HW 1: An Ultrasound Problem

Joshua Whalen
1/24/2020

Abstract

My dog Fluffy recently swallowed a marble. The marble needed to be removed by an intense acoustic wave, but the path and final position of the marble was unknown. In this assignment, I filtered noisy data from an ultrasound by averaging the signal over 20 time points to find the central frequency and applying a simple Gaussian filter set to the center frequency. This allowed me to filter out the noise generated by Fluffy's movement and the fluid of the intestines, and track only the position of the marble. A graph of the marbles movement is shown in Fig 2. In conclusion, to save Fluffy's life an intense acoustic wave needs to be focused at position [9.84, 4.92, 4.69] in the xyz plane.

Sec. I. Introduction and Overview

The purpose of this paper is to apply filtering and signal processing methods in a real world application. The original signal was obtained through ultrasound. The signal consisted of a 3 dimensional spatial recording taken at 20 time points. The data from 1 of the 20 time points is shown in Fig 1. as an isosurface. It is clear from the figure that there is a lot of noise and determining the position of the marble with the current data is impossible. In order to find the position of the marble, we must filter the data.

I will now described the overall process for filtering the data and obtaining the position of the marble. First, we need to define our spatial domain and Fourier modes. This will simply determine the space that we are sampling in and how many cuts we are making in this space. This is important for scaling between the spatial and frequency domain. Next, we needed to transform our spatial data to frequencies data. Then by averaging our 20 data points and taking the max of those averages we can find the central frequency, which represents the frequency from the marble. With the central frequency, we can now iterate through our original data with a simple Gaussian filter set at the center frequency. This should clear up the noise and allow us to see the movement of the marble in the 20 time points.

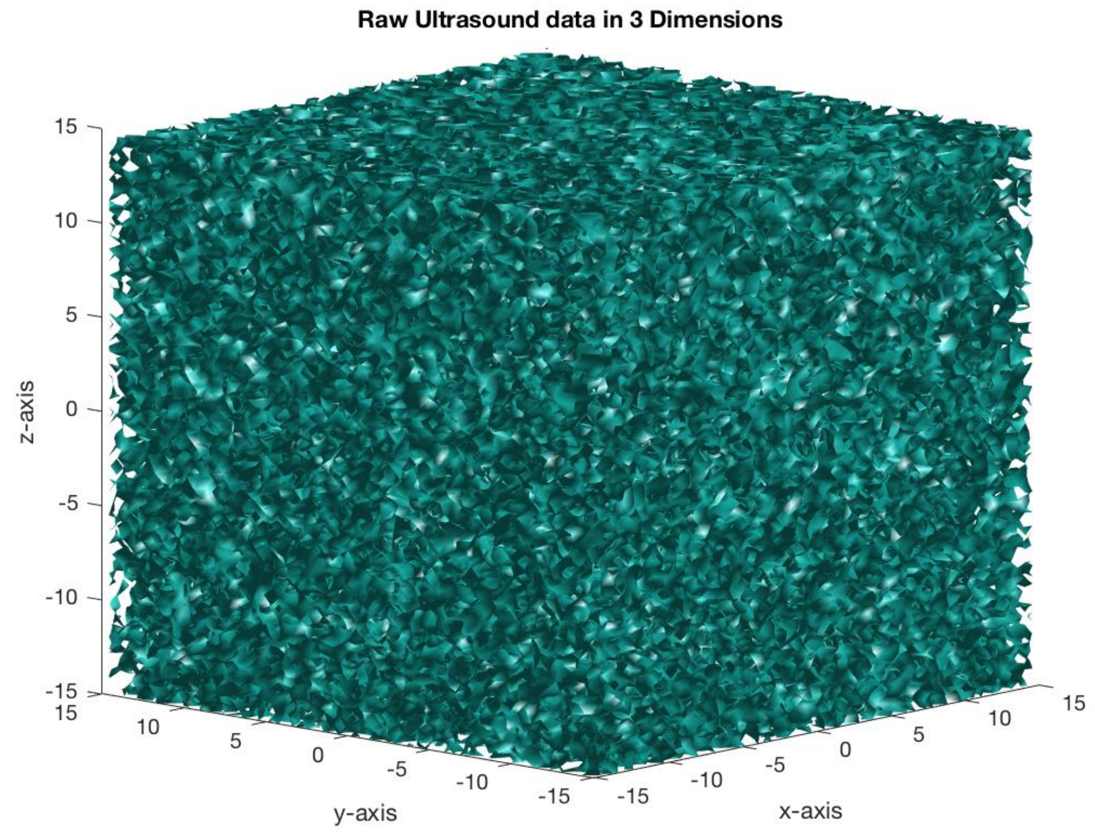

Fig 1. This figure shows the raw spatial data from the ultrasound shown as an isosurface.

Sec. II. Theoretical Background

Data and noise often come hand in hand, so it is nearly impossible to obtain a raw signal completely devoid of noise without applying some kind of filter. Filtering data involves moving between the spatial domain and frequency domain. This essentially is the difference between mapping all signals as a function of space and time verses mapping the abundance any independent signal (frequency) against all frequencies. This is extremely useful because as seen in Fig 1. discerning the marble frequency from the general noise in the spatial domain is difficult. Instead, by using the Fourier transform, we can split the signal into its separate frequencies. Then if we take the max of all of those frequencies we will obtain the frequency of the marble. The marble will be the max frequency because the only other signal recorded by the ultrasound is noise, which should be significantly lower in amplitude. Now that we know the frequency that corresponds with the marble location we can build a Gaussian filter centered around that frequency, and attenuate all of the other frequencies.

Sec. III. Algorithm Implementation and Development

As explained in the previous introduction sections, we need to implement two main signal processing methods to obtain the positional data of the marble. First, we need to average the spectrum to find the central frequency. Then, we need to generate and apply our filter. This section will go into the details for developing and implementing the code needed for both of these signal processing methods. All of the code used for this assignment was written in a MATLAB coding environment and posted in Appendix B. Additionally, all functions used throughout the code will be explained in Appendix A.

Averaging of the Spectrum

First, we need load the data using "load Testdata." The data from the ultrasound contains 20 rows, each row containing spatial information for each of the 20 time points. To make this data in a useful format, we will use the "reshape" command to change the vector into a 64 by 64 by 64 array at each time point. Then, we need to transform each array into the frequency domain using the command "fftn." Since, all of these arrays will be compiled into an average, there is no need to save them separately. Instead, we can implement a for loop that will iterate through all of the 20 rows, reshape them, transform them with fftn, and sum. Additionally, it is recommended that you define your array that holds the sum before the loop so that the array dimensions are not updating every iteration. After the loop, divide the average by 20 to represent the 20 time points and use the "max" and "ind2sub" command to find the indices of the max frequency, which will tell us the center frequency for the filter.

Filter Generation & Application

The filter needs to be applied to every point in the 64 by 64 by 64 array. To build the filter we need to first make a nested for loop for each dimension x, y, and z going from 1:64. Then we need to save our filter function in each index as a new array. The function itself is based on a Gaussian equation: $e^{-a(k-k_0)^2}$, where a represents the bandwidth of the filter, k_0 is the center frequency, and k is the wavenumber. But, since we the data is 3 dimensional the filter equation much be expanded: $e^{-a((k_x-k_{x0})^2+(k_y-k_{y0})^2+(k_z-k_{z0})^2)}$. Now that we've generated our filter we need to apply it to the original data. Similarly, to averaging the spectrum, we need to set up another for loop to iterate through the raw data reshape the array and transform it to the frequency domain, but now instead of finding the average with a summation, we multiple our filter to the data in the frequency domain. Multiplying in the frequency domain will convolve our filter and the data in the time domain. Then we need to transform our data back into the time domain with the function "iffn" and save the index of the max signal in 3 new position vectors, which we can plot later. Finally, using the "plot3" command and the 3 position vectors we can track the movement of the marble.

Sec. IV. Computational Results

In this section, I will discuss the results of applying the methods described in section III to the ultrasound data set. After averaging the spectrum and finding the max frequency, which correlates to the marble frequency, I found the center frequency was at the index [60,10,1]. With the center frequency, I was able to generate a Gaussian filter function $e^{-0.05((k_x-60)^2+(k_y-10)^2+(k_z-1)^2)}$. This equation will attenuate the frequencies away from the marble frequency at [60,10,1]. The bandwidth was set at 0.05 arbitrarily. Generally, as the bandwidth value increases the window size will decrease. So a smaller value will give a larger bandwidth. 0.05 was seen to be a good filter width. After iterating through the original data with the new filter function, I plotted the position vectors of the marble and obtained the graph shown in Fig 2. As shown in the figure, the final position of the marble is [9.84, 4.92, 4.69].

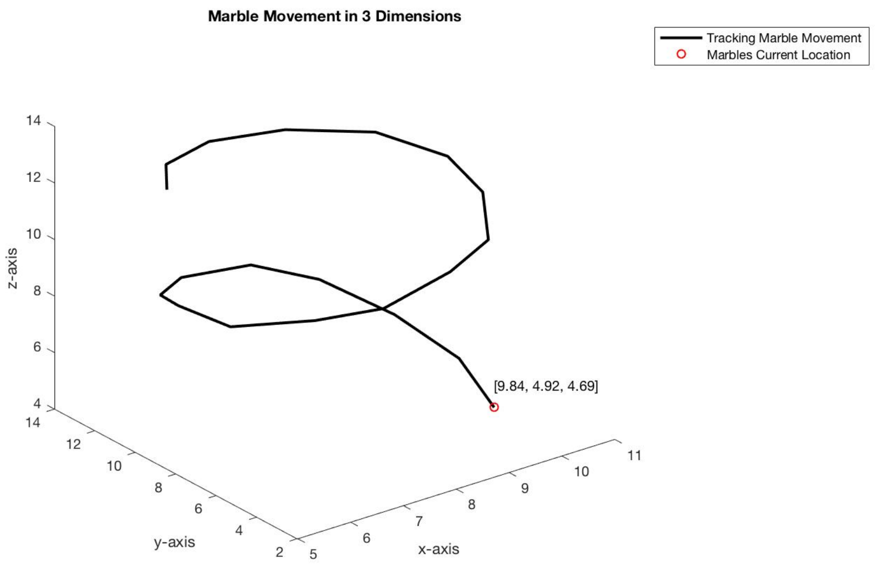

Fig 2. This figure shows the movement of the marble in 3 dimensions. The black line shows the path of the marble. The red circle denotes the current location of the marble.

Sec. V. Summary and Conclusions

In conclusion, to find the position of the marble in the noisy ultrasound data, I needed to implement an averaging of the spectrum and a filter based on the center frequency from the average. To dislodge the marble and save Fluffy, the intense acoustic wave must be focused at position [9.84, 4.92, 4.69] in the xyz plane. This assignment utilized the signal processing techniques essential to data collection. The analysis and filtering of signals is applicable in most any modern electronic device.

Appendix A. MATLAB functions used and brief implementation explanation

- load: Uploads variables saved in a .mat file. Used to upload raw ultrasound data.
- reshape: Changes a vector into an array for a determined size. Used to change the ultrasound data into 20 separate 64 by 64 by 64 arrays.
- fftn: Computes the fast Fourier transform of the given data set. Used to move our data set from the spatial domain to the frequency domain.
- max: Finds the max value in an array. Used to find the max amplitude among the frequency signals and position of the marble later on.
- ind2sub: Finds the indices of the max value.
- ifftn: Computes the inverse fast Fourier transform for a given data set. Used to move our data set from the frequency domain back to the special domain.
- plot3: Generates a plot in 3 dimensions. Used to visualize the path and location of the marble.

Appendix B. MATLAB codes

```
clear all; close all; clc;
load Testdata
L=15; % spatial domain
n=64; % Fourier modes

%Averaging The Spectrum
ave = zeros(n,n,n); %defining the average array
for j=1:20 %looping through the data reshaping, transform, and summation.
    Un(:,:,:)=reshape(Undata(j,:),n,n,n);
    Unt = fftn(Un);
    ave = Unt+ave;
end
[mxv,idx] = max(ave(:)/20); %find the max and indices
[r,c,p] = ind2sub(size(ave),idx);
```

```
%Generating and Applying the Filter
a = 0.05; %bandwidth
filter = zeros(n,n,n); %defining filter array
for x=1:64
    for y=1:64
        for z=1:64
            filter(x,y,z) = exp(-a*((x-r).^2+(y-c).^2+(z-p).^2)); %filter
equation
        end
    end
end

%defining position vectors for marble movement
xpos = zeros(1,20);
ypos = zeros(1,20);
zpos = zeros(1,20);

for j=1:20
    Un_filter(:,:,:)=reshape(Undata(j,:),n,n,n);
    Unt_filter = fftn(Un_filter).*filter; %applying filter
    marble = ifftn(Unt_filter);
    [m,id] = max(marble(:));
    [xpos(j),ypos(j),zpos(j)] = ind2sub(size(marble),id); %saving marble
position
end

%ploting marble position
plot3(xpos.*L/n,ypos.*L/n,zpos.*L/n,'k','LineWidth',2)
hold on
plot3(xpos(end)*L/n,ypos(end)*L/n,zpos(end)*L/n,'ro')
title('Marble Movement in 3 Dimensions')
legend('Tracking Marble Movement','Marbles Current Location')
text(xpos(end)*L/n,ypos(end)*L/n,zpos(end).*L/n+0.8, '[9.84, 4.92, 4.69]')
xlabel('x-axis')
ylabel('y-axis')
zlabel('z-axis')
%%
%Raw data visualization
test(:,:,:)=reshape(Undata(5,:),n,n,n);
isosurface(X,Y,Z,abs(test),0.4)
title('Raw Ultrasound data in 3 Dimensions')
xlabel('x-axis')
ylabel('y-axis')
```