AMATH 482 HW 1

Liyuan Tang

January 2019

Abstract

In this assignment, we are going to locate and compute the trajectory of the marble based on the data obtained by the ultrasound for 20 different measurements that were taken in time. We are going to find the center frequency by averaging the spectrum and then use the Gaussian filter in 3D to find the data around the center frequency and denoise the data in order to determine the path of the marble. Finally, we can find the location of the marble at 20th data measurement.

1 Introduction

The original signal data is highly noisy. It contains 20 rows of data for 20 different measurements that were taken in time while each row represents 1 data measurement. First, I am going to find the center frequency by first transforming the data into the spectral domain and then averaging those spectral contents, where the noise parts should decrease to 0. Then I am able to find the center frequency by focusing on the maximum value to get the corresponding frequency. As a result, I can build a spectral filter around this frequency that removes the noise and unnecessary frequencies at each time. Once we generated the refined data, we can do the inverse Fourier transform to transform the data back. We can focus on the maximum value again, but this time we are going to get the corresponding x, y, z coordinate in order to find the path of the marble at each step. Finally, I plot the path of the marble and mark the location at 20th measurement.

2 Theoretical Background

2.1 Fourier Transform

The Fourier transform is an integral transform defined over the entire line $x \in [-\infty, \infty]$ which is defined as

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

where k is the wavenumber. However our domain is only a finite domain where $x \in$[-L, L], so the continuous eigenfunction expansion becomes a discrete sum of eigenfunctions and associated wavenumber.

The FFT algorithm improves time complexity from $O(N^2)$ to $O(N \log N)$. This algorithm finds the transform on a finite interval and implies the solutions have periodic boundary condition. Also, it shifts the data from $x \in$ [-L, 0] to [0, L] and $x \in$ [0, L] to [-L, 0]. Besides, it assumes working on a 2π periodic domain. So when we define the wavenumber, we need to rescale it by $2\pi/L$, where L is length of the entire finite domain.

In Matlab, we can simply use "fft()" to do the Fourier transform and use "fftn()" if the input data is a multidimensional array. Inversely, we can use "ifft()" and "ifftn()" to do the inverse fast Fourier transform.

2.2 Spectrum Averaging

The white-noise can be modeled by adding a normal distributed variable with 0 mean and 1 variance to each of the Fourier component of the spectrum. Thus, if we average the white-noise over many signals, the noise should add up to 0. As a result, we can eliminate the noise in the signal.

The main process is to transform the data into spectral domain by Fourier transform during each iteration and add them up. Then, we need to rearrange the data by shifting the 0 frequency component to the center since "fft" shifts the data. Finally, we need to normalize the shifted data by taking absolute value and divided by its maximum value among all its elements. By searching the index of maximum value, we are able to find the corresponding frequency, which is the center frequency.

2.3 Spectral Filtering

Spectral filtering is a method that allows people to extract information at some specific frequencies. It can remove some unwanted components from the signal and then improve the ability to detect the signal in the noisy field. In the lecture notes, we are introduced to one type of the filters, which is the Gaussian filter:

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

where τ measures the bandwidth of the filter, k is the wavenumber and k_0 is the center frequency. This filter will highly reduce those frequencies away from the center frequency k_0. We can just do the Fourier transform to the original data to transform the data into the spectral domain. And then use that result times the filter constructed around a specific center frequency to eliminate the undesired frequencies. Finally, we can do the inverse Fourier transform to transform that product back into time domain to get the signal we want.

And in our problem, since the data is in 3-D, we also need to change the filter to:

$$\mathcal{F} = e^{-\tau((K_x-x)^2+(K_y-y)^2+(K_z-z)^2)}, \tag{3}$$

where x, y, z represents the center frequency.

3 Algorithm Implementation and Development

First, we are going to do the Fourier transform on a finite domain so we need to discretize the domain. Since it has periodic boundary condition, we should discretize into $n + 1$ points and take the first n points. Then, we should define the wavenumber (frequency) k from 0 to $n/2 - 1$ and $-n/2$ to -1 due to a shifting during the Fourier transform. We need to resale the wavenumbers k by $2\pi/L$ since the FFT assumes 2π

periodic signals, where L is the length of the domain. We can use "fftshift" to shift k which moves the 0 frequency to the center for convenience. Then we can use shifted k and x to create 3-D grid coordinates in spatial domain as well as frequency domain through "meshgrid".

To average the spectrum, each time we need to take a row of the data, reshape it to the size of $64 \times 64 \times 64$ and take the Fourier transform by "fftn" instead of "ffn" since the data is 3-D. We need to add up all of them in order to calculate the average. Then, we need to take "fftshift" of the result to shift the zero-frequency component to the center, since the data shifts during the "fftn", normalize it by taking the absolute value and then divided by its maximum value. Note that the data here is 3-D, with a size $64 \times 64 \times 64$, so when finding the max value, we should use "max(max(max(abs(data))))".

In order to find the center frequency, we should find the maximum value in this matrix. Here I used "[C, I] = max(uave(:))", which uave(:) will change the 3-D matrix into a column vector and I is the index of the maximum value. I used this index to find the subscripts of the maximum in the 3-D matrix through "ind2sub" with an input size [64,64,64]. Then, I use these subscripts to plug into K_x, K_y and K_z to get the corresponding frequency which is the center frequency.

Once we have the center frequency, we can construct the Gaussian filter around it through Equation(3) by changing the x, y and z in the equation to the center frequency we found. Since K_x, K_y and K_z are created by k_s which generates from shifting k, we need to shift the filter again. Then, I create a 20×3 matrix, "path", to store the coordinates of the path each time. I used a for loop and each time I took a row of data, reshaped it to $64 \times 64 \times 64$, used "fftn" and then multiply it by the filter. After that, I transformed the matrix from spectral domain back to spatial domain by "ifftn". Just like what we did before, we should find the subscripts of the maximum value in the matrix, plug into X, Y and Z instead of K_x, K_y and K_z since we are now in the spatial domain. Thus, we can get the x, y and z coordinates of the marble and store them into a row in the matrix "path". After 20 iterations, each row in "path" represents the location of the marble at every single data measurement while the columns in "path" represent the x, y and z coordinates. As a result, we can use "plot3" to plot the path of the marble with 3 columns vector input. And the 20th row in the matrix represents the x, y and z coordinates of the marble at 20th data measurement.

4 Computational Results

4.1 Frequency signature (center frequency)

The code is in Appendix B, part 1. x_freq = 1.8850, y_freq = -1.0472, z_freq = 0

4.2 Path of the marble

Based on part 2 in the code, I generated and plot the path of the marble (Figure 1). It starts from the top and goes down like a counterclockwise spiral.

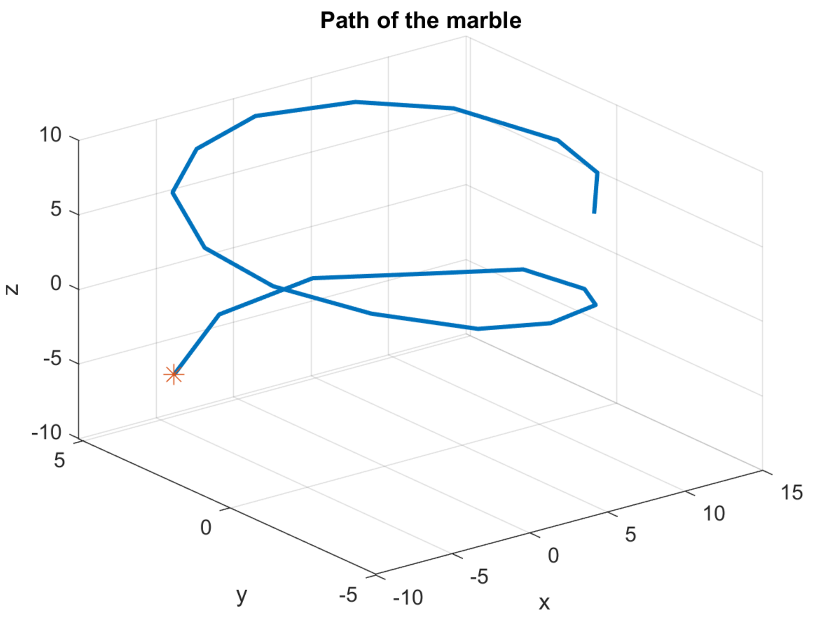

Figure 1: The path of the marble calculated by using filter around center frequency. The red asterisk represents the result of 20th data measurement

4.3 Coordinate of the marble at the 20th data measurement

The red star in Figure 1 represents the place that an intense acoustic wave should be focused to break the marble at the 20th data measurement. And its coordinate is (-5.6250, 4.2188, -6.0938) with respect to x, y and z.

5 Conclusion

In this assignment, we need to find the location of the marble in 3-D based on the given data of the ultrasound signal for 20 different measurements. I first found the center frequency based on the given data by using the spectrum averaging. Then, I created a filter around this frequency to denoise the data, found the path of the marble and its location at 20th data measurement and plot them. I used Fourier transform to transform the data from the spatial domain to frequency domain as the main tool to find the center frequency and the path of the marble. I found the index of the maximum value after summing up the transformed data of 20 iterations and found the corresponding center frequency. Then, I built a Gaussian filter around this frequency to denoise the data. I used inverse Fourier transform to transform the data back to spatial domain in order to find the path. As the same idea of finding the center frequency, I tried to get the index of the maximum value of the data and found the corresponding x, y and z coordinates which is the location of the marble at each iteration. I plotted the path in 3-D which looks like a counterclockwise spiral. The location of the marble at 20th data measurement is at the bottom of the path.

6 Appendix A

1. Uf = fftn(Un). "fftn" gives the multidimensional Fourier transform of an N-D array using fast Fourier transform. And "ifftn" is the multidimensional inverse fast Fourier transform for an N-D array.

2. ks = fftshift(k). "fftshift" will rearrange a Fourier transform by shifting the zero-frequency component to the center. "ifftshift" is its inverse.

3. [x,y,z] = ind2sub([n,n,n], I). It determines the equivalent subscript values corresponding to a single index into an array. While "sub2ind" converts subscripts to linear indices.

4. Un(:,:,:)=reshape(Undata(i,:),n,n,n). It will reshape the data input to the given size while the number of elements doesn't change.

5. isosurface(X, Y, Z, V, a). "isosurface" will compute and plot isosurface data from the volume data V at the isosurface value specified in a.

6. plot3(X, Y, Z). It plots line(s) in 3D space whose coordinates are element of X, Y and Z.

7 Appendix B

```
clear all; 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);

%% 1.

uave = zeros(n,n,n);
for i = 1:20
    Un(:,:,:)=reshape(Undata(i,:),n,n,n);
    Uf = fftn(Un);
    uave = uave + Uf;
end

% Normalize
uave = abs(fftshift(uave))./ max(max(max(abs(uave))));

% Find the index of the center frequency
[C, I] = max(uave(:));
[X_ind, Y_ind, Z_ind] = ind2sub([n,n,n], I);
% X_ind = 28; Y_ind = 42; Z_ind = 33;

% Find the corresponding frequency
x_freq = Kx(X_ind, Y_ind, Z_ind);
y_freq = Ky(X_ind, Y_ind, Z_ind);
z_freq = Kz(X_ind, Y_ind, Z_ind);

% Plot isosurface
figure(1)
isosurface(X, Y, Z, abs(uave), 0.4); grid on;
axis([-20 20 -20 20 -20 20]);


%% 2.

% Create a filter
filter = exp(-0.2.*(Kx - x_freq).^2 - 0.2.*(Ky - y_freq).^2 - ...
    0.2.*(Kz - z_freq).^2);
filter = fftshift(filter);

% Create an empty matrix to store the path
path = zeros(20,3);

for i = 1:20
    Un(:,:,:)=reshape(Undata(i,:),n,n,n);
    utn = fftn(Un);
    utnf = utn .* filter;
    unf = ifftn(utnf);

    % Find x, y, z coordinates
    [M, I_2] = max(unf(:));
    [X_path, Y_path, Z_path] = ind2sub([n,n,n], I_2);
    x_path = X(X_path, Y_path, Z_path);
    y_path = Y(X_path, Y_path, Z_path);
    z_path = Z(X_path, Y_path, Z_path);

    path(i, 1) = x_path;
    path(i, 2) = y_path;
    path(i, 3) = z_path;
end

% Plot the path
plot3(path(:,1), path(:,2), path(:,3), 'LineWidth', 2); grid on;
title('Path of the marble');
xlabel('x'); ylabel('y'); zlabel('z');
hold on;
%% 3.
% 20th data
plot3(path(20,1), path(20,2), path(20,3), '*', 'MarkerSize', 10);

hold off;
```