

1ST YEAR PROJECT REPORT

ELEC40006 – Electronics Design Project 1 (Circuit Simulator)

Staal, Simon T A

Dr Edward A Stott

Mrs Esther Perea

Simon Staal – CID 01719944

Salman Dhaif – CID 01722410

Ruwan Silva – CID 01505106

Word Count: 9986 words

Table of Contents

1- Introduction	2
1.1- Purpose, Intended Audience and Use:.....	2
1.2- Software Scope and Description:	2
1.3- Software Features and Requirements	3
2- Project Management.....	5
2.1- Belbin Team Roles.....	5
2.2- Team Roles	7
2.3- Gantt Chart	8
2.4- Team meetings.....	10
3- Design Process	12
3.1- Initial Research.....	12
3.2- Input Management	13
3.3- Matrix initialization	15
3.4- Matrix Calculations and Processing (Initialisation Loop)	18
3.5- Further input management.....	23
3.6- Matrix Decomposition.....	25
3.7- Final Processing (Transient Loop)	27
3.8- Output.....	29
4- Testing.....	31
4.1- Troubleshooting.....	31
4.2- Functional Testing	32
4.3- Performance Testing	36
5- Evaluation and Extension	41
Bibliography.....	43
Appendix A – Eigen Benchmark of dense decompositions (milliseconds)	45
Appendix B – Series-Resistor Netlist Generator.....	45
Appendix C – Processing time test script.....	47
Appendix D – Function Test 1 Results	47
Appendix E – Function Test 2 Results.....	49
Appendix F – Function Test 3 Results	51

1- Introduction

1.1- Purpose, Intended Audience and Use:

Circuit simulators have become an integral part of analogue and digital circuit design. They allow engineers to observe and alter the performance of a circuit before constructing it. Simulators are heavily used in industry as they provide insight into possible issues with the design and allow designers to evaluate the use of different components while saving resources in the design process; this makes them a vital stage of planning for electronic device manufacturing.¹ Moreover, they have been used in education to enhance understanding of electronics by providing different types of simulations and analysis. The aim of this project is to develop software packages that can provide transient analysis for an electrical network and simulate its behaviour.

1.2- Software Scope and Description:

Users of the developed simulator will need to input a circuit alongside analysis parameters in order to obtain results that demonstrate the circuit's behaviour. The program will work by solving for nodal voltages and currents in the circuit using the equation below where G is the conductance matrix (each index G_{ij} represents the total conductance connected to those nodes, when $i=j$ then it is the total conductance connected to node j), v is the nodal voltage vector and i is the vector for current at each node.

$$G \cdot v = i$$

The above equation essentially represents the system of equations that satisfy Kirchoff's current law for the circuit. Following the input, the program will construct and solve conductance, voltage and current matrices (as seen in figure 1) in order to obtain nodal voltages and currents in the circuit; this will be done for each instant of the simulation as specified by the user in the input. As the program simulates the circuit's transient response, it will demonstrate the response of components to the sudden change of turning on the circuit. This can be calculated numerically and hence the simulation needs to specify a duration and

¹ Sciencedirect.com. 2020. *Circuit Simulation - An Overview* | *Sciencedirect Topics*. [online] Available at: <<https://www.sciencedirect.com/topics/computer-science/circuit-simulation>> [Accessed 14 May 2020].

number of instances where values are calculated.² The results of the simulation will then be graphically displayed.

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

Figure 1: Matrix form of circuit's system of equations

1.3- Software Features and Requirements

The developed software has certain requirements that dictate how it will be designed. These can be broadly split into functional and non-functional requirements. Functional requirements provide insight as to what the software will do; the functional requirements in this case are:

- The program should be able to analyse an input netlist file (text file) which specifies a circuit in reduced SPICE format (seen in figure 2), which will specify the components of a circuit and how they are connected. The component and its name can be seen in the first string of the line, the next two strings are the nodes which it is connected to, followed by the value of the component. The file will specify a timestep and stop time as well. This will be specified in the second to last line where “10ms” represents the stop time and “1us” is the timestep. As well, the “.end” on the final line to indicate the end of the source file.

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

Figure 2: Example reduced SPICE netlist

² Hayt, W., Kemmerly, J. and Durbin, S., 2011. *Engineering Circuit Analysis*. 8th ed. McGraw-Hill Education, c. 8.

- A transient response analysis will be conducted for the circuit parsed in, with calculations for voltages and currents occurring at each timestep up until the stop time as specified in the input netlist file.
- The results will be outputted to a CSV (*comma-separated values*) file where the rows are the instances in time and the columns are all the nodes and components (values will represent their nodal voltages and instantaneous current). Following that, a MATLAB script will be used in order to plot and visualize the results onto a graph.
- When an invalid simulation or circuit is specified a detailed error message will be written to cerr: the standard error stream.

As for non-functional requirements, they evaluate how the system will perform. For our simulator:

- The final software is expected to match the accuracy and reliability of a transient simulation conducted on the analogue circuit simulator software LTSpice IV.
- Any inaccuracies in calculations should be under $10e-15$.
- Processing simple circuits should take under a second.
- Simulations are expected to be capable of processing circuits of up to 100 nodes.
- Developed packages will be written in C++17, using a wide range of libraries, and aim to be compatible with a variety of operating systems.

2- Project Management

Once functional and non-functional requirements had been decided, a communication plan was handled next. Discord would be the mode of communication used. This was decided due to the ease of sharing information and ability to call and screen share work with each other. Two meetings a week lasting around an hour were planned to discuss progress and future action. During these meetings a team member noted down all details of discussions to keep a clear record of project plans. Collaborative write-up on the project was done using a shared OneNote file to compile initial research and a word document on OneDrive to write the report. OneDrive and OneNote were chosen as they have been used previously by team members and are made for working on the same documents. A shared repository on Github was used so the team could work simultaneously on the coding files for the project. This was convenient as the second half of this year team members became familiarised with Github while progressing through the programming module of the course.


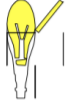


2.1- Belbin Team Roles



The 'Belbin Team Roles' are nine groupings of behaviours that are recognised as different profiles for members of a high performing team. As described on the Belbin website: "By using Belbin, individuals have a greater self-understanding of their strengths, which leads to more effective communication between colleagues and managers. Great teams can be put together, existing teams can be understood and improved, and everyone can feel that they are making a difference in the workplace."³ After completing the Belbin Self-Perception Inventory, the team's strengths and weaknesses were analyzed before deciding roles for this project. The two strongest and weakest roles of each team member are listed in this table below:

³ Belbin.com. 2020. *The Nine Belbin Team Roles*. [online] Available at: <<https://www.belbin.com/about/belbin-team-roles/>> [Accessed 20 May 2020].

	Simon	Salman	Ruwan
Strongest Roles	Completer Finisher	Plant	Monitor Evaluator
	Shaper	Completer Finisher	Teamworker
Weakest Roles	Plant	Resource Investigator	Shaper
	Teamworker	Monitor Evaluator	Completer Finisher

The table demonstrates that this is a diverse, but balanced group. This is displayed by Simon's two strongest roles being Ruwan's two weakest, while Simon's weakest roles are the strongest roles for Ruwan and Salman. This shows that the group compliments each other very well with virtually every members' weakest roles being one of the strongest for another team member. For a detailed look at the strengths and weaknesses that these roles describe refer to the figures below:

	SHAPER (SH)	<p>The driver of activity</p> <p>Strengths: Energetic, driven, bold. Challenges inertia, ineffectiveness and complacency in the team. Good at leading start-up or rapid-response teams.</p> <p>Weaknesses: Can offend people, is impatient and easily provoked. May lose sense of humour under pressure.</p>
	PLANT (PL)	<p>The creative engine-room</p> <p>Strengths: Unorthodox, knowledgeable and imaginative, turns out lots of radical ideas, good when everyone is stuck.</p> <p>Weaknesses: Needs careful handling to be effective. Individualistic, disregards practical details and protocol, can lose touch with reality and become an unguided missile.</p>
	RESOURCE INVESTIGATOR (RI)	<p>The enterprising extrovert</p> <p>Strengths: Outgoing, great energy, great motivator, good connections outside the team. Enjoys making new contacts, exploring new ideas, responds well to challenges.</p> <p>Weaknesses: Can be noisy, over-optimistic, lose interest and doesn't follow through. Can be lazy unless under pressure.</p>
	MONITOR EVALUATOR (ME)	<p>The objective observer</p> <p>Strengths: Unemotional, careful and discreet. Sees all options. Good at assessing proposals, using judgement, monitoring progress and preventing mistakes.</p> <p>Weaknesses: May not motivate others, may appear slow to act, seen as negative, critical and cynical.</p>

	TEAMWORKER (TW)	<p>The people supporter</p> <p>Strengths: Sensitive to others, perceptive, listens well, provides informal communication and support, is diplomatic and prevents feuding and fragmentation.</p> <p>Weaknesses: May be indecisive, avoids conflict, too concerned about keeping people happy to get the job done on time.</p>
	COMPLETER FINISHER (CF)	<p>The orderly perfectionist</p> <p>Strengths: Makes sure the team delivers, maintains a sense of urgency that can help the team. Polishes and perfects things. Good at follow-up and meeting deadlines.</p> <p>Weaknesses: Can be over-anxious, obsessively worrying about everything and reluctant to delegate.</p>

2.2- Team Roles

After discussion of strengths and weaknesses, these roles/focuses were decided: Simon – Head software architect and lead researcher, Salman – Technical lead and project vision, Ruwan –

Technical lead and timeline management. Simon was given these roles as his strengths included polishing and perfecting things, being driven and leading start-up/rapid response teams. These qualities meant that he would suit leading initial research as well as ensuring that we deliver a finished and polished simulator by leading the team's software implementation. Salman's strengths included being imaginative and knowledgeable as well as maintaining a sense of urgency and ensuring the team delivers. This meant that he was perfectly suited to look over technical calculations and watch over the vision of the project as he kept notes from team meetings. Ruwan's qualities include being perceptive and listening well and monitoring progress and preventing mistakes. He would monitor calculations helping prevent errors and keep track of progress to keep in line with the timeline.

	Simon	Salman	Ruwan
Team Roles	Head Software Architect	Technical Lead	Technical Lead
	Lead Researcher	Project Vision	Project Timeline

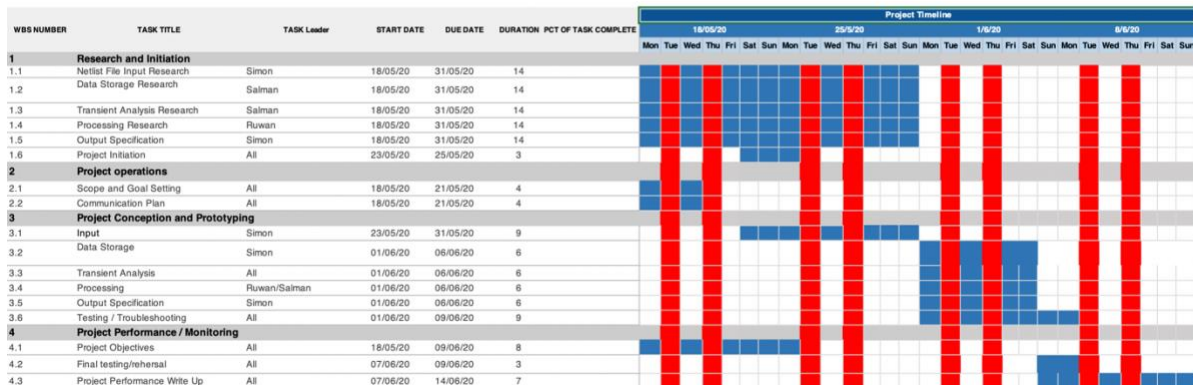
2.3- Gantt Chart

Part of project management is planning out a timeline of the project. A useful and commonly used method for this is a Gantt chart. A Gantt chart allows the user to display various tasks, when each task begins and ends, the duration of each task and how they overlap with each other. This is done by listing tasks on the left of the chart and dates along the top of the chart. Each task is represented by a bar with the position and length of the bar showing the duration as well as the start and end date of the task. It was decided that there would be an initial Gantt chart to plan out the ideal approach to the project and a separate one to track the actual progress of the project. This would improve evaluation of the project. Below is the initial Gantt chart:

YTB INITIAL GANTT CHART

PROJECT TITLE Transient Simulator
PROJECT MANAGER Simon Staal / Salman Chail / Ruwan Silva

COMPANY NAME Yeah The Boyz
DATE OF CREATION 09/05/20



On the left tasks were split up into 4 sections and assigned a task leader. The blue blocks depict when tasks are to be executed and red blocks are scheduled team meetings. The first section would ideally be complete by the start of the second week, so research could be implemented by the start of June and coding would become the focus for the next week. The project operations section was planned to finish before the end of the first week as it is important to set goals and a communication plan early on. Project conception and implementation was to be completed in the first week of June, so there would be a week to write up the finished report and complete some final tests. This seemed a realistic timescale as research should be completed by this stage and implementation could take a week alongside constant testing and troubleshooting. The final section mostly pertains to finishing the project write up and evaluation of the software package. It was thought that leaving the final week open for this was plenty of time to complete the report as well as conduct some final tests. However, the writing of the report would be started once initial research had been finished and would be updated as things progressed.

times for the next meeting and collaborative work to be done. An example of this is organising meeting times for multiple team members to troubleshoot passages of code together (Norming). Finally the team engaged in informal discussions to maintain team morale and make sure all members were motivated (Performing). As meetings were expected to last over an hour, a single member was assigned the task of documenting meeting minutes. This would help keep track of completed tasks and what still needs to be done hence giving a sense of awareness as to how much progress has been made.

3- Design Process

3.1- Initial Research

The design of the project was approached by considering several key points:

- What operations was the program going to perform?
- How would the required information be stored to perform the necessary operations?
- How would the information be extracted from the input file?
- How would the information produced by our program be outputted?

From the initial spec provided by the department, it was clear that the bulk of operations would rely on matrix manipulation, as a matrix equation in the form $[G][v] = [i]$ needed to be generated and solved. Since the C++ standard library does not contain any headers for matrix manipulation⁵, several specialized C++ libraries were considered.

GMTL⁶

This library is specially designed for graphics engines, but lacks general purpose matrices, matrix decomposition and solving. These limitations made the GMTL library unsuitable for this project.

Eigen⁷

Eigen is lightweight and versatile, meeting all the requirements regarding matrix manipulation and solving, with a variety of decompositions with benchmarks available to compare the effectiveness of their various functions. Its only dependencies are on the C++ standard library and it is a header-only library, meaning that it requires minimal installation time. Ultimately this was the library chosen for this project, as it seemed to be the most suited for the processing the program requires.

⁵ En.cppreference.com. 2020. *C++ Standard Library Headers - Cppreference.Com*. [online] Available at: <<https://en.cppreference.com/w/cpp/header>> [Accessed 24 May 2020].

⁶ Bierbaum, A., Meinert, K. and Scott, B., 2002. *GMTL Programmer's Guide*. [online] Ggt.sourceforge.net. Available at: <<http://ggt.sourceforge.net/gmtlProgrammersGuide-0.6.1-html/index.html#d0e36>> [Accessed 25 May 2020].

⁷ Jacob, B. and Guennebaud, G., 2020. *Eigen*. [online] Eigen.tuxfamily.org. Available at: <http://eigen.tuxfamily.org/index.php?title=Main_Page> [Accessed 25 May 2020].

IMSL⁸

Whilst IMSL also had all the functions and data structures needed for matrix manipulation, and is extremely fast, it requires a commercial license to use, so it was not selected.

Different data types, such as graphs, were considered to represent the circuit within the program, but it was quickly decided to be unnecessary, as values could be directly updated in the conductance matrix and current vector as the input netlist was processed. The chosen implementation was to store the netlist in a `vector<vector<string>>`, with each stored vector representing a line of the netlist, so each line could be evaluated to construct the matrices required to perform the transient analysis.

Therefore, extracting and storing information from the input file required no further research as the rest of the knowledge needed was provided in the programming module. Both the input and output would be done using the standard `cin/cout` `iostream`, allowing users to specify which files were to be used as inputs and where an output should be stored.

3.2- Input Management

Reading from a netlist input file required a model like the one used in one of the Q4 programming labs, creating a class `NetlistReader` (fig. 1) which contains a reference to an `istream`, and a method `get_line()`, which returns a vector containing a line of the netlist file, separated into strings. The method `get_line()` should also skip commented lines by checking the first character of a line and discarding it if it is `'*'`, and telling the program it has all the information by returning an empty vector if the line is `'end'`. The character extraction is done using `std::getline` overload contained in the `<string>` header⁹, which also stops extraction if the end of the file is reached. This means that a user could potentially input an invalid netlist without including a `'end'` at the end of the file. In order to prevent such issues, as well as other

⁸ Imsl.com. 2020. *IMSL C Library | High-Impact Functions For C/C++ Applications*. [online] Available at: <https://www.imsl.com/products/imsl-c-libraries> [Accessed 25 May 2020].

⁹ Cplusplus.com. 2020. *Getline (String) - C++ Reference*. [online] Available at: <http://www.cplusplus.com/reference/string/string/getline/> [Accessed 26 May 2020].

invalid syntax, it was specified that if the extracted string does not contain a ' ' (space) character and is not '.end', the reader will notify the user it has detected a syntax error and will exit the program with an error code. This means that if '.end' is never specified, getline() will eventually return an empty string, triggering the condition and exiting the program. Whilst the rest of input management has a stronger reliance on the user providing netlists with valid syntax, adding code which detects such errors was not a priority, and would be something considered in the final stages of polishing.

```
11  class NetlistReader
12  {
13  private:
14      istream &src;
15
16  public:
17      /*Creates a new netlist reader using the given input stream
18      To create an instance: NetlistReader <name>(cin);
19      */
20      NetlistReader(istream &src);
21
22      /*Reads a line of from the associated stream.
23      Should skip comment lines '*'
24      Once '.end' is reached returns an empty vector and should not be called again
25      */
26      vector<string> get_line();
27  };
```

Fig. 1: Netlist Reader

The primary goal of get_line() was to split up each line of the netlist into a format where any information required was easy to access for further processing in the simulator. It did this by pushing back each word in a line into a vector. This means that following the syntax provided in the initial specification, we could access the component type, nodes it was connected to and value at known vector indices. The method was also able to detect parentheses for the cases where voltage or current sources have a sinusoidal input, pushing back each of the values as separate strings.

At first, using this method to update the values in the conductance matrix line-by-line was considered, as this would mean that all of the information in the netlist wouldn't have to be stored, improving memory efficiency. However, this would mean that the total number of

nodes present in the circuit simulated would be unknown when constructing the matrix. This is an issue as time would have to be spent checking that the nodes included in each line are included in the matrix in every loop of processing, and would need to call `Eigen::conservativeResize()`¹⁰, an eigen method that changes the size of a dynamic matrix without changing the values of the coefficients already stored within it. This would likely be an expensive operation, and so instead it was decided that it would be more efficient to store the information contained in the netlist as a `vector<vector<string>>`, with each inner vector containing a line from the netlist. To store the information containing the simulation settings, denoted by `'.tran'`, a separate vector was used.

In order to count the number of nodes, the fact that all non-reference nodes were to be written as `'NXXX'`, where X represents some integer, was used. This means the vector elements indexed at [1] and [2] could be accessed, as this was where nodes for 2 terminal devices would be located in the vector, the `'N'` character would be removed, and the remaining value converted to an integer. Using another variable `node_max`, the largest node value found within a netlist could be tracked, and this would represent the number of nodes present in the circuit. Another limitation was that this means any nodes listed for 3-terminal devices would be ignored, but support for that would be created later if such devices were chosen to be included.

By the end of input processing, the simulator should now have a `vector<vector<string>>` input, containing all of the lines in the netlist which contain a line describing a component, a `vector<string>` `tran` containing the details of the transient analysis and an `int node_max` equal to the number of nodes in the circuit.

3.3- Matrix initialization

The next challenge was to choose suitable data types for representing the conductance matrix and voltage and current vectors. The matrix class in Eigen contains a total of 6 template parameters, but only the first 3 are mandatory:

```
Matrix<typename Scalar, int RowsAtCompileTime, int ColsAtCompileTime>11
```

¹⁰ Eigen.tuxfamily.org. 2020. *Eigen: The Matrix Class*. [online] Available at: http://eigen.tuxfamily.org/dox/group__TutorialMatrixClass.html [Accessed 26 May 2020].

¹¹ *Ibid.*

Scalar refers to the type of the coefficients stored in the matrix, and since conductance is a real value, the choice was limited to float or double. This choice would be a trade-off between speed and accuracy, as whilst doubles contain twice the number of bytes as floats, they take about twice the time to process for matrix decompositions¹², which would be the most computing intensive part of the simulator. From the benchmark of dense decompositions, it was noted that for matrices up to 100x100, a fractions of a millisecond of runtime at most would be sacrificed per timestep for using doubles over floats, depending on the decomposition used (see appendix A), so it was decided that the increased accuracy was worth the runtime cost, especially as the software may need to perform transient analysis on time steps in the nano or even pico-second range.

The immediately obvious choice for RowsAtCompileTime and ColsAtCompileTime was to set both as dynamic, as it was impossible to know the number of nodes in any given netlist at compile time. This meant that for a general matrix, the Eigen typedef MatrixXd could be used, which would create a dynamic matrix of doubles¹³. However, using fixed sized matrices for sizes smaller than 16 is hugely beneficial to performance¹⁴, and by using some of Eigen's optional template parameters, MaxRowsAtCompileTime and MaxColsAtCompileTime¹⁵, a dummy 'dynamic' matrix could be created that was initialized as a 16x16 matrix, avoiding dynamic memory allocation and giving the performance of a fixed sized matrix for circuits with 16 or fewer nodes (ignoring the reference node as it is not included in the conductance matrix). Since it was likely that there would be a large number of circuits that fit this <16 node requirement, this meant that by using a Matrix<double, Dynamic, Dynamic, 0, 16, 16>, the boost in performance could be used to run more detailed decompositions when solving for node voltages without having a huge drop in runtime performance.

This led to the use of 2 different matrix data structures, the 16x16 statically allocated matrix for circuits with less than 16 nodes, and a purely dynamic MatrixXd for circuits containing more

¹² Eigen.tuxfamily.org. 2020. *Eigen: Benchmark Of Dense Decompositions*. [online] Available at: <https://eigen.tuxfamily.org/dox/group__DenseDecompositionBenchmark.html> [Accessed 26 May 2020].

¹³ Eigen.tuxfamily.org. 2020. *Eigen: The Matrix Class*. [online] Available at: <http://eigen.tuxfamily.org/dox/group__TutorialMatrixClass.html> [Accessed 26 May 2020].

¹⁴ *Ibid.*

¹⁵ *Ibid.*

nodes. To ensure the use of the correct matrix, methods which allowed one of the matrices to be selected when processing were investigated. The first idea was to simply declare both, and then create a separate templated function to process whichever matrix chosen for use.

However, when passing an expression to a function taking a parameter of type Matrix, the expression will implicitly be evaluated into a temporary matrix due to Eigen's use of expression templates¹⁶. This means that while values are able to be read from a matrix, it is impossible to write to it, even when passing by pointer or reference. Tests were performed to confirm this was the case, but the Eigen library prevents changing any values as the matrices are defined as const.

Another alternative considered was adding a condition to the declaration of the two different matrices, only actually declaring the appropriate one for the number of nodes present in our circuit. Doing this in if statements is undesirable as the objects would fall out of scope outside the if brackets, meaning that all subsequent calculations would also have to be done in those if statements. Instead, the ternary operator was considered, which executes one of 2 expressions depending on a predefined condition¹⁷. Following the syntax provided, testing lines of the following form was done:

```
((node_max<17) ? Matrix<double, Dynamic, Dynamic,  
0, 16, 16> : MatrixXd) con
```

The aim of this was to select the type of the matrix being constructed to the variable con representing the conductance matrix, and to then use that single variable for all subsequent analysis. However, this sort of code is uncompileable due to declaration issues, and after trying different implementations, it was concluded that the C++ compiler does not allow this type of operation. This meant that the use of the 'dummy' dynamic 16x16 matrix was desired, and if statements for the 2 different

```
//Creating appropriate matrices / vectors  
Matrix<double, Dynamic, Dynamic, 0, 16, 16> con_s;  
if(node_max<17){  
    con_s.resize(node_max, node_max);  
}  
Matrix<double, Dynamic, Dynamic, 0, 16, 1> i_s;  
if(node_max<17){  
    i_s.resize(node_max, 1);  
}  
Matrix<double, Dynamic, Dynamic, 0, 16, 1> v_s;  
if(node_max<17){  
    v_s.resize(node_max, 1);  
}  
MatrixXd con_l;  
if(node_max>16){  
    con_l.resize(node_max, node_max);  
}  
VectorXd v_l;  
if(node_max>16){  
    v_l.resize(node_max, 1);  
}  
VectorXd i_l;  
if(node_max>16){  
    i_l.resize(node_max, 1);  
}
```

Matrix Initialisation version 1

¹⁶ Eigen.tuxfamily.org. 2020. *Eigen: Writing Functions Taking Eigen Types As Parameters*. [online] Available at: <<https://eigen.tuxfamily.org/dox/TopicFunctionTakingEigenTypes.html>> [Accessed 27 May 2020].

¹⁷ Cprogramming.com. 2020. *C++ Syntax Reference - Ternary Operator - Cprogramming.Com*. [online] Available at: <<https://www.cprogramming.com/reference/operators/ternary-operator.html>> [Accessed 27 May 2020].

matrices would have to be used in processing. This solution was inelegant but should have very little runtime cost as by having the processing for each matrix in different if statements, the program will only have to evaluate the condition ($\text{node_max} < 17$) before running the appropriate version of processing.

For the current and voltage matrices, the Eigen typedef `VectorXd` was used, which would create a dynamically sized vector that could be set equal to the number of nodes present in our circuit. Dummy 'dynamic' 16x1 matrices were also used for cases with fewer than 16 nodes (ignoring reference node), since separate if statements for the 2 conductance matrices were already required it added relatively little memory cost for the increase in performance. This should have a significant improvement in reducing runtime as any optimization will benefit vastly as calculations will have to be performed for each timestep of the transient analysis.

3.4- Matrix Calculations and Processing (Initialisation Loop)

It was decided that there needed to be a protocol that the simulator should follow when processing a circuit. The updating of the conductance matrix would be done with the most efficient method. To do this, priority of components and nodes were considered, and in which order the netlist lines would be processed to minimise time wasted overwriting values in the conductance matrix while maintaining a sensible order to the process. Once these factors had been considered the priority scheme was developed. This meant that the lines of the netlist would be re-arranged to suit the desired order of processing. By doing so, this would ensure the use of an efficient method for inputting values into the conductance matrix and the current vector.

Lines were first prioritised by the first node mentioned (Example below), starting with node 1. This meant that lines were first organised into blocks of lines to show all components connected to each node. However, if this was done with just the lines from the netlist, components would not be shown in the block of lines corresponding to the second node that they were connected to, so further changes were made to input processing to ensure all the information necessary was available (see section 3.5). This approach allows the conductance matrix to be processed node by node in the most efficient way possible.

Designator	First node mentioned	Second node mentioned	Value
R1	1	2	1k
V1	1	0	10
R1	2	1	1k
I1	2	0	1m

Once all the lines were split into sections dependent on the first node mentioned, components were the next priority (Example below). The original order of priority that was decided was voltage sources, capacitors, current sources, inductors and resistors. This order was decided with how these components affected the conductance matrix in mind. Voltage sources were deemed to be the first component to be dealt with because when they appear a 1, and also a – 1 when connected to two non-reference nodes, are placed in the row to show the voltage at this node is equal to the source or a difference in nodes is equal to the value of the source. By doing so no other conductances are noted as all other indexes in this row are 0. This means that other components are insignificant for this row and by addressing this first no time is wasted. However, this assumes that all other lines in this block for this node would then be skipped. This was noted when testing began as other indexes in the row of a voltage source were being updated from their correct 0 values. As it would be more complicated to code logic to skip these lines, the order was altered. In doing so, it meant that voltage sources would be addressed after current sources, inductors and resistors. The process of updating row indexes is explained later.

Capacitors were deemed to come after voltage sources, in terms of priority, as they act as voltage sources and therefore should be dealt with in the same way for the same reasons as listed above. The other three components can be done in any order technically as unlike voltage sources and capacitors they do not fill the row with 0s regardless of other components connected. However, the team chose to deal with current sources first followed by inductors and then resistors. This gave the finalised priority order of current sources, inductors, resistors, voltage sources and capacitors.

Designator	First node mentioned	Second node mentioned	Value
R1	1	2	2k
V1	1	3	10
C1	1	0	1n
I1	2	3	1m
R1	2	1	1k
C2	2	4	1n

Now that the lines have been ordered in terms of node each component is connected to and then within these groups sorted by component priority, the final thing to address was if there were two of the same component connected to the same node (Example below). This would be dealt with by ordering them by lowest second mentioned node. For example, if there were two resistors connected to node 2, one connected to node 1 and the other to node 3, the one connected to node 1 would come first. By sorting the lines in such a manner, there was now a systematic order in place that would ensure an efficient calculation time for filling in the conductance matrix.

Designator	First node mentioned	Second node mentioned	Value
R1	1	2	1k
R2	1	3	3k
V1	1	0	5
I1	2	0	1m

When it came to inputting values for voltage sources, there were two cases to consider. The first being when it is connected to the reference node and the second when it is between two non-reference nodes. An if statement was used to check for each case by checking if the second mentioned node was equal to 0. For the first case, after going through all other components connected to the node, a 1 was placed in G_{ii} position where “i” is the non-reference node and all other values in this conductance matrix row were updated to 0, by the use of a for loop that cycles through the row making all indexes 0. As for the current vector, the value of the voltage source was added into the corresponding node index (example: Voltage source between node 1 and 0, value added to current index for node 1) for when the voltage source was DC. If the voltage source was a sine source, just the DC offset was added into the current vector in the

initialisation stage. A check for a sine source was implemented by an if statement stating if line[3] was not equal to "SINE" it is a DC source.

The next case is more complicated as two rows on the conductance matrix need to be assigned to this voltage source. The original plan was to fill in the first row with the 1 and -1 when the voltage source appeared first and then fill in the conductances connected to both nodes, remembering to place a 0 in the slot that represented the conductance between the two mentioned nodes (this was later deemed to be incorrect as seen in troubleshooting section(4.1)), when we next came across the component. A check for this was placed using an if statement, which compared the first mentioned node with the second, if the first was larger than the second then this would be the second time it has been seen due to the sorting of lines. However, this would be difficult as the conductances connected to the first node would be needed, which would not be accessible as those components would have already been evaluated. So, the updated approach was to fill in all the conductances in the first row for components connected to the first node, including the value for current vector (like reference node case this value was either DC offset for sine source or just value for DC source), and when the source was seen a second time all the conductances would be copied into the second row, including current vector index. As for the first row, once all the values had been copied down to the second row, the first row would be filled with 0s and then a 1 and a -1 placed in the respective indexes for the two nodes and place the inverted value of the source in the current index for the first row.

Care must be taken when filling in the current vector as the value in the current index corresponds to when the source was first seen, and the nodes were flipped. The value here is the inverted version of the value in the second appearance of the source, so value in the line of the second appearance of the source must be inverted before adding it to the current vector of the first line. In the case of a DC source the value is multiplied by -1 and in the sine case it is multiplying the DC offset by -1 and placing only that in the current vector. Just like the voltage sources, capacitors were handled with the same approach with the only difference being the way the current vector was updated as the value of it acting as a voltage source was not readily available. This was handled in the transient loop and will be discussed later in the report.

```

//Resistor processing
if(line[0].find('R')==0){
    double r_con = 1/(ctod(line[3]));
    //Adding to total conductances indices
    if(stoi(line[1])!=0){
        con_s(stoi(line[1])-1, stoi(line[1])-1) += r_con;
    }
    //Allocate respective index in matrix
    if((stoi(line[1])!=0)&&(stoi(line[2])!=0)){
        con_s(stoi(line[1])-1, stoi(line[2])-1) -= r_con;
    }
}

```

While looping through the components of the netlist, if a resistor is found it can alter the conductance matrix in two ways. Firstly, the conductance will be added to the index representing the total conductance of the nodes it is connected to. If the resistor is found between two non-reference nodes, its conductance will be added to the index representing the nodes it is connected to. For example, if the following line is found in the netlist:

R1 N001 N002 20k

This indicates a 20 kilo-Ohm resistor (R1) is connected between nodes 1 and 2. The fragment of code above will first calculate its conductance by finding the reciprocal of its resistance. Following that it will increment the respective total conductance index using the syntactic sugar for compound assignment operator; this ensures all resistors connected to a node will need to be added to the appropriate total conductance index. Note that as each component will be found twice in the netlist with its nodes swapped around (as previously discussed), only one of the node indices will be checked for the total conductance operation. In the above example $1/20k$ will be added to the index representing conductances to node 1 G_{11} (then G_{22} the second time this component is found in the line vector). Afterwards, its conductance is added in the same way to the index G_{12} (G_{21} in the second time). The compound assignment operator “+=” is used once again instead of equating that index to the value of the conductance in order to account for parallel resistors.

For a current source, its value at 0s (DC value) will be added to the current vector in the initialisation loop. Note that when the current source appears a second time, its value will be inverted, and the opposite current will be added to the other node it is connected to. Once again the compound assignment operator is used to account for multiple currents in one

branch. In this simulator inductors will be treated as current source as the current through inductors must always be continuous as per Lenz's law. For inductors, there may be some current flowing into it at time 0s, hence a mapping between current into the node of the inductor and the inductor designator was initialised. Note that both current sources and inductors have a conductance of 0 and so do not make a difference to the conductance matrix.

```
//Current source processing
if(line[0].find('I')==0){
    int node = stoi(line[1]);
    //cout<<node<<endl;
    //If a current source is found, the value of its current will be added to respective node
    if(node!=0){
        // cout<<"here if"<<endl;
        //If sinusoidal then at t=0, only DC offset value
        if(line[3]=="SINE"){
            i_s((node-1), 0) += ctod(line[4]);
        }else{
            i_s((node-1), 0) += ctod(line[3]);
        }
        // cout<<"here else"<<endl;
    }
}
}
}
}
```

3.5- Further input management

After refining processing for the conductance and current matrices, some adjustments were made to the way the netlist was stored to streamline the calculations. In order to correctly process all components, each node should be iterated through from least to most significant, and from there go through each component in the order of priority outlined in the previous section. To do this, a custom sorting function was created using an overload of

```
bool netlist_sort(vector<string> &a, vector<string> &b)
{
    if(a[1]<b[1]){
        return true;
    }
    if(a[1]==b[1]){
        if(component(a[0][0], b[0][0])){
            return true;
        }
        if(a[0][0] == b[0][0]){
            return a[2]<b[2];
        }
    }
    return false;
}
```


std::sort()¹⁸ found in the <algorithm> header (see fig. 2). This algorithm would sort by first prioritizing the lowest value of node1, then the component type, and finally the lowest value of node2. However, since one line of the netlist contains information regarding 2 nodes, this could lead to potential issues when dealing with floating sources, as if the value in node1 is the higher value node, when performing the calculations on the smaller node (node2), the

```
bool component(char a, char b)
{
    if(a=='V' && b!='V'){
        return true;
    }
    if(a=='C' && b!='V' && b!='C'){
        return true;
    }
    if(a=='I' && b!='V' && b!='C' && b!='I'){
        return true;
    }
    if(a=='L' && b=='R'){
        return true;
    }
    return false;
}
```

fig. 2 Sorting comparison functions

order of the sorted netlist would mean that the existence of this floating source wouldn't be known when performing the calculations on that earlier node.

In order to ensure that both nodes have all the component information associated with them alternative data structures were considered to hold this information in a more memory efficient way. However, this was not ideal as for voltage and current sources, the order the nodes are given in the netlist are significant, meaning the value needs to be inverted when swapping the order of the nodes. This led to the decision that the best way to include this information was to swap node1 and node2 after storing the line in the vector representing the netlist, then check whether any changes needed to be made to the value of the component before pushing it back once again. For sinusoidal value, both the line[5] and line[6] elements, representing the DC bias and amplitude, were inverted as the offset as well as the sin function itself need to be inverted for swap cases involving them. Since the reference node "0" is not stored in the conductance matrix, it could be skipped in processing, as any information regarding links between non-reference nodes and node 0 would be found in the lines with the non-reference node as node1. Lines with node1 = "0" were therefore ignored and not pushed back into the input vector, as these would not be useful in analysis, allowing for some memory optimisation.

¹⁸ Cplusplus.com. 2020. *Sort - C++ Reference*. [online] Available at: <<http://www.cplusplus.com/reference/algorithm/sort/>> [Accessed 5 June 2020].

Whilst overall this meant that the time taken for input processing was almost doubled as well as the memory used to store the netlist, the calculation process should become much more efficient, which takes up a more significant portion of runtime.

```
//Creating appropriate matrices / vectors
Matrix<double, Dynamic, Dynamic, 0, 16, 16> con_s;
if(node_max<17){
    con_s = MatrixXd::Zero(node_max, node_max);
}
Matrix<double, Dynamic, Dynamic, 0, 16, 1> i_s;
if(node_max<17){
    i_s = VectorXd::Zero(node_max);
}
Matrix<double, Dynamic, Dynamic, 0, 16, 1> v_s;
if(node_max<17){
    v_s = VectorXd::Zero(node_max);
}
MatrixXd con_l;
if(node_max>16){
    con_l = MatrixXd::Zero(node_max,node_max);
}
VectorXd v_l;
if(node_max>16){
    v_l = VectorXd::Zero(node_max);
}
VectorXd i_l;
if(node_max>16){
    i_l = VectorXd::Zero(node_max);
}
\
```

Updated Matrix Initialisation

Another small change was to initialize the coefficients in the matrices to 0, saving more time in processing by allowing values to be ignored instead of inserting zeroes manually in the appropriate locations. Using the static method `Zero()`¹⁹, a desired matrix can be both resized and have its elements filled with 0, so the conditional vector initialisation was replaced with the following format:

`con_s = MatrixXd::Zeroes(node_max, node_max)` for matrices

`i_s = VectorXd::Zeroes(node_max)` for vectors

3.6- Matrix Decomposition

Now that the values in the conductance and current matrices are initialized, transient analysis can be performed. Before entering the timestep loop, any duplicate components in the input vector were removed, as now that conductance matrix was finished, multiple instances of components were no longer needed. To do this, the `std::remove_if()`²⁰ function in the algorithm header was used. In conjunction with the `erase()` method on the input vector, lines which had a `node1 > node2` were conditionally removed, ignoring cases including the reference node. This should leave a list of unique components from least to most significant nodes, starting at N001, allowing for easier calculations for currents going through each component, as well as updating the values in the current vector.

¹⁹ Eigen.tuxfamily.org. 2020. *Eigen: Advanced Initialization*. [online] Available at: <https://eigen.tuxfamily.org/dox/group__TutorialAdvancedInitialization.html> [Accessed 6 June 2020].

²⁰ Cplusplus.com. 2020. *Remove_If - C++ Reference*. [online] Available at: <http://www.cplusplus.com/reference/algorithm/remove_if/> [Accessed 6 June 2020].

Using the parameters specified in the “.tran” line in the netlist, a for loop was created to iterate through each timestep, solve for the node voltages, calculate the currents flowing through each component and update the values in the current vector.

In order to solve the voltage vector, the Eigen library offers a variety of decompositions, each rated differently based on their accuracy and performance²¹. By splitting processing using two different matrix types, with a more efficient data storage type for smaller matrices, more accurate decompositions which worked quickly for smaller matrices could be used without having to accept the drawback of long processing times for larger matrices. For the <16 node matrix (con_s), ColPivHouseholderQR()²² was selected, an extremely accurate decomposition with sub-millisecond processing times for the matrices it will be used to evaluate. However, for larger matrices using such accurate decompositions is not realistic, taking multiple milliseconds to process (see Appendix A). When considering that this decomposition will be performed hundreds of thousands of times, this would mean that the program would take several minutes just to perform these decompositions. For the >16 node matrix, using the fastest decompositions available was instead prioritised.

The LLT() and LDLT() decompositions boast the lowest benchmarking times within the Eigen library, but require a positive definite or positive or negative semidefinite matrix to be performed²³. This meant that they were incompatible, as the conductance matrix would typically include both positive and negative values. This left PartialPivLU()²⁴, whose only requirement is for a matrix to be invertible. It still has an extremely fast processing time, only 1.6 times slower than LLT() for 100x100 matrices²⁵. The conductance matrix must always be invertible, as the system of equations used for its creation is based on Kirchhoff’s current law,

²¹ Eigen.tuxfamily.org. 2020. *Eigen: Linear Algebra And Decompositions*. [online] Available at: <https://eigen.tuxfamily.org/dox/group__TutorialLinearAlgebra.html> [Accessed 6 June 2020].

²² Eigen.tuxfamily.org. 2020. *Eigen: Eigen::Colpivhouseholderqr<_Matrixtype> Class Template Reference*. [online] Available at: <https://eigen.tuxfamily.org/dox/classEigen_1_1ColPivHouseholderQR.html> [Accessed 6 June 2020].

²³ Eigen.tuxfamily.org. 2020. *Eigen: Linear Algebra And Decompositions*. [online] Available at: <https://eigen.tuxfamily.org/dox/group__TutorialLinearAlgebra.html> [Accessed 6 June 2020].

²⁴ Eigen.tuxfamily.org. 2020. *Eigen: Eigen::Partialpivlu<_Matrixtype> Class Template Reference*. [online] Available at: <https://eigen.tuxfamily.org/dox/classEigen_1_1PartialPivLU.html> [Accessed 6 June 2020].

²⁵ Eigen.tuxfamily.org. 2020. *Eigen: Benchmark Of Dense Decompositions*. [online] Available at: <https://eigen.tuxfamily.org/dox/group__DenseDecompositionBenchmark.html> [Accessed 6 June 2020].

which can be derived from Maxwell's equations²⁶. Solutions to Maxwell's equations are unique, and therefore the equations must be invertible, meaning that there should be no issues using this decomposition. After performing the appropriate decomposition, the voltage vector can then be solved by calling the solve() member function, with the current vector as a parameter, giving all of the node voltages at that timestep.

3.7- Final Processing (Transient Loop)

Current vector values were updated throughout the transient loop as sine sources (both voltage and current), capacitors and inductors do not have fixed values. This was done by having a for loop run through the lines with these components to update current vector values every timestep. For voltage and current sine sources, following the same rules as mentioned in the matrix calculations section about which current index is accessed for each case to set the DC offset, the current vector value would be updated each timestep using the equation of adding the DC offset to the amplitude multiplied by $\sin(2\pi f t)$. The values for the DC offset, amplitude and frequency would be available from the line indexes. The code above shows how this is done with current sources. In each time-step, the difference in current generated by the

```
//Current source processing
if(line[0].find('I')==0){
    if(line[3]=="SINE"){
        int node2 = stoi(line[2]);
        int node1 = stoi(line[1]);
        cout<<(ctod(line[4])+(ctod(line[5])*sin(2*M_PI*ctod(line[6])*t)))<<",";
        //If a current source is found, the value of its current will be added to respective node
        if(t!=0){
            //Initialise variable delta_i which calculates the increase or decrease in current from the previous instance to current one
            //it will then increment
            double delta_i = (ctod(line[5])*sin(2*M_PI*t)*ctod(line[6]))-(ctod(line[5])*sin(2*M_PI*(t-timeStep)*ctod(line[6])));
            if(node2 != 0){
                i_s((node2 -1),0)+= delta_i;
            }else if(node1 != 0){
                i_s((node1 -1),0)-= delta_i;
            }
        }
    }else{
        cout<<ctod(line[3])<<",";
    }
}
```

²⁶ Athavale, P., 2020. KIRCHOFF'S CURRENT LAW AND KIRCHOFF'S VOLTAGE LAW. [online] Ams.jhu.edu. Available at: <http://www.ams.jhu.edu/~prashant/KCL_KVL.pdf> [Accessed 6 June 2020].

source is calculated by subtracting the current value from that of the previous time step, then incrementing the correct indices in the current matrix.

The voltage supplied by capacitors was calculated by finding the additional charge stored on the capacitor during the previous timestep and adding this to the total charge before dividing by the capacitance to find the voltage provided by the capacitor. This was implemented by creating a map to store the total charge values for each capacitor, which would be updated each cycle. To find the additional charge, the current across the capacitor would be calculated and then multiplied by the timestep. To do this the difference in voltage between the nodes connected to the capacitor from the previous timestep would be divided by the timestep and multiplied by the capacitance to get the current (capacitor equation above) and then multiplied by the timestep to get the additional charge. The steps of dividing and multiplying by the timestep were removed as they just cancelled each other out and were unnecessary. This meant that the process was now calculate the difference between the voltage of the nodes connected to the capacitor from the previous timestep and multiply by its capacitance. The voltage values from the previous timestep were obtained from the voltage vector in the current loop as they would not have updated yet. Once the additional charge from this cycle had been calculated, it was added to the value stored in the total charge map and then this total value was divided by the capacitance (code below). This provides the new voltage value, which was placed in the current vector just like a voltage source. The key for each total charge stored for each capacitor would be the designator string, which is the first part of our netlist lines.

```
charges[line[0]] += (total*timeStep);  
//Divide total charge by capacitance to update current vector value  
i_s(stoi(line[1])-1) = charges[line[0]]/ctod(line[3]);  
//find total conductance connected positive end of supernode
```

The value of current generated by an inductor can be calculated using an inductors characteristic equation; this will be found using numerical integration for each time interval. The equation can be re-arranged to obtain the change in current (di) for each time interval as

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

The inductance (L) is read from the netlist and converted to a double using the `ctod()` function and stored in `induct_l`. The potential difference across the inductor was then found by subtracting the nodal voltages across the inductor. This can be seen in the fragment of code below while determining L_{pd} . Following that, the change in time (dt) is taken to be the double variable `timeStep` as specified by the simulation. The change current is then defined according to the equation above and stored in the variable `di_l`. For each iteration, it will increment and decrement the value of current coming in and going out of the inductor respectively by `di_l`. It then increments mapping between the respective inductor and the current going through it by `di_l`.

```
//Initialising variables for node numbers, inductance and PD across inductor
int l_node1 = stoi(line[1]);
int l_node2 = stoi(line[2]);
double induct_val = ctod(line[3]);
double l_pd;
//Finding l_pd, the PD across inductor
if(l_node1 == 0){
    l_pd = -v_s((l_node2-1),0);
}else if(l_node2 == 0){
    l_pd = (v_s((l_node1-1),0));
}else{
    l_pd = (v_s((l_node1-1),0))-(v_s((l_node2-1),0));
}
//Finding di, the change in current and the inductors contribution to that node's current
double di_l = (l_pd*timeStep)/induct_val;
if(l_node2!=0){
    i_s((l_node2 - 1), 0) += di_l;
}else if(l_node1!=0){
    i_s((l_node1 - 1), 0) -= di_l;
}
//Find add di to current going through inductor
induct_i[line[0]]+=di_l;
}
```

3.8- Output

All results produced by the program needed to be outputted in a comma-separated value (CSV) format²⁷ as outlined by the specification. These needed to include both the voltages at every node as well as the current going through each component. On the first line of the output, all columns were to be labelled, starting with all nodes followed by all components. To output the

```
for(int l=1; l<=node_max; l++){
    cout<<"N"<<setfill('0')<<setw(3)<<l<<" ";
}
for(int l=0; l<input.size(); l++){
    cout<<(input[l])[0]<<" ";
}
```

First line of output

²⁷ Edoceo.com. 2020. *Comma Separated Values (CSV) Standard File Format*. [online] Available at: <<http://edoceo.com/utilitas/csv-file-format>> [Accessed 8 June 2020].

list of nodes, simple loop was done using the `node_max` value. By using the `setfill()`²⁸ and `setw()`²⁹ functions in the `iomanip` header, the node listings were standardised in the form used by the input “NXXX”, where spaces were filled with leading 0’s. A second loop then goes through the input vector and outputs each of the components stored in it, producing the first line of output.

Accessing the voltages at each node can be done by directly accessing the elements in the voltage vector, as it is solved as the first step in each iteration of the timestep loop. By iterating through the vector within the transient loop, each element could be outputted in the appropriate column as the nodes are listed sequentially. When going through the input vector to update the values stored in the current matrix, the current flowing through each component could also be calculated in the same loop and outputted, matching the order of the components produced in the first output line. In order to find the current going through each component, the following logic was used:

- Voltage source = 0
- Current source = value
- Resistor = $\frac{V_{node1} - V_{node2}}{value}$
- Capacitor = $C \frac{dv}{dt}$
- Inductors = map[component]

As the current flowing through inductors was already stored in a map for the transient analysis, no additional logic was required, and the designator was simply used as the key to access the current value currently associated with the specific conductor. With each value outputted separated by a “,” and a newline character “\n” at the end of each timestep iteration, each appropriate value would be outputted in the correct order and formatted on separate rows for each timestep.

²⁸ Cplusplus.com. 2020. *Setfill - C++ Reference*. [online] Available at: <<http://www.cplusplus.com/reference/iomanip/setfill/>> [Accessed 8 June 2020].

²⁹ Cplusplus.com. 2020. *Setw - C++ Reference*. [online] Available at: <<http://www.cplusplus.com/reference/iomanip/setw/>> [Accessed 8 June 2020].

4- Testing

4.1- Troubleshooting

Troubleshooting resulted in a few adjustments to the initialisation of the conductance matrix and amendments to values in the current vector for both the initialisation stage and the transient loop. The first detail noticed in the initialisation loop was with floating voltage sources when the conductance matrix was outputted. When copying values from the first row to the second row in the conductance matrix values in the second row were being overwritten with 0s from the first row. This was adjusted by placing an if statement in the for loop that was copying values down, which only allowed values to be copied if they were not equal to 0 (code below). By doing so values in the second row would not be overwritten. It was then noticed that these copied values were not being added to the total conductance of the supernode, so a condition to add each copied conductance to the total supernode conductance was added to the for loop.

```
else{
    //Check for second time voltage source appears
    if(stoi(line[1]) > stoi(line[2])){
        //Copying values from first row into second row and overwrite first row
        for(int x=0; x<con_s.cols(); x++){
            if(x!=stoi(line[1])-1){
                con_s(stoi(line[1])-1, x) += con_s(stoi(line[2])-1, x);
                con_s(stoi(line[2])-1, x) = 0;
            }
        }
        //Move current vector value from first row into second row if current source present
        //i_s(stoi(line[1])-1) += i_s(stoi(line[2])-1);
        //Add in 1 and -1 to first row to represent voltage source
        con_s(stoi(line[2])-1, stoi(line[2])-1) = 1;
        con_s(stoi(line[2])-1, stoi(line[1])-1) = -1;
        //Place value of source into current vector
        //Check for if DC source
    }
```

The next error was an error in one index in the conductance matrix for the second row of a floating voltage source. We checked the conductance matrix multiplied by the voltage vector against the values in the current vector and compared this to the values from our decomposition. This gave us an error in the range of 10^{-16} , which is too small to be a problem. A response to a question on piazza stated that total conductance at each end of the supernode had to be placed in two separate indexes for each end. So, the 0 that had been incorrectly placed in one of these indexes was replaced with the conductance that was being added to the other index that had been treated as the total conductance connected to both ends of the supernode until now. This resulted in the correct values. These changes were then added to

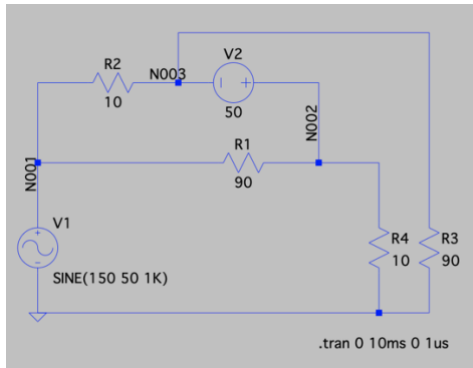
initialisation code for capacitors not connected to the reference node as these are supposed to be treated like voltage sources.

```
//Capacitor processing
if(line[0].find('C')==0){
    //Check for reference node
    double total = 0;
    if(stoi(line[2])==0){
        //multiply all conductances going into capacitor node by voltage difference of nodes to get tot
        for(int y=0; y<con_s.cols(); y++){
            if(y != stoi(line[1])-1){
                double i = (con_s(y, stoi(line[1])-1))*(v_s(stoi(line[1])-1)-v_s(y));
                //sum of currents
                total += i;
            }
        }
    }
    //output current into capacitor
    cout<<total<<",";
    //multiply current by timestep to get additional charge stored on capacitor
    charges[line[0]] += (total*timeStep);
    //Divide total charge by capacitance to update current vector value for voltage provided by cap
    i_s(stoi(line[1])-1) = charges[line[0]]/ctod(line[3]);
}
```

Once all conductance matrices and current vectors from test cases came out correct the transient loop was analysed. Capacitors connected to ground were tested first. It was found that no current was being recorded across the capacitor. The method for calculating current into capacitor was changed to finding all conductances connected to non-reference node and multiplying them with the voltage differences between this node and the other connected nodes, which conductance was measured over (code above). This was summed together in a for loop and then outputted. This was then multiplied by the timestep to find charge, which was added to our charge map, and then the total was divided by the capacitance to give the voltage supplied by the capacitor and placed in the current vector index of the row. This fixed the problem and was a better way to calculate current across capacitors and so was implemented in the non-reference node cases with some adjustments. However, when we tested capacitors between two non-reference nodes, we were not getting any current across it once again. After outputting the voltage difference and conductance values it appeared that the values from the conductance matrix were 0, which meant the wrong indexes were being called. After switching the variables, the problem was fixed.

4.2- Functional Testing

This section will go through some test cases ran to ensure the functionality of all components, comparing the results from the simulation with those from LT SPICE.

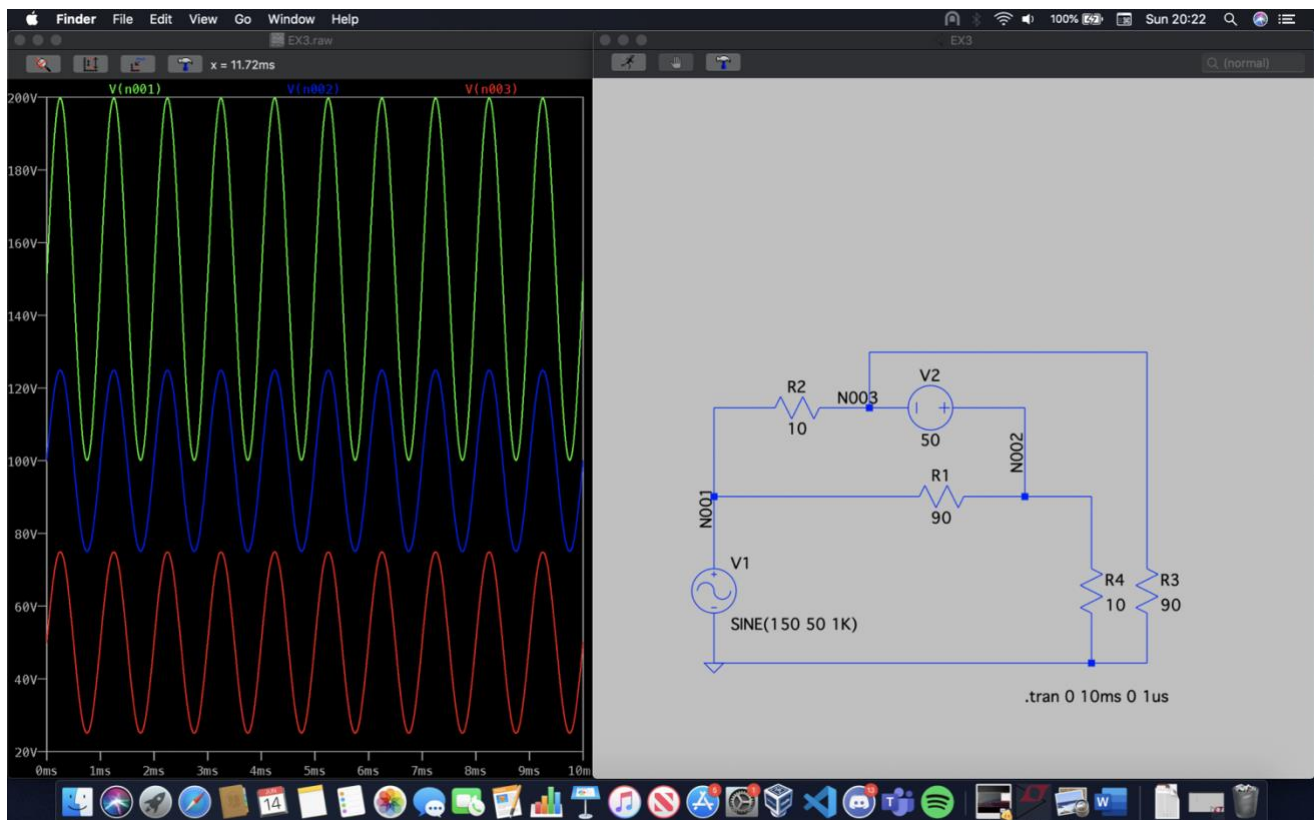


In this example there is a grounded sine voltage source used in conjunction with a floating voltage source and resistors. This simulation was done with a timestep of 1 microsecond, but now over 10 milliseconds, giving it 10,000 cycles of calculations and was completed in 0.24

```
V1 N001 0 SINE 150 50 1k
R1 N001 N002 90
R2 N001 N003 10
V2 N003 N002 50
R3 N003 0 90
R4 N002 0 10
.tran 0 10ms 0 1us
.end
```

seconds. Once again, the netlist, a diagram and the results are shown in Appendix D:

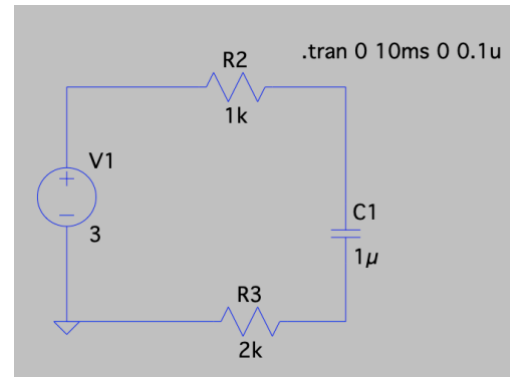
The results (see Appendix D) shows the sine source starting at its DC offset value and the next two show it reaching its limits of oscillation when node 1 goes to 200 and 100 as its amplitude is 50 giving a range of +/- 50 from its offset value. One can also view how node 3 and node 2 always have a difference of 50, which is in line with what is expected. These values also match those from the LTspice simulation that can be seen below:



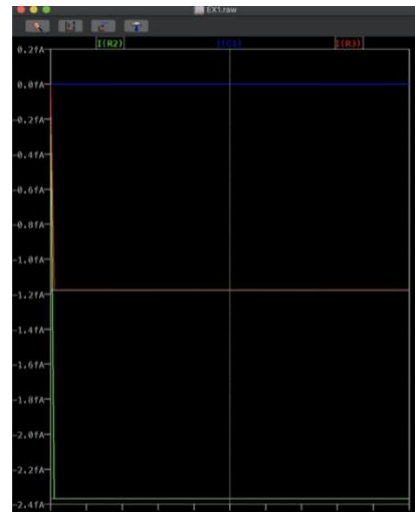
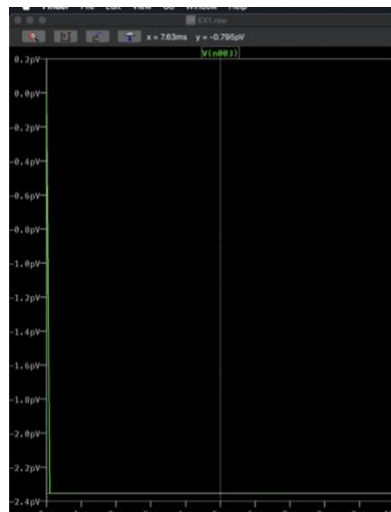
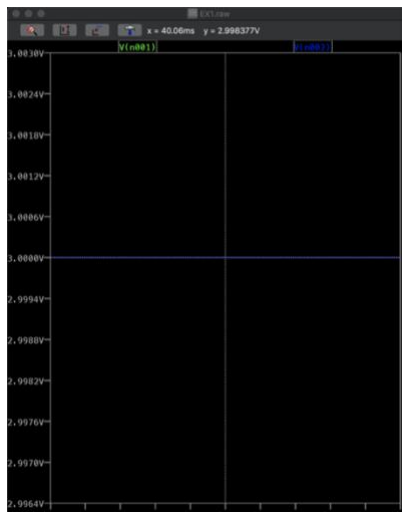
The next example shown here includes a capacitor and DC voltage source to show the transient behaviour of a capacitor. This time the runtime was 15 milliseconds while the timestep stayed at 1 microsecond. This took slightly longer than the other cases, taking 0.32 seconds to complete. Below is the netlist and circuit diagram, for results see appendix E:

```
R1 N001 N002 1k
R2 N002 N003 2k
V1 N001 0 5
C1 N003 0 1u
.tran 0 15ms 0 1u
.end
```

As can be seen above voltage into node 3 starts at 0 and current into capacitor starts at the value of the current through the resistors connected to it. Then by the end of the simulation the value in node 3 is almost the value of the voltage source and the current into the capacitor has decreased dramatically. This is a good representation of transient behaviour in a capacitor as described in an



article titled “Capacitor Transient Response”³⁰. This article illustrates how capacitors will store charge from the current flowing into them and as a result produce a voltage that opposes the current (becoming a voltage source) and increases over time and in doing so the current flowing into it decreases. It also talks of how its behaviour is asymptotic as the current into it



³⁰ Allaboutcircuits.com. 2020. *Capacitor Transient Response | RC And L/R Time Constants | Electronics Textbook*. [online] Available at: <<https://www.allaboutcircuits.com/textbook/direct-current/chpt-16/capacitor-transient-response>> [Accessed 10 June 2020]. (Kharagpur, 2020)

never quite reaches zero and the voltage produced by the capacitor never reaches the value of the voltage source, which is demonstrated above by the simulator. When this simulation was run on LTSpice the results did not show this behaviour and did not appear to show any transient behaviour:

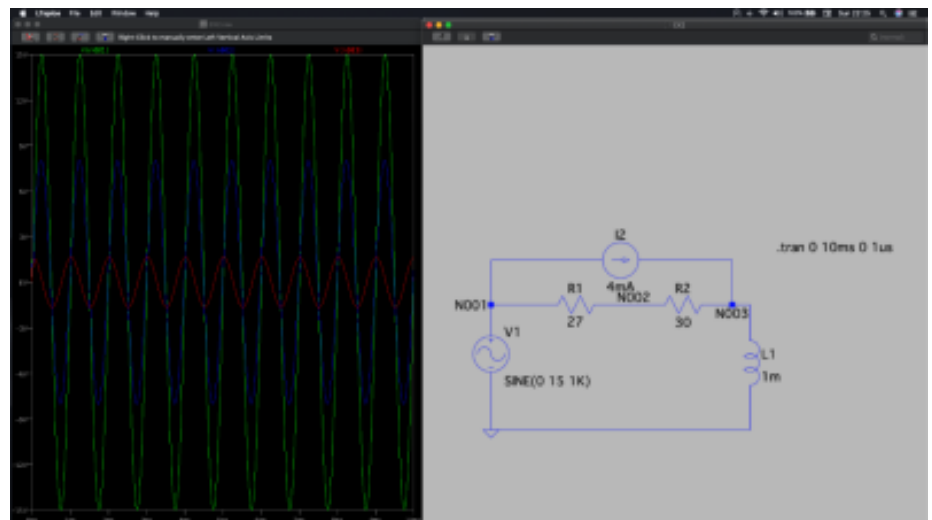
Another example simulated to test the capability of the software was one including a current source and an inductor as seen in the netlist below. The simulation conducted a transient analysis for 10ms with a timestep of 1us, meaning 10,000 instances were calculated.

```

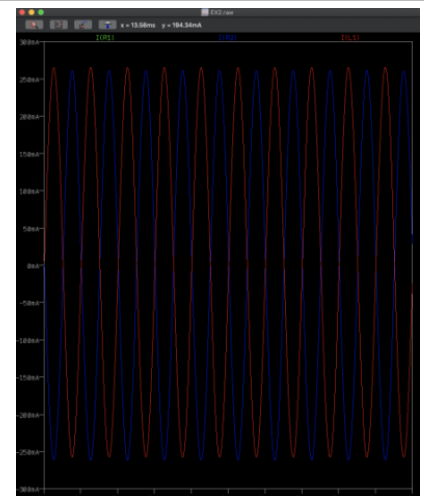
1 *
2 I2 N001 N003 4mA
3 R1 N002 N001 27
4 R2 N003 N002 30
5 L1 N003 0 1m
6 V1 N001 0 SINE(0 15 1K)
7 .tran 0 10ms 0 1us
8 .end

```

The results (see Appendix F) are in accordance with those simulated on LTSpice. The graphs generated from LTSpice for nodal voltages can be seen below alongside the circuit. For example, the voltage at node 2 reaches a minimum of around -8V during time $t=225\mu\text{s}$ which can be seen in both the graph and the simulation's results.



As well the current values on LTSpice can be seen below with the waveforms of R1 and R2 superimposing and the current going through the inductor being in anti-phase as expected. The values generated (see Appendix F) are in agreement with these waveforms.



4.3- Performance Testing

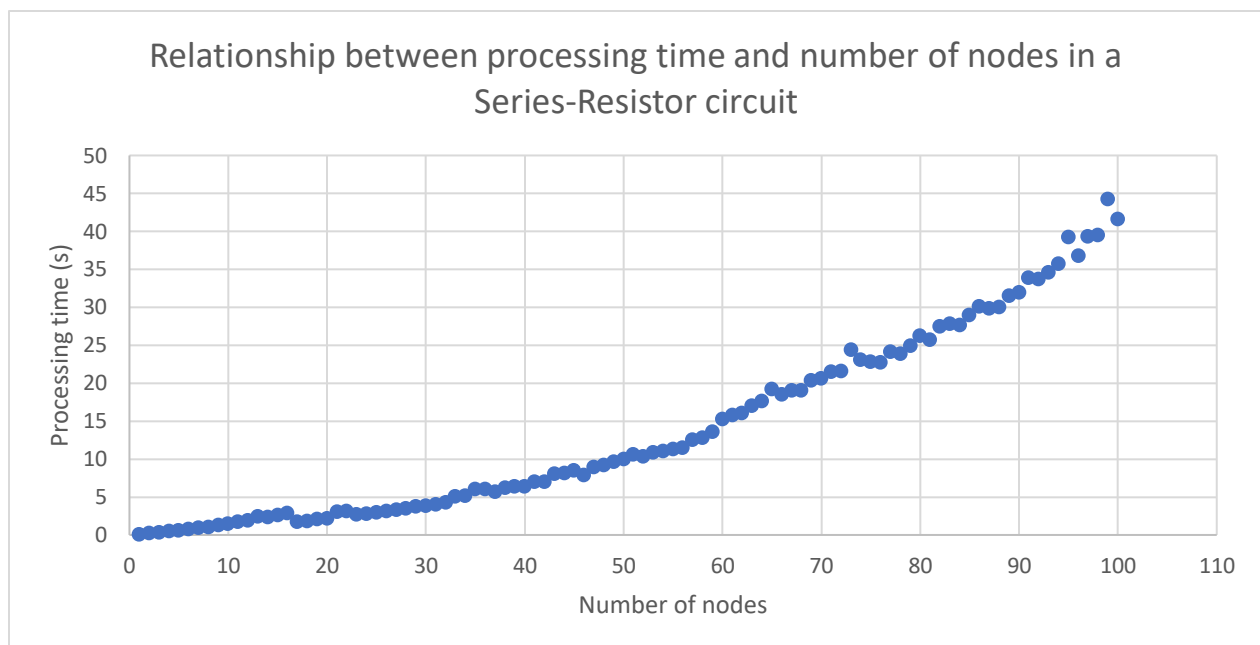
There were several performance benchmarks that were of particular importance:

- Processing time vs. Number of nodes
- Processing time difference between purely dynamic and 'dummy' dynamic matrices in the 1 to 16 node range
- The impact of different components within a circuit on the processing time

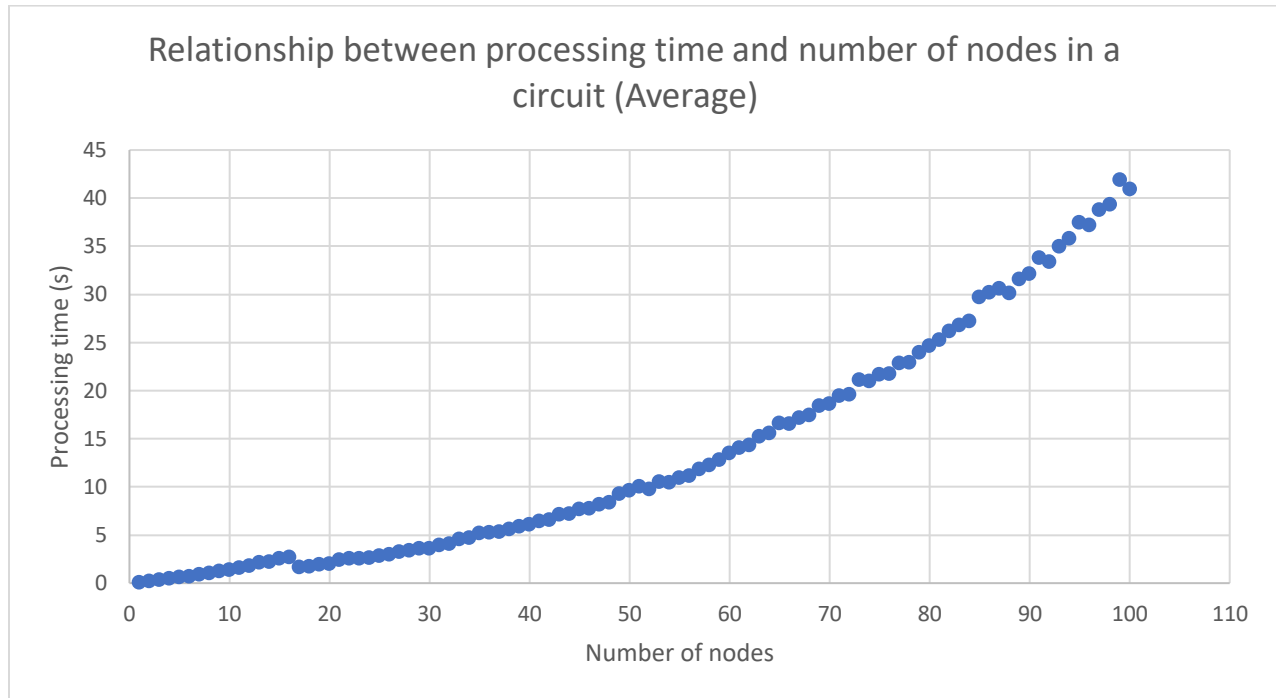
In order to evaluate the relationship between the number of nodes and processing time, it was decided that a circuit would be generated which contained a 5V DC voltage source and an n number of 100Ω resistors in series to create the appropriate number of nodes.

test_generator.cpp (see Appendix B) was created in order to generate 100 test cases ranging from 1 to 100 nodes (excluding the reference node), each with a transient setting of a 10ms simulation duration and $1\mu\text{s}$ step time, meaning each test case would go through 10,000 steps.

These were then all ran using a script, and the time command was used to determine the processing time for each input (see Appendix C). The elapsed, or real time, was used for this benchmark, and tests were performed in laptop connected to a power supply running a virtual machine running Ubuntu 18.04.3 with 3.8 GB of memory and an Intel Core i7-8569U CPU @ 2.80 GHz x 2. From all our data-points, the following graph was produced:



There is a clear non-linear increasing relationship between the number of nodes and processing time, as suggested by the benchmarking times for decompositions (see Appendix A), but there were several discontinuities on the graph that did not match expectations, indicating that the Eigen library had some unexpected behaviour or that outside factors were affecting the benchmark. In order to minimise the impact of such outside factors, the test was performed 2 more times and an average of all results was taken:



This new graph indicated that the outlying points were caused by outside factors, as the overall graph was much smoother after repeated measurements, but that simulations with more nodes had a higher variance on the time taken to perform them. A significant discontinuity was the jump from 16 to 17 nodes, where our simulator switched from using the statically allocated 16x16 matrix and the purely dynamic matrix, going from a 2.88 second to 1.79 second processing time. This drop in processing time was expected as there is a switch to a significantly faster but less accurate decomposition, but in order to decide whether this more expensive decomposition was worth the runtime cost, further investigation into the relative error from both decompositions would be performed. In order to evaluate the accuracy of the decompositions, the relative error can be calculated as follows:

```
double relative_error = (con*v - i).norm() / i.norm();
```

The `.norm()`³¹ method returns the norm of a vector, and since the equation solved by the simulator is of the form [conductance]*[voltage] = [current], the code above calculates the relative error according the definition: $error_{relative} = \frac{true-exp}{true}$. A running total of the relative error was then tracked through each simulation with the average calculated and outputted at the end of the transient analysis. To compare the relative errors of both decompositions, the average relative errors of the generated test cases 1-16 (inclusive) were used to find the overall average error of each decomposition:

Number of nodes	Relative error - colPivHouseholderQr()	Relative error - partialPivLu()
1	0	0
2	3.55E-16	0
3	3.55E-16	1.39E-18
4	3.55E-16	7.76E-19
5	3.55E-16	2.08E-18
6	3.55E-16	1.43E-18
7	3.55E-16	1.63E-18
8	1.78E-16	2.50E-18
9	1.78E-16	1.70E-18
10	1.78E-16	3.62E-18
11	1.78E-16	2.46E-18
12	3.55E-16	2.86E-18
13	3.56E-16	2.50E-18
14	1.78E-16	3.85E-18
15	3.55E-16	4.92E-18
16	3.56E-16	5.07E-18
AVERAGE	2.77698E-16	2.29893E-18

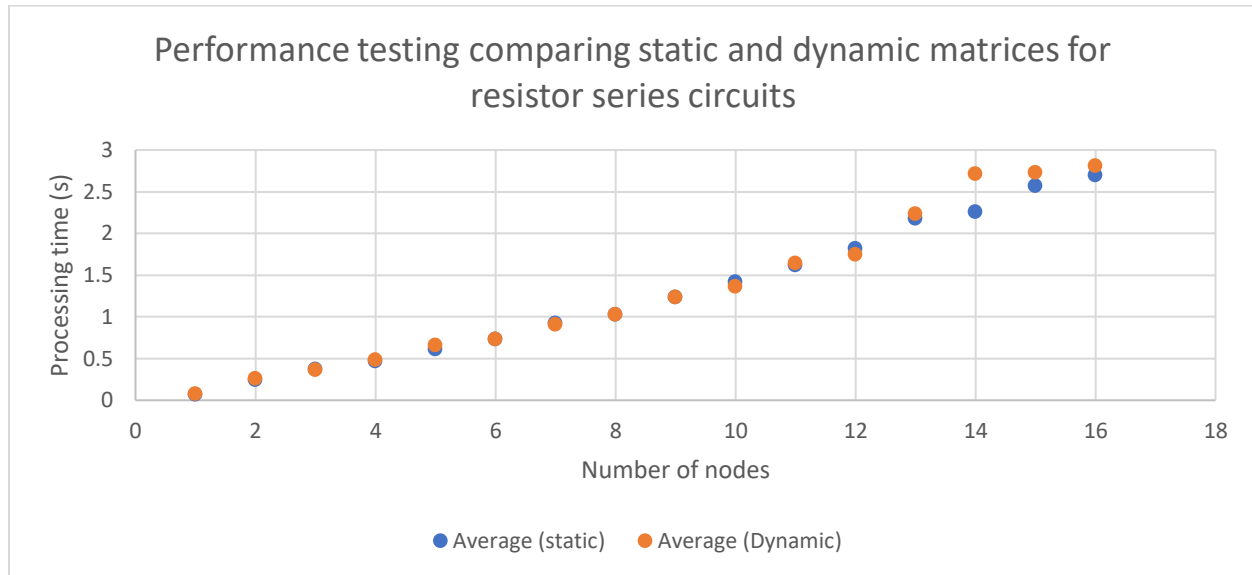
These results show that `partialPivLu()` is ~120 times more accurate than `colPivHouseholderQr()`, meaning that it is by far the best choice for the simulator as it is both faster (2.9 times, see appendix A) and more accurate. This was an unexpected result, but the reason `partialPivLu()` is more accurate is likely as it is a specialised decomposition for invertible matrices, whilst

³¹ Eigen.tuxfamily.org. 2020. *Eigen: Eigen::MatrixBase< Derived > Class Template Reference*. [online] Available at: <https://eigen.tuxfamily.org/dox/classEigen_1_1MatrixBase.html#a196c4ec3c8ffdf5bda45d0f617154975> [Accessed 11 June 2020].

colPivHouseholderQr() is a general decomposition that has no requirements on the matrix³².

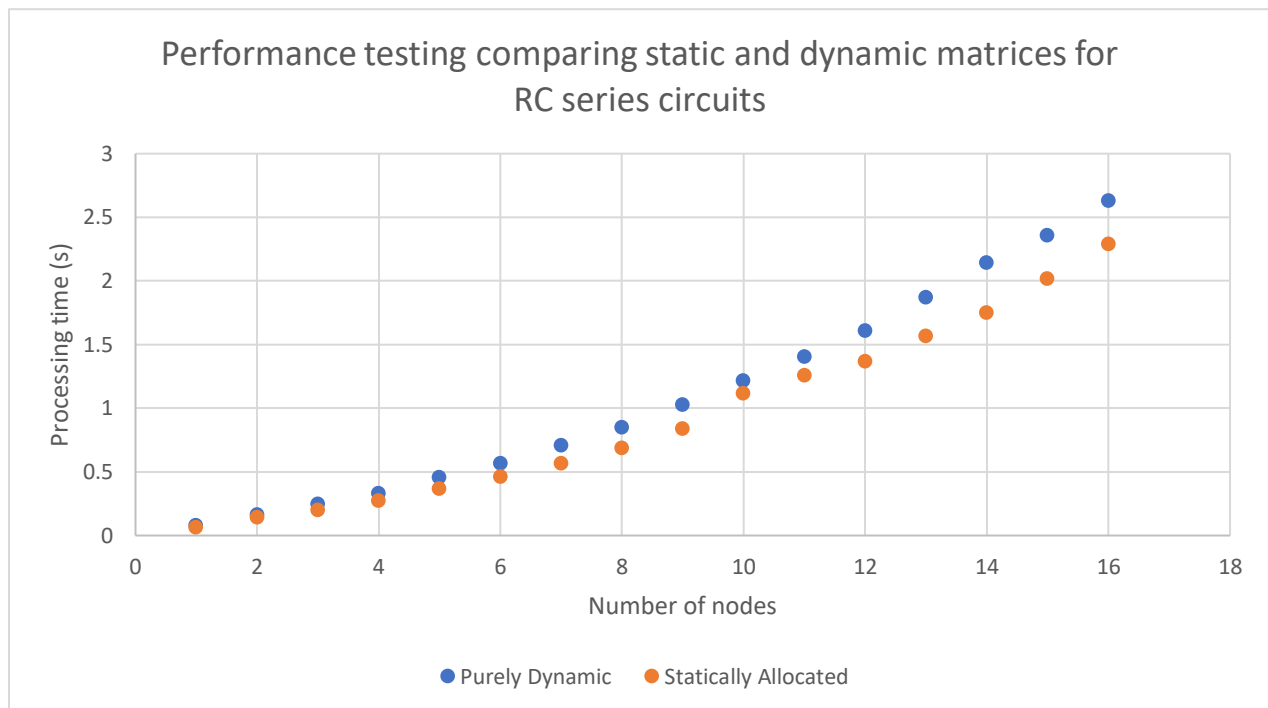
These results led to changing the decomposition used in both cases to partialPivLu() as it is clearly a superior decomposition for the simulator.

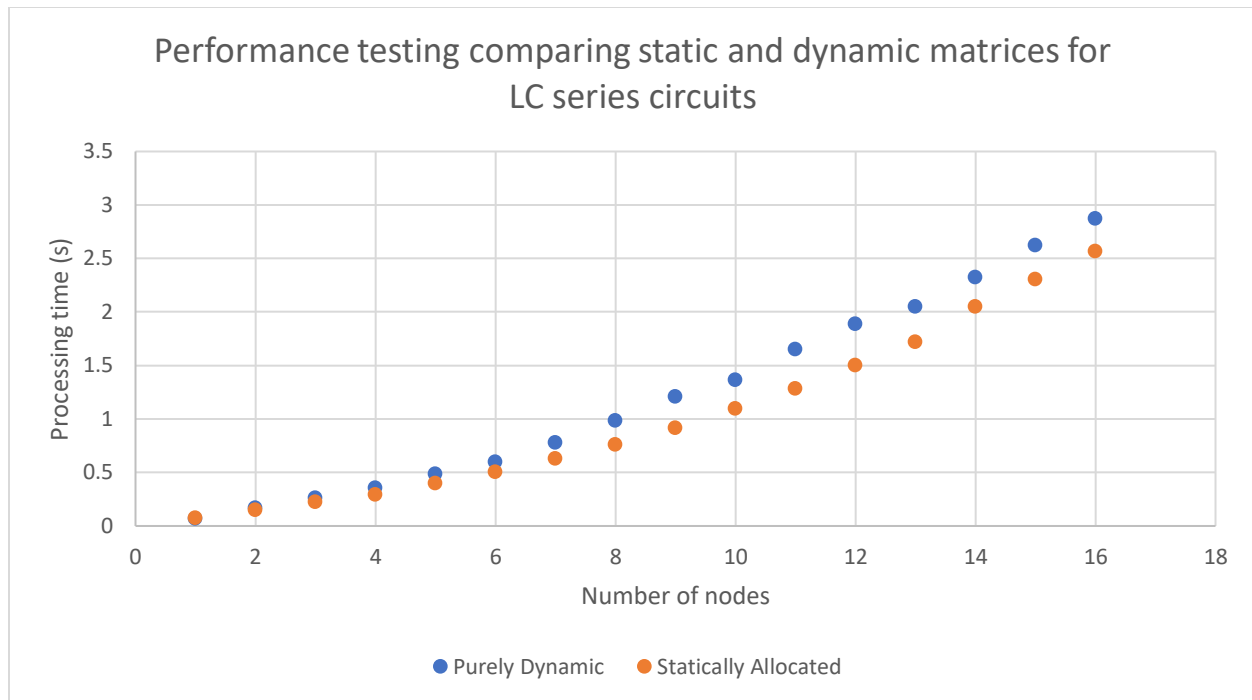
In order to evaluate the difference in the 2 matrix types used, benchmarking tests were run on using both the purely dynamic and statically allocated matrices in the 1-16 node range, with 3 trials of each to reduce the impact of variance on the results:



³² Eigen.tuxfamily.org. 2020. *Eigen: Linear Algebra And Decompositions*. [online] Available at: <https://eigen.tuxfamily.org/dox/group__TutorialLinearAlgebra.html> [Accessed 11 June 2020].

These results indicated a very minor boost in performance from the use of a static compared to a dynamically allocated matrix, with the statically allocated matrix saving on average 47ms. However, this is likely due to the fact that for the series-resistor networks being used in these benchmarking tests, relatively few matrix operations actually needed to be performed. For test cases with more complicated components, where repeated matrix operations need to be performed, such as sinusoidal sources or capacitors and inductors, a more significant increase in performance should be observed. In order to have a more revealing test, the `test_generator.cpp` was updated twice to generate a capacitor in parallel with a resistor, and an inductor in parallel with a capacitor. This should add more matrix operations to the transient analysis, making any differences in performance more apparent. Using the same benchmarking method as before, the following results were obtained:





There is now a clear difference in the performance of the two matrices, with statically allocated matrices saving on average 168ms for the RC circuit, and an average of 201ms for the LC circuit. From these results, it can be extrapolated that the more matrix operations a circuit needs to perform, the more apparent the boost in performance will be, and for circuits involving multiple capacitors, inductors and sinusoidal sources, which require calculations within the transient loop, the benefits of using the 2 data types will be non-negligible.

5- Evaluation and Extension

Overall, our simulator performs well, meeting all functional and non-functional requirements outlined in the introduction, meeting both speed and accuracy requirements. Additional work could have been done in creating more exhaustive test cases to ensure functionality for all components is present in any circuit, but unlike benchmarking tests, this would have required all test cases to be designed manually; and wouldn't fit in this report. Furthermore,

supplementary investigation into the impact of different components on processing time would have been beneficial to better understand the runtime costs of the operations performed, enabling further optimisation. Adding functionality for diodes and transistors was another possible extension, but polishing of the simulator for the currently compatible components was deemed more important. Another area that merited further development was our input error checking, as despite performing basic syntax checks, processing relies on all the information provided to be valid, with minimal assertions or other checks to lower runtime.

Bibliography

- Athavale, P., 2020. *Kirchoff's Current Law and Kirchoff's Voltage Law*. [Online]
Available at: http://www.ams.jhu.edu/~prashant/KCL_KVL.pdf
[Accessed 6 June 2020].
- Belbin, 2020. *The Nine Belbin Team Roles*. [Online]
Available at: <https://www.belbin.com/about/belbin-team-roles/>
[Accessed 20 May 2020].
- Bierbaum, A., Meinert, K. & Scott, B., 2002. *GMTL Programmer's Guide*. [Online]
Available at: <http://ggt.sourceforge.net/gmtlProgrammersGuide-0.6.1-html/index.html#d0e36>
[Accessed 25 May 2020].
- cplusplus.com, 2020. *Getline (String) - C++ Reference*. [Online]
Available at: <http://www.cplusplus.com/reference/string/string/getline/>
[Accessed 26 May 2020].
- cplusplus.com, 2020. *Remove_if - C++ Reference*. [Online]
Available at: http://www.cplusplus.com/reference/algorithm/remove_if/
[Accessed 6 June 2020].
- cplusplus.com, 2020. *Serfill - C++ Reference*. [Online]
Available at: <http://www.cplusplus.com/reference/iostream/setfill/>
[Accessed 8 June 2020].
- cplusplus.com, 2020. *Setw*. [Online]
Available at: <http://www.cplusplus.com/reference/iostream/setw/>
[Accessed 8 June 2020].
- cplusplus.com, 2020. *Sort - C++ Reference*. [Online]
Available at: <http://www.cplusplus.com/reference/algorithm/sort/>
[Accessed 5 June 2020].
- cppreference.com, 2020. *C++ Standard Library Headers - Cppreference.com*. [Online]
Available at: <https://en.cppreference.com/w/cpp/header>
[Accessed 24 May 2020].
- cprogramming.com, 2020. *C++ Syntax Reference - Ternary Operator*. [Online]
Available at: <https://www.cprogramming.com/reference/operators/ternary-operator.html>
[Accessed 27 May 2020].
- edoceo.com, 2020. *Comma Separated Values (CSV) Standard File Format*. [Online]
Available at: <http://edoceo.com/utilitas/csv-file-format>
[Accessed 8 June 2020].
- EETech Media LLC, 2020. *Capacitor Transient Respons | RC and L/R Time Constants | Electronics Textbook*. [Online]
Available at: <https://www.allaboutcircuits.com/textbook/direct-current/chpt-16/capacitor-transient-response>
[Accessed 10 June 2020].
- eigen.tuxfamily.org, 2020. *Eigen: Advanced Initialization*. [Online]
Available at: https://eigen.tuxfamily.org/dox/group_TutorialAdvancedInitialization.html
[Accessed 6 June 2020].

eigen.tuxfamily.org, 2020. *Eigen: Benchmark of Dense Decompositions*. [Online]
 Available at: https://eigen.tuxfamily.org/dox/group_DenseDecompositionBenchmark.html
 [Accessed 26 May 2020].

eigen.tuxfamily.org, 2020. *Eigen: Linear Algebra and Decompositions*. [Online]
 Available at: https://eigen.tuxfamily.org/dox/group_TutorialLinearAlgebra.html
 [Accessed 6 June 2020].

eigen.tuxfamily.org, 2020. *Eigen: The Matrix Class*. [Online]
 Available at: http://eigen.tuxfamily.org/dox/group_TutorialMatrixClass.html
 [Accessed 26 May 2020].

eigen.tuxfamily.org, 2020. *Eigen: Writing Functions Taking Eigen Types as Parameters*. [Online]
 Available at: <https://eigen.tuxfamily.org/dox/TopicFunctionTakingEigenTypes.html>
 [Accessed 27 May 2020].

eigen.tuxfamily.org, 2020. *Eigen::Colpivhouseholderqr<Matrixtype> Class Template Reference*. [Online]
 Available at: https://eigen.tuxfamily.org/dox/classEigen_1_1ColPivHouseholderQR.html
 [Accessed 6 June 2020].

eigen.tuxfamily.org, 2020. *Eigen::Matrixbase<Derived> Class Template*. [Online]
 Available at:
https://eigen.tuxfamily.org/dox/classEigen_1_1MatrixBase.html#a196c4ec3c8ffdf5bda45d0f617154975
 [Accessed 11 June 2020].

eigen.tuxfamily.org, 2020. *Eigen::Partialpivlu<Matrixtype> Class Template Reference*. [Online]
 Available at: https://eigen.tuxfamily.org/dox/classEigen_1_1PartialPivLU.html
 [Accessed 6 June 2020].

Hayt, W., Kemmerly, J. & Durbin, S., 2011. *Engineering Circuit Analysis*. 8 ed. s.l.:McGraw-Hill Education.

imsl.com, 2020. *IMSL C Library | High-Impact Functions for C/C++ Applications*. [Online]
 Available at: <https://www.imsl.com/products/imsl-c-libraries>
 [Accessed 25 May 2020].

Jacob, B. & Guennebaud, G., 2020. *Eigen*. [Online]
 Available at: http://eigen.tuxfamily.org/index.php?title=Main_Page
 [Accessed 25 May 2020].

Kharagpur, 2020. *R-L & R-C Transients*, s.l.: EE IIT.

Perea, E. & Popplewell, J., 2020. *Remote Teamwork*. s.l., Imperial College.

Sciencedirect, n.d. *Circuit Simulation - An Overview | Sciencedirect Topics*. [Online]
 Available at: <https://www.sciencedirect.com/topics/computer-science/circuit-simulation>
 [Accessed 14 May 2020].

Appendices

Appendix A – Eigen Benchmark of dense decompositions (milliseconds)

solver/size	8x8	100x100	1000x1000	4000x4000	10000x8	10000x100	10000x1000	10000x4000
LLT	0.05	0.42	5.83	374.55	6.79 *	30.15 *	236.34 *	3847.17 *
LDLT	0.07 (x1.3)	0.65 (x1.5)	26.86 (x4.6)	2361.18 (x6.3)	6.81 (x1) *	31.91 (x1.1) *	252.61 (x1.1) *	5807.66 (x1.5) *
PartialPivLU	0.08 (x1.5)	0.69 (x1.6)	15.63 (x2.7)	709.32 (x1.9)	6.81 (x1) *	31.32 (x1) *	241.68 (x1) *	4270.48 (x1.1) *
FullPivLU	0.1 (x1.9)	4.48 (x10.6)	281.33 (x48.2)	-	6.83 (x1) *	32.67 (x1.1) *	498.25 (x2.1) *	-
HouseholderQR	0.19 (x3.5)	2.18 (x5.2)	23.42 (x4)	1337.52 (x3.6)	34.26 (x5)	129.01 (x4.3)	377.37 (x1.6)	4839.1 (x1.3)
ColPivHouseholderQR	0.23 (x4.3)	2.23 (x5.3)	103.34 (x17.7)	9987.16 (x26.7)	36.05 (x5.3)	163.18 (x5.4)	2354.08 (x10)	37860.5 (x9.8)
CompleteOrthogonalDecomposition	0.23 (x4.3)	2.22 (x5.2)	99.44 (x17.1)	10555.3 (x28.2)	35.75 (x5.3)	169.39 (x5.6)	2150.56 (x9.1)	36981.8 (x9.6)
FullPivHouseholderQR	0.23 (x4.3)	4.64 (x11)	289.1 (x49.6)	-	69.38 (x10.2)	446.73 (x14.8)	4852.12 (x20.5)	-
JacobiSVD	1.01 (x18.6)	71.43 (x168.4)	-	-	113.81 (x16.7)	1179.66 (x39.1)	-	-
BDCSVD	1.07 (x19.7)	21.83 (x51.5)	331.77 (x56.9)	18587.9 (x49.6)	110.53 (x16.3)	397.67 (x13.2)	2975 (x12.6)	48593.2 (x12.6)

This benchmark has been run on a laptop equipped with an Intel core i7 @ 2,6 GHz, and compiled with clang with **AVX** and **FMA** instruction sets enabled but without multi-threading. It uses **single precision float** numbers. For double, you can get a good estimate by multiplying the timings by a factor 2.

Appendix B – Series-Resistor Netlist Generator

```
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
using namespace std;

int main(){
    ofstream output;
    vector<vector<string>> netlist;
    vector<string> tran = {".tran", "0", "10ms", "0", "1us" };
    netlist.push_back(tran);
    vector<string> v = {"V1", "N001", "0", "5"};
    netlist.push_back(v);
```

```

vector<string> r = {"R1", "0", "N001", "100"};
netlist.push_back(r);
for(int i=1; i<101; i++){
    string filename = "test";
    filename += to_string(i);
    filename += ".txt";
    output.open(filename);
    for(int m=0; m<netlist.size(); m++){
        int n = 0;
        for(n; n<netlist[m].size()-1; n++){
            output<<netlist[m][n]<<" ";
        }
        output<<netlist[m][n]<<endl;
    }
    output<<".end"<<endl;
    output.close();

    string des = "R" + to_string(i);
    r[0] = des;
    string n1 = "N";
    if(to_string(i).size()==1){
        n1 += ("00"+to_string(i));
    }
    if(to_string(i).size()==2){
        n1 += ("0"+to_string(i));
    }
    r[1] = n1;
    string n2 = "N";
    if(to_string(i+1).size()==1){
        n2 += ("00"+to_string(i+1));
    }
    if(to_string(i+1).size()==2){
        n2 += ("0"+to_string(i+1));
    }
    if(to_string(i+1).size()==3){
        n2 += to_string(i+1);
    }
    r[2] = n2;
    netlist[i+1] = r;
    vector<string> r_n = r;
    des = "R" + to_string(i+1);
    r_n[0] = des;
    r_n[1] = "0";
    r_n[2] = n2;

```

```

    netlist.push_back(r_n);
}
}

```

Appendix C – Processing time test script

```

#!/bin/bash
set -e

bash simulator_compile.sh

g++ test_generator.cpp -o test_generator
./test_generator

for i in {1..100}
do
    echo "Test $i :"
    <test$i.txt time ./simulator >test$i.s.txt
    echo ""
done

```

Appendix D – Function Test 1 Results

Time	N001	N002	N003
0	150	50	100
1.00E-06	150	50	100
2.00E-06	150.314	50.1571	100.157
3.00E-06	150.628	50.3142	100.314
4.00E-06	150.942	50.4712	100.471
5.00E-06	151.257	50.6283	100.628
6.00E-06	151.571	50.7853	100.785
7.00E-06	151.885	50.9423	100.942
8.00E-06	152.198	51.0992	101.099
9.00E-06	152.512	51.2561	101.256
1.00E-05	152.826	51.413	101.413
1.10E-05	153.14	51.5698	101.57
1.20E-05	153.453	51.7265	101.727
1.30E-05	153.766	51.8832	101.883
1.40E-05	154.08	52.0398	102.04
1.50E-05	154.393	52.1963	102.196

1.60E-05	154.705	52.3527	102.353
1.70E-05	155.018	52.509	102.509
1.80E-05	155.331	52.6653	102.665
1.90E-05	155.643	52.8214	102.821

Table above shows that values start at DC offset of sine source (150V)

0.000241	199.901	74.9507	124.951
0.000242	199.92	74.96	124.96
0.000243	199.937	74.9684	124.968
0.000244	199.952	74.9758	124.976
0.000245	199.964	74.9822	124.982
0.000246	199.975	74.9877	124.988
0.000247	199.984	74.9921	124.992
0.000248	199.991	74.9956	124.996
0.000249	199.996	74.998	124.998
0.00025	199.999	74.9995	125
0.000251	200	75	125
0.000252	199.999	74.9995	125
0.000253	199.996	74.998	124.998
0.000254	199.991	74.9956	124.996
0.000255	199.984	74.9921	124.992
0.000256	199.975	74.9877	124.988
0.000257	199.964	74.9822	124.982
0.000258	199.952	74.9758	124.976
0.000259	199.937	74.9684	124.968
0.00026	199.92	74.96	124.96
0.000261	199.901	74.9507	124.951

Results above show values hitting their peak when $\sin=1$ and therefore $N001 = (\text{DC offset}) + (\text{Amplitude})$, which are 150 and 50 respectively

0.000741	100.099	25.0493	75.0493
0.000742	100.08	25.04	75.04
0.000743	100.063	25.0316	75.0316
0.000744	100.048	25.0242	75.0242
0.000745	100.036	25.0178	75.0178
0.000746	100.025	25.0123	75.0123
0.000747	100.016	25.0079	75.0079
0.000748	100.009	25.0044	75.0044
0.000749	100.004	25.002	75.002
0.00075	100.001	25.0005	75.0005
0.000751	100	25	75
0.000752	100.001	25.0005	75.0005
0.000753	100.004	25.002	75.002
0.000754	100.009	25.0044	75.0044
0.000755	100.016	25.0079	75.0079
0.000756	100.025	25.0123	75.0123
0.000757	100.036	25.0178	75.0178
0.000758	100.048	25.0242	75.0242
0.000759	100.063	25.0316	75.0316
0.00076	100.08	25.04	75.04
0.000761	100.099	25.0493	75.0493

Results above show values hitting their trough when $\sin=-1$ and therefore $N001 = (\text{DC offset}) - (\text{Amplitude})$, which are 150 and 50 respectively

Appendix E – Function Test 2 Results

Time	N001	N002	N003	R1	V1	R2	C1
0	5	3.33333	0	0.00166667	0	0.00166667	0.00166667
1.00E-06	5	3.33389	0.00166667	0.00166611	0	0.00166611	0.00166611
2.00E-06	5	3.33444	0.00333278	0.00166556	0	0.00166556	0.00166556
3.00E-06	5	3.335	0.00499833	0.001665	0	0.001665	0.001665
4.00E-06	5	3.33555	0.00666333	0.00166445	0	0.00166445	0.00166445
5.00E-06	5	3.33611	0.00832778	0.00166389	0	0.00166389	0.00166389
6.00E-06	5	3.33666	0.00999167	0.00166334	0	0.00166334	0.00166334
7.00E-06	5	3.33722	0.011655	0.00166278	0	0.00166278	0.00166278
8.00E-06	5	3.33777	0.0133178	0.00166223	0	0.00166223	0.00166223
9.00E-06	5	3.33833	0.01498	0.00166167	0	0.00166167	0.00166167
1.00E-05	5	3.33888	0.0166417	0.00166112	0	0.00166112	0.00166112
1.10E-05	5	3.33943	0.0183028	0.00166057	0	0.00166057	0.00166057
1.20E-05	5	3.33999	0.0199634	0.00166001	0	0.00166001	0.00166001
1.30E-05	5	3.34054	0.0216234	0.00165946	0	0.00165946	0.00165946
1.40E-05	5	3.34109	0.0232828	0.00165891	0	0.00165891	0.00165891
1.50E-05	5	3.34165	0.0249418	0.00165835	0	0.00165835	0.00165835
1.60E-05	5	3.3422	0.0266001	0.0016578	0	0.0016578	0.0016578
1.70E-05	5	3.34275	0.0282579	0.00165725	0	0.00165725	0.00165725
1.80E-05	5	3.34331	0.0299152	0.00165669	0	0.00165669	0.00165669
1.90E-05	5	3.34386	0.0315718	0.00165614	0	0.00165614	0.00165614
2.00E-05	5	3.34441	0.033228	0.00165559	0	0.00165559	0.00165559

Table above shows that N003 starts at 0 as expected and current into C1 starts at same value as resistors and they all decrease as capacitor starts to output voltage

0.014978	5	4.9887	4.96609	1.13E-05	0	1.13E-05	1.13E-05
0.014979	5	4.9887	4.9661	1.13E-05	0	1.13E-05	1.13E-05
0.01498	5	4.9887	4.96611	1.13E-05	0	1.13E-05	1.13E-05
0.014981	5	4.98871	4.96612	1.13E-05	0	1.13E-05	1.13E-05
0.014982	5	4.98871	4.96614	1.13E-05	0	1.13E-05	1.13E-05
0.014983	5	4.98872	4.96615	1.13E-05	0	1.13E-05	1.13E-05
0.014984	5	4.98872	4.96616	1.13E-05	0	1.13E-05	1.13E-05
0.014985	5	4.98872	4.96617	1.13E-05	0	1.13E-05	1.13E-05
0.014986	5	4.98873	4.96618	1.13E-05	0	1.13E-05	1.13E-05
0.014987	5	4.98873	4.96619	1.13E-05	0	1.13E-05	1.13E-05
0.014988	5	4.98873	4.9662	1.13E-05	0	1.13E-05	1.13E-05
0.014989	5	4.98874	4.96621	1.13E-05	0	1.13E-05	1.13E-05
0.01499	5	4.98874	4.96623	1.13E-05	0	1.13E-05	1.13E-05

0.014991	5	4.98875	4.96624	1.13E-05	0	1.13E-05	1.13E-05
0.014992	5	4.98875	4.96625	1.13E-05	0	1.13E-05	1.13E-05
0.014993	5	4.98875	4.96626	1.12E-05	0	1.12E-05	1.12E-05
0.014994	5	4.98876	4.96627	1.12E-05	0	1.12E-05	1.12E-05
0.014995	5	4.98876	4.96628	1.12E-05	0	1.12E-05	1.12E-05
0.014996	5	4.98876	4.96629	1.12E-05	0	1.12E-05	1.12E-05
0.014997	5	4.98877	4.9663	1.12E-05	0	1.12E-05	1.12E-05
0.014998	5	4.98877	4.96632	1.12E-05	0	1.12E-05	1.12E-05
0.014999	5	4.98878	4.96633	1.12E-05	0	1.12E-05	1.12E-05
0.015	5	4.98878	4.96634	1.12E-05	0	1.12E-05	1.12E-05

Results above show that the voltage outputted from capacitor never reaches voltage source value and current into capacitor never hits 0, showing asymptotic behaviour

Appendix F – Function Test 3 Results

Time	N001	N002	N003	I2	R1	V1	R2	L1
0	0	-0.108	-0.228	0.004	0.004	0	0.004	0

This is the starting point for all components in this test

0.000241	14.9704	8.01054	0.277362	0.004	0.257773	0	0.257773	0.253773
0.000242	14.976	8.00867	0.267174	0.004	0.25805	0	0.25805	0.25405
0.000243	14.9811	8.00649	0.256977	0.004	0.258317	0	0.258317	0.254317
0.000244	14.9855	8.00399	0.246769	0.004	0.258574	0	0.258574	0.254574
0.000245	14.9893	8.00118	0.236551	0.004	0.258821	0	0.258821	0.254821
0.000246	14.9926	7.99805	0.226324	0.004	0.259057	0	0.259057	0.255057
0.000247	14.9953	7.9946	0.216088	0.004	0.259284	0	0.259284	0.255284
0.000248	14.9973	7.99084	0.205843	0.004	0.2595	0	0.2595	0.2555
0.000249	14.9988	7.98676	0.19559	0.004	0.259706	0	0.259706	0.255706
0.00025	14.9997	7.98237	0.18533	0.004	0.259901	0	0.259901	0.255901
0.000251	15	7.97766	0.175062	0.004	0.260087	0	0.260087	0.256087
0.000252	14.9997	7.97264	0.164788	0.004	0.260262	0	0.260262	0.256262
0.000253	14.9988	7.9673	0.154507	0.004	0.260426	0	0.260426	0.256426
0.000254	14.9973	7.96165	0.144219	0.004	0.260581	0	0.260581	0.256581
0.000255	14.9953	7.95568	0.133926	0.004	0.260725	0	0.260725	0.256725
0.000256	14.9926	7.9494	0.123628	0.004	0.260859	0	0.260859	0.256859

0.000257	14.9893	7.94281	0.113325	0.004	0.260983	0	0.260983	0.256983
0.000258	14.9855	7.9359	0.103017	0.004	0.261096	0	0.261096	0.257096
0.000259	14.9811	7.92868	0.0927057	0.004	0.261199	0	0.261199	0.257199
0.00026	14.976	7.92114	0.0823904	0.004	0.261292	0	0.261292	0.257292
0.000261	14.9704	7.9133	0.0720718	0.004	0.261374	0	0.261374	0.257374

These results show the peak of the sin values

0.000741	-14.9704	-8.01054	-0.277363	0.004	-0.257773	0	-0.257773	-0.261773
0.000742	-14.976	-8.00867	-0.267176	0.004	-0.25805	0	-0.25805	-0.26205
0.000743	-14.9811	-8.00649	-0.256978	0.004	-0.258317	0	-0.258317	-0.262317
0.000744	-14.9855	-8.00399	-0.24677	0.004	-0.258574	0	-0.258574	-0.262574
0.000745	-14.9893	-8.00118	-0.236552	0.004	-0.258821	0	-0.258821	-0.262821
0.000746	-14.9926	-7.99805	-0.226325	0.004	-0.259057	0	-0.259057	-0.263057
0.000747	-14.9953	-7.9946	-0.216089	0.004	-0.259284	0	-0.259284	-0.263284
0.000748	-14.9973	-7.99084	-0.205844	0.004	-0.2595	0	-0.2595	-0.2635
0.000749	-14.9988	-7.98676	-0.195591	0.004	-0.259706	0	-0.259706	-0.263706
0.00075	-14.9997	-7.98237	-0.185331	0.004	-0.259901	0	-0.259901	-0.263901
0.000751	-15	-7.97766	-0.175063	0.004	-0.260087	0	-0.260087	-0.264087
0.000752	-14.9997	-7.97264	-0.164788	0.004	-0.260262	0	-0.260262	-0.264262
0.000753	-14.9988	-7.9673	-0.154507	0.004	-0.260426	0	-0.260426	-0.264426
0.000754	-14.9973	-7.96165	-0.14422	0.004	-0.260581	0	-0.260581	-0.264581
0.000755	-14.9953	-7.95568	-0.133927	0.004	-0.260725	0	-0.260725	-0.264725
0.000756	-14.9926	-7.9494	-0.123629	0.004	-0.260859	0	-0.260859	-0.264859
0.000757	-14.9893	-7.94281	-0.113326	0.004	-0.260983	0	-0.260983	-0.264983
0.000758	-14.9855	-7.9359	-0.103018	0.004	-0.261096	0	-0.261096	-0.265096
0.000759	-14.9811	-7.92868	-0.0927062	0.004	-0.261199	0	-0.261199	-0.265199
0.00076	-14.976	-7.92114	-0.0823909	0.004	-0.261292	0	-0.261292	-0.265292
0.000761	-14.9704	-7.9133	-0.0720723	0.004	-0.261374	0	-0.261374	-0.265374

These results show the trough of the sin values