Application of Time-frequency Analysis: Locating Marble inside a dog

Quinn Luo

January 25th 2019

Abstract

This following project intends to extract useful information from fairly noisy data to track the trajectory of a marble within my dog's body using Fourier transformation. I will discuss the theoretical background regarding Fourier transform first, followed by averaging frequency content to denoise the data to get the center frequency. Then I will filter around the center frequency to locate the marble at each stage and determine the last-observed location.

Introduction

My dog Fluffy accidentally swallowed a marble that was stuck in his stomach. An ultrasound equipment was employed to detect its location and trajectory. Ultrasound emits a specific frequency that bounces back when hitting surfaces. However, because the dog is constantly moving and the fluids inside its body are flowing around, the data observed contain much noise. The following process that involves Fourier transformation help us to extract out useful information from to save the dog.

Background

- Limitations of analysis in time domain

  A wave, according to Fourier, can be represented by a set of cosine and sine functions in time domain, making the analysis impossible to conduct because the behavior of the sum of period functions is not intuitively understandable. However, Fourier transformation translates each cosine and sine function in time domain to a constant in frequency domain, which facilitates our analysis.

- Fourier series and its parameters

  Fourier series transform a function to the sum of cosine and sine functions with frequency w_0 by the following way:

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos nx + b_n \sin nx \tag{1}$$

  By multiplying both side by $\cos mx$ and integrate them from $-\pi$ to π, we can reach the parameters a_n and b_n given by the following:

$$a_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \cos nx dx \tag{2}$$

$$b_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \sin nx dx \tag{3}$$

  More generally, a Fourier transform on $-\infty$ to ∞ takes this form:

$$F(k) = \frac{1}{\sqrt{2\pi}} \int_{-\pi}^{\pi} f(x) e^{-ikx} dx \tag{4}$$

  Until now, we can transform data points between time domain and frequency domain. The command `fft` is an algorithm that transforms points to Fourier modes in Fourier domain. `fft` has a low floating-point operation count of $\mathrm{O}_N \cdot \log N$. Besides, `fft` is able to operate on the domain of $[-L, L]$ and displays a periodic behavior if the

number of points is 2^n for some $n \in \mathbb{Z}$. To point out is that the function `fft` shift the domain in a way that the left half and the right half exchange their position. This inconsistency can be resolved with `fftshift`.

- Filter around the target frequency Once we determine the frequency we want to explore around, a filter can be placed to intensify around that frequency. A common one is Gaussian filter e^{-cx^2}. With maximum value 1 and bell-shaped curve, the function gets magnified around the center.

Algorithm and results

- Average the spectrum to determine the center frequency A file that stores data in a matrix has 20 rows. Each row is a vector that can be reshaped into a three-dimensional tensor that stores the frequency content at that time.

    1. Choosing points in spatial domain

       15 is chosen as L so we are exploring on the space of $[-15, 15]$ on dimension of (x, y, z) and I take 2^6 as my number of modes to satisfy the requirement of `fft` command. Then I create $n + 1$ linearly assigned points on $[-L, L]$ and only use the first n points, followed by repeating this routine on all dimensions. Due to the periodicity of Fourier domain, the last mode has the same value of the first mode.

    2. Setting up Fourier modes in frequency domain

       Because `fft` operates on the interval $[-\pi, \pi]$, these equally spaced points must be projected to this 2π interval. Scaling factor $\frac{2\pi}{2L}$ is to be multiplied. Moreover, the points must be shifted to fit with the default order, starting with 0 to the right half followed by the left half and ending with -1, from 0 to $\frac{n}{2} - 1$ followed by $-\frac{n}{2}$ to -1. Also, points with the natural order from small to large are stored in ks. Finally, I use `meshgrid` to construct the coordinates in both spatial and

frequency domain and call them $[X, Y, Z]$ and $[Kx, Ky, Kz]$.

3. Denoising and determining center frequency By reshaping each row into a 3-D tensor and adding up I get a $64 \times 64 \times 64$ tensor. Subsequently, I divide every entry in the tensor by 20 and get the average spectrum. Through averaging the white noise is reduced, almost eliminated and the only significant frequency we can detect is the one reflecting on the marble. After an optional normalization, function `max(abs(Un_ave_norm(:)))` gives me the maximum value and its indices. These indices are then plugged into $[Kx, Ky, Kz]$ to get the frequency signature.

The maximum frequency happens at the $(28, 42, 33)$ index of frequency value $(1.8805, -1.0472, 0)$. Below are the visualization of frequency content obtained from ultrasound data. Command `isosurface` plots the content with corresponding isovalue. Isovalue is a measure of density and with certain isovalue we can see where the frequency is located.

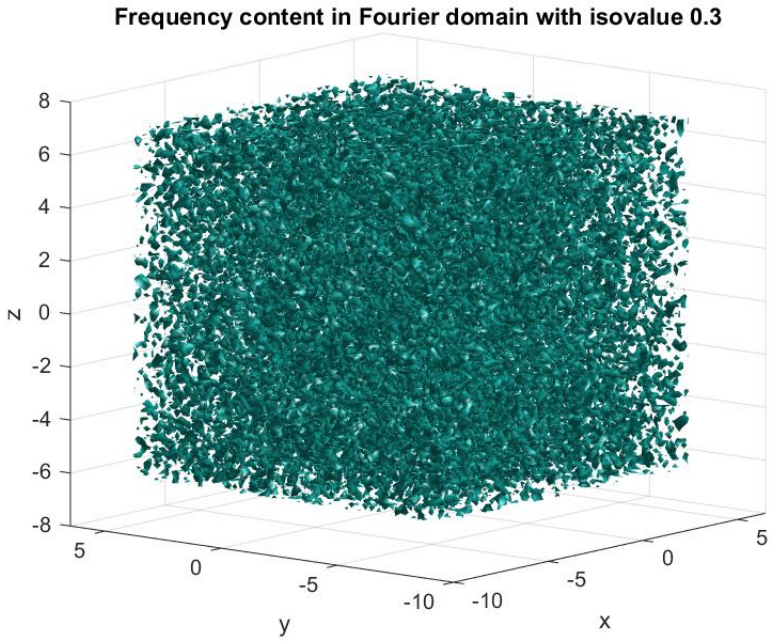

Figure 1: Isosurface of frequency content with isovalue 0.3: an intensity displaying noisy content

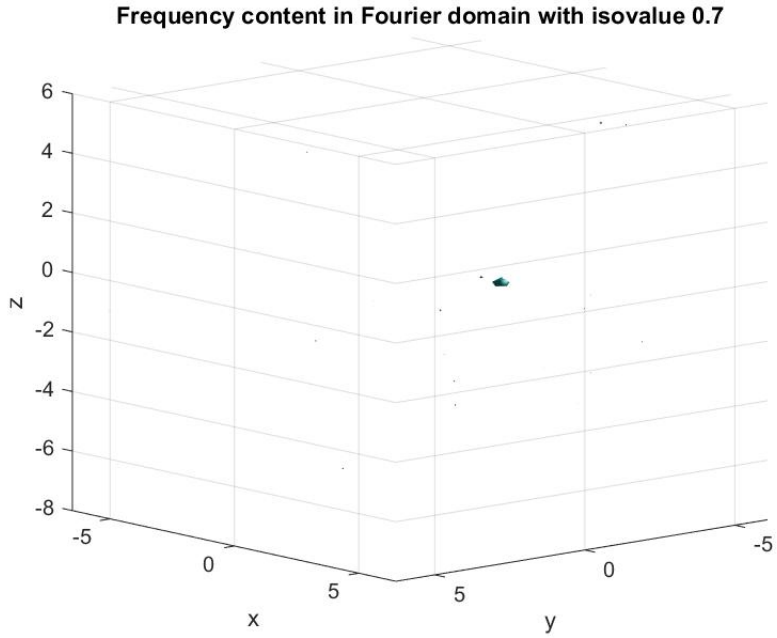

Figure 2: Isosurface of frequency content with isovalue 0.7: an density that produces cleaner image of location

- Plotting the trajectory and finding the last location of the marble

    1. Filtering around the frequency signature

       Since we are able to locate the frequency signature with known indices, we can test at time slot to get the location where that specific frequency happens. Thus, I Fourier transform every tensor I get by reshaping and apply a filter around the frequency. A 3-D Gaussian filter

       $$e^{-0.2[(Kx-1.8805)^2+(Ky-(-1.0472))^2+(Kz-0)^2]}$$

       is used based on the previous results.

    2. Finding the location where target frequency lives

       After filtering each row, I transform the frequency values back to spatial domain. Through the same routine of finding the indices of where center frequency is (achieved by finding the maximum frequency of filtered tensor), I get

three indices and plug them in $[X, Y, Z]$ to obtain spatial information. There are 20 rows so I end up with 20 locations. The last location ends up being at $(-5.6250, 4.2188, -6.0938)$ The trajectory is shown below.

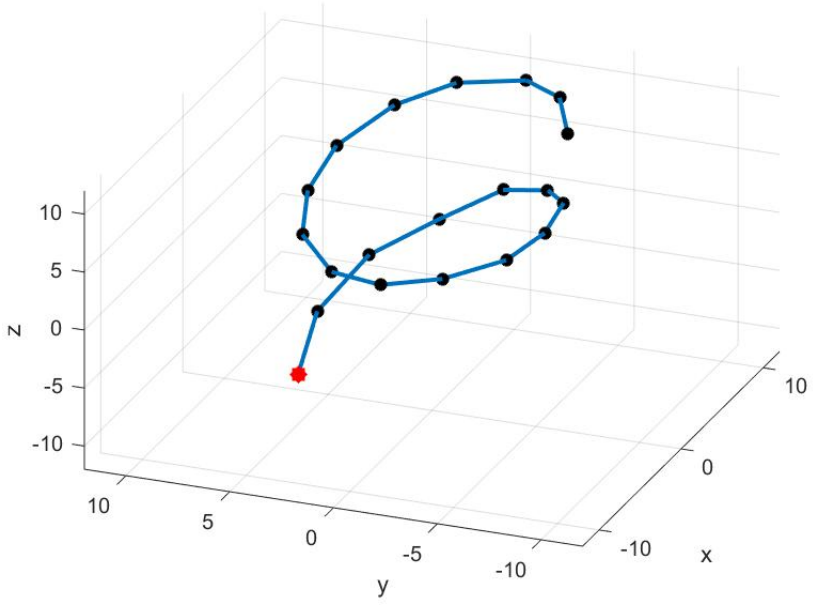

Figure 3: Trajectory of the marble in 20 time slots

Conclusion

In this project, I have analyzed data obtained from ultrasound scanner. First, I create a coordinate system in Fourier domain so that I can transform data between spatial domain and frequency domain. Then, by averaging the frequency content at each time slot, I am able to reduce noise and detect the frequency signature with its location in the data (the indices). Finally, with the target frequency known from the previous part, I filter around the center frequency and locate them at each time slot to create a trajectory and determine the last-observed location of the marble. Fluffy is done being saved.

Appendix I: MATLAB commands explained

- `fft/fftn` returns the Fourier transforms of each column of input matrix, working with different dimensions
- `ifft/ifftn` inverse transformation of `fft/fftn`
- `fftshift/ifftshift` swaps the left half and right half of the vector
- `[M, I] = max(a(:))` stores the maximum value of all dimensions as M an its indices as I
- `[a, b, c] = ind2sub(size, I)` breaks up a number into an array, being used to restore tensor indices here
- `reshape(A, sz)` reshapes matrix A to B of size sz
- `linspace(x1, x2, n)` generates n linearly spaced points between x1 and x2
- `isosurface(X, Y, Z, V, isovalue)` computes isosurface data with specified isovalue
- `plot3` displays three-dimensional plot of data points

Append II: 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);

Un_sum = zeros(n, n, n);

for i = 1:20
    Un_sum = Un_sum + fftn(reshape(Undata(i,:),n,n,n));
end

Un_ave = abs(fftshift(Un_sum))/20;
Un_ave_norm = Un_ave/max(Un_ave(:));
[M, I] = max(Un_ave_norm(:));
[a, b, c] = ind2sub([n, n, n], I);
frq = zeros(3, 1);
frq(1) = Kx(a, b, c);
frq(2) = Ky(a, b, c);
frq(3) = Kz(a, b, c);
isosurface(Kx, Ky, Kz, Un_ave_norm, 0.7); grid on
xlabel('x'); ylabel('y'); zlabel('z');
title("Frequency content in Fourier domain with isovalue 0.7")

%create a filter in frequency field
filter = fftshift(exp(-0.2*((Kx - Kx(28, 42, 33)).^2 + (Ky - Ky(28, 42, 33)).^2 + (Kz -
Unt = zeros(n, n, n);
Untf = zeros(n, n, n);
Unf = zeros(n, n, n);
loc = zeros(3, 20);
for i = 1:20
    Unt(:, :, :) = fftn(reshape(Undata(i,:), n, n, n));
    Untf = filter.*Unt;
    Unf = ifftn(Untf);
    [M, I] = max(abs(Unf(:)));
    [p, q, r] = ind2sub([n, n, n], I);
    loc(:, i) = [X(p, q, r), Y(p, q, r), Z(p, q, r)];
```

```
end


plot3(loc(1, :), loc(2, :), loc(3, :), 'LineWidth', 2);
hold on; grid on;
axis([-12 12 -12 12 -12 12])
xlabel("x")
ylabel("y")
zlabel("z")

for i = 1:20
    plot3(loc(1, i), loc(2, i), loc(3, i), 'k*', 'LineWidth', 2);
end
plot3(loc(1, 20), loc(2, 20), loc(3, 20), 'r*', 'LineWidth', 6);
```

References

Kutz, J. N., (2013). Data-driven modeling scientific computation: Methods for complex systems big data. Oxford: Oxford University Press.