Homework 1: An ultrasound problem

Jiayi Wang

Abstract

In this report, we will see how Fast Fourier Transform (FFT) and Gaussian filter are being used in real life. In this problem set, Fourier transform will be used to convert a signal from time domain to the frequency domain. Gaussian filter will be used to filter out the noise from data. By using both techniques, we are able to determine the frequency signature, the path of an object in a noisy field, and its location at a given time.

Introduction and Overview

As a specific application for spectral analysis in real life, suppose we have a dog swallowed a marble. The vet suspects that it has now worked its way into the intestines. Using ultrasound, data is obtained concerning the spatial variations in a small area of the intestines where the marble is suspected to be. 20 rows of data for 20 different measurements that were taken in time were stored in the given file: **Testdata.mat**. However, the data we obtained are highly noisy. In order to save the dog, we would need to locate and compute the trajectory of the marble so we can use an intense acoustic wave to break up the marble.

In the following sections, we will illustrate how we use FFT to analyze and manipulate ultrasound data in the frequency domain and Gaussian filtering to help significantly improve the ability to detect the location of the marble in the noisy field in order to save the dog.

Theoretical Background

- **Fourier Transform**

  The Fourier Transform is an integral transform defined over the entire line $x \in [-\infty, \infty]$. It converts a signal from its time domain to a representation in the frequency domain or vice versa. The Fourier Transform and its inverse are defined as

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

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

- *Fast Fourier Transform*

  In the mid 1960s, Cooley and Tukey developed the algorithm now known as fast Fourier Transform (FFT). FFT is an algorithm for computing the discrete Fourier Transform. It has low operation count: $O(N\,log\,N)$ and finds the transform on an interval $x \in [-L, L]$ with the solutions on this interval have periodic boundary condition. In order to lower the operation count to $O(N\,log\,N)$, the key is to discretize the range $x \in [-L, L]$ into 2^n points.[1]In Matlab, the following commands are often used when doing FFT: fft, $ifft$, $fftshift$, $ifftshift$, $fftn$, $ifftn$ where fft, $ifft$ are for $1-D$ and $fftn$, $ifftn$ are for N-D, that is multidimension. The detailed usage of these commands will be included in Appendix A at the end.

- **Filter**

  In real life, the data we obtain is often noisy due to many inevitable factors. Filtering then plays an important role in data processing; it helps improve the ability to detect the signal which is buried in a noisy field by removing undesired frequencies in the signal. There is a wide range of filters and Gaussian filter is one of them.

  - *Gaussian filter*

    Gaussian filter is a commonly used filter which helps remove noise from signal. It has the following form:

$$F(k) = exp(-\tau(k - k_0)^2) \quad (3)$$

    where τ is the width of the filter, k is the wavenumber, and k_0 is the frequency of the middle of the Gaussian (center-frequency).[2] By applying the

1 Page 311-313 from the course notes.
2 Page 323 from the course notes.

filter, frequencies that are farther away from the center frequency will be attenuated.

- **Averaging the spectral**
  Denoising the system by averaging is a technique people often used in stationary signals because white-noise should, on average, add up to zero in this case. Ideally, if more data is collected a better average signal is produced.[3]

Algorithm Implementation and Development

1. Load the data file **Testdat.mat** to Matlab.
2. Set the half spatial domain L to be 15 and Fourier modes n to be 64, that is 2^6 . The reason we choose n to be 64 is because we need to discretize the range $x \in [-L, L]$ into 2^n points in order to lower the operation count to $O(N\ log\ N)$ when applying FFT.
3. Create *x, y, z* vectors and each of them contains $n + 1$ points between $-L$ to L with spacing between the points is $\frac{2L}{n}$. Rescale frequencies k by $\frac{2\pi}{2L}$ since the FFT assumes 2π periodic signals and the reason we use$[0: (\frac{n}{2} - 1)\ -\frac{n}{2}: -1]$ is because we want k to match up with the ordering that FFT naturally does.
4. Build 3-D grid coordinates defined by the vectors *x, y, z* and *Kx, Ky, Kz* by using *meshgid()*.
5. Use a for loop to reshape each row of the data into the dimension: $64 \times 64 \times 64$ by using *reshape()* and generate the isosurface visualization of the data in spatial domain.
6. In order to clean up the data, we would like to sum up the spectrum and average it so the white noise would go to zero. To average the spectrum, we reshape the each row of data into the dimension: $64 \times 64 \times 64$ and transform them to frequency domain by using *fftn()*. Then we save the composite of all the frequencies across time in the variable: *Utave.* We rearrange a Fourier Transform *Utave* by shifting the zero-frequency component to the center of the array. Then we take the absolute value of it

[3] Page 325 from the course notes.

and divide by the 20 to get the averaged spectral. The reason we take the absolute value of it is because we care about the intensity of the signal and *Utave* here is a complex matrix. We also normalize *Utave* for visualization, so its value will be from 0 to 1.

7. Plot averaged spectrum using *isosurface()* to see the position of the marble clearly. To get a better plot, we choose the isovalue to be 0.6.
8. Use *indsub()* and *max()* to find the location of maximum value in frequency domain, that is the center frequency created by the marble and then output the position: *Kx*, *Ky*, and *Kz*.
9. Construct a Gaussian filter for 3D: $F(k) = exp(-\tau(k-k_x)^2) \cdot exp(-\tau(k-k_y)^2) \cdot exp(-\tau(k-k_z)^2)$ where k_x, k_y, k_z are the target frequency we found in step 8. Use a for loop to process every row of our data, and use *fftshift()* to unswap the 1st and 2nd half of our domains for every row of the Fourier transformed data since Gaussian filter need to be apply before the swapping. Use *ifftn()* to convert the data back to the time domain and normalize it. Then use the same technique in step 8 to find the location of the highest intensity signal at each time measurement and save the corresponding *x, y, z* values in a 20×3 matrix called *traject.*
10. Plot the path of marble over time using *plot3()* and output the final position at the 20th data measurement.

Computational Results

Below is the plot we obtain from step 5 which is the isosuface visualization of the 1st measurement. The visualizations of the rest 19 measurements have been omitted.

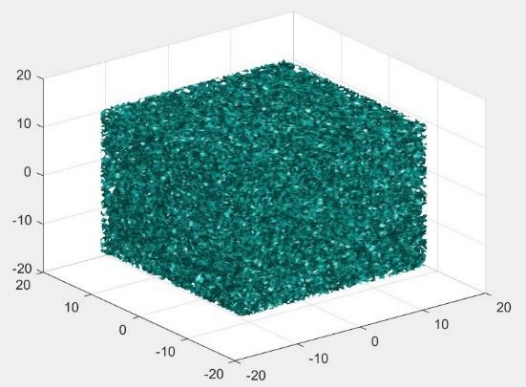

Figure 1: Isosuface visualization of the 1st measurement

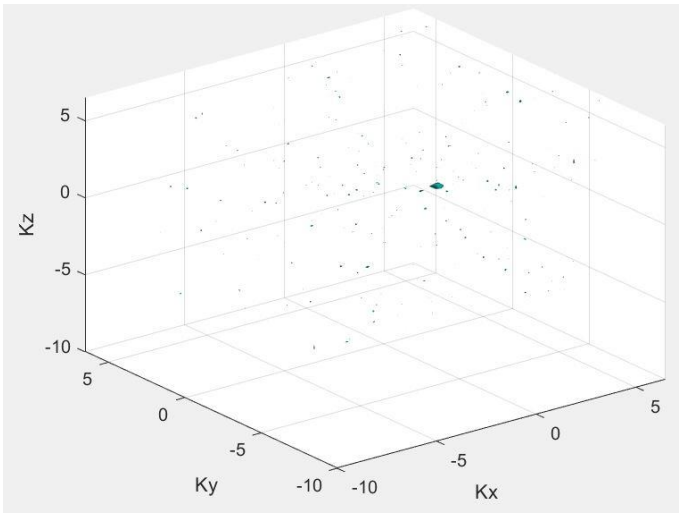

Figure 2: Averaged spectrum with isovalue: 0.6.

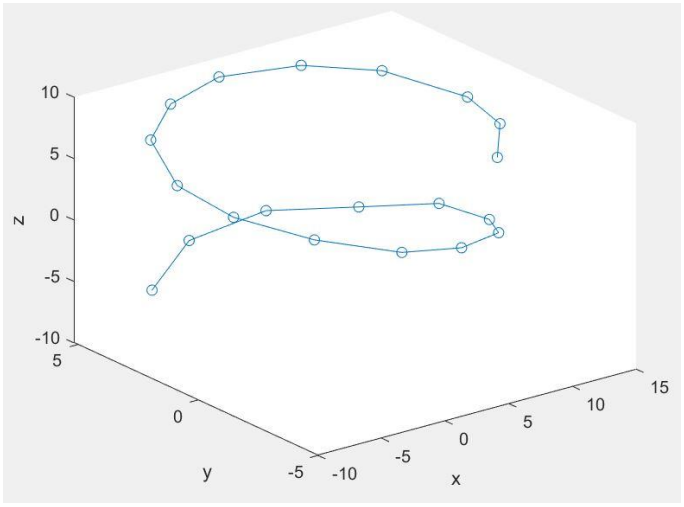

Figure 3: Path of the marble over time

Figure 2 is the plot we got from step 7 and we obtained the center frequency where: *Kx=1.8850, Ky=-1.0472, Kz=0* in step 8.

From step 10, we obtain the plot in Figure 3 which shows the path of marble over time. The final position at the 20th data measurement is *x= -5.6250, y= 4.2188, z=-6.0938.*

Summary and Conclusions

Since we were able to locate the marble, the dog's life will be saved once the intense acoustic wave is applied. In this application, we see how fast Fourier transform and Gaussian filter can be successfully applied in real life to extract useful data from a noisy field.

Appendix A. MATLAB functions used and brief implementation explanation

- *y = linspace(x1,x2,n)* generates n points. The spacing between the points is (x2-x1)/(n-1).[4]
- *[X,Y,Z] = meshgrid(x,y,z)* returns 3-D grid coordinates defined by the vectors x, y, and z. The grid represented by X, Y, and Z has size length(y)-by-length(x)-by-length(z).
- *B = reshape(A,sz1,...,szN)* reshapes A into a sz1-by-...-by-szN array where sz1,...,szN indicates the size of each dimension.
- *fv = isosurface(X,Y,Z,V,isovalue)* computes isosurface data from the volume data V at the isosurface value specified in isovalue. That is, the isosurface connects points that have the specified value much the way contour lines connect points of equal elevation.
- *Y = fftn(X)* returns the multidimensional Fourier transform of an N-D array using a fast Fourier transform algorithm. The N-D transform is equivalent to computing the 1-D transform along each dimension of X. The output Y is the same size as X.
- *Y = fftshift(X)* rearranges a Fourier transform X by shifting the zero-frequency component to the center of the array.
- *[row,col] = ind2sub(sz,ind)* returns the arrays row and col containing the equivalent row and column subscripts corresponding to the linear indices ind for a matrix of size sz. Here sz is a vector with two elements, where sz(1) specifies the number of rows and sz(2) specifies the number of columns.
- *X = ifftn(Y)* returns the multidimensional discrete inverse Fourier transform of an N-D array using a fast Fourier transform algorithm. The N-D inverse transform is equivalent to computing the 1-D inverse transform along each dimension of Y. The output X is the same size as Y.
- *k = find(X)* returns a vector containing the linear indices of each nonzero element in array X.
- *plot3(X,Y,Z)* plots coordinates in 3-D space.

[4] All the MATLAB functions and their brief implementation explanation are cited from www.mathwork.com.

Appendix B. MATLAB codes

```
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);
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

% Apply Fourier Transformation on data and average the spectrum
Utave=zeros(n,n,n);
for j=1:20
   Un(:,:,:)=reshape(Undata(j,:),n,n,n);
   Unt=fftn(Un(:,:,:));
   Utave=Utave+Unt;
end
utave=abs(fftshift(Utave))/20;
utave_normal=utave/max(utave(:));

% Plot the isosurface of averageing over the frequency domain
figure(2)
isosurface(Kx,Ky,Kz,utave_normal,0.6)
grid on
xlabel('Kx');
ylabel('Ky');
zlabel('Kz');

% Find the center frequency
ind=find(utave_normal==max(utave_normal(:)));
utave_size=size(utave_normal);
index=ind2sub(utave,ind);
cenfrex=Kx(index)
cenfrey=Ky(index)
cenfrez=Kz(index)

% Create the filter and plot the trajectory
filter=exp(-0.2*(Kx-cenfrex).^2).*exp(-0.2*(Ky-
cenfrey).^2).*exp(-0.2*(Kz-cenfrez).^2);
traject=zeros(20,3);
  for j=1:20
    Un(:,:,:)=reshape(Undata(j,:),n,n,n);
    unt=fftn(Un(:,:,:));
    unft=filter.*fftshift(unt);
```

```
unft_time_normal=abs(ifftn(unft))/max(abs(ifftn(unft(:))));
    ind=find(unft_time_normal==max(unft_time_normal(:)));
size2=size(unft_time_normal);
    index=ind2sub(size2,ind);
traject(j,:)=[X(index),Y(index),Z(index)];
  end figure(3)
plot3(traject(:,1),traject(:,2),traject(:,3),'o-')
xlabel('x');
ylabel('y');
zlabel('z');

% Output the location of marble at the 20th data measurement
traject(20,:)
```