Tutorial 3

March 19, 2019

1 Tutorial 3 Notebook - Fourier Transforms

In this notebook, we will be practicing implementing fourier transforms You need to be able to do this for your first assignment! As usual, please read through the code and understand it.

In [1]:
```
import numpy as np
import scipy as sp
import scipy.fftpack
from matplotlib import pyplot as plt
```

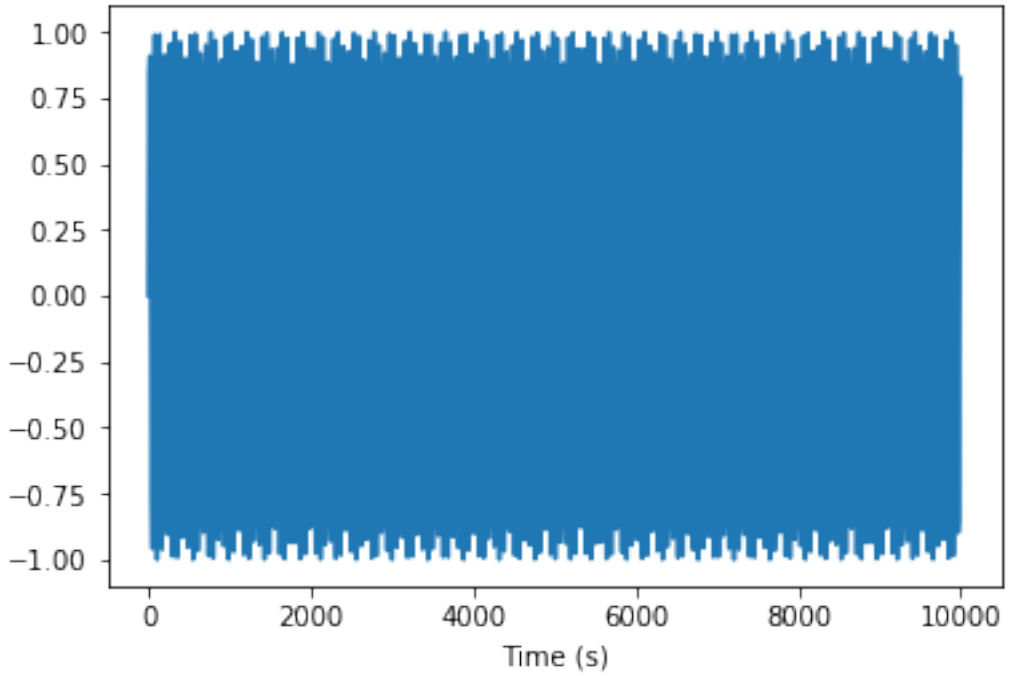
1.1 Part 1 - Baby's First Fourier Transform.

We are going to make a sinusoid with a certain frequency, then see if we can pick up the frequency from our fourier transform

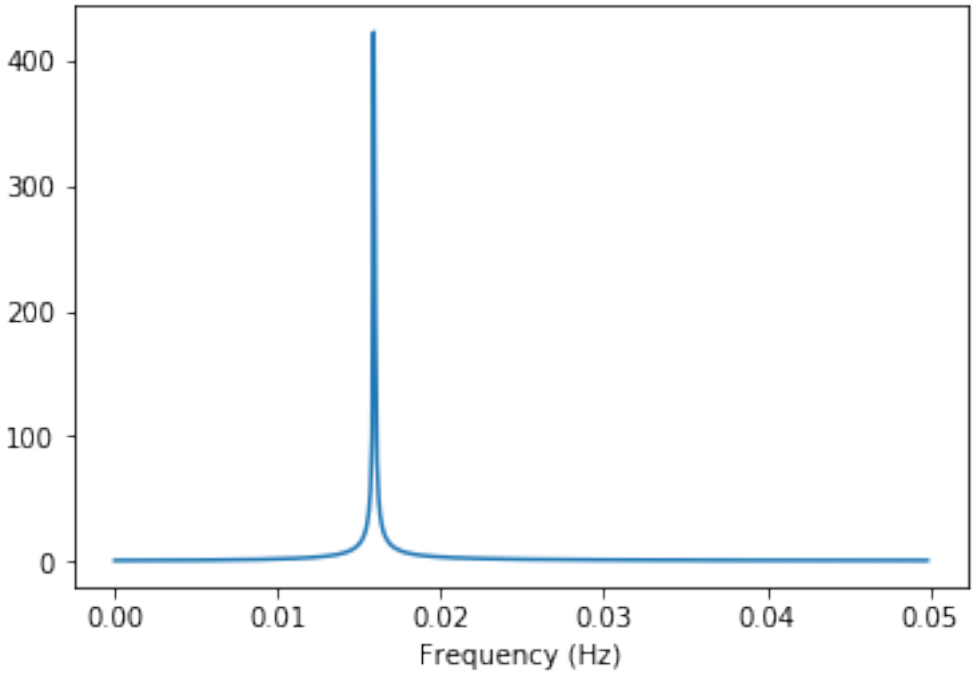
In [16]:
```
# Parameters that we might change (I like to keep these separate, to make it easier)
frequency = 0.1 #rads/sec, frequency of our sinusoid
sampling_time = 10000 #seconds, how long in time we will generate our sinusoid for.
timestep = 10 #seconds, the time in between each sample. (The inverse of this is our 

# Generate our timebase
num_samples = int(sampling_time/timestep) + 1  # Number of samples we will take
time_base = np.linspace(0, sampling_time, num_samples)
print("Time Base:")
print(time_base[0:5]) # Print out the first 5 items in the timebase just to check tha

# Generate our sinusoid, and plot it!
x = np.sin(time_base*frequency)
plt.plot(time_base, x)
plt.xlabel('Time (s)')
plt.show()
# Generate the fourier transform of our sinusoid
X = sp.fftpack.fft(x)
freq_base = sp.fftpack.fftfreq(num_samples, d=timestep) # Generates our frequency bas
print(freq_base)

# Prepare to plot our fourier transform (effort!)
# Step 1 - take the absolute value of our fourier transform
```

```
        X = np.abs(X)
        # Step 2 - take only the second half of our fourier transform (it's mirrored about f =
        X = np.copy(X[:num_samples//2])
        freq_base = np.copy(freq_base[:num_samples//2])
        # Step 3 - plot the thing!
        plt.figure()
        plt.plot(freq_base, X)
        plt.xlabel('Frequency (Hz)')

        print("We should be expecting a peak at " + str(frequency/(2*np.pi)) + " Hz")
```

```
Time Base:
[ 0. 10. 20. 30. 40.]
```




```
[ 0.00000000e+00  9.99000999e-05  1.99800200e-04 ... -2.99700300e-04
 -1.99800200e-04 -9.99000999e-05]
We should be expecting a peak at 0.015915494309189534 Hz
```



Homework Change the function! Find the fourier transform of the following functions: 1) Gaussian 2) Step Function 3) Point Function

Try to add noise to the sinusoid! See how this noise shows up on the fourier transform!

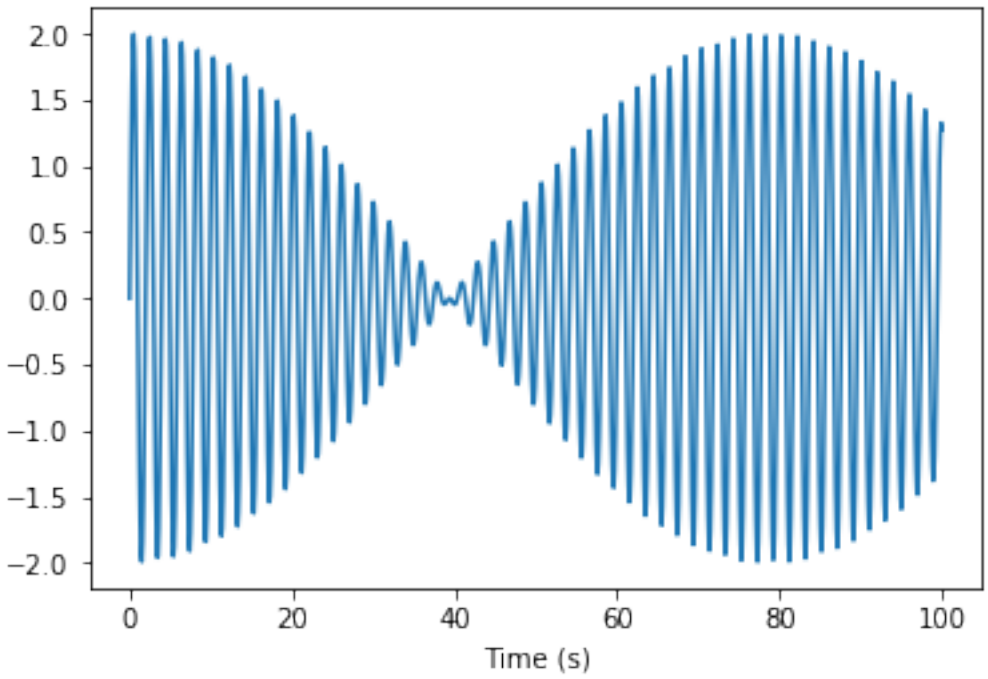
1.2 Part 2 - Russian Resolution

Playing around with the resolution of f tranforms. Our goal; understand how the number of samples affects the resolution of our fourier transform.

```
In [4]: # Let's generate a compound sinusoid with the frequencies very close together, and try

        # Parameters that we might change (I like to keep these separate, to make it easier)
        frequency_1 = 3.1514 #rads/sec
        frequency_2 = 3.2312 #rads/sec

        sampling_time = 100 #seconds, how long in time we will generate our sinusoid for.
        timestep = 0.1 #seconds, the time in between each sample. (The inverse of this is our

        # Generate our timebase
        num_samples = int(sampling_time/timestep) + 1  # Number of samples we will take
        time_base = np.linspace(0, sampling_time, num_samples)
        print("Time Base:")
        print(time_base[0:5]) # Print out the first 5 items in the timebase just to check that

        # Generate our sinusoid, and plot it!
        x = np.sin(time_base*frequency_1) + np.sin(time_base*frequency_2)
```

```
        plt.plot(time_base, x)
        plt.xlabel('Time (s)')
        plt.show()
```

```
Time Base:
[0.  0.1 0.2 0.3 0.4]
```

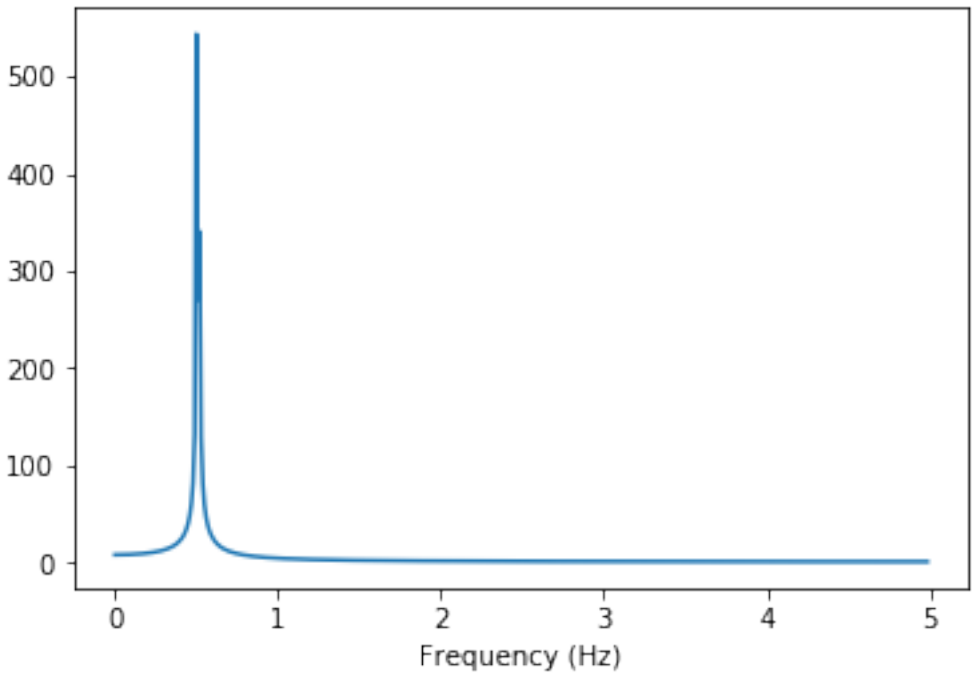


Note that we have a beat frequency on top of our two frequencies (what does THIS remind you of????)

```
In [5]: # Generate the fourier transform of our sinusoid
        X = sp.fftpack.fft(x)
        freq_base = sp.fftpack.fftfreq(num_samples, d=timestep) # Generates our frequency base
        print(freq_base)

        # Prepare to plot our fourier transform (effort!)
        # Step 1 - take the absolute value of our fourier transform
        X = np.abs(X)

        # Step 2 - take only the second half of our fourier transform (it's mirrored about f =
        X = np.copy(X[:num_samples//2])
        freq_base = np.copy(freq_base[:num_samples//2])
        # Step 3 - plot the thing!
        plt.figure()
        plt.plot(freq_base, X)
        plt.xlabel('Frequency (Hz)')
```

```
[ 0.          0.00999001  0.01998002 ... -0.02997003 -0.01998002
 -0.00999001]
```

Out[5]: Text(0.5, 0, 'Frequency (Hz)')



We now have two peaks - but we would like to improve their resolution a bit.

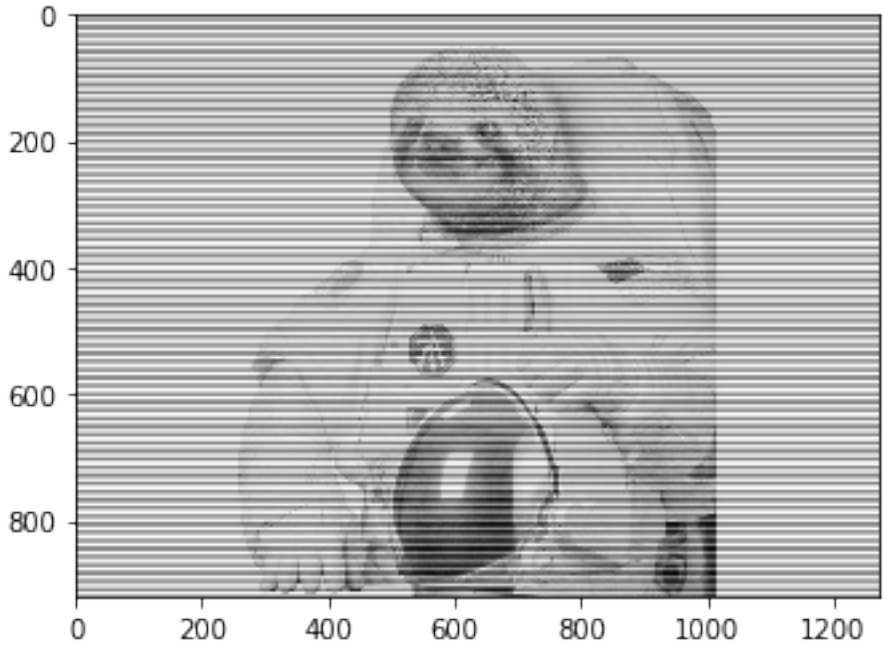
Homework - Using what you learned in the tute, make changes to improve the resolution of these two frequencies. You may wish to plot only a subset of the fourier transfrom data (do this using a slice, eg plot(X[0:100]))

1.3 Part 3 - Image Processing!

In this section, you're going to try and remove the noise from an image using the fourier transform.

In [6]:
```
import imageio # Useful image processing stuff
import scipy.fftpack
from matplotlib import pyplot as plt

# Read in the image (this last part just extracts the greyscale intensity values - we
Image = imageio.imread("SlothySinNoise.png")
plt.imshow(Image,cmap='gray')
```

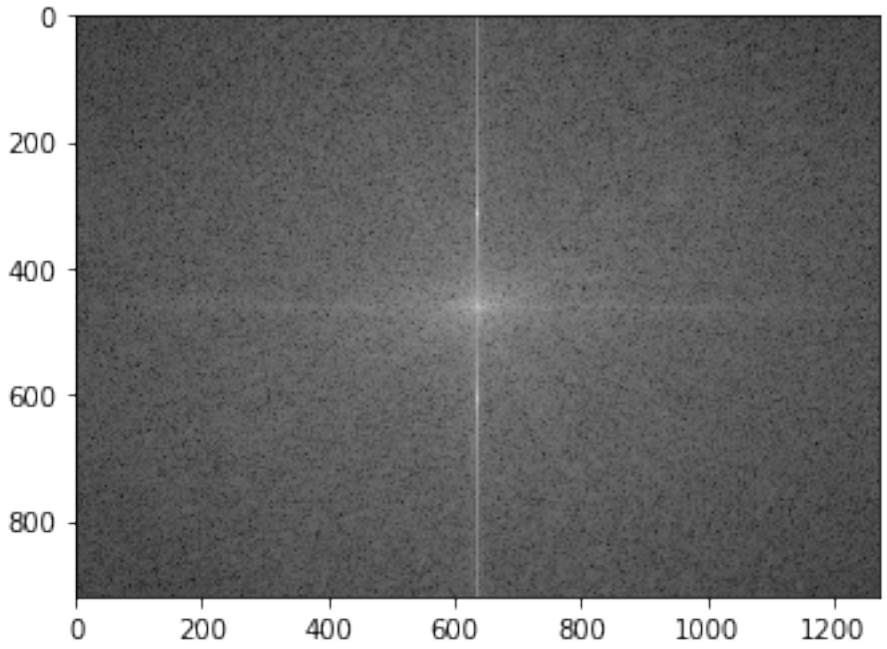
Out[6]: <matplotlib.image.AxesImage at 0x1e5a6098f28>

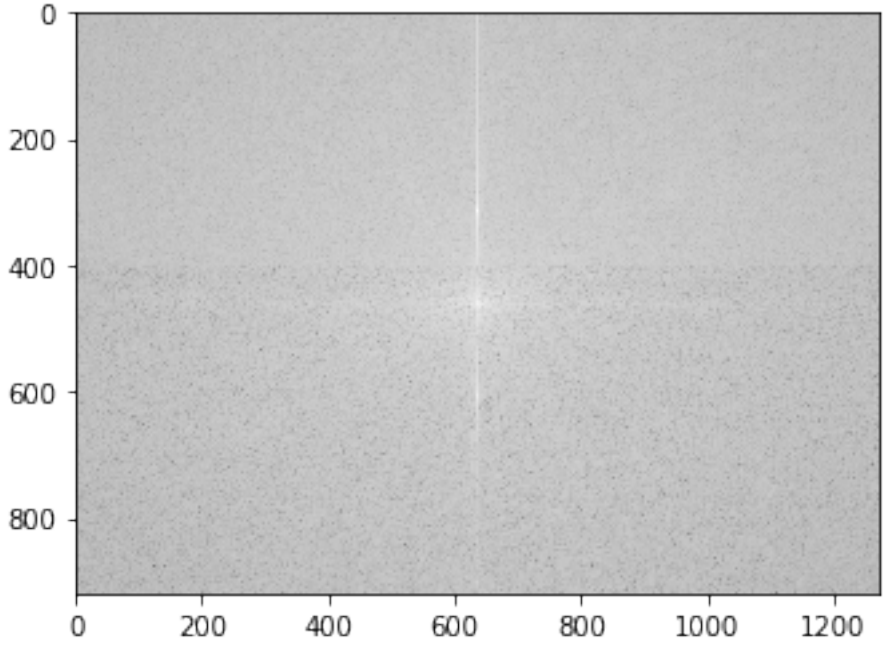


In [7]:
```
# Complete the 2D fourier transform of this image
fftRaw = scipy.fftpack.fft2(Image)
# Re-arrange the transform so that the centre is the 'low frequency' section
fftShifted = scipy.fftpack.fftshift(fftRaw)

# Take a look at the fourier transform (you have to take the log to see any useful con
# We are also taking the absolute value because we are looking at the intensity!
#      We don't want phase information (complex component of the FT)

plt.imshow(np.log(np.abs(fftShifted)), cmap='gray')
plt.show()

# INSERT PROCESSING HERE - I HAVE GIVEN YOU THE FRAMEWORK
#min_val = np.argmin(np.abs(fftShifted))
min_val = 0
for i in range(fftShifted.shape[0]):
    for j in range(fftShifted.shape[1]):
        if i > 400:
            fftShifted[i, j] = min_val + np.imag(fftShifted[i, j])


# Show our processing
plt.imshow(np.log(np.abs(fftShifted)), cmap='gray')
plt.show()

# Reverse the FFT and generate our final image
```

```
        image_out = scipy.fftpack.ifft2(scipy.fftpack.fftshift(fftShifted))
        print(image_out)
        plt.imshow(np.abs(image_out), cmap='gray')
```

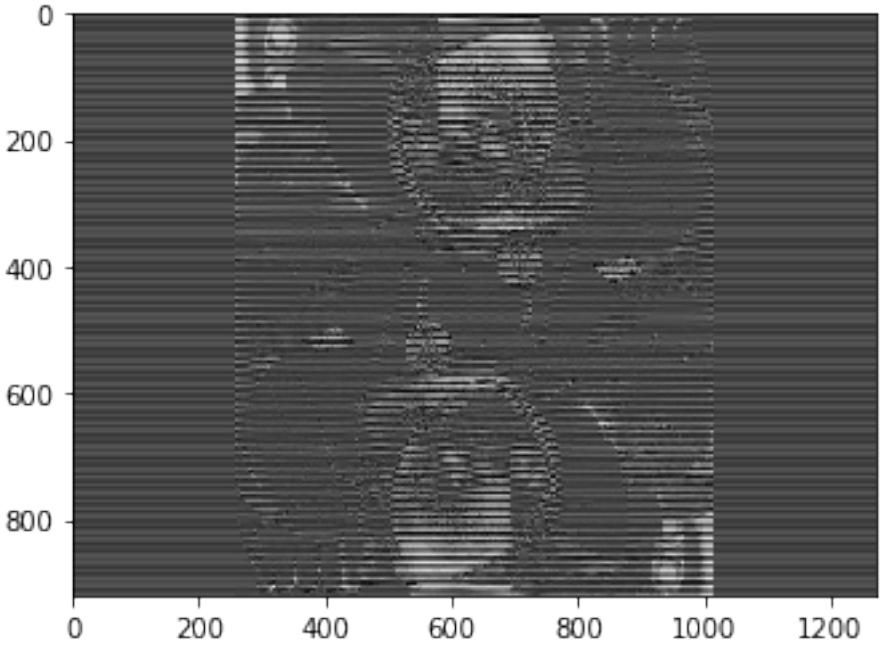
```
C:\Users\z3372528\Anaconda3\lib\site-packages\scipy\fftpack\basic.py:160: FutureWarning: Using
  z[index] = x
```





```
[[ -5.22889297 -1.5174039j   -5.22889297 -1.5174039j
   -5.22889297 -1.5174039j  ...  -5.22889297 -1.5174039j
   -5.22889297 -1.5174039j   -5.22889297 -1.5174039j ]
 [ 16.63052852+15.76273851j  16.63052852+15.76273851j
   16.63052852+15.76273851j ...  16.63052852+15.76273851j
   16.63052852+15.76273851j  16.63052852+15.76273851j]
 [ 17.68088999-25.70698119j  17.68088999-25.70698119j
   17.68088999-25.70698119j ...  17.68088999-25.70698119j
   17.68088999-25.70698119j  17.68088999-25.70698119j]
 ...
 [-13.69706915+35.04370544j -13.69706915+35.04370544j
  -13.69706915+35.04370544j ... -13.69706915+35.04370544j
  -13.69706915+35.04370544j -13.69706915+35.04370544j]
 [  8.58068223+35.9353075j    8.58068223+35.9353075j
    8.58068223+35.9353075j  ...   8.58068223+35.9353075j
    8.58068223+35.9353075j    8.58068223+35.9353075j ]
 [ 21.84115918 -8.24037859j  21.84115918 -8.24037859j
   21.84115918 -8.24037859j ...  21.84115918 -8.24037859j
   21.84115918 -8.24037859j  21.84115918 -8.24037859j]]
```

Out[7]: <matplotlib.image.AxesImage at 0x1e5a5a18da0>



1.4 Your challenge is to do the processing of the fourier tranform! You will need to exclude certain regions of the fourier transform, and then transform the image back.

To get an idea of what to do, look at the fourier transforms of the three files - one has no noise, one has a bit of noise, and one has a lot of noise - what's the difference?