DSP For Scientists and Engineers Notes

Philip Tracton

April 28, 2017

Contents

List of Figures

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
def Mean(self):
    """
    Calculate the mean of a list of values.
    The self.samples list should be set when instantiating
    this instance.
    """
    mean = 0
    for x in self.samples:
        mean = mean + x
    mean = mean/len(self.samples)
    return mean
```

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.

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

```
def StandardDeviation(self):
    """
    Calculate the standard deviation of a list of values.
    The self.samples list should be set when instantiating
    this instance.
    """
    mean = self.Mean()
    std = 0.0
    for x in self.samples:
        std = std + math.pow((x - mean), 2)
    std = std / (len(self.samples) - 1)
    std = math.sqrt(std)
    return std
```

- **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(self):
    """
    Calculate the variance of a list of values.
    The self.samples list should be set when instantiating
    this instance.
    """
    return math.pow(self.StandardDeviation(), 2)
```

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

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

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(self):
    """
    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(self.samples)
    for x in self.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)
        print("RunningStatistics: Mean {} Variance {} STD {}".format(
            mean, variance, std))
    return mean, variance, std
```

- 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

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

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

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

- 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.

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

```python
    def TypicalError(self):
        """
        Calculate the Typical Error based on the already stored
        self.samples and the StandardDeviation
        """
        error = self.StandardDeviation/math.pow(len(self.samples), 0.5)
        return error
```

- **Strong Law of Large Numbers** guarantees that the error becomes zero as N approaches infinity.

2.4 The Histogram, PMF and PDF

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

```python
#! /usr/bin/env python3

import math


class DSP():
    """
    DSP Demonstration class that implements the code
    from DSP For Scientists and Engineers, Second Edition
    """

    def __init__(self, samples=None):
        """
        """
        self.samples = samples
        return

    def Mean(self):
        """
        Calculate the mean of a list of values.
        The self.samples list should be set when instantiating
        this instance.
        """
        mean = 0
        for x in self.samples:
            mean = mean + x
        mean = mean/len(self.samples)
        return mean

    def StandardDeviation(self):
        """
        Calculate the standard deviation of a list of values.
        The self.samples list should be set when instantiating
        this instance.
        """
        mean = self.Mean()
        std = 0.0
        for x in self.samples:
            std = std + math.pow((x - mean), 2)
        std = std / (len(self.samples) - 1)
        std = math.sqrt(std)
        return std

    def Variance(self):
        """
        Calculate the variance of a list of values.
        The self.samples list should be set when instantiating
        this instance.
        """
        return math.pow(self.StandardDeviation(), 2)

    def RunningStatistics(self):
```

```python
        """
        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(self.samples)
        for x in self.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)
            print("RunningStatistics: Mean {} Variance {} STD {}".format(
                mean, variance, std))
        return mean, variance, std

    def SNR(self):
        """
        Calculate the Signal to Noise Ratio based on the
        already stored self.samples list
        """
        SNR = self.Mean()/self.StandardDeviation()
        return SNR

    def CV(self):
        """
        Calculate the Signal to Coefficient of Variation
        based on the already stored self.samples list
        """
        CV = (self.StandardDeviation()/self.Mean()) * 100
        return CV

    def TypicalError(self):
        """
        Calculate the Typical Error based on the already stored
        self.samples and the StandardDeviation
        """
        error = self.StandardDeviation/math.pow(len(self.samples), 0.5)
        return error
```

4.2 DSP Test Code

```python
#! /usr/bin/env python3

"""
Check Results:
http://www.calculator.net/standard-deviation-calculator.html?numberinputs=0%2C+1%2C+2%2C+3%2C+
"""


import DSP


if __name__ == "__main__":
    samples = [x for x in range(100)]
    dsp = DSP.DSP(samples)
    print(samples)
    print("Mean {} ".format(dsp.Mean()))
    print("StandardDeviation {} ".format(dsp.StandardDeviation()))
    print("Variance {} ".format(dsp.Variance()))
    mean, variance, std = dsp.RunningStatistics()
    print("SNR: {}".format(dsp.SNR()))
    print("CV: {}".format(dsp.CV()))
```