AMATH482: An Ultrasound Problem Writeup

Shuying Zhang
email: shuying3@uw.com

Abstract

This project simulates a real life situation that my dog called fluffy swallowed a marble and therefore, took an ultrasound check. The ultrasound result, however, was noisy somehow. We applied Fast Fourier Transform in MATLAB to average and filter the data in time and frequency resolution, then computed the true trajectory of the marble and found its final location in order to break it up.

1 Introduction and Overview

Fourier Transform, by summing cosines and sines to represent functions, is known as a powerful tool in solving biology and physical problems. Fast Fourier Transform (FFT) is an algorithm that has complexity O(NlogN) to perform Fourier Transform. This project will look into the applications of FFT in signal detection. Basically, it allows us to average and filter the noisy data and determine the true signals.

In the problem given, the noisy data contains 20 rows got in 20 measurements. We will first average the data in 20 realizations to determine the frequency signature, then filter around center to find the path.

There are five sections in this report. The introduction and overview section gives a brief description of the problem and the report. The theoretical part provides background knowledge for FFT and its applications. In algorithm implementation and development, we will formulate the problem into three steps, and apply FFT to it in MATLAB. The computational results shows the output and plots of the denoised data. A summary is concluded in summary and conclusions. All the MATLAB code and related MATLAB commands are in appendix.

2 Theoretical Background

We will go over what is Fourier Transform and how will FFT algorithm help us in signal detection. Some descriptions are from the class notes [1].

2.1 Fourier Transform

The *Fourier Tranform* and its inverse are defined as integrals over $[-\infty, \infty]$:

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

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

In real world, the function domain is always finite, say $[-L, L]$, so the kernel of the function e^{-ikx} has an oscillatory behavior. We call k wavenumbers, on the finite domain $[-L, L]$, Fourier Tranform is the discrete sum of eigenfunctions and coresponding k.

Besides, Fourier Transform also has one nice derivative property. It shows that the derivatives of Fourier Transform is related to the transform itself.

$$\widehat{f^{(n)}} = (ik)^n \widehat{f} \tag{3}$$

2.2 FFT and its advantages

FFT is an algorithm that runs faster than linear solvers. It is efficient as the runtime is O(NlogN) compared with Gaussian elimination ($O(N^3)$) and LU Decomposition ($O(N^2)$). With the properties of Fourier Transform, the solutions of FFT has periodic boundary conditions on the finite domain[-L,L]. To make the operation count O(NlogN), we will discretize the domain into 2^n, like 2, 4, 8, 16, 32, 64, 128 etc. points. Besides, FFT is also very accurate.

In MATLAB, *fftn*, *ifftn*, *fftshift* are commands related to Fast Fourier transform. More information is in appendix A.

2.3 FFT applications

FFT has many applications. Signal detection is one involved in this project. It mainly has two process: averaging and filtering.

Averaging In averaging process, we fit the white noise into standard normal distribution. The key fact is the zero mean. This tells us when adding up all the signals, the white noise should have sum zero. In this way, we are able to remove the white noise and extract the center frequency.

Filtering Filtering helps finding the signal hidden in the noisy data. In one realization, as long as we know where the center frequency is, we apply a filter there. The idea behind this is that the filter only has one hump around the center frequency and rest of the values are nearly zero. When multiplying the noisy data with the filter, the undesired frequencies located at other places got filtered out (become zero), and only the frequency near center frequency is left. By multiplying, we mean the dot product. Later, We can set a detection threshold for radar to determine whether a signal is recognized.

One commonly used filter is the Gaussian filter:

$$\mathcal{F}(k) = \exp\left(-\tau\left(k - k_0\right)^2\right) \tag{4}$$

where τ is the parameter of the filter bandwidth and k_0 is the location of the center frequency.

3 Algorithm Implementation and Development

3.1 Starting the problem

Preparation To perform FFT in MATLAB, we can just use the commands mentioned above, but before that, we need to discrete the domain [-L, L] into 2^n points (2,4,8,16,32,64...) and bring it to Fourier domain.

In this project, we set the Fourier mode n to be 64. First use *linspace* to divide domain [-L, L]. In this way, we get periodic boudaries. We do this in three dimensions and use *meshgrid* to create a 3-D space. Then, rescale the domain by multiplying $\frac{2*pi}{2*L}$ to make sure everything is on a periodic domain. What's more, we also need to shift k to center using *fftshift*. This shift is necessary for plotting all data in Fourier space. We use the *isosurface* command to take a look at the noise data.

3.2 Averaging the spectrum

Averaging First, we will perform FFT on all the noisy data. Now we add up 20 measurements (realizations) and find its sum. Doing this helps us averaging out noise in data and determines the center frequency. Similarly, we use *isosurface* to plot the result after averaging.

Determine the center frequency After averaging, although we may see the center frequency clearly through visualizing the data, we still haven't calculated the true locations of marble. Now we need to get its accurate location to apply a filter around it.

To do this, we use the MATLAB commands *ind3sub*, *find* and *reshape*. *Find* command returns the linear index of center frequency if all the data is an array or a vector. *reshape*

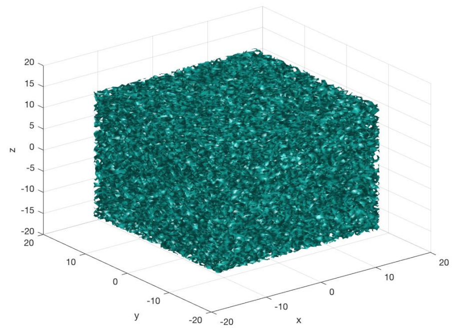

Figure 1: Noisy data at Measurement 20 with isovalue = 0.4

helps us reshape the 3-D data into a vector. In the end, we introduce *ind2sub*. By setting the size to N by N by N. *ind3sub* will return the subscript for that linear index if everything is in size N by N by N. This is exactly what we need. In this way, center frequency is successfully determined.

3.3 Filtering the data

Filter In this problem, we use the Gaussian filter in Equation 4 as we discussed in class but with some modifications since the problem is in 3-D and the center frequency has moved from (0,0,0) to the current center. Hence k_0 should change accordingly. We set $\tau = 0.2$.

Filtering each realization We now filter each realization by multiplying each measurement data by filter (dot product). Then, determine where the marble is at in each measurement. To do this, we use the same technique as we used above in locating the frequency center. One thing to point out, in this step, we perform inverse Fourier Transform on data to reveal the true location in space. After getting all the coordinates of the marble in 20 measurement, we plot the trajectory using *plot3*. The last location will be where we focus the acoustic wave.

4 Computational Results

At first, the data is very noisy as in Figure 1. Figure 1 displays the noisy data in space at measurement 20 with isovalue 0.4. Notice when plotting, we take the absolute value of the data.

After averaging, the data is much less noisy and clearly shows the frequency signature in Figure 2.

Then the filtering step gives is the location of the marble in 20 measurements in space. The locations are reflected in Figure 3.

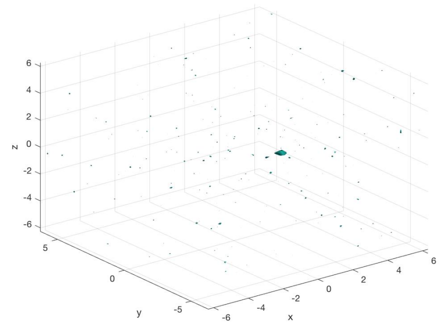

Figure 2: Frequency signature after averaging with isovalue 0.5

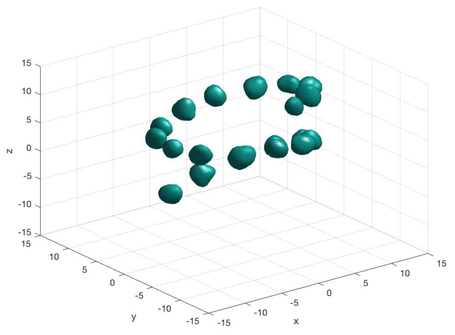

Figure 3: The true location of marble after denoising at isovalue=0.7

Finally, fter applying FFT to denoising the data with averaging and filtering. We find the true location of marble in space. The trajectory is in Figure 4.

We also get the position of marble in the last measurement. That is, (-5.6250, 4.2188, -6.0938).

Notice, when plotting, we take the absolute value of the data and normalize it to make the figures looks more reasonable.

5 Summary and Conclusions

In this project, we apply Fast Fourier Transform algorithm and solve a real world problem. This is one application of FFT in signal detection. The process of averaging and filtering enables us to denoise the data and capture the signal we are looking for. We successfully find the trajectory of the marble and gives its last location. Now we can focus the acoustic wave to breakup the marble. Fluffy got saved! I will take care of Fluffy better in the future to prevent this situation from happening again.

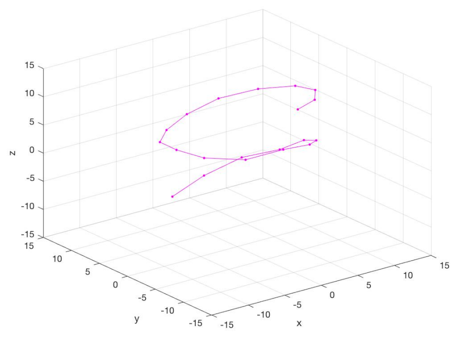

Figure 4: The trajectory of the marble in Fluffy's intestine

References

[1] Part3 : Computation Methods for Data Analysis. Data-Driven Modeling & Scientific Computation: Methods for Complex Systems Big Data, by J. Nathan Kutz, OUP Oxford, 2013, available at `http://faculty.washington.edu/kutz/582.pdf`.

A MATLAB commands

Documentation for those commands can be found in MATLAB using `help command_name`.

A.1 Data Modification

meshgrid(x,y,z) makes three vectors into grid (3-D).

linspace(-L,L,n+1) linearly divides [-L,L] into n equally-spaced parts, returns values of n points in between.

reshape(V,x,y) reshapes data V into x by y size. The number of elements remains unchanged.

find(X==x) returns the linear index of the element in Array X whose value is x.

ind2sub(size(V), INDEX) determines the subscripts in a size V data of elements with linear indices.

A.2 FFT related

fftn(V) uses FFT to do fourier transform on n-dim data V. n can be replaced into any dimensions, like fft, fft2.

ifftn(V) does inverse fourier transfrom over n-dim data V. Also see ifft, ifft2.

fftshift(V) shifts the FFT data V to center.

A.3 Plotting

isosurface(X,Y,Z,V,isovalue) computes isosurface geometry for data V at certain isovalue passed in.

plot3(X,Y,Z) plots 3-dim data.

B MATLAB code

```
clear all; close all; clc;

%%Load Data
load Testdata

%%Initial set up
L=15; % spatial domain
n=64; % Fourier modes
x2=linspace(-L,L,n+1); x=x2(1:n); y=x; z=x; %periodic boundaries
k=(2*pi/(2*L))*[0:(n/2-1) -n/2:-1]; %rescale the domain to fit FFT 2pi
ks=fftshift(k); %shift fft(k) in advance, k frequency components
[X,Y,Z]=meshgrid(x,y,z);
[Kx,Ky,Kz]=meshgrid(ks,ks,ks);

%start the problem
%Visulize the raw data
for j=1:20
Un(:,:,:)=reshape(Undata(j,:),n,n,n);
close all,isosurface(X,Y,Z,abs(Un),0.4)
axis([-20 20 -20 20 -20 20]), grid on, drawnow
pause(1)
end

%%Averaging
Uave = zeros(n,n,n); %Uave store avg data after adding up all measurements
for j=1:20
   Un(:,:,:)=reshape(Undata(j,:),n,n,n); %un - noise data in x,y,z at measurement
       j
   Utn = fftn(Un); %Utn- noise data in fourier
   Uave = Uave + Utn;
end
Uavep = reshape(Uave,1,n^3); %reshape the 3D data to a vector to normlize
                          %p - variable for visulization use only

isosurface(Kx,Ky,Kz,fftshift(abs(Uave)/max(abs(Uavep))),0.6)
axis([-2*pi 2*pi -2*pi 2*pi -2*pi 2*pi]), grid on, drawnow

%%Filtering
%find the max in Uave and return its indices
maxUavg = max(abs(Uavep)); %max in Uave: center of the hump

%find center frequency location
[kxc,kyc,kzc] = ind2sub(size(Uave), find(abs(Uavep) == maxUavg)); %returns
    coordinates of max in Uave
Kxc = Kx(kxc,kyc,kzc);
```

```
Kyc = Ky(kxc,kyc,kzc);
Kzc = Kz(kxc,kyc,kzc);

location = zeros(20,3);
%apply Gaussian filter around center at each measurements
filter = exp(-0.2*((Kx-Kxc).^2 + (Ky-Kyc).^2 + (Kz-Kzc).^2));
for j=1:20
   Un(:,:,:)=reshape(Undata(j,:),n,n,n);
   Utn = fftn(Un);
   Utnf = Utn.*filter; %Utnf filtered noise data in fourier
   Unf = ifftn(Utnf); %Unf inverse fft the filtered data: out of fourier
   Unfv = reshape(Unf,1,n^3); %Unfv Unf in vector
   maxUnf = max(abs(Unfv)); %find maximum of filter data

   %apply the similar process above to find the coordinates of the maxUnf
   [mxfi,myfi,mzfi] = ind2sub(size(Unf), find(abs(Unfv) == maxUnf)); %returns
      coordinates of max in Unf
    mxf = X(mxfi,myfi,mzfi);
    myf = Y(mxfi,myfi,mzfi);
    mzf = Z(mxfi,myfi,mzfi);

   %plot marble locations in 20 measurements
   isosurface(X,Y,Z,abs(Unf)/maxUnf,0.7);
   axis([-L L -L L -L L]), grid on, drawnow
   xlabel('x'); ylabel('y'); zlabel('z');

    %record the locations of the maxValue
   location(j,:) = [mxf,myf,mzf];
end

%plot trajectory
plot3(location(:,1),location(:,2),location(:,3),'m.-')
axis([-L L -L L -L L]), grid on, drawnow
xlabel('x'); ylabel('y'); zlabel('z');
finalPosition = location(20,:); %-5.6250 4.2188 -6.0938
```