Python Power Electronics

A free and open source circuit simulator for power electronics and power systems professionals

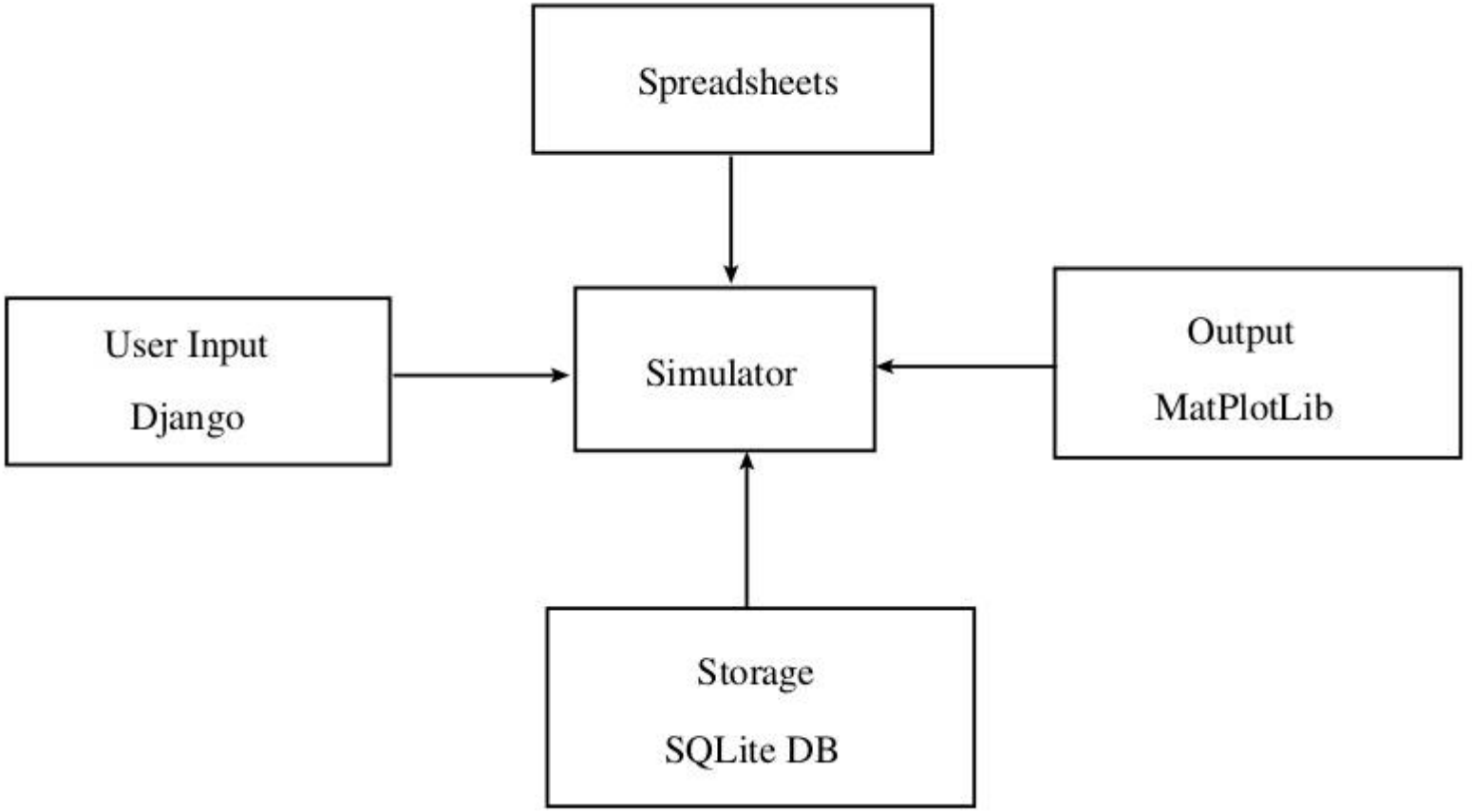
by

Shivkumar Iyer

Overview

- Simulators for power electronics are very expensive and have restrictive licensing.
- Most simulators are designed for small circuits and do not function well for multi converter interconnected systems.
- The major challenge in the future will be simulating smart grids.
- Specialized tools are needed for advanced power systems.

Components of the simulator



Components of the simulator (contd)

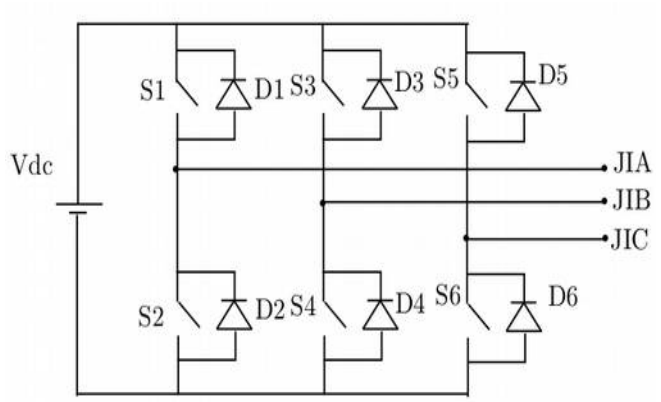
Simulator

- Written entirely in Python
- Uses network analysis and solves differential equations
- Details can be found in my book “Simulating non-linear circuits with Python Power Electronics: an open source simulator based on Python” - available at Gumroad: https://gumroad.com/l/lYQK#

User interface

- All user input except for the circuit schematics are from a web application.
- Django is used to develop the web application
  - Easy use of forms to get data from the user along with error checking and feedback.
  - Using data to create Python objects which then create database entries.
  - Conveniently manage backend databases in SQLite, MySQL or others.

Circuit schematics



Spreadsheets saved as .csv files are the most convenient way to “draw” a circuit but still results in the data being available as text.

File Edit View Insert Format Tools Data Window Help

Liberation Sans 10

A1 wire

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	wire	wire	wire	wire	wire	wire	wire	wire	wire	wire	wire	wire	wire	wire				
2	wire		wire			wire				wire				wire				
3	wire		wire			wire	wire	wire		wire	wire	wire		wire	wire	wire		
4	wire		wire			wire		wire		wire		wire		wire		wire		
5	wire		Resistor_Cdcinv1			wire		wire		wire		wire		wire		wire		
6	wire		wire			Switch_S1inv1		Diode_D1inv1		Switch_S3inv1		Diode_D3inv1		Switch_S5inv1		Diode_D5inv1		
7	wire		wire			wire		wire		wire		wire		wire		wire		
8	wire		wire			wire		wire		wire		wire		wire		wire		
9	wire		VoltageSource_Cdcinv1			wire	wire	wire		wire	wire	wire		wire	wire	wire		
10	wire		wire			wire				wire				wire				
11	Voltmeter_Vdcinv1		wire			wire				wire				wire				
12	wire		wire			wire	wire	jumpinv1pha		wire	wire	jumpinv1phb		wire	wire	wire	jumpinv1phc	
13	wire		wire			wire				wire				wire				
14	wire		wire			wire				wire				wire				
15	wire		wire			wire	wire	wire		wire	wire	wire		wire	wire	wire		
16	wire		wire			wire		wire		wire		wire		wire		wire		
17	wire		wire			Switch_S2inv1		wire		Switch_S4inv1		wire		Switch_S6inv1		wire		
18	wire		wire			wire		Diode_D2inv1		wire		Diode_D4inv1		wire		Diode_D6inv1		
19	wire		wire			wire		wire		wire		wire		wire		wire		
20	wire		wire			wire	wire	wire		wire	wire	wire		wire	wire	wire		
21	wire		wire			wire				wire				wire				
22	wire	wire	wire	wire	wire	wire	wire	wire	wire	wire	wire	wire	wire	wire				
23																		

Extracting components

- A component appears in the spreadsheet as follows

  Switch_S1inv1

- Essentially -->

  Component type + _ + Component name

- Component type has to be from the existing library – Resistor, Inductor, VariableResistor, VariableInductor, Capacitor, VoltageSource, ControllableVoltageSource, Diode, Switch

Extracting components (contd)

- We use Python's object oriented structutre
- A class is defined for every component type.
- The class has data (parameters) and functions (methods).
- All classes have the same functions – so Resistor class and Capacitor class will have same functions.
- The data within a class is different – the functions operate on data to produce different results.

Component Class

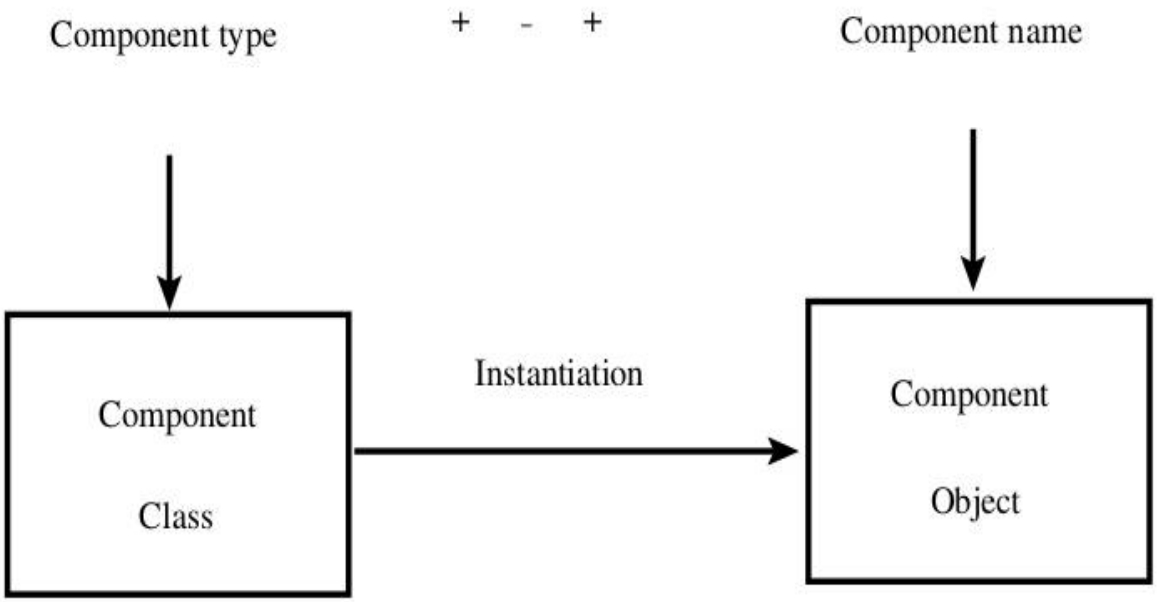
Look at the class definitions for Resistor and VoltageSource – parameters are different but they have the same function X. But the function performs completely different tasks.

```
class Resistor:
    Resistor = 100
    def X(self):
        self.output = self.Resistor
```

```
class VoltageSource:
    Mag = 100
    Freq = 50
    def X(self):
        self.output = self.Mag*sin(2*pi*50)
```

Same function, different behaviour

Component objects

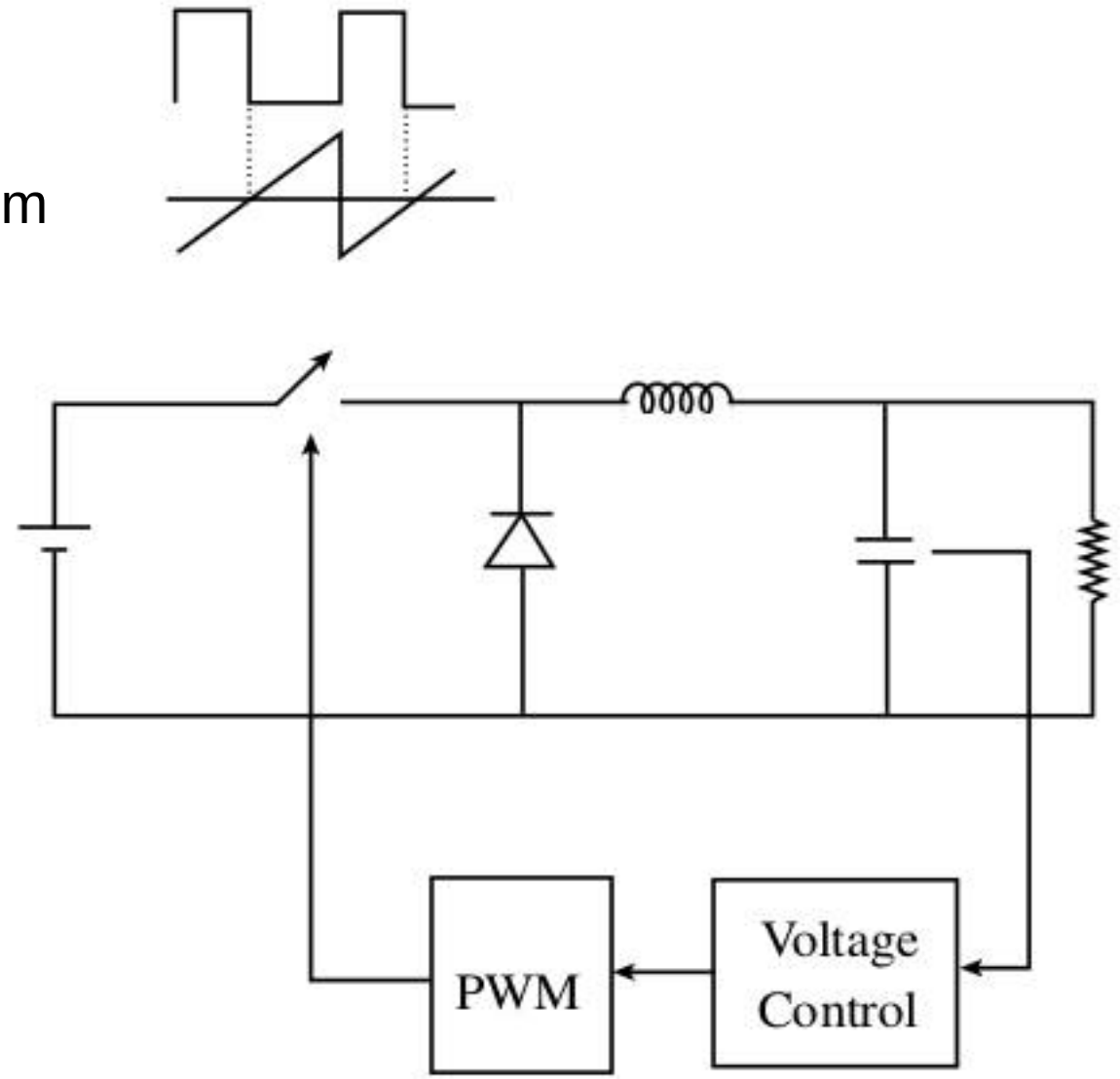


Every component in the circuit schematic will have an object created by instantiating a class of the component with the unique data related to the component such as name and cell position.

Sample circuit – buck converter

On time of switch changed

Sawtooth waveform sets switching frequency



Control strategy

- In Python Power Electronics, control code can be included in Python files.
- To know more about control, read the chapter on the website: http://pythonpowerelectronics.com/papers/chap4.pdf
- Every control file needs to have an input/output interface.
- Also, every control file can have special variables.

Voltage control

```
kp = 0.01
ki = 1.0
dt_sample = 200.0e-6
if t_clock>=t1:
    volt_err = 100 – voutput
    err_integ += volt_err*dt_sample
    duty_ratio = ki*err_integ + kp*volt_err
    t1 += dt_sample
buck_duty_ratio = duty_ratio
```

Control gains

Switching period

Control variables - input

- voutput is an input variable.
- An input variable specifies a measurement – in this case it is the measurement of the capacitor voltage.
- The measurement is then made available through the variable.
- So when voutput is used in code, it automatically connects to the measured voltage across the capacitor

Control variables – static variables

- In Python, there is no need to declare variables like in C/C++.
- But in a function, like a control function, a variable is local to the function. It is created when function is called and destroyed when exited.
- Static variable stores values of control. So err_integ is a static variable. At the end of the function, it is not destroyed. The value is stored and made available at the next iteration.

Control variables - VariableStorage

- Voltage Control and PWM blocks are connected together.
- Voltage Control block generates duty ratio.
- PWM block uses duty ratio to generate switching signal.
- The communication is through VariableStorage.
- buck_duty_ratio is a VariableStorage element. It is a global variable. All control files have access to this variable.
- So PWM can access this as will be shown next.

Control variables – Time event

- A control code runs at a sampling frequency.
- This sampling frequency is a part of the design and must be controlled.
- In DSP, this would be through Interrupt Service Routines.
- In simulator, it is through Time Event.
- t1 is a Time Event.
- By changing t1, the simulator will run control code at that time t1.

PWM

```
wave_freq = 5000.0
dt_sample = 0.1e-6
if t_clock>=t1:
    x_tri += dt_sample*wave_freq
    if x_tri>1.0:
      x_tri = 0.0
    if buck_duty_ratio>x_tri:
      gate_signal = 1.0
    else:
      gate_signal = 0.0
    t1 += dt_sample
```

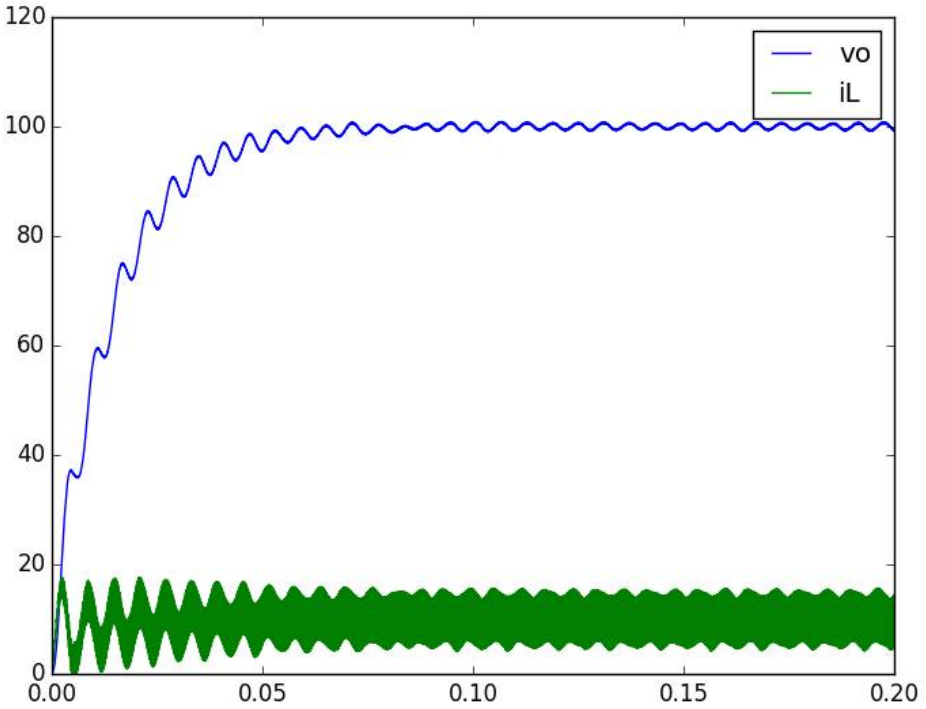
Sawtooth waveform

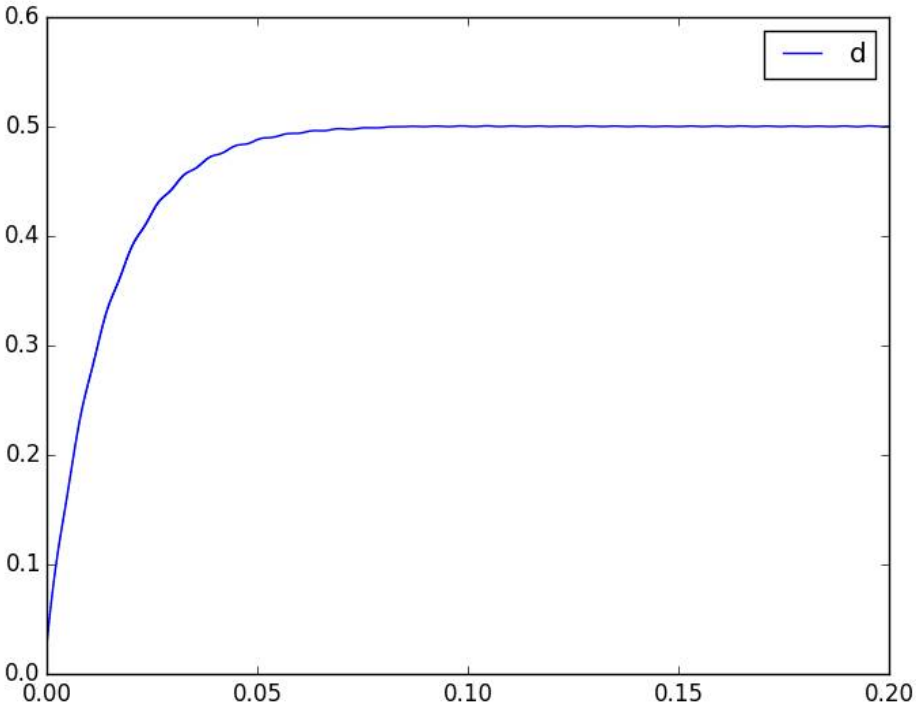
Variable storage is accessed here but generated in Voltage Control

Control variables – Output

- gate_signal is an output variable.
- An output variable lets you regulate a controllable element – Switch.
- When you change the output variable, you change the controllable element automatically.
- So by changing gate_signal between 1 and 0, the Switch is being turned on and off.

Simulation Result





Output regulated to 100V as desired and the duty ratio (input) is constant due to the controller settling in steady state.

Further resources

- Sample circuits in the link:
  - http://www.pythonpowerelectronics.com/blog.html
- Tutorials in the link:
  - http://pythonpowerelectronics.com/tutorials.html
- Documents and papers in the link:
  - http://pythonpowerelectronics.com/documentdownloads.html
- Software from the link:
  - http://pythonpowerelectronics.com/softwaredownloads.html
- For instant updates on the project:
  - https://www.facebook.com/pythonpowerelectronics

Thank
you

Questions
and
comments