Fourier Transforms

Ron Wilson

January 24, 2020

Abstract

In an attempt to save the fictional dog, Fluffy, who swallowed a marble, this project goes over numerical methods used to locate the marble. Specifically, it includes the use of Fourier transforms to find the center frequency of the marble, then track the path of the marble in time using a Gaussian filter and the inverse Fourier transform.

1 Introduction and Overview

The dog Fluffy swallowed a marble and was given an ultrasound to try to find the marble. However, Fluffy was moving too much, which caused the ultrasound to give very noisy data. By exploring Fourier transforms, we will denoise the data and locate the marble in order to determine where an intense acoustic wave should be focused to break up the marble.

This write-up focuses on three parts. In part one, I will average the spectrum from the ultrasound data to determine the center frequency generated by the marble. In part two, I will use this center frequency to apply a filter on the data to follow the path of the marble in time. In part three, I will visualize the path of the marble so that we can see where the acoustic wave should be focused.

2 Theoretical Background

In this section, I will give a brief background of the theory behind the numerical methods of this project.

2.1 Fourier Transform

We know that any given function $f(x)$ can be represented by a trigonometric series of sines and cosines:

$$f(x) = \frac{a_0}{2} + \sum_{i=1}^{\infty} (a_n \cos nx + b_n \sin nx) \quad x \in (-\pi, \pi]. \tag{1}$$

The coefficients a_n and b_n are called the Fourier coefficients, and can be solved for by multiplying both sides by cos mx, integrating over the spacial domain $x \in (-\pi, \pi]$, and then applying orthogonality. Then, these coefficients are found to be the following:

$$a_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \cos nx dx \quad n \geq 0. \tag{2}$$

$$b_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \sin nx dx \quad n > 0. \tag{3}$$

This Fourier series can be converted to a complex Fourier series in the domain $x \in [-L, L]$:

$$f(x) = \sum_{-\infty}^{\infty} c_n e^{\frac{in\pi x}{L}} \quad x \in [-L, L]. \tag{4}$$

The corresponding complex Fourier coefficients are

$$c_n = \frac{1}{2L}\int_{-L}^{L} f(x)e^{-\frac{in\pi x}{L}}dx. \tag{5}$$

This Fourier series governs a transform and inverse transform between the space and frequency domains. These transforms are called the Fourier transform and inverse Fourier transform:

$$F(k) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} e^{-ikx}f(x)dx. \tag{6}$$

$$f(x) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} e^{ikx}F(k)dk. \tag{7}$$

The ability to transform between the space and frequency domains will be extremely helpful in analyzing the ultrasound data. We can find the frequency given by the marble by using the Fourier transform. Then, we can use the inverse Fourier transform to follow the marble back into the space domain to find its position at a specific time-step. We will utilize the Fast Fourier transform (FFT) and its corresponding inverse (IFFT) algorithms in MATLAB to accomplish this. As the name suggests, the FFT algorithm is designed to find the Fourier transform very quickly. See Appendix A for the MATLAB functions corresponding to FFT and IFFT.

2.2 Gaussian Filter

Another important concept is applying a filter to remove any unwanted frequencies. This makes locating a target frequency and tracking its path much easier to visualize. There are many different types of filters to choose from, but for this problem I will be using a Gaussian filter:

$$\mathcal{F}(k) = e^{-\tau(k-k_0)^2} \tag{8}$$

Note that τ is the bandwidth of the filter centered on the signal center frequency k_0. After using the FFT to locate the center frequency given off by the marble, I will be able to place this filter around that signal to find the path of the marble.

3 Algorithm Implementation and Development

In this section, I will outline the numerical methods used to find the path of the marble through Fluffy's intestines. Refer to the MATLAB code in Appendix B to see how these methods are implemented.

1. To start, we first need to load the noisy data and setup the spacial and frequency domains. It is important to note that the FFT assumes that we have 2π periodic signals. So, the wavenumber k which corresponds to the eigenvalues of the Fourier transform (6) needs to be re-scaled by $\frac{2\pi}{L}$. Now that we have the frequency domain setup, we can average the spectrum to find the center frequency.

2. Next, I average the frequencies over each of the 20 time measurements of the data. Using this average frequency, I apply the multi-dimensional version of the Fast Fourier Transform to produce the frequency spectrum at each point in time, and then find the average of the spectrum's.

   (a) Note that I can take the absolute value of this average and normalize by its maximum value to plot the average frequency spectrum at each time-step. See Figure 1 for the average frequency spectrum at the 20th time-step.

3. Now, I can take the maximum of the average frequency in MATLAB to find the index corresponding to the peak frequency (the frequency given off by the marble). By taking this index and plugging it into the frequency domain, I can find the exact coordinates in the frequency domain of this peak frequency.

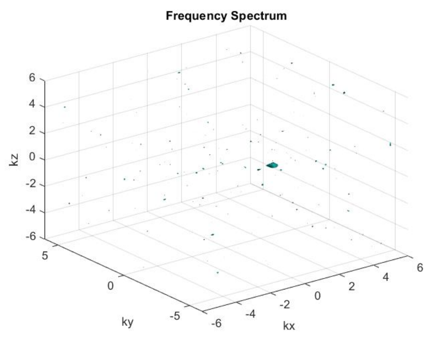

Figure 1: Average frequency spectrum

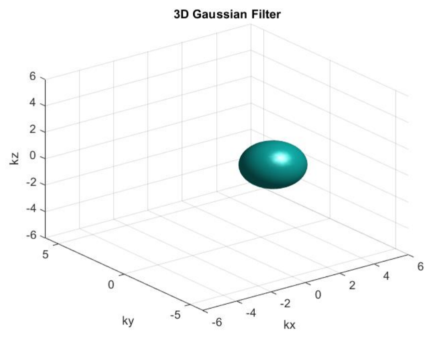

Figure 2: 3D Gaussian filter applied to center frequency

4. With the coordinates of the center frequency, I can place a filter around the frequency given off by the marble and track its path through each time-step. The Gaussian filter as specified by equation (8) can be converted to a 3D Gaussian filter around the center frequency. For this problem, I will use a bandwidth of $\tau = 0.2$. See Figure 2 to see what this 3D Gaussian filter looks like.

5. Multiplying the data measurements at each time-step by the Gaussian filter allows us to apply the filter to track the center frequency of the marble through time. A visual of the filter tracking the marble through the frequency domain can be seen in figure 3.

6. Taking the inverse Fourier transform allows us to generate the marble's path in space. The marble's path is seen in figure 4.

4 Computational Results

Figure 4 gives us the denoised ultrasound data. Using the final location of the marble in time (the 20th data point), an intense acoustic wave that will breakup the marble should be focused at $(x, y, z) = (-5.625, 4.219, -6.094)$.

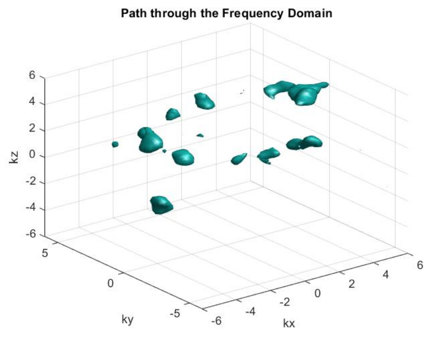

Figure 3: The path of the filter

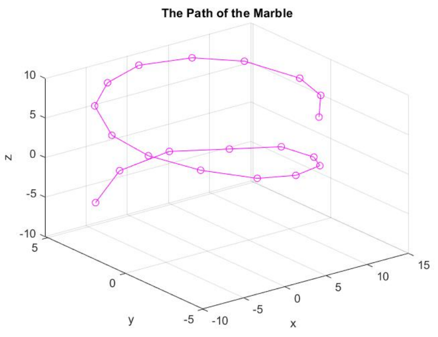

Figure 4: The marble's path in space

5 Summary and Conclusions

We were able to take the noisy data from the ultrasound and locate the frequency signature of the marble through the use of Fourier transforms. Then, we applied a Gaussian filter over this frequency signature and used the inverse Fourier transform to find the path of the marble at each data measurement. Finally, we used this path to find the exact location of the marble at the 20th data measurement, where we can breakup the marble with an intense acoustic wave to save Fluffy.

In conclusion, the FFT and IFFT algorithms are very useful tools to isolate large frequencies and map them back to the spatial domain. However, this only works well for the high frequencies given off by the marble. It would have been very difficult to find the marble if there were other high frequencies within the data measurements.

Appendix A MATLAB Functions

The following is a list of the important MATLAB functions that were used:

- `y = linspace(x1,x2,n)` returns a row vector of `n` evenly spaced points between `x1` and `x2`.
- `X = zeros(sz1,...,szN)` returns an `sz1-by-...-by-szN` array of zeros where `sz1,...,szN` indicate the size of each dimension.
- `[X,Y] = meshgrid(x,y)` returns 2-D grid coordinates based on the coordinates contained in the vectors `x` and `y`. X is a matrix where each row is a copy of `x`, and `Y` is a matrix where each column is a copy of `y`. The grid represented by the coordinates `X` and `Y` has `length(y)` rows and `length(x)` columns.
- `B = reshape(A,sz1,...,szn)` reshapes `A` into a `sz1-by-...-by-szN` array where `sz1,...,szN` indicates the size of each dimension.
- `Y = fftn(X)` returns the multidimensional Fourier transform of an N-D array using a fast Fourier transform algorithm. The N-D transform is equivalent to computing the 1-D transform along each dimension of `X`. The output `Y` is the same size as `X`.
- `Y = fftshift(X)` rearranges a Fourier transform `X` by shifting the zero-frequency component to the center of the array.
- `fv = isosurface(X,Y,Z,V,isovalue)` computes isosurface data from the volume data `V` at the isosurface value specified in `isovalue`.
- `M = max(A)` returns the maximum elements of an array.
- `[row,col] = ind2sub(sz,ind)` returns the arrays `row` and `col` containing the equivalent row and column subscripts corresponding to the linear indices `ind` for a matrix of size `sz`.
- `X = ifftn(Y)` returns the multidimensional discrete inverse Fourier transform of an N-D array using a fast Fourier transform algorithm. The N-D inverse transform is equivalent to computing the 1-D inverse transform along each dimension of `Y`. The output `X` is the same size as `Y`.
- `plot3(X,Y,Z,LineSpec)` creates a plot in 3-D space using the specified line style, marker, and color.

Appendix B MATLAB Code

The MATLAB code used to generate the results of this write-up (`Fourier_Transforms.m`) can be found at the following GitHub repository: `https://github.com/ronwilson016/Data-Science-Projects/tree/master/Fourier%20Transforms/Matlab%20Code`