ELEC40006: ELECTRONICS DESIGN PROJECT REPORT

Adam Horsler 01738592, Wuyang Zhou 01702964,

Timeo Schmidt 01707530

Dr Stott and Mrs Perea

Date of submission: 14/06/2020

Word Count: 9569

Imperial College

ELEC40006 - Electronics Design Project

Table of Contents

Abstract 5

1. Introduction 6

2. Technical Problem to be solved 6

   2.1 Overview 6

   2.2 Input file format 6

   2.3 Time-dependant components 8

   2.4 Types of nodes 8

   2.5 Capacitors and Inductors 9

   2.6 Non-linear components 9

   2.7 G, V and I matrices 9

   2.8 Output file format 9

3. Design Criteria 10

   3.1 Overview 10

   3.2 Assumptions and Dependencies 10

   3.3 Other Requirements 10

4. Design Process 11

   4.1 Overview 11

   4.2 Data Structure 12

   4.3 Parsing the input file 14

   4.4 Conductance Matrix Algorithm 16

      4.4.1 Algorithm 16

4.4.2 Linear Algebra Libraries 18

4.4.3 Eigen Implementation 20

4.5 Output File 21

5. Parsing the Netlist 22

6. Code for Building the G, V and I Matrices 24

6.1 Helper functions: 25

6.1.1 Is a node voltage known? 25

6.1.2 Are two nodes supernodes? 25

6.1.3 Separating supernodes 26

6.1.4 Indexing Nodes 27

6.1.5 Indexing components 27

6.1.6 Conductance Between Nodes 27

6.2 G, V and I matrix Construction 29

6.2.1 Overview 29

6.2.2 V Matrix 29

6.2.3 G Matrix 29

6.2.4 I Matrix 30

6.3 Capacitors and Inductors 31

6.3.1 Converting them to the equivalent sources 31

6.3.2 Updating the value of the equivalent sources 32

7. Current Through Components 33

7.1 Overview 33

7.2 Current through R: 33

7.3 Current through I source: 34

7.3.1 Current through V source: 34

7.3.2 Putting everything together....34
8. Testing....35
8.1 Overview....35
8.2 Simple Circuit Test....36
8.3 Supernode Test Case....37
8.4 Circuit with a Current Source....38
8.5 AC Circuit with Voltages and Current Sources....38
8.6 Circuit with a Capacitor....39
8.7 Circuit with an Inductor....39
8.8 Efficiency and Performance....39
9. Debugging Techniques....41
10. Project Planning....43
10.1 Overview....43
10.2 Meeting Structure....44
11. Conclusion....44
11.1 Overview....44
11.2 Reflection....45
11.3 Future work....46
12. References....49
13. Appendix....51
13.1 Appendix A: Software Requirements Specification....51
13.2 Appendix B: Gantt Chart....52
13.3 Appendix C: Notion....53
13.4 Appendix D: Replit....53
13.5 Appendix E: GitHub....53

13.6 Appendix F: Supernode Simulations 53

13.7 Appendix G: Current Source Simulations 54

13.8 Appendix H: Current and Voltage Sources 56

13.9 Appendix I: Capacitor Test Simulation 57

13.10 Appendix J: Inductor Test Simulation 59

13.11 Appendix J: Efficiency and Performance Data 61

Abstract

The Aim of the project was to create a software package that could perform transient simulations of circuits that contain linear components. These are resistors, capacitors and inductors. Optionally, non-linear components like diodes and transistors could be included.

The software package will create and systematically solve a system of linear equations. For circuits, it is often referred to as the conductance matrix algorithm. It involved constructing three matrices, a conductance matrix, voltage matrix and current matrix. The conductance and current matrix are populated using the information in the input file, while the voltage matrix is calculated by finding the inverse of the conductance matrix and multiplying it by the current matrix.

The algorithm and Object-Oriented Programming (OOP) approach to the problem proved to be an effective method for solving circuits analytically. This is because each component could be its own class, with the majority inheriting from a base class component. C++ libraries like Regular Expression and Eigen made pattern matching and linear algebra equations relatively simple and made the code easier to follow.

Our results showed that the software package successfully found the transient simulations of various circuits containing the aforementioned components. However, due to time constraints non-linear components were not implemented, although a potential implementation is described as possible future work.

1. Introduction

This product is based heavily on the functionality of the LTSpice software when solving circuits. It is designed to use a specifically formatted input file to calculate the transient simulation. The program will systematically solve systems of linear equations by writing them in matrix form and solving for the vector of unknowns. In this case, a conductance matrix algorithm is used to formulate and solve the circuit for the voltage at each node. After the voltage at each node is known, the current through each component is calculated. The product is created using the programming language C++, while the graphing of the results is done using MATLAB.

2. Technical Problem to be solved

2.1 Overview

The task was to create a software package that would take in an input file in a reduced LTSpice format of a circuit and produce a CSV file and MATLAB graph of the circuits' transient simulation.

The simulation can be set by two parameters, stop time and timestep. Stop time is the duration of the simulation and timestep is the resolution of the simulation. For example, a simulation with a stop time of 1s and a timestep of 1ms would need to calculate 1,000-time instances.

Given a circuit containing linear components (resistors, capacitors and inductors) as well as voltage and current sources, the voltage at each node needs to be found by solving $Vmatrix = Gmatrix^{-1} \cdot Imatrix$, where Gmatrix is the conductance matrix, Vmatrix is the voltage matrix and Imatrix is the current matrix. After knowing the voltage at every node, the current through each component needs to be calculated. Optionally, the product could support the non-linear components diodes and transistors.

2.2 Input file format

The specified input file defines the type, name, connected nodes and value of each component, as shown in Figure 1.

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

Figure 1

The format of each line is: *<designator> <node0> <node1> [<node 2] <value>.*

For example: “R1 N001 N003 1k” is a resistor named R1, connected between node 1 and node 3 with a resistance of 1k Ohms.

Voltage and current sources are represented in the form:

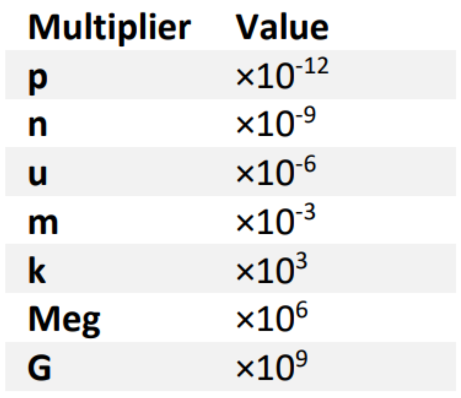
SINE(<dc offset> <amplitude> <frequency>)

For non-linear components, such as diodes and transistors, they are given a model name instead of a value. The product must then define their models internally. The model names used are shown in Figure 2.

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

Figure 2

Also, the product must support the value of a component given using the unit symbols shown in Figure 3.



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	$\times 10^{9}$

Figure 3

2.3 Time-dependant components

Some components have a value that changes over time with a predefined function. For example, a sinusoidal voltage source has a time varying voltage.

Some components have a value that depends on the previous history of the circuit. For example, a capacitor has a voltage proportional to the integral of the current through the capacitor: $Vcap(t) = \frac{1}{C}\int_{t0}^{t1} i(t)\,dt + V(t0)$. The program will therefore have to calculate the voltages and currents at each timestep throughout the simulation time to achieve the correct results. Thus, the program will need to execute fast enough to calculate each data point in time.

2.4 Types of nodes

A reference node is necessary in every circuit to ensure the correct voltages and currents are being calculated elsewhere. The reference node will have a voltage of 0. The values used in the conductance, voltage and current matrices change dependent on where the voltage sources are connected in relation to the reference node. If the voltage source has a terminal connected to the reference node, then the node connected to the other terminal can be expressed simply as something like $v1$ = $vsrc$, e.g. for a voltage source with Vsrc volts connected between node 1 and the reference node. If the voltage source does not have any terminal connected to the reference node, there are now two unknown voltages and a ‘supernode’ is formed. These scenarios must be considered to satisfy requirements.

2.5 Capacitors and Inductors

To correctly construct the systems of linear equations, inductors must be treated as current sources and capacitors as voltage sources. This is because an inductor’s current and the voltage across a capacitor cannot change instantly. Thus, the program must convert them into their respective source models and update their value at each timestep.

2.6 Non-linear components

Linear components will be solved analytically as their potential difference and current can be found exactly. However, non-linear components, such as diodes, have no analytical solution and so must be solved using an iterative Newton-Raphson method. Thus, the program must be able to solve circuits using analytical and/or iterative methods depending on the components present. It is therefore optional for the product to support non-linear components.

2.7 G, V and I matrices

G, V and I matrices are dependent on how the components are connected in a circuit. For example, G and I matrices change accordingly if there is a node with known voltage or supernodes. The I matrix changes if there are current sources connected to a node. After the matrices are correctly constructed, they should be solved to get the node voltages. Knowing the voltage at every node at a time instance, the current through each component needs to be calculated.

2.8 Output file format

The output file will show the voltage of each node and current through each component at each instance in time. The format requirements are that the file is written in Comma-Separated Value (CSV) format, where the columns are the nodes and components and the rows are the instances in time. A MATLAB script is then used to plot the results.

3. Design Criteria

3.1 Overview

Refer to Appendix A for the full design criteria as it contains the Software Requirements Specification (SRS) document. An SRS was used instead of a Product Design Specification (PDS) as the product is fully contained in software and thus an SRS is more appropriate.

3.2 Assumptions and Dependencies

- The system has only been tested on the gcc compiler, and on the operating systems Windows, Mac and Linux.
- The user will need to be running C++11 or above due to the inclusion of the Eigen library.
- The user needs to download the Eigen library.
- The input file is of the LTSpice format as described in Section 2.2. Otherwise, undefined behaviour will occur.

3.3 Other Requirements

- There are no additional hardware interface requirements.
- There are no addition communication interfaces required with the software.
- There are no additional safety or security requirements specified.
- The software package must pass various testing attributes for a variety of circuits. This is detailed in Section 8, and includes, but not limited to, circuits containing:
  - Single resistor and voltage source
  - Supernode
  - Current source
  - AC signal source
  - Capacitor
  - Inductor
  - If diodes are included, these will need be tested as well.

For linear components, the software package should produce an exact solution. For non-linear components, an iterative method is required with an accuracy of 3 significant figures.

4. Design Process

4.1 Overview

The overall flow chart below in Figure 4 illustrates the design of the system from input to output.

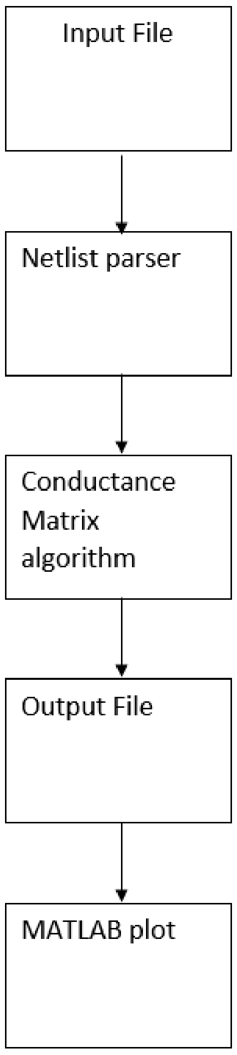


Figure 4

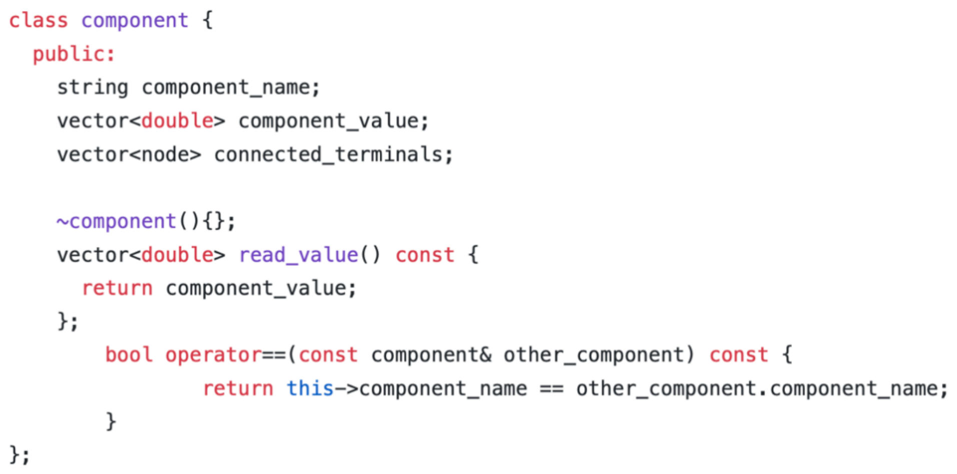
The software development platform GitHub was used to collaborate, with a GitHub repository created to track and share project code, the link of which can be found in Appendix B. GitHub was chosen as it keeps track of the various changes made to every iteration in the code. In addition, if a previous iteration is needed, it allows for quick and easy movement between iterations.

Since the product uses multiple libraries, the libraries are included in the file *dependencies.hpp*.

4.2 Data Structure

The first decision made was how to represent the components, nodes, and the overall circuit. An object-oriented programming (OOP) approach was agreed early on as this was deemed the most appropriate. The data structure is written in *simulator.hpp*. Each component, for example ‘resistor’ and ‘capacitor’, is its own class. These component classes then inherit from a base class ‘component’. As there is no physical generic ‘component’, the methods of the class are virtual. Separately, there would be a ‘node’ class and an overall ‘network_simulation’ class.

An OOP approach allows each component child class to hold its own variables and methods based on its type, e.g. a capacitor will hold its capacitance, as well as having common variables and methods associated with the base class component. Custom constructors are defined in these classes to assign information based off the netlist parser, which is described in Section 4.3 and Section 5.

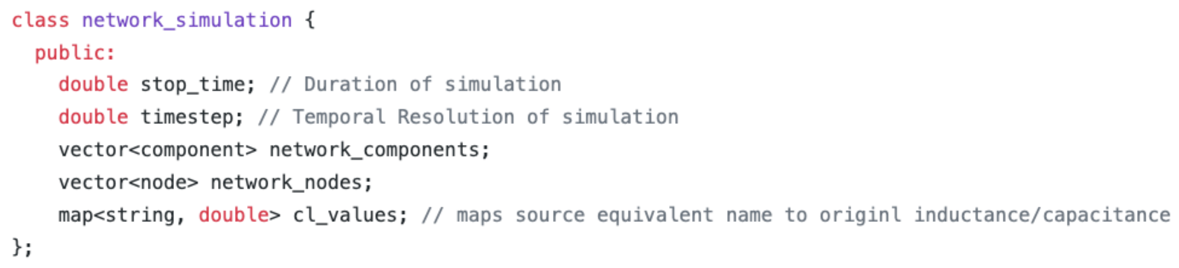


```
class component {
  public:
    string component_name;
    vector<double> component_value;
    vector<node> connected_terminals;

    ~component(){};
    vector<double> read_value() const {
      return component_value;
    };
        bool operator==(const component& other_component) const {
                return this->component_name == other_component.component_name;
        }
};
```

Figure 5

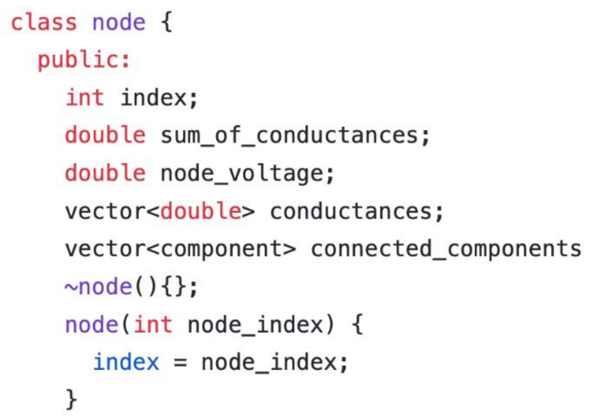
The network simulation class holds information about the whole circuit, for example a vector of all the nodes in a circuit and a vector of components in a circuit. In addition, the stop time and temporal resolution of the simulation is held here.



```
class network_simulation {
  public:
    double stop_time; // Duration of simulation
    double timestep; // Temporal Resolution of simulation
    vector<component> network_components;
    vector<node> network_nodes;
    map<string, double> cl_values; // maps source equivalent name to originl inductance/capacitance
};
```

Figure 6

The ‘node’ class holds the necessary information of each node in a circuit needed to insert values into the conductance matrix. For example, it holds a vector of the components connected to that node. Furthermore, the node_voltage member is updated after every time instance. Nodes are named by their indexes, for example N001 in LTSpice syntax is equal to Node 1.



```
class node {
  public:
    int index;
    double sum_of_conductances;
    double node_voltage;
    vector<double> conductances;
    vector<component> connected_components
    ~node(){};
    node(int node_index) {
      index = node_index;
    }
```

Figure 7

AC sources have three parameters: amplitude, frequency and DC off-set. A vector of doubles (*vector<double> component_value*) is used in the component class to represent the value of each component. For sources, the vector holds DC off-set, amplitude and frequency, in that order. For other components, the vector simply holds the device value, like resistance or capacitance.

Also, since there are non-cpp built in classes written, operators such as ‘==’ are overloaded for some classes.

The diagram in Figure 8 illustrates the final data structure:

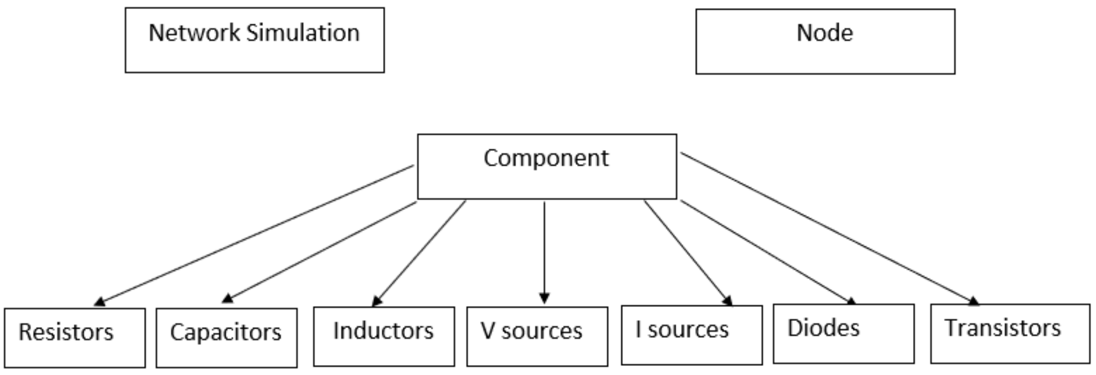


Figure 8

Although diodes and transistors inherit from the component class, they are used in the program differently to linear components due to their different behaviour physically. For example, they have a unique variable model_name which defines how they interact with the conductance matrix algorithm.

4.3 Parsing the input file

Initially, the input file would be parsed using the ifstream C++ library and custom operations. An ifstream object would be constructed with the parameter of the input file name. The benefits of this were that it gave full control on how to manipulate the information passed through. However, the disadvantages were that it was hard to assert whether the format of each input line was met. In addition, it would take a lot of nested loops and "if" statements to ensure the correct values are assigned to the data structure, e.g. considering the different multiplier symbols. The first iteration of this method can be seen in earlier commits from the GitHub repository in the file *read_file.cpp*.

Subsequently, it was decided to use Regular Expression's (regex) along with ifstream. The regular expressions library, "provides a class that represents regular expressions, which are a kind of mini language used to perform pattern matching within strings"[1].

[1] "Regular expressions library - cppreference.com", *En.cppreference.com*, 2020. [Online]. Available: https://en.cppreference.com/w/cpp/regex. [Accessed: 13- Jun- 2020].

The main operations with regexes can be characterised by operating on several of the following parameters:

- Target sequence: This is the sequence of characters that is searched when looking for a pattern.
- Pattern: The pattern which is searched for in the target sequence.
- Array matches: Some operations allow to retrieve information about matches.
- Replacement string: Some operations can replace matches[2].

The C++ regex standard library was introduced for C++ 11 and for this project was used to take the input file and assign the data to its respective components. The main benefit of using regex over a custom text processing method is that it makes data validation easy and simple. If a custom method were used, there would be too many possible variations of input text to cover and make the code unreadable/unfollowable. Thus, Regular Expression is needed to ensure the input file is of the format specified.

Netlist parser files are written in *netlist_parser.cpp* and *netlist_parser_helpers.cpp*.

The first use of regex in the software package is to identify the type of line being passed through the netlist parser. There are four different types:

- Component: *<designator> <node0> <node1> [<node 2] <value>*
- Comment: Begins with an Asterix (*)
- Transient simulation: *.tran 0 <stop time> 0 <timestep>*
- End of spice netlist: *.end*

The regex library function *regex_match* is then used to match the data structure with the type. *Regex_match* takes in, as a parameter, each netlist line and a type described above. It then returns

[2] "Regular expressions library - cppreference.com", *En.cppreference.com*, 2020. [Online]. Available: https://en.cppreference.com/w/cpp/regex. [Accessed: 13- Jun- 2020].

true if the parameter matches the type.[3] From this, the stringstream operator is used to pass the information to its respective data structure variables. Stringstream allows the assigning of string variables to be separated by whitespace. This is useful as all the types have their information separated by whitespace. The *regex_search* function is then used to check whether some sub-sequence in the target sequence matches the regular expression.[4]

4.4 Conductance Matrix Algorithm

4.4.1 Algorithm

The algorithm uses the data structure to populate three matrices', a conductance, voltage, and current matrix. This creates the equation $Ax = B$, where *A* is the conductance matrix (G matrix), x is the voltage matrix (V matrix) and *B* is the current matrix (I matrix). The solution for *x* is found by calculating the inverse of A then multiplying by *B*.

The conductance matrix, known as the G matrix, consists of the conductance's of resistors between nodes. A first example is shown in Figure 9:

$$\begin{bmatrix} G_{11} & -G_{12} & \cdots & -G_{1n} \\ -G_{21} & G_{22} & \cdots & -G_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ -G_{n1} & -G_{n2} & \cdots & G_{nn} \end{bmatrix} \cdot \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} i_1 \\ i_2 \\ \vdots \\ i_n \end{bmatrix}$$

Equation for finding a vector of unknown node voltages from a conductance matrix

Figure 9

The A matrix (G matrix) is populated differently under certain conditions. If 2 nodes are not connected to a voltage or current source, their value in the matrix is equal to the negative conductance of the resistor connecting them. For example, row 1 column 2, which corresponds to

[3] "regex_match - C++ Reference", *Cplusplus.com*, 2020. [Online]. Available: http://www.cplusplus.com/reference/regex/regex_match/. [Accessed: 13- Jun- 2020].

[4] "regex_search - C++ Reference", *Cplusplus.com*, 2020. [Online]. Available: http://www.cplusplus.com/reference/regex/regex_search/. [Accessed: 13- Jun- 2020].

node 1 and node 2, is shown above as -G12, where ‘G’ represents the conductance. The matrix is symmetrical, thus G12 = G21. The diagonals of the matrix are always positive while the off diagonals are negative.

If a non-reference node is connected to a voltage source, and the other terminal of that voltage source is connected to the reference node, the values in the G matrix change to reflect this, as shown in Figure 10. Node 1 is being used in this example. G11 takes the value of 1 as this is where column number equals row number. The rest of the values in the row are 0. The current matrix also changes, as instead of holding the value ‘i1” as in Figure 9, it now holds the voltage of the voltage source on the same row. It means that $V1 = Vsrc$.

$$\begin{bmatrix} 1 & 0 & \dots & 0 \\ -G_{21} & G_{22} & \cdots & -G_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ -G_{n1} & -G_{n2} & \dots & G_{nn} \end{bmatrix} \cdot \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} v_{src} \\ i_2 \\ \vdots \\ i_n \end{bmatrix}$$

Circuit analysis equation containing a voltage source between node 1 and reference

Figure 10

If a non-reference node is connected to a voltage source, and the voltage source is connected to another node which is not the reference node, this forms two rows in the G matrix to reflect the presence of a supernode. The first row represents the voltage source, but it now shows the difference between two nodes rather than an absolute, e.g. $V1 - V2 = Vsrc$. The second row uses Kirchhoff’s Current Law (KCL) for the supernode enveloping the voltage source. It is equal to the sum of the KCL equations for both the nodes in the supernode. For example, Gss in Figure 11 is equal to the sum of -G12 and G22. The node that the first row corresponds to is called the relationship supernode in our code. The node that the second row corresponds to is called the non-relationship supernode in our code.

$$\begin{bmatrix} 1 & -1 & \cdots & 0 \\ 0 & G_{SS} & \cdots & -G_{Sn} \\ \vdots & \vdots & \ddots & \vdots \\ -G_{n1} & -G_{n2} & \cdots & G_{nn} \end{bmatrix} \cdot \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} v_{src} \\ i_S \\ \vdots \\ i_n \end{bmatrix}$$

Circuit analysis equation containing a voltage source between node 1 and node 2

Figure 11

The principle is extended if more than one voltage source is connected to a node. Each voltage source creates one matrix row that defines its potential difference, and if the supernode enveloping all the connected voltage sources does not include the reference node, then there is also a row that describes KCL for the supernode.

The *x* matrix (V matrix) holds the voltages at each node and is what the algorithm solves. The size of the matrix is equal to the total number of voltage sources as its rows, with 1 column.

The *B* matrix (I matrix) holds the current source values going out of a node.

In the case of a relationship supernode, the value where the row corresponds to the relationship node of the supernode becomes the voltage value. In the case of a non-relationship supernode, the value where the row corresponds to the non-relationship supernode of the supernode becomes the total current source value going out of the supernodes.

The size of the matrix is also equal to the total number of voltage sources as its rows, with 1 column.

4.4.2 Linear Algebra Libraries

This section covers which linear algebra library is used and why it is chosen. C++ does not have an in-built matrix library. Researching potential linear algebra and matrix libraries showed there were a few potential ones that could be used in the product. These include Armadillo, Eigen and the

GNU Scientific Library. Armadillo is a library for linear algebra and scientific computing[5]; Eigen is a C++ template library for linear algebra, primarily matrices, vectors and numerical solvers[6]. The GNU Scientific Library (GSL) is more general purpose than the other two as it is a library for numerical computations in applied mathematics and science.[7]

All three software's are free to use, so cost is not relevant when choosing which one to use. However, because GSL is more general purpose, and the technical problems to be solved for this project only require linear algebra using matrices, it was not deemed suitable early in the project. With that being said, the GSL library offers numerical differentiation as one of its subject areas, something which the other two linear algebra libraries do not. As the non-linear components would require solving a Newton-Raphson iteration, it was considered to use both the GSL library for this and another library for the matrix manipulation. However, upon technical consultation it was agreed that it would be beneficial to program the Newton-Raphson iterations as a custom function as it was deemed simple enough to do so.

The Eigen library was chosen over Armadillo as Eigen comes with built-in matrix products and linear algebra decompositions. These are optimised for both small fixed-sized matrices and large dynamic sized ones. Armadillo, on the other hand, requires an external linkage to the Basic Linear Algebra Subprograms (BLAS)/Lapack libraries for matrix products and decompositions. In addition, an article on the performance of expression templates in C++, which looked to compare the most popular numerical computing libraries in C++, found that, "Eigen provides the best balance in

[5] "Armadillo: C++ library for linear algebra & scientific computing", *Arma.sourceforge.net*, 2020. [Online]. Available: http://arma.sourceforge.net/. [Accessed: 13- Jun- 2020].

[6] "Eigen", *Eigen.tuxfamily.org*, 2020. [Online]. Available: http://eigen.tuxfamily.org/index.php?title=Main_Page. [Accessed: 13- Jun- 2020].

[7] "GSL - GNU Scientific Library - GNU Project - Free Software Foundation", *Gnu.org*, 2020. [Online]. Available: https://www.gnu.org/software/gsl/. [Accessed: 13- Jun- 2020].

performance and compilation times for most numerical workload."[8] This was found by conducting various experiments using the same benchmarks.

To summaries, Eigen has essentially become an industry-standard scientific computing package for matrices, and so was used for all matrix construction and operations in the program. Using Eigen helps achieve the requirements due to its fast performance and well-documented characteristics over creating custom operations, for example using operator overloads. This is important as a certain level of efficiency and accuracy is required for the product to function as intended. Thus, having well-documented operations allows for easier monitoring of these requirements.

4.4.3 Eigen Implementation

The Eigen matrix class is used extensively in the program. The algorithm allows the program to know the column and rows in all three matrices at compile time, so although Eigen supports dynamic resizing, it was not needed. Thus, the matrices are constructed and populated at compile time. Matrices are then filled with values by functions like *create_G_matrix and create_I_matrix*.

As no storage order of Eigen matrices was specified, the default column-major storage order was used, as documented in Eigen 3.3.90[9]. However, the code was written to iterate by rows, so although this has posed a slight decrease in efficiency, due to time constraints it was decided that the performance hit was not essential to meet the project requirements.

Moreover, since the Eigen library is used, a different g++ compiling command needs to be used. It is mentioned in *readme.md*.

[8] R. Poya, "A look at the performance of expression templates in C++: Eigen vs Blaze vs Fastor vs Armadillo vs…", *Medium*, 2020. [Online]. Available: https://medium.com/@romanpoya/a-look-at-the-performance-of-expression-templates-in-c-eigen-vs-blaze-vs-fastor-vs-armadillo-vs-2474ed38d982. [Accessed: 13- Jun- 2020].

[9] "Eigen: Storage orders", *Eigen.tuxfamily.org*, 2020. [Online]. Available: https://eigen.tuxfamily.org/dox/group__TopicStorageOrders.html. [Accessed: 13- Jun- 2020].

4.5 Output File

The output file is of a Comma-Separated Value (CSV) format. The ‘fstream’ C++ library was used to create and write to the output file. Ofstream enables the program to open a file and write to it. It is used in the write_outputs_in_CSV.cpp file. Initially, the default constructor is called to construct the object that will hold the file.

This library is extensively used for these purposes due to its object-oriented interface and safety. Ofstream provides an overload to the “<<” operator which facilitates this function. Three separate void functions write to the file. Function write_csv_column_specifiers writes the first row, as this contains the headings like node indexes and component names. Function write_csv_voltage_row and function write_csv_current_row writes the time, node voltages, and currents through each component. In the end, the output csv file is inputted into MATLAB to generate the waveforms.

Input file name is set to be “netlist.txt”. Output file name is set to be “output.csv”.

To generate all the required outputs from an input file, different functions (mentioned in Section5, Section6 and section7) are utilised to carry out the correct operations. This is written in **the main function** in *write_outputs_in_CSV.cpp*.

The order of operations is as follows:

1. Converting conductors and inductors to their source equivalents in the network_components of the network_simulation.
2. Creating the V matrix vector.
3. Write column names into the CSV file.
4. A ‘for’ loop that increments the time by the timestep until it reaches the stop time.
   Inside the loop:
   1) Create the G, V and I matrices
   2) Solving the system of linear equations (Ax=b)
   3) Push the node voltages into the V matrix vector
   4) Write the node voltage into the CSV file
   5) Calculate the current through each component
   6) Write the currents into a CSV file

7) Update the values of the capacitor and inductor equivalent sources based on the state of the circuit at this time instance.

5. Parsing the Netlist

The netlist parser was heavily reliant on the specification for the Reduced Spice Format. Some backwards-compatible, additional features were added to allow the NETLISTs to be generated automatically from LTspice. This allowed a simple and direct comparison of simulation results, and was a very useful functionality that aided the overall development and debugging of the product.

In general, the input and output of a program are very important parts, as these are the interfaces, which other programs and further functionality, as well as the general product end-user would be using to communicate with the program.

The technical implementation of the parser was designed in a way, that it can be easily extended or modified, to accommodate for potential new features or syntax changes. Multiple technical strategies have been considered, discussed and evaluated prior to the development process.

In general, the process for parsing a netlist would be the following, regardless of the exact technical implementation:

1. Reading a netlist file into the program
2. Validating the netlist for correctness
3. Constructing the internal data structure of the circuit

In essence, this requires the validation, classifcation and processing of each netlist line. One option, which was considered was the “manual” deconstruction of the netlist lines, and then checking for patterns using nested-if statements. Although, this method would possibly be the fastest and most lightweight method, in terms of computation, there were various reasons why this approach was discarded. Firstly, the hard-coded patterns would be very hard to, implement, test, maintain and extend. There would be a lot of nested if-statements, and keeping a good overview over the code-base would be extremely difficult, which would also make the debugging process extremely hard, should there be any errors occurring.

Exactly for that reason, it was decided to use a string pattern-matching extension called regex. Regex easily permits the verification of netlist syntax, by simply defining regex string patterns, according to the “Reduced Spice Format”. These patterns can then be “matched”, and the netlist line can be easily verified and classified this way, with perfect precision. One example for this is illustrated below:

```
(V|I|R|C|L|D|Q)[0-9]+ N?[0-9]+ N?[0-9]+ (N?[0-9]+ )?.+
```

The above pattern is a regex pattern corresponding to a component line in the netlist. In the reduced netlist format specification, a component line is outlined to have the following format:

```
<designator> <node0> <node1> [<node2>] <value>
```

This directly corresponds to the regex format, which will be broken down in the following:

```
(V|I|R|C|L|D|Q)[0-9]+ N?[0-9]+ N?[0-9]+ (N?[0-9]+ )?.+
```

The highlighted section specifies, that the first character of a component line must be either a V or I or R …

```
(V|I|R|C|L|D|Q)[0-9]+ N?[0-9]+ N?[0-9]+ (N?[0-9]+ )?.+
```

This section specifies, that one or more digits must follow, with a space afterwards.

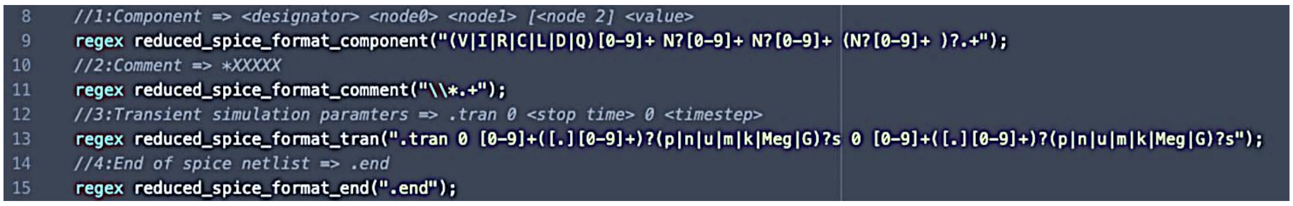
```
(V|I|R|C|L|D|Q)[0-9]+ N?[0-9]+ N?[0-9]+ (N?[0-9]+ )?.+
```

In regex-format, a node is specified by an optional (?=optional) leading N character, with one or more digits following afterwards, and again separated by a trailing space.

```
(V|I|R|C|L|D|Q)[0-9]+ N?[0-9]+ N?[0-9]+ (N?[0-9]+ )?.+
```

The other nodes follow a similar pattern, the highlighted end with a dot is used for the value of the component, which can have multiple formats, and is classified more precisely in a subsequent step. The dot designates any character in regex, and the plus means that any sequence of characters can be trailing of the end.

This walkthrough covers all regex-features and syntax that was used in the netlist parser. Another tool, which was used to create and debug regex expression was the web-application[10], which allows a simple creation of regex expression, and offers live-highlighting of the string that should be matched. This tool was heavily used during the development of the regex parser.



```
//1:Component => <designator> <node0> <node1> [<node 2] <value>
regex reduced_spice_format_component("(V|I|R|C|L|D|Q)[0-9]+ N?[0-9]+ N?[0-9]+ (N?[0-9]+ )?.+");
//2:Comment => *XXXXX
regex reduced_spice_format_comment("\\*.+");
//3:Transient simulation paramters => .tran 0 <stop time> 0 <timestep>
regex reduced_spice_format_tran(".tran 0 [0-9]+([.][0-9]+)?(p|n|u|m|k|Meg|G)?s 0 [0-9]+([.][0-9]+)?(p|n|u|m|k|Meg|G)?s");
//4:End of spice netlist => .end
regex reduced_spice_format_end(".end");
```

Figure 12

Above is an overview of the regex-patterns used to classify a netlist line. The patterns are simply stored as variables, and can therefore easily be adapted or appended, which makes maintenance and further developments of the product simple and fast.

6. Code for Building the G, V and I Matrices

To construct the G, V and I matrices, two Cpp files*, matrix_factory.cpp* and *matrix_helpers.cpp*, are written. *Matrix_helpers.cpp* contains the helper functions for the matrix construction functions in *matrix_factory.cpp*. All functions are first declared in *simulator.hpp.*

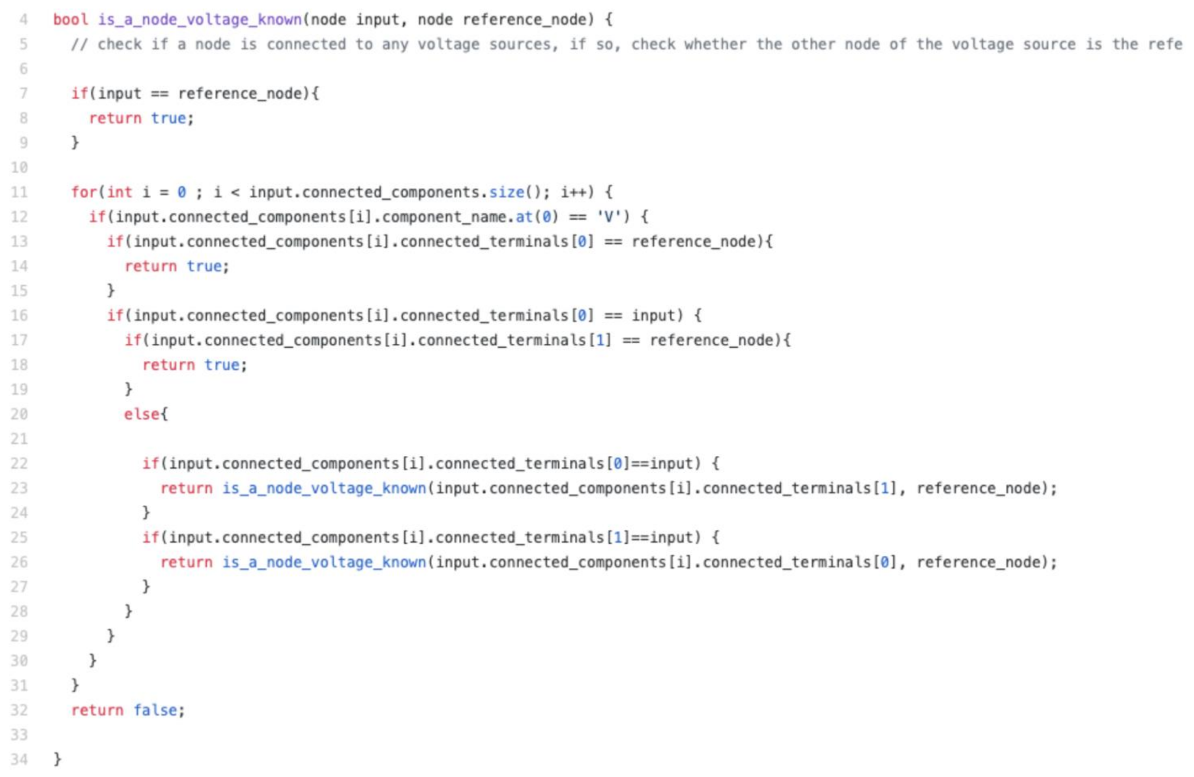
[10] RegExr. 2020. *Regexr: Learn, Build, & Test Regex*. [online] Available at: <https://regexr.com> [Accessed 14 June 2020].

6.1 Helper functions:

The helper functions are written for the purpose of separating different node types, which are known nodes, normal nodes and two types of supernodes. When the matrix construction functions are explained, the helper functions will be mentioned as well.

6.1.1 Is a node voltage known?

As shown in Figure 13, a Boolean function, which outputs whether the input node's voltage is already known comparing to the reference node, is written in *matrix_helpers.cpp line 4*.



```
4   bool is_a_node_voltage_known(node input, node reference_node) {
5     // check if a node is connected to any voltage sources, if so, check whether the other node of the voltage source is the refe
6
7     if(input == reference_node){
8       return true;
9     }
10
11    for(int i = 0 ; i < input.connected_components.size(); i++) {
12      if(input.connected_components[i].component_name.at(0) == 'V') {
13        if(input.connected_components[i].connected_terminals[0] == reference_node){
14          return true;
15        }
16        if(input.connected_components[i].connected_terminals[0] == input) {
17          if(input.connected_components[i].connected_terminals[1] == reference_node){
18            return true;
19          }
20          else{
21
22            if(input.connected_components[i].connected_terminals[0]==input) {
23              return is_a_node_voltage_known(input.connected_components[i].connected_terminals[1], reference_node);
24            }
25            if(input.connected_components[i].connected_terminals[1]==input) {
26              return is_a_node_voltage_known(input.connected_components[i].connected_terminals[0], reference_node);
27            }
28          }
29        }
30      }
31    }
32    return false;
33
34  }
```

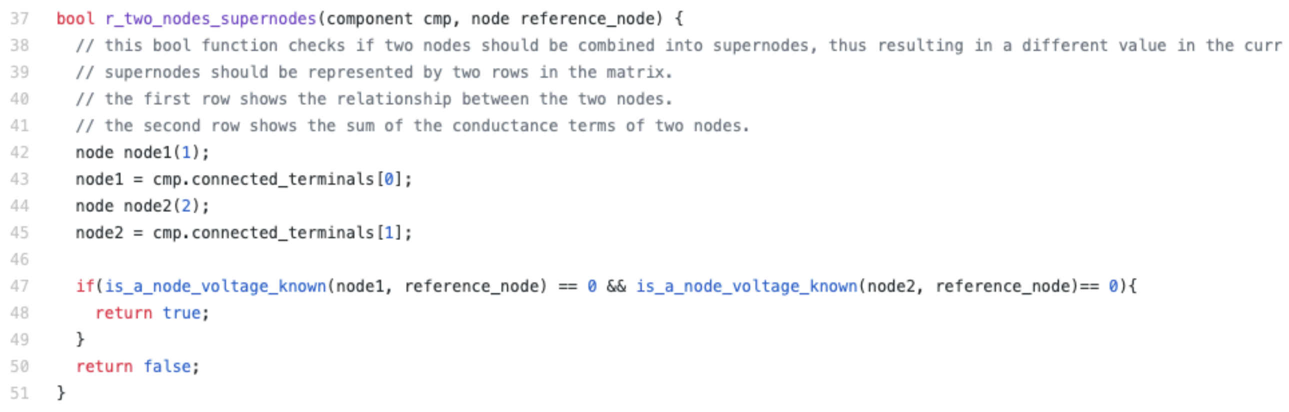
Figure 13

The function checks if the input node is the reference node and outputs true if it is. It then goes through all the connected components of the input node to see if there is any voltage source connected to the input node. If there is, it returns true if the voltage source is connected to the reference node (line 12 to 19) or another node with known voltages (line 20 to 26).

6.1.2 Are two nodes supernodes?

As shown in Figure 14, a bool function, that checks where the two nodes a voltage source is connected to are supernodes, is written in *matrix_helpers.cpp line 37*:

bool r_two_nodes_supernodes(component cmp, node reference_node)



```
bool r_two_nodes_supernodes(component cmp, node reference_node) {
  // this bool function checks if two nodes should be combined into supernodes, thus resulting in a different value in the curr
  // supernodes should be represented by two rows in the matrix.
  // the first row shows the relationship between the two nodes.
  // the second row shows the sum of the conductance terms of two nodes.
  node node1(1);
  node1 = cmp.connected_terminals[0];
  node node2(2);
  node2 = cmp.connected_terminals[1];

  if(is_a_node_voltage_known(node1, reference_node) == 0 && is_a_node_voltage_known(node2, reference_node)== 0){
    return true;
  }
  return false;
}
```

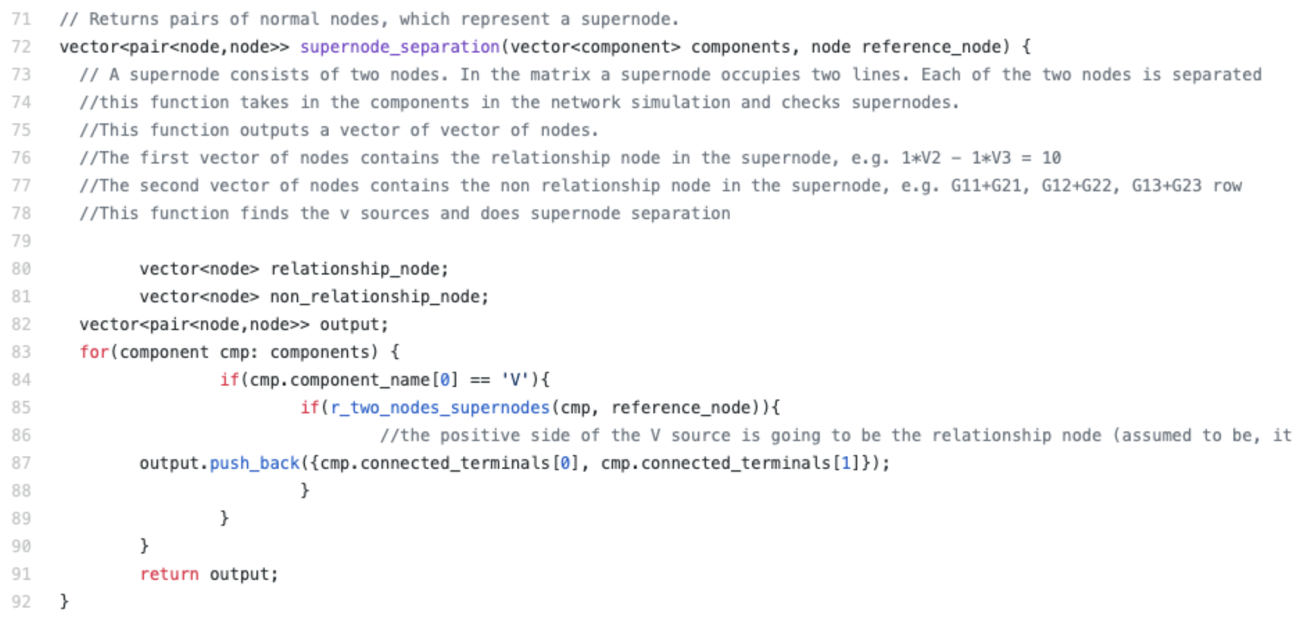
Figure 14

This function works by getting a voltage source component and the reference nodes as its inputs. It then checks if the two nodes that the voltage source is connected to are both nodes with unknown voltages.

6.1.3 Separating supernodes

The supernode separation function takes in a vector of components and finds out if supernodes exist in the connected nodes of the components. The function then classifies the supernodes to relationship supernodes and non-relationship supernodes and combines them into pairs. The first term in the pair is always a relationship node. The second term in the pair is always a non-relationship node. In the end, a vector that contains all supernode pairs is returned. Figure 15 shows the code for this function. The function is written in *matrix_helpers.cpp line 72*.

```
vector<pair<node,node>> supernode_separation(vector<component> components, node reference_node)
```



```
// Returns pairs of normal nodes, which represent a supernode.
vector<pair<node,node>> supernode_separation(vector<component> components, node reference_node) {
  // A supernode consists of two nodes. In the matrix a supernode occupies two lines. Each of the two nodes is separated
  //this function takes in the components in the network simulation and checks supernodes.
  //This function outputs a vector of vector of nodes.
  //The first vector of nodes contains the relationship node in the supernode, e.g. 1*V2 - 1*V3 = 10
  //The second vector of nodes contains the non relationship node in the supernode, e.g. G11+G21, G12+G22, G13+G23 row
  //This function finds the v sources and does supernode separation

        vector<node> relationship_node;
        vector<node> non_relationship_node;
  vector<pair<node,node>> output;
  for(component cmp: components) {
                if(cmp.component_name[0] == 'V'){
                        if(r_two_nodes_supernodes(cmp, reference_node)){
                                //the positive side of the V source is going to be the relationship node (assumed to be, it
        output.push_back({cmp.connected_terminals[0], cmp.connected_terminals[1]});
                        }
                }
        }
        return output;
}
```

Figure 15

6.1.4 Indexing Nodes

Function *which_is_the_node* returns the index of a node in a vector of nodes. If the node is not in the vector, the function returns -1. The function is in *matrix_helpers.cpp line 94*.

6.1.5 Indexing components

Function *which_is_cmp* and Function *which_is_cmp1* return the index of a component in a vector of components. If the node is not in the vector, they return -1. The functions differ a bit as one is used to find the index of the original capacitor or the inductor knowing the equivalent source name. The other one works just by looking for the component with the same component_name in the vector. The function are in *matrix_helpers.cpp line 104 and 114*.

6.1.6 Conductance Between Nodes

The *calculate_conductance_between_nodes* function in *matrix_factory.cpp* takes two nodes as inputs and returns their conductance term in the G matrix. If two input nodes are the same, the output is the diagonal terms in the G matrix. The function returns the sum of conductance of all resistors that the node is connected to. If the two input nodes are different, the output is the non-diagonal terms in the G matrix, which are negative. Thus, the function returns the negative sum of conductance of all common resistors which two nodes are both connected to. The function is shown in Figure 16.

```cpp
double impedance(component cmp) {
  if (cmp.component_name[0] == 'R') {
    return cmp.read_value()[0];
  }
  return 0.0; // avoids compiler warnings
}

double sum_conductance(vector<component> components) {
  double sum;
  for(component cmp: components) {
    if(cmp.component_name[0] == 'R')
    sum += 1.0/impedance(cmp);
  }
  return sum;
}


double calculate_conductance_between_nodes(node A, node B) {

  // For diagonal conductance matrix entries (G11 ,G22, G33 ...)
        if(A==B){
                return sum_conductance(A.connected_components);
        }

    // For all other conductance matrix entries (G12, G21, G13, G31 ...)
    vector<component> common_components_between_AB;
    for(int i = 0 ; i < A.connected_components.size(); i++){
      for(int c = 0; c < B.connected_components.size(); c++){
        if(A.connected_components[i] == B.connected_components[c]){
          common_components_between_AB.push_back(A.connected_components[i]);
        }
      }
    }
    return -sum_conductance(common_components_between_AB);
}
```

Figure 16

6.2 G, V and I matrix Construction

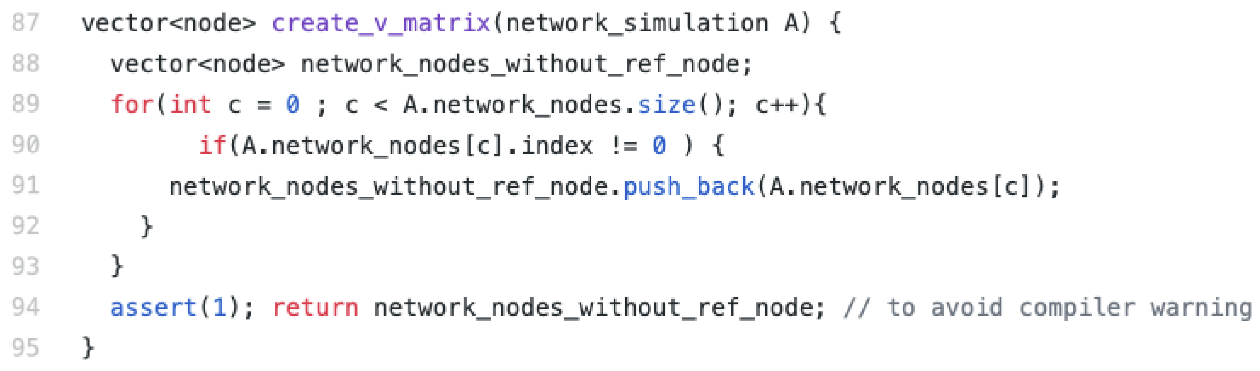
6.2.1 Overview

The matrix construction functions utilise the written helper functions and writes the value into the matrices. The V matrix will be first explained, as the rows in the G matrix and the I matrix need to correspond to the order of the node voltage terms in the V matrix.

The matrix construction functions are written in *matrix_factory.cpp.*

6.2.2 V Matrix

V matrix is created as a vector of nodes instead of a matrix because it is not used in linear algebra algorithm. Its function is to contain a fixed order of the nodes for the G matrix and I matrix to refer to.



```
vector<node> create_v_matrix(network_simulation A) {
  vector<node> network_nodes_without_ref_node;
  for(int c = 0 ; c < A.network_nodes.size(); c++){
        if(A.network_nodes[c].index != 0 ) {
      network_nodes_without_ref_node.push_back(A.network_nodes[c]);
    }
  }
  assert(1); return network_nodes_without_ref_node; // to avoid compiler warning
}
```

Figure 17

The *create_V_matrix* function is shown in Figure 17. It takes in a network_simulation object and outputs a vector of nodes that excludes the reference node. It iterates through the network_nodes member of the network_simulation class. If the node is not the reference node, which has node.index of ‘0’, it is pushed to a vector called network_nodes_without_ref_node.

6.2.3 G Matrix

After the V matrix is created, the order of the nodes in the V matrix is fixed, and the G matrix can be created. The *create_G_matrix* function takes in a network_simulation and outputs the conductance matrix of it.

The *create_G_matrix* function is in *matrix_factory.cpp line 149*.

In the *create_G_matrix* function, the *create_V_matrix* function is first called to retain the order of the nodes in the V matrix. Knowing the size of the V matrix vector, the G Matrix that contains values of type double is constructed with the number of rows and columns equal to the size of the V matrix vector.

The reference node is found by looking for the node with index '0' in the network_nodes member of network_simulation input. A vector that contains pairs of nodes is then created by function supernode_separation to hold the supernodes in the network.

The function assigns values into the columns of the matrix in one row first and then moves to the next row until the whole matrix is filled. The function addresses each row differently according to the four types of nodes in the V matrix, which are normal nodes, known nodes, relationship supernodes and non-relationship supernodes. This is done by various 'if' statements in the code.

For normal nodes, the function calls the calculate_conductance_between_nodes function to get the value at each column of the row.

For known nodes, the is_a_node_voltage_known function is called. The create_G_matrix finds the column that corresponds to the known nodes and makes it 1. All the other columns in the row are 0.

For supernodes, the function first checks if it is a relationship supernode or a non-relationship supernode. If it is a relationship supernode, the row is first filled with 0s. The column that corresponds to the supernode, which is the positive terminal of the voltage source, is filled with 1. The column that corresponds to the supernode, which is the negative terminal of the voltage source, is filled with -1.

If a node is a non-relationship supernode, the row is filled with the sum of the conductance terms from each node in the pair of supernodes. For example, the term at the first column of the row $(G_{supernodes\ 1})$ should be $G_{supernode1\ 1} + G_{supernode2\ 1}$.

After the whole matrix is filled, the G matrix is returned.

6.2.4 I Matrix

The I matrix takes in a network_simulation object and the simulation_progress as inputs. This is because the terms in the I matrix reflect the value of AC sources sometimes.

The *create_I_matrix* function is in *matrix_factory.cpp line 53*.

In the *create_I_matrix* function, *create_V_matrix* function is also first called to retain the order of the nodes in the V matrix. Knowing the size of the V matrix vector, the I Matrix that contains values of type double is initiated with the number of rows equal to the size of the Vmatrix vector and one column.

The reference node is found by looking for the node with index ‘0’ in the network_nodes vector of the network_simulation object. A vector that contains pairs of nodes is then created by function supernode_separation to hold the supernodes in the network.

The *create_I_matrix* function also addresses each row differently according to the type of nodes that each row corresponds to.

For regular nodes, the known currents going out of the node, which are produced by the current sources, are summed up and filled into the row.

For known nodes, the value of the voltage source that the node is connected to is filled into the row. If the positive side of the V source is connected to the node, the value is positive. If the negative side of the V source is connected to the node, the value is negative.

For supernodes, the function first checks the type of the supernode with the supernode pairs, so that it knows if a node is a relationship supernode or a non-relationship supernode. For relationship supernodes, the absolute value of the voltage source the supernodes are both connected to is written into the row.

For non-relationship supernodes, the sum of the known currents going out of each supernode in the pair is calculated and written into the row.

After every row is filled, the I matrix is returned.

6.3 Capacitors and Inductors

6.3.1 Converting them to the equivalent sources

Capacitors and inductors are considered as V sources and I sources when creating the matrix. Across a capacitor, the voltage cannot change abruptly. The current through an inductor also cannot change abruptly. Therefore, they can be considered as sources whose value depend on the history of the circuit.

The functions that calculate their value at a time instance are listed below:

$$Vcap(t) = \frac{1}{C}\int_{t0}^{t1} i(t)\,dt + V(t0)$$

$$i_{inductor}(t) = \frac{1}{L}\int_{t0}^{t1} V(t)\,dt + i(t0)$$

To convert them to their equivalent sources, function *convert_CLs_to_sources* is written in *matrix_helpers.cpp line 194*. It is a void function which changes the network_components and network_nodes member of an input network_simulation.

It first initiates the equivalent sources with all component values equal to zero and the same connected_nodes as the original capacitor or the original inductor has. Then, it iterates through the network_components to replace the capacitors and the inductors with their equivalent sources. The new sources are named like “I_L1” or “V_C1”.

After the network_components member is updated with the new equivalent sources, the network_nodes member is also updated to make sure that the new equivalent sources are included in their member vector connected_components. This is done by using the function *which_is_cmp* to find the index of the original capacitor or the original inductor in the connected_components member of the node and replacing them with the new equivalent source.

6.3.2 Updating the value of the equivalent sources

As the equivalent sources’ value is based on the history of the circuit, their values need to be changed after every time instance. This is done by function *update_source_equivalents* in *matrix_helpers.cpp line 125*.

Function *update_source_equivalents* updates the value of the equivalent sources by looking at the node voltages and current through components at the last time instance. It does not carry out integrations. Instead, it carries out the following two functions to approximate the integration value:

$$Vcap(tstep1) = \frac{1}{C} \times i(tstep0) * timestep + Vcap(tstep0)$$

$$i_{inductor}(tstep1) = \frac{1}{L} \times V(tstep0) * timestep + i_{inductor}(tstep0)$$

Since the results are only estimated, the accuracy of the estimation changes if timestep changes. With smaller timesteps, the lag in updating the equivalent source values is smaller. Thus, the result gets closer to the actual integration result when timestep is smaller.

The working principle of function *update_source_equivalents* is more or less the same as function *convert_CLs_to_sources*. The only difference is that the equivalent sources are initiated with component_value[0] equal to the value of the estimation.

After the equivalent source is initiated with the correct value based on the history of the circuit by function *update_source_equivalents*, the network_component and the network_nodes member of the input network_simulation are updated as in function convert_CLs_to_sources. The old equivalent sources are replaced by the new ones with the correct values.
Experimenting with different timesteps, it was found that if the timestep is about 100 times smaller than the lowest frequency of the source present in a circuit. The estimations are good enough to produce a simulation output which looks the same as the LTspice simulation result.

7. Current Through Components

7.1 Overview

After the conductance matrix algorithm is written correctly for all supported components, the voltage at each node at every time instance can be calculated. Therefore, the current through each component can also be calculated. The functions for this are in *matrix_helpers.cpp.*

7.2 Current through R:

The function for calculating the current through a resistor is called *calculate_current_through_R*. It first finds out which two nodes the resistor is connected to. The node voltages of these two nodes are found in Vvector, which always contains the updated node voltage value in a network. Thus, the voltage difference across the resistor is calculated. In the end, the following calculation is carried out to output the current.

$$i_R = \frac{voltage\ difference\ across\ R}{resistor\ value}$$

The function *calculate_current_through_R* is in *matrix_helpers.cpp line 310.*

7.3 Current through I source:

According to the definition of a current source, the amount of current going through it is always the source value of itself. Therefore, the following calculation is carried out.

$$i_{Isrc} = amplitude \times \sin(2\pi * frequency * time)$$

Inductors are treated as I sources when calculating their current.

7.3.1 Current through V source:

The current through a voltage source is calculated by the *tell_currents* function. It works by a process of elimination. It sums up the currents from all other nodes. According to KCL, the sum of currents going out of one node is always zero. Therefore, by summing up Rs' and Isources' currents going out or into the node that the voltage source is connected to, the current through a voltage source can be calculated. If there are multiple voltage sources connected to a node, the current through a voltage source is calculated by finding the node that only has one voltage source.

Capacitors are treated as V sources when calculating their current.

The *tell_currents* function is in *matrix_helpers.cpp line 238*.

7.3.2 Putting everything together

```
vector<double> calculate_current_through_component(vector<component> network_component,
vector<node> Vvector, double simulation_progress)
```

The function *calculate_current_through_component* utilises all the current calculation functions above and returns a vector of double which contains the current through each component in the input object of type network_component. It checks the name of each component and chooses the correct method for calculating the current accordingly.

The function *calculate_current_through_component* is written in *matrix_helpers.cpp line 290.*

8. Testing

8.1 Overview

Testing was done on Windows and Linux to ensure compatibility on both Operating Systems (OS). On Linux, the g++ command was used for the compilation, assembly and linking of source code to generate an executable file. *Compiled_Test.out* was then used for execution. Most of the test code was created using separate C++ files that would contribute to a main function for writing out the output. The exact compilation command is included in *readme.md*.

Before writing the conductance matrix algorithm, the netlist parser was tested. The requirements to pass the tests were as follows:

The netlist line must be correct when creating components for:
1.Simulation setting
2. Component name and type
3. The order of connected nodes
4. Value or function
5. Comments to be ignored

The conductance matrix algorithm was then tested, initially with just resistors due to the added complication of turning capacitors and inductors into voltage and current sources, respectively. There are two files to test, *matrix_factory.cpp* and *matrix_helpers.cpp*, which includes the functions that derive the three matrices, G matrix, V matrix, and I matrix.

As non-linear components were not implemented, they could not be tested.

8.2 Simple Circuit Test

Initially, the simplest circuit was tested. The circuit consists of one resistor, ‘R1’, and a DC voltage source ‘V1’ of value 5V.

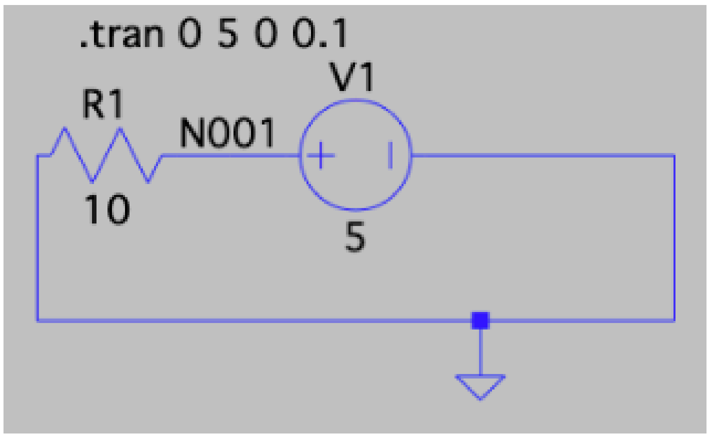


Figure 18

The simulation, shown in Figure 18 with the syntax “.tran 0 5 0 0.1” translates to a transient simulation with a stop time of 5 seconds and a timestep of 0.1 second. The LTSpice simulation results are shown in Figure 19:

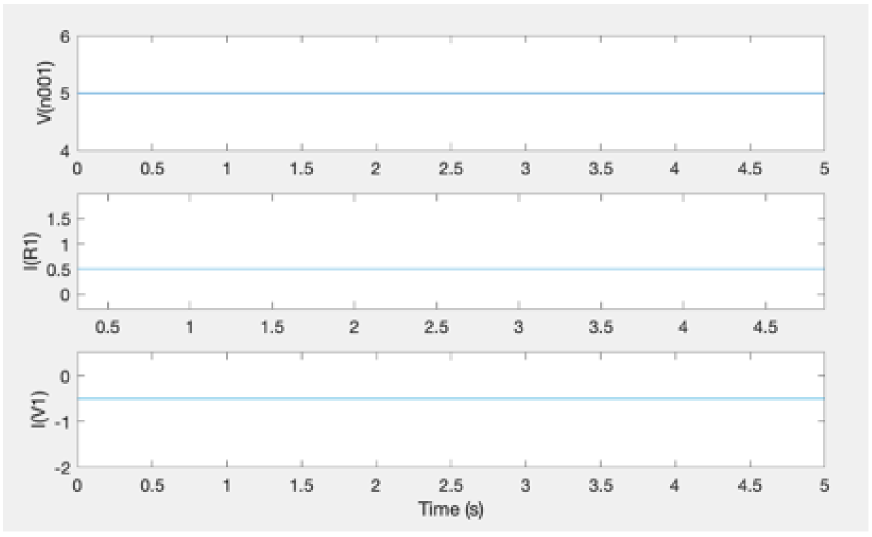


Figure 19

The results show that R1 has a value of 0.5Ω, the voltage at node N001 is 5V and the current through R1 is -0.5A. Shown in Figure 20 is the test results of the software package:

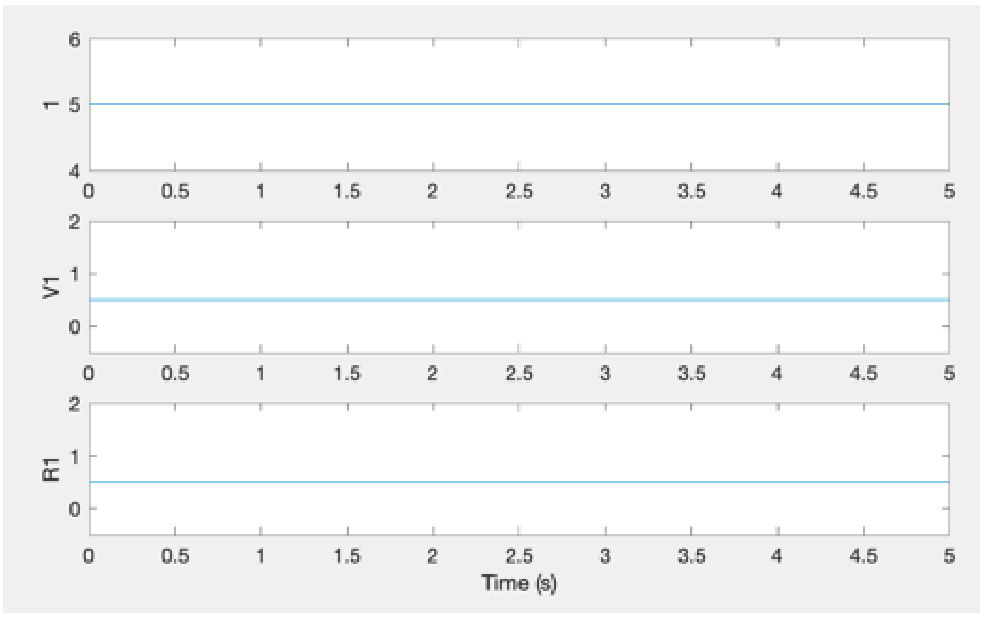


Figure 20

Therefore, the software is performing correctly given this simple circuit as R1 also has a value of 0.5Ω, node 1 has a voltage of 5V. In the project simulation, the current was specified as an absolute value, thus Figure 20 shows a current value of 0.5A whereas LTSpice simulation is -0.5A.

8.3 Supernode Test Case

A circuit containing a supernode was tested next. The circuit is shown in Figure 21:

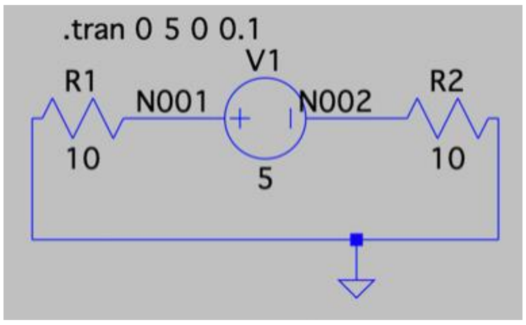


Figure 21

The same simulation setting was applied as in the simple circuit. Now, there are two resistors, R1 and R2, and thus two nodes (not including the reference node), N001 and N002. The voltage source, V1, is still 5V. The LTSpice simulation plot is shown in Appendix F. The LTSpice simulation yields the same results as our software package. This shows that the program can correctly manage supernodes.

8.4 Circuit with a Current Source

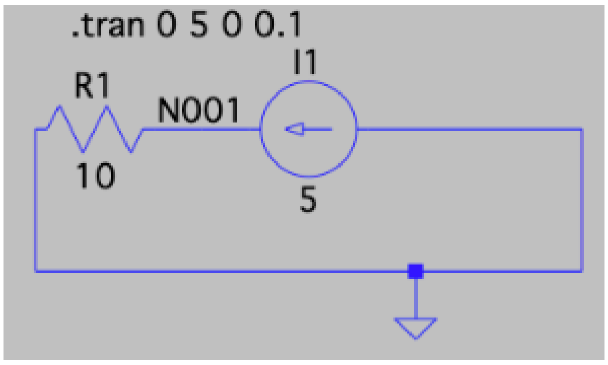


Figure 22

Like the simple circuit, however the voltage source V1 is replaced by the current source I1, shown in Figure 22. The circuit schematic and test simulations are shown in Appendix G, which shows that LTSpice simulation produces the same results as the software package. The significance of this test is that it shows the program works with current sources.

8.5 AC Circuit with Voltages and Current Sources

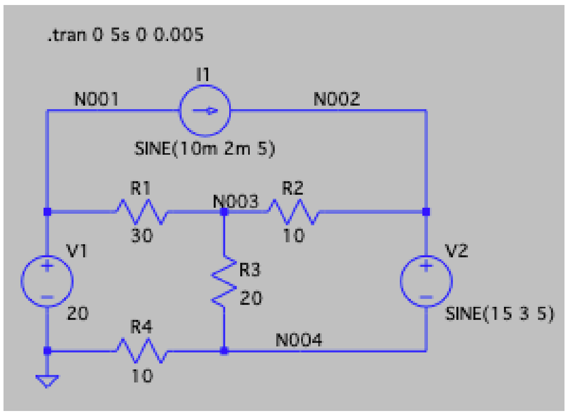


Figure 23

The last circuit tested before including capacitors and inductors is shown in Figure 23, with the results in Appendix G. The circuit contains supernodes, AC current source, AC voltage source and various resistors. The results show that the software package gives the exact same results as LTSpice. Therefore, it can be concluded with a high level of confidence that the software is correct for any circuit containing just resistors, current sources, and voltage sources.

8.6 Circuit with a Capacitor

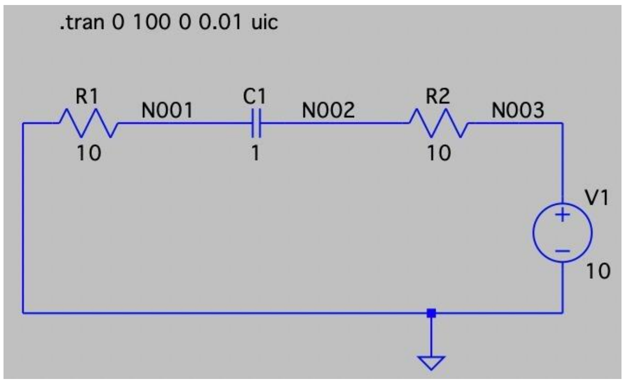


Figure 24

A circuit which had a capacitor was then tested, shown in Figure 24. The plots of the simulations can be found in Appendix I. The software package had the exact same simulation as the LTSpice simulation. Thus, the program works correctly with capacitors.

8.7 Circuit with an Inductor

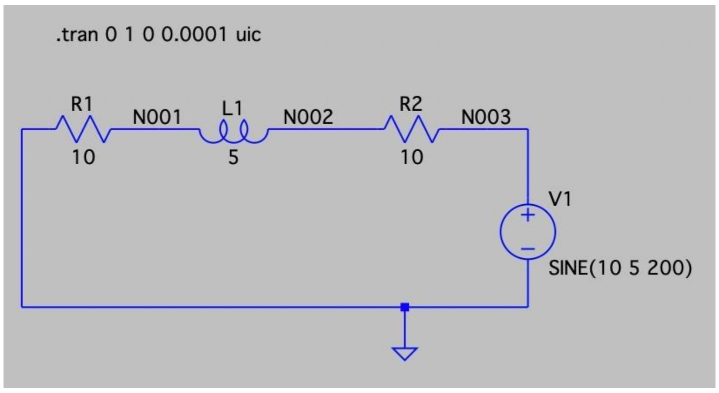


Figure 25

A circuit with an inductor was then tested, shown in Figure 25. The plots of the simulations can be found in Appendix J. The software package had the exact same simulation as the LTSpice Simulation. Thus, the program works correctly with inductors.

8.8 Efficiency and Performance

Finally, the software package was tested to see how a change in the number of nodes in a circuit affected efficiency and performance. The test results table can be found in Appendix I. The percentage of CPU used was always varying, so an average value is chosen.

The test circuits are carried out with the command “.tran 0 10s 0 0.001s”. In every test, the node voltages and current through components are written for 10,000-time instances. The test circuits consist of resistors and voltage sources in series.

The CPU used is the Intel Core i9-9880H which has a power consumption of 45watt (TDP = Thermal Design Power)[11]. The memory used is 32 GB 2400 MHz DDR4. Its power consumption is 11.85W[12].

TDP is a measure of the maximum amount of heat that a CPU can generate under any workload.

One watt is one joule per second. Thus, the amount of energy used by the CPU and memory is estimated using the following function.

$$Energy\ consumption_{test} = (Power_{CPU} \times \%CPU + \ Power_{MEM} \times \%MEM) \times \ time$$

It can be seen from the graph in Figure 26 that the energy consumption increases exponentially when the number of nodes increases. The x-axis is the number of nodes and the y-axis is the energy consumption. This might be caused by our implementation choice of the project. As no pointers are used, the nodes and components are created for multiple times which leads to power inefficiency. Memory usage was not affected by a change in the number of nodes.

[11] "Intel® Core™ i9-9880H Processor (16M Cache, up to 4.80 GHz) Product Specifications", *Ark.intel.com*, 2020. [Online]. Available: https://ark.intel.com/content/www/us/en/ark/products/192987/intel-core-i9-9880h-processor-16m-cache-up-to-4-80-ghz.html. [Accessed: 13- Jun- 2020].

[12] C. Angelini and I. Wallossek, "Intel Core i7-5960X, -5930K And -5820K CPU Review: Haswell-E Rises", *Tom's Hardware*, 2014. [Online]. Available: https://www.tomshardware.com/reviews/intel-core-i7-5960x-haswell-e-cpu,3918-13.html. [Accessed: 13- Jun- 2020].

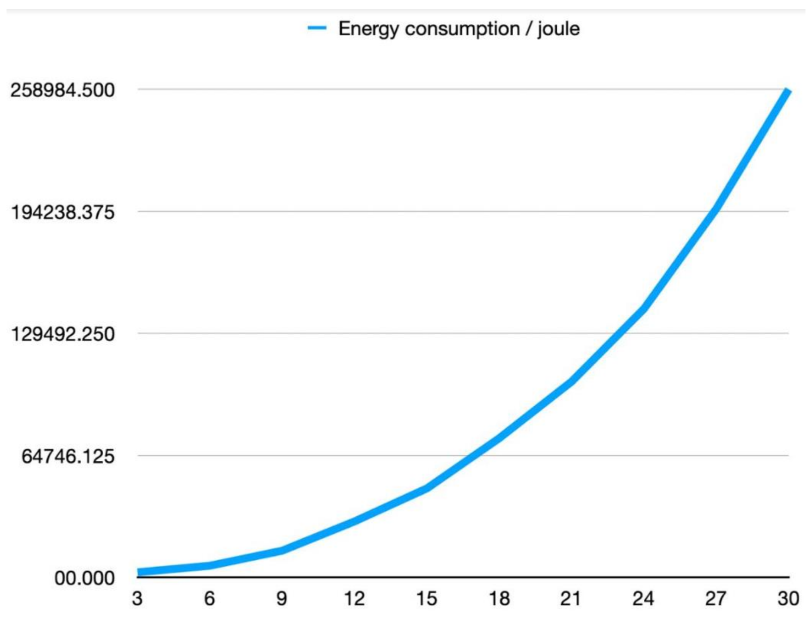


Figure 26

9. Debugging Techniques

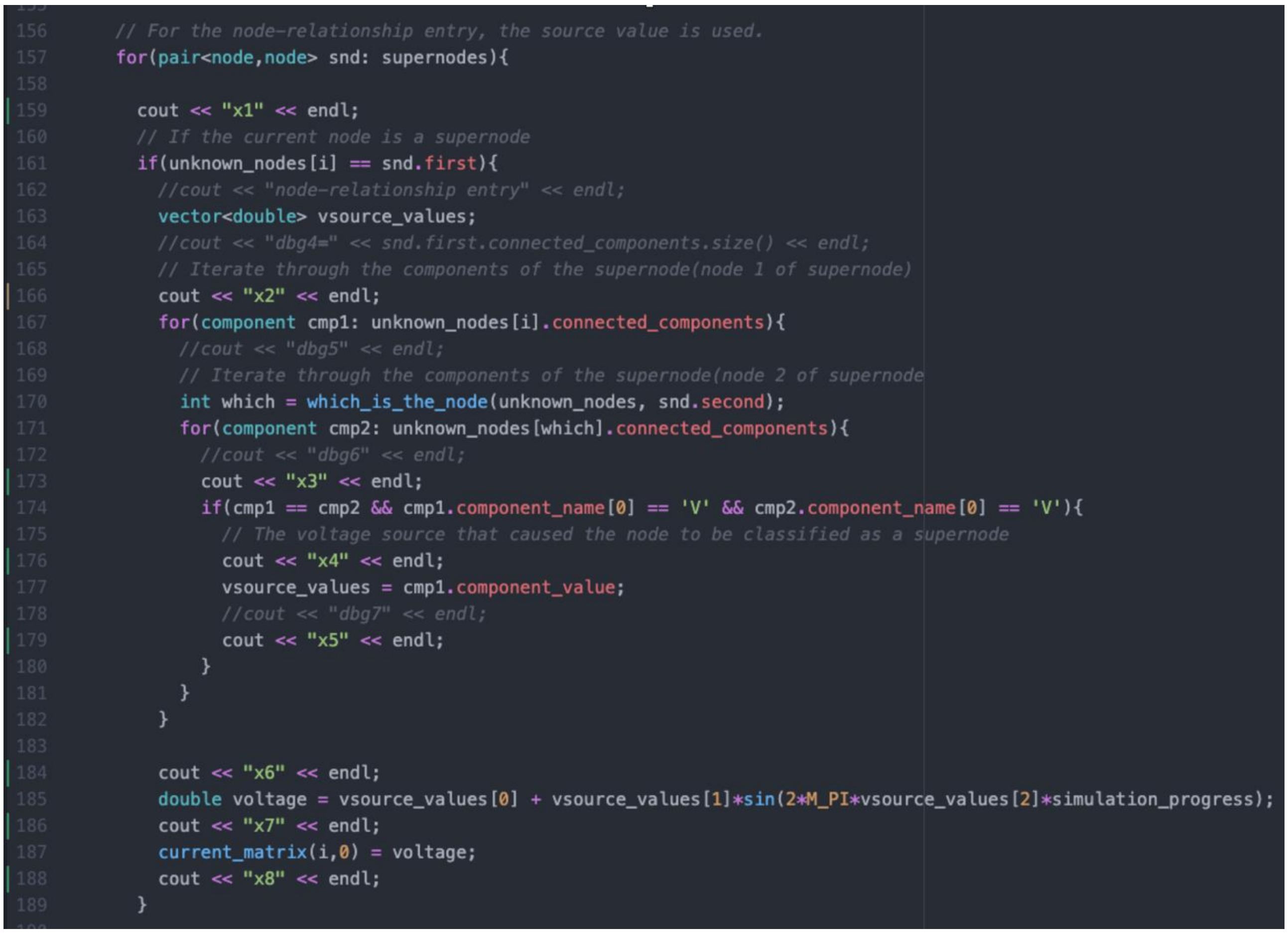
Errors are a natural part of the software development journey. The skill lies in managing the risk associated with potential errors. In most realistic settings this happens by rigorous testing of the software. Ideally, the testing of the program should happen as early as possible, as the development of further software components often relies on previous software components, hence testing of downstream developments can be difficult without good upstream test coverage.

Therefore, the team made sure that testing was done as early as possible, ideally immediately after the completion of new functionality, where this was possible. As is expected, debugging errors is a normal part of the testing process, and a variety of techniques were used, some of which is outlined in the following.

One tool which simplifies the debugging process of memory issues (including memory leakages) is Valgrind. It is a very popular tool, which enables the developer to trace back and identify the responsible locations in the source-code.

Another simpler, yet highly effective technique is the accurate and clever placement of *cout* << statements, or assert() statements within the code, that allow the programmer to identify faulty locations within the source code. Especially in a low-level language like C++, which gives the programmer a high level of control about the exact program behaviour, there can be a lot of sources for errors. Errors often express themselves in very unpredictable program outputs, which rarely allow exact pinpointing of the error's origin.

This manual debugging process has been utilised numerous times throughout the development journey of our product, and an example scenario is described in the following:



```
// For the node-relationship entry, the source value is used.
for(pair<node,node> snd: supernodes){

 cout << "x1" << endl;
 // If the current node is a supernode
 if(unknown_nodes[i] == snd.first){
  //cout << "node-relationship entry" << endl;
  vector<double> vsource_values;
  //cout << "dbg4=" << snd.first.connected_components.size() << endl;
  // Iterate through the components of the supernode(node 1 of supernode)
  cout << "x2" << endl;
  for(component cmp1: unknown_nodes[i].connected_components){
   //cout << "dbg5" << endl;
   // Iterate through the components of the supernode(node 2 of supernode
   int which = which_is_the_node(unknown_nodes, snd.second);
   for(component cmp2: unknown_nodes[which].connected_components){
    //cout << "dbg6" << endl;
    cout << "x3" << endl;
    if(cmp1 == cmp2 && cmp1.component_name[0] == 'V' && cmp2.component_name[0] == 'V'){
     // The voltage source that caused the node to be classified as a supernode
     cout << "x4" << endl;
     vsource_values = cmp1.component_value;
     //cout << "dbg7" << endl;
     cout << "x5" << endl;
    }
   }
  }

  cout << "x6" << endl;
  double voltage = vsource_values[0] + vsource_values[1]*sin(2*M_PI*vsource_values[2]*simulation_progress);
  cout << "x7" << endl;
  current_matrix(i,0) = voltage;
  cout << "x8" << endl;
 }
```

Figure 27

Somewhere in the above section of source code, a segmentation fault occurred. By simply placing different *cout* << statements at regular intervals through the code, the origin of the error can be precisely pinpointed by looking at the program output.

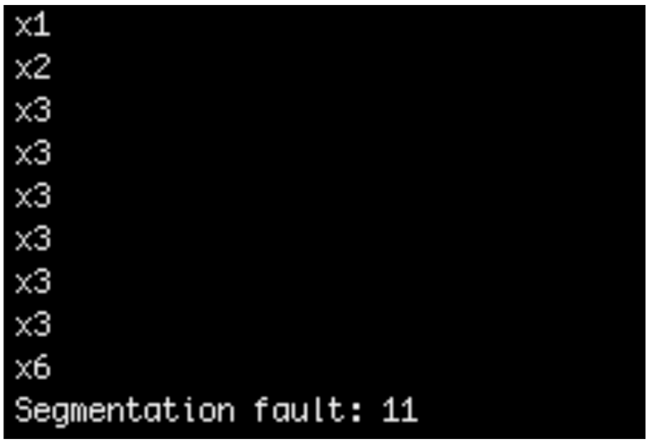

```
x1
x2
x3
x3
x3
x3
x3
x3
x6
Segmentation fault: 11
```

Figure 28

As can be in Figure 28, the segmentation fault happens after “x6” is printed to the console. The “x7” statement has not been printed to the console, hence it can be concluded that there must be an error in line 185 of the source code.

It is likely, that one of the accessed variables or indices within a vector are not existing or defined, causing this memory error. Proceeding similarly and outputting the variables separately, as well as the vector sizes, quickly pinpoint the erroneous statement in the source code.

To ensure potential bugs are found in the first place, it needs to be ensured that the tests are as representative as possible and cover ideally all the main functionality.

10. Project Planning

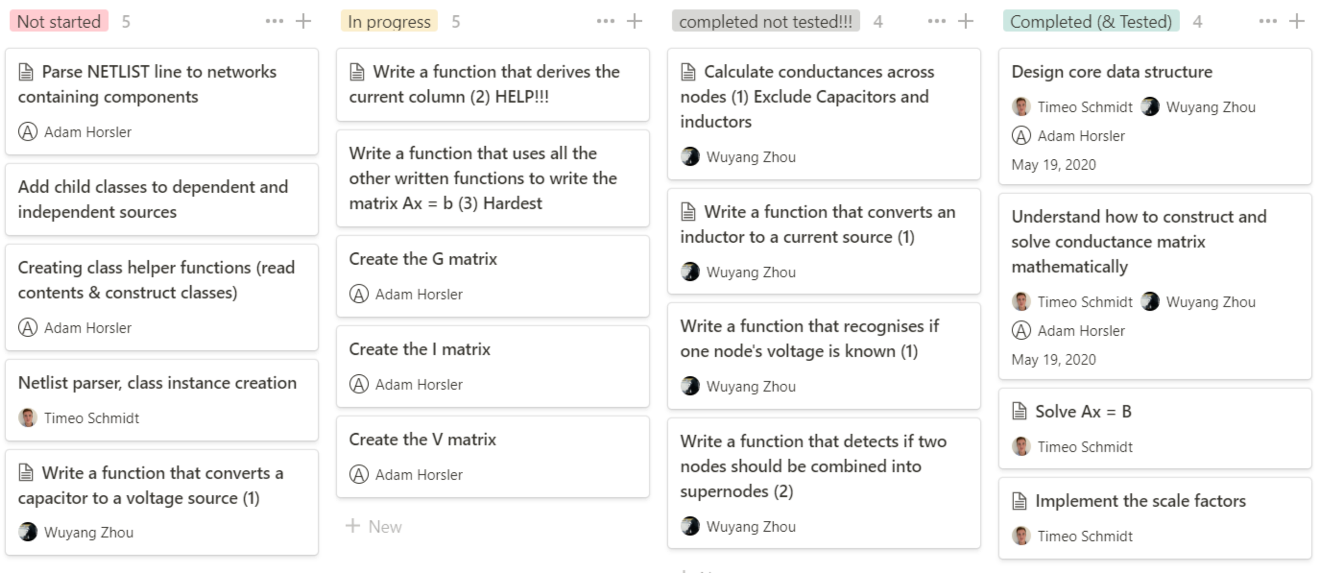
10.1 Overview

A Gantt chart, found in Appendix B, was used for the overarching plan of the project from start to finish. Notion was then used for the more in-depth technical planning. The aim of using these tools was to keep track of the tasks assigned to each member as well as the milestones achieved in the project. A ‘milestone’ in this case is only achieved once sufficient testing has been carried out. The Notion page link can be found in Appendix C.

Replit is an online Integrated Development Environment (IDE) that allows code to be written and compiled. It allows multiple users to edit the same files in real time. It can also be used in conjunction with GitHub to pull and push code to the team repository. This was used during meetings to illustrate ideas. A link to the Replit page is available in Appendix D.

10.2 Meeting Structure

Meetings were not organised to a specific time or day, and instead would occur on an ad-hock basis. This allowed for greater flexibility among team members and was deemed to be more efficient as meetings would revolve around a specific task or problem. At the end of every meeting, the Notion task tracker would be updated to reflect the contents of the meeting and assign tasks to be completed before next meeting. A screenshot of the tasks page is shown in Figure 29:



Not started 5

- Parse NETLIST line to networks containing components — Adam Horsler
- Add child classes to dependent and independent sources
- Creating class helper functions (read contents & construct classes) — Adam Horsler
- Netlist parser, class instance creation — Timeo Schmidt
- Write a function that converts a capacitor to a voltage source (1) — Wuyang Zhou

In progress 5

- Write a function that derives the current column (2) HELP!!!
- Write a function that uses all the other written functions to write the matrix Ax = b (3) Hardest
- Create the G matrix — Adam Horsler
- Create the I matrix — Adam Horsler
- Create the V matrix — Adam Horsler

+ New

completed not tested!!! 4

- Calculate conductances across nodes (1) Exclude Capacitors and inductors — Wuyang Zhou
- Write a function that converts an inductor to a current source (1) — Wuyang Zhou
- Write a function that recognises if one node's voltage is known (1) — Wuyang Zhou
- Write a function that detects if two nodes should be combined into supernodes (2) — Wuyang Zhou

Completed (& Tested) 4

- Design core data structure — Timeo Schmidt, Wuyang Zhou, Adam Horsler — May 19, 2020
- Understand how to construct and solve conductance matrix mathematically — Timeo Schmidt, Wuyang Zhou, Adam Horsler — May 19, 2020
- Solve Ax = B — Timeo Schmidt
- Implement the scale factors — Timeo Schmidt

Figure 29

The benefits of Notion when monitoring tasks include:

- Real-time editing between all members (like OneDrive and Google Docs).
- Can assign members to tasks – adds accountability to the task.
- Easy ordering of the progress of each task, shown in the highlighted colours.

11. Conclusion

11.1 Overview

The software package successfully calculates transient simulations of circuits containing the linear components resistors, capacitors and inductors. With the help of helper functions, the data structure is utilised to construct the G, V and I matrices using the Eigen library. The energy consumption of the simulator increases exponentially with an increase in the number of diodes in the circuit. Although non-linear components were not implemented, a potential method of implementation is described in Section 11.3.

11.2 Reflection

Upon reflecting on the project, the following lessons were learned:

- Tracking of code that was written but not tested as well as code that had been tested began halfway through the project. Before tracking, it was difficult to remember which code had or had not been tested. Thus, once tracking began it sped up meetings as it was no longer dependent on verbal communication and was instead shown in the Notion task chart.
- Although using GitHub was a great asset, the project could have benefited from using more of its features, especially branches (in addition to the master branch). Branches isolate development work without affecting other branches in the repository. The benefits of branches include:
    - Develop features
    - Fix bugs
    - Safely experiment with new ideas.

  At one point, the project had to be reverted to a previous commit due to conflicting code that was hard to track. Branches could have prevented this as they can easily be deleted or altered without affecting the master branch.
- When writing code, and for future reference, the team could have benefited from using comments more extensively. This would help other team members who did not contribute to a part familiarise themselves with how the code works faster, as oppose to trying to understand the code on its own. The comments would also help populate the task tracker, for example commenting when a function had been tested. When used in conjunction with GitHub, this would provide a history of information which is an additional benefit.

At the start of the project, all team members were working on the data structure as this would influence how the rest of the program operates. After this was completed and agreed upon, the next tasks were split up according to the Notion task tracker constructed from meetings. If there needed to be a change to the data structure, this would be discussed and implemented in a team meeting as it would likely affect everyone's task.

11.3 Future work

The following points are suggestions of future work:

- Instantiate the Eigen matrices with the implementation that they are stored in cache as row-major matrices, or similarly traverse the matrices by column rather than row. This would likely have a slight benefit to the compilation time due to better usage of data locality.
- Using pointers. If pointers are used, the repetitive creation of nodes and components can be avoided because pointers do not disappear unless they are deleted. Using pointers will make the code look much cleaner and might build up a better foundation for implementing non-linear components with three connecting nodes like BJTs or MOSFETs.
- Implementation of non-linear components.
- **Simulating diodes:**
  The implementation of semiconductor devices, such as diodes requires a more complex procedure, as the diode has got nonlinear IV-characteristics. To prepare the future implementation of diodes in our circuit simulator, this section outlines the mathematical algorithm, which then only needs to be translated into code.
  To illustrate the method, a simple example will be used.

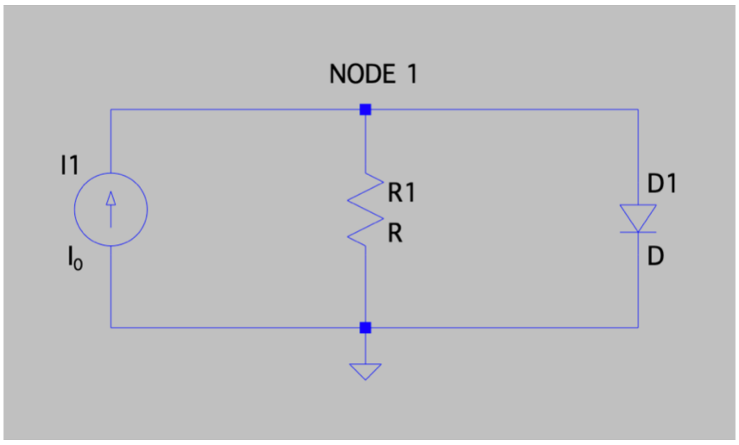


Figure 30

This example circuit contains a diode, and has got an unknown voltage at NODE1, which is to be calculated. In matrix form, this calculation simply is:

$$[G][V_1] = [I_0]$$

The Shockley diode equation, which models the current through a diode, given a voltage across it, is defined as:

$$I_d = I_S(e^{\frac{V_d}{V_T}} - 1)$$

Therefore, KCL at NODE1, can simply be expressed as:

$$0 = \frac{V_1}{R} + I_S(e^{\frac{V_d}{V_T}} - 1) - I_0 \qquad \text{(eq. 1)}$$

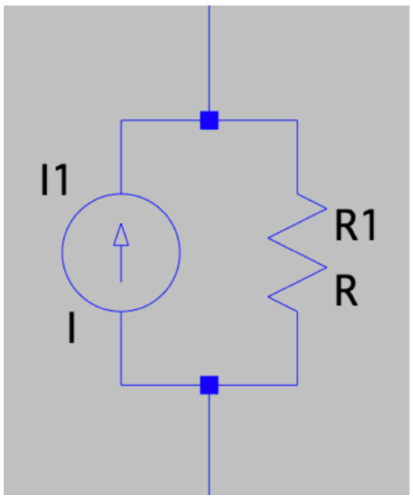


Figure 31

The end goal is to calculate a linear equivalent for the diode at specific operating conditions, that are consistent with the rest of the circuit. At this operating point, the diode can be modelled as a DC current source in parallel with a resistor. This is essential a Norton equivalent source of the diode, at an operating point of the IV-slope given by the Shockley equation.

The goal now is to determine the operating point of the diode. There is no direct calculation algebraic calculation possible, as the diode and circuit are inter-dependent. Therefore, a numerical approach is used to calculate the Norton equivalent source of the diode. The algorithm used for this is known as the “Newton-Raphson” method.

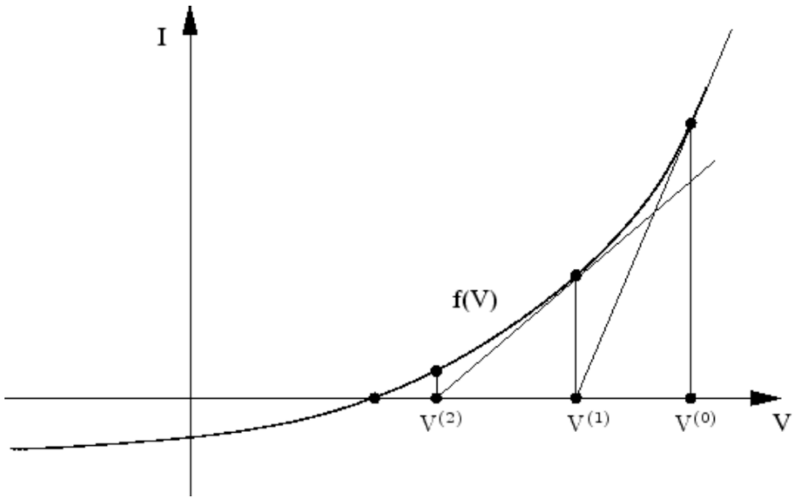


Figure 32 [13]

In Figure 32, the numerical method is visualised. The curve represents the IV-characteristic of the diode. The different tangents that are drawn correspond to the specific Norton-equivalent source.

The Newton-Raphson method works in the following way: First, a guess at an initial operating voltage is made. The algorithm then states, that the solution of a function's derivative at a given operating point is closer to the actual solution than the original point. This means, that by repeatedly differentiating eq. 1, the tangent would progressively converge to the actual solution. This can be seen in the above figure, where an initial guess is made at V(0), which then moves to v(1) and V(2) as the function gets differentiated. This process is repeated until the changes between iteration get reduced to a given threshold. The slope then represents the values of the equivalent Norton source.

[13] Qucs.sourceforge.net. 2020. *Non-Linear DC Analysis*. [online] Available at: <http://qucs.sourceforge.net/tech/node16.html> [Accessed 14 June 2020].

12. References

[1]"Regular expressions library - cppreference.com", En.cppreference.com, 2020. [Online]. Available: https://en.cppreference.com/w/cpp/regex. [Accessed: 18- May- 2020].

[2]"Regular expressions library - cppreference.com", En.cppreference.com, 2020. [Online]. Available: https://en.cppreference.com/w/cpp/regex. [Accessed: 18- May- 2020].

[3]"regex_match - C++ Reference", Cplusplus.com, 2020. [Online]. Available: http://www.cplusplus.com/reference/regex/regex_match/. [Accessed: 20- May- 2020].

[4]"regex_search - C++ Reference", Cplusplus.com, 2020. [Online]. Available: http://www.cplusplus.com/reference/regex/regex_search/. [Accessed: 20- May- 2020].

[5]"Armadillo: C++ library for linear algebra & scientific computing", Arma.sourceforge.net, 2020. [Online]. Available: http://arma.sourceforge.net/. [Accessed: 25- May- 2020].

[6]"Eigen", Eigen.tuxfamily.org, 2020. [Online]. Available: http://eigen.tuxfamily.org/index.php?title=Main_Page. [Accessed: 25- May- 2020].

[7]"GSL - GNU Scientific Library - GNU Project - Free Software Foundation", Gnu.org, 2020. [Online]. Available: https://www.gnu.org/software/gsl/. [Accessed: 25- May- 2020].

[8]R. Poya, "A look at the performance of expression templates in C++: Eigen vs Blaze vs Fastor vs Armadillo vs…", Medium, 2020. [Online]. Available: https://medium.com/@romanpoya/a-look-at-the-performance-of-expression-templates-in-c-eigen-vs-blaze-vs-fastor-vs-armadillo-vs-2474ed38d982. [Accessed: 26- May- 2020].

[9]"Eigen: Storage orders", Eigen.tuxfamily.org, 2020. [Online]. Available: https://eigen.tuxfamily.org/dox/group__TopicStorageOrders.html. [Accessed: 5- Jun- 2020].

[10]RegExr. 2020. Regexr: Learn, Build, & Test Regex. [online] Available at: <https://regexr.com> [Accessed 14 June 2020].

[11]"Intel® Core™ i9-9880H Processor (16M Cache, up to 4.80 GHz) Product Specifications", Ark.intel.com, 2020. [Online]. Available: https://ark.intel.com/content/www/us/en/ark/products/192987/intel-core-i9-9880h-processor-16m-cache-up-to-4-80-ghz.html. [Accessed: 9- Jun- 2020].

[12]C. Angelini and I. Wallossek, "Intel Core i7-5960X, -5930K And -5820K CPU Review: Haswell-E Rises", Tom's Hardware, 2014. [Online]. Available: https://www.tomshardware.com/reviews/intel-core-i7-5960x-haswell-e-cpu,3918-13.html. [Accessed: 10- Jun- 2020].

[13] Qucs.sourceforge.net. 2020. Non-Linear DC Analysis. [online] Available at: <http://qucs.sourceforge.net/tech/node16.html> [Accessed 14 June 2020].

13. Appendix

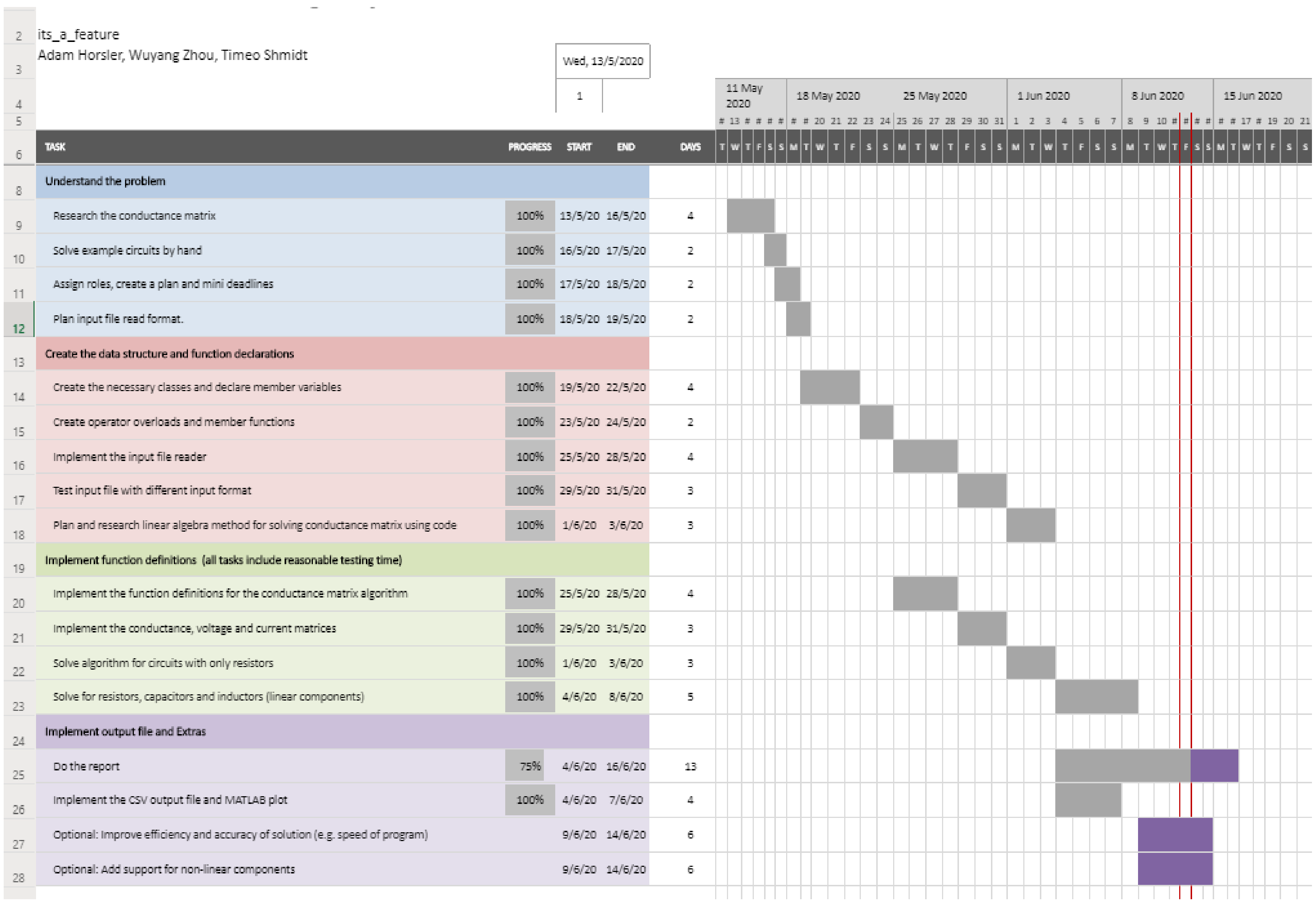
13.1 Appendix A: Software Requirements Specification

Everyone in Imperial College

The link to the document:

https://imperiallondon-my.sharepoint.com/:w:/g/personal/ah2719_ic_ac_uk/EXpbn2_v_VFMsYMW2yPK_SsB9XFVXHvSBTPkmyfEqbJQyw?e=fVIp0l

13.2 Appendix B: Gantt Chart



its_a_feature
Adam Horsler, Wuyang Zhou, Timeo Shmidt

Wed, 13/5/2020
1

TASK	PROGRESS	START	END	DAYS
Understand the problem				
Research the conductance matrix	100%	13/5/20	16/5/20	4
Solve example circuits by hand	100%	16/5/20	17/5/20	2
Assign roles, create a plan and mini deadlines	100%	17/5/20	18/5/20	2
Plan input file read format.	100%	18/5/20	19/5/20	2
Create the data structure and function declarations				
Create the necessary classes and declare member variables	100%	19/5/20	22/5/20	4
Create operator overloads and member functions	100%	23/5/20	24/5/20	2
Implement the input file reader	100%	25/5/20	28/5/20	4
Test input file with different input format	100%	29/5/20	31/5/20	3
Plan and research linear algebra method for solving conductance matrix using code	100%	1/6/20	3/6/20	3
Implement function definitions (all tasks include reasonable testing time)				
Implement the function definitions for the conductance matrix algorithm	100%	25/5/20	28/5/20	4
Implement the conductance, voltage and current matrices	100%	29/5/20	31/5/20	3
Solve algorithm for circuits with only resistors	100%	1/6/20	3/6/20	3
Solve for resistors, capacitors and inductors (linear components)	100%	4/6/20	8/6/20	5
Implement output file and Extras				
Do the report	75%	4/6/20	16/6/20	13
Implement the CSV output file and MATLAB plot	100%	4/6/20	7/6/20	4
Optional: Improve efficiency and accuracy of solution (e.g. speed of program)		9/6/20	14/6/20	6
Optional: Add support for non-linear components		9/6/20	14/6/20	6

13.3 Appendix C: Notion

https://www.notion.so/9f324c961e1748b3ac5cc1d174a3c94d?v=9b97d3e983d746a286bfd85c2b1fdb8e

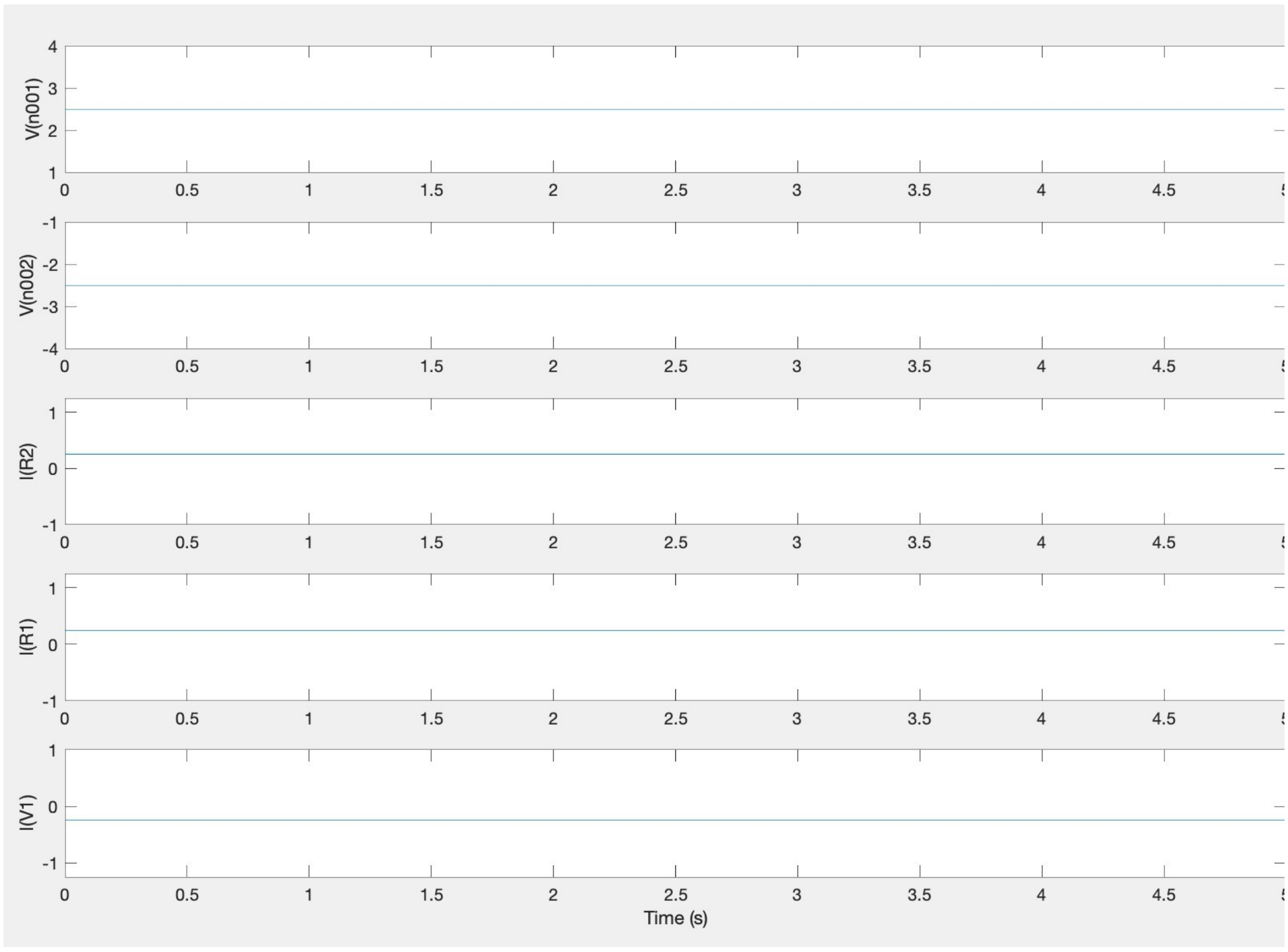
13.4 Appendix D: Replit

https://repl.it/@timeoschmidt1/yearoneproject

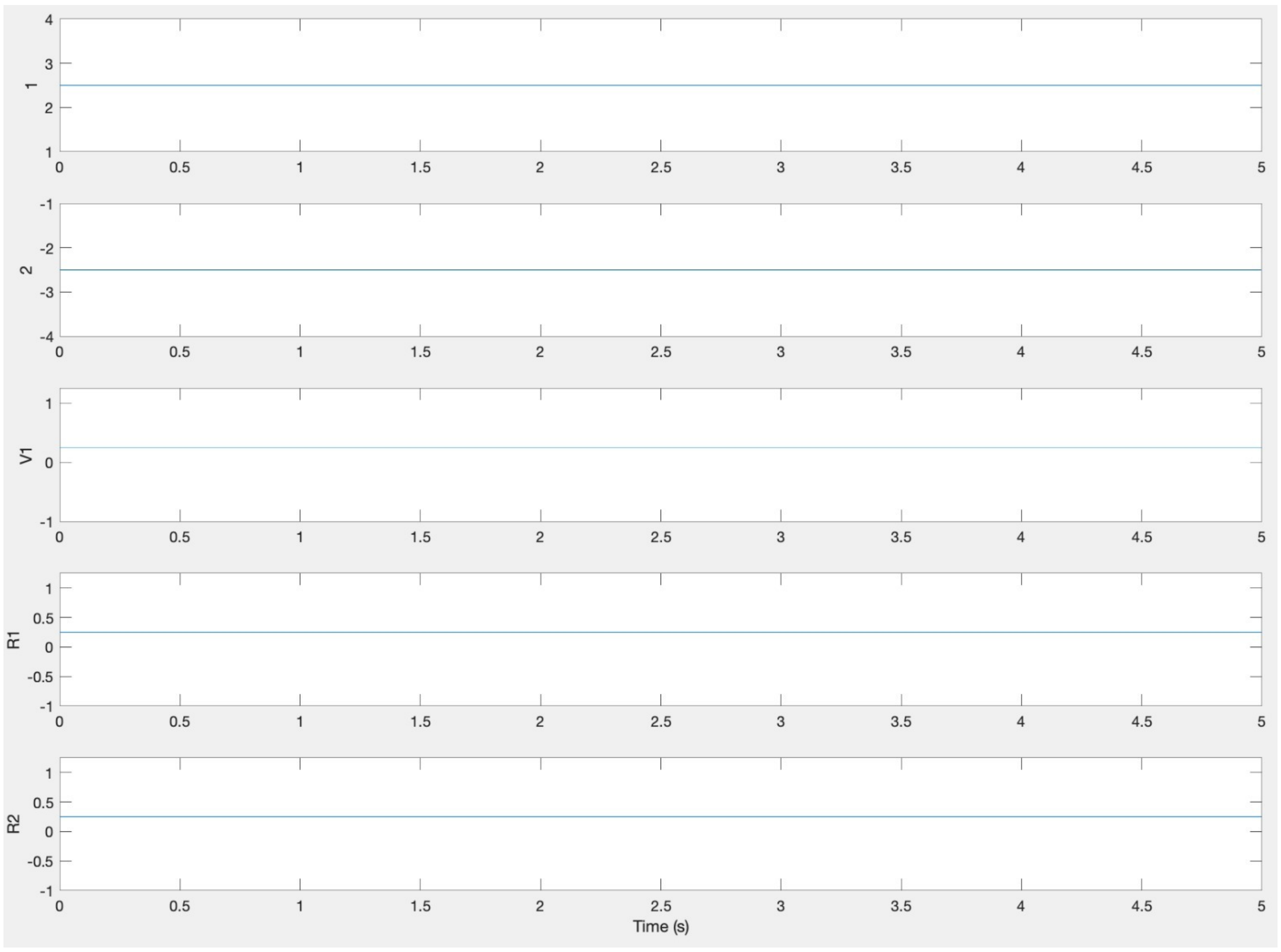
13.5 Appendix E: GitHub

https://github.com/timeo-schmidt/year_one_project

13.6 Appendix F: Supernode Simulations

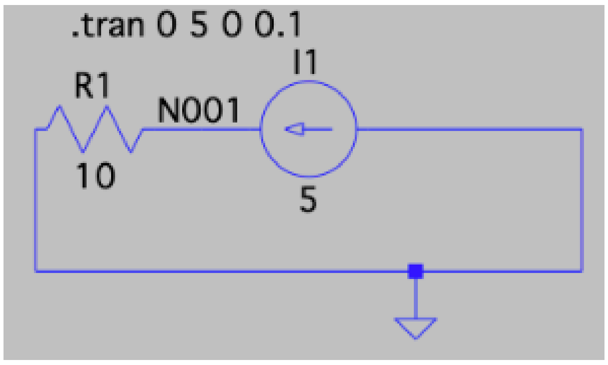


LTSpice Simulation of Supernode Circuit

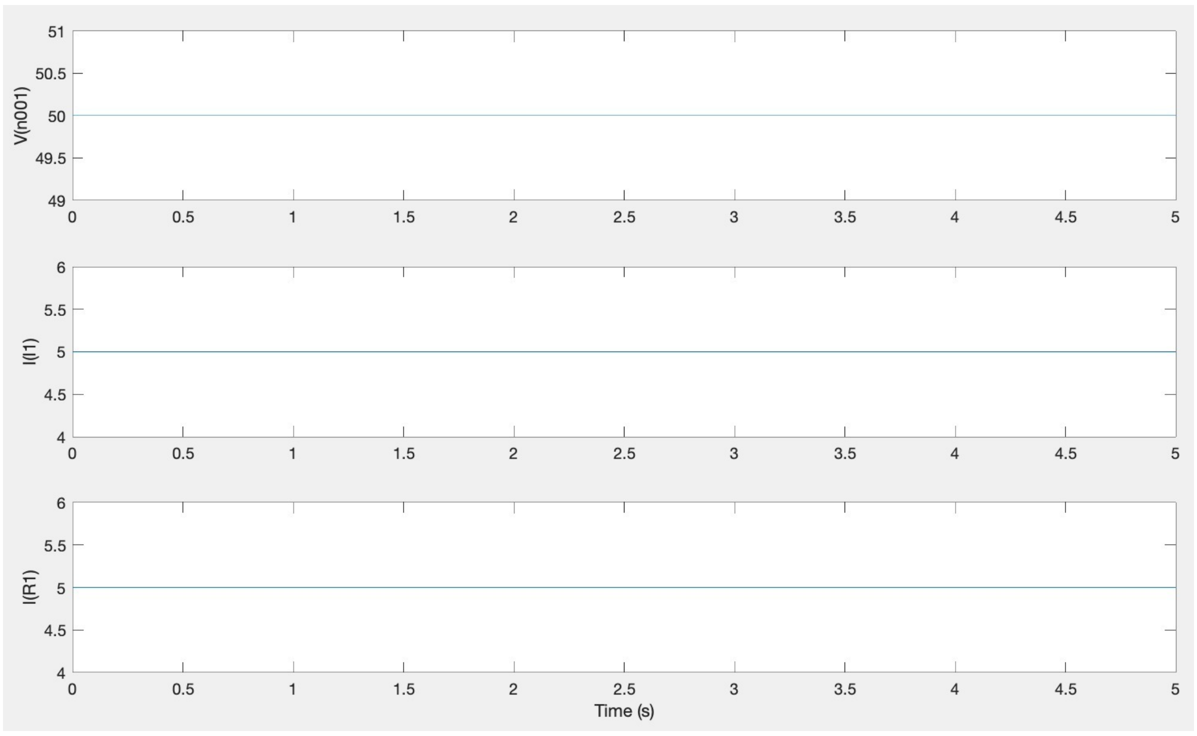


Software Package Simulation of Supernode Circuit

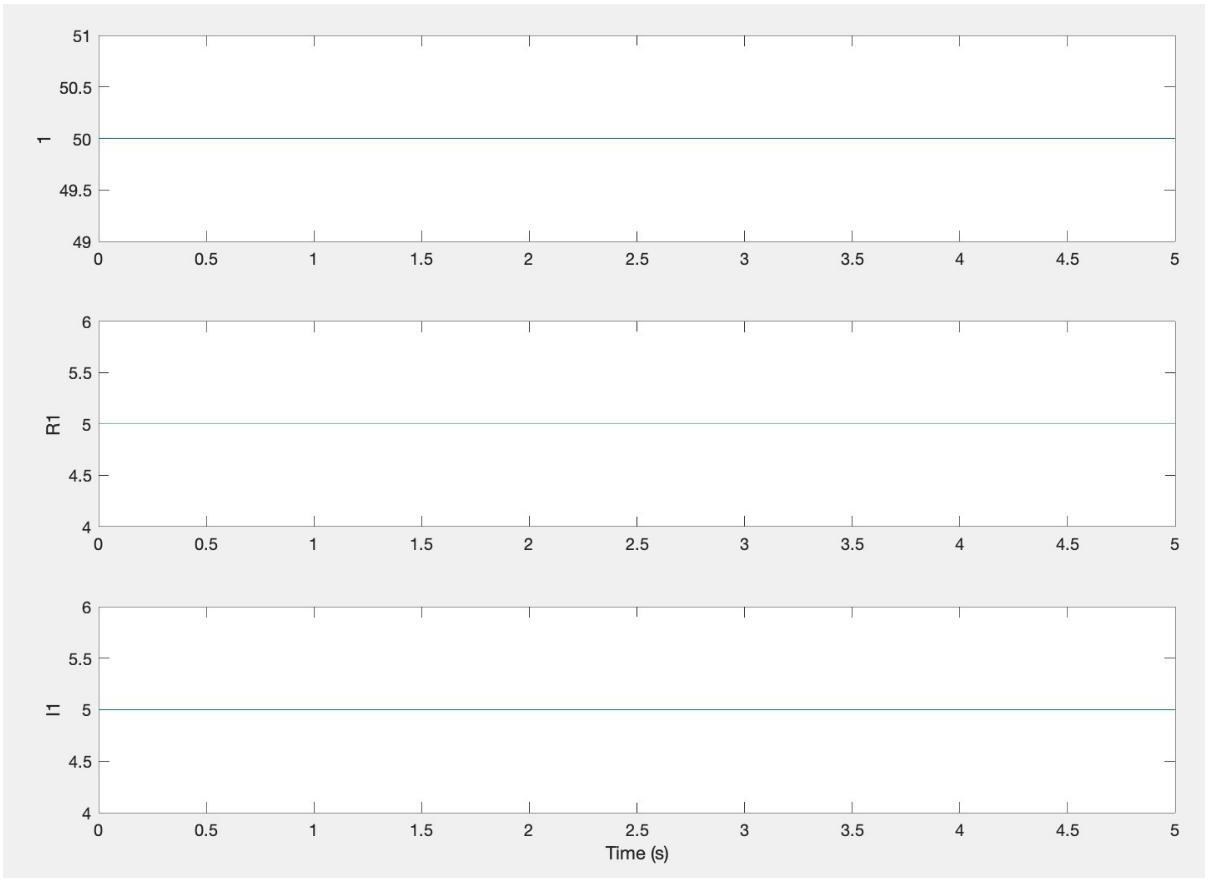
13.7 Appendix G: Current Source Simulations



Circuit with a Current Source

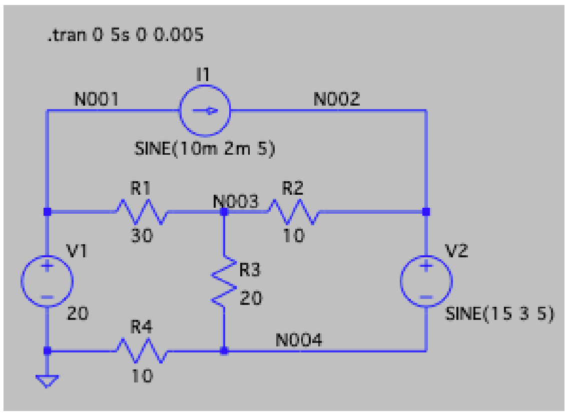


LTSpice Simulation of Circuit with a Current Source

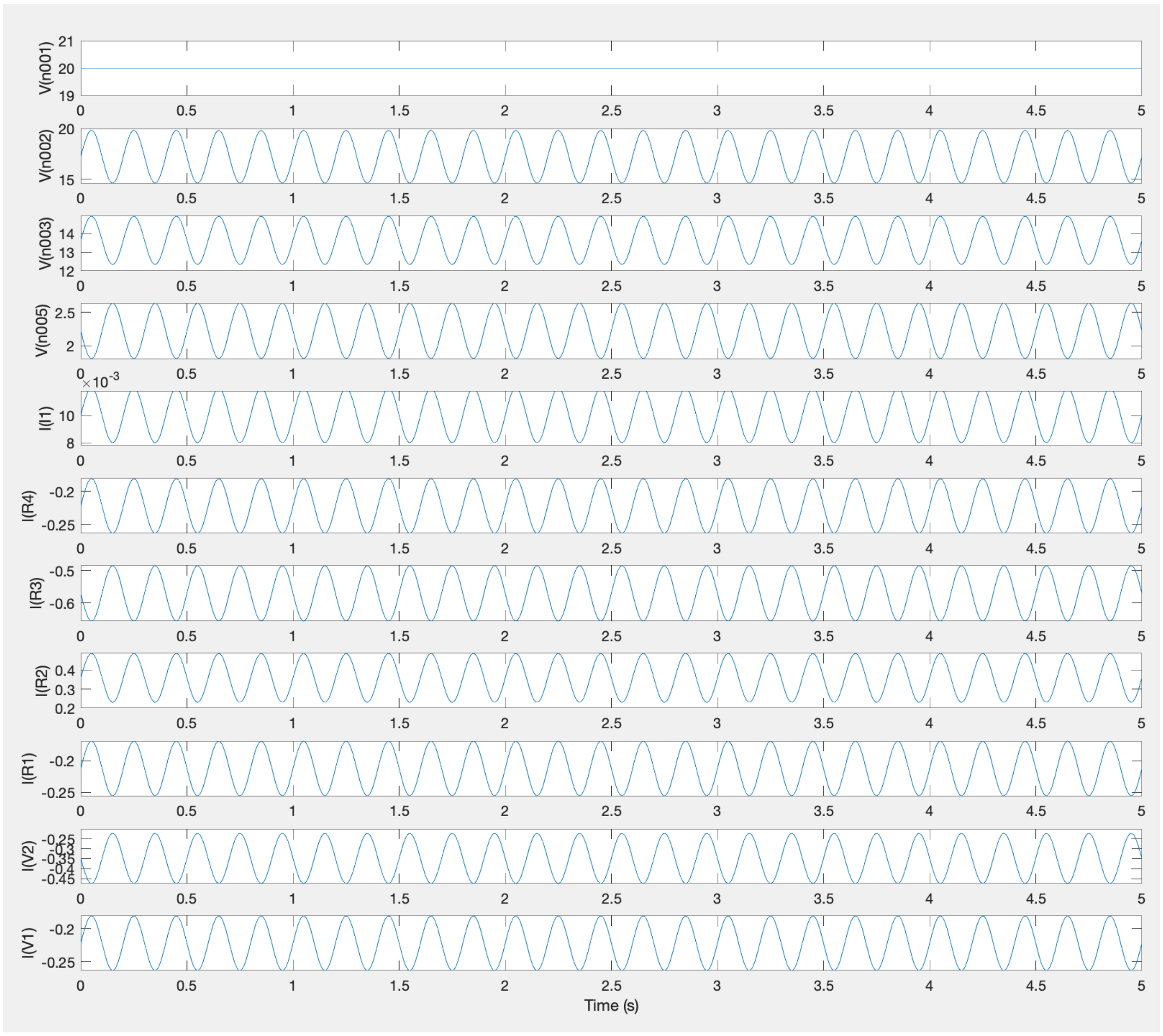


Software Package Simulation of Circuit with a Current Source

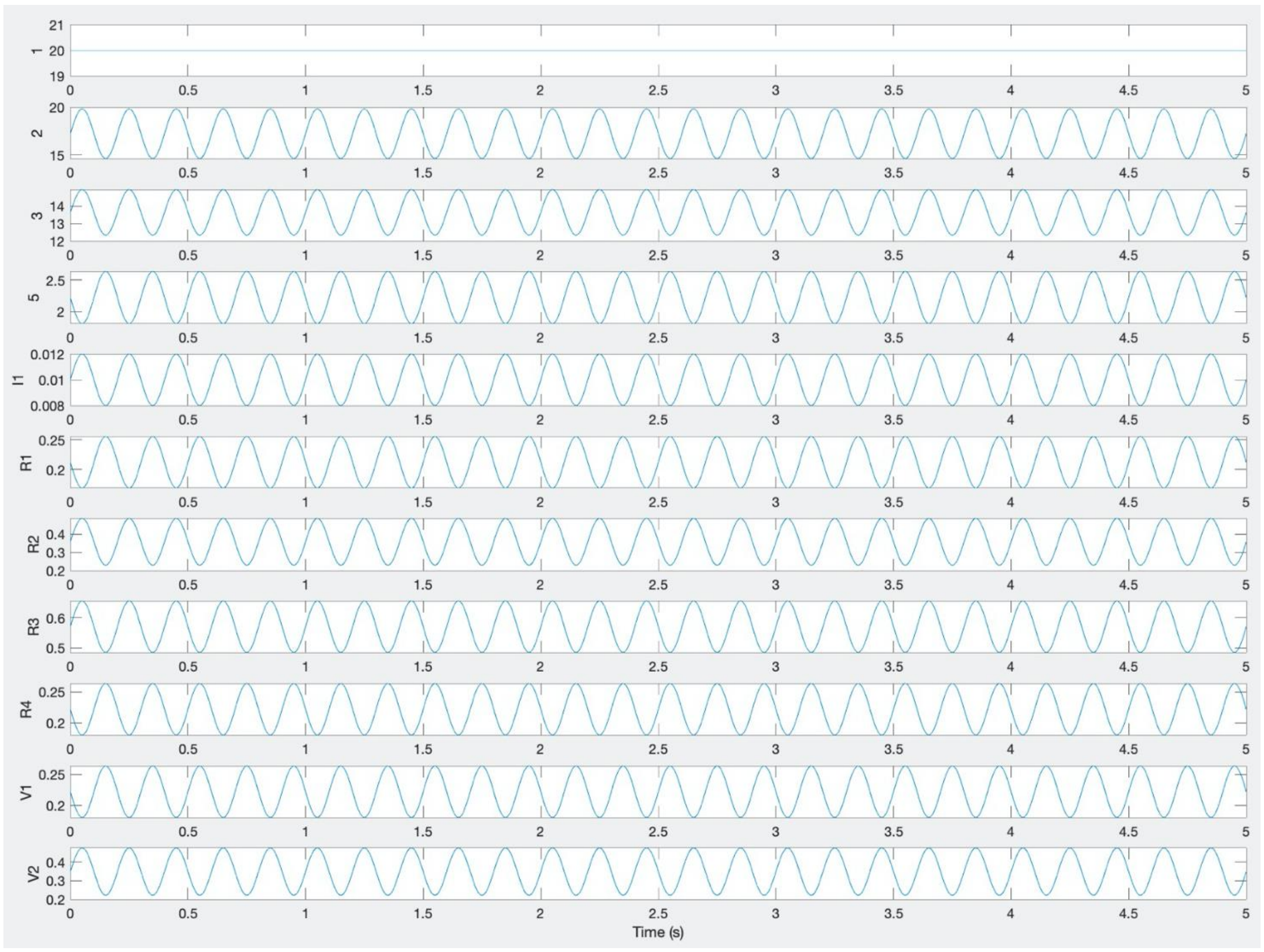
13.8 Appendix H: Current and Voltage Sources



Complicated Circuit

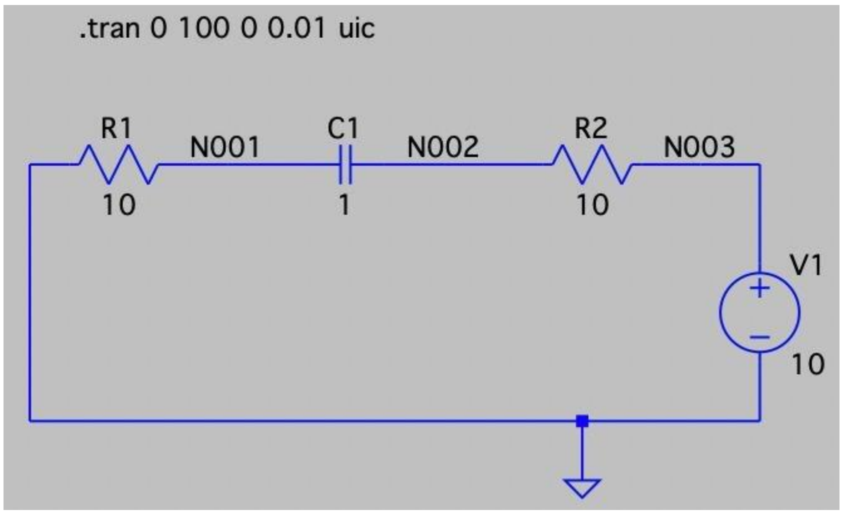


LTSpice Simulation of the Complicated Circuit

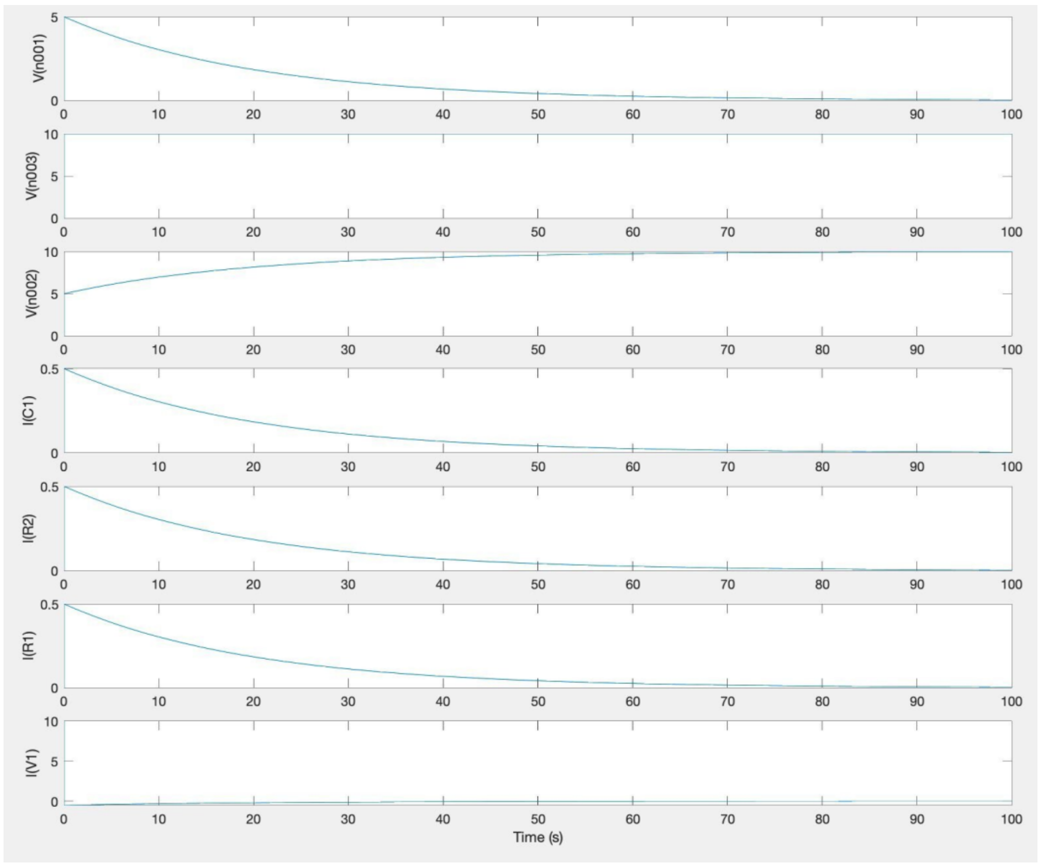


Software Package Simulation of the Complicated Circuit

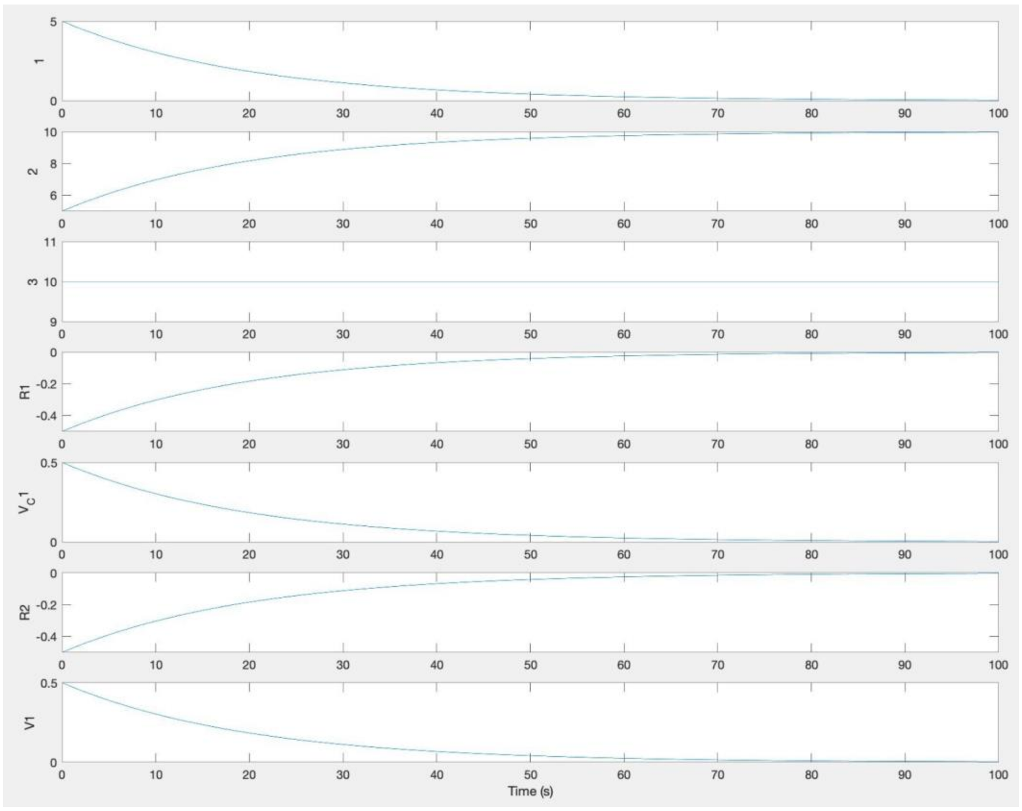
13.9 Appendix I: Capacitor Test Simulation



Circuit with a Capacitor

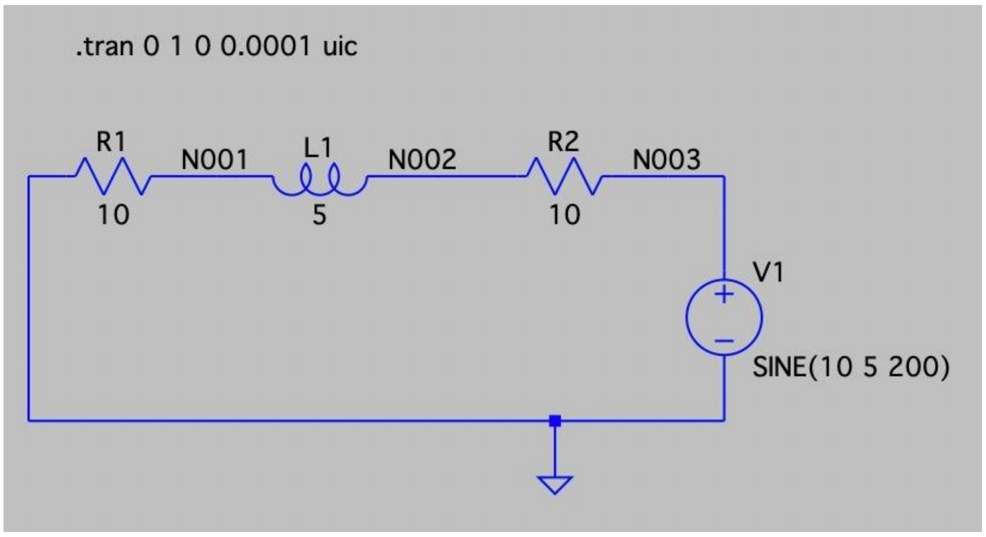


LTSpice Simulation of Circuit with a Capacitor

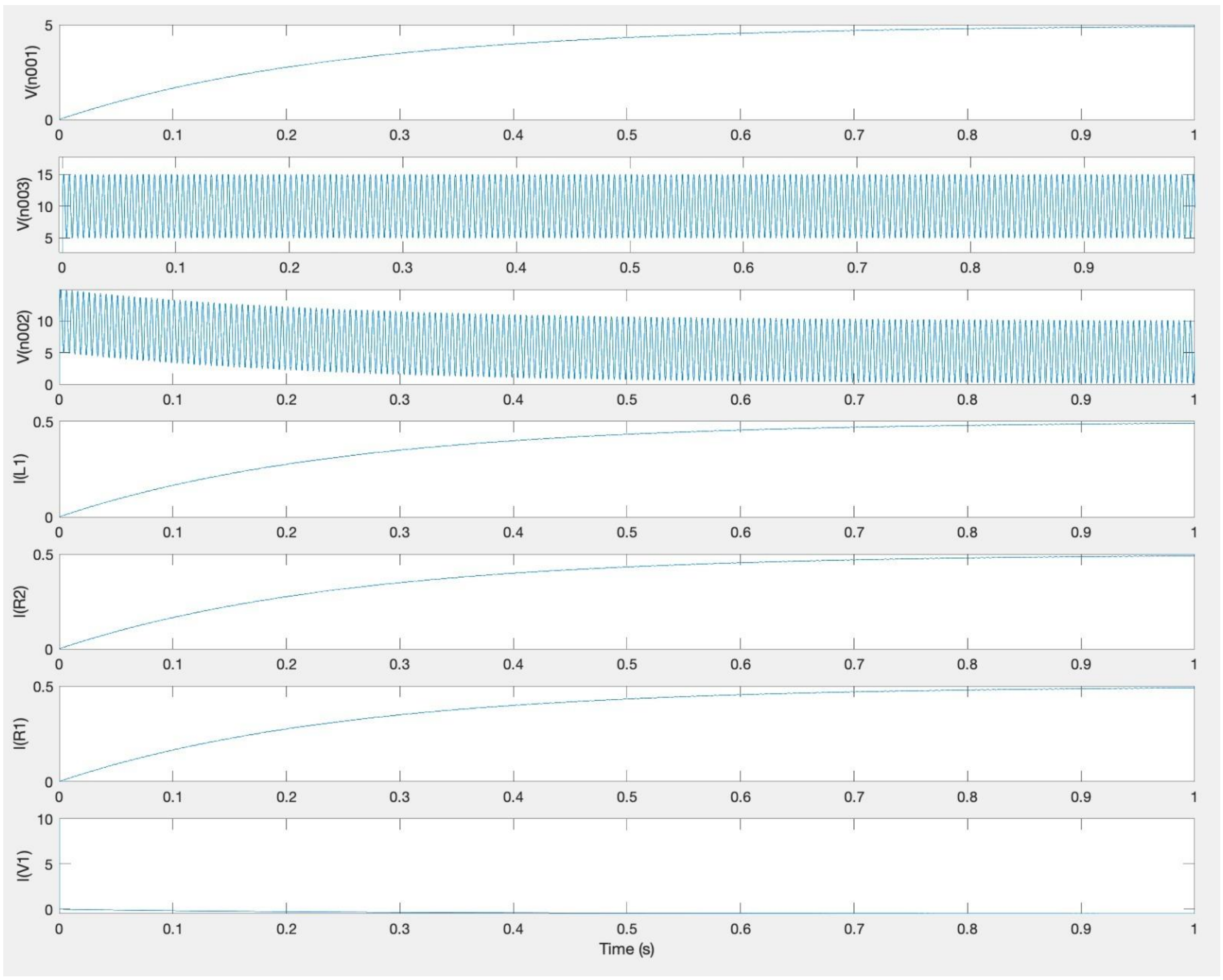


Software Package Simulation of Circuit with a Capacitor

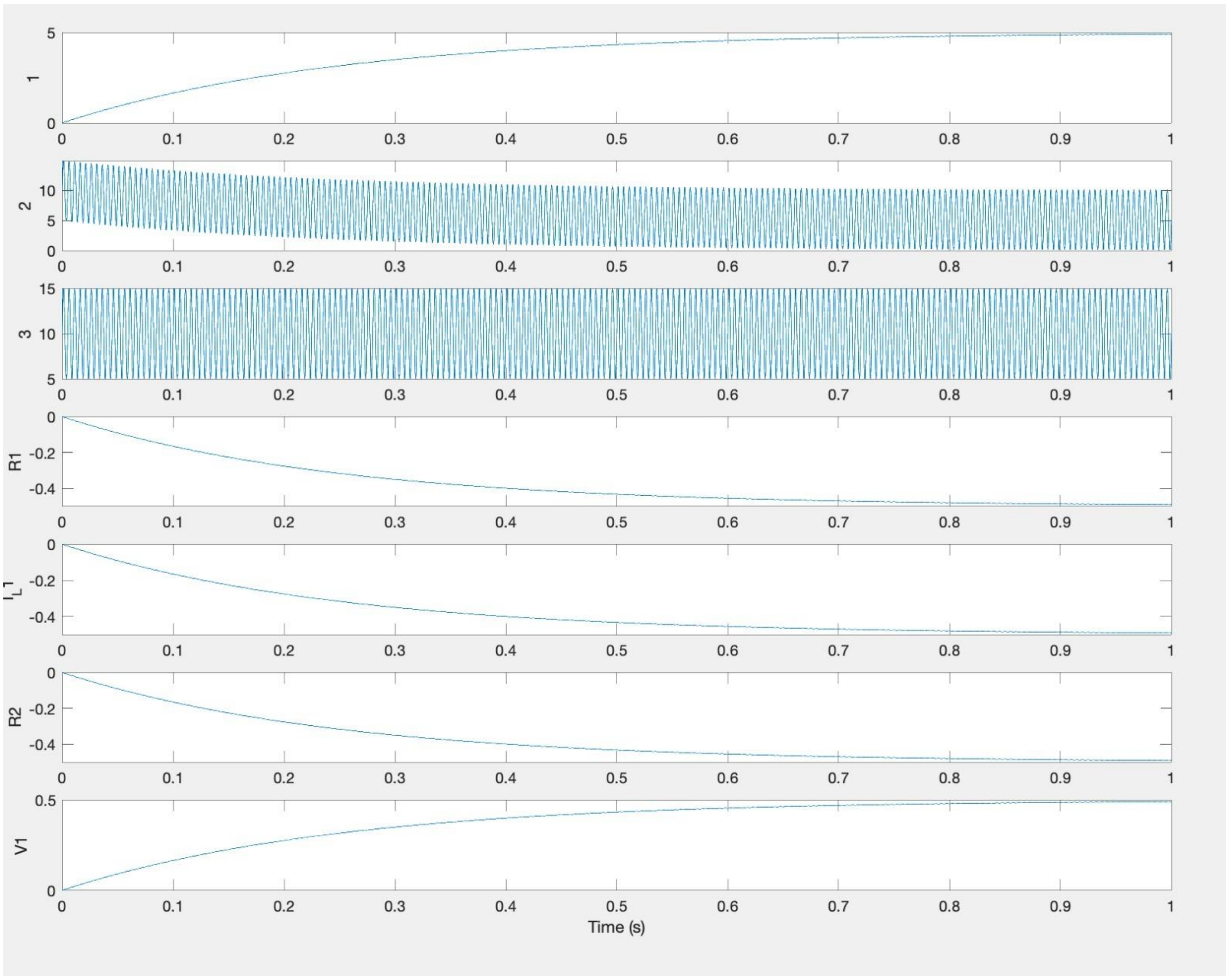
13.10 Appendix J: Inductor Test Simulation



Circuit with an Inductor

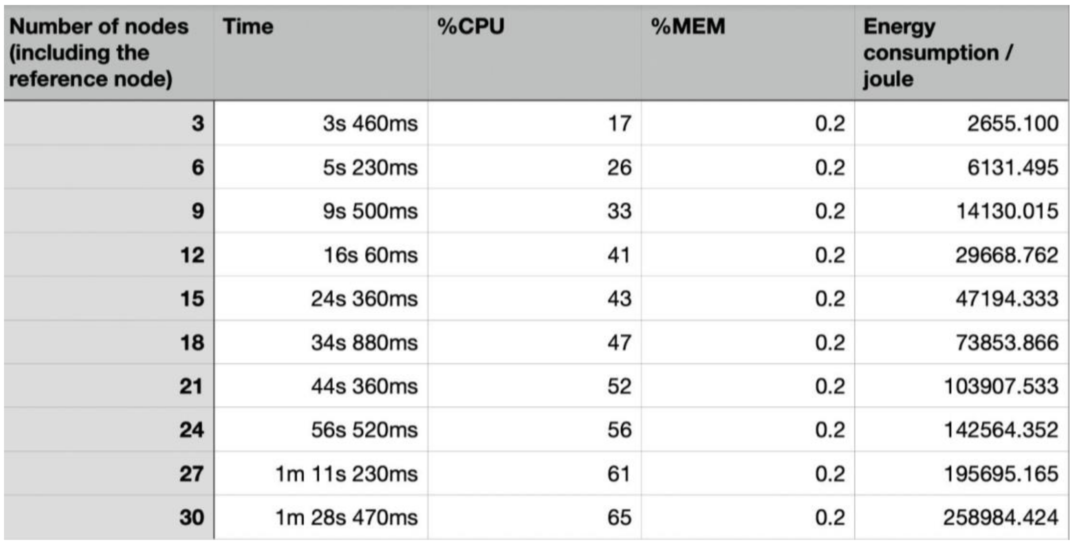


LTSpice Simulation of Circuit with an Inductor



Software Package Simulation of a Circuit with an Inductor

13.11 Appendix J: Efficiency and Performance Data



Number of nodes (including the reference node)	Time	%CPU	%MEM	Energy consumption / joule
3	3s 460ms	17	0.2	2655.100
6	5s 230ms	26	0.2	6131.495
9	9s 500ms	33	0.2	14130.015
12	16s 60ms	41	0.2	29668.762
15	24s 360ms	43	0.2	47194.333
18	34s 880ms	47	0.2	73853.866
21	44s 360ms	52	0.2	103907.533
24	56s 520ms	56	0.2	142564.352
27	1m 11s 230ms	61	0.2	195695.165
30	1m 28s 470ms	65	0.2	258984.424

Table comparing performance and efficiency to node number