Symbolic analysis of linear electric circuits with SymPy

Introduction

The aim of this tutorial is to show how to solve electrical circuits using SymPy and the SymPyCAP subroutine. Circuit solving will be presented step by step so that everyone will be able to easily apply the whole procedure to solve their own circuit. Everything shown here is based on SymPyCAP's documentation.

List of examples

List of single elements circuits

Example 1: RLC circuit

A simple RLC circuit in the frequency domain ***W*** and with the initial conditions of the elements (the initial condition of the capacitor is ***UC0***, and the coil ***IL0***) will be solved using SymPyCAP. The circuit is shown in Figure 1.

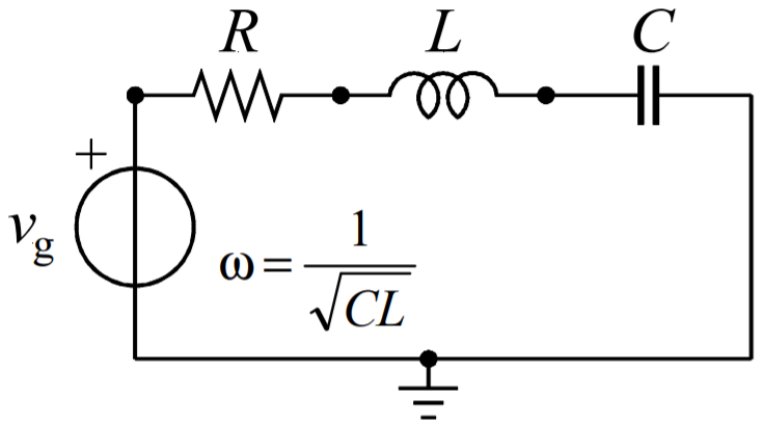

Figure 1: RLC circuit.

Step 1.1. Mark all nodes in the circuit with integer values starting from 0 (Figure 2).

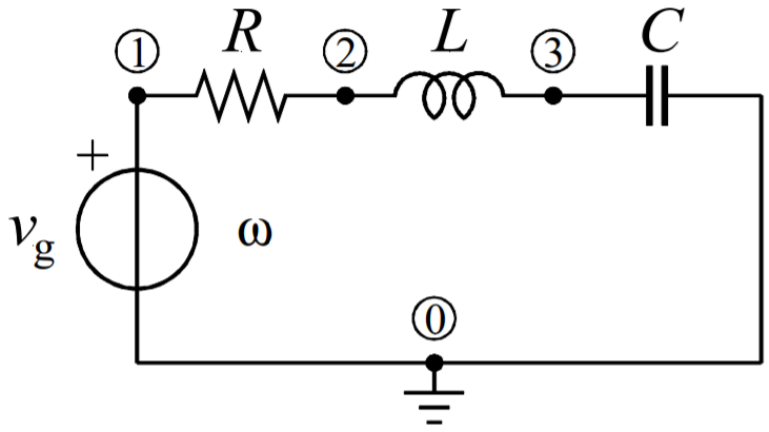

Figure 2: RLC circuit with marked nodes.

Step 1.2. In order to solve this circuit, import the `Circuit` class from symPyCAP and import SymPy.

[1]:
```
from symPyCAP import Circuit
import sympy
```

Step 1.3. Create a list of the elements that make up the circuit. Each element is listed in a specific format given in the documentation.

[2]:
```
RLC_schematic = [
    ["V","Vg",1,0],
    ["R","R1",1,2],
```

```
    ["C","C1",2,3,"UC0"],
    ["L","L1",3,0,"IL0"]
]
```

Step 1.4. In addition to the list of elements, create all the symbols needed to solve the circuit, as well as replacement symbols.

[3]:
```
R,C,L,W = sympy.symbols('R,C,L,W')
```

Step 1.5. Create an instance of the Circuit class with the list of elements `RLC_schematic` as an argument.

[4]:
```
RLC_circuit = Circuit(RLC_schematic)
```

Step 1.6. Call `symPyCAP` method to solve the circuit, that is, to find the response. Pass the symbolic values of the elements and the frequency.
Note: `sympy.sqrt()` should be used, not `math.sqrt()`.

[5]:
```
RLC_circuit.symPyCAP(w = 1/(sympy.sqrt(L*C)), replacement = {"R1" : R, "C1" : C,␣
↪"L1" : L})
```

Step 1.7. After the method has solved the circuit, print the solution.
Note: printing the solution that does not include the replacement can be done by the `print_solutions()` method.

[6]:
```
RLC_circuit.print_specific_solutions()
```

```
V1 : Vg

V2 : 0

V3 : I*Vg*(C*L)**(3/2)/(C**2*L*R)

IVg : -Vg/R
```

Note: The solution doesn't contain intial conditions of capacitor and coil because of jw analysis.
Note: Vg is the rms value of the sinusoidal voltage source with the angular frequency $w = \frac{1}{\sqrt{LC}}$

Step 1.8. It is possible to print the circuit specifications.

[7]:
```
RLC_circuit.electric_circuit_specifications()
```

```
Circuit specifications:
Number of nodes: 4
Input elements:
['V', 'Vg', 1, 0]
['R', 'R1', 1, 2]
['C', 'C1', 2, 3, 'UC0']
```

```
['L', 'L1', 3, 0, 'IL0']
Replacement rule:  {'R1': R, 'C1': C, 'L1': L}
Equations:  [IVg + (V1 - V2)/R1, I*C1*(V2 - V3)/sqrt(C*L) + (-V1 + V2)/R1,
I*C1*(-V2 + V3)/sqrt(C*L) - I*V3*sqrt(C*L)/L1, -Vg + V1]
Variables:  [V1, V2, V3, IVg]
Frequency:  1/sqrt(C*L)
```

Example 2: Simple capacitor circuit

A simple capacitor circuit is shown in Figure 3. The generator voltage is a step function of strength ***Vstep***. The capacitor is initially charged and its preinitial voltage, that is the initial condition, is ***V0***. Current of the ideal voltage source is presented to specify the reference direction.

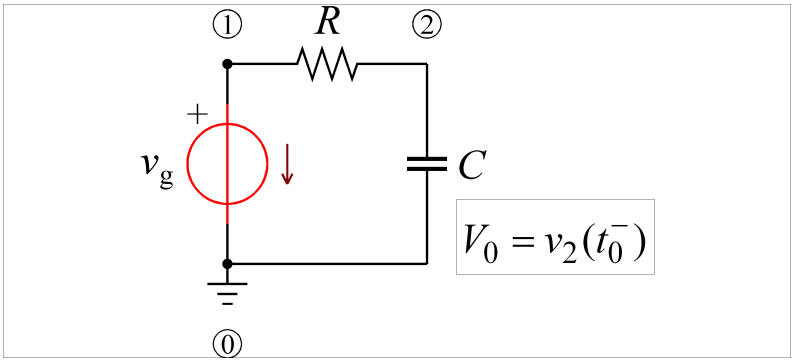

Figure 3: Simple capacitor circuit.

Step 2.1. Repeat steps 1.2 - 1.7.

[8]:
```
from symPyCAP import Circuit
from sympy import *
```

Note: Symbols like ***I[id]***, w, omega, r, replacement, ***V0***, ***V1***, ... are reserved symbols and must not be used for either symbol ids or for values in replacement rule list.

[9]:
```
Simple_capacitor_schematic = [
    ["V","Vg",1,0],
    ["R","R1",1,2],
    ["C","C1",2,0,"VC0"]
]
```

[10]:
```
Vstep,s,t = sympy.symbols('Vstep,s,t')
R = sympy.Symbol('R', positive=True)
C = sympy.Symbol('C', positive=True)
```

[11]:
```
simple_capacitor_circuit = Circuit(Simple_capacitor_schematic)
```

Note: calling `symPyCAP` without the frequency specification (w), solves the circuit in the s-domain, the Unilateral (one-sided) Laplace Transform domain.

[12]:
```
simple_capacitor_circuit.symPyCAP(replacement = {"R1" : R, "C1" : C, "Vg" :␣
↪Vstep/s})
```

Note: The Laplace Transform of the source voltage is Vstep/s, where s represents the complex frequency, that is the Laplace Variable.

[13]:
```
simple_capacitor_circuit.print_specific_solutions()
```

```
V1 : Vstep/s

V2 : (C*R*VC0*s + Vstep)/(s*(C*R*s + 1))

IVg : C*(VC0 - Vstep)/(C*R*s + 1)
```

Step 2.2. The time-domain capacitor voltage, for t > 0, can be computed by SymPy's Inverse Laplace Transform function.
Note: The getter `get_specific_solutions()` returns the dictionary of specific solutions which can be directly indexed.

[14]:
```
Uct = inverse_laplace_transform(system.get_specific_solutions()["V2"],s,t)
```

[15]:
```
Uct
```

[15]:
$$\left(VC_0 + Vstepe^{\frac{t}{CR}} - Vstep\right) e^{-\frac{t}{CR}} \theta\left(t\right)$$

Example 3: OTA-C (Operational Transconductance Amplifier with Capacitors)

An **OTA-C** (**O**PERATIONAL **T**RANSCONDUCTANCE **A**MPLIFIER with **C**APACITORS) low-pass and highpass 2^{nd}- order filter realization (Figure 4.) is to be solved.

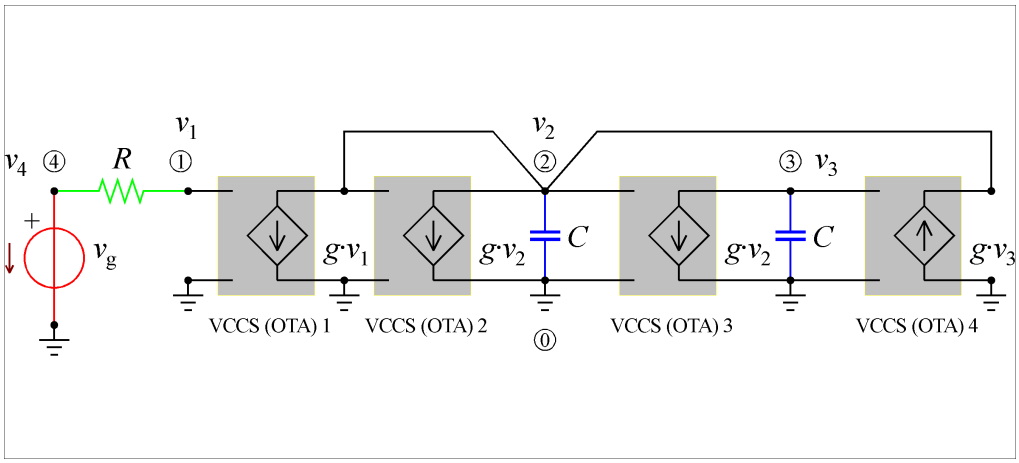

Figure 4: OTA-C (Operational Transconductance Amplifier with Capacitors).

Step 3.1. Repeat steps 1.2-1.7.

[16]:
```
from symPyCAP import Circuit
from sympy import *
```

[17]:
```
OTA_C_schematic = [
    ["V", "Vg", 4, 0],
    ["R", "R1", 4, 1],
    ["VCCS", "OTA1", [1,0], [2,0], "g"],
    ["VCCS", "OTA2", [2,0], [2,0], "g"],
    ["VCCS", "OTA3", [2,0], [3,0], "g"],
    ["VCCS", "OTA4", [3,0], [0,2], "g"],
    ["C", "C1", 2, 0],
    ["C", "C2", 3, 0]
]
```

[18]:
```
R,C = symbols('R,C')
```

[19]:
```
OTA_C_circuit = Circuit(OTA_C_schematic)
```

[20]:
```
OTA_C_circuit.symPyCAP(replacement = {"R1" : R, "C1" : C, "C2" : C})
```

[21]:
```
OTA_C_circuit.print_specific_solutions()
```

```
V1 : Vg

V2 : -C*Vg*g*s/(C**2*s**2 + C*g*s + g**2)

V3 : Vg*g**2/(C**2*s**2 + C*g*s + g**2)

V4 : Vg

IVg : 0
```

Step 3.2. Use node voltages to calculate capacitors bandwidths.

[22]:
```
Hs2bandpass = OTA_C_circuit.get_specific_solutions()['V2']/OTA_C_circuit.
↪get_specific_solutions()['V1']
```

[23]:
```
Hs2bandpass
```

[23]:
$$-\frac{Cgs}{C^2s^2 + Cgs + g^2}$$

[24]:
```
Hs3lowpass = OTA_C_circuit.get_specific_solutions()['V3']/OTA_C_circuit.
↪get_specific_solutions()['V1']
```

[25]:
```
Hs3lowpass
```

[25]:
$$\frac{g^2}{C^2s^2 + Cgs + g^2}$$

Example 4: Riordan gyrator network

Synthetic inductor, which is realized with the Riordan gyrator network, is shown in Figure 5. The proof-of-concept symbolic analysis follows. The circuit is inductorless but, theoretically, the impedance seen by the source is purely inductive.

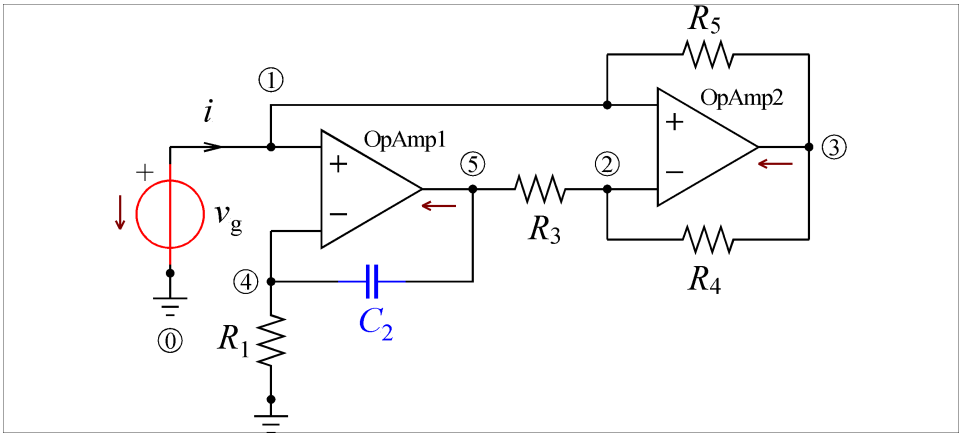

Figure 5: Riordan gyrator network

Step 4.1. Repeat steps 1.2.-1.7.

[26]:
```
from symPyCAP import Circuit
from sympy import *
```

[27]:
```
Riordan_schematic = [
    ["V", "Vg", 1, 0],
    ["OpAmp", "OpAmp1", [1,4], 5],
```

```
    ["R", "R1", 4, 0],
    ["C", "C2", 4, 5],
    ["R", "R3", 5, 2],
    ["OpAmp", "OpAmp2", [1,2], 3],
    ["R", "R4", 2, 3],
    ["R", "R5", 1, 3]
]
```

[28]:
```
Riordan_circuit = Circuit(Riordan_schematic)
```

Note: calling `symPyCAP` without any replacement results in a solution using only provided ids and can be printed using the `print_solutions()` method and fetched by getter method `get_solutions()`.

[29]:
```
Riordan_circuit.symPyCAP()
```

[30]:
```
Riordan_circuit.print_solutions()
```

```
V1 : Vg

V2 : Vg

V3 : Vg - R4*Vg/(C2*R1*R3*s)

V4 : Vg

V5 : Vg + Vg/(C2*R1*s)

IVg : -R4*Vg/(C2*R1*R3*R5*s)

IOpAmp1 : -Vg/R1 - Vg/(C2*R1*R3*s)

IOpAmp2 : Vg*(R4 + R5)/(C2*R1*R3*R5*s)
```

Step 4.2. Calculate the input impedance seen by the source.

[31]:
```
Zin = Riordan_circuit.get_solutions()['V1']/(-Riordan_circuit.
 ↪get_solutions()['IVg'])
```

[32]:
```
Zin
```

[32]:
$$\frac{C_2 R_1 R_3 R_5 s}{R_4}$$

Step 4.3. Calculate the value of the synthetic coil.

[33]:
```
S = Symbol('s')
```

[34]:
```
Lsynthetic = Zin/S
```

[35]:
```
Lsynthetic
```

[35]:
$$\frac{C_2 R_1 R_3 R_5}{R_4}$$

Example 5: Wilkinson power divider

Wilkinson power divider, which is realized with ideal lossless transmission line sections, is shown in Figure 6. The corresponding symbolic analysis, performed in the Phasor Transform domain, verifies that the circuit equally divides input power to the loads ***R2 = R*** and ***R3 = R***, i.e. ***V2 = V3***.

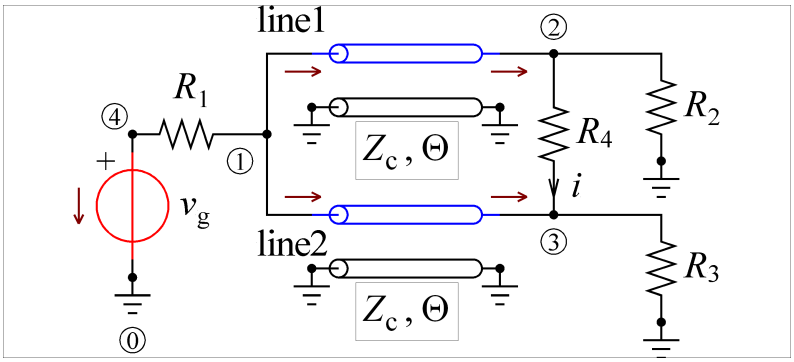

Figure 6: Wilkinson power divider.

Step 5.1. Repeat steps 1.2.-1.7.

[36]:
```
from symPyCAP import Circuit
import sympy
```

[37]:
```
R = Symbol('R')
W = symbols('W')
```

Note: `sympy.sqrt()` and `sympy.pi` should be used, not `math.sqrt()` and `math.pi`.

[38]:
```
Wilkinson_schematic = [
    ["V", "Vg", 4, 0],
    ["R", "R1", 1, 4],
    ["R", "R2", 2, 0],
    ["R", "R3", 3, 0],
    ["R", "R4", 2, 3],
    ["T", "T1", [1,0], [2,0], [sympy.sqrt(2)*R,sympy.pi/2]],
    ["T", "T2", [1,0], [3,0], [sympy.sqrt(2)*R,sympy.pi/2]]
]
```

[39]:
```
Wilkinson_circuit = Circuit(Wilkinson_schematic)
```

[40]:
```
Wilkinson_circuit.symPyCAP(w = W, replacement = {"R1" : R, "R2" : R, "R3" : R,␣
↪"R4" : 2*R})
```

[41]:
```
Wilkinson_circuit.print_specific_solutions()
```

```
V1 : Vg/2

V2 : -sqrt(2)*I*Vg/4

V3 : -sqrt(2)*I*Vg/4

V4 : Vg

IVg : -Vg/(2*R)

IT1_1 : Vg/(4*R)

IT1_2 : -sqrt(2)*I*Vg/(4*R)

IT2_1 : Vg/(4*R)

IT2_3 : -sqrt(2)*I*Vg/(4*R)
```

Note: I represents the imaginary unit in Python, $I = \sqrt{-1}$. It is not the current at a port of an element.

Step 5.2. Compare the voltages of the 2^{nd} and 3^{rd} node.

[42]:
```
simplify(Wilkinson_circuit.get_specific_solutions()['V2']-Wilkinson_circuit.
↪get_specific_solutions()['V3']) == 0
```

[42]:
```
True
```

Example 6: Transmission line circuit

Doubly terminated lossless transmission line section is shown in Figure 7. The corresponding symbolic analysis, performed in the s-domain, verifies that the circuit acts as a delay line.

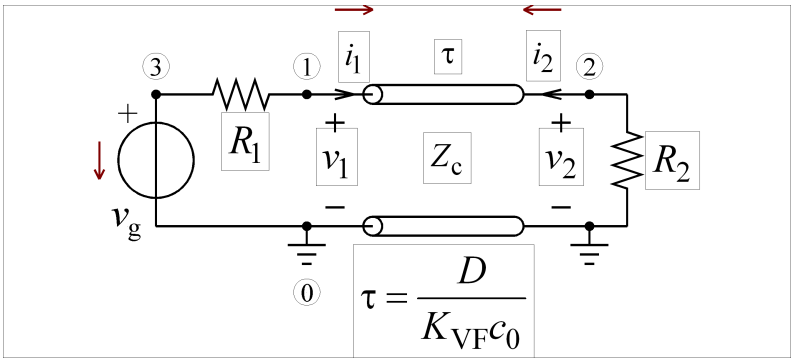

Figure 7: Transmission line circuit; the *s*-domain.

Step 6.1. Repeat steps 1.2.-1.7.

[43]:
```
from symPyCAP import Circuit
from sympy import *
```

[44]:
```
Zc = Symbol('Zc')
tau = symbols(r'tau')
```

[45]:
```
TLine_schematic = [
    ["V", "Vg", 3, 0],
    ["R", "R1", 3, 1],
    ["T", "T1", [1,0], [2,0], [Zc,tau]],
    ["R", "R2", 2, 0]
]
```

[46]:
```
TLine_circuit = Circuit(TLine_schematic)
```

[47]:
```
TLine_circuit.symPyCAP(replacement = {"R1" : Zc, "R2" : Zc})
```

[48]:
```
TLine_circuit.print_specific_solutions()
```

```
V1 : Vg/2

V2 : Vg*exp(-s*tau)/2

V3 : Vg

IVg : -Vg/(2*Zc)

IT1_1 : Vg/(2*Zc)

IT1_2 : -Vg*exp(-s*tau)/(2*Zc)
```

[49]:
```
TLine_circuit.get_specific_solutions()["V2"]
```

[49]:

$$\frac{Vge^{-s\tau}}{2}$$

Example 7: Adder

Adder, which is realized an with operational amplifier, is shown in Figure 8. The sum of all voltages at the input should be obtained at the output of the operational amplifier.

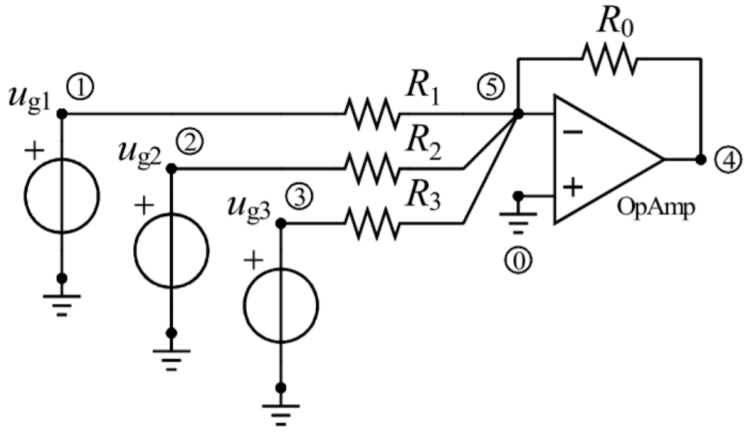

Figure 8: Adder.

Step 7.1. Repeat steps 1.2-1.7.

[50]:
```
from symPyCAP import Circuit
import sympy
```

[51]:
```
R = sympy.Symbol('R')
Ug1 = sympy.Symbol('Ug1')
Ug2 = sympy.Symbol('Ug2')
Ug3 = sympy.Symbol('Ug3')
```

[52]:
```
Adder_schematic = [
    ["R", "R1", 5, 1],
    ["R", "R2", 5, 2],
    ["R", "R3", 5, 3],
    ["R", "R0", 5, 4],
    ["V", "Vg1", 1, 0],
    ["V", "Vg2", 2, 0],
    ["V", "Vg3", 3, 0],
    ["OpAmp", "OpAmp1", [0,5], 4]
]
```

[53]:
```
Adder_circuit = Circuit(Adder_schematic)
```

[54]:
```
Adder_circuit.symPyCAP(replacement = {"R1" : R, "R2" : R, "R3" : R, "R0" : R,␣
↪"Vg1" : Ug1, "Vg2" : Ug2, "Vg3" : Ug3})
```

[55]:
```
Adder_circuit.print_specific_solutions()
```

```
V1 : Ug1

V2 : Ug2

V3 : Ug3

V4 : -Ug1 - Ug2 - Ug3

V5 : 0

IVg1 : -Ug1/R

IVg2 : -Ug2/R

IVg3 : -Ug3/R

IOpAmp1 : (Ug1 + Ug2 + Ug3)/R
```

Example 8: Subtractor

Subtractor, which is realized with an operational amplifier, is shown in Figure 9.

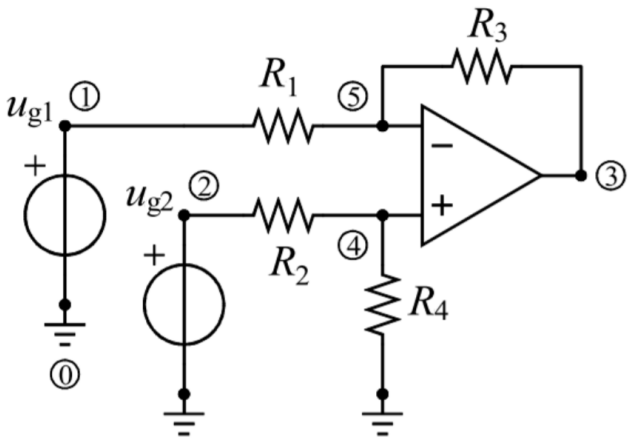

Figure 9: Subtractor.

Step 8.1. Repeat steps 1.2-1.7.

[56]:
```
from symPyCAP import Circuit
import sympy
```

[57]:
```
R = sympy.Symbol('R')
Ug1 = sympy.Symbol('Ug1')
Ug2 = sympy.Symbol('Ug2')
```

[58]:
```
Subtractor_shema = [
    ["R", "R1", 5, 1],
    ["R", "R2", 4, 2],
    ["R", "R3", 5, 3],
    ["R", "R4", 0, 4],
    ["V", "Vg1", 1, 0],
    ["V", "Vg2", 2, 0],
    ["OpAmp", "OpAmp1", [4,5], 3]
]
```

[59]:
```
Subtractor_circuit = Circuit(Subtractor_shema)
```

[60]:
```
Subtractor_circuit.symPyCAP(replacement = {"R1" : R, "R2" : R, "R3" : R, "R4" :␣
↪R, "Vg1" : Ug1, "Vg2" : Ug2})
```

[61]:
```
Subtractor_circuit.print_specific_solutions()
```

```
V1 : Ug1

V2 : Ug2

V3 : -Ug1 + Ug2

V4 : Ug2/2

V5 : Ug2/2

IVg1 : (-Ug1 + Ug2/2)/R

IVg2 : -Ug2/(2*R)

IOpAmp1 : (Ug1 - Ug2/2)/R
```

Example 9: Inductive Transformer

An inductive transformer with resistors, is shown in Figure 10.

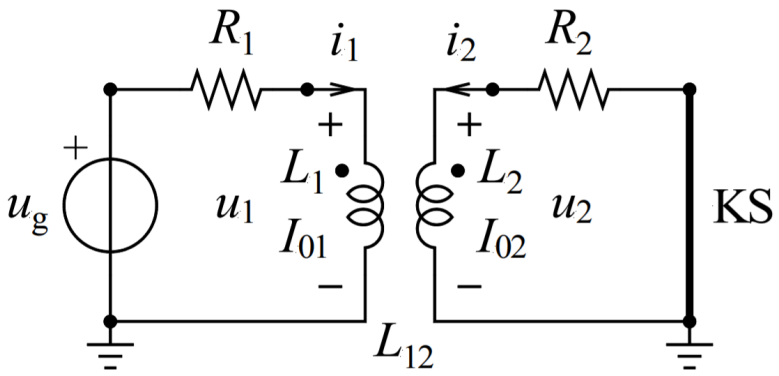

Figure 10: Inductive Transformer.

Step 9.1. Repeat steps 1.2-1.7.

[62]:
```
from symPyCAP import Circuit
import sympy
```

[63]:
```
R = sympy.Symbol('R', positive=True)
L = sympy.Symbol('L', positive=True)
t = sympy.Symbol('t')
```

[64]:
```
Inductive_schematic = [
    ["R", "R1", 3, 1],
    ["R", "R2", 0, 2],
    ["V", "Ug", 3, 0],
    ["InductiveT", "InductiveT1", [1,0], [2,0], ["LT1_1","LT1_2","LT1_12"],␣
 ↪["Io1","Io2"]]
]
```

[65]:
```
Inductive_circuit = Circuit(Inductive_schematic)
```

[66]:
```
Inductive_circuit.symPyCAP(replacement = {"R1" : R, "R2" : R, "LT1_1" : L,␣
 ↪"LT1_2" : L, "LT1_12" : L/2})
```

[67]:
```
Inductive_circuit.print_specific_solutions()
```

```
V1 : L*(-3*Io1*L*R*s - 4*Io1*R**2 - 2*Io2*R**2 + 3*L*Ug*s**2 +
4*R*Ug*s)/(3*L**2*s**2 + 8*L*R*s + 4*R**2)

V2 : -L*R*(2*Io1*R + 3*Io2*L*s + 4*Io2*R - 2*Ug*s)/(3*L**2*s**2 + 8*L*R*s +
4*R**2)

V3 : Ug

IUg : -(3*Io1*L**2*s + 4*Io1*L*R + 2*Io2*L*R + 4*L*Ug*s + 4*R*Ug)/(3*L**2*s**2 +
8*L*R*s + 4*R**2)
```

```
IInductiveT1_1 : (3*Io1*L**2*s + 4*Io1*L*R + 2*Io2*L*R + 4*L*Ug*s +
4*R*Ug)/(3*L**2*s**2 + 8*L*R*s + 4*R**2)

IInductiveT1_2 : L*(2*Io1*R + 3*Io2*L*s + 4*Io2*R - 2*Ug*s)/(3*L**2*s**2 +
8*L*R*s + 4*R**2)
```

Step 9.2. The time-domain for i_1 and i_2, for t > 0, can be computed by SymPy's Inverse Laplace Transform method.

[68]:
```
inverse_laplace_transform(Inductive_circuit.
↪get_specific_solutions()["IInductiveT1_2"],s,t)
```

[68]:
$$-\frac{Io_1 e^{-\frac{2Rt}{L}}\theta(t)}{2}+\frac{Io_1 e^{-\frac{2Rt}{3L}}\theta(t)}{2}+\frac{Io_2 e^{-\frac{2Rt}{L}}\theta(t)}{2}+\frac{Io_2 e^{-\frac{2Rt}{3L}}\theta(t)}{2}-\frac{Ug e^{-\frac{2Rt}{L}}\theta(t)}{L}+\frac{Ug e^{-\frac{2Rt}{3L}}\theta(t)}{3L}$$

[69]:
```
inverse_laplace_transform(Inductive_circuit.
↪get_specific_solutions()["IInductiveT1_1"],s,t)
```

[69]:
$$\frac{Io_1 e^{-\frac{2Rt}{L}}\theta(t)}{2}+\frac{Io_1 e^{-\frac{2Rt}{3L}}\theta(t)}{2}-\frac{Io_2 e^{-\frac{2Rt}{L}}\theta(t)}{2}+\frac{Io_2 e^{-\frac{2Rt}{3L}}\theta(t)}{2}+\frac{Ug e^{-\frac{2Rt}{L}}\theta(t)}{L}+\frac{Ug e^{-\frac{2Rt}{3L}}\theta(t)}{3L}$$

[70]:
```
ugt = Phi*DiracDelta(t)
```

[71]:
```
ugs = laplace_transform(ugt, t, s)
```

[72]:
```
ugs
```

[72]:
```
(Phi*(1 - Heaviside(0)), -oo, True)
```

[73]:
```
Inductive_circuit.get_specific_solutions()["IInductiveT1_2"].subs(R, L)
```

[73]:
$$\frac{\frac{Io_1 L^2}{2}+\frac{3Io_2 L^2 s}{4}+Io_2 L^2-\frac{LUgs}{2}}{\frac{3L^2s^2}{4}+2L^2s+L^2}$$

Example 10: LC circuit

An LC circuit is shown in Figure 11. Its resonant frequency is $w = \frac{1}{\sqrt{LC}}$. At this frequency, the circuit response diverges.

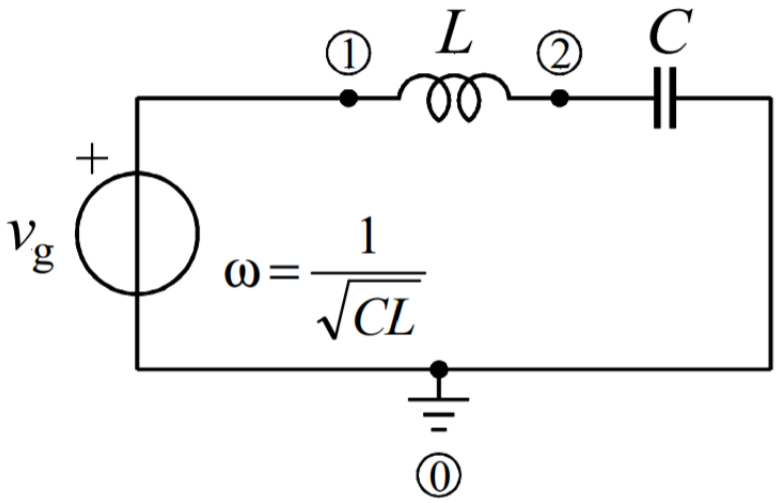

Figure 11: LC circuit.

Step 10.1. Repeat steps 1.2-1.7.

[70]:
```
from symPyCAP import Circuit
import sympy
```

[71]:
```
C = sympy.Symbol('C')
L = sympy.Symbol('L')
```

[72]:
```
LC_schematic = [
    ["L", "L1", 2, 1],
    ["C", "C1", 0, 2],
    ["V", "E1", 1, 0],
]
```

[73]:
```
LC_circuit = Circuit(LC_schematic)
```

[74]:
```
LC_circuit.symPyCAP(w = 1/(sympy.sqrt(L*C)), replacement = {"L1" : L, "C1" : C})
```

[75]:
```
LC_circuit.print_specific_solutions()
```

```
Steady-state response does not exist at frequency 1/sqrt(C*L)
```

Example 11: Parallel connection of voltage generators

An untypical example is two voltage generators of different voltage values. This circuit does not have a solution.

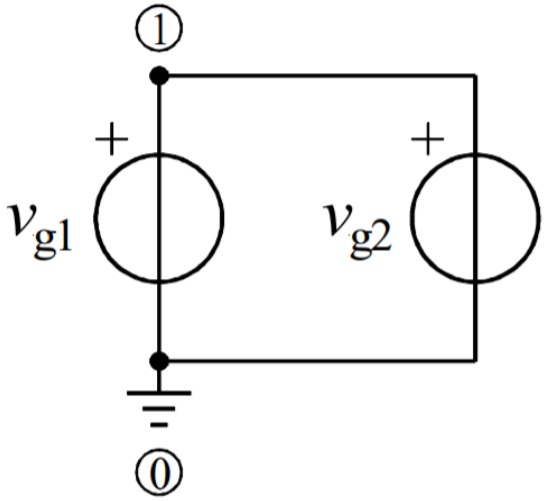

Figure 12: Parallel connection of voltage generators.

Step 11.1. Repeat necessary steps 1.2-1.7.

[76]:
```
from symPyCAP import Circuit
```

[77]:
```
EE_schematic = [
    ["V", "E1", 0, 1],
    ["V", "E2", 0, 1],
]
```

[78]:
```
EE_circuit = Circuit(EE_schematic)
```

[79]:
```
EE_circuit.symPyCAP()
```

[80]:
```
EE_circuit.print_solutions()
```

```
Solution does not exist!
```

The circuit is not well-behaved because the element equations, KCL and KVL cannot hold simultaneously.

Example 12: Ideal transformers

Two ideal transformers, which are cascaded to get the higher gain, are shown in Figure 13.

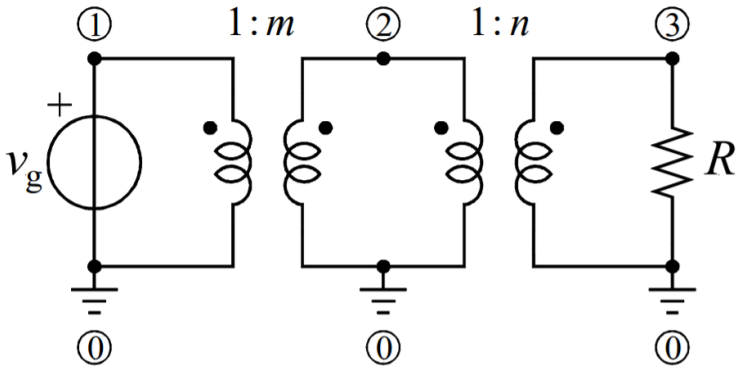

Figure 13: Ideal transformers.

Step 12.1. Repeat necessary steps 1.2-1.7.

[81]:
```
from symPyCAP import Circuit
import sympy
```

[82]:
```
m = sympy.Symbol('m')
n = sympy.Symbol('n')
```

[83]:
```
Ideal_schematic = [
    ["V", "E1", 1, 0],
    ["R", "R1", 3, 0],
    ["IdealT", "IT1", [2,0], [1,0], "m"],
    ["IdealT", "IT2", [3,0], [2,0], "n"]
]
```

[84]:
```
Ideal_circuit = Circuit(Ideal_schematic)
```

[85]:
```
Ideal_circuit.symPyCAP()
```

[86]:
```
Ideal_circuit.print_solutions()
```

```
V1 : E1

V2 : E1*m

V3 : E1*m*n

IE1 : -E1*m**2*n**2/R1

IIT1 : -E1*m*n**2/R1

IIT2 : -E1*m*n/R1
```

Example 13: Voltage divider

A voltage divider with two resistors and a voltage generator is shown in Figure 14.

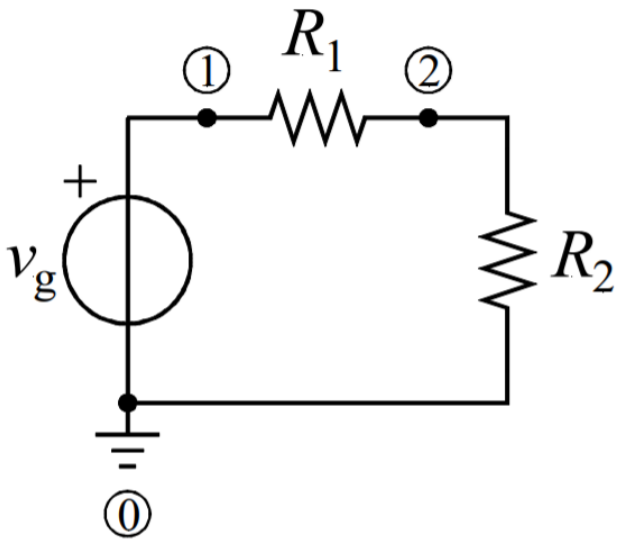

Figure 14: Voltage divider.

Step 13.1. Repeat necessary steps 1.2-1.7.

[91]:
```
from symPyCAP import Circuit
from sympy import *
```

[92]:
```
Voltage_divider_schematic = [
    ["V", "Vg", 1, 0],
    ["R", "R1", 1, 2],
    ["R", "R2", 2, 0]
]
```

[93]:
```
Voltage_divider_circuit = Circuit(Voltage_divider_schematic)
```

[94]:
```
Voltage_divider_circuit.symPyCAP()
```

[95]:
```
Voltage_divider_circuit.print_solutions()
```

```
V1 : Vg

V2 : R2*Vg/(R1 + R2)

IVg : -Vg/(R1 + R2)
```

Step 13.2. Easily obtain the voltages across resistors R1 and R2.

[96]:
```
UR1 = simplify(Voltage_divider_circuit.
 ↪get_solutions()["V1"]-Voltage_divider_circuit.get_solutions()["V2"])
```

[97]:
```
UR1
```

[97]: $$\frac{R_1 Vg}{R_1 + R_2}$$

[98]:
```
UR2 = Voltage_divider_circuit.get_solutions()["V2"]
```

[99]:
```
UR2
```

[99]: $$\frac{R_2 Vg}{R_1 + R_2}$$

Single element circuits

Example 14.1: Voltage generator - open circuit

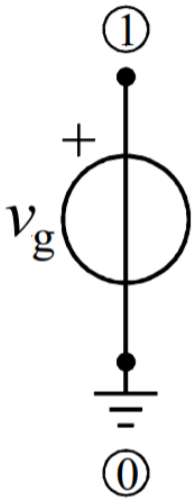

Figure 15: Voltage generator - open circuit.

Step 14.1.1. Repeat necessary steps 1.2-1.7.

[100]:
```
from symPyCAP import Circuit
```

[101]:
```
Generator_open_schematic = [
    ["V", "Vg", 1, 0]
]
```

[102]:
```
Voltage_open_circuit = Circuit(Generator_open_schematic)
```

[103]:
```
Voltage_open_circuit.symPyCAP()
```

[104]:
```
Voltage_open_circuit.print_solutions()
```

```
V1 : Vg

IVg : 0
```

Example 14.2: Voltage generator - closed circuit

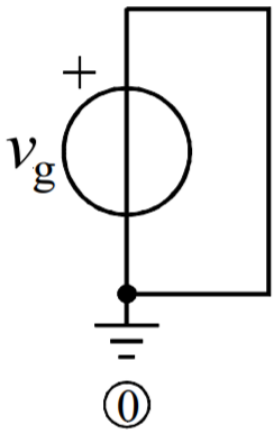

Figure 16: Voltage generator - closed circuit.

Step 14.2.1. Repeat necessary steps 1.2-1.7.

[105]:
```
from symPyCAP import Circuit
```

[106]:
```
Generator_closed_schematic = [
    ["V", "Vg", 0, 0]
]
```

[107]:
```
Voltage_closed_circuit = Circuit(Generator_closed_schematic)
```

[108]:
```
Voltage_closed_circuit.symPyCAP()
```

[109]:
```
Voltage_closed_circuit.print_solutions()
```

```
Solution does not exist!
```

Example 15.1: Current generator - closed circuit

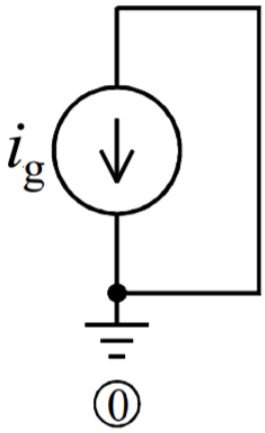

Figure 17: Current generator - closed circuit.

Step 15.1.1. Repeat necessary steps 1.2-1.7.

[110]:
```
from symPyCAP import Circuit
```

[111]:
```
Generator_closed_schematic = [
    ["I", "Ig", 0, 0]
]
```

[112]:
```
Current_closed_circuit = Circuit(Generator_closed_schematic)
```

[113]:
```
Current_closed_circuit.symPyCAP()
```

[114]:
```
Current_closed_circuit.print_solutions()
```

```
Solution does not exist!
```

Example 15.2: Current generator - open circuit

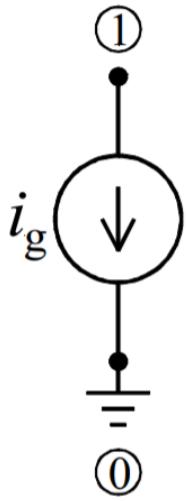

Figure 18: Current generator - open circuit.

Step 15.1.1. Repeat necessary steps 1.2-1.7.

[115]:
```
from symPyCAP import Circuit
```

[116]:
```
Generator_open_schematic = [
    ["I", "Ig", 1, 0]
]
```

[117]:
```
Current_open_circuit = Circuit(Generator_open_schematic)
```

[118]:
```
Current_open_circuit.symPyCAP()
```

[119]:
```
Current_open_circuit.print_solutions()
```

```
Solution does not exist!
```

Example 16.1: Capacitor with initial state - open circuit

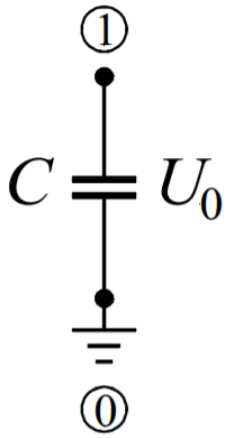

Figure 19: Capacitor with initial state - open circuit

Step 16.1.1. Repeat necessary steps 1.2-1.7.

[120]:
```
from symPyCAP import Circuit
```

[121]:
```
Capacitor_open_schematic = [
    ["C", "C1", 1, 0, "U0"]
]
```

[122]:
```
Capacitor_open_circuit = Circuit(Capacitor_open_schematic)
```

[123]:
```
Capacitor_open_circuit.symPyCAP()
```

[124]:
```
Capacitor_open_circuit.print_solutions()
```

```
V1 : U0/s
```

Since UC0=const, the time-domain voltage v1(t)=U0 for t>0, In other words, the Inverse Laplace Transform of UC0/s is UC0*HeavisideTheta(t).

Example 16.2: Capacitor with initial state - closed circuit

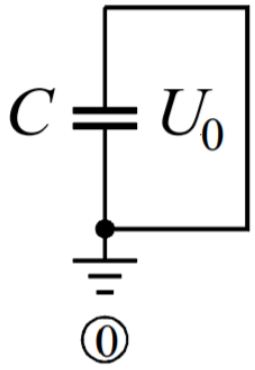

Figure 20: Capacitor with initial state - closed circuit.

Step 16.2.1. Repeat necessary steps 1.2-1.7.

[125]:
```
from symPyCAP import Circuit
```

[126]:
```
Capacitor_closed_schematic = [
    ["C", "C1", 0, 0, "U0"]
]
```

[127]:
```
Capacitor_closed_circuit = Circuit(Capacitor_closed_schematic)
```

[128]:
```
Capacitor_closed_circuit.symPyCAP()
```

[129]:
```
Capacitor_closed_circuit.print_solutions()
```

```
Solution does not exist!
```

Example 17.1: Inductor with initial state - open circuit

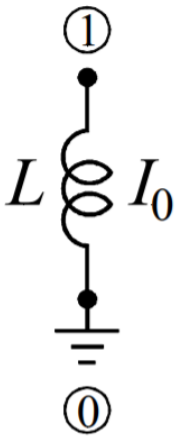

Figure 21: Inductor with initial state - open circuit.

Step 17.1.1. Repeat necessary steps 1.2-1.7.

[130]:
```
from symPyCAP import Circuit
```

[131]:
```
Inductor_open_schematic = [
    ["L", "L1", 1, 0, "I0"]
]
```

[132]:
```
Inductor_open_circuit = Circuit(Inductor_open_schematic)
```

[133]:
```
Inductor_open_circuit.symPyCAP()
```

[134]:
```
Inductor_open_circuit.print_solutions()
```

```
V1 : -I0*L1
```

Example 17.2: Inductor with initial state - closed circuit

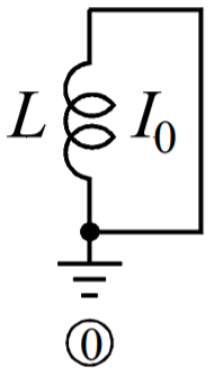

Figure 22: Inductor with inital state - closed circuit.

Step 17.2.1. Repeat necessary steps 1.2-1.7.

[135]:
```
from symPyCAP import Circuit
```

[136]:
```
Inductor_closed_schematic = [
    ["L", "L1", 0, 0, "I0"]
]
```

[137]:
```
Inductor_closed_circuit = Circuit(Inductor_closed_schematic)
```

[138]:
```
Inductor_closed_circuit.symPyCAP()
```

[139]:
```
Inductor_closed_circuit.print_solutions()
```

```
Solution does not exist!
```