University of Belgrade

School of Electrical Engineering

Software for analyzing linear time invariant electrical circuits using Symbolic Python "SymPyCAP"

Mentor:

prof. dr Dejan Tošić

Full Professor

Students:

Bogdan Žunić, 2018/0239

Vladimir Janković, 2018/0121

Miloš Milošević, 2018/0445

Dunja Đukić, 2018/0558

Belgrade, February 2021.

Table of Content

1. Introduction

In the modern world, many problems are solved using software. Software programs are fast and reliable ways for solving problems and complicated analysis in the fields of mathematics, physics, as well as other problems like managing data bases, traffic control and many more.

Electric circuits are no exception, and there are many programs for solving simple and complicated circuits. Programs like Maxima, *Mathematica* and WolframAlpha are used for analyzing electric circuits and solving equations of the given circuit, or symbolic analysis in other words. There are programs like Hspice, LTspice, Micro-Cap 12 which are used for simulating and analyzing real electric circuits by creating a circuit using different elements (voltage, resistors, capacitors, amplifiers, …).

SymPyCAP (Symbolic Python Circuit Analysis Program) is a library for Python for symbolic analysis of linear time invariant electric circuits. SymPyCAP is based on SALECx, which is software created by professor Dejan Tošić, that runs using the program Maxima. This Python library uses SymPy which is a powerful Python library used for equation solving (polynomial, linear, nonlinear, differential, …), matrices, limits, geometry, combinatorics, graphics, and data analysis. SymPyCAP uses different functions from the SymPy library for making symbols with the Symbol() function, exponential expressions with the exp() function, trigonometry with the sin(), cos() functions and for solving the linear equations of circuits using the linsolve() function.

This documentation will explain how to set up and use the SymPyCAP library, how to initialize the form and parameters for different electric components used in the circuit and how to use the functions from the SymPyCAP library.

The document also contains a few important notes, which explain some constraints and proper use of the SymPyCAP library, to inform users about problems that can occur if they are not careful.

2. Installation guide

The first step of the installation guide is to install Python 3.8.6 (if the user has already installed Python and the required libraries listed on page 5, the user can skip this part). Python can be downloaded from its official site: https://www.python.org/downloads/release/python-386/. Once there, under the "Files" section, choose the right version based on the operating system and installation method. Here, an example will be shown using Windows executable installer.

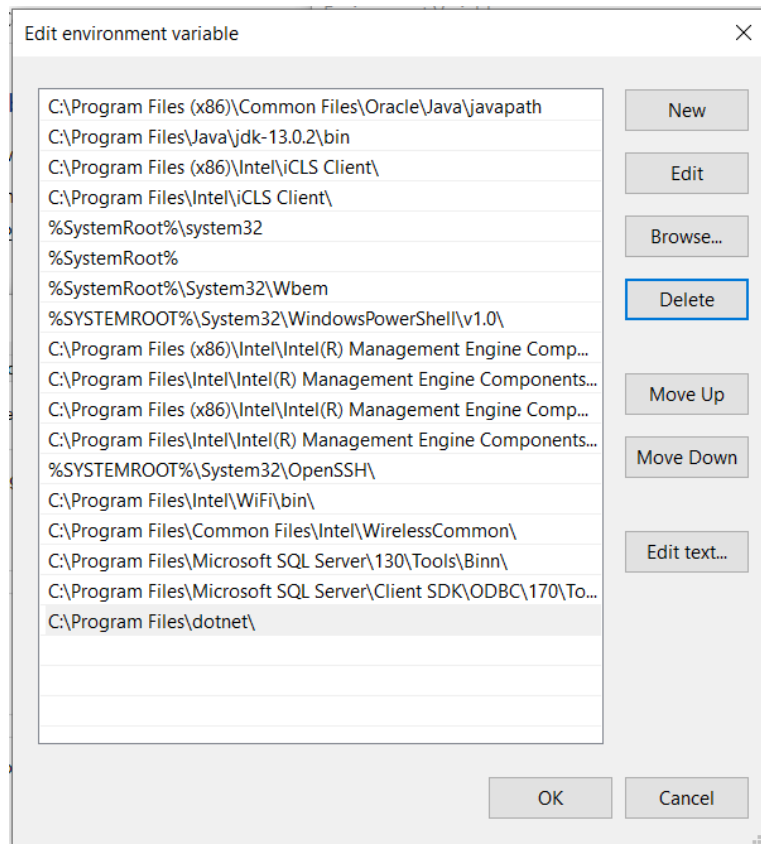Opening the downloaded installer results in this Python Setup window:
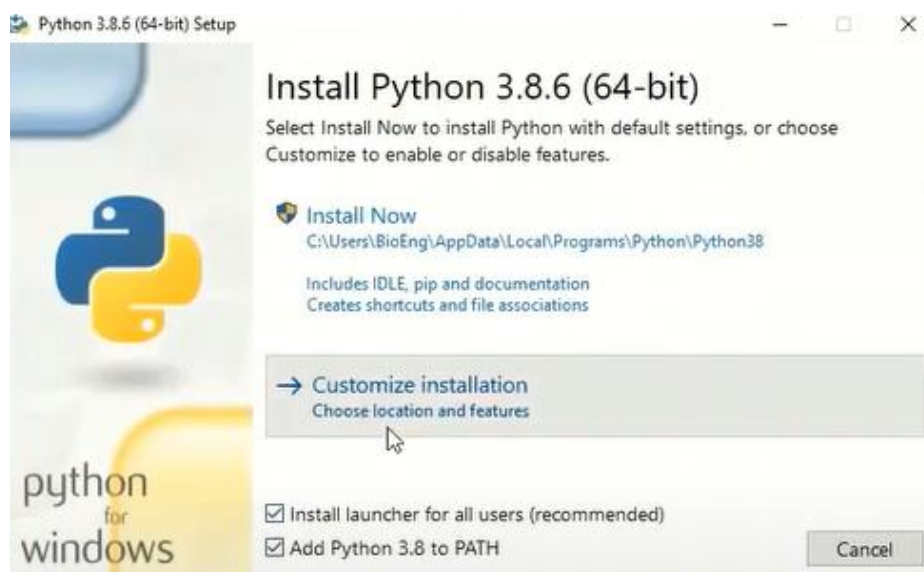
Selecting the custom installation option allows user to have more control over what is being installed and where.
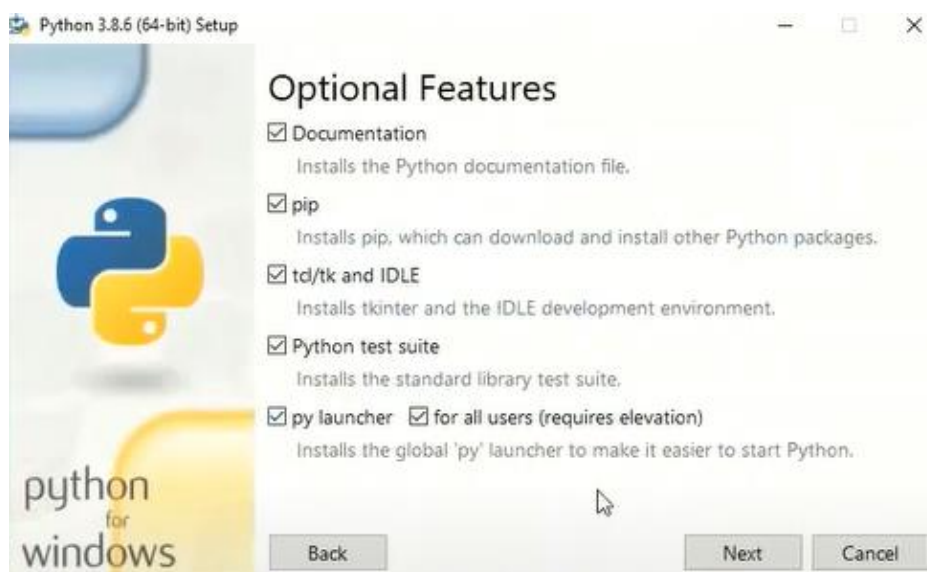

Out of the additional options that are available, it is recommended to install all of them, as most will be necessary like pip and IDLE.

After selecting optional features, a couple more options are available. Additionally, of all the non-checked options in the picture above, it is recommended to check the option "Precompile standard library". There is also the option to customize the location where Python will be installed using the "Customize install location" field. This location (referred to as *path_to_python* in the rest of the document) can be set arbitrarily or left on default option, however it is important to remember where the Python is installed as this information is needed in the future. All that is left now is to press Install button and wait until the installation is complete, after which the Setup window can be safely closed.

After installing Python, the system variable "Path" should be updated.

For Windows users, under Control Panel => System => Advanced system settings => Environment variables, one can find the System variables and one of them is "Path".

Editing the "Path" variable opens the following window:

Selecting the New button, a new text field will be created. In the first text field one should write the *path_to_python,* and then make a new text field which contains *path_to_python*/Scripts and then click OK until all windows are closed.
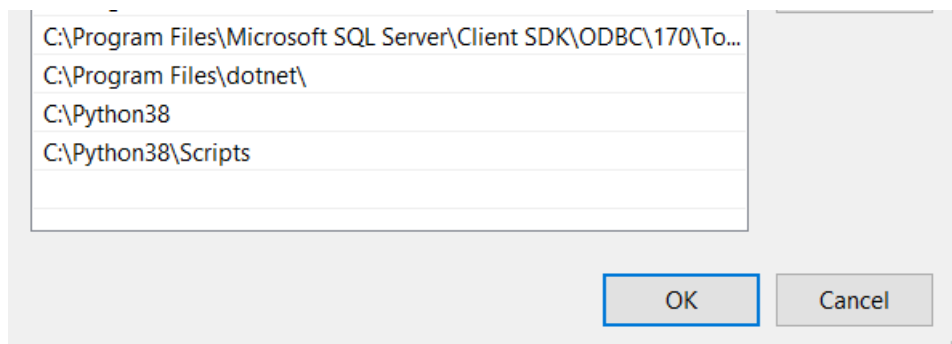
There are different ways for Linux/Mac OS to update the path variable using different scripts, but that will not be explained here, and we encourage to search for solutions online.

Once Python is installed, a couple of libraries should be installed using the pip tool. This requires opening a Command Prompt and executing the command: pip install *module*

Example: pip install matplotlib

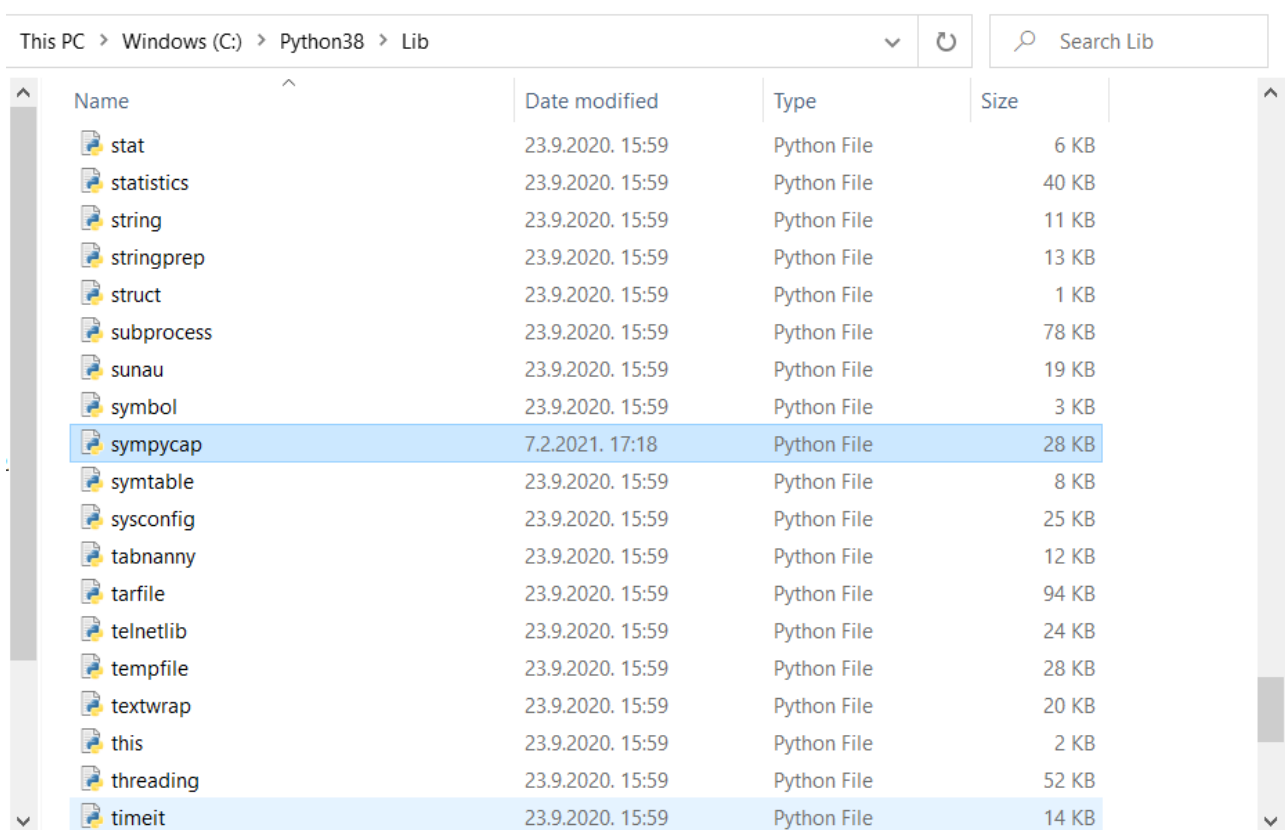The *module* represents which library the user wants to install. Libraries are as follows:

1. matplotlib
2. numpy
3. pyserial
4. vpython
5. drawnow
6. sympy

All of these will be needed for the SymPyCAP to function properly.

In order to use SymPyCAP, its library named ***sympycap.py*** must first be downloaded using the following link:

https://github.com/vladajankovic/SymPyCAP

After downloading the sympycap library, move it to the *path_to_python*\Lib folder where all .py libraries are placed (example: C:\Python38\Lib) and everything is ready to start using the sympycap library.

Name	Date modified	Type	Size
stat	23.9.2020. 15:59	Python File	6 KB
statistics	23.9.2020. 15:59	Python File	40 KB
string	23.9.2020. 15:59	Python File	11 KB
stringprep	23.9.2020. 15:59	Python File	13 KB
struct	23.9.2020. 15:59	Python File	1 KB
subprocess	23.9.2020. 15:59	Python File	78 KB
sunau	23.9.2020. 15:59	Python File	19 KB
symbol	23.9.2020. 15:59	Python File	3 KB
sympycap	7.2.2021. 17:18	Python File	28 KB
symtable	23.9.2020. 15:59	Python File	8 KB
sysconfig	23.9.2020. 15:59	Python File	25 KB
tabnanny	23.9.2020. 15:59	Python File	12 KB
tarfile	23.9.2020. 15:59	Python File	94 KB
telnetlib	23.9.2020. 15:59	Python File	24 KB
tempfile	23.9.2020. 15:59	Python File	28 KB
textwrap	23.9.2020. 15:59	Python File	20 KB
this	23.9.2020. 15:59	Python File	2 KB
threading	23.9.2020. 15:59	Python File	52 KB
timeit	23.9.2020. 15:59	Python File	14 KB

This PC > Windows (C:) > Python38 > Lib   Search Lib

3. Using the function

To start using the library functionality first we need a Python file. One can be simply created by making a new text file and changing the extension of the .txt file to .py. After that simply opening that file with IDLE 3.8 will allow you to start working. (Right click on the .py file will give user the option to edit with IDLE.)
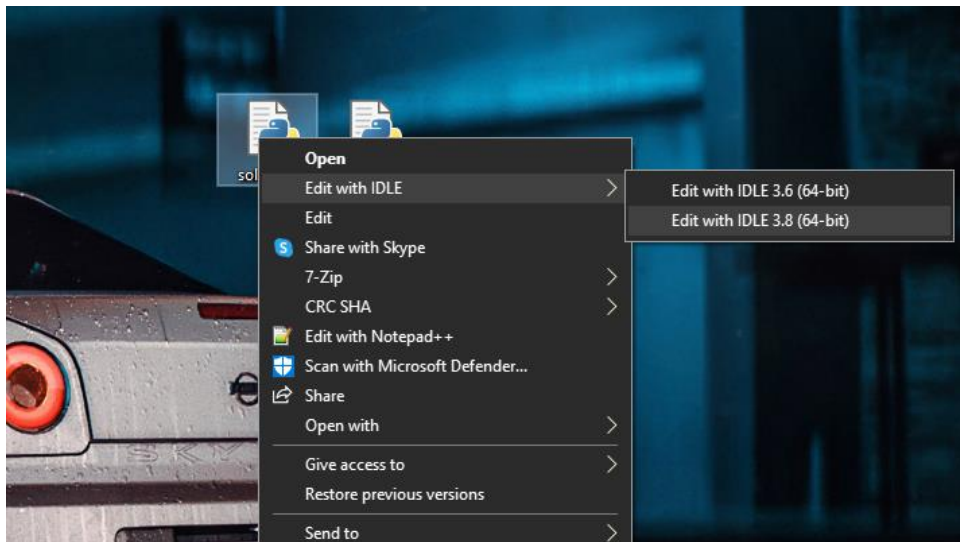
Another way of making a .py file is to launch IDLE from your desktop, start menu or source folder, which will open a Python shell. Under File => New File, an untitled .py file will be created which must later be saved somewhere.
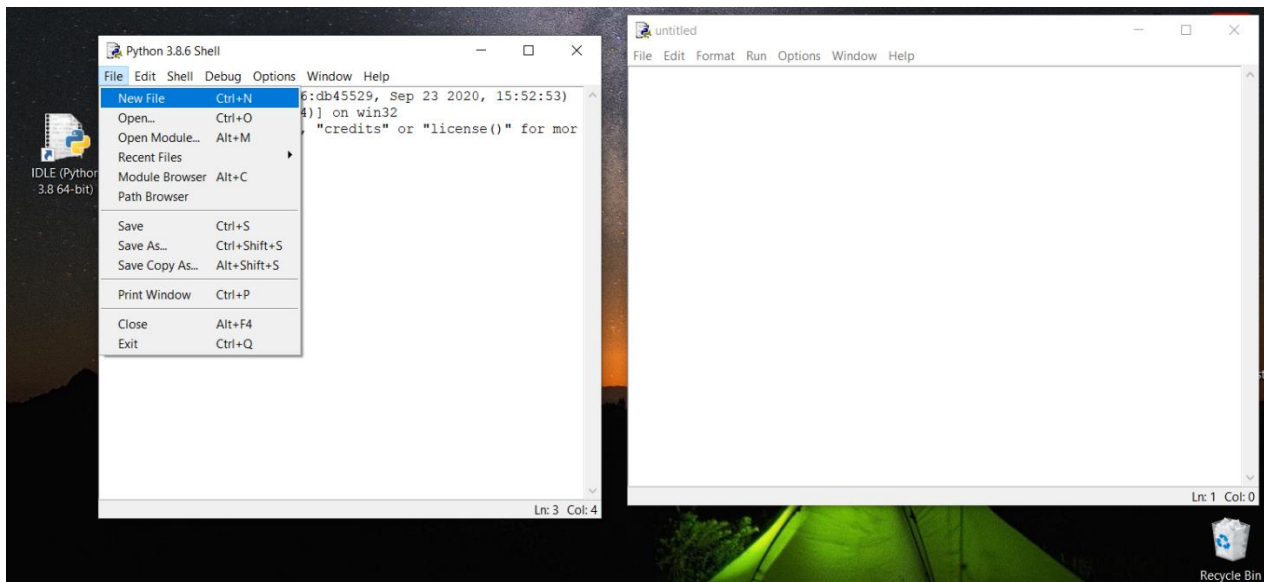
Once opened, the user needs to import the sympycap library, define the circuit using the format that will be presented next and call the function *runcap*.

```python
from sympycap import *

input =[4,
        ["UG", 3, 0, "E"],
        ["R", 1, 0, "R"],
        ["L", 3, 1, None, "L"],
        ["Z", 2, 0, "Z2"],
        ["Z", 3, 2, "Z1"]
]

runcap(input, False, True)
```

The first line imports all functionality of the sympycap library to this file. This is required to use the function *runcap*, as without this inclusion Python would not be able to find where the function source code is located.

Format of the circuit description - *input* (variable name, can be freely changed) is an array consisting of the number of nodes in the circuit (4 in this example) followed by the description of each element using the forms from the next section of this document.

IMPORTANT NOTE: Circuit elements that are connected in **SERIES** are represented as if they have a node between them. For example, between node A and B are 2 resistors and a voltage source that are connected in series and that implies that there are 4 nodes in total. One is A, one between the two resistors, one between the resistor and voltage source and one is B.

IMPORTANT NOTE: Nodes are numerated with consecutive integers starting from 0!! The reference node (ground) must exist and is designated by 0. Errors occur when a number is missing between the largest and smallest number (e.g. 0, 1, 2, 4, 6). That's why it is important to numerate the nodes appropriately without skipping a number. This also helps in figuring out the total number of nodes in the circuit. The total number of nodes will always be the largest node number +1.


The last thing is calling the function *runcap* itself by providing it with 3 arguments, first one being the circuit description. The second and third arguments are Boolean type (True, False), where the second represents if Phasors are used (primarily when using transmission lines in circuits) and the third one for Latex output of the results. By default, the second and third argument are set to False which implies that the function can be called only with the circuit description (without using phasors and no Latex display of the results).

Phasor value indicates whether the transmission lines should be solved using Phasors or the Laplace transform, while setting the Latex output to True opens a new window and displays the results in a much suitable way to read.

However due to some limitations in the way the text is processed a situation can occur where the equation cannot be displayed fully onscreen, and for that reason the result will always be printed in the console as the standard string of characters and it is up to the user to choose if they want the Latex output or not.

Also, if a user wants to change the value of one of these two arguments while leaving the other one at its default value, it can be done by simply calling the function like so:

runcap(input, Phasor=True) or *runcap(input, LatexOutput=True).*
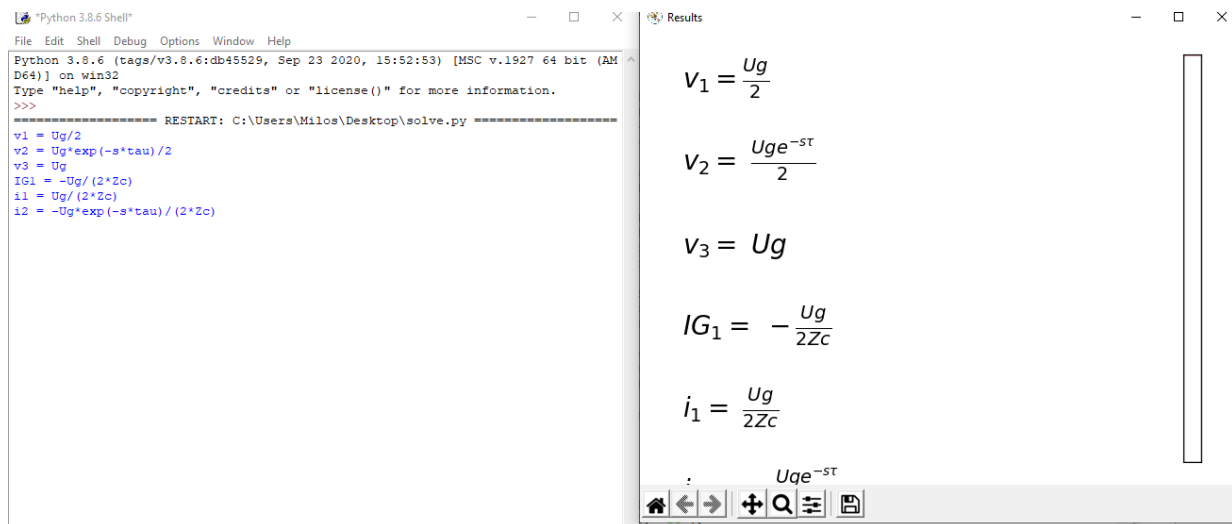
Here is the output of our example that shows how the regular output (blue text) and the Latex output (Results window) looks. Note that the bar on the right simulates scroll bar so that all the equations can be seen, but an equation could be complex enough to the point that the width of the window is not sufficient to display it fully and it would get cut off on the right edge of the window even when the plot is set to Fullscreen view.

4. Circuit elements

This section will contain information on how to write a circuit element correctly as an array of parameters, as well as what each parameter represents for the given element.

IMNORTANT NOTE: Since the value of an element is a symbol or symbolic expression, if writing "2R" and the value is squared/raised to a power, the output shows $2 \cdot R^2$, so it's sometimes better to write "(2R)".

- ## *Resistor*

  ["R", plusTerm, minusTerm, "value"]

  Example: ["R", 2, 3, "R1"]

  1. "R" – type of the element (string)
  2. plusTerm – plus terminal (int)
  3. minusTerm – minus terminal (int)
  4. "value" – symbolic value of the element (string)
     (example: "R1", "R2", …, "R", "2R", "3R/5", …)
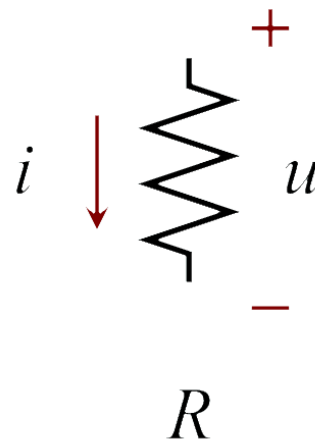
- ## *Inductor*

  ["L", plusTerm, minusTerm, "init_I", "value"]

  Example 1: ["L", 2, 3, "I0", "L2"]

  Example 2: ["L", 1, 4, None, "3L/4"]

  1. "L" – type of the element (string)
  2. plusTerm – plus terminal (int)
  3. minusTerm – minus terminal (int)
  4. "init_I" – initial condition for the current through the inductor:
     If there are no initial conditions => None
     If there are initial conditions => "I0", "I1", …, "I", "2I", … (string)
  5. "value" – symbolic value of the element (string)
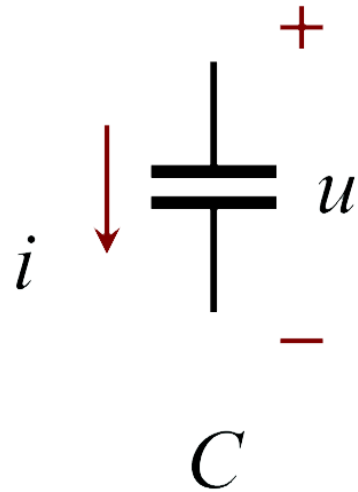   (example: "L1", "L2", …, "L", "2L", "L/5", …)
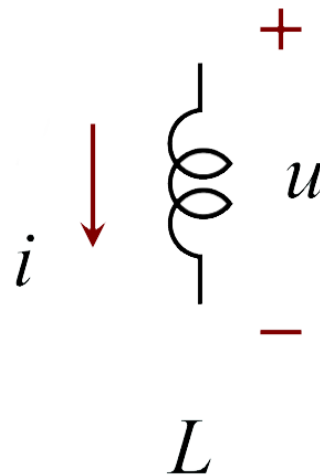
- *Capacitor*

["C", plusTerm, minusTerm, "init_U", "value"]

Example 1: ['C', 2, 3, "U0", "C1"]

Example 2: ['C', 1, 4, None, "2C"]

1. "C" – type of the element (string)
2. plusTerm – plus terminal (int)
3. minusTerm – minus terminal (int)
4. "init_U" – initial condition for the voltage through the capacitor:
   If there are no initial conditions => None
   If there are initial conditions => "U0", "U1", …, "U", "2U", … (string)
5. "value" – symbolic value of the element (string)
   (example: "C1", "C2", …, "C", "2C", "2C/3", …)

- *Current source*

["IG", in_node, out_node, "value"]
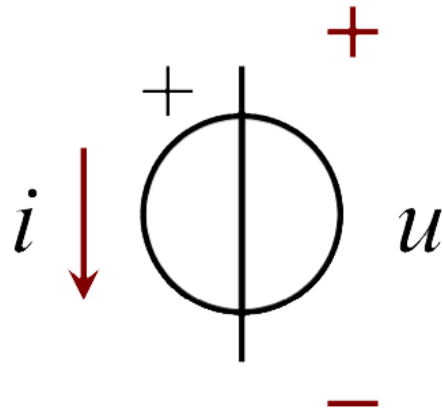
Example: ["IG", 2, 3, "i1"]
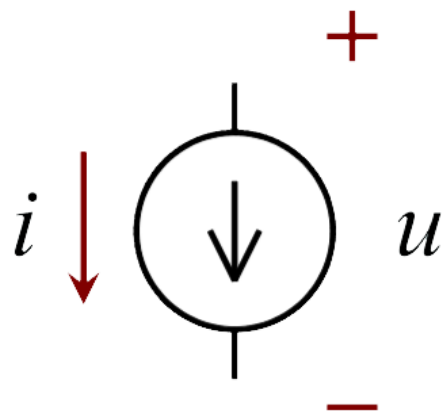
1. "IG" – type of the element (string)
2. in_node – node where the current enters, plus terminal (int)
3. out_node – node where the current exits, minus terminal (int)
4. "value" – symbolic value of the element (string)
   (example: "I1", "I2", …, "I", "2I", "I/5", …)

- *Voltage source*

  ["UG", plusTerm, minusTerm, "value"]
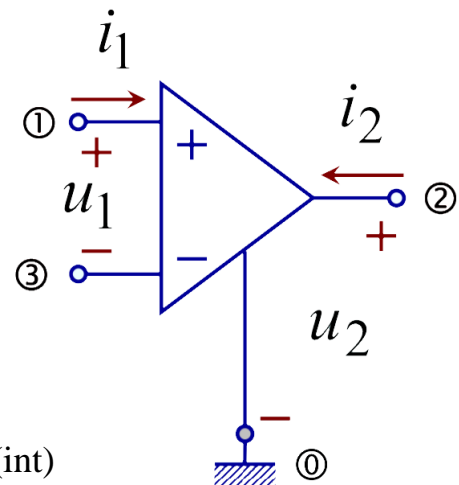
  Example: ["UG", 2, 3, "ug"]

  1. "UG" – type of the element (string)
  2. plusTerm – plus terminal (int)
  3. minusTerm – minus terminal (int)
  4. "value" – symbolic value of the element (string)
     (example: "ug", "E2", "U2", …, "U", "2U", "U/5", …)

- *Operational Amplifier - OpAmp*

  ["OpAmp", [inverting, non-inverting], out]

  Example: ["OpAmp", [3, 2], 4]

  1. "OpAmp" – type of the element (string)
  2. inverting – the inverting terminal (-) (int)
  3. non-inverting – the non-inverting terminal (+) (int)
  4. out – output node (+) (int)

  Note: The minus node (-) on the second terminal (u2) is implicitly
   connected to the GND (0, reference node) (the electric potential is 0)

• *Voltage Controlled Voltage Source - VCVS*

["VCVS", [plusTerm1, minusTerm1],

   [plusTerm2, minusTerm2], "a", "curr_I2"]

Example: ["VCVS", [2, 3], [4, 5], "3a", "I2"]

1. "VCVS" – type of the element (string)
2. plusTerm1 – plus terminal 1st port (int)
3. minusTerm1 – minus terminal 1st port (int)
4. plusTerm2 – plus terminal 2nd port (int)
5. minusTerm2 – minus terminal 2nd port (int)
6. "a" – VCVS parameter (string)
   (example: "a", "2a", "a/5", …, "1", "2.3", "3.5", …)
7. "curr_I2" – symbol for the I2 current (string)
   (example: "i2", "I2")
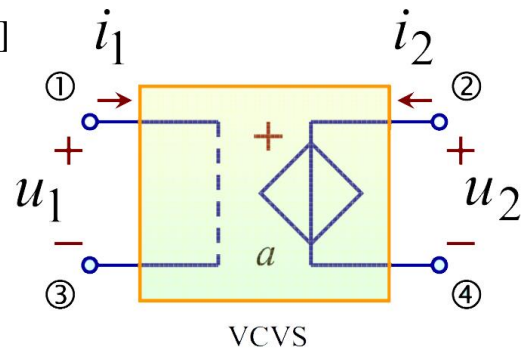
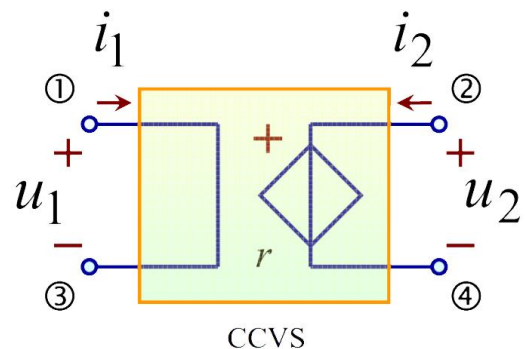$$u2 = a * u1$$
$$i1 = 0$$

• *Current Controlled Voltage Source - CCVS*

["CCVS", [plusTerm1, minusTerm1],

   [plusTerm2, minusTerm2], "r", "curr_I1"]

Example: ["CCVS", [2, 3], [4, 5], "r", "I1"]

1. "CCVS" – type of the element (string)
2. plusTerm1 – plus terminal 1st port (int)
3. minusTerm1 – minus terminal 1st port (int)
4. plusTerm2 – plus terminal 2nd port (int)
5. minusTerm2 – minus terminal 2nd port (int)
6. "r" – CCVS parameter (string)
   (example: "r", "2r", "r/5", …, "1", "2.3", "3.5", …)
7. "curr_I1" – symbol for the I1 current (string)
   (example: "i1", "I1")
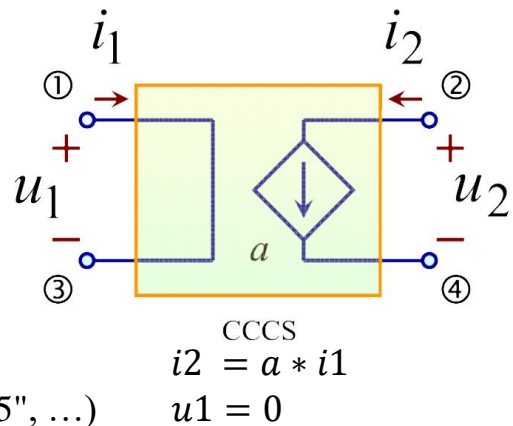
$$u2 = r * i1$$
$$u1 = 0$$

- ## *Current Controlled Current Source - CCCS*

  ["CCCS", [plusTerm1, minusTerm1],

     [plusTerm2, minusTerm2], "a", "curr_I1"]

  Example: ["CCCS", [2, 3], [4, 5], "2a", "I1"]

  CCCS
  $$i2 = a * i1$$
  $$u1 = 0$$

  1. "CCCS" – type of the element (string)
  2. plusTerm1 – plus terminal 1st port (int)
  3. minusTerm1 – minus terminal 1st port (int)
  4. plusTerm2 – plus terminal 2nd port (int)
  5. MinusTerm2 – minus terminal 2nd port (int)
  6. "a" – CCCS parameter (string)
     (example: "a", "2a", "a/5", …, "1", "2.3", "3.5", …)
  7. "curr_I1" – symbol for the I1 current (string)
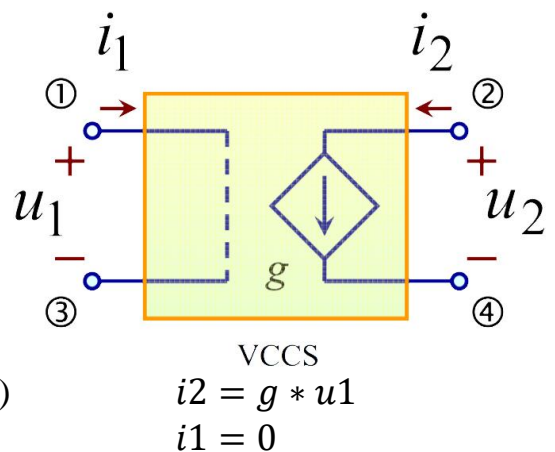   (example: "i1", "I1")

- ## *Voltage Controlled Current Source - VCCS*

  ["VCCS", [plusTerm1, minusTerm1],

     [plusTerm2, minusTerm2], "g"]

  Example: ["VCCS", [2, 3], [4, 5], "g"]

  VCCS
  $$i2 = g * u1$$
  $$i1 = 0$$

  1. "VCCS" – type of the element (string)
  2. plusTerm1 – plus terminal 1st port (int)
  3. minusTerm1 – minus terminal 1st port (int)
  4. plusTerm2 – plus terminal 2nd port (int)
  5. minusTerm2 – minus terminal 2nd port(int)
  6. "g" – VCCS parameter (string)
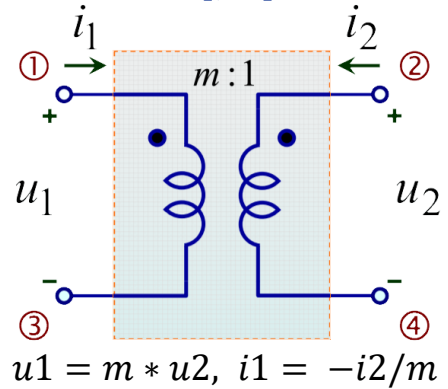   (example: "g", "2g", "g/5", …, "1", "2.3", "3.5", …)

- *Ideal Transformer*

  ["IT", [plusTerm1, minusTerm1], [plusTerm2, minusTerm2], m]

  Example: ["IT", [2, 3], [4, 5], 5]

  1. "IT" – type of the element (string)
  2. plusTerm1 – plus terminal 1st port (int)
  3. minusTerm1 – minus terminal 1st port (int)
  4. plusTerm2 – plus terminal 2nd port (int)
  5. minusTerm2 – minus terminal 2nd port (int)
  6. m – Ideal transformer parameter (string)

  $$u1 = m * u2, \quad i1 = -i2/m$$

- *Impedance*

  ["Z", plusTerm, minusTerm, "impedance"]

  Example: ["Z", 2, 5, "Z1"]

  1. "Z" – type of the element (string)
  2. plusTerm – plus terminal (int)
  3. minusTerm – minus terminal (int)
  4. "impedance" – symbolic value of the element (string)
     (example: "Z1", "Z2", …, "Z", "2Z", "3Z/5", …)

- *Admittance*

  ["Y", plusTerm, minusTerm, "admittance"]

  Example: ["Y", 2, 3, "Y2"]

  1. "Y" – type of the element (string)
  2. plusTerm – plus terminal (int)
  3. minusTerm – minus terminal (int)
  4. "admittance" – symbolic value of the element (string)
   (example: "Z1", "Z2", …, "Z", "2Z", "3Z/5", …)

- ## *Linear inductive transformer*

  ["K", [plusTerm1, minusTerm1], [plusTerm2, minusTerm2],

   ["L1", "L2", "L12"], ["initI_L1", "initI_L2"], ["curr_I1", "curr_I2"]]

  Example: ["K", [2, 3], [4, 5], ["L1", "L2", "L12"], [None, "I0"], ["i1", "i2"]]

  1. "K" – type of the element (string)
  2. plusTerm1 – plus terminal of the 1st port (int)
  3. minusTerm1 – minus terminal of the 1st port (int)
  4. PlusTerm2 – plus terminal of the 2nd port (int)
  5. minusTerm2 – minus terminal of the 2nd port (int)
  6. "L1" – value of the inductor L1 (string)
  7. "L2" – value of the inductor L2 (string)
  8. "L12" – value of the inductor L12(string)
  9. "initI_L1" – initial current in L1 (string) or None
  10. "initI_L2" – initial current in L2 (string) or None
  11. "curr_I1" – the symbol of i1 current (string)
  12. "curr_I2" – the symbol of the i2 current (string)

  $$L12 = k * \sqrt{L1 * L2}$$
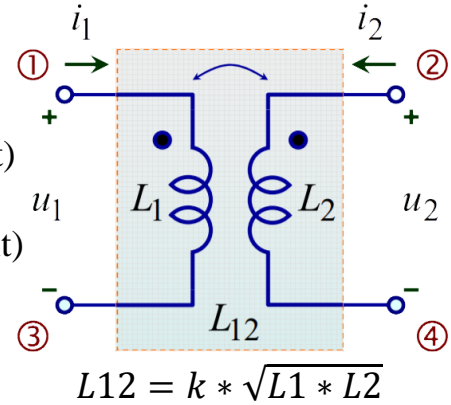
- ## *Transmission lines - the Phasor Transform analysis*

  ["T", [plusSending, minusSending], [plusReceiving, minusReceiving],

   ["curr_I1", "curr_I2"], ["Zc", "Theta"]]

  Example: ["T", [1, 3], [2, 4], ["i1", "i2"], ["Zc", "theta"]]

  1. "T" – type of element (string)
  2. plusSending – plus terminal of the 1st port (int)
  3. minusSending – minus terminal of the 1st port (int)
  4. plusReceiving – plus terminal of the 2nd port (int)
  5. minusReceiving – minus terminal of the 2nd port (int)
  6. "curr_I1" – the symbol of the i1 current (string)
  7. "curr_I2" – the symbol of the i2 current (string)
  8. "Zc" – characteristic impedance (string)
 9. "Theta" – electrical length (angle in radians or degrees) (string)

- *Transmission lines - the Laplace Transform analysis*

  ["T", [plusSend, minusSend], [plusReceive, minusReceive],

  ["curr_I1", "curr_I2"], ["Zc", "Tau"]]]

  Example: ["T", [1, 3], [2, 4], ["i1", "i2"], ["Zc", "tau"]]

1. "T" – type of element (string)
2. plusSend – plus terminal of the 1st port (int)
3. minusSend – minus terminal of the 1st port (int)
4. plusReceive – plus terminal of the 2nd port (int)
5. minusReceive – minus terminal of the 2nd port (int)
6. "curr_I1" – the symbol of the i1 current (string)
7. "curr_I2" – the symbol of the i2 current (string)
8. "Zc" – characteristic impedance (string)
9. "Tau" – transmission line delay (string)
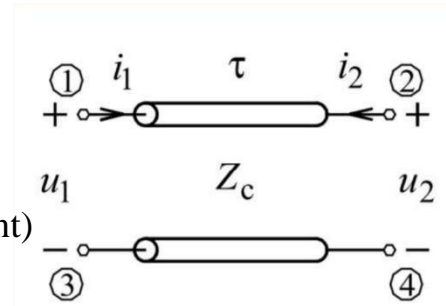
- *ABCD two-port*

  ["T", [plusTerm1, minusTerm1], [plusTerm2, minusTerm2],

  ["A", "B", "C", "D"], ["curr_I1", "curr_I2"]]

  Example: ["ABCD", [1, 3], [2, 4], ["A", "B", "C", "D"], ["I1", "I2"]]

1. "ABCD" – type of element (string)
2. plusTerm1 – plus terminal of the 1st port (int)
3. minusTerm1 – minus terminal of the 1st port (int)
4. plusTerm2 – plus terminal of the 2nd port (int)
5. minusTerm2 – minus terminal of the 2nd port (int)
6. "A", "B", "C", "D" – symbols or symbolic expressions (string)
7. "curr_I1" – the symbol of the i1 current (string)
8. "curr_I2" – the symbol of the i2 current (string)

$$U1 = A * U2 + B * (-I2)$$
$$I1 = C * U2 + D * (-I2)$$
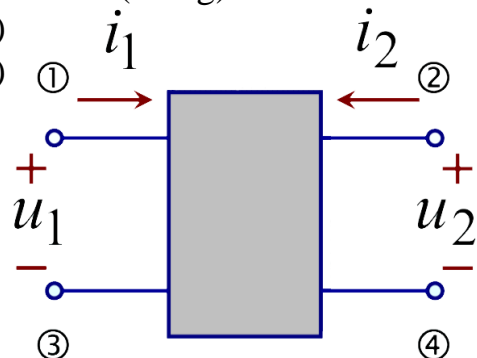
5. Examples

In this section we shall present some examples to familiarize the user with the various elements that can be used in the SymPyCAP function.

For the first couple of examples we will focus on the basic elements of any circuit: Resistors and ideal voltage and current generators.

In the below picture a basic circuit on the left and on the right is how we translate it into an input list for the program. The nodes are numbered from 0 to 5 (inclusive). In our input list we use one configuration of the node numbers but first time users are encouraged to change the configuration and mix and match with ideal current generators in order to familiarize themselves with the interface.
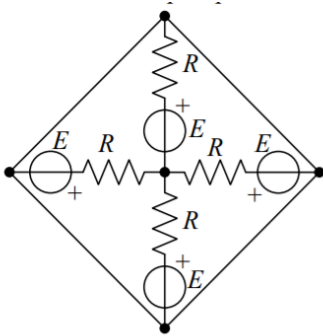

```python
from sympycap import *

input =[6,
        ["UG", 1, 0, "E"],
        ["R", 5, 1, "R"],
        ["UG", 2, 0, "E"],
        ["R", 5, 2, "R"],
        ["UG", 3, 0, "E"],
        ["R", 5, 3, "R"],
        ["UG", 4, 5, "E"],
        ["R", 0, 4, "R"]
]

runcap(input, False, True)
```
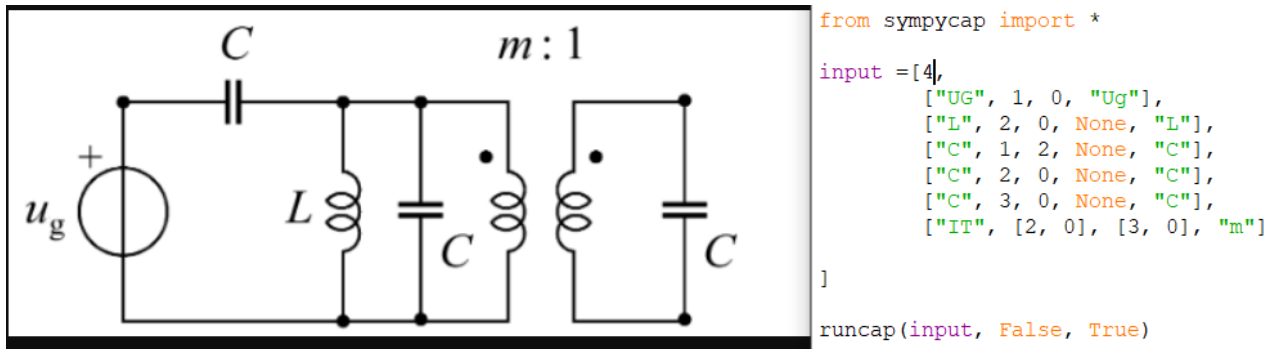
Example 1: OET1 exam 28. June 2019. Question 5.

The second example uses the same set of elements as the first but in a more complex circuit.
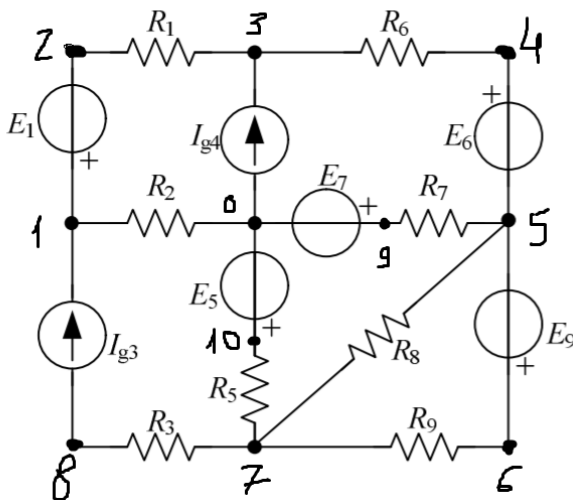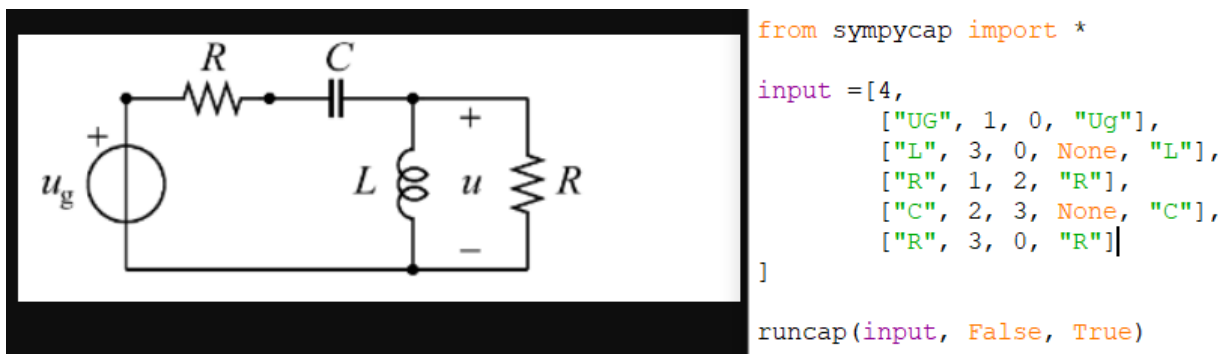


```python
from sympycap import *

input =[11,
        ["UG", 1, 2, "E1"],
        ["UG", 10, 0, "E5"],
        ["UG", 4, 5, "E6"],
        ["UG", 9, 0, "E7"],
        ["UG", 6, 5, "E9"],
        ["IG", 0, 3, "Ig4"],
        ["IG", 8, 1, "Ig3"],
        ["R", 2, 3, "R1"],
        ["R", 1, 0, "R2"],
        ["R", 8, 7, "R3"],
        ["R", 7, 10, "R5"],
        ["R", 3, 4, "R6"],
        ["R", 5, 9, "R7"],
        ["R", 7, 5, "R8"],
        ["R", 7, 6, "R9"]
]

runcap(input, False, True)
```

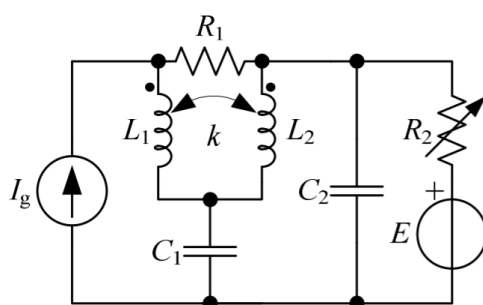*Example 2: OET1 exam 20. January 2019. Exercise 1.*

The following examples demonstrate the use of capacitors inductors and transformers. Please note that all initial conditions of the dynamic elements are set to *None*, meaning that there are no initial conditions, that is initial conditions are equal to zero. We encourage users to input initial conditions in order to see how they impact the results and to familiarize themselves with the use of dynamic elements.



```python
from sympycap import *

input =[4,
        ["UG", 1, 0, "Ug"],
        ["L", 2, 0, None, "L"],
        ["C", 1, 2, None, "C"],
        ["C", 2, 0, None, "C"],
        ["C", 3, 0, None, "C"],
        ["IT", [2, 0], [3, 0], "m"]
]

runcap(input, False, True)
```

Example 3: LC circuit with an ideal transofrmer


```python
from sympycap import *

input =[4,
        ["UG", 1, 0, "Ug"],
        ["L", 3, 0, None, "L"],
        ["R", 1, 2, "R"],
        ["C", 2, 3, None, "C"],
        ["R", 3, 0, "R"]
]

runcap(input, False, True)
```

Example 4: RLC circuit


```python
from sympycap import *

input =[5,
        ["UG", 4, 0, "E"],
        ["IG", 1, 0, "Ig"],
        ["R", 2, 4, "R2"],
        ["R", 1, 2, "R2"],
        ["C", 3, 0, None, "C1"],
        ["C", 2, 0, None, "C2"],
        ["K", [1, 3], [2, 3], ["L1", "L2", "L12"], [None, None], ["i1", "i2"]]
]

runcap(input, False, True)
```
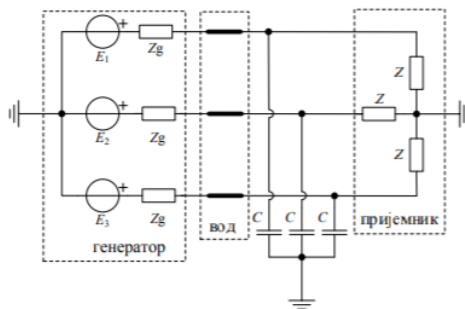
Example 5: OET2 exam 1. February 2019. Exercise 2.
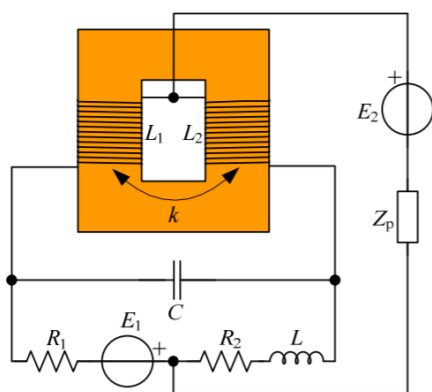
```
from sympycap import *

input =[7,
        ["UG", 0, 1, "E1"],
        ["UG", 6, 5, "E2"],
        ["R", 1, 2, "R1"],
        ["R", 3, 0, "R2"],
        ["L", 4, 3, None, "L"],
        ["C", 2, 4, None, "C"],
        ["Z", 5, 0, "Zp"],
        ["K", [2, 6], [4, 6], ["L1", "L2", "L12"], [None, None], ["i1", "i2"]]
]

runcap(input, False, True)
```
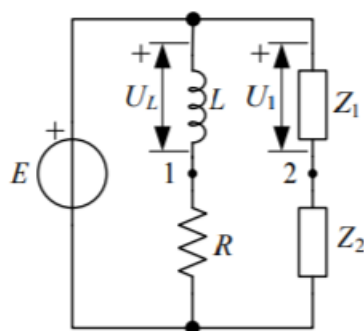
Example 6: OET2 exam 16. June 2019. Exercise 2.

Here are some more examples using impedance and some dynamic elements.


```
from sympycap import *

input =[7,
        ["UG", 1, 0, "E1"],
        ["UG", 2, 0, "E2"],
        ["UG", 3, 0, "E3"],
        ["Z", 4, 0, "Z"],
        ["Z", 5, 0, "Z"],
        ["Z", 6, 0, "Z"],
        ["C", 4, 0, None, "C"],
        ["C", 5, 0, None, "C"],
        ["C", 6, 0, None, "C"],
        ["Z", 4, 1, "Zg"],
        ["Z", 5, 2, "Zg"],
        ["Z", 6, 3, "Zg"]
]

runcap(input, False, True)
```

Example 7: OET2 exam 7. July 2019. Exercise 2.


```
from sympycap import *

input =[4,
        ["UG", 3, 0, "E"],
        ["R", 1, 0, "R"],
        ["L", 3, 1, None, "L"],
        ["Z", 2, 0, "Z2"],
        ["Z", 3, 2, "Z1"]
]

runcap(input, False, True)
```

Example 8: OET2 exam 7. July 2019. Exercise 1.

The next couple of examples will focus on transmission lines. The examples use the Laplace transform for the solution, but the user can switch to the Phasor variant by setting the appropriate boolean argument.
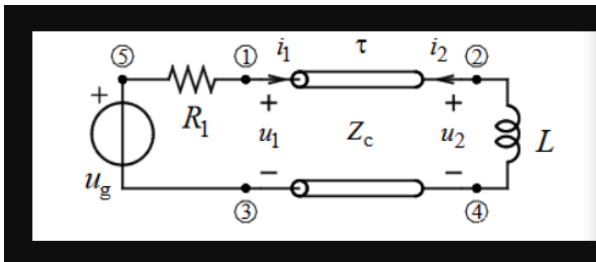


```
from sympycap import *

input =[4,
        ["UG", 1, 0, "Ug"],
        ["L", 3, 0, None, "L"],
        ["R", 1, 2, "R"],
        ["T", [2, 0], [3, 0], ["i1", "i2"], ["Zc", "tau"]]
]

runcap(input, False, True)
```

Example 9: Transmission line with resistor and inductior

Задатак 1

Неоклопљена парица (Unshielded Twisted Pair, UTP) за повезивање рачунара у мрежу се може приближно представити као вод без губитака чија је карактеристична импеданса $Z_c = 100\,\Omega$. Електрично коло чине три оваква вода и отпорници отпорности $R_1 = 2R_2 = R_3 = R_4 = Z_c$. Одзив је устаљен и простопериодичан, а побуда је u_g.
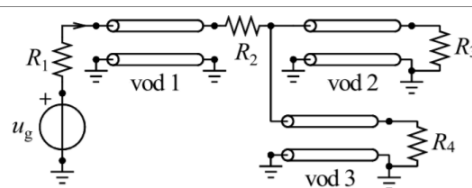
```
from sympycap import *

input =[7,
        ["UG", 1, 0, "Ug"],
        ["R", 2, 1, "Zc"],
        ["T", [2, 0], [3, 0], ["i1", "i2"], ["Zc", "tau"]],
        ["R", 4, 3, "2Zc"],
        ["T", [4, 0], [5, 0], ["i3", "i4"], ["Zc", "tau"]],
        ["R", 5, 0, "Zc"],
        ["T", [4, 0], [6, 0], ["i5", "i6"], ["Zc", "tau"]],
        ["R", 6, 0, "Zc"],
]

runcap(input, False, True)
```

Example 10: UTP represented using transmission lines

Задатак 2

Вилкинсонов делитељ снаге (Wilkinson power divider/combiner), који се користи у радарским системима и бежичним комуникационим системима (WLAN, WiFi, WiMAX, GPS, RFID), приказан је на слици. Одзив је устаљен и простопериодичан, $u_g(t) = \sqrt{2}U_g \cos(2\pi f\,t)$. Водови су без губитака, карактеристичне импедансе $Z_c = \sqrt{2}R$ и електричне дужине $\Theta = \pi/2$ на учестаности f. Отпорности отпорника су $R_1 = R_2 = R_3 = R$, $R_4 = 2R$.
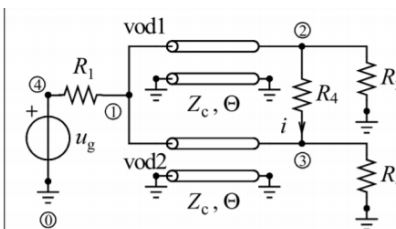

```
from sympycap import *

input =[5,
        ["UG", 4, 0, "Ug"],
        ["R", 4, 1, "R"],
        ["T", [1, 0], [2, 0], ["i1", "i2"], ["Zc", "Theta"]],
        ["R", 2, 3, "2R"],
        ["T", [1, 0], [3, 0], ["i3", "i4"], ["Zc", "Theta"]],
        ["R", 2, 0, "R"],
        ["R", 3, 0, "R"]
]

runcap(input, True, True)
```

Example 11: Wilkinson power divider/combiner

6. Literature

- http://tek.etf.rs/
- http://oet.etf.rs/
- https://maxima.sourceforge.io/
- https://github.com/pssoh/SALECx
- https://docs.python.org/3/tutorial/
- https://www.sympy.org/en/index.html
- https://matplotlib.org/3.1.1/index.html
- https://numpy.org/
- http://www.spectrum-soft.com/index.shtm
- https://www.wolfram.com/mathematica/
- https://www.analog.com/en/design-center/design-tools-and-calculators/ltspice-simulator.html