Homework 1

Daniel Finn

University at Buffalo
PHY 506: Computational Physics II

1 Introduction

Homework 7 from PHY 505: Computational Physics I was reproduced on the NVIDIA Jetson Nano 2GB.

2 Getting the NVIDIA Jetson Working

A few issues were encountered trying to get Jupyter installed and working. The NVIDIA Jetson Nano 2GB image, provided by NVIDIA, comes with python and some other related packages pre-installed, however, all of the pre-installed packages appear to be several generations old. For examples, the Python 3.6 was pre-installed despite 3.9 being the current working version. Therefore, to fix issues with the Jupyter install, Python and python-related packages were updated.

Instead of Python 3.9, Python 3.8 was installed and linked. Based on recent personal experience, not all related packages have been updated for 3.9 so 3.8 is a safer option.

Problem1

February 17, 2021

1 Problem 1

[6]:
```
import matplotlib.pyplot as plt
import sys
from fft import fft as fft
import numpy as np
import math
```

[7]:
```
# Make the plots a bit bigger to see
# NOTE: Must be done in a separate cell
plt.rcParams['figure.dpi'] = 150
```

1.1 CO_2 Data

1.1.1 Problem 1

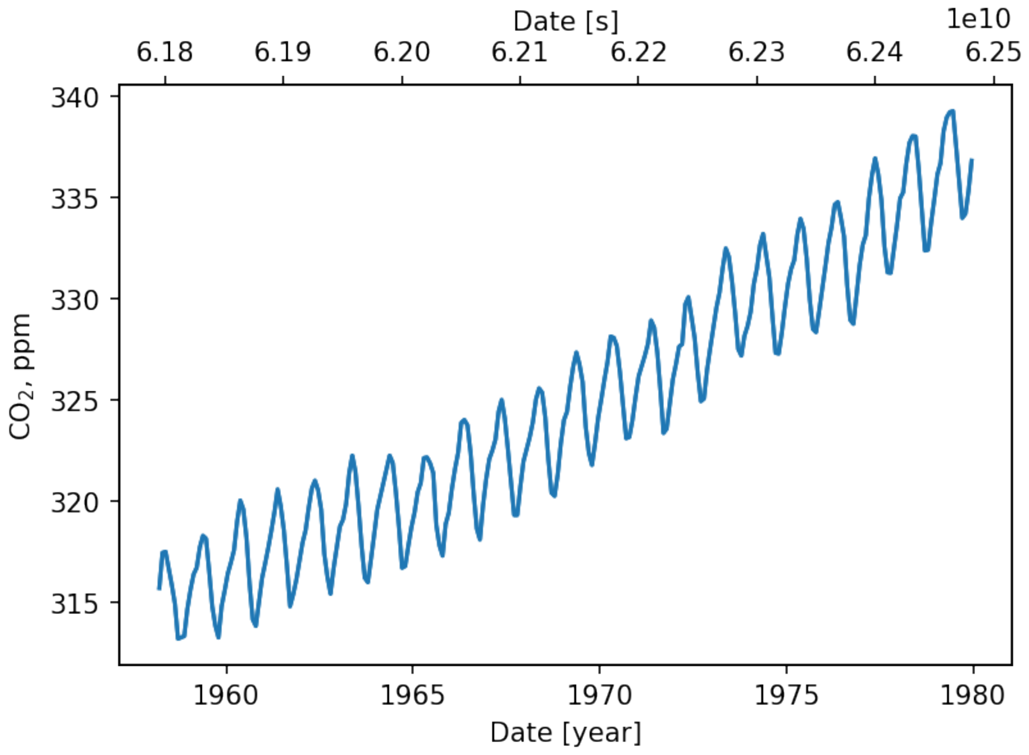
The CO_2 data used in homework 6 was first uploaded. The x data, originally in decimal years, was converted to seconds so frequencies can be in units of Hertz. A function, *findPeriod()*, was written to find the peaks of the CO_2 data and from that the period of oscillations. A second function, *findPeaks()*, was written to calculate the frequency of the transformed data. *findPeaks()* will be used in Part 3.

[8]:
```
import p1_functions as p1

x,y = p1.read_co2_2('co2_mm_mlo.txt')

y_valid = y >= 0.
y = y[y_valid]
x = x[y_valid]

y =y[0:256]
x =x[0:256]

#Convert to seconds
x_s = x*31556952
```

```
T = p1.findPeriod(x_s, y)
P = 1/T
print("Period = "+str(T)+" [s]")
print("Frequency = "+str(P)+" [Hz]")

f1 = plt.figure(1)
ax1 = f1.add_subplot(111)
ax2 = ax1.twiny()
ax1.plot( x, y )
ax2.plot( x_s, y)
ax1.set_ylabel('CO${_2}$, ppm')
ax1.set_xlabel('Date [year]')
ax2.set_ylabel('CO${_2}$, ppm')
ax2.set_xlabel('Date [s]')
```

```
Period = 2868813.8181818184 [s]
Frequency = 3.485761235749257e-07 [Hz]
```

```
[8]: Text(0.5, 0, 'Date [s]')
```



1.1.2 Part 2 - Index to Frequency Domain

A function was written in p1_functions.py to convert from the index domain to the frequency domain. The equation,

$$f = \frac{k}{N},$$

was used for this conversion, where k is the x data in the index domain and N is the total number of samples.

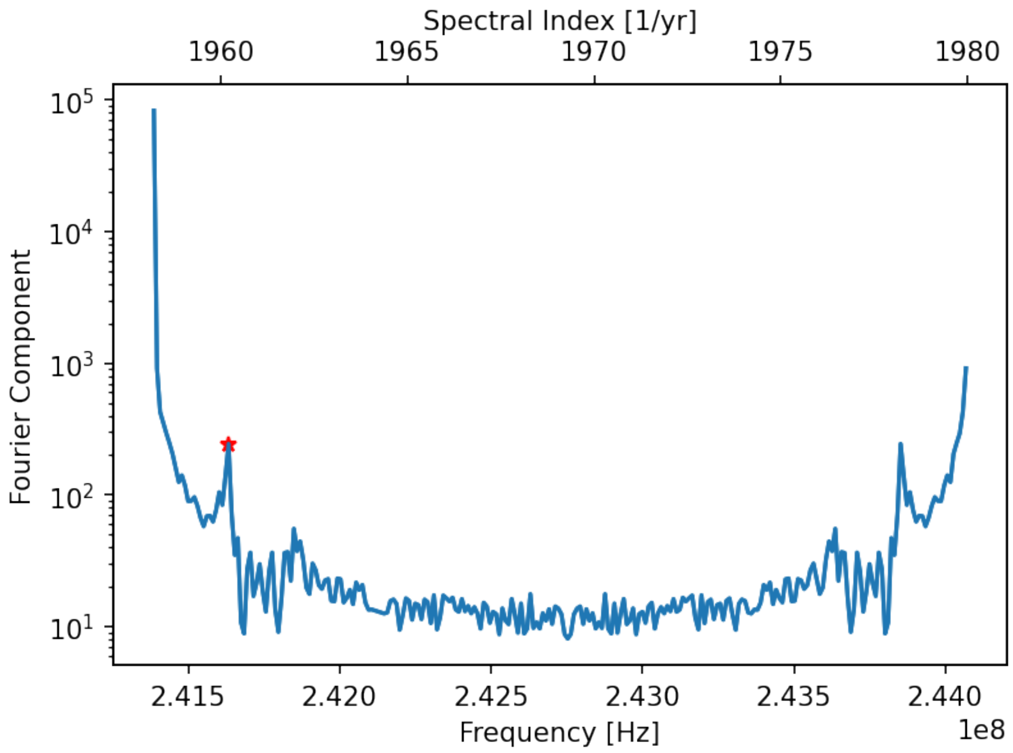
[9]:
```
freq_domain = p1.indexToFreq(x_s, len(y))

Y = fft(y)
Y_abs = abs(Y)
```

1.1.3 Part 3 - Frequency of Data

The functions *indexToFreq()* and *findPeaks()* were used to find the frequency values for this data set.

[13]:
```
N = len(y)
xpeaks, Ypeaks = p1.findPeaks(freq_domain, Y_abs[:int(N/2)], 45)
print(xpeaks)
print("Peak Frequency Values: ("+str(xpeaks[0])+" [Hz])")
print("Peak Fourier Components: ("+str(Ypeaks[0])+")")

f2 = plt.figure(2)
ax3 = f2.add_subplot(111)
ax4 = ax3.twiny()
ax3.plot( freq_domain, Y_abs)
ax4.plot( x, Y_abs)
ax3.plot( xpeaks, Ypeaks, 'r*')
plt.yscale('log')
ax3.set_ylabel('Fourier Component')
ax3.set_xlabel('Frequency [Hz]')
ax4.set_xlabel('Spectral Index [1/yr]')


plt.show()
```

```
[241633553.7735]
Peak Frequency Values: (241633553.7735 [Hz])
Peak Fourier Components: (244.46603240281848)
```


Spectral Index [1/yr]
1960
1965
1970
1975
1980
Fourier Component
10^5
10^4
10^3
10^2
10^1
2.415
2.420
2.425
2.430
2.435
2.440
1e8
Frequency [Hz]

```
[ ]:
```

Problem2

February 17, 2021

1 Problem 2: Filtering

[51]:
```
import matplotlib.pyplot as plt

import sys

from fft import fft, fft_power, ifft
import numpy as np
import math
```

[52]:
```
# Make the plots a bit bigger to see
# NOTE: Must be done in a separate cell
plt.rcParams['figure.dpi'] = 150
```

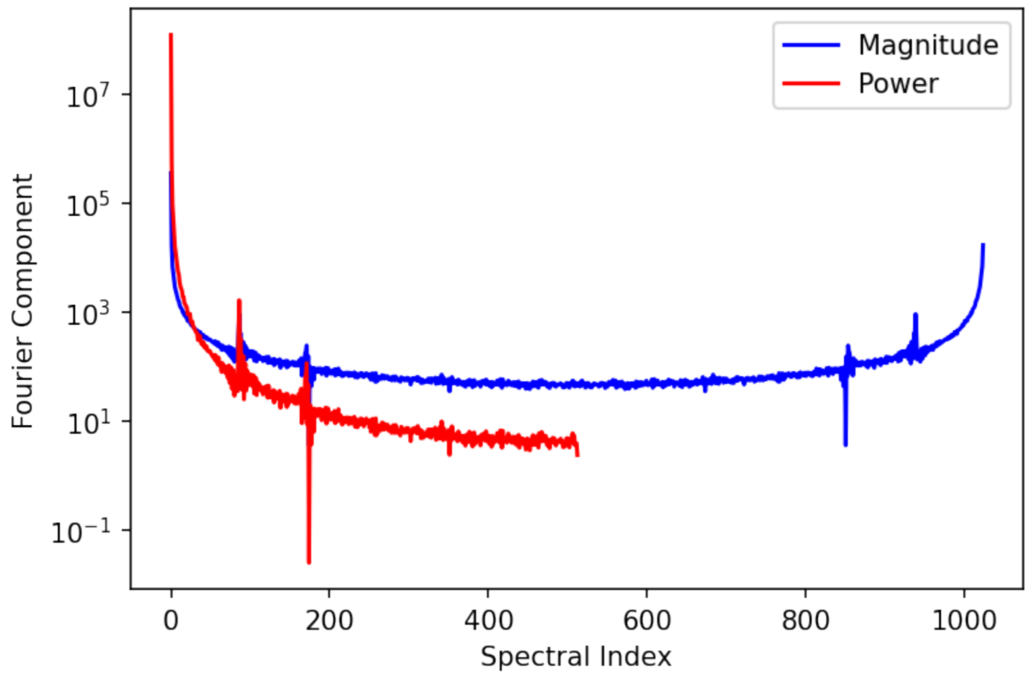
1.0.1 Part 1 - Smoothed Power Spectrum

The power spectrum was smoothed using a padding techniques. The first data value *y[0]* was appended to the end of the CO_2 data to make the total length of the array 1024.

[53]:
```
import math
from p1_functions import read_co2_2

# Read CO2 data
x,y = read_co2_2('co2_mm_mlo.txt')
ind = y > 0
x = x[ind]
y = y[ind]

print("Initial length of data: "+str(len(y)))

# Pad with values
N = 1024
Npad = N - len(y)
padding = np.ones(Npad)*y[0]
y = np.concatenate( (y, padding) )
x = np.arange(len(y))
```

```
Y = fft(y)

Y_abs = abs(Y)
powery = fft_power(Y)
powerx = np.arange(powery.size)

f1 = plt.figure(1)
plt.plot( x, Y_abs ,'b-', label='Magnitude')
plt.plot(powerx, powery, 'r-', label='Power')
plt.yscale('log')
plt.ylabel('Fourier Component')
plt.xlabel('Spectral Index')
plt.legend()

plt.show()
```

```
Initial length of data: 713
```

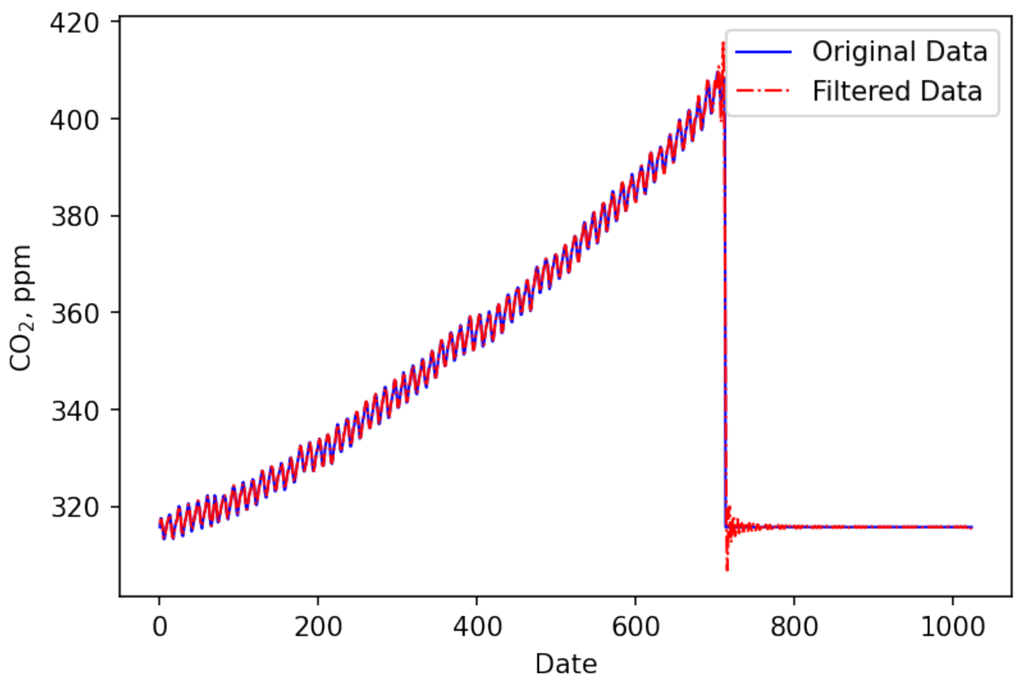


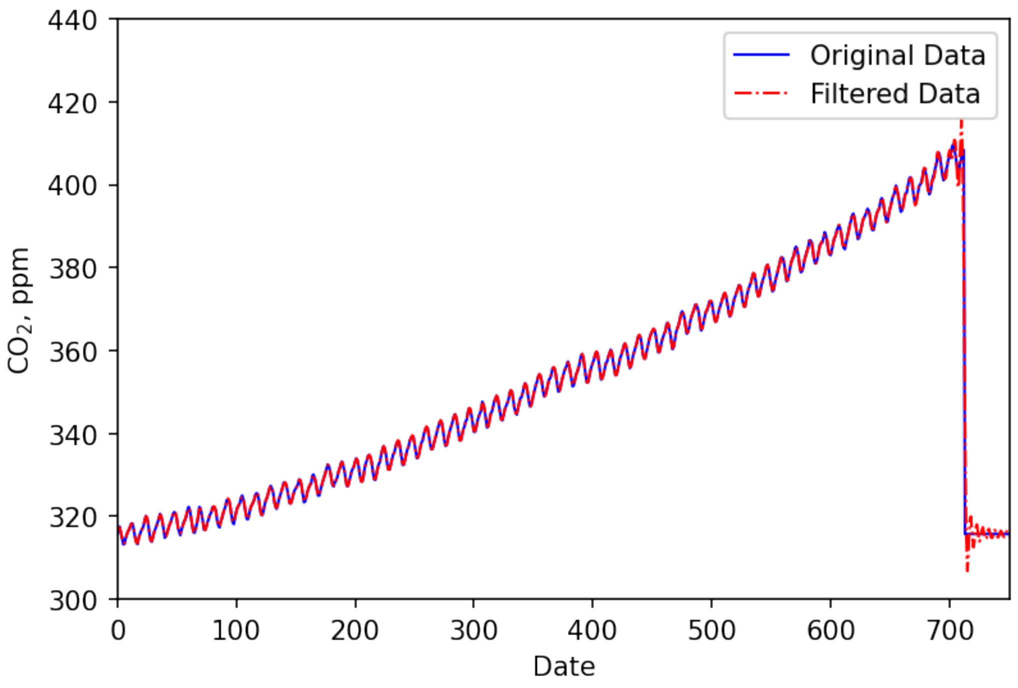
1.0.2 Part 2 - Filtered Data

The padded data was used to clean up the CO_2 data. Frequencies above 200 were then filtered out of the data set.

```
[54]: maxfreq = 200
      Y_filtered = Y
      Y_filtered[maxfreq:len(Y)-maxfreq] = 0.0

      yfiltered = ifft(Y_filtered)

      f2 = plt.figure(2)
      plt.plot( x, y , 'b-', label='Original Data', linewidth=1)
      plt.plot( x, np.real(yfiltered), 'r-.',label='Filtered Data', linewidth=1)
      plt.ylabel('CO${_2}$, ppm')
      plt.xlabel('Date')
      plt.legend()

      f3 = plt.figure(3)
      plt.plot( x, y , 'b-', label='Original Data', linewidth=1)
      plt.plot( x, np.real(yfiltered), 'r-.',label='Filtered Data', linewidth=1)
      plt.ylabel('CO${_2}$, ppm')
      plt.xlabel('Date')
      plt.xlim([0,751])
      plt.ylim([300,440])
      plt.legend()
```

```
[54]: <matplotlib.legend.Legend at 0x7f5c3b43d0>
```




Original Data
Filtered Data
CO_2, ppm
Date
440
420
400
380
360
340
320
300
0
100
200
300
400
500
600
700

Problem3

February 17, 2021

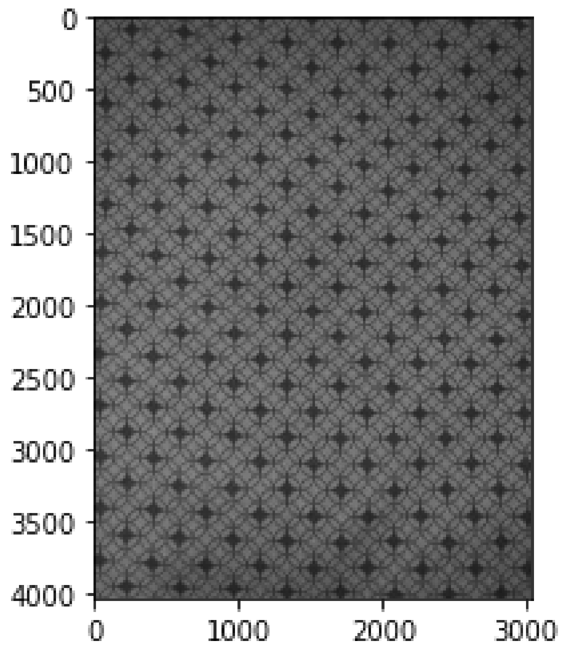
0.1 Problem 3 - 2D Fast Fourier Transform

An greyscale image was taken using my cellphone of a shirt with a periodic pattern. Filtering was applied to the image using the Fast Fourier Transform in an attempt to cancel out the pattern.

[3]:
```
from PIL import Image, ImageOps
import numpy as np
import matplotlib.pyplot as plt
```

[4]:
```
bilayer = np.array(ImageOps.grayscale(Image.open('IMG_2358.png')))
plt.imshow(bilayer, "gray")
```

[4]:
```
<matplotlib.image.AxesImage at 0x7f5594f940>
```

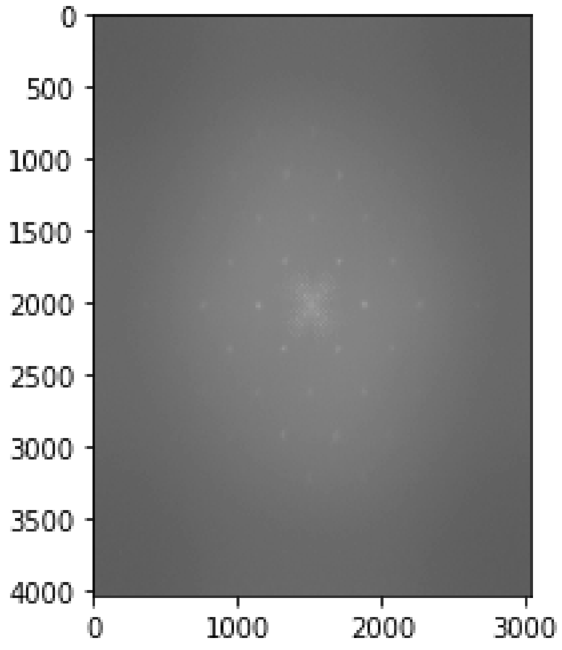


First calculate the FFT of the image above.

[5]:
```
Nx = len(bilayer[0])
Ny = len(bilayer)
x = np.linspace(-10,10,Nx)
y = np.linspace(-10,10,Ny)
```

[6]:
```
Fbilayer = np.fft.fft2(bilayer)
Fbilayercenter = np.fft.fftshift(Fbilayer)
plt.imshow(np.log(1+np.abs(Fbilayercenter)), "gray")
```

[6]:
```
<matplotlib.image.AxesImage at 0x7f558ff160>
```



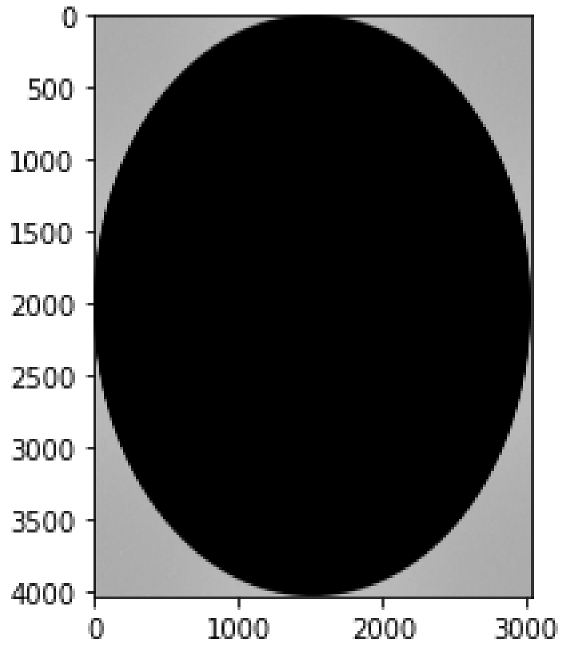
0.1.1 Filter 1 - High Pass Filter

Looking at the image's fourier data, peaks in the data can be seen in a diamond shape around the center with one larger peak at the center of the image. The area surrounding diamond patterned peaks also appears to be slightly lighter than the outer edges of the image. This larger, dimmer peak could be the result of lower frequency periodicities in the pattern. The goal of the high pass filter should therefore be to filter out all of these peaks.

A simple high pass filter in the shape of an ellipse was applied to the image's matrix in the spectral domain. All frequency data inside the ellipse $R = x^2 + y^2$, where x and y are arrays of same size as the image and R = 8, were zeroed. The ellipse filters out a considerable amount of the fourier data, however this is necessary because the larger, dimmer peak extends to near the edges of the fourier image. The inverse FFT was then applied to the image fourier data multiplied by the highpass filter.

[7]:
```
R = 10
xx, yy = np.meshgrid(x, y, sparse=True)
highpass = (np.sqrt(xx**2 + yy**2) >= R)*1
plt.imshow(np.log(1+np.abs(highpass*Fbilayercenter)), "gray")
```

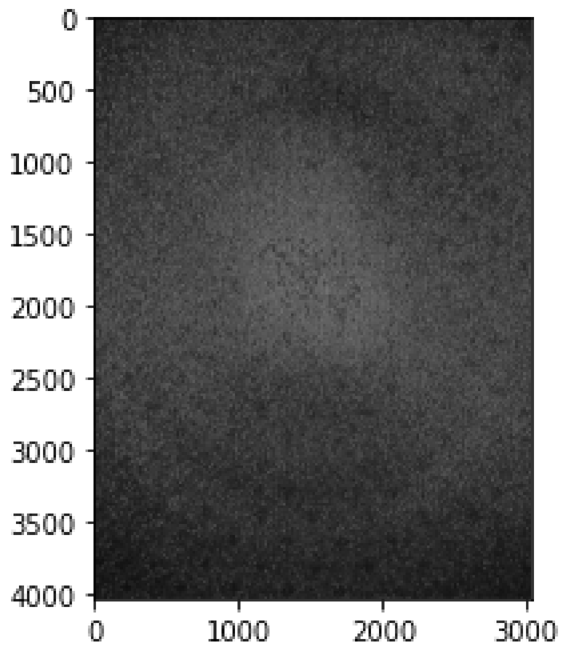
[7]:
```
<matplotlib.image.AxesImage at 0x7f51d9e430>
```



[8]:
```
filtered_hp = np.fft.ifft2(highpass*Fbilayercenter)
```

[9]:
```
plt.imshow(np.log(1+np.abs(filtered_hp)), "gray")
```

[9]:
```
<matplotlib.image.AxesImage at 0x7f51d5d1f0>
```



0.1.2 High Pass Filter Result

The high pass filter was mostly successful in filtering out the periodic pattern.

```
[ ]:
```