# Amath 482 homework 1: An ultrasound problem

Andy Zhang

January 24, 2020

**Abstract**

This report describes the way to locate the marble swallowed by a dog by operating on data in frequency space. Due to the noise generated by the fluid movement in the dog's intestines, we need to locate the center frequency of the marble by averaging 20 lines of data and find the maximum. Then we can determine the position of the marble by filtering out the noise around center frequency and form a trajectory in spatial domain. The final position of the marble can be determined by the 20th line of data.

## 1 Introduction

My dog swallowed a marble, which went into its intestines. By using ultrasound, the vet collected the data of the marble's trajectory. However, the dog kept moving and fluid inside its intestines generated highly noisy data. Our goal is to filter out the noises and determine the position of the marble so that I can save my dog!

## 2 Theoretical Background

### 2.1 Fourier Transform

The Fourier Transform is the basis of this method, which helps us to transform the data from time or spatial domain to frequency domain. The Fourier Transform and its inverse are defined as:

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

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

We can understand Fourier Transform by thinking it as disassembling the periodic function into sums of sines and cosines, proved by Euler's formula: $e^{ikx} = cos(kx) + isin(kx)$, where k represents wavenumbers and the term $e^{ikx}$

represents oscillatory integration kernel. Fourier Transform is defined over the entire real line $x \in [-\infty, \infty]$, whereas our computational domain is finite over $x \in [-L, L]$.

We use commands ***fft(u)*** and ***ifft(u)*** in Matlab to do Fast Fourier Transform and its inverse, with operation order $O(NlogN)$. The lower operation order of Fast Fourier Transform makes more practical use when dealing with large amounts of data. We also need **L** (spatial or time domain) and **n** (Fourier modes) to determine the frequencies **k**. Note that Fast Fourier Transform assumes a periodic signals, that is, frequency **k** is in unit of **rad/time or spatial unit**. That's why we rescale **k** by $2\pi/L$. In this project, commands ***fftn(u)*** and ***ifftn(u)*** are used to deal with multi-dimensional arrays.

## 2.2 Gaussian Filter

Applying a Gaussian filter around the center frequency of the given data helps us eliminate the noises and determine the spatial trajectory of marble. A commonly used Gaussian filter is defined as:

$$F(k) = e^{-\tau(k-k_0)^2}, \tag{3}$$

where $\tau$ determines the width of filter window and **k** represents the wavenumbers(the frequencies over finite domain). $k_0$ is the frequency we are interested in the frequency space, in this case, the center frequencies of marble.

The Gaussian Filter can effectively eliminate the noise away from the center frequencies. However, since we're dealing with data in 3-d dimensions, so we have to modify the filter to:

$$F(k) = e^{-\tau((Kx-xf)^2+(Ky-yf)^2+(Kz-zf)^2)}, \tag{4}$$

where xf, yf, zf represent the center frequency at x, y, and z dimensions, and Kx, Ky, Kz are grids that that consist of frequencies in three dimensions. It filters out the noise in all dimensions.

## 2.3 Averaging

Based on the idea that white-noise can be modeled as normally distributed variables with zero mean to each Fourier component, we can then average up the sum of many data, with a zero mean for added-up white noises as well. This is practical for continuous signal processing because the more signals we add up and take average, the more precisely and clearly the center frequency is shown. If we are dealing with data instead of plots, like in this case, the center frequency can be determined by the coordinates of the maximum element in the averaged up data.

# 3 Algorithm Implementation and Development

## Prepare for Fourier Transform

To start with, we need the spatial domain **L** and Fourier modes **n**, which are pre-set to 15 and 64 respectively. Since we are solving it numerically, so we have to discrete the domain into **n+1** points and take the first **n** points to form periodic boundary conditions.

Then we need to rescale the frequencies **k** by $2\pi/2L$ because Fast Fourier Transform assumes a $2\pi$ periodic signals and the domain ranges from $-L$ to $L$, that is a total $2L$ units long. We also have to **fftshift()** the frequencies so that the zero frequency is staying at the middle of the array.

Since we are dealing with multidimensional data, we have to create spatial grids for X, Y, and Z, and frequency grids for Kx, Ky, and Kz, with the size of 64 by 64 by 64.

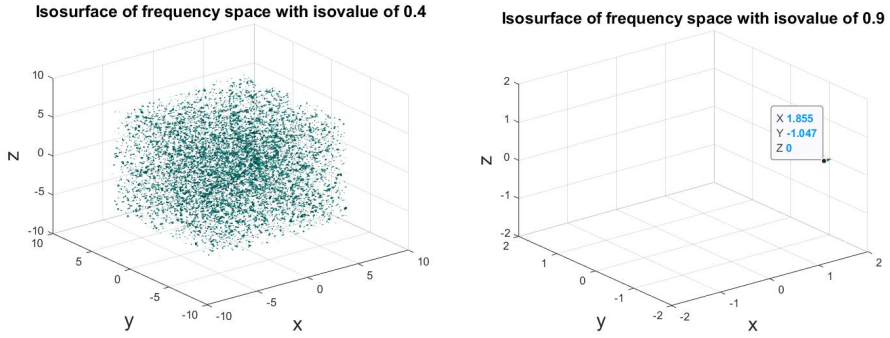
## Reshape the data, add them up, and take average

Then we need to reshape the 20 lines of data into a 64 by 64 by 64 grid named **u** and apply **fftn()** to **u** because we are curious about the center frequencies in the frequency space. After taking average by dividing 20, we normalize the grid and name it **uave**.

## Find the center frequency

The normalized **uave** has its maximum element of 1 and we are going to find the **index** of this **aveMax** in **uave** by calling $[aveMax, Index] = max(uave(:))$, in which **(uave(:))** changes the grid into a vector so that we can use an index to represent three-dimensional coordinates. Then we call $[Px, Py, Pz] = ind2sub(size(uave), Index)$ to find the arrays of **uave** that **aveMax** is in. Finally, we apply the coordinate (Px, Py, Pz) to **Kx**, **Ky**, and **Kz** to find the center frequencies in all dimensions of frequency space.

Figure 1: Isosurface plot with different isovalues



## Make sure we find the correct center frequency

We are calling the **isosurface** command to present the frequencies according to given isovalues. As Figure(1) shows, with an isovalue of 0.4, it's still hard to find the frequency of marble under the influence of noises. Then I gradually increase the isovalue up to 0.9, which is close to the ceiling of isovalue that we can't see anything in the plot. We can check the coordinate of frequency by clicking onto it, which shows the same frequency found by the above algorithm. The center frequency calculated is provided in the ***computational results*** section below.

## Build a Gaussian filter and find the trajectory

As shown above in the **Gaussian filter** section, we build such filter that filters out noises in all dimensions.

$$F(k) = e^{-\tau((Kx-xf)^2+(Ky-yf)^2+(Kz-zf)^2)} \tag{5}$$

We reshape the data again, and transform it into frequency space by **fftn()**. Then we apply **fftshift()** to match with corresponding frequencies and array multiply the resulting grid and filter. We then have to do **ifftshift()** because the filter is based on frequencies being applied **fftshift()** and the resulting grid also gets shifted once. We finally take the frequency information back to spatial domain to get **uf**.

We apply the same idea as finding the center frequency. $(xt, yt, zt)$ is the coordinate of the max frequency per line of data. Applying this coordinate to $X, Y, Z$ results in the actual position of marble according to each data line. Connecting them all forms the trajectory.

I've tested on three different values of $\tau$, that are 0.1, 0.3, and 0.9. Figure(2) represents the downward view of trajectories for three cases and I find the trajectory seems most stable when $\tau = 0.3$, thus I take it as the appropriate $\tau$ for the data in this case.

Figure 2: Downward trajectory of the marble applying filters with different $\tau$

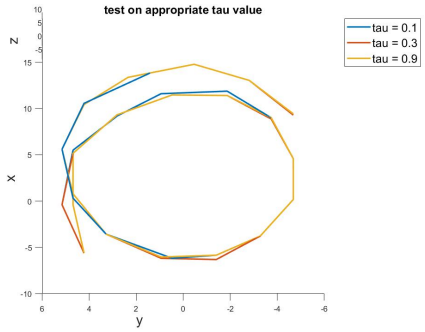
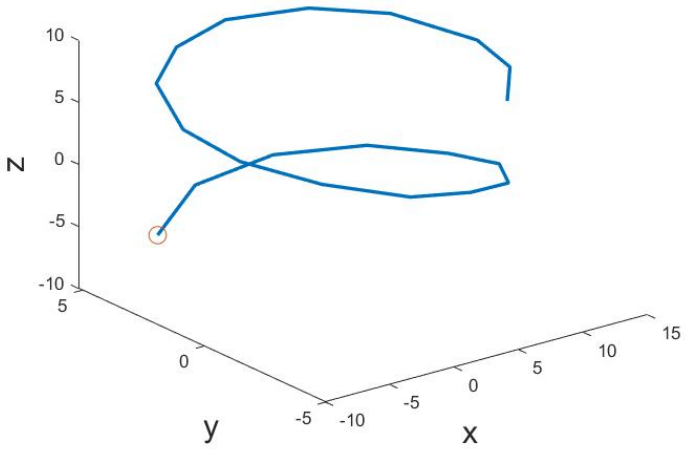


Figure 3: Trajectory of the marble in spatial domain with desired $\tau$




# 4 Computational Results

## Center frequency

The center frequency of the marble is:

$$xf = 1.8850, yf = -1.0472, zf = 0 \tag{6}$$

We need this center frequency so that we can build a filter and trace the marble in spatial domain.

## Trajectory of marble

As shown by Figure(3), it represents the marble's trajectory from all 20 lines of data. The highlighted marker represents the 20th data measurement and it's the position for the vet to use an intense acoustic wave to break the marble.

## The spacial position of marble in 20th data measurement

It's the last row vector in **trajectory**, with coordinate:

$$x = -5.250, y = 4.2188, z = -6.0938 \tag{7}$$

# 5 Summary and Conclusion

From this assignment, we can solve the case by alternately operate on data in spatial or frequency domain. Generally, we can solve the problem by simply

finding the center frequency through averaging the data, build an appropriate Gaussian filter that denoise the data, and finding the positions of marble by analysing the data. One needs to grasp Fourier Transform and the ideas behind filter and averaging in order to solve similar problems.

# 6 Appendix A

**ut = fftn(u)** returns the multidimensional Fourier Transform of N-dimensional array u.

**uf = iffrn(utf)** returns the multidimensional inverse Fourier Transform of N-dimensional array utf.

**ks = fftshift(k) or k = ifftshift(k)** rearranges the Fourier Transform by shifting the zero frequency component to the middle of the array. For N-dimensional arrays, it swaps "half-space" along each dimension. "ifftshift()" is the inverse.

**u(:,:,:)=reshape(Undata(i,:),n,n,n)** reshapes the i-th row of Undata to a grid **u** with size of n by n by n.

**[Px,Py,Pz] = ind2sub(size(uave), Index)** returns the array of Px, Py, and Pz containing the equivalent subscripts in different dimensional arrays according to the size of grid and given index.

**isosurface(X, Y, Z, V, isovalue)** computes the isosurface geometry for data V at given isovalue.

**plot3(X, Y, Z)** plots 3 dimensional graph. It can plot points or lines according to the data type of X, Y, and Z.

# 7 Appendix B

```
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);

uave = zeros(n, n, n);

% reshape the data into 64*64*64 matrix, add them up
for j = 1:20
    u(:,:,:)=reshape(Undata(j,:),n,n,n);
    ut = fftn(u);
    uave = uave + ut;
end

% take average and and normalize it
uave = abs(fftshift(uave)) / j;
uave = uave / max(uave(:));

% find the position (Px, Py, Pz) of aveMax in uave
[aveMax, Index] = max(uave(:));
[Px,Py,Pz] = ind2sub(size(uave), Index);

% find the corresponding position (xf, yf, zf) of aveMax
    in frequency space
xf = Kx(Px,Py,Pz);
yf = Ky(Px,Py,Pz);
zf = Kz(Px,Py,Pz);

% make sure we find the correct frequencies xf, yf, zf
```

```
figure(1)
isosurface(Kx,Ky,Kz,uave,0.4)
title('Isosurface of frequency space with isovalue of 0.4','FontSize',15)
xlabel('x','FontSize',18)
ylabel('y','FontSize',18)
zlabel('z','FontSize',18)
axis([-10 10 -10 10 -10 10])
grid on

figure(2)
isosurface(Kx,Ky,Kz,uave,0.9)
title('Isosurface of frequency space with isovalue of 0.9','FontSize',15)
xlabel('x','FontSize',18)
ylabel('y','FontSize',18)
zlabel('z','FontSize',18)
axis([-2 2 -2 2 -2 2])
grid on


%for test purpose only
%{
tau = [0.1, 0.3, 0.9];
for k = 1 : 3
    filter = exp(-tau(k) * ((Kx - xf).^2 + (Ky - yf).^2 + (Kz - zf).^2));

    trajectory = zeros(20, 3);
    for i = 1 : 20
        u(:,:,:)=reshape(Undata(i,:),n,n,n);
        ut = fftshift(fftn(u));
        utf = ifftshift(filter.* ut);
        uf = ifftn(utf);
        [traceMax, Index] = max(uf(:));
        [xt, yt, zt] = ind2sub(size(uf), Index);
        trajectory(i, 1) = X(xt, yt, zt);
        trajectory(i, 2) = Y(xt, yt, zt);
        trajectory(i, 3) = Z(xt, yt, zt);
    end
    plot3(trajectory(:, 1), trajectory(:, 2), trajectory(:, 3), 'Linewidth', 2)
    hold on
end
title('test on appropriate tau value', 'FontSize', 15)
xlabel('x', 'FontSize', 18)
```

```
ylabel('y', 'FontSize', 18)
zlabel('z', 'FontSize', 18)
legend('tau = 0.1', 'tau = 0.3', 'tau = 0.9', 'FontSize',
    15)
%}

% build a filter and center on data around (xf, yf, zf)
filter = exp(-0.3 * ((Kx - xf).^2 + (Ky - yf).^2 + (Kz -
    zf).^2));

% trace the trajectory of the marble through each data
    spatially
trajectory = zeros(20, 3);
for i = 1 : 20
    u(:,:,:)=reshape(Undata(i,:),n,n,n);
    ut = fftshift(fftn(u));
    utf = ifftshift(filter.* ut);
    uf = ifftn(utf);
    [traceMax, Index] = max(uf(:));
    [xt, yt, zt] = ind2sub(size(uf), Index);
    trajectory(i, 1) = X(xt, yt, zt);
    trajectory(i, 2) = Y(xt, yt, zt);
    trajectory(i, 3) = Z(xt, yt, zt);
end

figure(3)
plot3(trajectory(:, 1), trajectory(:, 2), trajectory(:,
    3), 'Linewidth', 2)
title('trajectory of the marble in spatial domain', '
    FontSize', 15)
hold on
plot3(trajectory(20, 1), trajectory(20, 2), trajectory
    (20, 3), 'o', 'MarkerSize', 10)
xlabel('x', 'FontSize', 18)
ylabel('y', 'FontSize', 18)
zlabel('z', 'FontSize', 18)
```