LTS_V2

Imperial EE 1st Year Group Project

Ignacio Bricchi, Nicholas Pfaff, Igor Silin
CID: 01719233, 01709264, 01756268

Imperial College London

14 JUNE 2020
ELEC40006

Lecturers: Dr. Stott and Mrs. Perea

WORD COUNT: 10711

TABLE OF CONTENTS

1.INTRODUCTION

i. Abstract

LTS_V2 was implemented with the goal of creating a circuit simulator which is accurate, energy efficient, user friendly, and provides support for both linear and non-linear components. The development used an Agile workflow, as well as breaking the implementation of every feature into research, implementation, and testing.

The project results show a software which can accomplish both linear and non-linear analysis with high degrees of accuracy, and linear analysis with high efficiency. Non-linear analysis falls behind in time and energy efficiency.

ii. Specifications

The prompt for this project was to develop a circuit simulator which could take in a simplified SPICE-style netlist, and successfully simulate it using a transient simulation. The simulator should be able to simulate a circuit consisting of a mixture of linear and non-linear elements.

The functional requirements of this project were clear: Run transient analysis on a provided SPICE-style netlist. As a minimum the program should have the ability to simulate linear circuits. Non-linear circuits were specified as being “advanced”.

The non-functional requirements were less precise. Overall, it was decided to aim for a program that runs quickly and is accurate when compared to LTspice. Moreover, it should be easy to use.

We decided on the following for our Product Design Specifications:

1) **Performance**. It is vital that our simulator produces results at a rapid rate. A slow simulator can be considered useless as the user would switch to a quicker alternative.
2) **Maintenance**. Useful software products require easy maintenance. New components get released, and when this happens it should be easy to get them up and running.
3) **Quality and Reliability**. This program is meant to simulate reality. As such, it must be able to produce accurate and consistent results. Users should be able to model a circuit and be confident that when the circuit is built, the real-life output will be almost identical to the simulation.
4) **Testing**. Testing needs to be conducted constantly making sure that our product is reliable and functional.

5) **Documentation**. Documentation is vital to achieve maintainability: a third party should be able to look at the application and instantly understand how something works and what changes can safely be made without breaking existing functionality.

iii. Team Working

To work effectively and efficiently, we broke down each of our Belbin roles in the team and used them as a guide to allocate tasks.

Top two roles for each member:

- Ignacio Bricchi: Shaper + Team worker
- Nicholas Pfaff: Resource Investigator + Completer/Finisher
- Igor Silin: Implementer + Plant

Communication was handled completely through Skype. Skype was chosen as we were familiar with it, and it has screen sharing, voice chat, text chat, and file transfer capabilities.

Meetings were integral to our workflow. We had two main types of meetings: scheduled update meetings and peer working meetings.

The former was done at least two times per week. During these we discussed our progress since previous communication and any problems we had. Additionally, we discussed future plans and proposals.

The latter was done at varying frequencies, but typically several times a week: when something went wrong, we had group calls that involved screen sharing and working through the problem together. This proved an invaluable tool, as having two or three minds on the same issue sped up debugging exponentially.

The best example of this peer coding helping solve problems was our BJT implementation. We used a simplified BJT model of the BJT to simulate it. This model used voltage dependant current sources. The person who implemented this was having problems with the simulation, values were blowing up to infinity. A second team-mate offered help and hoped on a call. This person was better versed in dependant sources and was able to identify an error with the implementation of them in the BJT model. This call lasted maybe 30 minutes at most and prevented hours of debugging.

Overall, having frequent meetings allowed everyone to stay informed about LTS_V2's current state. This knowledge enabled us to support each other constantly and allowed new features to be added quickly. Without this, LTS_V2 would not have advanced as far as it did.

iv. Project Philosophy

A. AGILE DEVELOPMENT

For this project we decided that an agile programming method would be the most effective: It would enable us to progressively build up to the final application while having smaller, intermediate beta versions. These would already meet parts of the specification while being fully functional, self-contained, and testable.

Our Agile timeline:

1.) We built a simple program which could solve linear equations. With this, we compared the performance of two different matrix libraries that we considered.
2.) We then built a simple circuit simulator that could simulate basic circuits consisting of resistors and DC sources. The simulator's results confirmed that the logic used to design the simulator was working as expected.
3.) We built a simple input and output system. The input system could parse a simplified version of a SPICE netlist and generate a circuit model. The output system could simulate this model and output the results to a CSV file. This enabled us to build and test circuits more rapidly.
4.) We expanded our linear model to include components which required updates and changed over time. Reactive components like capacitors and inductors, as well as time varying sources such as sinusoidal waveforms.
5.) We had a major re-organisation of our code into individual sub-modules, which should function as independently as possible. Each sub-module could now be expanded and changed on its own without affecting other parts of the application.

At this point we had a fully functional transient simulator for linear circuits. From here, the timeline became much less linear: we worked simultaneously on multiple feature branches. Hence, the following order of tasks is approximate.

6.) We implemented two different approaches for a non-linear simulation using diodes and chose one of them after testing.
7.) We worked on starting the simulations with a DC operating point calculation. So far, our transient analysis started all circuit values at zero, regardless of the starting source values, making all sources act like pulses.
8.) We decided to roll back the non-linear analysis implementation and re-implement it, using a combination of both previously tested methods. The one previously chosen was not working well

for more complex components and test cases. We choose a combination of the two, as even though the first implementation showed better performance, the second one resulted in simpler code.

9.) We expanded the input options to be more like SPICE.

10.) We implemented the BJT and MOSFET using our new non-linear approach.

11.) We added a dynamic timestep to the non-linear analysis.

12.) We added source stepping to the non-linear analysis to supplement the DC operating point calculation.

13.) We added output interpolation to the non-linear analysis

B. MODULARITY AND POLYMORPHISM

After getting to stage five in the timeline, we had a meeting to discuss our code's structure. The application started becoming large and we needed to decide on how to proceed. We agreed on two main goals regarding our code's organisation.

1) Modularity
2) Polymorphism

Modularity is important as it makes code scalable and easy to collaborate on.

Scalability was needed due to a combination of our AGILE development and our desire to create a system where it would be straightforward to add new features and optimisations, without having to modify unrelated parts of the application. Ideally our implementation should enable someone to improve a sub-module without knowing anything about other sub-modules.

Clear separation of modules also allows team work to be carried out more seamlessly: when working on a feature one could be certain that other people's changes (in different modules) would not cause conflicts with one's changes.

Polymorphism enabled us to design an application without having to worry about extra components being added in the future: Every new component should work independently internally and provide the application with the required information through a set of common functions. The application does not need to differentiate between components or worry about their internal implementation. This allowed us to set up our code such that any new component only requires the creation of a component child class, with minor changes to the input module

2.PROJECT BREAK DOWN

i. Project Plan

The project plan and development focused on three stages: Research, implementation, and testing.

Research played an important role: It became clear early on that there was no need to re-invent the wheel. Circuit simulators exist and hence time can be better spend taking inspiration from years of research rather than recreating everything from scratch.

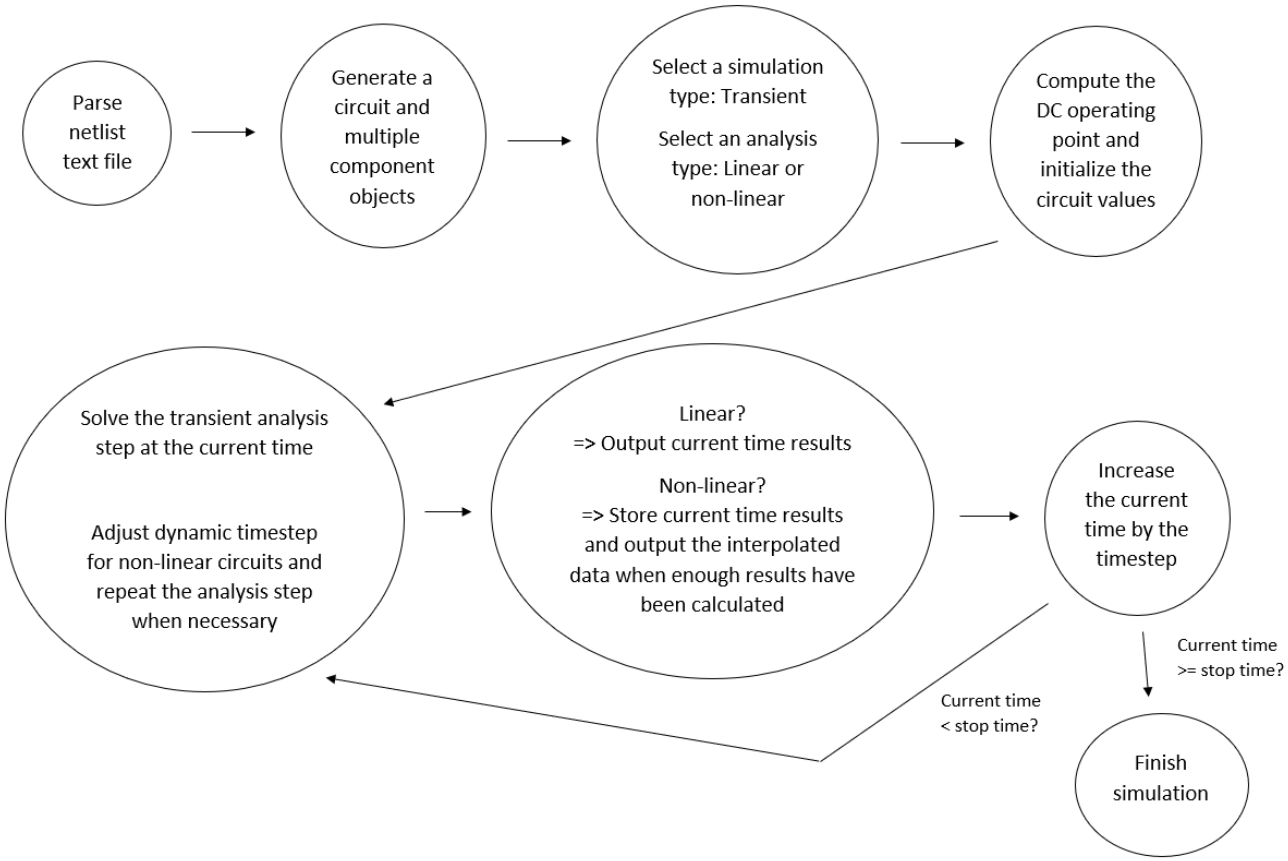
Implementation could occur quickly and efficiently once a concept was well understood.

Testing after implementation provided confidence that the current stage of development was functional, and, as such, the project could be expanded via the agile methodology used.

An overview of the plan consisted of researching and testing a matrix library that would form the backbone of the circuit evaluation, building a solution for linear components, and finally expanding this solution for non-linear components. This approach of splitting the project into parts based on functional requirements ensured that every functional requirement would be met.

After complying with the minimum requirements (simulation with linear components), the development process gained the freedom and flexibility to test additional features. This involved adding additional elements, such as ideal op-amps, and improving efficiency, accuracy, and user friendliness. The focus was more on the latter, as extra components were not mentioned in the specification and hence appeared to have a lower priority than improving the overall application.

ii. Simulator state diagram



iii. General Newton-Raphson explanation

A fundamental part of LTS_V2 is the Newton-Raphson method for obtaining the roots of an equation. Newton-Raphson is based on the idea that the root of the tangent of a function at a point A, is closer to the root of the function itself than point A [1].

Iterative Newton-Raphson formula [2]:

$$X_{n+1} = X_n - \frac{f(X_n)}{f'(X_n)} \qquad \textit{Eq. 1}$$

Every consecutive iteration produces an x closer to the root of the function. The root is found when $X_{n+1} = X_n$.

The iterative formula above is best demonstrated graphically. Rewriting the equation:

$$f(X_n) + f'(X_n)(X_{n+1} - X_n) = 0 \quad \textit{Eq. 2}$$

$$y = f(X_n) + f'(X_n)(X_{n+1} - X_n) = 0 \quad \textit{Eq. 3}$$

$$y = f(X_n) + f'(X_n)(x - X_n) \quad \textit{Eq. 4}$$

Eq. 4 describes a line with gradient $f'(X_n)$, passing through the point $(X_n, f(X_n))$: The tangent to the curve $y = f(x)$.

In *Eq. 4*, using $x = X_{n+1}$ and $y = 0$, X_{n+1} is the root of the tangent as becomes clear when comparing with *Eq. 3*. This is shown in Figure 1. Figure 1 also shows how every Newton-Raphson iteration gets closer to the root of the function than the previous one.

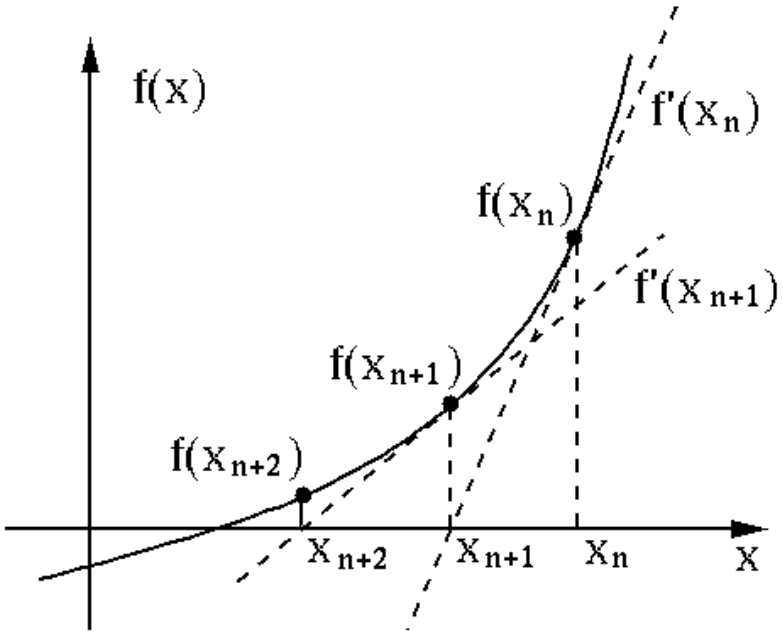


Figure 1. Graphical Newton-Raphson representation. Taken from [3]

iv. Matrix Generation

A circuit schematic can be described by a set of equations. These equations can be easily solved by a computer when written in matrix form. One method to obtain such equations is nodal analysis. This technique is demonstrated using the example circuit in Figure 2.

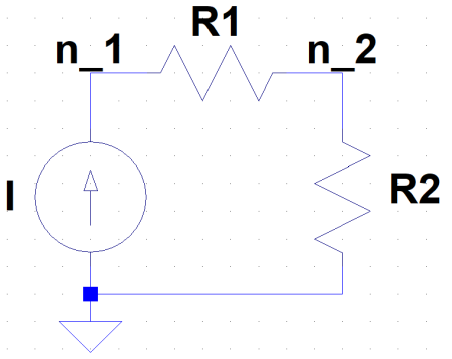


Figure 2. Resistor

For the circuit from Figure 2, nodes one and two have been labelled and ground is defined as node zero:

$$\frac{v_1 - v_2}{R_1} - I1 = 0 \qquad \textit{Eq. 5}$$

$$\frac{v_2 - v_1}{R_1} + \frac{v_2}{R_2} = 0 \qquad \textit{Eq. 6}$$

Rewriting into matrix form:

$$\begin{bmatrix} \frac{1}{R_1} & -\frac{1}{R_1} \\ -\frac{1}{R_1} & \frac{1}{R_1} + \frac{1}{R_2} \end{bmatrix} \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = \begin{pmatrix} I1 \\ 0 \end{pmatrix} \qquad \textit{Eq. 7}$$

Solving will give the nodal voltages v_1 and v_2.

One problem of nodal analysis are voltage sources: They have infinite conductance and hence cannot appear in the matrix above (conductance matrix). A solution is modified nodal analysis (MNA) [4]: The conductance matrix, the solution vector and the current vector **b** are extended to include an extra part that deals with voltage sources. MNA was chosen as the method to deal with voltage sources, as it has advantages compared to other methods.

The main other method suggested in the project brief deals with supernodes. MNA is a well-defined and a very systematic approach and is hence much easier to construct than supernodes. Also, MNA gives the current through the voltage source directly inside the solution vector, while for supernodes this current must be found using the process of elimination. Elimination involves extra steps and hence appears less efficient than MNA.

MNA is demonstrated using the example circuit in Figure 3.

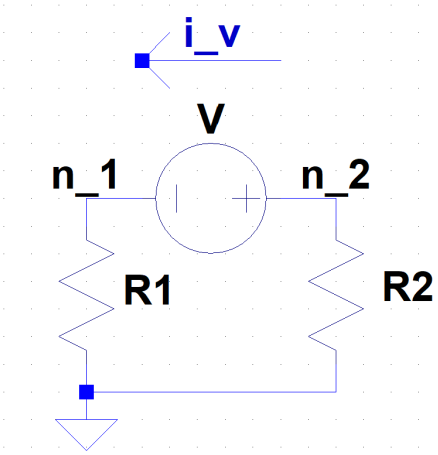


Figure 3. Circuit for MNA

Applying nodal analysis including the added current through the voltage source:

$$\frac{v_1}{R_1} - i_{V_1} = 0 \qquad \textit{Eq. 8}$$

$$i_{V_1} + \frac{v_2}{R_2} = 0 \qquad \textit{Eq. 9}$$

Add an equation to describe the effect of the voltage source on its adjacent nodes:

$$v_2 - v_1 = V1 \qquad \textit{Eq. 10}$$

Rewriting in matrix form:

$$\begin{bmatrix} \frac{1}{R_1} & 0 & -1 \\ 0 & \frac{1}{R_2} & 1 \\ -1 & 1 & 0 \end{bmatrix} \begin{pmatrix} v_1 \\ v_2 \\ i_{V_1} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ V1 \end{pmatrix} \qquad \textit{Eq. 11}$$

Solving gives the node voltages v_1, v_2 and the current through the voltage source i_{V1}.

A computer can easily solve the matrix equations above. However, it cannot form the nodal equations and construct the matrices the same way that it was done above. To create the matrixes, a method called construction by inspection [5] or element stamping [4] is used as explained further in later sections.

v. Matrix Library

It was decided to use a matrix library rather than designing a custom one for this project: many matrix-libraries already exist for C++. Producing a better or an equivalently good one would consume too much time, which could be better spent on the actual task of the project.

To choose one of the many available matrix libraries, research was conducted, considering reviews and forum activity. The library should fulfil certain criteria: it must be free, fast, have an easy to understand syntax and a big community of users. The big community is important as having lots of information and already answered questions online speeds up development time significantly. Two libraries were found that best fulfilled the criteria above: Eigen3 and Armadillo.

To select one, further performance testing was done to assess their speed and accuracy. The first of these tests was meant to find the fastest solver for each library. This test was adjusted to the needs of the planned linear circuit solver: The A matrix would only need to be computed once as only **b** changes between successive timesteps. This test produced the following results:

1. Eigen3's fastest solver is the normal inverse method ($\boldsymbol{x} = A^{-1}\boldsymbol{b}$)
2. Armadillo's fastest solver is LU decomposition

These results make sense as both methods above do not need to recalculate the entire solution at each timestep. A does not change for the linear solver and hence Eigen3 only needs to obtain the inverse and Armadillo only needs to obtain the LU decomposition of A once. See Appendix A for a list of the tested equation solvers and detailed test results.

The second one of these tests compared Eigen3's and Armadillo's fastest solvers using the same test program and the same test cases. The test was performed using 6x6 matrices and 1M iterations to simulate the timesteps. The following results were produced:

1. Eigen3's inverse method is around 8x faster than Armadillo's LU decomposition
2. Eigen3 is more accurate when compared with MATLAB's solution

However, despite Armadillo having a nicer syntax than Eigen3 and more detailed documentation, Eigen3 was still decided to be the winner due to its significantly better speed and accuracy.

The third test compared Eigen3's different solvers based on the non-linear simulator's requirements: all matrices must be recomputed at every iteration. The test was performed using 30x30 matrices and 100k iterations. The results showed that Eigen3's default inverse method was still the fastest for the relatively small matrices tested. These matrix dimensions would be common for LTS_V2.

vi. Possible solutions for linear components

When researching different approaches for analysing linear circuits, two main methods were found. The first idea was to use Newton-Raphson, the second one was to solve the nodal equations directly using linear algebra.

Both methods were considered, as Newton-Raphson would be necessary for non-linear components and it could permit the reuse of code. However, using Newton-Raphson would require re-calculating the conductance matrix A, and therefore A^{-1}, at every iteration. The direct linear algebra method avoids this, only calculating A and A^{-1} once at the beginning of the simulation, solely updating the **b** vector at successive timesteps.

This means that the second, direct method is significantly faster, as discussed in the Matrix Library section: Calculating the inverse of a matrix is very computationally intensive, especially as the matrix dimensions increase.

vii. Possible Solutions for non-linear

A. DIRECT NEWTON RAPHSON

One solution that was considered, is using Newton-Raphson directly to solve the nodal equations. Rewriting *Eq. 1* in terms of the change of x between successive iteration:

$$\Delta X_n = -\frac{f(X_n)}{f'(X_n)} \qquad \textit{Eq. 12}$$

$f(x)$ is the non-linear function for which a root must be found. It must be differentiable at X_n.

To use this equation, one first makes an initial guess of what the solution might be. This becomes X_0. The equation is then called recursively, adding ΔX_n to X_n to from X_{n+1}. This is repeated until the magnitude of ΔX_n falls below a threshold.

This approach can be expanded to work for multiple variables, to find the roots of a set of simultaneous equations. This is achieved using a set of matrices and linear algebra. Rewriting *Eq. 12* for multiple variables [6]:

$$J(X_n) * \Delta X_{n+1} = f(X_n) \qquad \textit{Eq. 13}$$

$\boldsymbol{f}(x)$ is the vector of simultaneous equations and $\boldsymbol{J}(x)$ the Jacobian of $\boldsymbol{f}(x)$. The Jacobian matrix contains all the first order partial derivatives of each equation in $\boldsymbol{f}(x)$.

An original guess, ΔX_n, is set to a vector of all zeros. Then, *Eq. 13* is solved repeatedly, continuously changing the X used for the following iteration. The solution is found once every value in ΔX_{n+1} is below the threshold.

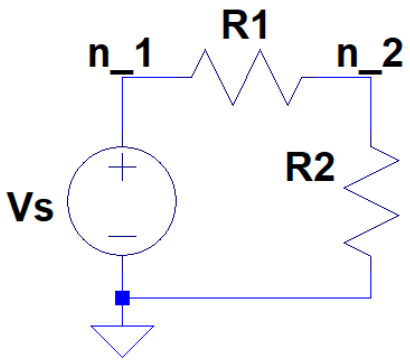


Figure 4. Newton-Raphson example circuit

Taking Figure 4 as an example, constructing $\boldsymbol{f}(\boldsymbol{x})$ from the MNA equations results in:

$$\boldsymbol{f}(\mathrm{x}) = \begin{bmatrix} -\mathrm{i_v} + (v_1 - v_2)\dfrac{1}{\mathrm{R1}} \\ (v2 - v1)\dfrac{1}{\mathrm{R1}} + v_2\dfrac{1}{\mathrm{R2}} \\ \mathrm{Vs} - v1 \end{bmatrix} \qquad \textit{Eq. 14}$$

Because $\boldsymbol{f}(\boldsymbol{x})$ and the Jacobian are constructed using MNA, their construction can be implemented similarly to the way the matrix generation is implemented. The Jacobian from *Eq. 13* can be obtained:

$$\mathrm{J}(\mathrm{x}) = \begin{bmatrix} \dfrac{1}{R1} & -\dfrac{1}{R1} & -1 \\ -\dfrac{1}{R1} & \dfrac{1}{R1} + \dfrac{1}{R2} & 0 \\ -1 & 0 & 0 \end{bmatrix} \qquad \textit{Eq. 15}$$

This algorithm can be used with any component that can be described using IV characteristics, including those with more than two nodes.

B. INDIRECT NEWTON RAPHSON

To incorporate non-linear components into the system of equations produced by MNA, they must first be linearised[1] (modelled with linear components). Newton Raphson uses linear tangents, over several iterations to approximate a solution. The same can be done with circuits, creating linear models which when solved iteratively lead to a solution.

If $f(x) = i$, where i is the IV characteristic of a component, using Eq. 3:

$$i_{n+1} = i_n + i'(v_n)(v_{n+1} - v_n) \qquad \textit{Eq. 16}$$

Rewriting:

$$i_{n+1} = (i_n - i'(v_n)v_n) + i'(v_n)v_{n+1} \qquad \textit{Eq. 17}$$

Letting $i' = g$

$$i_{n+1} = (i_n - g * v_n) + \mathrm{g} * v_{n+1} \qquad \textit{Eq. 18}$$

This can be represented by a current source and a resistor in parallel (Figure 5).

[1] Inspiration for this method was given by [1]

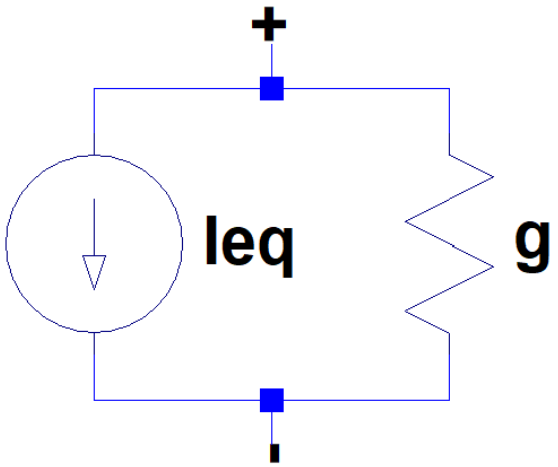


Figure 5. Linear diode companion model

Where $Ieq = (Ic(V_n) - V_n * g)$ and $g = \frac{dIc}{dV}(V_n)$.

A linear circuit can be obtained by replacing all non-linear components with their linear equivalent models. Its solution corresponds to the n+1 iteration and hence the next Newton-Raphson guess v_{n+1}. Using the new guess, eventually the voltages will converge, and a solution will be obtained.

This method is best demonstrated using the diode which is the simplest non-linear component. An example circuit for this is shown below:

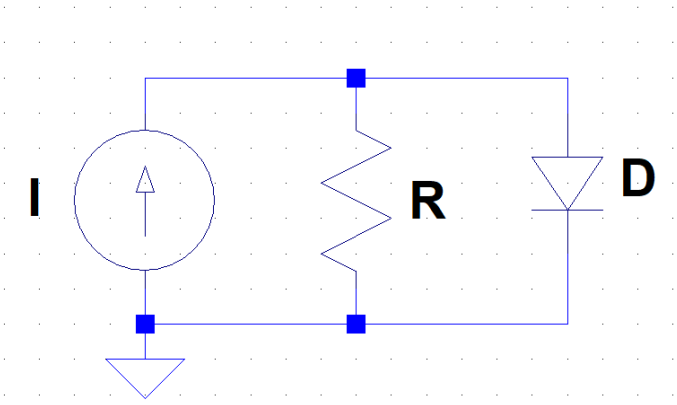


Figure 6. Diode example

The IV characteristic of the diode is given by Shockley's equation:

$$i = I_s\left(e^{\frac{v}{v_t}} - 1\right) \qquad \textit{Eq. 19}$$

Linearising using Newton-Raphson:

$$i_{n+1} = \left(i_n - \frac{I_s}{v_t}e^{\frac{v}{v_t}}v_n\right) + \frac{I_s}{v_t}e^{\frac{v}{v_t}}v_{n+1} \qquad \textit{Eq. 20}$$

Letting

$$g_n = \frac{I_s}{v_t}e^{\frac{v}{v_t}} \qquad \textit{Eq. 21}$$

$$Ieq_n = i_n - g_n v_n \qquad \textit{Eq. 22}$$

Hence:

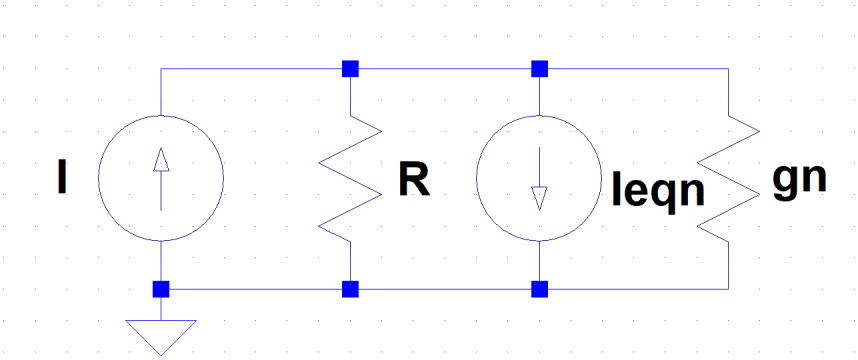


Figure 7. Figure 3 with companion model

Applying nodal analysis to the n+1 iteration of the circuit shown in Figure 7:

$$\frac{v_{n+1}}{R} + G_n v_{n+1} + i'_n - I1 = 0 \qquad \textit{Eq. 23}$$

Solving for the next Newton-Raphson guess v_{n+1}:

$$v_{n+1} = \frac{I1 - I_s\left(e^{\frac{v}{v_t}} - 1\right) + \frac{I_s}{v_t}e^{\frac{v}{v_t}}v_n}{\frac{1}{R} + \frac{I_s}{v_t}e^{\frac{v}{v_t}}} \qquad \textit{Eq. 24}$$

The result corresponds to the one that would be obtained by applying Newton-Raphson directly as is demonstrated below.

Applying nodal analysis directly to the circuit shown in Figure 6:

$$\frac{v}{R} + I_s\left(e^{\frac{v}{v_t}} - 1\right) - I1 = 0 \qquad \textit{Eq. 25}$$

Applying Newton-Raphson:

$$v_{n+1} = v_n - \frac{\frac{v}{R} + I_s\left(e^{\frac{v}{v_t}} - 1\right) - I1}{\frac{1}{R} + \frac{I_s}{v_t}e^{\frac{v}{v_t}}} \qquad \textit{Eq. 26}$$

$$v_{n+1} = \frac{I1 - I_s\left(e^{\frac{v}{v_t}} - 1\right) + \frac{I_s}{v_t}e^{\frac{v}{v_t}}v_n}{\frac{1}{R} + \frac{I_s}{v_t}e^{\frac{v}{v_t}}} \qquad \textit{Eq. 27}$$

Mathematically both methods yield the same results. However, after testing both (See Appendix B) it became apparent that the second provided better convergence. Additionally, because the direct method required having both the MNA equations and their derivatives, more calculations would need to be performed per iteration than with the indirect method. For these reasons, the indirect method was chosen.

3.IMPLEMENTATIONS

i. Linear Component Solutions

A. RESISTORS AND LINEAR SOURCES

When considering the equation $A\boldsymbol{x} = \boldsymbol{b}$ generated using MNA, **x** will be the solution vector containing the node voltages and voltage source currents. The current through any resistor is the product of the conductance and the voltage difference across it. Keeping this idea in mind, a pattern can be spotted about how resistors appear in the previous example matrices: The diagonal elements of the conductance matrix A are the sum of the conductance of the resistors corresponding to that node. For example, the first diagonal element is the sum of the conductance connected to node one. The other elements of the conductance matrix are the negative conductance of the resistor connected between the two corresponding nodes. For example, a resistor connecting node one and node two results in its negative conductance in element (1,2) and element (2,1) of the conductance matrix.

Linear current sources affect the **b** vector. Current through a current source is always a constant at a given time and is independent of the voltage across it. For this reason, it is placed in the **b** vector. For a current source pointing from node one (positive node) to node two (negative node), current is added to index one and subtract it from index two of **b**.

As mentioned previously, MNA includes an extra equation for voltage sources $Vs = V_+ - V_-$. In the row/column corresponding to the voltage source, a one must be inserted at the element corresponding to the source's positive node, and a minus one at the source's negative node.

As each row represents a nodal equation, any equation which requires the current through a voltage source must include a positive or negative one in the appropriate column, corresponding to that current in the $\boldsymbol{x}$ matrix.

All remaining elements of A, **x** and **b** are zero.

B. CAPACITORS AND INDUCTORS

As the terminal equations of capacitors and inductors involve derivatives, some method of numerical integration is necessary to model them. The most widely used integration scheme for circuit analysis was chosen [7] – Trapezoidal method. It is superior to other first order methods (Forward/Backward Euler) as

the order of the error due to the trapezoidal method is h^3, as opposed to h^2. So, for small values of h (<<1), the largest term in the error is related to h^3, which is much smaller than h^2.

Higher order methods have smaller errors (h^4 for parabolic, h^5 for Simpson's Rule [7]) and give results that are curves, but they have disadvantages. They are not self-starting, which means that a first order method must be used to generate the first two values, and therefore cannot be used independently. They are also less stable than the trapezoidal method, meaning that there is a greater probability of results not converging. The trapezoidal method strikes a good balance between accuracy and stability.

The derivation of the inductor model is shown below. The derivation of the similar capacitor model is given in Appendix C.

The inductor equation is:

$$v = L\frac{di}{dt} \qquad \textit{Eq. 28}$$

This rearranges to give the following equation when t = n+1:

$$i_{n+1} = \frac{1}{L}\int_0^{n+1} v\ dt \qquad \textit{Eq. 29}$$

According to the trapezoidal method, $\int_0^{n+1} v\ dt = \int_0^n v\ dt + \frac{h}{2}(v_{n+1} + v_n)$, where *h* is the timestep:

$$i_{n+1} = \frac{1}{L}\int_0^{n+1} v\ dt = \frac{1}{L}\int_0^n v\ dt + \frac{h}{2L}(v_{n+1} + v_n) = i_n + \frac{h}{2L}v_n + \frac{h}{2L}v_{n+1} \qquad \textit{Eq. 30}$$

This can be modelled as a current source in parallel with a resistor (Figure 8).

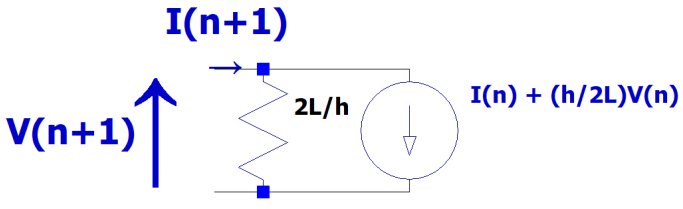


Figure 8. Linear inductor model

Similar results are obtained for the capacitor, shown in Figure 9 and *Eq. 31* (see Appendix C):

$$i_{n+1} = -i_n - \frac{2C}{h}v_n + \frac{2C}{h}v_{n+1} \qquad \textit{Eq. 31}$$

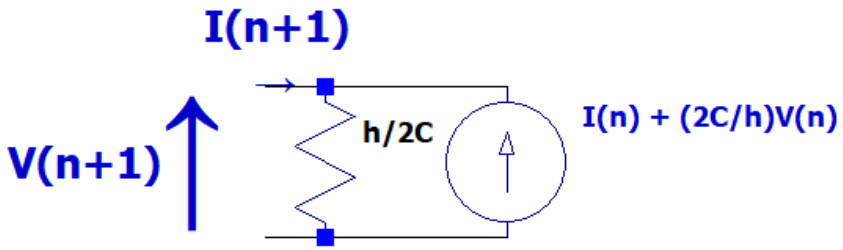


Figure 9. Linear capacitor model

A benefit of these models is that resistors and current sources are simple components: a capacitor/inductor can be treated as a resistor when calculating conductance for the A matrix, and as a current source when calculating the **b** matrix.

Additionally, given a static timestep, the resistor's value remains constant for both models, allowing the A matrix to remain constant throughout a simulation.

The current source is updated at the end of each iteration, using the nodal voltages just calculated and the previous value of the current source.

Capacitors/inductors were tested using the circuit from Figure 10.

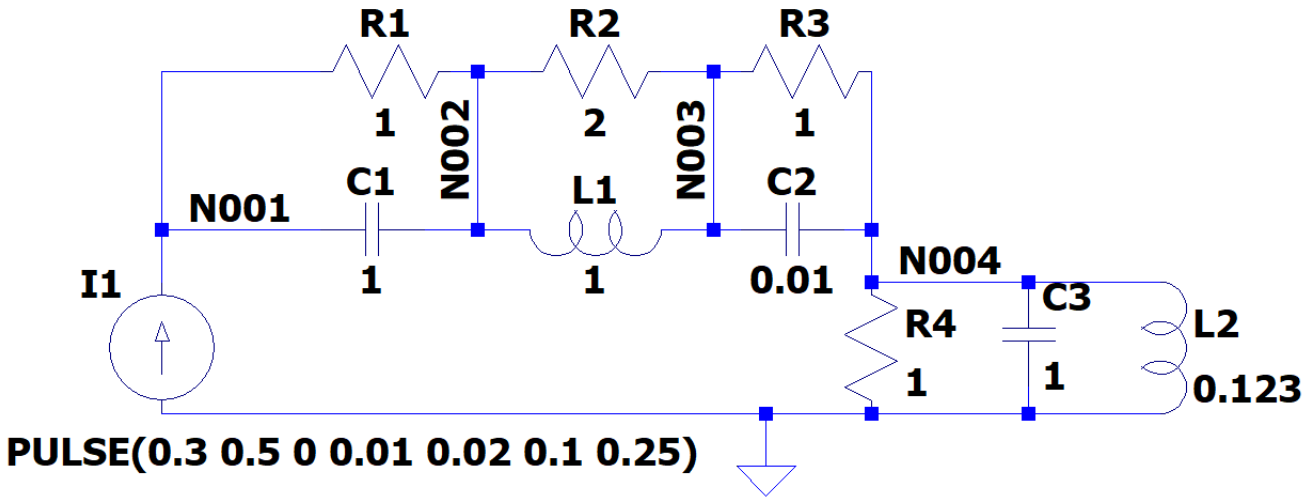


Figure 10. Capacitor/inductor test circuit

Simulating:

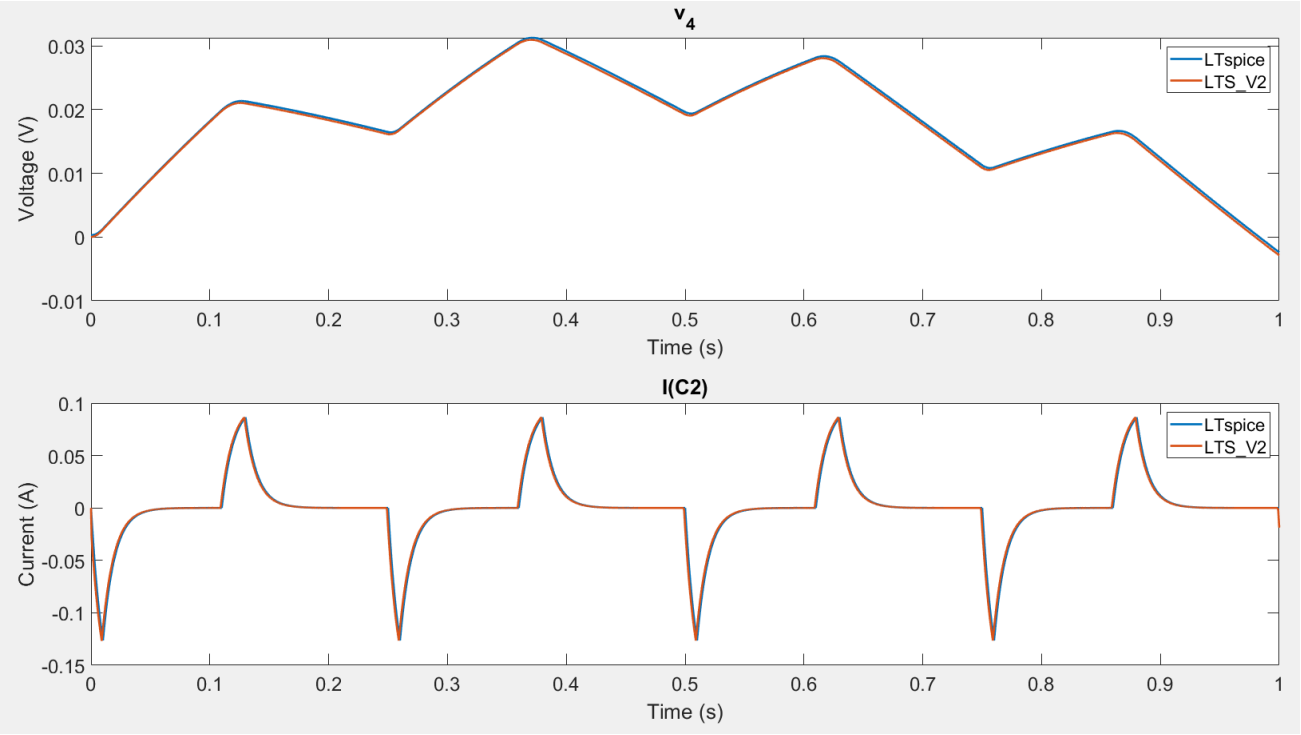


Figure 11. Two of the waveforms produced by the circuit shown in Figure 10 when simulated with a static timestep of 10µs

Figure 11 shows the most complex voltage and current output from the circuit in Figure 10. Both are almost identical to LTspice's results. This shows that the Trapezoidal method gives excellent results, even for complex circuits with multiple inductors/capacitors.

C. IDEAL OP-AMPS

The ideal op-amp implemented[2] assumes that negative feedback exists, that $Vin(+) = Vin(-)$, where Vin are the input terminals, that no current flows into the input terminals and that there are no limits imposed by the supply rails.

The output of an ideal op-amp can be treated as a voltage source while the inputs are defined by $Vin(+) - Vin(-) = 0$. This is demonstrated using the circuit shown in Figure 12.

[2] Inspiration for this method was given by [12]

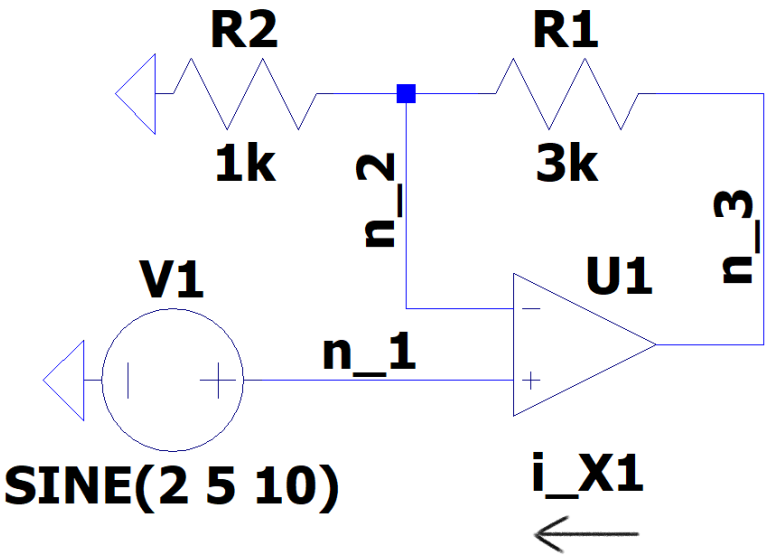


Figure 12. Non-inverting configuration

Applying MNA, the following equations are obtained where the op-amp is treated as described above:

$$V1 = v_1 \qquad \textit{Eq. 32}$$

$$\frac{v_2}{R_2} + \frac{v_2 - v_3}{R_1} = 0 \qquad \textit{Eq. 33}$$

$$\frac{v_3 - v_2}{R_1} + i_{X1} = 0 \qquad \textit{Eq. 34}$$

$$v_1 - v_2 = 0 \qquad \textit{Eq. 35}$$

Rewriting in matrix form:

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & \frac{1}{R_1}+\frac{1}{R_2} & -\frac{1}{R_1} & 0 & 0 \\ 0 & -\frac{1}{R_1} & \frac{1}{R_1} & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 \end{bmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ i_{V1} \\ i_{X1} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ V1 \\ 0 \end{pmatrix} \qquad \textit{Eq. 36}$$

The major difference between the op-amp and a voltage source is that the op-amp is not described in terms of its output voltage, but by its input terminals being at the same potential.

The column corresponding to the op-amp (last column) represents the current flowing out of the op-amp while the row corresponding to the op-amp (last row) represents the input terminals being at the same potential.

It can be noted that adding an op-amp breaks the symmetry of the A matrix. This also occurs when dependent sources are added to the circuit.

Simulating the circuit from Figure 12:

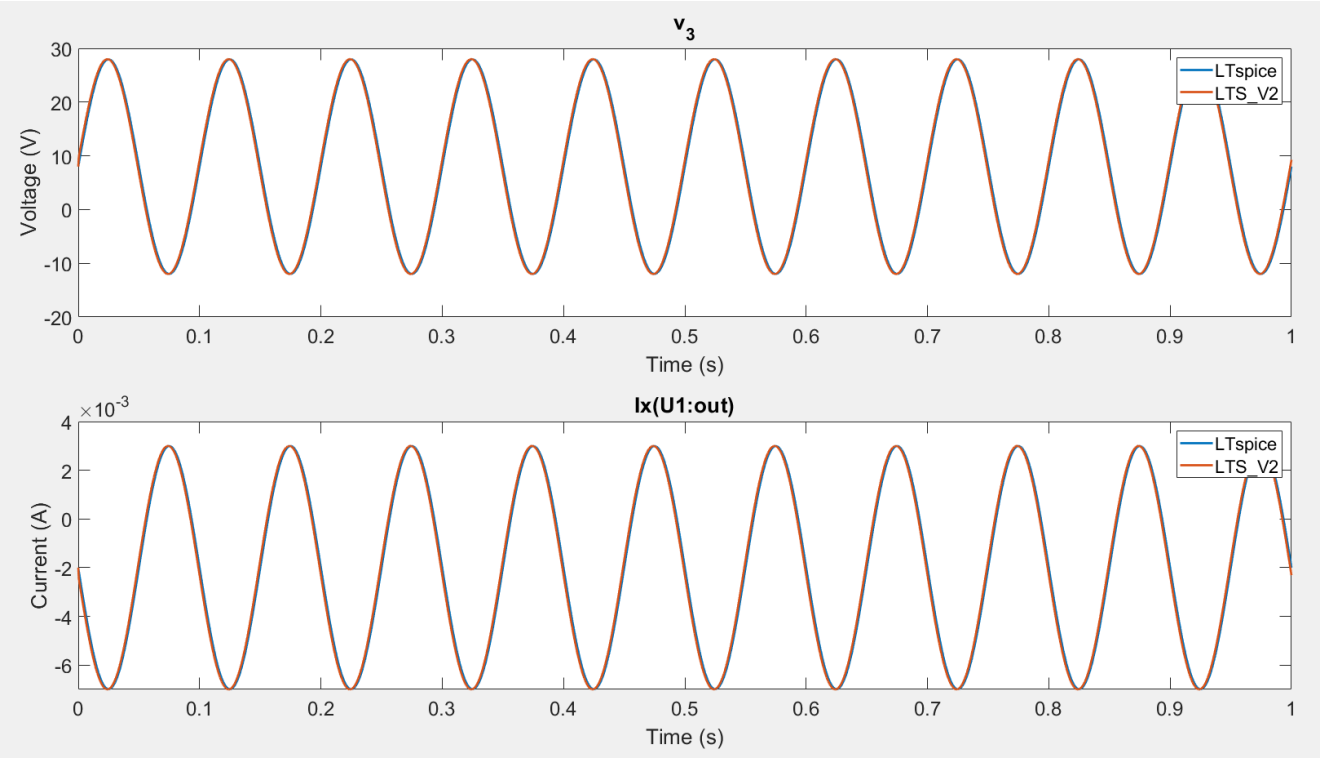


Figure 13. Waveforms produced by the circuit from Figure 12, using a static timestep of 10µs

Both the voltage and current at the op-amp's output are basically identical to the ones produced by LTspice.

D. DEPENDENT SOURCES

Dependent sources are described by equations as was done for the ideal op-amp[3].

Voltage-controlled voltage source

A voltage-controlled voltage source creates a potential difference between its two terminals that is equal to its gain multiplied by the potential difference between the two controlling nodes. This can be

[3] Inspiration for this method was given by [8].

described as $v_{N(+)} - v_{N(-)} = Gain * (v_{NC(+)} - v_{NC(-)})$ where V_N is the voltage at the source's node and V_{NC} the voltage at the controlling node. Rearranging gives $v_{N(+)} - v_{N(-)} + Gain * (v_{NC(-)} - v_{NC(+)}) = 0$.

To implement a voltage-controlled voltage source using MNA, it is noticed that the source still behaves like a normal voltage source as indicated by the first half of the equation. This behaviour does not change, but some additional behaviour must be added to the row corresponding to the source as described by the second half of the equation: "+Gain" must be added to the column corresponding to NC(-) and "-Gain" to the column corresponding to NC(+).

This is demonstrated using the circuit shown in Figure 14.

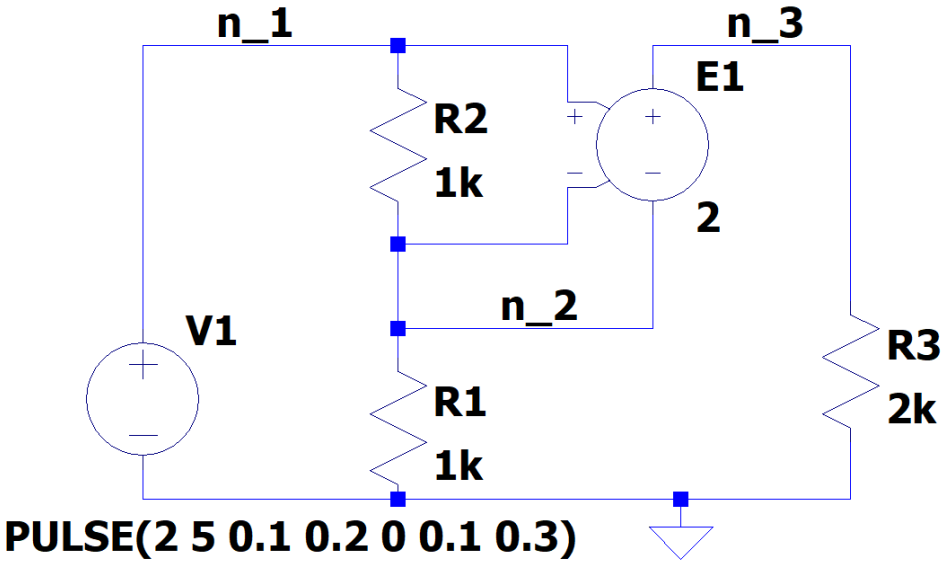


Figure 14. Voltage-controlled voltage source. Taken from [8].

Here, $\boldsymbol{n_1}$ is the positive and $\boldsymbol{n_2}$ the negative controlling node. The previous examples demonstrated how the MNA equations correspond to their matrix from. Hence, this step is skipped here and for the following examples. The implemented algorithm builds these matrix equations directly by element stamping as is explained here and in the previous subsections of Linear Component Solutions.

Resulting matrix equations:

$$\begin{bmatrix} \frac{1}{R_1} & -\frac{1}{R_1} & 0 & 1 & 0 \\ -\frac{1}{R_1} & \frac{1}{R_1}+\frac{1}{R_2} & 0 & 0 & -1 \\ 0 & 0 & \frac{1}{R_3} & 0 & 1 \\ -1 & 0 & 0 & 0 & 0 \\ -Gain & Gain-1 & 1 & 0 & 0 \end{bmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ i_{V_1} \\ i_{E1} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ V1 \\ 0 \end{pmatrix} \quad \textit{Eq. 37}$$

The “1” and “-1” at position (3,5), (5,3) and (2,5), (5,2) respectively, describe the normal voltage source behaviour of E1. The “Gain” and “-Gain” at position (5,1) and (5,2) respectively, describe the additional behaviour of E1 as described by the source’s defining equation.

Simulating the circuit from Figure 14:

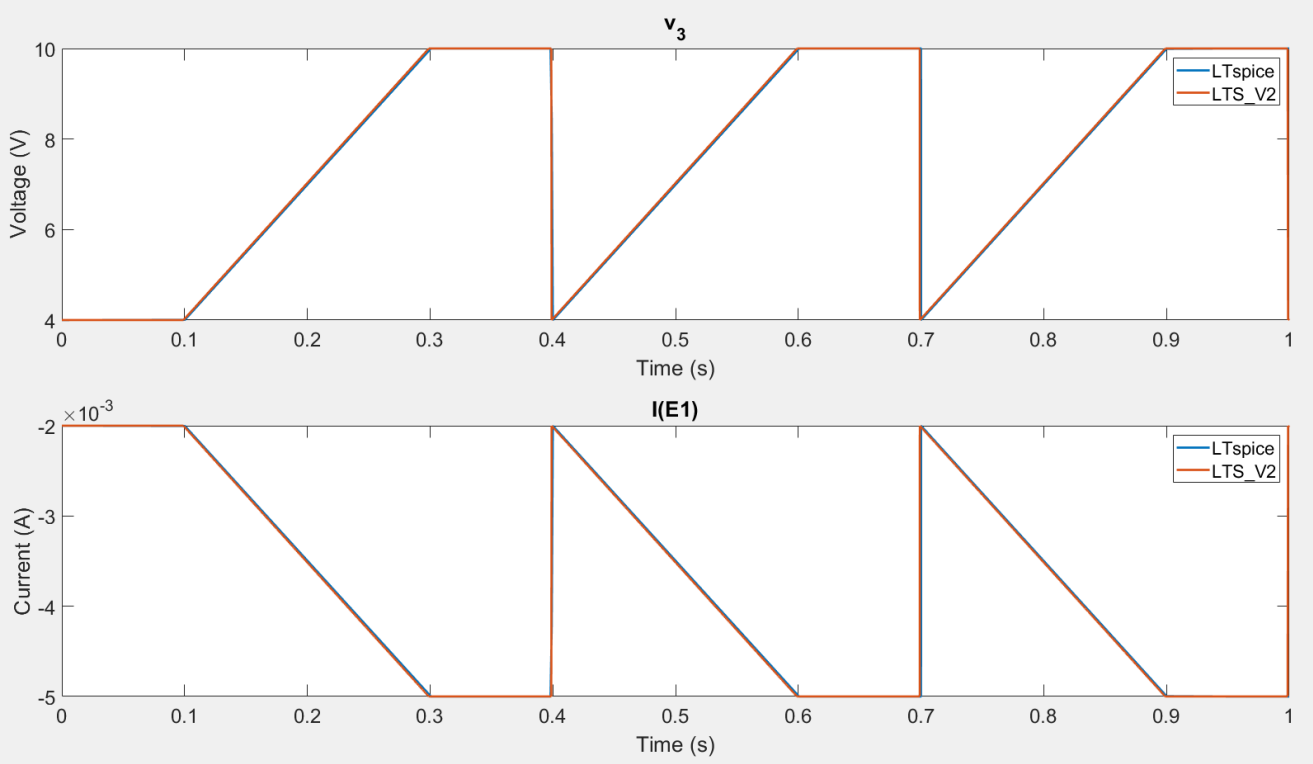


Figure 15. Waveforms produced by the circuit from Figure 14, using a static timestep of 10µs

Both the voltage and current at the source’s output are basically identical to the ones produced by LTspice.

Voltage-controlled current source

Voltage-controlled current sources behave like voltage-controlled voltage sources in that their current values are equal to their gain multiplied by the potential difference between the two controlling nodes. The source’s defining equations are based on the extra current that must now be considered at its two nodes.

Node N(+): $$Gain * \left(v_{NC(+)} - v_{NC(-)}\right) + other\ currents\ into\ N(+) = 0$$ *Eq. 38*

Node N(-): $$-Gain * \left(v_{NC(+)} - v_{NC(-)}\right) + other\ currents\ into\ N(-) = 0$$ *Eq. 39*

To implement these equations in matrix form, “+Gain” is added to the element described by the row corresponding to N(+) and the column corresponding to NC(+) in A. “-Gain” is added to the element described by the row corresponding to N(+) and the column corresponding to NC(-). The second equation is implemented using the same approach but with the row corresponding to N(-) and the signs reversed.

This is demonstrated using the circuit shown in Figure 16.

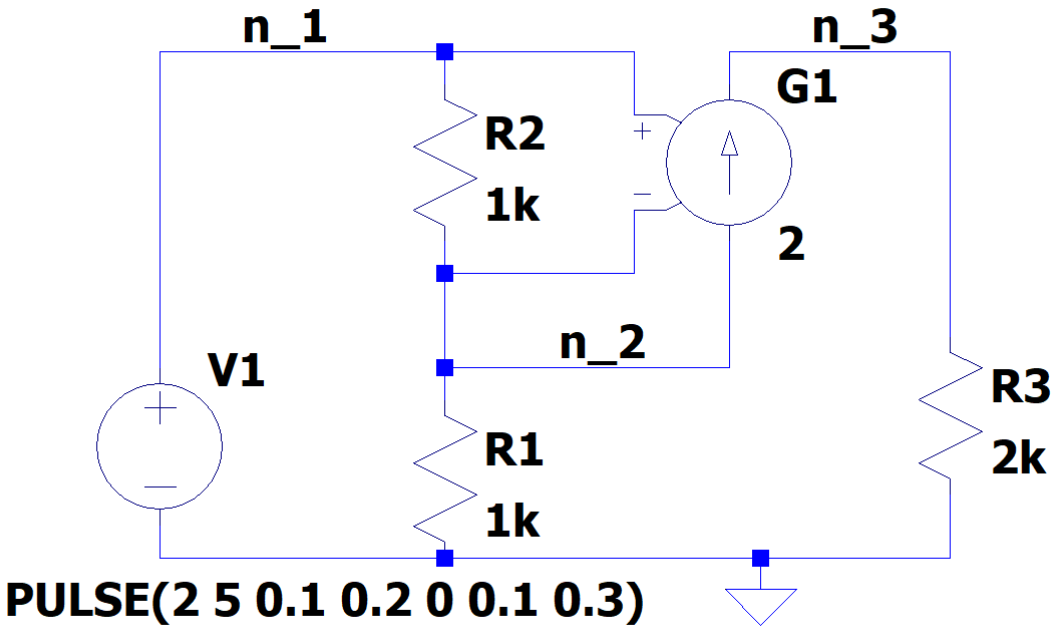


Figure 16. Voltage-controlled current source. Taken from [8].

Resulting matrix equation:

$$\begin{bmatrix} \frac{1}{R_1} & -\frac{1}{R_1} & 0 & 1 \\ -\frac{1}{R_1} - Gain & \frac{1}{R_1} + \frac{1}{R_2} + Gain & 0 & 0 \\ Gain & -Gain & \frac{1}{R_3} & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ i_{V1} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ V1 \end{pmatrix} \qquad \textit{Eq. 40}$$

The source changes four elements of the A matrix by adding or subtracting its gain value to/from them as described by the defining equations.

Simulating the circuit from Figure 16:

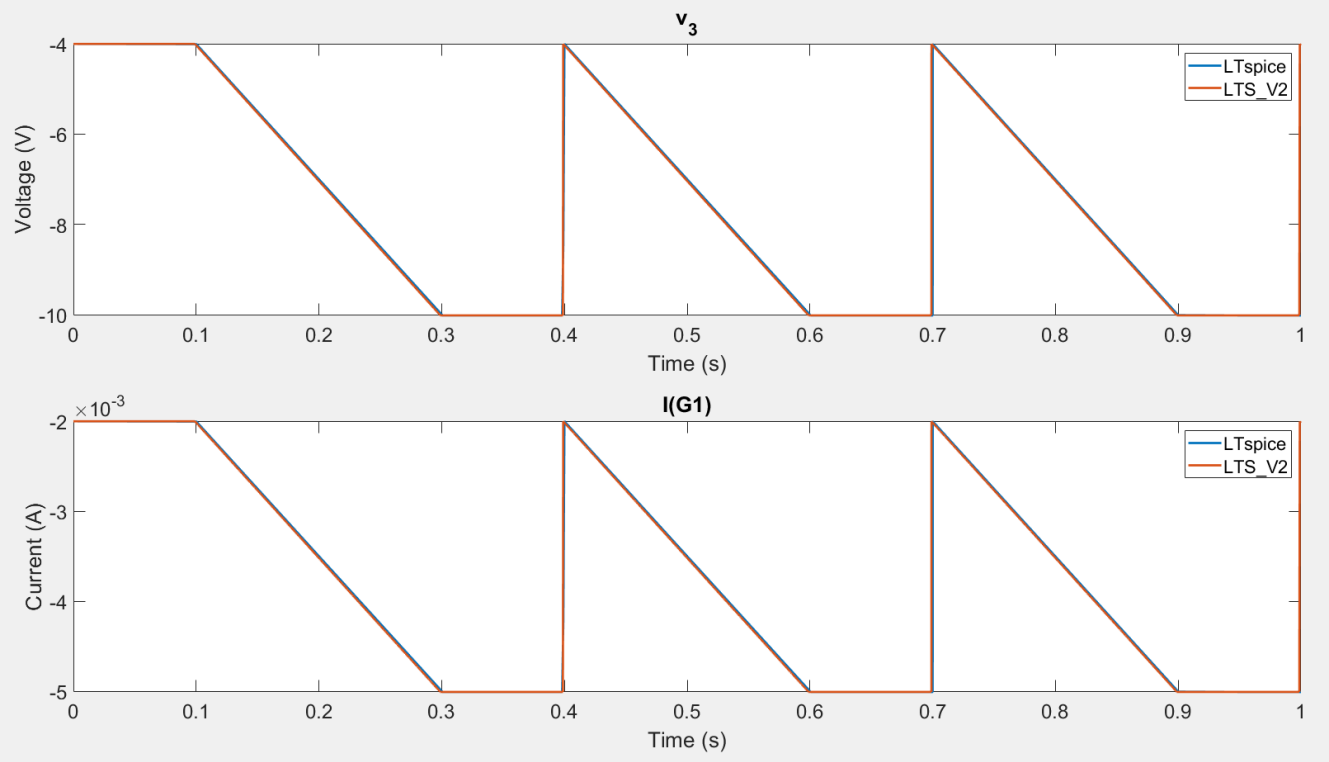


Figure 17. Waveforms produced by the circuit from Figure 16, using a static timestep of 10µs

Both the voltage and current at the source's output are basically identical to the ones produced by LTspice.

Current-controlled voltage source

Current-controlled voltage sources are controlled by the current flowing through a specified voltage source. Their potential difference is equal to a gain multiplied by this controlling current. The current must flow through a voltage source as these are the only currents calculated by MNA. A zero voltage source must be inserted, and taken as the controlling source, when a controlling current is required that does not flow through an already existing voltage source. The current-controlled source can be described by $v_{N(+)} - v_{N(-)} = Gain * I_{controlling}$ where $I_{controlling}$ is the current through the specified voltage source. Rearranging gives $v_{N(+)} - v_{N(-)} - Gain * I_{controlling} = 0$. Again, the current-controlled voltage source still behaves like a normal voltage source as described by the first part of the equation. However, some additional behaviour must be added that is described by the second half of the equation: The gain value must be subtracted from the element described by the row corresponding to the current-controlled voltage source and the column corresponding to the controlling voltage source.

This is demonstrated using the circuit shown in Figure 18.

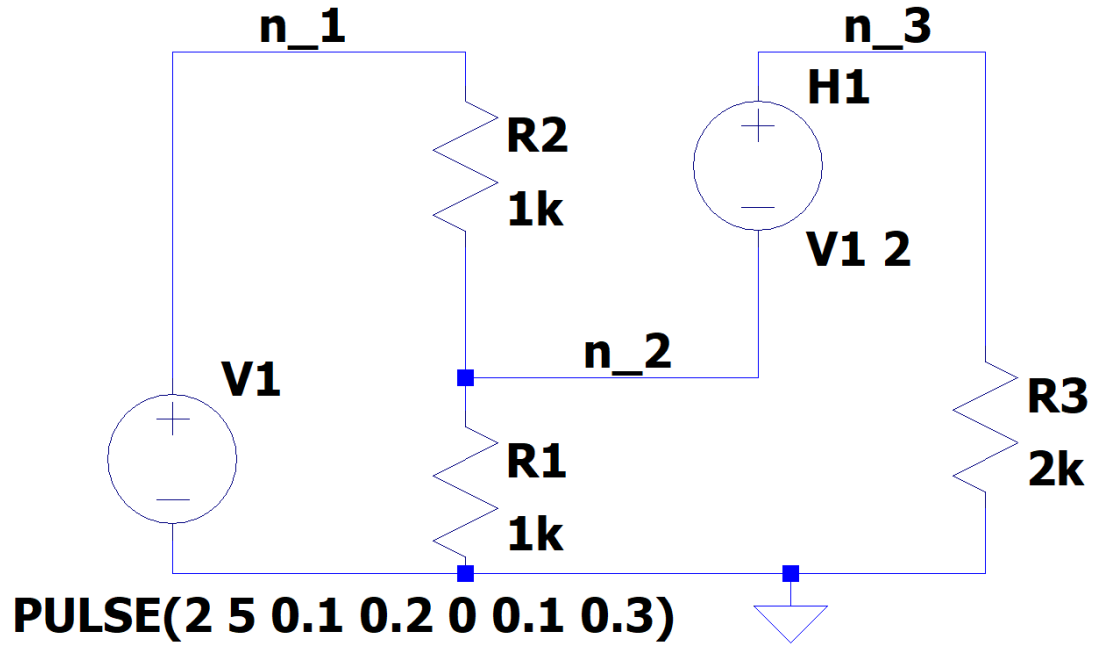


Figure 18. Current-controlled voltage source. Taken from [8].

Resulting matrix equation:

$$\begin{bmatrix} \frac{1}{R_1} & -\frac{1}{R_1} & 0 & 1 & 0 \\ -\frac{1}{R_1} & \frac{1}{R_1}+\frac{1}{R_2} & 0 & 0 & -1 \\ 0 & 0 & \frac{1}{R_3} & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & -Gain & 0 \end{bmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ i_{V1} \\ i_{H1} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ V1 \\ 0 \end{pmatrix} \quad \textit{Eq. 41}$$

The source adds the ± 1s to A due to its normal voltage source behaviour, and "-Gain" into element (5,4) as described by the defining equation. In this example, $I_{controlling} = i_{V1}$ as specified in Figure 18.

Current-controlled sources are more complex to implement than voltage-controlled ones due to the required reference to another circuit component. It was decided to link the index of the controlling voltage source, inside the solution vector, to the current-controlled source directly after parsing the netlist. This avoids the need to loop through all existing voltage sources every time that A must be recomputed and makes accessing the controlling current trivial.

Simulating the circuit from Figure 18:

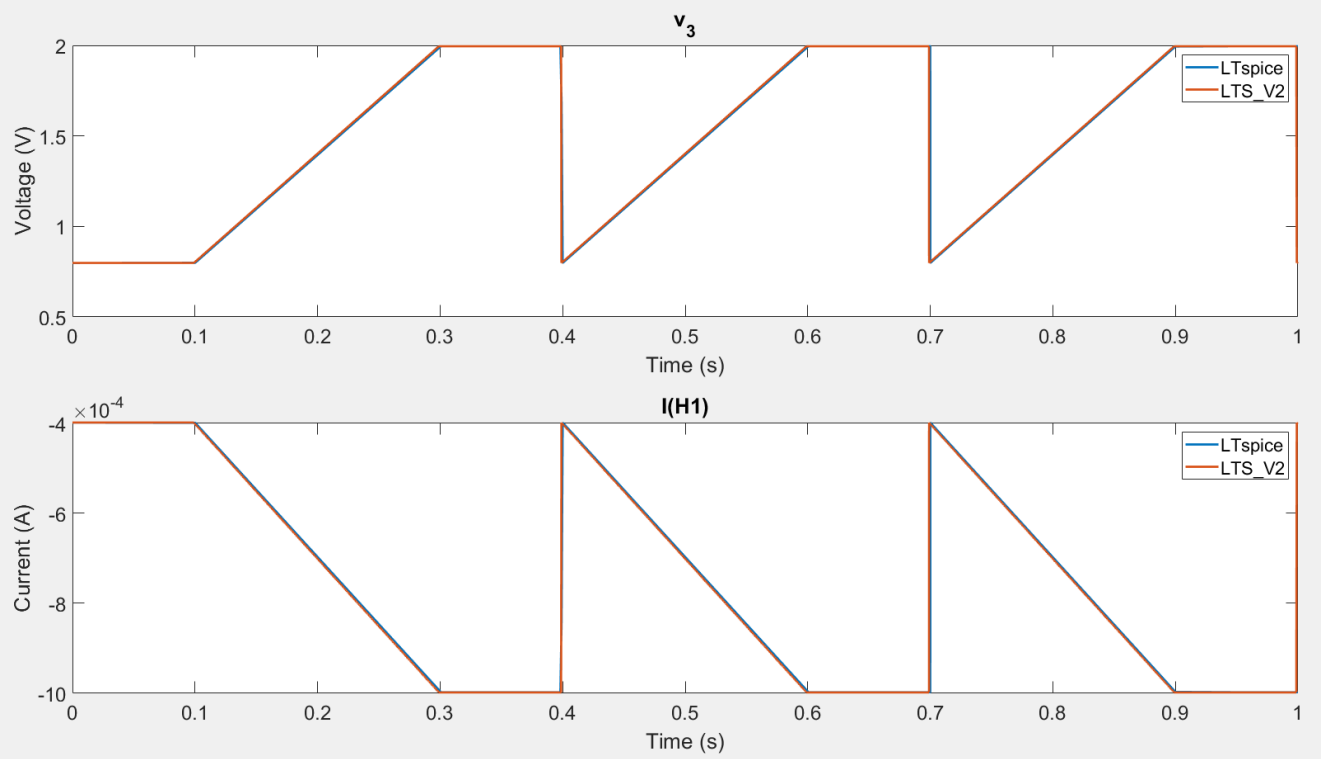


Figure 19. Waveforms produced by the circuit from Figure 18, using a static timestep of 10µs

Both the voltage and current at the source's output are basically identical to the ones produced by LTspice.

Current-controlled current source

Current-controlled current sources behave like current-controlled voltage sources in that their current value is determined by the current through a specified voltage source multiplied by a gain. The source's defining equations are based on the extra current that must now be considered at its two nodes.

Node N(+): $$Gain * I_{controlling} + other\ currents\ into\ N(+) = 0$$ *Eq. 42*

Node N(-): $$-Gain * I_{controlling} + other\ currents\ into\ N(+) = 0$$ *Eq. 43*

To implement these equations in matrix form, the column in A that corresponds to the voltage source through which the controlling current flows must be modified: "+Gain" must be inserted into the row corresponding to N(+) and "-Gain" into the row corresponding to N(-) of the current source. This is demonstrated using the circuit shown in Figure 20.

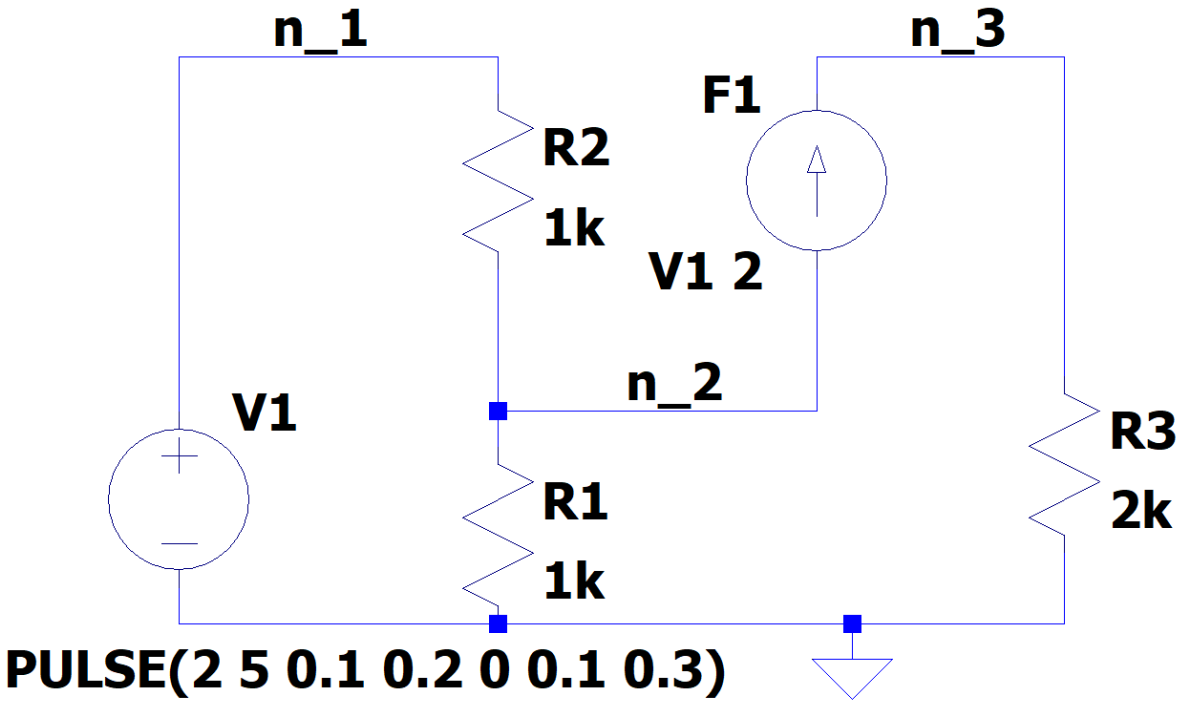


Figure 20. Current-controlled current source. Taken from [8].

Resulting matrix equation:

$$\begin{bmatrix} \frac{1}{R_1} & -\frac{1}{R_1} & 0 & 1 \\ -\frac{1}{R_1} & \frac{1}{R_1}+\frac{1}{R_2} & 0 & -Gain \\ 0 & 0 & \frac{1}{R_3} & Gain \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ i_{V1} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ V1 \end{pmatrix} \quad \textit{Eq. 44}$$

Again, $I_{controlling} = i_{V1}$.

Simulating the circuit from Figure 20:

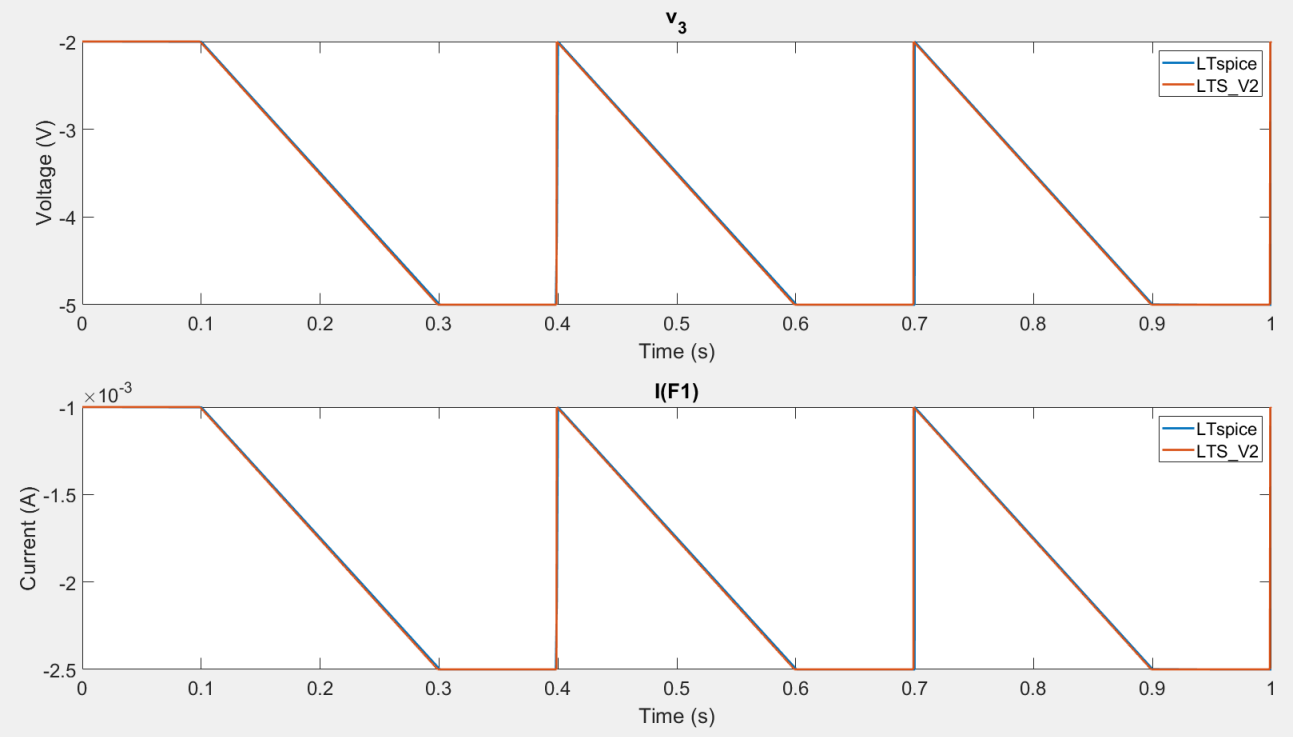


Figure 21. Waveforms produced by the circuit from Figure 20, using a static timestep of 10µs

Both the voltage and current at the source's output are basically identical to the ones produced by LTspice.

ii. Non-linear Sources

Non-linear sources are sources whose values change over time. Their values are recalculated at the end of each timestep for the next timestep.

They were implemented separately and then linked to the sources using a HAS-A relationship. This avoided to need to implement the same sources multiple times for both current and voltage sources.

A. SINE

Netlist syntax for the implemented sine wave:

<Vxxx or Ixxx> <N(+)> <N(-)> SIN(<offset> <amplitude> <frequency> <time delay> <damping factor> <phase shift in degrees>)

Delay, damping factor and phase shift are optional.

The circuit shown in Figure 22 is used to compare the sine waves produced by LTS_V2 with the ones produced by LTspice. The same waveform could be produced by a current source.

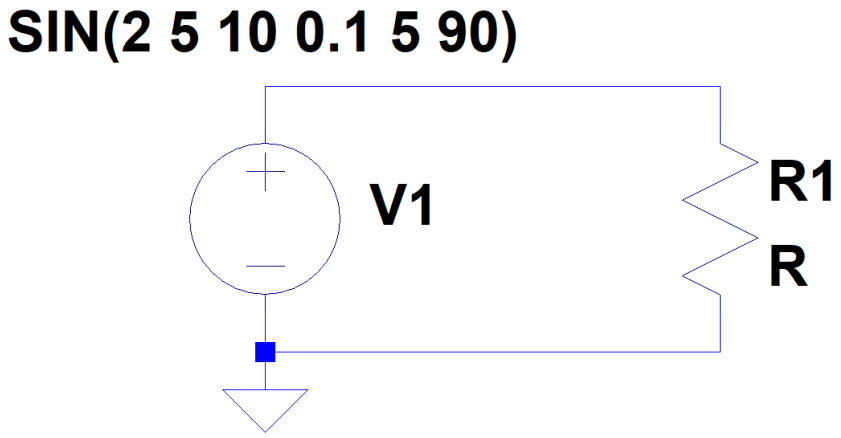


Figure 22. Sine waveform

Plotting LTS_V2's and LTspice's results on the same graph:

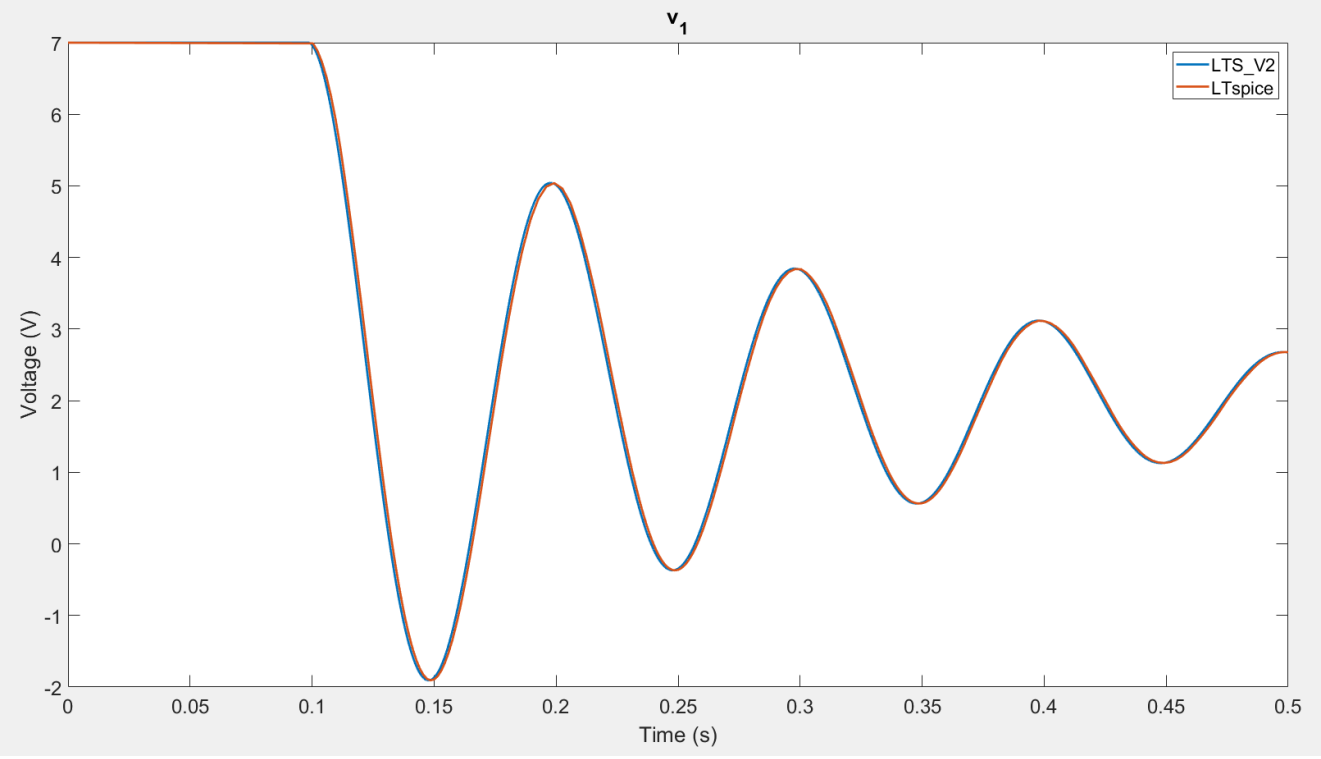


Figure 23. LTS_V2's and LTspice's results for Figure 22

The two waveforms appear to be almost identical. This confirms the accuracy of LTS_V2's sine waves.

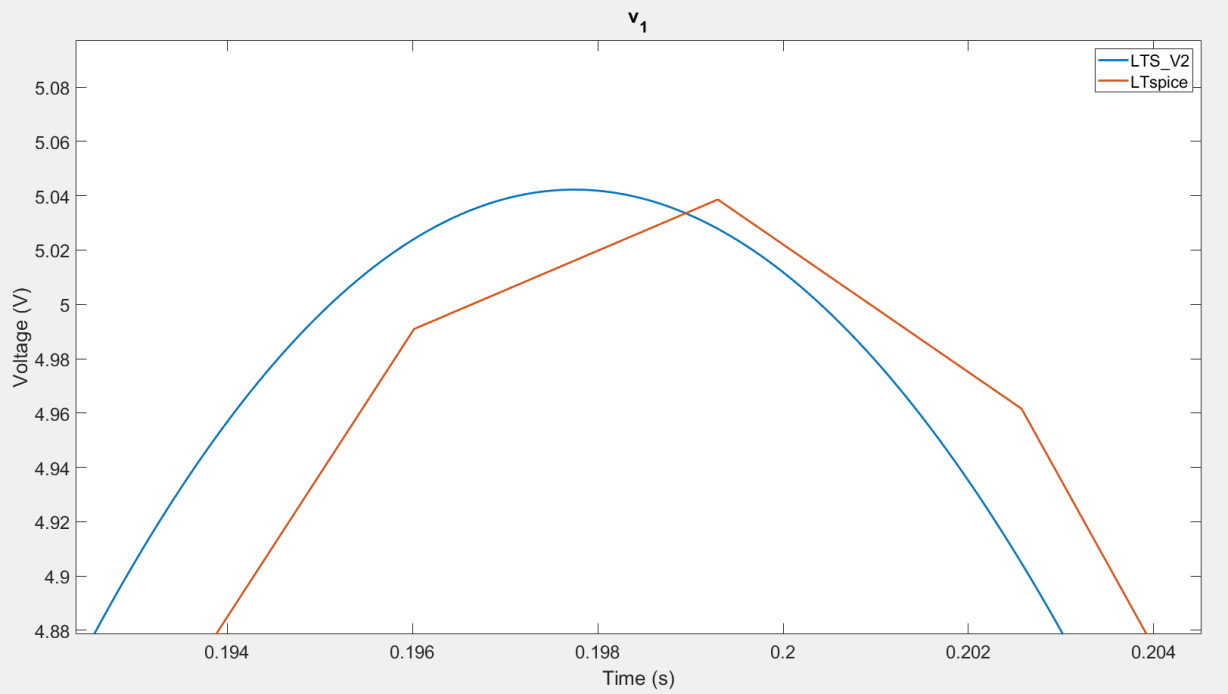


Figure 24. LTS_V2's and LTspice's results for Figure 22: Zoomed in view

When comparing the two more closely, a clear difference emerges. LTspice uses a rough approximation for the sinusoid when compared with LTS_V2. The error is likely there due to a more efficient approximation, which results in increased speed but slightly decreased accuracy.

B. PWL (PIECEWISE LINEAR)

Netlist syntax for the implemented PWL wave:

<Vxxx or Ixxx> <N(+)> <N(-)> PWL(<t1> <v1> <t2> <v2> <t3> <v3> ...)

<vx> specifies the voltage/current at time = <tx>.

At least one time and one value argument must be provided. There is no limit on how many arguments can be provided.

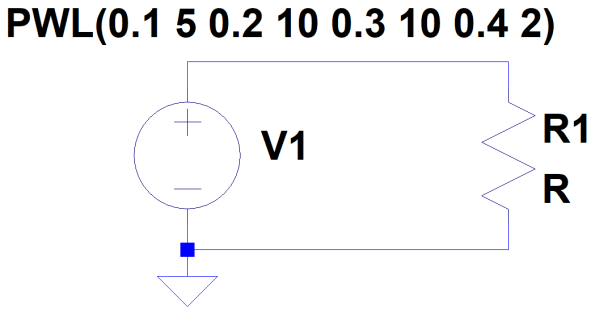


Figure 25. PWL waveform

Plotting LTS_V2's and LTspice's results on the same graph:

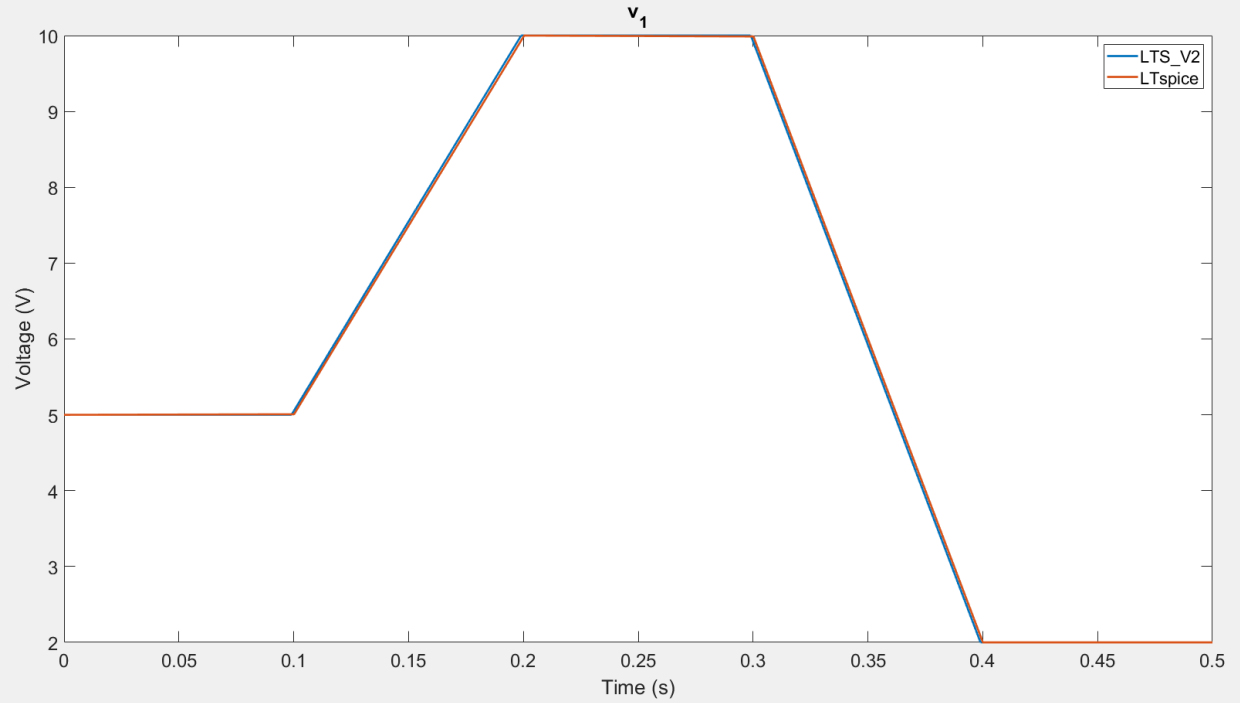


Figure 26. LTS_V2's and LTspice's results for Figure 25

Again, the two waveforms are almost identical. This confirms the accuracy of LTS_V2's PWL waves.

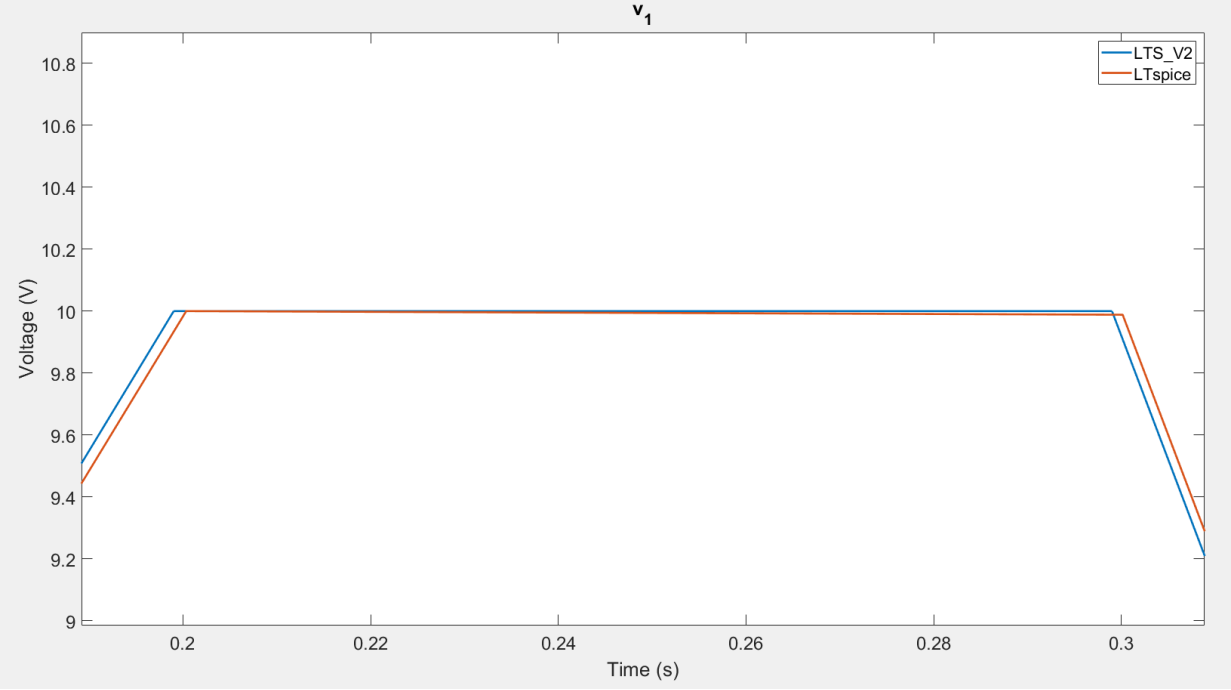


Figure 27. LTS_V2's and LTspice's results for Figure 25: Zoomed in view

Figure 27 shows that LTspice's waveform has some errors, being slanted when expected to be flat. This does not happen in LTS_V2. The error once again is small and likely insignificant.

C. PULSE

Netlist syntax for the implemented pulse wave:

<Vxxx or Ixxx> <N(+)> <(N(-)> PULSE(<initial value> <pulsed value> <delay> <rise time> <fall time> <on time> <period>)

There are no optional arguments.

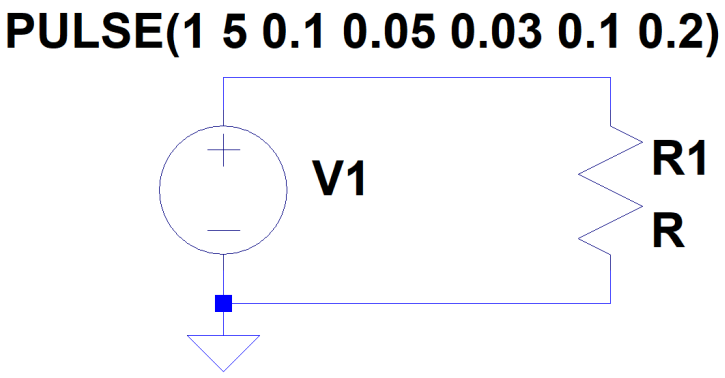


Figure 28. Pulse waveform

Plotting LTS_V2's and LTspice's results on the same graph:

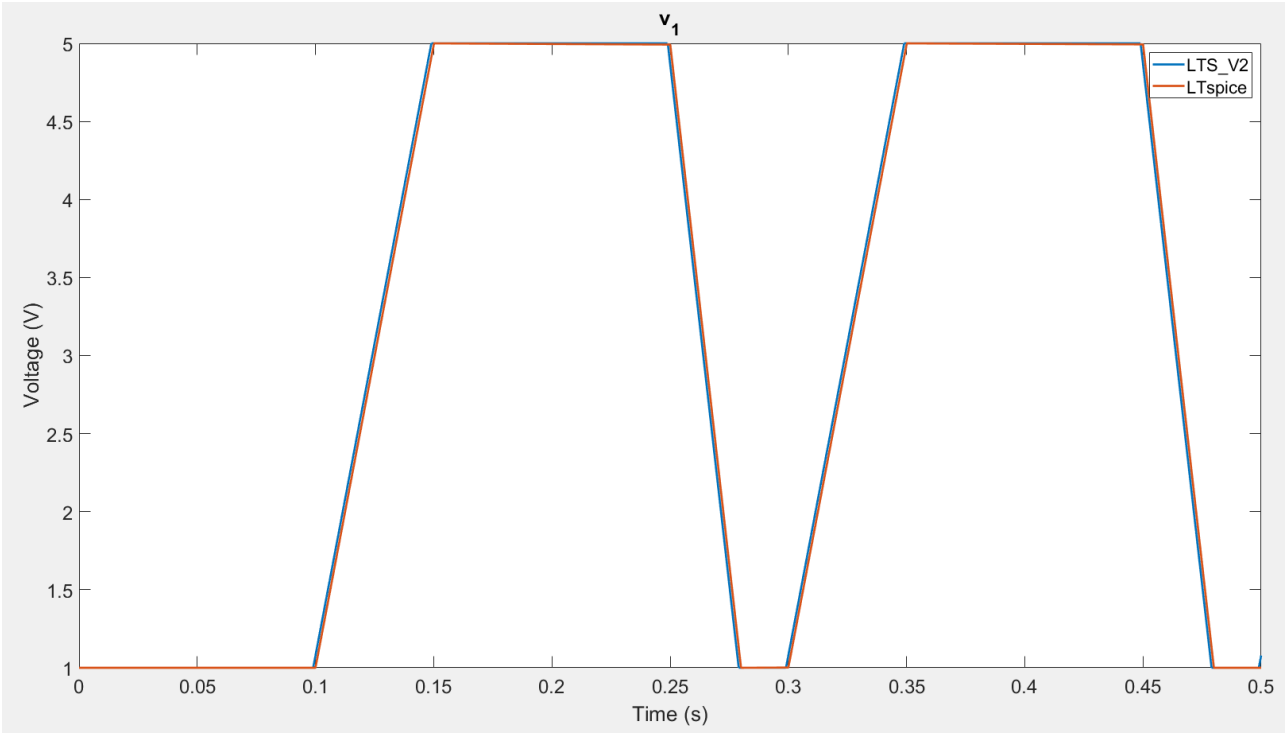


Figure 29. LTS_V2's and LTspice's results for Figure 28

The same negative slope of LTspice's results, as was the case for the PWL waveform, can be observed at the top.

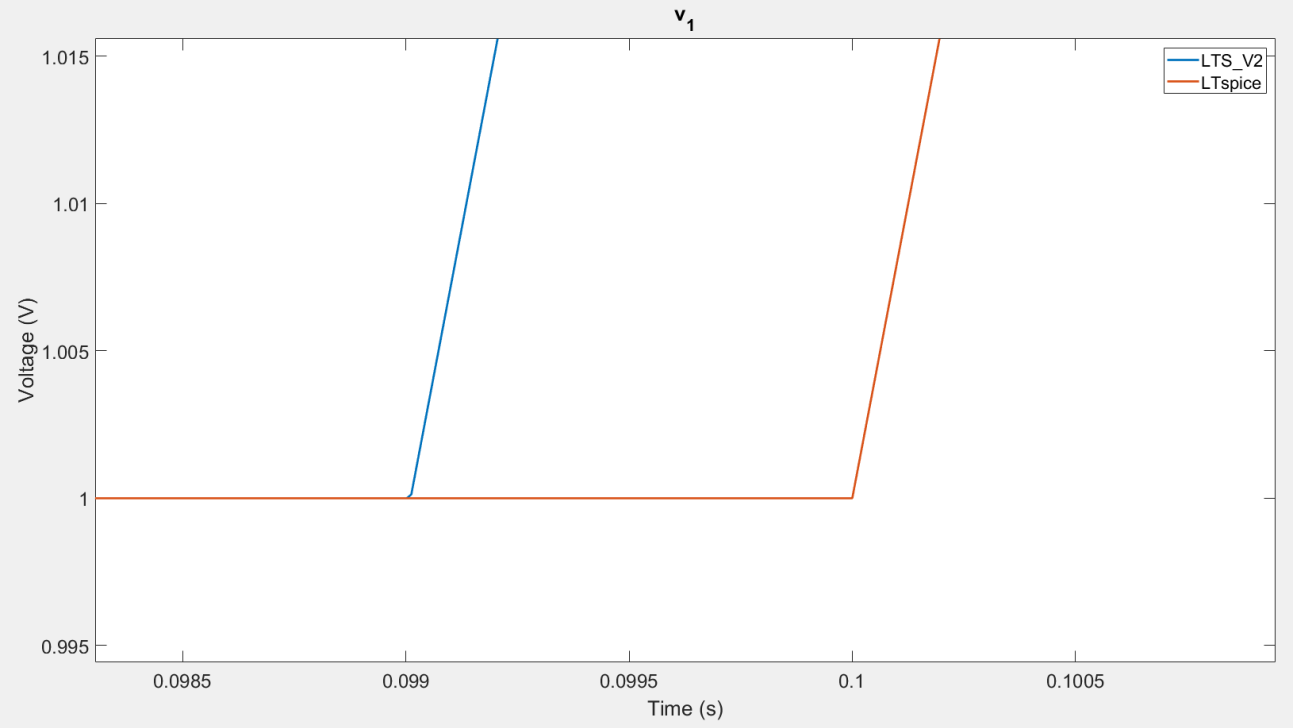


Figure 30. LTS_V2's and LTspice's results for Figure 28: Zoomed in view

For all the tested waveforms, a slight discrepancy between the start time of the two waveforms was noticed: one always seems to be slightly delayed compared to the other. Figure 30 shows that LTS_V2 starts slightly too late (at 0.099 rather than 0.1s). This is likely due to rounding errors and not significant.

iii. Non-Linear component solutions

A. MATRIX GENERATION FOR NON-LINEAR CIRCUITS

The method that was chosen for non-linear circuits came from the result of experimenting with both initial ideas. The code structure used to implement the previously described direct Newton-Raphson method was picked due to it being cleaner and more flexible. However, the mathematical principles described in the indirect Newton-Raphson method were used, as these showed a greater range of convergence (See Appendix B) and proved to be easier to work with.

Using *Eq. 15*, the linearized circuits for the following components can be derived.

B. DIODE

From the Poisson diode equation $I = Is * \left(e^{\frac{V}{N*VT}} - 1\right)$ [9], the linearized model can be derived using the above method, to obtain the following equations for Ieq, and g (See Appendix D):

$$Ieq = I(V) * \left(1 - \frac{V}{N * VT}\right) \quad \textit{Eq. 45}$$

$$g = \frac{I(V)}{N * VT} \quad \textit{Eq. 46}$$

As demonstrated previously, the diode can be modelled as shown in Figure 31.

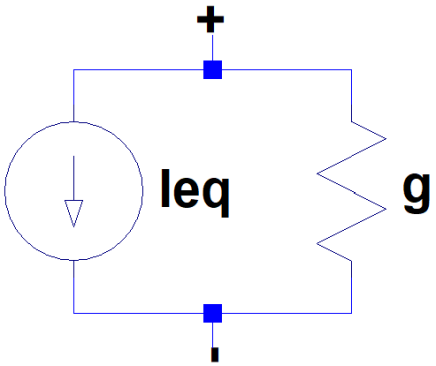


Figure 31. Diode model

Where $Ieq = I(V) * \left(1 - \frac{V}{N*VT}\right)$ and $g = \frac{I(V)}{N*VT}$

After implementing this model, the test circuit shown in Figure 32, was simulated with results plotted on the same graph as LTspice's (Figure 33).

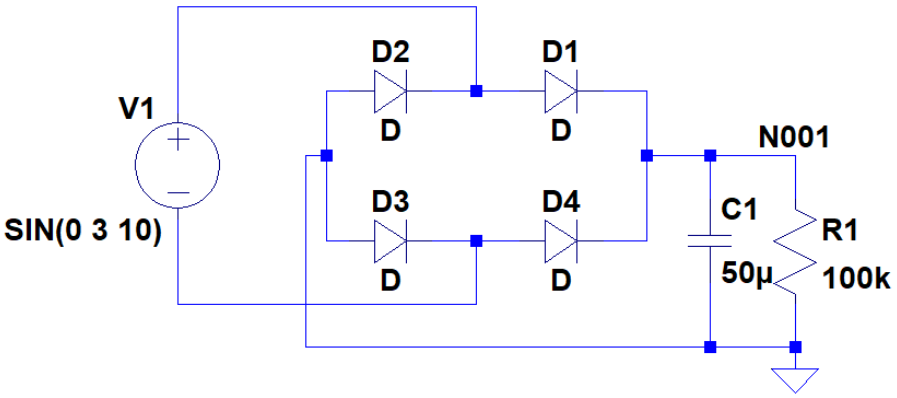


Figure 32. Diode test circuit

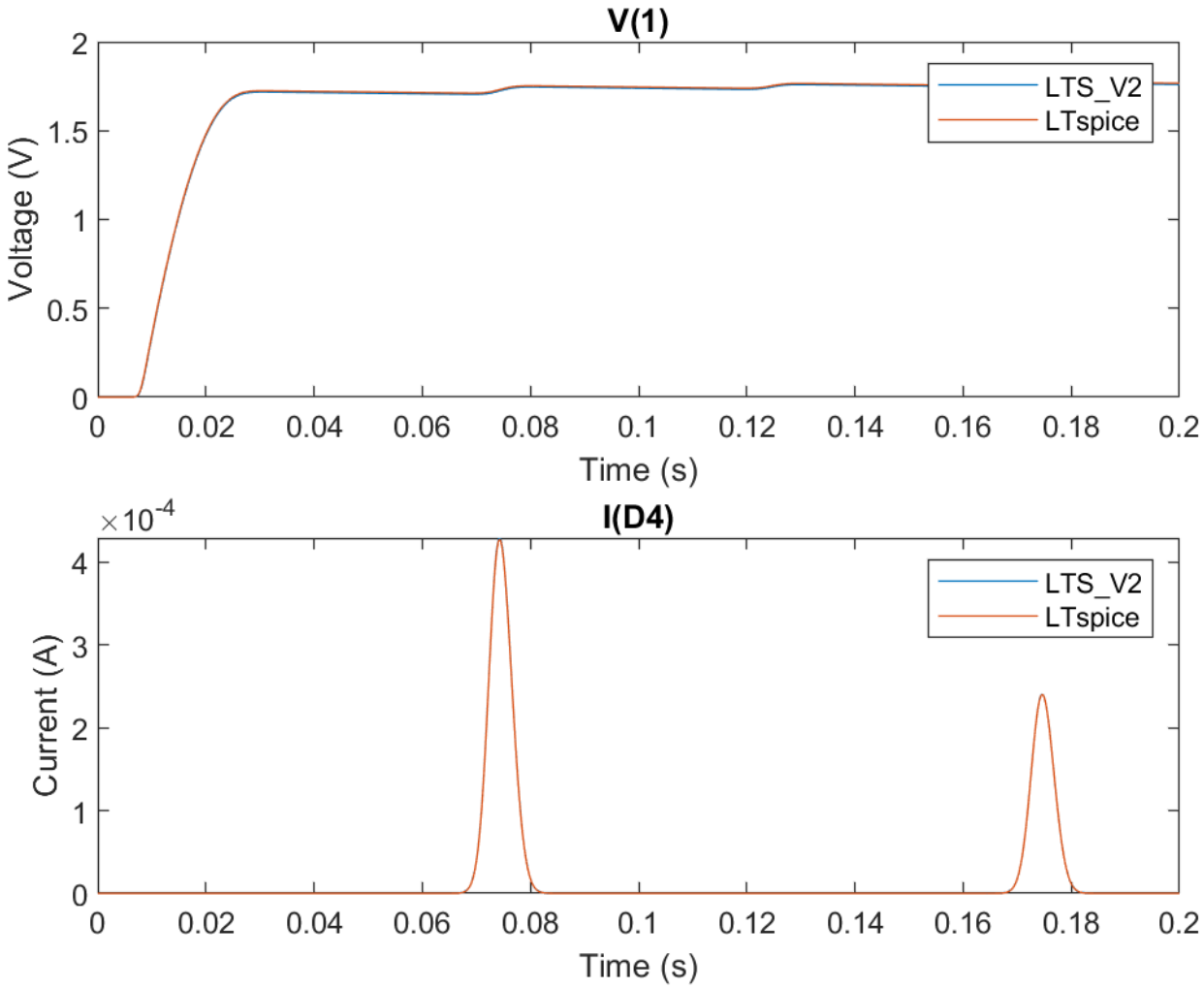


Figure 33. Waveforms produced for the voltage N001 and current through D4 in Figure 32 using a dynamic timestep

The fit is almost perfect. Only a tiny discrepancy can be observed: The voltage at node one lags slightly in LTS_V2 compared to LTspice. This could be due to LTspice's diodes using a more complex model with several more parameters than LTS_V2's. However, it could also be a difference in rounding as this is small enough to be considered insignificant.

C. MOSFET

The MOSFET derivation is not as simple that of the diode. The method developed for linearizing non-linear equations only works for two nodes at a time. For multi-node components an expanded version of the previously described equation must be used (See Appendix E).

The expanded equation gives the current at a node due to a current source Ieq as:

$$Ieq = I_{x_a,n} - \sum_{b=1,b\neq a}^{k} \left(V_{x_a x_b,n} * g_{x_a x_b,n}\right) \qquad \textit{Eq. 47}$$

x_a represents the node from which the current is being calculated, and each subsequent x_b represents every other node in the component. $g_{x_a x_b,n} = \frac{dI_{x_a}}{dV_{x_a x_b}}\left(V_{x_a x_b,n}\right)$ is the conductance between nodes a and b.

This system works well with the implemented matrix generating algorithm: when constructing the A matrix, the algorithm requires the conductance of each component between each pair of nodes. This is represented by $g_{x_a x_b}$.

Additionally, the current equations for the MOSFET change depending on the operating mode. The current at each node, shown in Figure 34, at each operating point for an N-Chanel MOSFET is described in Table 1:

Table 1. N-Channel MOSFET currents

	Cut-Off VGS-VT < 0	Saturation 0 <= VGS-VT < VDS	Triode 0 <= VDS <= VGS-VT
Drain Current (ID)	0	$\frac{K}{2}(VGS-VT)^2 * \left(1+\frac{VDS-VDS_{sat}}{V_a}\right)$	$K\left((VGS-VT)-\frac{VDS}{2}\right)VDS$
Gate Current (IG)	0	0	0
Source Current (IS)	0	$\frac{K}{2}(VGS-VT)^2 * \left(1+\frac{VDS-VDS_{sat}}{V_a}\right)$	$K\left((VGS-VT)-\frac{VDS}{2}\right)VDS$

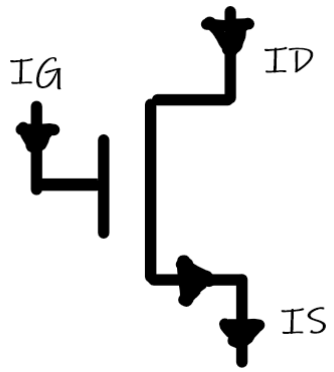


Figure 34. NMOS

Using these current relationships to develop a linearised model using the above method results in the model shown in Figure 35, which has its variables defined in Table 2. (See Appendix F for derivation)

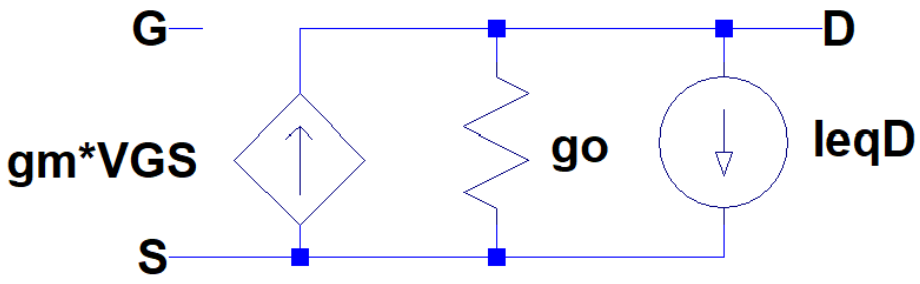


Figure 35. NMOS linearized circuit

Table 2. N-Channel MOSFET parameters

	Cut-Off	Saturation	Triode
g_m	0	$\sqrt{2KI_D}$	$K * VDS$
g_o	0	$\frac{K(VGS-VT)^2}{2V_a}$	$K\big((VGS-VT)-VGS\big)$
Ieq_D	0	$I_{D,n} - g_m VGS_n - g_o VDS_n$	$I_{D,n} - g_m VGS_n - g_o VDS_n$

The P-Channel MOSFET current equations are similar to the N-Channel ones. The only differences being the direction of the current at each node and that the voltages are flipped: VGS in the N-Channel becomes VSG in the P-Channel.

Figure 36 and Table 3 represent the P-Channel MOSFET linearized circuit.

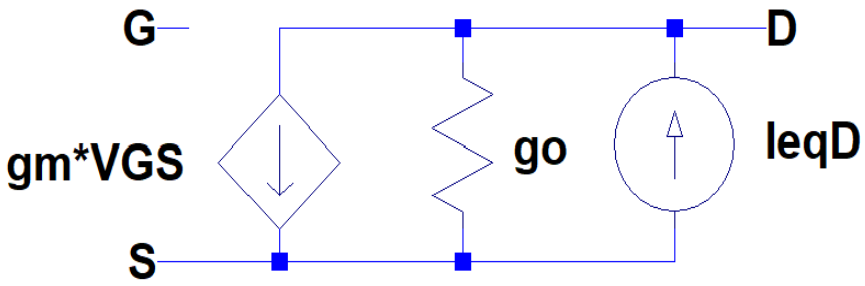


Figure 36. PMOS linearized circuit

Table 3. P-Channel MOSFET parameters

	Cut-Off VGS – VT < 0	Saturation VGS-VT < 0 AND VDS+VGS+VT < 0	Triode VDS
g_m	0	$\sqrt{2KI_D}$	$K * VDS$
g_o	0	$\frac{K(VGS-VT)^2}{2V_a}$	$K\big((VGS-VT)-VGS\big)$
Ieq_D	0	$I_{D,n} - g_m VGS_n - g_o VDS_n$	$I_{D,n} - g_m VGS_n - g_o VDS_n$

Once implemented both the NMOS and PMOS simulations can be run, with the circuits Figure 37 and Figure 38 respectively.

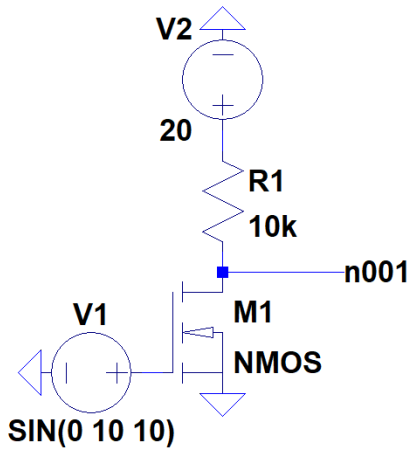


Figure 37. NMOS example

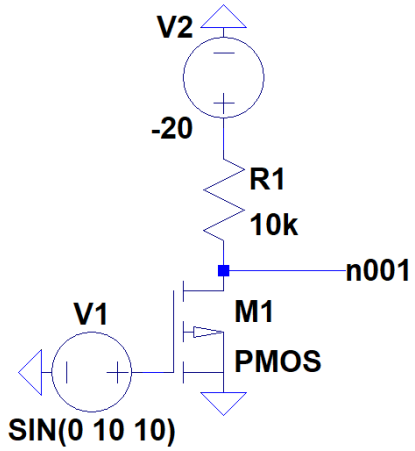


Figure 38. PMOS example

When simulated in LTS_V2 (Using a dynamic timestep) and comparing to LTspice, the results are very accurate as shown in Figure 39 and Figure 40.

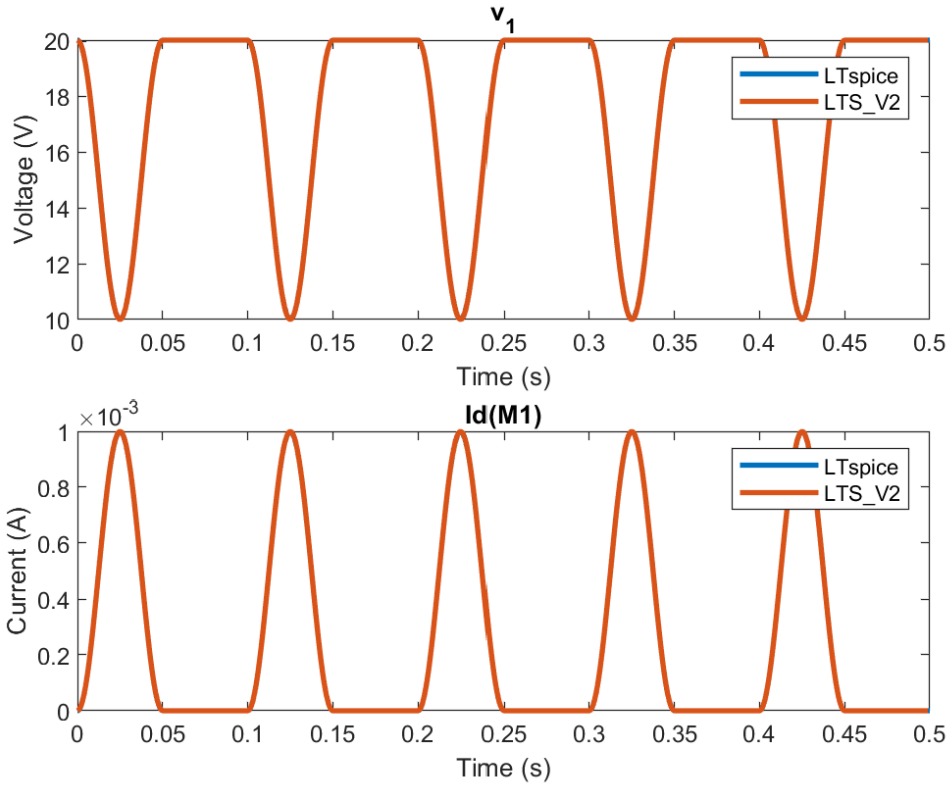


Figure 39. Waveform produced at N001 and current Id(M1) in Figure 37

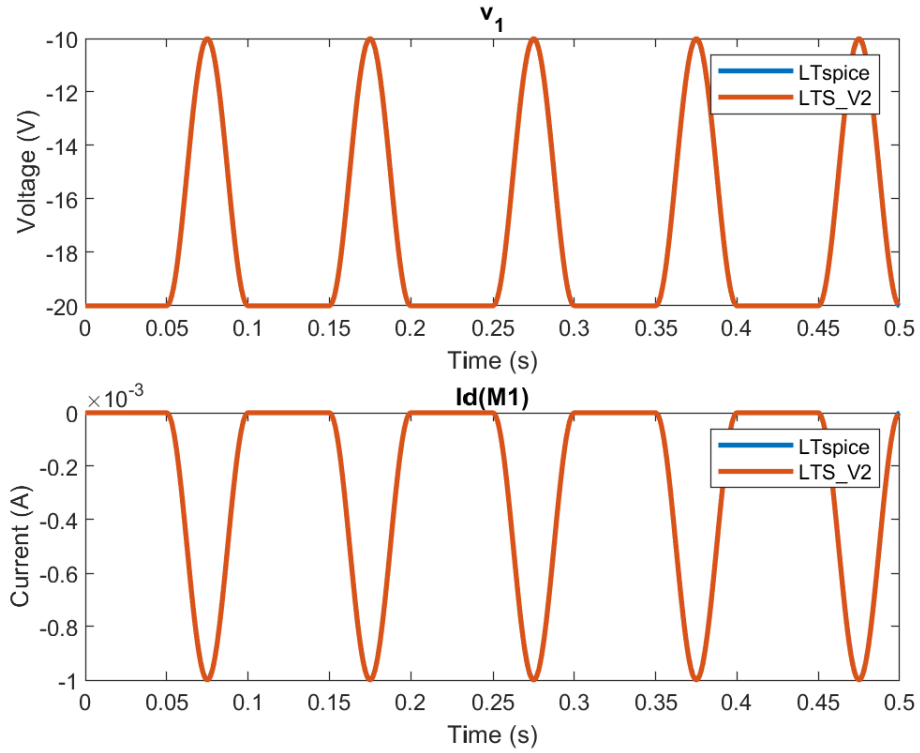


Figure 40. Waveform produced at N001 and current Id(M1) in Figure 38

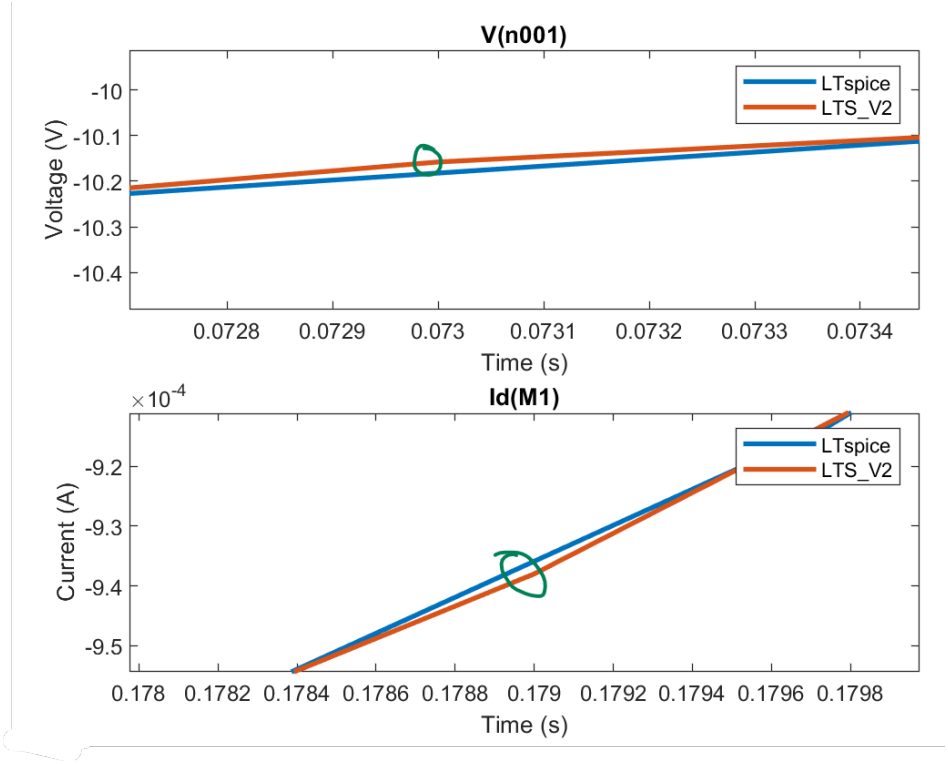


Figure 41 Zoomed in version of Figure 40

By inspection, the similarity between the two results is uncanny. They are perfectly identical. There is some small discrepancy at the peaks when zoomed in shown in Figure 41, however this is miniscule in the scale of the results.

Looking with more detail at the zoomed in graph in Figure 41, the LTS_V2 print step can bee seen at the sharp edges. It is observed that the maximum error occurs at these print steps. This is expected, as LTS_V2's maximum error is directly dependant on the "print step". The error can be made smaller by making the print step smaller; however, this increases simulation time. Print step is discussed in detail in (section 4.vii. Dynamic timestep)

D. BJT

The bipolar junction transistor is more complicated. The expanded linearization method can be applied to it, as was done for the MOSFET. However, this becomes complex to deal with. A simpler method replaces the BJT with an equivalent large signal model shown in Figure 42: the Gummel-Poon Mid Band model [9].

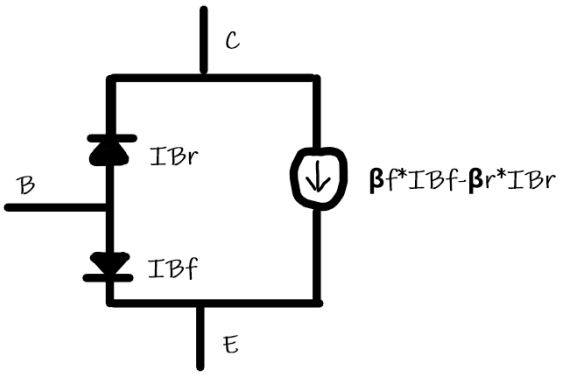


Figure 42. Mid Band Gummel-Poon NPN BJT model

One benefit of this model is that it encapsulates the behaviour of the NPN BJT in all operating regions, which simplifies the calculations. This is accomplished by suitable voltage differences activating/disactivating the appropriate diodes.

Using this model, the linearized model is derived in Appendix G, and the results shown in Figure 43 and Figure 44, with parameters described in Table 4.

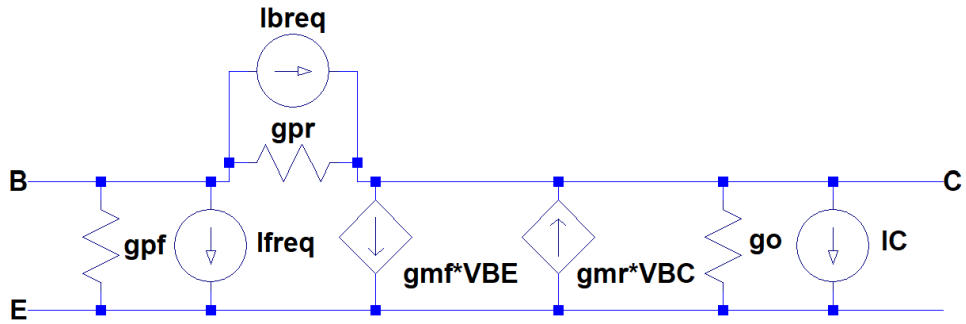


Figure 43. Mid Band Gummel-Poon NPN BJT linearized circuit

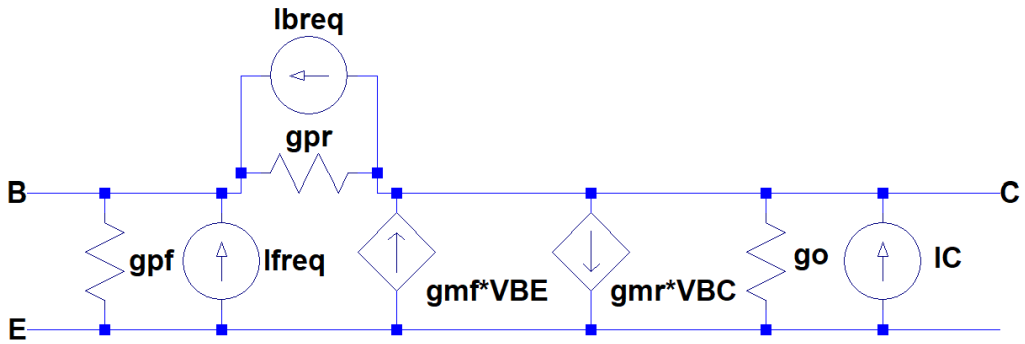


Figure 44. Mid Band Gummel-Poon PNP BJT linearized circuit

Table 4. Mid Band Gummel-Poon model parameters

	NPN	PNP
$g_{\pi,f}$	$\frac{I_{fs}}{\beta_f * VT} * e^{\frac{VBE}{VT}}$	$\frac{I_{fs}}{\beta_f * VT} * e^{\frac{VEB}{VT}}$
$g_{\pi,r}$	$\frac{I_{rs}}{\beta_R * VT} * e^{\frac{VBC}{VT}}$	$\frac{I_{rs}}{\beta_R * VT} * e^{\frac{VCB}{VT}}$
$g_{m,f}$	$\frac{I_{fs}}{VT} * e^{\frac{VBE}{VT}}$	$\frac{I_{fs}}{VT} * e^{\frac{VEB}{VT}}$
$g_{m,r}$	$\frac{I_{rs}}{VT} * e^{\frac{VBC}{VT}}$	$\frac{I_{rs}}{VT} * e^{\frac{VCB}{VT}}$
g_o	$\frac{I_E}{V_a}$	$\frac{I_E}{V_a}$
$I_{bf,eq}$	$I_{bf}(VBE) - g_{\pi,f} * VBE$	$I_{bf}(VEB) - g_{\pi,f} * VEB$
$I_{br,eq}$	$I_{br}(VBC) - g_{\pi,r} * VBC$	$I_{br}(VCB) - g_{\pi,r} * VCB$
$I_{C,eq}$	$\mathrm{IC}(VBE,VBC) - g_{m,f} * \mathrm{VBE} + g_{m,r} * \mathrm{VBC} - g_o * \mathrm{VCE}$	$\mathrm{IC}(VEB,VCB) - g_{m,f} * VEB + g_{m,r} * VCB - g_o * VEC$

Once implemented, both the NPN and PNP BJT's can be tested with the circuits in Figure 45 and Figure 46 respectively.

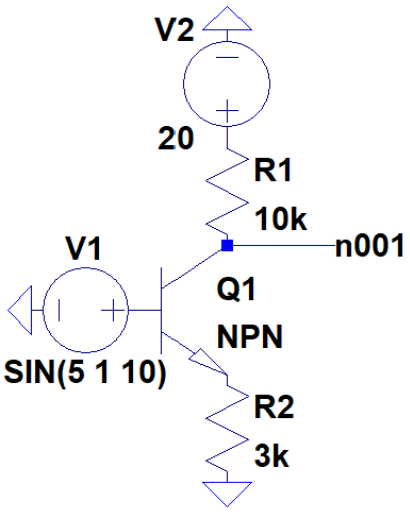


Figure 45. NPN example

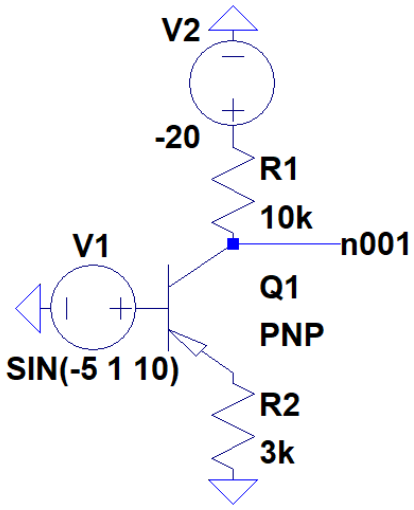


Figure 46. PNP example

When simulating in LTS_V2 (Using a dynamic timestep) and comparing to LTspice's, the results shown in Figure 47 and Figure 48 are very accurate.

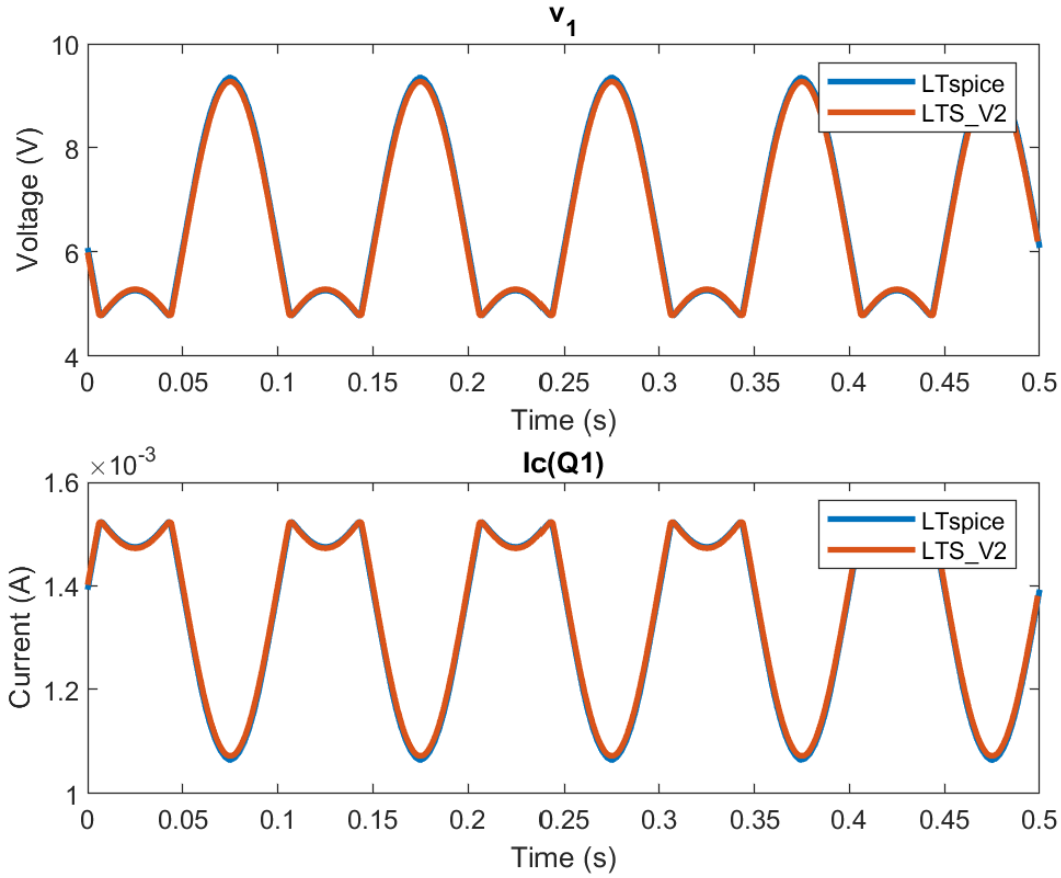


Figure 47. Waveform produced at n001 and current IC(Q2) in Figure 45

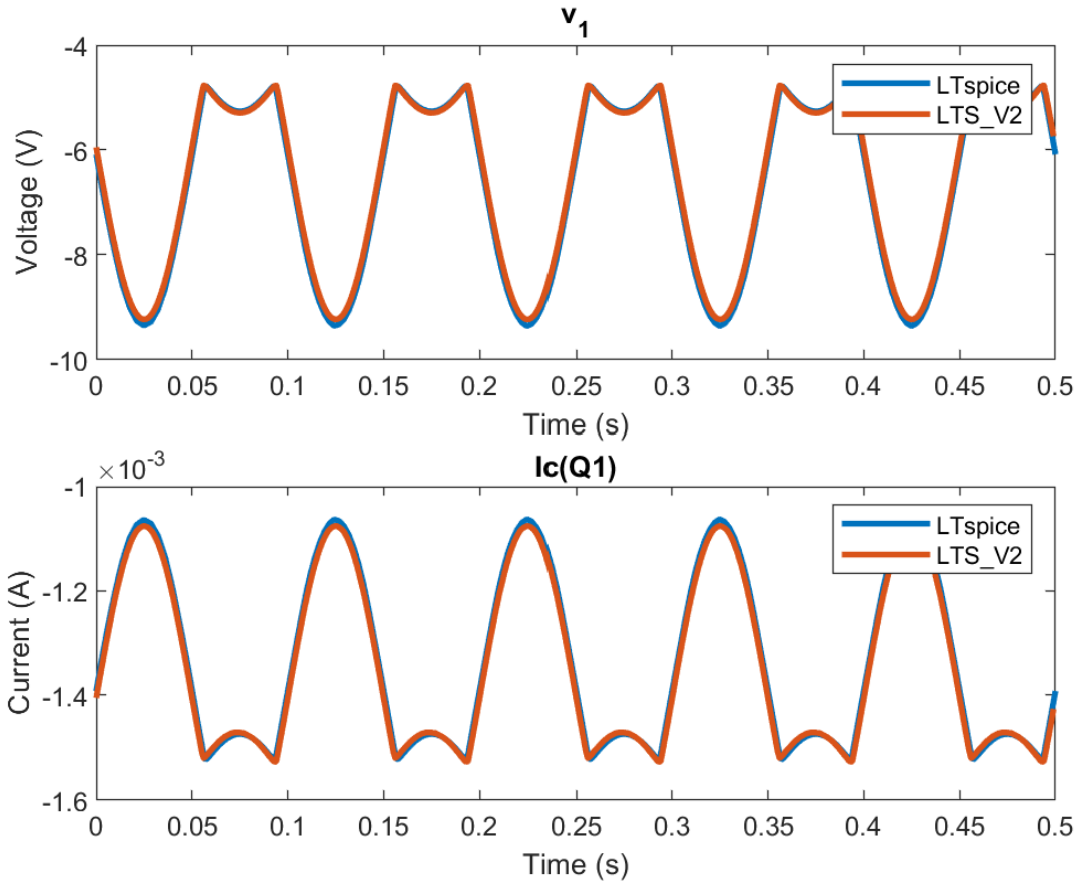


Figure 48. Waveform produced at n001 and current IC(Q2) in Figure 46

It can be observed that LTS_V2 produces very accurate results for both NPN and PNP. There are small errors near the extrema, which is apparent at the large peaks. This is most likely is due to the simple model used, as well as certain simplified parameters: Parasitic capacitances, thermal effects, and more are not modelled by LTS_V2. Nevertheless, the accuracy is remarkable.

iv. Operating point for linear circuits

The DC operating point gives the steady-state voltages for each node in the circuit [10]. Without this, all circuit values would be zero at time equals zero. This causes all sources to act as pulses from zero to their true value. Instead, the simulation should begin with all previous transients having died away.

The operating point is obtained by treating all capacitors as open-circuits and all inductors as short-circuits. Once the node voltages of this DC circuit have been obtained, they are used to initialize the capacitor and inductor voltages/currents. These values are then used to start the transient analysis at the values of the operating point rather than at zero.

Open circuits are implemented by simply removing capacitors from the circuit. Short- circuits are implemented by replacing inductors with a zero-voltage voltage source.

The circuit from Figure 27 is used to demonstrate linear transient analysis without and with a DC operating point calculation.

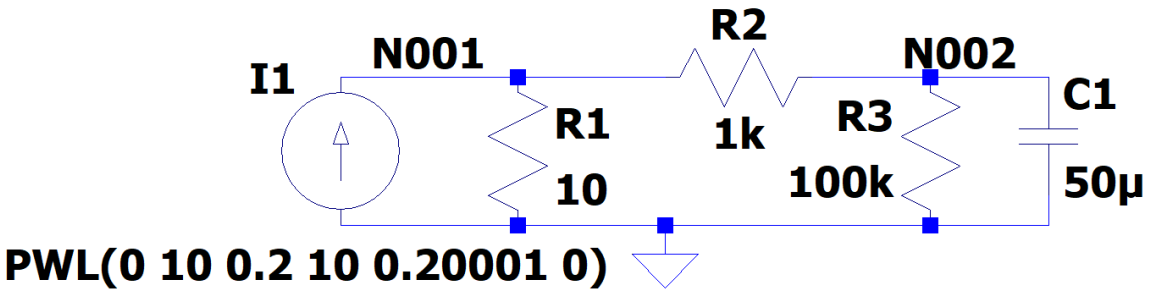


Figure 27. Operating point example circuit

Without DC operating point calculation:

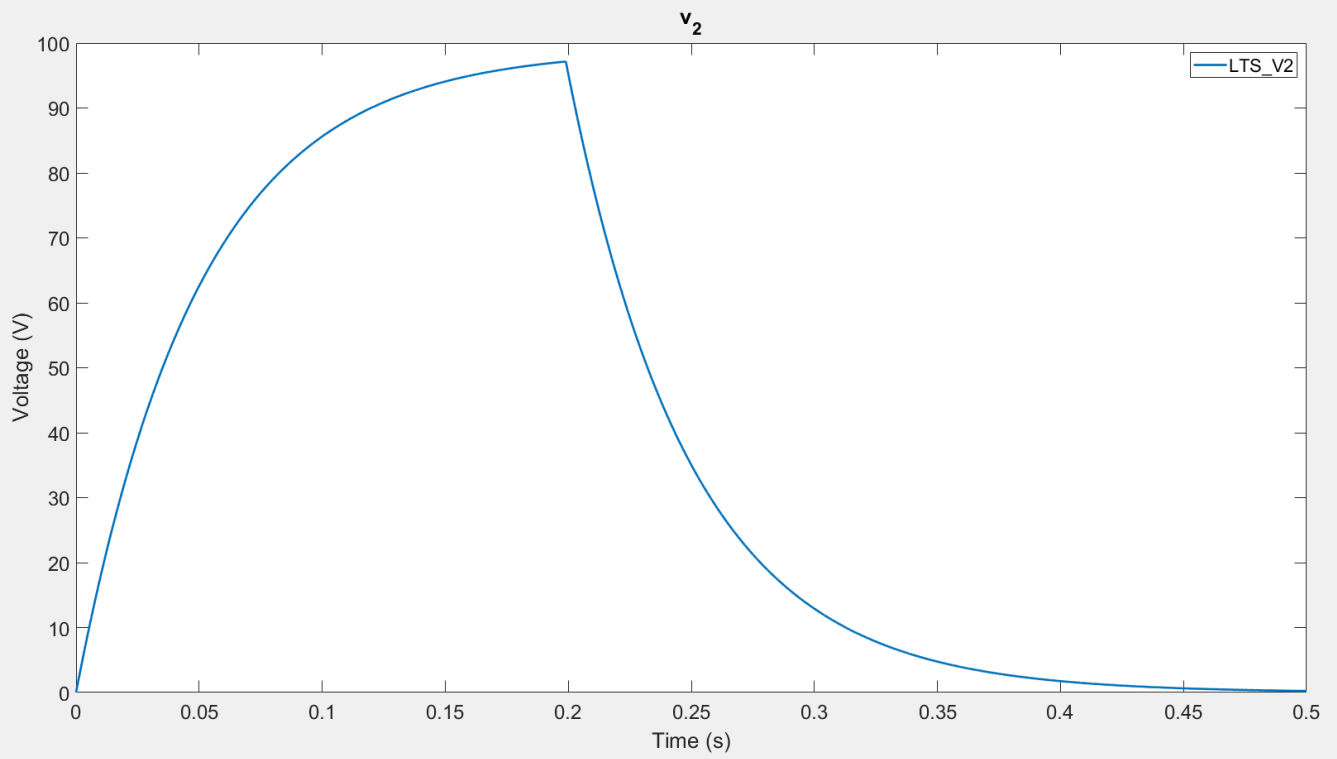


Figure 28. Results for Figure 27 when no operating point calculation is performed

With DC operating point calculation:

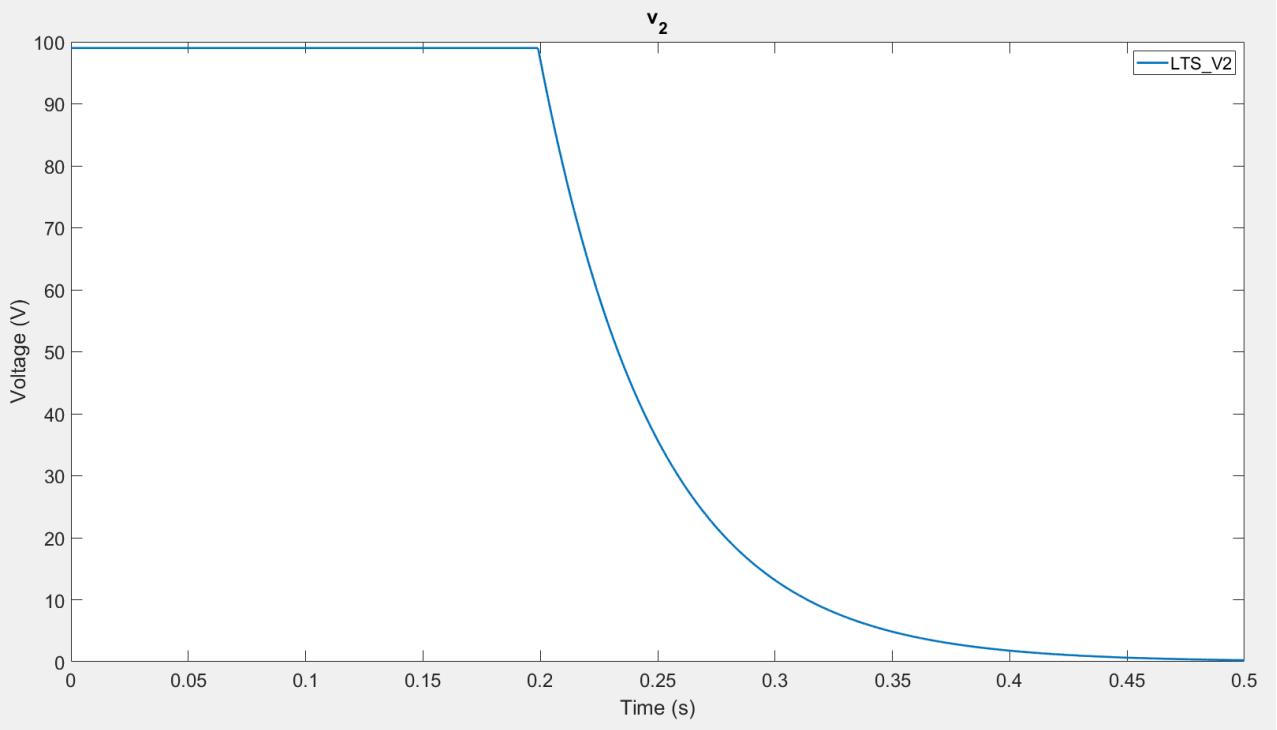


Figure 29. Result for Figure 27 when an operating point calculation is performed

The difference between the two results (Figure 28 and Figure 29) can be clearly observed: The DC operating point works as desired.

4. ALGORITHMS (LTS_V2'S CORE)

i. Reading input file

LTS_V2 attempts to stay as faithful to SPICE format as possible: It can support most SPICE directives[4]. The software reads a text file, containing a netlist, from the standard input stream. It considers four basic rules while analysing the input line by line:

- The first line is the title of the circuit
- If the line begins with a "*", it is a comment
- If the line begins with a ".", it is a command
- Otherwise the line indicates a component

After reading the first line, an empty circuit class with the specified title is created. After the first line, any comment line is ignored.

Linear components syntax:

<Component type><Component name> <node 1> <node 2> <component value>

Example: V1 N001 N002 SINE(0 1 10)

Non-linear components syntax:

<Component type><Component name> <node 1> <node 2> … <node n> <model name>

Where model names are optional.

Example: Q1 N002 N001 0 0 NPN

Commands start with a "." and can be MODEL, TRANS, OPTIONS, or END. They assist with defining model types, running transience analysis, specifying analysis options, and ending the netlist file.

See Appendix H for more detail regarding the netlist syntax and input implementations.

[4] As specified by [13], [14]

ii. Generating circuit and components objects

A. CIRCUIT

The simulation starts by initializing a circuit object. This object is responsible for storing components and computing the **x** vector, that contains the nodal voltages and voltage source currents. It also contains the matrices A and **b** that are required to compute **x**: $A\boldsymbol{x} = \boldsymbol{b}$.

Additionally, methods to generate the above matrices exist. Finally, there are vectors containing references to components, grouping components that have something in common. These are used to optimize efficiency: not every method needs to be called on every component. For example, methods that update AC sources have no effect when called on a resistor.

B. COMPONENT

The component class is an abstract class that contains general attributes and implements methods that are the same for every component. It provides the basis for component polymorphism.

Most methods defined in the component class are solely meant to be overloaded and throw errors when called on a child class that does not implement them. Not every child must implement every method due to the existence of the different component vectors in the circuit class. An example would be methods that return the current produced by a component, for insertion into the **b** vector. Furthermore, all components must implement a constructor.

iii. Selecting the simulation type and the analysis type

After the circuit's creation, a simulation type is determined from the netlist. Currently only transient analysis is supported. Dependent on the existing components, a type of analysis is selected: Linear or non-linear analysis. The two different types exist as this improves performance for linear circuits: computationally intensive steps that are necessary for non-linear circuits can be skipped.

iv. Calculate DC operating point

Any simulation starts by calculating the DC operating point, which is used to set the initial circuit values. This is necessary, as by default all components start initialized at zero voltage/current. Hence,

components that depend on previous values act as if all sources started at zero, rather than their initial values, leading to an incorrect output.

v. Solving the transient analysis step at the current time

Depending on the circuit's linearity, A and **b** are constructed appropriately, to solve $A\boldsymbol{x} = \boldsymbol{b}$. **x** is used to update components, such as capacitors, for the next iteration.

For linear circuits, **x** represents the final values for that timestep. For non-linear circuits, **x** is compared to the previous **x**: Newton-Raphson is continued until the difference between these two is less than a specified threshold.

vi. Output current time results

A CSV output file is generated at the start of the analysis. It contains one header row, and numerous data rows. The first column represents time while the other columns represent voltages and currents.

The nodal voltages are taken directly from **x** and have the form v_xxx, where xxx represents the node number. The same holds for voltage source currents that have the form i_xxx, where xxx represents the component's name.

Other component currents are more complex: a component's method is called that returns a comma-separated string of its terminal currents. For two-terminal devices this is a single value, whereas for multi-terminal devices it contains multiple values. For example, the MOSFET's output format would be "id_M1,ig_M1,is_M1".

For a linear transient analysis, these results are written to the CSV file as they are calculated, which minimises memory usage. As linear interpolation of results is used in non-linear analysis, results must be stored and processed before outputting. At most, two sets of results are stored at any given time, and the interpolation is output whenever the two sets are correctly positioned in time. The memory usage does not increase significantly, but the time taken does.

vii. Optimisations

A. GMIN

Diodes, MOSFETS, and BJTs all have PN-junctions. A problem arises when the voltage across these junctions becomes zero. This is demonstrated using the example from Figure 49.

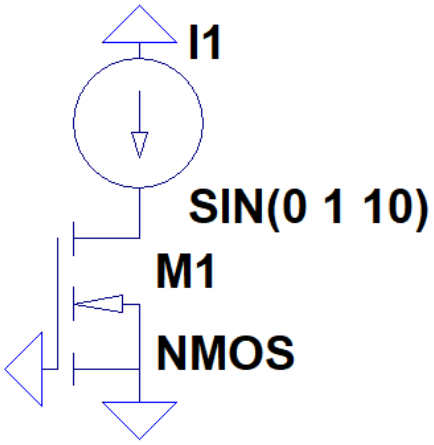


Figure 49. PN-Junction problem example

The current source starts with a value of zero amps, which results in an operating point with all nodes at 0V and hence a PN-junction with zero potential difference.

This is easy to observe, but in practice the transience analysis still must use an algorithm. As part of this algorithm a conductance matrix is formed. Looking at the MOSFET, when the voltage difference between any pair of nodes is zero, the conductance between them is also zero.

This results in the conductance matrix only containing a zero. Its determinant is zero and hence no inverse can be computed. The determinant is zero whenever an entire row of A is zero. This can happen when a MOSFET and a current source are the only two components connected to a node and all MOSFET voltages are equal.

The method used to avoid this issue is known as GMIN [5]. GMIN models reality by setting a minimum conductance between two nodes. With GMIN, the conductance of a PN-junction never reaches zero. Hence, an entire row will never become zero.

GMIN can be specified using .OPTIONS.

B. SOURCE STEPPING (OPERATING POINT FOR NON-LINEAR CIRCUITS)

Computing the DC operating point for non-linear circuits is more complex than for linear circuits. This is the case, as the Newton-Raphson method tends to diverge when the initial guess is too far away from the solution, as demonstrated in Figure 50.

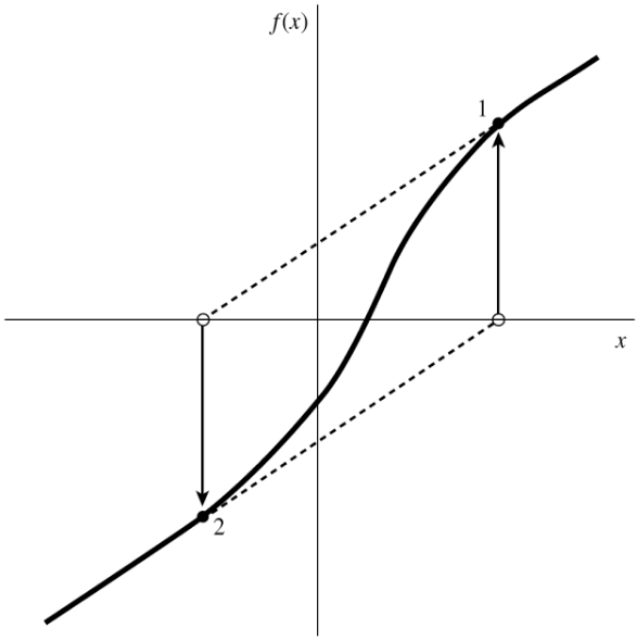


Figure 50. Newton-Raphson divergence caused by the initial guess being too far away from the solution. Taken from [11].

The same problem with Newton-Raphson divergence may occur when there are no inductors/capacitors: the initial circuit values forced by current/voltage sources are too far away from a stable operating point of the non-linear component and hence Newton-Raphson diverges on the first time-step.

There are multiple methods that aim to solve this Newton-Raphson divergence problem such as Source Stepping and GMIN Stepping [5]. GMIN Stepping works by inserting a resistor in parallel with every PN-junction in the circuit. The conductance of this resistor is decreased every time that Newton-Raphson converges, until the resistor has zero conductance and hence represents an open circuit.

Source Stepping works by starting the source values at zero and increasing them every time that Newton-Raphson converges until they reach their full values. For both methods, the last converging solution is used as the next Newton-Raphson guess with the decreased GMIN or increased source values. Both methods work by bringing the circuit's solution closer to the Newton-Raphson guess and then slowly bringing the circuit's solution back to the real solution.

Source Stepping was chosen as the DC operating point convergence helper. The main reason for this is the ease of implementation compared to GMIN Stepping: only the source values need to be adjusted. Moreover, a circuit generally contains fewer sources than PN-junctions and hence Source Stepping might be slightly more efficient as fewer values need to be changed for every successive set of Newton-Raphson iterations.

The implemented Source Stepping algorithm works by first replacing all capacitors with open circuits, all inductors with short-circuits and multiplying all source values by a factor alpha. Alpha must be between zero and one. It starts at one and checks for convergence. If the circuit does not converge, alpha is halved. This continues until the first convergence is found.

From this point, alpha is increased by a step. If the circuit converges again, alpha is increased by the same step and so on until alpha becomes one. If the circuit does not converge again, alpha is reset to the last converging alpha and the step is halved. If the step becomes smaller than a minimum value, the program terminates with an error message. If alpha reaches one, the DC operating point has been determined and the transient simulation starts, using the DC operating point values as the initial circuit values.

Figure 51 shows an example circuit that does converge with source stepping but diverges without.

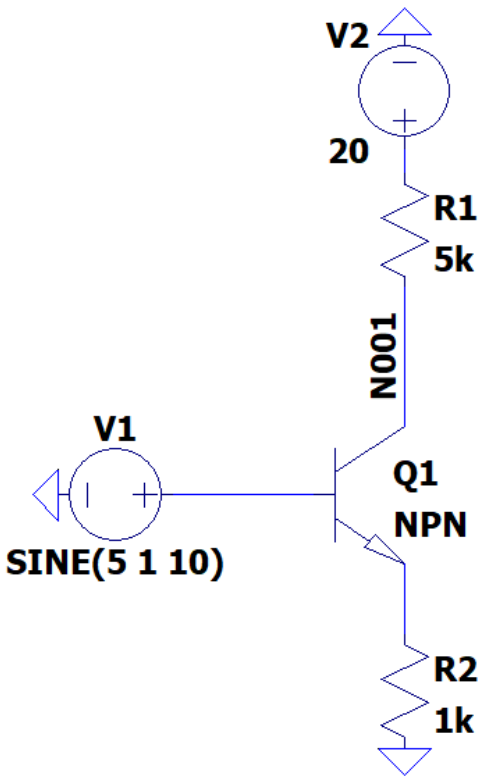


Figure 51. Circuit demonstrating the need for source stepping

When no source stepping is used, Newton-Raphson fails to converge on the first timestep. This is indicated by an error message stating that the maximum number of Newton-Raphson iterations has been exceeded. This is the expected behaviour, as the initial value of the BJT's base is 5V, which is far from the initial guess of 0V.

When source stepping is used, the output shown in Figure 52 is produced:

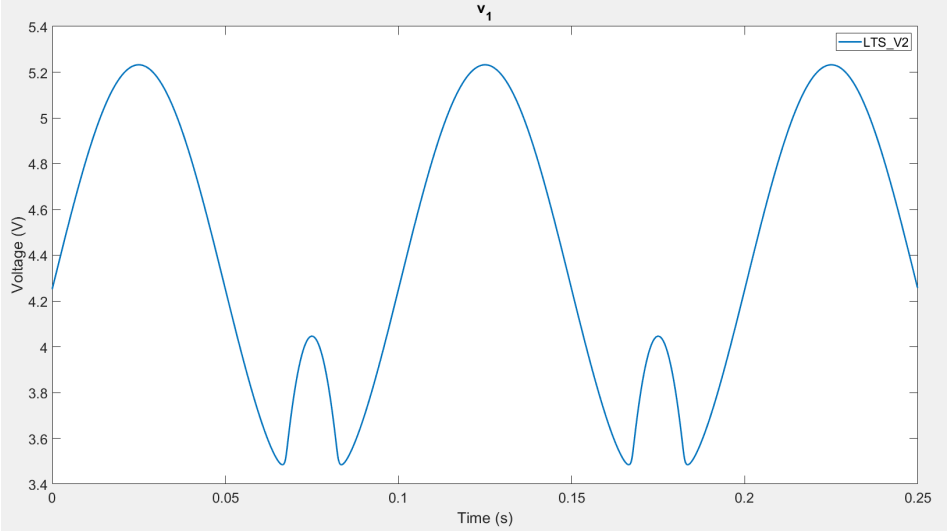


Figure 52. Voltage at node N001 in Figure 51

The initial voltage at node one is also far from the initial guess of zero volts. This indicates that the circuit is unlikely to converge without source stepping or an equivalent algorithm.

Error! Reference source not found. shows how the source stepping algorithm arrived at its solution:

alpha: 1 step: 0.05	alpha: 0.175 step: 0.05	alpha: 0.9 step: 0.025
alpha: 0.5 step: 0.05	alpha: 0.15 step: 0.025	alpha: 0.925 step: 0.025
alpha: 0.25 step: 0.05	alpha: 0.175 step: 0.025	alpha: 0.95 step: 0.025
alpha: 0.125 step: 0.05	alpha: 0.2 step: 0.025	alpha: 0.975 step: 0.025
alpha: 0.175 step: 0.05	alpha: 0.225 step: 0.025	alpha: 1 step: 0.025

The first column shows how alpha is decreased at the beginning while the step is not used. It also shows how alpha is increased again after the first converging alpha has been found (0.125). The second column shows how the step size is reduced and alpha is reset to the last converging alpha, when the next convergence cannot be found. The third column shows how the algorithm continues until alpha is one and converges: The circuit's DC operating point has been found.

Figure 53. Sample of the alpha/step values for the circuit from Figure 51

C. DYNAMIC TIMESTEP

Generally, a small timestep is preferred due to its accuracy: any function is better approximated by many points close together than by points spaced out. Good examples of this in LTS_V2 are sinusoidal sources and transients from capacitors/inductors. However, it wastes time to use a constant small timestep for the whole simulation, particularly if nodal voltages or branch currents differ only slightly between iterations: a larger timestep would be faster (due to fewer calculations) and have similar accuracy. However, using a large timestep for the whole simulation would lose substantial accuracy, and could entirely ignore very rapid fluctuations in source voltages e.g. a voltage spike. A compromise is a dynamic timestep. This is when the difference in time between two consecutive iterations changes throughout the course of the simulation, depending on how the circuit voltages and currents change. Whenever there is only a small difference in the **x** matrix and no sudden fluctuation is expected to have occurred between timesteps, the timestep is doubled. If there is a large change in the **x** matrix or the Newton-Raphson method requires too many iterations to converge, the last calculated values are discarded, and that iteration is repeated with a timestep eight times smaller. This ensures high resolution for rapid changes in voltage/current. The difference in values between two consecutive **x** matrices that causes a change in timestep is set to a constant times ABSTOL. A dynamic timestep balances speed and accuracy by doing fewer calculations when the circuit values change slightly and doing many whenever there is a large jump in values, as demonstrated in Figure 54.

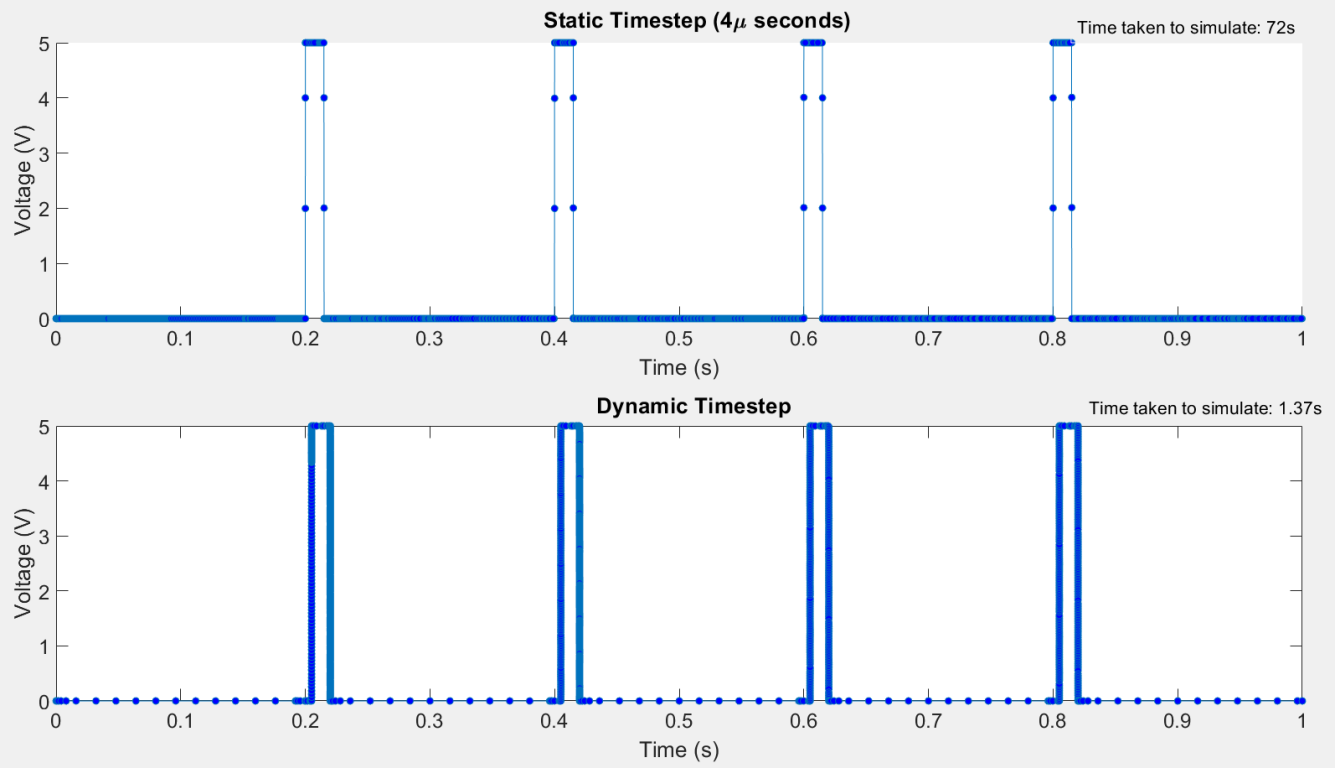


Figure 54. Static vs dynamic timestep

Figure 54 also shows how the dynamic timestep increases both speed and accuracy. The simulation time is decreased massively from 72s to 1.37s.

Implementing dynamic time-stepping is complicated by capacitors and inductors, which both use the timestep in their internal parameters. The program iterates through a vector of capacitors/inductors and updates the values of each associated conductance. This means that A^{-1} must be recalculated several times throughout a simulation. Therefore, the dynamic timestep was only implemented for non-linear circuits, where the inverse must be calculated at every iteration anyway.

At this stage, each valid result could be outputted. However, this would not fulfil the requirement of the SPICE .tran statement, which specifies a fixed time interval between printed results, as opposed to a dynamic one. Therefore, linear interpolation of results was implemented to satisfy this condition.

D. ERROR HANDLING

Errors can occur throughout the simulation. It was attempted to identify all these situations and circumvent them if possible. An example is the addition of source stepping: it was discovered that non-linear circuits did not converge at t=0, if they included sources with large values.

Errors can arise either due to software limitations (processing errors) or user errors (input errors). On detection of such an error, the program terminates with an error message.

Input

Any violation of the strict netlist format causes an error, that results in the program being terminated with an error message. Examples include using non-supported units and not including the .END command.

The error message is dependent on the error type. For example, the message for a non-supported statement has the form shown in Figure 55:

```
}else{
    cerr << "Unsuported netlist statement. Statement: " << compTypeC <<endl;
    exit(1);
}
```

Figure 55. Example error

compTypeC is the statement. This explains both the problem that occurred, and what exactly caused it. For example, if “N” is used instead of “M” to indicate MOSFET, the message makes identifying this trivial.

Processing

Processing errors are errors which happen during the simulation, such as Newton-Raphson divergence. These errors are more difficult to fix than input errors as they are associated with fundamental limitations of the application.

It was attempted to identify every possible processing error and return a unique error message, that informs the user about the exact problem and where it happened.

5. TESTING LTS_V2

i. Speed Scalability

To quantify LTS_V2's speed, a test script was set up which systematically creates increasingly complex circuits, to test how simulation time scales with increasing circuit size.

A sinusoidal input voltage was connected to x number of a components in series. This was repeated for various components.

The sinusoid was chosen, as a DC input would simply result in large dynamic time steps for non-linear components, which would improve speed, but would not be a fair comparison with linear circuits. Moreover, sinusoids are more demanding than DC inputs and hence test LTS_V2 more rigorously.

Device name	DESKTOP-C09D4BK
Processor	Intel(R) Core(TM) i7-8650U CPU @ 1.90GHz 2.11 GHz
Installed RAM	8.00 GB
Device ID	AE3E3E4D-0694-4305-8222-3D53540E0AC5
Product ID	00330-63022-24431-AAOEM
System type	64-bit operating system, x64-based processor

All simulations were run on the same computer with specs shown in Figure 56, using the Windows Linux Subshell to run the program.

Figure 56 Computer Specs

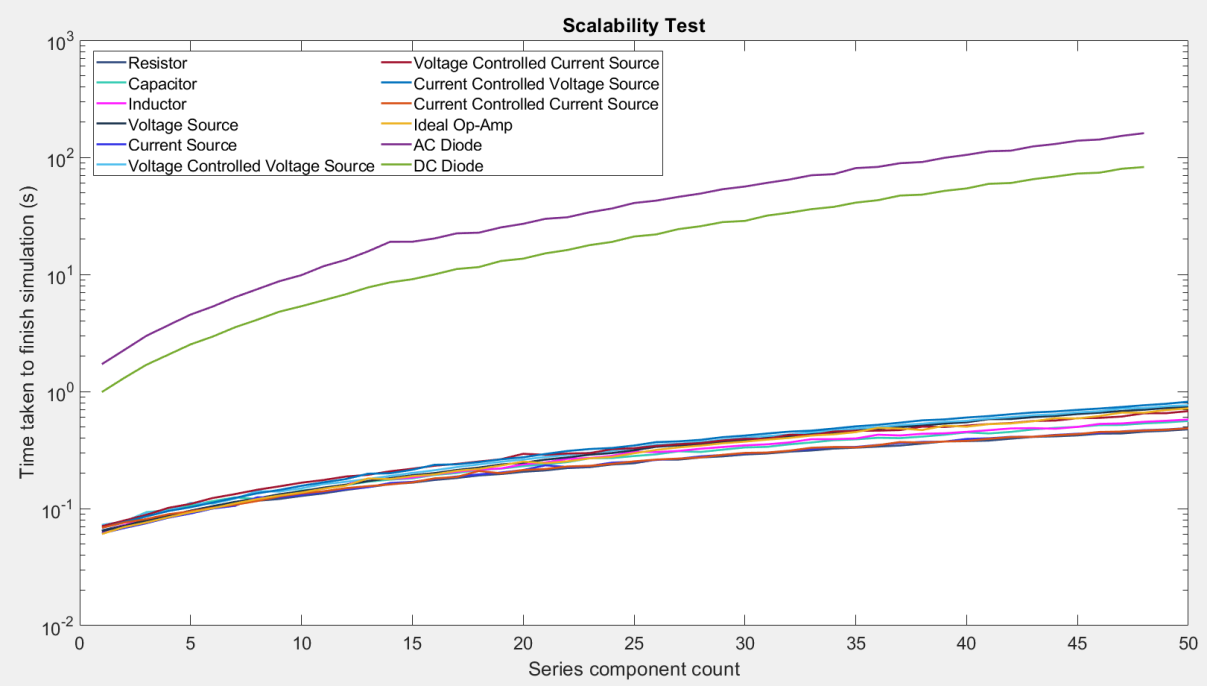


Figure 57. Scalability log plot

Figure 57 shows that non-linear components take longer to simulate than linear ones, and scale slightly worse. This is expected as non-linear components require more calculations, including the computation of multiple inverses.

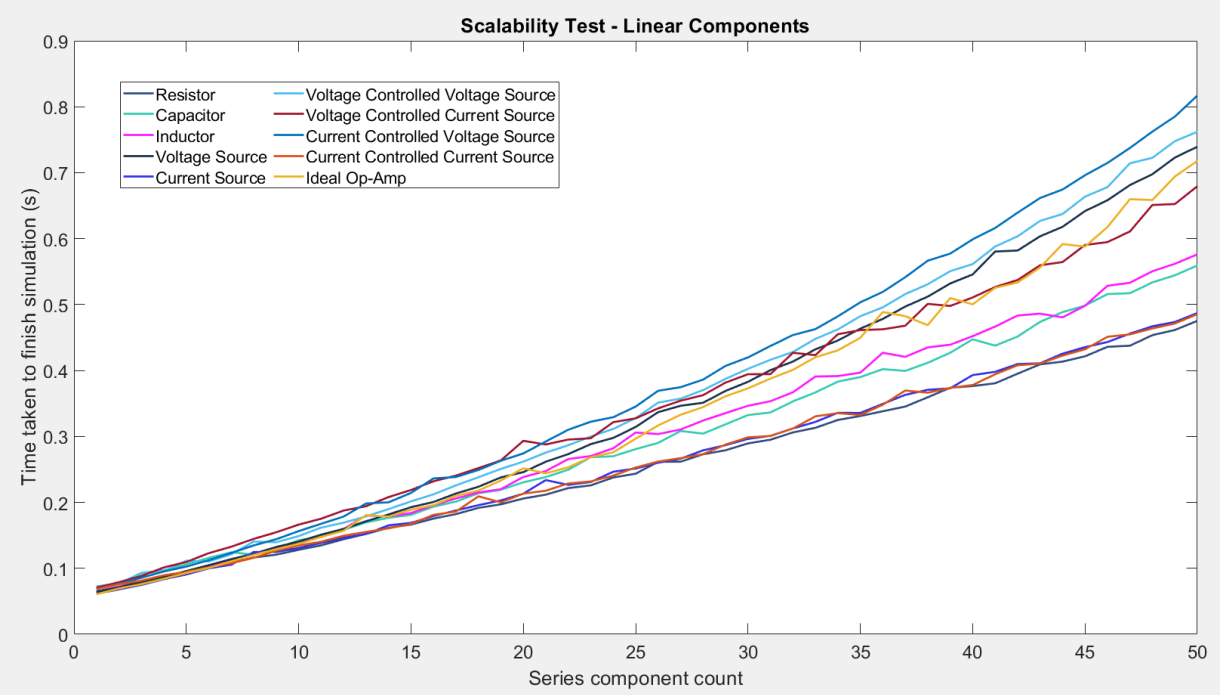


Figure 58. Scalability, linear circuits

It is noticeable that voltage sources (Normal, op-amp, dependent) scale worse compared to other linear components. This is expected, as every voltage source increases the matrix dimensions by one due to MNA. However, this is not a problem, as circuits generally use few voltage sources.

Voltage controlled current sources scale worse than expected. They only slightly add to the conductance matrix and do not change its dimensions. Hence, they were expected to be relatively fast.

Finally, current sources, current controlled current sources, and resistors are the quickest of the components. This is expected, as they require the least amount of matrix modifications.

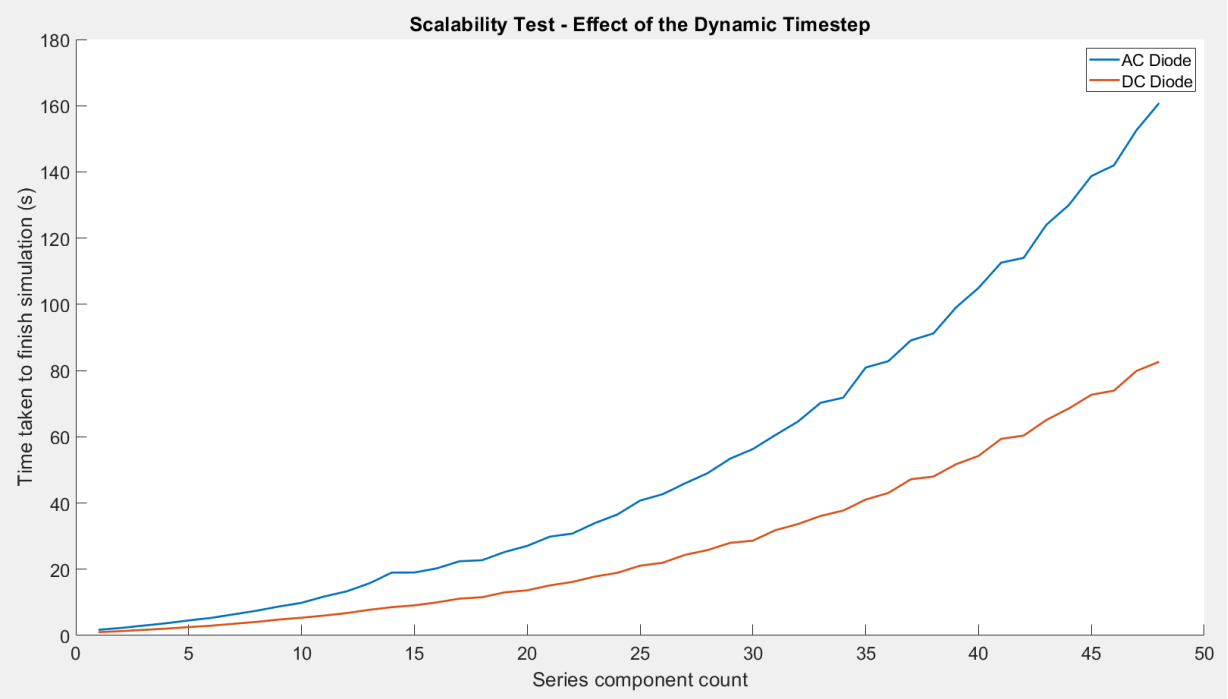


Figure 59. Scalability, diode comparison

Figure 59 demonstrates how the dynamic timestep increases speed: DC diodes scale significantly better than AC ones. DC inputs produce no changes in circuit values and hence the dynamic timestep maxes out while it stays smaller for AC inputs.

Finally, it is generally observed that LTS_V2 is significantly slower than LTspice: LTspice uses advanced optimisation methods that are not implemented in LTS_V2 and does not interpolate results, even when specified in the .tran statement.

ii. Energy Efficiency

Energy efficiency was tested using a similar circuit construction method to the scalability testing. Linear circuits were tested using resistors and non-linear ones using diodes.

The third-party software powertop was used to estimate a simulation's energy consumption.

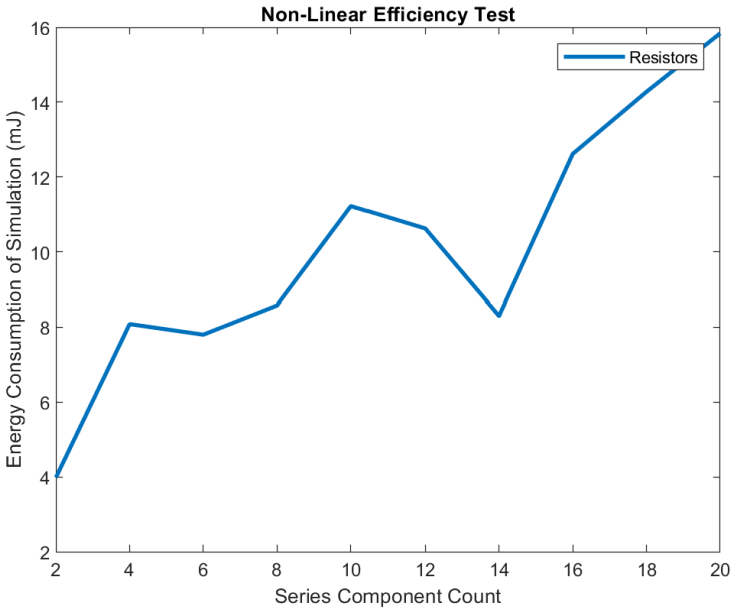


Figure 60. Linear energy consumption

Energy usage seems chaotic for linear circuits (Figure 60). This is likely due to the expected increase in energy consumption being small compared to changes that could be caused by random spikes: the resolution of the software is too small for such low energies.

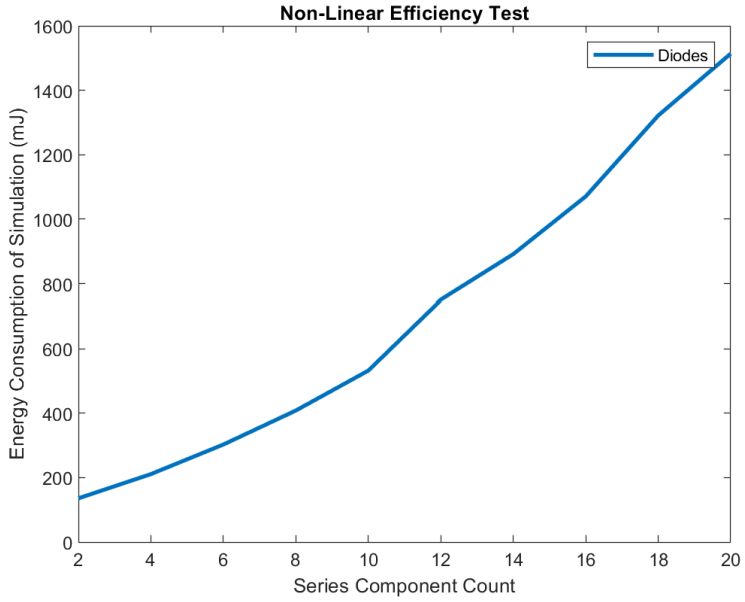


Figure 61. Non-Linear energy consumption

Non-linear circuits have a higher energy drain than linear circuits (**Figure 61**).

Furthermore, the power consumption of both circuits is compared in Figure 62, to examine the effect of time.

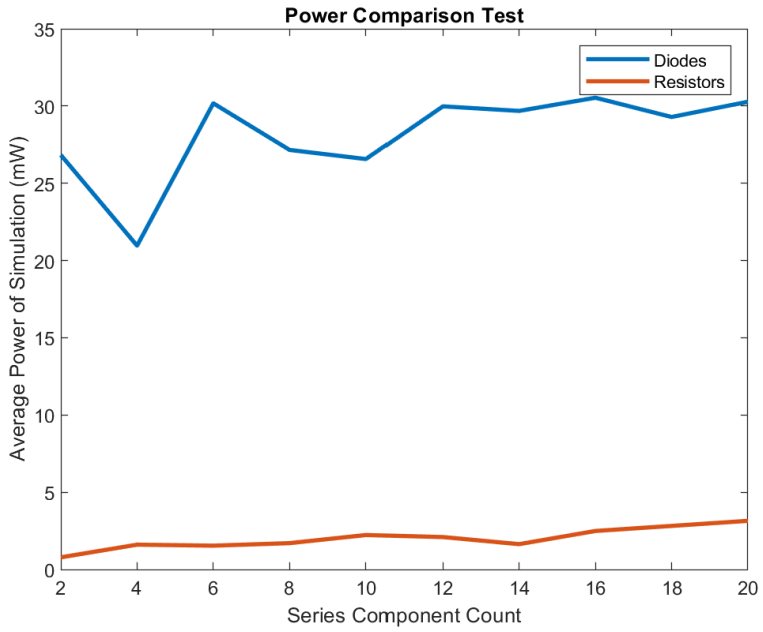


Figure 62. Power comparison

Power consumption is affected less by component count than energy consumption. This suggests that the increasing energy consumption mainly comes from an increased simulation time rather than an increased power consumption.

Like energy consumption, power consumption is affected significantly by the analysis method. This is expected as many more calculations are required for the non-linear analysis than for the linear one. For example, the non-linear method requires multiple matrix inverse calculations that tend to be resource intensive and hence contribute heavily to power consumption.

Throughout the development of the non-linear analysis algorithm, the amount of required calculations was minimized. Optimisations like a dynamic timestep reduce unnecessary calculations, improving energy efficiency.

iii. Final test circuits

Two test circuits were chosen that demonstrate every component type implemented and how they interact.

A. LINEAR CIRCUIT

Figure 63 shows a circuit, containing all linear component types.

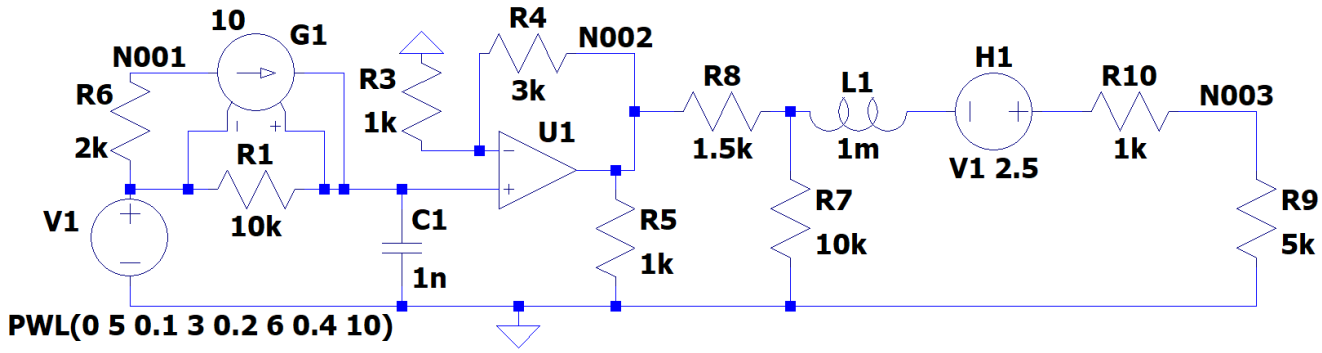


Figure 63. Linear circuit

A sample of results are plotted on Figure 64, Figure 65 and Figure 66. LTS_V2 used a static timestep of 10µs.

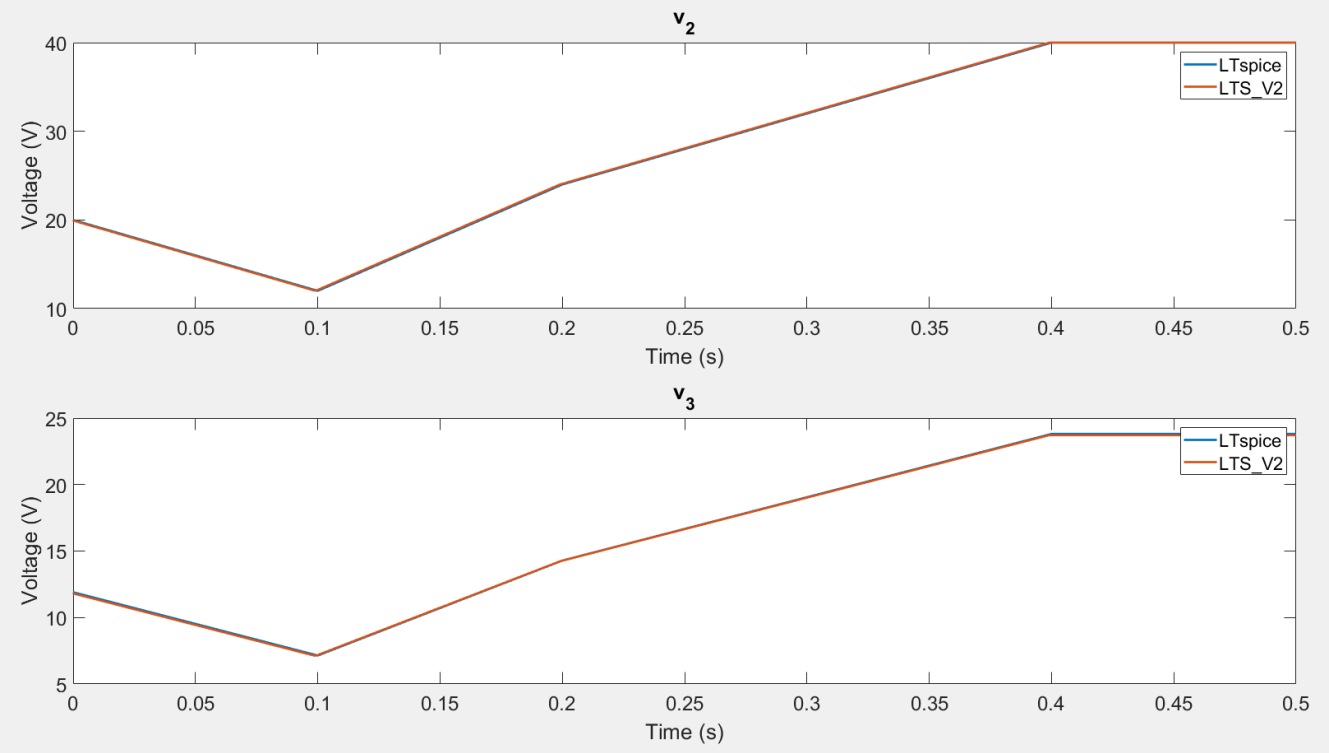


Figure 64. Figure 63, selected voltages

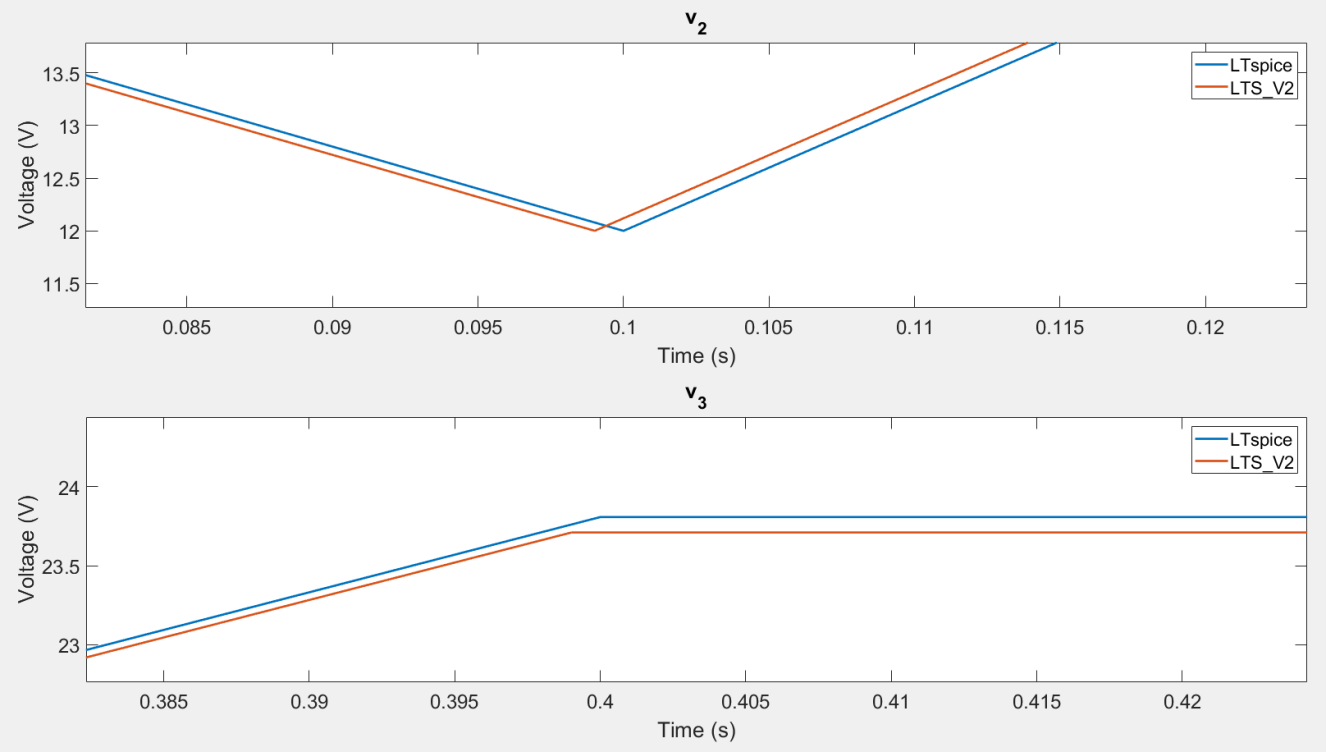


Figure 65. Figure 64, zoomed-in

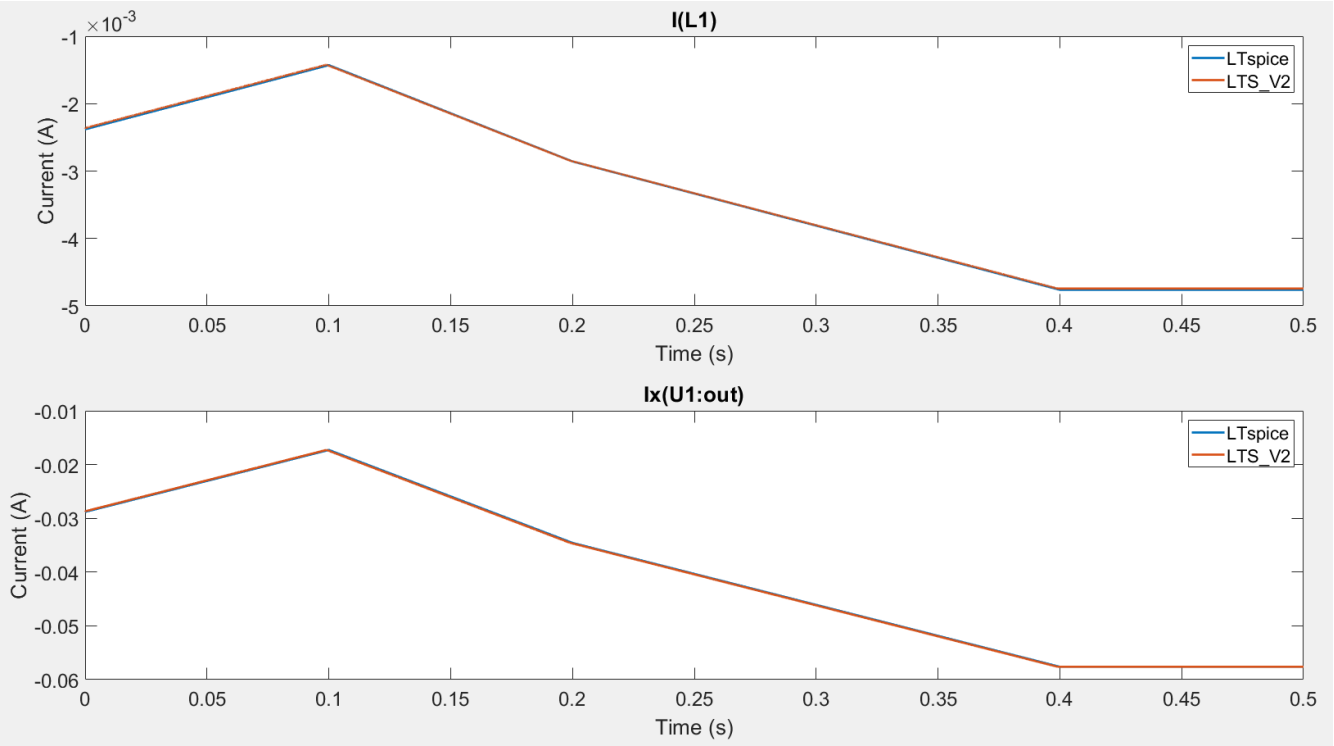


Figure 66. Figure 63, selected currents

All results are almost identical to the ones produced by LTspice, but small deviations can be observed. Figure 65 highlights these deviations: LTspice's and LTS_V2's waveforms change directions after each other and one is always slightly above the other. The deviations' magnitudes suggest that these errors might come from rounding or the timestep used.

B. NON-LINEAR CIRCUIT

Figure 67 shows a non-linear circuit consisting of three cascaded stages. LTS_V2 used a dynamic timestep and a printing-interval of 0.1ms.

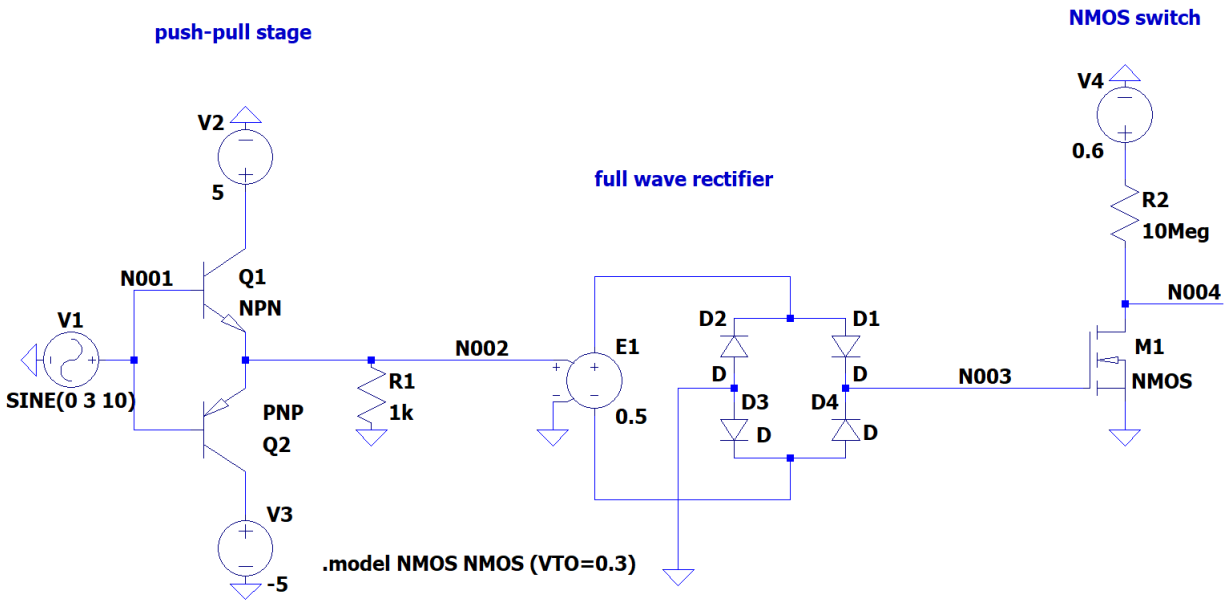


Figure 67. Non-linear circuit

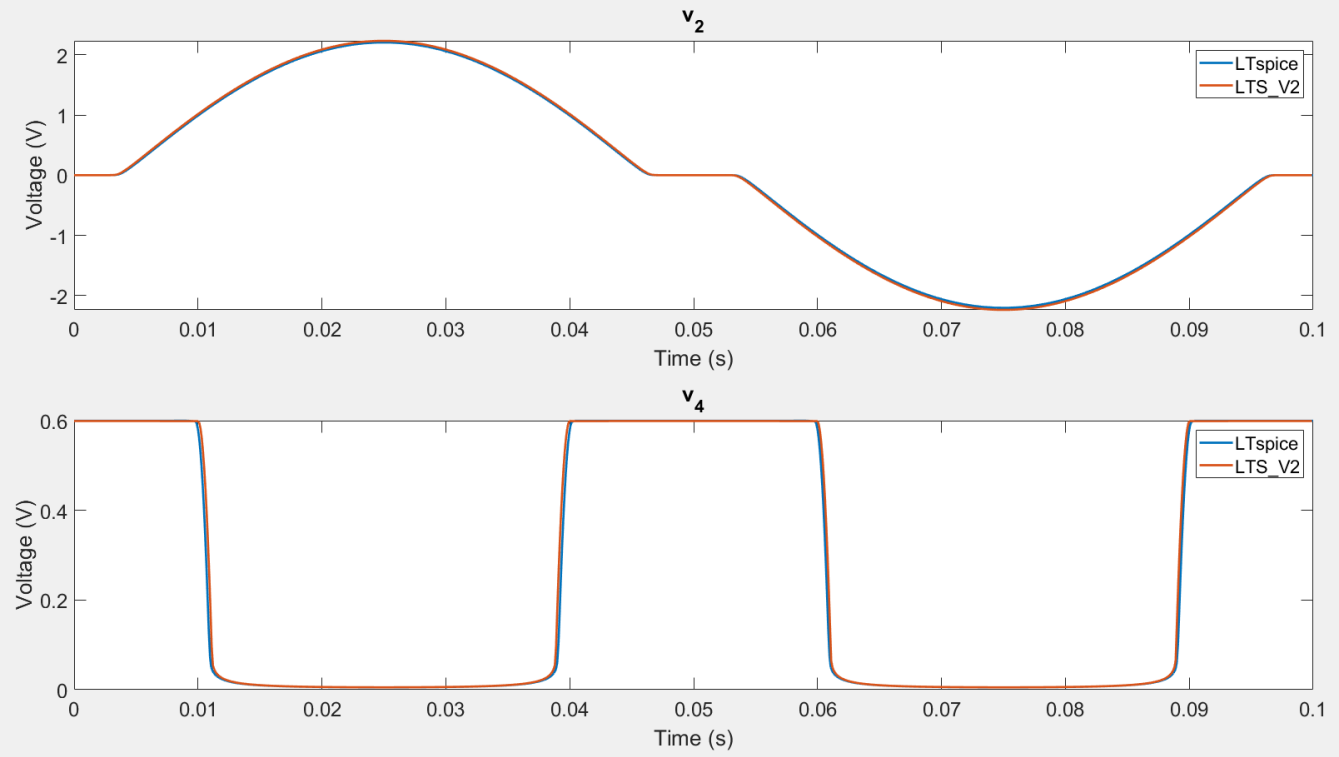


Figure 68. Figure 67, selected voltages

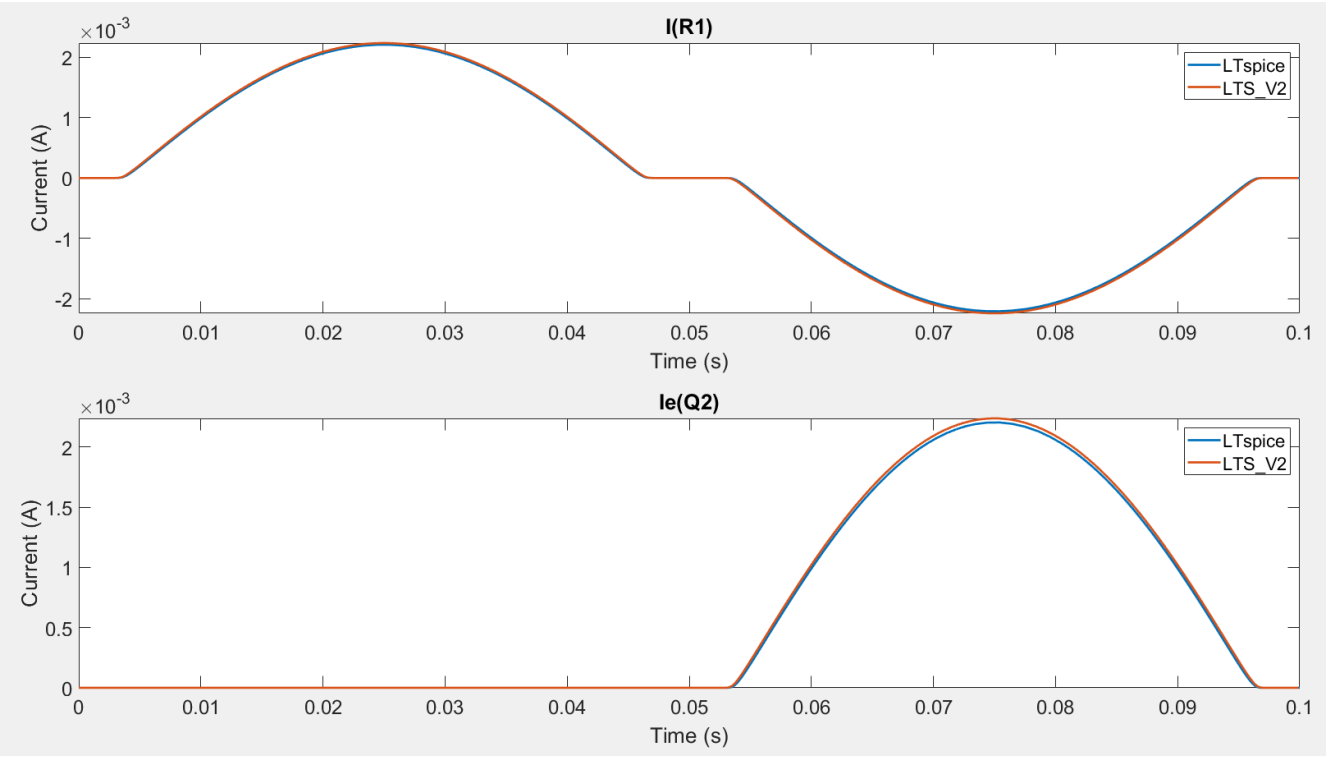


Figure 69. Figure 67, selected currents

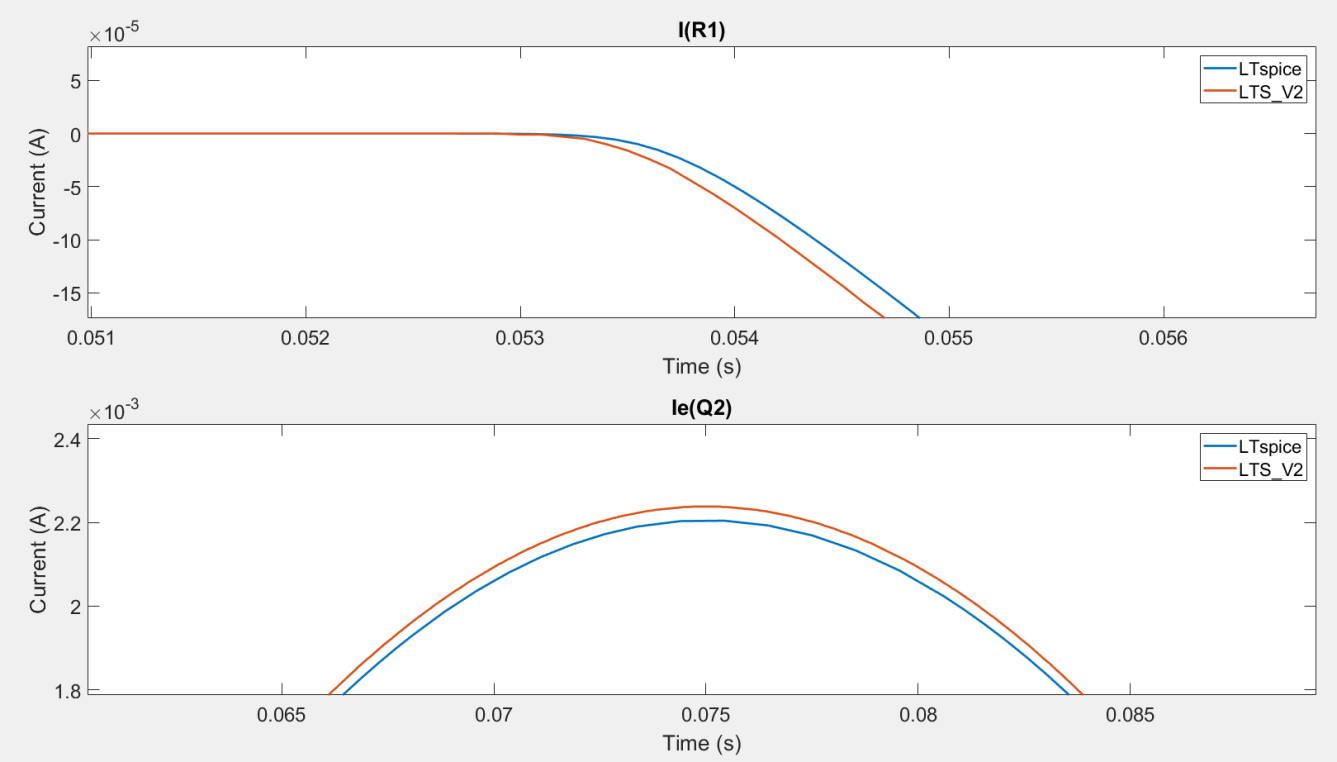


Figure 70. Figure 69, zoomed-in

Like for the linear circuit, the results are almost identical to LTspice's. Moreover, the deviations are of the same nature. However, the BJT current has a slightly bigger deviation: Exact to four decimal places. These slightly bigger deviations for the non-linear circuit are likely caused by the printing-interval used (also four decimal places). It would not be expected to achieve a greater accuracy than the printing-interval.

C. TIMING

Although LTspice and LTS_V2 have very similar accuracy, they differ significantly in terms of speed. To demonstrate this, five tests were run for the circuits in Figure 63 and Figure 67 to compare LTS_V2's speed with LTspice's.

For the linear circuit, LTS_V2 took 33.8ms on average, whereas LTspice took 14.3ms. LTS_V2 is less than half as quick, but very close in absolute terms.

LTS_V2 was found to be much slower than LTspice for the non-linear circuit. LTspice took 12.0ms on average, whereas LTS_V2 took 3.47s, which makes LTS_V2 approximately 300 times slower. Although LTS_V2 has many fewer advanced features to maximise efficiency, the main reason for LTS_V2's relative slowness in the non-linear analysis is interpolation: The algorithm used is not very optimised.

6. REFLECTION

Having gone through the process of describing what was achieved and how, it is good to reflect upon the process.

Teamwork was the greatest strength of LTS_V2, everyone was constantly providing ideas, constructive criticism and aid to other members where needed. This accelerated the project development significantly.

The agile working methodology, although instrumental in allowing the project to grow as it did, had some drawbacks. For example, there are several versions of the update method, to update components, that are used differently in linear and non-linear analysis. This occurred due to a more flexible update method being required for the latter but updating the first being too monumental of a task to be worthwhile.

Using MNA did provide many benefits, but, as voltage sources were treated differently from the beginning, full component polymorphism was not possible. Special functions that did not apply to other components were included in the component class to allow for voltage sources to work properly.

Furthermore, many additional features/improvements were researched but could not be implemented due to time constraints. These would be useful extensions for LTS_V2.

i. Sub-Circuits

Sub-circuits are integral when including large models into a simulation. They allow a project to be segmented into chunks, which work individually, but are then put together to form a complex circuit. An example is a non-ideal op-amp.

ii. More components

Currently only basic components are supported. However, modularity makes it straightforward to add additional components. Such components include switches and non-ideal op-amps.

iii. Graphical user interface (GUI)

Nowadays, no one likes typing netlists. For user friendliness, adding a GUI to design circuits is the most important extension. Manually typed netlists are hard to visualize and modify. Without this, consumers are likely to never use LTS_V2 at all.

A well-designed GUI might also be an important selling point: Many existing simulators have outdated GUIs that are not appealing to the user.

iv. More complex models

Non-linear and charge storing components are approximated using a model. These are mainly ideal or have trivial non-ideal traits. For example, capacitors use first order integration methods and BJTs are not affected by the substrate.

7. CONCLUSION

To close this report, the original specifications must be revisited. Looking at the minimum requirements for the functional aspect of the program, linear circuits can be simulated. Furthermore, they are simulated in a rapid and accurate manner as demonstrated by testing.

Moving to a more advanced aspect of the simulation, the non-linear components are functional and return accurate results. The simulation speed is slower than for linear components, but still reaches good efficiency. Accuracy was harder to quantify due to the circuits' complexity (See section 5). Nevertheless, high accuracy is demonstrated.

On top of the minimum requirements, several extra features were added. The input system allows for near true SPICE-style netlists, instead of only simplified versions. Extra components have been added, including ideal op-amps. Additionally, advanced optimisations have been implemented to maximize the software's utility.

Throughout the process, every implementation was carefully considered, alternatives proposed, evaluated, and tested before deciding on a final solution. When at a later point it was discovered that a poor choice was made, the team worked backwards to fix and optimize for a better alternative. Everything was done with user-friendliness and scalability in mind. The code base is highly modular and polymorphic, allowing for easy expansion to meet future needs.

Overall, LTS_V2 meets and exceeds the required specifications and compares to LTspice in accuracy, but not in speed.

REFERENCES

[1] M. Ossowski, S. Kozlowska, M. Korzybski, A. Kuczyński and M. Tadeusiewicz, "The Newton-Raphson method," Institute of Electrical Engineering Systems - Lodz University of Technology, 18 December 2019. [Online]. Available: https://matel.p.lodz.pl/wee/i12zet/The%20Newton-Raphson%20method.pdf. [Accessed 15 May 2020].

[2] M. D. Smith, "Newton-Raphson Technique," 1 Oct 1998. [Online]. Available: https://web.mit.edu/10.001/Web/Course_Notes/NLAE/node6.html. [Accessed 10 May 2020].

[3] R. Wang, "Newton-Raphson Method (Univariate)," 16 February 2020. [Online]. Available: http://fourier.eng.hmc.edu/e176/lectures/ch2/node5.html. [Accessed 25 May 2020].

[4] . C.-W. Ho, A. Ruehli and P. Brennan, "The modified nodal approach to network analysis," *IEEE Transactions on Circuits and Systems,* vol. 22, no. 6, pp. 504-509, June 1975.

[5] R. Kielkowski, Inside Spice, 2nd ed., McGraw-Hill, 1998.

[6] R. Wang, "Newton-Raphson method (multivariate)," 12 Feb 2015. [Online]. Available: http://fourier.eng.hmc.edu/e176/lectures/NM/node21.html. [Accessed 10 May 2020].

[7] C. Thompson, "A study of numerical integration techniques for use in the companion circuit method of transient circuit analysis," ECE Technical Reports, West Lafayette, 1992.

[8] E. Cheever, "Advanced SCAM with dependent sources," Swarthmore College, 2019. [Online]. Available: https://lpsa.swarthmore.edu/Systems/Electrical/mna/MNA1.html. [Accessed 10 May 2020].

[9] A. L. Steven Herbst, "Companion Models for Basic Non-Linear and," 30 Dec 2008. [Online]. Available: http://dev.hypertriton.com/edacious/trunk/doc/lec.pdf. [Accessed 18 May 2020].

[10] Multism Live, "DC operating point," [Online]. Available: https://www.multisim.com/help/simulation/grapher/dc-operating-point/#:~:text=DC%20operating%20point,with%20no%20input%20signal%20applied.. [Accessed 1 June 2020].

[11] W. Press, S. A. Teukolsky, W. Vetterling and B. Flannery, Numerical Recipes: The Art of Scientific Computing, 3rd ed., New York: Cambridge University Press, 2007.

[12] E. Cheever, "Modified Nodal Analysis with Inductors, Capacitors and Op-Amps," Swarthmore College, 2019. [Online]. Available: https://lpsa.swarthmore.edu/Systems/Electrical/mna/MNA5.html. [Accessed 10 May 2020].

[13] “SPICE Quick Reference Sheet,” 2001. [Online]. Available: https://web.stanford.edu/class/ee133/handouts/general/spice_ref.pdf. [Accessed 15 May 2020].

[14] G. Rincón-Mora, “LTSPICE Manual,” [Online]. Available: http://rincon-mora.gatech.edu/classes/ltspice.pdf. [Accessed 23 May 2020].

[15] R. Nave, “Hyper Physics,” GSU, [Online]. Available: http://hyperphysics.phy-astr.gsu.edu/hbase/Solids/pnjun.html. [Accessed 7 June 2020].

[16] K. S. K. a. I. Clifford, “Achieving Accurate Results with a Circuit Simulator,” [Online]. Available: https://kenkundert.com/docs/eda+t93-paper.pdf. [Accessed 7 Jun 2020].

APPENDIX

Appendix A

TEST 1

Solvers tested for Eigen3:

- colPivHouseholderQr
- fullPivLu
- completeOrthogonalDecomposition
- Normal inverse method ($\boldsymbol{x} = A^{-1}\boldsymbol{b}$)
- LU decomposition (2 steps)

Screenshot of Eigen3 test program output:

```
Time taken by colPivHouseholderQr: 2172457 microseconds
Time taken by FullPivLU: 3192497 microseconds
Time taken by CompleteOrthogonalDecomposition: 1950554 microseconds
Time taken by inverse method: 33448 microseconds
Time taken by LU decomp: 68814 microseconds
Inverse method faster
```

Figure 71. Test results produced by the test program responsible for comparing Eigen3's different solvers for the linear analysis.

Solvers tested for Armadillo:

- LAPACK
- SuperLU
- Normal inverse method ($\boldsymbol{x} = A^{-1}\boldsymbol{b}$)
- LU decomposition (2 steps)

Screenshot of Armadillo's test program output:

```
Time taken by LAPACK: 33374854 microseconds
Time taken by SuperLU: 35005591 microseconds
Time taken by LU decomp: 497831 microseconds
Time taken by Inverse method: 834096 microseconds
LU decomp faster
```

Figure 72. Test results produced by the test program responsible for comparing Armadillo's different solvers for the linear analysis.

TEST 2

Screenshot of Armadillo VS Eigen3 test program output:

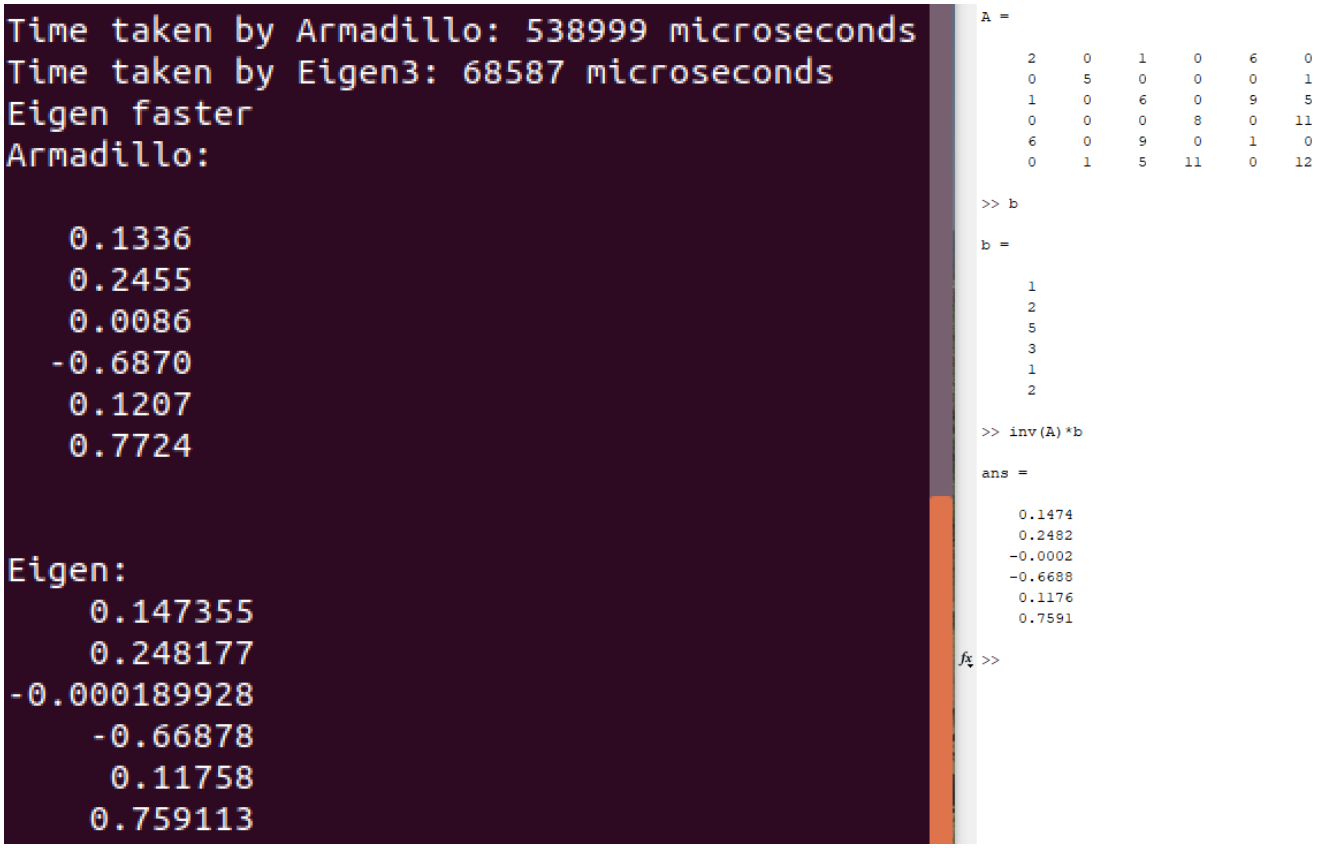


Figure 73. Test results produced by the test program responsible for comparing Armadillo with Eigen3. The right side shows a screenshot of MATLAB's results.

The results included on the right side of Figure 73 show the results produced by MATLAB for the same A and **b** that were used to produce the results on the left. These were used as a reference to assess the accuracy of Armadillo and Eigen3.

TEST 3

Screenshot of Eigen3's fastest non-linear solver test program:

```
Time taken by colPivHouseholderQr: 151360827 microseconds
Time taken by FullPivLU: 159573055 microseconds
Time taken by CompleteOrthogonalDecomposition: 149299489 microseconds
Time taken by inverse method: 121855884 microseconds
Time taken by LU decomp: 188314955 microseconds
Inverse method faster
```

Figure 74. Test results produced by the test program responsible for comparing Eigen3's different solvers for the non-linear analysis.

Appendix B

For the direct Newton Raphson there was a clear divergence problem. If we attempted to simulate a simple BJT amplifier, the result is shown in **Error! Reference source not found.**.

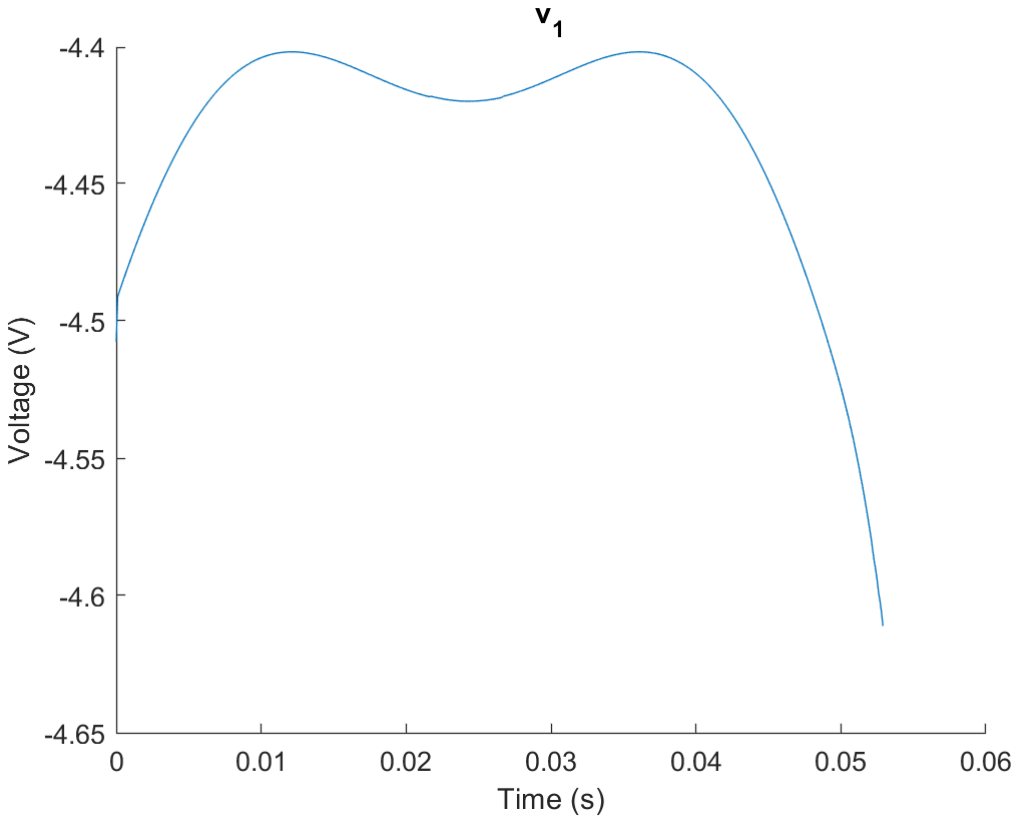


Figure 75. Diverging simulation

What was observed was that at around 0.05 seconds the number of Newton-Raphson iterations required to calculate the next iterations started increasing, and at the cut-off point it would not converge, going into well over 1000 iterations with no solution.

This problem was resolved by using indirect Newton-Raphson method which showed a greater range of convergence. After this, any extra divergence issues were solved with extra algorithms such as source stepping, dynamic time stepping and GMIN.

Appendix C

The equation for a capacitor is:

$$i = C\frac{dv}{dt} \qquad \textit{Eq. 48}$$

This rearranges to give the following equation when t = n+1.

$$Cv_{n+1} = \int_0^{n+1} i\ dt \qquad \textit{Eq. 49}$$

According to the trapezoidal method, $\int_0^{n+1} i\ dt = \int_0^n i\ dt + \frac{h}{2}(i_{n+1} + i_n)$, where *h* is the timestep.

$$Cv_{n+1} = \int_0^{n+1} i\ dt = \int_0^n i\ dt + \frac{h}{2}(i_{n+1} + i_n) = \int_0^n i\ dt + \frac{h}{2}i_{n+1} + \frac{h}{2}i_n \qquad \textit{Eq. 50}$$

Rearranging gives:

$$i_{n+1} = -\frac{2}{h}\int_0^n i\ dt - i_n + \frac{2C}{h}v_{n+1} = -\frac{2C}{h}v_n - i_n + \frac{2C}{h}v_{n+1} \qquad \textit{Eq. 51}$$

Appendix D

Given the diode Poisson equation:

$$I = Is * \left(e^{\frac{V}{N*VT}} - 1\right) \qquad \textit{Eq. 52}$$

Its derivative can be computed as:

$$\frac{dI}{dV} = \frac{Is}{N * VT} * e^{V/N*VT} \qquad \textit{Eq. 53}$$

Using the equations for linearizing non-linear components:

$$Ieq = (Ic(V_n) - V_n * g) \qquad \textit{Eq. 54}$$

$$g = \frac{dIc}{dV}(V_n) \qquad \textit{Eq. 55}$$

For a diode:

$$g = \frac{Is}{N * VT} * e^{V/N*VT} \sim \frac{Is}{N * VT} * (e^{V/N*VT} - 1) = \frac{I(V)}{N * VT} \quad \textit{Eq. 56}$$

$$Ieq = I(V) - V * \frac{I(V)}{N * VT} = I(V) * \left(1 - \frac{V}{N * VT}\right) \quad \textit{Eq. 57}$$

Appendix E

To generalize using the Newton-Raphson method, to linearize non-linear components, a few variables must be defined:

x_m represents the node m of the component.

I_{x_m} represents the current going into the component at node m.

$V_{x_a x_b}$ represents the voltage across the component between nodes a and b $(V_a - V_b)$

$g_{x_a x_b}$ represents the conductance between nodes a and b through the component.

$I_{x_a x_b}$ representing the current going through the component at node a, only considering that caused by the $V_{x_a x_b}$

A subscript of n represents the value of that variable at iteration n of the Newton Raphson method, following the same principle a subscript of $n + 1$ represents the value of that variable at iteration $n + 1$ of the Newton Raphson method.

Using this notation, it is realized that I_{x_m} is a function of $V_{x_a x_b}$ for every pair of nodes a and b.

Using the current equation previously derived:

$$I(V_{n+1}) = I(V_n) + (V_{n+1} - V_n) * \frac{dI}{dV}(V_n) \quad \textit{Eq. 58}$$

This can be adjusted to allow multi-variable current functions. Firstly $I(V)$ should only represent the current at node a based on the voltage difference $V_{x_a x_b}$ and should be replaced with $I_{x_a x_b}(V_{x_a x_b})$. Furthermore, the derivative $\frac{dI}{dV}(V_n)$ should now represent the partial derivative of $I_{x_a x_b}$ with respect to the voltage difference $V_{x_a x_b}$, $\frac{dI_{x_a x_b}}{d\, V_{x_a x_b}}(\, V_{x_a x_b,n})$. This leads to a current equation of the form:

$$I_{x_a x_b}(V_{x_a x_b,n+1}) = I_{x_a x_b}(V_{x_a x_b,n}) + (V_{x_a x_b,n+1} - V_{x_a x_b,n}) * \frac{dI_{x_a x_b}}{d\, V_{x_a x_b}}(\, V_{x_a x_b,n}) \quad \textit{Eq. 59}$$

Letting

$$g_{x_a x_b,n} = \frac{dI}{dV_{x_a x_b}}\left(V_{x_a x_b,n}\right) \quad \textit{Eq. 60}$$

$$I_{x_a x_b}\left(V_{x_a x_b,n+1}\right) = I_{x_a x_b}\left(V_{x_a x_b,n}\right) + \left(V_{x_a x_b,n+1} - V_{x_a x_b,n}\right) * g_{x_a x_b,n} \quad \textit{Eq. 61}$$

This equation gives the current at node a based on its relationship with node b. The total current is then the sum of all the currents at node a due to all other nodes.

$$I_{x_a,n+1} = \sum_{b=1,b\neq a}^{k} \left(I_{x_a x_b}\left(V_{x_a x_b,n}\right) + \left(V_{x_a x_b,n+1} - V_{x_a x_b,n}\right) * g_{x_a x_b,n}\right) \quad \textit{Eq. 62}$$

$$I_{x_a,n+1} = \sum_{b=1,b\neq a}^{k} \left(I_{x_a x_b}\left(V_{x_a x_b,n}\right)\right) + \sum_{b=1,b\neq a}^{k} \left(\left(V_{x_a x_b,n+1} - V_{x_a x_b,n}\right) * g_{x_a x_b,n}\right) \quad \textit{Eq. 63}$$

$$I_{x_a,n+1} = I_{x_a,n} + \sum_{b=1,b\neq a}^{k} \left(\left(V_{x_a x_b,n+1} - V_{x_a x_b,n}\right) * g_{x_a x_b,n}\right) \quad \textit{Eq. 64}$$

$$I_{x_a,n+1} = \left(I_{x_a,n} - \sum_{b=1,b\neq a}^{k} \left(V_{x_a x_b,n} * g_{x_a x_b,n}\right)\right) + \sum_{b=1,b\neq a}^{k} \left(V_{x_a x_b,n+1} * g_{x_a x_b,n}\right) \quad \textit{Eq. 65}$$

Where k is the total number of nodes.

This leads to the following equation for Ieq, for a node:

$$Ieq = I_{x_a,n} - \sum_{b=1,b\neq a}^{k} \left(V_{x_a x_b,n} * g_{x_a x_b,n}\right) \quad \textit{Eq. 66}$$

Appendix F

Starting with the P-Channel MOSFET-Current equations at each node and each operating point.

Table 5. P-Channel MOSFET currents

	Cut-Off VGS-VT < 0	Saturation 0 <= VGS-VT < VDS	Triode 0 <= VDS <= VGS-VT
Drain Current (ID)	0	$\frac{K}{2}(VGS-VT)^2\left(1+\frac{VDS-VGS+VT}{V_a}\right)$	$K\left((VGS-VT)-\frac{VDS}{2}\right)VDS$
Gate Current (IG)	0	0	0
Source Current (IS)	0	$\frac{K}{2}(VGS-VT)^2\left(1+\frac{VDS-VGS+VT}{V_a}\right)$	$K\left((VGS-VT)-\frac{VDS}{2}\right)VDS$

As the method of making a linear model, relies on only two nodes, a linear equivalent, considering each pair of nodes in the MOSFET, must be worked out: Drain-Gate (DG), Drain-Source (DS), Gate-Source (GS).

This is achieved using the current equation for multi-variable current functions, derived in Appendix E. See Appendix E for an explanation for the meaning of each symbol. Giving the conductance between two nodes as:

$$g_{x_a x_b,n} = \frac{dI}{dV_{x_a x_b}}\left(V_{x_a x_b,n}\right) \quad \textit{Eq. 67}$$

And the sum of the current sources connected to the node a is given by

$$Ieq_{x_a,n+1} = I_{x_a,n} - \sum_{b=1,b\neq a}^{k}\left(V_{x_a x_b,n} * g_{x_a x_b,n}\right) \quad \textit{Eq. 68}$$

Starting with cut-off mode it can be shown by inspection that both g and Ieq are zero for all nodes, as all currents are zero in that operating mode. Additionally, through the same logic it can be determined that all gate currents are zero, therefore g_{GD}, g_{GS} and Ieq_G are all equal to zero as well for all operating points.

That only leaves the DS pair. Looking at the drain current at saturation mode:

$$I_D = \frac{K}{2}(VGS - VT)^2 * \left(1 + \frac{VDS - VGS + VT}{V_a}\right) \sim \frac{K}{2}(VGS - VT)^2 \quad \textit{Eq. 69}$$

$$g_m = \frac{dI_D}{dV_{GS}} = K(VGS - VT) = \sqrt{2KI_D} \quad \textit{Eq. 70}$$

$$g_o = \frac{dI_D}{dV_{DS}} = \frac{K(VGS - VT)^2}{2V_a} \quad \textit{Eq. 71}$$

$$Ieq_{D,n+1} = I_{D,n} - g_m VGS_n - g_o VDS_n \quad \textit{Eq. 72}$$

Because the current equations for $I_E = I_D$ with current flowing in the opposite direction it is trivial to see that the values for g and Ieq are identical.

With these equations, before constructing a linearized model, the case of g_m must be considered first. It forms part of the conductance between the DS nodes, however the current across it is not dependant on the voltage VDS, but rather the voltage VGS. Hence, the effects of this conductance are modelled as a voltage dependant current source. The model can be constructed as shown in Figure 76.

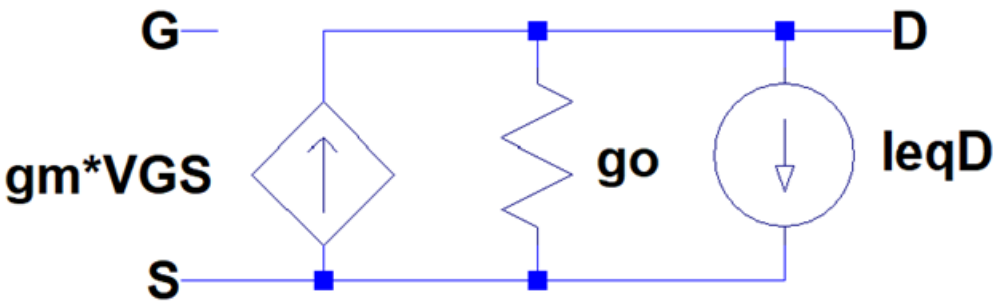


Figure 76. PMOS linearized circuit

Repeating this process for the triode region:

$$I_D = K\left((VGS - VT) - \frac{VDS}{2}\right)VDS = K\left((VGS - VT)VGS - \frac{VGS^2}{2}\right) \quad \textit{Eq. 73}$$

$$g_m = \frac{dI_D}{dV_{GS}} = K * VDS \quad \textit{Eq. 74}$$

$$g_o = \frac{dI_D}{dV_{DS}} = K\big((VGS - VT) - VGS\big) \quad \textit{Eq. 75}$$

$$Ieq_{D,n+1} = I_{D,n} - g_m VGS_n - g_o VDS_n \quad \textit{Eq. 76}$$

These formulae can be used to construct a table of values (Table 6) for each of the regions of operation for the P-Channel MOSFET.

Table 6. P-Channel MOSFET Linearized Parameters NMOS linearized parameters

	Cut-Off	Saturation	Triode
g_m	0	$\sqrt{2KI_D}$	$K(VDS)$
g_o	0	$\frac{K(VGS-VT)^2}{2V_a}$	$K\big((VGS - VT) - VGS\big)$
Ieq_D	0	$I_{D,n} - g_m VGS_n - g_o VDS_n$	$I_{D,n} - g_m VGS_n - g_o VDS_n$

This same method can be used to create the P-Channel MOSFET with the diagram in Figure 77 and parameters in Table 7.

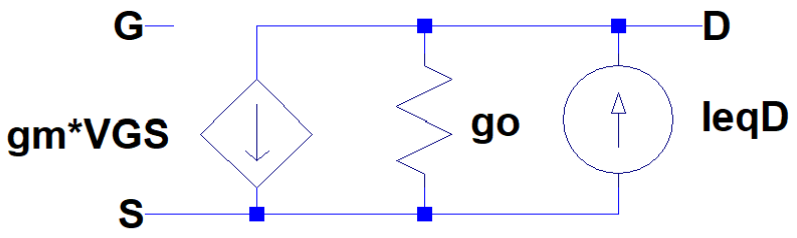


Figure 77. PMOS linearized circuit

Table 7. PMOS linearized parameters

	Cut-Off VGS – VT < 0	Saturation VGS-VT < 0 AND VDS+VGS+VT < 0	Triode VDS
g_m	0	$\sqrt{2KI_D}$	$K * VDS$
g_o	0	$\frac{K(VGS-VT)^2}{2V_a}$	$K\big((VGS-VT)-VGS\big)$
Ieq_D	0	$I_{D,n} - g_m VGS_n - g_o VDS_n$	$I_{D,n} - g_m VGS_n - g_o VDS_n$

Appendix G

The BJT is complex as every node has currents depending on the voltages at every other node. For this reason, instead of approaching this model directly as was done for the MOSFET, a simplified large signal equivalent model must be used first.

The Gummel-Poon Mid Band model [9] is one such model shown in Figure 78:

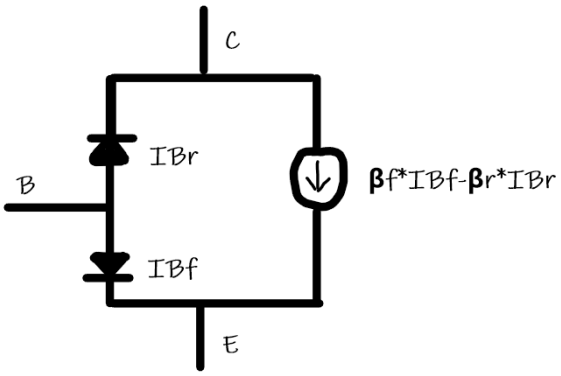


Figure 78. NPN large signal Gummel-Poon model

To linearize this, we could work out the parameters using the linearization method described in Appendix E. However, because we have a large signal model, we can linearize each of those components first, and then build put them together. Starting with the diodes, we already have the equations:

$$Ieq = (I(V_n) - V_n * g) \quad \textit{Eq. 77}$$

$$g = \frac{dI}{dV}(V_n) \quad \textit{Eq. 78}$$

Starting with the equations for the Gummel-Poon [9] model IBf has a current:

$$\boldsymbol{I_{bf} = \frac{I_{fs}}{\beta_f}\left(e^{(vBE/Vt)} - 1\right)} \quad \textbf{\textit{Eq. 79}}$$

$$\text{Ieq} = \left(I_{bf}(V_{BE,n}) - V_{BE,n} * g\right) \quad \textit{Eq. 80}$$

$$g_{pf} = \frac{I_{fs}}{\beta_f * VT} * e^{\frac{VBE}{VT}} \quad \textit{Eq. 81}$$

Repeating the process for the IBr current:

$$\boldsymbol{I_{bf} = \frac{I_{rs}}{\beta_r}\left(e^{(vBC/Vt)} - 1\right)} \quad \textbf{\textit{Eq. 82}}$$

$$\text{Ieq} = \left(I_{br}(V_{BC,n}) - V_{BC,n} * g\right) \quad \textit{Eq. 83}$$

$$g_{pr} = \frac{I_{rs}}{\beta_R * VT} * e^{\frac{VBC}{VT}} \quad \textit{Eq. 84}$$

Following this the only extra component required is the current source. For this method we start by breaking the current into two separate current sources as shown in Figure 79:

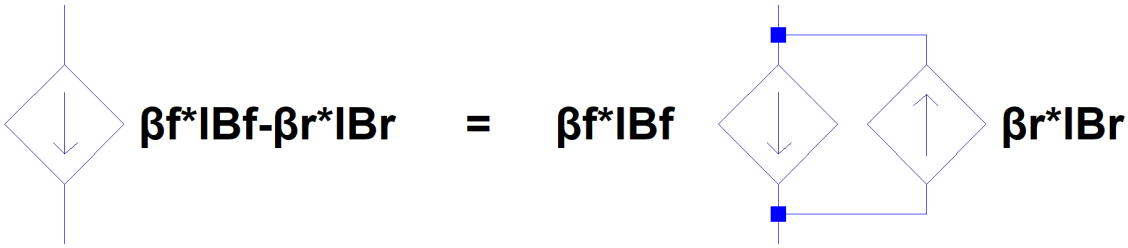


Figure 79. Breaking the current source into two current sources

What is interesting to see here is that each of these are just a product of the already derived diode currents. As such we simply multiply the previous linearized parameters by the specified β and result in the following parameters:

$$\boldsymbol{g_{mr} = \frac{I_{fs}}{VT} * e^{\frac{VBE}{VT}}} \quad \textbf{\textit{Eq. 85}}$$

$$g_{mf} = \frac{I_{rs}}{VT} * e^{\frac{VBC}{VT}} \quad \textit{Eq. 86}$$

The last step is using Kerkhof's current law we can define IC:

$$\boldsymbol{IC = \beta_f * IBf - (\beta_r + 1) * Ibr} \quad \textbf{\textit{Eq. 87}}$$

$$\text{ICeq} = \text{IC}(VBE, VBC) - g_{m,f} * \text{VBE} + g_{m,r} * \text{VBC} \quad \textit{Eq. 88}$$

We also add an extra resistor between the emitter and the collector GO to model early voltage:

$$\boldsymbol{IE = (\beta_f + 1) * IBf - \beta_r * Ibr} \quad \textbf{\textit{Eq. 89}}$$

$$go = \frac{IE}{Va} \quad \textit{Eq. 90}$$

Using these variables, we can create the circuit diagram Figure 80:

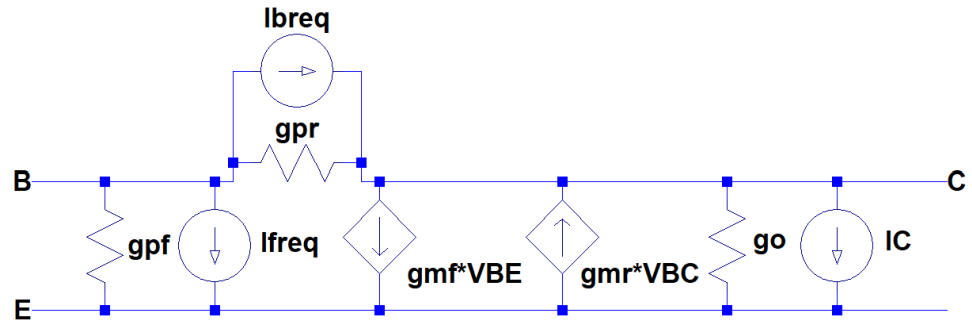


Figure 80. NPN linearized circuit

This same method can be repeated to obtain the PNP linearized circuit in Figure 81.

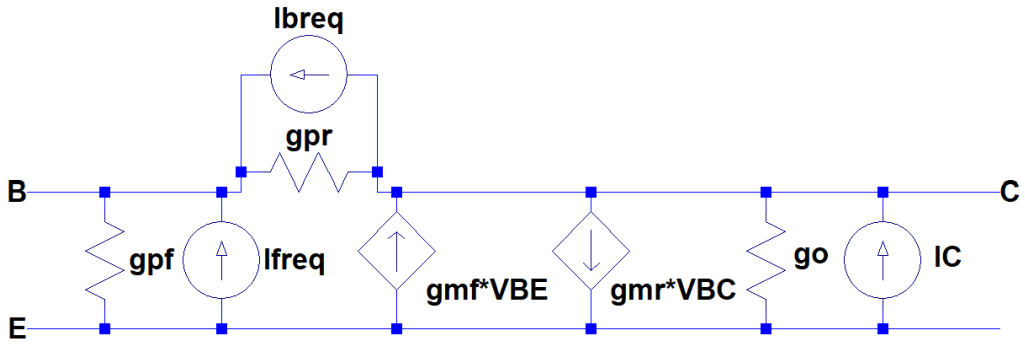


Figure 81. PNP linearized circuit

The NPN and PNP parameters are defined in Table 8.

Table 8. Linearized BJT parameters

	NPN	PNP
$g_{p,f}$	$\frac{I_{fs}}{\beta_f * VT} * e^{\frac{VBE}{VT}}$	$\frac{I_{fs}}{\beta_f * VT} * e^{\frac{VEB}{VT}}$
$g_{p,r}$	$\frac{I_{rs}}{\beta_R * VT} * e^{\frac{VBC}{VT}}$	$\frac{I_{rs}}{\beta_R * VT} * e^{\frac{VCB}{VT}}$
$g_{m,f}$	$\frac{I_{fs}}{VT} * e^{\frac{VBE}{VT}}$	$\frac{I_{fs}}{VT} * e^{\frac{VEB}{VT}}$
$g_{m,r}$	$\frac{I_{rs}}{VT} * e^{\frac{VBC}{VT}}$	$\frac{I_{rs}}{VT} * e^{\frac{VCB}{VT}}$
g_o	$\frac{I_E}{V_a}$	$\frac{I_E}{V_a}$
$I_{bf,eq}$	$I_{bf}(VBE) - g_{\pi,f} * VBE$	$I_{bf}(VEB) - g_{\pi,f} * VEB$
$I_{br,eq}$	$I_{br}(VBC) - g_{\pi,r} * VBC$	$I_{br}(VCB) - g_{\pi,r} * VCB$
$I_{C,eq}$	$\mathrm{IC}(VBE, VBC) - g_{m,f} * \mathrm{VBE} + g_{m,r} * \mathrm{VBC} - g_o * \mathrm{VCE}$	$\mathrm{IC}(VEB, VCB) - g_{m,f} * VEB + g_{m,r} * VCB - g_o * VEC$

Appendix H

Supported netlist input syntax

COMMANDS

Commands are case insensitive. The command type is specified by the command title which follows the ".". Currently, four commands are supported: TRANS, END, MODEL and OPTIONS.

The TRANS command indicates that a transient analysis should be run.

Syntax: .TRANS <TSTEP> <TSTOP> <TSTART> <TMAX>

TSTART and TMAX are optional parameters. TSTEP represents either the static timestep for linear analysis or the printing interval for non-linear analysis. TSTOP represents the simulation's end time. TSTART represents the start time and defaults to zero. TMAX gives the maximum dynamic timestep for the non-linear analysis.

The END command indicates the end of the netlist. It is required and does not accept any arguments. Anything after the END command is ignored.

Syntax: .END

The MODEL command specifies parameter values for non-linear components. The build-in default values will be used if no model statement is provided. MODEL statements are a useful netlist extension, as non-linear components have lots of parameters that can be specified. This becomes messy when done on the component's netlist line, additionally, working with component of the same model becomes cumbersome: The same parameters must be listed next to each of these components.

Syntax: .MODEL <model name> <component type> (<parameter name>=<parameter value> …)

The component type is one of D, NPN, PNP, NMOS and PMOS.

The () indicates a list, of variable length, that contains space separated parameter names and values.

Both the component type and the parameter names are case insensitive, and each parameter can only appear once. An example is:

Q1 N002 N003 N004 0 MyBJT

.MODEL MyBJT NPN (IS=0.01p VAF=100 BF=200)

Here, Q1 is associated with the model "MyBJT". This model is later defined by the model command, to be an NPN model where IS, is set to 1p, VAF to 100 and BF to 200.

An unordered map, mapping from a string to an enum, is used to convert the component type and the parameter names into integer IDs. It is checked that the IDs are valid parameters for the specified component type. The program is terminated if an invalid parameter is specified. This ID is then inserted, with its associated value, into another unordered map, mapping from int to float.

All this information is stored inside a struct, that is created for each model statement, and added to the associated components, once the END command has been detected.

The OPTIONS command is used to adjust general circuit parameters such as GMIN and ABSTOL. The default values are used if the parameters are not specified in an OPTIONS command.

Syntax: .OPTIONS <parameter name>=<parameter value> …

Parameter names are case insensitive.

When an option command is parsed, the specific commands determined through a set of if statements, after which a specific function is run on the circuit component to set up the appropriate option with the specified value.

COMPONENT

Syntax: <Component Type><Component Name> <arg1> <arg2> <arg3> … <argx>

The component type is case insensitive.

Components give a great example for modularity. The input module only checks the first character of the netlist line to determine the component's type. It then passes all the netlist arguments into the component's constructor for it to deal with.

This is possible because all the components implement a constructor, which takes in exactly three parameters: A string, a vector of strings and a vector of floats. The string is the component's name, the string vector contains additional arguments such as nodes or model name, and the float vector contains general circuit parameters such as the static time step for linear circuits. Modularity and polymorphism mean that the input module can initialize components without knowing anything about their internal implementation. This makes it trivial to modify and add components later.

MULTIPLIERS

All numerical inputs can be used with a multiplier, such as "k" to indicate x10^3. This is implemented using a helper function that separates the multiplier from the rest of the string argument. The numerical part is converted into a float and multiplied by the multiplier. Any units that come after the multiplier, such as "s" for seconds, are ignored.

For example, "1.3ks" will be separated into 1.3, "k" and "s". The 1.3 is converted into a float, the "k" is converted into 1000 and the "s" is ignored. The returned value will equal $1.3 * 1000 = 1300$.