User Guide

Heather Y.

June 20, 2018

1 Introduction

This document is meant to be an introduction to the analog-comp programming system. First, the setup of the most complete simulation is discussed, along with the use of the plain module's functions. Next, an explanation is given of how to adjust parameters in the example simulation, and how to call the functions. Next, there is a discussion of how to write your own simulations using the analog-comp system. Finally, the files other than the primary simulation files and main module are discussed briefly.

The analog-comp main files require that you have installed pigpio and matplotlib; all other modules used are built-ins. If you wish to use some of the other files, you may need to install RPi.GPIO or scipy. Installation of these is discussed for linux in the setup section. The analog-comp files are also meant to be run on a Raspberry Pi; see the Overall Documentation file for more information.

2 Setup

2.1 Raspberry Pi

To begin using the analog-comp system, begin by booting up your Raspberry Pi and opening the command line. You will need to install pigpio and matplotlib. Begin by using the command `sudo apt-get install python3-matplotlib` to install matplotlib. To test this, type `python3` into the command line and type in `import matplotlib` into the prompt. If there are no errors, the installation has taken place successfully.

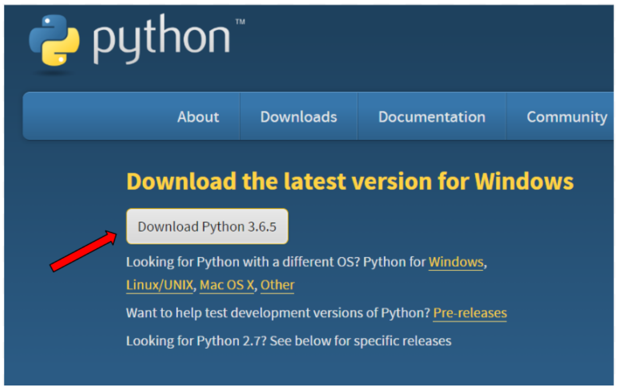
To install pigpio, type into the command line `sudo apt-get install python3-pigpio`. Once installed, type into the command line `sudo pigpiod` which starts the pigpio daemon. Once this is done, check that pigpio is properly installed by typing `python3` into the command line. Once in the prompt, enter `import pigpio`. If there are no errors, the installation has taken place successfully.

Once you have these packages installed, open the analog-comp folder in your files directory (if you have put the files onto the Raspberry Pi using a jump drive, there should have been a prompt to open the relevant folder in the file directory; the files should be located in the /media/pi folder under the name of the jump drive). Check that you have both the module-ion.py file and the basicfunctions.py file and open the module-ion.py file using IDLE 3. Once this has opened, select Run > Run Module from the top bar.

A new window should open labelled "Python 3.5.3 Shell" and under the RESTART heading, your program will begin running. Any text outputs will show up here. The graph produced at the end of most of the programs will pop up as a separate window.

2.2 Windows

On Windows, begin by installing Python 3. To do this, go to `https://python.org/downloads` and click on the "Download Python 3.6.5" button, which should be right below a heading that says "Download the latest version for Windows".



Download the latest version for Windows

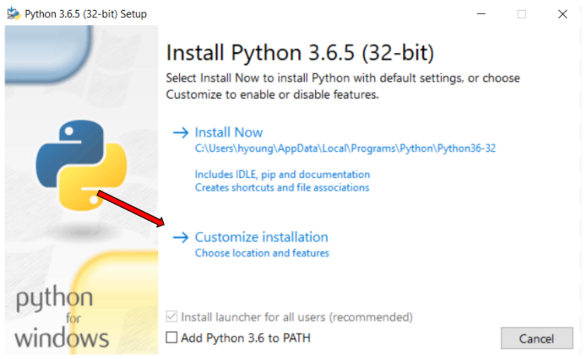
Looking for Python with a different OS? Python for Windows, Linux/UNIX, Mac OS X, Other

Want to help test development versions of Python? Pre-releases

Looking for Python 2.7? See below for specific releases

Figure 1: Click the button shown by the arrow above to begin installing python.

Selecting this button should automatically download the python installer. Open the python installer and select the bottom button, which should look something like that in figure 2.



Install Python 3.6.5 (32-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

→ Install Now
C:\Users\hyoung\AppData\Local\Programs\Python\Python36-32

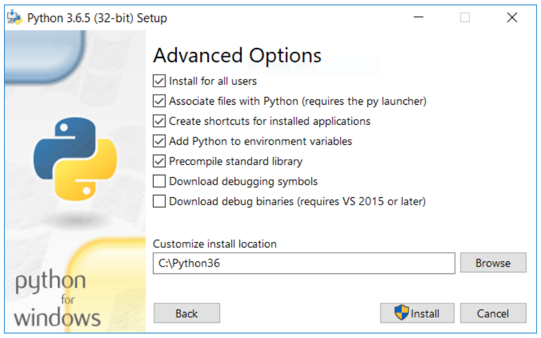
Includes IDLE, pip and documentation
Creates shortcuts and file associations

→ Customize installation
Choose location and features

☑ Install launcher for all users (recommended)
☐ Add Python 3.6 to PATH

Figure 2: Select the button marked by the arrow to begin setting up python.

In the new window that pops up, click next. In the next window, check the boxes shown in Figure 3 (however, keep the path name the same as what automatically pops up).



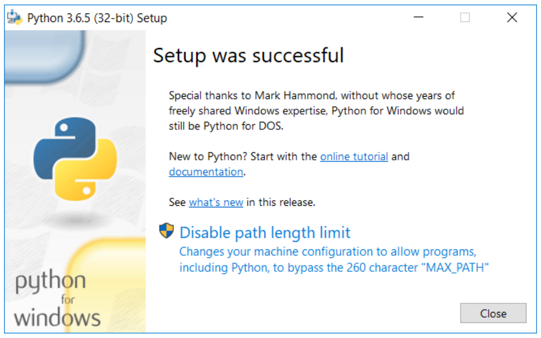
Advanced Options

☑ Install for all users
☑ Associate files with Python (requires the py launcher)
☑ Create shortcuts for installed applications
☑ Add Python to environment variables
☑ Precompile standard library
☐ Download debugging symbols
☐ Download debug binaries (requires VS 2015 or later)

Customize install location
C:\Python36

Figure 3: Select the boxes shown for a proper installation.

When it has finished installing (a new screen will pop up after the green progress bar), select the "Disable path length limit" option shown and then close the installer (see Figure 4).



Setup was successful

Special thanks to Mark Hammond, without whose years of freely shared Windows expertise, Python for Windows would still be Python for DOS.

New to Python? Start with the online tutorial and documentation.

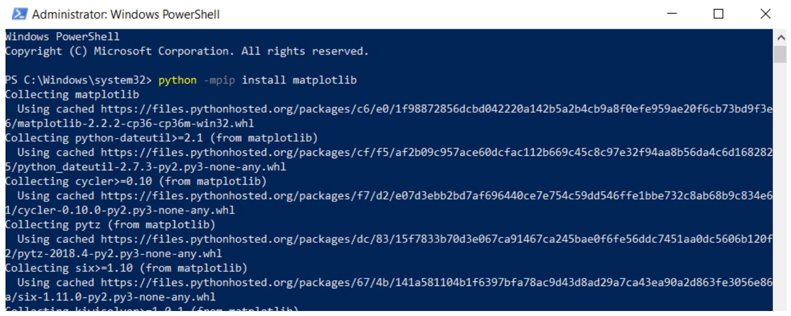
See what's new in this release.

Disable path length limit
Changes your machine configuration to allow programs, including Python, to bypass the 260 character "MAX_PATH"

Figure 4: Click the "disable path length limit" option shown before closing the installer. You will know it has run because the option will no longer show up.

Next, open your applications tab as accessed by the windows key. Find the python 3 folder, and then open the drop down and select IDLE. This will open the interactive prompt for python. If this is successful, python has been installed properly.

To install matplotlib, press Windows+X and click on "Windows Powershell (Admin)". Once this is opened run `python -mpip install matplotlib` (see Figure 5).



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Windows\system32> python -mpip install matplotlib
Collecting matplotlib
  Using cached https://files.pythonhosted.org/packages/c6/e0/1f98872856dcbd042220a142b5a2b4cb9a8f0efe959ae20f6cb73bd9f3e
6/matplotlib-2.2.2-cp36-cp36m-win32.whl
Collecting python-dateutil>=2.1 (from matplotlib)
  Using cached https://files.pythonhosted.org/packages/cf/f5/af2b09c957ace60dcfac112b669c45c8c97e32f94aa8b56da4c6d168282
5/python_dateutil-2.7.3-py2.py3-none-any.whl
Collecting cycler>=0.10 (from matplotlib)
  Using cached https://files.pythonhosted.org/packages/f7/d2/e07d3ebb2bd7af696440ce7e754c59dd546ffe1bbe732c8ab68b9c834e6
1/cycler-0.10.0-py2.py3-none-any.whl
Collecting pytz (from matplotlib)
  Using cached https://files.pythonhosted.org/packages/dc/83/15f7833b70d3e067ca91467ca245bae0f6fe56ddc7451aa0dc5606b120f
2/pytz-2018.4-py2.py3-none-any.whl
Collecting six>=1.10 (from matplotlib)
  Using cached https://files.pythonhosted.org/packages/67/4b/141a581104b1f6397bfa78ac9d43d8ad29a7ca43ea90a2d863fe3056e86
a/six-1.11.0-py2.py3-none-any.whl
```

Figure 5: Running the installation command for matplotlib in the windows administrator powershell.

There is no need to install pigpio on windows as it will only work on a Raspberry Pi.

When this is done, type `python` into the shell and then `import matplotlib`. If there are no errors, the installation has been successful. (To exit the shell, type `quit()`.)

You can now open any python file from IDLE by opening IDLE and then clicking File > Open... which will allow you to open the file of your choice. To run off of windows, first open the file basicfunctions.py and comment out lines three and five using the # symbol.

You can then select any file from the analog-comp suite, such as module-ion.py. If you wish to run module-ion.py, see the "Parameter Adjustment" section. If you wish to create your own simulation, see the "New Simulations" section.

3 Parameter Adjustment

3.1 module-ion.py

Once the module-ion.py file has been opened, you can adjust the initial simulation parameters by changing lines five through nine of the code (line numbers show up in the bottom right of the IDLE editor, prefixed by "Ln: "). To adjust the starting x position of the item, change the line marked x_pos. To do this, set your cursor after the equals sign and delete the number currently there, replacing it with your new starting value. To change the starting y position, change the line marked y_pos. To change the axial magnetic field, change the line marked axial_mag. To change the electric potential, change the line marked elect_pot. Finally, to change the number of steps the program will run, change the line marked max_steps.

A word of warning on this last change - 25,000,000 steps, for example, can take almost half an hour on the Raspberry Pi. Choose this number carefully based upon the accuracy you are running the system at, as you want the minimum number of steps needed to show you your result. To change the system accuracy, scroll down to line 31 and change the number in the parentheses to the precision of the system. The maximum recommended number is -15, and the minimum -11. -13 should be sufficient for most precision needs; at this level of precision, only 5,000,000 steps are necessary, which takes around ten minutes of running time.

If you wish the output to show up as a graph on the Raspberry Pi, there are no changes that need to be made. However, if you wish to change the output to show up on an oscilloscope, you will first need to remove the hashtag ('#') symbol on lines 68, 69, 75, 76, 77, and 93 (after doing this, the text on these lines should appear black instead of red). If you wish to change back to graph-only output after this, you will need to replace the hashtags at the beginning of those lines.

If you want the output to show up on the oscilloscope, you must wire up the pins marked 'G4' and 'G27' on the Sparkfun Pi Wedge up to a low-pass filter (see Figure 6). Finally, if you wish the output to show up on an oscilloscope and have also changed the precision and step number from the default, you must first view the standard graph produced and look at the minimum and maximum values printed on the graph. You must then change the "pos_mini" and "pos_maxi" values (lines 34 and 35) based on the minimum and maximum values outputted by the program.

To change the integration method between simpson and trapezoidal, go to line 38 of the module-ion.py file and change the text between the quotation marks to ”simp” or ”trap”. The text should be green. If anything other than simp, trap, or the quotation marks are green, you are missing a quotation mark, without which the program will not work.

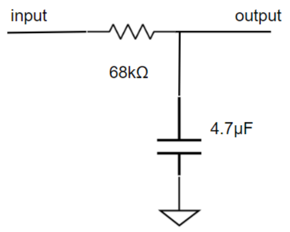


Figure 6: Schematic for a low-pass filter, necessary for converting the PWM output produced by pigpio into something that will make sense on an oscilloscope.

3.2 basicfunctions.py

If you want to use the functions contained in basicfunctions.py in another file, you must first make sure your other python file is in the same directory as basicfunctions.py. You can easily make sure this is achieved by opening basicfunctions.py in the IDLE editor and clicking File > New File while in basicfunctions.py. In this new file, you must add the line `import basicfunctions` to use the functions in this file. If you want to make using the functions easier, you can use code along the lines of `import basicfunctions as b`. To call a function from the file, use the code `basicfunctions.function(parameters)` (or if you used the latter import command, the code `b.functions(parameters)`). Replace function with your function name and parameters with the parameters required for that function.

The parameters required for each function are described in each function’s definition in the file. Some general notes: never import `chop_num()` or `decorator()`; these are both functions used by other functions. If you wish the `chop_num()` function to be used or not used, you must change the variable chop_flag, on line 7, to True or False respectively (remember, edit the text after the equals sign).

4 New Simulations

Begin by creating a new file in IDLE. To do this, open the IDLE shell and select File > New File. Save the new file into the same folder that basicfunctions.py is located in. First, type `import basicfunctions` as described in the ”Parameter Adjustment - basicfunctions.py” section. Your file will now be able to access all of the custom analog-comp functions. (If you are curious about how these functions work, see the Overall Documentation.)

Next, begin to map out your simulation. To show how to do this, the example of solving the equations of motion for a damped harmonic oscillator will be given. We know that the motion of a damped harmonic oscillator with no external forces can be described as

$$\frac{d^2x}{dt^2} + 2\zeta\omega_0\frac{dx}{dt} + \omega_0^2 x = 0$$

where $\omega_0 = \sqrt{\frac{k}{m}}$ and $\zeta = \frac{c}{2\sqrt{mk}}$ where k is the spring constant, m is the mass, and c is the viscuous damping coefficient.

From here, we identify the inputs for our system, that is, k, m, and c. We also note the values that can be immediately calculated from these inputs, ω_0 and ζ. We also make note of the basicfunctions we'll need - multiply, divide, and sqroot (for a full listing of basicfunctions and their parameters, see the basicfunctions.py docs). Thus, our program now looks something like this:

```python
import basicfunctions as b
import matplotlib.pyplot as plt

k = 5
m = 5
c = 1
omega = b.sqroot(b.divide(k, m))
zeta = b.divide(c, b.multiply(2, b.sqroot(b.multiply(m,k))))
```

Now we begin outlining how we must solve the differential equation, or rather, how an analog computer would do this. A useful guide to this is "Analog Computing Technique" by Robert Paz; see especially pages 13 through 17. We can do it using an analog block diagram (see Figure 7).

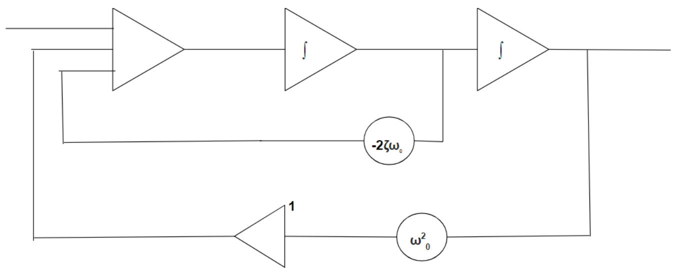


Figure 7: Analog block diagram of how to solve our differential equation. Note the loose line at the left is the input signal and the loose line at the right is the output signal.

We then translate this diagram into code.

The code first initializes all the different variables needed to integrate in lines 1 through 13. Each integral requires a variable that is set to the previous

```python
signal = 0
signal_array = [0]
position_array = []
velocity_array = []
position_init = 1
velocity_init = 0
position_val = 0
velocity_val = 0
step = 0
max_steps = 1000
t_step = 0.1
time = []
method = 'simp38'

while step <= max_steps:
    time.append(t_step*step)
    if step == 0:
        velocity_val = velocity_init
        position_val = position_init
        position_array.append(position_init)
        velocity_array.append(velocity_init)
    elif step == 1:
        signal = b.subtract(b.multiply(-2,omega,zeta,velocity_val), b.
            ↪ multiply(b.square(omega),position_val))
        signal_array.append(signal)
        velocity_val = b.integrate(step, t_step, signal_array,
            ↪ velocity_val, method, velocity_init)
        velocity_array.append(velocity_val)
        position_val = b.integrate(step, t_step, velocity_array,
            ↪ position_val, method, position_init)
        position_array.append(position_val)
    else:
        signal = b.subtract(b.multiply(-2,omega,zeta,velocity_val), b.
            ↪ multiply(b.square(omega),position_val))
        signal_array.append(signal)
        velocity_val = b.integrate(step, t_step, signal_array,
            ↪ velocity_val, method, velocity_init)
        velocity_array.append(velocity_val)
        position_val = b.integrate(step, t_step, velocity_array,
            ↪ position_val, method, position_init)
        position_array.append(position_val)
    step+=1

plt.plot(time, signal_array, time, velocity_array, time, position_array
    ↪ )
plt.show()
```

integral's value, a list that contains the values of the integral (note that for this it is sometimes convenient at high step numbers to use a deque), and an initialization value. A variable must also hold the method the user wishes the integrate function to use. The other variables in this section set up the iterating step variable, the maximum number of steps, the time step, and an array used for the x axis of the output plot.

The code next enters runtime, which has three phases (runtime is lines 15 through 36). The first runtime phase finishes the initialization of variables (lines 17 through 21) and appends the current time step to the time list (line 16). The second runtime phase starts up the integration process (lines 22 through 28). Finally, the third runtime phase enters the standard integration process (lines 29 through 36).

The output is produced in lines 38 and 39 using matplotlib.pyplot (described in the Overall Documentation). The output produced by this code is shown in Figure 7.

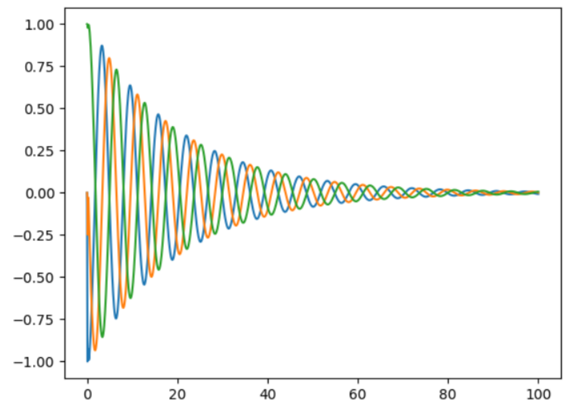


Figure 8: Output of our simulation for a damped harmonic oscillator. This shows the signal, velocity, and position.

5 Other Files

Most of the other files included in the analog-comp system are test files or older versions of the current program; however, a few files of interest will be mentioned here. The ion-with-scipy.py file integrates using the `scipy.cumtrapz()` function; however, this is very slow and is not used in the main files. The xy-pwm-test.py and pwm-test.py files can be used to ensure you have pigpio, the low-pass filter, and the oscilloscope properly setup. The opampblocks.py file is a test of the integrate functions on simpler functions, like sine and square waves. Finally, the ion-sim-timings.py file is the next-oldest iteration of the main ion simulation code, and may be looked at for interest. Information about them can be found in their individual doc files. Most of these files, however, as stated, are obsolete or meant for testing obsolete code.