Tracking Marble via Ultrasound Data

Makenna Barton - bartonmj@uw.edu

January 2020

Abstract This report outlines the theories and algorithms used to locate and calculate the trajectory of a marble swallowed by a dog. Through the use of the Discrete Fourier Transform, noisy signal averaging and filtering by frequency, we were able to successfully map the marble and target treatment. The theory behind the Fourier Transform, signal averaging and filtering, along with the corresponding MATLAB functions used to implement said strategies, are discussed and outlined in this report. The MATLAB code used to solve this problem is attached.

1 Introduction and Overview

Data is provided and was obtained using ultrasound to detect spatial variations in the abdominal area of the dog where the marble is suspected to be. This data is very noisy as the ultrasound can not specifically locate the marble. To make sense of the noisy data we will average out the noise to find the frequency signature of the marble, filter the data around that signature frequency to show the marble's path over time, and finally predict where the marble is going next. This information should lead us to where we should focus an intense acoustic wave to breakup the marble and save the dog.

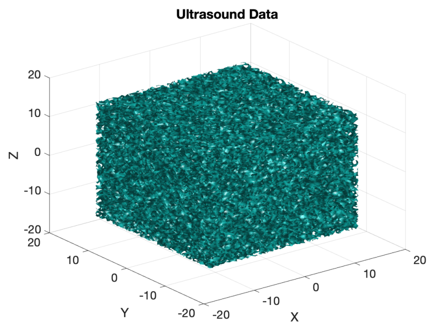

Figure 1: Original, unfiltered, noisy ultrasound data

2 Theoretical Background

The solution to this problem was reliant on the Fourier Transform to help make sense of the noisy data received. The Fourier Transform is a very powerful tool in mathematics because it allows us to represent a given function as a series of sines and cosines. The Fourier Transform is defined over all real numbers as an integral. The formula is written using exponential representation of sine and cosine:

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

where k is the frequency. The Inverse Fourier Transform is equally important. As it sounds, is transforms the function back to the spacial domain from the frequency domain (sines and cosines). It is written:

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

. For most applications, such as this one, it is more effective to use the Discrete Fourier Transform, defined on a finite spacial domain $x \in [-L, L]$. The formula is:

$$\hat{x}_k = \sum_{n=0}^{N-1} x_n e^{\frac{-2\pi ikn}{N}} \tag{3}$$

and the inverse:

$$x_n = \frac{1}{N} \sum_{n=0}^{N-1} \hat{x_n} e^{\frac{2\pi ikn}{N}} \tag{4}$$

where N is the total number of points and k is the frequency.

In most practical settings, such as our setting here, you are not provided a function to investigate, you are given a set of sample points. The catch with this is that, without every point defined, the transformation of a function into scaled sines and cosines cannot capture the large frequencies (k). This is when it becomes most helpful and appropriate to use the Discrete Fourier Transform. The issue with using the DFT is that you cannot tell from only the points, what the original function is. This leads us to a concept called Aliasing. Aliasing occurs when multiple functions are the same at a group of points, as sine and cosine are at different frequencies, making them indistinguishable from each other when you are only given a set of points. Periodic functions such as the ones used in the DFT, commonly have aliasing as a property, and when used in MATLAB, this needs to be considered (see Appendix A - fft()).

A final important property of the Fourier Transform should be considered here and it is described by the Heisenberg Uncertainty Principle. The principle states that when considering looking at a function in two domains, as we are here with the spacial domain (pre-transform) and the frequency domain (post-transform), the more you know about one space, the less you know about the other. Specifically, the more information you have about position of a wave, the less you know about its frequency. It is important to keep this principle in mind when using the Fourier Transform because the Transform is our means of jumping from one space to the other.

Because the data provided from the ultrasound is very noisy, we need a way to distinguish the noise from the true signal of the marble. Common ways to reduce noise are: filtering the data around a center frequency by applying a filter (most commonly, a Gaussian filter) or averaging the signals at different time steps together. To solve our problem specifically, we will use both methods, but we must use the averaging method first due to the fact that we do not know the center frequency of the marble. Averaging the signals over time will leave us with the frequency of the marble's signal because a property of white noise is that it has an average value of 0. Averaging of a signal must occur in the frequency domain as the spectrum remains fixed at the transmitted frequency signal, while time is changing.

Next, we follow with the filtering method on the data because we now know the frequency of the marble's signal. A Gaussian filter is written:

$$F(k) = exp(-\tau(k - k_0)^2) \tag{5}$$

where τ measures the bandwidth of the filter, k is the wave number, with k_0 as the the center frequency. Applying this filter to a Fourier Transformed function will isolate the signal input around the center frequency k_0. The frequencies close to k_0 are amplified, while the frequencies far away from the center frequency are muted. Again, filtering is only helpful in the spacial domain when you already know the desired center frequency. If you try to filter around an incorrect frequency, your filter will not detect any frequencies.

3 Algorithm Implementation and Development

To start to solve this problem I first looked at what the data provided looked like (see Figure 1). It was clear this data was very noisy and it needed to be simplified. As I said previously, this problem required the use of Signal Averaging due to the fact we did not know the frequency of the item we were looking for (the marble). After averaging over all the time steps (Appendix B, lines 36-42), I was left with a less noisy set of data. From this data, I obtained the signature frequency of the marble by finding the coordinates of the maximum frequency. This frequency was the center frequency that I could then filter around. After applying a Gaussian filter to the transformed data at each time step, I used the Inverse Discrete Fast Fourier Transform to obtain the filtered data in the spacial domain, now amplified around the frequencies caused by the marble. Next, at each time reading I found the position of the signal at its maximum, which was the position of the marble at each time step. I saved the 3D coordinates of the marble at each time step into X, Y, and Z vectors and then plotted the vectors against each other in 3D space. This resulted in a visualization of how the marble was moving over time in the stomach of the dog (Figure 2).

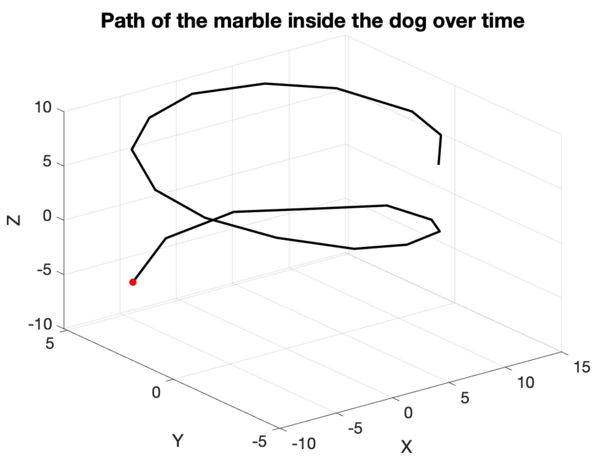

Figure 2: Above shows the path the marble took inside the dog during the ultrasound reading. The red dot shows where to focus an acoustic wave to break up the marble

4 Computational Results

After averaging the original signal over time, I found the signature frequency of the marble to have the frequency coordinates of $kx = 60, ky = 10, kz = 1$. Using this signature frequency I was able to filter the signal and obtain a plot of the movement of the marble inside the dog. Figure 2 shows the path of the marble through the dog over the time of the ultrasound. The computations above have lead me to the conclusion that to break up the marble at the 20th data measurement and save my dog, an acoustic wave should be focused at $x = -5.625, y = 4.2188, z = -6.0938$

5 Summary and Conclusions

Solving the problem of finding a marble inside the abdomen of a dog through ultrasounds showed the power of the Fourier Transform and combing though noisy data. The Fourier Transform gave us the ability to look at the signal's frequencies in more detail and over the time of the ultrasound. Through simplifying the signals in the ultrasound by averaging and filtering the data, we were able to determine the path of the marble in the stomach of my dog and pinpoint its final position. The use of averaging, the Discrete Fourier Transform, and filtering allowed us to target and break up the marble with an acoustic wave and save my dog.

6 Appendix A - MATLAB functions used and brief implementation explanation

`fftn()`- Fast Fourier Transform — Order: O(NlogN)
This function performs the transform on the function you specify. It also has some additional properties. When using `fftn()`you must define a spatial domain and a frequency domain (see Appendix B, lines 6-7). As you can see

when I am defining the frequency domain (k), I re-scaled the frequencies by $\frac{2\pi}{L}$, where L is the length of the spatial domain. I did this because in MATLAB, `fftn()` assumes that the frequencies are 2π periodic. Finally, `fftn()` also shifts the domain of the function to match the corresponding frequencies due to Aliasing. It should be noted that the `fftn()` function performs best when the number of points (n) is a power of 2 (i.e. 2^n).

`ifftn()`- Inverse Discrete Fast Fourier Transform
This function performs the inverse discrete transform, bringing the function back to the spatial domain from the frequency domain.

`fftshift()`- Fast Fourier Transform Shift
This function shifts the transform back to the mathematically correct spatial domain, the domain prior to the Aliasing property. In a vector x, `fftshift()` swaps the left and right halves of the vector.

`ind2sub()` This function requires you to pass in an index and the size of a matrix, and it determines the equivalent coordinates corresponding to the index for a matrix of the size passed in. This function was used to get the coordinates of the max frequency from the averaged function.

`reshape()` The reshape function "reshapes" a passed in matrix to the size parameter that was passed into the function. I used this function to represent the data as a 3D 64x64x64 matrix of points, rather than one long 1x262144.

`isosurface()` According to the Mathworks Documentation,
"fv = isosurface(X,Y,Z,V,isovalue) computes isosurface data from the volume data V at the isosurface value specified in isovalue. That is, the isosurface connects points that have the specified value much the way contour lines connect points of equal elevation." This was helpful for us to visualize out 3D data.

`plot3()` This function produces a 3-dimensional line plot of a given set of points.

7 Appendix B - MATLAB Codes

Matlab Code

```
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);
[Kx,Ky,Kz]=meshgrid(ks,ks,ks);
for j=1:20
    Un(:,:,:)=reshape(Undata(j,:),n,n,n);
end
isosurface(X,Y,Z,abs(Un),0.4)
axis([-20 20 -20 20 -20 20]), grid on, drawnow
set(gca,'Fontsize',14)
title('Ultrasound Data','Fontsize',16)
xlabel('X')
ylabel('Y')
zlabel('Z')

print(gcf,'-dpng','original_data.png')

%%
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);
[Kx,Ky,Kz] = meshgrid(k,k,k);

UnTAve = zeros(n,n,n);

% Average the transformed data
for j=1:20
    UnT = fftn(reshape(Undata(j,:),n,n,n));
    UnTAve = UnTAve + UnT;
end
UnTAve = abs(UnTAve)/j;

% Get the coodinates of the max frequency
[maxVals,indices1] = max(UnTAve(:));
[kx0,ky0,kz0] = ind2sub(size(UnTAve),indices1);
signatureFreq = [kx0 ky0 kz0]

% Filter the transformed data at each time step
tau = .2;
Kx0 = Kx(kx0,ky0,kz0);
Ky0 = Ky(kx0,ky0,kz0);
```

```
Kz0 = Kz(kx0,ky0,kz0);

filter =  exp(-tau.*((Kx-Kx0).^2 + (Ky-Ky0).^2 + (Kz-Kz0).^2));

for jj = 1:20
    UnT = fftn(reshape(Undata(jj,:),n,n,n));
    UnFT = UnT .* filter;
    UnF = ifftn(UnFT);
    [maxVals,indices] = max(abs(UnF(:)));
    [mx,my,mz] = ind2sub(size(UnF),indices);
    marbX(1,jj) = X(mx,my,mz);
    marbY(1,jj) = Y(mx,my,mz);
    marbZ(1,jj) = Z(mx,my,mz);
end
marb20 = [marbX(20) marbY(20) marbZ(20)];

plot3(marbX,marbY,marbZ,'k-','Linewidth',2)
set(gca,'Fontsize',14)
xlabel('X')
ylabel('Y')
zlabel('Z')
title('Path of the marble inside the dog over time','Fontsize',18)
grid on, hold on
plot3(marb20(1),marb20(2),marb20(3),'r.','Markersize',16)
print(gcf,'-dpng','marble_path.png')
```