Programmatic Symbolic Circuit Analysis

Implemented as a MATLAB Toolbox

Nicklas Vraa, Electrical Engineering B.Sc. Aarhus University - nkvraa@gmail.com - Bachelor’s Thesis

Abstract

This report introduces the basics of my MATLAB toolbox, which is capable of symbolic circuit analysis. I later named the project ELABorate. It explains the approach, I decided upon, when designing the software, as well as the actual implementation in code. It also outlines the motivation for the project, and lastly demonstrates some of the capabilities of the software.

To try the project, please visit:

<https://github.com/NicklasVraa/ELABorate>

1. Introduction

This project attempts to partially automate the process of analyzing electrical circuits by abstracting low-level tasks of the analysis to high-level functions to be called in program. For the implementation, I’ve chosen MATLAB for it’s live-script functionality, which allows automatic latex-style formatted output. The implementation should be easily transferable to Octave or Python with SymPy. MATLAB can also export to standalone c-code GUI applications.

1. Motivation

Today, circuit analysis is done either analytically by hand, or numerically using software like SPICE (LTSpice, PSPICE etc.). When doing symbolic analysis by hand, one gains great insight into not only the circuit, one works on, but also into any numerical variations of that circuit. The downside is that it typically gets complicated and takes considerable time.

Is it feasible to combine the advantages of symbolic analysis with the computational power of a computer in an intuitive manner? Currently, there is no software, which accomplishes this. SSPICE (Michigan State University, 2022) is an attempt at this but seems abandoned.

1. Project specification

The goal is to develop a toolbox for MATLAB, which is capable of symbolic circuit analysis. The objective is outlined as follows:

* Develop a program, which constructs a circuit model in code from a simple input. It should handle various common circuit elements, such as:
  + Independent AC- and DC sources.
  + Passives: resistors, capacitors, and inductors.
  + Dep. sources: CCCS, CCVS, VCCS, VCCS.
  + Larger structures: Op-Amps, transformers.
  + Non-linear elements: MOSFETs, BJTs.
  + Arbitrary sub-circuit models.
* Develop and implement methods for symbolically determining:
  + Circuit transfer functions.
  + Input- and output resistances.
  + AC/DC equivalents of a given circuit.
  + Thevenin/Norton equivalents.
  + Stability parameters.
* Develop and implement additional methods for:
  + Connecting the outputs of this toolbox to the rest of MATLAB’s system analysis functions.
  + Automate input validation and debugging.
  + After-the-fact manipulation, such as removing or inserting elements, and simplifying circuits.

1. Methodology

I take an object-oriented approach, defining each circuit element as its own class to ensure modularity and extendibility. The circuit is itself a class, which contain lists of elements-classes.

The element-class only contain information about its own nodal connections. Additional type-specific attributes are implemented by defining sub-classes: As an example, the resistor-class inherits from the passive-class, which inherits from the element-class. Inheriting ensures easy implementation of new circuit elements in the future.

I use MATLAB’s symbolic toolbox for the symbolic manipulation, and the modified nodal analysis (NMA) approach for relating each circuit element to each other.

* 1. Modified Nodal Analysis

When symbolically analyzing electrical circuits, the electrical engineer usually employs the node voltage method and loop current method. Another similar, but more recent approach is modified nodal analysis (Chung-Wen, Ruehli, & Brennan, 1975), which uses linear algebra to speed up the analysis.

Modified nodal analysis (MNA) uses the element’s branch constitutive equations (BCEs) i.e., their voltage- and current characteristics and Kirchhoff’s current- and voltage laws. The approach is usually broken down into 3 steps.

1. Write the KCL equations of the circuit. At each node of the circuit, write the currents coming into and out of the node. The currents of the independent voltage sources are taken from the positive to negative. Note that the right-hand-side of each equation is always equal to zero, so that the branch currents that come into the node are given a negative sign and those that go out are given a positive sign.
2. Use the BCEs in terms of the node voltages of the circuit to eliminate as many branch currents as possible. Writing the BCEs in terms of the node voltages saves one step. If the BCEs were written in terms of the branch voltages, one more step, i.e., replacing the branches voltages for the node ones, would be necessary.
3. Write down any unused equations.

Exactly how this approach will be handled by a computer, is described in the next section.

* 1. Algorithmic MNA

Converting MNA to an algorithm, that can be performed by a computer is relatively straight-forward, assuming I already have an abstract circuit-object, which contain all the information needed for complete analysis.

Much of the approach has already been described in a paper by Litovski but it must be modified and extended for additional circuit elements.

The following is a high-level description of the algorithm but modified and extended to fit my project specification.

View setting a variable not as a numeric evaluation, but as defining a symbolic expression, like appending to an equation. Firstly, I define specific notation to shorten the description.

Let be the number of nodes

Let be the number of voltage sources

Let be the number of processed voltage sources

Let *passive* be either a resistor, capacitor, or inductor

Let denote a mathematical expression

Allocate matrices and fill with 0

Let denote element within matrices

Allocate vectors and fill with 0

Let denote anode and cathode connections

We then fill the matrices. This is the crux of this extended, programmatic, modified nodal analysis approach, which allows for fast and efficient symbolic computation. We start with the most basic circuit elements. Mind the notation, where ‘i’ and ‘j’ refers to both a vectors and matrix-indices.

For all passives

If passive is resistor, set

Else if passive is capacitor, set

Else if passive is inductor, set

If anode is ground

Set

Else if cathode is grounded

Set

Else

Set , , ,

For all independent voltage sources

If anode is not grounded

Set ,

If cathode is not grounded

Set ,

Add parsed voltage source id’s to as and as

For all independent current sources

If anode is not grounded

Set

If cathode is not grounded

Set

For operational amplifiers, the approach is surprisingly simple and resembles the algorithm for the previous parts.

Let denote 1st and 2nd input connections.

For all op-amps

If first input is not grounded

Set

If second input is not grounded

Set

Add parsed op-amps id’s to as

Now for the active sources. The approach is very similar, but with some additional complexity, especially for VCCS’s. When talking about control nodes, I am referring to the nodes, on which the source’s output depend.

Let denote anode and cathode

Let denote controlled anode and cathode.

For all VCVS’s

If anode is not grounded

Set ,

If cathode is not 0

Set ,

If 1st control node is not grounded

Set

If 2nd control node is not grounded

Set

Add parsed VCVS id’s to as

For all VCCS’s

If nothing is grounded

Set ,

Set ,

If only anode is grounded

Set ,

If only anode and control anode are grounded

Set

If only anode and control cathode are grounded

Set

If only cathode is grounded

Set ,

If only cathode and control anode are grounded

Set

If only both cathodes are grounded

Set

If only control anode is grounded

Set ,

If only control cathode is grounded

Set , ,

For all CCVS’s

If anode is not grounded

Set ,

If cathode is not grounded

Set ,

Add parsed CCVS id’s to as

Insert ’s in D

For all CCCS’s

Store index of current CCCS

If anode is not grounded

If cathode is not grounded

1. Implementation

This chapter details the implementation of the program in MATLAB code. Only the larger structures of the program will be described. See the code on the GitHub repository for more detail. The code is extensively commented.

* 1. Input to the program

The industry standard for defining circuits is using the netlist format. Simply a text file, where each line is a component. Each component is defined by a symbol identifying the type, a name, its nodal connections, and some additional information specific to the type of component. For the MNA algorithm to work properly, the reference node must be ground. Each node must also obey the sign convention.

The syntax used by this program is SPICE-like with a few simplifications. I have chosen the custom file extension to be ‘.circ’ for easier distinction between circuit-files and regular text files, but it is still a raw text file.

* 1. A screenshot of a computer screen

     Description automatically generated with low confidenceModelling the Circuit

Fig. 1 outlines the class-tree. Only the fenced classes are exposed to the user to maintain user-friendliness. All the element-classes are only interfaced with through the circuit- and ELAB-class.

* 1. Programmatic Analysis

1. Evaluation
   1. Using the Program

The program is designed to be used in conjunction with MATLAB’s LiveScript, as it neatly outputs the results of the program.

* 1. Future expansions

1. Conclusion

# References

1. Chung-Wen, H., Ruehli, A., & Brennan, P. (1975, June). The modified nodal approach to network analysis. *IEEE Transactions on Circuits and Systems, 22*(6), 504-509. doi:10.1109/TCS.1975.1084079
2. Michigan State University. (2022, March 10). *Symbolic Spice*. From egr.msu.edu: https://www.egr.msu.edu/~wierzba/index\_Page533.htm