Software Development for Circuit Analysis and Solving Documentation

Release

Carmen Legarreta

Jun 21, 2019

CONTENTS

The aim of this project is provide a free python program to solve circuits.

CHAPTER

ONE

CIRCUIT

The class Circuit has been made to represent, analysed, and solve circuits. This class provides some solving modes.

1.1 Documentation

This script contains the Circuit class. It allows the user to save the parameters of the circuit, such as; elements and each nodes. Once the circuit is defined, the user can get tableau equations, voltage solution, current solution....

class circuito.**Circuit**(*circuit*)

Bases: object

A class used to represent a circuit.

circuit: list It is a list, with elements and lists, which represents a circuit. circuit parameter must have the following structure; [[r1, [0, 1]], [r2, [1, 2]], ..., [V, [n, 0]]] [r1, [0, 1]]–>r1 resistive element connected to 0 and 1 nodes. The current will pass from 0 node to 1. [r2, [1, 2]]–>r1 resistive element connected to 1 and 2 nodes. The current will pass from 1 node to 2. . . .

variables: list It is a list with circuit variables in a symbolic form.

nl: boolean It determinates if the circuit is linear, or non-lineal, that it: if nl is True the circuit will be non-lineal, and if it False, lineal.

add(n): It adds a element/group of elements, defined by the input n,in the circuit attribute.

reset(circuit): It erases the defined circuit attribute and introduces another one.

get(): when this method is called, the user will get the objectÂ´s circuit attribute value.

getTableua(): when this method is called, the user will get the matrix representation of the tableau equations.

solution(position, port, variable, initcondition): It returns the solution of the circuit.

dcAnalysis(position, ports, variable, dcelement, t0, t1, steps): It makes a graph which shows the change of a element variable, as a VoltageSource type element changes its value

timeAnalysis(position, ports, variable, t0, t1, steps): It makes a graph which shows the change of a element variable, as the time change.

theveninEquivalent(node1, node2): It returns the thevenin equivalent of the circuit.

NortonEquivalent(node1, node2): It return the Norton equivalent of the circuit.

InputError: If the Circuit object initiate argument structure is not what supposed to be. The structure it must follow is described in the Parameters section. If the Circuit initiate argument does not meet the khirchoffs law criteria.

Circuit 0 node represents ground.

add(*n*)

This method adds elements, with each respective node notation. The adding element must have the same structure as the initiate arguments; [[element1,[node1, node2], [node3, node4],...], [element2, [node1Â´, node2Â´], [node3Â´, node4Â´],...],...]

n: list

copy()

It return a copy of this object.

dcAnalysis(*position*, *ports*, *variables*, *dcelement*, *v0*, *vi*, *steps*)

This function allows the user to see a circuit variable change as the VoltageSource type element, defined with the input variavle dcelement value change from v0, to vi.

position: list This parameter must contain integers which the position of the wanted elements to be analysed.

ports: list Specify from which ports will bet get the variable information, respectively.

variables: list the variable to be studied. There are four options; voltage, current, node1, node2. These variables are related to the Element type object.

dcelement: list It must be a VoltageSource Type element with each nodes

v0: float Voltage at which the analysis starts.

vi: float Voltage at which the analysis finishes.

steps: int number of steps from v0 to vi.

It is possible to use this function in circuits with capacitors and inductor. The step size will be 1e-5. However, elements with t dependency are not allowed.

InputError If the element parameter is not in the circuit

get()

It returns it's circuit attribute.

getTableau()

This method returns a tuple with two list. The first one holds tableu for currents, voltages, and tableau for elements. The second one has the values of the excited sources.

t, u: tuple t: list u: list

nortonEquivalent(*nodo1*, *nodo2*)

It calculates the Norton equivalent of the circuit.

nodo1: int It is equivalent to the positive node, node+

nodo2: int It is equivalent to the negative node, node-

reset(*circuit=[]*)

When reset method is called, the defined circuit attribute will be replaced by the input parameter. If there is no input parameter, then the circuit attribute will be defined by a default variable.

circuit: list It represents a circuit that will replaced the already defined circuit attribute.

solution (*position=None*, *port=None*, *variable=None*, *initcondition=None*)
It returns a dictionary which keys are the circuit variables: i_1, i_2,..., e_1, e_2,... . The values of the keys are the solutions of the circuit.

position: list This parameter must contain integers which the position , from 0 to the number of elements-1, of the wanted elements to be analysed.

port: list Specify from which ports will bet get the variable information, respectively.

variable: list The variable to be studied. There are four options; voltage, current, node1, node2. These variables are related to the Element type object.

initcondition: dictionary A dictionary which keys are the circuit variables, and each values are the initial conditions.

theveninEquivalent (*nodo1*, *nodo2*)
It calculates the Thevenin equivalent of the circuit.

nodo1: int It is equivalent to the positive node, node+

nodo2: int It is equivalent to the negative node, node-

timeAnalysis (*position*, *ports*, *variable*, *t0*, *t1*, *steps*)
This function allows the user to see a circuit variable change as the time changes.

position: list This parameter must contain integers which the position of the wanted elements to be analysed.

ports: list Specify from which ports will bet get the variable information, respectively.

variable: list the variable to be studied. There are four options; voltage, current, node1, node2. These variables are related to the Element type object.

t0: float Time at which the analysis starts.

t1: float Time at which the analysis finishes.

steps: int number of steps from t0 to t1.

InputError If the element parameter is not in the circuit

CHAPTER

TWO

ELEMENTS

The Elements class represents differents elements, all of them have terminals, equations to represent the physical behaviour, etc.

2.1 Documentation

class elementos.**ACCurrentSource**(*value*, *frequency*, *phase=0*)

Bases: *elementos.Elements*

A class used to represent a AC voltage source.

value: float The amplitude of the AC current source..

frequency: float The frequency of the AC current source.

phase: float The phase of the AC current source

equation: list The representation of the voltage source constitutive equation.

class elementos.**ACVoltageSource**(*value*, *frequency*, *phase=0*)

Bases: *elementos.Elements*

A class used to represent a AC voltage source.

value: float The amplitude of the AC voltage source..

frequency: float The frequency of the AC voltage source.

phase: float The phase of the AC voltage source

equation: list The representation of the voltage source constitutive equation.

class elementos.**CCCS**(*u*)

Bases: *elementos.Hybrid2*

A class used to represent a current controlled current source.

u: float The relation between the first and second port current when the first portÂ´s voltage is 0 V.

class elementos.**CCVS**(*r*)

Bases: *elementos.CurrentControlled*

A class used to represent a current controlled voltage source element.

r: float The impedance value of the voltage source controlled by a current.

class elementos.**Capacitor**(*value*, *v0=0.0*)
Bases: *elementos.Elements*

A class used to represent a capacitor.

value: float The capacitance value of the capacitor.

initcondition: float Determinates the voltage value across the capacitor when t=0.

equation: list The representation of the capacitor constitutive equation.

vk(*value*)

class elementos.**CurrentControlled**(*r11*, *r12*, *r21*, *r22*)
Bases: *elementos.Elements*

A class used to represent a current controlled element.

r11: float The impedance value of the first port when the second port current is 0 A, open port.

r12: float The impedance value of the first port when the first port current is 0 A.

r21: float The impedance value of the second port when the second port current is 0 A.

r22: float The impedance value of the second port when the first port current is 0 A.

equation: list The representation of the current controlled constitutive equation.

class elementos.**CurrentSource**(*value*)
Bases: *elementos.Elements*

A class used to represent a current source.

value: float The impedance value of the resistance.

equation: list The representation of the current source constitutive equation.

class elementos.**D1N4002**(*i0=1.411e-08, v=[0.4], i=[0.001]*)
Bases: *elementos.Elements*

A class used to represent a diode.

equation: list The representation of a diode constitutive equation.

v: list It is a list with the initial values.

getDiscrete()
This method return True when the equation has been linealized.

setDiscrete(*dis*)
The input value determinates wheter the element equation is going to be linealized or not.

dis: boolean True to make the equation linear. False, otherwise.

setParamValues(*v=None*, *i=None*)
To linearization a set of values are needed. So, this method takes each input arguments, and it uses to make the linealization and change the equation.

v: list v is a list with the voltage values of the elements port.

i: list i is a list with the current values of the elements port

class elementos.**Elements**(*equation=[]*, *differential=False*, *difValue=None*, *nl=False*, *tvariant=False*, *subcir=False*)

Bases: object

A class used to represent a set of electric elements.

equation: list The representation of the constitutive equations of the element.

equation: list The representation of the constitutive equations of the element.

current(terminal): It returns the way in which goes the current in the terminal, input value.

nEquations(): It returns the number of constitutive equations of the element.

nPorts(): It returns the number of ports of the element.

VoltageValue(equation, port): It takes a equation, specified by the input parameter, of the constitutive equations and it will return the value that is within the specified port voltage.

CurrentValue(equation, port): It takes a equation, specified by the input parameter, of the constitutive equations and it will return the value that is within the specified port current.

UValue(equation): It takes a equation, specified by the input parameter, of the constitutive equations and it will return the u value.

setEquation(equation): It changes the element equation with the input argument.

getEquation(): It return the element equation.

setuValue(equation, value): The u value of the equation is changed with the value input.

getDif(): It determinates if the equation has any differentiation.

getDifValue(): It specifies which variable, v or i, has been differentiate.

getNL(): It specifies whether the element is non-linear, or not.

setNL(value): It allows the user to specify the element non-linearity.

getTvariant(): It determinates wheter the element has a time dependency.

setTvariant(value=True): This method allows the user to determinate whether the elements is timer variant or not.

getSubcir(value): It allows the user to determinate whether the element is a sub-circuit or not.

getDiscret(): It determinates whether the element equation has beenlinearized.

getCir(): It returns the solution of the inner circuit of a SubCircuit type object.

setCir(cir): It gets the solution of the inner circuit of a SubCircuit type object

current(*terminal*)

It returns the way in which goes the current in the terminal, input value.

terminal: int The number of the terminal. It will take 0 or 1 value.

int 1 if the terminal number is 0, -1 otherwise.

currentValue(*equation*, *port*)

It takes a equation, specified by the input parameter, of the constitutive equations and it will return the value that is within the specified port current.

equation: int It is a number that specifies the equation that the user wants to take from the set of constitutive equations.

port: int It is a number that specifies the port from which the user wats to get current information.

float The parameter value multiplying with the current variable, specified by equation and port input parameters.

getCir()
It returns the solution of the inner circuit of a SubCircuit type object.

getDif()
It determinates if the equation has any differentiation.

boolean True if the equation has any differentiation. False, otherwise

getDifValue()
It specifies which variable, v or i, has been differentiate.

getDiscrete()
It determinates whether the element equation has been linearized.

getEquation()
It return the element equation.

getNL()
It specifies whether the element is non-linear, or not.

getSubcir()
This method determinates wheter the element is a SubCircuit type objetc

getTvariant()
It determinates whether the element has a time dependency.

nEquations()
It returns the number of constitutive equations of the element.

int number of constitutive equations of the element.

nPorts()
It returns the number of ports of the element.

int number of ports of the element.

setCir(*cir*)
It gets the solution of the inner circuit of a SubCircuit type object

cir: dictionary the keys are equal to the circuit variables, and the dictionary values are the solution of the circuit.

setEquation(*equation*)
It changes the element equation with the input argument.

equation: list A representation of the constitutive equation with lists.

setNL(*value*)
It allows the user to specify the element non-linearity.

value: boolean True to became non-linear. False, to linear.

setSubcir(*value=True*)
It allows the user to determinate whether the element is a sub-circuit or not.

value: boolean True to sub-circuits. False, otherwise

setTvariant(*value=True*)
This method allows the user to determinate wheter the elements is timer variant or not.

value: boolean True to time variant element. False, otherwise.

setuValue(*equation*, *value*)
The u value of the equation is changed with the value input.

equation: int It is a number that specifies the equation that the user want to take to change each u value

value: float It is a float number, and it will replace the u value of the specified equation

uValue(*equation*)
It takes a equation, specified by the input parameter, of the constitutive equations and it will return the u value.

equation: int It is a number that specifies the equation that the user wants to take from the set of constitutive equations.

float The u value of the equation.

voltageValue(*equation*, *port*)
It takes a equation, specified by the input parameter, of the constitutive equations and it will return the value that is within the specified port voltage.

equation: int It is a number that specifies the equation that the user wants to take from the set of constitutive equations.

port: int It is a number that specifies the port from which the user wats to get voltage information.

float The parameter value multiplying with the voltage variable, specified by equation and port input parameters.

class elementos.**Gyrator**(*g*)
Bases: *elementos.VoltageControlled*

A class used to represent a gyrator.

g: float The admittance value of the gyrator

class elementos.**Hybrid1**(*h11*, *h12*, *h21*, *h22*)
Bases: *elementos.Elements*

A class used to represent Hybrid circuit.

h11: float The impedance value of the first port when the second port voltage is 0 V.

h12: float The relation between the first and second port voltages when the first portÂ´s current is 0 A.

h21: float The relation between the first and second por currents when the second portÂ´s voltage is 0 V.

h22: float The admittance value of the second port when the first port current is 0 A.

equation: list The representation of the Hybrid1 constitutive equation.

class elementos.**Hybrid2**(*h11*, *h12*, *h21*, *h22*)
Bases: *elementos.Elements*

A class used to represent a inverse Hybrid circuit.

h11: float The admittance value of the first port when the second port current is 0 A.

h12: float The relation between the first and second port current when the first portÂ´s voltage is 0 V.

h21: float The relation between the second and first port voltage when the second port current is 0 A.

h22: float The admittance value of the second port when the second port current is 0 A.

equation: list The representation of the inverse Hybrid constitutive equation.

class elementos.**Inductor**(*value*, *i0=0*)
Bases: *elementos.Elements*

A class used to represent a inductor.

value: float The inductance value of the inductor.

initcondition: float Determinates the current value across the inductor when t=0.

equation: list The representation of the capacitor constitutive equation.

ik(*value*)

class elementos.**Q2N2222**(*ics=3.307157571149511e-15, ies=1.1403e-15, ar=0.8602961925565159, af=0.9943, v=[-0.7, -0.7], i=[0.001, 0.001]*)
Bases: *elementos.Elements*

A class used to represent a NPN transistor.

equation: list The representation of the NPN transistor constitutive equation.

getDiscrete()
This method return True when the equation has been linealized.

setDiscrete(*dis*)
The input value determinates wheter the element equation is going to be linealized or not.

dis: boolean True to make the equation linear. False, otherwise.

setParamValues(*v=None*, *i=None*)
To linearization a set of values are needed. So, this method takes each input arguments, and it uses to make the linealization and change the equation.

v: list v is a list with the voltage values of the elements port.

i: list i is a list with the current values of the elements port

class elementos.**Q2N3904**(*ics=5.135668043523316e-15, ies=6.734423e-15, ar=0.42749, af=0.9976, v=[-0.7, -0.7], i=[0.001, 0.001]*)
Bases: *elementos.Elements*

A class used to represent a NPN transistor.

equation: list The representation of the NPN transistor constitutive equation.

getDiscrete()
This method return True when the equation has been linealized.

setDiscrete(*dis*)
The input value determinates wheter the element equation is going to be linealized or not.

dis: boolean True to make the equation linear. False, otherwise.

setParamValues(*v=None*, *i=None*)
To linearization a set of values are needed. So, this method takes each input arguments, and it uses to make the linealization and change the equation.

v: list v is a list with the voltage values of the elements port.

i: list i is a list with the current values of the elements port

class elementos.**Q2N3906**(*ies=1.1537553902975205e-15, ics=1.3908311155608809e-15, ar=0.8278688524590164, af=0.994, v=[-0.7, -0.7], i=[-0.0001, -0.0001]*)

Bases: *elementos.Elements*

A class used to represent a NPN transistor.

equation: list The representation of the NPN transistor constitutive equation.

getDiscrete()

This method return True when the equation has been linealized.

setDiscrete(*dis*)

The input value determinates wheter the element equation is going to be linealized or not.

dis: boolean True to make the equation linear. False, otherwise.

setParamValues(*v=None*, *i=None*)

To linearization a set of values are needed. So, this method takes each input arguments, and it uses to make the linealization and change the equation.

v: list v is a list with the voltage values of the elements port.

i: list i is a list with the current values of the elements port

class elementos.**Resistance**(*value*)

Bases: *elementos.Elements*

A class used to represent a resistance.

value: float The value of the resistance.

equation: list The representation of the resistance constitutive equation.

class elementos.**Transformer**(*n*)

Bases: *elementos.Hybrid1*

A class used to represent a transformer.

n: float The numbers of turns in a winding.

class elementos.**Transmission1**(*t11*, *t12*, *t21*, *t22*)

Bases: *elementos.Elements*

A class used to represent a two port network.

t11: float The relation between the first and the second port voltages values when the second port current value is 0 A.

t12: float The impedance of the first port when the second port current flows in the opposite direction and its portÂ´s voltage is 0 V.

t21: float The admitance of the first port when the second port current is 0 A.

t22: float The relation between the firs and second port currents when the second portÂ´s currents flows in the opposite way and the itÂ´s voltage value is 0 V.

equation: list The representation of the transmission constitutive equation.

class elementos.**Transmission2**(*t11*, *t12*, *t21*, *t22*)

Bases: *elementos.Elements*

A class used to represent a two port network with inverse transmission parameters.

t11: float The relation between the second and first port voltages when the first port current value is 0 A.

t12: float The second port impedance value when the first port voltage is 0 V.

t21: float The second port admittance value when the first port current is 0 A.

t22: float The relation between the second and first port currents when first port voltage is 0 V.

equation: list The representation of the inverse transmission constitutive equation.

class elementos.**VCCS**(*g*)
Bases: *elementos.VoltageControlled*

A class used to represent a voltage controlled current source element.

g: float The admittance value of the current source controlled by a voltage.

class elementos.**VCVS**(*u*)
Bases: *elementos.Hybrid1*

A class used to represent a voltage controlled voltafe souce.

u: float The relation between the first and second port voltages when the first portÂ´s current is 0 A.

class elementos.**VoltageControlled**(*g11*, *g12*, *g21*, *g22*)
Bases: *elementos.Elements*

A class used to represent a voltage controlled element.

g11: float The admittance of the first port when the second portÂ´s voltage is 0 V.

g12: float The admittance of the first port when the first portÂ´s voltage is 0 V.

g21: float The admittance of the second port when the second portÂ´s voltage is 0 V.

g22: float The admittance of the second port when the first portÂ´s voltage is 0 V.

equation: list The representation of the voltage controlled constitutive equation.

class elementos.**VoltageSource**(*value*)
Bases: *elementos.Elements*

A class used to represent a voltage source.

value: float The excitation value of the source.

equation: list The representation of the voltage source constitutive equation.

elementos.**pi = 3.141592653589793**
==========Elements==========

CHAPTER

THREE

SUBCIRCUITS

The SubCircuit class is used to represent and implement differents integrated circuit, such as the Operational Amplifier.

3.1 Documentation

Created on Mon May 6 10:30:44 2019

@author: Carmen

class subcircuito.**SubCircuit**(*circuito*, *v*, *FT*, *nl=False*, *differential=False*, *cir=None*)

Bases: *elementos.Elements*

A class used to represent a set of subcircuits.

circuito: list It represents the inner circuit of the subcircuit

v: list A list with initial values of the subcircuit.

FT: list It determinates which are the input/output of the inner circuit.

nl: boolean True if the inner circuit is non-linear. False, otherwise.

differentia: boolean True if the inner circuit has any capacitor/inductor.

cir: dictionary The keys of the dictionary are the variables of the inner circuit, and the values are the solution of the circuit.

nl: boolean True if the inner circuit is non-linear. False, otherwise.

circuito: list It represents the inner circuit of the subcircuit

setDiscrete(): The input value determinates wheter the element equation is going to be linealized or not.

getCir(): It returns the solution of the inner circuit of a SubCircuit type object.

setCir(cir): It gets the solution of the inner circuit of a SubCircuit type object.

setParamValues(): This methods get values for linearization, and gets the equivalent equation.

setLinearEquations(): It recalculates the equations.

difElements(): It reuturns a list of elements with differential values on its equations.

setIVk(): It is a method for the application of the Euler method in capacitors, and inductors.

difElements()
It reuturns a list of elements with differential values on its equations.

getCir()
It returns the solution of the inner circuit of a SubCircuit type object.

setCir(*cir*)
It gets the solution of the inner circuit of a SubCircuit type object.

cir: dictionary The keys of the dictionary are the variables of the inner circuit and the values are the solutions of the circuit.

setDiscrete(*dis*)
The input value determinates wheter the element equation is going to be linealized or not.

setIVk(*V=None*, *I=None*)
It is a method for the application of the Euler method in capacitors, and inductors.

V: list V contains the voltage solutions of the ports.

I: list I contains the current solutions of the ports.

setLinearEquations()
This mehtod is only used for linear subcircuits. It is used to when a inner circuit component has a capacitor/inductor type element, that is: when those elements equations change, the equation must be recalculated.

setParamValues(*v=None*, *i=None*)
This methods get values for linearization, and gets the equivalent equation.

v: list v is a list with the voltage values of the elements port.

i: list i is a list with the current values of the elements port.

PYTHON MODULE INDEX