In [3]:
```
#Transforming audio signals to the frequency domain

import numpy as np
import matplotlib.pyplot as plt
from scipy.io import wavfile
```

In [4]:
```
# Read the audio file
sampling_freq, signal = wavfile.read('file_example_WAV_1MG.wav')
sampling_freq
```

Out[4]:
```
44100
```

In [5]:
```
signal
```

Out[5]:
```
array([ 4395, 15134, 19572, ..., -5859,   701,  7220], dtype=int16)
```

In [7]:
```
# Normalize the values
signal = signal / np.power(2, 15)
signal
```

Out[7]:
```
array([ 4.09316272e-06,  1.40946358e-05,  1.82278454e-05, ...,
       -5.45661896e-06,  6.52857125e-07,  6.72414899e-06])
```

In [8]:
```
# Extract the length of the audio signal
len_signal = len(signal)
len_signal
```

Out[8]:
```
176400
```

In [9]:
```
# Extract the half length
len_half = np.ceil((len_signal + 1) / 2.0).astype(np.int)
len_half
```

```
<ipython-input-9-01ad0ebda8d0>:2: DeprecationWarning: `np.int` is a deprecated alias for the builtin `int`. To silence this warning, use `int` by itself. Doin
g this will not modify any behavior and is safe. When replacing `np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you w
ish to review your current use, check the release note link for additional information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  len_half = np.ceil((len_signal + 1) / 2.0).astype(np.int)
```

Out[9]:
```
88201
```

In [10]:
```
# Apply Fourier transform
freq_signal = np.fft.fft(signal)
freq_signal
```

Out[10]:
```
array([ 1.35409559e+00+0.j        , -3.86887177e-04-0.00086196j,
       -1.41686190e-04-0.00163807j, ...,  4.18115153e-04+0.00230829j,
       -1.41686190e-04+0.00163807j, -3.86887177e-04+0.00086196j])
```

In [11]:
```
# Normalization
freq_signal = abs(freq_signal[0:len_half]) / len_signal
freq_signal
```

Out[11]:
```
array([7.67627886e-06, 5.35606006e-09, 9.32077569e-09, ...,
       2.24123364e-09, 1.80002006e-09, 4.80554529e-09])
```

In [12]:
```
# Take the square
freq_signal **= 2
freq_signal
```

Out[12]:
```
array([5.89252571e-11, 2.86873793e-17, 8.68768595e-17, ...,
       5.02312823e-18, 3.24007223e-18, 2.30932655e-17])
```

In [13]:
```
# Extract the length of the frequency transformed signal
len_fts = len(freq_signal)
len_fts
```

Out[13]:
```
88201
```

In [14]:
```
# Adjust the signal for even and odd cases
if len_signal % 2:
    freq_signal[1:len_fts] *= 2
else:
    freq_signal[1:len_fts-1] *= 2
freq_signal
```

Out[14]:
```
array([5.89252571e-11, 5.73747586e-17, 1.73753719e-16, ...,
       1.00462565e-17, 6.48014446e-18, 2.30932655e-17])
```

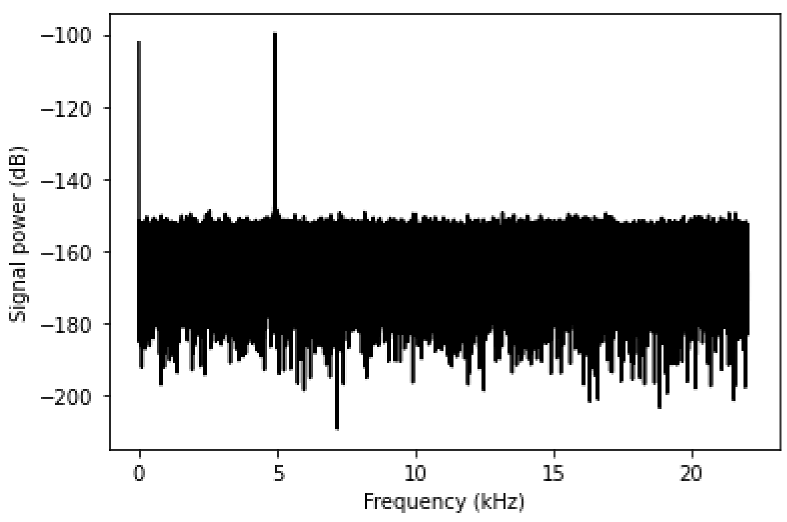
In [15]:
```
# Extract the power value in dB
signal_power = 10 * np.log10(freq_signal)
signal_power
```

Out[15]:
```
array([-102.29698514, -162.41279128, -157.60065891, ..., -169.9799574 ,
       -171.88415313, -166.36514651])
```

In [16]:
```
# Build the X axis
x_axis = np.arange(0, len_half, 1) * (sampling_freq / len_signal) / 1000.0
x_axis
```

Out[16]:
```
array([0.000000e+00, 2.500000e-04, 5.000000e-04, ..., 2.204950e+01,
       2.204975e+01, 2.205000e+01])
```

In [17]:
```
# Plot the figure
plt.figure()
plt.plot(x_axis, signal_power, color='black')
plt.xlabel('Frequency (kHz)')
plt.ylabel('Signal power (dB)')
plt.show()
```



In [ ]: