SymPyCAP Reference Manual

1 Authors

- Katarina Stanković
- Nikola Ilić
- Matija Dodović
- Jelena Bakić

University of Belgrade – School of Electrical Engineering

2 License

Creative Commons

3 Acknowledgment

We thank Prof. dr Dejan V. Tošić and Prof. dr Milka M. Potrebić for recommending this software project to us and for all discussions and help with the project.

4 About SymPyCAP

SymPyCAP is a program for solving linear, time-invariant electric circuits. This program is Python-based (It's written entirely in Python) and uses SymPy, a Python library for symbolic mathematics.
SymPyCAP uses MNA (Modified Nodal Analysis) to formulate and solve equations.

5 Why SymPy?

- SymPy is completely free, open source and licensed under the BSD license. So, you can modify the source code end sell it if you want to.
- SymPy uses Python as its language. This means that if you know Python, it is much easier to get started with SymPy (because you already knows the syntax). And if you don't know Python, it is really easy to learn.
- Third advantage of SymPy is that it is lightweight program. It has no dependencies other than Python, so it can be used almost anywhere easily.
- And finally, it can be used as a library. You can just import it in your own Python application.

Anaconda

- For monitoring this work we recommend Anaconda, free open-source Python distribution. Within it, the environment we recommend is *Jupyter Notebook*.

6 Algorithm

Nodes:

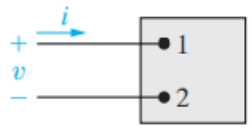
- Reference node - one node, labeled by zero, 0 (default node). The node voltage of this node (reference node) is set to zero, 0.
- Other nodes - labeled by consecutive integers, starting from one, 1.

The Kirchhoff's current law equations (KCL)

- SymPyCAP formulates the KCL equations for all nodes, except the reference node.

Passive sign convention

- Whenever the reference direction for the current in an element is in the direction of the reference voltage drop across the element (as in this picture), use a positive sign in any expression that relates the voltage to the current. Otherwise, use a negative sign.
- We apply this sign convention.



An ideal basic circuit element.

Modified Nodal Analysis

- ***MNA variables:*** node voltages and currents which cannot be expressed in terms of node voltages.
- Node voltages are labeled by V_1, V_2, V_3...
- $V_0 = 0$, by default
- Currents are labeled by I"id" ("id" specifies a circuit element).

Reserved symbols

- *I* – MNA current variables (I[id])
- *V* – MNA voltage variables (V_0, V_1, V_2...)
- *r* – dictionary of replacements in the form:
  {..., "id" : symbolic_value, ...}
- *replacement* - another name for *r*
- *w* – symbol/symbolic expression of frequency for the Phasor transform analysis
- *omega* – another name for *w*

7 Electric Circuit

Input to SymPyCAP (the circuit to be analyzed) is specified as a list of circuit elements (list of lists):

```
[list_1, list_2, list_3, ... list_N]
```

A circuit element ($list_i$) is specified as a list:

- for one-port element:
  ```
  [type, id, a, b]
  ```
  ```
  [type, id, a, b, IC]
  ```
- for two-port element:
  ```
  [type, id, [a1,a2], [b1,b2], p]
  ```
  ```
  [type, id, [a1,a2], b]
  ```
  (b = b1 when b2 is ground node)

type – string that specifies type of element ("R", "L", "C", "Z", "Y", "I", "V", "OpAmp", "IdealT", "InductiveT", "VCVS", "VCCS", "CCCS", "CCVS", "T")

id – string that identifies circuit element ("R1", "L1", "C1", "Ug", "OpAmp1", "I1", "VCVS1", etc.)

a – integer, positive terminal

b – integer, negative terminal

IC – initial conditions at $t_{[0]}$- ("U0" for capacitors, "I0" for inductors, ["I_01", "I_02"] for linear inductive transformers)

a1 – integer, positive terminal of the 1^{st} port

a2 – integer, negative terminal of the 1^{st} port

b1 – integer, positive terminal of the 2^{nd} port

b2 – integer, negative terminal of the 2^{nd} port

p – parameter or list of parameters

One-port elements:

- **Resistor**
  ```
  ["R", "id", plusTerm, minusTerm]
  ```
- **Capacitor**
  ```
  ["C", "id", plusTerm, minusTerm, "U0"]
  ```
  ```
  ["C", "id", plusTerm, minusTerm]
  ```
  U_0 is here 0, by default.

- **Inductor**

  ```
  ["L", "id", plusTerm, minusTerm, "I0"]
  ["L", "id", plusTerm, minusTerm]
  ```

  I_0 is here 0, by default.

- **Impedance**

  ```
  ["Z", "id", plusTerm, minusTerm]
  ```

- **Admitance**

  ```
  ["Y", "id", plusTerm, minusTerm]
  ```

- **Current source – ideal current generator**

  ```
  ["I", "id", plusTerm, minusTerm]
  ```

- **Voltage source – ideal voltage generator**

  ```
  ["V", "id", plusTerm, minusTerm]
  ```
  (V = V [plusTerm] - V [minusTerm])

Two-port elements:

- **Operational Amplifier – Ideal OpAmp**

  ```
  ["OpAmp", "id", [nonInvertingTerm, invertingTerm], 2ndTerm]
  ```

- **Two-port specified by ABCD-parameters (transmission parameters, chain parameters)**

  ```
  ["4-A", "id", [plusPrimaryTerm, minusPrimaryTerm],
  [plusSecondaryTerm, minusSecondaryTerm], ["A", "B", "C", "D"]]
  ```

Controlled Sources:

- **VCVS – Voltage Controlled Voltage Source**

  ```
  ["VCVS", "id", [plusControllingTerm, minusControllingTerm],
  [plusControlledTerm, minusControlledTerm], "voltageGain"]
  ```

- **VCCS – Voltage Controlled Current Source**

  ```
  ["VCCS", "id", [plusControllingTerm, minusControllingTerm],
  [plusControlledTerm, minusControlledTerm], "transconductance"]
  ```

- **CCCS – Current Controlled Current Source**

  ```
  ["CCCS", "id", [plusControllingTerm, minusControllingTerm],
  [plusControlledTerm,  minusControlledTerm], "currentGain"]
  ```

- **CCVS – Current Controlled Voltage Source**

```
["CCVS", "id", [plusControllingTerm, minusControllingTerm],
[plusControlledTerm, minusControlledTerm], "transresistance"]
```

Transformers:

- **Ideal Transformer**

```
["IdealT", "id", [plusPrimaryTerm, minusPrimaryTerm],
[plusSecondaryTerm, minusSecondaryTerm], "turnsRatio"]
```

- **Inductive Transformer**

```
["InductiveT", "id", [plusPrimaryTerm, minusPrimaryTerm],
[plusSecondaryTerm, minusSecondaryTerm], ["L1_id", "L2_id", "L12_id"]]
```

```
["InductiveT", "id", [plusPrimaryTerm, minusPrimaryTerm],
[plusSecondaryTerm, minusSecondaryTerm], ["L1_id", "L2_id", "L12_id"], ["I_01", "I_02"]]
```

"L1_id", "L2_id", "L12_id" are unique ids for coupled coils of transformator.

Transmission lines

- **Transmission line, Phasor Transform**

```
["T", "id", [plusSendingTerm, minusSendingTerm],
[plusReceivingTerm, minusReceivingTerm], [Zc, theta]]
```

Zc – symbolic expression
theta [radian] – symbolic expression, electrical length (real number)
I[“id”,plusSendingTerm] current **into** plusSendingTerm
I[“id”,plusReceivingTerm] current **out of** plusReceivingTerm

- **Transmission line, Laplace Transform**

```
["T", "id", [plusSendingTerm, minusSendingTerm],
[plusReceivingTerm, minusReceivingTerm], [Zc, tau]]
```

Zc – symbolic expression
tau [second] – symbolic expression, delay
I[“id”,plusSendingTerm] current **into** plusSendingTerm
I[“id”,plusReceivingTerm] current **into** plusReceivingTerm

8 Calling SymPyCAP

8.1 Importing symbols:

```
import sympy
S = sympy.Symbol('S')
S1, S2,.. = sympy.symbols('S1, S2')
```

SymPy's Symbol() function's argument is a string containing symbol which can be assigned to a variable.
SymPy's symbols() function returns a *sequence of symbols* with names taken from names argument, which can be a comma or whitespace delimited string, or a sequence of strings.
S1, S2 – user symbols that will be used for circuit analysis (for example: E, R, L, W..). In this sequence can't be reserved symbols.
It is very important to define symbols which will be used in the program.

```
import sympy
S = sympy.Symbol('S', real=True, positive=True)
```

Parameters *real* and *positive* are optional, which introduce assumptions about the properties of symbols used in the symbolic calculation. Without these parameters, S represents a complex number, by default.

- **For the Laplace Transform analysis:**

```
from symPyCAP import Circuit
import sympy
system = Circuit(elements)
system.symPyCAP()
```

elements – arbitrary name for list of circuit elements (it can be any other word..)
system – instance of class Circuit (main class of the program)
`symPyCAP()` – this method initializes V to V_i, user defined symbols, creates MNA equations, for every element in circuit, solves linear system of equations, checks validity of every element.

Also, it can read replacement list for user symbols, for example:

```
system.symPyCAP(replacement = ["R1" : R, "R2" : R])
```

```
system.symPyCAP(r = ["R1" : R, "R2" : R])
```

- **For the Phasor Transform analysis:**

```
from symPyCAP import Circuit
import sympy
system = Circuit(elements)
system.symPyCAP(w = W)
```

W – angular frequency [rad/s]

It can be replaced with:

" " `system.symPyCAP()`
this means that frequency is not specified. By default, it will be marked as "s" in the solution

w = W `system.symPyCAP(w = W)`

omega = W `system.symPyCAP(omega = W)`

In this version, also, method can read replacement list, for example:
`system.symPyCAP(w = W, replacement = {"R1" : R, "R2" : R})` etc.

- **Outputs**

1) circuit specifications

```
from symPyCAP import Circuit
import sympy
system = Circuit(elements)
system.symPyCAP()
```

`system.electric_circuit_specifications()` – this function returns:

1.1) for the Laplace Transform analysis

Circuit specifications:
Number of nodes: <"positive_integer">
Input elements: <"list of elements">
Replacement rule: { <"element_values"> }
Equations: [<"list of equations">]
Variables: [V1, . . . Vn, I["id"]. . .]

1.2) for the Phasor Transform analysis

Circuit specifications:
Number of nodes: <"positive_integer">
Input elements: <"list of elements">
Replacement rule: { <"element_values"> }
Equations: [<"list of equations">]
Variables: [V1, . . . Vn, I["id"]. . .]
Frequency: jw

`Equations` are automatically equal to 0.

2) `system.print_solutions()` – returns solution in form:
variable1: solution(variable1)
variable2: solution(variable2)
...

If the entered circuit is not valid, the program will print: *Solution does not exist!*

3) `system.print_specific_solutions()` – returns solution in the same form as 2), but with applied replacement rules ("R1" : R, "C2" : C,...)

Replacement rule physically changes id with symbols, so if this function returns the solution in the form $\frac{1}{0}$, the program will print: *Steady-state response does not exist at frequency 1/sqrt(C*L)*.

- **Getters**

1) `get_solutions()` – gets dictionary of solutions.

2) `get_specific_solutions()` – gets dictionary of specific solutions (with applied replacement rules).

References

[1] Allen Downey, *Think Python: How to Think Like a Computer Scientist*, Green Tea Press, 2008.

[2] Paul Gerrard, *Lean Python: Learn Just Enough Python to Build Useful Tools*, Apress, 2016.

[3] Anaconda Software Distribution. Computer software. Vers. 2-2.4.0. Anaconda, Nov. 2016. Web. https://anaconda.com (last visited January 27th 2021).

[4] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damián Avila, Safia Abdalla, Carol Willing, Jupyter development team. Jupyter Notebooks - a publishing format for reproducible computational workflows. In Fernando Loizides and Birgit Scmidt, editors, Positioning and Power in Academic Publishing: Players, Agents and Agendas, pages 87–90, Netherlands, 2016. IOS Press.

Classic

[5] Charles A. Desoer, Ernest S. Kuh, *Basic Circuit Theory*, New York, NY, McGraw-Hill, 1969.

[6] Leon O. Chua, Charles A. Desoer, and Ernest S. Kuh, *Linear and nonlinear circuits*, New York, NY, McGraw-Hill, 1987.

General

[7] Charles K. Alexander, Matthew N. O. Sadiku, *Fundamentals of Electric Circuits*, 6/e, New York, NY, McGraw-Hill, 2017.

[8] James W. Nilsson, Susan A. Riedel, *Electric Circuits*, 10/e, Upper Saddle River, NJ, Prentice Hall, 2015.

[9] J. David Irwin, R. Mark Nelms, *Basic Engineering Circuit Analysis*, 11/e, Hoboken, NJ, Wiley, 2015.

[10] James A. Svoboda, Richard C. Dorf, *Introduction to Electric Circuits*, 9/e, Hoboken, NJ, Wiley, 2014.

[11] William H. Hayt, Jr., Jack E. Kemmerly, Steven M. Durbin, *Engineering circuit analysis*, 8/e, New York, NY, McGraw-Hill, 2012.

[12] Farid N. Najm, *Circuit Simulation*, Hoboken, New Jersey, John Wiley & Sons, 2010.

[13] Omar Wing, *Classical Circuit Theory*, Springer Science+Business Media, LLC, New York, NY, 2008.

[14] Wai-Kai Chen (Editor), *Circuit Analysis and Feedback Amplifier Theory*, CRC Press, Taylor & Francis Group, Boca Raton, FL, 2006.

[15] Dejan Tosic, Milka Potrebic, http://tek.etf.rs, Electric Circuit Theory, in Serbian, accessed April 2021.

[16] Predrag Pejovic, http://tnt.etf.bg.ac.rs/~oe4sae/, Free software, accessed February 2021.

[17] Tošić V., Dejan, Potrebić M., Milka. (2019). Symbolic analysis of linear electric circuits with Maxima CAS (Version paper presentation) (pp. 15–18). Presented at the Primena slobodnog softvera i otvorenog hardvera (in English "Applicaton of Free Software and Open Hardware") (PSSOH), Belgrade, Serbia: University of Belgrade - School of Electrical Engineering and Academic Mind. http://doi.org/10.5281/zenodo.3533544

Power Engineering

[18] Arieh L. Shenkman, *Transient Analysis of Electric Power Circuits Handbook*, Springer, Dordrecht, The Netherlands, 2005.

[19] Arieh L. Shenkman, *Circuit Analysis for Power Engineering Handbook*, Springer, Dordrecht, The Netherlands, 1998.

Transmission Lines

[20] Paul R. Clayton, *Analysis of Multiconductor Transmission Lines*, 2/e, Hoboken, NJ, Wiley IEEE Press, 2008.