Homework 1

Eli Miller

March 15, 2019

Abstract

The Fourier Transform is an incredibly powerful and easy to implement tool with applications rooted in digital signal processing. By transforming data into the frequency domain, signatures that are moving in the spatial domain with time are fixed in the frequency domain. In this report, I derive some of the theory behind the Fourier Transform and explore it as a tool to understand noisy ultrasound data.

1 Introduction and Overview

In this report I first review theoretical background on the Fourier Transform. I then discuss it's implementation and use for numerical computation. Then, using data from a marble-swallowing dog's ultrasound, I work in the frequency domain to determine a characteristic frequency of a signal. This insight is used to design a filter that works to suppress the noise in the data and back out cleaner spatial data.

Our specific problem of interest is saving our poor dog Fluffy, who has swallowed a marble. We have obtained noisy ultrasound data from Fluffy's intestines, and our objective is to find the marble's characteristic frequency, filter around that frequency, and extract a path of the marble through the intestines. We can then determine where and when to focus intense waves to break up the marble and save Fluffy!

2 Theoretical Background

The Fourier Transform and its inverse are defined as

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

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

While the transform is defined over the entire real line, for computation we will only look at a finite spatial domain $x \in [-L, L]$. Likewise, in the frequency domain, we limit the frequency domain $k \in [-\frac{n\pi}{2L}, \frac{n\pi}{2L}]$, which has been scaled by π/L because of how the FFT algorithm expects a 2π periodic domain.

Once in the frequency domain, the goal becomes to filter out noise so that meaningful data can be pulled back into the spatial domain using the Inverse Fourier Transform. One way of doing this is through the use of a Gaussian filter. In 1-dimension, this takes the form of

$$\mathcal{F}(k) = \exp\left(-\tau(k - k_0)^2\right) \tag{3}$$

with filtering occurring around a characteristic frequency k_0. This filter can be extended to higher dimensions quite easily. We will use a 3-dimensional Gaussian as a method of filtering in the frequency domain, which takes the form

$$\mathcal{F}(k) = \exp\left(-\tau((k_x - k_{x0})^2 + (k_y - k_{y0})^2 + (k_z - k_{z0})^2\right) \tag{4}$$

with filtering occurring around a 3-D characteristic frequency k_0.

3 Algorithmic Implementation and Development

3.1 Data Preparation

Before applying transforms, we need to set up coordinate systems as determined by the spatial and frequency resolution of our measuring equipment. As such, we create a spatial domain where each axis $x, y, z \in [-L, L]$, where L is given to be 15. Similarly, our frequency domain $k_x, k_y, k_z \in [-n/2, n/2]$; however, the fft method expects a 2π periodic domain. The frequency domain is reshaped to be 2π periodic. Additionally, we must reshape the data from a 20 by 64^3 array into 20 arrays of size $(64, 64, 64)$. Let S_i represent the spatial data

3.2 Transform to Frequency Domain

For each time slice, take the 3-D Fourier Transform (FFTN). This outputs a shifted frequency spectrum of our spatial data, with amplitude of alternating signs. In practice we need to shift the output of this transform to be zero-centered, and take the absolute value in order to produce the expected frequency-space representation. The importance of this shifting comes up in filtering, where the functional form of the filter assumes a zero-centered frequency-space representation.

$$U_i = \text{fftshift}(F(S_i)) \tag{5}$$

3.3 Determine Characteristic Frequency

We assume that our signal is composed of a response at a characteristic frequency and random noise at all other frequencies. Assuming that this noise is truly random with mean centered at zero, we take the average over all time slices frequency representation and normalize. We expect that the average contribution of the noise at each frequency will tend toward zero, while the characteristic frequency will remain constant.

$$U_{ave} = \frac{1}{\max(|U_{ave}|)} \sum_{i=1}^{20} U_i \tag{6}$$

This results in a frequency-space representation with a strong component corresponding to the characteristic frequency K_0. To find k_0, simply take the index of the maximum component value.

$$U_{ave}(k_0) = \max(U_{ave}) \tag{7}$$

3.4 Filter Around Characteristic Frequency

Using the determined characteristic frequency, we construct a Gaussian filter around k_0 as shown in equation (4). It is important to recognize that in addition to filtering around the correct frequency k_0, we need to select an appropriate bandwidth τ. Filtering in the frequency domain is a simple component-wise multiplication. For each time slice $i \in [1, 20]$ the filtered frequency component is given

$$U_{filtered,i} = \mathcal{F}(k_0) \cdot U_i \tag{8}$$

As stated above, it is important to note that this method of filtering expects that the frequency representation will be centered at zero.

3.5 Transform to Spatial Domain

To extract meaningful data, we need to transform from frequency back to spatial domain. This process employs the Inverse Fourier Transform (IFFT). It is important to note that in practice, the IFFT expects a shifted frequency-space representation of the signal. Thus, the filtered frequency data is 'unshifted' before transforming back to spatial components.

$$S_{filtered,i} = f(\text{fftshift}(U_{filtered,i})) \tag{9}$$

The marble's spatial position is computed by taking the magnitude of the filtered spatial data at each timestep. To find the marble's position, we simply find the maximum component at each timestep. This corresponds to the filtered data signature of the marble's location, and is used in figure 1 to plot the marble's path over time.

4 Computational Results

Through averaging, it is determined that the characteristic frequency $(k_{x0}, k_{y0}, k_{z0}) = (3.76, -2.09, 0.0)$

The plot of the marble's path is shown in figure 1. To aid in visualization, traces of the path on the coordinate planes are shown in figure 2.

At time $t = 20$ an intense acoustic wave should be focused at the marble's final location of $(4.22, -5.63, -6.09)$.

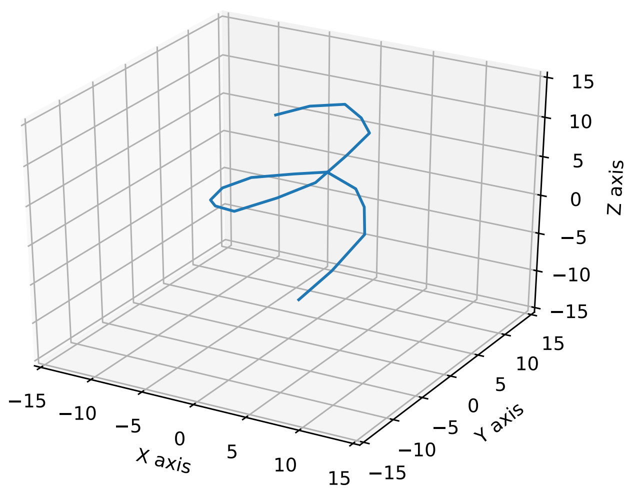

Figure 1: Plot of marble position extracted from filtered data

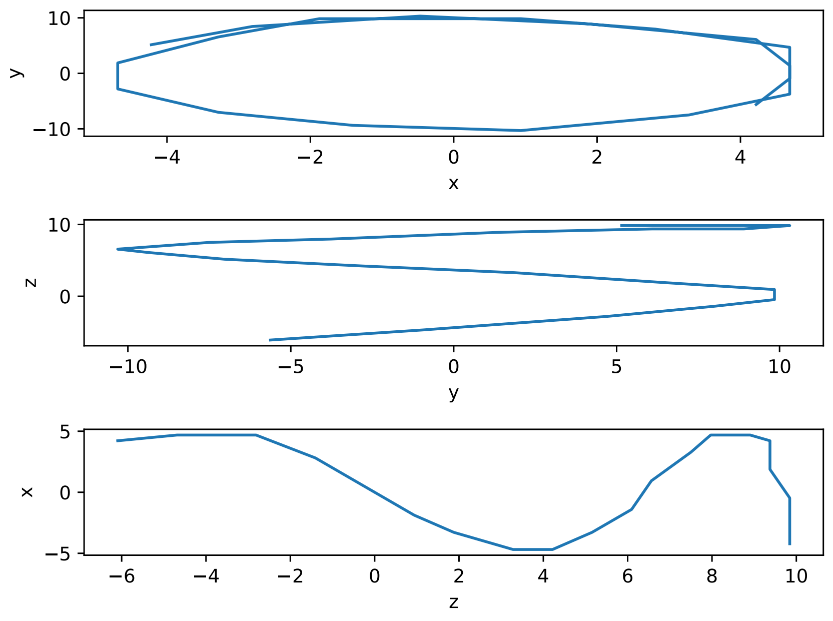

Figure 2: Projections of marble path on principal coordinate planes

5 Summary and Conclusions

This analysis employed the use of the Fourier Transform to change coordinate systems from spatial to frequency space, determine characteristic frequency of an object, filter, and determine the path of an object over time.

Like any application, problem-specific details arise with computational implementation. It is important to remember that the Fast Fourier Transform expects a 2π periodic domain, and we must reshape our frequency space accordingly. Additionally, FFT outputs a shifted frequency representation with oscillating sign. The output must be shifted manually to a zero-centered frequency representation for filtering and visualization. While not as critical in this problem, bandwidth selection and filter design can play a major role in what we are able to pull from noisy data. We can use our knowledge of the system and some Electrical Engineering friends to help guide our decisions in future analysis.

In Python, re-indexing commands may behave differently than Matlab or other languages. While this can be controlled by toggling FORTRAN and C style indexing, it an important place to check that the code is performing as expected. Additionally, the lack of a 1-line isosurface function requires a bit more work for data visualization.

At present, the methods used to determine characteristic frequency and position work well for the current problem, but rely heavily on well-behaved data. The Gaussian filter works well for this dataset as the noise appears to be distributed randomly and drop out significantly after only 20 timesteps of averaging. This allows the application of the simple low-pass style filter. If the noise's frequency content evolved dramatically over time, other filtering techniques may be required.

In addition, the determination of the characteristic frequency simply takes the maximum value. Other methods to be explored could be a weighted average of all values above a threshold, or other more sophisticated criteria for determining the proper frequency around which to filter. This discussion extends to the determination of the marble's position from the cleaned spatial data. Current methods use a single point maximum signature as the position. Because we know that the marble occupies some finite volume, we could use this information to shape our analysis. We could again take a weighted average or leverage stochastic-style location determination.

6 References

J. N. Kutz, *Data-driven modeling & scientific computation: methods for complex systems & big data.* Oxford University Press, 2013.

7 Appendix A: Functions Used

For this analysis, I relied heavily on the numpy library for python, and a bit on the scipy.io module. Below are some specific functions used.

`np.fft.fftn`: returns the n-dimensional Fourier Transform. This function does not require that the dimension is given explicitly.

`np.fft.fftshift`: returns a zero-centered frequency domain representation of our data.

`np.argmax`: returns the maximum argument of an array and its flattened index.

`np.unravel_index`: returns the index in given dimensions from the flattened index (options for indexing style). Useful in combination with argmax for multi-dimensional arrays.

`scipy.io.loadmat`: loads matlab files into a dictionary with relevant information. The key 'data' extracts the necessary variables.

8 Appendix B: Code

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri Jan 11 12:34:00 2019

@author: elimiller
"""

import numpy as np
from numpy import pi
import scipy.io as sio
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

#Import matlab datafile
mat_contents = sio.loadmat('Testdata.mat')
data = mat_contents['Undata']

PLOTPATH = True
SAVEPATH = False

L = 15
n = 64

#Create spacial grid in a n,n,n space
x = np.linspace(-L, L, n+1)[0:n]
y = x
z = x
X, Y, Z = np.meshgrid(x, y, z)

#Create frequency space noramlized to 2pi
k = pi/L * np.concatenate((np.arange(0, (n/2)), np.arange(-n/2, 0)))
#fftshift frequency domain
```

```
ks = np.fft.fftshift(k)
kx, ky, kz,  = np.meshgrid(ks, ks, ks)
average = np.zeros((n, n, n))


for j in range(len(data)):
    #Take a time slice of a (n,n,n) spacial data
    u = np.reshape(data[j, :],(n, n, n))
    #3d fft
    u_transform = np.fft.fftn(u)
    #fftshift of freqnecny data
    u_transform_shift = np.fft.fftshift(u_transform)
    #add each component to average out white noise
    average = average + u_transform_shift
#normalize the averaged data
normalized_average = np.abs(average / np.max(np.abs(average)))
#Find max of average data to have center of filter
max_index = np.unravel_index(
        np.argmax(normalized_average),(n,n,n), order='F')
#make a gaussian filter
k0 = [kx[max_index], ky[max_index], kz[max_index]]

def gauss_filter(noisy_data, index, alpha=.2, inputshifted=True):
    kxmax = kx[index]
    kymax = ky[index]
    kzmax = kz[index]
    filt = np.exp(-alpha*((kx-kxmax)**2 + (ky-kymax)**2 + (kz-kzmax)**2))

    if inputshifted:
        noisy_data = np.fft.fftshift(noisy_data)
    ReturnThis = noisy_data * filt
    return ReturnThis

#initalize data storage arrays before loop
pt_index = np.zeros((20,3))
pt_location = np.zeros((20,3))
filt_spacial = np.zeros_like(data)

#This loop does the same as above, with added returning to spatial domain
#We could probably eliminate with shaping as (20, 60, 60, 60)
for j in range(len(data)):
    spacial = np.reshape(data[j, :],(n, n, n))
    spacial_transform = np.fft.fftn(spacial)
    filtered_data = gauss_filter(spacial_transform, max_index)
    filtered_spacial = np.fft.ifftn(filtered_data)
    filt_spacial[j,:] = filtered_spacial.reshape(1, n**3)
    index = np.unravel_index(
            np.argmax(np.abs(filtered_spacial)),(n,n,n), order='F')
    pt_index[j,:] = index
    pt_location[j,:] = [x[index[0]], y[index[1]], z[index[2]]]

#Get the locaiton at time t=20
final_location  = pt_location[-1,:]
print('the marble\'s final location is' + str(final_location))

#Plot output
if PLOTPATH:
    fig = plt.figure(1)
    ax = fig.add_subplot(111, projection='3d')
    ax.plot(pt_location[:,0], pt_location[:,1], pt_location[:,2])

    ax.set_xlim(-L, L)
    ax.set_ylim(-L, L)
    ax.set_zlim(-L, L)
    ax.set_xlabel('X axis')
    ax.set_ylabel('Y axis')
    ax.set_zlabel('Z axis')


    fig2 = plt.figure(2)
    label = [['x', 'y'],['y','z'],['z', 'x']]
    for i in range(3):

        ax2 = plt.subplot(3,1,i+1)
        if i <= 1:
            ax2.plot(pt_location[:,i], pt_location[:, i+1], c='C0')
        else:
            ax2.plot(pt_location[:,i], pt_location[:, 0], c='C0')
        ax2.set_xlabel(label[i][0])
        ax2.set_ylabel(label[i][1])
        plt.tight_layout()

    if SAVEPATH:
        fig.savefig('marble_path.pdf')
        fig2.savefig('marble_traces.pdf')
```