Imperial College London

Assignment Title: Circuit Simulator

Module Reference and Name: ELEC40006 Electronics Design Project

Group Name: Rectifiers

Students:

Chau, Yan To,

Kitikul, Chakrarat,

Viswakumar, Raghav

Date of Submission: 14th June 2020

Word Count: 10,986

Table of Contents

Document Conventions

For clarity, quantities of various circuit components and sources are referenced with the following symbols, with their respective unit symbols omitted, unless otherwise specified.

Component	Quantity	Symbol	Unit
-	Voltage	V	Volts (V)
-	Current	I	Amps (A)
Resistor	Resistance	R	Ohms (Ω)
Capacitor	Capacitance	C	Farads (F)
Inductor	Inductance	L	Henrys (H)
Resistor	Conductance	G	Siemens (S)
Inductor	Magnetic Flux	Φ	Weber (Wb)

Following standard conventions, upper case is used to represent biased or DC values, while lower case is used for time-varying values.

Times are given in seconds unless otherwise stated.

Notes

The word count provided on the cover page accounts for content in sections 2 to 6, excluding headers, code excerpts, as well as captions for figures, tables or equations.

Despite every effort to achieve the greatest degree of clarity, this report, like any other report, may still contain some errors, such as typographical and grammar mistakes. Any feedback for errors, as well as guidance would very much be appreciated.

1. Abstract

The purpose of this project is to apply engineering practices to solve a problem in a team-based environment. The assignment is to use C++ to program a simulator that can perform a subset of LTSpice functionality, more specifically transient simulations for linear components and a subset of nonlinear components; this involves an in-depth study of modifying nodal analysis for computerised analysis and the Newton Raphson method, implementing the actual program, and continuously performing tests and optimisations to improve the simulator. This process, albeit the numerous hardships and hurdles, would eventually be proven exceptionally satisfying, not only because of the deliverables achieved, but also for the team-building and project management experiences throughout the process.

This report is a document for both the technical and non-technical details of the project. The former refers to the research undergone and the actual details of the implementation and testing of the simulator; the latter refers to the project management procedures adopted throughout the project, and details such as the software development life cycle and collaboration framework are documented.

2. Introduction

2.1. Background and Objectives

SPICE, or “Simulation Program with Integrated Circuit Emphasis”, is a well known open-source analog electronic circuit simulator written by Laurence Nagel at the Electronics Research Laboratory of UC Berkeley. The motivation was to create a program that could be made available in the public domain, unlike many other circuit simulators at the time which were developed for the United States Department of Defense. Since its release in 1973, SPICE has been utilised to perform computerised analysis of nonlinear circuits with the aid of various techniques, such as Newton’s method, which would be further explored in this project. In contrast to the complex and impractical process of breadboard-testing board-level circuit designs, SPICE is offered as a reliable and cost-efficient way to verify the integrity of various integrated circuits. This is why SPICE has been quickly and widely adopted as the industry’s standard integrated circuit simulator [25].

Although an extremely versatile program in its own right, SPICE, due to its command line driven interface, is considered difficult to use. Many commercial versions with schematic GUI input and oscilloscope-like output interfaces were consequently created, each with their own variety of semiconductor components. One of these versions is LTspice, which is extremely versatile in terms of its interface, the variety of components supported, as well as the range of simulation modes available. The program automatically captures the netlist from the user-drawn schematic and allows users to plot traces for AC, DC, transfer function, transient and a range of other analyses. At its heart, however, lies the original SPICE solver.

At the Electrical and Electronic Engineering (EEE) department of Imperial College London, LTspice is used as a circuit simulation tool for standard laboratory exercises. For Year 1 students, it has been selected as the tool for examining the terminal characteristics of bipolar transistors and different configurations of the CE Amplifier and other amplifier stages. With respect to the wide adoption of SPICE, the core of the SPICE solver, despite its complexity, is an object worthy of scrutiny.

There are various objectives for this project.

1) To apply team-working and project management skills throughout;
2) Utilise the mathematics and programming skills gained in the academic year, including but not limited to C++ programming and matrices manipulation; and,
3) To ultimately create a software package that performs transient simulations of circuits, like LTSpice.

The third point refers to the actual deliverable product of this project, which is the Circuit Simulation software. The implied audience of this software may include the following:

- Lecturers and GTAs of the Imperial College EEE department,
- Other Year 1 EEE students; and,
- Any other parties of interest with a basic knowledge of computerised analysis for circuits.

The software quality attributes of this project have been divided into the following functional and non-functional requirements.

2.1.1. Functional Requirements

Functional requirements are the tasks to be fulfilled by the actual deliverable. For this circuit simulator, it refers to the accuracy and correctness when solving various circuit configurations using the circuit simulator. With respect to the specifications outlined in appendix 8.4, it must support a subset of functions currently available on SPICE, more specifically transient simulations for circuits incorporating the following elements:

- Linear passive components (resistors)
- Linear reactive components (capacitors and inductors)
- Current sources of a DC value or sinusoidal function
- Voltage sources of a DC value or sinusoidal function

Due to the nature of this project, support for some elements are left as an extension, including:

- Nonlinear components (diodes and transistors)
- Any other components compatible with SPICE

In addition, details for the input and output interface requirements of the program is outlined in section 4.

2.1.2. Non-functional Requirements

Non-functional requirements refer to the efficiency and usability aspects of the deliverable. The code deliverable contains numerous recursions and loops due to:

- The adoption of the iterative Newton-Raphson Method; and,
- Regenerating outputs during each timestep of a transient simulation.

Hence, the non-functional requirements include, but not limited to the following:

- Usability, which is determined by the quality of the documentation and the user-friendliness of the program,
- Efficiency, which is determined by the compile and run-time of programs under fixed processor speed and memory
- Reliability, or how stable the program is when errors occur,
- Testability, which is its effectiveness in testing a wide range of inputs for debugging
- Adaptability, which is the program's ability to adapt to irrelevant inputs such as comments or tolerate faulty inputs,
- Readability, such that the program can be easily modified, maintained and managed
- Resilience, that is the program's ability to resolve large inputs (large circuit configurations)
- Portability and reusability, which is the interface and platform compatibility of the program; and,
- Extensibility, which means that the program can be increased into a larger scale, and that new features can be integrated to enhance functionality.

These requirements are non-behavioral constraints that are each achieved to a certain extent; likewise, an evaluation on these aspects are discussed in section 5.

2.2. Software Requirement Specifications (SRS)

SRS is a crucial part of software development, and it serves to document the development of this circuit simulator software in a systematic manner. An SRS may contain many elements, as shown in the SRS outline created by Helm [8]; the relevant elements for this project are:

- An abstract of the project and the simulator (1),
- Purpose (2.1),
- Intended audiences and their characteristics (2.1),
- Scope, which includes the functions and features of the product (2.1.1 and 2.1.2),
- User and system interfaces, as well as required hardware and software interfaces,
- Design and Implementation Constraints (4.2),
- Evaluation on the functional and non-functional requirements achieved (5),
- References (7); and,
- Revision History, presented in the form of a log (8.2).

These elements are placed in various other sections, denoted by the number in brackets; this is done to maintain the overall readability of an academic report. For the purposes of this project, licensing requirements, legal details and applicable standards are omitted.

2.3. Literature Review

2.3.1. Advanced circuit analysis

Circuit analysis involves deploying various mathematical techniques to find unknown elements of a circuit from a circuit with known elements and a known configuration. For standard nodal analysis, the aim is to find the unknown node voltages. This involves:

1) Examining a circuit with components of known resistances, capacitances and/ or inductances, nonlinear components of known terminal characteristics, as well as sources of known voltage or current value;
2) Deriving the relevant equations using Kirchoff's voltage and current laws; and,
3) Solving the system of equations.

The equations outlined in step 2) are established by the fundamental principles of Kirchhoff's Current Law (KCL) and Kirchhoff's Voltage Law (KVL) in Electrical Engineering. KCL states that the net sum of currents in a node is zero, while KVL states that the net sum of voltages around a closed loop is zero.

In a transient simulation environment, this process is extended to evaluate the system of equations at consecutive time steps of the simulation.

The key, however, for simulating circuits, is to devise an algorithm that can be derivable in a straightforward manner, such that complex circuits with a greater number of nodes and components can be consistently and systematically solved through computerised analysis as well. This section outlines the algorithms devised for various circuit configurations processed by this project, and these are the premises to understanding the operations of the circuit simulator.

2.3.1.1. Scenario 1 - Resistors and current sources only

This section outlines the process of performing circuit analysis on circuits with resistors and current sources, as described by Mitcheson [15]. Take the following circuit in figure 1 as an example.

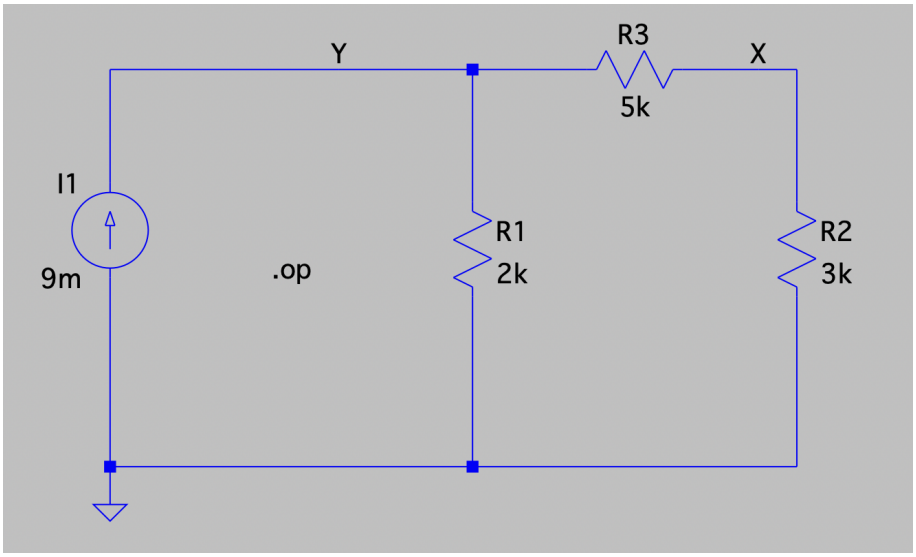


Figure 1 Circuit with only current sources and resistors

For consistency, currents exiting a node are positive and currents entering a node are negative. Hence, the KCL equations are:

$$\frac{Y}{R_1} + \frac{Y - X}{R_3} - I_1 = 0$$

$$\frac{X - Y}{R_3} + \frac{X}{R_2} = 0$$

Equation 1 KCL about node Y

Equation 2 KCL about node X

By simultaneous equations, the solutions are:

$$X = 5.4, Y = 14.4$$

Nonetheless, considering the bigger picture, rearranging equation 1 and equation 2, and converting resistances to their inverse, conductances, results in the following:

$$(G_1 + G_3)Y - G_3X = I_1$$

$$-G_3Y + (G_2 + G_3)X = 0$$

Equation 3 Rearranged KCL equation about node Y

Equation 4 Rearranged KCL equation about node X

It is now apparent that equation 3 and equation 4 can be expressed as a system of equations:

$$\begin{bmatrix} (G_1+G_3) & -G_3 \\ -G_3 & (G_2+G_3) \end{bmatrix} \begin{bmatrix} Y \\ X \end{bmatrix} = \begin{bmatrix} I_1 \\ 0 \end{bmatrix}$$

Equation 5 System of equations in matrix form

Consequently, the vector of two unknown voltages can be calculated by obtaining the inverse of the leftmost matrix, known as the conductance matrix. When this algorithm is extended to a circuit with larger number of unknown node voltages, the following system of equations is obtained:

$$\begin{bmatrix} G_{1,1} & -G_{1,2} & \cdots & -G_{1,n} \\ -G_{2,1} & G_{2,2} & \cdots & -G_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ -G_{n,1} & -G_{n,2} & \cdots & G_{n,n} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} i_1 \\ i_2 \\ \vdots \\ i_n \end{bmatrix}$$

Equation 6 System of equations for a circuit with n nodes

As before, the solution is found by calculating the inverse of the conductance matrix. The following observations can be made for the conductance matrix in equation 6:

1) The size of the matrix is n by n, where n is the total number of nodes in the circuit, excluding the reference (ground) node.
2) The diagonal entries contain the sum of conductances connected to that corresponding node. For instance, $G_{1,1}$ and $G_{2,2}$ refers to the sum of conductances connected to nodes 1 and 2 respectively.
3) The remaining entries contain the sum of conductances connected in between the two corresponding nodes. For instance, $G_{1,2}$ and $G_{2,1}$ both refer to the sum of conductances between nodes 1 and 2, and they are both the same.
4) The conductance matrix is symmetrical, since connections of components between 2 nodes are mutual.
5) All entries in the diagonal are positive while the remaining entries are negative.

The voltage matrix is size n by 1, and each entry represents the node voltage. The current matrix is also size n by 1, where each entry contains the net sum of currents at a node, or the difference in currents entering and exiting a node.

2.3.1.2. Scenario 2 - Resistors, voltages and current sources only

The conductance matrix is slightly altered with the inclusion of voltage sources in the circuit configuration. Take the following circuit in figure 2 as an example:

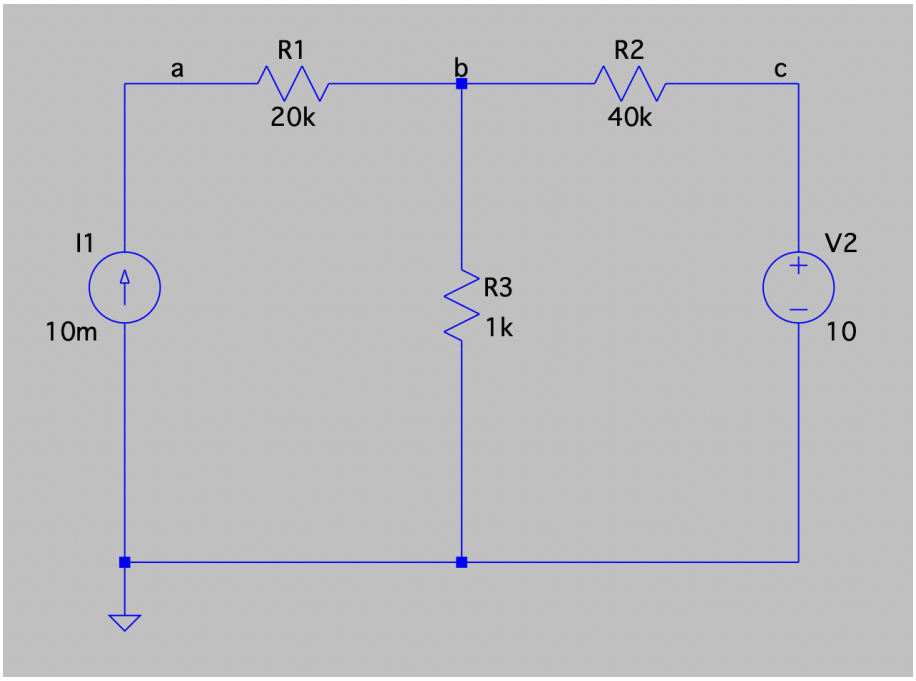


Figure 2 Circuit with resistors, voltage and current sources only

The defining KCL equations are:

$$\frac{v_a - v_b}{R_1} - I_1 = 0$$

$$\frac{v_b - v_a}{R_1} + \frac{v_b - v_c}{R_2} + \frac{v_b}{R_3} = 0$$

$$v_c = V_2$$

Equation 7 KCL equation at node a

Equation 8 KCL equation at node b

Equation 9 Node c equation

By manual calculations, the solution for this system of equations is:

$$v_a = 210, v_b = 10, v_c = 10$$

Likewise, this process can be streamlined for computerised analysis. Analogous to the principle outlined in scenario 1, rearranging the three equations into a matrix format results in the following:

$$\begin{bmatrix} G_1 & -G_1 & 0 \\ -G_3 & (G_1+G_2+G_3) & -G_2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_a \\ v_b \\ v_c \end{bmatrix} = \begin{bmatrix} I_1 \\ 0 \\ V_2 \end{bmatrix}$$

Equation 10 System of equations for figure 2 circuit in matrix form

Several observations can be made:

1) Most entries of the conductance matrix resemble the original matrix. However, since the KCL equation defining node c is just $v_c = V_2$, connections to node c are ignored for the equation corresponding to node c.
2) The current matrix no longer contains only the net sum of currents at a node. Instead, the entry corresponding to node c contains V_2.

Nevertheless, this process is insufficient to accommodate circuit configurations with voltage sources connected between two non-reference nodes. Consider the following circuit in figure 3 as an example.

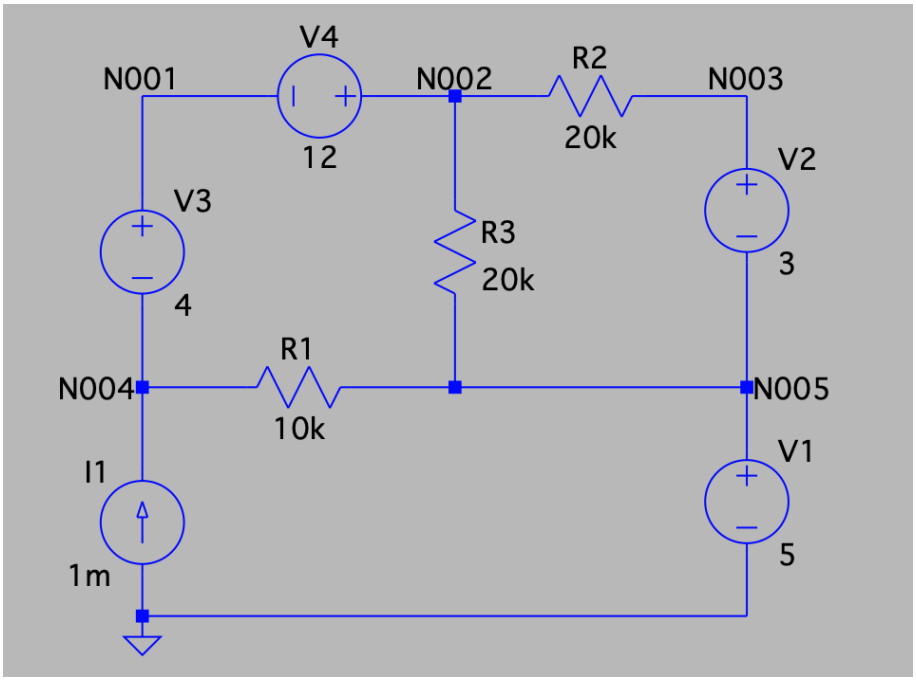


Figure 3 Circuit with multiple supernodes.

For simplicity, the voltage at each node is represented by v_n, where n is the least significant digit of the node. Using the conventional approach outlined in scenario 1, the relevant equations are:

$$v_1 - v_4 = V_4$$

$$v_2 - v_1 = V_5$$

$$v_3 - v_5 = V_3$$

$$\frac{v_4 - v_5}{R_1} + \frac{v_2 - v_5}{R_3} + \frac{v_2 - v_3}{R_2} = I_1$$

$$v_5 = V_1$$

Equation 11 Set of equations required for solving circuit in figure 3

In this set of equations, the 4th equation shows the KCL equation for the supernode N001, N002 and N004, which envelopes the two voltage sources V_4 and V_5. The fourth line is particularly complex, and there appears to be little consistency in relation to the way different lines of equations are constructed.

While the current algorithm is straightforward in terms of manual calculations, the key concerns when constructing this conductance matrix with computerised analysis are:

1) Identifying larger supernodes.
2) The inconsistencies when completing entries corresponding to different nodes, including:
    a) Nodes connected to only resistors and current sources which are completed with the algorithm outlined in scenario 1,
    b) Supernode row: entries are filled in like the conductance matrix if the node corresponding to that entry is only connected to current sources and resistors, and positive if the node is within the supernode. In this scenario, v_b and v_c are within the supernode; and,
    c) Rows that correspond to potential differences.

These problems may potentially lead to the use of multiple unreliable if-else statements and inefficient while loops during computerised analysis.

To mitigate this issue, a modified algorithm has been devised by Cheever [2], which accounts for the currents entering and exiting the voltage sources. The following equations are obtained:

$$I_{V_3} - I_{V_4} = 0$$

$$\frac{v_2 - v_3}{R_2} + \frac{v_2 - v_5}{R_3} + i_{V_4} = 0$$

$$\frac{v_3 - v_2}{R_2} + I_{V_2} = 0$$

$$\frac{v_4 - v_5}{R_1} - I_{V_3} = I_1$$

$$\frac{v_5 - v_2}{R_3} + \frac{v_5 - v_4}{R_1} + I_{V_1} - I_{V_2} = 0$$

$$v_1 - v_4 = V_3$$

$$v_2 - v_1 = V_4$$

$$v_3 - v_5 = V_2$$

$$v_5 = V_1$$

Equation 12 System of equations for circuit in figure 3 derived using the modified algorithm

, where i_{V_n} refers to the current flowing through the voltage source V_n.

The crucial difference is that the line 4 (supernode KCL equation) of equation 11 from the previous system of equations is split into multiple equations here. Rearranging the system of equations to a matrix format leads to an “augmented” conductance matrix:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \\ 0 & (G_2+G_3) & -G_2 & 0 & -G_3 & 0 & 0 & 0 & 1 \\ 0 & -G_2 & G_2 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & G_1 & -G_1 & 0 & 0 & -1 & 0 \\ 0 & -G_3 & 0 & -G_1 & (G_1+G_3) & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ I_{V_1} \\ I_{V_2} \\ I_{V_3} \\ I_{V_4} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ I_1 \\ 0 \\ V_1 \\ V_2 \\ V_3 \\ V_4 \end{bmatrix}$$

Equation 13 System of equations of circuit in figure 3 in a matrix format

Likewise, the solutions can be computed by obtaining the inverse of the conductance matrix. Several observations can be made for the augmented version of the conductance matrix:

1) The entire matrix is symmetrical, though this behavior is related to the ordering of the equations.
2) The dimension of the matrix is $m \times m$, where m is the sum of the number of non-reference nodes and the number of voltage sources.

The advantage of the previous method is that it results in a smaller matrix, specifically a conductance matrix of size $n \times n$, where n is the number of non-reference nodes. However, in comparison, the modified algorithm provides a lot more advantages:

1) The top left $n \times n$ entries of the conductance matrix are completed using the method in scenario 1, which was seen as a consistent approach to solving circuits with resistors and current sources. This part remains the same and only depends on the placement of resistors.
2) The entire matrix is symmetrical when the equations at each node and the equations corresponding to the potential difference of the voltage sources are inserted in the correct order.
3) While the size of the matrix is substantially larger, it should be argued that the currents through voltage sources, such as I_{V_1} or I_{V_2} are useful quantities. This is because a capacitor behaves as a voltage source at each timestep of a transient simulation, and hence the previous current through the capacitor is needed for calculating the relevant quantities of the capacitor.

All in all, this modified algorithm can be recursively extended to support a greater number of voltage sources connected together, without forming unnecessarily large and complex supernode equations. The process of dealing with large matrices is a topic to be further elaborated in section 4, which outlines various technical barriers during coding development.

2.3.1.3. Scenario 3 - Capacitors and Inductors

Capacitors and inductors are reactive components that depend on the history of the circuit, as proposed by Stott [22]. The key attributes of these 2 components are summarised in the table 1, with details elaborated throughout this section:

Attributes	Capacitor	Inductor
Average current	Infinite	Defined by circuit configuration
Average voltage	Defined by circuit configuration	Zero
Model in steady state (for a long period of time, for instance in DC circuits)	Open circuit	Closed circuit
Obeys	Voltage continuity (at a certain instance voltage is the same regardless of the value of current flowing)	Current continuity (at a certain instance current is the same regardless of the value of voltage across it)
Defining equation	$i = C\frac{dv}{dt}$	$v = L\frac{di}{dt}$

Table 1 Key attributes of capacitors and inductors

Hence, a capacitor can be treated as a voltage source, whereas an inductor can be treated as a current source. The first example briefly outlines how and why capacitors are converted into voltage sources during specific time instances of a transient simulation. Consider the following circuit in figure 5.

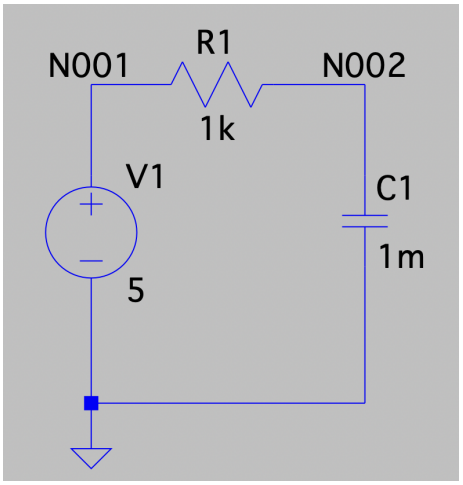


Figure 4 Circuit with a capacitor

From a relatively ground-level perspective, a capacitor is made of two conducting but separated pieces of plate. When a current flows through the capacitor, positive charge due to current is accumulated on one plate, and in regards to charge neutrality, a balancing and equal negative

charge exists on the other other plate. A potential difference is formed between the plates, which is proportional to the charge accumulated, and these two variables are related by the quantity capacitance, as shown in equation 14:

$$v = \frac{q}{C}$$

Equation 14 Relationship between voltage and charge accumulated of a capacitor

Consequently, using the definition of current as a time integral of charge, this leads to the derivation of equation 15, the capacitor equation:

$$i = \frac{dq}{dt} = C\frac{dv}{dt}$$

Equation 15 Capacitor equation

As illustrated by the equation, a change in voltage requires infinite current. Therefore, the voltage of a capacitor cannot change instantaneously - in other words, at a certain time instance in a transient simulation, it acts as a voltage source, with the same voltage regardless of the current flowing through. The calculations are as follows:

1) Assume the capacitor starts with charge of zero. Since voltage is proportional to the charge accumulated, replace it with a 0V voltage source.
2) Node voltage at N002 is therefore zero. With respect to the node voltages, the capacitor current, which is the current through the resistor, is:

   $$i = \frac{v_1 - v_2}{R_1} = \frac{5}{1k} = 5mA$$

3) Current is the time integral of charge. Using the numerical method for calculus, the change charge during the next 1μs time step is the product of the timestep and current:

   $$q = It = 5m \times 1\mu s = 5nC$$

4) In context, 5nC is the change in charge, and the total charge for the first time step. With respect to equation 17, the capacitor is now equivalent to the voltage source with value:

   $$v = \frac{q}{C} = \frac{5nC}{1\mu F} = 5mV$$

5) Using this new node voltage, the capacitor current for the next time step is:

   $$i = \frac{v_1 - v_2}{R_1} = \frac{5 - 0.005}{1k} = 4.995mA$$

Repeating the above procedure for the next few time steps, the following values are obtained for the next few iterations, as shown in table 2:

Time (μs)	Change in charge (C)	Total Charge (C)	Voltage across capacitor (V)	Current through capacitor (A)
0	0	0	0	0.005
1	5E-09	5E-09	0.005	0.004995
2	4.995E-09	9.995E-09	0.009995	0.00499
3	4.99E-09	1.4985E-08	0.014985005	0.004985
4	4.985E-09	1.997E-08	0.01997002	0.00498

Table 2 Values for voltage, current and charge of capacitor at the first 5 timesteps.

In summary, the algorithm for solving the node voltages and current through the capacitor at a specific time step is as follows:

1) Calculate the branch current through the capacitor, initially assuming the capacitor starts with a charge (and hence voltage source) of value zero.
2) Obtain the value of change in charge by multiplying by current with the time step.
3) Obtain the total charge accumulated, by adding the value obtained in step 2 with the previous charge.
4) Find the potential difference of the capacitor proportional to this charge, by dividing total charge by the capacitance. This effectively replaces the capacitor with a voltage source.
5) Using this voltage, solve for the new capacitor current, and iterate to the next time step.

On the other hand, inductors are treated as current sources. Figure 5 shows an example of a simple circuit consisting of an inductor.

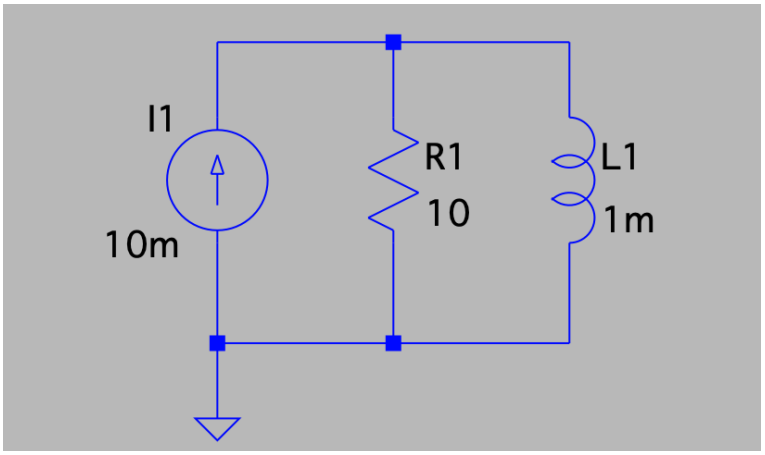


Figure 5 Circuit with an inductor

Inductors are fundamentally coils of wire around a ferromagnetic core. By Faraday's law of induction, the induced voltage across the inductor is proportional to the rate of change of magnetic

flux through the coils, in which the change of magnetic flux is generated current flowing through the inductor. This can be expressed as the following:

$$V \propto \frac{d\phi}{dt}$$

Equation 16 Faraday's Law of induction

Accounting for various factors, the governing inductor equation is:

$$V = L\frac{di}{dt}$$

Equation 17 Inductor Equation

As shown in the inductor equation, equation 18, a change in current requires infinite voltage. Therefore, the current of an inductor cannot change instantaneously - in other words, at a certain time instance, it acts as a current source, with the same current regardless of the voltage across it. As shown in [21], the calculations for the circuit in figure 5 are as follows:

1) Assume the inductor is initially a current source with value 0A, which is an equivalent open circuit. Initially the node has a voltage of:

   $v = i \times R = 10m \times 10 = 100m$

2) As the inductor develops a magnetic field, the new current in the inductor, according to equation 18, is:

   $$i = \frac{1}{L}\int vdt + i_0 = \frac{1}{1m}100m \times 1\mu + 0 = 0.0001$$

3) Note that this value is added to the initial current in the inductor, which is only zero for the first time step. Then, in the next 1μs time step, the current in the adjacent resistor is

   $i = 1m - 0.0001 = 9.9m$

4) Likewise, the new nodal voltage is:

   $v = 9.9m \times 10 = 99m$

Table 3 shows the values obtained by repeating the procedure for the next consecutive time steps:

Time (μs)	Current through resistor (A)	Node Voltage (V)	Current through inductor (A)
0	0.01	0.1	0.0001
1	0.0099	0.099	0.000199
2	0.009701	0.09701	0.00029601
3	0.00940499	0.0940499	0.00039006
4	0.00901493	0.090149301	0.00048021

Table 3 Values for currents and node voltage at the first 5 timesteps.

In summary, the algorithm for solving the node voltages and current through the capacitor at a specific time step is as follows:

1) Find the voltage v across the inductor
2) Obtain the current through the inductor by:

$$i = \frac{1}{L}\int v dt + i_0 = \frac{1}{L}vt + i_0$$

Equation 19 Inductor Equation rearranged, where t is the timestep.

3) Find the new node voltage across the inductor. This may involve calculating currents in adjacent components.
4) Iterate to the next time step.

This algorithm can even be extended to larger circuit configurations with inductors. Consider the following circuit in figure 6:

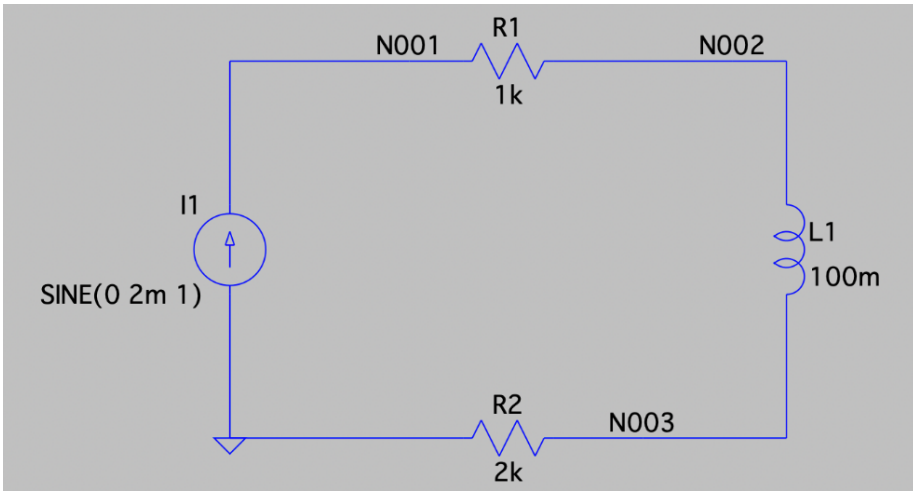


Figure 6 Inductor circuit with resistors on both ends.

With respect to equation 19, incorporating the procedure for the inductor equation in the system of equation gives:

$$\begin{bmatrix} G_1 & -G_1 & 0 & 0 \\ -G_1 & G_1 & 9 & -1 \\ 0 & 0 & G_2 & -1 \\ 0 & -\frac{t}{L} & \frac{t}{L} & 1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ V_3 \\ i_L^{new} \end{bmatrix} = \begin{bmatrix} I_1 \\ 0 \\ 0 \\ i_L^{old} \end{bmatrix}$$

Equation 20 System of equations for circuit in figure 6

Using the principles outlined in scenarios 1 and 2, capacitors and inductors can be changed into voltage and current sources respectively, and hence circuit configurations with resistors, capacitors, inductors, as well as voltage and current sources can be solved.

When capacitors or inductors are placed in series or parallel configurations, they can simply be treated as multiple sources. This however raises the concern of placing two capacitors in parallel, since ideal voltage sources cannot be placed in a parallel configuration, in which currents infinitely circulate between the two sources. Otherwise, these reactive components can be converted into one component with a certain equivalent value, which can then be treated as a voltage or current source following the principles detailed above.

The justifications for selecting a numerical integration method for reactive components rather than a differentiator model is discussed and evaluated alongside test results in section 5.

2.3.2. Solving circuits with nonlinear components

A subset of nonlinear components are supported in this circuit simulator, including diodes and transistors. Unlike linear components, such as the aforementioned resistors, capacitors or inductors, nonlinear components do not have analytical solutions, and are instead approximated with numerical methods like the Newton-Raphson Method [17]. This is because the behavior of linear components can be intuitively described with a single number, like resistance, inductance or capacitance, whereas that of nonlinear components are described by transcendental functions which can only be solved numerically; for instance, diodes are described by the well known Shockley equation, which involves the natural exponential function:

$$i_d = I_s e^{\frac{v_d}{V_T}}$$

Equation 21 Shockley's equation for diodes

In equation 21, the subscript d represents voltage or current through the diode, I_s is the temperature dependent saturation current, and V_T is the thermal voltage or the "volt equivalent of temperature". The typical values of I_s and V_T are as follows:

$$I_s = 1 \times 10^{-15} A$$

$$V_T = 25mV$$

For the purposes of this simulator, diodes in breakdown mode will not be considered. They are only either "ON" within the forward biased region, or "OFF" within the reverse biased region., The typical value for the boundary in between the "ON" and "OFF" modes is 0.7, though it should be noted that this is just an approximated value which is not used for the final nodal values.

To ease the understanding of solving circuits with diodes, this section is split into several segments. The three segments are:

1) Introduction to the Newton-Raphson Method
2) Parameters for performing computerised analysis using the Newton-Raphson Method
3) Example of solving circuits with diodes

2.3.2.1. Introduction to the Newton-Raphson Method

The iterative Newton-Raphson method is the cornerstone for solving circuits with nonlinear components. The aim is to determine the operating point of the diode, $(v_d^{(n)}, i_d^{(n)})$, after iteratively applying the Newton-Raphson Method. Given that the initial conditions are correct, the method offers a systematic way to iteratively converge to the root of a nonlinear equation, and is suitable for computerised analysis owing to its consistent implementation [18]. Consider a diode in figure 7, labelled with passive sign convention:

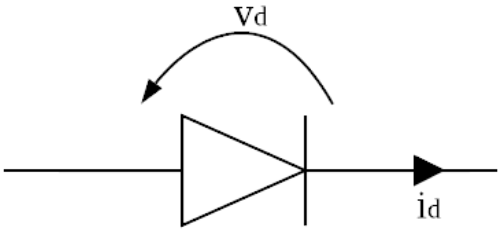


Figure 7 Non-ideal diode, labelled with passive sign convention

With respect to the Shockley equation, equation 21, i_d is a function of v_d. In context, the Newton-Raphson method states that at the given initial point $v_d^{(0)}$, the zero point of the continuous function i_d is closer to the zero point of the function than v_d. In mathematical terms, the equation is:

$$v_d^{(1)} = v_d^{(0)} - \frac{i_d^{(0)}}{\frac{\partial i_d}{\partial v_d}\Big|_{v_d = v_d^{(0)}}}$$

Equation 22 First iteration of the Newton-Raphson Method

This process is repeated n times to reach a sufficient degree of precision $v_d^{(n+1)}$, which will be regarded as the operating point. Consequently, the general formula is:

$$v_d^{(n+1)} = v_d^{(n)} - \frac{i_d^{(n)}}{\frac{\partial i_d}{\partial v_d}\Big|_{v_d = v_d^{(n)}}}$$

Equation 23 (n+1)th iteration of the Newton-Raphson Method

Equation 23 can then be rewritten in the following form:

$$i_d^{(n)} + \frac{\partial i_d}{\partial v_d}|_{v_d=v_d^{(n)}}(v_d^{(n+1)} - v_d^{(n)}) = 0$$

Equation 24 Newton-Raphson Method rearranged

Upon closer examination, equation 24 describes a tangent at the point $(v_d^{(n)}, i_d^{(n)})$, after performing the Newton-Raphson method n times on equation 22. Hence, with respect to the Newton-Raphson method again, equation 24 represents the (n+1)th iterative solution of the equation, or in other words:

$$i_d^{(n+1)} = i_d^{(n)} + \frac{\partial i_d}{\partial v_d}|_{v_d=v_d^{(n)}}(v_d^{(n+1)} - v_d^{(n)})$$

Equation 25 (n+1)th iteration of the Newton-Raphson Method rearranged

To further manipulate equation 25 into a relatively recognisable form, $\frac{\partial i_d}{\partial v_d}|_{v_d=v_d^{(n)}}$ is denoted with the symbol $g_d^{(m)}$ and the bracketed term is distributed. This results in the following equation:

$$i_d^{(n+1)} = i_d^{(n)} + g_d^{(m)}(v_d^{(n+1)} - v_d^{(n)}) = (i_d^{(n)} - g_d^{(m)}v_d^{(n)}) + g_d^{(m)}v_d^{(n+1)}$$

Equation 26 (n+1)th iteration of the Newton-Raphson Method, as a sum of currents

As shown in [13], under this frame of reference, the following “companion model” is obtained:

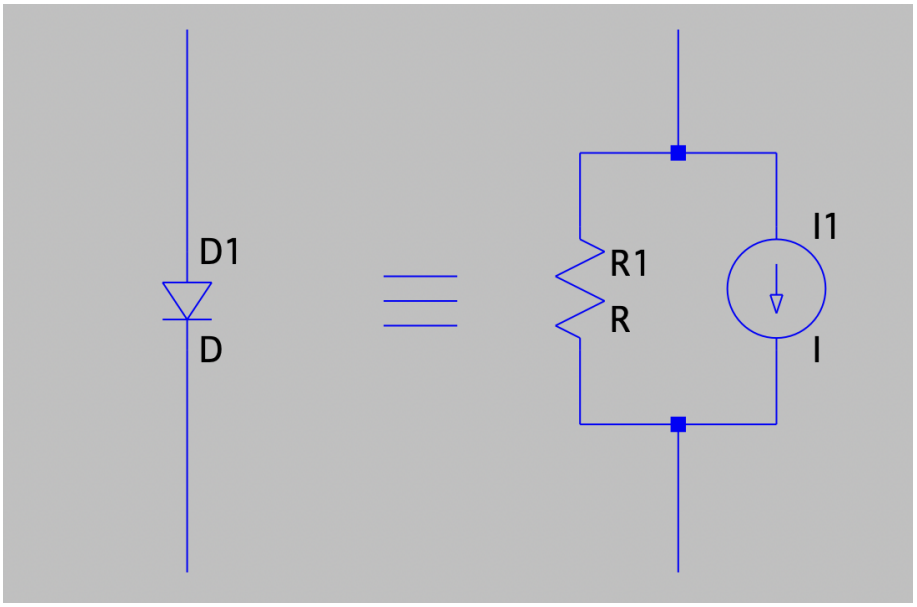


Figure 8 Companion model of the diode

In figure 8, $i_d^{(n+1)}$ can be understood as a sum of currents, with the first term representing a current source of value $(i_d^{(n)} - g_d^{(m)}v_d^{(n)})$ and the second term representing a resistor with a conductance of value $g_d^{(m)}$.

Since this model contains a current source and a resistor, certain entries of the conductance matrix and current matrix have to be iterated multiple times before reaching the final values. The efficiency of this operation during transient simulations is a topic of interest that will be discussed in section 4.

2.3.2.2. Parameters for the Newton-Raphson Method

The parameters for the Newton-Raphson method were intentionally disregarded in the previous section, in order to maintain the flow of relevant derivations and calculations. However, these parameters must be understood to prevent errors throughout the calculations, and to ensure a trouble-free execution during computerised analysis. The two main parameters, as described by Davis [5], are:

1) The initial point, $v_d^{(0)}$; and,
2) The number of iterations to reach the final $i_d^{(n+1)}$

For the Newton-Raphson method to work, the initial point $v_d^{(0)}$ must be “near” the solution, and the function must be smooth and well-behaved. For the purposes of this project, the function is always smooth and well-behaved, since the function i_d in its original form (the Shockley equation, equation 21), is an exponential equation, which is smooth within its entire domain.

Obtaining the correct initial point is important. Consider figure 9, which graphs the diode characteristic g (in green) and the load line f (in black), which is determined using the open circuit voltage and short circuit current of the diode. Note that this graph is only used for illustrating numerical overflow, and that this load-line method is actually not necessary for computation.

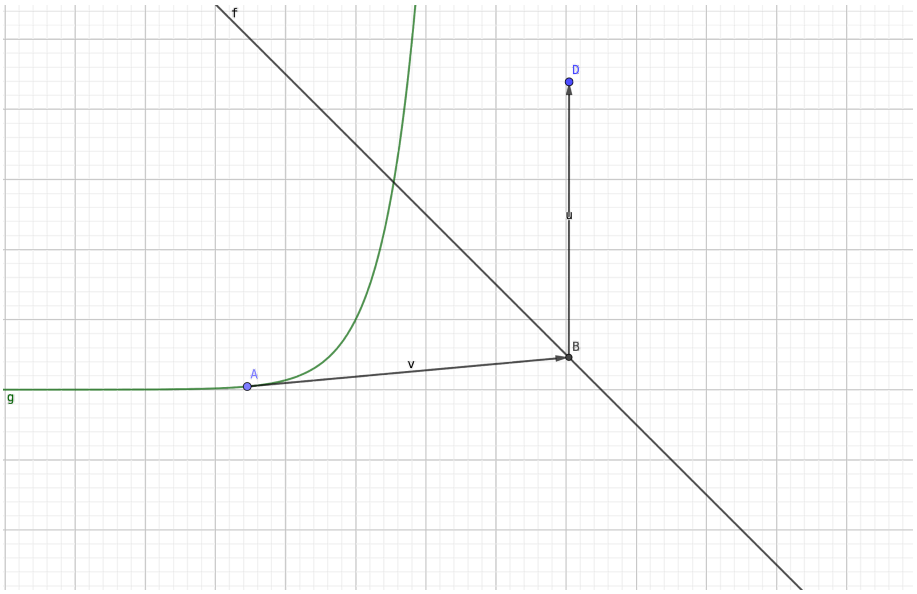


Figure 9 Graph of diode characteristic and load line

In this case, selecting a small initial $v_d^{(0)}$ will will lead to numerical overflow, where the slope of the function is infinite and hence insolvable. In other words, it prevents the algorithm from converging towards a solution.

To achieve the highest possibility for convergence, the following guidelines are established [5]:

Create a thevenin equivalent for the rest of the circuit, at the diode terminals. Let V_{OC} be the open circuit voltage of the thevenin equivalent, and $v_{d_{on}}$ be the typical threshold voltage for the diode, which is specific to the model of the diode.

1) If $V_{OC} < v_{d_{on}}$, then choose V_{OC} as the initial point.
2) If $V_{OC} > v_{d_{on}}$ or $V_{OC} \approx v_{d_{on}}$, then choose $v_{d_{on}}$ as the initial point.

In addition, an algorithm must be established to monitor the precision of the final result, so as to efficiently achieve a result of certain degree precision. In spite of the fact that comparing the differences in between consecutive values may reduce the number of iterations, fixing the number of iterations to reach the final $i_d^{(n+1)}$ remains a quick and viable alternative.

According to Davis [5], it typically takes 3-6 iterations to reach a solution that is correct to 6 decimal places. For consistency, the program always performs 6 iterations to reach the final $i_d^{(n+1)}$.

2.3.2.3. Example of solving circuits with diodes

As a result of the companion model, the diode can now be treated like other components in the circuit. Consider the circuit in figure 10.

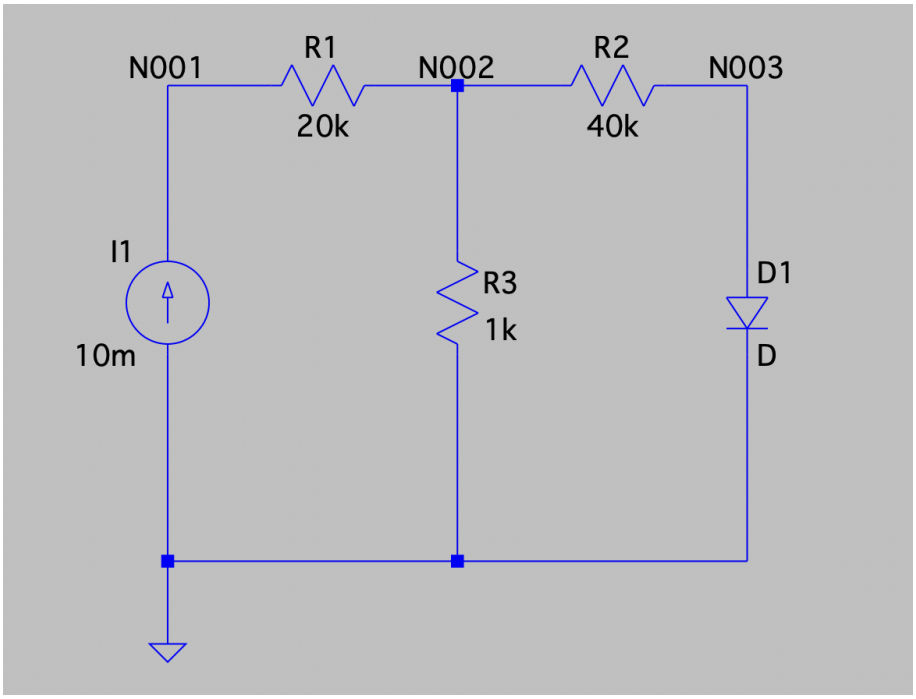


Figure 10 Diode circuit

Assume the typical threshold voltage for the diode is $v_{d_{on}}$. Using the advanced circuit analysis tools outlined in section 2.2.1, the following system of equations is obtained, in its matrix form. Note that the equivalent conductance and current source of the companion model for the diode is now accounted within the equations.

$$\begin{bmatrix} G_1 & -G_1 & 0 \\ -G_1 & (G_1+G_2+G_3) & -G_3 \\ 0 & -G_3 & (G_3+g_d^{(n+1)}) \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} I_1 \\ 0 \\ -i_d^{(n+1)}+g_d^{(n+1)}v_d^{(n+1)} \end{bmatrix}$$

Equation 27 System of equations for circuit in figure 10, in matrix form

In context, the parameters must be obtained iteratively. For example:

$$i_d^{(n+1)} = i_d^{(n)} + \frac{\partial i_d}{\partial v_d}|_{v_d=v_d^{(n)}}(v_d^{(n+1)} - v_d^{(n)})$$

$$g_d^{(1)} = \frac{\partial i_d^{(0)}}{\partial v_d}|_{v_d=v_d^{(0)}} = \frac{I_S}{V_T}e^{\frac{v_d^{(0)}}{V_T}}$$

While the overall method appears to be straightforward, determining certain nodal voltages before iteration remains an area of concern, since those values are required for determining the initial starting point. This issue will be discussed in section 4.

3. Project Management

Planning and management is always a crucial element for large scale projects, even if the actual development process deviates from the original plan. The following section outlines the initial plannings involved, the key changes throughout the course of the project, and a revised high-level software development life cycle of the project.

3.1. Project Planning

Project planning was initiated immediately following the release of the initial project specification. After forming the team, as regards deadlines, the following initial plan was drafted:

Date	To do
7-8 May	● Identify operational, functional and non-functional requirements of the circuit simulator project in the initial meeting. ● Initiate team-building activities to develop team dynamics.
11 - 20 May	● Research on Modified Nodal Analysis, the Newton-Raphson Method, and explore various methods to perform matrix arithmetic in C++. ● Develop conceptual designs of the simulator, and discuss features to implement in simulator. ● Initiate source control
21 - 22 May	● Decision: agree on conceptual understanding of conductance matrices, select library for matrix inversion. ● Construct initial design of various base classes
25 - 30 May	● Implement preliminary design for simulator - should support circuit configurations with resistors, current sources and voltage sources for DC (biased) simulations. ● Perform testing to verify accuracy of simulator.
1 - 5 June	● Complete detailed design for simulator - should support transient simulations for all components specified in the specification. ● Perform testing to verify accuracy of simulator.
8 - 12 June	● Optimise simulator to support larger circuit configurations. ● Review final design of the simulator ● Complete video demonstration of circuit simulator, formal report, and prepare code files for submission
14 June	Submit project

Although there were no clear directions given initially, the goal at the time was to establish a clear understanding of the resources and time required upfront.

3.2. Project Development

Throughout the development of the circuit simulator, critical evaluation had been constantly performed. This involved:

- Constantly testing and verifying the design for new problems;
- Implementing repetitive amendments to the conceptual and actual design for various elements within the project; and,
- Building new or alternative elements to optimise the efficiency of current elements and to extend the functionality of the simulator.

3.2.1. Milestones and Key changes throughout the project

Some problems led to a complete outlook of certain elements. The key changes made during development were:

Date	Event
21 May	Completed class architecture for components and sources
24 May	Instead of keeping track of nodes in a vector of strings, a new class called “node” was made, which contains information about its own identity, the components connected to it, and the nodes to which these components are connected to. It was developed to aid the insertion of voltage sources in the conductance matrix.
27 May	Deliverable can support current sources and resistors.
3 June	Used the new nodal analysis algorithm which accounts for currents through voltage sources. Booleans (v_source_pos, v_source_neg, v_source_ground) and relevant methods previously used to identify ways nodes were connected to voltage sources were removed. The new method, albeit the larger size of the conductance matrix, provides better support for capacitors, since their equivalent voltage depends on the current from the previous timestep.
3 June	Deliverable supports voltage sources as well, and transient simulation outputs are MATLAB compatible.
8 June	Adopted the companion model for the diode, which allows a diode to be treated as a parallel combination of a resistor and controlled current source.

For reference, a formal log for the entire project development is included in Appendix 8.3 .

3.2.2. Project Development Cycle

A project development cycle refers to the stages of development that a team progresses through during the course of a project; it is the process of managing the efficiency and quality of goals achieved, and to relieve stumbling stages during development [20]. However, the development cycle of this software oriented project greatly differs from that of conventional hardware projects, and a number of adjustments has been made to sustain the efficiency for development in a remote working environment. In context, the crucial phases of development involved in this project are:

1) Analysing the functional and non-functional requirements,
2) **Research and planning,**
3) **Software architectural design, or the process of defining components and interfaces required for the program,**
4) **Build software, during which most of the coding is done,**
5) **Testing and verification,**
6) **Documentation in various means, including logs, reports and videos,**
7) Deployment, or in this context, submission; and,
8) Formative and summative feedback, for instance asking questions on Piazza.

For Rectifiers, a revised agile model was later adopted, with each sprint cycle consisting of **stages 2-6** from the list above. In other words, following the iterative development cycle of Agile [1], these stages were repeated for numerous iterables, where each iterable is continuously delivered to add support for more components. Since a Gantt Chart provides a useful outlook of the entire development [28], the revised software development life cycle is displayed in a Gantt Chart:

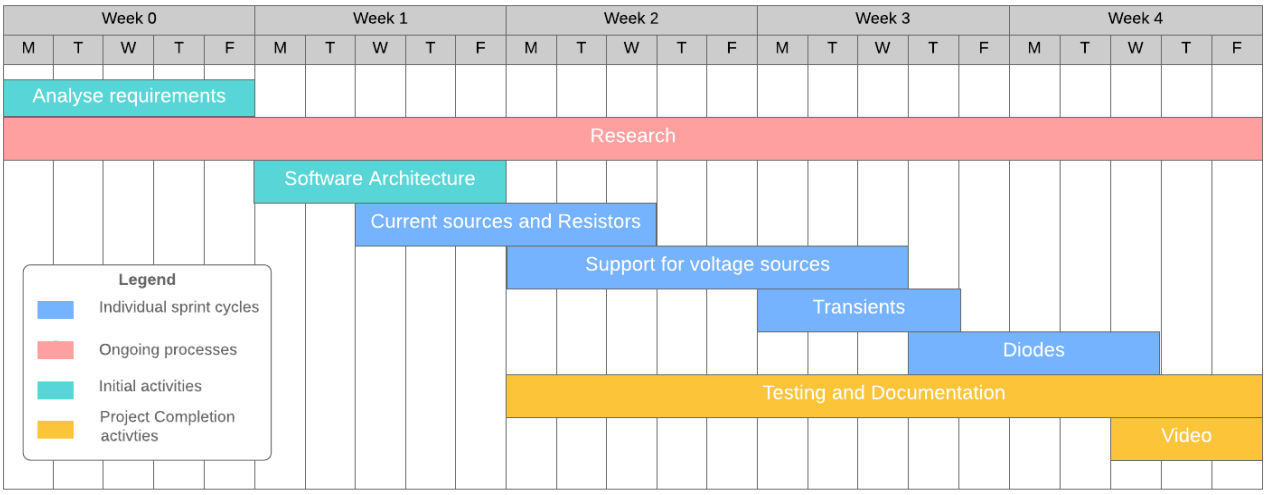


Figure 11 Gantt Chart for the Software Development Life Cycle of this project

9)

3.3. Managing collaboration and teamwork

Effective collaboration is highly critical for completing large projects, as different team members work and think in naturally dissimilar ways [24]. Various communication channels and source control management platforms have been adopted, such that the progress of the project remains transparent to every member.

3.3.1. Communication Channels

Regular discussions are held to review the current progress; this includes making plans together, discussing the architecture of the code, and verifying the accuracy of tests cooperatively. In addition, 5 Discord conferences have been held throughout the course of the project. Typically, this occurs at the beginning of each individual sprint cycle as outlined in figure 11, where research findings and initial designs are discussed. For instance, the first meeting was mostly discussions on creating a versatile class architecture that is highly extensible for future developments.

Finally, and most important of all, the logs and deliverables are always accessible by every member of the team. This includes:

- To-do list and and log,
- Draft report; and,
- The repository of the entire circuit simulator project on Github - this platform of source control is reliable for areas with subpar internet connection, and maintains a consistent record of versions for previously written code.

A complete record of logs and relevant links is included in Appendix 8.3 . This is a compiled log for the minutes of Discord conferences held, as well as relevant conversations on Whatsapp and Discord Chat, in which a total of over 4000 messages and 200 images have been sent.

3.3.2. Collaboration

To accommodate for different team roles, the three members are each involved at various stages of development, in order to take the full advantage of the strengths of the team. For instance, the software development phase is split across members according to the reported Belbin roles:

Belbin Roles	Assigned Task
Resource Investigator Shaper Monitor Evaluator	- Conduct research on various abstract methods for approaching circuit analysis, - Maintain comprehensive documentation over the course of the entire project; and, - Verify accuracy and precision of outputs from the deliverable.
Coordinator Specialist Completer Finisher	- Initiate software architecture design of project and maintain structural integrity of members and methods of classes, - Manage overall structure of code; and, - Optimise efficiency of methods
Shaper Implementer Team-worker	- Design and implement new features, - Utilise the established members and methods for building code in the main file; and, - Troubleshoot bugs.

table 4 Scheme of work distributed, according to self-reported Belbin roles

In spite of distributing the work according to the Belbin roles, the actual work pattern resembles a big bang model; that is, members working together on features simultaneously to reach a goal in a shorter period of time.

A more ideal scenario would be to converse ideas at an earlier stage, such that each member is better informed of the overall direction of the project. This would allow errories to be identified and an earlier stage, reducing the number of times the entire program has to be reconstructed with a different mindset.

4. Circuit Simulator Design

For the following section, the technical details of the development undergone throughout the circuit simulator is outlined, though a great emphasis is placed on exhibiting the development and implementation of the final design. Various references are made to section 2.2 “Literature Review” to aid the understanding of the final design, and to justify the features implemented for final design of the circuit simulator.

To illustrate the designs of various features in an efficient and concise manner, flowcharts and diagrams are utilised to demonstrate key characteristics and processes within the simulator. Code excerpts are also occasionally included, to display the members of the class and the declarations for various methods. A full copy of the code is available in appendix 8.1.

4.1. Interfaces

This section describes the interfaces involved in the application. For the nature of this project, the hardware and communication interfaces are omitted.

4.1.1. Input Interface of the Simulator

Since the deliverable is only a program rather than a complete package, the simulator does not have a graphical user interface.

In terms of the input interface, the program reads a SPICE-like netlist which contains information about the sources and components of a circuit and the simulation directive. Details regarding the format and constraints of the input are outlined in appendix 8.3, which is the “EE1 Project 2020 - Circuit Simulator File Format” document provided initially. As a side note, there is also a simulation directive called `.op`, which takes in no parameters, and performs DC analysis.

The entire process is as follows:

1) Navigate to the directory of the simulator project in the Ubuntu 18.04 terminal
2) Compile the program by typing:

```
chmod u+x build_simulator.sh
./build_simulator.sh
```

3) Then run the simulation by typing:

```
<$NETLIST.txt$ ./simulator >simdata.txt
```

Where `$NETLIST.txt$` is the SPICE netlist file for the simulation.

There is also a variant of the simulator available that can be built using the script file `./build_simulator_improved.sh` using the procedure above. This program differs in that:

- It offers better efficiency
- By default, the system simulation timestep is $t_s = \frac{t_{stop}}{10000}$. The program chooses the timestep differently depending on the specified timestep,
    - If the user specifies a timestep that is sufficiently small, the user’s timestep is chosen as the system timestep for the simulation; whereas,
    - If the user chooses a timestep that is too large, more specifically one that is larger than t_s, then t_s is chosen as the simulation timestep instead. This is because small inductors and capacitors should ideally be simulated with smaller timesteps to avoid

abrupt changes in current or voltage. For the majority of reactive components available in the lab, this timestep is a sufficient and conservative estimate.

4.1.2. Output Interface of the Simulator

In terms of the output interface, the program returns a tab-delimited table of results, where:

- Row 1 contains headers,
- Column 1 contains time values,
- Remaining columns contain branch current or node voltage values; and
- Every row is data for a subsequent timestep.

For example, the first few lines of output for a transient simulation of the circuit in figure 1, with the transient directive `.op`, is as follows:

```
time    V(1)    V(2)    I(R1)    I(R2)    I(R3)    I(I1)
0    14.4    5.4    -0.0072    -0.0018    0.0018    0.009
```

Since `simdata.txt`, contains all the node voltages and branch currents, the data points specified by this text file then can be plotted on MATLAB using the provided script file `plotsim.m`, attached in appendix 8.1.

4.1.3. Software Interfaces

The program is designed to be performed in an Ubuntu 18.04 environment. Hence the program must be run on the terminal within:

- The Ubuntu 18.04 operating system,
- An Ubuntu 18.04 virtual machine; or,
- Docker, WSL or similar applications with an Ubuntu 18.04 engine.

While the following interfaces are not required, they are recommended additions:

1) LTSpice, where users can draw a circuit and export the netlist.
2) MATLAB and the Symbolic Math Toolbox, which plots the results for transient simulations. The `plot_sim.m` file is used to plot the tab-delimited table of results.

4.2. Design and Implementation Constraints

The following section briefly illustrates the actual implementation of the simulator. A list of links are available in appendix 8.1, which points to the relevant cpp files on the team's GitHub repository.

4.2.1. Classes and Constructors

Since object oriented programming is adopted, a diagram, figure 12 on the overleaf, is used to illustrate the overall structure of the base and inherited classes, as well as their members and methods. The diagram follows a simplified version of UML (unified modelling language) [29], where '#' denotes protected members, and '+' denotes public methods. The type for members and methods has been omitted for clarity.

The class "base_class" acts as an umbrella class for all the other classes, which allows for inherited functionality. It contains virtual methods that are accessible and overloaded by sources, basic and non-linear components when iterating through arrays of components. The 3 inherited classes are:

1) "basic_component" : It contains information about resistors, capacitors or inductors.
    - `prev_cv` refers to the current or voltage across capacitors and inductors in the previous iterations; this is necessary since these reactive components depend on the precious history of the circuit
    - `tot_acc` is a temporary variable used for calculations, which is appended `prev_cv` to at the end of each timestep.
2) "source": It is a class for current and voltage sources. It contains 3 additional members, namely "output_type", "frequency" and "amplitude" to output relevant waveforms or traces.
3) "Nonlinear_component" contains members and methods for diodes, and transistors (yet to be implemented).

In addition, the class "node" has been defined, which contains a unique node ID (derived from the node name), a vector of pointers to the nodes connected to it, and a vector of pointers to components and sources connected to it. Having a vector of pointers to components is useful for constructing the conductance matrix, and having a vector of pointers to sources, and the other node to which these sources are connected to, allows voltage sources to be properly inserted into the augmented conductance matrix.

Figure 12 Diagram of objects in this program.

4.2.2. Netlist Parser

The netlist parser is a part of the main cpp file, and the role of the parser is to convert the netlist into a format readable by the program. The input can either be passed in via stdin or as a text file, and they key stages are:

1) Declare temporary variables, which are passed in as parameters during the construction of nodes, components and sources;
2) Perform various checks, identify sources and components, construct them with their relevant parameters, and add them to their respective vectors; and at the meantime,
3) Identify nodes, and insert them (as well as information on connected nodes and components) to the nodes vector if they haven't been inserted
4) At the end of file, identify the transient directive.

In addition, there are several helper functions and other processes in place throughout parsing:

1) Scientific converter function has been made to multiply values with their multipliers at various stages of the netlist parser; for instance, a value is multiplied by 1000 if it is labelled with the multiplier "k" ,
2) Ignore comments, or lines starting with `'*'`; and,
3) Ignore shorted components, that is, components with both terminals pointing to the same node.

Figure 13 provides a simplified flowchart of this entire process.

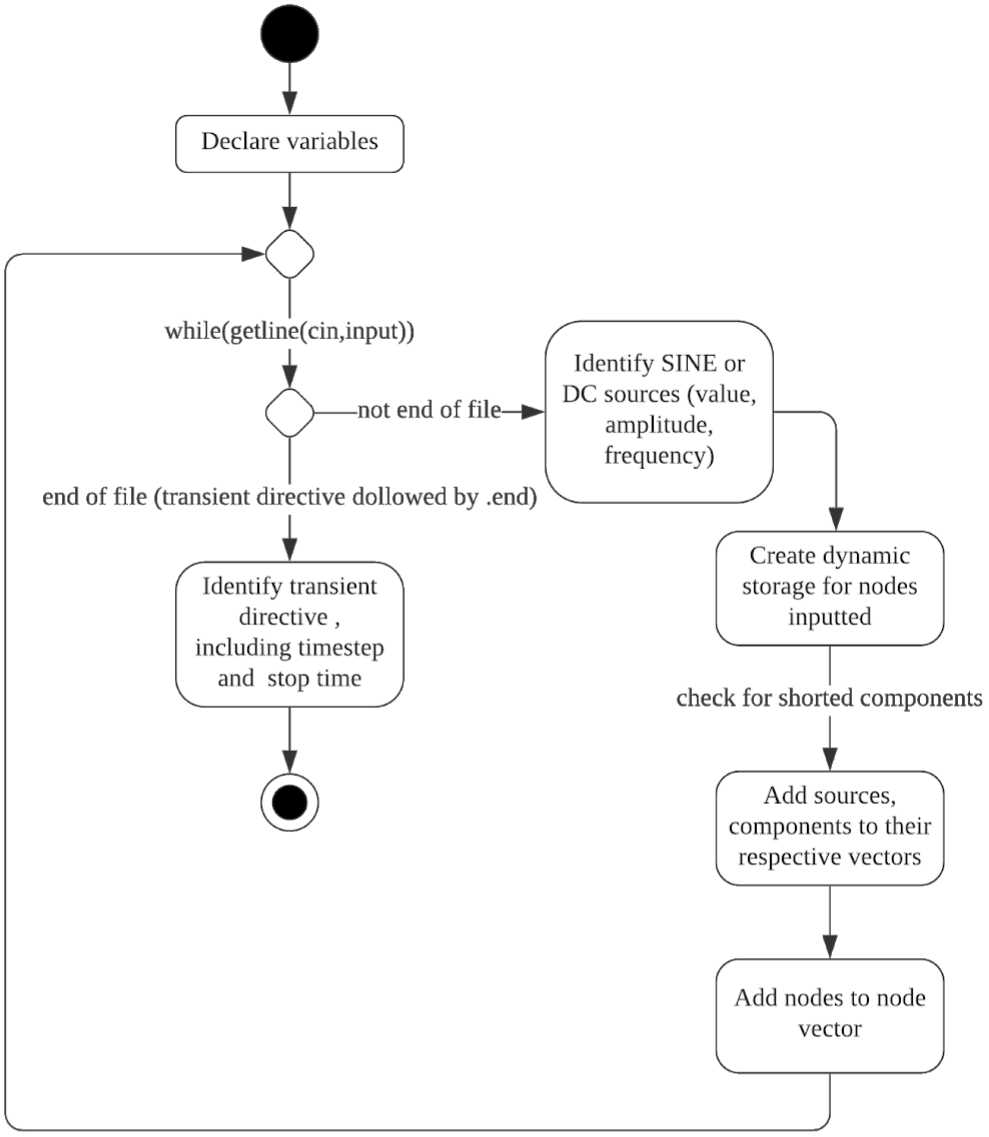


Figure 13 Flowchart for parsing netlist

4.2.3. Conductance Matrices

This section is based on section 2.3.1 “Advanced circuit analysis”. The key iterations of the conductance matrix code are:

1) Support for current sources and resistors only, based on section 2.3.1.1,
2) Support for voltage sources, current sources and resistors, based on section 2.3.1.2,
3) Revised version of iteration 2, using the modified algorithm that accounts for currents through voltage sources,
4) Support for reactive components; and,
5) Support for diodes.

The flowchart overleaf, figure 14, shows the top-level view of the conductance matrix.

Notes:

- Conductances are added as positive increments at diagonals and added as negative increments elsewhere.
- Voltage sources and capacitors are offset with the variable pos during insertion, since they are added to the “augmented” part of the matrix, where the position of entry is offset by the size of the original conductance matrix. “1” is inserted to the entry which corresponds to the node where current is entering the negative terminal of a voltage source or capacitor, whereas “-1” is inserted to the entry which corresponds to the node where current is exiting the positive terminal.

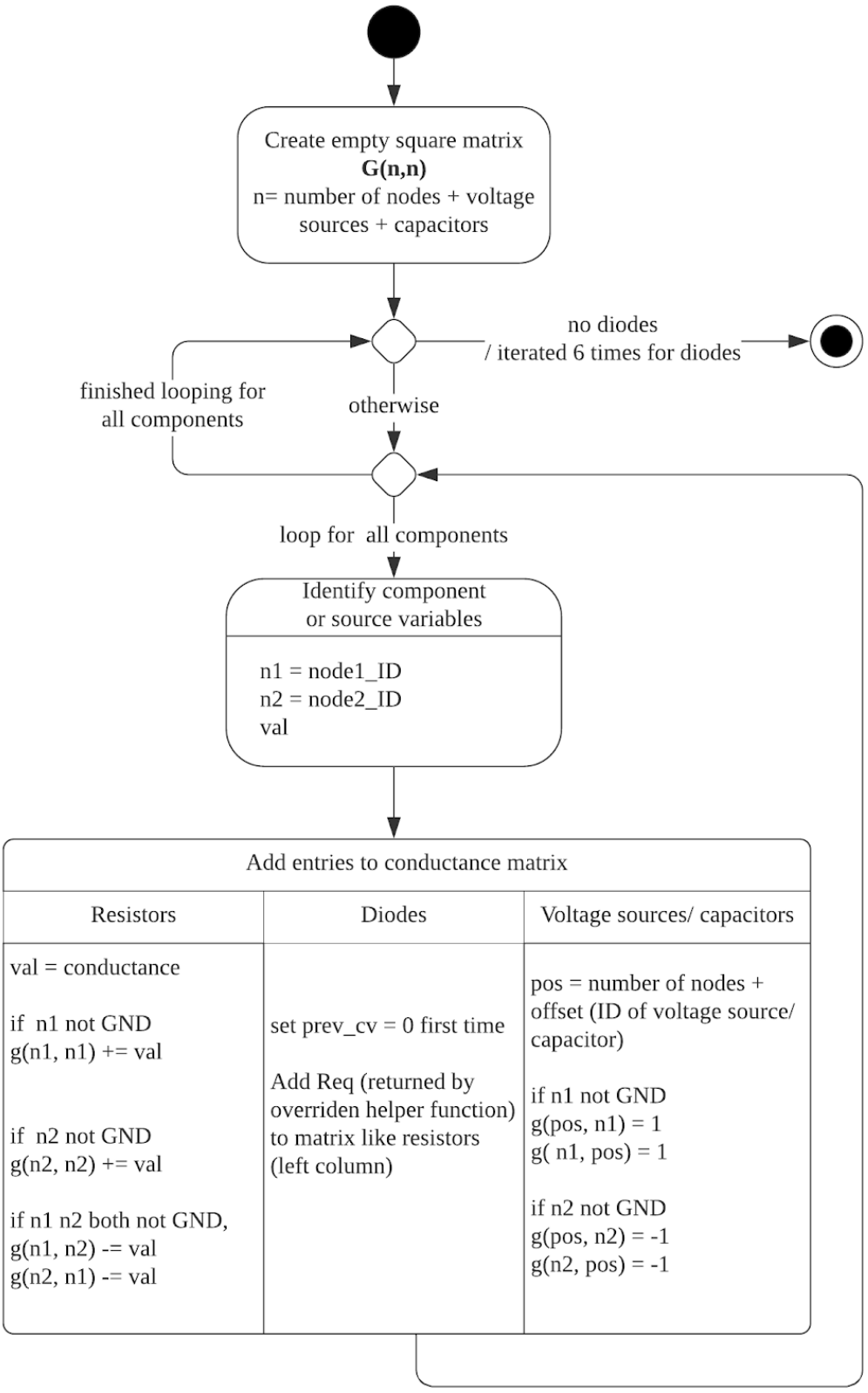


Figure 14 Flowchart for conductance matrix insertion

4.2.4. Current Matrices

Figure 15 shows the overall flow of inserting entries into the current matrix. For non-zero entries, the values are due to 5 types of components or sources:

1) Current sources - the value of the current is inserted at the entry corresponding to the node where the current exits,
2) Inductors (which act as current sources) - same as current sources, except the value returned accounts for the voltage across the inductor in the previous iteration, which is obtained using the `return_value()` function, elaborated below,
3) Voltage sources - the value of the voltage is inserted at the entry corresponding to the node of the positive terminal of the voltage source.
4) Capacitors (which act as voltage sources) - same as voltage sources, except the value returned accounts for the current through the capacitor during the previous timestep.
5) Diodes - contains an equivalent current source, which is found by iteration of the Shockley equation using the Newton-Raphson Method.

The function `return_value()` is a function of timestep, overloaded to return the values used for the current matrix at the present timestep. The details are as follows:

1) For capacitors and inductors, their previous currents or voltages are stored in `prev_cv`.
2) With respect to the numerical integration methods for reactive components outlined in section 2.2.1.3, the previous value current or voltage is multiplied by the timestep and added to the previous charge or voltage to give the total `(tot_acc)`, which is then divided by the capacitance or inductance value to give the current or voltage value at the present timestep.
3) Afterwards, `tot_acc` is stored back in `prev_cv`.

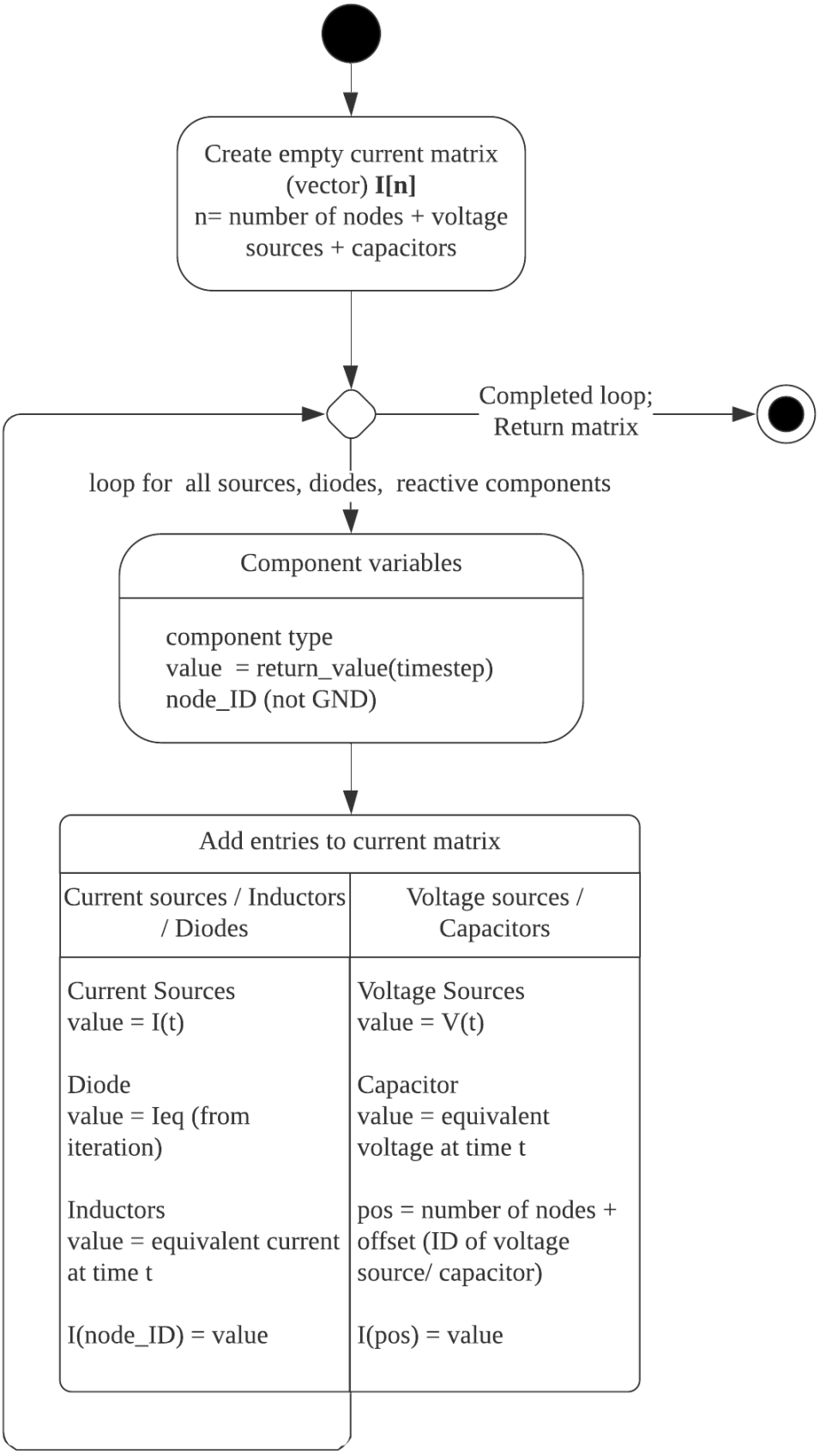


Figure 15 Flowchart for Current matrix insertion

4.2.5. Dependency: Obtaining Voltage Matrix by Inverting Matrices with Eigen

Arithmetically, the voltage matrix is determined by multiplying the inverted conductance matrix with the current matrix. For the purpose of the project, a high level matrix-computing library has been selected to determine the voltage matrix, resulting in the following code:

```
MatrixXd v(h,1); // matrix dimensions, h = no. of nodes + v. sources + caps
v = g.fullPivLu().solve(current);
```

Under the hood, the conductance matrix (or the augmented version of it) is inverted using Eigen, a popular C++ template library that supports various linear algebra operations, most importantly matrix inversion [6]. The advantages of using Eigen are:

- Flexibility, in terms of the range of methods available for matrix decomposition and pivoting,
- Performance, in terms of compilation and execution time,
- Accuracy of results,
- Efficiency - it supports sparse matrices, which is especially useful when completing rows defining potential differences in the conductance matrix, where entries are mostly left uninitialised as zero; and.
- Other features like compute, which extracts the matrix which has undergone LU decomposition. This enables easier debugging.

During LU factorisation, there must be a non-zero pivot at every step of the elimination process such that Gaussian Elimination is properly completed. Although partial pivoting for LU decomposition is the method taught in the Year 1 Engineering Mathematics: Principles and Applications course, for computerised analysis, pivoting is employed not only to make sure that Gaussian elimination is completed, but also to reduce roundoff errors.

Completing pivoting involves both row and column interchanges to place the largest entry of the submatrix in the pivot position before every stage of elimination. In summary, partial pivoting involves row permutation, whereas full pivoting involves both row and column permutation, which pushes the largest entries to the diagonal.

According to Cheney and Kincaid [4], this enables:

- Elimination for singular matrices - there is a probability that submatrices of the conductance matrix, which are sparse matrices where most entries are zero, will not be completed with partial pivoting.
- The selection of the best pivot - it is always ideal to have a diagonally dominant matrix, which is always true if the circuit contains only resistors and current sources, but not necessarily so when there are voltage sources.

- Reduced roundoff errors - it removes near-zero values from the pivots. Roundoff errors are a significant problem when arithmetic operations are performed in between numbers of vastly different magnitudes, since numbers have to conform to the native data type of the CPU.

Therefore, despite its relatively slow execution speed, FullPivLU has been selected over PartialPivLU for numerical stability concerns.

4.2.6. Overall Product Perspective

This section summarises all the consecutive stages that the program advances through. To best illustrate this process, figure, a flowchart is provided overleaf.

The following list outlines several features that deserve further justifications:

- For circuits with only sources and linear components, the conductance matrix is evaluated once within a timestep. The matrix is re-evaluated multiple times within a timestep, only if there are diodes, since the Newton-Raphson Method requires 6 iterations to reach a precise numerical solution for the diode.
- Likewise, the conductance matrix will not be recreated every timestep if there are no diodes in the circuit.
- However, the current matrix (and consequently the voltage matrix) is re-evaluated at every timestep since the voltage and current sources are functions of time. Simultaneously, equivalent voltages of capacitors vary as well.

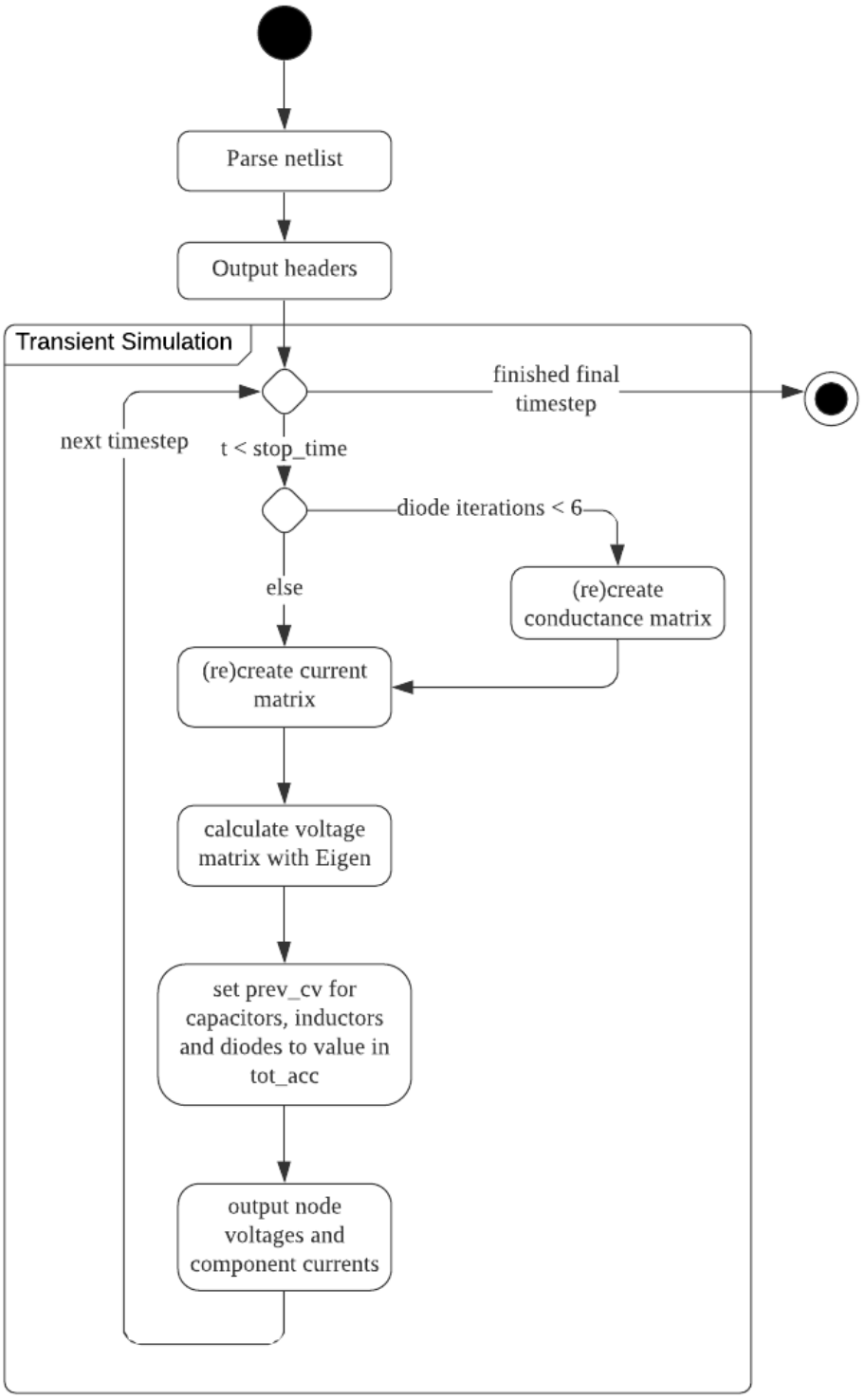


Figure Flowchart for the entire program

5. Discussion

5.1. Evaluation of results

On top of the preliminary tests performed concurrent to code development, this section includes a series of tests designed to test the program against the functional and non-functional requirements listed in sections 2.1.1 and 2.1.2.

For each of the following tests, the following will be outlined:

- Circuit schematic, with the corresponding SPICE netlist attached in the appendices,
- Purpose: for example, this may include verifying specific functionality, or assessing non-functional aspects like efficiency, memory footprint, or scalability; and,
- Simulation Directive,
- Results, which shows whether the functional requirement regarding accuracy is met; a list of steady state values provided for DC circuits, and graphs are provided for AC circuits,
- Execution time, in which user time is considered a relatively reliable proxy indicator for execution time, since:
    - Real is the wall clock time, which may include time used by other processes running on the CPU,
    - Sys is the the CPU time spent in background (kernel) process; and,
    - User is the CPU time in executing the user-specified process.
- Energy consumption, calculated by:

$$Energy = \text{Max. Pwr Consumption} \times \text{Execution Time (User)} \times \% \text{ of CPU used}$$

- Evaluation.

Reference values from LTSpice are included in appendix 8.2.

Nonetheless, certain non-functional requirements like reliability and testability are aspects that are based on the entire evaluation process. Furthermore, other non-functional requirements such as usability, readability, portability or extensibility are unfortunately not testable under current circumstances.

5.1.1. Test 1 - DC Sources and Resistors

Circuit Schematic and Simulation Directive

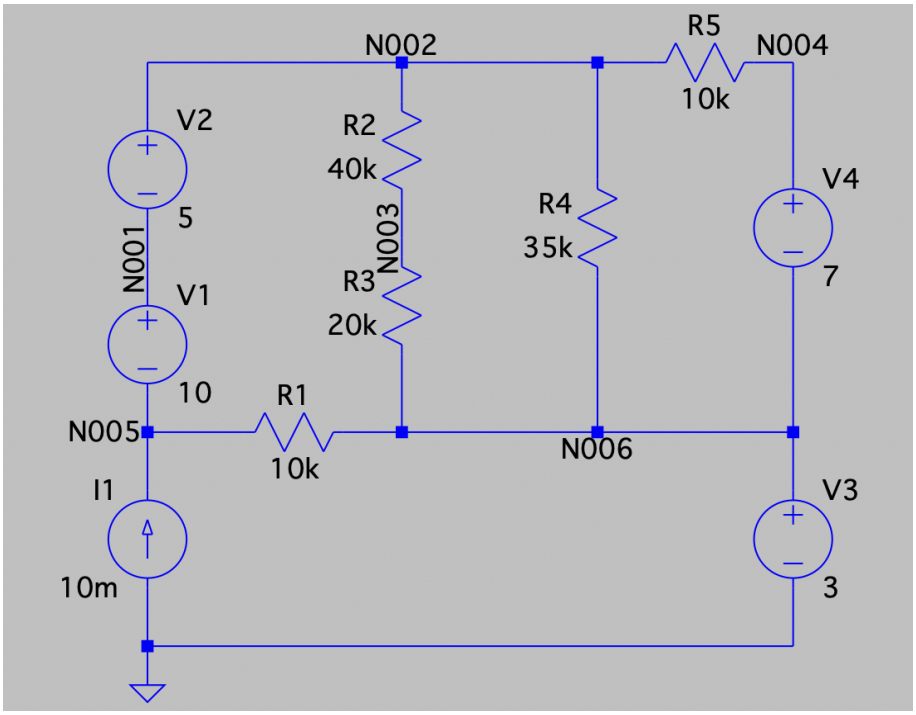


Figure 16 Test 1 Schematic

```
.op (Simulation for operating bias point)
```

Purpose

The circuit tests for:

- Scientific notation in the netlist,
- Non-grounded sources and supernodes,
- Concatenated sources,
- Parallel resistors; and,
- Series resistors.

Operation Efficiency Benchmarks

Performance Time	`real 0m0.098s`
	`user 0m0.003s`
	`sys 0m0.006s`

Energy Consumption	$Energy = 28W \times 0.003s \times 12.8\% = 10.8mJ$

Results: Operating Bias Points (Raw Data)

```
time    V(5)    V(1)    V(2)    V(6)    V(3)    V(4)    I(I1)    I(V1)    I(V2)
I(R1)    I(R2)    I(R3)    I(R4)    I(V3)    I(V4)    I(R5)
0    37.7476    47.7476    52.7476    3    19.5825    10    0.01    -0.00652524
-0.00652524    0.00347476    0.000829126    0.000829126    0.00142136    0.01
0.00427476    0.00427476
10000    37.7476    47.7476    52.7476    3    19.5825    10    0.01
-0.00652524    -0.00652524    0.00347476    0.000829126    0.000829126
0.00142136    0.01    0.00427476    0.00427476
```

Evaluation

Results are correct and output is available in a negligible amount of time with a minimal energy impact on the CPU. This test gives us confidence that supernodes work, even when connected to multiple components in series and parallel.

5.1.2. Test 2 - DC Sources, Capacitors, Inductors and Resistors

Circuit Schematic and Simulation Directive

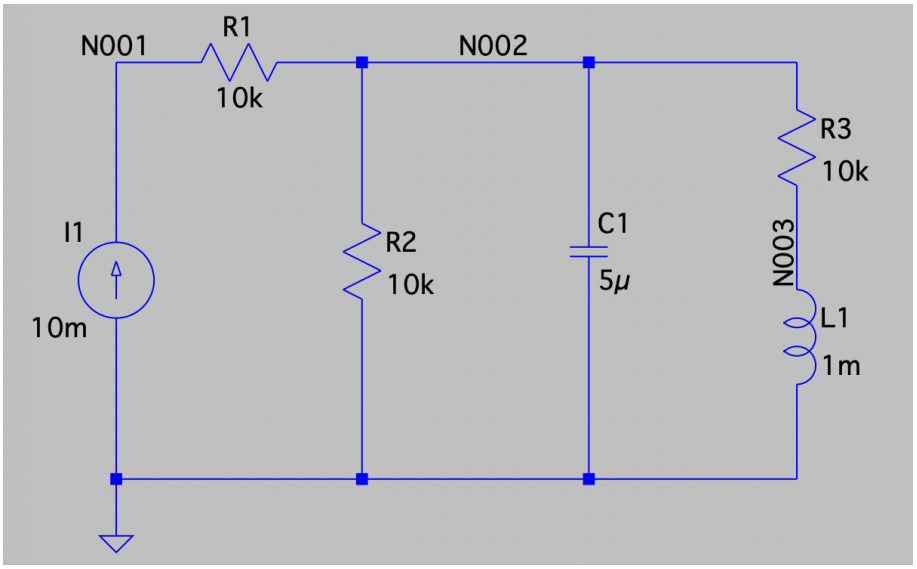


Figure 17 Test 2 Schematic

```
.tran 0 100000 5
```

Purpose

The circuit tests for:

- Behaviour of reactive components when average voltage is applied,
- Continuity characteristics of reactive components; and,
- Transient simulation functionality.

Operation Efficiency Benchmarks

Performance Time	`real 0m17.666s`
	`user 0m2.518s`
	`sys 0m1.101s`

Energy Consumption	$Energy = 28W \times 2.518s \times 11.6\% = 8.18J$

Results: Graph of all nodal voltages and component currents

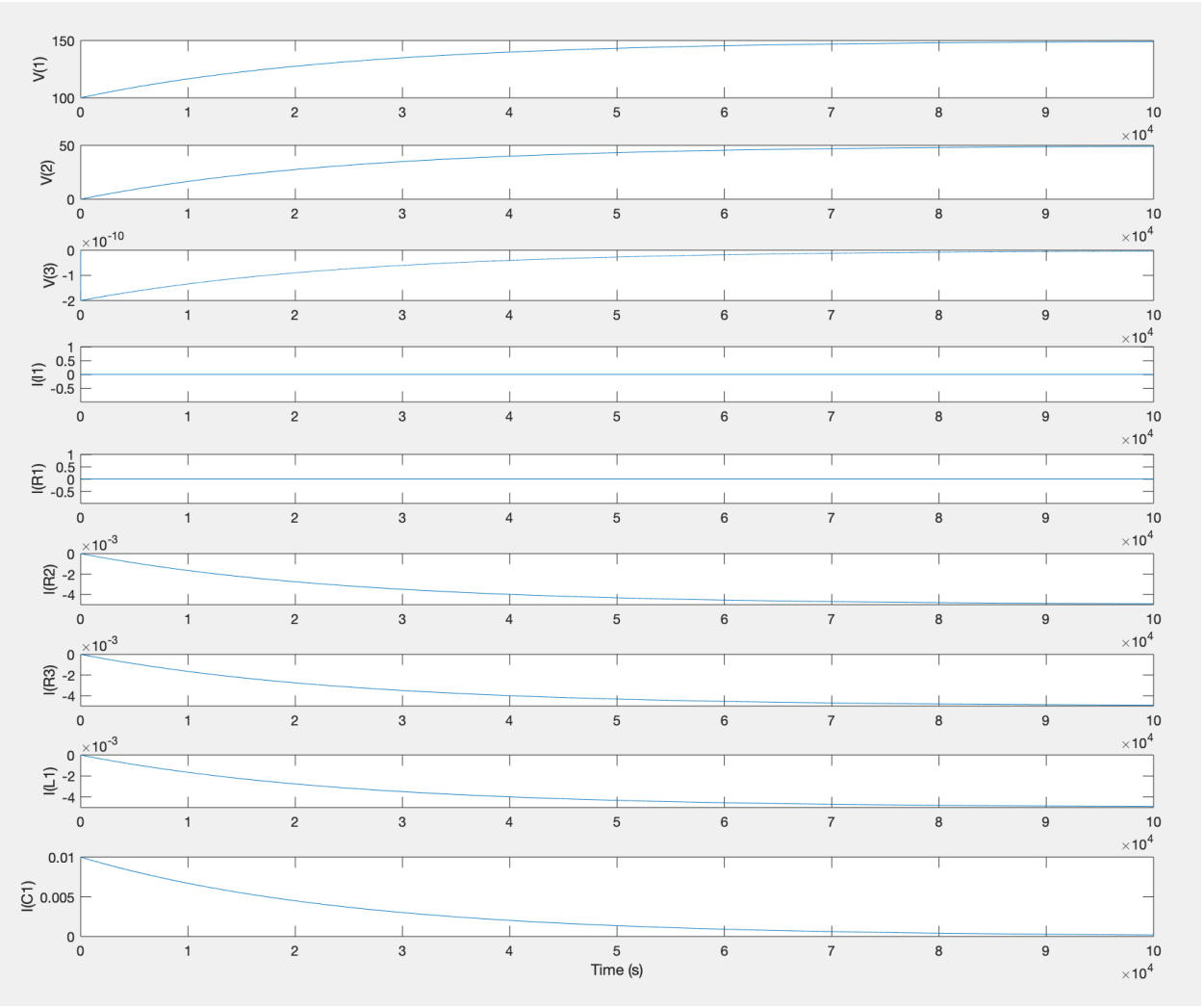


Figure 18 Graphs of results for Test 2

Evaluation

The final steady state values are in line with the LTSpice operating bias point results outlined in appendix 8.2.2. It should be noted that the program took 10,000 seconds to reach this steady state, which is expected given the small value of the capacitor in the circuit.

Performance time is relatively longer than test 1, which was also anticipated, since there are 2000 data points in total, rather than 2 in test 1. Correspondingly there is a significantly higher energy consumption.

This test is proof that our simulator works for capacitors and inductors in the same circuit while doing DC analysis.

5.1.3. Test 3 - AC Sources, Capacitors, Inductors and Resistors

Circuit Schematic and Simulation Directive

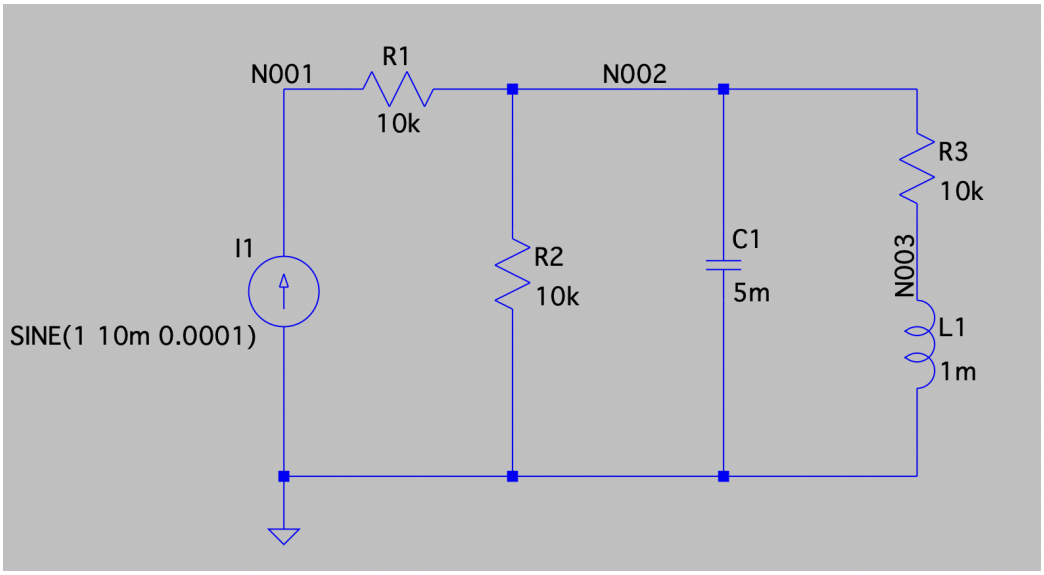


Figure 19 Test 3 Schematic

```
.tran 0 100000 5
```

Purpose

The circuit tests for:

- Behaviour of reactive components when node voltages are time variant,
- Accuracy of AC sources; and,
- Transient simulation functionality and efficiency.

Operation Efficiency Benchmarks

Performance Time	`real 0m16.267s`
	`user 0m2.704s`
	`sys 0m1.119s`

Energy Consumption	$Energy = 28W \times 2.704s \times 12.3\% = 9.31J$

Results: Graph of all nodal voltages and component currents

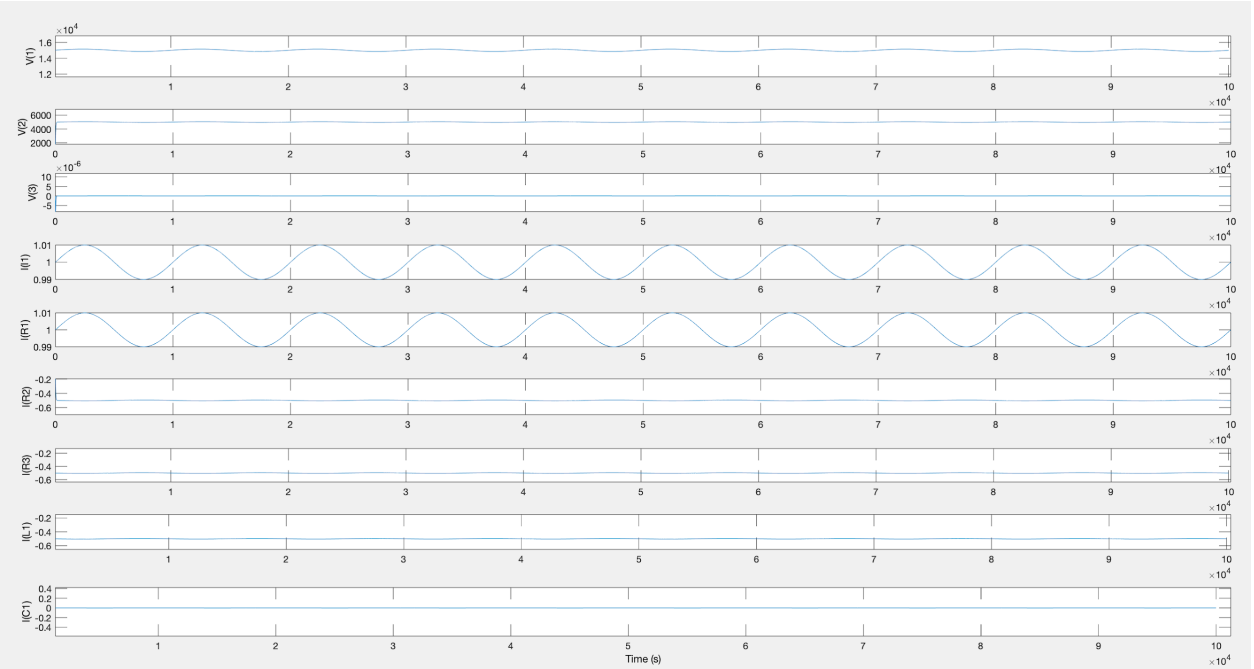


Figure 20 Graphs of results for Test 3

Evaluation

The following graph outlines the results achieved if the capacitor had a smaller capacitance, and the simulation were performed within a shorter time frame.

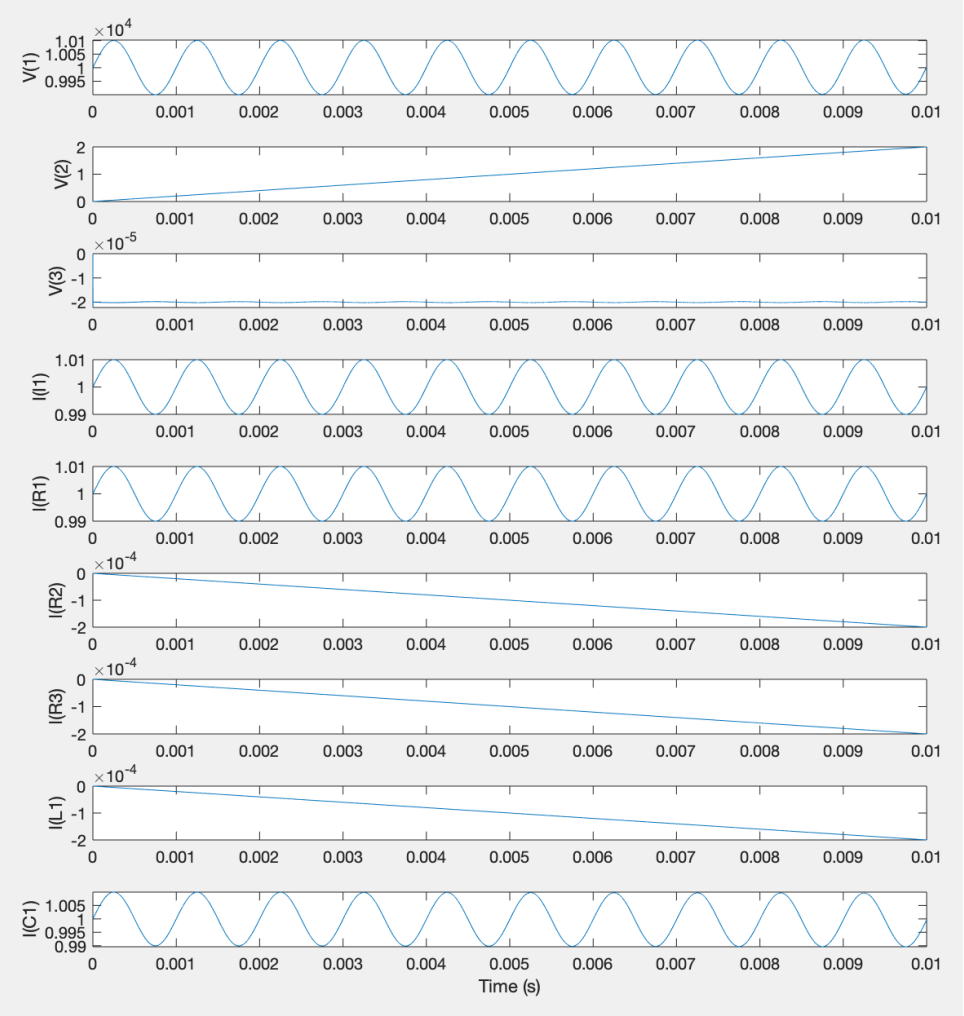


Figure 21 Test 3, with a smaller capacitor and a shorter simulation time

As the graph in figure 21 shows, insufficient time was given for the capacitor to charge to its steady state value, and hence the sine wave from node 1 was not “passed on” to node 2. This reveals the limitations of adopting numerical integration, more specifically the Riemann Sum - a smaller timestep is required for small inductances and capacitances. Nevertheless, it should be noted that modelling the capacitor as a differentiator and current source would have created the possibility of positive feedback and oscillation, when the change in values between time steps are too abrupt. The ideal solution would be to adopt the differential method for circuits with small inductors and capacitors, such that these circuits could be simulated with smaller timestep, and to adopt the present integral method for circuits with larger capacitances and inductances, such that the steady state solution for these components are stable.

Nevertheless, provided that a sufficient time is allowed for the reactive components to reach their steady state, the overall results are correct. In terms of non-functional requirements, the simulation time is approximately 7% longer than test 2, which has the same number of timesteps; it can be deduced that simulation time is highly correlated with the number of timesteps for the simulation.

5.1.4. Test 4 - DC Sources, Resistors and Diodes

Circuit Schematic and Simulation Directive

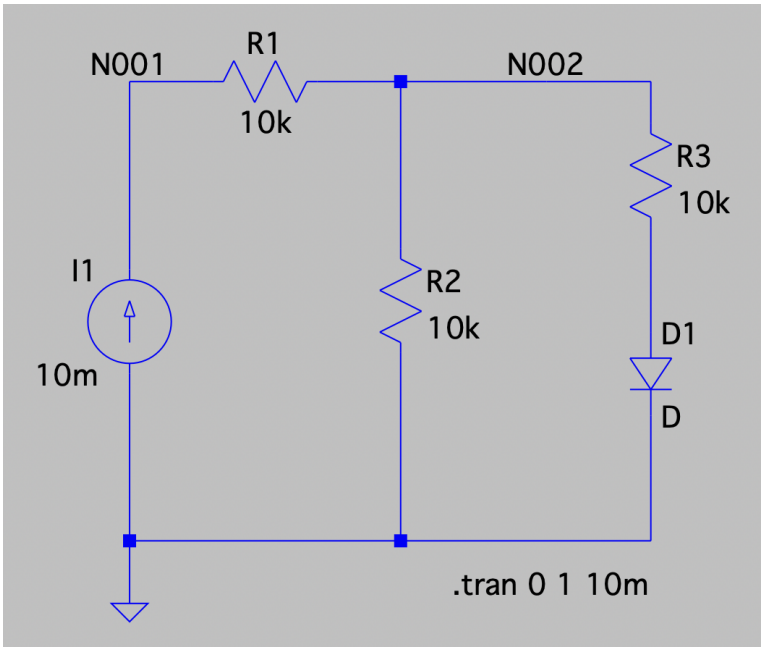


Figure 22 Test 4 Schematic

```
.tran 0 1 0.1m
```

Purpose

The circuit tests for the behaviour of diodes, more specifically the terminal characteristics of diodes at the operating boundary conditions.

Operation Efficiency Benchmarks

Performance Time	`real 0m6.518s`
	`user 0m2.182s`
	`sys 0m0.458s`

Energy Consumption	$Energy = 28W \times 2.182s \times 2.8\% = 1.71J$

Results: Operating Bias Points (Raw Data)

```
time    V(1)    V(2)    V(3)    I(I1)    I(R1)    I(R2)    I(R3)    I(D1)
0    150.35    50.3501    0.700202    0.01    0.01    -0.00503501
-0.00496499    0.00496499
```

Evaluation

Although this circuit can be tested with `.op` (operating bias point), a transient simulation was performed nonetheless, so as to ensure that the values do not actually change over time. This is a necessary test since conductance and current matrices for reactive components do actually change over time. The results indicate that the present simulator is accurate for diodes, and efficient to an appropriate degree, especially considering that 10,000 data points have been produced.

5.1.5. Test 5 - AC Sources, Resistors and Diodes

Circuit Schematic and Simulation Directive

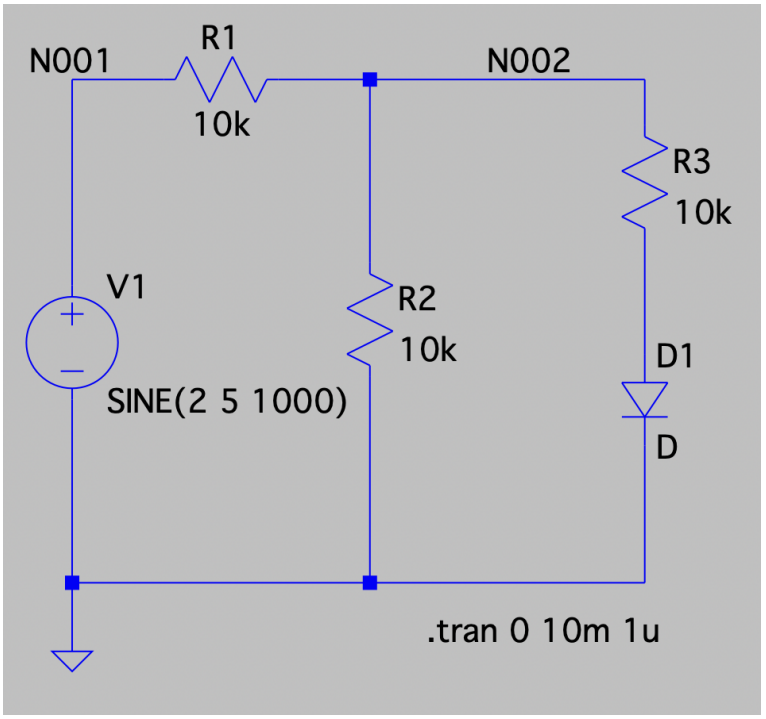


Figure 23 Test 5 Schematic

```
.tran 0 10m 1u
```

Purpose

The circuit tests for:

- Behaviour of diodes with varying voltages at terminals; and,
- Transient simulation functionality and efficiency.

Operation Efficiency Benchmarks (With simulator_improved adopted)

Performance Time	`real 0m8.103s`
	`user 0m2.487s`
	`sys 0m0.488s`

Energy Consumption	$Energy = 28W \times 2.487s \times 4.5\% = 3.13J$

Results: Graph of values

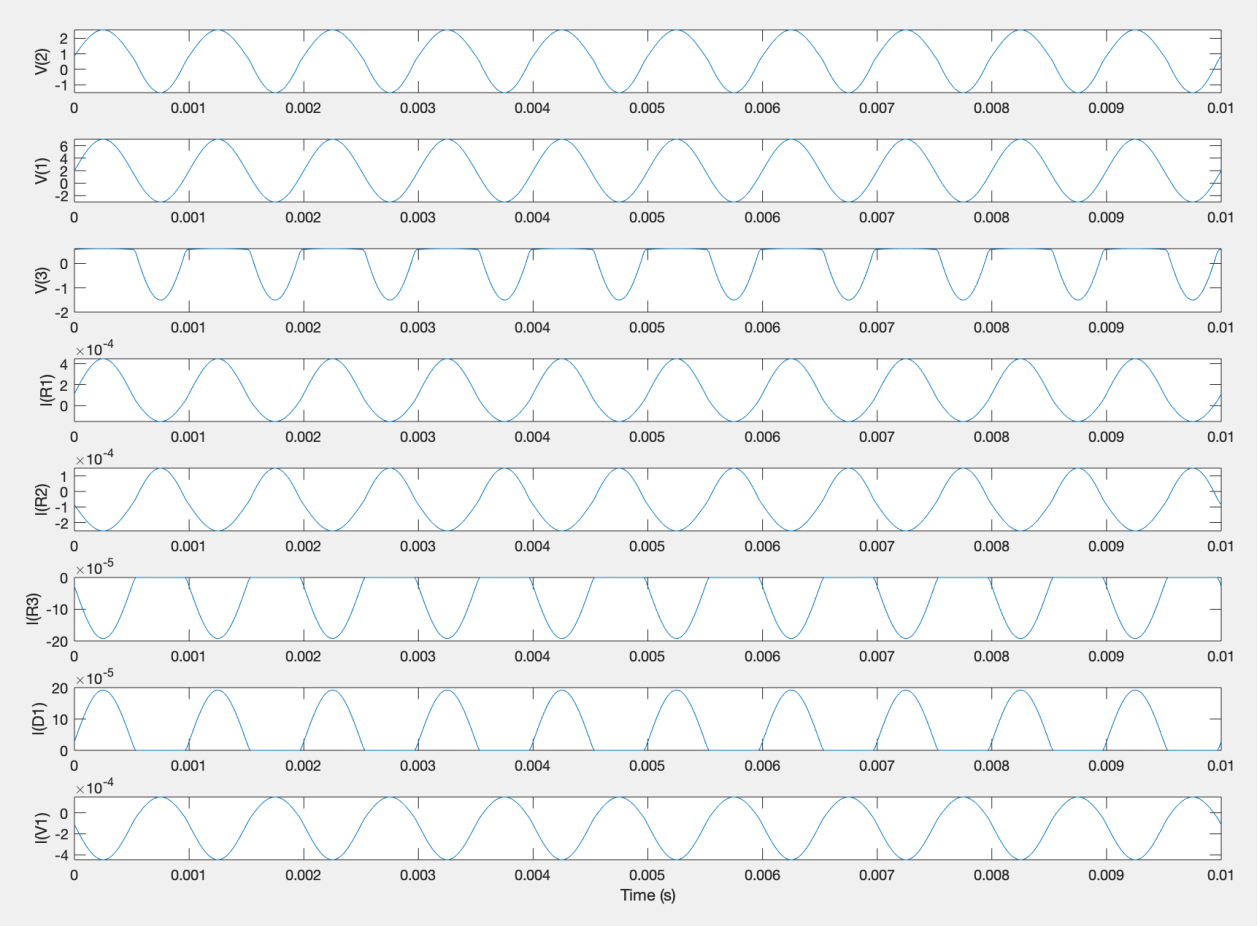


Figure 24 Graph of results for test 5

Evaluation

This circuit has the identical configuration as test 4, but with the DC current source replaced with an AC voltage source. This is done in an attempt to observe the change in characteristics at the boundary conditions of the diode. As expected, I(D1) shows that there is current through the diode only and V(3) shows that the potential across the diode is 0.7 under the same conditions.

In terms of non-functional requirements, this test adopted the “improved” version of the simulator, which significantly reduced execution time and energy consumption, despite the greater number of data points produced.

It is also important to note the results of this test, as we have just proved that diodes work with AC circuits.

5.1.6. Test 6 - All components and AC sources

Circuit Schematic and Simulation Directive

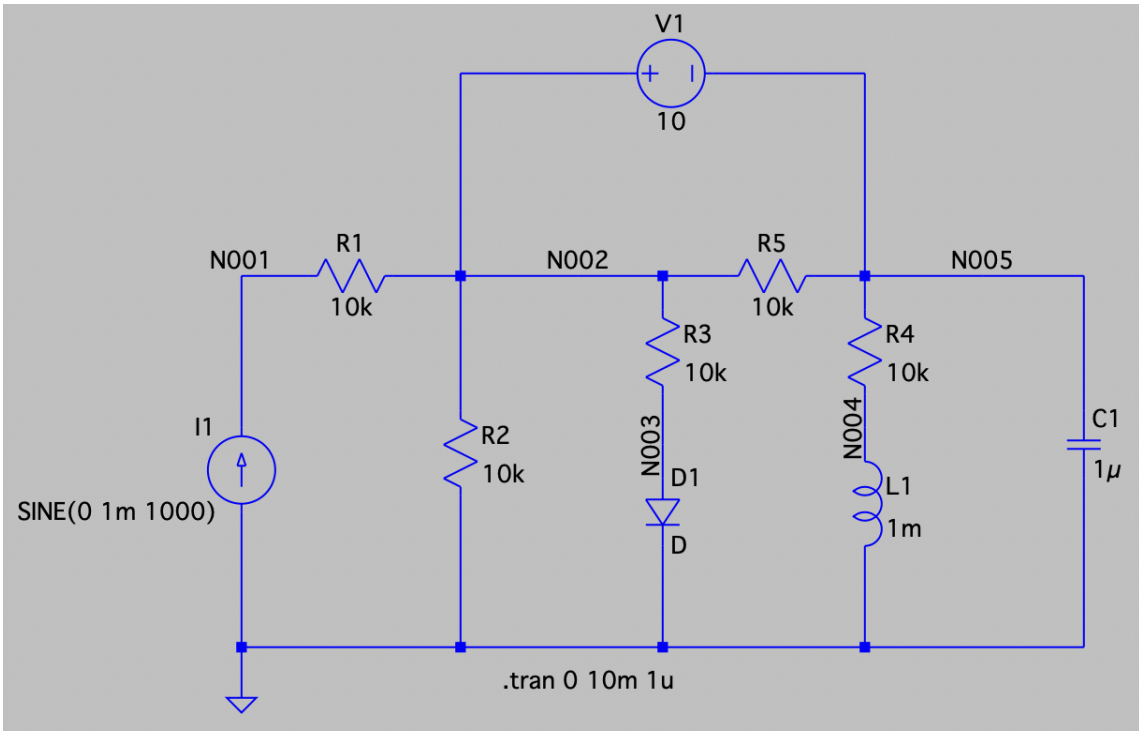


Figure 25 Test 6 Schematic

```
.tran 0 10m 1u
```

Purpose

The circuit tests for:

- Transient simulation functionality and efficiency;
- Supernodes in AC circuits; and,
- Reliability of the program when it comes to reactive components.

Operation Efficiency Benchmarks

Performance Time	`real 0m0.098s`
	`user 0m0.003s`
	`sys 0m0.006s`

Energy Consumption	$Energy = 28W \times 0.003s \times 12.8\% = 10.8mJ$

Results: Graph of values

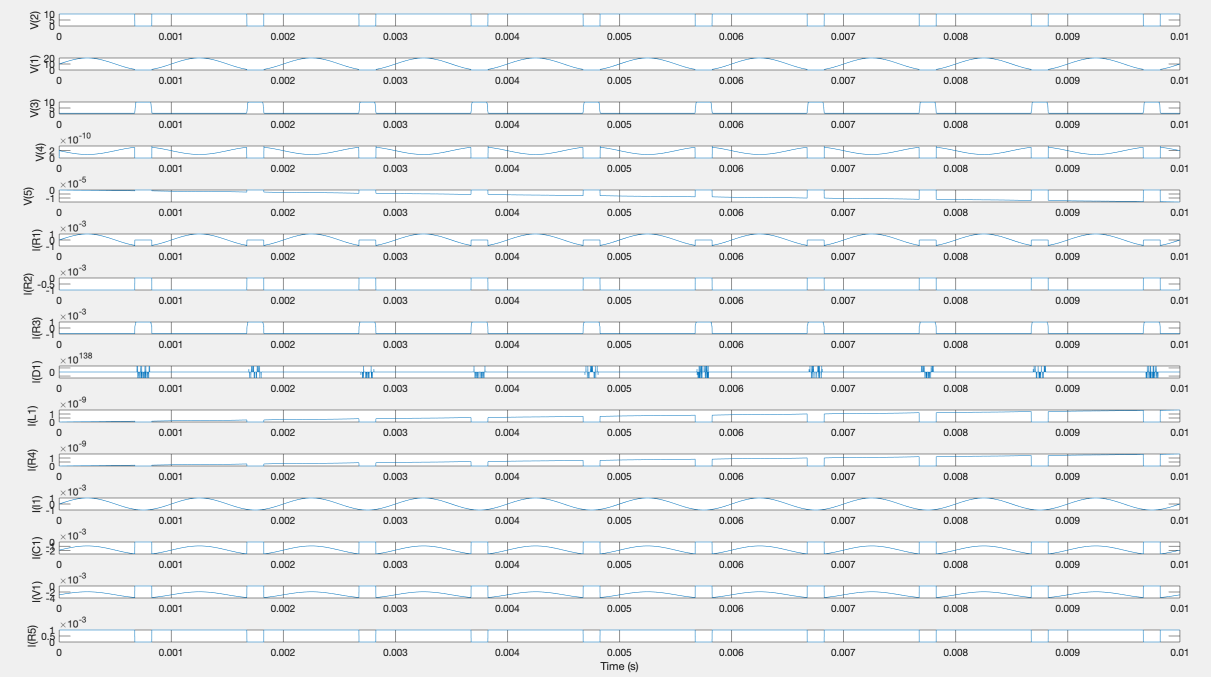


Figure 26 Graph of results for test 6

Evaluation

Unfortunately, non-functional requirements will not be considered for this test since the results are not correct. As seen in the results, the conducting conditions for the diodes are correct, which means the problem is unlikely caused by the diode. In fact, removing the diode made little to no effect on the other nodal voltage and branch currents of the circuit. In addition, a diode is by definition a DC offset during conduction, which reduces the amount of uncertainty around diodes. We can also note the fact that the diodes have been tested by themselves along with resistors in circuits previously, which lowers the probability of the error originating from the diodes themselves.

The key problem here may be:

- The parallel placement of reactive components, in combination with the non-grounded voltage source; or,
- The placement of supernodes in AC circuits. To test this possibility, a revised version of test 1 is performed.

5.1.7. Test 1 Revised- Supernode Circuit revised with an AC Current Source

Circuit Schematic and Simulation Directive

Circuit schematic identical to test 1, except I1 is replaced with : `SINE(0 10m 1000)`

`.tran 0 10m 1u`

Purpose

The circuit tests for:

- Scientific notation in the netlist,
- Non-grounded sources and supernodes,
- Concatenated sources,
- Parallel resistors; and,
- Series resistors.

Operation Efficiency Benchmarks

Performance Time	`real 0m5.203s`
	`user 0m1.848s`
	`sys 0m0.158s`

Energy Consumption	$Energy = 28W \times 1.848s \times 17\% = 8.8J$

Results: Graph of Values

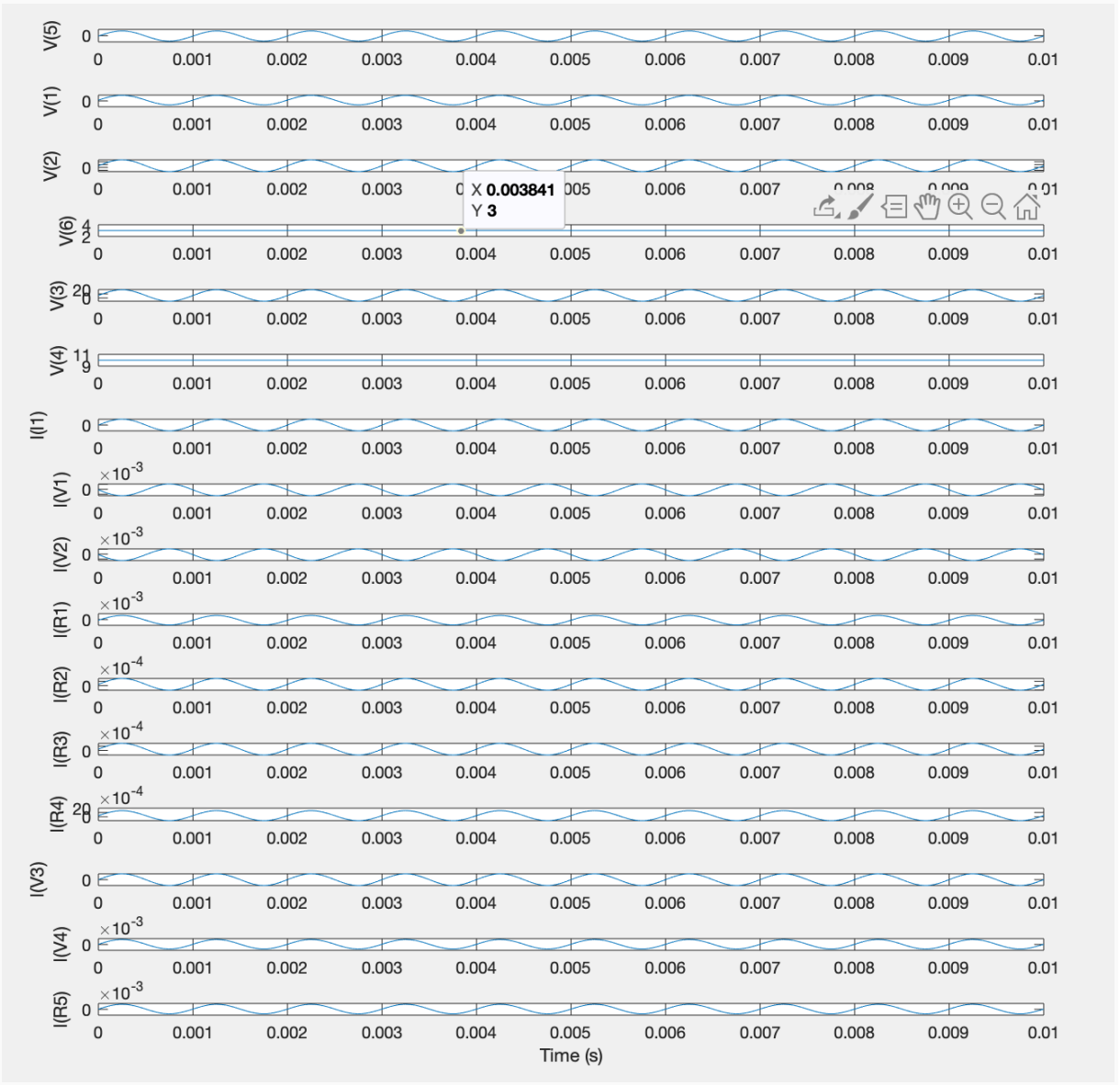


Figure 27 Graph of results for test 1, modified

Evaluation

Results are correct when there are supernodes in an AC circuit, which indicates that results in test 6 are incorrect most likely due to the placement of the reactive components. From a general viewpoint, it is a high likelihood that the capacitors and inductors in test 6 have reacted to the delay in changes of current and voltage in other reactive components, which have led to unanticipated feedback effects.

5.1.8. Test 7 - More complex test on reactive components

Circuit Schematic and Simulation Directive

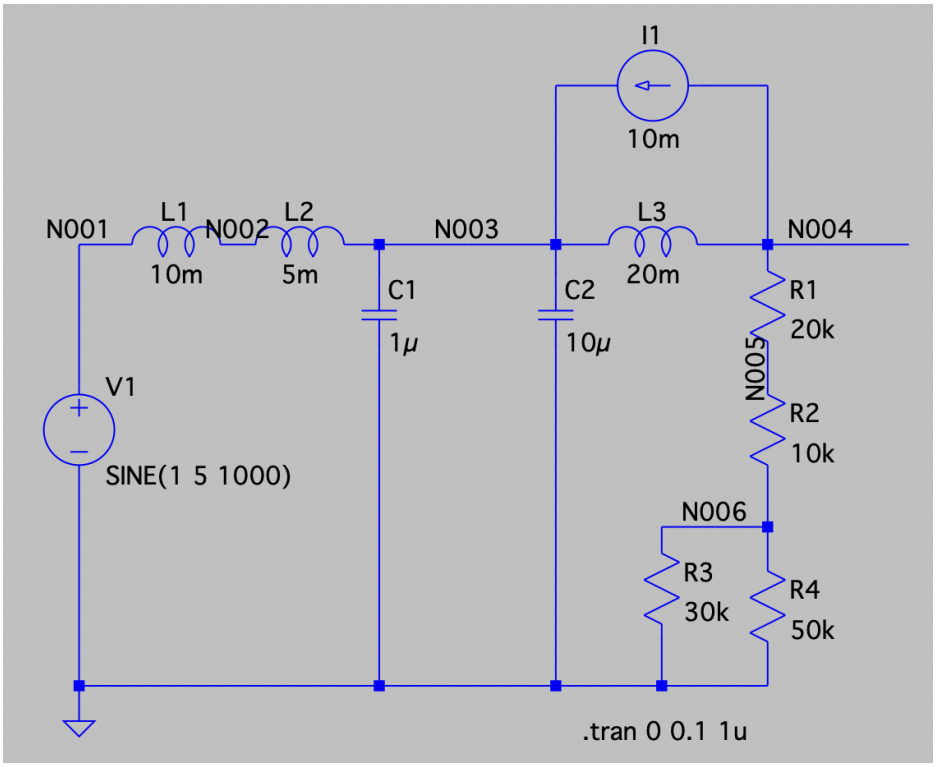


Figure 28 Test 7 Schematic

```
.tran 0 10m 1u
```

Purpose

The circuit tests for the program's resilience to unstable outputs due to reactive components, more specifically series inductors and parallel capacitors.

Operation Efficiency Benchmarks

Performance Time	`real 0m8.736s`
	`user 0m3.166s`
	`sys 0m0.376s`

Energy Consumption	$Energy = 28W \times 3.166s \times 12.4\% = 11.0J$

Results: Graphs for node voltages and component currents

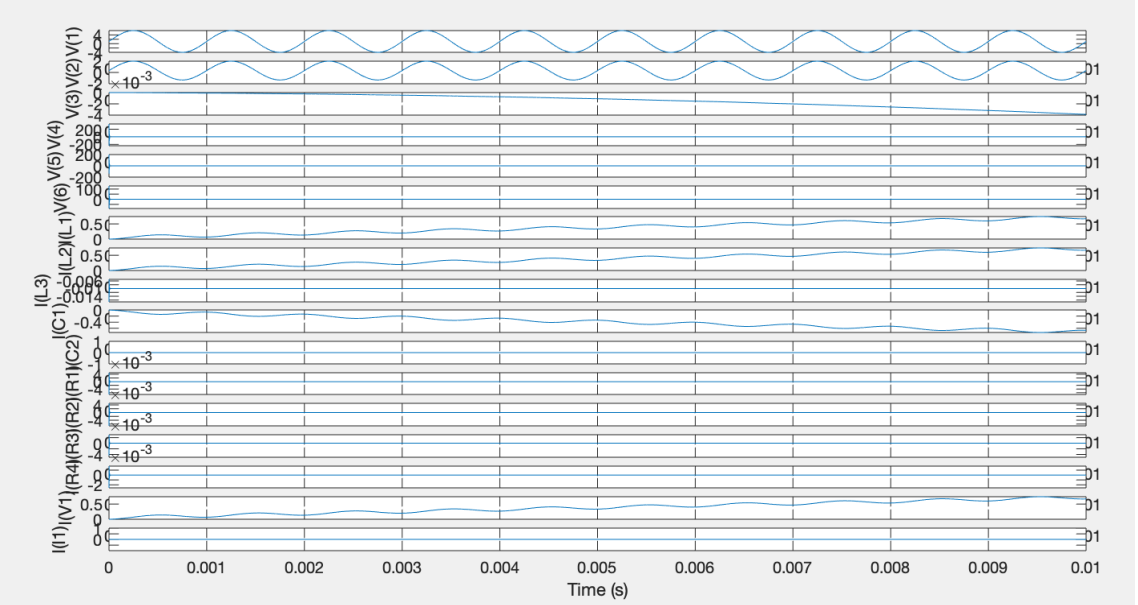


Figure 29 Graph of results for test 7

Evaluation

As expected, the simulation did not work. This can be attributed to the modelling of the reactive components in this circuit simulator:

- An inductor is modelled as a current source, but the shortcoming is that current sources cannot be placed in a series configuration. In this scenario, it is likely that the change in current through the inductor L1 is not immediately detectable in L2, causing the value at node 2 to be incorrect. This might have been solved if the two inductors were combined into an equivalent inductor, resulting in the inductance L1+L2.
- A capacitor is modelled as a voltage source, but by the same token, voltage sources cannot be placed in parallel with each other. This might have been solved if the 2 capacitors were combined into one single capacitor, with value $(\frac{1}{C1} + \frac{1}{C2})^{-1}$.

5.1.9. Test 8 - Double diodes

Circuit Schematic and Simulation Directive

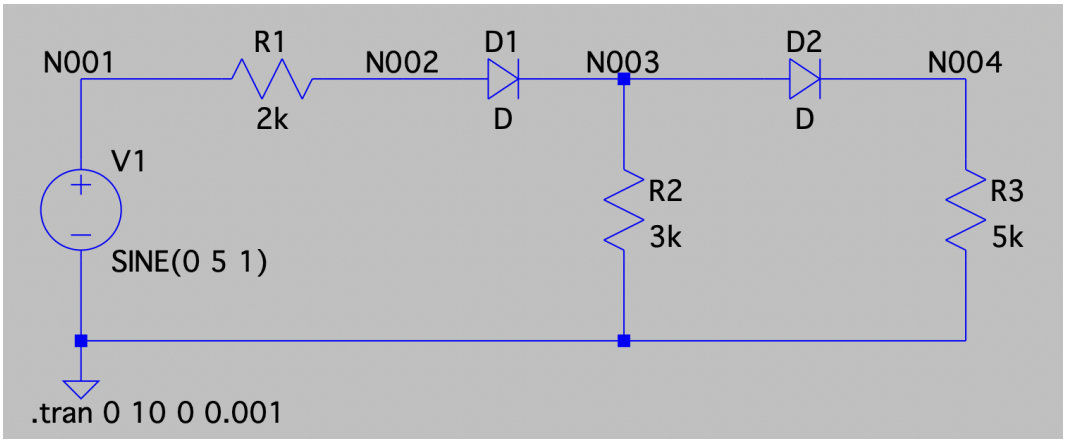


Figure 30 Test 8 Schematic

```
.tran 0 10 0 0.001
```

Purpose

The circuit tests for diodes placed consecutively in a series configuration.

Operation Efficiency Benchmarks

Performance Time	`real 0m16.151s`
	`user 0m5.717s`
	`sys 0m0.813s`

Energy Consumption	$Energy = 28W \times 5.717s \times 16.3\% = 26.1J$

Results

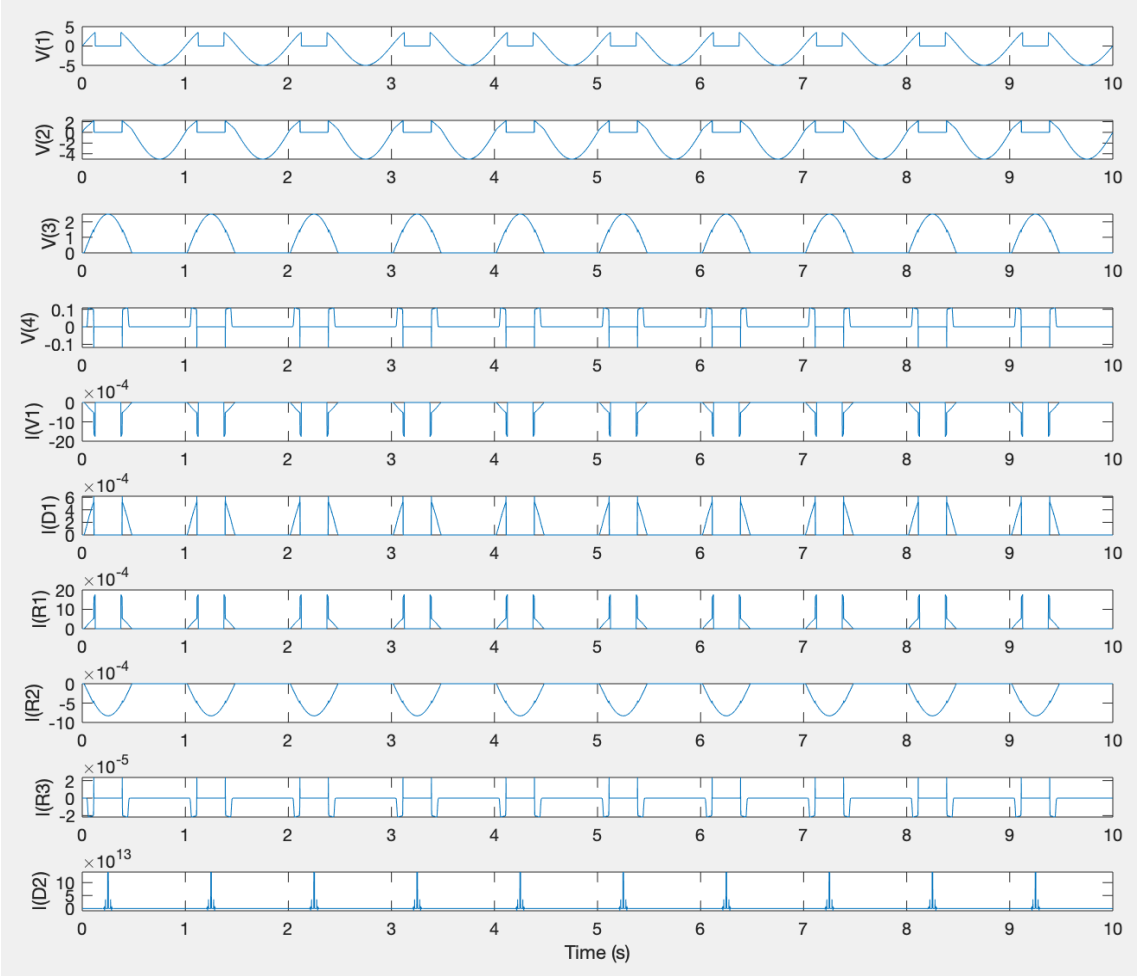


Figure 31 Graph of results for test 8

Evaluation

While the simulator was able to perform simulations for circuits with one diode, the situation completely changes when diodes are placed consecutively in series. Without examining the actual mathematics behind the operation of series diodes, it is apparent that the characteristic equation is altered, which changes the initial condition for the newton raphson method.

By contrast, the analytical solution outlined in appendix 8.2.9 suggests that the diode is better treated as a single block with a 1.4V drop in potential. Nevertheless, obtaining the actual numerical solution through iteration is difficult, as the method would have to be applied recursively. It should be noted that given extra time, an appropriate recursive function could have been created that will fix all of the diode's known problems. However, this was not considered due to diodes being an extension and current time limitations with the project.

5.2. Comparison with projects on other platforms

Albeit the relatively reduced scope of this circuit simulator project, a comprehensive investigation must be carried out for other projects that perform the same or similar functions, so as to establish a robust and holistic evaluation on the program's achieved functional and non-functional requirements.

There are various electronic design automation (EDA) software available that do not rely on the SPICE solver, such as:

- Simscape Electrical, which offers much greater applications for hydraulics or mechatronic systems, as well as grid level electrical power systems [19],
- The Lcapy Python Package, which uses the Sympy library to perform linear circuit analysis for circuits configurations with a range of high and low level electronic components, and outputs numerical data or expressions in numerous formats [7,27],
- SAPWIN (Symbolic Analysis Program for Windows), a circuit simulator written in C++, which can generate Laplace domain equations for analog circuits, convenient for algebraically analysing time-variant circuits; and,
- Various other platforms which take advantage of both MATLAB and SPICE for different stages of the circuit analysis process.

Nonetheless, these are all platforms designed with purposes other than circuit analysis in mind. Furthermore, some of the named projects are either licensed or open-source, so accordingly they are developed with a substantial set of functionality. Consequently, this section places an emphasis on comparing with other projects of a similar scale and with like-minded aims.

Since the core SPICE solver has been explored in the introduction, two other circuit analysis platforms are reviewed, namely SCAM (Symbolic Circuit Analysis in MATLAB) and CircuitNAV.

5.2.1. SCAM

SCAM is a circuit analysis tool developed by Erik Cheever at Swarthmore College [3]. It takes in a similarly reduced SPICE netlist, and outputs its contents directly to a MATLAB object. The pros and cons of this tool, compared to that of the specified functional and non-requirements of this project, are as follows:

	Functional Requirements	Non-functional Requirements
Pros	- Supports circuit configurations with ideal operational amplifiers, - Can solve DC circuits and generate transfer functions with relevant step response and bode plots; and, - Can perform analysis for circuits with either numeric or symbolic components, which is beneficial for analysing abstract circuits.	- Input and output interfaces are particularly integrated with MATLAB, which enhances usability for current MATLAB users; and, - Results, such as the transfer function, can be directly converted to a MATLAB object and used for further calculations.
Cons	- Does not support diodes, - Does not support voltage or current sources defined by functions; and, - Does not perform transient simulations	- MATLAB, albeit its versatility, is a paid program and it has its proprietary programming language, which makes this less accessible for non academic users; and, - The program requires a large dependency called Symbolic Math Toolbox, though it contains valuable utilities such as graph plotters and equation solvers.

Table 5 Pros and cons of SCAM

5.2.2. CircuitNAV

On the other hand, CircuitNAV is a relatively simple online tool that provides only algebraic solutions for specific nodes or branch currents of a circuit [26]. The pros and cons of this tool, compared to that of the specified functional and non-requirements of this project, are as follows:

	Functional Requirements	Non-functional Requirements
Pros	- Provides algebraic or symbolic representations of circuit parameters - node voltages and branch currents; and, - Supports dependent sources.	- Exceptionally accessible and reliable online interface, - Input interface entirely compatible with SPICE netlist; and, - Execution time and memory footprint is considerably low.
Cons	- Does not support nonlinear components, - Does not offer numerical solutions; and, - Does not perform any simulations.	- Output format is an image of equation, which is only convenient for studying purposes; and, - Output interface does not allow parsing into other programs.

Table 6 Pros and cons of CircuitNAV

The 2 tables above provide comprehensive comparisons for the functional and non-functional requirements between the Circuit Simulator Project and SCAM or CircuitNAV. In summary, all three products, while developed with similar purposes in mind, support a different subset of functionality; from a bigger picture, the SPICE solver remains the most versatile platform for circuit analysis. Ultimately, the goal of this project is to produce a software package that performs transient simulations for circuits with linear and non-linear components, and indeed the specified functional and non-functional requirements have been fulfilled.

5.3. Evaluation on Project Management

Everyone has fulfilled their self-appointed roles perfectly, which has helped us to finish this project smoothly because we were able to divide up the tasks to match each team member's personal strengths. Communication problems concerning time zones differences between each member were handled by leaving notes in the group chat regarding what they have done each day, and what issues they have encountered. This has helped other members catch up on progress that has been going on while they were asleep.

The agile project development cycle has helped with managing the project greatly, it kept everyone on track and allowed for regular evaluations of the progress made, and concerns each team member had. This was found to be especially useful when multiple revisions of the code had to be made in order to integrate new features into the existing code efficiently. It also helped us with version control, so we can revert back to previous working versions should an error be found in the current revision.

While the project has been successful on the whole, there were several things we could have done better in regards to timing. We had unanimously decided as a group to postpone the start of the project to work on our lab orals, this meant that we had 2 weeks less to finish our project. This resulted in a few intended features not being in the final product, such as, finishing the efficiency improvements for the improved version of the code, adding in BJTs, and doing an in-depth troubleshooting of the tests that failed to find out the reason so we can fix it properly.

6. Conclusion

6.1. Summary

The purpose of this project was to strengthen project management skills for an engineering environment, to study literature regarding computerised circuit analysis in order to create a simulator that can perform a subset of LTSpice functionality. Ultimately, the outcomes are indeed tremendously rewarding, in terms of both the actual deliverable achieved and the soft skills gained.

Throughout the course of this project, the algorithm of achieving the simulation has gradually evolved alongside the increasing complexity of circuits dealt with; key changes were made to accommodate supernodes, then reactive components, and finally diodes. Considerations were also made to meet non-functional requirements such as efficiency and usability, to a lesser but still appreciable extent; for example, loops were modified to perform selective updates on entries of matrices, and considerations were made to prevent parsing faulty netlists.

On the other hand, the project was also a perfect environment to develop engineering practices; at the current stage, this involves acquiring skills such as team-building, communicating engineering designs, professional skills, or project management in general. The current outcomes would not have been achieved if it were not the resilience the team had towards upholding principles such as frequent communication and collaboration, or adhering to the project development cycle.

6.2. Areas for future Development

Some of the functional and non-functional requirements of the project have yet to be accomplished, and it is undeniable that computerised circuit analysis is a topic that has boundless opportunities for further development, both in terms of feature set and efficiency.

For instance, conducting a further examination on the Newton-Raphson method would eventually lead to the implementation of transistors. This would make analysis for semiconductor devices possible, which is important as the manufacturing of electronics increasingly relies on semiconductor devices such as bipolar junction transistors and MOSFETs. This would also pave the way towards the implementation of operational amplifiers, arguably one of the most important devices in today’s electronics industry.

In addition, the differential method can be adopted as well to solve circuits with small inductances and capacitances, alongside the present integral method for relatively larger capacitances. This combination would enable a greater degree of flexibility when it comes to choosing the values of reactive components and the timestep for the simulation.

7. References

[1] M. Alexander, “Agile project management: 12 key principles, 4 big hurdles,” CIO, 19-Jun-2018. [Online]. Available: https://www.cio.com/article/3156998/agile-project-management-a-beginners-guide.html. [Accessed: 14-Jun-2020].

[2] E. Cheever, “An Algorithmic Approach to Modified Nodal Analysis,” *Analysis of Resistive Circuits*. [Online]. Available: https://www.swarthmore.edu/NatSci/echeeve1/Ref/mna/MNA1.html. [Accessed: 12-Jun-2020].

[3] E. Cheever, “SCAM- A tool for symbolically solving circuit equations,” *MATLAB Central File Exchange*, 21-Nov-2003. [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/3443-scam-a-tool-for-symbolically-solving-circuit-equations. [Accessed: 12-Jun-2020].

[4] E. W. Cheney and D. R. Kincaid, Numerical Mathematics and Computing, 7th ed. Cengage Learning, 2012.

[5] A. J. Davis, “Diode Models,” *ELE 338*, Dec. 2009.

[6] “Eigen::FullPivLU< _MatrixType > Class Template Reference,” Eigen 3.3.7. [Online]. Available: https://eigen.tuxfamily.org/dox/classEigen_1_1FullPivLU.html. [Accessed: 14-Jun-2020].

[7] M. Hayes, “Overview,” *Overview - Lcapy 0.52 documentation*. [Online]. Available: http://lcapy.elec.canterbury.ac.nz/overview.html#introduction. [Accessed: 12-Jun-2020].

[8] J. C. Helm, “Software Requirements Specification .” [Online]. Available: https://sceweb.uhcl.edu/helm/RationalUnifiedProcess/webtmpl/templates/req/rup_srs.htm. [Accessed: 13-Jun-2020].

[9] “IEEE REFERENCE GUIDE,” *IEEE Author Centre*, 2018. [Online]. Available: https://ieeeauthorcenter.ieee.org/wp-content/uploads/IEEE-Reference-Guide.pdf. [Accessed: 13-Jun-2020].

[10] S. Jahn, “Non-linear DC Analysis,” *sourceforge.net*, 30-Dec-2007. [Online]. Available: http://qucs.sourceforge.net/tech/node16.html. [Accessed: 12-Jun-2020].

[11] V. Jiménez, “Solving circuits with Python,” *R6500*, 09-Apr-2018. [Online]. Available: http://r6500.blogspot.com/2018/04/solving-circuits-with-python.html. [Accessed: 12-Jun-2020].

[12] Keysight Technologies, “Simulation Time Steps,” *Simulation Time Steps*. [Online]. Available: http://literature.cdn.keysight.com/litweb/pdf/genesys200801/sim/cayenne/simulation_time_steps.htm. [Accessed: 12-Jun-2020].

[13] K. A. Kuhn, “Using Newton’s Method to Solve the Classic Diode Problem,” Jul. 2001.

[14] W. J. McCalla, Fundamentals of Computer-aided Circuit Simulation, 1st ed. Kluwer Academic Publishers Group, 1988.

[15] P. D. Mitcheson, “SPICE algorithms and internals ,” *EE2.3 Semiconductor Modelling in SPICE* . [Online]. Available: http://www3.imperial.ac.uk/pls/portallive/docs/1/7292571.PDF. [Accessed: 13-Jun-2020].

[16] “Modern trends in circuit analysis and design,” *EE 321 Advanced Circuit Theory 2004: A systems approach to circuits, measurements and control*, 2004. [Online]. Available: https://uom.lk/sites/default/files/elect/files/ee321s3.3.pdf. [Accessed: 13-Jun-2020].

[17] *Non-Linear DC Analysis*. [Online]. Available: http://www.ecircuitcenter.com/SpiceTopics/Non-Linear Analysis/Non-Linear Analysis.htm. [Accessed: 13-Jun-2020].

[18] G. A. Pavliotis, “EE1 Mathematics Numerical Methods,” *Imperial College London*. [Online]. Available: http://wwwf.imperial.ac.uk/~pavl/numerical_methods.pdf. [Accessed: 13-Jun-2020].

[19] “Simscape Electrical,” *MATLAB & Simulink*. [Online]. Available: https://www.mathworks.com/products/simscape-electrical.html. [Accessed: 12-Jun-2020].

[20] Stackify, “What Is SDLC? Understand the Software Development Life Cycle,” *Stackify*, 08-Apr-2020. [Online]. Available: https://stackify.com/what-is-sdlc/. [Accessed: 12-Jun-2020].

[21] E. Stott, “question@168 Inductor in Circuit Simulator.” .

[22] E. Stott, “question@58 Circuit simulator project: dealing with capacitor and inductor.” .

[23] “The Newton-Raphson method,” 2004. [Online]. Available: https://matel.p.lodz.pl/wee/i12zet/The Newton-Raphson method.pdf. [Accessed: 13-Jun-2020].

[24] “The Nine Belbin Team Roles,” Belbin. [Online]. Available: https://www.belbin.com/about/belbin-team-roles/. [Accessed: 14-Jun-2020].

[25] C. Warwick, “Everything you always wanted to know about SPICE ** But were afraid to ask.,” The EMC Journal, no. 82, pp. 27–29, May 2009.

[26] “Welcome to CircuitNAV,” *CircuitNAV*, 2019. [Online]. Available: https://circuitnav.pythonanywhere.com/. [Accessed: 12-Jun-2020].

[27] “Welcome to SymPy's documentation!,” *SymPy 1.6 documentation*, 23-May-2020. [Online]. Available: https://docs.sympy.org/latest/index.html. [Accessed: 12-Jun-2020].

[28] “What is a Gantt Chart?,” Gantt.com. [Online]. Available: https://www.gantt.com/. [Accessed: 14-Jun-2020].

[29] “What is Class Diagram,” Visual Paradigm. [Online]. Available: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-class-diagram/. [Accessed: 14-Jun-2020].

8. Appendices

8.1. Code

This section

8.1.1. Matlab plotsim.m File

This is the file for plotting simulation on MATLAB, distributed alongside the project specifications.

```
close all;

%File is a tab-delimited table of simulation results
%Row 1 contains headers
%Column 1 contains time values
%Generate an example in LTspice by running a simulation
% and choosing File -> Export data as text

%Input file name
datafile = "simdata.txt";

%Get simulation data.
simfile = importdata(datafile,'\t',1);
simdata = simfile.data;

%Loop over columns of data
figure;
for n = 2:(size(simdata,2))
      subplot(size(simdata,2),1,n);   %Add a set of axes
      plot(simdata(:,1),simdata(:,n)); %Add the data
      ylabel(simfile.colheaders(n));  %Label the Y axis with the column
header
end
xlabel('Time (s)'); %Label the x axis of the bottom plot
```

8.2. Test Reference

The following section provides the results for performing the same tests on LTSpice. Similarly, operating basis point solutions are provided for circuits with only DC sources, and graphs are provided for circuits with AC sources.

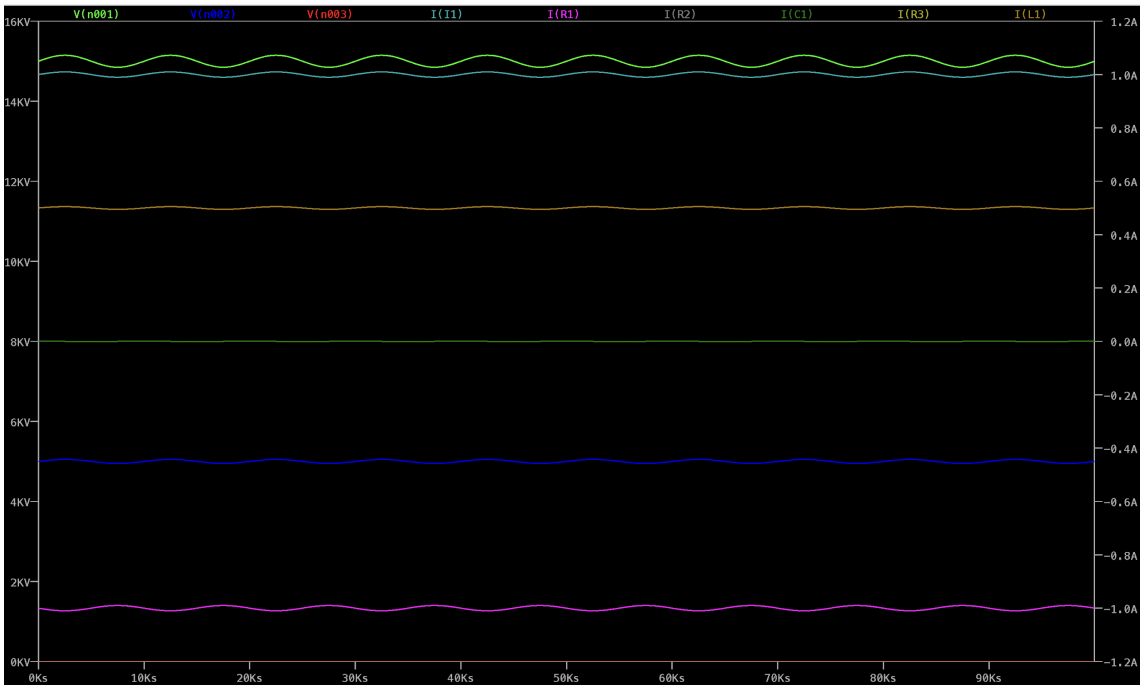
8.2.1. Test 1

```
Direct Newton iteration for .op point succeeded.
Operating Bias Point Solution:
V(n005)           37.7476   voltage
V(n001)           47.7476   voltage
V(n002)           52.7476   voltage
V(n006)                 3   voltage
V(n003)           19.5825   voltage
V(n004)                10   voltage
I(I1)             0.01   device_current
I(R5)       -0.00427476   device_current
I(R4)       -0.00142136   device_current
I(R3)       -0.000829126   device_current
I(R2)       -0.000829126   device_current
I(R1)       -0.00347476   device_current
I(V4)       0.00427476   device_current
I(V3)             0.01   device_current
I(V2)       -0.00652524   device_current
I(V1)       -0.00652524   device_current
```

8.2.2. Test 2

```
Direct Newton iteration for .op point succeeded.
Operating Bias Point Solution:
V(n001)                150   voltage
V(n002)                50   voltage
V(n003)           5e-06   voltage
I(C1)             2.5e-16   device_current
I(L1)             0.005   device_current
I(I1)             0.01   device_current
I(R3)             0.005   device_current
I(R2)             0.005   device_current
I(R1)             -0.01   device_current
```

8.2.3. Test 3

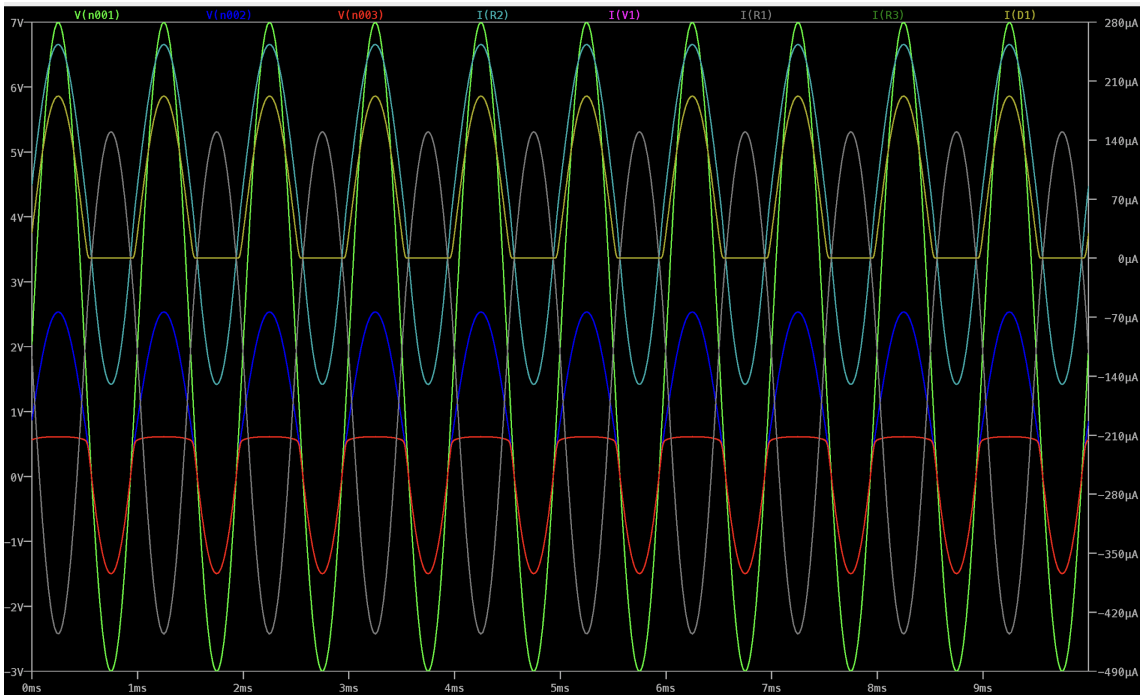


8.2.4. Test 4

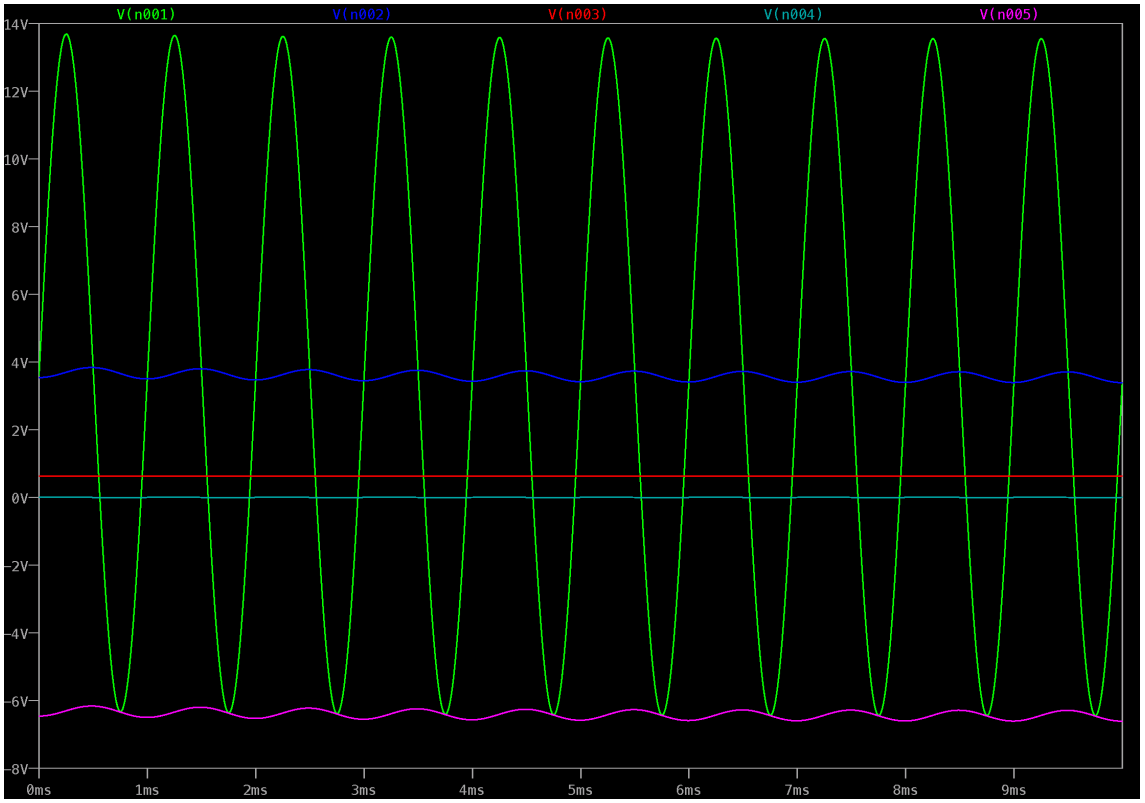
```
Direct Newton iteration for .op point succeeded.
Semiconductor Device Operating Points:
                        --- Diodes ---
Name:       d1
Model:      d
Id:   4.97e-03
Vd:   6.97e-01
Req:  5.21e+00
CAP:  0.00e+00


Operating Bias Point Solution:
V(n001)           150.348   voltage
V(n002)           50.3483   voltage
V(n003)           0.696567   voltage
I(D1)       0.00496538   device_current
I(I1)             0.01   device_current
I(R3)       0.00496517   device_current
I(R2)       0.00503483   device_current
I(R1)             -0.01   device_current
```

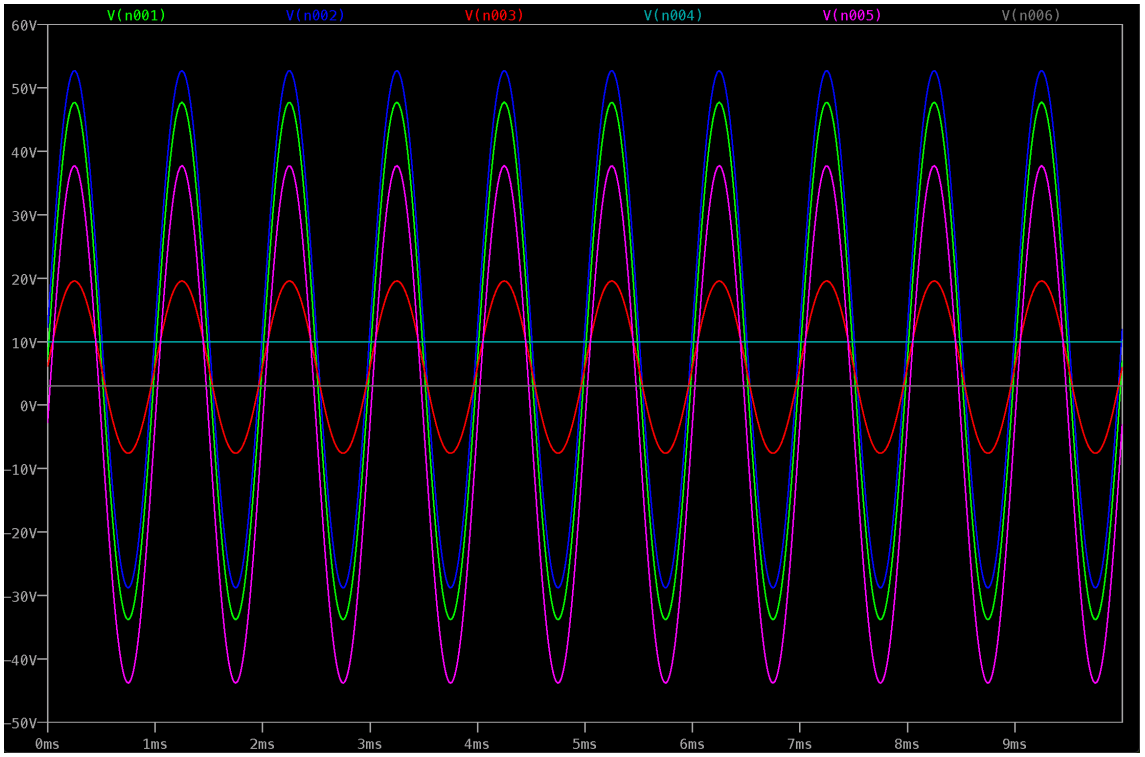
8.2.5. Test 5



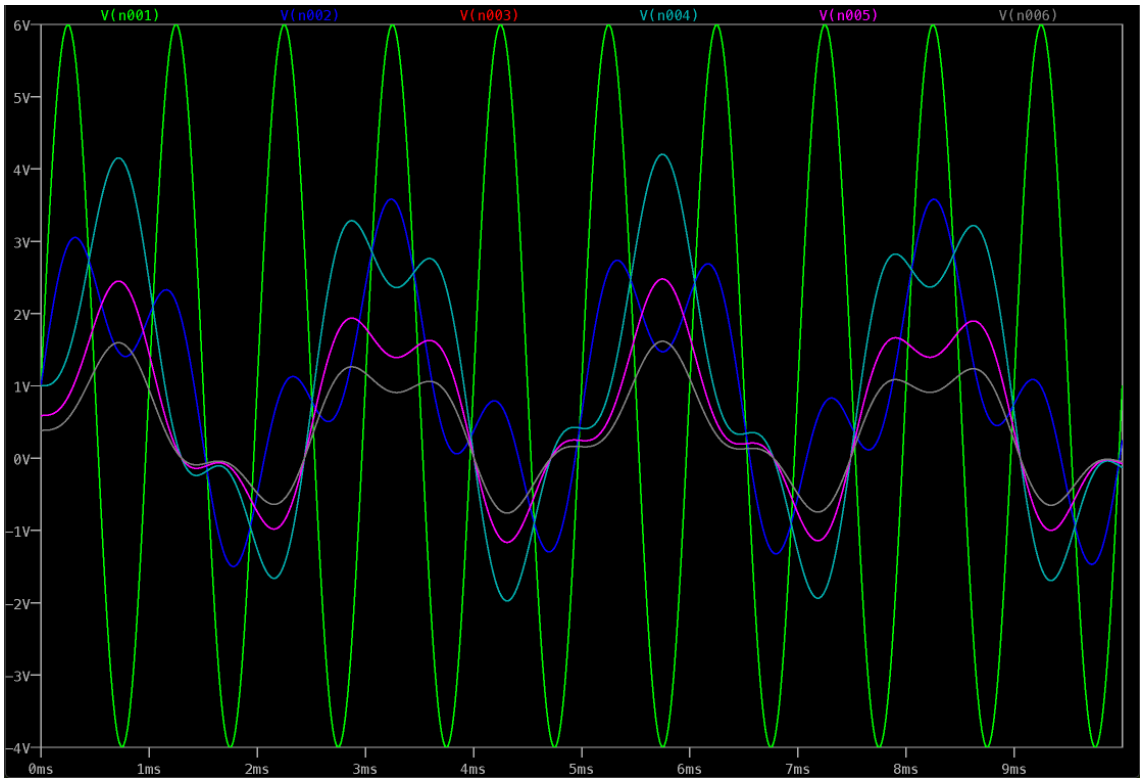
8.2.6. Test 6



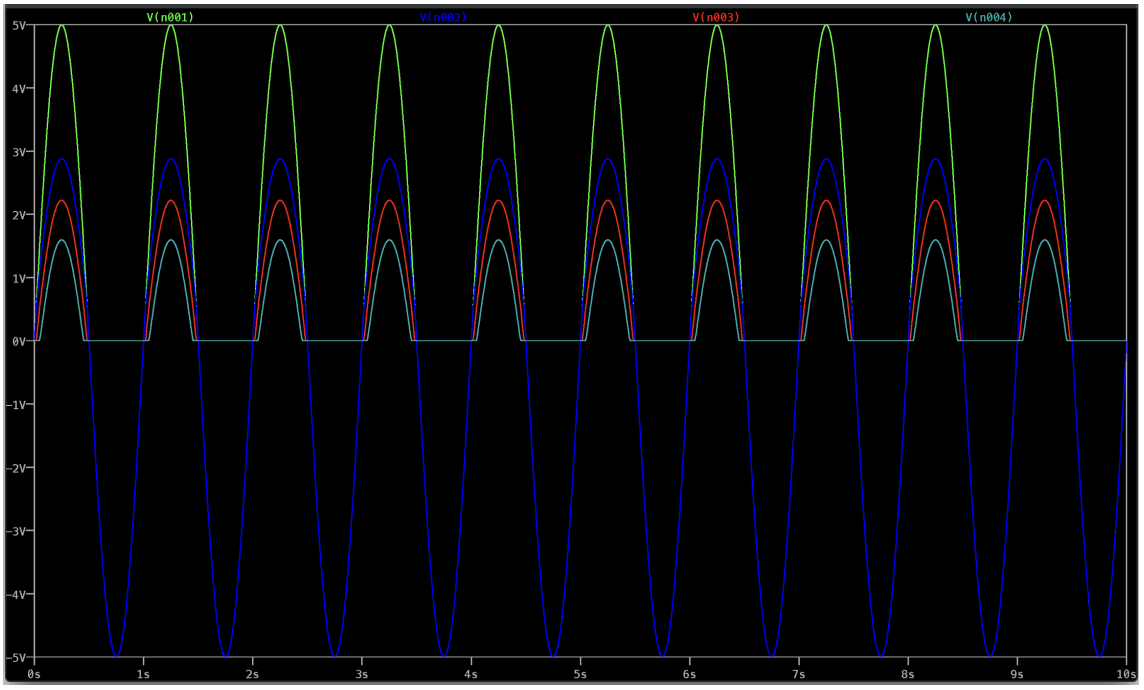
8.2.7. Test 1 Modified



8.2.8. Test 7



8.2.9. Test 8


V(n001)
V(n002)
V(n003)
V(n004)
5V
4V
3V
2V
1V
0V
-1V
-2V
-3V
-4V
-5V
0s
1s
2s
3s
4s
5s
6s
7s
8s
9s
10s

8.3. Complete Project Log

This section provides the log written over the course of the entire project. Note that certain texts have been reformatted for consistency.

21st May:

- Came up with a basic to-do list of what exactly needs to be completed systematically in order to be able to create the program
- Started to understand what the specification wanted from us

23rd May:

- Made a basic class for defining LCR components as well as voltage/current sources. The classes will store all the relevant component to be used later by other parts of the program
- Made a factory function which takes in the LTSpice netlist and turns it into a vector of these component variables
- Realized the best plan of action is to create a basic program which works for easy simulations for just resistors and current sources and then slowly add more functionality
- Deciding to discuss and implement our ideas as rather than brainstorming the best storage, we feel that completing a basic format will allow us to see what needs improvement in terms of efficiency and then we will be able to directly tackle that.
- We discovered when looking through a vector of nodes, the find function doesn't look at the last element in the vector and thus, when we input the node of a circuit into a vector, we can't check all the elements in the vector to see if they are the same. As such, we decided to move onto for loops.
- Tested whether our netlist reader is capable of taking the basic component variables (resistors and current sources) and able to put them in matrix form and created a method for finding values inside of the conductance matrix for resistors.

24th May:

- Made a basic class structure for nonlinear components that supports both 2 and 3 node components. This was done by having 2 constructors, the 3rd node value will be left blank. A separate boolean variable was added into the class so we can tell whether 2 or 3 nodes are being used
- Made a basic framework for newton-raphson approximation in the new nonlinear class by adding variables that stores the previous iteration's value for use in the next loop

- Added newton-raphson method for diodes via the shockley ideal diode equation. Starting values for an ideal diode are unknown so it is left blank and will be filled in later when more research has been done
- Added a matrix library to the program and implemented the method for finding the conductance matrix with the new matrix library we have included within the main function More research is required on the new matrix library, Eigen, in order to do that.
- Realized that in order to calculate current matrix for future components, we will need to also define components in terms of their currents in their circuits. Currently our classes do not support this, so we wanted to check all of our classes to see what is really necessary and what can be improved upon.
- Created a node class which tells us which nodes are connected to which other nodes and added variables to store information on components/nodes connected to a single node into the node class
- In our component base class, we also added voltage and current values which we can apply to components to get their direct current/voltage values to input into the matrix.
- We realized that our current conductance matrix will have to undergo a lot of change in order to incorporate voltage sources since these cause the matrix to no longer be symmetric. As a result, we came to the consensus that it is best to code how to create a matrix using resistors, voltage sources and current sources initially. (This means that we will have to re-write our code for producing the conductance matrix)
- We also realized after some discussion that our understanding for using conductance matrices was different from one another. This is due to the sources we visited using different methods in order to calculate nodal voltages using matrices compared to the specification we were given. This made us realize the importance of communication and looking at the specification.

25th May:

- We realized that several new methods were needed after adding the new variables added a missing function to return the current value across a component
- Realized that the previous method of implementing the current matrix generation code was overly complicated. Redid the entire code and reduced it to around 30% of its original size.
- Redid the classes so that they now reference each other via pointers instead of a location in the vector. This will make it easier and more efficient to traverse the circuit to look at nearby components/nodes without the need for big loops and recursion.

- Redid the netlist parser to support the new changes made to the classes by creating pointers to components instead, and storing the pointers into vectors separated into groups (components and nodes).
- Made current matrix generation function take in the pointer instead so it uses less RAM when called. This will be very useful when dealing with larger circuits.

27th May:

- Now that both the vectors of nodes and components are all pointers, a lot of the previous code needs to be changed to incorporate this using dynamic memory. The benefit of this is that any functions we run will be extremely cheap in terms of copying because we will no longer be copying the vector. This means that there is effectively a single copy of every node and component which is linked to other nodes and components through pointers.
- A major issue that popped up as a result of this was to do with the change from variables to pointers. This caused a lot of the function calls inside of the class to change from nodes.return(node) to nodes->return(node).
- The final major issue was with regards to printing information because there were a couple problems where the code would print out an address instead of a tangible value. At this point, we have yet to work on fixing the conductance matrix to incorporate these new classes.

28th May:

- Realized that our node vector wasn't storing the components connected to each node properly, so we quickly fixed our node vector insertion function in order to take this into account.
- We worked on fixing our conductance matrix calculations for only resistors to incorporate the new classes which we are using, namely the new node and component class. Since the node vector contains a vector of all the connected components to that node, we can actually simplify our previous conductance matrix code so that it only requires one for loop to input all the values rather than two for loops for both the diagonal and the upper triangular sections of the matrix.
- An issue came up where the netlist parser wasn't adding connected components and connected nodes to the nodes inside of the node vector. It was actually adding them to the new instance of node that was created whenever we read in a new component, which meant that all the data wasn't getting stored on the nodes inside of our vector, preventing

us from completing the calculation. This problem would have caused a lot of wasted memory in a larger file.

29th May

- We came to the realization that the specification actually told us not to include the ground node in the matrices. As a result, we had to fix this and we did so by making the for loop for the conductance matrix continue everytime it saw the ground node.
- A problem with this is that there is now one less node in the matrix, this had to be fixed by adding for the node's ID to make sure it wasn't ground.
- There were two minor mistakes when implementing this, firstly, when re-writing the code, the other_conductances section was accidentally reverted back to g(node_id_1, node_id_2) = other_conductances. This means that we took away a minus and a +=, effectively robbing our program of its compatibility with parallel resistors.
- Secondly, the program which checks and confirms what node 2 is by comparing with node 1, wasn't doing it correctly and as such sometimes node 2 took the value of node 1. Once these two were fixed, we completed our conductance code.
- After this we decided to check our eigen library, conductance matrix and current matrix through creating some code that would calculate the voltage matrix for simple circuits with just resistors and current sources in DC, as this is all we have programmed in terms of cases.
- Fixed the problem with the current matrix where all the values were swapped. The issue was that the LTSpice notation for current sources (in, out) were opposite the voltage sources equations (out, in). This was easily fixed by changing the location of the node array the program was looking at for current sources
- Added support for scientific notation (eg. p, m, k, Mega) using a series of if-else statements.
- Ran checks with different circuits to make sure that the current code is working properly
- Separated functions in the main file to their own hpp files so that we can work on each function simultaneously without creating merge conflicts

30 May

- Added support for transient simulations in a different cpp file to avoid merge conflicts between people working on the same file. The code will have to be manually merged later

on once both parts are done. This will allow simulations with sine wave sources once the netlist parser has been updated to support them

- Added support for sine wave sources in the separate code mentioned above. This will allow the main code to run transient simulations once combined back with the original code.
- Added labels for voltage source pos and voltage source neg and voltage source ground in the node class for easy references when dealing with supernodes
    - V source pos is true if the node is at the positive terminal of a voltage source
    - V source neg is true if the node is at the negative terminal of a voltage source
    - V source ground is true if the voltage source to which the node is connected to is grounded.
    - Added relevant methods for setting these booleans for individual nodes and returning these booleans

31 May

- Improved transient code made previously in a different cpp file and merged it into the main file. Changes were also made to most of the previous functions to allow transient analysis. This has only been tested with IR, but it should support voltage sources once they are added into the main file.
- Fixed an for infinite loop which occurs when not doing a transient simulation due to the timestep being 0 by default

2 June

- Added a recursive function to check for supernodes in the current matrix. This will allow us to properly find the total voltage easily for larger supernodes
- Found a problem when changing values in the node objects within the parser. The value change has to be done after adding all the nodes to the vector of nodes to prevent problems occurring from missing data due to the way we handled pointers and creations of new objects.
- Came to the realisation that the booleans (v_source_pos, v_source_neg and v_source_ground) are useful for efficiency, but useless when two voltage sources with opposite terminals are connected to the same node. These variables and all their associated functions were removed from the code.

3 June

- Redid the conductance matrix and current matrix generation for a new method that allows for supernodes. We did this because super nodes are easier to deal with alongside voltage sources. This also means that the supernode recursive function from before is now removed from the code.
- Voltage sources simply require their own entry (row and column) inside of the matrix and this allows for multiple voltage sources to be easily added to the matrix. The only downside to this is that the matrix ends up being larger than previously, however the logic is immensely reduced and super nodes are easily dealt with in this model through the voltage source entries.

4 June

- Added capacitor and inductor support to the current matrix since we decided based on a piazza post and the numerical integration method in the specification that we would treat capacitors as a voltage source and inductors as a current source. We will use the results from the voltage matrix to slowly change the outputs of the sources each timestep. This will allow them to store the previous current/voltages across them like real inductors and capacitors
- After some testing we found that inductors/capacitors work with DC source transient analysis, but not AC. We couldn't properly see how exactly the capacitors/inductors were behaving during AC analysis since these are components which don't act linearly towards currents/voltages like resistors do and this made it hard to track them properly while looking at the actual values in a table.
- We decided to reconfigure our output of matrices to matlab output in order to see the output waves through the matlab simulator provided. We made it so that you input the node you want to see initially and you get the matlab format for that node's voltage output.
- We tested our circuit simulation on some simple RL and RC circuits with 1 resistor and 1 capacitor/inductor. (Although during this time we thought that as such our program was working correctly, we eventually found out this was not the case.)

6th June

- We came to a decision as a group and decided to implement diodes since we have the free time available: as such we have started researching various methods to implement diodes. The only information we have available currently is the fact that we should try to do it iteratively using the newton-raphson method since you can't directly analyze a diodes voltage/current since it involves a transcendental equation inside of the matrix.

- We decided to stick with the shockley equation which describes a diode's current in terms of its voltage and iteratively use the newton-raphson method in order to eventually come across a good estimate for the voltage/current of the diode.

8th June

- We found out that it can be difficult to model the diode using the newton-raphson method along with Shockley's equation, monitoring each of the currents and then iteratively solving for the diode voltage/current. The major problem we discovered is that when a diode affects 2 nodes rather than just 1 because 1 is ground, we have to use gaussian elimination on the matrix initially to make sure the diode row of the matrix is in terms of 1 unknown.
- After researching the eigen library, we realized that it's generally unrealistic to be able to code in such a complex behaviour when we can't find the right function. Through some more research we decided to switch to a new model where the diode is modelled as a resistor in parallel with a current source. It also uses the shockley equation, but in a much more convenient way since we know how to model resistors/current sources, so it's easy for us to input into our matrices without any major issue.

9th June

- We decided to follow the companion model and as such some of the classes were changed: namely the nonlinear_component class, which now uses a single function which takes in Vd in order to calculate Id, Ieq and Req, which is all the information required for the matrix which recalculates Vd at a more accurate level.
- This function and new class which now directly points to a resistor and a current source will make it easier to simply call the function inside of the main code whenever we need the new diode values outputted almost immediately. This means that we don't need to expand the vectors storing all the components since it will be part of the diode.
- We also decided to split apart the component.hpp and component.cpp file into multiple .hpp files after realizing that the code and functions are hard to follow when they're split across a cpp and hpp file.
- On top of that we are no longer compiling the main code alongside multiple .cpp files as we feel that .hpp files are easier to work with as seen with the current_matrix.hpp and scientific_converter.hpp. As such each class got it's own .hpp file.

10th June

- We managed to finish coding in diodes using the new model. It is currently uncompiled and is supposed to loop over the conductance matrix multiple times in order to accurately find Vd and Id when there is a diode in the circuit. It then models the diode as a resistor and current source in the matrices in order to properly solve for Vd and Id.
- An issue was found where the newton-raphson loop will go on indefinitely because the process to stop it was mistakenly inserted outside of the loop itself.
- We improved our outputs so that it fits the specification, which is to make the program output the voltages at every node and the currents across every component. It also helps to see all the other voltages in order to bug fix and identify problems that may pop up during testing
- We discovered that although a couple of the previous circuits work, inductors aren't working at all anymore. This has led us to look through our previous code and try to identify what exactly has changed. Inductors only seem to be working in basic circuits now and not in any complex ones.
- We've decided to revert back to an older version of our code to compare the inductor and capacitor circuits to identify the differences and easily see exactly what has changed in the new version of our code. This was possible because we made commits after each change so we can easily revert back to any old version incase something broke
- After testing between the two versions we realize that the voltages are actually identical for all the circuits. This means that our old code is also not working for inductors. It seems that the testing we had previously done was too simplistic and didn't fully investigate whether inductors were working. This led us to believe that inductors were never implemented correctly.
- We have a feeling that the issue we have is in relation to the inductor equation itself. After some testing, we found out that the inductors still don't seem to be working. They are stuck at a voltage of 0, when they are supposed to be letting voltage through them.

11th June

- Now that we have found out that our inductor code was never working at all and our new code is correct, because it copies our old code, we need to find a fix for inductors and since our suspicion is the equation, we have decided to conduct research on the modelling itself.
- We did tests with various circuits to narrow down the scenarios in which the inductor didn't work. The main problems are when the inductors are connected to a component on both

sides. This suggested to us that the current output wasn't working properly on both sides of the component.

12 June

- We used a new diode model that was suggested by an instructor on Piazza. This was to give inductors their own row and have the old current across it as a value in the current matrix, to be used in calculating the next current value across the inductor.
- Running the complex circuits with this model proved to be successful, all the results matched what was expected.
- Now that we have time, we did the tests for diodes and it isn't working properly. The main issue seems to be the direction of current and it reaching -nan and -inf when inspecting the actual output files.
- The issue was fixed by changing all the data types concerning diodes from floats to double. The problem was that during the internal calculations for diodes, the values were either too large or small to be properly stored within a float, which then messed up all future calculations.

13 June

- Fixed issues in the nonlinear class related to diodes. The main issue was that the node orders were wrong when creating the component from the netlist format, due to the fact that a current goes into node1, and out of node2 as opposed to the voltage coming out of node1 for voltage sources.
- Ran tests for the existing code to make sure that everything works correctly. We found two main problems which were that multiple diodes in series will break the program, and capacitors and inductors in parallel won't output correct values. We suspect that this is because having several reactive components connected to the same nodes on both sides create an infinite feedback effect with our current algorithm that drives it up to infinity
- Cleaned up the github repo, and appropriately named all the files that will be submitted.
- Created a copy of the code, with the suffix "_improved" to fix some of the old redundant code caused by multiple people working on the same file. This is so that we can compare the time saved using the new algorithms
- Each component is now stored in their own vector in the new code so that functions that require select components will not have to loop through the entire list of components each timestep. This gave a noticeable time improvement.

14 June

- Added for invalid circuits (eg. short circuits). The code will now throw out an error message and exit the program with an error code.
- Added a system defined time step that overrides the user's timestep if the user's is too large to have a decent output. This is required due to the riemann sum method used for reactive components. If the user's timestep is smaller than the system's the user's will be used instead.
- Changed the initial values for diodes to match LTSpice's ideal diode model initial values so that our simulations are closer to their values. The new initial values mean that our simulations with diodes have values that are basically the same.
- Did some final checks on the code before submitting to make sure that everything is working properly

8.4. EE1 Project 2020 - Simulation File Specification

EE1 Project 2020 – Circuit Simulator File Format

Your circuit simulator should read an input file that defines the circuit and the simulation. The format of the input file is compatible with SPICE, but you only need to support a subset of the features of SPICE, which are described in this document.

Components

The netlist is described by lines defining the type, designator (name), nodes and value of each component

Designator letter	Component	Node order	Value
V	Voltage source	+, -	Volts or function
I	Current source	In, out	Amps or function
R	Resistor	N/A	Ohms
C	Capacitor	N/A	Farads
L	Inductor	N/A	Henries
D	Diode	Anode, Cathode	Model name
Q	Transistor	Collector, Base, Emitter	Model name

The format of each line is:

```
<designator> <node0> <node1> [<node 2] <value>
```

Designator

The designator field contains a letter and a number. The letter specifies the type of component (see table above) and the number is added to make a unique identifier for each component

Node list

A list of two or three node names occurs after the designator. Node names have the format `N123`. The component is connected to these nodes, in the order given in the table above. The reference (ground) node is named `0`

Value

The value is the component value in the units given in the table above. Numbers can be followed by a multiplier:

Multiplier	Value
p	$\times 10^{-12}$
n	$\times 10^{-9}$
u	$\times 10^{-6}$
m	$\times 10^{-3}$
k	$\times 10^{3}$
Meg	$\times 10^{6}$
G	$\times 10^{9}$

Voltage and current sources can have a DC value, or a function. The only function required is a sinusoid written in the form:

```
SINE(<dc offset> <amplitude> <frequency>)
```

DC offset and amplitude have units of Volts or Amps. Frequency has units of Hertz.

Diodes and transistors are given a model name instead of a value. If you support them, the following models should be defined internally in your software:

Model name	Type
D	Silicon diode
NPN	NPN BJT
PNP	PNP BJT

Simulation settings

The transient simulation is described with a line of the form:

```
.tran 0 <stop time> 0 <timestep>
```

Stop time is the duration of the simulation. Timestep is the resolution of the simulation. `0` fields are included to maintain compatibility with SPICE and do not need to be interpreted.

End of file

The file ends with the line:

```
.end
```

Comments

The input file may contain comments on lines beginning with *

These lines should be ignored.

Extensions to the specification

You can extend the file format if necessary to support any additional features you have added. Maintain compatibility with SPICE if possible.

Example

```
* A test circuit to demonstrate SPICE syntax
V1 N003 0 SINE(2 1 1000)
R1 N001 N003 1k
C1 N001 0 1µ
I1 0 N004 0.1
D1 N004 N002 D
L1 N002 N001 1m
R2 N002 N001 1Meg
Q1 N003 N001 0 NPN
.tran 0 10ms 0 1us
.end
```

You can generate your own examples by drawing a circuit in the LTspice software and choosing the option View → SPICE Netlist. LTspice includes some additional lines which aren't required for your simulator. You may wish to configure your simulator to ignore these lines so you can produce test circuits directly from LTspice.

8.5. List of Figures

This section provides a list of figures in the main body of this report.