McGill Physical Journal

Introduction to Programming

Maclean Rouble (260423190), Michael Caouette-Mansour (260581053), Leo Thomas (260574268)

McGill University Department of Physics

January 16, 2017

Abstract

This report presents a discussion of various aspects of data processing and representation in an experimental setting, using Python programming language. Topic covered include distribution analysis with Histogram representation and associated mean and variance calculations, error propagation through operations including derivation, integration and root-mean-squared, as well as a brief look at frequency-filtering.

1 Introduction

Steadily increasing advancements in processing and computational capabilities of current computers has allowed for automated data generation, collection, and analysis, on a much large scale than previously possible. Automated data analytic techniques have an impact that reaches far beyond the physics laboratory with companies such as Facebook, Netflix, and Amazon using these techniques on massive scales to create individually-tailored product recommendations, advertising placement, network suggestions.

2 Theory

The first part of this report is concerned with error propagation though a series of operations on different signals. For simple operations like addition, multiplication, or any function of n variables $Y = f(X_1, X_2, ..., X_n)$, the function is simply applied to each array element. The propagation of the error σ_Y is computed using the following formula (which is based on the continuous approximation, which assumes that neighboring elements in the array are close):

$$\sigma_Y^2 = \sum_{i=1}^{n} \left(\frac{\partial f}{\partial X_i}\right)^2 \sigma_i^2 \tag{1}$$

The method used for computing the derivative of an array Y with respect to X (also in array form) is based on the limit definition of the derivative. For each pair of points $(X_i, Y_i), (X_{i+1}, Y_{i+1})$, the derivative is computed for a point in the middle using the approximation that the two points are linearly related. The derivative is then:

$$\frac{\delta Y}{\delta X} \approx \frac{Y_{i+1} - Y_i}{X_{i+1} - X_i} \tag{2}$$

The error on the derivative is computed by plugging Eq. 2 in Eq. 1, where $f = \frac{\delta Y}{\delta X}$. Since there is no error in the abscissa used, this gives:

$$\sigma_{\frac{dY}{dX}}^2 = \left(\frac{1}{X_{i+1} - X_i}\right)^2 \left(\sigma_{Y_{i+1}}^2 + \sigma_{Y_i}^2\right)^2 \tag{3}$$

The definition of an integral by a trapezoidal Riemann sum is used to compute the integral of the array. The i^{th} element of the array of the integral F is computed using:

$$F_i = \int_{X_1}^{X_i} Y dX \approx \sum_{j=1}^{i} (X_{j+1} - X_j) \frac{Y_j + Y_{j-1}}{2} \tag{4}$$

The error on the integral is computed by plugging Eq. 4 into Eq. 1, with, again, there was no errors in the absissa. This gives:

$$\sigma_{F_i}^2 = \frac{1}{i^2} \sum_{k=1}^{i} (X_k - X_{k-1})^2 \frac{\sigma_{Y_k}^2 + \sigma_{Y_{k-1}}^2}{2} \tag{5}$$

3 Data

3.1 Square of Signal A

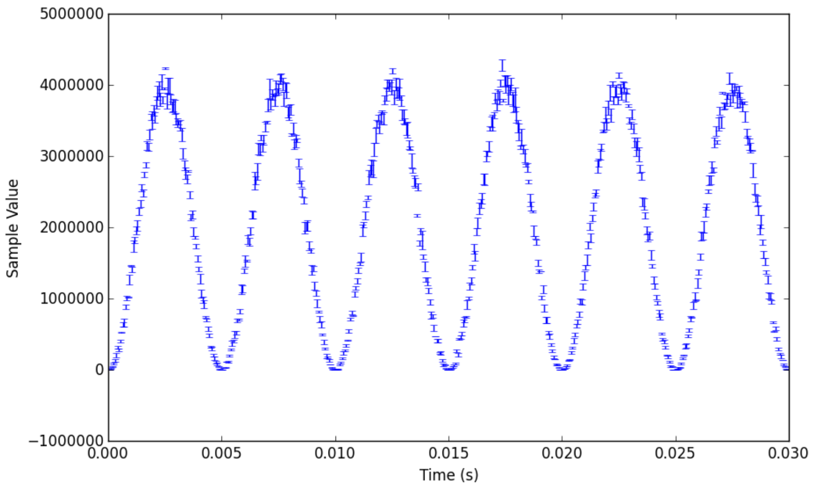

Figure 1: The data of signal A having been processed by the multiply function in order to square its values. Note this these values remain unitless since A is unitless

3.2 Derivative of Signal A

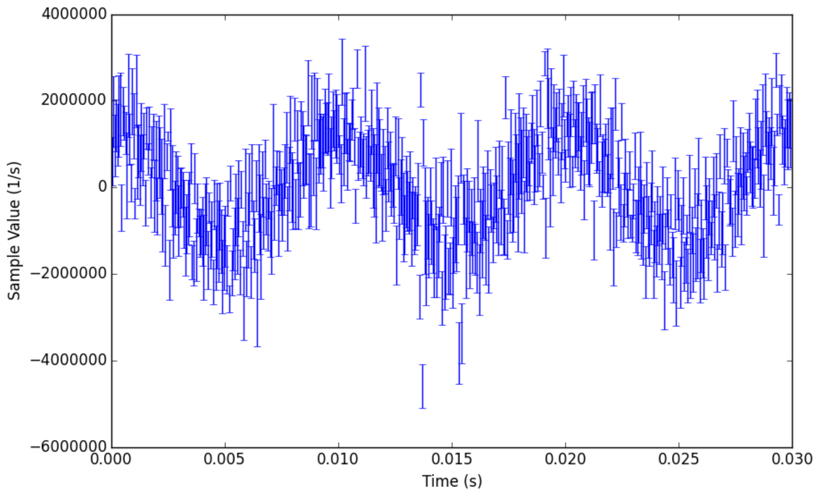

Figure 2: The data of signal A having been processed by the derivative function.

Upon looking at the data generated by the derivative function, it is obvious that while the data follows a definite trend (resembling a cosine function, as is expected for the derivative of a sine function), it is not always in very close agreement. This scattering is due to the nature of the original data itself, which has points that do not always follow a smooth sine function but instead have small irregularities. These irregularities are made particularly visible in the depiction of the derivative, due to the method of computing the derivative (and the definition of the derivative itself), which compares the increase in the quantity to the increase in time. As these data points occur at high quantities with very little time between them, the slopes of these lines (the derivatives between each set of points) tend to be very steep and varies sharply from one interval to the next.

However, it is also possible to use an averaging function to interpolate a smooth function from this noisy scatter, which one would expect to match the theoretically-predicted cosine function. This then raises the question of which representation of the data is more useful. Presumably, each has uses to which it is most suited.

3.3 Integral of A

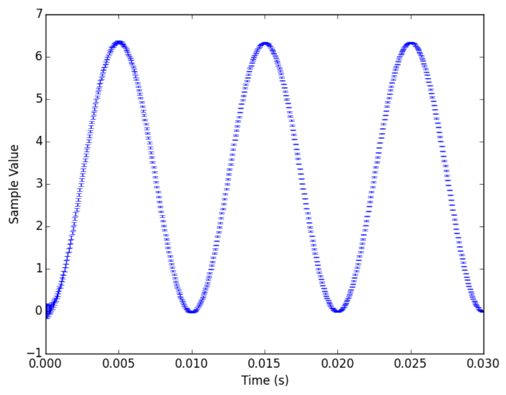

Figure 3: Signal a processed by the integral function. Note that signal A is a valueless quantity, therefore its integral remains in units of time

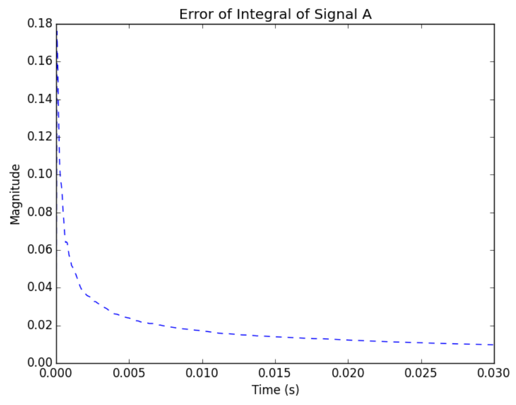

Figure 4: Magnitude of error bars on the integral of signal a. Unit-less quantity as signal A is unit-less as well.

Note that while the indefinite integral of a sine function is a negative cosine function, represented above is the definite integral from time 0 to time t, Which oscillates between a maximal value at half of the original sine function's period, and 0 after a full period. Note also that we have chosen to represent the relative error, as opposed to the total error. This is due to the nature of the integral, which involves an infinite sum. Since the total absolute error is a linear combination of all elements in the sum, this error tends to infinity. However, the error relative to the number of points used to evaluate this integral decreases exponentially. This is logical, since as we decrease the size of the interval of approximation, one expects the error to decrease, since the approximation becomes more precise as there are more data points within the same interval.

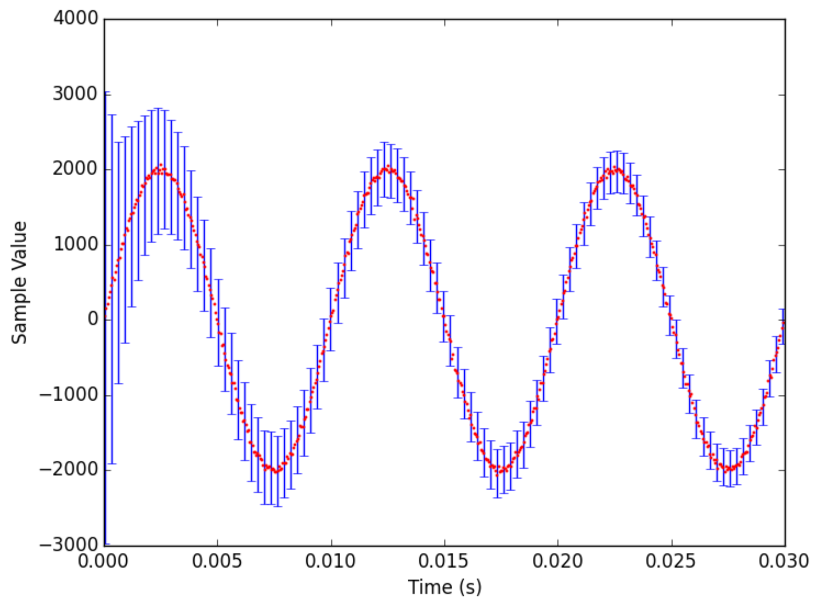

Figure 5: The derivative of the integral of A (blue), overlayed with A (red). Note that the integral has units of time, and the derivative has units of (1/t), therefore the derivative of the integral is unit-less, as is the original signal. Since the derivative function has more or less constant error magnitude and the integral function has errors that decrease exponentially, the errors on the derivative of the integral also decrease exponentially.

3.4 RMS of D

Figure 6 shows the Signal D with its RMS. Figure 7 shows the error on the corresponding RMS curve as the time domain increases. The RMS values approach a fixed point as the time domain of integration increases. This is to be expected, because the RMS is a type of averaging function. As the domain of averaging increase, so does the precision and the less the values oscillate around this fixed point.

Theoretically, this fixed value is $\frac{1}{\sqrt{2}}$ multiplied by the amplitude of the function being processed (in this case, a sine wave). Since the average amplitude of Signal D is 518 ± 30, we can compare this (multiplied by $\frac{1}{\sqrt{2}}$) to the RMS curve to determine how accurate it is. Figure 8 shows that the computed RMS values approach the expected value quite accurately. The large initial oscillation is due to the domain of integration for RMS not yet having completed a full cycle.

The decrease in the error is to be expected, because the more cycles that are completed in the domain of integration, the better the averaging becomes.

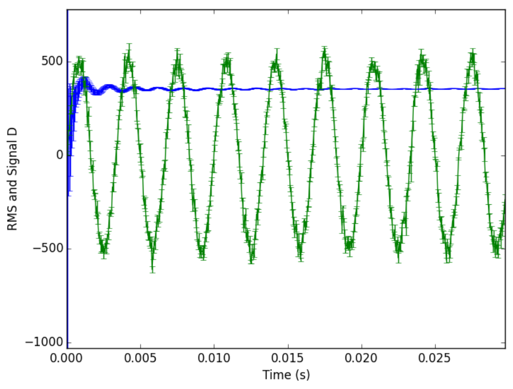

Figure 6: Signal D (green) and its RMS values (blue). At each point the RMS is computed from time zero to the corresponding time.

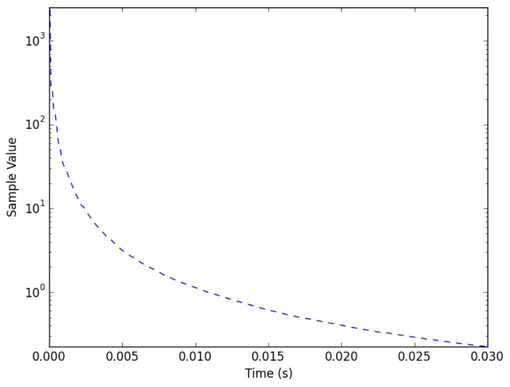

Figure 7: Error on the RMS of figure 6, plotted on a logarithmic scale to emphasis the exponentially decaying error magnitude as the domain of computation increases.

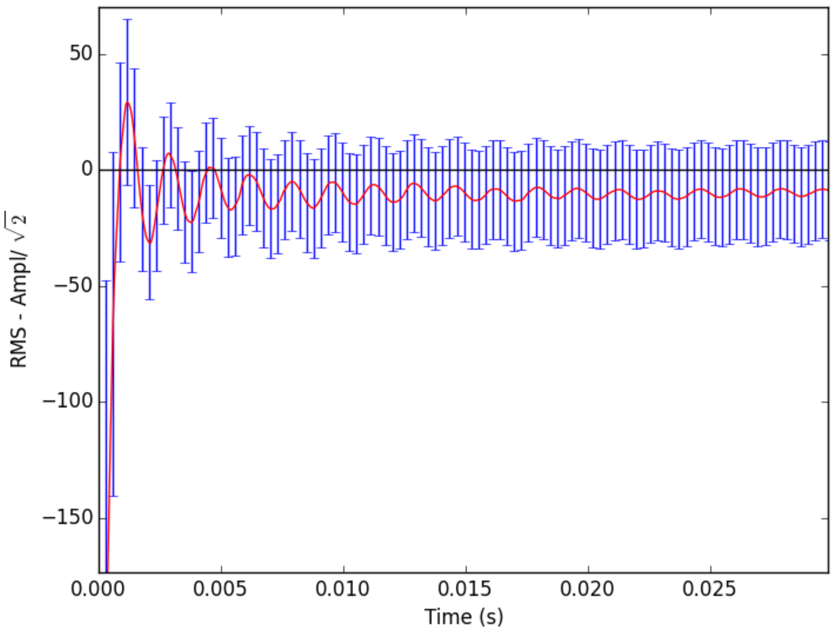

Figure 8: The difference between the computed RMS values and the amplitude of the original signal multiplied by $\frac{1}{\sqrt{2}}$.

Figure 6 shows the signal d with its RMS. Figure 7 shows the error on the corresponding

RMS curve as the time domain increase. The RMS approach a fixed value as the time domain of integration increases. This is to be expected, because the RMS is a kind of average function. As the domain of averaging increase, so is for the precision and less are the oscillator around the fixed value. This fixed value is suppose to be $\frac{1}{\sqrt{2}}$ time the amplitude of a sine wave. This is what Figure 6 shows. The large initial oscillation is due to the fact that the domain of integration for RMS does not complete a full cycle at this time.

The decrease in the error of propagation is to be expected, because the more cycles are in the domain of integration, the better the averaging method is.

3.5 Low Pass Filtering of Signal C

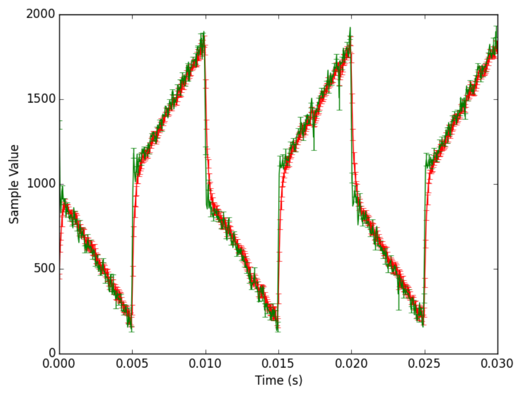

Figure 9: Signal C processed by the given low pass filter function, with cutoff frequency 55Hz.

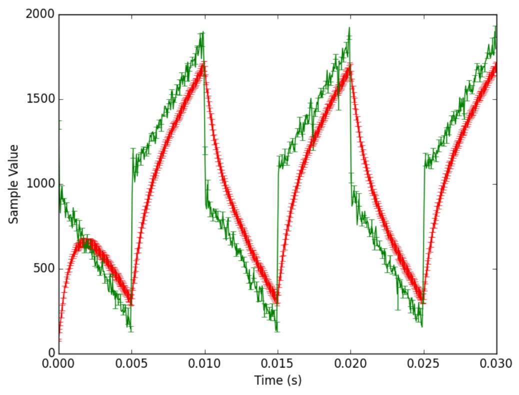

Figure 10: Signal C processed by the given low pass filter function, with cutoff frequency 10Hz.

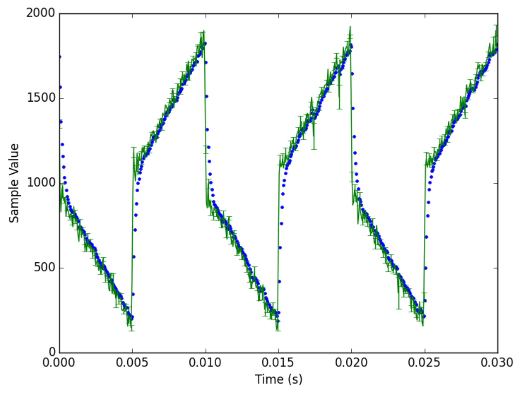

Figure 11: Signal C processed by the given fast Fourier transform low pass filter function, with cutoff frequency 55Hz.

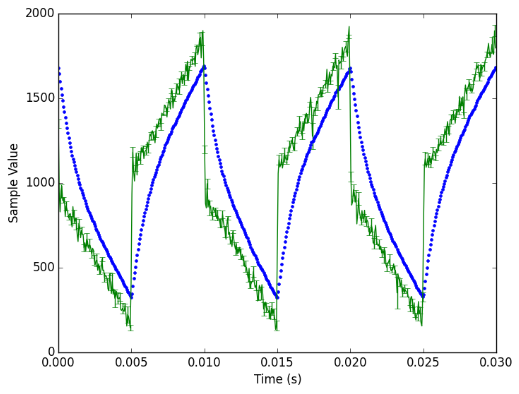

Figure 12: Signal C processed by the given fast Fourier transform low pass filter function, with cutoff frequency 10Hz.

Applying a low-pass filter to the signal has the effect of diminishing all frequencies above the cutoff frequency (on the left, 55Hz, on the right, 10Hz). The filter utilizing a fast Fourier transform (FFT) has slightly smaller error associated with it, as visible in the images.

The waveforms also become more triangular as the cutoff frequency is decreased, due to the removal of the higher frequencies. This is, perhaps, unexpected at first glance, but is explained when considering the construction of waveforms by the addition of sines. For example, a sawtooth wave contains many small-amplitude high frequency sine waves added to larger-amplitude lower frequency sine waves, and the more extensive the range of these high frequency waves, the more perfect the overall shape of the wave. Take away these high frequencies, and what remains is overall much more sinusoidal.

3.6 Frequency Analysis of the Warmer/Colder Game Solver

Table 1: Mean number of guesses and associated variance per batch of 20 trials

Batch N^o	1	2	3	4	5	6	7	8	9	10
Mean	16.9	16.6	16.05	17	16.6	16.75	16.7	16.65	17.45	17.1
Variance	2.624881	4.164133	2.801339	2.121320	2.49780	3.031089	3.465545	2.743629	2.616773	2.406242

The above table shows a consistent mean and variance throughout each batch of 20 trials. If one then takes the variance of these means themselves, the resulting value is 0.1236. This is notably smaller than the mean of the individual variances per batch of data, meaning that while the individual trials may vary significantly, overall the data falls closely about a consistent mean.

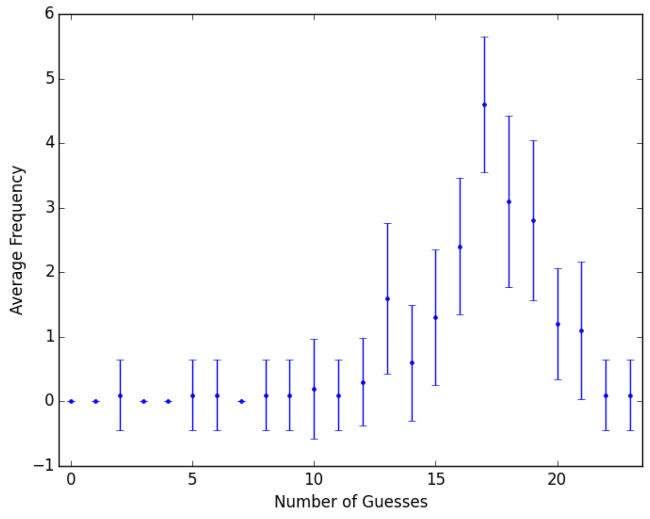

Figure 13: Mean frequency and associated standard deviation of each number of guesses within batches of 20 trials. The standard deviation is defined here as the square root of the variance.

4 Conclusions

As accessibility of huge data sets grows exponentially with current technological developments, understanding the effect of the acquisition and manipulation of this data on the precision of the analysis becomes a crucial part of any interpretation. A second important point to note is that not all error analyses are consequent, and techniques used to establish confidence intervals must be carefully chosen to ensure a meaningful analysis, as in the case of the use of the relative versus total error on the integral function.

References

A Raw Data for Warmer-Colder

Raw data generated by the warmer-colder game. Each batch (table row) represents a histogram of the number of guesses necessary for 20 different trials

Table 2: Raw Data

N^o of guesses	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
Batch N^o 1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	2	1	1	5	6	3	0	1	0	0
2	0	0	1	0	0	0	0	0	0	0	0	0	0	3	0	0	3	4	3	2	3	0	0	1
3	0	0	0	0	0	0	0	0	0	1	0	0	1	3	0	1	3	6	2	1	1	1	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	2	4	4	5	0	1	2	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	1	2	1	4	1	4	0	5	1	1	0	0
6	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	1	2	7	3	3	2	0	0	0
7	0	0	0	0	0	1	0	0	0	0	0	0	0	3	0	0	2	4	3	5	1	1	0	0
8	0	0	0	0	0	0	0	0	0	0	2	0	0	0	1	1	4	3	5	2	1	1	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	1	3	4	1	3	1	4	0	0
10	0	0	0	0	0	0	0	0	0	0	0	1	0	0	2	2	1	5	3	4	1	0	1	0

Table 3: Mean frequency of each number of guesses within batches of 20 trials

$N^o$of guesses	1	2	3	4	5	6	7	8	9	10	11	12
Mean frequency	0.0	0.0	0.1	0.0	0.0	0.1	0.1	0.0	0.1	0.1	0.2	0.1
Variance	0.0	0.0	0.3	0.0	0.0	0.3	0.3	0.	0.3	0.3	0.6	0.3

N^o of guesses	13	14	15	16	17	18	19	20	21	22	23	24
Mean Frequency	0.3	1.6	0.6	1.3	2.4	4.6	3.1	2.8	1.2	1.1	0.1	0.1
Variance	0.458256	1.3565	0.75	1.1	1.113553	1.113553	1.757840	1.536229	0.748331	1.135782	0.3	0.3

B Python code

B.1 Warmer-Colder

```
# helper method: returns midpoint of range
def get_midpoint(low, high):
    return (low + high)/ 2


# Algorithm works by testing boundary points, checking on what
# side of the midpoint the value lies and updating the upper/lower
# limit of the boundary accordingly.
def play_warmer_colder(connection):
    connection.send("PLAY WARMER_COLDER")
    resp = connection.getResp()
    if not "OK_PLAY" == resp[0:7]:
        print "Something is wrong, Arduino replied '%s'" % (resp)
        return False

    # Define limits
    hlim = 4095
    llim = 0

    # send first guess as the upper limit, the result of this
    # guess is always warmer, so we don't check response.
    guess = hlim
    connection.send("GUESS %d" % (guess))
    resp = connection.getResp()

    # next guess will be lower limit. If response is WARMER, we check
    # the bottom half the range, if not, we check the upper half
```

```
nextGuess = llim

while 1:
    # send current guess, this is updated at the end of the loop
    guess = nextGuess
    print "Guessing %d" % (guess)
    connection.send("GUESS %d" % (guess))
    resp = connection.getResp()

    if not "OK_GUESS" == resp[0:8]:
        print "Something is wrong, Arduino resplied '%s'" % (resp)
        return False

    if "CONGRATULATIONS" == resp[10:10+15]: return resp

    # either higher or lower limit will be updated to the midpoint
    midpoint = get_midpoint(llim, hlim)

    # If the previous guess was the upper limit, we check the
    # upper half, and if it was the lower limit, we check the
    # lower half
    if "WARMER" == resp[10:10+6]:

        if guess == llim:
            hlim = midpoint

        if guess == hlim:
            llim = midpoint

        nextGuess = midpoint
    # Do the oposite of the WARMER cases.
    if "COLDER" == resp[10:10+6]:

        if guess == llim:
            # current upper limit has already been tested,
            # so we can eliminate it and update the boundary
            # to one less than this value
            hlim=hlim-1
            nextGuess = hlim

        if guess == hlim:
            llim = llim +1
            nextGuess = llim

    if "SAME" == resp[10:10+4]:
        nextGuess = midpoint
```

B.2 Data Processing Methods

```
def multiply(x,y):
    values = x[:,0] * y[:,0]
    errorbars=numpy.sqrt(((y[:,0]**2)*x[:,1]**2) + ((x[:,0]**2)*y[:,1]**2))
    return numpy.column_stack((values,errorbars))


def sqrt(x):
    values = numpy.sqrt(x[:,0])
    errorbars=0.5*((x[:,0]**(-0.5))*x[:,1])
    return numpy.column_stack((values,errorbars))


#IMPT: returns 2 lists: (modified t, column stack)
```

```
# Returns the midpoint of the linear interpolation between any two
# pairs of points in t. Therefore modt has one element less than t
def deriv(t,x):
    modt = numpy.zeros(len(t)-1)
    values = numpy.zeros(len(t)-1)
    errorbars = numpy.zeros(len(t)-1)

    for i in range(0,len(x)-1):
        values[i]=(x[i+1,0]-x[i,0])/(t[i+1]-t[i])
        modt[i] = (t[i+1] + t[i])/2
        errorbars[i] = (1/(t[i+1]-t[i]))*numpy.sqrt(x[i,1]**2 + x[i+1,1]**2)

    return modt, numpy.column_stack((values,errorbars))

# Trapezoidal integration. Returns the interval width multiplied by
# average of the function at either end of the interval.
def integrate(t,y):
    val = numpy.zeros(len(y))
    error = numpy.zeros(len(y))
    values = numpy.zeros(len(y))
    errorbars= numpy.zeros(len(y))

    for i in range(1, len(y)):
        val[i] = (t[i]-t[i-1])*((y[i,0]+y[i-1,0])/2)
        error[i] = numpy.sqrt(((t[i]-t[i-1])/2)*(y[i,1]**2+y[i-1,1]**2))
        values[i] = sum(val)
        errorbars[i] = (numpy.sqrt(sum((error[:]**2))))/i

    return numpy.column_stack((values, errorbars))
```

C Third Appendix

Your work will likely build on, rely on, or otherwise make use of the results or work of others. In all such cases it is essential that you properly 'reference' the appropriate original source. References should be numbered consecutively in order of first appearance throughout the text. Software is available to the McGill community to make this very easy (EndNote for Window/Word, BibTeX for LaTeX) which is available to all McGill students. References to Web URL's are to be discouraged but are sometimes necessary.

If you state a fact that is based on someone else's research or work you MUST cite it!