```
/*
Author
*/

/*
Symbolic Analysis of Linear Electric Circuits with Maxima
SALECx version 1.0 (2019-10-09) for Maxima 5.38+, wxMaxima 16
*/

/*
Dr. Dejan Tošić, Full Professor, tosic@etf.rs
University of Belgrade -- School of Electrical Engineering
11000 Belgrade, Serbia
*/

/*
License
*/

/*
Creative Commons
*/

/*
Acknowledgement
*/

/*
I thank Prof. Dr. Predrag Pejović for permanent encouragement
valuable discussions related to this project.
*/

/*
Presented and Published
*/

/*
Application of Free Software and Open Hardware,
PSSOH 2019, International Conference,
University of Belgrade -- School of Electrical Engineering,
Belgrade, Serbia, October 26, 2019. http://pssoh.etf.bg.ac.rs
*/

/*
SALECx in a Nutshell
*/
```

```
/*
SALECx is a Maxima program for solving linear time-invariant
finite electric circuits in the complex domain of
the Unilateral Laplace Transform or Phasor Transform.
*/

/*
Algorithm
*/

/*
SALECx uses Modified Nodal Analysis (MNA) to formulate
equations and solve circuits.
*/

/*
One node, referred to as the reference node is
labeled by zero, 0. Other nodes are labeled by
consecutive integers starting from one, 1.
*/

/*
For all nodes except the reference node, Node 0,
SALECx formulates the Kirchhoff's current law
(KCL) equations. The reference direction for current
is OUT OF the node (leaving the node).
*/

/*
The currents are expressed in terms of node voltages.
The node voltage of the reference node is set to zero, 0.
*/

/*
If a current cannot be expressed in terms of node voltages
then the current becomes a MNA variable and the corresponding
element equation is added to the system of the MNA equations
*/

/*
MNA variables are node voltages, V[1], V[2], V[3], ...
and currents of the ports which are not voltage controlled,
i.e. the currents which cannot be expressed in terms of
node voltages. These currents are labeled by I["id"] or
I["id",pin] where "id" uniquely specifies a circuit element
and pin stands for an integer assigned to a circuit node,
1, 2, 3, ...
*/
```

```
/*
Reserved symbols
*/

/*
s -- complex frequency, the Laplace variable [radian/second]
*/

/*
I -- MNA current variables
I[label] or I[label,node]
*/

/*
V -- MNA voltage variables, node voltages
V[1], V[2], V[3] ...
V[0] is set to 0
*/

/*
Units
*/

/*
All quantities are assumed to be in SI units,
the International System of Units (SI), adopted by
the General Conference on Weights and Measures in 1960.
*/

/*
Electric Circuit Specification
*/

/*
The circuit to be analyzed is specified as a list
 [circuitElement_1, circuitElement_2 ..., circuitElement_N].
*/

/*
A circuit element is specified as a list of the form
 [type, label, a, b, p]
 [type, label, a, b, p, IC]
 [type, label, [a1,a2], b]
 [type, label, [a1,a2], [b1,b2], p]
 [type, label, [a1,a2], [b1,b2], p, IC]
*/

```

```
/*
type -- string that specifies the element type:
 "R", "L", "C", "I", "V", "Z", "Y", "OpAmp",
 "VCVS", "VCCS", "CCCS", "CCVS", "IT", "K", "T".
*/

/*
label -- string that uniquely identifies circuit element, e.
 "Vgen", "Isource", "Rin", "Cfb", "Lprimary", "Zload", etc.
*/

/*
one-port element
 a -- positive terminal
 b -- negative terminal
*/

/*
two-port element
 a1 -- positive terminal of the 1st port
 a2 -- negative terminal of the 1st port
 b1 -- positive terminal of the 2nd port
 b2 -- negative terminal of the 2nd port
*/

/*
p -- parameter or parameters if p is list
*/

/*
IC -- initial conditions at 0-minus
 Vo for capacitors
 Io for inductors
 [Io1,Io2] for linear inductive transformers
*/

/*
Element Catalog
*/

/*
One-port elements
*/

/*
Resistor
["R", "id", plusTerminal, minusTerminal, resistance]
*/
```

```

/*
Inductor
["L", "id", plusTerminal, minusTerminal, inductance]
["L", "id", plusTerminal, minusTerminal, inductance, Io]
Io -- initial condition, initial current at 0-minus
from plusTerminal, across the element, to minusTerminal
*/

/*
Capacitor
["C", "id", plusTerminal, minusTerminal, capacitance]
["C", "id", plusTerminal, minusTerminal, capacitance, Vo]
Vo -- initial condition, initial voltage at 0-minus
Vo = V[plusTerminal] - V[minusTerminal]
*/

/*
Current source (ideal independent current generator)
["I", "id", plusTerminal, minusTerminal, excitation]
excitation is the source (generator) current
from plusTerminal, across the element, to minusTerminal
*/

/*
Voltage source (ideal independent voltage generator)
["V", "id", plusTerminal, minusTerminal, excitation]
excitation is the source (generator) voltage
voltage = V[plusTerminal] - V[minusTerminal]
*/

/*
Impedance
["Z", "id", plusTerminal, minusTerminal, impedance]
*/

/*
Admitance
["Y", "id", plusTerminal, minusTerminal, admittance]
*/

/*
Operational Amplifier
*/

/*
Operational Amplifier (Ideal OpAmp)
["OpAmp", "id", [nonInvertingTerminal, invertingTerminal], ou
```

```
I["id"] is current into outputTerminal, MNA current variable
*/

/*
Controlled Sources
*/

/*
["VCVS", "id", [plusControllingTerminal, minusControllingTerm
 [plusControlledTerminal, minusControlledTerminal], voltageGa
I["id"] is current into plusControlledTerminal, MNA current v
*/

/*
["VCCS", "id", [plusControllingTerminal, minusControllingTerm
 [plusControlledTerminal, minusControlledTerminal], transcond
*/

/*
["CCCS", "id", [plusControllingTerminal, minusControllingTerm
 [plusControlledTerminal, minusControlledTerminal], currentGa
I["id"] is current into plusControllingTerminal, MNA current
*/

/*
["CCVS", "id", [plusControllingTerminal, minusControllingTerm
 [plusControlledTerminal, minusControlledTerminal], transresi
I["id"] is current into plusControlledTerminal, MNA current v
*/

/*
Transformers
*/

/*
Ideal Transformer
["IT", "id", [plusPrimaryTerminal, minusPrimaryTerminal],
 [plusSecondaryTerminal, minusSecondaryTerminal], turnsRatio]
I["id"] is current into plusPrimaryTerminal, MNA current vari
*/

/*
Linear Inductive Transformer
["K", "id", [plusPrimaryTerminal, minusPrimaryTerminal],
 [plusSecondaryTerminal, minusSecondaryTerminal], [L1,L2,L12]
["K", "id", [plusPrimaryTerminal, minusPrimaryTerminal],
 [plusSecondaryTerminal, minusSecondaryTerminal], [L1,L2,L12]
I["id",plusPrimaryTerminal] is
```

```
 current into plusPrimaryTerminal, MNA current variable
I["id",plusSecondaryTerminal] is
 current into plusSecondaryTerminal, MNA current variable
*/

/*
ABCD two-port
*/

/*
["ABCD", "id", [plusPrimaryTerminal, minusPrimaryTerminal],
 [plusSecondaryTerminal, minusSecondaryTerminal], [[A,B],[C,I
I["id",plusPrimaryTerminal] current into plusPrimaryTerminal
I["id",plusSecondaryTerminal] current OUT OF plusSecondaryTe
*/

/*
Transmission lines
*/

/*
Transmission Line, Phasor Transform
["T", "id", [plusSendingTerminal, minusSendingTerminal],
 [plusReceivingTerminal, minusReceivingTerminal], [Zc,theta]]
theta [radian] -- electrical length
I["id",plusSendingTerminal] current into plusSendingTerminal
I["id",plusReceivingTerminal] current OUT OF plusReceivingTe
*/

/*
Transmission Line, Laplace Transform
["T", "id", [plusSendingTerminal, minusSendingTerminal],
 [plusReceivingTerminal, minusReceivingTerminal], [Zc,tau]]
tau [second] -- delay (one-way time delay)
I["id",plusSendingTerminal] current into plusSendingTerminal
I["id",plusReceivingTerminal] current into plusReceivingTerm
*/

/*
Calling SALECx
*/

/*
Laplace Transform s-domain
 SALECx[circuitSpecification]
*/

/*
```

```
Phasor Transform j*omega-domain, sinusoidal steady state
 SALECx[circuitSpecification, omegaPhasorTransform]
omegaPhasorTransform [radian] -- angular frequency
*/

/*
Options
*/

/*
Return only the response
 SALECxPrint: false
*/

/*
Return some analysis details and the response
 SALECxPrint: true
*/

/*
Declaration and Initialization
*/

/*
Declare complex domain
 domain: complex$
*/

/*
Remove values of symbols, e.g.
 remvalue(Ig, s, Vg, Z, Yeq)$
*/

/*
Declare complex variables, e.g.
 declare([Ig, s, Vg, Z, Yeq], complex)$
*/

/*
Declare real variables, e.g.
 declare([Cload, L12, R, Vgeff, omega1], real)$
*/

/*
Declare integer variables, e.g.
 declare(nHarmonic, integer)$
*/

```

```
/*
Make assumptions, e.g.
 assume(C > 0, L2 > 0, Vgeff > 0, notequal(m, 0), n > -1)$
*/

/*
Introduce aliases, e.g.
 alias(j, %i)$
*/

/*
Circuit Graph Assumption
*/

/*
The electric circuit graph is assumed to be connected.
*/

/*
If the graph is not connected then
(1) identify the disconnected components,
(2) choose one node in each component, and
(3) connect the chosen nodes to make the graph connected.
*/

/*
The refence node (ground) is numbered by zero, 0.
The other nodes are numbered by consecutive integers starting
*/

/*
References
*/

/*
Classic
*/

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

/*
Leon O. Chua, Charles A. Desoer, and Ernest S. Kuh,
Linear and nonlinear circuits, New York, NY, McGraw-Hill, 198
*/

```

```
/*
General
*/

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

/*
James W. Nilsson, Susan A. Riedel,
Electric Circuits, 10/e, Upper Saddle River, NJ, Prentice Ha
*/

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

/*
James A. Svoboda, Richard C. Dorf,
Introduction to Electric Circuits, 9/e, Hoboken, NJ, Wiley, 2
*/

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

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

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

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

/*
Power Engineering
*/
```

```

/*
Arieh L. Shenkman, Transient Analysis of Electric Power Circu
Springer,  Dordrecht, The Netherlands, 2005.
*/

/*
Arieh L. Shenkman, Circuit Analysis for Power Engineering Har
Springer, Dordrecht, The Netherlands, 1998.
*/

/*
Transmission Lines
*/

/*
Paul R. Clayton, Analysis of Multiconductor Transmission Line
Hoboken, NJ, Wiley IEEE Press, 2008.
*/

/*
ElementStamp (subprogram)
*/

ElementStamp(e_) := block([type_, label_,
a_, b_, p_, IC_:0, Zc_, theta_, supported_:true],

if length(e_) = 4 then
 [type_, label_, a_, b_]: e_
elseif length(e_) = 5 then
 [type_, label_, a_, b_, p_]: e_
else
 [type_, label_, a_, b_, p_, IC_]: e_,

if type_ = "R"
 then (J[a_]: J[a_] + (V[a_]-V[b_])/p_,
       J[b_]: J[b_] + (V[b_]-V[a_])/p_)

elseif type_ = "L"
 then (if PhasorTransform_ then IC_: 0,
       J[a_]: J[a_] + (V[a_]-V[b_])/(s*p_) + IC_/s,
       J[b_]: J[b_] + (V[b_]-V[a_])/(s*p_) - IC_/s)

elseif type_ = "C"
 then (if PhasorTransform_ then IC_: 0,
       J[a_]: J[a_] + p_*s*(V[a_]-V[b_]) - p_*IC_,
       J[b_]: J[b_] + p_*s*(V[b_]-V[a_]) + p_*IC_)

```

```
elseif type_ = "I"
 then (J[a_]: J[a_] + p_,
       J[b_]: J[b_] - p_)

elseif type_ = "V"
 then (J[a_]: J[a_] + I[label_],
       J[b_]: J[b_] - I[label_],
       JJ: cons(V[a_]-V[b_]=p_, JJ),
       VV: cons(I[label_], VV) )

elseif type_ = "VCVS"
 then (a1_: first(a_), a2_: second(a_),
       b1_: first(b_), b2_: second(b_),
       J[b1_]: J[b1_] + I[label_],
       J[b2_]: J[b2_] - I[label_],
       JJ: cons(V[b1_]-V[b2_]=p_*(V[a1_]-V[a2_]),
       JJ),
       VV: cons(I[label_], VV) )

elseif type_ = "VCCS"
 then (a1_: first(a_), a2_: second(a_),
       b1_: first(b_), b2_: second(b_),
       J[b1_]: J[b1_] + p_*(V[a1_]-V[a2_]),
       J[b2_]: J[b2_] - p_*(V[a1_]-V[a2_]) )

elseif type_ = "CCCS"
 then (a1_: first(a_), a2_: second(a_),
       b1_: first(b_), b2_: second(b_),
       J[a1_]: J[a1_] + I[label_],
       J[a2_]: J[a2_] - I[label_],
       J[b1_]: J[b1_] + p_*I[label_],
       J[b2_]: J[b2_] - p_*I[label_],
       JJ: cons(V[a1_]-V[a2_]=0, JJ),
       VV: cons(I[label_], VV) )

elseif type_ = "CCVS"
 then (a1_: first(a_), a2_: second(a_),
       b1_: first(b_), b2_: second(b_),
       J[a1_]: J[a1_] + (V[b1_]-V[b2_])/p_,
       J[a2_]: J[a2_] - (V[b1_]-V[b2_])/p_,
       J[b1_]: J[b1_] + I[label_],
       J[b2_]: J[b2_] - I[label_],
       JJ: cons(V[a1_]-V[a2_]=0, JJ),
       VV: cons(I[label_], VV) )

elseif type_ = "IT"
 then (a1_: first(a_), a2_: second(a_),
       b1_: first(b_), b2_: second(b_),
```

```
        J[a1_]: J[a1_] + I[label_],
        J[a2_]: J[a2_] - I[label_],
        J[b1_]: J[b1_] + (-p_)*I[label_],
        J[b2_]: J[b2_] - (-p_)*I[label_],
        JJ: cons(V[a1_]-V[a2_]=p_*(V[b1_]-V[b2_]),
        JJ),
        VV: cons(I[label_], VV) )

elseif type_ = "OpAmp"
 then (a1_: first(a_), a2_: second(a_),
        J[b_]: J[b_] + I[label_],
        JJ: cons(V[a1_]-V[a2_]=0, JJ),
        VV: cons(I[label_], VV) )

elseif type_ = "K"
 then (if PhasorTransform_ then IC_: [0,0],
        [L1_, L2_, L12_]: p_, [I01_, I02_]: IC_,
        a1_: first(a_), a2_: second(a_),
        b1_: first(b_), b2_: second(b_),
        J[a1_]: J[a1_] + I[label_, a1_],
        J[a2_]: J[a2_] - I[label_, a1_],
        J[b1_]: J[b1_] + I[label_, b1_],
        J[b2_]: J[b2_] - I[label_, b1_],
        JJ: cons(V[a1_]-V[a2_] =
         L1_*s*I[label_,a1_] - L1_*I01_ +
         L12_*s*I[label_,b1_] - L12_*I02_,
        JJ),
        JJ: cons(V[b1_]-V[b2_] =
         L12_*s*I[label_,a1_] - L12_*I01_ +
         L2_*s*I[label_,b1_] - L2_*I02_,
        JJ),
        VV: cons(I[label_, a1_], VV),
        VV: cons(I[label_, b1_], VV)
  )

elseif type_ = "Z"
 then (J[a_]: J[a_] + (V[a_]-V[b_])/p_,
        J[b_]: J[b_] + (V[b_]-V[a_])/p_)

elseif type_ = "Y"
 then (J[a_]: J[a_] + (V[a_]-V[b_])*p_,
        J[b_]: J[b_] + (V[b_]-V[a_])*p_)

elseif type_ = "ABCD"
 then ([a11_,a12_]: first(p_), [a21_,a22_]: second(p_),
        a1_: first(a_), a2_: second(a_),
        b1_: first(b_), b2_: second(b_),
        J[a1_]: J[a1_] + I[label_, a1_],
```

```
        J[a2_]: J[a2_] - I[label_, a1_],
        J[b1_]: J[b1_] - I[label_, b1_],
        J[b2_]: J[b2_] + I[label_, b1_],
        JJ: cons(V[a1_]-V[a2_] =
         a11_*(V[b1_]-V[b2_]) + a12_*I[label_, b1_],
        JJ),
        JJ: cons(I[label_, a1_] =
         a21_*(V[b1_]-V[b2_]) + a22_*I[label_, b1_],
        JJ),
        VV: cons(I[label_, a1_], VV),
        VV: cons(I[label_, b1_], VV)
  )

elseif type_ = "T" and PhasorTransform_
 then (Zc_: first(p_), theta_: second(p_),
        a1_: first(a_), a2_: second(a_),
        b1_: first(b_), b2_: second(b_),
        J[a1_]: J[a1_] + I[label_, a1_],
        J[a2_]: J[a2_] - I[label_, a1_],
        J[b1_]: J[b1_] - I[label_, b1_],
        J[b2_]: J[b2_] + I[label_, b1_],
        JJ: cons(V[a1_]-V[a2_] =
         cos(theta_)*(V[b1_]-V[b2_]) +
         %i*Zc_*sin(theta_)*I[label_, b1_],
        JJ),
        JJ: cons(I[label_, a1_] =
         %i*(1/Zc_)*sin(theta_)*(V[b1_]-V[b2_]) +
         cos(theta_)*I[label_, b1_],
        JJ),
        VV: cons(I[label_, a1_], VV),
        VV: cons(I[label_, b1_], VV)
  )

elseif type_ = "T"
 then (Zc_: first(p_), tau_: second(p_),
        a1_: first(a_), a2_: second(a_),
        b1_: first(b_), b2_: second(b_),
        J[a1_]: J[a1_] + I[label_, a1_],
        J[a2_]: J[a2_] - I[label_, a1_],
        J[b1_]: J[b1_] + I[label_, b1_],
        J[b2_]: J[b2_] - I[label_, b1_],
        JJ: cons(V[a1_]-V[a2_] =
         Zc_*I[label_, a1_] +
         Zc_*I[label_, b1_]*exp(-tau_*s)+
         (V[b1_]-V[b2_])*exp(-tau_*s),
        JJ),
        JJ: cons(V[b1_]-V[b2_] =
         Zc_*I[label_, b1_] +
```

```
         Zc_*I[label_, a1_]*exp(-tau_*s)+
         (V[a1_]-V[a2_])*exp(-tau_*s),
        JJ),
        VV: cons(I[label_, a1_], VV),
        VV: cons(I[label_, b1_], VV)
   )

else supported_: false,

supported_) $

/*
SALECx (main program)
*/

SALECx(circuit_, [w_]) := block([i_, n_],
 if w_=[] then PhasorTransform_: false
          else PhasorTransform_: true,

 if w_#[] then
  print("Phasor Transform at angular frequency ", first(w_)),
 if w_=[] then remvalue(s)
          else s: %i*first(w_),

 n_: lmax(flatten(
  map(lambda([x], part(x,[3,4])), circuit_)
 )),

 elementValues_: map(lambda([x],
  if length(x)>4 then part(x,5) else false), circuit_
 ),

 initialConditions_: map(lambda([x],
  if length(x)=6 then part(x,6) else false), circuit_
 ),

 remvalue(I, J, JJ, V, VV),
 for i_: 0 thru n_ do J[i_]: 0,
 JJ: [],
 V[0]: 0,
 potentials_: makelist(V[i_], i_, n_),
 VV: [],

 m_: map(ElementStamp, circuit_),

 equationsVn_: makelist(J[i]=0, i, n_),
 equationsMNA_: append(equationsVn_, JJ),

```

```
 variablesMNA_: append(potentials_, VV),

 responseMNA_: linsolve(equationsMNA_, variablesMNA_),

 if SALECxPrint then (
 print("Symbolic Analysis of Linear Electric Circuits with Ma
 print("SALECx version 1.0, Prof. Dr. Dejan Tošić, tosic@etf
 print("Number of nodes excluding 0 node: ", n_),
 print("Electric circuit specification:", circuit_),
 print("Supported element: ", m_),
 print("Element values: ", elementValues_),
 print("Initial conditions: ", initialConditions_),
 print("MNA equations: ", equationsMNA_),
 print("MNA variables: ", variablesMNA_)
 ),

responseMNA_) $

/*
SALECxPrint (reserved symbol, verbose option)
*/

SALECxPrint: false $

print("Dejan Tosic, SALECx 2019 v1.0");
print("Symbolic Analysis of Linear Electric Circuits with Ma
```