Filtering ultra sound signal with Fourier Transformation

Author: Kuan-Wei Lee

Abstract: In this project, I filter the ultra sound signal to locate a marble inside a dog's intestine. The ultra sound signal was stored in a 20*262144 vector that include the ultra sound signals in the 3D space taken at 20 different period of time. The 20*262144 vector is reshaped to 20*64*64*64 matrix and the Fourier transform is applied to the matrix. I found the frequency of the signal produced by the marble by averaging the Fourier transformation of 20 measurements and create a Gaussian filter center at the frequency of the marble to denoise the data and locate the marble.

Introduction

In this project, I am going to analyze a noisy ultra sound signal and locate a marble inside a dog's intestine. When I plot the data, it looks very noisy and it is impossible to pick out the signal of marble from the noise. To denoise the ultra sound signal, I used the Fourier transformation (Figure 1). The Fourier transformation allows me to transform the data from position space to the frequency space. Since the signal from marble has a unique frequency signature that is different from the frequency of the noise, we can filter out the noise and pick out the signal from marble if we know the frequency of the marble signal. To find out the frequency of the marble signal and filter out the noise, I use the Fast Fourier Transform tools in MATLAB to perform Fourier transform and Inverse Fourier Transform. The concepts of Fourier transform and the implementation of the algorithm will be introduced in the following sections.

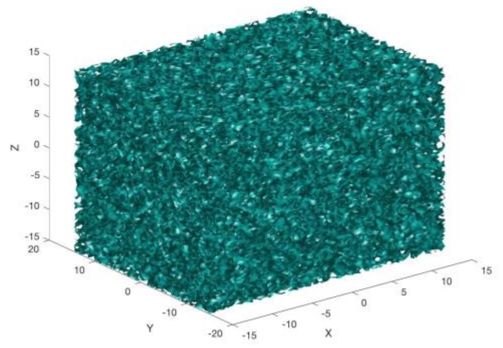

Figure 1. The plot of unfiltered ultrasound signal. It is very noisy and the location of marble cannot be identified.

Concept of Fourier transformation

Before I talk about the algorithm developed for this project, I would like to go over the concepts of the Fourier transformation and how it works.

The Fourier transform is defined as follow:

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

Equation (1) is the Fourier transform and equation (2) is the inverse Fourier transform.
You might have noticed that the Fourier transform (FT) and inverse Fourier transform(IFT) are defined as integrals from negative infinity to positive infinity. It is impossible to do FT and IFT with integration from negative infinity to positive infinity on the computer since we can not give computers infinitely small steps dx, dk and infinitely large boundary. Everything has to be finite. Therefore, we are going to perform Discrete Fourier transform, or more precisely the Fast Fourier Transform, with MATLAB. FFT is basically a faster version of DFT. DFT transform a sequence of N complex numbers $\{x_0, x_1, \ldots, x_{n-1}\}$ to another sequence of complex numbers $\{X_0, X_1, \ldots, X_{n-1}\}$. The mathematical form of the DFT are as follow:

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-\frac{i2\pi}{N}kn} \quad (3)$$

The set $\{X_k\}$ contains N numbers and it takes N operations to get each term. Therefore, the computation complexity of DFT is $O(N_2)$. DFT gets very slow as N gets very large. To speed up the computation, we can use FFT. FFT speeds up the computation by dividing each term of X_k into two sub-terms. One term is the odd term while the other is the even term. The mathematical form of the FFT is the following:

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-\frac{i2\pi}{N}kn} \quad (4)$$
$$= \sum_{m=0}^{N/2-1} x_{2m} \cdot e^{-\frac{i2\pi}{N}k(2m)} + \sum_{m=0}^{N/2-1} x_{2m+1} \cdot e^{-\frac{i2\pi}{N}k(2m+1)} \quad (5)$$

As you can see, each term of X_k is split into two smaller DFT terms. Those smaller DFT terms can be split into even smaller DFT terms as long as N/2 is even. FFT speeds up the DFT by splitting bigger DFT problem into smaller DFT problems until the DFT problem is too small to be split. The computation complexity of FFT is O(N ln(N)), which makes a huge difference if N is very large. The other reason why we choose FFT to perform Fourier transformation on the signal besides faster computational speed is that there is a FFT algorithm in MATLAB that we can use.

Algorithm Implementation and computation results

The Fourier transformation is suitable to filter out the noise and get the marble signal from noise data because the majority of the noise are white noise. When we perform Fourier transform on the ultra sound signal, we transform the data from position space to frequency space. If we know the frequency of the signal from marble, we can create a filter around the marble signal frequency to filter out white noise. Although we don't know the frequency of the marble signal in the beginning, we can find it out by averaging the Fourier transformed ultra sound signal taken at 20 different times. The frequency of the marble signal is not time dependent while the frequency of the white noise fluctuates and tend to have zero value in frequency space on average. If we average the Fourier transform of the ultra sound signal at 20 different times, the magnitude of the marble signal in frequency space will be the largest since the magnitude of the white noise in frequency space will be close to zero. Therefore, the strategy of locating the frequency of the marble signal is to average 20 Fourier transformation of the ultra sound signal. Since the data at each time frame is a 3 dimensional matrix, we need to perform 3D FFT.

Before I perform 3D FFT on the ultra sound signal, I reshape the data so it is easier to do Fourier transformation on it and plot. The data contains the strength of the ultra sound signal on a given x,y,z coordinate. X,Y, and Z ranges from -15 to 15 and there are 64 points on each dimension. I reshape the 20*262144 matrix to 20*64*64*64 matrix. After the data is reshaped, I then perform Fourier transformation on the ultra sound signal taken at 20 different times, which gives me 20 Fourier transformed ultra sound signal. I averaged all 20 Fourier transformed data because the averaged value of the white noise tends to be zero in frequency space. Since the FFT is 3D in our case, the plotting of Fourier transformed data is different from the plotting for 1D FFT. The frequency space of 3D FFT has frequencies in three different dimension (kx,ky,kz). To plot the matrix of 3D FFT, I first normalized the magnitude of the frequency to [0,1] by dividing all the values in the matrix by the maximum value in the matrix. After the normalization, I use MATLAB function "isosurface" to plot the data in frequency space. The function "isosurface" can plot all the points in 3D space that has the same "isovalue". To put it simply, "isosurface" will produce a 3D surface plot that contains all the points in 3D space that have the same function value if we interpret our matrix as values of a function at different points in space. Since I expect that the frequency with the greatest magnitude is the frequency of the marble signal, I set the isovalue to be 0.95 and plot the isosurface (Figure 2). The reason for choosing 0.95 instead of 1 for the isovalue is because there are too few points in the frequency space that have isovalue of 1.

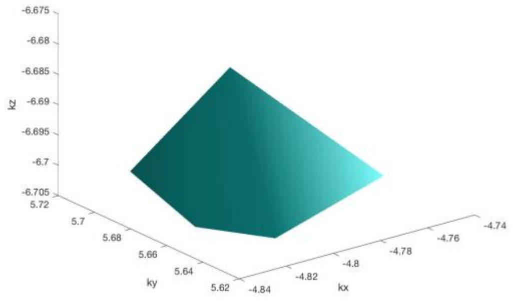

Figure 2. The isosurface consists of all the points with value 0.95.

The isosurface in figure 2 is a three dimensional surface that has some width in kx, ky, and kz dimension. kx,ky, and kz are the coordinates in frequency space and all the points on the isosurface are the points that have high strength. The kx, ky, kz values of the points on the surface basically suggests the frequency of the marble signal. For simplicity, I estimate the marble signal frequency by averaging kx, ky, and kz. In the end, I will get the frequency of marble signal as three values $(\overline{k_x}, \overline{k_y}, \overline{k_z})$. After finding the frequency of the marble signal, we can then create a Gaussian filter centered at the marble signal frequency.
The mathematical form of the filter is :
$e^{-\tau(k_x-(\overline{k_x})^2} \cdot e^{-\tau(k_y-(\overline{k_y})^2} \cdot e^{-\tau(k_z-(\overline{k_z})^2}, \tau\ is\ the\ width\ of\ the\ filter\ and\ is\ set\ to\ be\ 0.2$

By multiplying this filter function with the Fourier transformed data, we can filter out the frequency that is far away from $(\overline{k_x}, \overline{k_y}, \overline{k_z})$. In another words, most of the white noise will be filtered out because the frequency of the white noise are not close to $(\overline{k_x}, \overline{k_y}, \overline{k_z})$. After the noise are filtered out, I then apply inverse Fourier transform to transform the data from frequency space to position space. Since most of the noise were removed, we can locate the marble by plotting the data. The plotting of the data is done with the MATLAB function “isosurface”. I plot the isosurface with isovalue 0.25 because there are decent amounts of points that have 0.25 as their function values. When I plot the isosurface of the ultra sound data at 20 different time, spherical objects that look like a marble show up (Figure 3).

The graph in figure 3 suggests that the locations of the marble at different point in time has been successfully located. To know where the intense acoustic wave should focus at 20th data measurement, I extracted all the points of the isosurface plotted with the 20th data measurement and find the average x,y,z values of all the points. It turns out that the average x,y,z of the 20th isosurface is (-5.3831,4.2533,-6.0622). Therefore, the intense acoustic wave should be focused at (x,y,z) = (-5.3831,4.2533,-6.0622) at the 20th data measurement.

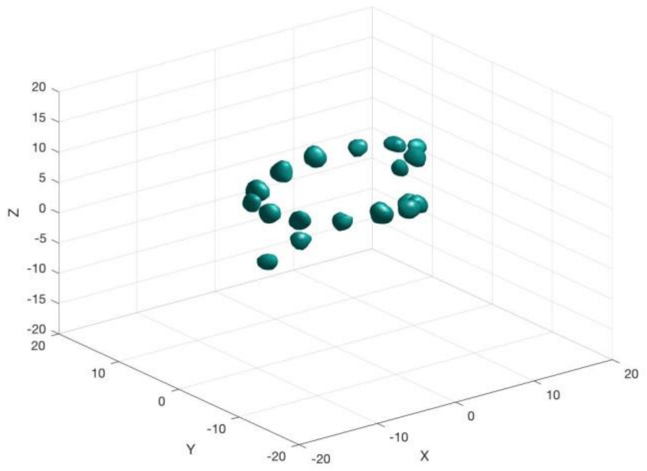

Figure 3. The isosurface of the ultra sound data at 20 different time. The isosurface at each time looks like a marble and we can see the marble has a spiral trajectory.

Conclusion

Fourier transformation is a powerful tool for analyzing noisy data. It can not only be used in analyzing periodic audio signal but can also be applied to analyzing images and ultra sound signals. The ability of Fourier transformation to break down the signals into different frequency signatures allow us to filter out certain parts of the signal and pick out the signal that we are interested. Even if we don't know the frequency of the signal in interest at first, we can find out the frequency by averaging the Fourier transform of multiple measurements assume that other signal besides the signal in interest are white noise. With Fast Fourier Transform tool box in MATLAB, I was able to denoise the signal and locate the marble at different points in time.

Appendix A

MATLAB functions used in the project:

```
[X,Y,Z]=meshgrid(x,y,z);
[Kx,Ky,Kz]=meshgrid(ks,ks,ks);
```

The function meshgrid is to create "coordinates" that matches the values in the matrix. The meshgrid is made in a way that the dimension of the meshgrid matches the dimension of the matrix that we want to plot.

```
k=(2*pi/(2*L))*[0:(n/2-1) -n/2:-1]; ks=fftshift(k);
fftn(Un(:,:,:),[n,n,n]);
```

The two lines of code above is for performing FFT on the data. It starts with making a vector k that represents the frequency. The factor `(2*pi/(2*L))` is there because fft function in MATLAB assumes that we work on a 2π period. fftshift(k) will reorganize the order of k so that the plot will come out accurately.

```
isosurface(Kx,Ky,Kz,Un_ftavg3d,0.95)
```

The function isosurface is for plotting a 3D matrix. The code above plots the Fourier transformed ultra sound signal in frequency space. The input for this function are coordinates, the matrix we want to plot, and the isovalue. As I explained in the previous section, isosurface picks out the indices in the matrix contains the isovalue and plot it on a 3D graph.

Appendix B

```
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);
Un_ft = zeros(length(Undata(:,1)),n,n,n);
for i = 1:length(Undata(:,1))
    Un(:,:,:) = reshape(Undata(i,:),n,n,n);
    Un_ft(i,:,:,:) = fftn(Un(:,:,:),[n,n,n]);
end

isosurface(X,Y,Z,abs(Un),0.4)
xlabel('X')
ylabel('Y')
zlabel('Z')
Un_ftavg = mean(Un_ft,1);
Un_ftavg3d(:,:,:) = Un_ftavg(1,:,:,:);
Un_ftavg3d = abs(Un_ftavg3d);
Un_ftavg3d = Un_ftavg3d./max(Un_ftavg3d,[],'all');
fv_fft = isosurface(Kx,Ky,Kz,Un_ftavg3d,0.95);
isosurface(Kx,Ky,Kz,Un_ftavg3d,0.95)
xlabel('kx')
ylabel('ky')
zlabel('kz')

kx = fv_fft.vertices(:,1);
ky = fv_fft.vertices(:,2);
kz = fv_fft.vertices(:,3);
tau = 0.2;
x_filt = (min(kx)+max(kx))/2;
y_filt = (min(ky)+max(ky))/2;
z_filt = (min(kz)+max(kz))/2;
filter = exp(-1*tau*(Kx-(x_filt)).^2).*exp(-1*tau*(Ky-
y_filt).^2).*exp(-1*tau*(Kz-(z_filt)).^2);
Un_new = zeros(20,n,n,n);
for m = 1:20
    Un_ft1(:,:,:) = Un_ft(m,:,:,:);
    Un_ft1new = Un_ft1.*filter;
    Un_new(m,:,:,:) = ifftn(Un_ft1new,[n,n,n]);

end
```

```
figure(2)
for i = 1:20
    Un_s(:,:,:) = Un_new(i,:,:,:);
    isosurface(X,Y,Z,abs(Un_s(:,:,:)),0.25)
    axis([-20 20 -20 20 -20 20]), grid on, hold on
    xlabel('X')
    ylabel('Y')
    zlabel('Z')
    pause(0.1)
end
fv = isosurface(X,Y,Z,abs(Un_s(:,:,:)),0.25);

x_marb = mean(fv.vertices(:,1));
y_marb = mean(fv.vertices(:,2));
z_marb = mean(fv.vertices(:,3));
```