MCGILL FACULTY OF ENGINEERING
Fall 2021
ECSE 343

Group project -13 (Bug-Captures')

Lawi Mwirigi : 260831614

Atef Halawi : 260830789

Introduction

The primary objective of this project was to solve for the output voltage V2 given a set of matrices and vectors containing unknowns derived from the circuit's nodal current equation, $F(X) = G\,X + g(X) - U = 0$, where **X** is the unknown vector, **U** is the source vector, the matrix **G** containing resistor contributions, and finally vector $g(X)$ containing the diode currents, which depend on the vector **X**. The circuit in question is a full wave rectifier, which is used to convert alternating current (AC) to pulsating direct current (DC) for use in power supplies via a diode arrangement.
We solved for the output voltage of v2 using a variety of methods we learned in class for solving nonlinear equations, including Newton-Raphson and solve implemented in Matlab.

Main Body

The circuit we were solving for was implemented as shown in figure 1.

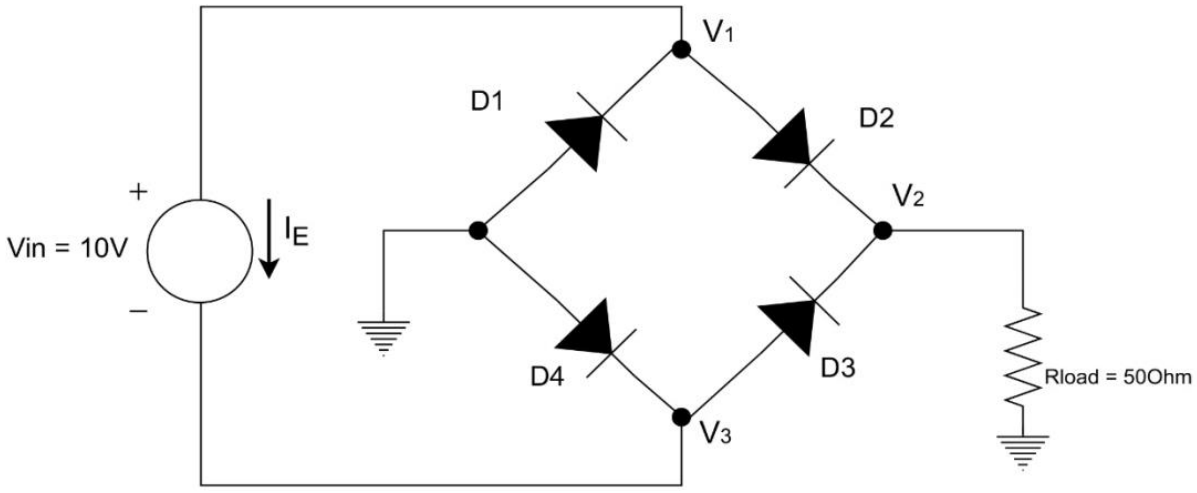

Figure 1: Full wave rectifier

A full wave rectifier is a circuit arrangement which makes use of both half cycles of input AC and converts them to DC. This process of converting both half cycles of the input supply in AC to DC is termed full wave rectification.

Regarding the first half cycle, current flows originating from the source through the diode D2 from v1 to v2, continuing into the load, whilst part of the current flows back to the source through the diode D4 and the node V3 thus completing the circuit. During this period, diodes D1 and D3 are reverse biased and thus no current flows through them.

During the second half cycle the reverse occurs and diodes D3 and D1 conduct current, whilst diodes D1 and D4 are in reverse bias, thus do not conduct.

Keeping this in mind, the load can now make use of both cycles, as it consumes current in a DC like fashion. So with this background knowledge and with the provided nodal equation mentioned earlier, we can now discuss in further detail the two methods implemented in order to derive the value of v2, the second component in the unknown vector X.

Part 1

In part 1, given a 10V DC source, firstly the function F and its Jacobian equivalent J were derived via the given nonlinear function “nonlinearFunc()”, in which we pass our initial unknown X vector. We then proceed to solve for the unknown component v2, or x2, the second entry of the X vector, by making use of the inbuilt matlab function “Solve” in which we pass the function in question F, which we derived from nonlinearFunc method, the unknown vector X, and finally we specify which component of the unknown vector X we are seeking, in this case x2 being v2, from which we got v2 = 8.5914 volts.

With regards to the result, this was approximately what we anticipated. However to verify the accuracy of the result, we constructed a simulink circuit mimicking the given circuit from figure 1, as seen in figure 2, and obtained the actual value of the voltage V2 to be equal to 8.552614V, as seen from the oscilloscope attached to voltmeter in figure 3, by placing a voltmeter in between the nodes of the resistor, obviously representing diode D2.

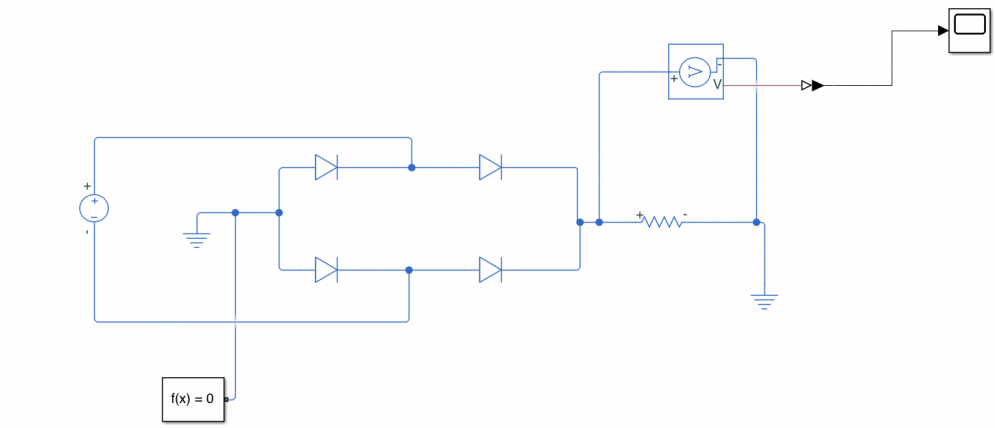

Figure 2: Simulink equivalent circuit with DC source voltage

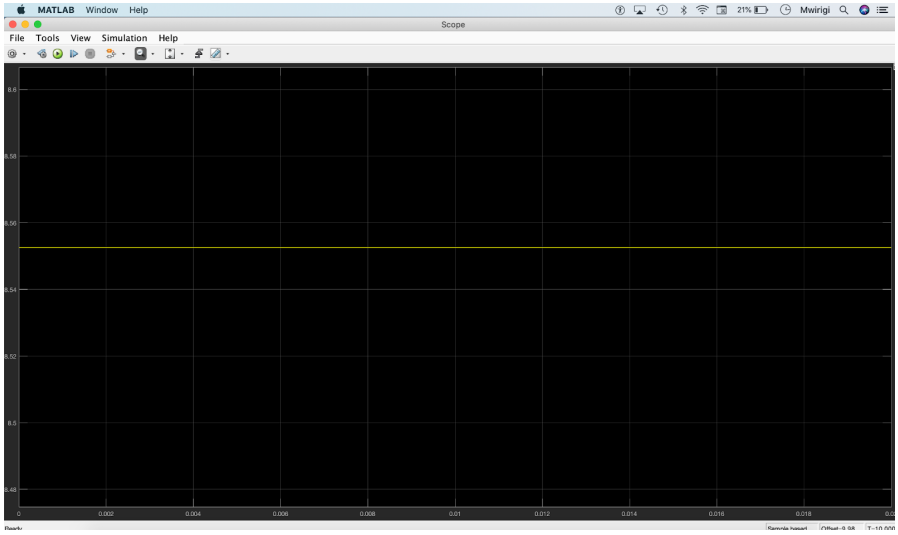

Figure 3: Voltage v2 as seen from oscilloscope

When computing the errors, we obtain an absolute error of 0.038786 and relative error of 0.4535%, which are very minimal in nature. Therefore we conclude that this relative error is acceptable and the solve method is highly accurate.
Moreover, it was observed that the runtime is 5.432195 seconds.

Part 2

Here, we were asked once again to derive the value of v2, but this time by passing values of the unknown vector X without any meaningful guess by the user, thus we updated our implementation method. We introduced our own Newton-Raphson implementation to solve for X instead of the solve method provided by matlab, as unlike the solve method, the Newton-Raphson method makes use of the initial guess provided by the user.

The Newton-Raphson method, “NewtonRaphson()” takes as input the initial guess of the user Xguess, and a tolerance value, provided to us. This function then outputs X_final, where it’s supposed to represent the actual values for the X vector. The mechanism of this method involves a for loop, in which the number of iteration is set to a maximum of 1000 for functionality purposes. This algorithm makes use of the previously implemented nonlinearFunc method as well. Beginning from it’s first iteration, the initial Xguess is set as X_prev, in which we pass it into the nonlinearFunc method, to retrieve the according function and it’s Jacobian equivalent set as F_prev and J_prev. The algorithm then set’s a new X vector value represented as X_current, by subtracting the quotient F_prev/J_prev from the X_prev. We then once again make use of the nonlinearFunc method by passing X_current into it in order to retrieve the updated equivalent function set as F_current and it’s Jacobian equivalent set as J_current. This is followed by computing the normDeltax, which is set by making use of the norm function of matlab in which we pass X_current - X_prev. The X_prev is then set as X_current and is iterated by a value of 1 in the case the loop is not broken. This loop is repeated several times, with each iteration returning a X_current to be more and more

accurate when compared to it's previous looped values until the following condition is met; that is that both the normDeltaX and the norm of F_current are both less than the tolerance, the loop breaks and sets the current X vector as the final X vector, X_final. We then simply obtained the value of v2 by getting the second element in the X vector, which we found to be v2 = 8.5914 volts. We immediately observed that this was similar to the value we obtained in part 1.

With regards to the accuracy of this method, when comparing them to actual voltage of 8.552614V, the computed relative error is that of 0.4535%, which is the same as the relative error of the previous method and is also extremely low, thus it can be inferred that the two methods are as accurate.

The only advantage of using this implementation over that of part 1 is to do with the rapidness of this method, it is faster as compared to the one in part 1. i.e the Elapsed time is 0.606159 seconds.This is a 896.17% improvement compared to the first implementation thus validating the choice of our algorithm and pseudocode.
Also we can note that our algorithm is working for a trivial guess of all zeros.

Part 3

For this part of the project, we were to implement the same exact circuit however this time we replaced the DC voltage source with an AC voltage source, 5sin(377t). This means for every time the value of Vin was different, due to the sinusoidal nature of the voltage source. In order to adapt to the situation, we slightly modified our Part 2 code, as now there is a time dependency to be taken in account for, due the time dependency nature of the AC voltage source.

With regards to the Newton-Raphson method we simply added an extra input time "t" as well as adjusting the value of U(4,1)=5 * sin(377*t) in the nonlinearFunc() method, as that value was once the DC source voltage value. We then called these methods inside a time loop starting at t=0, that had a delta t of 1e-3 and was running up to 0.05. After

retrieving our final X2 values, at different times, we plotted the result as a function of time as seen in figure 4.

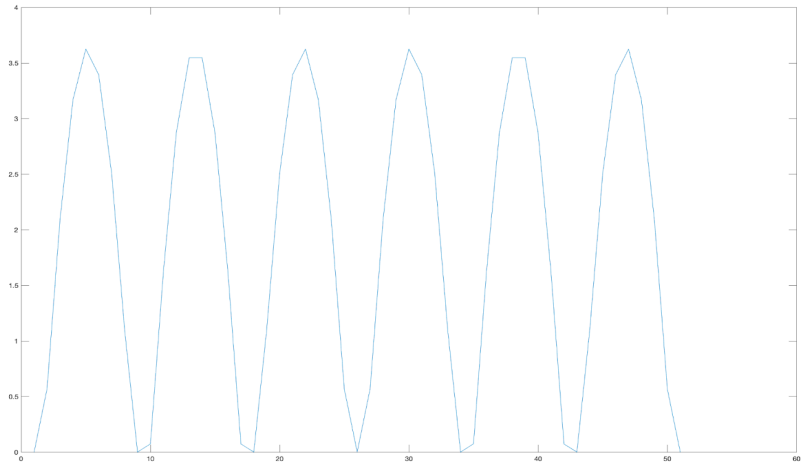

Figure 4: v2 voltage from t=0 to to t=0.05

We also obtained an Elapsed time of 0.490326 seconds.

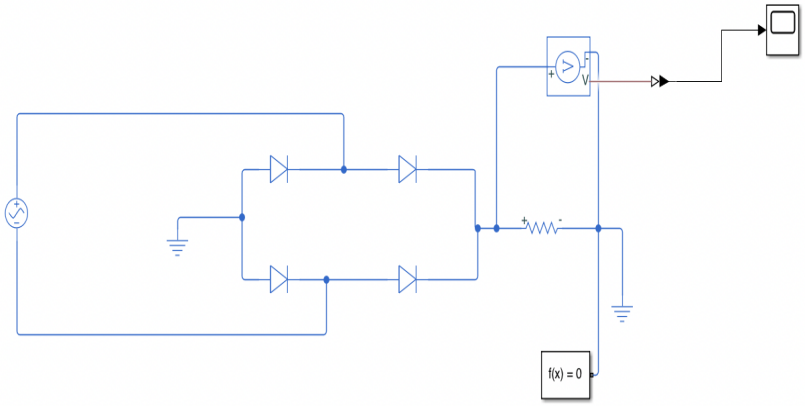

Figure 5 : Simulink equivalent circuit with AC source voltage

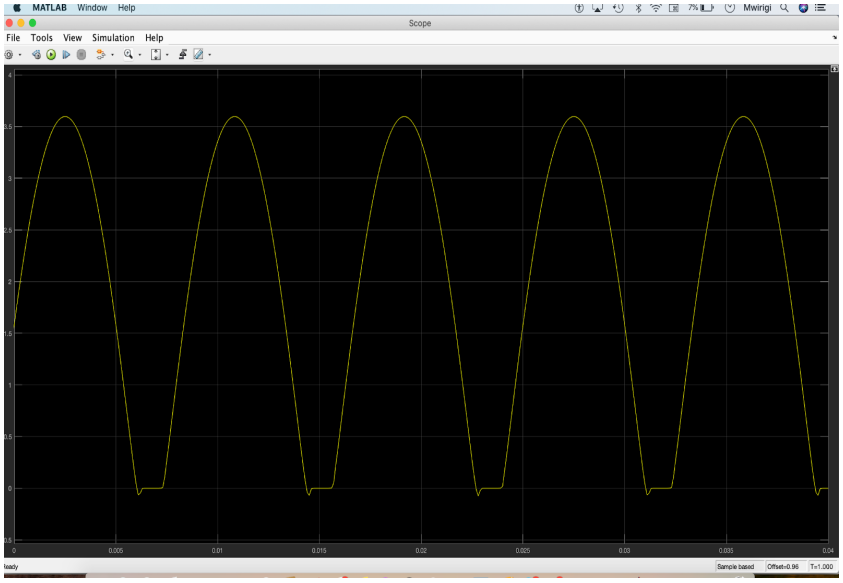

Figure 6: Voltage v2 as seen from oscilloscope

With regards to the actual values retrieved for v2, when comparing them with the actual output of v2 as seen in figure 6, we note they are very much similar as expected however we can identify small difference in the troughs of the waves, the matlab ones are smooth while the simulink scope has some dent in it. This is due to the hidden capacitance inbuilt in the simulink.

Conclusion

In conclusion, we can see that the algorithm implemented in part 2 is more desirable as explained earlier, as even though it's as accurate when comparing the relative error, the speed of computation is far better. With regards to part 3, we were able to derive a good replica of the actual voltage v2 however…..

Contribution of each team member

The key goal here was to develop teamwork experience and responsibility sharing from each of the members.
Mostly Lawi implemented the Matlab code and the simulink circuit with the help of Atef, while Atef on the other hand mostly worked on the project report, the powerpoint with the help of Lawi.

Moreover, we both worked on a 10 min video describing what we had done with the use of the powerpoint.