PROGRAMMATIC SYMBOLIC CIRCUIT ANALYSIS

BACHELOR'S PROJECT
23 JUNE 2022

NICKLAS VRAA
STUDENT

OUTLINE

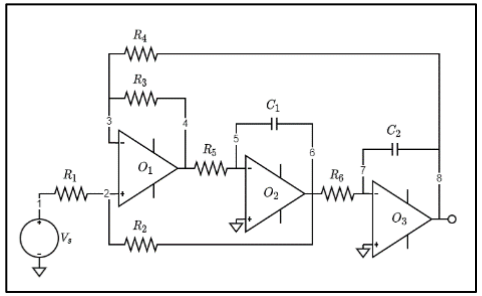
- Motivation
- Methodology, The programmatic approach
- Basic use, Circuit objects
- The ELAB class
- Modified Nodal Analysis by example
- Key features, Symbolic analysis, Transmuting
- Digitizing the standard toolset
- The project in action
- Conclusion and future opportunities

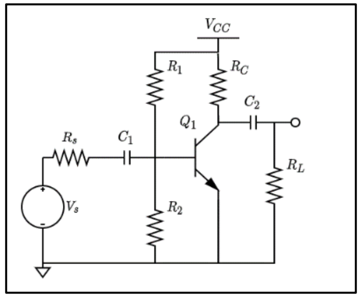
MOTIVATION

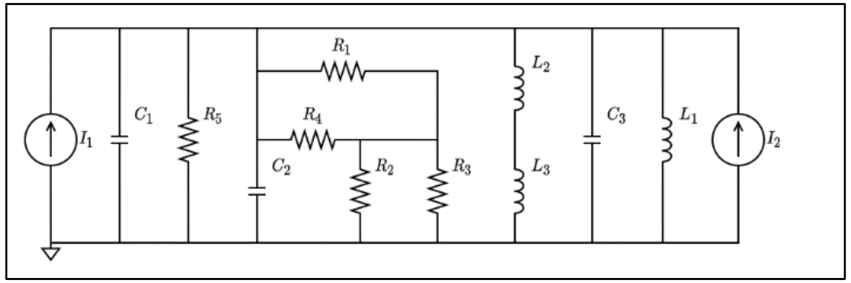
Complicated problems
- Time-consuming
- Repetitive in nature

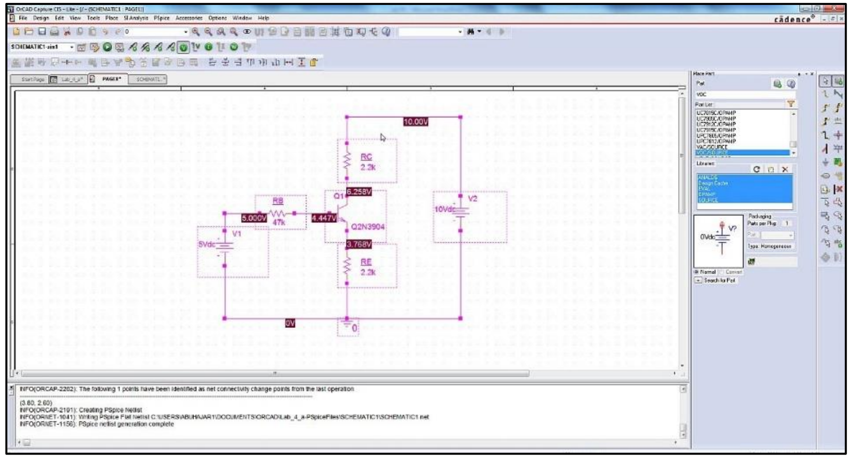
Heavy software
- Hard to install and run
- Unfamiliar environment
- Focus on numerical simulation, Assumes knowledge of values
- Limited understanding to be gained

Symbolic vs Numerical Analysis
- Symbolic analysis increases understanding but is time-consuming.
- Numerical analysis is near-instant but provides to no semantic understanding.









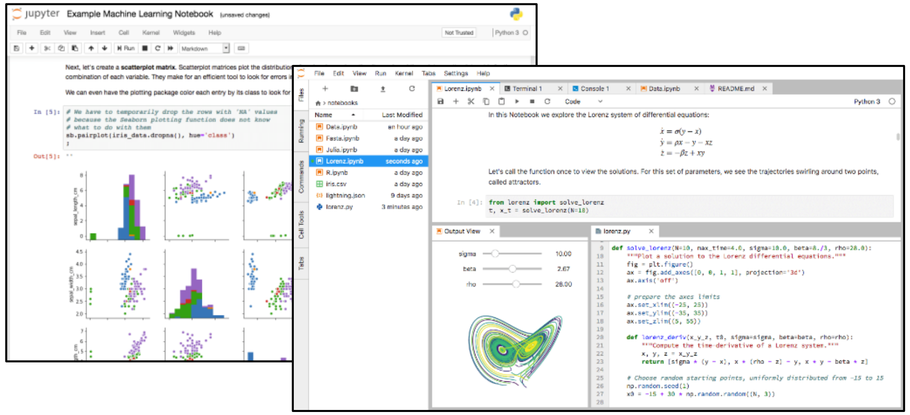
NEW APPROACH

Inspiration
- The field of data science, control systems, etc.
- Use pre-existing generalized tools
- Practically no standalone software

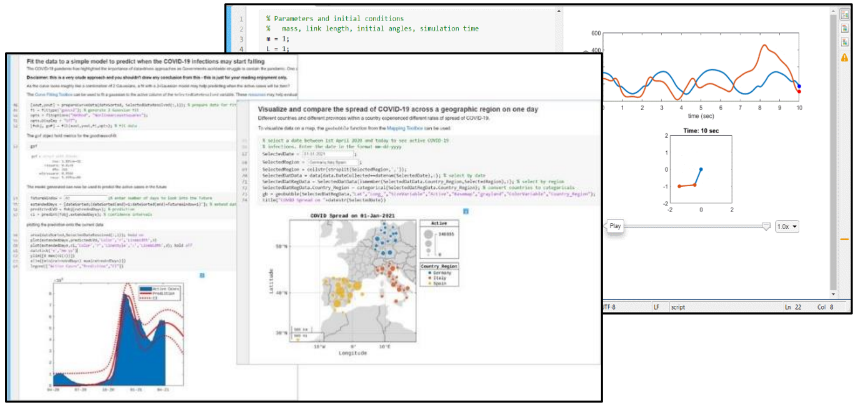
The "programmatic" approach
- A general skill with infinite versatility
- Unbound by isolated software
- Choice of environment

Why a MATLAB toolbox?
- Live script, formatted output
- Symbolic math toolbox
- Pre-existing systems analysis tools
- Familiarity, rapid prototyping
- The industry standard, very fast

Jupyter Notebooks / Python for Data Science



Live Scripts / MATLAB for control systems, statistics



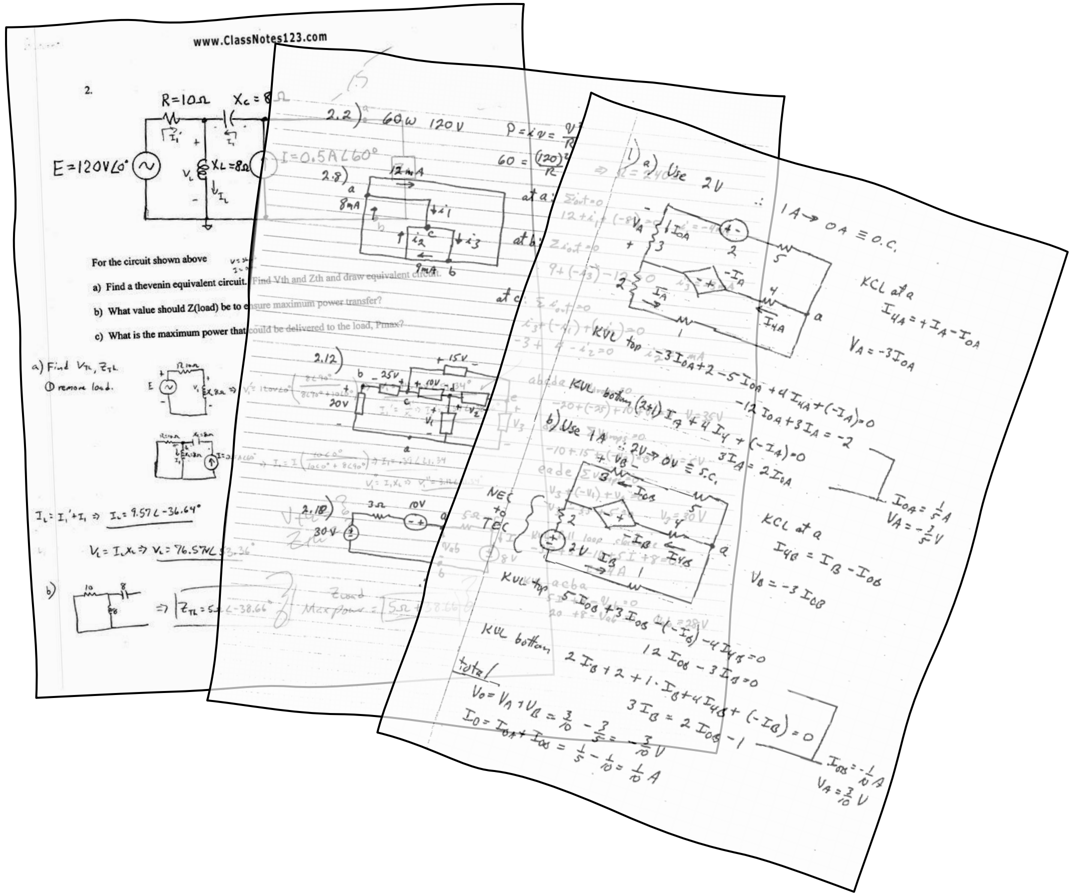
THE KEY IDEA

ELABorate

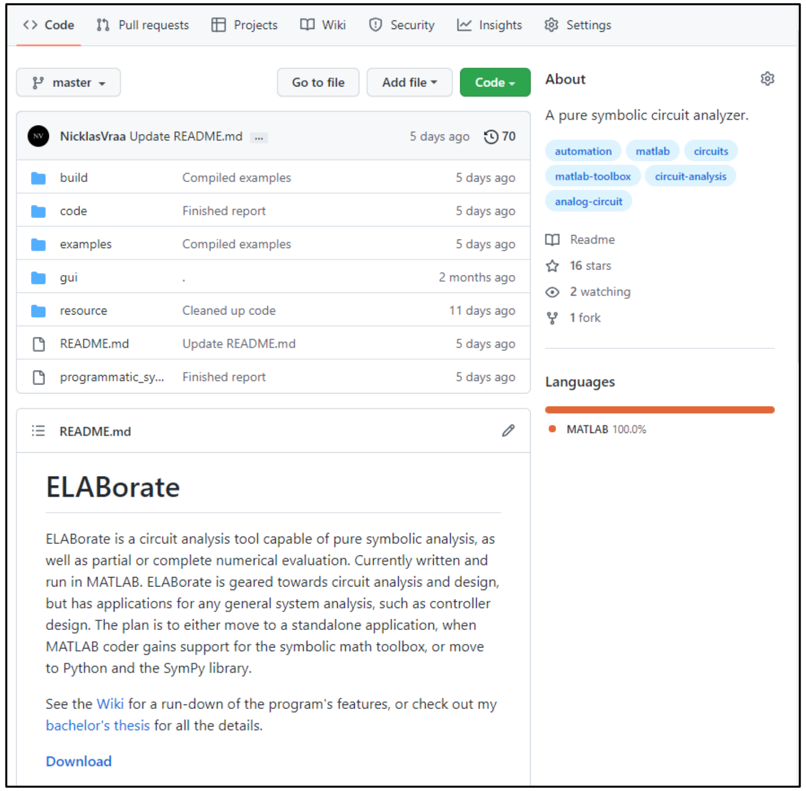
Electronic **Lab**oratory

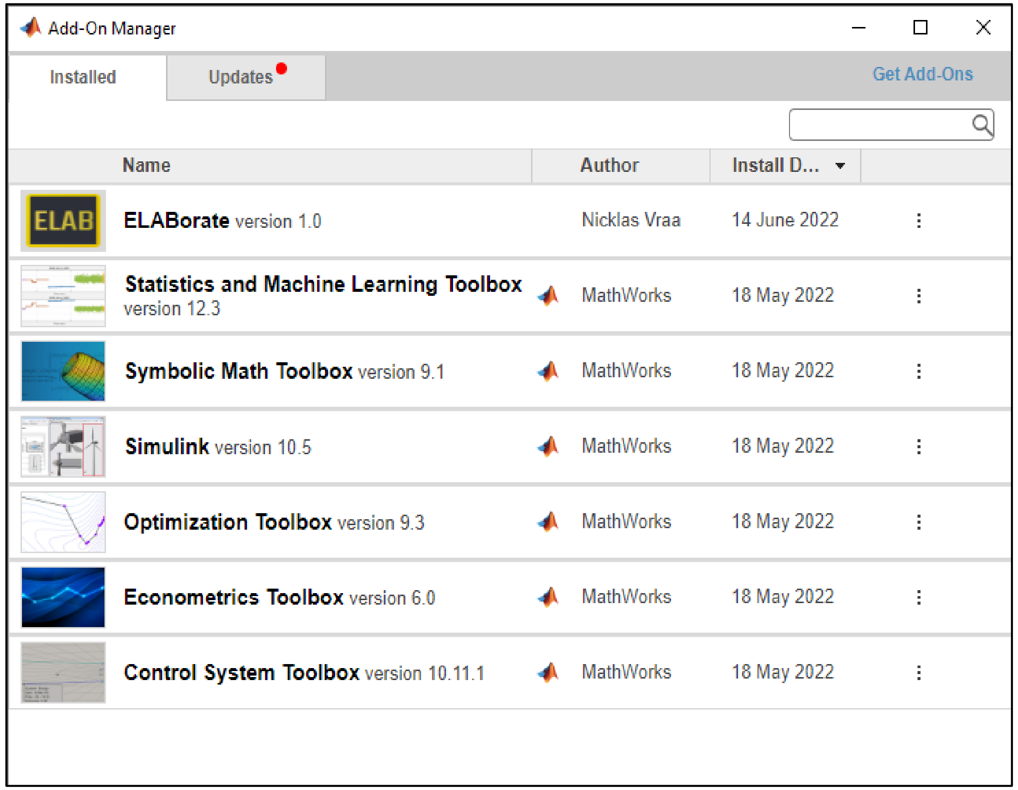
Why?

- Digitalizing our tools and research subjects
- Symbolically, we are still stuck with pen-and-paper

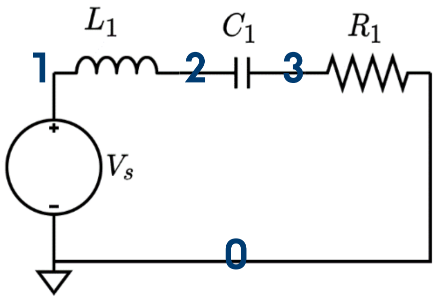


THE RESULTS


Code
Pull requests
Projects
Wiki
Security
Insights
Settings
master
Go to file
Add file
Code
About
A pure symbolic circuit analyzer.
automation
matlab
circuits
matlab-toolbox
circuit-analysis
analog-circuit
Readme
16 stars
2 watching
1 fork
Languages
MATLAB 100.0%
NicklasVraa Update README.md
5 days ago
70
build
Compiled examples
5 days ago
code
Finished report
5 days ago
examples
Compiled examples
5 days ago
gui
2 months ago
resource
Cleaned up code
11 days ago
README.md
Update README.md
5 days ago
programmatic_sy...
Finished report
5 days ago
README.md
ELABorate
ELABorate is a circuit analysis tool capable of pure symbolic analysis, as well as partial or complete numerical evaluation. Currently written and run in MATLAB. ELABorate is geared towards circuit analysis and design, but has applications for any general system analysis, such as controller design. The plan is to either move to a standalone application, when MATLAB coder gains support for the symbolic math toolbox, or move to Python and the SymPy library.
See the Wiki for a run-down of the program's features, or check out my bachelor's thesis for all the details.
Download


Add-On Manager
Installed
Updates
Get Add-Ons
Name
Author
Install D...
ELAB
ELABorate version 1.0
Nicklas Vraa
14 June 2022
Statistics and Machine Learning Toolbox version 12.3
MathWorks
18 May 2022
Symbolic Math Toolbox version 9.1
MathWorks
18 May 2022
Simulink version 10.5
MathWorks
18 May 2022
Optimization Toolbox version 9.3
MathWorks
18 May 2022
Econometrics Toolbox version 6.0
MathWorks
18 May 2022
Control System Toolbox version 10.11.1
MathWorks
18 May 2022

BASIC USE



rlc_series.txt

```
Vs 1 0 DC 5
L1 1 2 1
C1 2 3 0.0001
R1 3 0 1000
```

rlc_circuits.mlx

```
circuit = Circuit('circuits/rlc_series.txt');
```

Constructor in Circuit.m class

```
circuit =
  Circuit with properties:

          file_name: 'circuits/rlc_series.txt'
          num_nodes: 3
       num_elements: 4
               list: 'Vs 1 0 DC 5↵R1 3 0 1000↵  …
          Indep_VSs: [1×1 Indep_VS]
          Indep_ISs: [0×0 Indep_IS]
          Resistors: [1×1 Resistor]
          Inductors: [1×1 Inductor]
         Capacitors: [1×1 Capacitor]
       Ideal_OpAmps: [0×0 Ideal_OpAmp]
        Generic_zs: [0×0 Impedance]
             VCVSs: [0×0 VCVS]
             CCCSs: [0×0 CCCS]
             VCCSs: [0×0 VCCS]

                  ⋮
```

THE CIRCUIT CONSTRUCTOR

Data

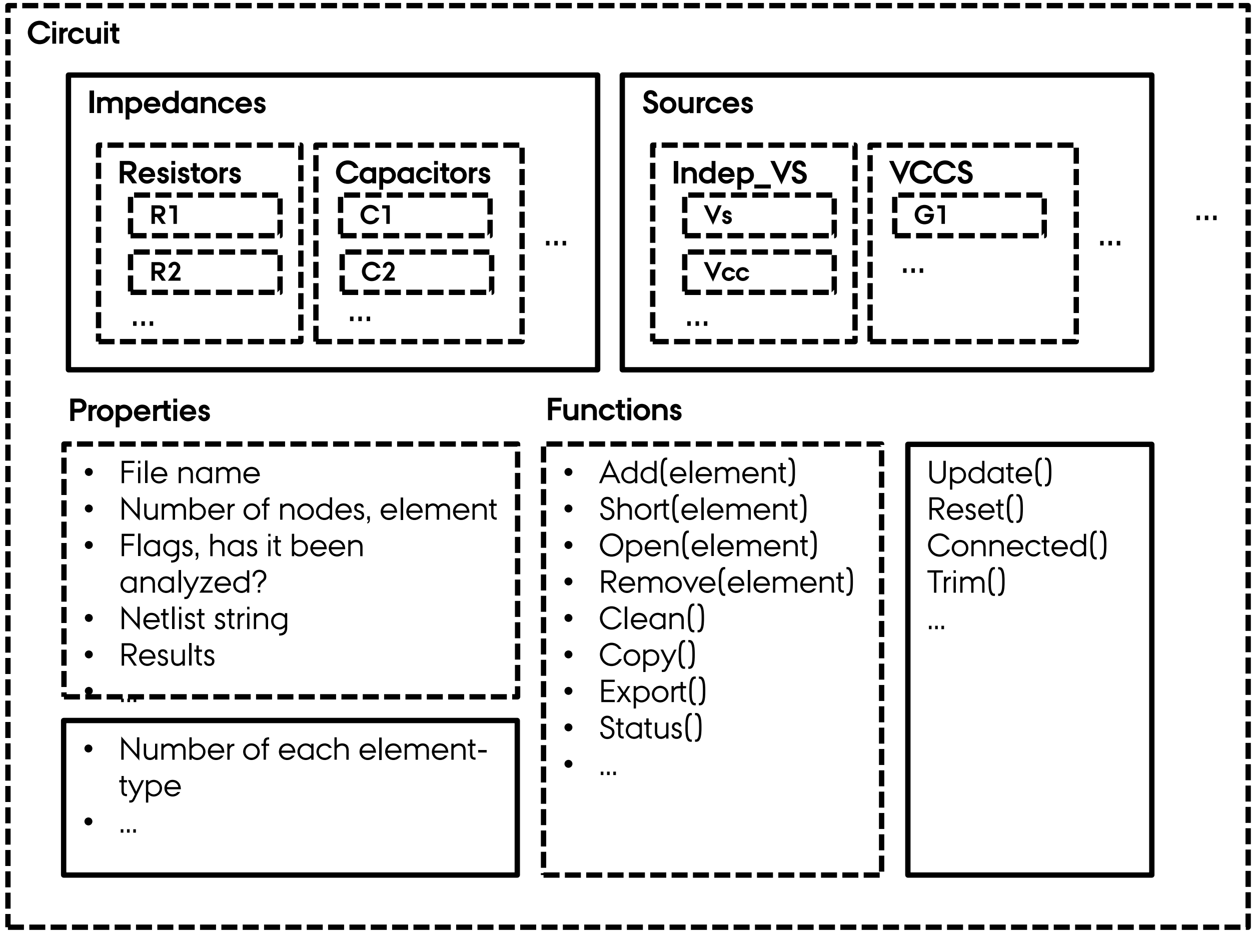
```
classdef Circuit < Base_System
% Circuit model, utilizing the Element subclasses.

    properties ...

    properties (Access = {?ELAB, ?Analyzer, ?Modeller, ?Transmuter, ?Visualizer}) ...

    methods
    % Methods pertaining to this circuit.

        function obj = Circuit(file_name)
        % Constructor of the circuit class.

            % Parse the text file at the given path (file_name).
            obj.file_name = file_name;
            file = fopen(file_name);

            % Convert to cell matrix, based on line number and spaces.
            netlist = textscan(file,'%s %s %s %s %s %s %s %s', 'CollectOutput', 1);
            obj.netlist = {netlist{1}(:,1) netlist{1}(:,2) netlist{1}(:,3) ...
                           netlist{1}(:,4) netlist{1}(:,5) netlist{1}(:,6) ...
                           netlist{1}(:,7) netlist{1}(:,8)};
            fclose(file);

            % Split cell matrix into vectors.
            [Name, N1, N2, arg3, arg4, arg5, arg6, arg7] = obj.netlist{:};

            % Convert node values from string-type to number-type.
            N1 = str2double(N1); N2 = str2double(N2);

            % Find number of elements by number of entries in the vectors.
            obj.num_elements = length(Name);
            obj.num_nodes = length(unique([N1; N2])) - 1;
```

```
    % This creates compound element arrays and handles setup.
    obj.update();
end
```

```
% Creating element objects based the parsed netlist information.
for i = 1:obj.num_elements

    % Element id is always in the same place.
    id = Name{i}(1:end);

    % First letter in ID tells the type of element.
    switch id(1)
        case {'V'}
            obj.Indep_VSs(end+1) = Indep_VS(id, N1(i), N2(i), arg3{i}, arg4{i});
        case {'I'}
            obj.Indep_ISs(end+1) = Indep_IS(id, N1(i), N2(i), arg3{i}, arg4{i});
        case {'R'}
            obj.Resistors(end+1) = Resistor(id, N1(i), N2(i), arg3{i});
        case {'L'}
            obj.Inductors(end+1) = Inductor(id, N1(i), N2(i), arg3{i});
        case {'C'}
            obj.Capacitors(end+1) = Capacitor(id, N1(i), N2(i), arg3{i});
        case {'Z'}
            obj.Impedances(end+1) = Impedance(id, N1(i), N2(i), arg3{i});
        case {'E'}
            obj.VCVSs(end+1) = VCVS(id, N1(i), N2(i), str2double(arg3{i}), ...
                                    str2double(arg4{i}), arg5{i});
        case {'G'}
            obj.VCCSs(end+1) = VCCS(id, N1(i), N2(i), str2double(arg3{i}), ...
                                    str2double(arg4{i}), arg5{i});
        case {'H'}
            obj.CCVSs(end+1) = CCVS(id, N1(i), N2(i), arg3{i}, arg4{i});
        case {'F'}
            obj.CCCSs(end+1) = CCCS(id, N1(i), N2(i), arg3{i}, arg4{i});
        case {'O'}
            obj.Ideal_OpAmps(end+1) = Ideal_OpAmp(id, N1(i), N2(i), ...
                                                  str2double(arg3{i}));
        case {'Q'}
            obj.BJTs(end+1) = BJT(id, N1(i), N2(i), str2double(arg3{i}), arg4{i});
        case {'M'}
            obj.MOSFETs(end+1) = MOSFET(id, N1(i), N2(i), ...
                                        str2double(arg3{i}), arg4{i});
    end
end
```

THE CIRCUIT OBJECT

```matlab
% Part of ELABorate, all rights reserved. ...

classdef Circuit < Base_System
% Circuit model, utilizing the Element subclasses.

    properties ...

    properties (Access = {?ELAB, ?Analyzer, ?Modeller, ?Transmuter, ?Visualizer}) ...

    methods
    % Methods pertaining to this circuit.

        function obj = Circuit(file_name) ...

        function export(obj, name) ...

        function copy = clone(obj) ...

        function status(obj) ...

        function update_netlist(obj) ...

        function short(obj, X) ...

        function open(obj, X) ...

        function clean(obj) ...

        function add(obj, X) ...

        function remove(obj, X) ...
    end

    methods(Access = {?ELAB, ?Analyzer, ?Modeller, ?Transmuter, ?Visualizer})
    % Methods only available to ELABorate modules.

        function connected = get_connected(obj, X, node) ...

        function update(obj) ...

        function reset(obj) ...

        function trim(obj) ...
    end

    methods(Access = private)
    % Methods only available to this circuit object.

        function update_num_nodes(obj) ...

        function update_nums(obj) ...

        function update_arrays(obj) ...
    end

    methods(Static)
    % Methods shared among objects of this class.

        function connected = check_elem_array(Xs, Y, node) ...
    end
end
```

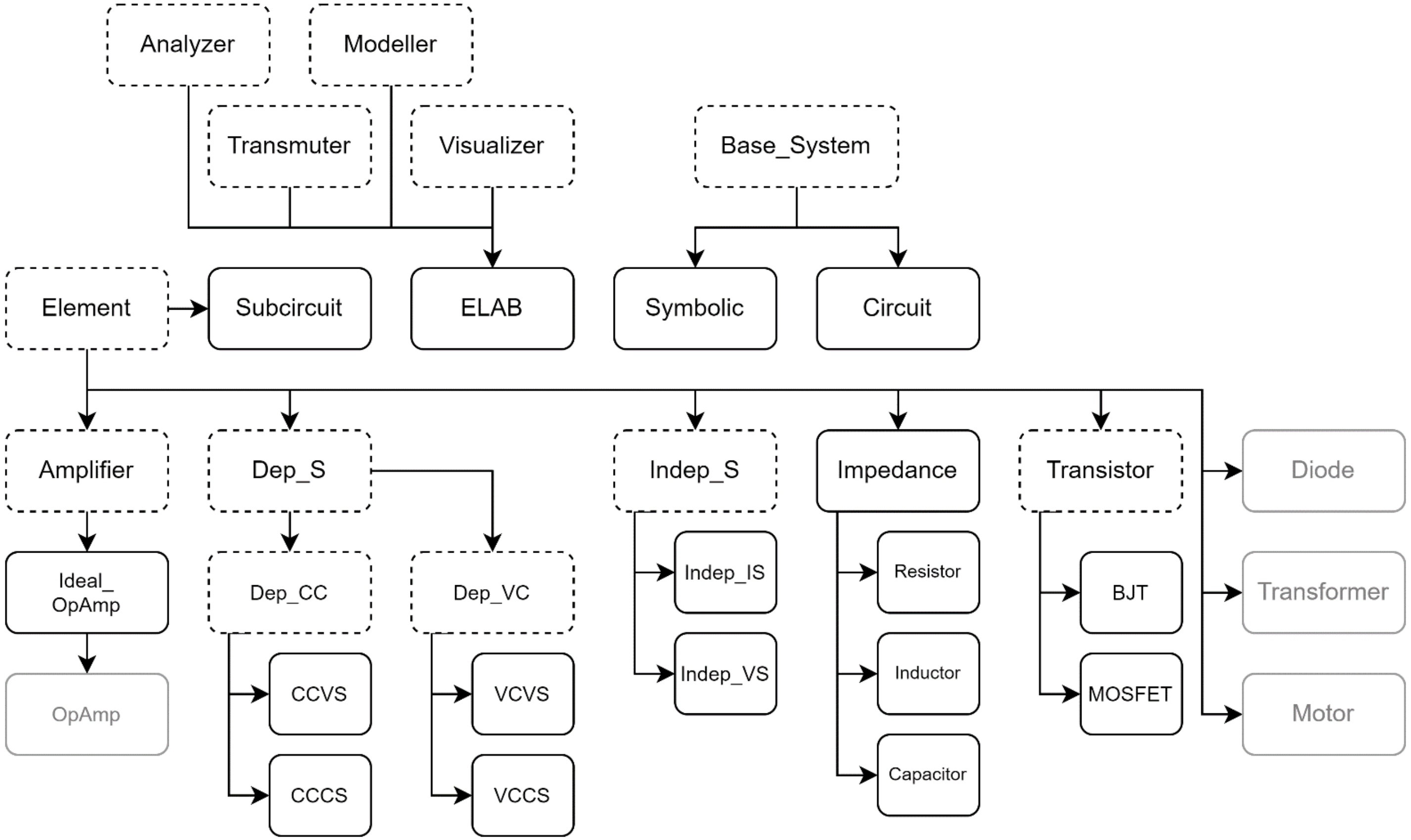


--- Accessible by user

INHERITANCE

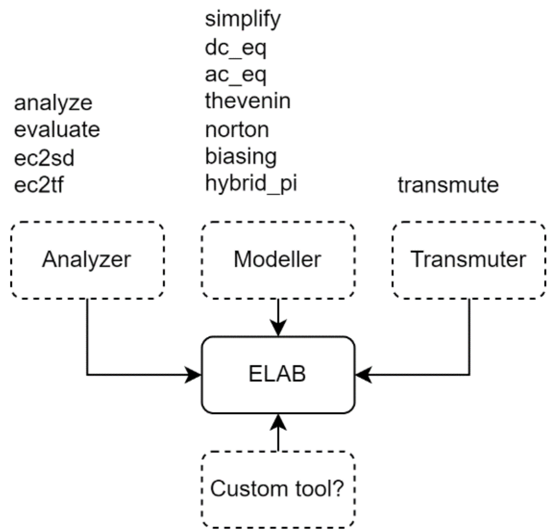
The Intent
- Have the program be communal
- Open-source, easily extended by third parties

Solution
- Rely on a purely object-oriented approach
- Modularity
- Strong inheritance tree
- Limit access to core



User need only interact with Circuit and ELAB

THE ELAB CLASS



classdef ELAB < ... & Custom

Modeller

```
classdef Modeller
% A collection of functions, specifically for safely altering the
% circuit object or even converting the circuit to another form.

    methods(Static)

        function obj = simplify(obj, z_eq) ...

        function obj = dc_eq(obj) ...

        function obj = ac_eq(obj) ...

        function obj = remove_sources(obj) ...

        function obj = thevenin(obj, load) ...

        function obj = norton(obj, load) ...

        function orig = biasing(obj) ...

        function obj = hybrid_pi(obj, freq, early) ...

        function bool = is_same_type(X1, X2) ...

        function bool = is_parallel(X1, X2) ...

        function bool = is_series(obj, X1, X2) ...

    end

    methods(Static, Access = private)

        function done = simplify_parallel(obj, z_eq) ...

        function done = simplify_series(obj, z_eq) ...

        function pair = find_parallel(~, Xs) ...

        function pair = find_series(obj, Xs) ...

        function rename(obj) ...

        function rename_eqs(Xs) ...

        function bool = check_shared(Xs, X1, X2, node) ...

        function nodes = unique_nodes(X1, X2) ...

        function [v_th, Z_th] = equivalent(obj, load) ...
    end
end
```

Analyzer

```
classdef Analyzer
% A collection of functions designed to facilitate symbolic
% and numerical analysis of a given circuit object.

    methods(Static)

        function analyze(obj) ...

        function evaluate(obj) ...

        function out = ec2sd(obj, node_in, node_out) ...

        function out = ec2tf(obj, node_in, node_out) ...

        function RA = routh(poly_coeffs, show, epsilon) ...

        function cr = critical(RA, show) ...

        function bp = breakaway(sys, gain, show) ...

        function [ord, type] = order_type(G, show) ...

        function [Kp, Kv, Ka] = static_error_K(G, show) ...

        function dp = dominant(G) ...

        function [wn, zeta] = damp(G, show) ...

        function N = period(x) ...

        function out = observable(in) ...

        function out = controllable(in) ...

        function out = jordan(in) ...
    end

    methods(Static, Access = private)

        function find_v_and_i(obj) ...
    end
end
```

Transmuter

```
classdef Transmuter
% A collection of functions designed to transmute a system into
% another form, that may be more helpful for analysis or design.

    methods(Static)

        function out = transmute(in, type_in, type_out, show) ...

        function out = sd2tf(in, show) ...

        function out = zd2tf(in, show) ...

        function out = tf2sd(in, show) ...

        function out = tf2ss(in, show) ...

        function out = ss2tf(in, show) ...

        function out = tf2zp(in, show) ...

        function out = zp2tf(in, show) ...

        function out = de2sd(in, show) ...

        function out = sd2de(in, show) ...

        function out = sd2ce(in, show) ...

        function out = ce2sd(in, show) ...

        function out = sd2td(in, show) ...

        function out = td2sd(in, show) ...

        function out = nd2zd(in, show) ...
    end
end
```

```
classdef ELAB < Analyzer & Modeller & Visualizer & Transmuter
% The combination of the main classes of the ELABorate project.
```

ANALYZER - MODIFIED NODAL ANALYSIS

The Basic MNA Algorithm (for a computer)
- Build a **L**inear **T**ime-**I**nvariant system

$$\boldsymbol{Ax} = \boldsymbol{z}$$

$$A = \begin{bmatrix} G & B \\ C & D \end{bmatrix} \quad x = \begin{bmatrix} v \\ j \end{bmatrix} \quad z = \begin{bmatrix} i \\ e \end{bmatrix}$$

```
function analyze(obj)
% Symbolic analysis of given circuit object.

    if obj.symbolically_analyzed
        return
    end

    tic;

    % Array allocation.
    G = cell(obj.num_nodes, obj.num_nodes);  [G{:}] = deal('0');
    B = cell(obj.num_nodes, obj.num_VSs);    [B{:}] = deal('0');
    C = cell(obj.num_VSs,   obj.num_nodes);  [C{:}] = deal('0');
    D = cell(obj.num_VSs,   obj.num_VSs);    [D{:}] = deal('0');
    i = cell(obj.num_nodes, 1);              [i{:}] = deal('0');
    e = cell(obj.num_VSs,   1);              [e{:}] = deal('0');
    j = cell(obj.num_VSs,   1);              [j{:}] = deal('0');
    v = compose('v_%d', (1:obj.num_nodes)');

    % Building circuit equations:
    num_vs_parsed = 0;
```

For the intermediate matrices comprising **A**:

- **G** is a diagonal matrix holding the sum of conductances of each circuit element connected between nodes i and j.
- **B** is an $n \times m$ matrix denoting the location of each voltage source. For entry $b_{i,j}$ a 0 means no voltage source, whereas a ± 1 means a voltage source. The sign indicates its orientation.
- **C** is an $m \times n$ matrix and simply the transpose of **B** so long as there are no dependent sources.
- **D** is an $m \times m$ matrix of zeros.

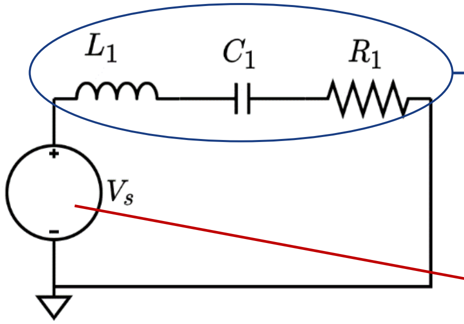
For the intermediate matrices comprising **x**:

- **v** is a vector holding the n unknown node voltages, excluding the ground voltage assumed to be 0.
- **j** is a vector holding the m unknown currents running through each voltage source in the circuit.

For the intermediate matrices comprising **z**:

- **i** is a vector holding the n sum of all current sources going into any given node.
- **e** is a vector simply holding the values of all independent current sources.

MNA- EXAMPLE



$$A = \begin{bmatrix} G & B \\ C & D \end{bmatrix} \quad x = \begin{bmatrix} v \\ j \end{bmatrix} \quad z = \begin{bmatrix} i \\ e \end{bmatrix}$$

Algorithm

```
For all impedances
  If impedance is generic or resistor, set exp = 1
  Else if impedance is capacitor, set exp = 1/s
  Else if impedance is inductor, set exp = s

  If anode is ground
    Set g_{j,j} = g_{j,j} + exp
  Else if cathode is grounded
    Set g_{i,i} = g_{i,i} + exp
  Else
    Set g_{i,i} = g_{i,i} + exp, g_{j,j} = g_{j,j} + exp, g_{i,j} = g_{i,j} + exp, g_{j,i} = g_{j,i} + exp
```

```
For all independent voltage sources
  If anode is not grounded
    Set b_{i,p} = b_{i,p} + 1, c_{p,i} = c_{p,i} + 1
  If cathode is not grounded
    Set b_{j,p} = b_{j,p} - 1, c_{p,j} = c_{p,j} - 1
Add parsed id's to e as V_id and to j as I_vid
```

i , j = anode , cathode
p = # processed v-sources

A =

$$\begin{pmatrix} \frac{1}{L_1 s} & -\frac{1}{L_1 s} & 0 & 1 \\ -\frac{1}{L_1 s} & C_1 s + \frac{1}{L_1 s} & -C_1 s & 0 \\ 0 & -C_1 s & C_1 s + \frac{1}{R_1} & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} * \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ I_{Vs} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ Vs \end{pmatrix}$$

x =

z =

Solve for x

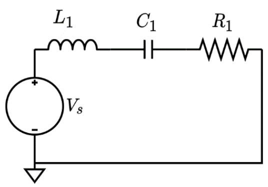
$$x = A^{-1} z$$

solutions =

$$\begin{pmatrix} v_1 = Vs \\ v_2 = \frac{Vs\,(C_1 R_1 s + 1)}{C_1 L_1 s^2 + C_1 R_1 s + 1} \\ v_3 = \frac{C_1 R_1 Vs\, s}{C_1 L_1 s^2 + C_1 R_1 s + 1} \\ I_{Vs} = -\frac{C_1 Vs\, s}{C_1 L_1 s^2 + C_1 R_1 s + 1} \end{pmatrix}$$

Circuit is fully defined.
Evaluation is easy.

KEY FEATURES




```
ELAB.analyze(circuit)
```

Symbolic analysis successful (0.300184 sec).

```
ELAB.ec2sd(circuit, 1, 3)
```

Symbolic transfer function calculated successfully (4.291000e-03 sec).
ans =

$$\frac{v_3}{v_1} = \frac{C_1\,R_1\,s}{C_1\,L_1\,s^2 + C_1\,R_1\,s + 1}$$

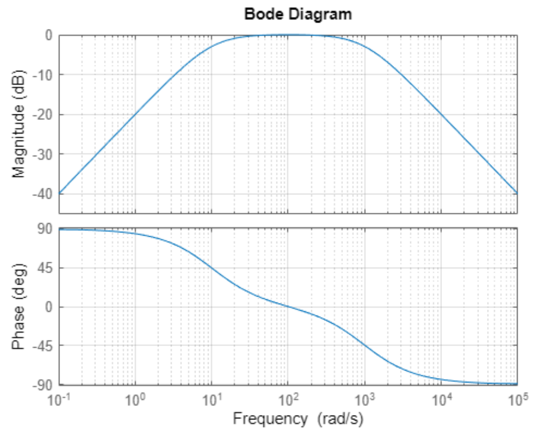
```
TF = ELAB.ec2tf(circuit, 1, 3)
```

```
Numerical evaluation successful (0.0717553 sec).
Transfer function object created successfully (1.429922e-01 sec).

TF =

        1000 s
  --------------------
  s^2 + 1000 s + 10000

Continuous-time transfer function.
```

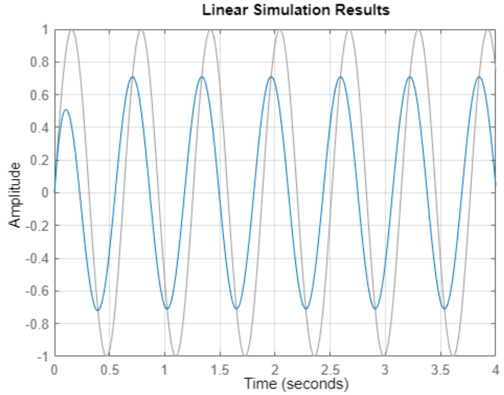
```
pole(TF), zero(TF) ...
```

```
ans = 2×1
   -989.8979
    -10.1021

ans = 0
```

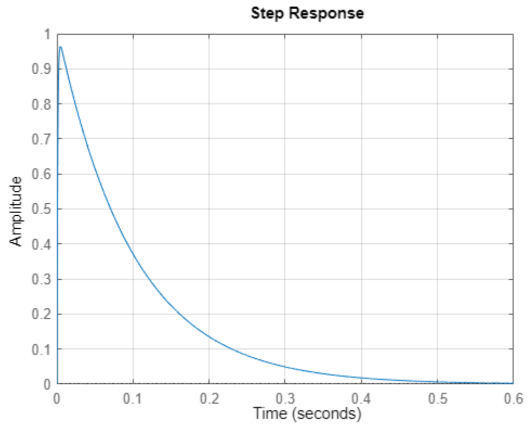
```
bode(TF); grid on;
```



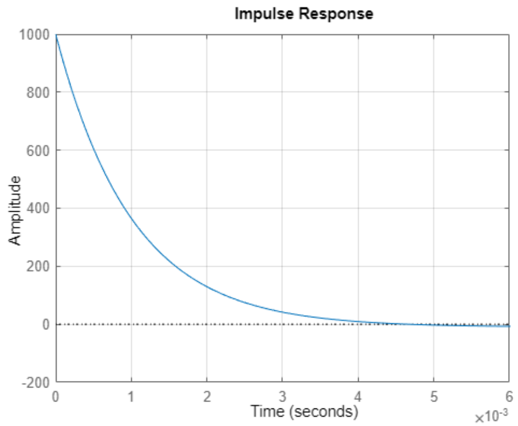

```
t = 0:0.01:4;  % Arbitrary interval.
u = sin(10*t); % Arbitrary signal.
lsim(TF,u,t); grid on
```




```
step(TF); grid on;
```

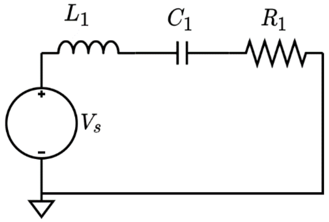



```
impulse(TF); grid on;
```



The Visualizer class should improve these outputs

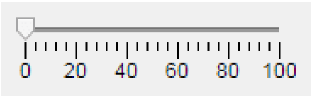
KEY FEATURES - II

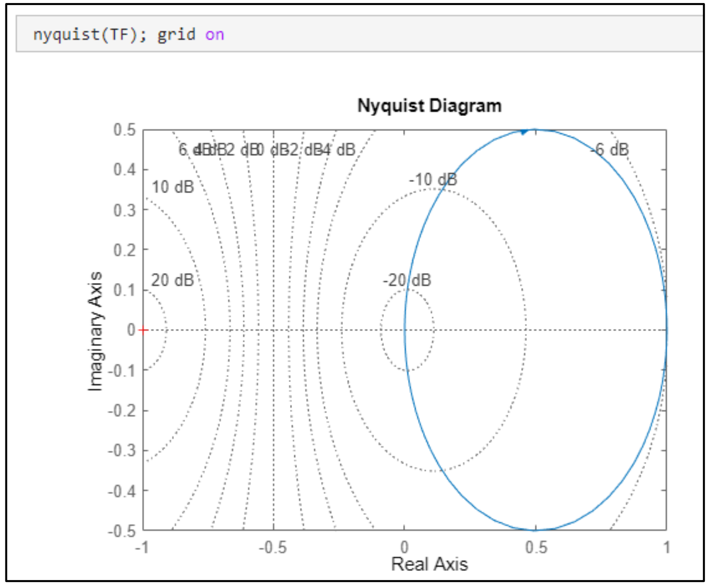


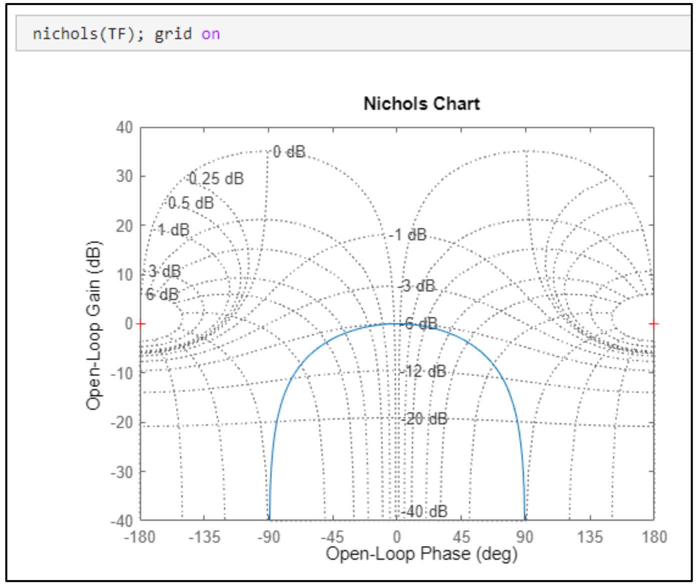
rlc_series.txt

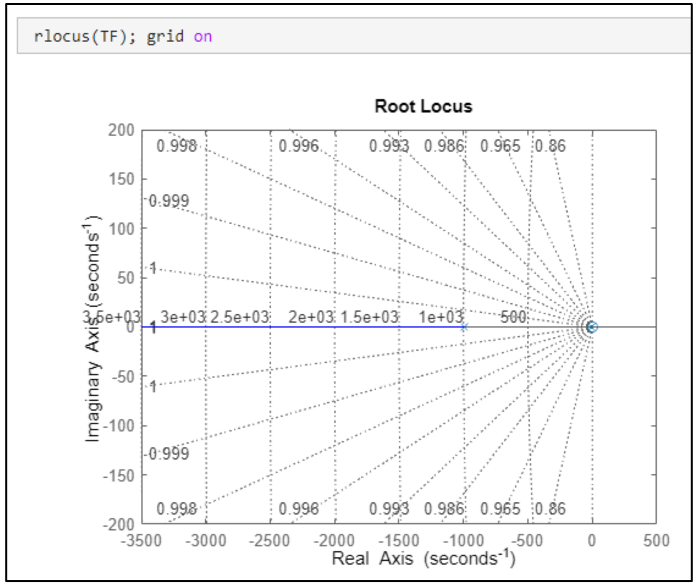
```
Vs 1 0 DC 5
L1 1 2 1
C1 2 3 0.0001
R1 3 0 1000
```

The Visualizer class should be able to manipulate the numerical values for dynamic graphs









Can be done for any circuit file

TRANSMUTING

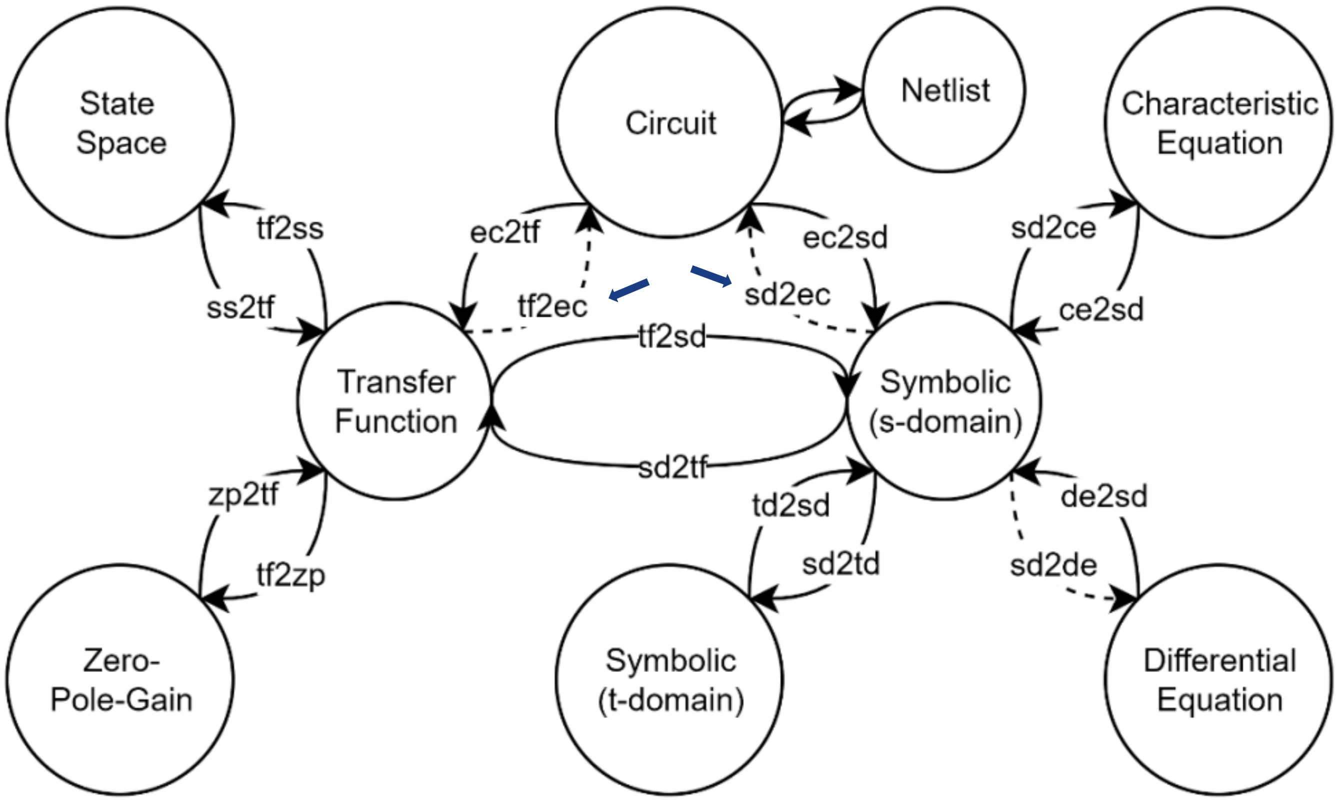


Figure 4 - Map of how the transmuter converts from one domain to another. Arrows correspond to functions.

```
syms s t y(t) Y
in = ODE(diff(y(t),2)+3*diff(y(t))+2*y(t) == 0, [0.1, 0.05]);

ELAB.transmute(in, 'de', 'td', true)
```

Diff. equation

$$\frac{\partial^2}{\partial t^2}\,y(t) + 3\,\frac{\partial}{\partial t}\,y(t) + 2\,y(t) = 0$$

Laplace transform:

$$2\,Y - \frac{s}{10} + 3\,Y\,s + Y\,s^2 - \frac{7}{20} = 0$$

Solve for tf:

$$\frac{2\,s+7}{20\,s^2+60\,s+40}$$

In s-domain

$$\frac{2\,s+7}{20\,s^2+60\,s+40}$$

Partial fraction decomp:

$$\frac{1}{4\,(s+1)} - \frac{3}{20\,(s+2)}$$

Inverse Laplace:

$$\frac{e^{-t}}{4} - \frac{3\,e^{-2\,t}}{20}$$

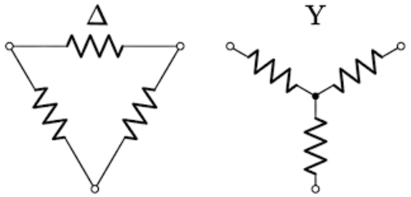
ans =

$$\frac{e^{-t}}{4} - \frac{3\,e^{-2\,t}}{20}$$

- 'sd' = Symbolic s-domain.
- 'td' = Symbolic t-domain.
- 'tf' = Transfer function.
- 'ss' = State space.
- 'zp' = Zero-pole-gain.
- 'de' = Differential equation.
- 'ce' = Characteristic equation.
- 'ec' = Electric circuit.

DIGITIZING CIRCUIT INTUITION

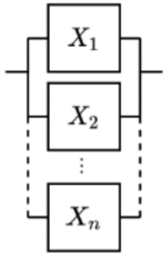
Detection
- is_parallel
- is_series
- is_same_type
- is_delta
- is_wye



ID	Series equivalent	Parallel equivalent
R	R1+R2	(R1*R2)/(R1+R2)
C	(C1*C2)/(C1+C2)	C1+C2
L	L1+L2	(L1*L2)/(L1+L2)
V	V1+v2	V1 (only when v1=v2)
I	Not allowed	I1+I2

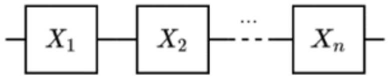
Detecting Parallels

```
Let $x_1$ and $x_2$ be circuit elements

If $z_1$'s anode = $z_2$'s anode and $z_1$'s cathode = $z_2$'s cathode
or $z_1$'s anode = $z_2$'s cathode and $z_1$'s cathode = $z_2$'s anode
  Return true
Else
  Return false
```



Detecting Series

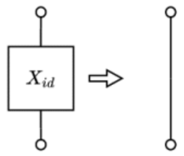
```
If parallel
  Return false
Else
  If $z_1$'s anode = $z_2$'s anode or $z_1$'s anode = $z_2$'s cathode
    Let *shared* be $z_1$'s anode
  Else if $z_1$'s cathode = $z_2$'s anode or $z_1$'s cathode = $z_2$'s cathode
    Let *shared* be $z_1$'s cathode
  Else
    Return false
  If anything else is connected to *shared*
    Return false

Return true
```



DIGITIZING CIRCUIT MANIPULATION

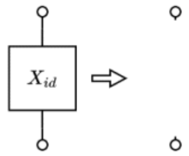
Shorting

```
Given an element x
Find node n connected to x with the lowest number.

For all terminals T1 of x
  Find all elements Y connected to t1
  For all y in Y
    For all terminals T2 of y
      If t2 is equal to t1
        Change t2 to n

Remove x from circuit
Update global properties of the circuit
```



Opening

```
Given an element x
Remove x from circuit
Let Y be all elements in the circuit
Initialize a list L of items to be removed
```



Trimming

```
For all y in Y
  Let sum = 0
  For all terminals t of y
    If node n at t is not ground
      Let Z be y's connected to n
      sum += length of Z

  If sum = 0
    Append Z to L
```

Any elements not connected to anything else

```
For all y in Y
  Let found = false
  For all terminals t1 of y
    For all terminals t2 of y excluding t1 itself
      If t2 is not equal t1
        Set found = true
        Break loop

    If not found
      Append y to L

  Remove all elements in L from circuit

Update global properties of the circuit
```

Any elements only connected to themselves

Cleaning

```
For the number of nodes in the circuit
  Let k be the current node value
  Let L be an empty list

  For all elements in the circuit
    Add all element terminal values to L

  Let m be the smallest value in L which is also above k

  For all elements in the circuit
    Let X be the current element

    For all terminals of X
      Let t be the current terminal
      If  t = m
        Set t = k+1

  Update global properties of the circuit
```

MODELLING – HIGHER LEVEL MANIPULATION

Takes advantage of short(), open(), simplify(), etc.

- dc_eq, ac_eq
- thevenin, norton
- biasing, hybrid_pi

Finding a Thevenin or Norton equivalent circuit is a widely used tool for analyzing circuit. Thevenin- and Norton-equivalents share the same initial approach.

1. **Open** the element being considered the load.
2. **Analyze** the circuit, finding the voltage across the open gap.
3. **Open** all voltage-sources and **short** all current-sources.
4. **Simplify** the circuit to a single impedance.

- For the Thevenin equivalent circuit:

  a. **Add** a voltage source with the value found in step 2.
  b. **Add** the impedance found in step 4 in series with the voltage source.
  c. **Add** back the load element.

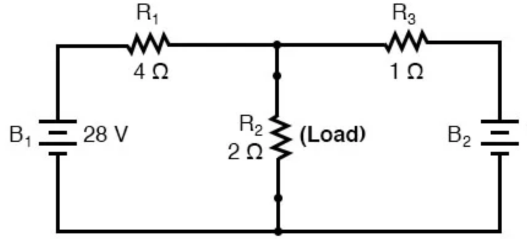
- For the Norton equivalent circuit.

  a. Apply Ohm's law: $I_{Norton} = V_{step\ 2} \,/\, Z_{step\ 4}$
  b. **Add** a current source with the value found in step a.
  c. **Add** the impedance found in step 4 in parallel with the current source.
  d. **Add** back the load element.

The words in bold will be standalone functions.

Point: Higher-level manipulation is easily implemented when following this approach

Thevenin



Norton

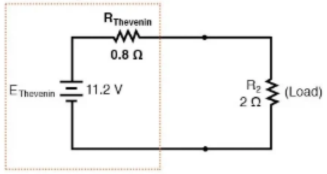
```
circuit = Circuit('circuits/th_no_equivalents.txt');
ELAB.thevenin(circuit, circuit.Resistors(2));
```

```
Symbolic analysis successful (0.237976 sec).

Numerical evaluation successful (0.0397994 sec).
```

```
circuit.list
```

```
ans =
    'Vth 1 0 DC 56/5
     Zth 1 2 4/5
     R2 2 0 2
     '
```

Thevenin Equivalent Circuit



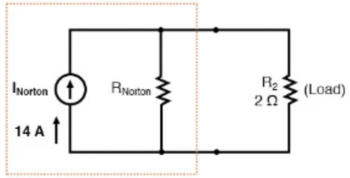

```
circuit = Circuit('circuits/th_no_equivalents.txt');
ELAB.norton(circuit, circuit.Resistors(2));
```

```
Symbolic analysis successful (0.245657 sec).

Numerical evaluation successful (0.0372693 sec).
```

```
circuit.list
```

```
ans =
    'Ino 1 0 DC 14
     Zno 1 0 4/5
     R2 1 0 2
     '
```

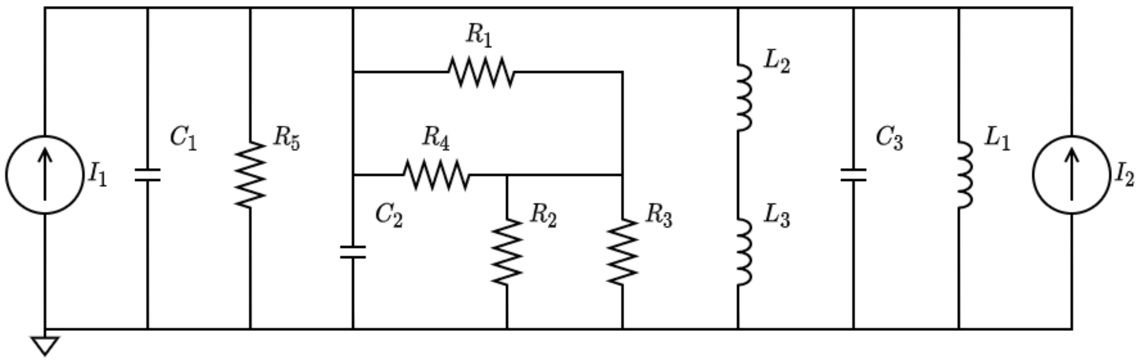
Norton Equivalent Circuit



IN ACTION – SIMPLIFYING

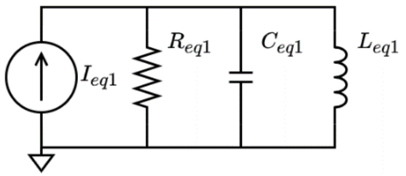
complex.txt

```
I1 1 0 AC 5
I2 1 0 AC 10
R1 1 2 1000
R2 2 0 2000
R3 2 0 2000
R4 1 2 3000
R5 1 0 1000
C1 1 0 0.000002
C2 0 1 0.000003
C3 1 0 0.00002
L1 3 1 0.02
L2 3 0 0.03
L3 1 0 0.2
```




```
ELAB.simplify(circuit);
circuit.list
```

```
ans =
    'I_eq1 1 0 AC 15
     R_eq1 1 0 7000/11
     L_eq1 1 0 0.04
     C_eq1 1 0 0.000025
     '
```



Notice

- Complexity is arbitrary – O(N) time, N = #pairs
- Layout is arbitrary – Only connections matter
- Works numerically, symbolically and mixed
- Always appropriate naming
- Can treat all impedances as same type
- Output is simply another circuit object

```
circuit = Circuit('circuits/multiple_eqs.txt');
circuit.list
```

```
ans =
    'Vs 1 0 DC Vs
     R1 1 2 R1
     R2 2 3 R2
     R3 4 5 R3
     R4 4 5 R4
     R5 5 0 R5
     C1 3 4 C1
     '
```

```
ELAB.simplify(circuit);
circuit.list
```

```
ans =
    'Vs 1 0 DC Vs
     Req1 2 1 R1+R2
     Req2 3 0 R5+(R3*R4)/(R3+R4)
     C1 2 3 C1
     '
```

```
circuit = Circuit('circuits/eq_impedance.txt');
circuit.list
```

```
ans =
    'Vs 1 0 AC Vs
     R1 2 3 3
     R2 4 0 8
     L1 2 4 0.2
     C1 1 2 0.002
     C2 3 0 0.01
     '
```

```
ELAB.evaluate(circuit)
```

```
Symbolic analysis successful (0.502269 sec).

Numerical evaluation successful (0.211514 sec).
```

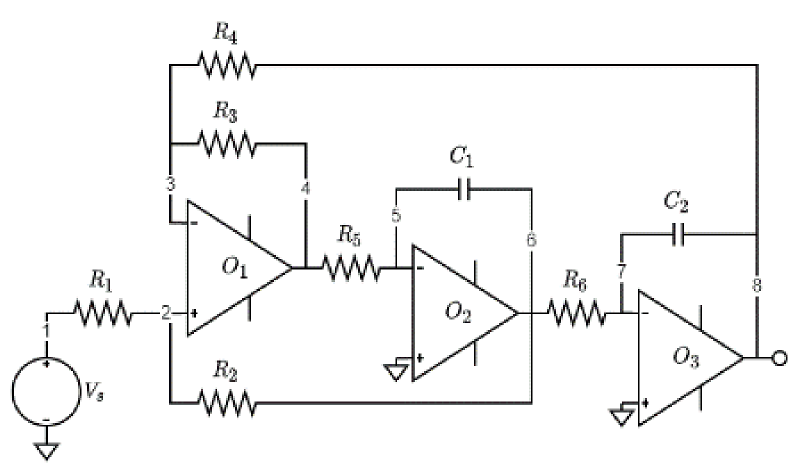
```
ELAB.simplify(circuit, true);
circuit.list
```

```
ans =
    'Vs 1 0 AC Vs
     Zeq1 1 0 0.002*s+(2405.0*s+8.0*s^2+1500.0)/(1100.0*s+s^2+500.0)
     '
```

IN ACTION – COMPLEX FILTER ANALYSIS

state_variable_filter.txt

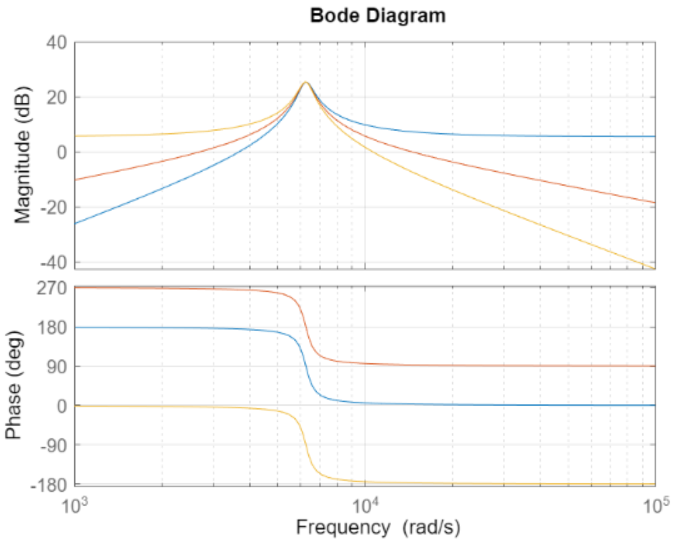
```
Vs 1 0 AC
R1 1 2 1000
R2 2 6 19000
R3 3 4 10000
R4 3 8 10000
R5 4 5 16000
R6 6 7 16000
C1 5 6 10e-9
C2 7 8 10e-9
O1 2 3 4
O2 0 5 6
O3 0 7 8
```




```
ELAB.ec2sd(circuit,1,8)

Symbolic transfer function calculated successfully (5.072700e-03 sec).
ans =
```

$$\frac{v_8}{v_1} = \frac{R_2\ (R_3 + R_4)}{R_1\, R_3 + R_2\, R_3 + C_2\, R_1\, R_3\, R_6\, s + C_2\, R_1\, R_4\, R_6\, s + C_1\, C_2\, R_1\, R_4\, R_5\, R_6\, s^2 + C_1\, C_2\, R_2\, R_4\, R_5\, R_6\, s^2}$$

```
bode(ELAB.ec2tf(circuit,1,4)); grid on; hold on;

Symbolic analysis successful (1.07754 sec).

Numerical evaluation successful (0.241102 sec).
Transfer function object created successfully (2.675553e-01 sec).

bode(ELAB.ec2tf(circuit,1,6));

Transfer function object created successfully (2.028830e-02 sec).

bode(ELAB.ec2tf(circuit,1,8)); hold off;

Transfer function object created successfully (1.757560e-02 sec).
```



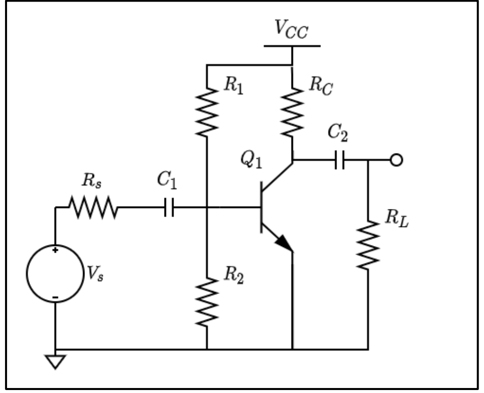
MATLAB does automatic simplification, if the symbols match

IN ACTION – TRANSISTOR MODELLING

Pre-modelling

bjt_complex_amp.txt

```
V_cc 4 0 DC
V_s 1 0 AC
R_s 1 2
R_1 3 4
R_2 3 0
R_C 4 5
R_L 6 0
C_1 2 3
C_2 5 6
Q_1 3 5 0
```

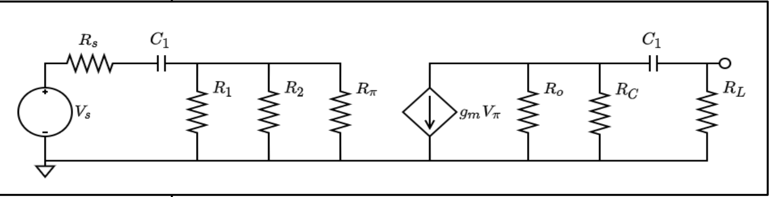


Notice

- Will handle an arbitrary number of transistors
- Biasing calculations are possible

Post-modelling

```
ELAB.hybrid_pi(circuit, 'lf', true);
circuit.list

ans =
    'V_s 1 0 AC V_s
     R_s 1 2 R_s
     R_1 3 0 R_1
     R_2 3 0 R_2
     R_C 0 4 R_C
     R_L 5 0 R_L
     R_pi_Q_1 3 0 R_pi_Q_1
     R_o_Q_1 4 0 R_o_Q_1
     C_1 2 3 C_1
     C_2 4 5 C_2
     G_Q_1 4 0 3 0 G_Q_1
     '
```



Post-simplification

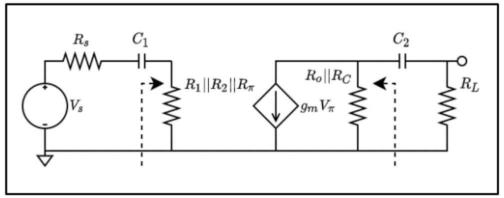
```
ELAB.simplify(circuit);
circuit.list

ans =
    'V_s 1 0 AC V_s
     R_s 1 2 R_s
     R_L 5 0 R_L
     R_eq1 0 4 (R_C*R_o_Q_1)/(R_C+R_o_Q_1)
     R_eq2 3 0 (R_1*R_2*R_pi_Q_1)/(R_1*R_2+R_1*R_pi_Q_1+R_2*R_pi_Q_1)
     C_1 2 3 C_1
     C_2 4 5 C_2
     G_Q_1 4 0 3 0 G_Q_1
     '
```

```
circuit.Resistors(3).resistance

ans =
```

$$\frac{R_C\, R_{o,Q,1}}{R_C + R_{o,Q,1}}$$

```
circuit.Resistors(4).resistance

ans =
```

$$\frac{R_1\, R_2\, R_{\pi,Q,1}}{R_1\, R_2 + R_1\, R_{\pi,Q,1} + R_2\, R_{\pi,Q,1}}$$



Input- and output resistance

FUTURE WORK - CONCLUSION

In-progress
- Delta-wye simplification
- Transformers, diodes
- Direct text input

Possibilities
- Computer vision (object detection), Image to netlist
- Graph theory for simplification?
- CircuiTikZ, schematic output

Scrapped
- GUI

New development!

< *PeerJ Computer Science*

Lcapy: symbolic linear circuit analysis with Python

Research article | Computer Aided Design | Computer Education | Scientific Computing and Simulation

Michael Hayes

February 18, 2022

Author and article information

Electrical and Computer Engineering, University of Canterbury, Christchurch, New Zealand

DOI
10.7717/peerj-cs.875

AARHUS
UNIVERSITY