AC Module Reference

This document is the reference for the AC sub module of the Python SLab system.

Index

Introduction

The “SLab System” is a software solution that provides a low cost access to circuit measurements by the use of low cost development boards.
The complete system has a natural tendency to feature creeping. In order to make the dimension of the project easy to manage, it has been divided in several modules.
The main module, associated to the file slab.py, contains all the code that directly access the Hardware Board and the most immediate low complexity commands.
The functionality of the main module is extended by the use of additional modules. This document describes the AC module, associated to the file slab_ac.py that includes a set of functions to work with the measurement of the frequency response of circuits.

If you want to use the AC module you will also need to use the main slab module as it is the one that includes the ***connect*** command that starts the communication with the Hardware Board. That means that you will need to use two imports.

```
>>> import slab
>>> import slab_ac as ac
```

If you need to use other additional modules you can add more lines for other sub modules. Observe that we have used an alias name for the module namespace so, a command with the name “command” will be accessed as “ac.command”.

Frequency Response Commands

Commands in this section are useful to measure the response of a circuit against tones of constant frequency. The commands in this section use the slab.py module wave commands to generate and measure the response of the circuit against sine waves.

Note that the SLab system is not designed for frequency response characterization. As the hardware boards don't feature any amplification circuit, dynamic range is poor. Using 12 bit ADCs and DACs we can get a maximum dynamic range of 72dB but that's only true at ideal conditions. Taking into account that the signals won't usually reach the full range of the converters and the measurement noise, the dynamic range will be much lower. For instance, with a 10mV noise and signals that only reach half Vdd range, the dynamic range will be reduced to 44dB. That's only two decades for a first order zero or pole.

In practice, using an experimental setup that makes uses of most of the ADC range you can obtain about 60 dB dynamic range in magnitude measurements as dB measurements are logarithmic and thus are quite forgiving for noise errors. Phase measurements, however, will be contaminated with noise much before reaching this point.

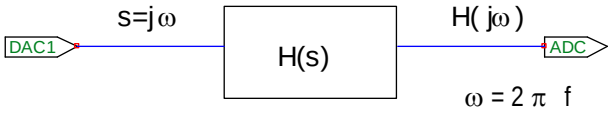
You should also take care of low frequency noise as it will limit the available dynamic range.

Frequency response commands use the base transient and wave commands to operate. So, they modify any previous setting configured with ***setSampleTime***, ***setTransientStorage***, ***setWaveFrequency*** and erase any wavetable loaded with the any waveform generation command.

sineGain(v1, v2, freq, channel, npre, maxfs)

Arguments	v1	Low peak value (V)	V1
	v2	High peak value (V)	
	freq	Tone frequency (Hz)	
	channel	ADC channel to read (Defaults to 1)	
	npre	Number of preset cycles (Defaults to 5)	
	maxfs	Maximum sample frequency (Optional parameter)	
Returns		Complex gain	

Obtains the complex gain of a circuit against a sine wave tone. The tone will be applied using DAC 1 and the response of the circuit will be read at the indicated ADC channel (1 by default).



The sine wave will be generated at DAC 1 as:

$$A_{DAC} = \frac{v2 - v1}{2} \qquad M_{DAC} = \frac{v1 + v2}{2} \qquad DAC\ 1(t) = M_{DAC} + A_{DAC} \cdot sin(2\pi \cdot freq \cdot t)$$

So the low peak of the wave will be at v1 and the high peak of the wave at v2.

The output of the circuit read at the ADC is processed to obtain the unknowns M_{ADC}, A_{ADC} and φ_{ADC} that give the best approximation to:

$$ADC\ 1(t) \approx M_{ADC} + A_{ADC} \cdot sin(2\pi \cdot freq \cdot t + \varphi_{ADC})$$

After solving for the unknown values, the complex gain is reported as:

$$gain = \frac{A_{ADC}}{A_{DAC}} cos(\varphi_{ADC}) + j \frac{A_{ADC}}{A_{DAC}} sin(\varphi_{ADC})$$

Parameter ***npre*** selects the number of sine cycles to produce before start measuring the circuit and defaults to 5 cycles.
Paremeter ***maxfs*** selets the maximum sample frequency to use and defaults the maximum reported by on the board. If you get a ***Sample Overrun*** exception it means that measurements are not fast enough for the maximum sample frequency so you will need to select a lower sample frequency. Note that the command will generate an exception if the maximum sample frequency is not, at least, 4 times the tone frequency to use.

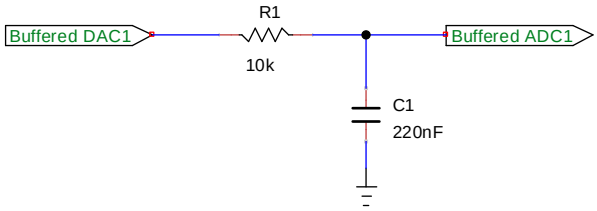
At tone frequencies close to the maximum limit, several cycles will be measured to reduce the errors. However, errors will increase the closer you get to the limit, especially in the phase response as any uncalibrated delay between the DAC and the ADC will produce a phase response error.

If the maximum or minimum of the output waveform is too close to the supplies, a saturation warning will be generated.

Example 19:

Show the response of a first order low pass filter at the corner frequency.

We will test the same circuit of example 18, but now, we will only use the DAC 1 and ADC 1 connections:



The corner frequency is:

$$f_c = \frac{1}{2\pi(R1 \cdot C1)} \approx 72Hz$$

We can obtain the response of the circuit ay this frequency using the following code. Note that we use the default ADC channel 1.

```
>>> print "Response at 72Hz"
>>> gain = slab.sineGain(1.0,2.0,72.0)
>>> print "  Gain = " + str(gain)
>>> m,p = slab.magnitudePhase(gain)
>>> print "  Magnitude = " + str(m)
>>> print "  Phase = " + str(p) + " deg"
```

Running the code, we will get on screen:

```
Response at 72Hz
  Gain = (0.521542448508-0.498544530811j)
  Magnitude = 0.721493710851
  Phase = -43.7084822512 deg
```

That is quite close to the expected value of 1/√2 magnitude and 45º phase.

sineGainAll(v1, v2, freq, npre, maxfs)

Arguments	v1	Low peak value (V)	V1
	v2	High peak value (V)	
	freq	Tone frequency (Hz)	
	npre	Number of preset cycles (Defaults to 5)	
	maxfs	Maximum sample frequency (Optional parameter)	
Returns		List of complex gains	

Obtains the complex gain of a circuit against a sine wave tone for all ADCs.
The tone will be applied using DAC 1 and the response of the circuit will be read at all ADC channels in sequence.
Data is returned as a list of complex gains with one element for each ADC.

freqResponse(v1, v2, fvector, channel, npre, maxfs)

Arguments	v1	Low peak value (V)	V1
	v2	High peak value (V)	
	fvector	List of frequencies (Hz)	
	channel	ADC channel to use (Defaults to 1)	
	npre	Number of cycles before measurement (Defaults to 5)	
	maxfs	Maximum sample frequency (Defaults to board reported)	
Returns		Complex gain numpy array	

This command computes the complex gain of a circuit by calling the ***sineGain*** command for the list of frequencies contained in the **fvector** parameter. The command ***bodeResponse*** issues this command to get the plot data.

That way, the ***bodeResponse*** command:

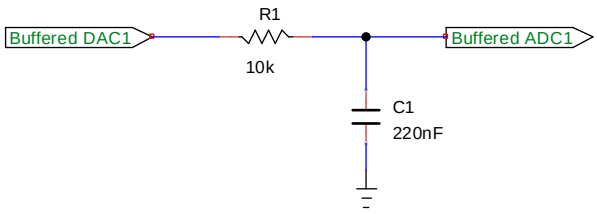
```
>>> slab.bodeResponse(0.5,2.5,10.0,8000.0,10,npre=5)
```

can be implemented as:

```
>>> fv = logRange(10.0,8000.0,ppd=10)
>>> gv = freqResponse(0.5,2.5,fv,npre=5)
>>> plotBode(fv,gv)
```

Example 23:
Compare the ideal and real response of a first order low pass filter

We use the following low pass circuit:



Then we use the following code to compare the real and ideal response:

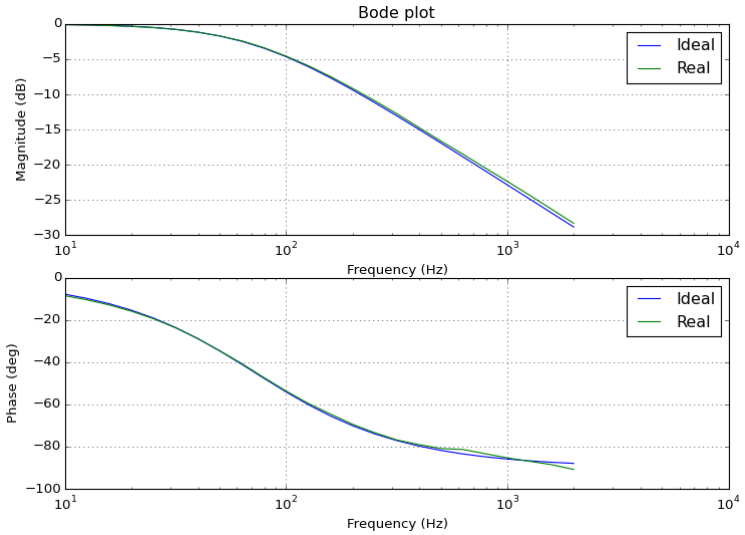
```
>>> fv = slab.logRange(10.0,8000.0,10.0)

>>> R = 10000.0
>>> C = 220.0e-9
>>> fc = slab.w2f(1/(R*C))

>>> g0 = 1.0/(1.0+1j*fv/fc)
>>> g = slab.freqResponse(0.5,2.5,fv,npre=5,maxfs=35000)

>>> slab.plotBode([fv,fv],[g0,g],["Ideal","Real"])
```

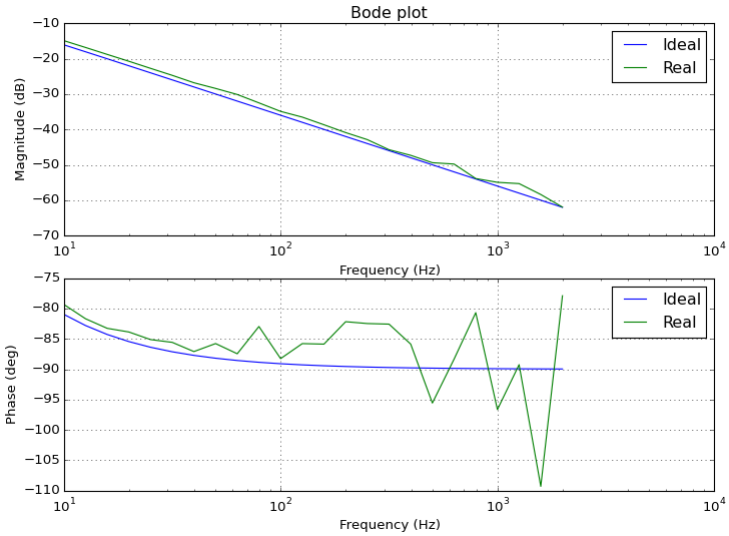
After the last command we will get the plot:



Note that we have measured frequency values up to 2 kHz. This is 1/20 of the maximum sample frequency of the board. Above this frequency we get a slight degradation of the magnitude plot and a notable degradation of the phase plot.

Dynamic range of the frequency response is limited due to the lack of amplification on the system. For a peak to peak amplitude of 2V, your dynamic range is limited to 68dB if there is not any noise at all.

If we change the capacitor C of the previous filter by a 10uF value, the corner frequency is reduced to 1.6Hz. The response will be in that case:



Observe that the magnitude response is quite correct at 60 dB attenuation, but the phase response measurement is bad for attenuations over 30 dB.

freqResponseAll(v1, v2, fvector, npre, maxfs)

Arguments	v1	Low peak value (V)	V1
	v2	High peak value (V)	
	fvector	List of frequencies (Hz)	
	npre	Number of cycles before measurement (Defaults to 5)	
	maxfs	Maximum sample frequency (Defaults to board reported)	
Returns		List of complex gain numpy array	

This command computes the complex gain of a circuit at all ADCs by calling the ***sineGain*** command for the list of frequencies contained in the **fvector** parameter.
So it is similar to ***freqResponse*** but measurements are performed at all ADCs instead of only using ADC 1.
Data is returned as a list of complex gains with one element for each ADC.

Frequency Plot Commands

Commands in this section draw plots of magnitude and phase against a horizontal frequency axis.

plotBode(fvector, gvector, labels, linear)

Arguments	f	Frequencies	V1
	g	Gains	
	labels	List of labels	
	linear	Select if we use linear magnitudes (Defaults to False)	
Returns		Nothing	

Auxiliary

Generates a bode plot from frequencies and complex gains.
Frequency horizontal axis will feature logarithmic spacing.

This function can be called in two modes:

In the first mode f and g are lists or numpy arrays that contain frequencies and complex gains. The command will show a bode plot for this data.

In the second mode both f and g are lists of several frequency and gain vectors. The command will superpose several bode plots in this case.

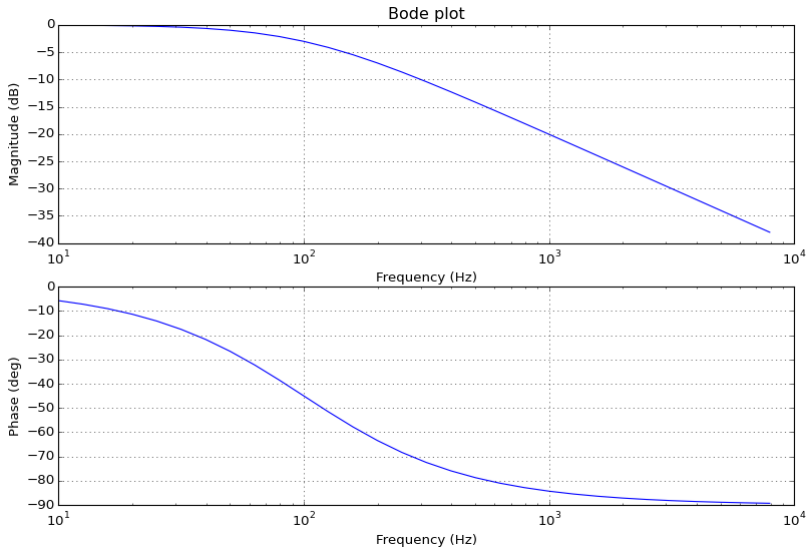
If we use ***linear = True*** , magnitudes will show in linear values instead of dB so the plot will no longer be “Bode” but “Frequency Response”.

Example 21A:
Draw a bode plot of a first order 100 Hz low pass filter

We can calculate the ideal response of the filter

```
>>> freq = slab.logRange(10.0,10000.0)
>>> g1 = 1.0/(1.0+1j*freq/100.0)
>>> slab.plotBode(freq,g1)
```

We will obtain the following plot:



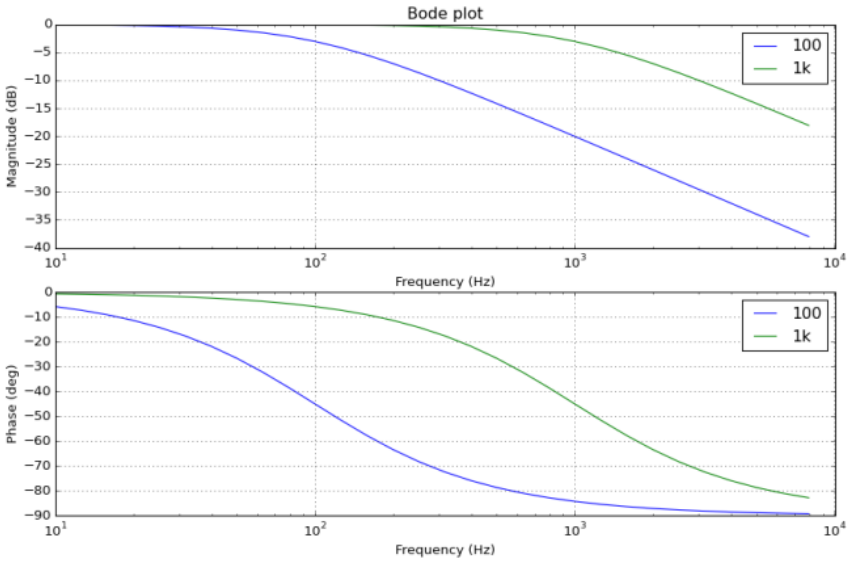
Example 21B:

Draw a bode plot of two first order 100 Hz and 1 kHz low pass filters

We add the following code after the previous "A" example:

```
>>> g2 = 1.0/(1.0+1j*freq/1000.0)
>>> slab.plotBode([freq,freq],[g1,g2],["1k","10k"])
```

We will obtain the following plot:



plotFreq(fvector, gvector, labels)

Arguments	f	Frequencies	V1
	g	Gains	
	labels	List of labels	
Returns		Nothing	

Auxiliary

This command is similar to the ***plotBode*** command but the spacing in the frequency horizontal axis will be linear instead of logarithmic.
Data in the vertical Y axis will always be linear also.

Bode Response Commands

Commands in this section combine a frequency response command with a frequency plot command.

bodeResponse(v1, v2, fmin, fmax, ppd, channel, npre, maxfs, returnData)

Arguments	v1	Low peak value (V)	V1
	v2	High peak value (V)	
	fmin	Minimum frequency (Hz)	
	fmax	Maximumfrequency (Hz)	
	ppd	Points per decade (Defaults to 10)	
	channel	ADC channel to use (Defaults to 1)	
	npre	Number of cycles before measurement (Defaults to 5)	
	maxfs	Maximum sample frequency (Defaults to board reported)	
	returnData	Return plot data (Defaults to False)	
Returns		See text. Generates a plot	

This command makes successive calls to ***sineGain*** to obtain the frequency response of a circuit. After all measurements, results are shown on a bode plot.

Parameteres **v1**, **v2**, **npre** and **maxfs** are the same than in the ***sineGain*** command. Refer to this command for a deeper explanation.
Frequencies are spaced geometricaly (linear in logarithmic space). Parameter **fmin** set the minimum frequency to measure, **fmax** the maximum and **ppd** the number of frequencies to test at each decade. If **ppd** is omitted it defaults to 10.

The command returns a tuple of two vectors with the frequency values and the complex gains for the circuit.

```
fvector, gvector = bodeResponse( .... )
```

Phase measurement is very sensitive to timing errors. Errors in the phase response can be high for frequencies over 1/20 of the maximum sample frequency of the board if signal level is low.
In the same way, high attenuation values can get outside of the measurement limits for the SLab system and give wrong results. Special care should be taken for very low frequencies as there could be high levels of low frequency noise,

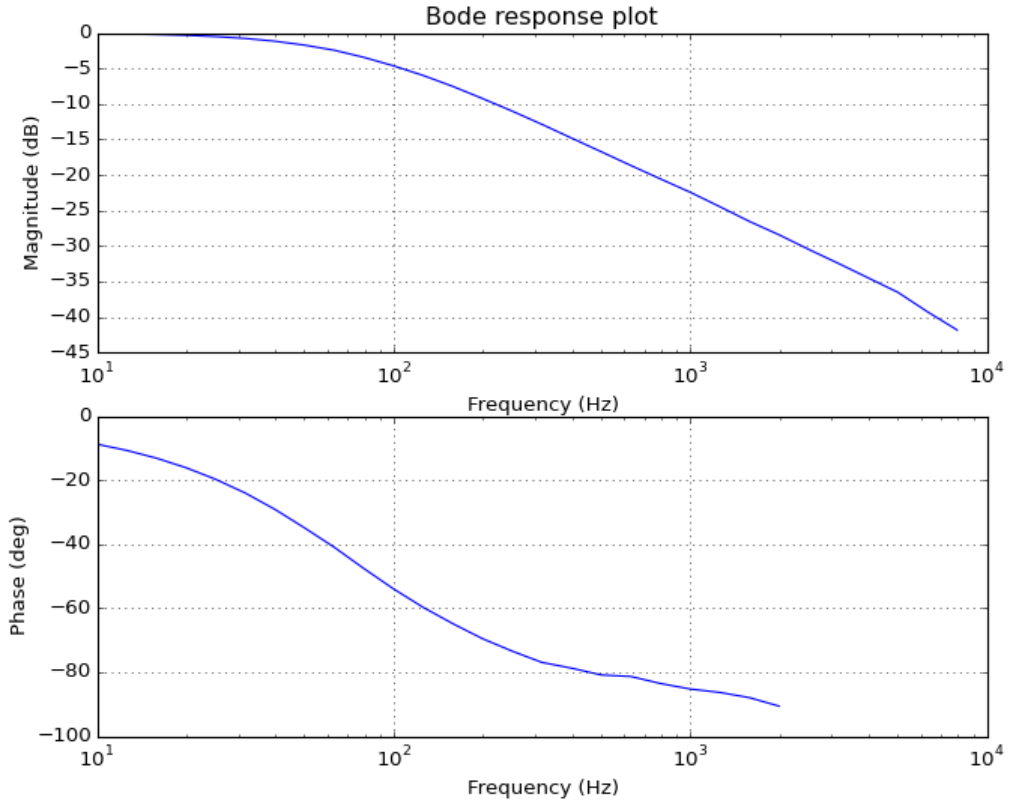
If optional parameter ***returnData*** is True or ***setPlotReturnData(True)*** was called previously, this command returns a tuple with two numpy arrays: frequencies and complex gains.

Example 20:
Show the bode response of a first order low pass filter.

We use the same filter of example 18 with the same connections.
In order to obtain the bode response we just need to use one command that sweeps between 10Hz and 8kHz with a sine wave between 0.5V and 2.5V. We have set the maximum sample frequency to 35kHz because the default 40kHz value for the Nucleo 303RE board gave a Sample Overrun exception.

```
>>> slab.bodeResponse(0.5,2.5,10.0,8000.0,10,npre=5,maxfs=35000)
```

The obtained bode plot will be:



Note that the phase plot is shown only up to 2 kHz. That is 1/20 of the maximum 40 kHz sample frequency of the board.

Utility Functions

Functions in this section take a float value or a numpy array and return an object of the same kind with the same size. For floats, the function is executed once. For numpy arrays, the function is executed for each element of the array returning an array with the same number of elements than the input array.
As these commands don't use the hardware board, they are all auxiliary and you don't need to connect to use them.

logRange(start, end, ndec ,ppd)

Arguments	start	Start value	V1
	end	End value (One of *end* or *ndec* must be provided)	
	ndec	Number of decades	
	ppd	Points per decade (default to 10)	
Returns		Numpy array of values	

Auxiliary

Creates a geometrically spaced (linear log spaced) numpy array that starts at the given value. The end of the space can be indicated with the ***end*** parameter or the ***ndec*** parameter.
The ***ppd*** parameter indicates the number of points per decade which defaults to 10.

Examples:

```
>>> f = logRange(fstart,fend)          # Range with default 10 ppd
>>> f = logRrange(fstary,fend,ppd=20)  # Range with 20 ppd
>>> f = logRrange(fstart,ndec=4)       # 4 decades from fstart with default 10 ppd
>>> f = logRange(fstrat,ndec=4,ppd=5)  # 4 decades with custom ppd
```

f2w(value)

Arguments	value	Float or numpy array	V1
Returns		Converts from frequency (Hz) to angular speed (rad/s)	

Auxiliary

w2f(value)

Arguments	value	Float or numpy array	V1
Returns		Converts from angular speed (rad/s) to frequency (Hz)	

Auxiliary

dB(value)

Arguments	value	Float or numpy array of gains	V1
Returns		Convert from linear to dB values	

Auxiliary

Formula applied is:

$$dB(value) = 20 \cdot log_{10}(value)$$

magPhase(value)

Arguments	value	Float or numpy array of complex numbers	V1
Returns		magnitudes and phases	

Auxiliary

Take one complex value or a numpy array of complex values and generates two floats or arrays that include the magnitude and phase of each input element.
Phase is given in degrees (0 to 360).

Example:
Convert from complex gain to magnitude and phase

```
>>> g = 0.7 + 1j*0.7
>>> m,p = slab.magPhase(g)
>>> print m,p
0.989949493661 45.0
```

mag(value)

Arguments	value	Float or numpy array of real or complex numbers	V1
Returns		Absolute value (magnitude)	

Auxiliary

Take one complex or real value or a numpy array of values and generates one float or one array that include the absolute value of each input element.

phase(value)

Arguments	value	Float or numpy array of real or complex numbers	V1
Returns		Phase value (deg)	

Auxiliary

Take one complex value or a numpy array of values and generates one float or one array that include the phase value, in degrees, of each input element.

References

STM32 Nucleo Page
http://www.st.com/en/evaluation-tools/stm32-mcu-nucleo.html

MBED Developer Page
https://developer.mbed.org/

Python page:
https://www.python.org/

Anaconda page:
https://www.continuum.io/downloads

TinyCad
https://sourceforge.net/projects/tinycad/
Circuit images on this document have been drawn using the free software TinyCad