AMATH 482 HW1
An Ultrasound Problem

Oliver Speltz

January 25, 2019

I Introduction

My dog Fluffy has swallowed a marble! At the vet's office, he is moving around too much on the counter for the ultrasound to get a good read on where it is. In order to save my dog, I must denoise the data using the Fourier transform.

II Theoretical Background

The Fourier transform is a key piece of technology in signal processing. In its essence, it takes an interval of any function and turns it into a representation of the function in sines and cosines. The discrete form of the Fourier transform of some function $f(x)$ over the interval $[-L, L]$, looks like this:

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

The Fourier transform is like a change of coordinate systems, where our new coordinates are the different frequencies of sines and cosines and the a_n and b_n are the distance in each direction. If you let $L \to \infty$, this becomes a transform of the function over the whole line.

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

Now we've taken our function $f(x)$ and have a new representation of it in the k domain. The value $F(k)$ is like the strength of the frequency k in our original function. Before it was discrete, now we have a continuum of frequencies. Fortunately, the Fourier transform is linear, so we can go back from this k domain to our original x domain. This is done with the inverse Fourier transform.

$$f(x) = \int_{-\infty}^{\infty} F(k) e^{ikx} dk \tag{3}$$

In practical terms, this allows us to examine a signal in the frequency space, potentially make some modifications to the frequencies, then go back to the original domain and see the modified signal. A very fortunate consequence of the Fourier transform is that if a signal is moving in space (or time), the frequency of the signal is the same. An example of this phenomenon is given in Figure 1, on the left in blue are two Gaussian bell curves centered at 0 and 3, then on their right are their Fourier transforms, which are identical.

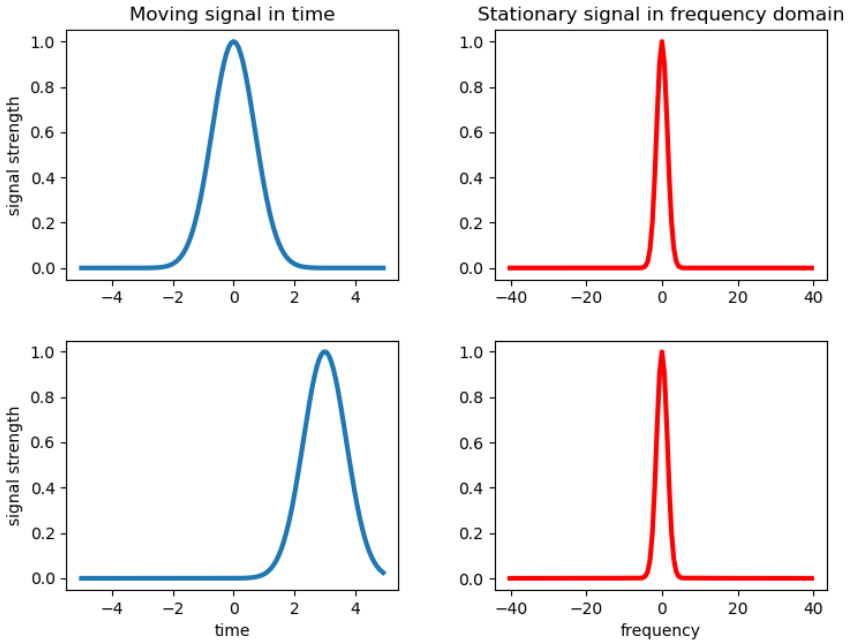

Figure 1. On the left, similar signals centered at different points in time, on the right, the Fourier transform of each is the same.

In most applications, you have many different readings of your signal at different points in time. If your signal also contains noise, it may be hard to distinguish your signal from the junk. However, if you take the Fourier transform of your signal at many points in time, the signal will remain stationary. This allows you to sum up many points in time and the noise will average out, leaving only the signal. An important thing to note, however, is that this will only work if the frequency of the signal is actually not changing. Once you have an idea of the frequency of your signal, you can apply a filter around this frequency to clean up the data. A filter, as seen in Figure 2, is a function that you multiply your signal by to set mostly everything but a specific range to zero. So, if you were to apply a filter centered around the maximum of average frequencies of your signal over time, you would in theory be filtering out the noise and keeping your signal intact. Then due to the linearity of the Fourier transform, we can inverse this filtered signal to see how it looks in the domain it was recorded in.

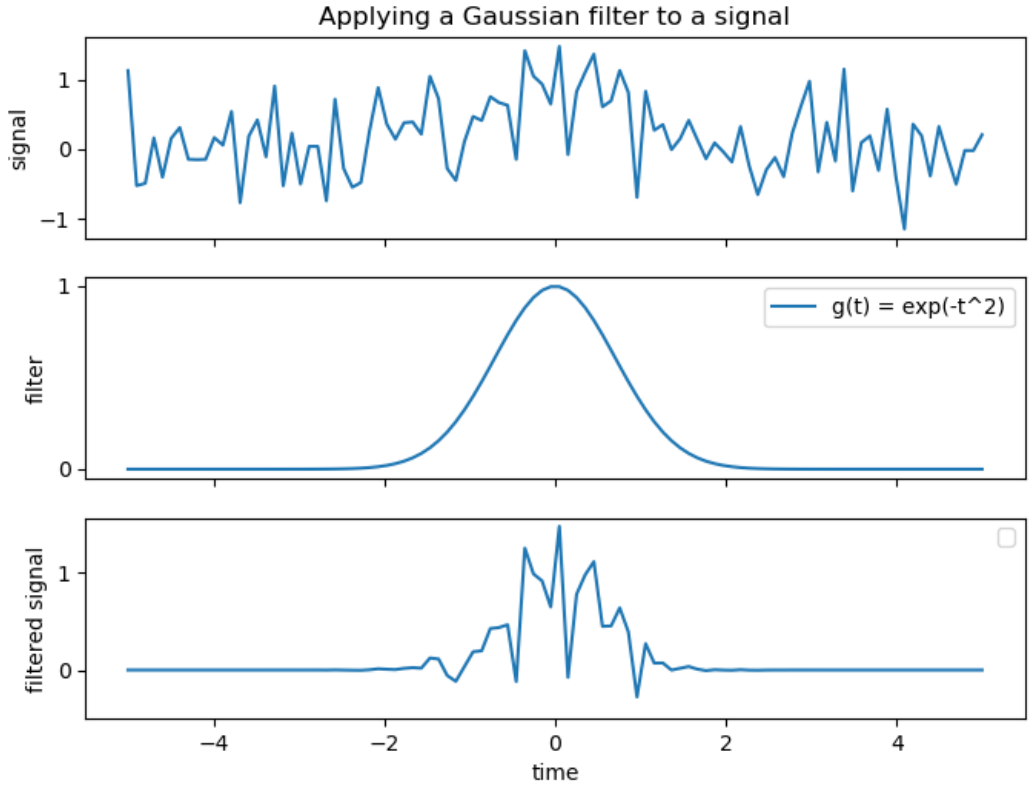

Figure 2. Applying a Gaussian filter centered at zero to the noisy signal at top.

III Algorithm Implementation and Development

We can take advantage of these characteristics of the Fourier transform to save my poor dog Fluffy. He is moving around too much for us to make any sense of the readings in the spatial domain, luckily we have 20 different readings from different points in time. If we take the 3-dimensional Fourier transform of each time slice, the frequency center of the marble should stay put. Then, we can add up the transform of each point in time to average out the noise and find the center. Once we have a center established, we can create a filter around that point and filter everything in the frequency space that isn't near it. Finally, taking the inverse transform of the filtered signal, we have a more clear idea of the path of the marble as it wrecks havoc through Fluffy's digestive system.

Implementing this in in Python is easily done thanks to the `numpy`, `scipy.io` and `scipy.fftpack` modules. First load the MATLAB data contained in `Testdat.mat` with `scipy.io.loadmat()`. After reshaping the data to be 20 64x64x64 cubes, we can use `scipy.fftpack.fftn()` to take the Fourier transform of each time point, then sum them all up. The location of the maximum value of this sum is the central frequency of the marble. We can then make a Gaussian filter centered around this frequency and multiply our transformed signal by it. Reversing the transform using `fftpack.ifftn()` gives us a filtered signal in the spatial domain. Taking the indices of the maximum of this cube of data and seeing where they relate to in the X, Y and Z directions will give us the trajectory of the marble in my dog.

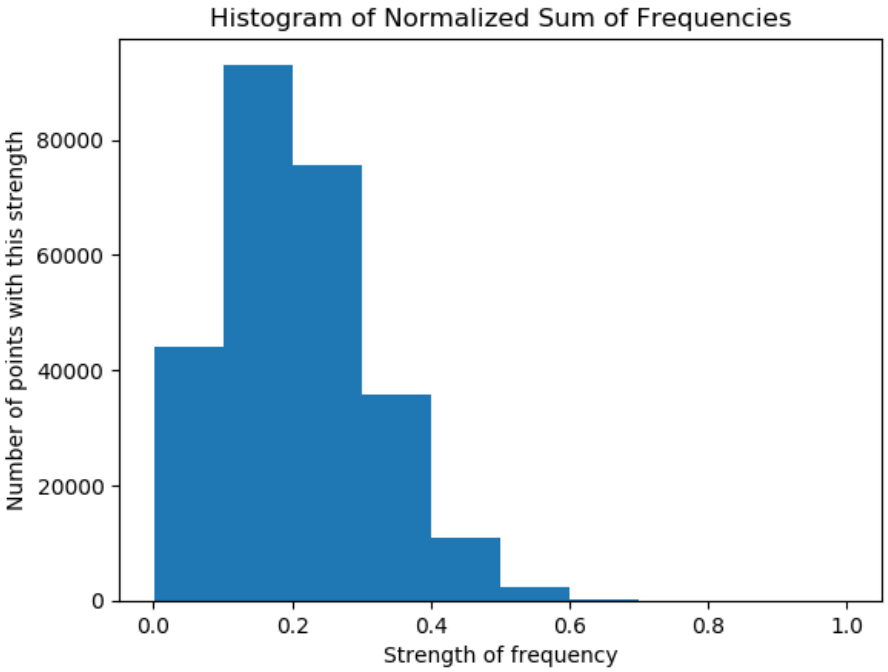

Figure 3. This histogram shows that very few data points were close to the maximum data point, meaning it is well distinguished.

IV Computational Results

Using this technique, I found the central frequency of the marble to be:

$$(k_x, k_y, k_z) = (0, 1.885, -1.047)$$

As seen in Figure 3, when the averaged frequencies are normalized to the maximum, most points have frequencies around 0.2. This indicates that the maximum is likely significant. Applying the filter around the location of this maximum and then looking at the filtered signal in the space domain we see a clear helical trajectory of the marble. In Figure 4 you can see a 3D plot of the marble's trajectory.

Now that we have located the trajectory of the marble, we must act fast and destroy it before it moves too much more. At the last time point, the marble is located at

$$(x, y, z) = (-6.09375, -5.625, 4.21875).$$

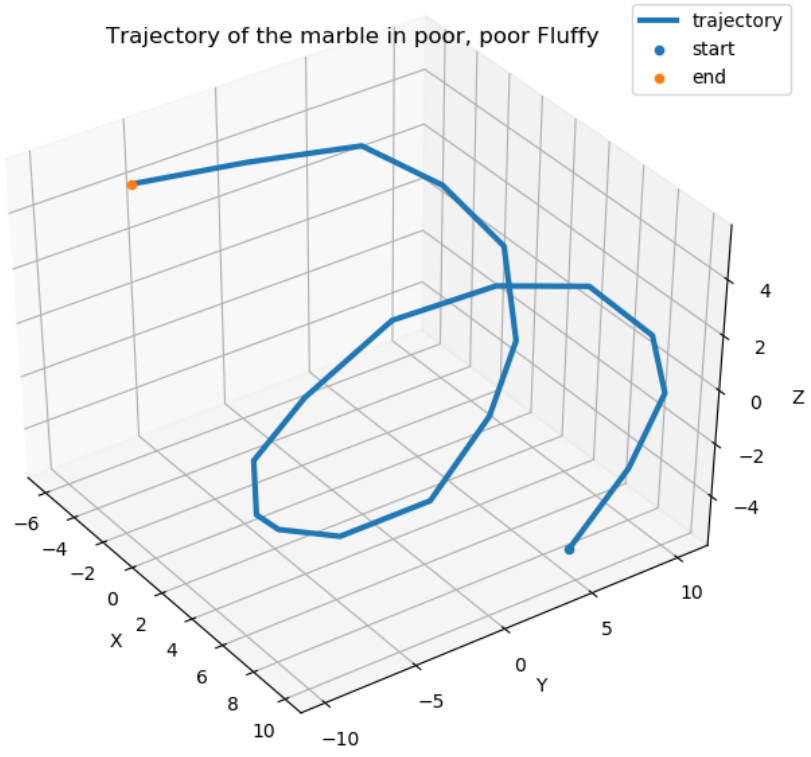

Figure 4. The filtered signal shows the marble is moving in a clear helical pattern down Fluffy's small intestine.

V Summary and Conclusions

In summary, the Fourier transform is a useful tool for picking out unchanging signals. If the frequency of a signal is constant, then even if it is moving in space or time, the Fourier transform can be used to isolate and denoise it.

There is quite a bit of room for refinement and tweaking in this algorithm. One example is the filter size used. I used 0.2 rather arbitrarily, but in my experiments with different sizes, I found that larger filters resulted in a similar helical shaped trajectories but more jagged and noisy. Wider filters tend to let more noise in. Too small of a filter and the trajectory would become complete nonsense and be all over the place. So too small of a filter doesn't let enough of the signal through. One could also use completely different filters besides the Gaussian if different aspects of the data needed to be captured. Another area for improvement would be in the selection of the coordinates of the marble. In this implementation, I selected the maximum signal strength of each time point to be the location of the marble. Perhaps a more accurate way to make this selection would be to take a weighted average of the signal over the space. Essentially, think of the signal strength as a density function and find the "center of mass" of the signal, or a subset of the signal, in which it is over some threshold.

In conclusion, I am so grateful that a French guy thought of this 200 years ago and saved my dog.

Appendix A: Python Functions Used

`numpy.meshgrid(x,y,z,...)`
: Creates grids that are defined by the space created by `x,y,z...` Each grid is `(len(x),len(y),len(z),..)` in shape, the first varies only in dimension 1, the second only in dimension 0, the third and following vary only in the $d-1$ dimension. Use to set up a 2d or 3d coordinate space over which to apply functions.

`scipy.fftpack.fftn(x,axes=None)`
: Does the fast Fourier transform in n dimensions. By default it uses all the dimensions of `x`, if axes is supplied, it does the fft in those directions only, maintaining the shape in the axes left out. Returns the Fourier transform in the shape of `x`, or in the shape of the axes of `x` provided. The dimensions of `x` should be a power of two. Returns the transform in the butterfly shifted pattern.

`numpy.argmax(a)`
: Returns the index of the maximum element in the array `a`, the index is into the flattened array.

`scipy.fftpack.ifftn(x,axes)`
: Works in a similar manner to `fttn()` but returns the inverse Fourier transform. Assumes the input is in the butterfly shift.

Appendix B: Main Python Code

```
import numpy as np
import scipy.io as sio
import scipy.fftpack as ft
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D # for 3d plotting

L = 15 # spatial domain
n = 64 # fourier nodes

# set up spatial domain, throw out last point because fourier assumes a
# periodic domain, that is f(-L) = f(L)
x = np.linspace(-L,L,n,endpoint=False); y = x.copy(); z = x.copy();
# set up frequency domain, fft assumes a interval of 2pi so we must rescale
# our 2L interval accordingly
# the fft algorithm puts things out of order, "butterfly shift"
k = np.pi/L * np.hstack((np.arange(0,n/2),np.arange(-n/2,0)))

# set up grids for X,Y and Z so any point is accesses as u[xi,yi,zi]
Y,X,Z = np.meshgrid(x,y,z)
# use butterfly shifted domain so we dont have to shift our transforms later
Ky,Kx,Kz = np.meshgrid(k,k,k)

# load data from .mat file and reshape it so it has the form
# [time slice,x,y,z]
dat = sio.loadmat('Testdata.mat')
un = dat['Undata'].reshape((20,n,n,n))

# fourier transform on the 1st, 2nd and 3rd axes so that the 0th axis,
# our time slices, is maintained
unt = ft.fftn(un,axes=(1,2,3))
```

```
# sum up all the time slices to average out the noise
ave = np.sum(unt,axis=0)

# plotda a histogram of the normalized signal strength
#a = (np.abs(ave)/np.max(np.abs(ave))).flatten()
#plt.hist(a)

# get the index of the maximum value of the average transformed signal
# and then get the corresponding (kx,ky,kz)
ind = np.argmax(ave)
x_cen = Kx.flatten()[ind]
y_cen = Ky.flatten()[ind]
z_cen = Kz.flatten()[ind]
print('The central frequency of the marble is:')
print('kx =',x_cen,'ky =',y_cen,'kz = ',z_cen)

# make a Gaussian filter around this frequency center
filt = np.exp(-0.2*((Kx-x_cen)**2 + (Ky-y_cen)**2 + (Kz-z_cen)**2))

# apply the filter to each slice of time in the frequency domain
untf = filt*np.ones_like(unt)*unt
# reverse the transform so we have our filtered signal in the spatial domain
unf = ft.ifftn(untf,axes=(1,2,3))

coords = np.zeros((20,3))
for j in range(20):
    # take the maximum value of the filtered signal to
    # to be the location of the marble
    ind = np.argmax(np.abs(unf[j]))
    coords[j,:] = [X.flatten()[ind],Y.flatten()[ind],Z.flatten()[ind]]

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot(coords[:,0],coords[:,1],coords[:,2],label='trajectory',
        linewidth=3)
ax.scatter(coords[0,0],coords[0,1],coords[0,2],label='start')
ax.scatter(coords[19,0],coords[19,1],coords[19,2],label='end')
ax.legend()
ax.set(xlabel='X',ylabel='Y',zlabel='Z',
       title='Trajectory of the marble in poor, poor Fluffy',
       yticks=np.arange(-10,11,5))
plt.show()

print('To break up the marble with an acoustic wave, we should place it at:')
print('x =', coords[19,0], 'y =', coords[19,1], 'z = ', coords[19,2])
```