# AMATH 582 - Homework 1 - An ultrasound problem (vF)

February 6, 2020

**Github**
Link: https://github.com/b-goldney/AMATH_582
Username: b-goldney

**Abstract:**
The purpose of this assignment is twofold: i) to highlight some of the math behind Fourier Transforms and ii) identify the path of a marble which was ingested by a dog.

Fourier Series and Fourier Transforms are considered to be one of the "secrets of the universe". A Fourier Series is used to represent a periodic signal as a sum of *sin* and *cos* waves. The Fourier Transform is utilized to represent an arbitrary function over the real number line.

Regarding the application of Fourier Transform in this paper, data is provided from 20 images that were taken via ultrasound of a dog that swallowed a marble. However, the dog was moving during the images causing the signal from the ultrasound to be noisy. Fourier Transform is used to determine the location of the marble. The spectrum is averaged and Fourier Transform is applied to determine the location of the marble. There are 20 images captured and each image contains 262,144 data points (a 64,64,64 image).

## Sec. I. Introduction and Overview

The Fourier Transform is a technique to transorm a function of time, $x(t)$, to a function of frequency, $X(\omega)$. The benefit of switching from the time domain to the frequency domain (and vice versa) is obvious. However, the fact that any function can be written as a sum of *sin* and *cos* waves is not obvious. This paper will aim to clarify some of the math behind Fourier Seris and Fourier Transforms, as well as show an example of how this is applicable.

## Sec. II. Theoretical Background

The purpose of this section is to clarify the math underlying Fourier Analysis.

There are couple definitions to clarify first.
- Fourier Transform: decomposes a function of time into its constituent frequencies (i.e. *sin* and *cos* waves). - Fourier Series: is an expansion of a periodic function in terms of an infinite sum of *sin* and *cos* waves.

There are two different, but equivalent, ways to state Fourier Series: Trigonmetric and Exponential:

$$x(t) = a_0 + \sum_{n=1}^{\infty}(a_n cos(n\omega_0 t) + b_n sin(n\omega_0 t)) \longleftarrow \text{Trigonmetric} \quad (1)$$

$$= \sum_{-\infty}^{\infty} c_n e^{ej\omega_0 t} \longleftarrow \text{Exponential} \quad (2)$$

The Fourier Transform is defined as:

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

**Let's look at the basics and a few examples first** - If the periods of two periodic functions do not have a common multiple then their sum is not periodic.
- However, sums of two periodic functions are often periodic. Suppose that $f$ is periodic with period $P$, and $g$ is periodic with at least period $Q$. If $P$ and $Q$ have a common multiple $R$, then $f$ and $g$ are periodic with period $R$.

**Example 1**: Adding 2 waves with different frequencies.

$$1.0Hz + 0.80Hz = 0.20Hz$$

Adding a 1.0 Hz wave to a 0.80 Hz wave results in a wave with 0.20 Hz. This is evidenced by looking at how often the cycles between the waves repeat themselves. The 1.0 Hz wave repeats itself every second; wheras, the 0.80 Hz wave repeats itself every 1.25 seconds (1 second / 0.80 wave = 1.25 seconds per wave). We can see that the two waves will "meet" at 5.0 seconds. The 1.0 Hz wave will be on its fifth cycle, and the 0.80 Hz wave will be on its 4 th cycle (1.25 seconds * 4 cycles = 5.0 seconds). Therefore, the combined wave cycle repeats itself every 5 seconds, resulting in 0.20 Hz (1 wave cycle / 5 seconds = 0.20 Hz). Plotting the charts shows the two waves meet at exactly 5.0 seconds.

$$0.20Hz = 1 \text{ wave cycle / } 5 \text{ seconds}$$

**Example 2: Adding 2 waves with different frequencies**

$$0.40Hz + 1.2Hz = 0.40Hz$$

The 0.40 Hz wave will repeat itself every 2.5 seconds (1 second / 0.40 waves = 2.5 seconds per wave). The 1.2 Hz wave will repeat itself every 0.833 seconds (1 second / 1.2 waves = 0.833 seconds per wave). Multiplying 0.833 * 3 equals 2.5 seconds. Therefore, the two waves will meet every 2.5 seconds, resulting in 0.40 Hz.

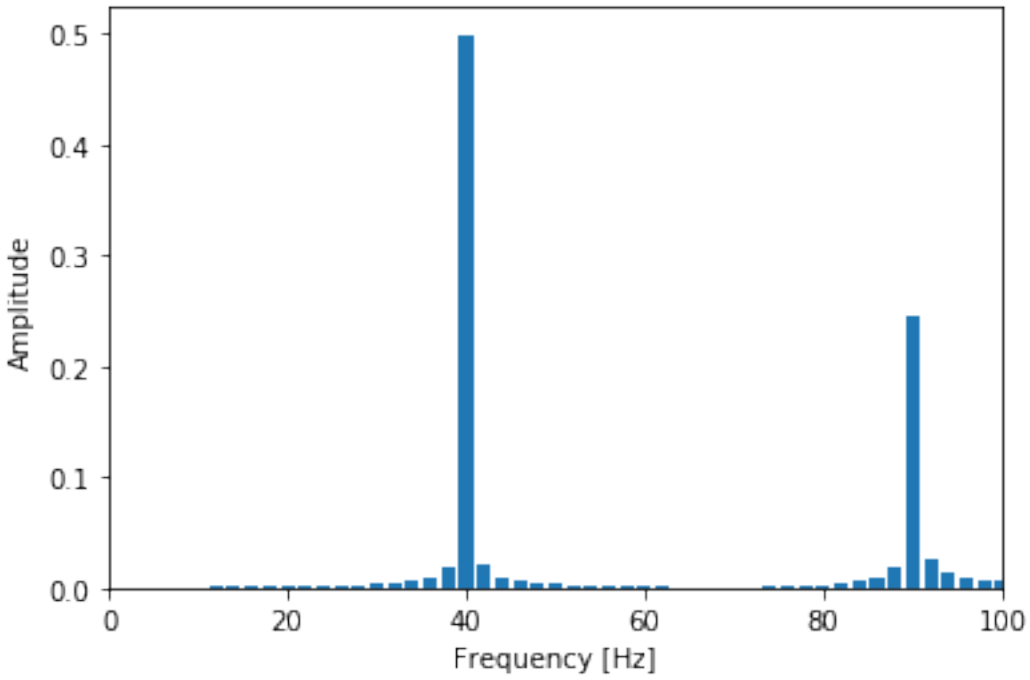
$$0.40Hz = 1 \text{ wave cycle / } 2.5 \text{ seconds}$$

**Example 3: Fourier Transform on two waves added together** - Let's look at how to perform a Fourier Transform on 2 sine waves added together (e.g. a 40 Hz wave added to a 90 Hz wave)

[347]:
```
t = np.linspace(0, 0.5, 500)
s = np.sin(40 * 2 * np.pi * t) + 0.5 * np.sin(90 * 2 * np.pi * t)
#plt.ylabel("Amplitude"), plt.xlabel("Time [s]"), plt.plot(t, s), plt.show()
```

The below graph shows the Fourier Transform correctly caculates there is a 40 Hz and 90 Hz wave

[35]:
```
fft = np.fft.fft(s), T = t[1] - t[0]  # sampling interval
N = s.size, f = np.linspace(0, 1 / T, N) # 1/T is the frequency
plt.xlim(0,100,10), plt.ylabel("Amplitude"), plt.xlabel("Frequency [Hz]")
plt.bar(f[:N // 2], np.abs(fft)[:N // 2] * 1 / N, width=1.5)  # 1 / N is a␣
 ↪normalization factor
plt.show()
```




**Takeaway**
Notice that this only works if the wave frequencies are a scalar multiple of each other. If the waves are not multiples of each other then the waves will never "meet" again, consequently there will be no cycles to repeat.

**Derivations**

**Let's begin by defining Euler's Equation and understanding how complex numbers are introduced into Fourier's Transform**

Let's derive Euler's Equation $e^{ix} = cos(x) + sin(x)$

First, Maclaurin Series on $e^x$:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{x^5}{5!} + ...$$

Second, Maclaurin Series on cos(x):

$$cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \frac{x^{10}}{10!} + ...$$

Third, Maclaurin Series on sin(x):

$$sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} + ...$$

Now, let's rewrite the Maclaurin Series of $e^x$ but multiply every x by $i$.

$$e^{ix} = 1 + ix + \frac{ix^2}{2!} + \frac{ix^3}{3!} + \frac{ix^4}{4!} + \frac{ix^5}{5!} + ... \tag{3}$$

$$= 1 + ix - \frac{x^2}{2!} - i\frac{x^3}{3!} + \frac{x^4}{4!} + i\frac{x^5}{5!} - \frac{x^6}{6!} + ... \tag{4}$$

To rewrite $e^{ix}$, the following facts are used:

$$i^2 = -1 \tag{5}$$

$$i^3 = i^2 * i = -i \tag{6}$$

$$i^4 = i^2 * i^2 = 1 \tag{7}$$

$$i^5 = i^4 * i = i \tag{8}$$

Notice, the Maclaurin Series for the *sin* and *cos* waves are represented by the Maclaurin Series for $e^{ix}$. The first term in $e^{ix}$ is the first term in *cos*. The second term in $e^{ix}$ is the first term in *sin*. The third term in $e^{ix}$ is the second term in *cos*. The fourth term in $e^{ix}$ is the second term in *sin*. So on and so forth. Therefore, the following is true:

$$e^{ix} = (1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!}) + i(x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!}) \tag{9}$$

$$= cos(x) + i\ sin(x) \tag{10}$$

Finally, let's look at the equations to determine the coefficients, $a_n$ and $b_n$, for the trigonmetric definition.

$$a_n = \frac{1}{2\pi}\int_0^{2\pi} f(x)cos(nx)dx, \quad n = 0, 1, 2...b_n = \frac{1}{2\pi}\int_0^{2\pi} f(x)sin(nx)dx, \quad n = 0, 1, 2...$$

**Sec. III. Algorithm Implementation and Development**

Background on Fast Fourier Transform Algorithm. - The FFT algorithm factorizes the Discrete Fourier Transform matrix into a product of mostly sparse factors. - The FFT algorithm computes the DFT in $O(n\ log(n))$ time compared to the DFT which calculates it in $O(N^2)$ - A divide and conquer approach is implemented to reduce computation time

The general process for the algorithm implementation and development is as follows. - Load data and duplicate the code provided in the homework, which provides spatial and spectral resolution of the ultrasound equipment - Resize grid vectors by $\frac{2\pi}{L}$ for the domain because FFT assumes $2\pi$ periodicity - Subsequently, the grid vectors are shifted via np.fft.fftshift because FFT will shift the data - Reshape each row (i.e. image) of data to a 64x64x64 matrix - Once the data is in a

3D matrix, the N-Dimension FFT algorithim is applied - The results of the Fourier Transform are added to a variable which is subsequently normalized by its largest value - The result of the Fourier Transform is multiplied by a Gaussian Filter (noise is assumed to be white) - The inverse of the Fourier Transform is applied - Plot the trajectory of the marble

### Sec. IV. Computational Results

The spatial resolution refers to the number of pixels utilized in construction of a digital image. Spectral resolution describes the ability of a sensor to define fine wavelength intervals.

The center frequency is (4.5, 0.0, -2.5) which is calculated as:

[331]:
```
(ki*L)/(2*np.pi),  (kj*L)/(2*np.pi),   (kk*L)/(2*np.pi)
```

[331]:
```
(4.5, 0.0, -2.4999999999999996)
```

The spatial resolution is 0.46875 which is calculated as:

[333]:
```
x2[1]-x2[0]
```

[333]:
```
0.46875
```

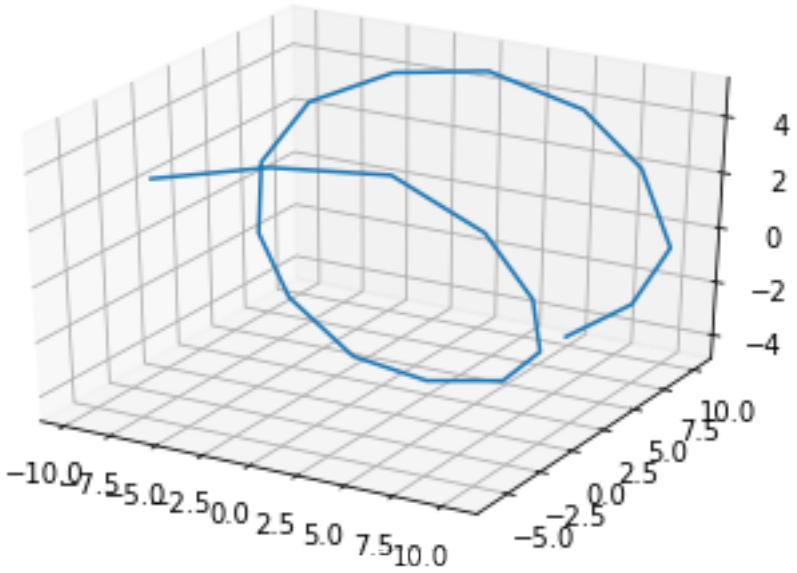
The path of the marble is in a spiral pattern. The coordinates of the last position of the marble are at (-5.625, -6.09375, 4.21875), which are captured in the "marble" variable:

[335]:
```
marble[-1]
```

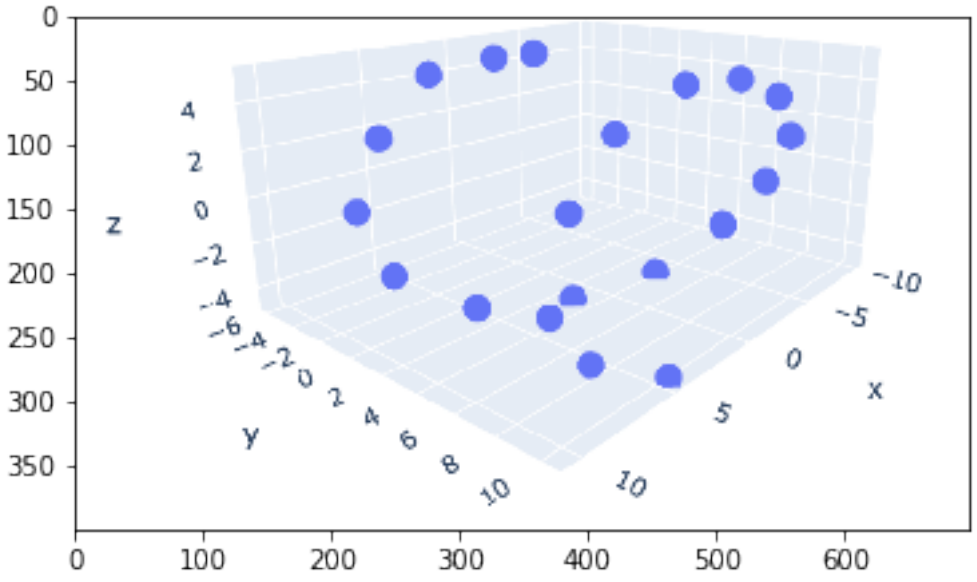
[335]:
```
array([-5.625  , -6.09375,  4.21875])
```

### Plot path of the marble

[297]:
```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
%matplotlib inline
ax = plt.axes(projection='3d'), ax.plot3D(df.iloc[:,0],df.iloc[:,1],df.iloc[:
 ↪,2])
plt.show()
```




[361]:
```
import plotly.graph_objects as go
fig = go.Figure(data=[go.Scatter3d(x=df.iloc[:,0], y=df.iloc[:,1], z=df.iloc[:
 ↪,2],
                                   mode='markers')])
#fig.show()
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('plotly_3d.png')
plt.imshow(img)
plt.show()
```




### Sec. V. Summary and Conclusions

The goal of this assignment was to use Fourier Transform to identify the path of a marble that was ingested by a dog. We denoised data from 20 ultrasound images and applied a Fourier Transform to identify the frequency of the marble. Once the frequency of the marble was identified, we were able to filter out all other data and plot the path of the marble.

A key assumption is that the noise in the images is white (i.e. zero mean).

### Appendix A: Python functions used and brief implementation explanation

- np.asarray: The asarray() function is used to convert an given input to an array. Input data, in any form that can be converted to an array. This includes lists, lists of tuples, tuples, tuples of tuples, tuples of lists and ndarrays
- np.fft.fftn: Compute the N-dimensional discrete Fourier Transform. This function computes the N-dimensional discrete Fourier Transform over any number of axes in an M-dimensional array by means of the Fast Fourier Transform (FFT)
- np.fft.ifftn(): Compute the N-dimensional inverse discrete Fourier Transform. This function computes the inverse of the N-dimensional discrete Fourier Transform over any number of axes in an M-dimensional array by means of the Fast Fourier Transform (FFT). In other words, ifftn(fftn(a)) == a to within numerical accuracy
- np.argmax(): Returns the indices of the maximum values along an axis.
- np.unravel_index(): Converts a flat index or array of flat indices into a tuple of coordinate arrays.

### Appendix B: Python Code

[281]:
```
import pandas as pd
%matplotlib inline
import matplotlib.pyplot as plt
```

```
#plt.style.use('style/elegant.mplstyle')
import numpy as np
```

[282]:
```
Undata = pd.read_csv('Undata.csv', header=None)
Undata.shape # confirms the data has 20 rows and 262,144 columns, which matches␣
 ↪Matlab
Undata.head()
# The code below replaces each "i" with a "j" so we can load the numbers as␣
 ↪complex values and not strings
values = list()
with open('Undata.csv') as h:
    for line in h:
        values.append(eval(line.replace('i', 'j'))) #

Undata2 = np.reshape(values, (20,262144)) # There are 20 rows with 262,144␣
 ↪values (262,144 = 64^3)
Undata2 = pd.DataFrame(Undata2)

Undata = Undata2
del Undata2
```

[318]:
```
# Duplicate Code Provided in homework
L = 15, n = 64

x2 = np.linspace(-L,L,n+1)
x = x2[0:n+1], y=x, z=x

# create the following code for the k variable [0:(n/2-1) -n/2:-1];]
list1 = range(0,32,1) , list2 = range(-32,0,1), list3 = []

for i in list1:
    list3.append(i)

for i in list2:
    list3.append(i)

# Convert list3 to a numpy array so it can be multiplied
array3 = np.asarray(list3)

k = ((2*np.pi)/(2*L)) * array3 # Rescale wavenumbers because FFT assumes 2*pi␣
 ↪periodic signals

ks = np.fft.fftshift(k)
# Create meshgrids
[X,Y,Z] = np.meshgrid(x,y,z)
[Kx,Ky,Kz]= np.meshgrid(k,k,k)
Utave = np.zeros((64,64,64))
```

```
Uj = np.zeros((64,64,64), dtype=complex), ave = np.zeros((64,64,64))
```

[319]:
```
for i in range(0,20,1):
    Uj[:,:,:]= np.reshape(np.array(Undata.iloc[i,:]),(64,64,64))
    ave = ave + np.fft.fftn(Uj)
```

Return the max value and index of the max value

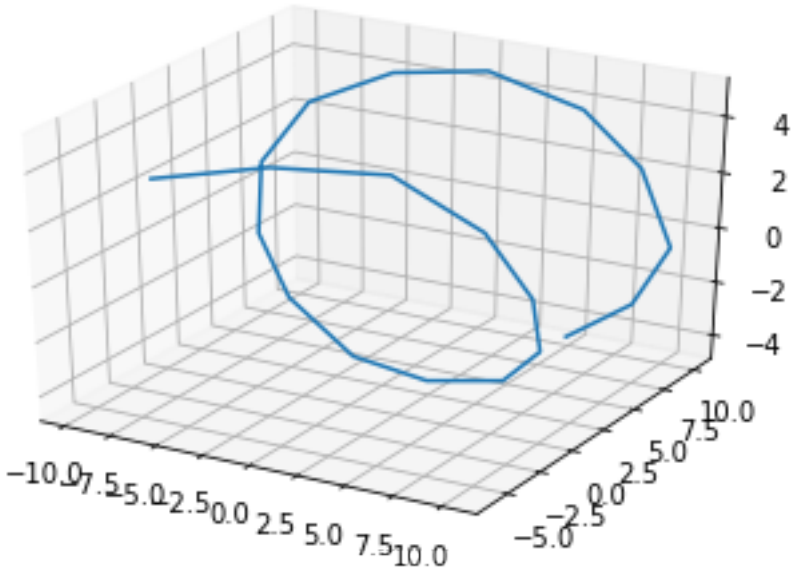
[285]:
```
ave_max = np.argmax(abs(ave.flatten()))
# Unravel takes the max value coordinates and outputs coordinates in a␣
 ↪(64,64,64) coordinate grid
I,J,K = np.unravel_index(ave_max,(64,64,64))
```

[286]:
```
ki = Kx[I,J,K], kj = Ky[I,J,K], kk = Kz[I,J,K]

delta_x = Kx - ki, delta_y = Ky - kj, delta_z = Kz - kk
gaussfilter = np.exp(-1 * ((delta_x**2) + (delta_y)**2 + (delta_z)**2))
```

[305]:
```
marble = np.zeros((20,3))

for i in range(0,20,1):
    Uj = np.reshape(np.array(Undata.iloc[i,:]),(64,64,64))
    Ujtf = np.fft.fftn(Uj) * gaussfilter
    Ujf = np.fft.ifftn(Ujtf)

    ave_max = np.argmax(abs(Ujf.flatten()))

    ave_max = np.argmax(abs(Ujf.flatten()))
    xx,yy,zz =  np.unravel_index(ave_max, [64, 64, 64])
    marble[i,0] = X[xx,yy,zz]
    marble[i,1] = Y[xx,yy,zz]
    marble[i,2] = Z[xx,yy,zz]

df = pd.DataFrame(marble)
```

**Plot path of the marble**

[297]:
```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
%matplotlib inline
ax = plt.axes(projection='3d')
ax.plot3D(df.iloc[:,0],df.iloc[:,1],df.iloc[:,2]), plt.show()
```




[293]:
```
import plotly.graph_objects as go

#fig = px.scatter_3d(df,x='xx', y='yy', z='zz')

fig = go.Figure(data=[go.Scatter3d(x=df.iloc[:,0], y=df.iloc[:,1], z=df.iloc[:
 ↪,2],
                                   mode='markers')])
fig.show()
```

### Appendix C: References


- https://en.wikipedia.org/wiki/Fast_Fourier_transform
- https://lpsa.swarthmore.edu/Fourier/Xforms/FXformIntro.html
- https://ocw.mit.edu/courses/mathematics/18-03sc-differential-equations-fall-2011/unit-iii-fourier-series-and-laplace-transform/fourier-series-basics/
- https://docs.scipy.org/doc/numpy/reference/routines.fft.html