Overall Documentation

Heather Y.

June 14, 2018

1 Project Goals

The overall goal of the project is to simulate an analog computer using digital methods so that analog systems can be tested and constructed without the difficulties involved with obtaining and building large groups of op-amps - in other words, so that analog systems may effectively be prototyped.

This particular phase of the project concerned constructing an op-amp block devoted to integration. The methods used have been updated over the course of the project, from the trapezoidal method to the simpson method, and a function which varies the time step has been added, all in search of accuracy. Multiple modules have also been used. This document is an attempt at describing overarching methods used, describing commands from modules used in this project, and other notes to make future programming in this project easier.

Much of this code will not work if run on anything other than a Raspberry Pi. If you wish to run anything on a computer other than the Raspberry Pi, see the pigpio section for which lines to comment out, and the matplotlib section for which lines to add to view the output. The current program that is in use is the module-ion.py file (which imports the basicfunctions.py file) which contains custom integration, differentiation, summation, and other such methods.

For further description of any of the programs written for this project, see the individual documentation files written for each of those programs. These may reference other individual files or this file for further information.

2 matplotlib.pyplot

This module is used for graphing throughout the project. To graph, you must have a list or array of values. The commands involved are as follows:

- `import matplotlib.pyplot as plt` - the standard import format.
- `plt.plot(list1, list2)` - list1 and list2 are the two "axes".
- `plt.xlabel('text')` - label for x axis. You can also use `plt.ylabel()` for the y axis.

- `plt.plot(time, list1, 'bs', time, list2, 'r--')` - another valid plot. Here, time is a list of time values that is treated as the x-axis, and list1 and list2 are two functions graphed with respect to time. 'bs' and 'r–' refer to the method of identification - 'bs' is blue squares, and 'r–' red dashes.
- `plt.plot(time, list1, time, list2, time, list3)` - another valid plot. No need for 'bs' or 'r–' here - matplotlib can automatically assign separate styles.
- `plt.show()` - actually shows your plot.

3 pigpio

The pigpio module is used for communicating with the GPIO pins of the Raspberry Pi throughout the project. To run, one must run sudo pigpiod in the command line beforehand, which starts the pigpio daemon. The commands involved are as follows:

- `import pigpio`
- `pi = pigpio.pi()` - pi can be replaced with a variable name of your choice, though pi or my_pi or something similar is standard. `pigpio.pi()` simply starts things up.
- `pi.set_mode(pin, pigpio.OUTPUT)` - pin can be replaced with the pin number of your choice, pi with the variable name you set up on the previous item, and `pigpio.OUTPUT` with `pigpio.INPUT`, though all pins used in this project are outputs.
- `pi.set_PWM_frequency(pin, frequency)` - replace pi with your variable name, pin with pin number, and frequency with a frequency.
- `pi.set_PWM_dutycycle(pin, duty cycle)` - replace pi with variable name, pin with pin number, and duty cycle with a number from 0 to 255 - if you input 255, it will read as on 100% of the time, and if you input 0, it will read as off 100% of the time. Because it is a range from 0 to 255, the `pwm_map()` function is used (see the relevant documentation).
- `pi.stop()` - stops your use of pi as an interface with the GPIO pins. Does not stop all variables' interface, just pi. There is no cleanup equivalent to the `GPIO.cleanup()` (see pwm-test.py docs for description of RPi.GPIO commands; that module is not used in any in-use programs).

4 Timing with cProfile

To test the efficiency of the program (see description in ion-sim-timings.py) cProfile was used. Use the following command from the command line: `python3`

`-m cProfile -o profile-output.txt program.py` where profile-output.txt is the file you want the analysis outputted to and program.py is the name of the file to be profiled. In the analysis file, you want to read the third column to see the time it took for each function to be run once. Finally, to see the time it took for the whole program to run, do the following in the command line: `time python3 program.py` where program.py is the name of the file to be profiled.

In both cases rewrite your program so that any user inputs are defined in the code. If you're testing with a graph, you must close out of the graph for the results to appear.

5 Trapezoidal Method

The trapezoidal method works by adding the two most recent values for the integral, multiplying by the time step divided by two, and then adding the sum of all past integrals. It is fairly accurate, though not as accurate as the simpson method. There is an older version of this function in a commented out line in the integrate function in the opampblocks.py file. The cumulative method of the trapezoidal method is not new, and there is a scipy implementation called cumtrapz (see alt-integrate.py and ion-with-scipy.py docs); however, this is much slower than our own function.

6 Modified Cumulative Simpson Method

One can see the history of the iterations of the Simpson method that were gone through in the "integral test values" spreadsheet. Here will be given a brief overview of the most recent Simpson method.

Simpson's rule is defined as

$$\int_a^b f(x)\,dx \approx \tfrac{\Delta x}{3}\left(f(x_0)+4f(x_1)+2f(x_2)+4f(x_3)+2f(x_4)+\cdots+f(x_n)\right)$$

Note that this doesn't work by itself for our application because it requires knowledge of the function - $f(x_1), f(x_2), ..., f(x_{n-1})$ are unknown and can only be found via a linear regression, which produces effectively the same error as the trapezoidal function. However, there is another method. The main advantage of the Simpson method comes in the repeated weighting of values (which is where the multiplication by 4 and 2 comes from).

Searches resulted in the paper "A Modified Simpson's Rule and Fortran Subroutine for Cumulative Numerical Integration of a Function Defined by Data Points" by L.V. Blake, which is used to implement the Simpson's method. It redefines the Simpson method for cumulative numerical integration where the function is unknown. It does this by defining two equations, A_1 and A_2, which define the area under the curve from the middle point (i.e., the point $f(x_1)$) to the beginning point ($f(x_0)$) and to the end point of the subinterval ($f(x_n)$) (see Figure 1).

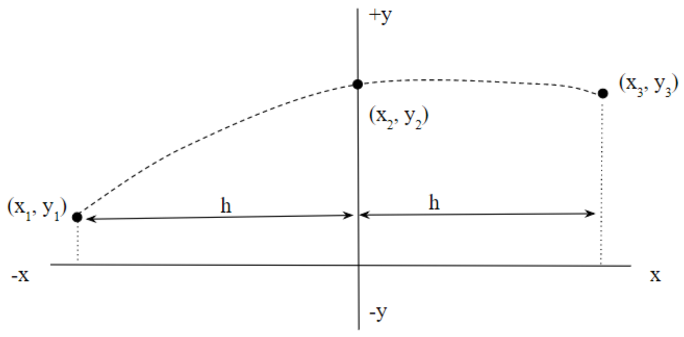

Figure 1: Diagram from page 4 of "A Modified Simpson's Rule"; a triplet of equispaced points in the xy coordinate system. The second degree polynomial approximation is shown dotted.

A_1 and A_2 are defined on page 5 of Blake's paper as follows:

$$A_1 = \frac{h}{3}\left(\frac{5}{4}y_1 + 2y_2 - \frac{1}{4}y_3\right)$$

and A_2 is defined as

$$A_2 = \frac{h}{3}\left(-\frac{1}{4}y_1 + 2y_2 + \frac{5}{4}y_3\right)$$

However, only A_2 was used in our method, as using A_1 and A_2 would result in adding sections of the integral twice. As can be seen in the "simptrap" column of the spreadsheet, this dramatically reduced error compared to the trapezoidal method.

This version was programmed, and did not improve accuracy, which after much testing lead to the conclusion that the ion simulation integrals could not reasonably be approximated by a second degree polynomial. This method was kept, however, because although it was not any more accurate in this particular case, it was not any less accurate, and it could be much more accurate in other situations.

The updated function has to check if the step is equal to one, in which case it simply uses the trapezoidal method, so that the cumulative modified simpson's method has three points of data to work with. The key to the new method is the weighting coefficients for each value; if these are tweaked at all (they are -.25, 2, and 1.25) the whole function will go horribly awry.

The one disadvantage to the simpson function relative to the trapezoidal is that, as it stands, it does not allow for time step scaling. The next step for this project is calculating the modification to the simpson rule that allows for universal time step scaling; as per page 8 of Blake's paper, this is possible, although the updated formulas are very cumbersome.

This method we have used for the Simpson method may be generalized to higher order Newton-Cotes methods - see the paper *Cumulative High-Order Newton-Cotes Algorithms* by the same authors as these programs for a description of the Simpson 3/8ths cumulative method and its use.

7 Time Step Scaling

See in particular the ion-sim-step-vary.py docs for more details on this.

It returns either the 'normal' time step or the 'fine' time step, that is, 10^{-12} or 10^{-15}, based upon the ratio between the previous two x acceleration, y acceleration, x velocity, and y velocity values. It does by taking the last two values from the array, dividing them by each other to find their ratio (did it increase by five percent? 0.1 percent? etc) and then taking the absolute value. It then checks if any of these are .000075% above or below one, in which case it switches to fine, otherwise, it keeps it at the normal setting.

It does all of this in a try except block to account for when acceleration or velocity is near zero at the outer tips; when a ZeroDivisionError is excepted it simply returns the normal setting.

However, it is slightly unclear as to whether or not this works as intended, or if it simply speeds up the program and helps improve the accuracy by cutting out the fraction of steps that are small and running those at lower accuracy, reducing the step size. Using this function, it takes five million steps for the ion to make one complete "loop" about the origin.

8 `pwm_map` function

For confirmation of this function, which is used with the pigpio module, see the spreadsheet entitled "pwm values" on the right-hand side. This has test cases and their results.

The function is meant to appropriately scale the values you want as outputs to a duty cycle percentage, which is required by the pigpio module. It takes as inputs the value, the minimum possible value, the maximum possible value, and the number of bits of resolution the system has. The minimum and maximum are labelled mini and maxi due to the built in python functions `min()` and `max()`.

It takes two to the power of the number of bits and then subtracts one, which it defines as the scale factor. Then, to account for shifts of the minimum possible value from zero, it either adds or subtracts the distance to zero from all input values, then divides by the range and multiplies by the scale factor. This is basically where the input value is at in the range, and then scales it to the new range provided by the resolution.

This is all needed because as noted in the pigpio section, the `.set_PWM_dutycycle()` command requires a number between 0 and 255 so all output values of functions must be scaled appropriately if they are to be read out in this manner.

9 Notes on Scientific Programming in Python

Firstly, a note on fixed point versus floating point math. When it was believed that numerical instabilities were resulting from floating point math, fixed point math was looked at as an alternative. It was rejected because it can cause information loss and inaccuracies as per Wikipedia and other more reliable sources.

Further, it was eventually determined that numerical instabilites were most likely not arising from floating point math, as the decrease of the time step improved the results, even though the numbers were smaller and therefore more prone to floating point error.

The use of deque and decimal are discussed in the ion-sim.py docs. Finally: as evidenced by our find of the "A Modified Simpson's Rule" paper by Blake, and our find of the Elementary Numerical Analysis text, most of these methods and their edge case uses will have been looked at before. A good google search, in particular, could have saved us a good deal of effort on some particular trains of thought for improving our system.