HW1: An Ultrasound Problem

AMATH 482 | Celeste Becker

Abstract

Given a noisy Ultrasound dataset, the goal of this problem is to determine the frequency signature of a marble, its trajectory, and its location at any point in time.

The original noisy data is in the spatial domain. Using Fourier transforms to convert the data into the frequency domain, the data is then averaged to find the frequency signature (center frequency) of the marble. Next, a Gaussian filter is used to remove noise. Finally, by using an inverse Fourier Transform to convert the data back into the spatial domain, we have a clean data set to observe the marble.

Sec. I. Introduction and Overview

In this problem, my dog Fluffy has swallowed a marble. Ultrasound data is collected about the spatial variations in a small area of the intestines. This is where the marble is suspected to be. However, the data is filled with random noise due to Fluffy moving while getting the Ultrasound.

The Ultrasound data is in the spatial domain and contains 20 rows of data corresponding to 20 different measurements taken in time.

To save my dogs life, I must find the marble and its trajectory. Using MATLAB, I will analyze the data to find where the marble is located at the 20^{th} time measurement (the last time measurement) and thus where an intense acoustic wave should be focused to break up the marble.

I will use averaging, Fourier transforms, and Gaussian filtering on this data set to find the location of the marble.

Sec. II. Theoretical Background

The three main mathematical methods used in this program are *Averaging*, *Fourier Transforms* and *Gaussian Filtering*. The information here is mainly referenced from sections 13.1 through 13.4 of Data-Driven Modeling & Scientific Computation: Methods for Complex Systems & Big Data by J. Nathan Kutz.

Averaging

Given a dataset that has random noise (also known as white noise), if you average the data by taking the Mean, the white noise on average should add up to zero. This leaves behind only the signals in the dataset that are not random. “Averaging over the frequency domain produces a clear signature at the center frequency of interest” Kutz, page 331.

Fourier Transform

The Fourier Transform takes a function in the time domain and represents it in the frequency domain. It uses **Euler's Identity** ($e^{\pm ix} = \cos(x) \pm isin(x)$) to represent the function and its derivatives in terms of the sums of cosines and sines:

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

In the equation above, the **kernel** $= e^{-ikx}$. This describes the oscillatory behavior of the function.

The inverse Fourier Transform converts a function from the frequency domain to the time domain:

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

To use the Fourier Transform in MATLAB, we use the **Fast Fourier Transform algorithm**, FFT. Key features of the FFT algorithm are:

1. It finds the transform on a finite interval: $x \in [-L, L]$.
2. The solutions of this finite interval have periodic boundary conditions because the integration of the kernel is oscillatory.
3. It integrates over the discrete interval of -2π to 2π. This means data must be re-scaled to fit this domain.
4. The number of data points is assumed to be 2^n. With n being any positive real number.

Gaussian Filtering

Gaussian Filtering allows us to extract information at specific frequencies, effectively removing noise from a signal.

$$G(k) = e^{-\tau(k - k_0)^2}$$

The τ is a measurement of the bandwidth of the filter, and k is the wave number. The center frequency is when $k = k_0$. The gaussian function is centered around the center frequency. When multiplying data in the frequency domain by a gaussian filter, we are removing all frequencies around the chosen center frequency. This filters the data, so you are only left with the center frequency.

Sec. III. Algorithm Implementation and Development

Each section labeled here corresponds to the commented section with the same name in *Appendix B. Matlab Codes* at the end of this report. Code reference lines are also included.

Load Data – (lines 3-5):

In this section, the Ultrasound Data is loaded into the program. (Line 4).

Set up – (lines 6 – 13):

In the spatial domain, we have vectors of $x, y, z \in [-15, 15]$. We then turn this into a 3D grid matrix with size of 64 x 64 x 64. In the frequency domain, we have vectors Kx, Ky, and Kz represented by $Kx, Ky, Kz \in \left[\frac{2\pi}{2*15}(-32),\ \frac{2\pi}{2*15}(31)\right]$. We rescale the frequencies by factors of $\frac{2\pi}{L}$ because FFT assumes 2π periodic signals. Then we also turn this into a 3D grid matrix with size of 64 x 64 x 64. By splitting up the data by 64, we see that 64 $= 2^6$, which allows us to use the FFT algorithm.

Average in the Frequency Domain - (lines 14-24):

To average in the frequency domain, an average variable is created as an empty matrix of 64 x 64 x 64. Each row in the Ultrasound dataset is reshaped into a 64 x 64 x 64 matrix. The average is equal to the sum of the Fourier transform of each reshaped row in the Ultrasound dataset. Then the average value is divided by 20 using element wise division and the absolute value is taken. The averaged dataset now contains almost only the frequencies corresponding to the center frequency of the marble.

Find Center Frequency from Averaged Data - (lines 25-32):

To find the center frequency, we find the maximum value and its corresponding index in our averaged dataset. By plugging this index value into each frequency vector, Kx, Ky, and Kz, we find the value of the center frequencies of the marble: Kx0, Ky0, and Kz0.

Gaussian Filtering - (lines 33-63):

First the the bandwidth variable Tau is set to 1. Then using the values for the center frequencies we just found, we build the gaussian filter function.

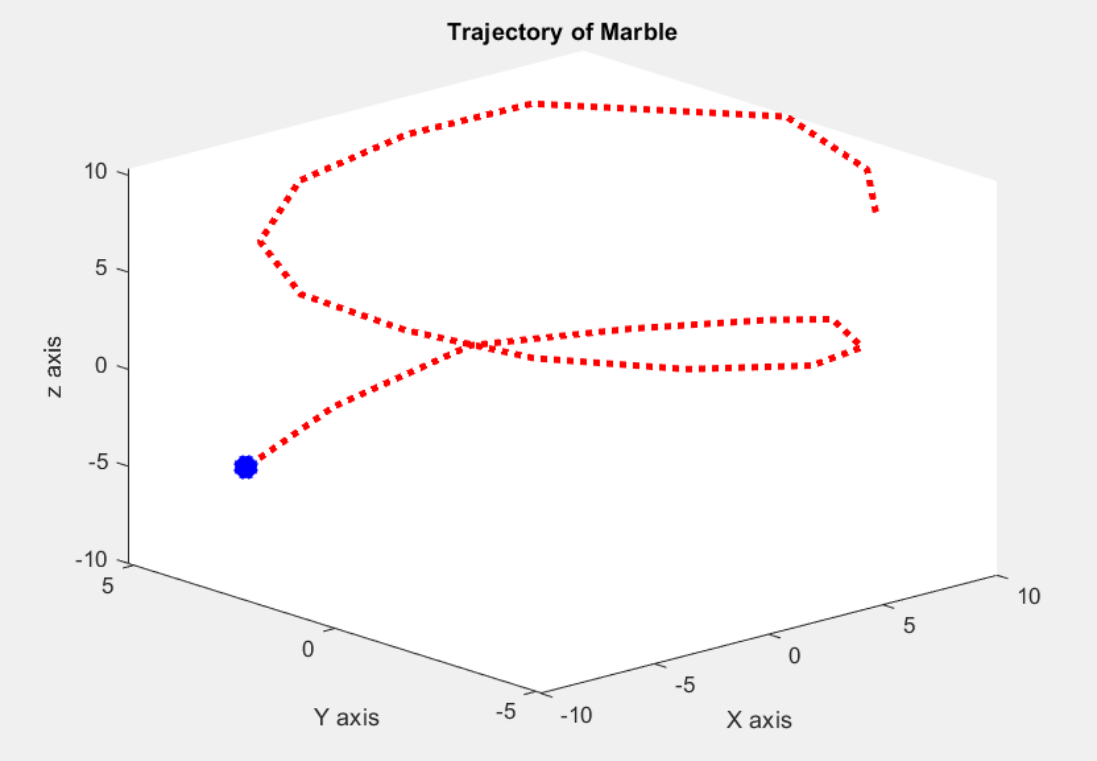
$$filter = e^{-\tau[(k-kx_0)^2+(k-ky_0)^2+(k-kz_0)^2]}$$

Next, we create blank vectors to track the marble position's x, y, and z coordinates at each of the 20 points in time.

For each row of Ultrasound data, we reshape it into the 64 x 64 x 64 matrix and take it's Fourier transform. Then we multiply this Fourier transformed data by the filter to filter out the noisy data. Now only the center frequency of the marble is left. Then by taking the inverse Fourier Transform of the filtered data, we convert back to the spatial domain. Then we store the marble's x, y, and z position at that given moment in time into the position vectors.

Plot Marble Trajectory and Last Data Point - (lines 64-78):

Here, plot commands are used to plot the marble's trajectory and its final value at the last 20th datapoint.



The red dashed line is the trajectory of the marble, the blue dot is the marble at the last time measured.

Sec. IV. Computational Results

The marble's trajectory is plotted in the spatial domain seen in red in the image above. As time progresses the marble spirals down.

$(-5.625, 4.687, -6.0938)$ is the final position of the marble at the 20th data point represented by the blue dot.

Sec. V. Summary and Conclusions

In summary, I was able to successfully take a noisy Ultrasound data set, and apply averaging, Fourier transforms, and gaussian filtering to find the trajectory and location of the marble in Fluffy's intestines.

Now an intense acoustic wave can be focused at the final (X,Y,Z) position:
$(-5.625, 4.687, -6.0938)$ to break up the marble and save Fluffy!

Appendix A. MATLAB functions used and brief implementation explanation

Meshgrid

- [X, Y, Z] = meshgrid(x, y, z)

- Returns a 3-D grid of coordinates defined by vectors x, y, and z
- *Implementation*: Used to make a 3D grid of the spatial domain, and of the frequency domain.

Reshape

- Reshape(A, [a,b,c])
- Reshapes an input array A into an matrix of dimensions a,b,c
- *Implementation*: Used to reshape the Ultrasound data into a 64 x 64 x 64 matrix.

fftn

- Y = fftn(X)
- Returns the N dimensional Fourier Transform of an N-D array using the fast Fourier Transform algorithm.
- *Implementation*: Used to convert data from the spatial domain into the frequency domain.

ifftn

- X = ifftn(Y)
- Returns the multidimensional discrete inverse Fourier Transform an N-D array using the fast Fourier transform algorithm.
- *Implementation*: Used to convert data from the frequency domain into the spatial domain

max

- [M, I] = max(A)
- Returns the index into the operating dimension that corresponds to the maximum value of A
- *Implementation*: Used to find the center frequency of the marble, and the position of the marble

ind2sub

- [i1, i2, i3] = ind2sub([n,n,n],I)
- Converts the linear index I into row and column subscripts for a matrix of size n
- *Implementation*: Used convert the linear index I into row and column subscripts for the spatial domain

Appendix B. MATLAB codes

```
clear; close all; clc;

%% Load Data - (lines 3-5)
load Testdata

%% Set up - (lines 6-13)
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);

%% Average in Frequency Domain - (lines 14-24)
avg = zeros(n,n,n);
for row = 1:20
    Un = reshape(Undata(row,:),n,n,n);

    Unf = fftn(Un);

    avg = avg + Unf;
end
avg = abs(avg./20);

%% Find Center Frequency From Averaged Data - (lines 25-32)
[maxValue, indexOfMax] = max(avg,[],'all', 'linear');

%center frequencys in Kx, Ky and Kz
Kx0 = Kx(indexOfMax);
Ky0 = Ky(indexOfMax);
Kz0 = Kz(indexOfMax);

%% Gaussian Filtering - (lines 33-63)
%Tau = the bandwidth of the filter
t = 1;

%Gaussian filter function
filter = exp(-t*((Kx-Kx0).^2 + (Ky-Ky0).^2 + (Ky - Ky0).^2));

%Marble Position vectors
x0 = zeros(1, 20);
y0 = zeros(1, 20);
z0 = zeros(1, 20);
```

```
for row = 1:20

    Un = reshape(Undata(row,:),n,n,n);

    Unf = fftn(Un);

    UfilterFS = Unf.*filter;

    UfilterSS = abs(ifftn(UfilterFS));

    [maxPos, I] = max(UfilterSS(:));
    [i1, i2, i3] = ind2sub([n,n,n], I);

    x0(1, row) = X(i1, i2, i3);
    y0(1, row) = Y(i1, i2, i3);
    z0(1, row) = Z(i1, i2, i3);

end

%% Plot Marble Trajectory and Last Data Point - (lines 64-78)
plot3(x0,y0,z0, 'r:', 'Linewidth', [3]);
hold on
plot3(x0(20), y0(20), z0(20),'b*', 'Linewidth', [10]);
title('Trajectory of Marble');
xlabel('X axis');
ylabel('Y axis');
zlabel('z axis');
```