Imperial College
London

Circuit Simulator Report

Electronics Design Project 1 - ELEC40006
Lecturers: Dr Stott and Mrs Perea

Group - “Error 404”

Samuel Adekunle	01737380
Neel Dugar	01722569
Jonah Lehner	01704401

Word Count: 10520

0. Table of Contents

0. Table of Contents 2

Abstract 5

Introduction 5

2. Technical Requirements of Project 6

2.1 Parsing SPICE Netlist 6

2.1.1 Abstracting Electrical Circuits in C++ 6

2.1.2 Compliance with SPICE netlist format 6

2.2 Modelling Linear Components 6

2.3 Solving Linear Equations of the form “G*V=I” 6

2.4 Modelling Nonlinear Components 6

3. Design Criteria (SRS) 7

3.1 Introduction to SRS 7

3.1.1 Purpose 7

3.1.2 Intended Audience and Use 7

3.1.3 Scope 7

3.2 Overall Description of Product 7

3.2.1 User Needs 7

3.2.2 Assumptions and Dependencies 7

3.3 System Features and Requirements 8

3.3.1 Functional Requirements 8

3.3.2 Nonfunctional requirements 8

3.3.3 External Interface Features 8

4. Design Process 9

4.1 Project Planning and Structure 9

4.1.1 CMake Build System and Folder Structure 9

4.1.2 Documentation 9

4.2 Continuous Review of Data Structures for Circuit Abstractions 9

4.2.1 Initial Rough Design 9

4.2.2 Refactored Code Design 10

4.3 Solving Linear Equations of form G * V = I” 11

4.3.1 SymbolicC++ 11

4.3.2 Eigen C++ 11

4.3.3 Comparing Solvers 11

4.4 Multiple Attempts at Solving Nonlinear Equations 12

4.5 Graphing Tool 12

4.6 User Experience 13

5. Detailed Technical Description of API **14**
5.1 Circuit Abstraction 14
5.2 Circuit Math 14
5.2.1 Handling Voltage Sources 15
5.3 Netlist Parser and Circuit::Simulator 16
5.4 Linear Components Modelling 17
5.5 Source Modelling 19
5.6 Nonlinear Components Modelling 20
5.6.1 Diode Parasitic Capacitance Modelling 20

6. Demonstration Circuits and Evaluation of Simulator **22**
6.1 Evaluation and Benchmarking Process 22
6.2 Effect of number of nodes on runtime (Time Complexity) 22
6.3 Circuit Component Tests 23
6.3.0 External Interactive Graphs and Netlists 23
6.3.1 Test 1 - Simple RC Charging network 24
6.3.2 Test 2 - Single Diode Circuit 27
6.3.3 Test 3 - Two Diode Crossover Distortion 30
6.3.4 Test 4 - Full Wave Rectifier Circuit 33
6.3.5 Test 5 - Parasitic Capacitance Properties 35
6.3.6 Test 6 - LCR Multiple Phase Circuit 37

7. Project Planning and Management **40**
7.1 Trello Boards 40
7.2 Code Collaboration 40
7.3 Communication Channels 40

8. Conclusion and Future Extensions **41**
8.1 Critical Analysis and Reflection 41

9. Appendix **43**
9.1 Trello Board 43
9.2 Netlists Used for Testing and Evaluation of Simulator. 43
9.2.1 Simple RC Test 43
9.2.2 Single Diode Test 43
9.2.3 Double Diode Cross Over Distortion Test 44
9.2.4 Full Wave Rectifier 44
9.2.5 Parasitic Capacitance Properties 44
9.2.6 LCR Multiple Phase Circuit 44
9.3 List of Accepted Commands and Components 45
9.3.1 Accepted Components 45
9.3.2 Accepted Prefixes (to use with values) 45
9.3.3 Accepted Commands 46
9.4 Circuit Simulator Manual 46

9.5 Valgrind Output 46
9.6 Diode Parameters 47

10. References 48

Abstract

This project aims to produce a command-line interface program, which fully implements the functionality of circuit simulators like LTspice. While mimicking this software, the project also seeks to address and fix some of the issues faced while using GUI simulators such as poor cross-platform support and poor control of simulator output.

Introduction

The choice to embark on this project was motivated by an interest to develop a deeper understanding of the working of circuit simulators. The team also thought that, out of all the tasks available to pick from, the circuit simulator had the most significant technical challenges. The team would have to apply knowledge gained from the programming module (ELEC40004), using C++ as the primary programming language, the Analysis and Design of Circuits module (ELEC40002) and linear algebra from the Mathematics module (ELEC40001).

The project is entirely compliant with the SPICE netlist format. This allows for easy corroboration of this project's results with other simulators. The output format is a delimiter separated list of values but, unlike LTspice, the delimiter can be altered by the user. The team also designed an internal graphing tool, which creates interactive and flexible graphs with more features than currently found in other circuit simulators. The implementation of a standard output format like comma-separated values (CSV) means that output from this program can easily be used in other graphing software such as MATLAB.

This report begins with a detailed technical analysis of the problems trying to be solved, followed immediately by a section outlining design criteria for the project in detail. Proceeding that, the report explains the design process followed while developing the simulator and gives a quick technical overview of the API developed. Finally, the simulator was put through a series of evaluation tests to measure its performance and check for correct operation, by comparing its result with those obtained by LTspice.

2. Technical Requirements of Project

2.1 Parsing SPICE Netlist

2.1.1 Abstracting Electrical Circuits in C++

Before actually parsing the SPICE Netlist, an abstraction of the circuit was needed which met the task's requirements:

- Convert the netlist to the internal object instances.
- Generate the circuit matrices.
- Obtain circuit parameters after simulation such as current and voltage

The team decided to go with an object-oriented approach (OOP). OOP allowed the program to abstract each line in the netlist as an object. This method made parsing easier and allowed the storage of member variables and functions in classes.

2.1.2 Compliance with SPICE netlist format

The parser needed to work with a netlist generated by LTspice. The simulator ignored lines of commands that were not actively being used by the simulator. As an extension, the parser added support for multiple simulations of different types on a single netlist. See appendix for a full list of commands and components supported by the simulator.

2.2 Modelling Linear Components

The simulator needs to model the most basic circuit components. The simulator can handle Resistors, Capacitors, Inductors, Voltage Sources and Current Sources. Models that represent the current and conductance behaviour of these components were implemented to provide support for these components.

2.3 Solving Linear Equations of the form "$G * V = I$"

In order to generate the conductance matrix and current vector, the simulator carried out nodal analysis on all the nodes in the circuit and factorised the equation into the required format.

2.4 Modelling Nonlinear Components

The simulator also needed to model more advanced components which do not have linear relationships between current and voltage. These components required implementing iterative algorithms to find solutions.

3. Design Criteria (SRS)

3.1 Introduction to SRS

3.1.1 Purpose

The purpose of the software produced is to simulate circuits with a command-line interface, which shall produce an output of node voltages and component currents. This project should make use of linear algebra in solving the systems of linear equations representing the circuit.

3.1.2 Intended Audience and Use

The primary intended audience should be engineers who want to use this circuit simulator to design actual products. The program should also be accessible to students who want to deepen their understanding of electronics. The simulator should, therefore, be used to model both ideal components and non-ideal components.

3.1.3 Scope

This simulator aims to be a command-line equivalent of the LTspice simulator. The goal is to create a more performant alternative, which can be used from the convenience of the command line. The benefits of using this program should include speed, control of IO and ease of use.

3.2 Overall Description of Product

3.2.1 User Needs

The simulator's target users are engineers. The simulator must parse netlist input and provide output such that simulation results can inform design decisions. The users would probably have used LTspice or some variant beforehand, so the input must be compliant with the standard Spice netlist.

3.2.2 Assumptions and Dependencies

The simulations carried out by the program will assume ideal characteristics of circuit components, e.g. connecting wires have zero resistance. Hence, when the simulator output is compared to results obtained in a lab, they might differ slightly. The results obtained by the simulator are only correct in a theoretical sense. They should not be used as definitive proof of real-life circuit behaviour, but merely as a guide.

All linear algebra manipulations are done via floating-point arithmetic, and any inaccuracies in their implementation may negatively impact the precision of the simulator's results.

3.3 System Features and Requirements

3.3.1 Functional Requirements

- Takes in SPICE compliant netlists as input.
- Correctly generate and solve circuit equations of the form $G \times V = I$
- Outputs circuit values in appropriate formats for further use.
- Display output through graph plotting software

3.3.2 Nonfunctional requirements

- Memory Safety - No memory leaks during program execution.
- Speed - The program should take no more than 1 second to simulate "simple circuits". In this project, a "simple circuit" is defined as one of fewer than 10 components, 10 nodes and 10,000 timesteps. This specification is relatively arbitrary but captures the idea that the users are engineers and students, who do not want to be waiting long periods for their designs to simulate. Ultimately, the program should have simulation speeds which are competitive with LTspice.
- Accuracy and Precision - For this project, the output of the simulator should try to follow, as carefully and precisely as possible, the expected result from ideal components.
- Clarity - The API designed should be easy to read, understand and use. This would allow for other programs to extend the program library.

3.3.3 External Interface Features

- OS: Ubuntu 18.04 or higher
- Shell: Bash
- Build System: CMake 3.16 or higher
- Graphing Tool: Python 3.7 or higher

4. Design Process

4.1 Project Planning and Structure

4.1.1 CMake Build System and Folder Structure

It was decided to use CMake as the build tool to make compiling the source files less cumbersome and to leverage its inbuilt testing system CTest. An online guide was followed [1] for setting up the folder structure that made the project more modular, allowing the team to work collaboratively without interfering with others' work.

```
.
├──── bin
│    ├──── simulatorplot
├──── build
│    ├──── build.sh
│    ├──── install.sh
├──── CMakeLists.txt
├──── docs
├──── Doxyfile
├──── include
├──── lib
├──── README.md
├──── requirements.txt
├──── spec.pdf
├──── src
│    └──── main.cpp
├──── test
```

File Hierarchy used during development

4.1.2 Documentation

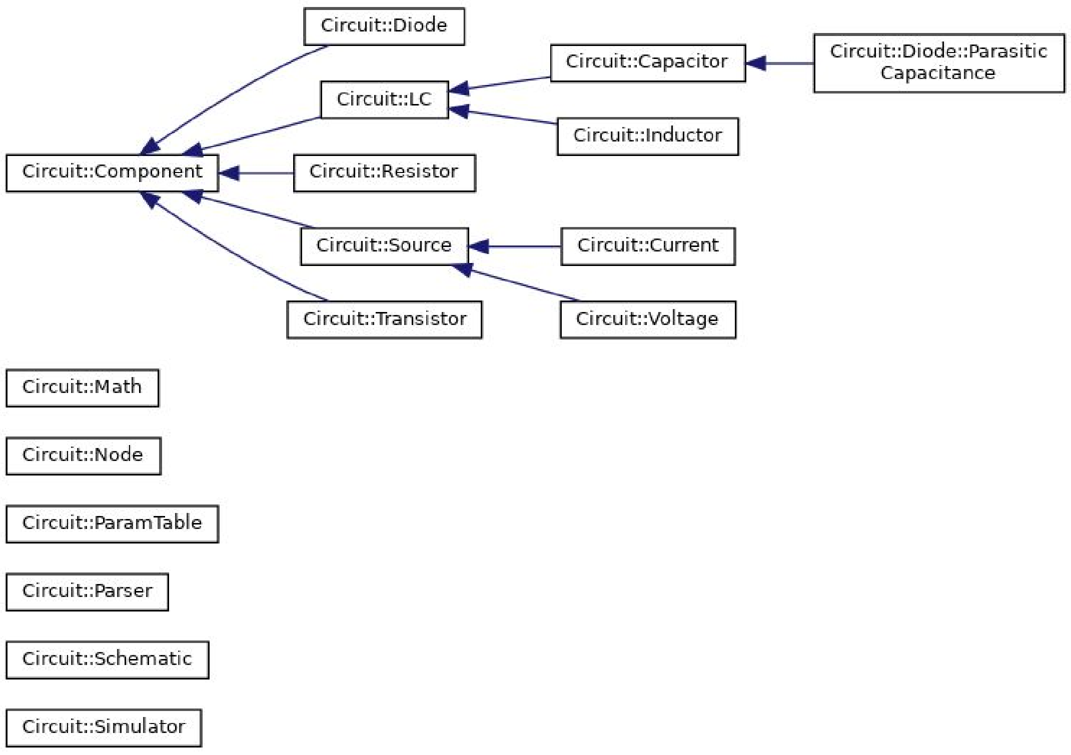
The Doxygen [2] documentation tool was used to generate technical documentation for the API developed during this project. It can be found at https://jjlehner.github.io/404CircuitSim/.

4.2 Continuous Review of Data Structures for Circuit Abstractions

4.2.1 Initial Rough Design

It was decided to namespace everything under "Circuit" for uniformity. The decision was then made to abstract the SPICE schematic as a container, which would store all the nodes and components, as well as set up the connections between them. It was decided to use maps from the Standard C++ Library, rather than vectors, based on the assumption that the simulator would be doing many lookups based on component/node names and computational time could be saved by using maps, which have faster search times than vectors.

The team then began designing the node and an abstract component class, which all circuit components would inherit from. The node class had a vector containing pointers to the components that were connected to it, and the component class had an identical vector containing the pointers to the nodes it was connected to. This bidirectional relationship would make the process of generating the conductance matrix to be quicker.



Class Hierarchy of Namespace Circuit used in this project.

4.2.2 Refactored Code Design

As the design progressed, the code structure and circuit API were constantly re-evaluated to embed more specification defined functionality, while reducing code duplication and complexity. This code re-evaluation and refactoring was particularly acute during the construction of the netlist parser. During this stage the intermediate abstract classes Circuit::LC and Circuit::Source were inserted between the base class Circuit::Component and child classes Circuit::Capacitor, Circuit::Inductor, Circuit::Current and Circuit::Voltage. The pairs Circuit::Capacitor, Circuit::Inductor and Circuit::Voltage, Circuit::Current were found to employ similar behaviour and function calls. Inserting these intermediate classes, therefore, reduced the amount of maintained code and simplified handling within data structures.

A second refactoring choice was made in the movement of the parasitic capacitance properties of diodes to a separate parasitic capacitance class. This choice meant that the code contained within Circuit::Diode encapsulated ideal diode properties and any parasitic capacitance without code duplication - the parasitic capacitance class inherited its behaviour from the capacitor class(see diagram above). This modular design allows for the quick construction of more complex, non-ideal components while simplifying the code for individual classes.

4.3 Solving Linear Equations of form $G * V = I$"

4.3.1 SymbolicC++

The project started out using SymbolicC++ [3] as its linear algebra library because it was assumed that the project might need its symbolic algebraic features. As the project progressed, it was observed that there was no need for these features. Symbolic's C++ system for solving the equations was found to be inefficient and sometimes yielded incorrect results, and so the switch to Eigen [4] was made.

4.3.2 Eigen C++

The switch to Eigen [4] and its LU solver for Sparse Matrices, resulted in a faster runtime for linear circuits and gave access to much more advanced and well-documented API. Eigen also imported trusted Fortran libraries, which included iterative nonlinear solvers. The simulator made use of these libraries later to solve nonlinear components.

4.3.3 Comparing Solvers

Eigen has different solvers built into its library. It was decided initially to use the LU solver, as LTspice uses this solver for its matrices. The speed and efficiency of other solvers were tested and compared, such as the QR solver. The benchmark test below uses the Hayai [5] benchmarking library.

```
[==========] Running 1 benchmark - LU..
[   RUNS   ]     Average time: 105365.987 us (~6914.195 us)
                 Fastest time: 90644.017 us (-14721.970 us / -13.972 %)
                 Slowest time: 126579.117 us (+21213.130 us / +20.133 %)
                 Median time: 105612.425 us

[==========] Running 1 benchmark - QR..
[   RUNS   ]     Average time: 109450.176 us (~7340.099 us)
                 Fastest time: 93525.951 us (-15924.225 us / -14.549 %)
                 Slowest time: 129052.130 us (+19601.954 us / +17.909 %)
                 Median time: 109031.467 us
```

Ultimately the LU solver was chosen because its computation times were more consistent(smaller standard deviation) but not always faster than the QR solver. Further investigations of the optimum algorithm were limited by the occurrence of thermal throttling on this benchmark.

4.4 Multiple Attempts at Solving Nonlinear Equations

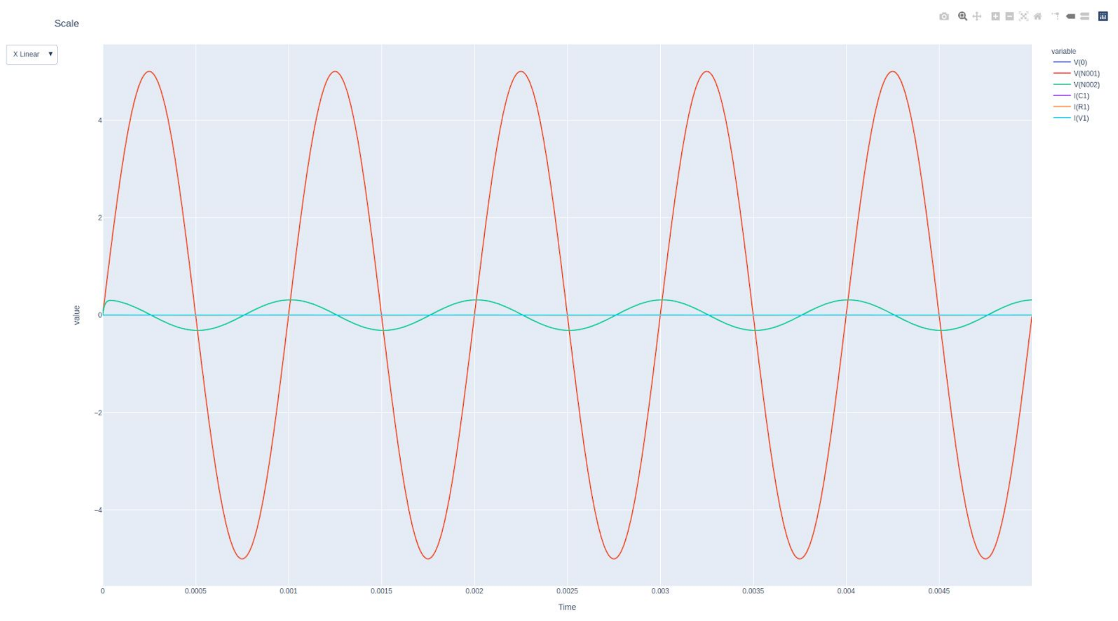
It took two separate algorithm designs to successfully handle nonlinear components. The initial attempt began by “guessing” the value of the voltage at each node. From that, the simulator then calculated the voltage across each diode and determined the current flowing through it with the Shockley equation. Then, the simulator generated the corresponding current vector. With Newton-Raphson and other techniques, the program attempted to minimise the function f(V) = G*V - I(V), where V is the voltage vector, G is the conductance matrix, and I(V) is the current vector, which is itself a function of V. This method was believed to be mathematically sound, as it worked for a circuit of one diode and one source. Moreover, it was based on a model successfully outlined by Professor Mitcheson [6]. However, no matter which iteration technique was tried, more complex circuit voltage vectors failed to converge.

After a design review, it dawned on the team that this method was far more complicated than it needed to be. At this point, the algorithm was improved to consider treating diodes as a resistor with a variable instantaneous conductance. For each diode, the simulator would guess the voltage across it, rather than than the nodes it was attached to. Guessing the value of the voltage across the diode allowed the simulator to generate the current through it and hence find its instantaneous conductance. This was mainly the reverse of the first method outlined. Instead of modelling diodes as current sources, the simulator modelled them as resistors. For more complex circuits, this method meant having to guess far fewer values of voltage than the initially outlined algorithm. Ultimately this method converged for all circuits tested.

The Levenberg-Marquardt algorithm was also included in the Eigen library (which Eigen imported from the MINPACK Fortran library). The algorithm was tested and compared to Newton-Raphson for minimising the set of nonlinear equations in precision and iterative time. There was some concern about the suitability of the Leveberg-Marquardt algorithm. However, during testing, the algorithm always produced results which were just as, or more precise, than the Newton-Raphson method. Precision here was determined by comparisons to the output of LTspice.

4.5 Graphing Tool

In order to improve the rate of debugging and testing, it was decided to write a graphing script to help with plotting the results of the simulator. The Plotly[7] graphing libraries were used to build an interactive graph dashboard, similar to the graphs made by LTspice. The user can add and remove plots on the graph, change the scale from linear to logarithmic and zoom in on specific regions of the graph.



Demo plot produced by the inbuilt graphing tool.

4.6 User Experience

The simulator was designed to be user friendly and easy to use. The simulator has various flags that give the user full control over the inputs and outputs of the simulator. For example, the shell command below would generate a space-separated output file in the out directory and plot the results of the simulation showing only nodes V(N001) and V(N002). See appendix for a full description of possible simulation flags.

```
simulator -i rcnetwork.cir -o out -f space -p 'V(N001) V(N002)'
```

5. Detailed Technical Description of API

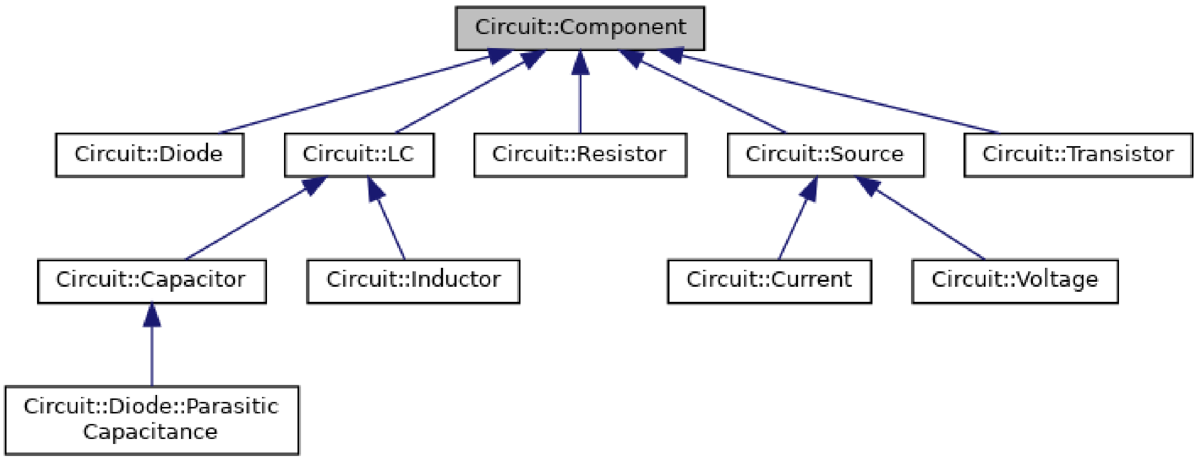
Detailed annotation of the API codebase can be found on the project's Doxygen page: https://jjlehner.github.io/404CircuitSim/

5.1 Circuit Abstraction

The Circuit::Schematic, Circuit::Node and Circuit::Component classes are used to achieve circuit abstraction. The Circuit::Schematic has two main data fields which contain all the nodes and components in the circuit:

```
std::map<std:::string, Circuit::Node *> nodes;
std::map<std::string, Circuit::Component *> comps;
```

The Circuit::Node class stores its voltage and a vector containing all the components connected to it. The Circuit::Component is an abstract base class that all other circuit components inherit. It contains a vector containing all the nodes it is connected to as well as a host of member functions which allow the components to interact with the circuit simulator.



Inheritance diagram for Circuit::Component

5.2 Circuit Math

The Circuit::Math class contains the algorithms that generate conductance and current matrices for a given circuit (at a given timestep) allowing the simulator to solve for voltage.

The conductance algorithm works by iterating through the component map stored in the schematic and adding the conductance of each component to the conductance matrix.

```
init_matrix(conductance); // initialize the conductance matrix
for (auto comp_pair : schem->comps){
      Component *comp = comp_pair.second;
      if(comp->isSource()) continue;
      double value = comp->getConductance(...);
```

```
        handleConductanceMatrixTwoNodes(conductance,
            comp->nodes[0]->getId(), comp->nodes[1]->getId(), value);
}
```

The current algorithm works similarly. It iterates through the component map in the schematic and, if the component is a source of some kind (voltage source, current source, capacitor, inductor, diode), it handles the source logic. It uses dynamic casts to check if a component can be cast down to a type that acts as a source.

Voltage sources add a level of complexity to the algorithm because a voltage source virtually ignores the KCL equations and arbitrarily sets the voltage irrespective of the other currents. As a consequence, voltage sources are handled last and after all the current sources have been dealt with.

```
init_vector(current); // initialize the current vector
// handle current sources
for (auto comp_pair : schem->comps){
      Component *comp = comp_pair.second;
       if (Current *source = dynamic_cast<Current *>(comp)){
            int posId = source->getPosNode()->getId();
            Int negId = source->getNegNode()-getId();
            double value = source->getSourceOutput(...);
            handleCurrentSource(current, posId, negId, value);
      }
      // repeat if block for all types that act as current sources
}
// handle voltage sources
for (auto comp_pair : schem->comps){
      Component *comp = comp_pair.second;
       if (Voltage *source = dynamic_cast<Voltage *>(comp)){
            int posId = source->getPosNode()->getId();
            Int negId = source->getNegNode()-getId();
            double value = source->getSourceOutput(...);
      handleVoltageSource(conductance, current, posId, negId, value);
      }
}
```

5.2.1 Handling Voltage Sources

There were a few unique challenges while trying to implement voltage sources in the simulator. Firstly, the handling of voltage sources in the conductance matrix needed to account for the fact that voltage sources set the voltage between nodes. This was represented by replacing rows in the conductance matrix with rows which encoded the voltage source's positional relationships between nodes. While handling floating voltage sources, however, those replaced rows need to be preserved and added together to form supernodes. Finally, to obtain the current through a voltage source, Ohm's law couldn't be used because the conductance through the voltage source wasn't defined. The workaround was to use KCL to determine the current flowing through the voltage source, using the currents flowing through the components connected to it.

```
// current and conductance matrices have been defined and calculated
above
Eigen::VectorXd new_conductance(current.rows());
init_vector(new_conductance); // new conductance row which defines
voltage source

if (posId != -1)
{
    new_conductance(posId) = 1.0;
    if (negId == -1)
    {
        // if the voltage source is connected to ref node,
        conductance.row(posId) = new_conductance;
        current[posId] = val;
        return;
    }
    else
    {
        // if voltage source is floating,
        new_conductance(negId) = -1.0;
        conductance.row(negId) += conductance.row(posId);
        conductance.row(posId) = new_conductance;

        current[negId] += current[posId];
        current[posId] = val;
        return;
    }
}
else
{
    // when the pos terminal is connected to the ref node
    new_conductance(negId) = -1.0;
    current[negId] = val;
    conductance.row(negId) = new_conductance;
    return;
}
```

5.3 Netlist Parser and Circuit::Simulator

The Circuit::Parser class handles the parsing of the SPICE netlist and its conversion into a Circuit::Schematic Object, which the rest of the simulator interacts with. It has one basic function:

```
static Circuit::Schematic* parse( std::istream& inputStream );
```

Given the time constraint, it was not possible to implement every SPICE command, directive or simulation type. Any unsupported commands were skipped. See appendix for a list of accepted commands.

In order to parse each line, regular expressions were used to check and match patterns, rather than a chain of for loops and if-else blocks. If the line is a component declaration, a matching object is then created and added to the schematic object. The Circuit::Schematic class has a particular function that handles all the complexities of adding a component. This includes setting up the bidirectional relationship of linking nodes to components and components to nodes.

```
// takes in a components and the nodes it should be connected to
void Circuit::Schematic::setupConnections2Node(...)
```

Conversely, if the line is a simulation directive or command, a Circuit::Simulator object is generated, which contains the logic that runs simulations. This design choice was in an attempt to support multiple simulations in one netlist. Circuit::Schematic contains a vector of simulator objects which are run by the main executable.

```
// main executable loop
for (Circuit::Simulator *sim : schem->sims)
    {
        std::ofstream &out;
        // output path is title + simulation type + output format
        out.open(outputPath);
        sim->run(out, outputFormat);
        out.close();
}
```

Also, in order to support runtime variables, like ones defined in the .STEP directive, parameter tables were introduced to store variables. The Circuit::Schematic class contains a vector of parameter tables, and each simulation is run multiple times for each parameter table, or once if no variables are defined.

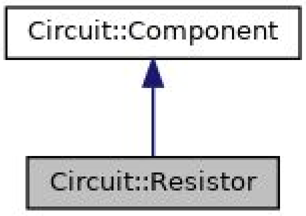
```
// simulator parameter table loop
void run(std::ostream &dst, OutputFormat format){
      for(auto param : schem->tables){
            switch(type){
                  // run simulation for a given type and parameter table
            }
      }
}
```

5.4 Linear Components Modelling

All components inherit from the component class and are expected to overload the functions that allow components to interact with the circuit simulator. These functions are:

```
virtual double getConductance(...) const;
virtual double getCurrent(...) const;
virtual bool isSource() const;
```

Resistors were the easiest to model and overloading the defining functions for a resistor is straightforward. The method isSource() just returns false. The getCurrent() method returns the voltage difference across its nodes multiplied by the conductance. The getConductance() method returns the reciprocal value of the resistor(the conductance).



Inheritance Diagram for resistors

Capacitors and Inductors had much functionality in common. An intermediary abstract class called LC was introduced, to reduce repeating code which defined the same behaviour. For both components, an equivalent model was obtained from simpler components (resistors and current sources). An analysis was carried out on the governing equation for the components using simple approximations to obtain the mathematical expressions for the equivalent models.

$Capacitors\ Governing\ Equation\ :\ i\ =\ C\frac{dv}{dt}$

$\frac{dv}{dt}\ =\ rate\ of\ change\ of\ voltage\ across\ capacitor\ with\ respect\ to\ time$

$An\ approximation\ (more\ precise\ for\ smaller\ \Delta t)\ :\ i\ =\ C * \frac{(v_n - v_{n-1})}{\Delta t}$

$where : \Delta t\ =\ timestep$

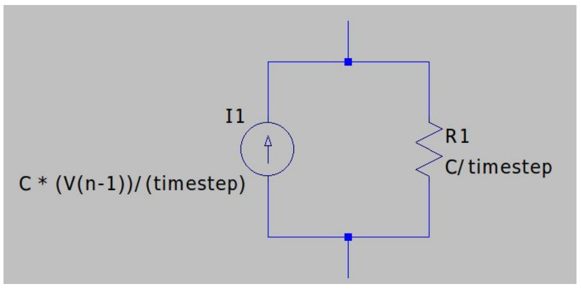
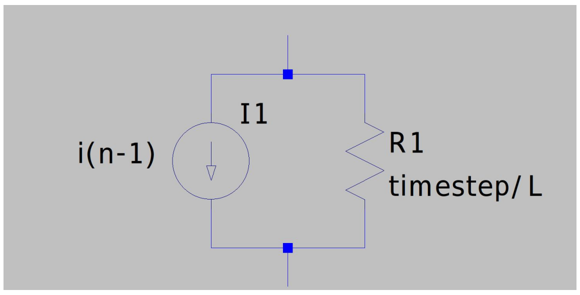
$(v_n\ -\ v_{n-1}) = change\ in\ voltage\ across\ the\ capacitor\ for\ each\ timestep$

$\Rightarrow i_n\ =\ \frac{Cv_n}{\Delta t}\ -\ \frac{Cv_{n-1}}{\Delta t}$

$\frac{Cv_{n-1}}{\Delta t}\ \rightarrow a\ current\ source\ based\ on\ the\ previous\ value\ of\ voltage$

$\frac{C}{\Delta t} \rightarrow a\ constant\ conductance\ based\ on\ the\ timestep$

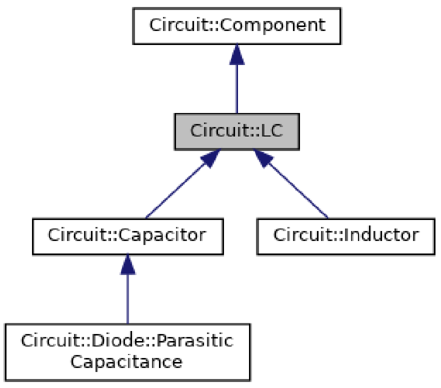
$\Rightarrow i_n\ =\ -i_{source}\ + G * v_n$

 *How the simulator models capacitors*	 *How the simulator models inductors*

Hence, a capacitor can be modelled as a current source with some parallel resistor, and both work together to set the current through the capacitor.

An inductor is modelled very similarly, but the equation results in a voltage source with some series resistance. This is transformed into its Norton equivalent with a current source and parallel resistance.

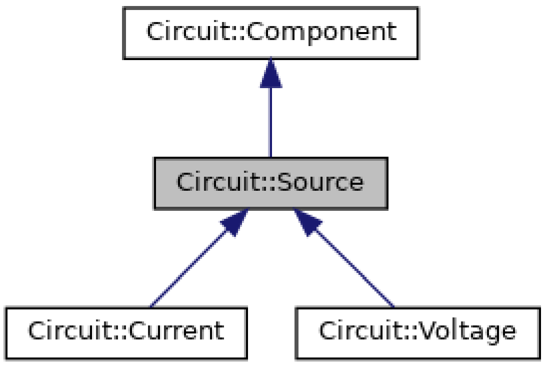
$$\text{Inductor Governing Equation} \; : \; v = L\frac{di}{dt}$$
$$\text{where} : \; \frac{di}{dt} = \text{rate of change of current through inductor with respect to time}$$
$$\Rightarrow v_n = \frac{Li_n}{\Delta t} - \frac{Li_{n-1}}{\Delta t}$$
$$\frac{Li_{n-1}}{\Delta t} \rightarrow \text{a voltage source based on the previous value of current}$$
$$\frac{L}{\Delta t} \rightarrow \text{a constant resistance based on the timestep}$$
$$\Rightarrow i_n = \frac{v_n}{L}\Delta t + i_{n-1}$$
$$\Rightarrow i_n = -i_{source} + G * v_n$$



Inheritance Diagram for LC components

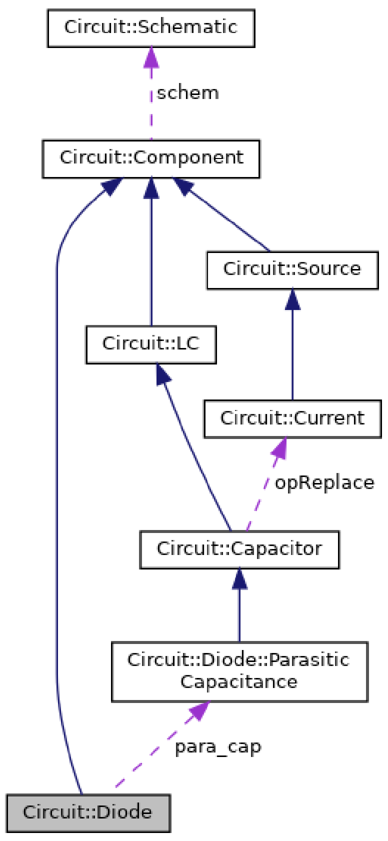
5.5 Source Modelling

Sources were modelled as a particular type of component which didn't need to overload getConductance(). An intermediary abstract class, Circuit::Source, is added which would overload isSource() from Circuit:Component, and add an extra function, getSourceOutput(), which is used to obtain the value of the voltage/current source at any given timestep.



Inheritance Diagram for Source Components

5.6 Nonlinear Components Modelling



Currently, the only nonlinear component supported is diodes. The simulator models them as resistors and iterates through multiple estimations of voltage across them to find the immediate conductance of the component. The voltage across the diode is initially guessed to be 0.0V, and this then produces an estimate for the current through the diode using the Shockley Equation:

The program, having found both the current and voltage across the diode, takes the ratio between them to find the instantaneous conductance. This value is then used to generate the conductance matrix and find the corresponding voltage across the resistor used to model the diode. The difference between the “guess voltage” across the diode and the calculated voltage across the resistor model is calculated. This value is saved as the error voltage. The error voltage is then subsequently used to update the guessed voltage across the diode. The simulator eventually converges to the correct voltage difference, which is usually 0V or ~0.7V. As described above, iterative techniques are used to converge on an answer for the equations.

The iterative minimisation method used is Newton-Raphson. Overall it tries to minimise the error vector, which is a function of guessed voltages across each diode in the netlist.

If more than one nonlinear component is entered into a netlist, the error vector function will be dependent on more than one variable. This requires updating the Newton-Raphson iterative formula from one-dimensional root-finding to n-dimensional root finding. Below is the updated equation [8] :

$$\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} - J(\mathbf{x}^{(k-1)})^{-1}\mathbf{F}(\mathbf{x}^{(k-1)})$$

5.6.1 Diode Parasitic Capacitance Modelling

For diodes, it was desired to demonstrate how the simulator would handle non-ideal components. The modularity of the API design meant that it was easy to add parasitic capacitance to the diode model. A member object was added to the class diode called “parasitic capacitance”. This object inherited its behaviour from the capacitor class. Effectively now, whenever a diode is placed in a netlist, the simulator inserts a parasitic capacitor in parallel with it. This capacitor’s capacitance value is dependent on the guessed voltage across the diode and is modelled by the class Circuit::Diode::ParasiticCapacitance.

In the diode model, the value of capacitance is set based on the voltage across it. The equation used is a copy of the one used in LTspice [6]:

$$C_j = \frac{C_j(0)}{\sqrt{1 - \frac{V}{V_0}}}$$

In

The simulator used LTspice's default value for Cj0 : $1 * 10^{-14}$

6. Demonstration Circuits and Evaluation of Simulator

6.1 Evaluation and Benchmarking Process

To benchmark the speed of the simulator, the team made use of the Hayai [5] benchmarking library. For each test, the average simulation time was determined by running each simulation 100 times and outputting the mean, fastest, slowest and median runtime. These results were then compared with those produced by LTspice. In running the benchmarks, the following optimisation flags were used: -O2 -march=native -flto as they ended up producing the most effective runtime performance boost. It was also attempted to use -O3, but this resulted in a performance decrease that seems to be due to the cmov instruction being used for loops, which has an increased cycle count [9].

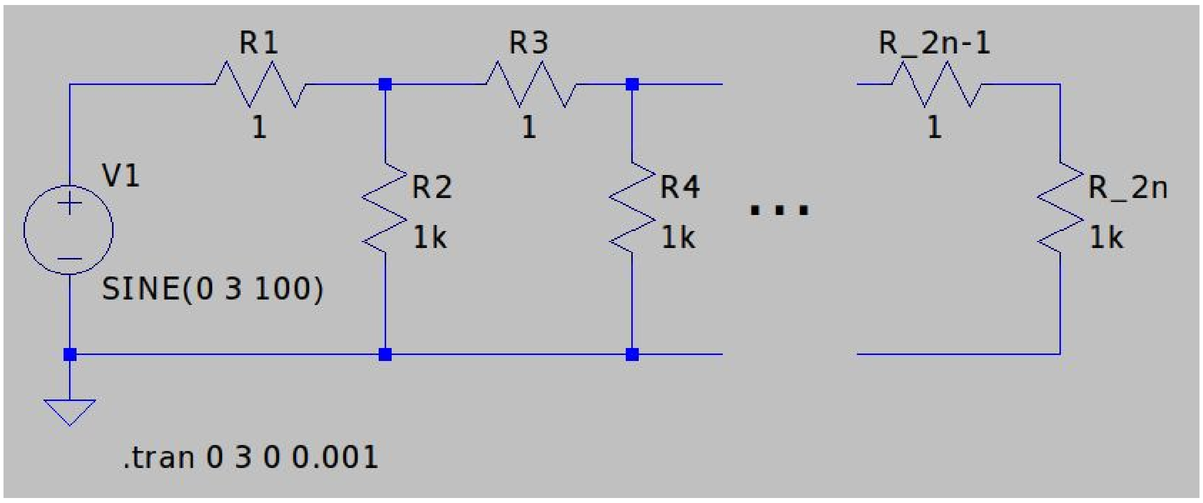
In addition to checking the runtime of the simulator, the memory usage of the simulator was observed. Using Valgrind [10] memory debugging tool, the amount of memory allocated was checked while the simulator was running. This was to ensure that there were no memory leaks. See appendix for Valgrind output.

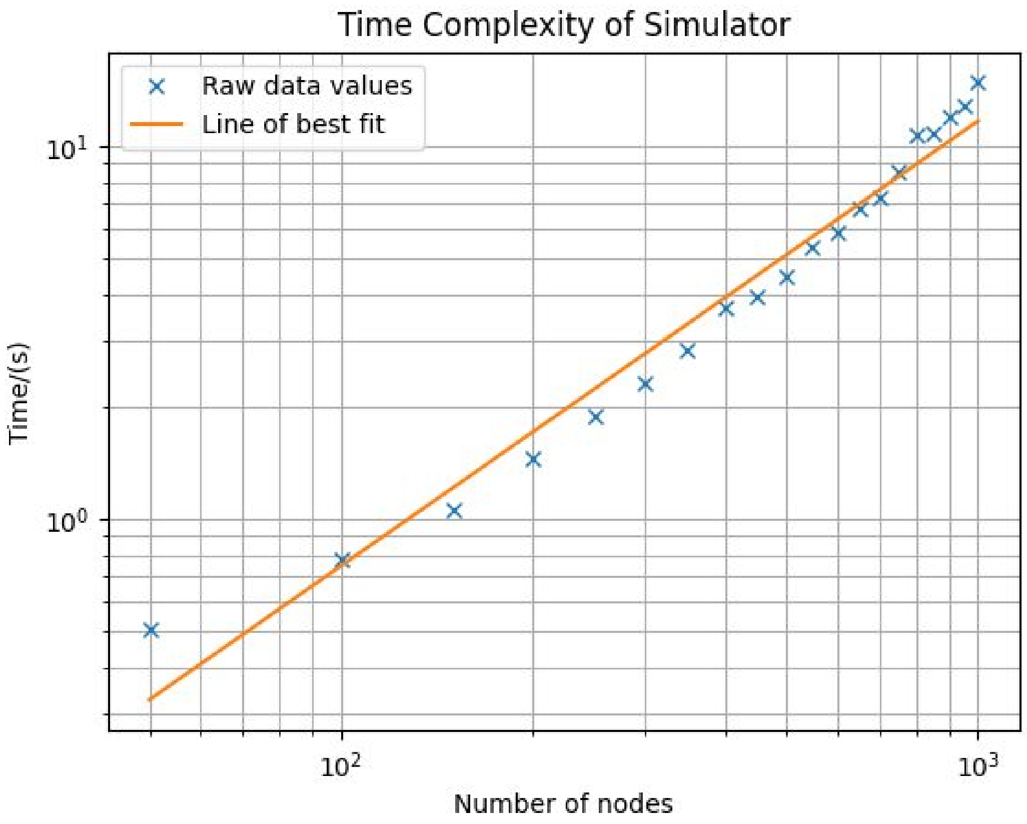
To evaluate the simulator, where appropriate, the accuracy of the simulator's results were compared to LTspice. The rate of convergence when employing an iterative solution was also observed to see if the algorithm could be modified to improve efficiency. See appendix for netlists of circuits used for evaluation and testing.

In order to measure the power consumption of the simulator, Powermetrics [11] was used to measure the CPU time taken for the simulator to run. This turned out to be roughly 26674.8576ms for a netlist with 3 components running 9 transient simulations (of about 1000 timesteps) using the .STEP directive. Then, by using the CPU's average power consumption during execution, it was estimated that the program used 791J for the entire simulation. This corresponds to the power consumption of roughly 88J per step.

6.2 Effect of number of nodes on runtime (Time Complexity)

In order to test the scalability of the project, a script was used to generate netlists of increasing size. For a given input value n, the script generated a netlist with 2n resistors and n nodes based on a simple repeating pattern. The repeating pattern used to generate the circuits can be seen below. Every netlist produced by the script was run 50 times, and the average simulation runtime was recorded for each value n. The data collected was plotted on a log-log graph of simulation run time vs the number of nodes in the netlist. The line of best fit was generated from the collected data and was found to have a gradient of 1.2. This was suggestive of a polynomial(roughly linear) computation time for the number of nodes added. This matched Eigen's documentation for sparse matrices having linear time complexity in LU decomposition.





The gradient of the line of best fit = 1.1932909942554772
R-Squared Value = 0.9855725664949567

6.3 Circuit Component Tests

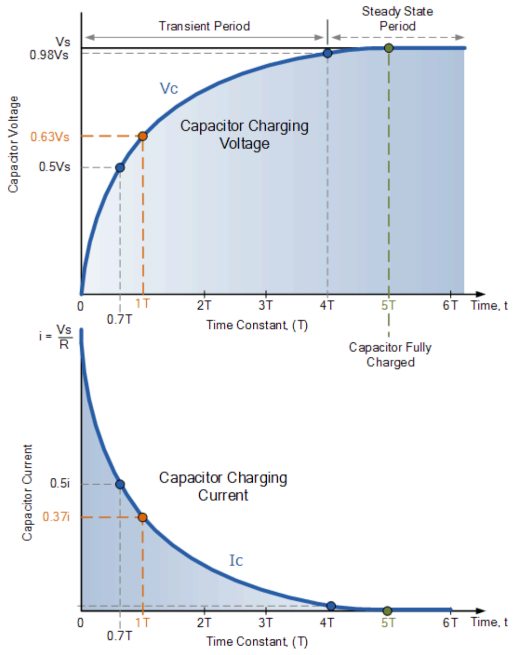
6.3.0 External Interactive Graphs and Netlists

For each of the evaluation tests, an interactive HTML plot was generated which contained the results produced by the simulator. Those links are supplied with the result of each test. The netlist used for each test can be found in the appendix.

6.3.1 Test 1 - Simple RC Charging network

Interactive graph - https://jjlehner.github.io/404CircuitSim/rc.html

The first test performed was the simulation of an RC charging network. In the early stage of the simulator output, assuming that just before the test starts the voltage across the capacitor is zero, a transient effect of the capacitor charging should be observed. As time progresses, the transient simulation ought to tend to the steady-state solution observed by an operating point simulation. (.OP). The steady-state solution of this circuit should be 5V across the capacitor, as a capacitor acts as an open circuit to DC signals. In the steady-state, no current through any component should be observed.



←The expected results for the LC charging network

Diagram taken from https://www.electronics-tutorials.ws/rc/rc_1.html
Date - 8/6/20

As the capacitor is changing a negative exponential decay, tending towards zero should be observed for current and voltage through and across the resistor. The time constant of these exponential decays should be equivalent to resistor value * capacitor value.

For this test, the error value of a reading is determined by taking the difference between LTspice's simulator output and the project's simulator output for an equivalent point in time. While it is possible to have mathematically determined all values in this circuit rather than relying on LTspice to produce an error, it was thought that since all other tests were being performed against LTspice, it would be inconsistent not to use it here.

Specification and Functional Points Tested:

1. Correct parsing of components and SPICE commands
2. Modification of the conductance matrix to accommodate a voltage source.

3. Accurate modelling of a capacitor and resistor, notably the transient effect of the capacitor charging. (Current through resistor should always be the same as the current through the capacitor as they are in series)

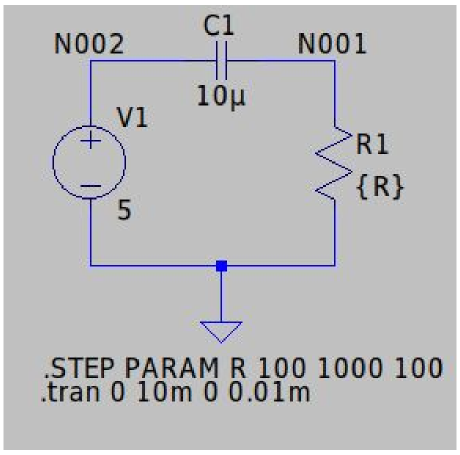
The Simulation benchmark results

Average time: 2396878.466 us (~236307.808 us)
Fastest time: 1893799.966 us (-503078.500 us / -20.989 %)
Slowest time: 2728756.966 us (+331878.500 us / +13.846 %)
Median time: 2344749.966 us

LTspice simulation time:

Total elapsed time: 0.141 seconds.

The project's simulator runs 2.552s slower than LTspice for this particular netlist. Despite being slower, the simulator's runtime is within the time constraint demanded by the specification.



```
.STEP PARAM R 100 1000 100
.tran 0 10m 0 0.01m
```

Fig1

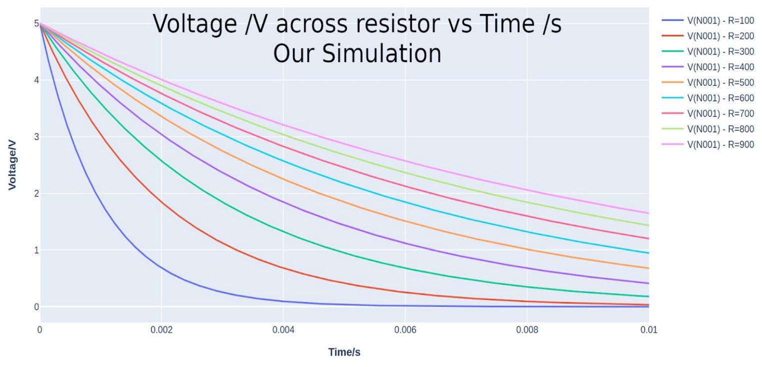


Fig2

Ostensibly from figure 2 and figure 3, the simulator performs as expected. The voltage across the resistor seems to demonstrate exponential decay - this assumption is confirmed by plotting natural logarithms of voltage and time against each other and noting the straight lines produced. The gradient of these lines is also found to be the negative reciprocal of the time constant, just as the theory suggests [12].

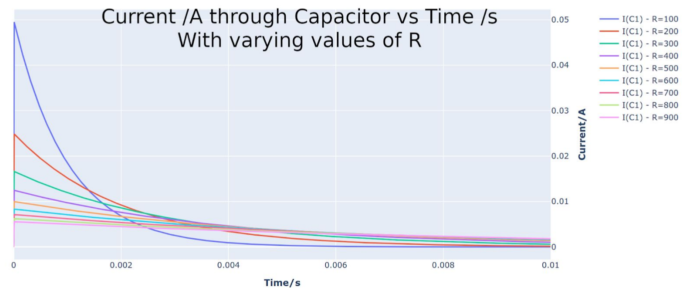


Fig3

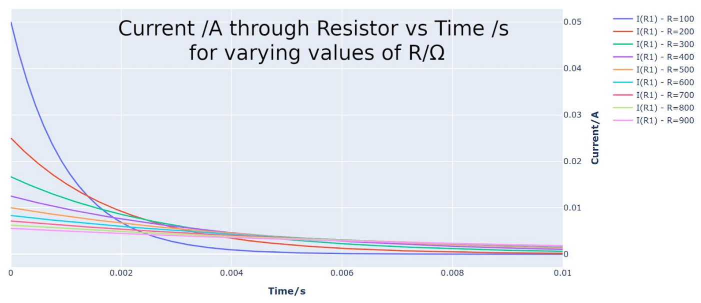
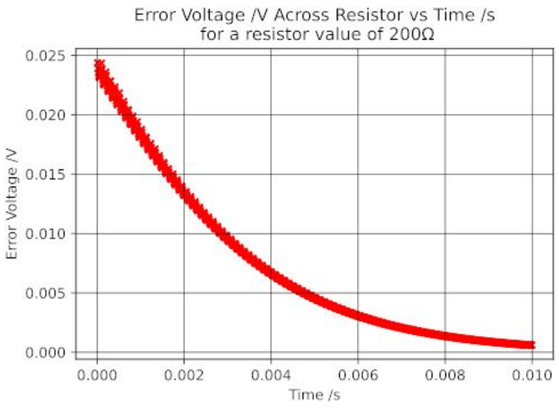


Fig4

Figure 3 and Figure 4 represent the current through the capacitor and resistor respectively. From the circuit diagram(figure 1), it is obvious to see that the current should be the same through both components for any given moment in time. The output of the simulation matches this assumption(the values of current for both components are identical at each timestep)

Comparison to LTspice Output

Fig5



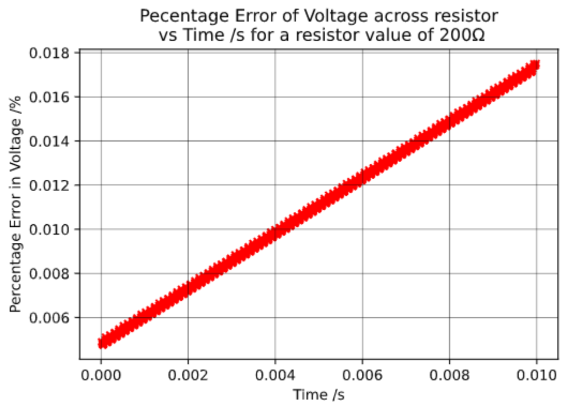


Fig6

Fig7

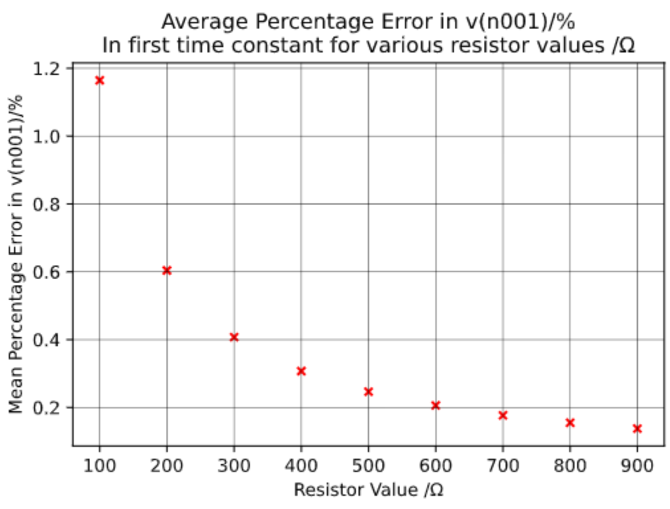


Fig8

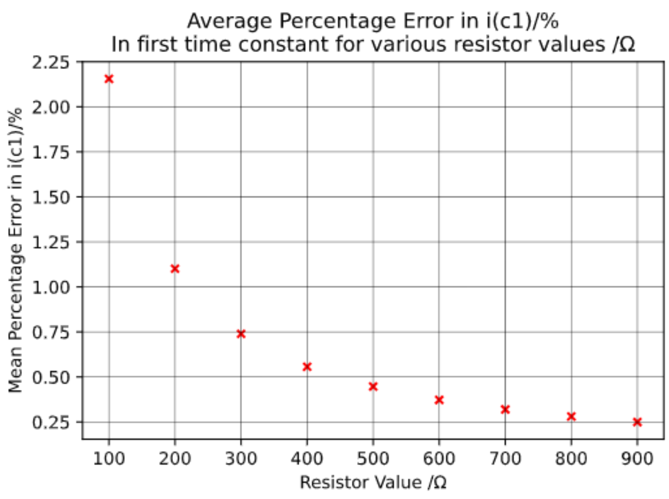


Figure 5 is a plot of the difference in voltage for each timestep between the simulation prediction and that of LTspice, for a resistor value of 200Ω. Despite Figure 5 illustrating the most significant raw error at the earliest point in the simulation, figure 6 demonstrates that the percentage error of voltage is most precise early in the simulation. Percentage error is calculated by:

$$\frac{|spice\ prediction(t) - the\ prediction(t)|}{spice\ prediction(t)}$$

The increase in percentage error is most likely caused by the inaccuracies of the numerical integration of charge across the capacitor used in the capacitor modelling (see above). All errors in voltage noted in the graph are within the defined specification of 5% from LTspice's prediction.

For this particular simulation, it was important to determine how the precision of the simulation would change with varying values of R from the step command. Figure 7 and 8 both demonstrate the circuit parameters v(n001) and I(c1) decreasing in average error for increasing resistor values. Average error for a circuit was determined by summing the error at each timestep and dividing it by the number of timesteps. For consistency, error values in figure 7 and 8 were taken from the first time constant for all circuits. This prevented the error being artificially inflated for circuits of small value resistors, which tend to zero after a shorter time than circuits with larger resistors.

In regards to the circuit simulated, a more significant error for circuits with smaller R-value corroborates the idea that the capacitor charges fastest(has a smaller time constant) with smaller values of resistance. It, therefore, is logical, that the simulator has the most significant source of error for circuits with small resistors as there is the greatest rate of change in circuit currents and voltages. Having used a fixed timestep in the project's simulator means that: the numerical integration is least precise for faster-changing signals. This could represent a possible extension for us: to decrease the time step during the simulation (and therefore increase the precision) if the rate of change for a particular signal is noticed to be large or increasing. As both figure 7 and 8 demonstrate, the percentage error never increases above 5% for these resistor values. The error does increase above 5% for resistor values less than 50Ω. This can be manually fixed by decreasing the size of the timestep in the netlist.

All technical and non-technical specifications are met for this circuit as detailed above.

6.3.2 Test 2 - Single Diode Circuit

Interactive graph - https://jjlehner.github.io/404CircuitSim/diode.html

This is the first test which includes a diode and therefore relies upon an iterative method to be solved in the simulator. In conduction, one should expect a ~0.7V across the diode. Proper evaluation of diode performance was required as it took two design iterations to get diodes to perform correctly and it was essential to see if the addition of diode support still meant that the simulator met the specification requirements for speed, precision and memory safety.

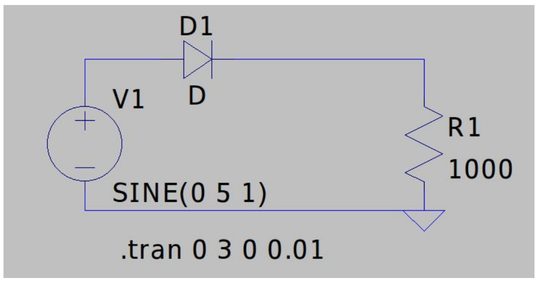


Figure 1 - Circuit Diagram represented by netlist inputted into the simulator

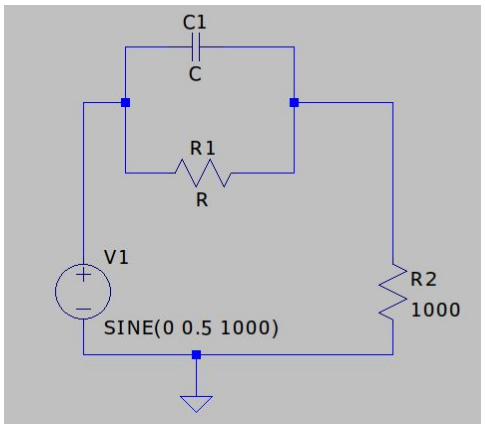


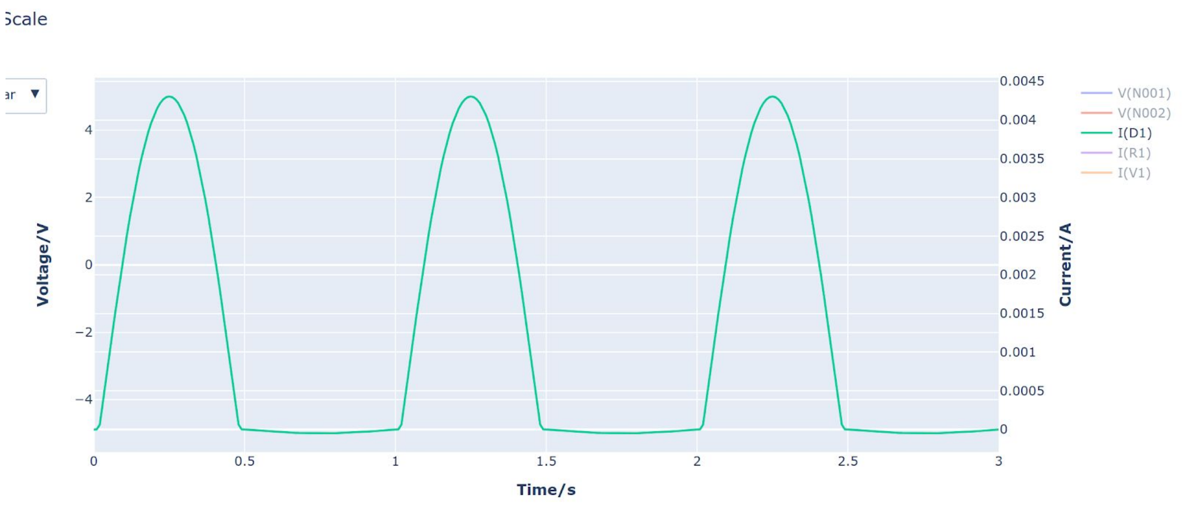
Figure 2 -
How the simulator internally represents the same circuit.

Where the values of R and C are determined by the "guessed voltage" across them on each iterative step (see above).

Error 404 Simulator Times:

Average time: 55742.426 us (~11550.734 us)
Fastest time: 34271.907 us (-21470.519 us / -38.517 %)
Slowest time: 100553.907 us (+44811.481 us / +80.390 %)
Median time: 61245.407 us

LTspice's simulation time - 0.032 seconds
LTspice ran 0.024s faster



Error 404 Simulator Output
Figure 3 - Current through the diode (use axis on the right)

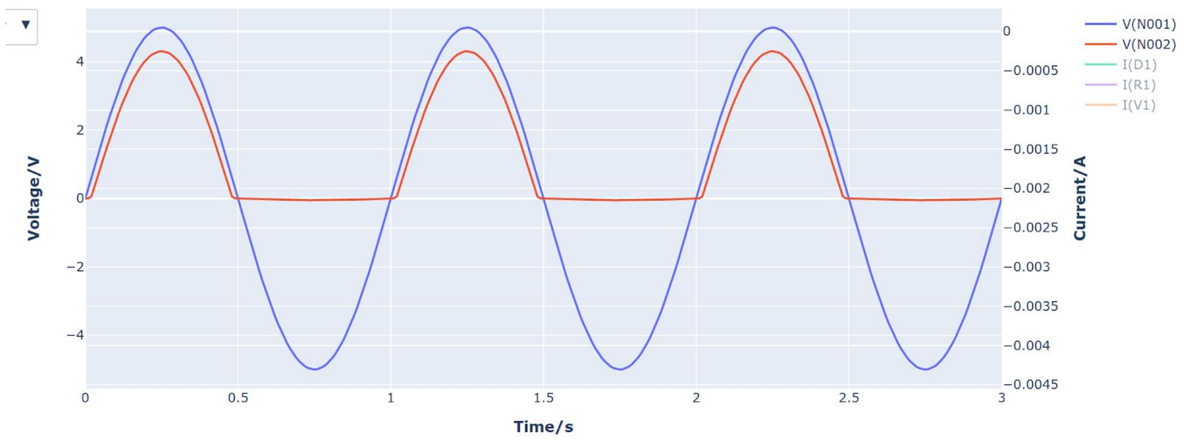


Figure 4: Error 404 Simulator Output
Blue Line - Voltage from the source.
Red Line - Voltage across resistor

Figure 3 and figure 4 qualitatively demonstrate the circuit performing as expected. Figure 3 illustrates that there is a significant current flowing through the circuit for only half the time period. Figure 4 also demonstrates that when the diode is in conduction (in the positive section of the source output) voltage across the resistor follows the source but with a ~0.7V difference, caused by the voltage drop over the diode.

Investigation of Simulator Behaviour

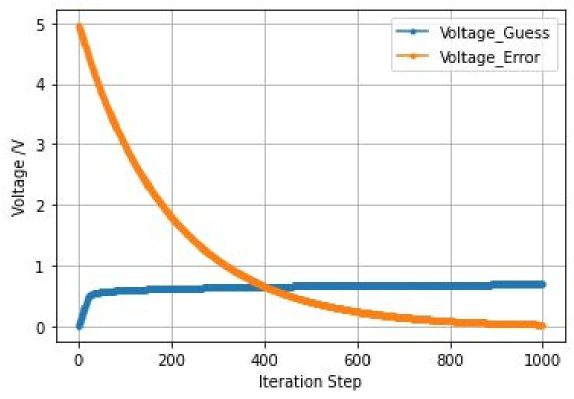


Figure 5
Convergence Plot at 260us

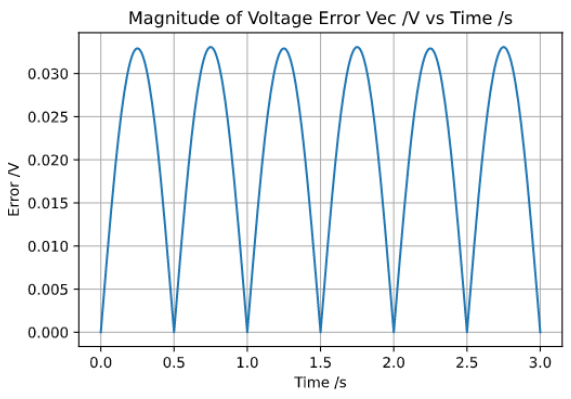


Figure 6 - Magnitude of voltage error vec after 1000 iterations for each time step

The blue line of figure 5 represents the simulator iteratively improving its estimation of the voltage across the diode. The data for this particular graph is taken on the iterative step for the time value of 260us into the simulation. One can see that at the first iterative step, the simulator has the initial guess of zero, which correctly increases within the first 20 steps to a value around 0.7V, as one would expect. The orange line indicates the magnitude of the voltage error at each time step for a corresponding guess at the voltage across the diode(shown by the blue line). Interestingly it then takes the simulator another 980 iterations to move closer to a value which minimises the error to reasonable levels. For a circuit of only one diode, the rather large number of iterations required to converge surprised the team. A further discussion in test 3 can be seen about the possibility of adding some code to handle this.

Figure 6 is a graph of the magnitude of the voltage error vector after 1000 iterations against time. Description of how the voltage error was calculated can be seen in the detailed discussion of API. An interesting point of note was that the magnitude of the voltage error vector was itself in phase with the voltage source. If more time could be devoted to this project, it would be interesting to investigate further the cause of this relationship as it might help to improve the iterative algorithm. As one can see the voltage error vector was always quite small for this circuit (less than 0.5% of the voltage value) meaning that the voltage vector converges within the 1000 iteration limit imposed by the simulator.

Error Statistics:
After 1000 iterations

- Average Error - 0.015V
- Minimum error - 0.00005V
- Maximum error - 0.030V

In conclusion, all specification requirements were met with regards to speed, although this test showed the project to be significantly behind the performance of LTspice. The model for diodes seems to hold up against the expected results qualitatively.

6.3.3 Test 3 - Two Diode Crossover Distortion

Interactive graph - https://jjlehner.github.io/404CircuitSim/double_diode

The following circuit has historical use in Diode Protection for Shield Resistance Measurement of GMR Recording Heads [13]. As the ultimate goal of this project is to mimic tools used in industry, it is vital to discover the precision of the simulator in a practical circuit. This circuit also tests the simulator's ability to iteratively find solutions for circuits with more than one nonlinear component, a significantly more difficult challenge.

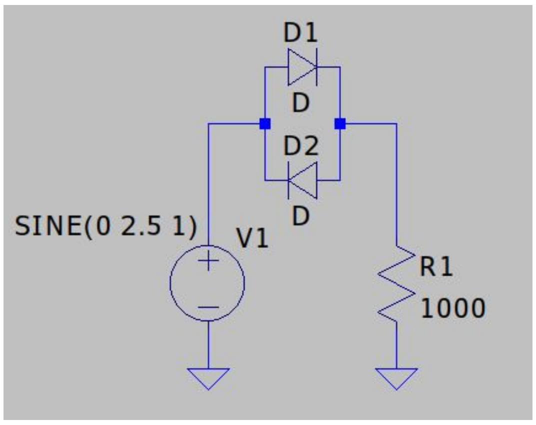


Figure 1
Simulated Circuit

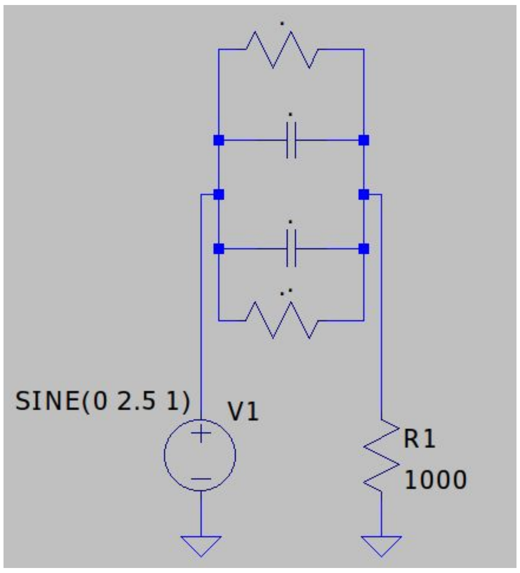


Figure 2
How the simulator sees the same circuit. Where the values of R and C are dependent on the voltage across them

Error 404 Simulator Times:

Average time: 19390.291 us (~1484.050 us)
Fastest time: 18388.971 us (-1001.320 us / -5.164 %)
Slowest time: 26780.971 us (+7390.680 us / +38.115 %)
Median time: 18705.471 us

LTspice’’s simulation time - 0.037 seconds

For this circuit, the project’s simulator runs 0.0176s faster than LTspice. This was a surprising result because, in the previous test, which was a simpler circuit, LTspice outperformed the project’s simulator.

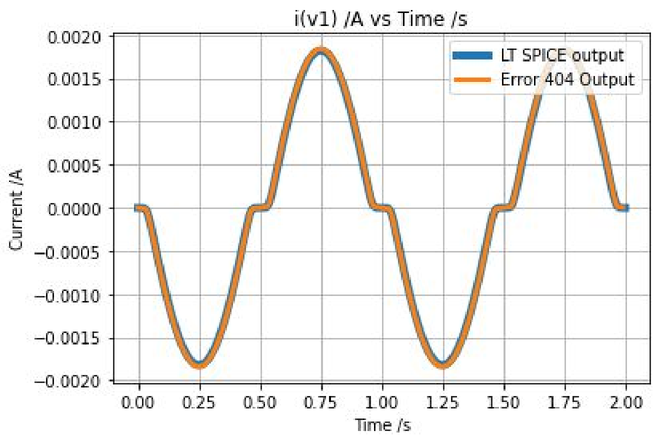


Figure 3

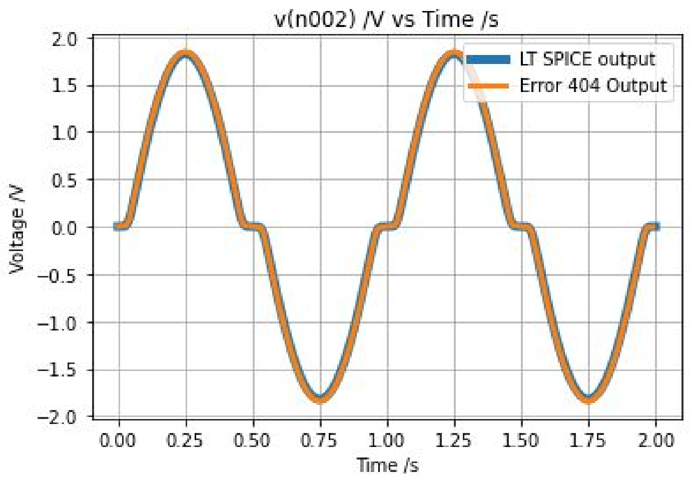


Figure 4

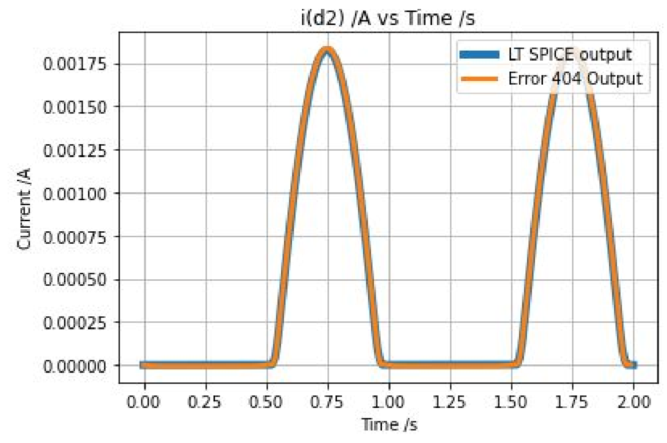


figure 5

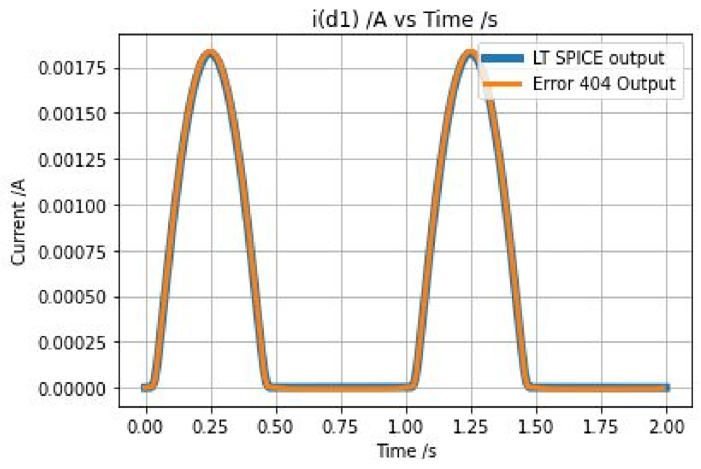
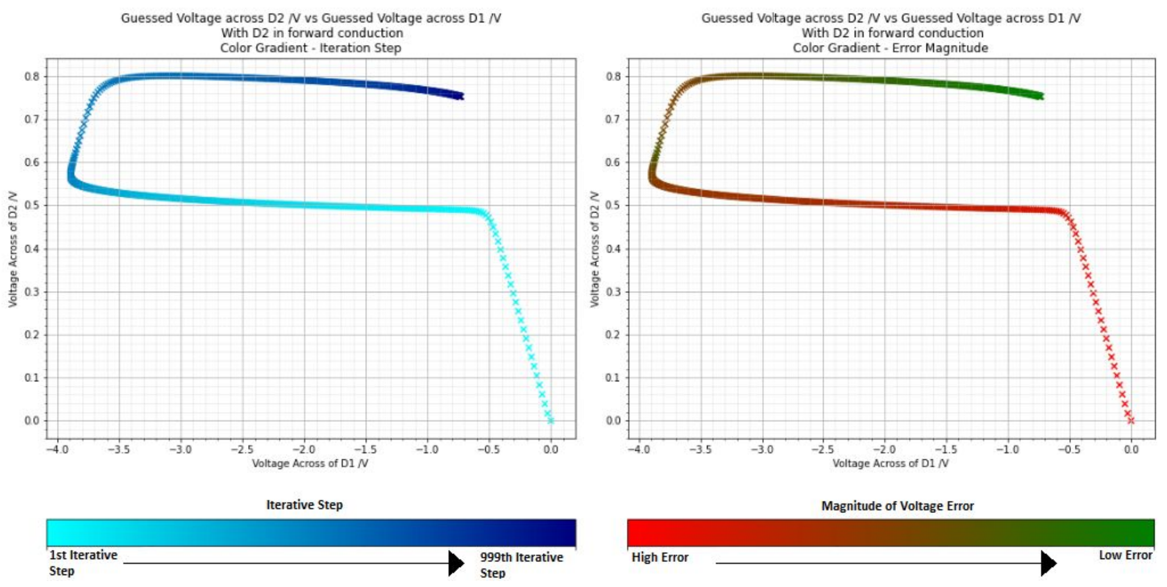


figure 6

As one can see in figures 3-6, the simulator follows the output of LTspice exactly. The blue line represents LTspice's output and has had to be thickened as the simulator's output completely masked it otherwise. This demonstrates the diode model functioning and interacting well with other components. Overall this test was a huge success, not only did the simulator meet the specification's demand for precision, but also the speed requirement, which was initially of slight concern due to the number of iterations on each timestep.



These diagrams demonstrate the program using the Newton-Raphson iterative technique to converge on voltages across both diodes for one timestep. Each mark on the scatter diagram is a guess made by the simulator for a voltage across each diode. At this particular moment in the simulation, diode D2 is in forward conduction. As both diodes are in parallel but orientated in opposite directions, the voltage across them should be equal in size but negatives of each other. As D2 is in forward conduction, it should have ~ 0.7V across it whereas D1 should have ~ -0.7V across it. The left diagram shows that by the final iterative step (coloured in dark blue), the simulator has converged on a value that was predicted. The diagram on the right shows that the magnitude of the error voltage vector at this point has reached a minimum. For a description of how the error voltage vector is calculated, see above. These graphs highlight two fundamental properties of the simulator: firstly, that the simulator successfully navigates a local minimum of voltage error for this circuit and secondly, the number of iterations it took for the simulator to converge on the correct value. The point (-3.75,0.62) has a more greenish colour than the points surrounding it. This point illustrates that the simulator found a diode voltage vector which reduced the voltage error but which was not the global minimum. If the simulator had converged to this point rather than allowing the error to increase to find the global minimum, the simulator would output bogus results. The Newton-Raphson iteration technique optimises this circuit well. However, this result is not a guarantee that it could find the minimum in circuits with more nonlinear components which produce higher dimensional voltage guess vectors.

Addressing the second issue of the number of iterations until successful convergence: the left graph demonstrates that the simulator reaches the correct voltage vector values just as it reaches the last steps of the iteration. The gradual change from light blue to dark blue means that the value is continually improving throughout the iterations, not finding a minimum quickly and staying there. This means that for this circuit, the simulator is performing near the ideal number of iterations for each timestep. A possible weakness of the initial simulator design this highlights is the fixed number of iterations for each timestep. After performing this test, a section of code was written to increase the number of iterations if the voltage error vector magnitude were still large or decrease the number of iterations for a timestep if the voltage error vector magnitude were small. Although a slight speed increase was noticed, it was decided not to add this feature in the final master branch as it was believed that, without more thorough testing, it might lead to non-convergence for some circuits.

6.3.4 Test 4 - Full Wave Rectifier Circuit

Interactive graph - https://jjlehner.github.io/404CircuitSim/FWRec

This circuit was one of the first shown to first-year students by Professor Holmes illustrating the usefulness of diodes and nonlinear components. Given the extreme usefulness of the following circuit, it is essential to know if the simulator can handle it precisely. This circuit further increases the number of diodes in the circuit to four and therefore, the number of voltages the simulator needs to guess iteratively. Qualitatively, one would expect this circuit to have a DC output across the terminals of R1. The value of C was deliberately chosen to be too small, such that the capacitor discharging signal could be observed in the output voltage.

Error 404 Simulator Times:

```
        Average time: 848732.973 us (~89537.076 us)
        Fastest time: 728088.973 us (-120644.000 us / -14.215 %)
        Slowest time: 1150283.973 us (+301551.000 us / +35.530 %)
        Median time: 812862.473 us
```

LTspice’S simulation run time - 0.268 seconds

Project simulator ran 0.580s slower than LTspice.

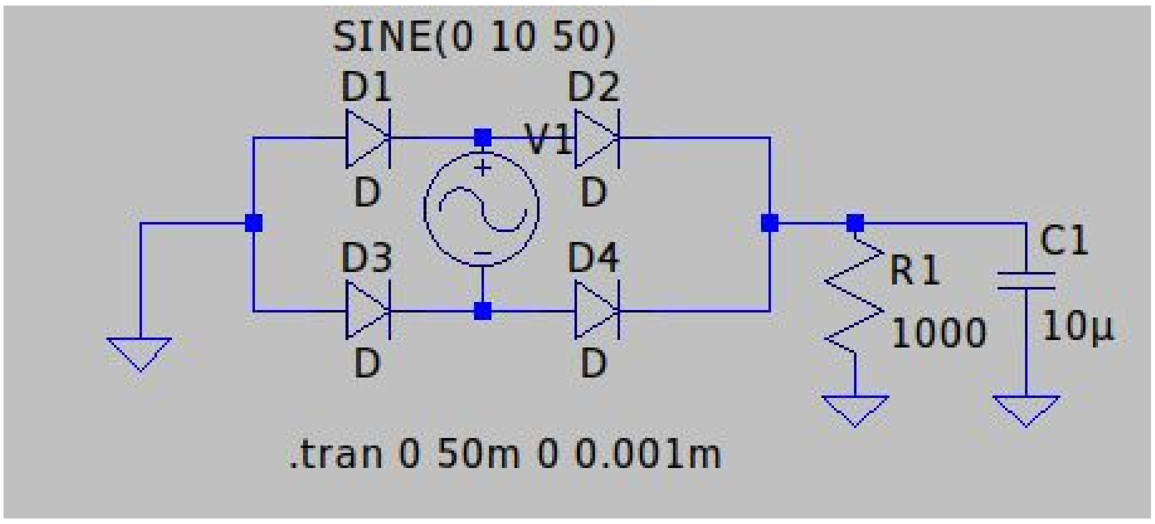
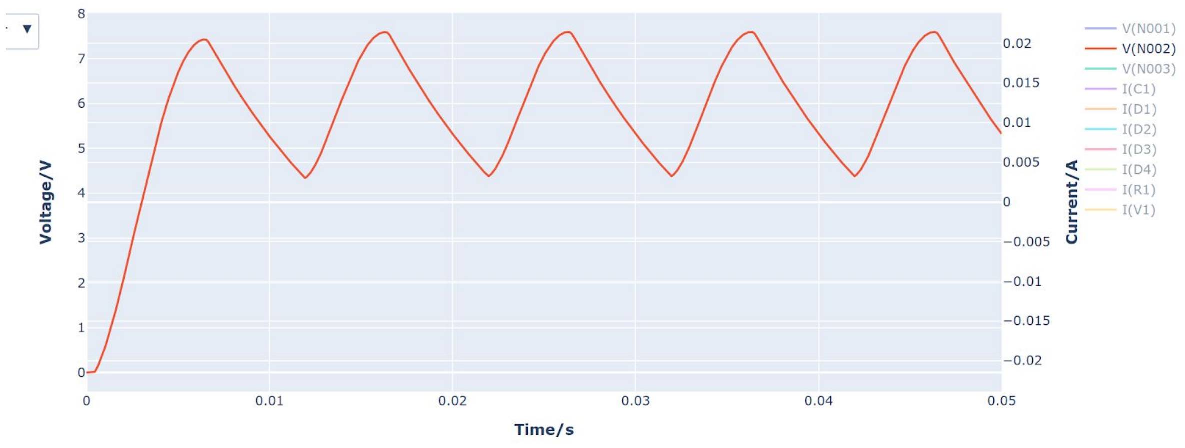


Figure 2 - In-built plotter graph of voltage across R1 and C1



The results from figure 3-6 show the value of the project's simulator output, against that of LTspice. Of all the evaluation tests performed, this tests ostensibly shows the simulator struggling to match the precision demanded by this project's specification. Overall, however, it was not believed that this discrepancy was a result of a failure in the iterative algorithm, rather LTspice modelling some non-ideal characteristics not yet modelled by this project. The graphs below are the results of many attempts to reduce the distance between the program's output and LTspice's output. This was the closest match that could be formed and involved adding 100Ω series resistance to the diode model. One possible cause of this discrepancy could have been LTspice's use of a quadratic function between the on and off state of the diode [14]. Ultimately, this simulator output qualitatively follows the expected result. If more time could be devoted to the project to prepare it for industrial use, the yet unknown non-ideal characteristic would be found.

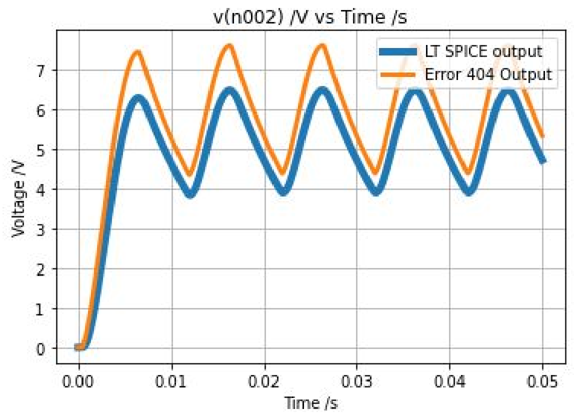


Figure 3

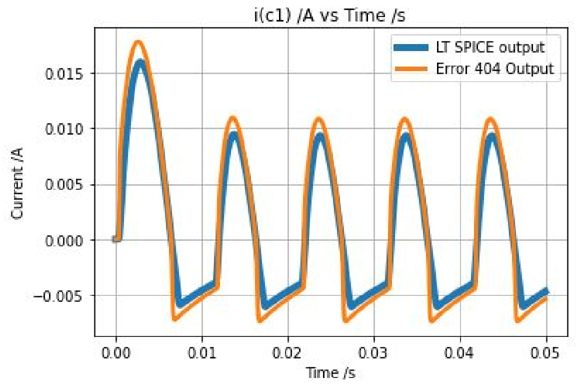
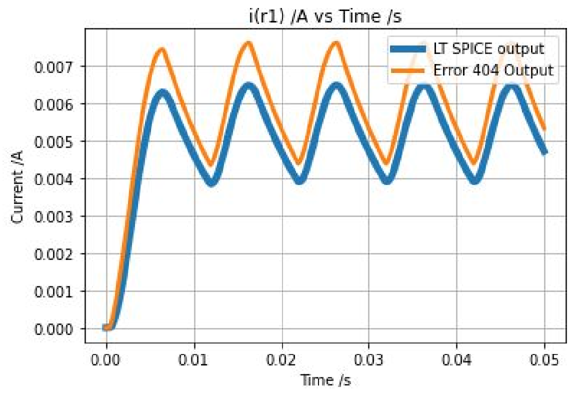


Figure 4

Figure 5



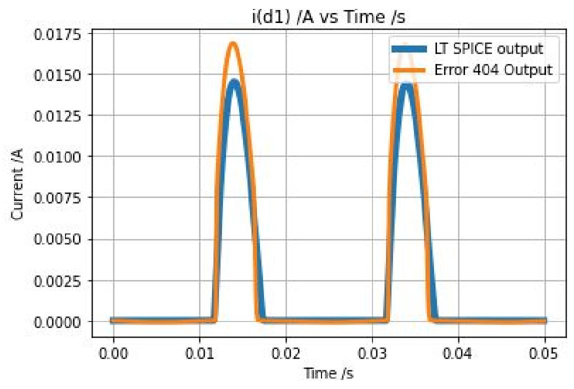


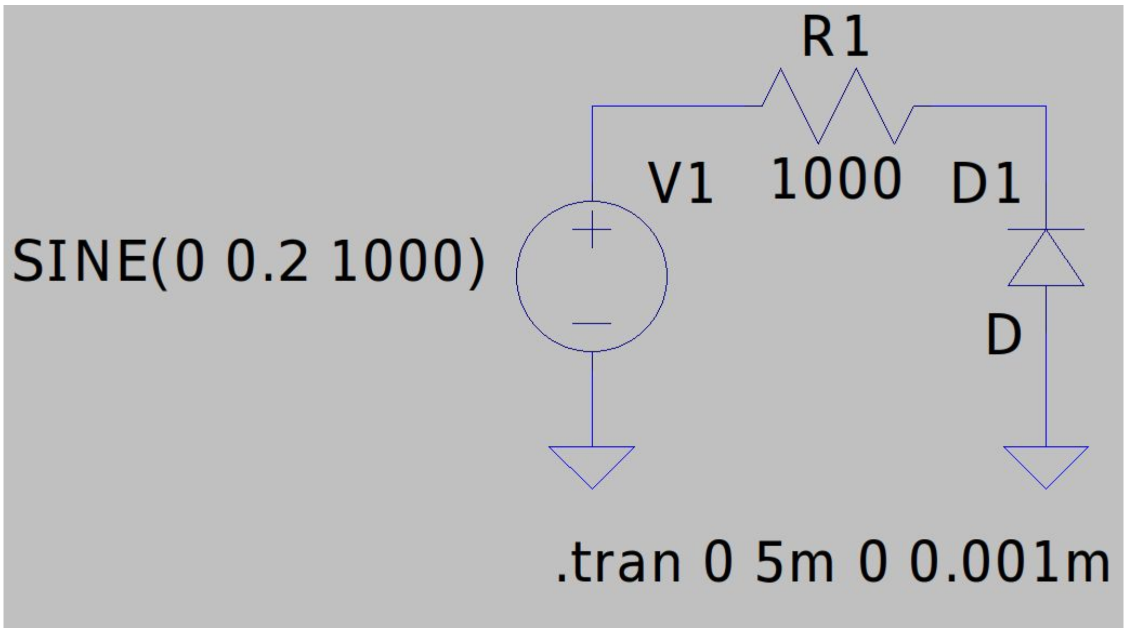
Figure 6

In conclusion, although the specification's demand for precision was not met for this test, it was deemed not a failure of the diode model, but rather a difference in non-ideal characteristics between this project and LTspice's model. In regards to speed, the specification's demand is met.

6.3.5 Test 5 - Parasitic Capacitance Properties

Interactive graph - https://jjlehner.github.io/404CircuitSim/parasitic

In order to demonstrate the modularity of the design, the team desired that some non-ideal characteristics should be added into the diode model. This came in the form of adding some series resistance as well as adding some parasitic capacitance. This test demonstrates the successful integration of parasitic capacitance into the simulator's design. The following circuit diagram has a voltage source of amplitude 0.2V. This amplitude is too small to allow for forward conduction but allows capacitive effects to be observed.



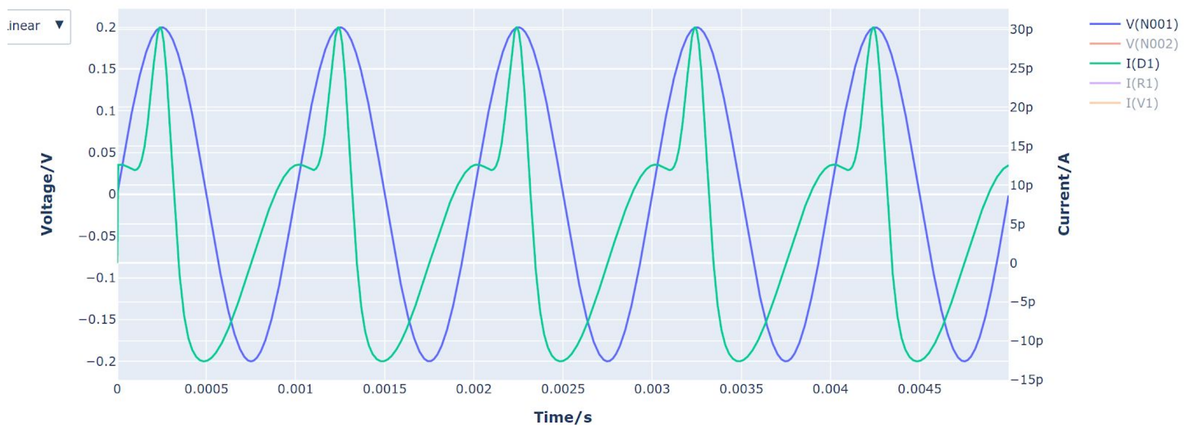


Figure 2
V(n001) is voltage across diode
I(D1) is current through diode

Error 404 Simulator Times:

> Average time: 140333.550 us (~7477.423 us)
> Fastest time: 134099.970 us (-6233.580 us / -4.442 %)
> Slowest time: 177966.970 us (+37633.420 us / +26.817 %)
> Median time: 137996.470 us

The results from figure 2 demonstrate the circuit in figure 1 having some voltage-dependent reactive properties. This is evident from the fact the current moves in and out of phase with the voltage across it. When the voltage across the diode is most negative, one would expect that the capacitor should have a capacitance of:

$$C = \frac{C_{j0}}{\sqrt{1-\frac{vd}{V0}}} = 9.13 * 10^{-15}$$

Values for Cj0 and v0 can be found in the appendix. This equation was taken from Professor Mitcheson's lectures [6] on Spice internals.

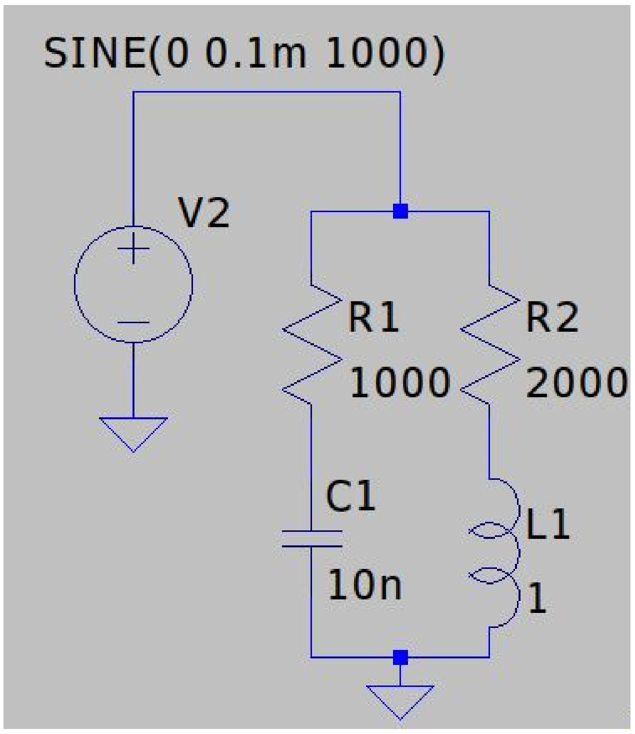
Doing phasor analysis on the circuit above, one would expect the phase angle between voltage and current to be 1.57 radians for the capacitance value calculated. This circuit has a source frequency of 1000Hz meaning that this phase angle would correspond to a phase shift of 250us. The maximum phase difference in figure two, found at -0.2 V, is 254us. A difference of 4us is acceptable here for the level of precision defined by the specification.

In conclusion, implementation of parasitic capacitance functions correctly for this circuit, within the bounds of the specification. Simulation runtime is also acceptable.

6.3.6 Test 6 - LCR Multiple Phase Circuit

Interactive graph - https://jjlehner.github.io/404CircuitSim/phase

In the initial debugging stage and qualitative tests of the simulator, it was noticed that the simulator struggled terribly with circuits that had multiple phases. The tending of some circuit parameters to zero while others might have been near or at their peak caused floating-point arithmetic error within the Symbolic C++ library. The switch to the Eigen linear algebra library fixed these issues and led to a tremendous speed increase. As a result of these early bugs in the simulator, it was essential to test how the simulator would respond in a circuit which deliberately caused signals out of phase with each other. The simulator also needed to be tested in its handling of inductor based components.

 SINE(0 0.1m 1000) V2 R1 1000 R2 2000 C1 10n L1 1	The circuit to the left was designed to have many signals of current and voltage out of phase with each other. The capacitor and inductor perform this through their reactive properties.

Error 404 Simulator Times:

Average time: 428684.608 us (~21144.252 us)
Fastest time: 386728.928 us (-41955.680 us / -9.787 %)
Slowest time: 475816.928 us (+47132.320 us / +10.995 %)

Median time: 426364.928 us

LTspice Simulation Run Time - 0.037 seconds.

LTspice performs this circuit much faster than achieved by the project’s simulator - 0.392 faster. This simulation is still within the specification demands of less than 1 second for “simple circuits” (simple circuits are defined in specification)

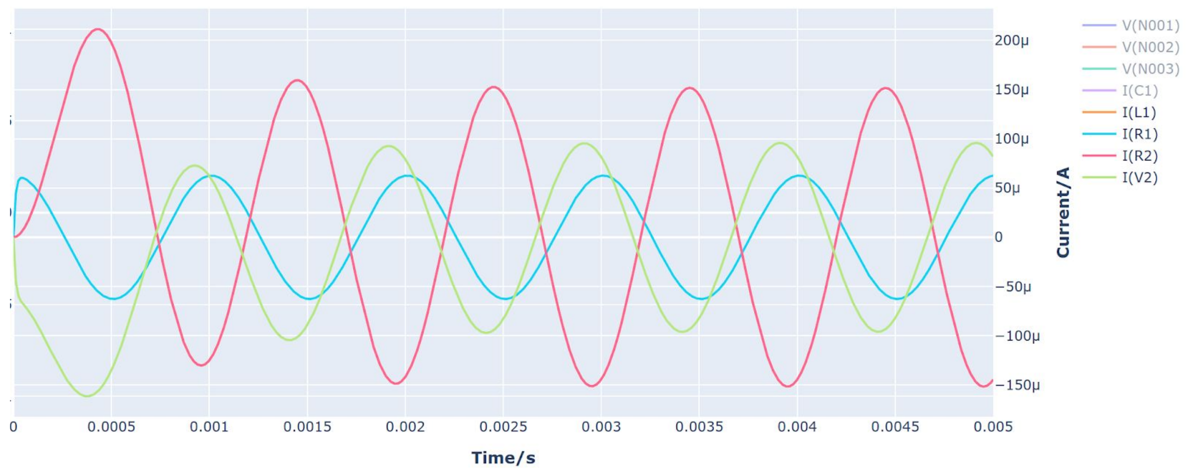
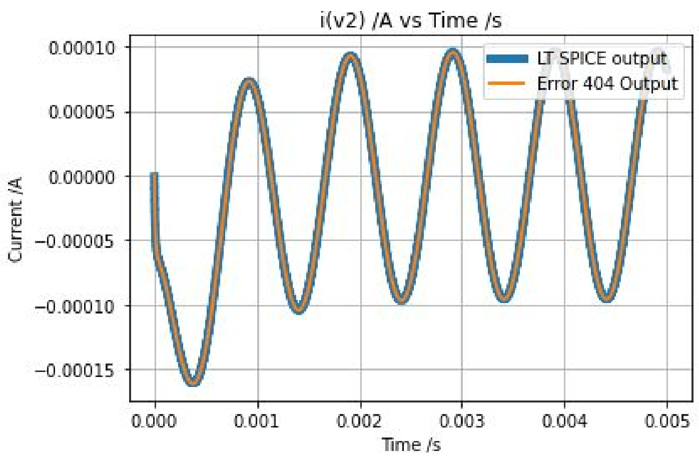
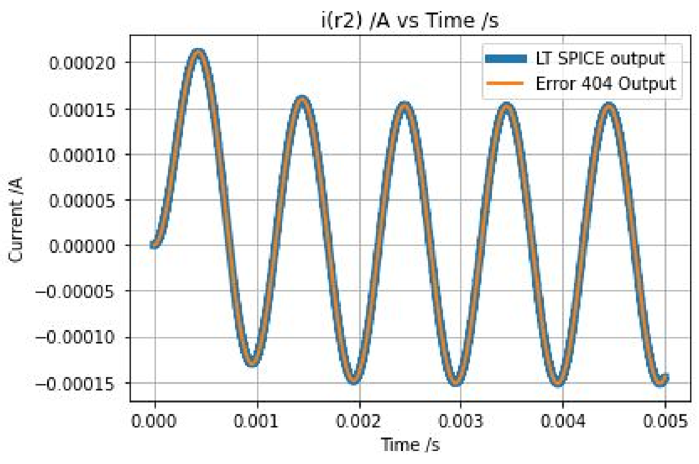
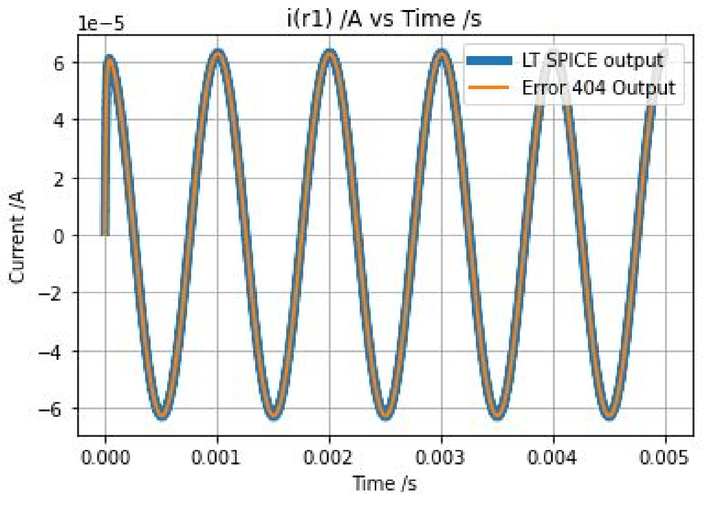


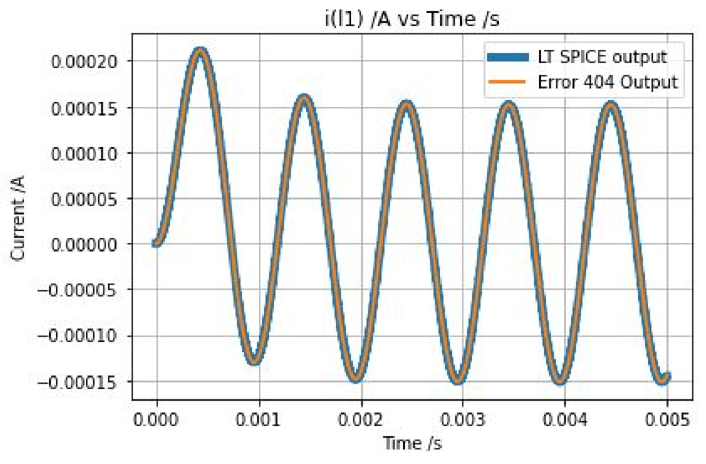
Figure 2 - Interactive graph output https://jjlehner.github.io/404CircuitSim/phase

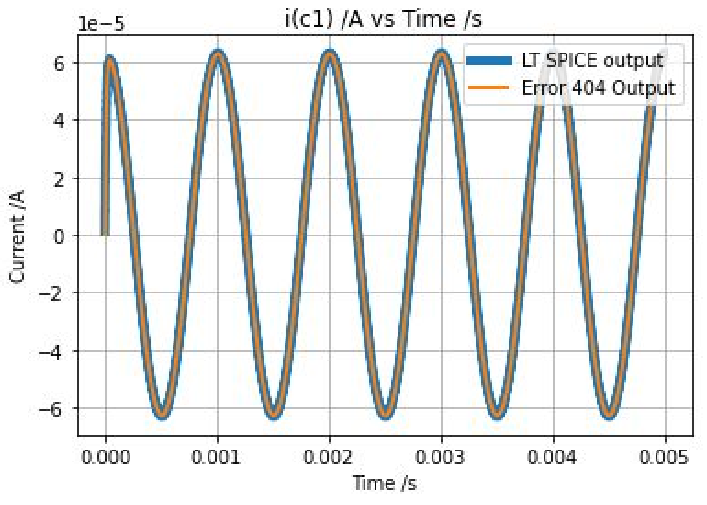
Figures 3-10 below, demonstrate the high performance of the simulator with linear component modelling. As one can observe, the orange line of the simulator output traces the blue line produced by LTspice accurately. Critically, the simulator not only handles the fact that many of the signals are out of phase, as demonstrated by figure 2, but it also manages to capture the very noticeable early transient effects, before stabilising into the steady-state solution. For this circuit, the simulator’s linear component modelling is performing very well, and simulation runtime is within the specification requirements.

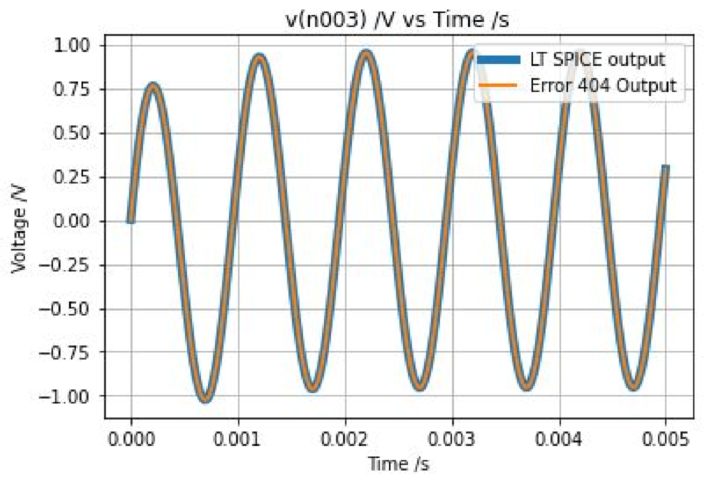


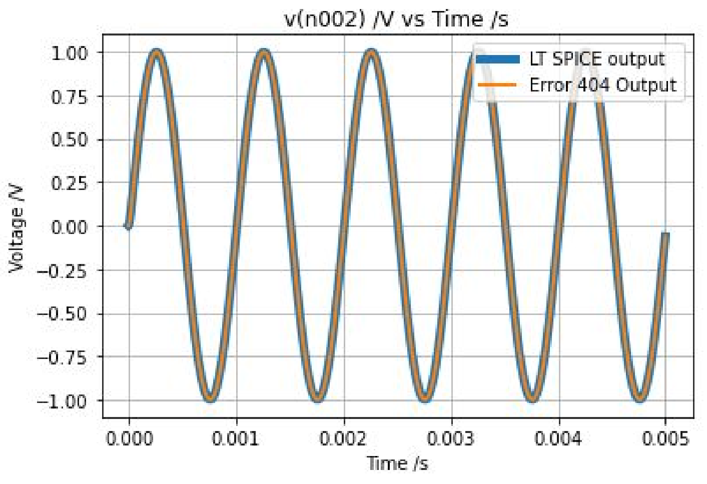


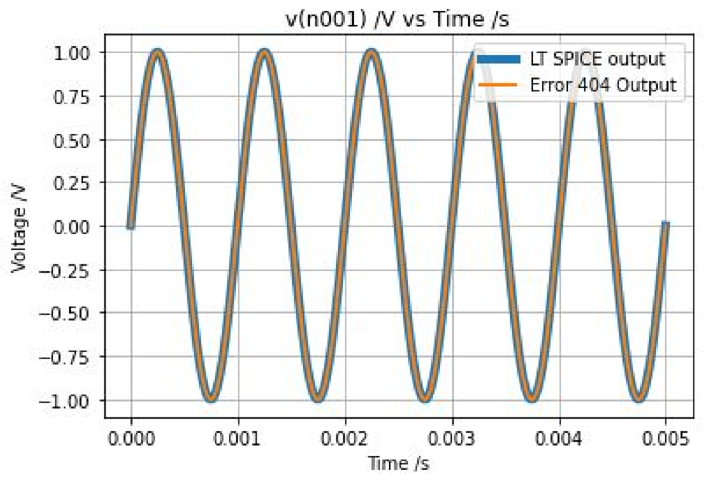

i(r1) /A vs Time /s
1e−5
Current /A
Time /s
LT SPICE output
Error 404 Output


i(l1) /A vs Time /s
Current /A
Time /s
LT SPICE output
Error 404 Output


i(c1) /A vs Time /s
1e−5
Current /A
Time /s
LT SPICE output
Error 404 Output


v(n003) /V vs Time /s
Voltage /V
Time /s
LT SPICE output
Error 404 Output

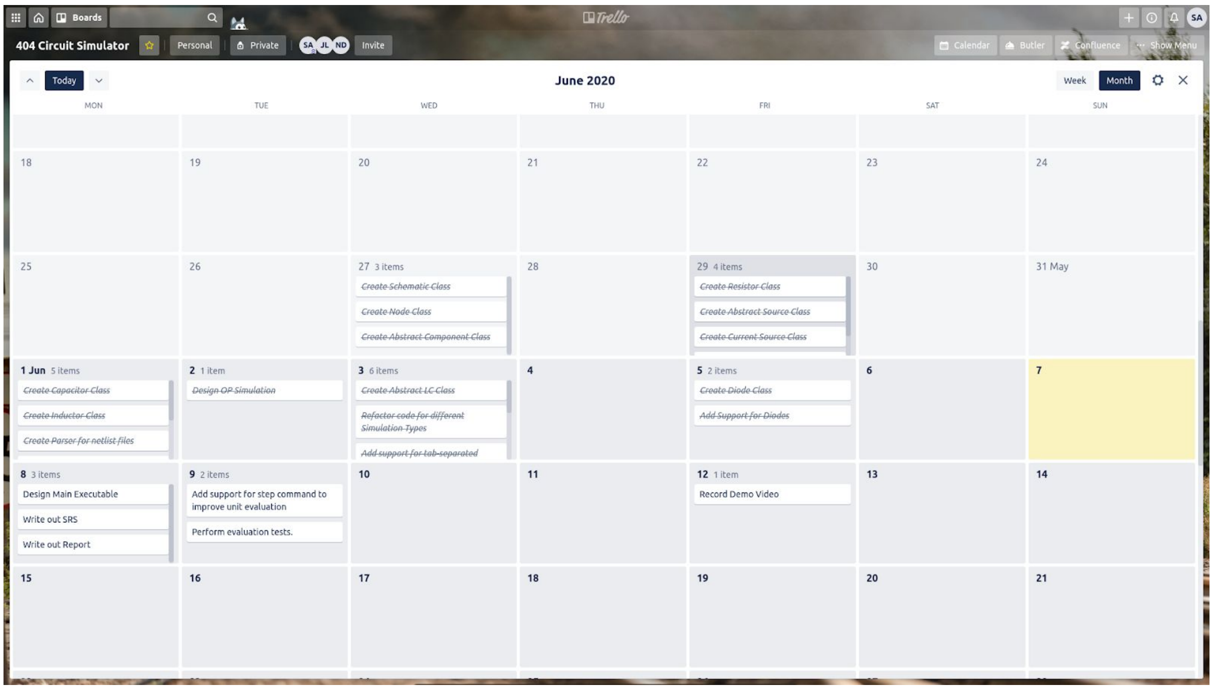

v(n002) /V vs Time /s
Voltage /V
Time /s
LT SPICE output
Error 404 Output


v(n001) /V vs Time /s
Voltage /V
Time /s
LT SPICE output
Error 404 Output

7. Project Planning and Management

7.1 Trello Boards

The team decided to go with Trello as its project management system, where lists were created to represent the subtasks. Each member then assigned themselves to various cards in each list and gave each card due dates. See appendix for full board structure.



MON	TUE	WED	THU	FRI	SAT	SUN
18	19	20	21	22	23	24
25	26	27 3 items ~~Create Schematic Class~~ ~~Create Node Class~~ ~~Create Abstract Component Class~~	28	29 4 items ~~Create Resistor Class~~ ~~Create Abstract Source Class~~ ~~Create Current Source Class~~	30	31 May
1 Jun 5 items ~~Create Capacitor Class~~ ~~Create Inductor Class~~ ~~Create Parser for netlist files~~	2 1 item ~~Design OP Simulation~~	3 6 items ~~Create Abstract LC Class~~ ~~Refactor code for different Simulation Types~~ ~~Add support for tab-separated~~	4	5 2 items ~~Create Diode Class~~ ~~Add Support for Diodes~~	6	7
8 3 items Design Main Executable Write out SRS Write out Report	9 2 items Add support for step command to improve unit evaluation Perform evaluation tests.	10	11	12 1 item Record Demo Video	13	14
15	16	17	18	19	20	21

Trello Calendar as at 12:41 PM June 7th 2020.

7.2 Code Collaboration

The team used Git Version Control and Github (https://github.com/jjlehner/404CircuitSim) to host the repository online, leveraging feature branches to prevent code conflicts as much as possible. The team also took advantage of Visual Studio Code's Live Share and Atom's Teletype to help each other with coding challenges in real-time.

7.3 Communication Channels

The group created a Microsoft Teams® page and set up daily recurring meetings where progress reports and challenges faced were shared and discussed. The group also used teams to share relevant documents and files and had a GitHub bot integrated into the channel which provided a notification whenever there was an issue or pull request.

8. Conclusion and Future Extensions

Overall, the simulator matches the points demanded by the specification for linear components and diodes for the tests performed. With diodes, however, the simulator was unable to compete with the speed of LTspice's simulation. The simulator is still currently a Work In Progress, which is currently incomplete on transistor and MOSFET implementation. On reflection, the simulator would be able to quickly implement transistor simulations, as the API has support for 3-terminal components. The API also allows for the component hierarchy to be expanded with a transistor or MOSFET class, which would inherit its behaviour from the component class with any specific transistor functions being overridden. The iterative simulator used for diodes was designed to be as general as possible (the nonlinear optimiser has no understanding of diode behaviour or knowledge of the Shockley equations). As a consequence, it can be easily extended to support the nonlinear equations produced by transistors.

Another extension would be to modify the iterative technique to iterate to a tolerance, rather than to a fixed number of iterations. This would vary the compute time, for simpler circuits this might shorten the runtime, whereas in more complicated cases the runtime might increase. This would be rather simple to implement as the simulator could simply already know the magnitude of the voltage error vector.

Throughout the design of the project code structure, care was taken to make the simulator as modular as possible so that it would be easy to add extra functionality if time permitted. (Notably through an object-orientated component hierarchy). Simulation of more non-ideal components would also be impressive - this could include series resistance of components or parasitic conductance and inductance. This behaviour would be simple to integrate into the project, as demonstrated by the integration of parasitic capacitance in diodes.

Another essential step that should be taken if time allowed would be to perform further evaluation testing. This is necessary to check that the project could meet the high standards of the industry, which is the ultimate intended use case for this simulator.

Furthermore, in future, adding multithreading support might lead to a significant decrease in runtime.

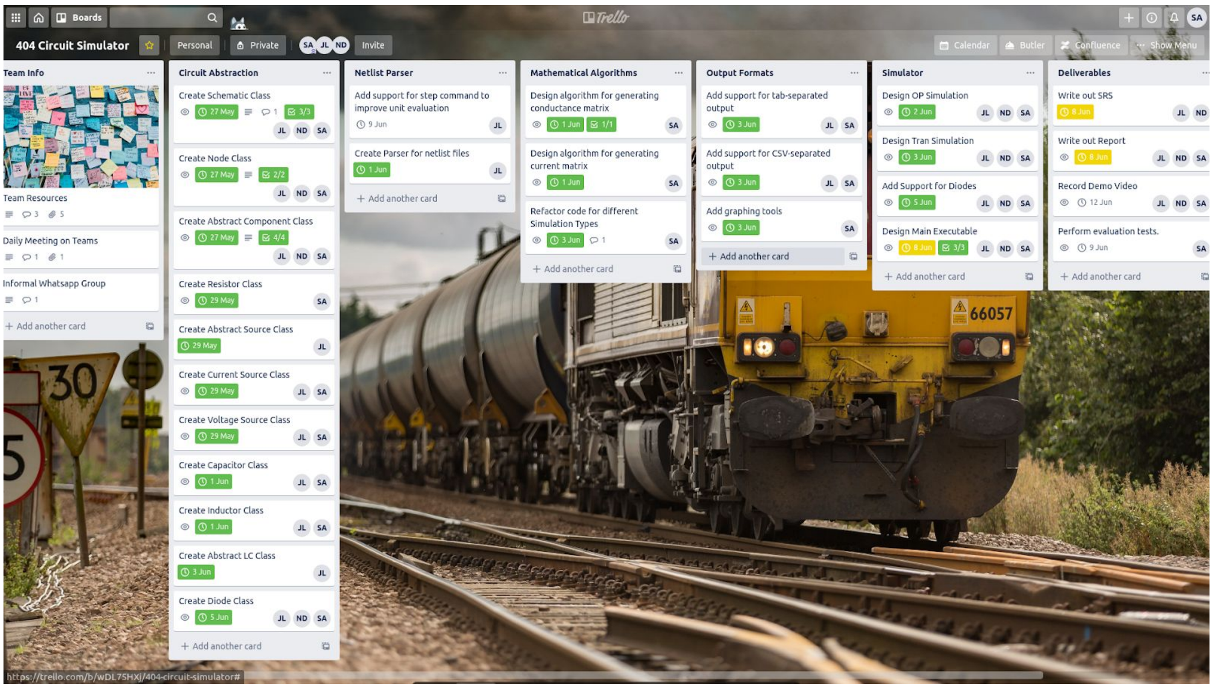
8.1 Critical Analysis and Reflection

In testing, the simulator produced results similar in precision to LTspice. It failed, however, to match LTspice's speed, but did fall within the requirements of the specification. A vital strength of the project is the simplicity of the API produced and its ease of understanding. All the data structures and models component class interactions are well-documented on the project's Doxygen page: https://jjlehner.github.io/404CircuitSim/. The API itself is very easily expanded due to the choice of making a header-only library, where the only prerequisite to adding a component is deriving an object from the component class. In reflection, it is evident that the graphing software employed by LTspice is much more lightweight than the project's produced solution. This is perhaps due to the chosen graphing libraries struggling with the large number of data points produced. It is also a certainty that the solution produced is less reliable than LTspice.

Despite efforts detailed in the report, the produced simulator has not undergone the thorough QA testing that is expected from industry tools. The most significant weakness against LTspice, however, is that the project does not seem to have the same scalability. This is evident when looking at simulation runtimes for LTspice on 1000 node analysis compared to this project. This perhaps could be improved by taking advantage of multithreading the simulator's processes and implementing other matrix solvers which can deal with larger matrices more efficiently.

9. Appendix

9.1 Trello Board



Team Info
- Team Resources — 3 comments, 5 attachments
- Daily Meeting on Teams — 1 comment, 1 attachment
- Informal Whatsapp Group — 1 comment
- + Add another card

Circuit Abstraction
- Create Schematic Class — 27 May, 1 comment, 3/3 — JL ND SA
- Create Node Class — 27 May, 2/2 — JL ND SA
- Create Abstract Component Class — 27 May, 4/4 — JL ND SA
- Create Resistor Class — 29 May — SA
- Create Abstract Source Class — 29 May — JL
- Create Current Source Class — 29 May — JL SA
- Create Voltage Source Class — 29 May — JL SA
- Create Capacitor Class — 1 Jun — JL SA
- Create Inductor Class — 1 Jun — JL SA
- Create Abstract LC Class — 3 Jun — JL
- Create Diode Class — 5 Jun — JL ND SA
- + Add another card

Netlist Parser
- Add support for step command to improve unit evaluation — 9 Jun — JL
- Create Parser for netlist files — 1 Jun — JL
- + Add another card

Mathematical Algorithms
- Design algorithm for generating conductance matrix — 1 Jun, 1/1 — SA
- Design algorithm for generating current matrix — 1 Jun — SA
- Refactor code for different Simulation Types — 3 Jun, 1 comment — SA
- + Add another card

Output Formats
- Add support for tab-separated output — 3 Jun — JL SA
- Add support for CSV-separated output — 3 Jun — JL SA
- Add graphing tools — 3 Jun — SA
- + Add another card

Simulator
- Design OP Simulation — 2 Jun — JL ND SA
- Design Tran Simulation — 3 Jun — JL ND SA
- Add Support for Diodes — 5 Jun — JL ND SA
- Design Main Executable — 8 Jun, 3/3 — JL ND SA
- + Add another card

Deliverables
- Write out SRS — 8 Jun — JL ND
- Write out Report — 8 Jun — JL ND SA
- Record Demo Video — 12 Jun — JL ND SA
- Perform evaluation tests. — 9 Jun — SA
- + Add another card

https://trello.com/b/wDL75HXj/404-circuit-simulator#

Trello Board as at 12:41 PM June 7th 2020.

9.2 Netlists Used for Testing and Evaluation of Simulator.

9.2.1 Simple RC Test

```
* Simple RC Test
V1 N002 0 PWL(0.00 0.0 0.0000001p 5)
R1 N001 0 {R}
C1 N001 N002 10µ
.STEP PARAM R 100 1000 100
.tran 0 10m 0 0.01m
.backanno
.end
```

9.2.2 Single Diode Test

```
* Single Diode Test
D1 N001 N002 D
R1 N002 0 1000
V1 N001 0 SINE(0 5 1)
.model D D
.tran 0 3 0 0.01
```

```
.backanno
.end
```

9.2.3 Double Diode Cross Over Distortion Test

```
* Double Diode Test
D1 N001 N002 D
D2 N002 N001 D
V1 N001 0 SINE(0 2.5 1)
R1 N002 0 1000
.model D D
.tran 0 2 0 0.01
.backanno
.end
```

9.2.4 Full Wave Rectifier

```
* Full Wave Rectification
D1 0 N001 D
D2 N001 N002 D
D3 0 N003 D
D4 N003 N002 D
V1 N001 N003 SINE(0 10 50)
R1 N002 0 1000
C1 N002 0 10µ
.model D D
.tran 0 50m 0 0.005m
.backanno
.end
```

9.2.5 Parasitic Capacitance Properties

```
* Parasitic Test
V1 N001 0 SINE(0 0.2 1000)
D1 N002 0 D
R1 N002 N001 1000
.model D D
.tran 0 5m 0 0.001m
.backanno
.end
```

9.2.6 LCR Multiple Phase Circuit

```
* Multi Phase Circuit
R1 N001 N003 1000
R2 N001 N002 2000
C1 N003 0 10n
L1 N002 0 1
```

```
I1 0 N001 SINE(0 0.1m 1000)
.tran 0 5m 2m 0.001m
.backanno
.end
```

9.3 List of Accepted Commands and Components

Extensions to SPICE format:

- Allows multiple types of simulations per netlist. e.g. a .tran and .op could both run at once on the same netlist

9.3.1 Accepted Components

Format: <designator> <node0> <node1> <value>

Component	Designator	Node Order	Value
Voltage Source	V	+, -	Constant or SINE function
Current Source	I	out, in	Constant or SINE function
Resistor	R	N/A	Ohms
Capacitor	C	N/A	Farad
Inductor	L	N/A	Henry
Diode	D	Anode, Cathode	Model (D)

9.3.2 Accepted Prefixes (to use with values)

Multiplier	Power of 10
p	-12
n	-9
u	-6
m	-3
k	3
Meg	6
G	9

9.3.3 Accepted Commands

- .OP: dc operating point
  - Usage: .op
- .TRAN: time domain analysis
  - Usage: .tran 0 STOP SAVE STEP
- .STEP: repeat simulation for multiple values of parameter
  - Linear Mode: .step param VAR START END INC
  - Octave Mode: .step oct param VAR START END POINTS_PER_OCTAVE
  - Decade Mode: .step dec param VAR START END POINTS_PER_DECADE

9.4 Circuit Simulator Manual

When running simulator -h, a manual will be printed out as such:

```
-i     <file>      path to input netlist
-o     <dir>       path to output directory
-f     <format>    specify output format, either csv or space
-p     <list>      plots output, list specifies columns to plot
-c                 shows names of columns in output file, blocks -p
-h                 shows this help information

Usage: simulator -i file -p list [ -ch ] [ -o dir ] [ -f format ]

Examples

Plot Specific Columns:
      simulator -i test.net -p 'V(N001) V(N002)'

Plot All Columns:
simulator -i test.net -p ''
```

9.5 Valgrind Output

```
==3978== HEAP SUMMARY:
==3978==     in use at exit: 0 bytes in 0 blocks
==3978==   total heap usage: 205,683 allocs, 205,683 frees, 96,285,617
bytes allocated
==3978==
==3978== All heap blocks were freed -- no leaks are possible
==3978==
==3978== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

9.6 Diode Parameters

```
double inst_conductance = 0;
double IS = 1e-14;
double RS = 0;
double CJ0 = 1e-14;
double TT = 0;
double BV = 100;
double IBV = 0.1e-10;
double VJ = 1;
const double GMIN = 1e-5;
const double V_T = 25e-3;
```

10. References

[1] H. Lipschitz, "A Simple C++ Project Structure," July 3rd, 2013.
[2] (). *Doxygen Documentation*. Available: https://www.doxygen.nl/manual/index.html.
[3] (). *SymbolicC++ Documentation*. Available: https://issc.uj.ac.za/symbolic/symbolic.html.
[4] (). *Eigen Documentation*. Available: http://eigen.tuxfamily.org/dox/index.html.
[5] (). *Easy C++ bench-marking*. Available: https://bruun.co/2012/02/07/easy-cpp-benchmarking.
[6] Anonymous "SPICE Diode and BJT models," .
[7] (). *Plotly Python Graphing Library*. Available: https://plotly.com/python/.
[8] Anonymous "Newton's method," 2020. Available: https://en.wikipedia.org/w/index.php?title=Newton%27s_method&oldid=961349456.
[9] Anonymous (). *c++ - gcc optimization flag -O3 makes code slower than -O2*. Available: https://stackoverflow.com/questions/28875325/gcc-optimization-flag-o3-makes-code-slower-than-o2.
[10] (). *Valgrind Documentation*. Available: https://www.valgrind.org/docs/manual/index.html.
[11] Anonymous (). *Power profiling overview*. Available: https://developer.mozilla.org/en-US/docs/Mozilla/Performance/Power_profiling_overview.
[12] Anonymous (-08-27T07:01:21+00:00). *RC Charging Circuit Tutorial & RC Time Constant*. Available: https://www.electronics-tutorials.ws/rc/rc_1.html.
[13] A. Siritaratiwat *et al*, "A study of noise effects due to the diode protection for shield resistance measurement of GMR recording heads," *Tmag,* vol. 41, *(10),* pp. 2941-2943, 2005. Available: https://ieeexplore.ieee.org/document/1519167. DOI: 10.1109/TMAG.2005.855324.
[14] (). *LTspice: Simple Idealized Diode | Analog Devices*. Available: https://www.analog.com/en/technical-articles/LTspice-simple-idealized-diode.html.