An Ultrasound Problem

Yuanheng (Jonathan) Zhao

January 20th, 2020

Abstract

This report discusses a method using Fourier transforms to denoise an ultrasound dataset. Our goal is to locate a marble at twenty measurements in some spatial domain. To obtain the solution, we will apply the fast Fourier transform and its inverse to switch the data between the spatial domain and frequency domain, the method of averaging to find the frequency signature, and the technique of filtering to extract intuitive information from the spectrum.

Sec I: Introduction and Overview

Nowadays, the Fourier transform is considered one of the most powerful and efficient methods for solving problems related to time-frequency analysis in diverse fields of study, such as physics and biology. It is widely used in signal processing and analyzing. We are going to apply the fast Fourier transform, an efficient and accurate algorithm that performs forward and backward Fourier transform, to extract crucial information we want from a noisy dataset.

In this problem, our dog fluffy swallowed a marble by accident. By using ultrasound, the vet obtained the data concerning the spatial variations in a small area of the intestines where the marble is suspected to be. However, the dataset is filled with noises that we can find nothing useful to locate the marble, since fluffy keeps moving and the internal fluid movement generates highly noisy data! To save our dog's life, we are going to compute the trajectory of the marble and provide the position of the marble at the 20th data measurement so that an intense acoustic wave can be focused to break the marble.

In the next sections, I will present some theoretical concepts and explanations about FFT, filters, and technique of averaging, which are essential for us to solve the problem. Then we will discuss the implementation of the algorithm in MATLAB, and realize the functionalities of the algorithm based on the computational results. At the end of the report is appended with MATLAB functions implemented and codes.

Sec II: Theoretical Background

2.1 Fourier Transform

Fourier transform is a tool to decompose a function from time domain to frequency domain. It is an integral defined over the entire line $x \in [-\infty, \infty]$ and computed over a finite domain $x \in [-L, L]$. This powerful tool can be used to transform a sophisticated function into another form, decomposing it into a frequency-amplitude image.

For $x \in [-\infty, \infty]$, the Fourier transform (in one dimension) is defined as:

$$F(k) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-ikx} f(x) dx \tag{2.1.1}$$

The inverse of Fourier transform is defined as:

$$f(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{ikx} F(k) dk \tag{2.1.2}$$

2.2 The Fast Fourier Transform

Through the discussion about the problem, we will apply the Fast Fourier transform, commonly known as the FFT algorithm, to perform the Fourier transform or its inverse. It is widely used because of its low operation count O(Nlog$N$), and excellent accuracy. The relatively low operation count is realized by recursively dividing the range $x \in [-L, L]$ into two pieces, and finally get the domain discretized into 2^n points. That is why we say the FFT prefers 2^n.

In matlab, the related functions such as fft(x), ifft(x), fftshift(x), ifftshift(x), fftn(x), are ready to perform forward and backward, one-dimensional and multi-dimensional FFTs. The algorithm associated with the FFT assumes we are working on a 2π periodic domain, and so that we are going to rescale the frequencies k by $2\pi/L$ later.

2.3 Gaussian Filtering

As we talked about the algorithm of Fourier transform, we might notice how practical it is to realize the functionalities of putting and manipulating data in the frequency domain. However, for most of the applications in the real world, there exist noises mixed with the signals we receive or we want. For example, a radar sends electromagnetic fields to detect planes might receive reflections coming from undesired targets, such as geographical objects and atmospheric phenomena. That is why we introduce the procedure of noise attenuation via frequency filtering.

Spectral filtering is a method that allows us to extract information at specific frequencies. By diminishing or removing undesired frequencies and white-noise, the spectral filter around a specific frequency can be used to filter out desired components. And one of the simplest filters we will use later is the Gaussian filter, which is defined as:

$$\mathcal{F}(k) = \exp\left(-\tau\left(k - k_0\right)^2\right) \tag{2.3.1}$$

where τ measures the bandwidth of the filter and k_0 represents the center-frequency of the desired signal field.

2.4 Averaging

Averaging is a method we use to extract a clean spectral signature, or central frequency, which is the specific frequency or frequency range of the target object, from many noisy signals with different measurements. The key idea is that white-noise over many signals should, on average, add up to zero. Thus, by averaging a given set of realizations, we can remove the white-noise and get a distinguishable structure of center frequency. The more realizations we have, the more accurately we can locate the center frequency. However, in the averaging process, we lose the time-domain dynamics.

Sec III: Algorithm Implementation & Development

Part 1: Initialization

First we load the data from **Testdata.mat**, which in some sense gives the locations of the marble at twenty different points of time. Then we define the spatial domain and frequency domain, while the frequency domain indicates the number of frequencies we are looking at the spatial domain in each direction. Thus we divide x, y, z $\in$ [-*L*, *L*] in spatial domain into *n* discrete modes. And we also need to rescale the frequency domain by multiplying them with $2\pi/L$, since the functions associated with the FFT assumes we are working on a 2π periodic domain. Then we shift the axes of the frequency domain via *fftshift* to make x $\in$ [0, *L*] $\longrightarrow$ x $\in$ [-*L*, 0] and x $\in$ [-*L*, 0] $\longrightarrow$ x $\in$ [0, *L*], which exactly corresponds to the way MATLAB shifts the data.

We use *meshgrid* to make 3-D arrays to construct the X-Y-Z coordinate system and Kx-Ky-Kz frequencies system.

Part 2: Averaging

Construct a 3-D array to store the accumulated frequencies from the twenty realizations. In this process, we use *reshape* to fit the realization data into three dimensions and *fftn* to do multi-dimensional FFT to get the frequencies to be added up.

Once we get the *ave*, the accumulated frequencies of signals where most of the noises have been cancelled out, we rescale it by the shifting rule mentioned above. Since *ave* includes complex numbers and we just want the intensity of the signals, we take the absolute value of the accumulated frequencies, *ave*. Then we search for the strongest signal through the whole array. Divide *ave* by the strongest signal. At this time we can plot the isosurface about the averaged signals. Of course we are not going to determine the center signal by just moving the data cursor on the graph by hand. We will apply *find* to look for the coordinates in frequency domain with the strongest signal, which are respectively the center frequency in each direction.

Part 3: Filtering

Setting the bandwidth $\tau = 0.2$, we can create a multi-dimensional Gaussian filter around the center frequency following the form of equation 2.3.1, through dot product of filters in each dimension. Do the same thing we did in the averaging process: Reshape and do multi-dimensional on the data for each realization. Then we apply this Gaussian filter by multiplying the transformed data in each direction with the filter to get *unft*. And then we apply *ifftn* to transform the *unft* in frequency domain back to spatial domain. Take absolute value of it to remove the effects of complex numbers and search for the strongest signal with maximum absolute value. Based on the coordinates with the strongest signal, we can locate the marble in spatial domain for each realization, and store the position information into some array.

Part 4: Output

Based on the position information we stored for each realization in the last step, we can plot the path of the marble by using plot3. And the position of the marble at the 20th data measurement corresponds to the coordinates stored in the last row of the position array.

Sec IV: Computational Results

The visualization of the first measurement, in the spatial domain, created by *isosurface(X,Y,Z, abs(Un), 0.4)* is shown in Figure 1. We can gain nothing intuitive at the current status.

After the procedure of averaging, we successfully remove the white-noise and plot the isosurface visualization that shows the center frequency of the data in frequency domain (Figure 2). The right one in Figure 2 is a magnifying version with a data cursor on it to show approximate values of center frequency, while the computed center frequency [*kx*,*ky*, *kz*] = [1.8850, -1.0472, 0].

By applying the Gaussian filter, we get to denoise the twenty measurements and plot the isosurface visualization of all the twenty measurements in spatial domain, and the path of the marble at the twenty points of time in Figure 3.

The position of the marble at the 20th data measurement is computed as:
[x, y, z] = [-5.625, 4.21875, -6.09375]

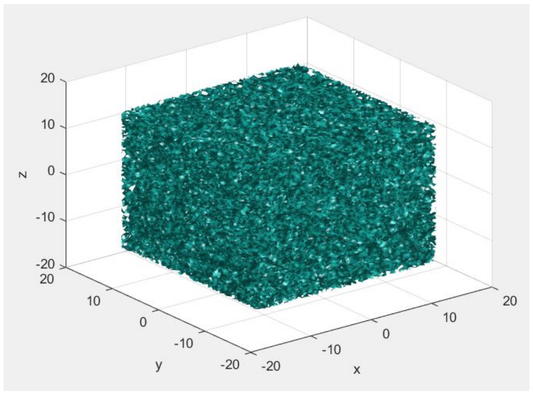

Figure 1: Isosureface for the first measurement in spatial domain (original data).

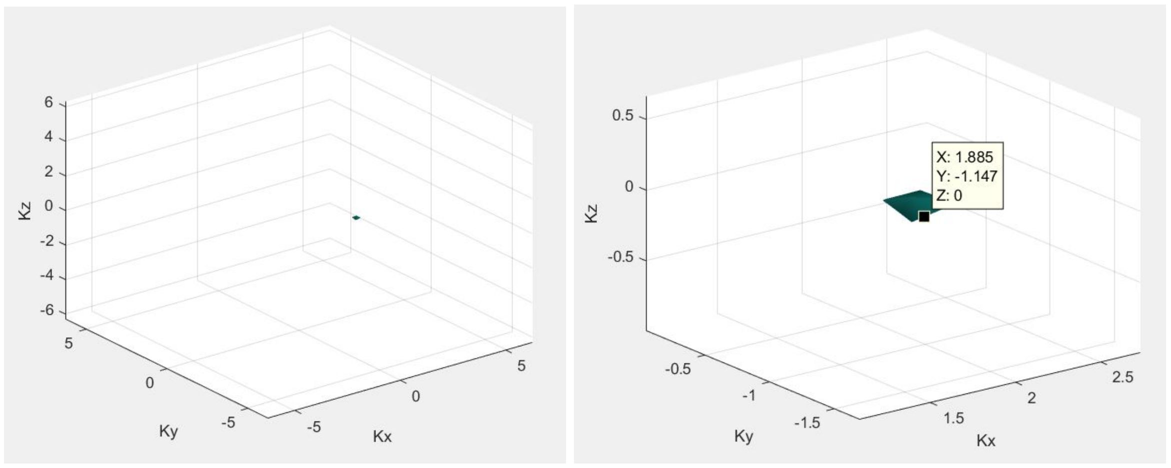

Figure 2: Isosureface for averaged spectrum in frequency domain.

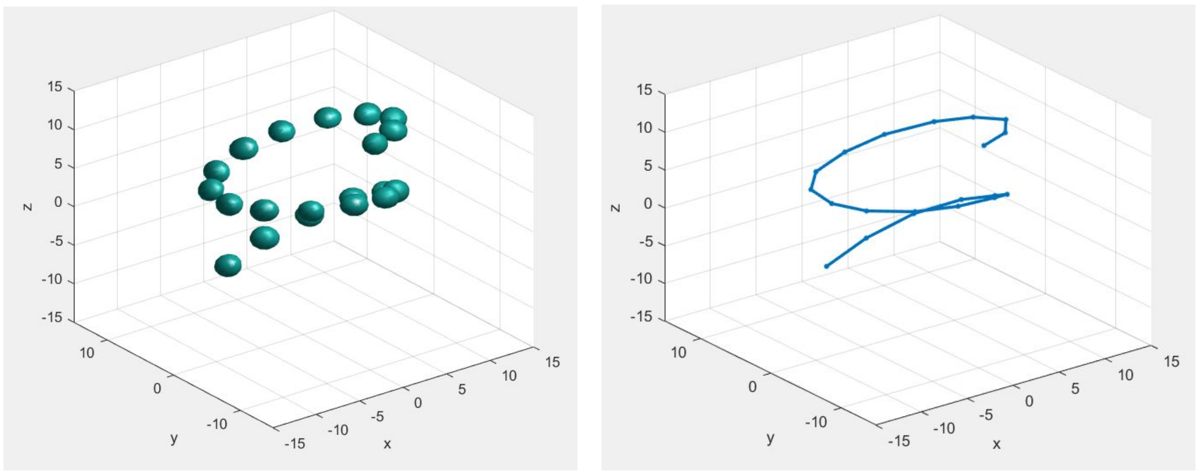

Figure 3: The isosurface of all denoised measurements in spatial domain (left) and the path of the marble at twenty points of time (right).

Sec V: Summary and Conclusions

Through the procedure of averaging, we remove the white-noise and obtain the frequency signature. By applying the Gaussian filter with frequency signature in each direction, we denoise the dataset in spatial domain and successfully track the marble. It illustrated that the method we use with FFT is pretty helpful in solving problems about signal denoising. The final solution is (-5.625, 4.219, -6.094), where an intense acoustic wave can be focused to break up the marble to save our dog.

Appendix A: MATLAB functions used

functions	Syntax we used	Brief explanation
find	*[row, col] = find(__)*	Returns the row and column subscripts of each nonzero element in an array with given some input arguments.
fftn	*fftn(X)*	Returns the multidimensional Fourier transform of an N-D array *X* using FFT.
fftshift	*fftshift(X)*	Rearranges a Fourier transform *X* by shifting the zero- frequency component to the center.
ifftn	*ifftn(Y)*	Returns the multidimensional discrete inverse Fourier transform of an N-D array using FFT.
ifftshift	*ifftshift(Y)*	Rearranges a shifted Fourier transform Y back to the original transform output. The inverse of *fftshift(X)*.
isempty	*TF = isempty(A)*	Returns logical 1 (true) if A is empty, and logical 0 (false) otherwise.
isosurface	*isosurface(X,Y,Z,V,isovalue)*	Computes isosurface data from the volume data V at the isosurface value specified in isovalue, and creates 3-D visualizations.
meshgrid	*[X,Y,Z] = meshgrid(x,y,z)*	Returns 3-D grid coordinates defined by the vectors x, y, and z.
plot3	*plot3(X,Y,Z,LineSpec)*	Creates the plot using the specified line style, marker, and color.
reshape	*B = reshape(A,sz1,...,szN)*	Reshapes a multidimensional array into a matrix with specific sizes.

Appendix B: MATLAB codes

```
%% Part1: Initialization

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);


%% Part2-1: Adding up the signals of all realizations

ave = zeros(n, n, n);

for realize=1:20

   Un(:,:,:)=reshape(Undata(realize,:),n,n,n);

   Utn = fftn(Un);

   ave = ave + Utn;


%     close all, isosurface(X,Y,Z,abs(Un),0.4)

%     axis([-20 20 -20 20 -20 20]), grid on, drawnow
%     xlabel('x'); ylabel('y'); zlabel('z');
%     pause
end

%% Part2-2: Averaging & Finding the center frequency
ave = abs(fftshift(ave));
flatAve = reshape(ave, n^3, 1);
strongest = max(flatAve);
ave = ave/strongest;
close all, isosurface(Kx,Ky,Kz,ave,0.8)
axis([-2*pi 2*pi -2*pi 2*pi -2*pi 2*pi]), grid on, drawnow
xlabel('Kx'); ylabel('Ky'); zlabel('Kz');

ave = ifftshift(ave); % shift back frequency domain
% look for the strongest signal
for m = 1:n
   [j, i] = find(ave(:,:,m) == 1);
   if isempty(i)~=1
      center_frequency = [i, j, m];
      break
   end
end
kx=k(center_frequency(1));
ky=k(center_frequency(2));
```

```
kz=k(center_frequency(3));


%% Part3-1: Create a Filter
tau = 1.0;
filter = exp(-tau*(fftshift(Kx)-kx).^2).*...
    exp(-tau*(fftshift(Ky)-ky).^2).*...
    exp(-tau*(fftshift(Kz)-kz).^2);


%% Part3-2: Filtering & locating the marble in each realization
position = zeros(20, 3);
for realize = 1:20
    % apply filter on each realization
    Un(:,:,:)=reshape(Undata(realize,:),n,n,n);
    Utn = fftn(Un);
    unft = filter.* Utn;
    unf = ifftn(unft); % inverse multi-dimensional FFT
    unf = abs(unf);
    flat_unf = reshape(unf, n^3, 1);
    strongest = max(flat_unf);
    % look for the strongest signal
    for m = 1:n
        [j, i] = find(unf(:,:,m) == strongest);
        if isempty(i)~=1
            position(realize,:) = [x(i), y(j), z(m)];
            break
        end
    end

    isosurface(X,Y,Z,unf/strongest,0.8)
    axis([-L L -L L -L L]), grid on, drawnow
    xlabel('x'); ylabel('y');zlabel('z');
    pause(1)
end

%% Part4: Solution
% plot the path of the marble
plot3(position(:,1), position(:,2), position(:,3), '.-', 'MarkerSize',...
    10, 'Linewidth', 2)
axis([-L L -L L -L L]), grid on, drawnow
xlabel('x'); ylabel('y'); zlabel('z');

% Center frequency
[kx, ky, kz]
% The position of the marble at the 20th data measurement.
solution = [position(20,1), position(20,2),position(20,3)];
```