Digital Signal Processing

Lecture 6 – Noise

상명대학교
컴퓨터과학과
강상욱 교수

Definition of noise

- As in an English dictionary
  - An unwanted signal of any kind. If two signals interfere with each other, each signal would consider the other to be noise.
- Second, in the signal processing
  - A signal that contains components at many frequencies, so it lacks the harmonic structure of the periodic signals.

Uncorrelated uniform noise

- The simplest kind to generate a noise is uncorrelated uniform noise (UU noise)
  - Uniform : the signal contains random values from a uniform distribution; that is, every value in the range is equally likely.
  - Uncorrelated : the values are independent; that is, knowing one value provides no information about the others.

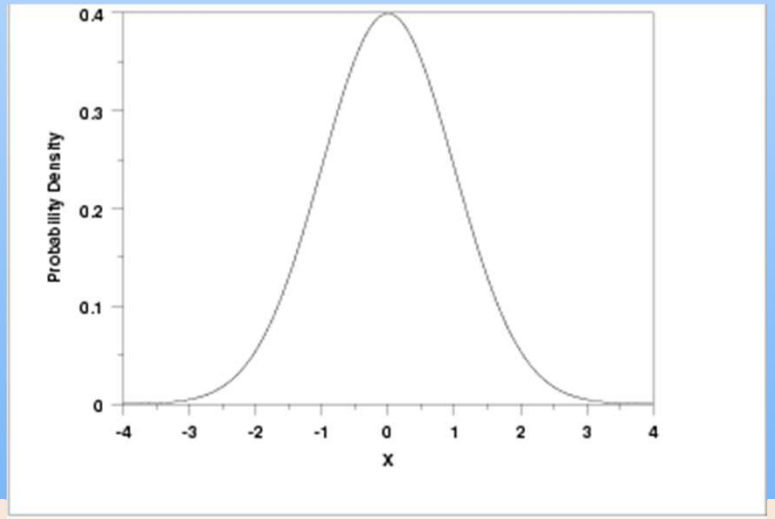
```
class UncorrelatedUniformNoise(_Noise):

    def evaluate(self, ts):
        ys = np.random.uniform(-self.amp, self.amp, len(ts))
        return ys
```

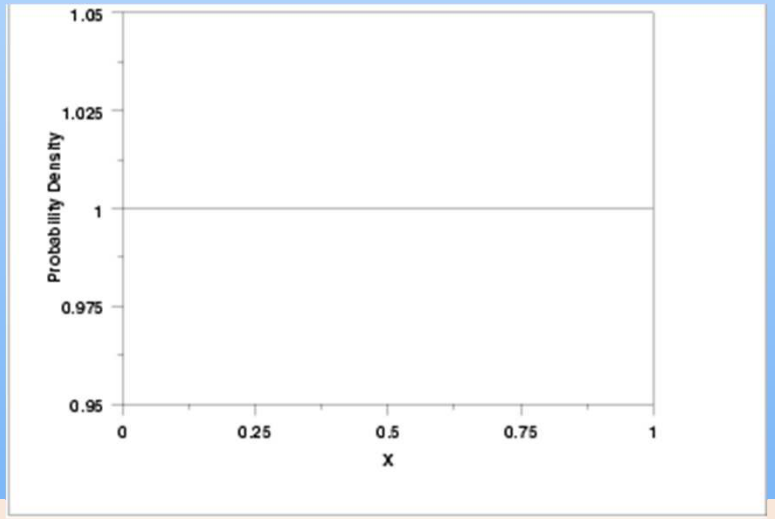
As usual, the evaluate function takes `ts`, the times when the signal should be evaluated. It uses `np.random.uniform`, which generates values from a uniform distribution. In this example, the values are in the range between `-amp` to `amp`.

Uniform distribution

- Probability density function
  - The probability that x can take a specific value.
  - $P[X = x] = p(x) = p_x$
  - $p(x)$ is non-negative for all real x.
  - $\sum_j p_j = 1$
  - $0 \leq p(x) \leq 1$



Normal distribution



Uniform distribution

UU noise signal

```
signal = thinkdsp.UncorrelatedUniformNoise()
wave = signal.make_wave(duration=0.5, framerate=11025)
```

Generate UU noise with duration 0.5 seconds at 11,025 samples per second

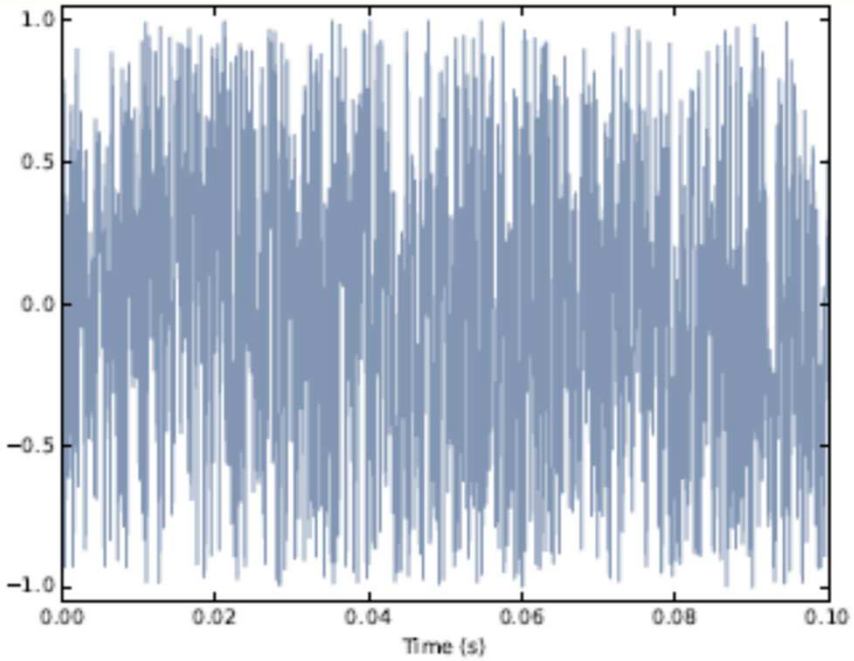


Figure 4.1: Waveform of uncorrelated uniform noise.

UU noise spectrum

```
spectrum = wave.make_spectrum()
spectrum.plot_power()
```

Similar to spectrum.plot( ), except that it plots power instead of amplitude. Power is the square of amplitude.

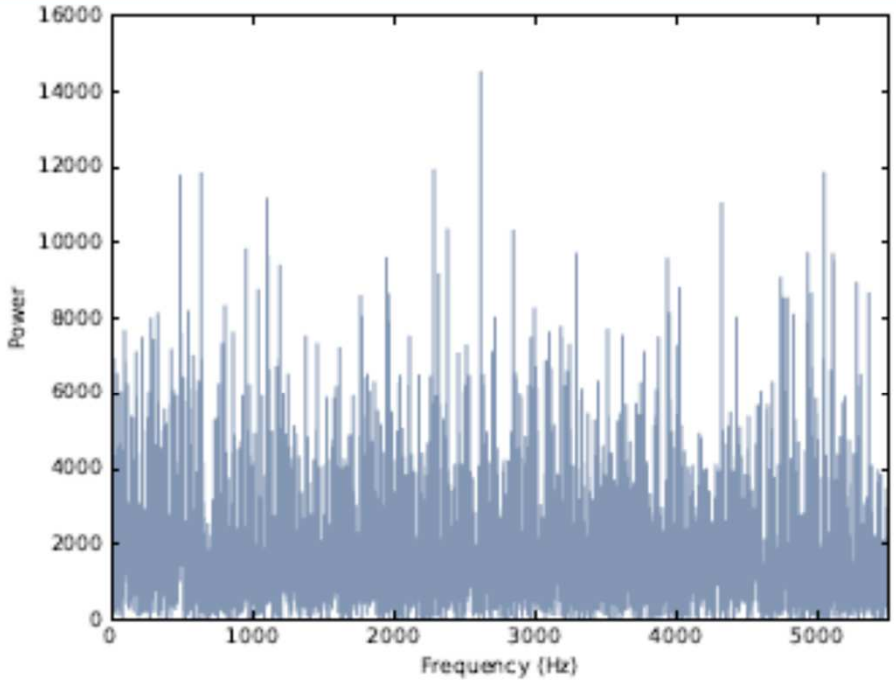


Figure 4.2: Power spectrum of uncorrelated uniform noise.

Three things to know about a noise

- Distribution
  - The set of possible values and their probabilities.
  - In the uniform noise signal, the set of values is the range from -1 to 1, and all values have the same probability.
  - In the Gaussian noise, the set of values is the range from negative to positive infinity, but values near 0 are the most likely, with probability that drops off according to "bell" curve.
- Correlation
  - Is each value in the signal independent of the others?
  - In UU noise, each noise value is independent.
  - In Brownian noise, each value is the sum of the previous value and a random "step". So each value is dependent on the previous values.
- Relationship between power and freq.
  - In the spectrum of UU noise, the average power is the same for all frequencies.
  - In the spectrum of pink noise, the power is inversely related to freq.

Integrated spectrum I

- Integrated spectrum shows the cumulative power in the spectrum up to f.
- Spectrum provides a method that computes the IS.

```
def make_integrated_spectrum(self):
    cs = np.cumsum(self.power)
    cs /= cs[-1]
    return IntegratedSpectrum(cs, self.fs)
```

`self.power` is a NumPy array containing power for each frequency. `np.cumsum` computes the cumulative sum of the powers. Dividing through by the last element normalizes the integrated spectrum so it runs from 0 to 1.

- IntegratedSpectrum provides plot_power.

```
integ = spectrum.make_integrated_spectrum()
integ.plot_power()
```

Integrated spectrum 2

Noise with equal power at all frequencies is called white noise

White noise is an analogy with light, because an equal mixture of light at all visible frequencies is white

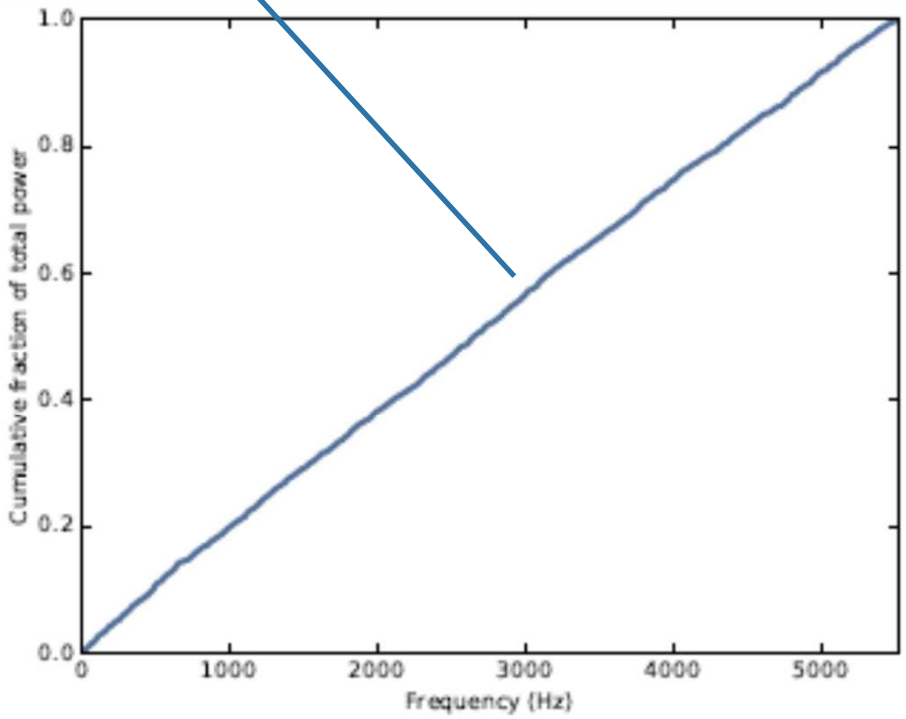


Figure 4.3: Integrated spectrum of uncorrelated uniform noise.

Brownian noise (red noise) I

- Each value is the sum of the previous value and a random "step".
  - It is an analogy with Brownian motion, in which a particle suspended in a fluid moves apparently at random, due to unseen interactions with the fluid.
  - Random walk : a mathematical model of a path where the distance between steps is characterized by a random distribution.
  - In a one-dimensional random walk, the particle moves up or down by a random amount at each step.

- The location of the particle at any point in time is the sum of all previous steps.
- When the amplitude is high, it tents to be stay high, and vice versa.

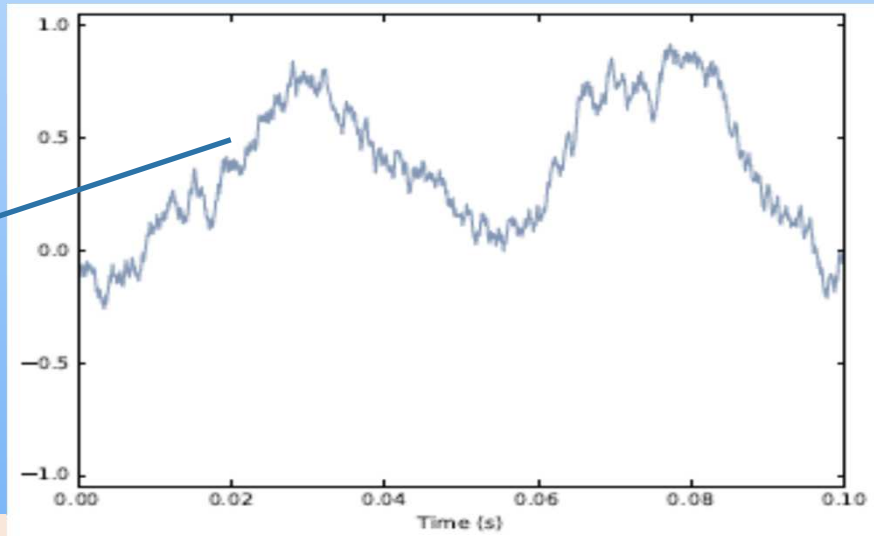


Figure 4.4: Waveform of Brownian noise.

Brownian noise 2

```
class BrownianNoise(_Noise):

    def evaluate(self, ts):
        dys = np.random.uniform(-1, 1, len(ts))
        ys = np.cumsum(dys)
        ys = normalize(unbias(ys), self.amp)
        return ys
```

`evaluate` uses `np.random.uniform` to generate an uncorrelated signal and `np.cumsum` to compute their cumulative sum.

Since the sum is likely to escape the range from -1 to 1, we have to use `unbias` to shift the mean to 0, and `normalize` to get the desired maximum amplitude.

Brownian noise 3

```
signal = thinkdsp.BrownianNoise()
wave = signal.make_wave(duration=0.5, framerate=11025)
wave.plot()
```

Waveform

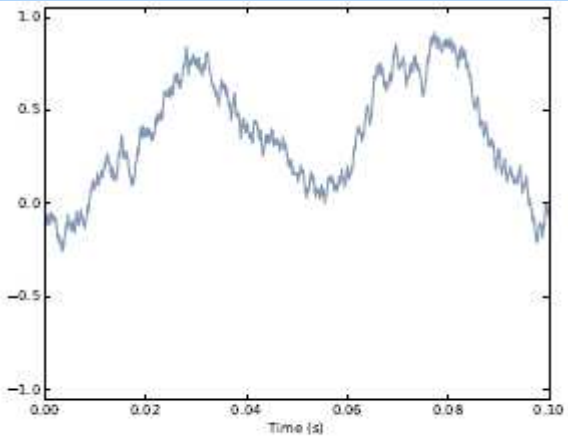
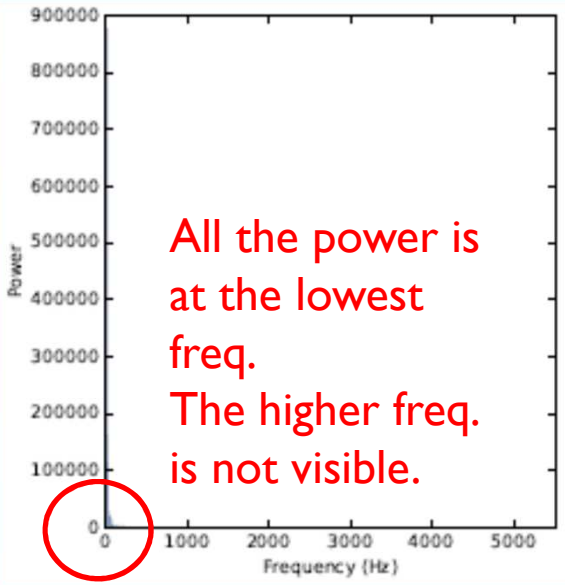
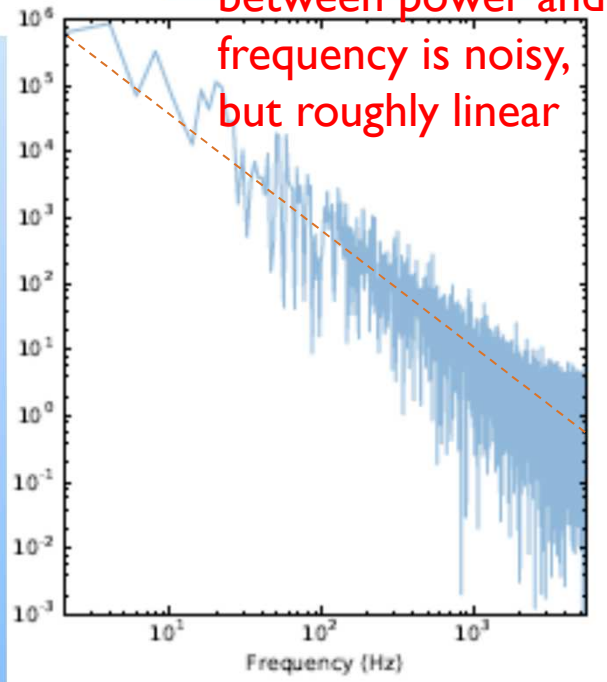


Figure 4.4: Waveform of Brownian noise.



All the power is at the lowest freq.
The higher freq. is not visible.

The relationship between power and frequency is noisy, but roughly linear



Linear scale

Log-log scale

```
spectrum = wave.make_spectrum()
spectrum.plot_power(linewidth=1, alpha=0.5)
thinkplot.config(xscale='log', yscale='log')
```

Brownian noise - slope

```
#class Spectrum

    def estimate_slope(self):
        x = np.log(self.fs[1:])
        y = np.log(self.power[1:])
        t = scipy.stats.linregress(x,y)
        return t
```

For Brownian noise, the slope of the power spectrum is -2.

$$logP = k - 2logf$$

,where P is power , f is freq, and k is the intercept.

It discards the first component of the spectrum because this component corresponds to $f = 0$, and $\log 0$ is undefined.

`estimate_slope` returns the result from `scipy.stats.linregress`, which is an object that contains the estimated slope and intercept, coefficient of determination (R^2), p-value, and standard error. For our purposes, we only need the slope.

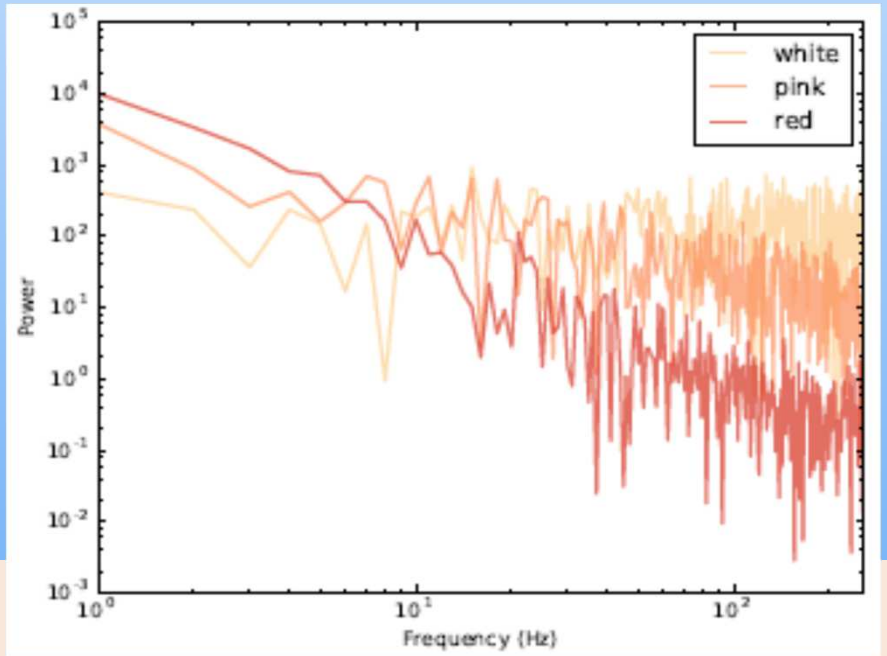
Exponentiating both sides yields.

$$P = \frac{K}{f^2}, K = e^k$$

Power is proportional to $1/f^2$.

Pink noise 1

- We can synthesize noise with any exponent β such that $P = K/f^{\beta}$
  - When $\beta = 0$, power is constant at all frequencies, which is white noise.
  - When $\beta = 2$, it is red noise.
- When β is between 0 and 2, the result is between white and red noise, so it is called pink noise.
- Generating a pink noise
  - One way to generate a pink noise is to generate white noise and then apply a low-pass filter.
  - Thinkdsp provides a class



Pink noise 2

```
class PinkNoise(_Noise):

    def __init__(self, amp=1.0, beta=1.0):
        self.amp = amp
        self.beta = beta
```

amp is the desired amplitude of the signal. beta is the desired exponent. PinkNoise provides make_wave, which generates a Wave.

White noise wave

Low pass filtering

```
def make_wave(self, duration=1, start=0, framerate=11025):
    signal = UncorrelatedUniformNoise()
    wave = signal.make_wave(duration, start, framerate)
    spectrum = wave.make_spectrum()

    spectrum.pink_filter(beta=self.beta)

    wave2 = spectrum.make_wave()
    wave2.unbias()
    wave2.normalize(self.amp)
    return wave2
```

Pink noise 3

Spectrum provides pink_filter:

```
def pink_filter(self, beta=1.0):
    denom = self.fs ** (beta/2.0)
    denom[0] = 1
    self.hs /= denom
```

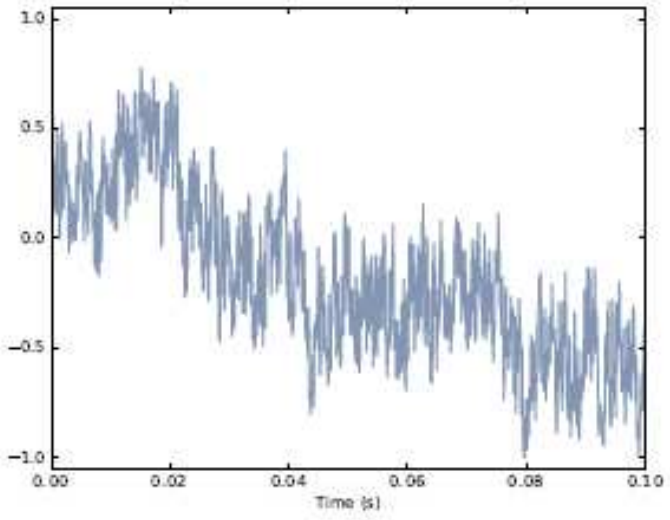


Figure 4.6: Waveform of pink noise with $\beta = 1$.

Like Brownian noise, it wanders up and down, but at least visually, it looks more random.

Gaussian noise

- Another "white noise".
  - White noise means "equal power at all frequencies, on average."
  - UG (uncorrelated Gaussian) noise is also called "white noise".
  - The spectrum of UG noise is also UG noise. (Check during the lab)

```
class UncorrelatedGaussianNoise(_Noise):

    def evaluate(self, ts):
        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.