DSP For Scientists and Engineers Notes

Philip Tracton

September 20, 2017

Contents

1 The Breadth and Depth of DSP **3**

2 Statistics, Probability and Noise **4**
- 2.1 Signals and Graph Terminology . . . 4
  - 2.1.1 Definitions . . . 4
  - 2.1.2 Concepts . . . 4
- 2.2 Mean and Standard Deviation . . . 4
  - 2.2.1 Mean . . . 4
  - 2.2.2 Standard Deviation and Variance . . . 5
  - 2.2.3 Running Statistics . . . 7
- 2.3 Signal vs. Underlying Process . . . 8
- 2.4 The Histogram, PMF and PDF . . . 9
- 2.5 The Normal Distribution . . . 11
- 2.6 Digital Noise Generation . . . 12
- 2.7 Precision and Accuracy . . . 13

3 ADC and DAC **14**
- 3.1 Quantization . . . 14
- 3.2 The Sampling Theorem . . . 15
- 3.3 Digital to Analog Conversion . . . 16
- 3.4 Analog Filters for Data Conversion . . . 16
- 3.5 Selecting the Anti-Alias Filter . . . 16
- 3.6 Multirate Data Conversion . . . 18
- 3.7 Single Bit Data Conversion . . . 18

4 DSP Software **19**
- 4.1 Computer Numbers . . . 19
- 4.2 Fixed Point . . . 19
- 4.3 Floating Point . . . 19
- 4.4 Number Precision . . . 19
- 4.5 Execution Speed: Programming Language . . . 19
- 4.6 Execution Speed: Hardware . . . 19
- 4.7 Execution Speed: Programming Tips . . . 19

1 The Breadth and Depth of DSP

These are my personal notes from learning DSP using The Scientists and Engineers Guide to Digital Signal Processing, Second Edition.

The first chapter goes over the history and where you can find use for DSP work. The book writes some psuedo-code in a "BASIC" like language. I will be attempting to move it over to Python. Matlab would be better but I don't have it.

2 Statistics, Probability and Noise

2.1 Signals and Graph Terminology

2.1.1 Definitions

- **Signal** is how one parameter is related to another parameter
- **Continuous Signal** is if BOTH parameters can assume a continuous range
- **Discrete Signal** is if BOTH parameters are quantized in some manner
- **Time Domain** is if the X axis (the independent variable) is time
- **Frequency Domain** is if the X axis (the independent variable) is frequency

2.1.2 Concepts

- The two parameters of a signal are not interchangeable
- The parameter on the Y axis is a function of the one on the X axis
- Mathematicians tend to do 1-N, everyone else does 0-(N-1)

2.2 Mean and Standard Deviation

2.2.1 Mean

- **Mean** μ is the average of the signal. Add all samples together and divide by N. In electronics this is the DC (direct current) value.

$$\mu = \frac{1}{N}\sum_{i=1}^{N-1} x_i$$

```
1  import random
2  samples = random.sample(range(1, 101), 20)
3  def Mean(data=None):
4      """
5      Calculate the mean of a list of values.
6      """
7      if data is None:
8          return
9      mean = 0
10     for x in data:
```

```
        mean = mean + x
        mean = mean/len(data)
    return mean

samples

Mean(samples)
```

```
Python 3.5.2 |Continuum Analytics, Inc.| (default, Jul  2 2016, 17:53:06)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-1)] on linux
Type "help", "copyright", "credits" or "license" for more information.
python.el: native completion setup loaded
[24, 32, 11, 68, 46, 72, 100, 5, 2, 54, 27, 19, 75, 30, 4, 78, 74, 23, 14, 38]
1.9383619639071976
```

2.2.2 Standard Deviation and Variance

- **Average Deviation** is not commonly used. Sums up all the deviations, from the mean, for each sample and divided by the number of samples. Use absolute values for deviation otherwise differences could cancel out.

- **Standard Deviation** averages the power. This is the AC portion of the signal.

$$\sigma = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N-1}(x_i - \mu)^2}$$

```
import math
def StandardDeviation(samples):
    """
    Calculate the standard deviation of a list of values.
    Uses the Mean method from previous examples
    """
    mean = Mean(samples)
    std = 0.0
    for x in samples:
        std = std + math.pow((x - mean), 2)
        std = std / (len(samples) - 1)
        std = math.sqrt(std)
    return std

samples

StandardDeviation(samples)
```

```
[24, 32, 11, 68, 46, 72, 100, 5, 2, 54, 27, 19, 75, 30, 4, 78, 74, 23, 14, 38]
8.282051868140895
```

- **Variance** is commonly used in statistics. Notice variance and standard deviation both divide by N-1, not N!

$$\sigma^2 = \frac{1}{N-1}\sum_{i=1}^{N-1}(x_i - \mu)^2$$

```
def Variance(samples):
    """
    Calculate the variance of a list of values.
    """
    return math.pow(StandardDeviation(samples), 2)

samples

Variance(samples)
```

```
[24, 32, 11, 68, 46, 72, 100, 5, 2, 54, 27, 19, 75, 30, 4, 78, 74, 23, 14, 38]
68.5923831465761
```

- **Root Mean Square (rms)** measures both the AC and DC components.

$$x_{rms} = \sqrt{\frac{1}{N}\sum_{i=0}^{N-1}(x_i)^2}$$

```
def RootMeanSquare(samples):
    """
    Calculate the Root Mean Square of an input list
    """
    rms = 0
    N = len(samples)
    for x in range(N-1):
        square = samples[x] * samples[x]
    divide = square/N
    rms = math.sqrt(divide)
    return rms

samples

RootMeanSquare(samples)
```

```
[24, 32, 11, 68, 46, 72, 100, 5, 2, 54, 27, 19, 75, 30, 4, 78, 74, 23, 14, 38]
3.1304951684997055
```

2.2.3 Running Statistics

- **Running Statistics** is often needed. In this situation we want to recompute mean and standard deviation of new signal added in without redoing all of the calculations

$$\sigma^2 = \frac{1}{N-1}(\sum_{i=0}^{N-1}(x_i)^2 - \frac{1}{N}(\sum_{i=0}^{N-1}x_i)^2)$$

```
def RunningStatistics(samples):
    """
    Calculate the mean, variance and std while running through a list of
    values. The self.samples list should be set when instantiating
    this instance.
    """
    mean = 0
    variance = 0
    std = 0
    temp_sum = 0
    sum_squares = 0
    N = len(samples)
    for x in samples:
        temp_sum = temp_sum + x
        sum_squares = sum_squares + math.pow(x, 2)
        mean = temp_sum/N
        variance = (sum_squares - (math.pow(temp_sum, 2)/N)) / (N - 1)
        std = math.sqrt(variance)
    return mean, variance, std

samples

RunningStatistics(samples)
```

```
[24, 32, 11, 68, 46, 72, 100, 5, 2, 54, 27, 19, 75, 30, 4, 78, 74, 23, 14, 38]
(39.8, 856.2736842105263, 29.262154469733193)
```

- In some situations mean decribes what is being measured and standard deviation measures noise
- **Signal to Noise Ration (SNR)** is a comparison of mean to standard deviation

$$SNR = \frac{\mu}{\sigma}$$

```
def SNR(samples):
    """
```

```
    Calculate the Signal to Noise Ratio
    """
    SNR = Mean(samples)/StandardDeviation(samples)
    return SNR

samples

SNR(samples)
```

```
[24, 32, 11, 68, 46, 72, 100, 5, 2, 54, 27, 19, 75, 30, 4, 78, 74, 23, 14, 38]
0.23404368805798234
```

- **Coefficient of Variance (CV)** is the standard deviation divided by the mean and multiplied by 100%.

$$CV = \frac{\sigma}{\mu} * 100\%$$

```
def CV(samples):
    """
    Calculate the Signal to Coefficient of Variation
    """
    CV = (StandardDeviation(samples)/Mean(samples)) * 100
    return CV

samples

CV(samples)
```

```
[24, 32, 11, 68, 46, 72, 100, 5, 2, 54, 27, 19, 75, 30, 4, 78, 74, 23, 14, 38]
427.27065544799416
```

- High SNR and Low CV is a good signal!

2.3 Signal vs. Underlying Process

- **Statistics** is the science of interpreting numerical data
- **Probability** is used in DSP to understand the process that generated the signals
- **Statistical Variation or Fluctuation or Noise** is random irregularity found in actual data
- **Typical Error** is the standard deviation over the square root of the number of samples. For small N, expect a large error. As N grows larger the error should be shrinking.

$$TypicalError = \frac{\sigma}{N^{\frac{1}{2}}}$$

- **Strong Law of Large Numbers** guarantees that the error becomes zero as N approaches infinity.
- The Standard Deviation equation measures the value of the underlying process, not the actual signal. Divide through by N to get the value of the signal.
- **Non Stationary** processes that change their underlying behavior. This causes a slowly changing mean and standard deviation.

2.4 The Histogram, PMF and PDF

- **Histogram** displays the number of samples there are in the signal at this value or range of values.
  - This is formed from the acquired signal and works on a finite number of samples.
  - Only operates on discrete data
  - If the number of levels a signal can take are larger than the number of samples, this is a problem! This happens all the time if you use floating point numbers.

```
import numpy as np
import matplotlib
matplotlib.use('Agg')
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt

mu, sigma = 100, 15
x = mu + sigma * np.random.randn(10000)

# the histogram of the data
n, bins, patches = plt.hist(x, 50, normed=1, facecolor='green', alpha=0.75)

# add a 'best fit' line
y = mlab.normpdf(bins, mu, sigma)
l = plt.plot(bins, y, 'r--', linewidth=1)

foo = plt.xlabel('Smarts')
foo = plt.ylabel('Probability')
foo = plt.title(r'$\mathrm{Histogram\ of\ IQ:}\ \mu=100,\ \sigma=15$')
foo = plt.axis([40, 160, 0, 0.03])
foo = plt.grid(True)
foo = plt.savefig('../Notes/histogram.png', bbox_inches='tight')
```

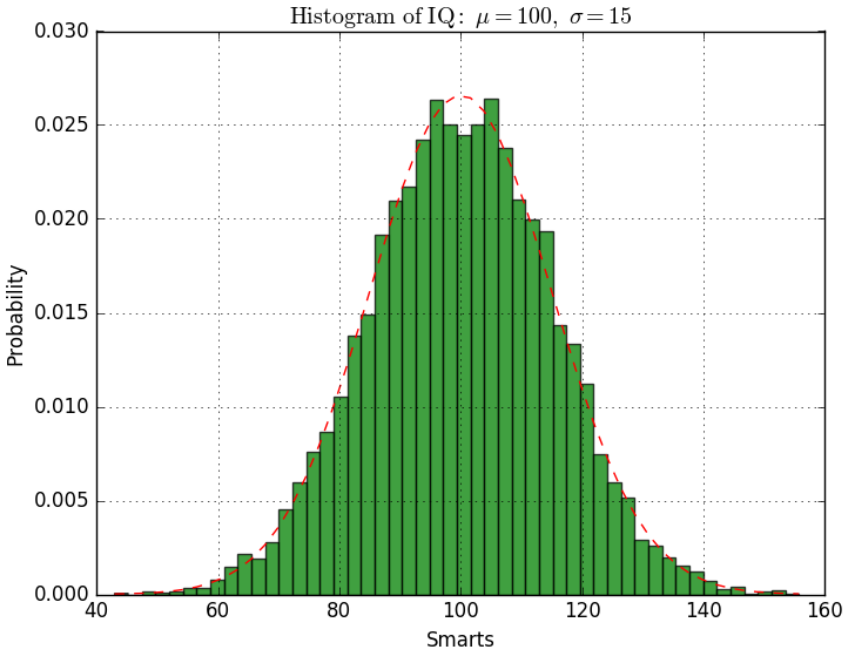

Figure 1: Histogram graph example

Code and example lifted from Matplotlib Examples

- **Probability Mass Function (PMF)** is the corresponding curve for the underlying process.
  - This is what the histogram would be with infinite samples. This means the pmf must be between 0 and 1 with the sum of all values to be equal to 1.
  - This describes the *probability* that a certain value will be generated.
  - Only operates on discrete data
- **Probability Density Functions (PDF)** is to continuous signals what the probability mass function is to discrete signals.
  - The vertical axis is in units of **probability density**
  - Probability is calculated by multiplying the probability density by the range of values
  - If the PDF is not constant over a range, the multiplication becomes an integral.
- **Binning** is done by creating arbitrary ranges for the values to be counted in. These are the bins. The value of the bin is the number of samples in that range.
  - Too many bins makes it difficult to estimate the amplitude of the PMF
  - Too few bins makes it difficult to estimate the PMF in the horizontal direction
  - The number of bins is a trade off of the resolution in X or Y directions.

2.5 The Normal Distribution

- **Normal Distribution or Gaussian Distribution** is the bell shaped PDF formed from random signals.
  - The basic shape of the curve is generated from the *negative squared exponent*

- You can convert the raw curve with the mean (μ) and standard deviation (σ).
- You also need to normalize the equation so that the are under the curve is equal to 1.
- The mean (μ) centers the curve over a particular value.
- The standard deviation (σ) controls the width of the bell shape
- The sharp drop indicates that the extremes are very rare

$$y(x) = e^{-x^2}$$

$$P(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(x-\mu)^{\frac{2}{2\sigma^2}}}$$

- **Cumulative Distribution Function (CDF)** is the integral of the PDF. The Gaussian can not be easily integrated. Numerical methods are required. The symbol Φ(x) is used for this.

2.6 Digital Noise Generation

- **Randonm Number Generation** is the heart of digital noise generator.
- **Central Limit Theorem** the sum of random numbers become normally distributed as more and more random samples are added together.
  - The resulting PDF becomes Gaussian.
  - X is a normally distributed random number. Requires 2 random numers R_1 and R_2

$$X = (-2logR_1)^{\frac{1}{2}} cos(2\pi R_2)$$

- **Seed** is the number that the random number generator starts with. The following equation generates the randomd number based off of the seed. This allows us to replay a sequence of random numbers.

$$R = (aS + b) moduloc$$

- **Psuedo-random** numbers generated this way are not absolutely random since they are based on the previous value.

2.7 Precision and Accuracy

- **Accuracy** is hitting your goal
  - difference between mean and the actual value
  - Measure of calibration
  - If calibration corrects the error it was an accuracy issue
- **Precision** is performing the same way repeatedly.
  - Poor precision results from random errors.
  - Width of histogram
  - Measure of random noise
  - Poor accuracy results from systemic errors
  - If averaging successive reading provide a better measurement then it was a precision error

3 ADC and DAC

3.1 Quantization

- Analog to digital conversion is usually done in 2 steps.
  - **Sample and Hold (S/H)** keeps the voltage constant for the actual ADC conversion. This value is only allowed to change at periodic intervals. Changes to the input inbetween these periods is ignored!
  - **Analog to Digital Conversion (ADC)** converts analog level to digital value
- **Sampling** converts the independent variable (time) from continuous to discrete
- **Quantization** converts the dependent variable (voltage or other input signal) from continuous to discrete. Small changes in between samples can be lost. Errors here appear like random noise. In most cases, quantization is adding in a specific amount of random noise to the signal. Random noise signals are combined by adding their *variances*. The number of bits determines the precision of the data. This error *should* be small. Small changes in amplitude can be lost.
- It is important to keep sampling and quantization separate and analyze them that way since they degrade the signal in different ways
- Any digitized sample can have a maximum error of $\pm\frac{1}{2}$ **LSB (Least Significant Bit)** This is the distance between 2 adjacent quantization levels.
- When figuring out how many bits are needed ask 2 questions
  - How much noise is already present in the signal?
  - How much noise can be tolerated in the signal?
- If the signal is very slow moving, the error can not be treated as random since the value presented to the ADC will be constant for long periods. It makes the output seem stuck.
- **Dithering** common technique for improving the digitization of slow moving signals. Even if the signal is moving by less than $\pm\frac{1}{2}$ LSB

the added noise causes the output to randomly toggle between 2 adjacent levels. Even though a single measurement has the inherent limit of $\pm\frac{1}{2}$ LSB the statistics of large number of samples can do better. Adding noise can provide more information. These circuits can be quite elaborate.

3.2 The Sampling Theorem

- **Proper Sampling** is when you can reconstruct the analog signal from the samples exactly. The samples need to be a unique representation of the analog signal. Too low of a sampling rate can lead to improper sampling.
- **Aliasing** happens when sinusoids change frequency when sampling. This makes an unambiguous recreation of the signal impossible. This is improper sampling. Aliasing can loose information about both the high and low frequencies. It can also change the phase of the signal. Can either phase shift 0 or 180 degrees.
- **Sampling Theorem** *Shannon* or *Nyquist* sampling states that a continuous signal can be properly sampled only if it does not contain frequency components above one half of the sampling rate. *EX* a sampling rate of 2000 samples/second requires the analog signal to be made of signals less than 1000 samples/second
- **Nyquist Sampling** and **Nyquist rate** are not standardized terms.
- When the continuous signal is above one half of the sampling rate *aliasing* changes the frequency into something that *can* be represented by the data.
- **Impulse Train** each impulse is infinitely narrow and the value between each one is 0. This is a theoretical concept only. It can be treated as an continuous signal and has identical content to the digital signal.
- Every waveform can be viewed as being composed of sinusoids of varying frequency and amplitudes.
- **Upper Sideband** is a copy of the multiple of the original frequency spectrum
- **Lower Sideband** is a left or right flipperd copy of the original frequency spectrum.

3.3 Digital to Analog Conversion

- **Zeroth Order Hold** - Nearly all DACs operate by holding the last value until another sample is received. This is the DAC equivalent of Sample and Hold.
- **Sinc Function** In the frequency domain the zeroth order hold results in the impulse train being multiplied by the following.

$$H(f) = |\frac{sin(\frac{\pi f}{f_s})}{\frac{\pi f}{f}}|$$

- The zeroth order hold can be understood as the convolution of the impulse train with the rectangular pulse with a width equal to the sampling period.
- Analog filter that does this needs to remove all frequency components above one half of the sampling rate and boost the frequencies of the reciprocal of the zeroth order hold's effect $\frac{1}{sinc(x)}$
- Analog signals suffer from the same limitations (noise and bandwidth) as digital signals

3.4 Analog Filters for Data Conversion

Figure 2: Analog Filters

3.5 Selecting the Anti-Alias Filter

- **Antialiase Filter** This is an analog low pass filter designed to remove all frequencies above the nyquist frequency.
- **Reconstruction Filter** This is the output filter for a DAC and may include zeroth order hold filter and frequency boost.
- Filter complexity can be adjusted by number of **poles** and **zeros**. More poles is more complicated electronics but better performance.

- Three types of analog filters are commonly used. Each is designed to optimze a different design parameter.
  - **Chebyshev**
  - **Butterworth**
  - **Bessel or Thompson**

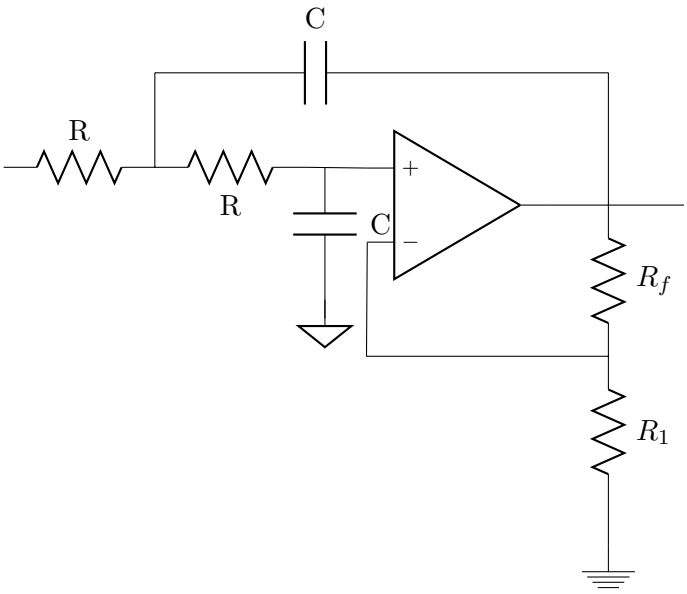

Figure 3: Modified Sallen-Key Circuit

3.6 Multirate Data Conversion

3.7 Single Bit Data Conversion

4 DSP Software

4.1 Computer Numbers

4.2 Fixed Point

4.3 Floating Point

4.4 Number Precision

4.5 Execution Speed: Programming Language

4.6 Execution Speed: Hardware

4.7 Execution Speed: Programming Tips