In [1]:
```
import os

if not os.path.exists('thinkdsp.py'):
    !wget https://github.com/AllenDowney/ThinkDSP/raw/master/code/thinkdsp.py
```

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt

from thinkdsp import decorate
```

exer01

In [3]:
```
if not os.path.exists('132736__ciccarelli__ocean-waves.wav'):
    !wget https://github.com/AllenDowney/ThinkDSP/raw/master/code/132736__ciccarelli
```

In [4]:
```
from thinkdsp import read_wave

wave = read_wave('132736__ciccarelli__ocean-waves.wav')
wave.make_audio()
```

Out[4]:

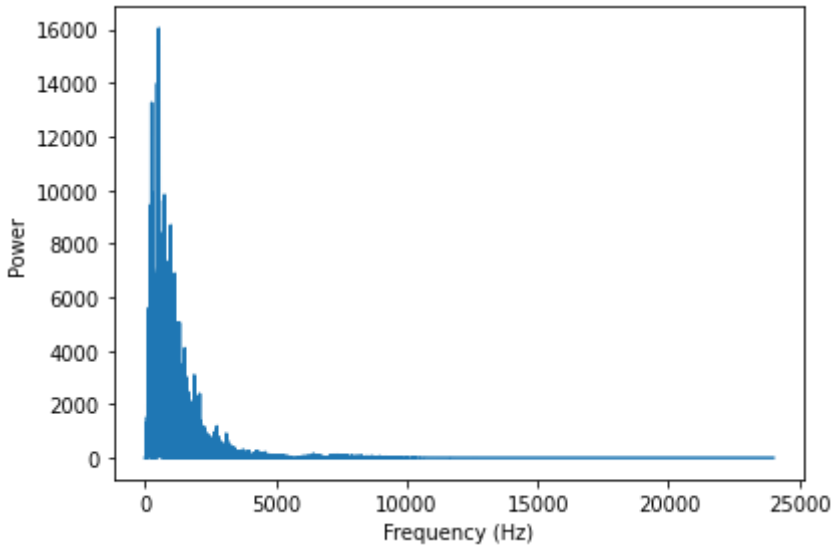
0:00 / 1:14

In [5]:
```
segment = wave.segment(start=1.5, duration=1.0)
segment.make_audio()
```

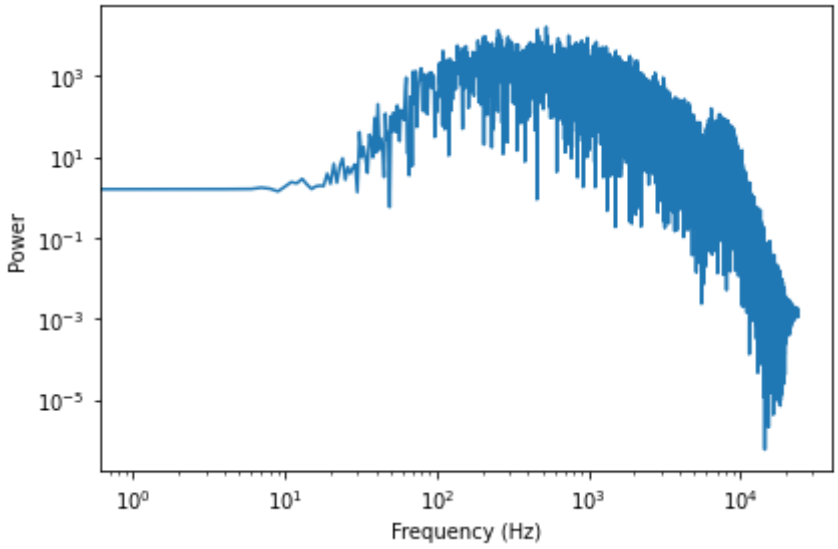
Out[5]:

0:00 / 0:01

In [6]:
```
spectrum = segment.make_spectrum()
spectrum.plot_power()
decorate(xlabel='Frequency (Hz)',
         ylabel='Power')
```



In [7]:
```
spectrum.plot_power()

loglog = dict(xscale='log', yscale='log')
decorate(xlabel='Frequency (Hz)',
         ylabel='Power',
         **loglog)
```

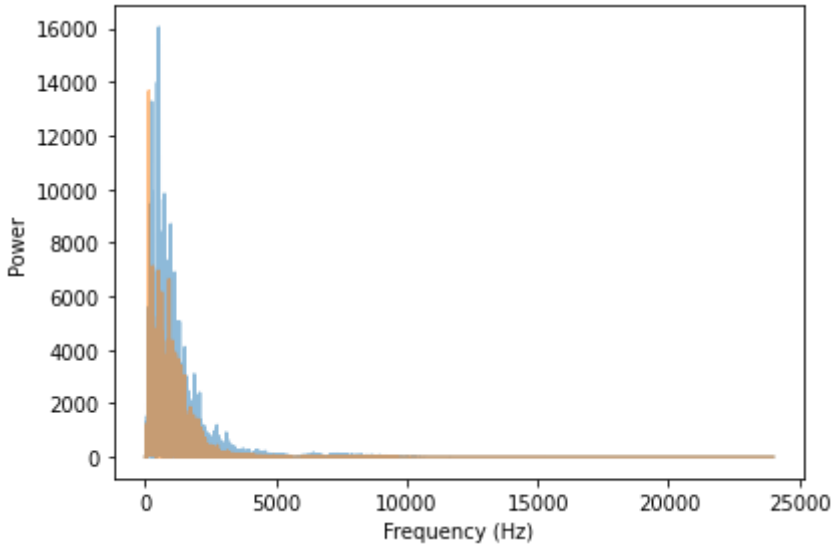


In [8]:
```
segment2 = wave.segment(start=2.5, duration=1.0)
segment2.make_audio()
```

Out[8]:

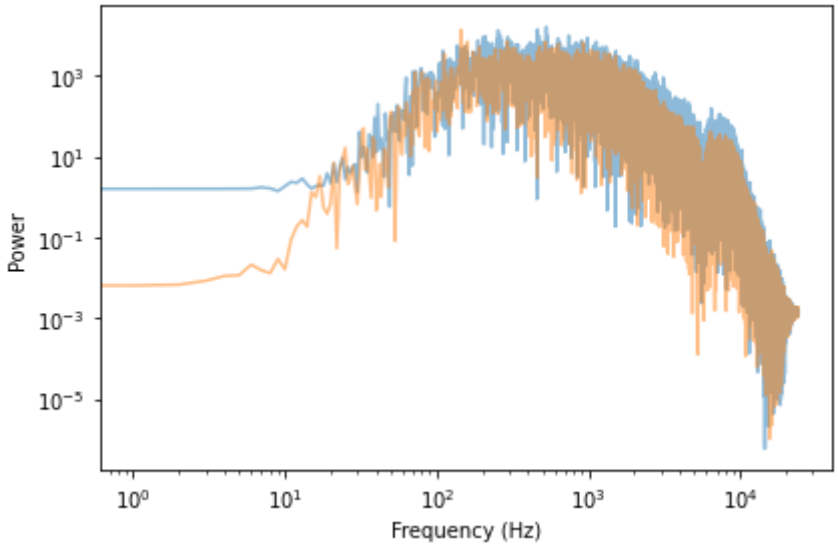
0:00 / 0:01

In [9]:
```
spectrum2 = segment2.make_spectrum()

spectrum.plot_power(alpha=0.5)
spectrum2.plot_power(alpha=0.5)
decorate(xlabel='Frequency (Hz)',
         ylabel='Power')
```



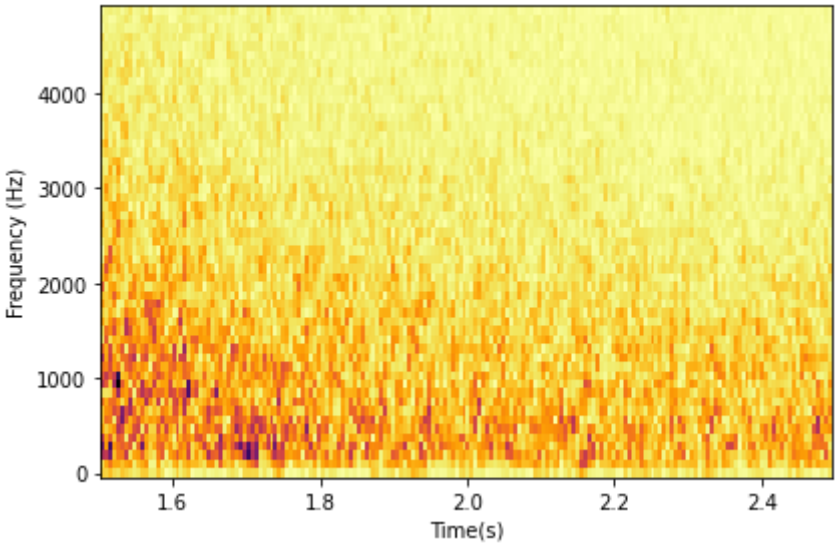
In [10]:
```
spectrum.plot_power(alpha=0.5)
spectrum2.plot_power(alpha=0.5)
decorate(xlabel='Frequency (Hz)',
```

```
        ylabel='Power',
        **loglog)
```



In [11]:

```
segment.make_spectrogram(512).plot(high=5000)
decorate(xlabel='Time(s)', ylabel='Frequency (Hz)')
```



exer02

In [12]:

```
if not os.path.exists('BTC_USD_2013-10-01_2020-03-26-CoinDesk.csv'):
    !wget https://github.com/AllenDowney/ThinkDSP/raw/master/code/BTC_USD_2013-10-01
```

In [13]:

```
import pandas as pd

df = pd.read_csv('BTC_USD_2013-10-01_2020-03-26-CoinDesk.csv',
                 parse_dates=[0])
df
```

Out[13]:

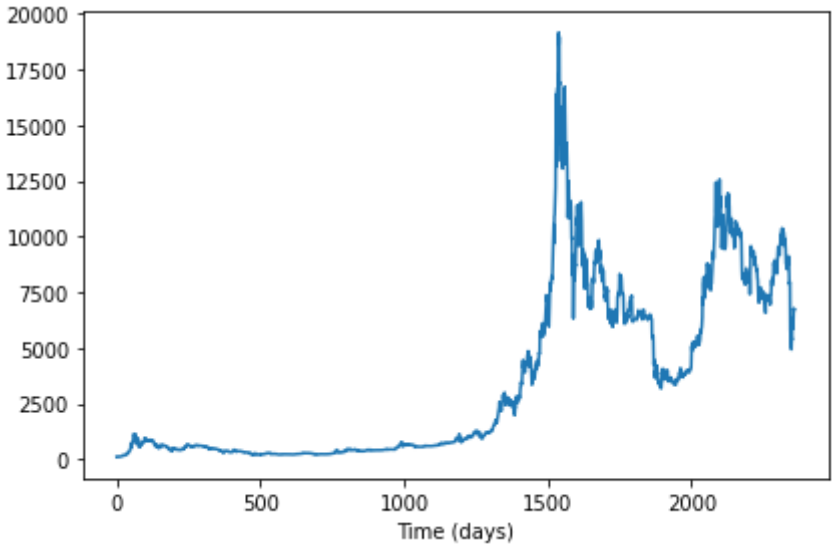
	Currency	Date	Closing Price (USD)	24h Open (USD)	24h High (USD)	24h Low (USD)
0	BTC	2013-10-01	123.654990	124.304660	124.751660	122.563490

	Currency	Date	Closing Price (USD)	24h Open (USD)	24h High (USD)	24h Low (USD)
1	BTC	2013-10-02	125.455000	123.654990	125.758500	123.633830
2	BTC	2013-10-03	108.584830	125.455000	125.665660	83.328330
3	BTC	2013-10-04	118.674660	108.584830	118.675000	107.058160
4	BTC	2013-10-05	121.338660	118.674660	121.936330	118.005660
...	...	...	...	...	...	...
2354	BTC	2020-03-22	5884.340133	6187.042146	6431.873162	5802.553402
2355	BTC	2020-03-23	6455.454688	5829.352511	6620.858253	5694.198299
2356	BTC	2020-03-24	6784.318011	6455.450650	6863.602196	6406.037439
2357	BTC	2020-03-25	6706.985089	6784.325204	6981.720386	6488.111885
2358	BTC	2020-03-26	6721.495392	6697.948320	6796.053701	6537.856462

2359 rows × 6 columns

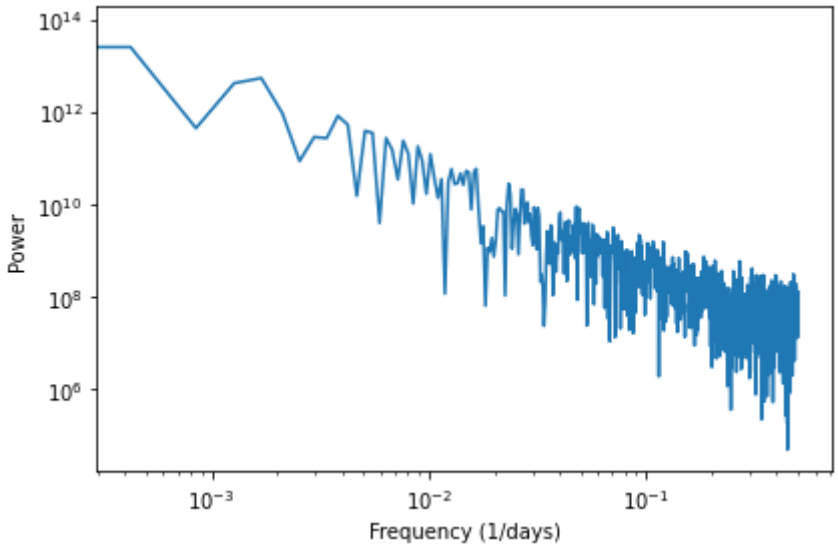
In [14]:
```
ys = df['Closing Price (USD)']
ts = df.index
```

In [15]:
```
from thinkdsp import Wave

wave = Wave(ys, ts, framerate=1)
wave.plot()
decorate(xlabel='Time (days)')
```



In [16]:
```
spectrum = wave.make_spectrum()
```

```
spectrum.plot_power()
decorate(xlabel='Frequency (1/days)',
         ylabel='Power',
         **loglog)
```



In [17]:
```
spectrum.estimate_slope()[0]
```

Out[17]: -1.733254093675894

exer03

In [23]:
```
from thinkdsp import Noise

class UncorrelatedPoissonNoise(Noise):
    """Represents uncorrelated Poisson noise."""

    def evaluate(self, ts):
        """Evaluates the signal at the given times.

        ts: float array of times

        returns: float wave array
        """
        ys = np.random.poisson(self.amp, len(ts))
        return ys
```

In [24]:
```
amp = 0.001
framerate = 10000
duration = 1

signal = UncorrelatedPoissonNoise(amp=amp)
wave = signal.make_wave(duration=duration, framerate=framerate)
wave.make_audio()
```

Out[24]: 0:00 / 0:01

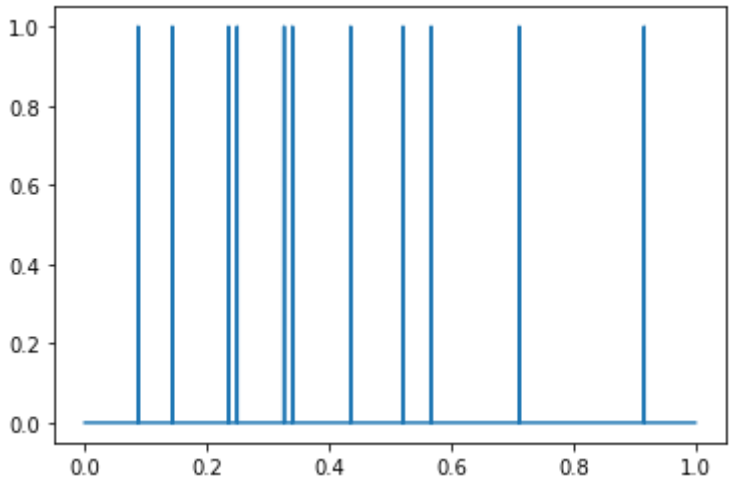
In [25]:
```
expected = amp * framerate * duration
actual = sum(wave.ys)
```

```
print(expected, actual)
```

```
10.0 11
```

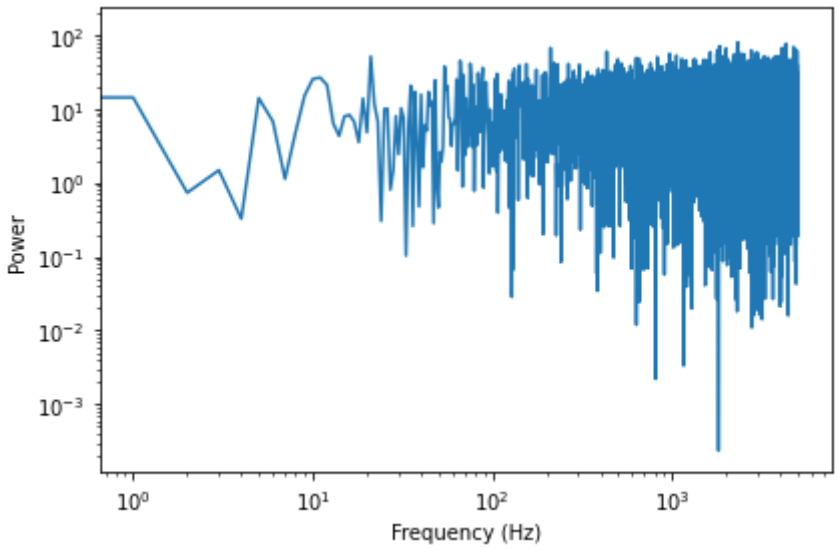
In [26]:

```
wave.plot()
```



In [27]:

```
spectrum = wave.make_spectrum()
spectrum.plot_power()
decorate(xlabel='Frequency (Hz)',
         ylabel='Power',
         **loglog)
```



In [28]:

```
spectrum.estimate_slope().slope
```

Out[28]:

```
0.014075885614474239
```

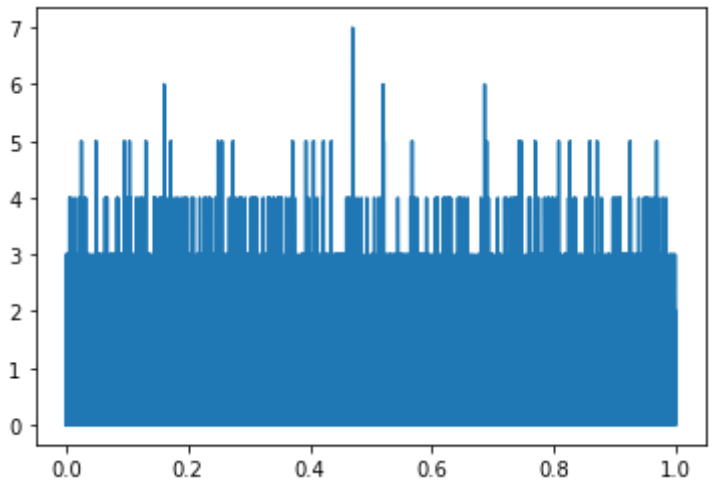
In [29]:

```
amp = 1
framerate = 10000
duration = 1

signal = UncorrelatedPoissonNoise(amp=amp)
wave = signal.make_wave(duration=duration, framerate=framerate)
wave.make_audio()
```

Out[29]:

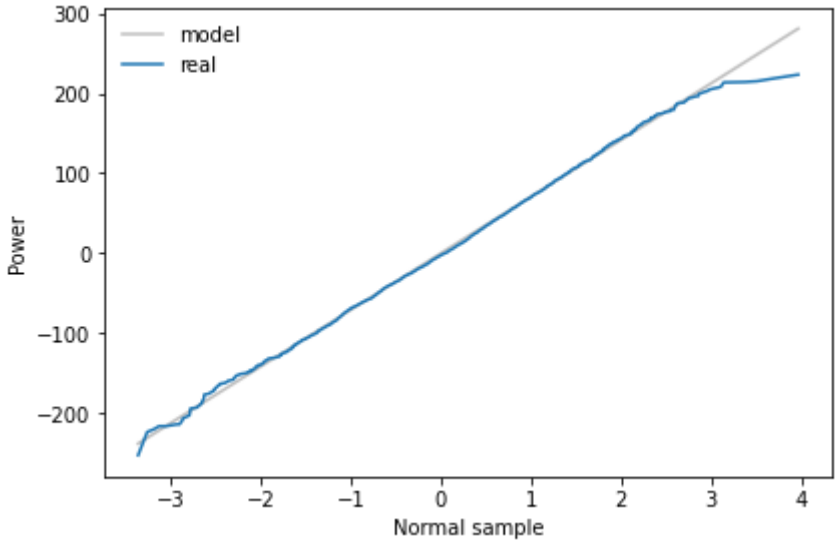
0:00 / 0:01

In [30]:
```
wave.plot()
```

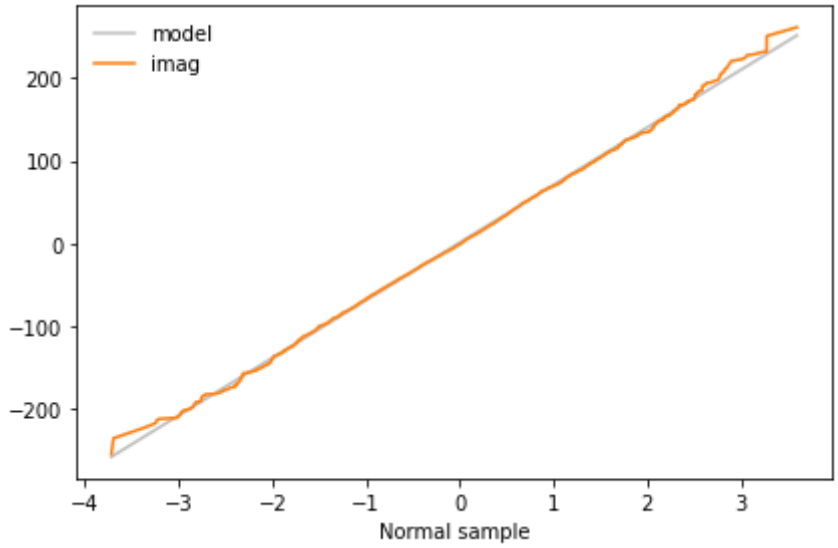


In [31]:
```
import matplotlib.pyplot as plt

def normal_prob_plot(sample, fit_color='0.8', **options):
    """Makes a normal probability plot with a fitted line.

    sample: sequence of numbers
    fit_color: color string for the fitted line
    options: passed along to Plot
    """
    n = len(sample)
    xs = np.random.normal(0, 1, n)
    xs.sort()

    ys = np.sort(sample)

    mean, std = np.mean(sample), np.std(sample)
    fit_ys = mean + std * xs
    plt.plot(xs, fit_ys, color='gray', alpha=0.5, label='model')

    plt.plot(xs, ys, **options)
```

In [32]:
```
spectrum = wave.make_spectrum()
spectrum.hs[0] = 0

normal_prob_plot(spectrum.real, label='real')
decorate(xlabel='Normal sample',
         ylabel='Power')
```



In [33]:
```
normal_prob_plot(spectrum.imag, label='imag', color='C1')
decorate(xlabel='Normal sample')
```



In [ ]:
```
```