Homework 1: An Ultrasound Problem

Joshua Yap
AMATH 482
Winter 2020

March 16, 2020

Abstract

This paper goes through the process of denoising signals in a real-world application. MATLAB functions are used to perform Fourier transforms, average and filter the signals to obtain the end result of a marble's position in 3D.

1 Introduction and Overview

My dog Fluffy has swallowed a marble. Ultrasound data shows it in the intestines, but movement of internal fluid around the marble generates highly noisy data. To save Fluffy's life, the trajectory of the marble must be computed using the given data. This paper (a) determines the frequency signature of the marble, (b) filters data around the frequency signature to denoise it, (c) computes the path of the marble and (d) determines the location an intense acoustic wave should be focused at the 20th measurement to break up the marble.

2 Theoretical Background

Noisy data is typical in real-world applications. In the case of signal processing, it can cause ambiguity in signal detection, especially in cases of low signal-to-noise ratio (where signal amplitude is low or noise amplitude is high) or when position of a signal in time is required. For our application, we are required to determine the path of the marble through Fluffy's intestines.

The given data shows ultrasound wave detection in the spatial domain. We are provided code that plots this data 20 times, each plot corresponding to a different point in time. One such plot is shown in Figure 1. The plot looks like a solid cube because there is noise encoded in the data. This paper presents the denoising process to get the data to give us a plot that looks like a marble.

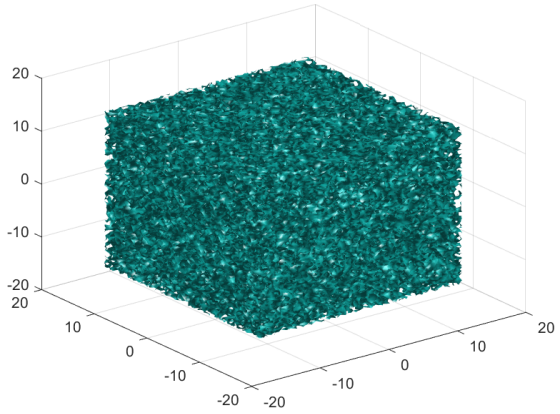


Figure 1: An example of a plot generated using the given data at the 20th time point. The surface connects points of equal magnitude in the given data.

2.1 *Fourier Transform*

The problem requires us to use Fourier transforms to denoise the data. The Fourier transform takes a function in spatial or temporal bases and generates a new function in the frequency basis. Specifically, the transform and its inverse are defined as

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

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

The transform allows us to take the data we are given and analyze the frequencies it contains so that we can isolate the frequency signature of waves bouncing off the marble and filter out much of the noise.

In MATLAB, the Fourier transform is done in a discrete manner known as the Fast Fourier transform (FFT). It has the following key features:

(i) It has a low operation count of $O(N\log N)$.

(ii) It assumes that input signals are 2π periodic, so we must scale our signal by $2\pi/L$, where L is the length of our spatial domain.

(iii) We need to discretize our spatial domain into 2^n points to take advantage of the low operation count.

(iv) It has excellent accuracy properties.

Implementing the FFT gives us a small imaginary component due to numerical round off errors, so we use absolute values in our analysis.

2.2 *Averaging and Filtering*

Another aspect of the analysis is averaging. We use this to find the frequency signature of the marble. We construct the average by summing the signal in the frequency domain at each time point, and then dividing by the number of time points. This takes advantage of the fact that the noise is white - it affects all frequencies equally and has zero mean. In other words, the noise should sum to zero as we average over more and more time points. What remains is the marble's frequency signature since that does not sum to zero.

Once we have averaged the signal, we construct a filter around the marble's frequency signature - the filter lets through signals with similar frequencies and suppresses those with frequencies far away from it. In our case, we use a Gaussian filter defined as

$$G(k) = e^{-\tau(k-k_0)^2} \tag{3}$$

τ and k_0 are parameters we tune to get the desired filter. τ controls the width of the filter and k_0 determines the frequency at which the filter is centered. In our application, this will be the marble's frequency signature. To 'apply the filter' means to multiply the filter by the signal.

3 Algorithm Implementation and Development

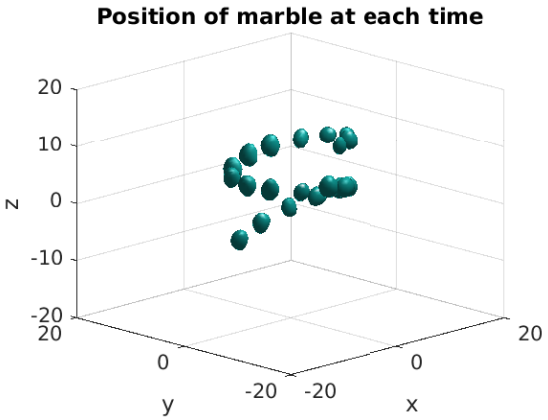
Define domains: The algorithm begins by defining the domains in space and frequencies in 3 dimensions.

Average: We then iterate through each time point in the given data and transform it to the frequency domain using the fft() command. We sum them and then divide the sum by 20, the number of time points.

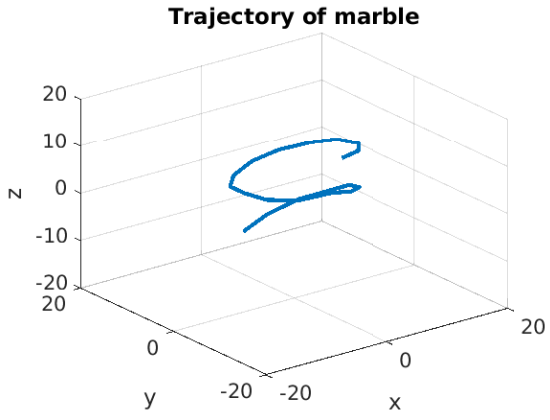
Find peak: In 1 dimension, the frequency signature we are looking for would look like a peak on the y-axis. We generalize to 3 dimensions by finding the maximum value in our averaged signal. The index corresponding to this maximum in the array maps to the frequencies in the x, y and z directions.

Construct filter: We vectorize the Gaussian filter above to work in 3 dimensions and center it at the frequencies found using the index. We choose τ by trial and error - plots were made with different values of τ and $\tau = 0.4$ yielded reasonable results.

Apply filter: Next we again iterate through each time point, this time applying the filter to each one. The process is to Fourier transform each signal, then apply the filter, then inverse Fourier transform it to get the signal back in the spatial domain.



(a) Filtered data plotted on the same axes. Each sphere represents the marble at each time point.



(b) Trajectory of marble obtained from positions of marble.

Figure 2: Plots generated from filtered signals

Determine marble position: Once we have the filtered signal, the plot looks like a marble. We determine its position by finding the index in the signal array that gives us the maximum value.

Plot trajectory: Using a line to connect the position of the marble at each time gives us its trajectory.

4 Computational Results

The domain defined is $x, y, z \in [-15, 15]$ with 2^6 (or 64) frequencies. Averaging the signals gave us the frequency signature $k_0 = (-4.8171, -6.7021, -1.0472)$, where each value corresponds to the x, y and z frequencies respectively. Our filter was constructed with the width $\tau = 0.4$ determined by trial-and-error - plots were made with different values of τ and $\tau = 0.4$ yielded reasonable results. Upon applying the filter to the signal, we obtained plots that resembled a marble. Figure 2(a) shows the result of the plots at different times. The filter was effective in getting rid of the noise and left mostly just the marble.

We find the trajectory of the marble by interpolating its position at different times. The result is Figure 2(b), with the marble starting at the top of the blue line and spiraling down. At the 20th time point, the marble ends up at the point $(-5.1562, 4.2188, -6.0938)$. It is at this point that an intense acoustic signal should be focused to break up the marble.

5 Summary and Conclusions

We have taken noisy data and used the FFT and filtering to denoise it such that it yields the signal we want - the marble's position in time. It was a fairly quick algorithm - running the code each time took a total of about 10 seconds, during which we obtained plots of the marble's position and trajectory.

Appendices

Appendix A

Summary of MATLAB functions used

linspace: Creates a vector of equally-spaced points.

meshgrid: Used to define the spatial and frequency domains in 3D.

fft: Transforms original signal from 3-space to frequency space.

fftshift: Shifts values of Fourier transformed signal to match basis frequencies.

max: Find the maximum value in array.

reshape: Rearrange dimensions in an array to perform fft or to plot.

ifft: Inverse Fourier transform to get signal back to 3-space.

ifftshift: Undo the shift of signal values to get back to the original arrangement of values in the array.

isosurface: Plots the marble surface in 3D.

plot3: Plots the marble trajectory as a line in 3D.

Appendix B

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);
```

```
Unt_avg = zeros(1,n^3);
for j=1:20
    Un = Undata(j,:);
    Unt = fft(Un);
    Unt_avg = Unt_avg + Unt;
end
Unt_avg = abs(fftshift(Unt_avg))./20;
[M,I] = max(Unt_avg);

% Convert indices to frequencies
Kxmax = Kx(I);
Kymax = Ky(I);
Kzmax = Kz(I);

% [Kxmax,Kymax,Kzmax] = [-4.8171,-6.7021,-1.0472]

% Construct Gaussian filter centered at [Kxmax,Kymax,Kzmax]
tau = 0.4;
filter = exp(-tau*((Kx-Kxmax).^2+(Ky-Kymax).^2+(Kz-Kzmax).^2));
filter = reshape(filter,[1,n^3]);

% Initialize position variable, stores (x,y,z) coordinates of
% marble at each time point
pos = zeros(20,3);

% Loop through time points of given data
for j = 1:20
    % Filter frequencies from each time point in given data and
    % transform back to spatial domain to plot
    Un1 = Undata(j,:);
    Un1t = fft(Un1);
    Un1t = fftshift(Un1t);
```

```
    Un1tf = filter.*Un1t;
    Un1f = ifft(ifftshift(Un1tf));

    % Determine position of marble at each time
    [m,ii] = max(Un1f);
    xj = X(ii);
    yj = Y(ii);
    zj = Z(ii);
    pos(j,:) = [xj yj zj];

    isosurface(X,Y,Z,abs(reshape(Un1f,[n,n,n])),0.2)
    hold on
    view([-1,-1,0.5])
    title('Position of marble at each time')
    xlabel('x')
    ylabel('y')
    zlabel('z')
    axis([-20 20 -20 20 -20 20])
    grid on
    pause(0.2)
end
print -depsc m_pos.eps
hold off

% Plot path of marble
figure()
plot3(pos(:,1),pos(:,2),pos(:,3),'LineWidth',2)
title('Trajectory of marble')
xlabel('x')
ylabel('y')
zlabel('z')
axis([-20 20 -20 20 -20 20])
grid on
```

```
print -depsc m_traj.eps


% Determine final position of marble
final_pos = pos(end,:);
```