Using Fourier Transforms for an Ultrasound Problem
Seong Hyun Han
January 24, 2020

Abstract

In this problem, Fourier transformation and spectral filtering is used in order to help locate a marble that was swallowed by a dog. In the ultrasound data that was collected from the vet to locate the position of the marble in the dog's intestines, noise was introduced into the data by the fluid movement inside the intestines. To reduce and filter the unwanted noise in the data, the signal data is transformed to the frequency domain, averaged, and filtered. Ultimately, we hope to find the trajectory of the marble and use an intense acoustic wave to break it up.

Sec. I Introduction and Overview

The main motivation for this problem is to locate a marble that was swallowed by a dog named Fluffy. The marble is assumed to be located inside the intestines of the dog. Ultrasound was used by the vet to collect data relating to the spatial variation in a portion of the intestine where the marble is suspected to be located. However, due to the movement of the dog, the internal fluid movement inside the intestines introduce a lot of noise into the data. Therefore, in order to accurately locate and break up the marble by an intense acoustic wave, we must compute the trajectory of the marble.

The ultrasound data collected consists of 20 measurements that were taken in time, with each measurement correlating to a 3-dimential spatial image of the dog's intestine. In order to reduce the noise in the data due to intestinal fluid movement, the 20 collected data will be transformed into the frequency domain and then averaged. The averaged data can be used to find signature frequencies that correlate to the marble. Furthermore, a filter centered around the signature frequencies can be used to isolate signal correlating to the marble. The filtered frequency data can then be transformed back to the spatial domain, where the trajectory of the marble can be traced.

Sec. II. Theoretical Background

In order to solve this problem, two key mathematical concepts must be understood. The first concept is the Fourier Transform. The cardinal concept of the Fourier Transform is used to take a signal and break it up into sines and cosines of different frequencies. Viewing a signal as a function, $f(x)$, the Fourier Transform (Eqn. 1) can be defined as the integration from $x \in [-\infty, \infty]$ of a function multiplied by $e^{\pm ikx}$, which relates to Euler's formula (Eqn. 2), which describes something that oscillates [1].

$$F(k) = \frac{1}{2\pi}\int_{-\infty}^{\infty} e^{ikx} f(x)dx \qquad \text{(Eqn. 1)}$$

$$e^{\pm ix} = \cos(x) \pm isin(x) \qquad \text{(Eqn. 2)}$$

The transformed values, $F(k)$, can be viewed as the strength of each frequency value k. Moreover, the Inverse Fourier Transform (Eqn. 3) can be defined as the integration from $x \in [-\infty, \infty]$ of $F(k)$ by $e^{\pm ikx}$

$$f(k) = \frac{1}{2\pi}\int_{-\infty}^{\infty} e^{ikx} F(k) dk. \quad \text{(Eqn. 3)}$$

The Inverse Fourier Transform is used to bring the signal from the frequency domain back to its original domain, usually time or space. The above transforms allow us to process signals in either domains in order to obtain our desired signal. The Fourier Transform and Inverse Fourier Transform can be carried computationally using a method called the Fast Fourier Transform and the Inverse Fast Fourier Transform. This method was developed with a low operation count of $O(N\ logN)$ and takes the Fourier Transform from a finite range $x \in [-L,\ L]$. The low operation count could be achieved by discretizing the range into 2^n points. Furthermore, the integration Kernel, e^{ikx}, assumes that the solutions have periodic boundary conditions [1].

The second key mathematical concept is spectral filtering. One type of spectral filtering and the filtering method used for this problem is using a Gaussian filter. The general purpose of a filter is to attenuate unwanted signals or noise in our data. This process is usually achieved by multiplying a function by the signal of interest in the frequency domain, where the function is close to zero away from the desired signal frequency and close to one near the desired frequency. One type of function that exemplifies this behavior is the Gaussian function (Eqn. 4), which has a symmetric "bell curve" shape [1].

$$f(x) = e^{-\alpha x^2} \quad \text{(Eqn. 4)}$$

For multiplication of the above function in frequency to operate as a filter, the Gaussian function can also be rewritten as

$$F(k) = e^{-\tau(k-k_o)^2} \quad \text{(Eqn. 5)}$$

where τ defines the width or the attenuation of the filter and k_o equals the center frequency of the filter. This filter can also be applied in a three-dimensional frequency space by using the equation

$$F(k) = e^{-\tau[(k_x-k_{xo})^2 + (k_y-k_{yo})^2 + (k_z-k_{zo})^2]}\ [1]. \quad \text{(Eqn. 6)}$$

Sec. III. Algorithm Implementation and Development

This problem is solved in a MATLAB by using the methods mentioned above. Starting with **Section I** in **Appendix B. MATLAB codes**, the ultrasound data stored in **Testdata.mat** is loaded as a variable *Undata*. The file contains 20 measurements taken over time with each measurement containing 64^3 values correlating to a $64 \times 64 \times 64$ 3D array in the spatial domain. The x, y, z spatial domain is defined as -15 to 15 and discretized by dividing each axis into 64 points, defining a $64 \times 64 \times 64$ Cartesian grid in 3D space using the *meshgrid()* function. The time interval is then converted into a frequency interval with 64 Fourier modes. This frequency interval must be scaled by a factor of $\frac{2\pi}{2L}$, where $L = 15$, in order to rescale the frequency components (k) to the 2π domain as the Fast Fourier Transform in MATLAB assumes 2π periodic signals. The frequency components are then shifted using *fftshift()* to have the value zero at the center and stored in the variable *ks*. The variable *ks* is used to define a 3D grid for the frequency domain using *meshgrid()*.

In **Section II** in **Appendix B. MATLAB codes**, the average of the 20 collected data will be transformed into the frequency domain and then averaged to reduce noise in the data. To do so, the values for each measurement from the loaded data (*Undata*) is reshaped into a $64 \times 64 \times 64$ 3D array using the function *reshape()*. The reshaped measurements are then transformed into the Fourier domain using the *fftn()* function. The transformed measurements are then added all together and divided by the number of measurements (20) and stored to the variable *ut_ave*. The averaging of the data results in relative reduction in the unwanted noise and increases the magnitude of the frequencies that correlates to the marble. Next, the maximum value and its correlating index number of the absolute value of our zero centered *ut_ave* is found using the function *max()* (*fftshift()* of *ut_ave* was used since the frequency domain axes are zero centered). The central frequencies are needed to construct our filter. After the maximum value and its index value is found, the *ind2sub()* function was used to find the subscript values equivalent to the index value for a $64 \times 64 \times 64$ 3D array. These subscript values (*r, c, and p*) were then used to find the correlated central frequencies for the frequency domain axes (*Kx, Ky, and Kz*).

In **Section III** in **Appendix B. MATLAB codes,** a three-dimensional Gaussian filter is applied using Eqn. 6 with $\tau = 0.2$ as it was found to be sufficient. For each reshaped and transformed measurements, the Gaussian filter was multiplied and the Inverse Fourier Transform was taken of the filtered signal using the function *ifftn()*, bring the filtered signal back into the spatial domain. The marble position was then determined for each time measurement based on the largest value in the spatial domain. The largest value in the spatial domain was found through the *max()* function and its position through the *ind2sub()* function as previously done in the frequency domain. The x, y, z locations were then stored for each of the max values for each measurement for all 20 measurements. The 20 x, y, z locations were then plotted in 3D using the *plot3()* function to trace the trajectory of the marble.

Sec. IV. Computational Results

The central frequencies obtained from **Section II** in **Appendix B. MATLAB codes** could be approximately checked to see if they were correct by plotting the absolute values of *ut_ave* with values equal to 230. Thus, a surface can be seen with similar frequency ranges of the computed central frequencies (Figure 1).

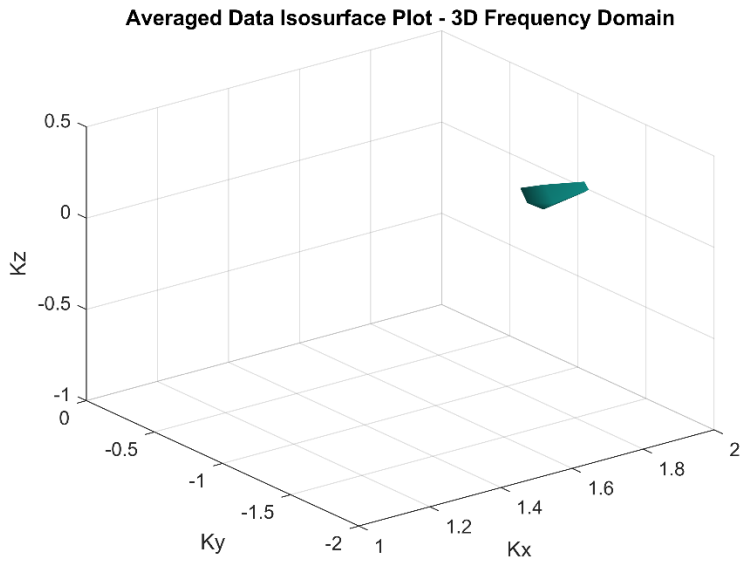

Figure 1. Isosurface plot of the averaged data in a three-dimensional frequency domain. The center frequencies $kxo = 1.8850, kyo = -1.0472, and\ kzo = 0.0000$.

The path of the marble can be traced by locating the highest value of the filtered data in the spatial domain for each measurement in time. As seen in Figure 2, the marble moves in a spiral/cylindrical trajectory.

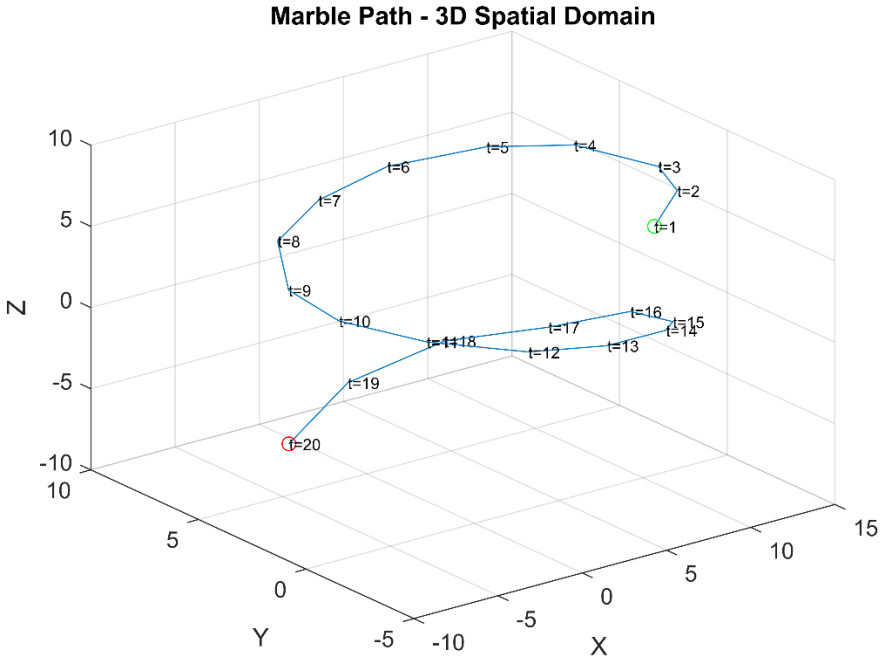

Figure 2. Marble path in the 3D spatial domain. The marble starts from the green circle (t=1), which correlates to the first measurement and ends at the red circle (t=20), which correlates to the last measurement.

Sec. V. Summary and Conclusions

All in all, as determined in **Section III** in **Appendix B. MATLAB codes,** the marble position at the 20th data measurement is at (x, y, z) location (-5.6250, 4.2188, -6.0938). Therefore, the vet should break up the marble by an intense acoustic wave at this location in order to save Fluffy's life. Moreover, through solving this problem, the importance and capabilities of Fourier Transforms could be exemplified along with how to apply spectral filters like the Gaussian filter.

Appendix A. MATLAB functions used and brief implementation explanation

- **fftshift():** shift the zero-frequency component to the center of the spectrum.
- **meshgrid():** replicates input grid vectors to create 3D rectangular grid coordinates.
- **reshape():** reshapes inputted elements into a nxnxn 3D array.
- **fftn():** takes the n-dimensional discrete Fourier Transform.
- **max():** finds the maximum value in array vector and its correlated index value.
- **ind2sub():** find the subscripts (row, column, page) from linear index.
- **isosurface():** computes isosurface geometry for a given isovalue in 3D.
- **plot3():** plots points in 3D space.

Appendix B. MATLAB codes

```
%% AMATH 482 Homework 1
% Seong Hyun Han
% 1/24/20

%% Section I
%Starter code with modification

clear; close all; clc;
load Testdata
[data_num, N] = size(Undata); % set data_num to the number of signals obtained
L=15; % computational spatial domain
n=64; % number of Fourier modes (2^n)
x2=linspace(-L,L,n+1); % define the spatial domain discretization
x=x2(1:n); % only consider the first n points for periodicity
y=x; z=x; % set spatial domain discretization for y and z axes
k=(2*pi/(2*L))*[0:(n/2-1) -n/2:-1]; % frequency components of FFT rescaled to 2pi domain
ks=fftshift(k); % Fourier components with zero at the center

[X,Y,Z]=meshgrid(x,y,z); % Define Cartesian grid in 3D space
[Kx,Ky,Kz]=meshgrid(ks,ks,ks); % Define 3D grid for the frequency domain

%% Section II
% (Problem 1) Average the 20 3D spatial data in frequency to reduce noise and find the central
frequencies
ut_ave = zeros(n,n,n);
for j=1:data_num
Un(:,:,:)=reshape(Undata(j,:),n,n,n); % reshape Undata array into a 3D nxnxn array
ut_ave = ut_ave + fftn(Un);
end

ut_ave = ut_ave/data_num; % average of the Fourier transformed signal
ut_ave_fft = fftshift(ut_ave); % fftshifted ut_ave
ut_ave_abs = abs(ut_ave); % absolute value of ut_ave

% Find the maximum value in ut_ave_fft and its corresponding index value
[M,I] = max(abs(ut_ave_fft(:)));
% Find subscript values equivalent to the found index (I) for a nxnxn 3D array
[r,c,p] = ind2sub(size(ut_ave_fft),I);

% using subscripts to find the correlated central frequencies
kxo = Kx(r,c,p); % center-frequency for Kx-axis
```

```
kyo = Ky(r,c,p); % center-frequency for Ky-axis
kzo = Kz(r,c,p); % center-frequency for Kz-axis

% plot of the absoluate values of ut_ave with values equal to 230
figure(1)
isosurface(Kx,Ky,Kz,fftshift(ut_ave_abs),230)
axis([1 2 -2 0 -1 0.5]), grid on, drawnow
title('Averaged Data Isosurface Plot - 3D Frequency Domain')
xlabel('Kx')
ylabel('Ky')
zlabel('Kz')
saveas(figure(1),'AMATH482_fig1.png');
print(gcf,'AMATH482_fig1.png','-dpng','-r600');

%% Section III
tau = 0.2; % filter bandwidth
% 3D Gaussian filter with all filter bandwidths equal to tau
filter=exp(-tau*(((Kx-kxo).^2)+((Ky-kyo).^2)+((Kz-kzo).^2)));

% Define variables to store 20 x,y,z locations
x_loc = zeros(1,data_num);
y_loc = zeros(1,data_num);
z_loc = zeros(1,data_num);

% Determine the marble path
for j=1:data_num
Un(:,:,:)=reshape(Undata(j,:),n,n,n);
ut_filter = filter.*fftshift(fftn(Un)); % filter signal in frequency
u_filter = abs(ifftn(ut_filter)); % inverse fft of filtered signal into spatial domain

% Determine the marble position based on largest signal in spatial domain
[Max,Idx] = max(u_filter(:));
[row,col,pag] = ind2sub(size(abs(ut_ave)),Idx);

% using subscripts to find the correlated marble locations
x_loc(1,j) = X(row,col,pag);
y_loc(1,j) = Y(row,col,pag);
z_loc(1,j) = Z(row,col,pag);

end

labels =
{'t=1','t=2','t=3','t=4','t=5','t=6','t=7','t=8','t=9','t=10','t=11','t=12','t=13','t=14','t=15',
't=16','t=17','t=18','t=19','t=20'};

% plot marble path
figure(2)
plot3(x_loc,y_loc,z_loc), grid on
hold on
plot3(x_loc(1,1), y_loc(1,1), z_loc(1,1),'go')
plot3(x_loc(1,20), y_loc(1,20), z_loc(1,20),'ro')
title('Marble Path - 3D Spatial Domain')
xlabel('X')
ylabel('Y')
zlabel('Z')
text(x_loc,y_loc,z_loc,labels,'FontSize',7)
saveas(figure(2),'AMATH482_fig2.png');
print(gcf,'AMATH482_fig2.png','-dpng','-r600');

%% Section IV
% marble position at the 20th data measurement
breakup_loc = [x_loc(1,20), y_loc(1,20), z_loc(1,20)];
```

References

1. Kutz, Jose Nathan. *Data-Driven Modeling & Scientific Computation: Methods for Complex Systems & Big Data*. Oxford University Press, 2013.