HW1 An Ultrasound Problem

Mingzhong Deng

Janurary 25, 2019

Abstract

Using data, obtained by ultrasound, to find out the position of a marble. The marble is swallowed by a dog and located in the dog's intestine. Data collected from the dog's intestine is highly noisy. This project denoise the data and find out the position of the marble in the dog's intestine.

1 Introduction and Background

Fast Fourier Transform (FFT) is an algorithm that is used to compute the discrete Fourier transform of a sequence or its inverse. Fourier analysis converts a signal from its space time to Fourier space. Fast Fourier Transform imporves the runtime of the computing the discrete Fourier transform from $O(n^2)$ to $O(nlogn)$. For the marble problem, the most important feature of FFT is to use FFT to denoise data, finding out the most important frequency and ignoring the undesirable frequency at the same time. The test data contains twenty rows of ultrasound data for twenty different measurements that were taken in different time. Due to the constant movement of the dog, the data is noisy. The way to solve the problem is using FFT to find the significant frequency of the signals and the trajectory of the marble.

In the next sections, I will describe more of the other algorithm and concepts being used for the problem including Gaussian filtering and averaging the frequency of the signals, and their implementation in MATLAB. Also, I will present the actual trajectory of the marble and the result of saving the dog.

2 Theoretical Background

2.1 Fourier Transform

Fourier transform decomposes a singal in time into its frequency components. The following equation is the detail of Fourier Transform.

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

The inverse of Fourier Transform is the following equation:

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

Some of the properties of Fourier transformation include narrow signal in time space has large spread in Fourier space and wide signal in time space has a small spread in Fourier space. So, the significant changes in time space can be captured easily by Fourier transform. Plus, Fourier transform has a derivative property, by doing integration by parts, the following equation can be easily acquired, and it might be helpful in the future:

$$F^N(k) = F(k)(ik)^N \tag{3}$$

2.2 FFT Algorithm

The FFT algorithm is used to calculate the Fourier Transform for a function. The algorithm reduced the runtime to $O(nlogn)$ from $O(n^2)$. It spreads x into a period from $-L$ to L because x is considered to be periodic. The domain is discretized to 2^n points. The following equation shows the detail of the FFT algorithm:

$$F(k) = \sum_{k=1}^{N} f(n) e^{\frac{-i*2\pi(k-1)(n-1)}{N}} \tag{4}$$

The inverse of FFT is the following equation:

$$f(n) = \frac{1}{N} \sum_{k=1}^{N} F(k) e^{\frac{i*2\pi(k-1)(n-1)}{N}} \tag{5}$$

2.3 Guassian Filtering

The main reason I use filtering in this project is nnoise attenuation. The filtering process is common in electronics and signal detection. Filtering can help improve the ability to detect the signal buried in the noise field. The equation for Guassian filter is the following:

$$\mathscr{F}(k) = exp(-\tau(k - k_0)^2) \tag{6}$$

The Gaussian filter attenuates the frequencies away from the center frequency.

2.4 Time Averaging

Noise can be modeled to have a normal distributed random variable with zero mean and finite standard deviation. If averaging over many signals, the noise should add up to zero. It is tremendously powerful in practice for signal processing. The requirement to perform time averaging can be easily achieved by performing Guassian filtering. Averaging over the frequency realizations produces a clean, localized signature in the frequency domain, which will be Fourier in this problem.

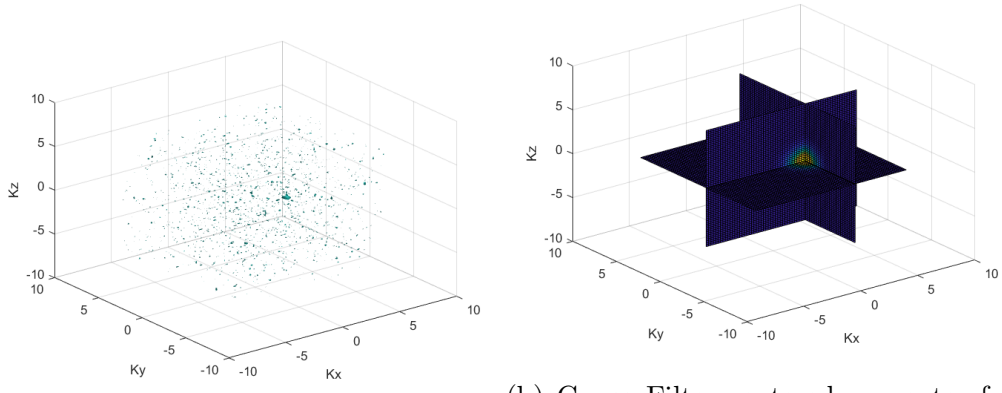
3 Algorithm Implementation and Development

This section explains the procedure of the MATLAB program and the detail information of the implementation.

- Load `testdata.mat`.
- Define domain size and grid vectors.
- Construct grid in 3D in normal space with `meshgrid` function.
- Reshape each row of data into 3D.
- Use `fftn` function for 3D Fast Fourier Transform to send the data to Fourier space.
- Normalize all data with the largest element for each row of data and take the absolute value.

- Visualize the data with `isosurface`.
- Find the center frequency and its spread.
- Use a 3D Gaussion function with center frequency found above as a center and use it to project the entire row of data.
- Multiply each signal by the Gaussian filter in Fourier space.
- Use `ifftn`, 3D inverse FFT function to send the data back to normal space.
- Use `fftshift` to shift the data.
- Normalize all data with the largest element and take the absolute value.
- Visualize the data with `isosurface` and `plot3`.
- Find the position of the marble in the final time step.

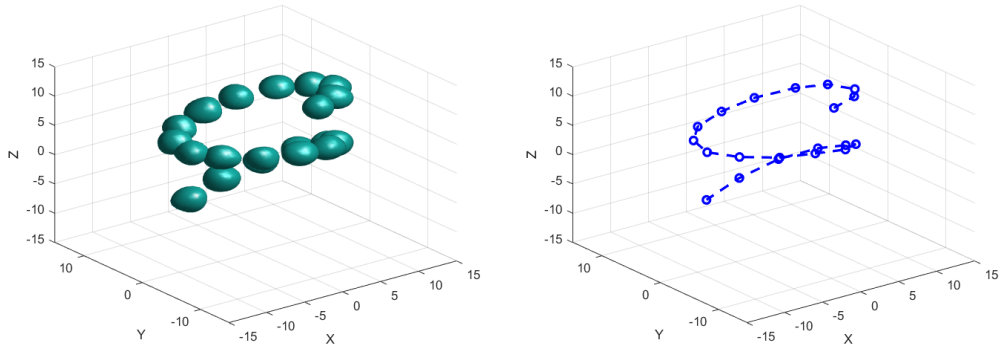
4 Computational Results



(a) Frequency signature of signals

(b) Gauss Filter centered on center frequency

Figure 1: Frequency distribution of signal and Gaussian Filter



(a) Isosurface Visualization (b) plot3 Visualization

Figure 2: Visualization of the trajectory of the marble

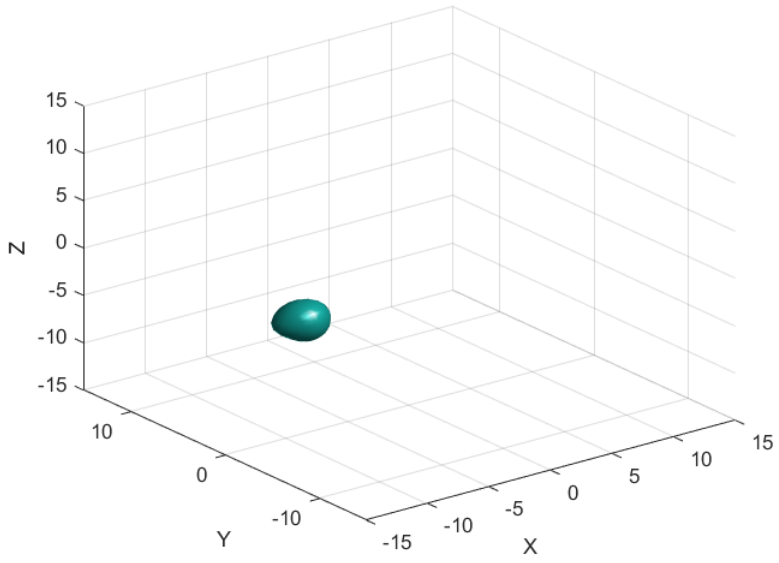


Figure 3: Final position of the marble

Figure 1(a) shows the frequency signature of the signals after time averaging. The frequency signature is located at (9, -5, 0) by multiplying by $\frac{L}{2\pi}$. Figure 1(b) shows the 3D Gaussian filter constructed at the center frequency.

Figure 2(a) shows the isosurface visualization of the trajectory of the marble by applying Gaussian filter. Figure 2(b) shows the plot3 visualization of the trajectory of the marble by ploting out the center of the marble at each time step.

Figure 3 shows the isosurface visualization of the final position of the marble from the last row of the input data. The final position is (-5.625, 4.2188, -6.0396).

5 Summary and Conclusions

The use of gaussian filtering, time averaging, and FFT for capturing the siganl in Fourier space are able to denoise the data and find the position of the marble at each time step. The Gaussian filter is able to construct around the frequency signature and time averaging method helps to denoise the data. The final position is (-5.625, 4.2188, -6.0396). I have successfully saved the dog.

6 References

https://faculty.washington.edu/kutz/KutzBook/page14.html
Data-Driven Modeling & Scientific Computation

Appendix A - MATLAB functions

- `load`: load the data into a tensor.
- `linespace(X, Y, n)`: generate n points between X and Y.
- `fftshift(X)`: rearranges a Fourier transform X by shifting the zero-frequency component to the center of the array.
- `meshgrid`: creates a grid with grid vectors in desired dimension(s).
- `zeros`: generate a tensor filled with zeroes with the desired dimension or size.
- `abs`: returns the absolute value of the input value.

- `max`: returns the maximal value within the input list, matrix, or multiple dimensional tensor.
- `fftn`: returns the N-dimensional discrete Fourier transform of the input N-dimensional tensor by running the FFT algorithm.
- `ifftn`: returns the N-dimensional tensor in normal time space of the input N-dimensional tensor in Fourier space by running the ivnerse FFT algorithm.
- `slice(X,Y,Z,V)`: draws slices for the data V, specify X,Y, and Z as the coordinate data.
- `isosurface(x,y,z,f,value)`: computes the isosurface for data f at isosurface value. x,y,z arrays specify the points at the data given.
- `reshape`: returns the matrix or tensor with desired size from the different dimensional input.
- `plot3(x,y,z)`: plot a line in 3D through the points from vector x, y, and z.

Appendix B - MATLAB code

```
clear all; close all; clc;
load Testdata.mat

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);

avg = zeros(n,n,n);
for j = 1:20
    Un(:,:,:) = reshape(Undata(j,:), n,n,n);
    avg = avg + fftn(Un);
```

```
end

avg = abs(fftshift(avg));
max_num = 0;
for i = 1:n
    for j = 1:n
        for k = 1:n
            if avg(i,j,k) > max_num
                max_num = avg(i,j,k);
                a = i;
                b = j;
                c = k;
            end
        end
    end
end

%%
figure(1)
isosurface(Kx,Ky,Kz,abs(avg)/max(avg(:)), 0.5);
grid on;
xlabel('Kx');
ylabel('Ky');
zlabel('Kz');

%%
gaussfilter = exp(-((Kx-ks(b)).^2 + (Ky-ks(a)).^2 + (Kz-ks(c)).^2)/2);
figure(2)
slice(Kx, Ky, Kz, gaussfilter, 2, -1, 0);
xlabel('Kx');
ylabel('Ky');
zlabel('Kz');

%%
% apply for all 20 signals
maximum = 0;
matrix = [];
figure(3)
```

```
for iter = 1:20
    Un(:,:,:) = reshape(Undata(iter,:),n,n,n);
    fftUn = fftshift(fftn(Un));

    datT = fftUn.*gaussfilter;
    dat = ifftn(datT);

    for i = 1:64
        for j = 1:64
            for k = 1:64
                if abs(dat(i,j,k)) > maximum
                    maximum = abs(dat(i,j,k));
                    b = X(1,i,1);
                    a = Y(j,1,1);
                    c = Z(1,1,k);
                end
            end
        end
    end
    maximum = 0;
    matrix = [matrix; [a b c]];

    isosurface(X,Y,Z, abs(dat)/max(abs(dat(:))), 0.5);
    axis([-L L -L L -L L]);
    grid on;
    xlabel('X');
    ylabel('Y');
    zlabel('Z');
    pause(0.2);
end

%%
figure(4)
plot3(matrix(:,1), matrix(:,2), matrix(:,3), ...
    'b--o', 'LineWidth', 2);
axis([-L L -L L -L L]);
grid on;
xlabel('X');
```

```
ylabel('Y');
zlabel('Z');

%%
figure(5)
isosurface(X,Y,Z, abs(dat)/max(abs(dat(:))), 0.5);
axis([-L L -L L -L L]);
grid on;
xlabel('X');
ylabel('Y');
zlabel('Z');
```