DSP For Scientists and Engineers Notes

Philip Tracton

April 29, 2017

Contents

List of Listings

List of Tables

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), 100)
def Mean(data=None):
    """
    Calculate the mean of a list of values.
    """
    if data is None:
        return
    mean = 0
    for x in data:
```

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

samples

Mean(samples)
```

```
[10, 40, 6, 75, 3, 4, 91, 100, 28, 37, 85, 57, 13, 2, 31, 67, 48, 11, 51, 8, 22, 33,
0.9572176842143284
```

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.

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

```python
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)
```

```
[10, 40, 6, 75, 3, 4, 91, 100, 28, 37, 85, 57, 13, 2, 31, 67, 48, 11, 51, 8, 22, 33,
9.45547040062438
```

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

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

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

samples

Variance(samples)
```

```
[10, 40, 6, 75, 3, 4, 91, 100, 28, 37, 85, 57, 13, 2, 31, 67, 48, 11, 51, 8, 22, 33,
89.40592049708377
```

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

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

```python
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)
```

```
[10, 40, 6, 75, 3, 4, 91, 100, 28, 37, 85, 57, 13, 2, 31, 67, 48, 11, 51, 8, 22, 33,
7.2
```

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

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

```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)
```

```
[10, 40, 6, 75, 3, 4, 91, 100, 28, 37, 85, 57, 13, 2, 31, 67, 48, 11, 51, 8, 22, 33,
(50.5, 841.6666666666666, 29.011491975882016)
```

- 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)
```

[10, 40, 6, 75, 3, 4, 91, 100, 28, 37, 85, 57, 13, 2, 31, 67, 48, 11, 51, 8, 22, 33,
0.10123427430444071

- **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)
```

[10, 40, 6, 75, 3, 4, 91, 100, 28, 37, 85, 57, 13, 2, 31, 67, 48, 11, 51, 8, 22, 33,
987.8077428526934

- 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.

-

2.5 The Normal Distribution

2.6 Digital Noise Generation

2.7 Precision and Accuracy

3 ADC and DAC

3.1 Definitions

3.2 Concepts

4 Appendix A – Code Listing

4.1 DSP Source Code

"../src/DSP.py"

4.2 DSP Test Code

"../src/DSP$_Test.py$"