A Quick Demonstration of MCSA Using Raw Data in Python Part 2

Done in a Jupyter Notebook

Howard W Penrose, Ph.D., CMRP - President MotorDoc LLC, info@motordoc.com

In [1]:
```
#Here is where we import the tools that will be used for our rough pass.
import numpy as np
import csv
import matplotlib.pyplot as plt
import scipy as sp
from scipy.fft import fft, fftfreq
from scipy.fft import rfft, rfftfreq
from scipy.fftpack import fft,fftshift
from scipy import signal
from scipy.signal import welch
from scipy.signal import flattop
import pandas as pd
```

In [2]:
```
#This is the data that was used from a csv file.  There are 100,000 rows of data taken at 10,000 samples per second
#for 10 seconds.
df = pd.read_csv(r"Cbe.c
df.head()
```

Out[2]:

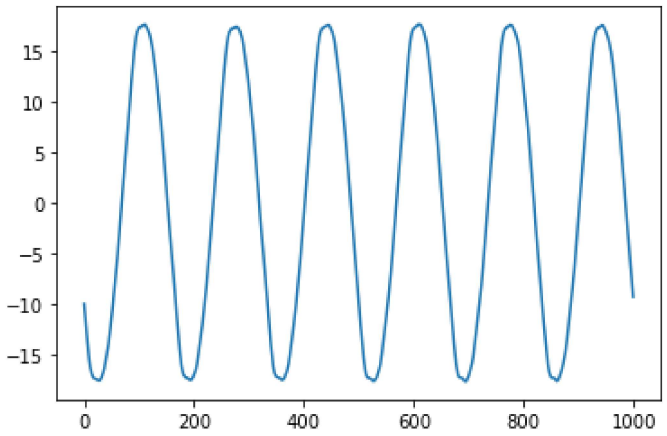
	Va	Vb	Vc	Aa	Ab	Ac
0	-394.499725	254.709045	195.858688	-10.017016	17.856047	-8.399207
1	-402.846680	235.570038	217.189835	-10.709504	17.899864	-7.772016
2	-403.015320	222.332916	228.319138	-11.359893	17.884399	-7.156853
3	-403.268250	211.540878	238.942551	-12.010282	17.917047	-6.539113
4	-410.603455	193.666565	245.856201	-12.616853	17.923922	-5.917936

Following is what we performed in Part 1

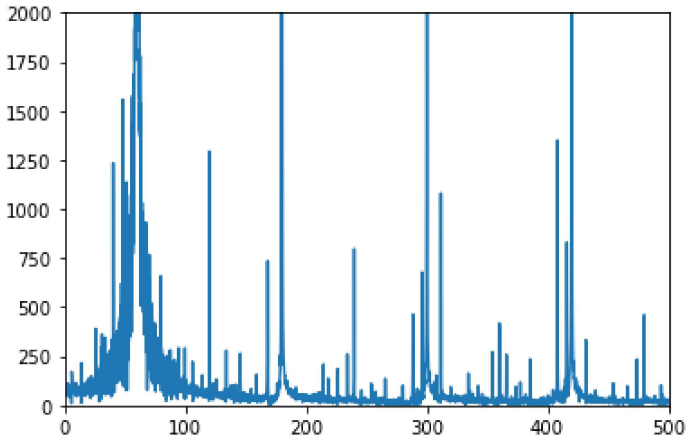
In [3]:
```
#So, while we prep for performing an FFT analysis, let's see what it looks like.  Wow - a perfect sine wave.
#There cannot be information there, can there?
```

```
Sec = 10 #seconds of data collected
normalized_data = np.double(df['Aa'])
plt.plot(normalized_data[:1000])
plt.show
```

Out[3]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



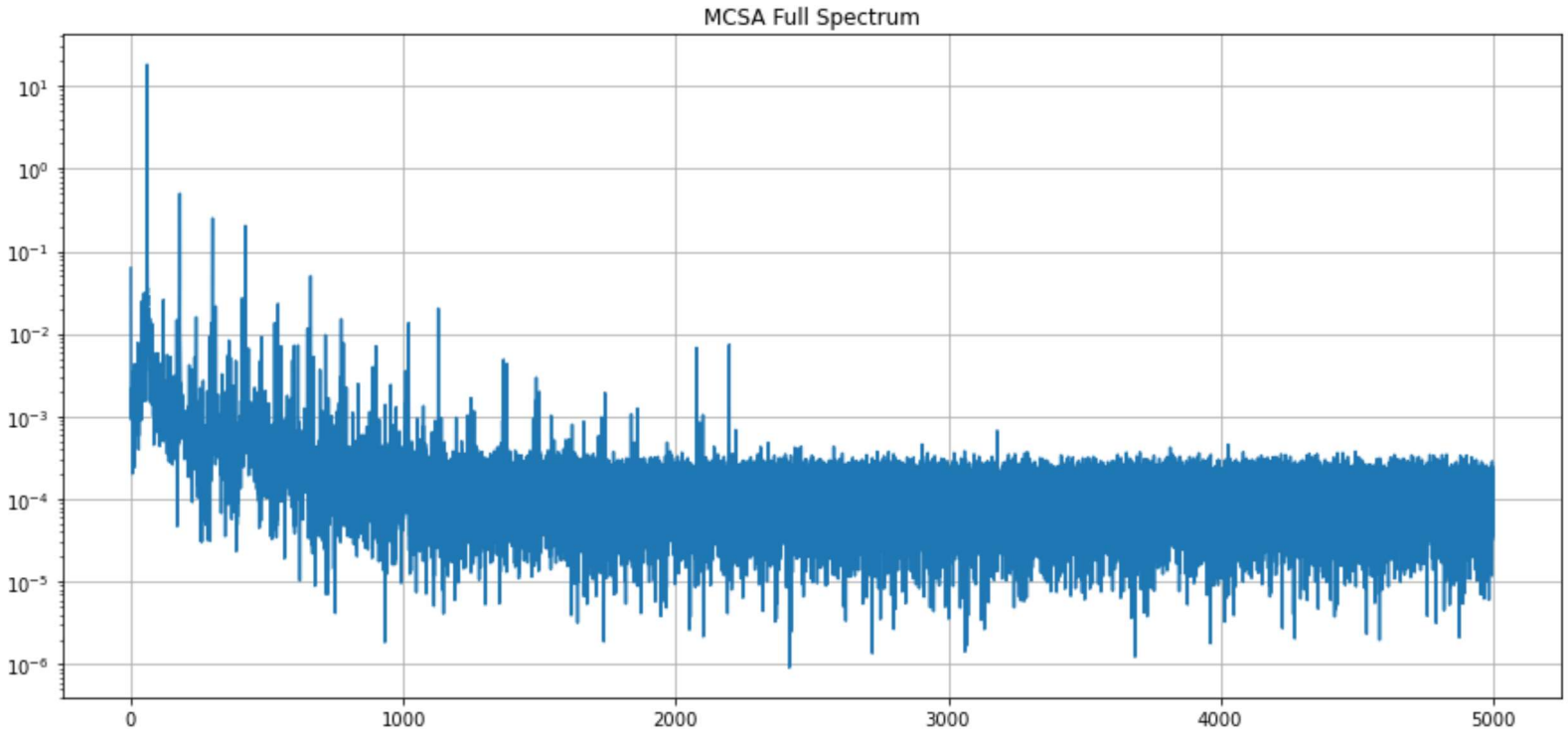
In [4]:
```
#What happens if we zoom in and ignore the rediculously high line frequency peak - or peaks.
Sample_Rate = 10000
N = Sample_Rate*10
yf = rfft(normalized_data)
xf = rfftfreq(N,1/Sample_Rate)

plt.plot(xf,np.abs(yf))
plt.xlim([0,500])
plt.ylim([0,2000])
plt.show()
```

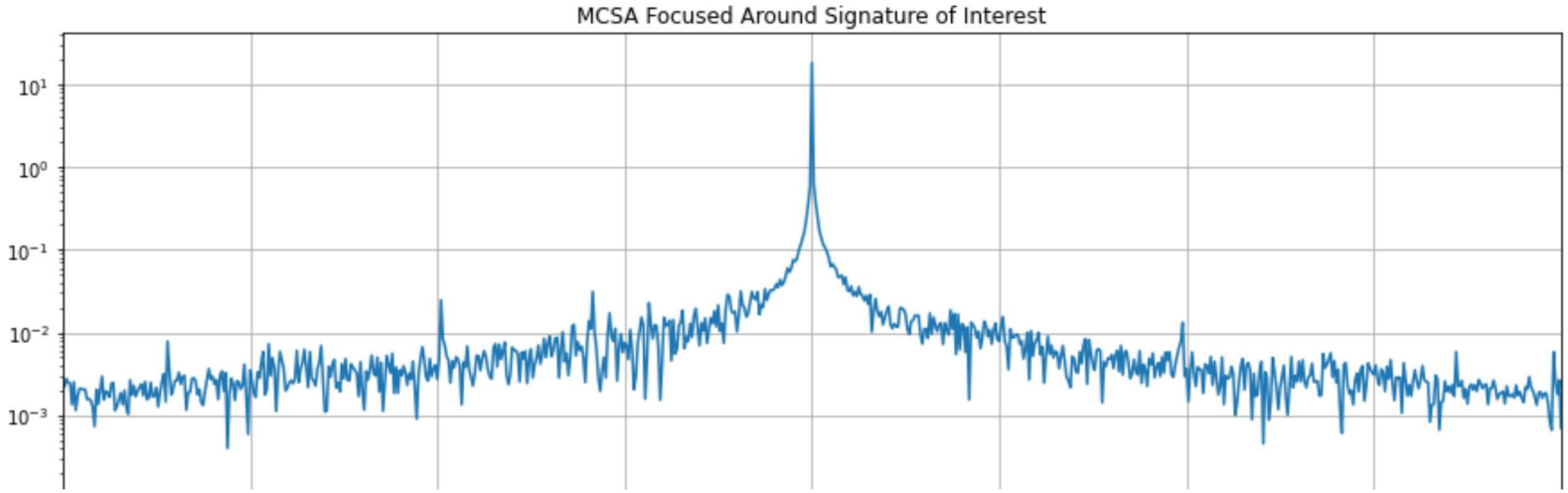


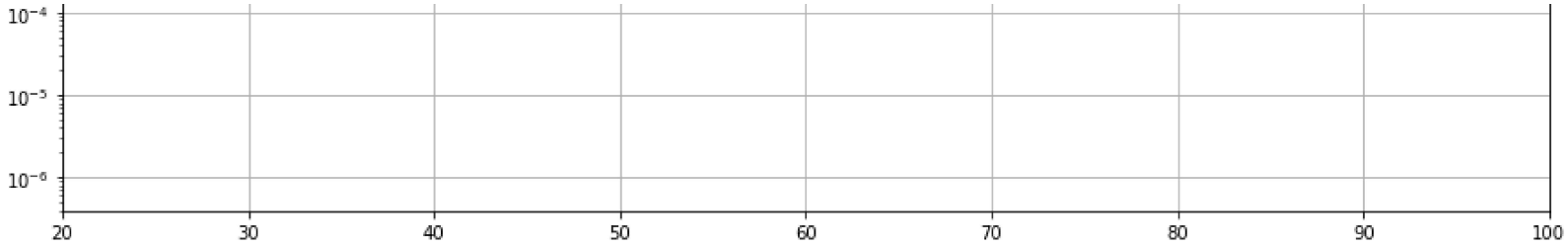
In [5]:

```
#OK, now lets look at the real logarithmic graph in dB.  These are not the values we expect, but it gets us in the right
#direction.  What's wrong with this graph?  The dB values are a factor of 10 instead of 20, which is used for current,
#leaving us well under what we should see.

#The motor is real.  It is a 6-pole, 1200 RPM (20Hz) motor with a bit of an issue.  It is an 1175 RPM, 10HP, 460Vac, 14.3Amp
#machine that is misaligned.  As you can see below, there is a peak just above 40Hz and below 80 Hz, which are +/- running
#speed around the line frequency of 60Hz.

#Now we are off to a good start.  What else can we do with this data?  Visit next week as we start working on resolution,
#windows, and a bunch of other things to get this signature to look like what we see on an analyzer.
T = 1/10000 #sampling frequency (sec)
x = np.linspace(0.0,N*T,N)
y = normalized_data

yf = abs(fft(y)) #perform fft returning magnitude
xf = np.linspace(0.0,1.0/(2.0*T), N//2)#determine frequency bins

freqs = fftfreq(N,T)

#ax1 = plt.subplot(211)
#ax1.plot(x,y)
ax2 = plt.figure(figsize=(15,15))
#plt.grid()

ax2 = plt.subplot(212)
yf2 = 2/N*np.abs(yf[0:N//2]);
ax2.semilogy(xf,yf2)
plt.grid()
ax2.set_title("MCSA Focused Around Signature of Interest")
ax2.set_xlim([20,100])

ax3 = plt.subplot(211)
yf2 = 2/N*np.abs(yf[0:N//2]);
```

```
ax3.semilogy(xf,yf2)
plt.grid()
ax3.set_title("MCSA Full Spectrum")
#ax3.set_xlim([20,100])
plt.show()
```







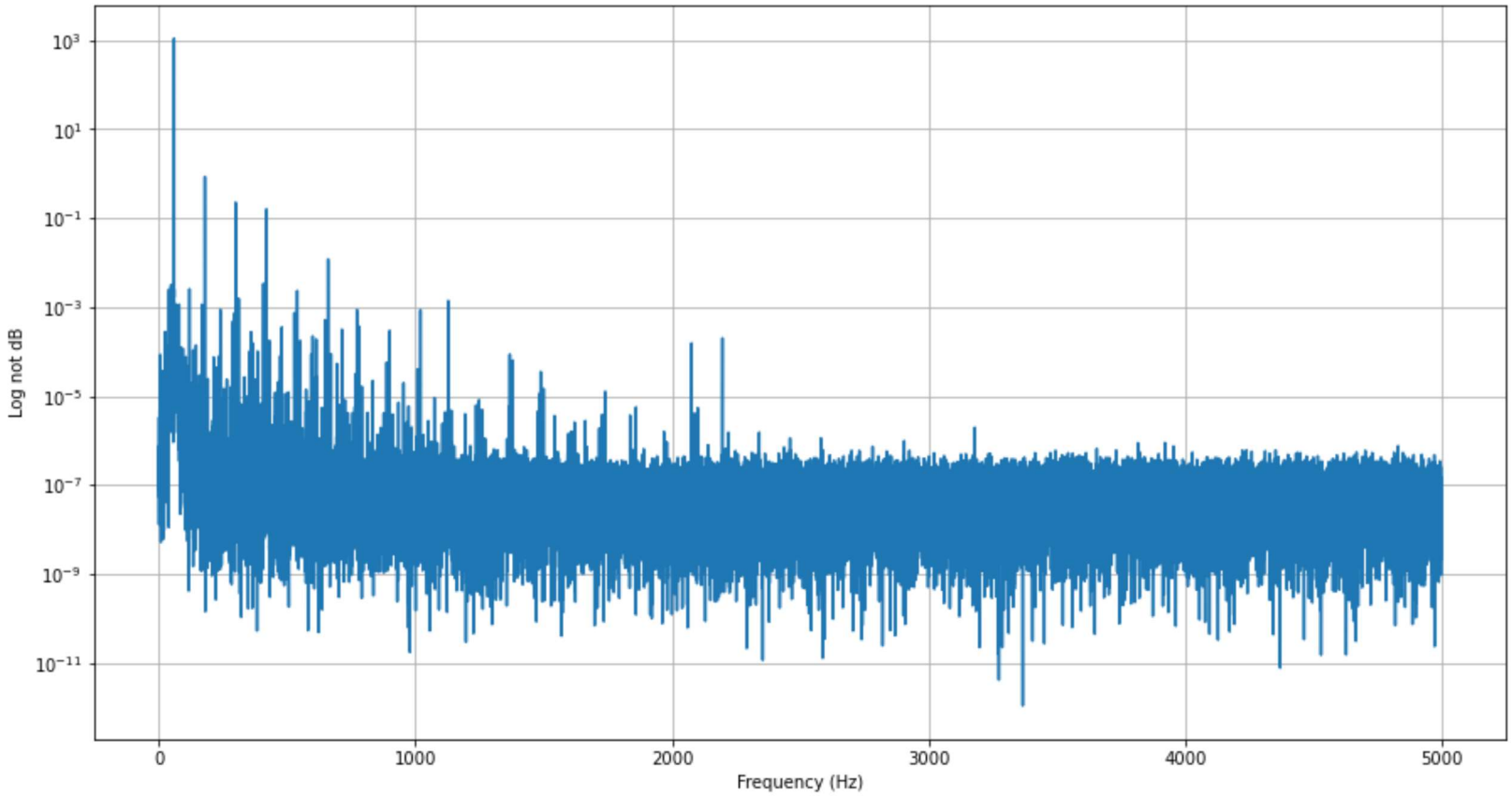
What we will do now is review a few filters.

The first one of interest would be the Hanning Window followed by the Flat Top Window and a Hamming Window. There are your more common filters used for current signature analysis in most rotating machinery systems.

The purpose of the filters is to smooth the data. For vibration analysis one of the most common is the Hanning Window. However, there will tend to be less sharp peaks and it requires enough data collection time to be effective. Following is the above data in a Hanning Window.

In [6]:
```
freqs,psd = welch(normalized_data, fs=10000,window='hanning',nperseg=100000)
plt.figure(figsize=(15,8))
plt.xlabel("Frequency (Hz)")
plt.ylabel("Log not dB")
plt.grid()
plt.semilogy(freqs, psd)
```

Out[6]:
```
[<matplotlib.lines.Line2D at 0x22b1c327310>]
```

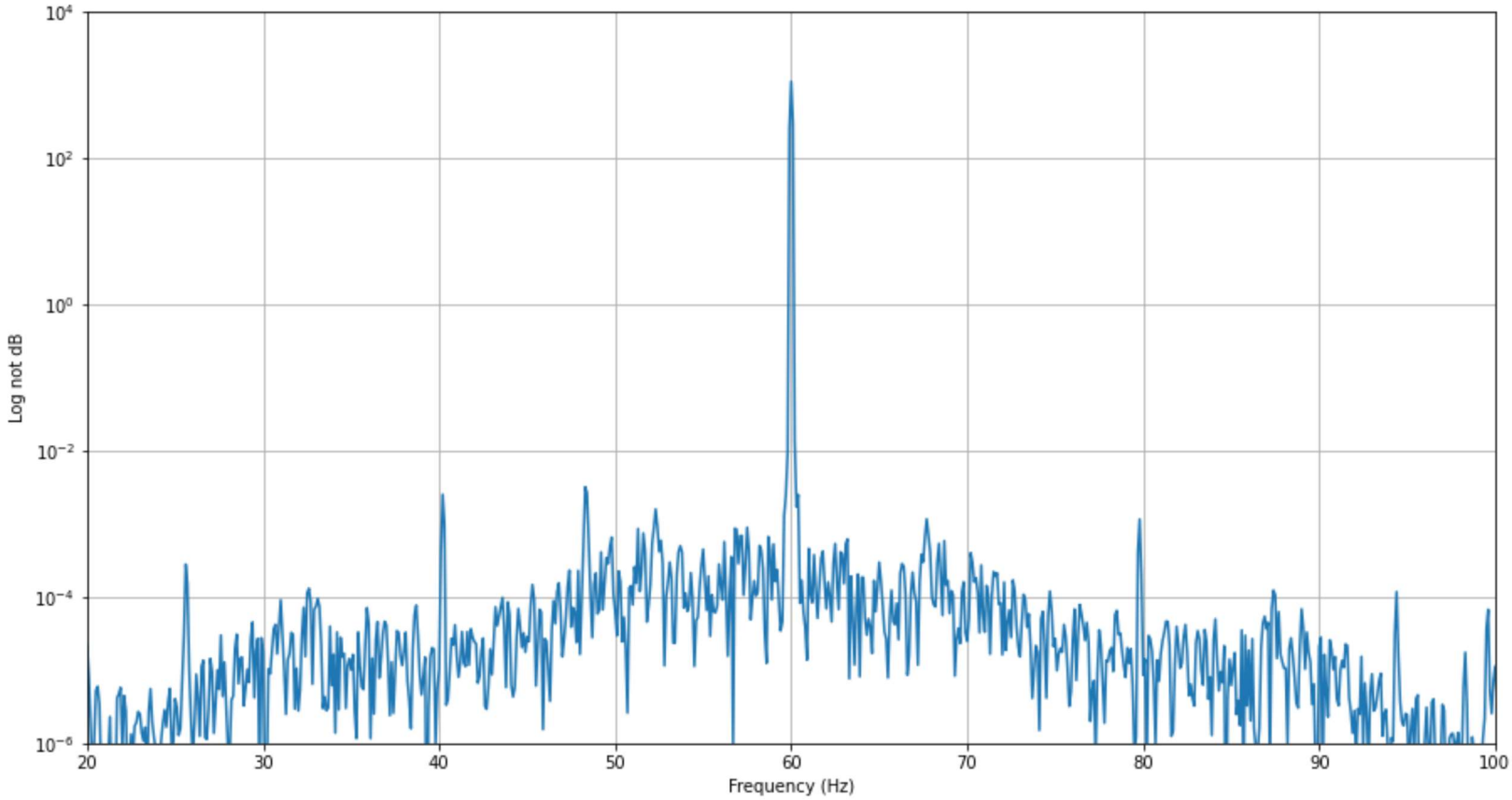


In [7]:

```
plt.figure(figsize=(15,8))
plt.xlim([20,100])
plt.ylim([10e-7,10e3])
plt.xlabel("Frequency (Hz)")
plt.ylabel("Log not dB")
plt.grid()
plt.semilogy(freqs,psd)
```

Out[7]:

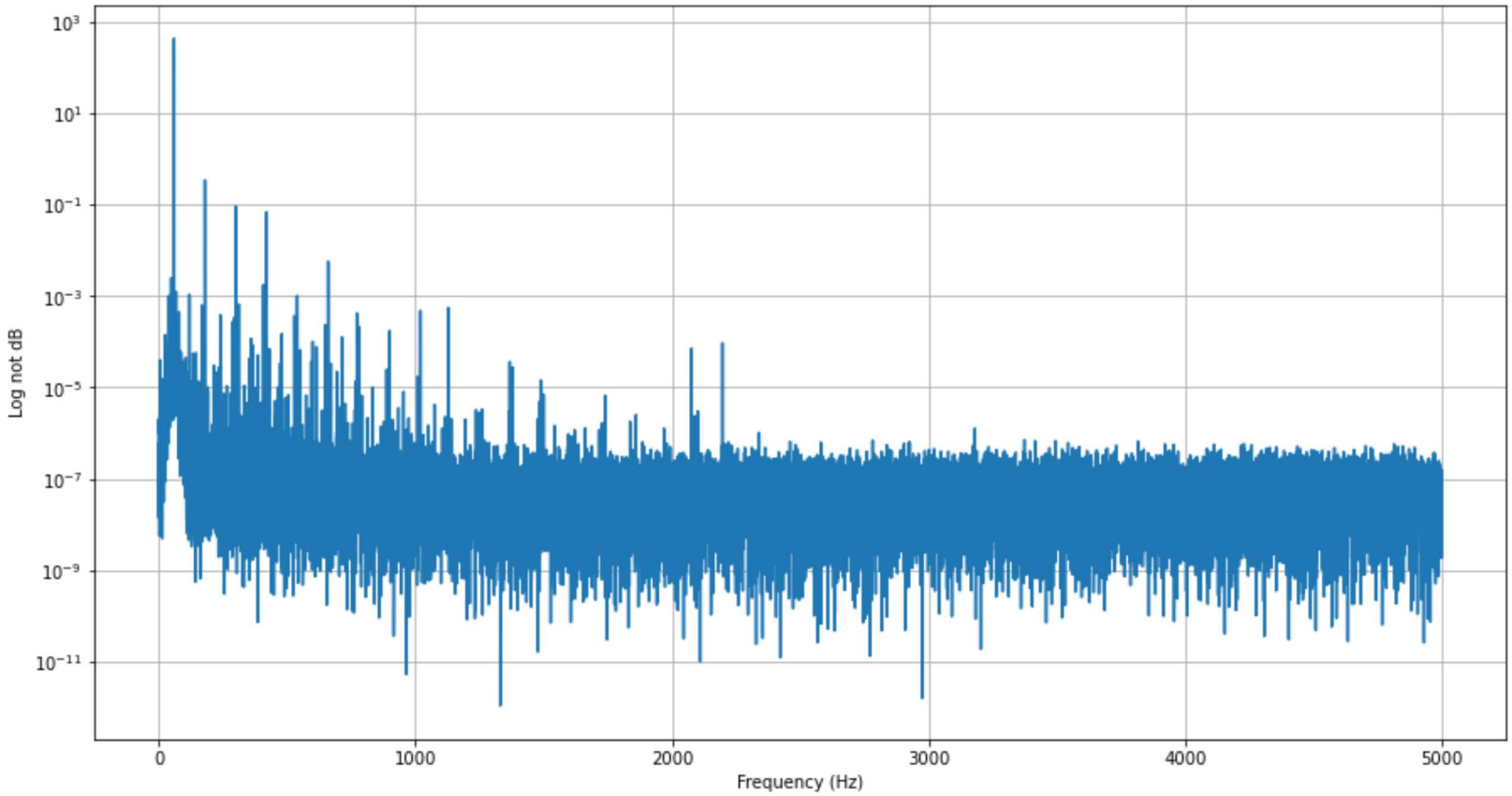
```
[<matplotlib.lines.Line2D at 0x22b1cab00a0>]
```



Now lets see what happens with a Flat Top Window

In [8]:
```
freqs,psd = welch(normalized_data, fs=10000,window='flattop',nperseg=100000)
plt.figure(figsize=(15,8))
plt.xlabel("Frequency (Hz)")
plt.ylabel("Log not dB")
plt.grid()
plt.semilogy(freqs, psd)
```

Out[8]:
```
[<matplotlib.lines.Line2D at 0x22b20b612b0>]
```

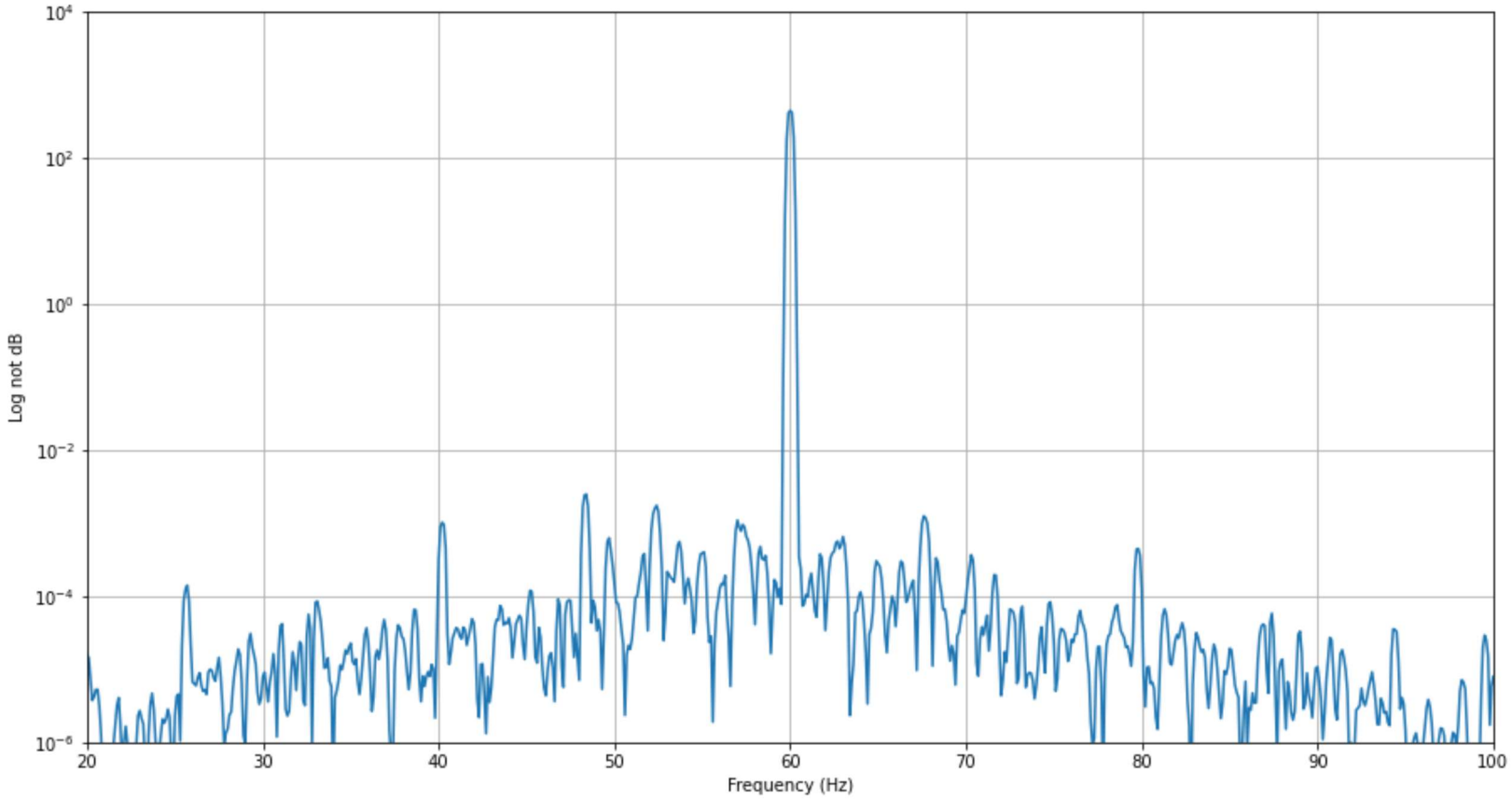


In [9]:

```
plt.figure(figsize=(15,8))
plt.xlim([20,100])
plt.ylim([10e-7,10e3])
plt.xlabel("Frequency (Hz)")
plt.ylabel("Log not dB")
plt.grid()
plt.semilogy(freqs,psd)
```

Out[9]:

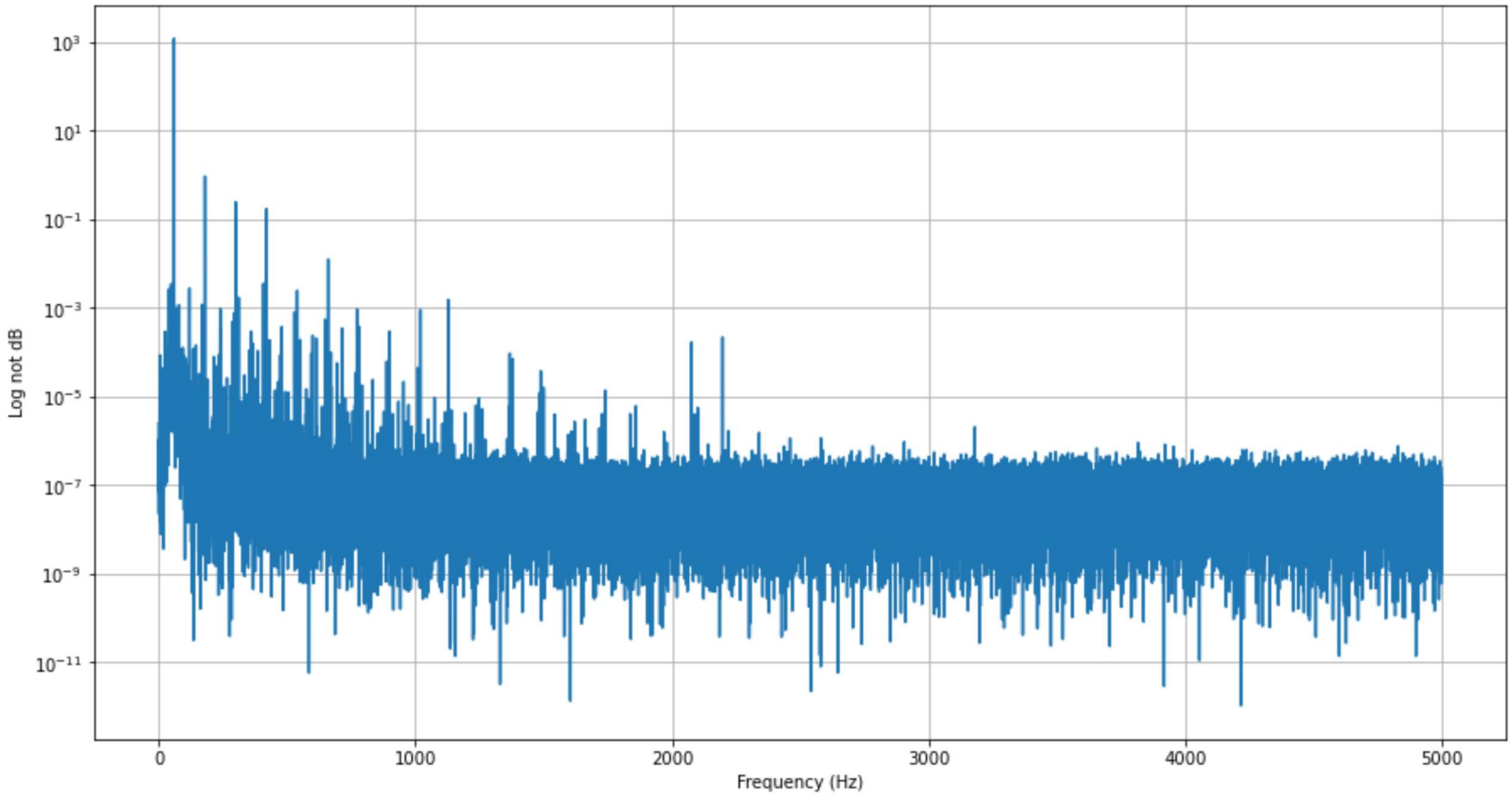
```
[<matplotlib.lines.Line2D at 0x22b22039280>]
```



Or a Hamming Window

In [10]:
```
freqs,psd = welch(normalized_data, fs=10000,window='hamming',nperseg=100000)
plt.figure(figsize=(15,8))
plt.xlabel("Frequency (Hz)")
plt.ylabel("Log not dB")
plt.grid()
plt.semilogy(freqs, psd)
```

Out[10]:
```
[<matplotlib.lines.Line2D at 0x22b22948c70>]
```

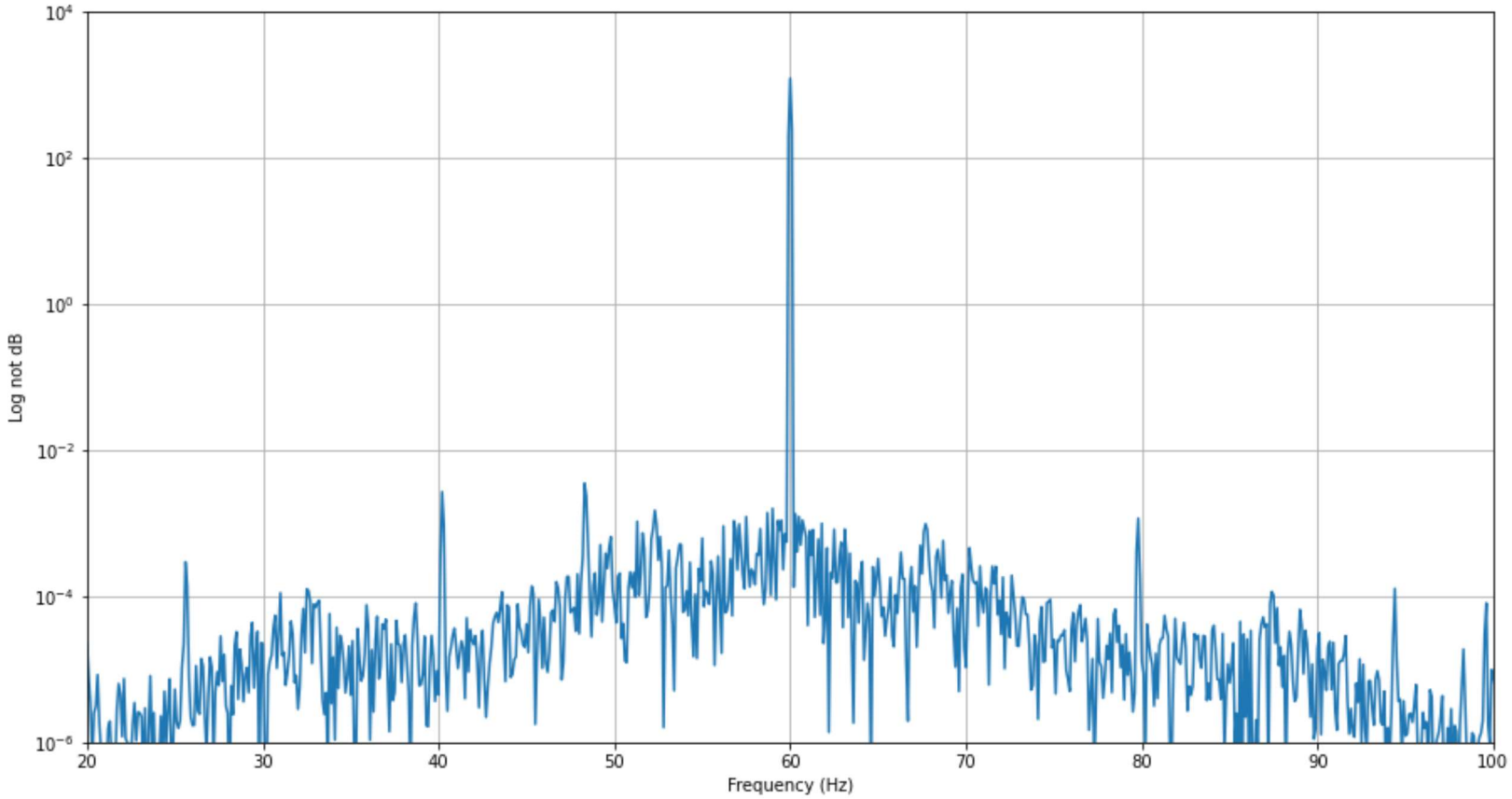


In [11]:

```
plt.figure(figsize=(15,8))
plt.xlim([20,100])
plt.ylim([10e-7,10e3])
plt.xlabel("Frequency (Hz)")
plt.ylabel("Log not dB")
plt.grid()
plt.semilogy(freqs,psd)
```

Out[11]:

```
[<matplotlib.lines.Line2D at 0x22b257b7850>]
```



Convert the Y-Axis to dB in Current

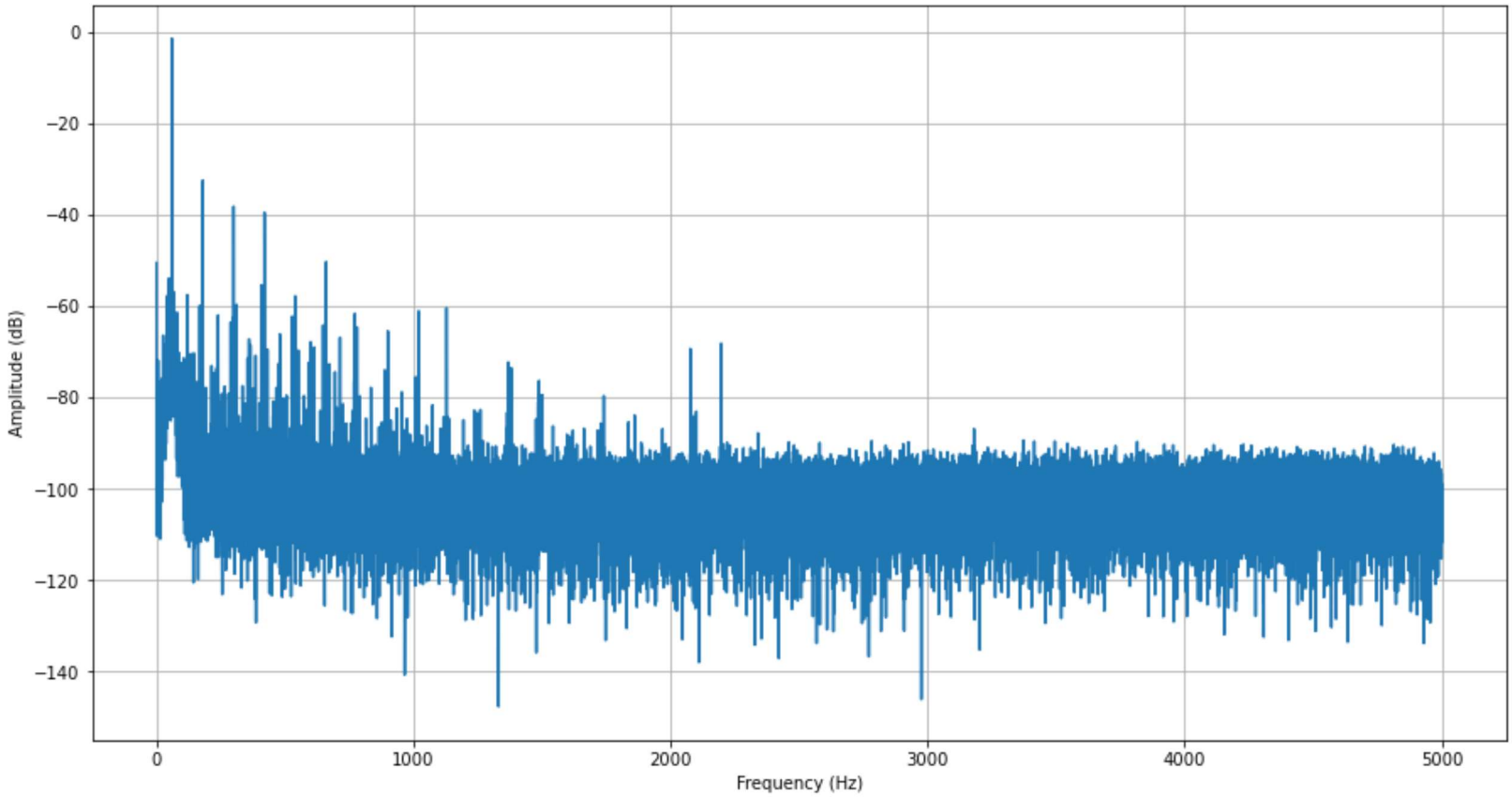
To this point in our exersize we have not dealt with averages nor changing the y-axis from logarithmic mean to actual -dB (decibels), which is what we use. For current dB the conversion is 20*log10. We will first convert everything we need to current dB (-dB) and then will review the data in a Flat Top window. One of the challenges we have with the dataset above is that it is too short to perform any reasonable averages and have sharp peaks as we would have to change 'nperseg' to some other value. For this exercise we will start with dB and then work on averages next week. We will also compare a larger sampling to this sample as a comparison.

For our purposes, we will not go through the extra code to show the values as negative. There's another reason for this.

In [46]:

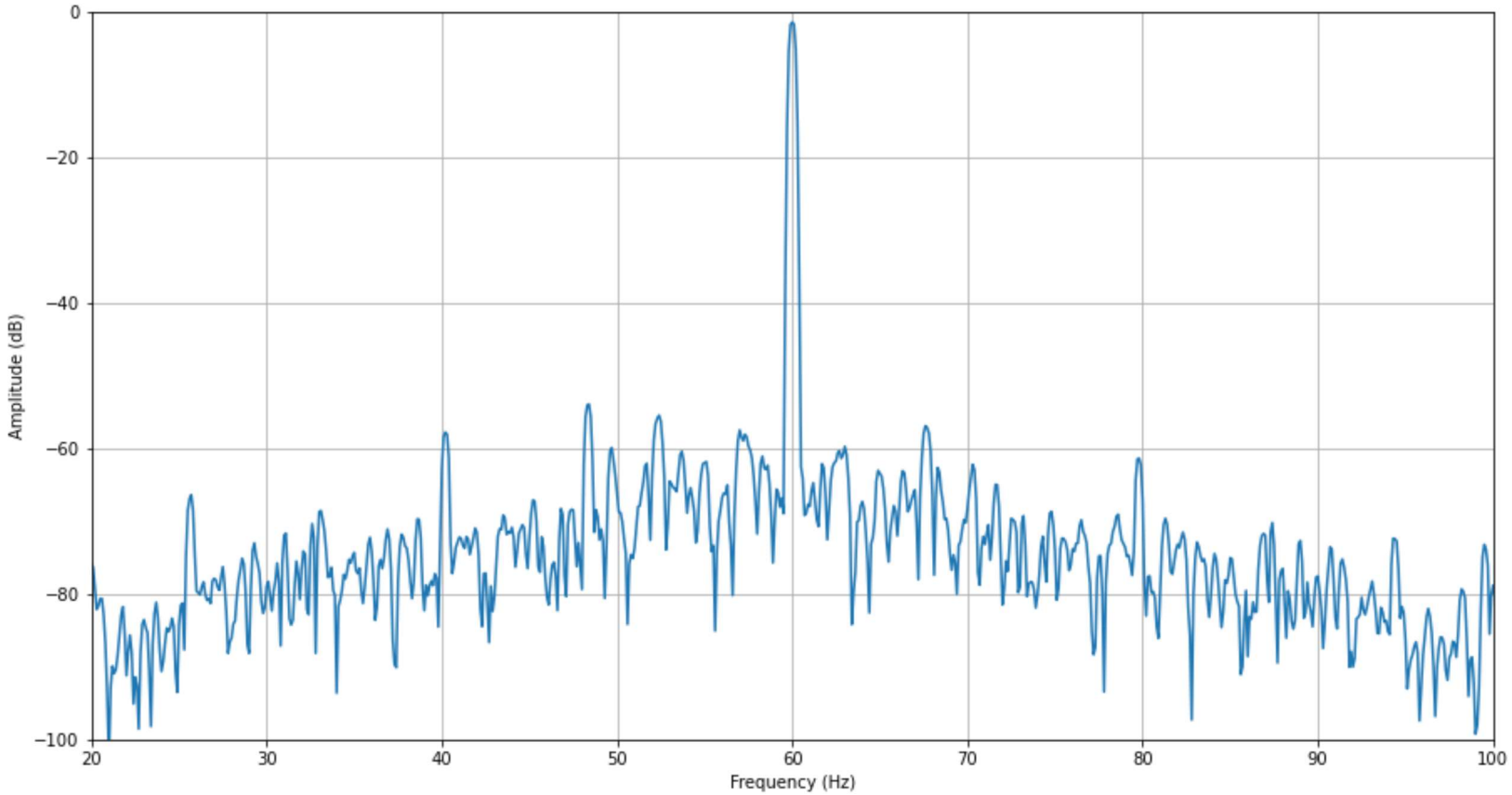
```
#This set of code will convert everything from before to dB as positive values.  Just to make it confusing,
#we still need to compare the peak of the line frequency to the peak of the value of interest.  In commercial
#systems the peak will normally be presented as '0' and all other values as '-dB' or 'dB down.'

#first we will define a function as dbfft
def dbfft(x, fs, win=None, ref=21474836):
```

```
    #Calculate spectrum in dB scale
    #Args:
        #x: input signal
        #fs: sampling frequency
        #win: vector containing window samples (same length as x)
        #ref: reference value used for dBFS scale.  32768 for int16 and 1 for float.
    #Returns:
        #freq: frequency vector
        #s_db: spectrum in dB scale

    N = len(normalized_data) #Length of input sequency.  We will use the conditioned data.
    if win is None:
        win=np.ones(1,N)
    if len(normalized_data)!=len(win):
        raise ValueError('Signal and window must be of the same length')
    x = normalized_data*win

    #calculate the real FFT and frequency vector
    sp = rfft(x)
    freq = np.arange((N/2)+1)/(float(N)/fs)

    #scale the magnitude of FFT by window and factor of 2
    #as we are using half of FFT spectrum
    s_mag = np.abs(sp)*2/np.sum(win)

    #convert to dBFS
    s_dbfs = 20*np.log10(s_mag/ref)

    return freq, s_dbfs

#def main():
#load the file
freq, s_dbfs = welch(normalized_data, fs=10000,window='flattop',nperseg=100000)

N=100000
win=flattop(N)
fs=10000
freq, s_dbfs=dbfft(normalized_data[0:N],fs,win)

#scale from dbfs to db
K=120
s_db=s_dbfs + K

plt.figure(figsize=(15,8))
plt.plot(freq, s_db)
plt.grid()
plt.xlabel('Frequency (Hz)')
plt.ylabel('Amplitude (dB)')
plt.show()
```



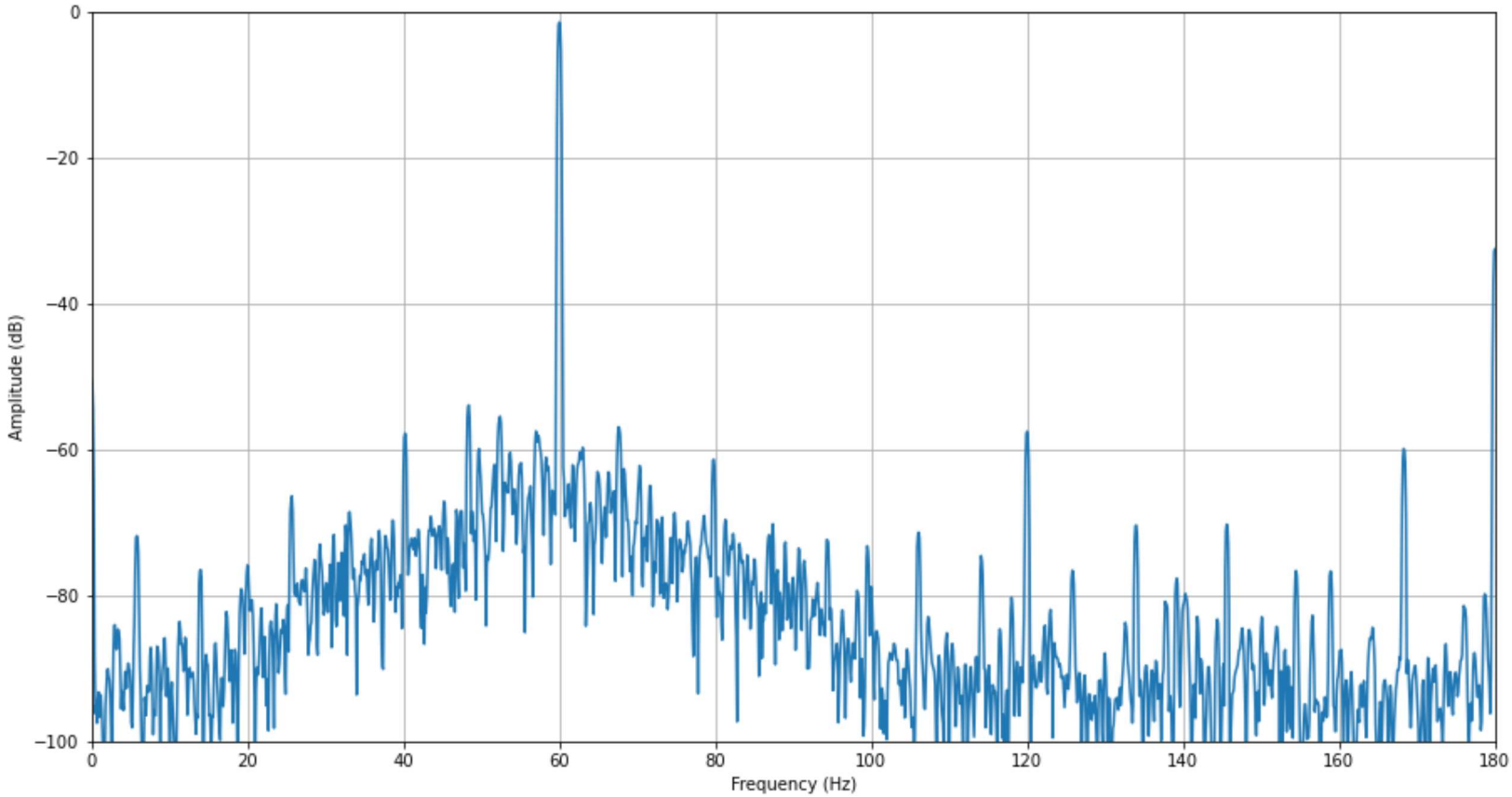
In [50]:

```
#now we will come in closer and crop the data so that we are looking at the 1X RPM and around the line frequency.
plt.figure(figsize=(15,8))
plt.plot(freq, s_db)
plt.xlim([20,100])
plt.ylim([-100,0])
plt.grid()
plt.xlabel('Frequency (Hz)')
plt.ylabel('Amplitude (dB)')
plt.show()
```



In [52]:

```
#in this one lets zoom out to see up to 180 Hz.
plt.figure(figsize=(15,8))
plt.plot(freq, s_db)
plt.xlim([0,180])
plt.ylim([-100,0])
plt.grid()
plt.xlabel('Frequency (Hz)')
plt.ylabel('Amplitude (dB)')
plt.show()
```



For more information follow the Motor Diagnostics and Motor Health Newsletter by signing up on any of the MotorDoc LLC official websites: www.motordoc.com www.empathcms.com or www.motordocai.io

In [ ]: