DSP For Scientists and Engineers Notes

Philip Tracton

May 12, 2017

Contents

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.

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

```python
import random
samples = random.sample(range(1, 101), 20)
def Mean(data=None):
    """
    Calculate the mean of a list of values.
    """
    if data is None:
        return
    mean = 0
    for x in data:
```

```
11          mean = mean + x
12          mean = mean/len(data)
13      return mean
14
15  samples
16
17  Mean(samples)
```

```
Python 3.4.5 |Anaconda custom (64-bit)| (default, Jul  2 2016, 17:47:47)
[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
[42, 47, 20, 86, 65, 45, 95, 25, 30, 54, 21, 69, 50, 2, 34, 73, 100, 15, 48, 57]
2.9725233474027126
```

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.

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

```
1   import math
2   def StandardDeviation(samples):
3       """
4       Calculate the standard deviation of a list of values.
5       Uses the Mean method from previous examples
6       """
7       mean = Mean(samples)
8       std = 0.0
9       for x in samples:
10          std = std + math.pow((x - mean), 2)
11          std = std / (len(samples) - 1)
12          std = math.sqrt(std)
13      return std
14
15  samples
16
17  StandardDeviation(samples)
```

[42, 47, 20, 86, 65, 45, 95, 25, 30, 54, 21, 69, 50, 2, 34, 73, 100, 15, 48, 57]
12.416681888991043

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

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

```
1   def Variance(samples):
2       """
3       Calculate the variance of a list of values.
4       """
5       return math.pow(StandardDeviation(samples), 2)
6
7   samples
8
9   Variance(samples)
```

[42, 47, 20, 86, 65, 45, 95, 25, 30, 54, 21, 69, 50, 2, 34, 73, 100, 15, 48, 57]
154.1739891323982

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

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

```
1   def RootMeanSquare(samples):
2       """
3       Calculate the Root Mean Square of an input list
4       """
5       rms = 0
6       N = len(samples)
7       for x in range(N-1):
8           square = samples[x] * samples[x]
9       divide = square/N
10      rms = math.sqrt(divide)
11      return rms
12
13  samples
14
15  RootMeanSquare(samples)
```

[42, 47, 20, 86, 65, 45, 95, 25, 30, 54, 21, 69, 50, 2, 34, 73, 100, 15, 48, 57]
10.73312629199899

2.2.3 Running Statistics

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

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

```python
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)
```

[42, 47, 20, 86, 65, 45, 95, 25, 30, 54, 21, 69, 50, 2, 34, 73, 100, 15, 48, 57]
(48.9, 717.3578947368422, 26.78353775618229)

- 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

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

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

samples

SNR(samples)
```

[42, 47, 20, 86, 65, 45, 95, 25, 30, 54, 21, 69, 50, 2, 34, 73, 100, 15, 48, 57]
0.23939755999050197

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

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

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

samples

CV(samples)
```

[42, 47, 20, 86, 65, 45, 95, 25, 30, 54, 21, 69, 50, 2, 34, 73, 100, 15, 48, 57]
417.7152014580577

- 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.

```python
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')
```

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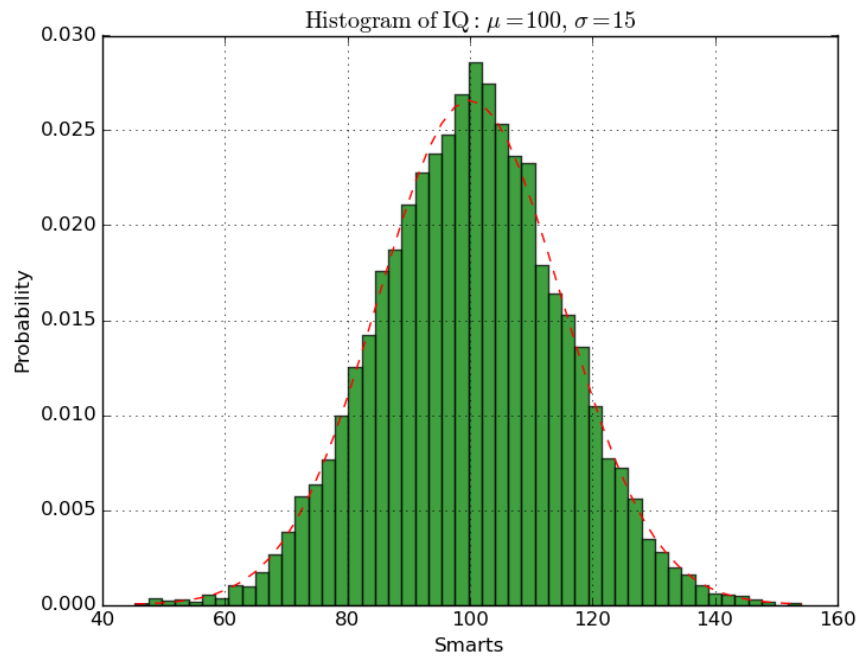

Figure 1:   Histogram graph example

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 gener-
    ated.

  – Only operates on discrete data

10

- **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 $\Phi(x)$ 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 Definitions

3.2 Concepts