```
        ys = np.random.normal(0, self.amp, len(ts))
        return ys
```

`np.random.normal` returns a NumPy array of values from a Gaussian distribution, in this case with mean 0 and standard deviation `self.amp`. In theory the range of values is from negative to positive infinity, but we expect about 99% of the values to be between -3 and 3.

UG noise is similar in many ways to UU noise. The spectrum has equal power at all frequencies, on average, so UG is also white. And it has one other interesting property: the spectrum of UG noise is also UG noise. More precisely, the real and imaginary parts of the spectrum are uncorrelated Gaussian values.

To test that claim, we can generate the spectrum of UG noise and then generate a "normal probability plot", which is a graphical way to test whether a distribution is Gaussian.

```
signal = thinkdsp.UncorrelatedGaussianNoise()
wave = signal.make_wave(duration=0.5, framerate=11025)
spectrum = wave.make_spectrum()

thinkstats2.NormalProbabilityPlot(spectrum.real)
thinkstats2.NormalProbabilityPlot(spectrum.imag)
```

`NormalProbabilityPlot` is provided by `thinkstats2`, which is included in the repository for this book. If you are not familiar with normal probability plots, you can read about them in Chapter 5 of *Think Stats* at `http://thinkstats2.com`.

Figure 4.8 shows the results. The gray lines show a linear model fit to the data; the dark lines show the data.

A straight line on a normal probability plot indicates that the data come from a Gaussian distribution. Except for some random variation at the extremes, these lines are straight, which indicates that the spectrum of UG noise is UG noise.

The spectrum of UU noise is also UG noise, at least approximately. In fact, by the Central Limit Theorem, the spectrum of almost any uncorrelated noise is approximately Gaussian, as long as the distribution has finite mean and standard deviation, and the number of samples is large.

4.6 Exercises

Solutions to these exercises are in `chap04soln.ipynb`.

Exercise 4.1 "A Soft Murmur" is a web site that plays a mixture of natural noise sources, including rain, waves, wind, etc. At `http://asoftmurmur.com/about/` you can find their list of recordings, most of which are at `http://freesound.org`.

Download a few of these files and compute the spectrum of each signal. Does the power spectrum look like white noise, pink noise, or Brownian noise? How does the spectrum vary over time?

Exercise 4.2 In a noise signal, the mixture of frequencies changes over time. In the long run, we expect the power at all frequencies to be equal, but in any sample, the power at each frequency is random.

To estimate the long-term average power at each frequency, we can break a long signal into segments, compute the power spectrum for each segment, and then compute the average across the segments. You can read more about this algorithm at `http://en.wikipedia.org/wiki/Bartlett's_method`.

Implement Bartlett's method and use it to estimate the power spectrum for a noise wave. Hint: look at the implementation of `make_spectrogram`.

Exercise 4.3 At `http://www.coindesk.com` you can download the daily price of a BitCoin as a CSV file. Read this file and compute the spectrum of BitCoin prices as a function of time. Does it resemble white, pink, or Brownian noise?

Exercise 4.4 A Geiger counter is a device that detects radiation. When an ionizing particle strikes the detector, it outputs a surge of current. The total output at a point in time can be modeled as uncorrelated Poisson (UP) noise, where each sample is a random quantity from a Poisson distribution, which corresponds to the number of particles detected during an interval.

Write a class called `UncorrelatedPoissonNoise` that inherits from `thinkdsp._Noise` and provides `evaluate`. It should use `np.random.poisson` to generate random values from a Poisson distribution. The parameter of this function, `lam`, is the average number of particles during each interval. You can use the attribute `amp` to specify `lam`. For example, if the framerate is 10 kHz and `amp` is 0.001, we expect about 10 "clicks" per second.

Generate about a second of UP noise and listen to it. For low values of `amp`, like 0.001, it should sound like a Geiger counter. For higher values it should sound like white noise. Compute and plot the power spectrum to see whether it looks like white noise.

Exercise 4.5 The algorithm in this chapter for generating pink noise is conceptually simple but computationally expensive. There are more efficient alternatives, like the Voss-McCartney algorithm. Research this method, implement

it, compute the spectrum of the result, and confirm that it has the desired relationship between power and frequency.