ENSC 895: SPECIAL TOPICS: THEORY, ANALYSIS, AND SIMULATION OF NONLINEAR CIRCUITS

Computing DC operating points of non-linear circuits using homotopy methods

Spring 2004

Final Project Report

Renju S. Narayanan
rsn@sfu.ca

Bozidar Vujicic
bvujicic@cs.sfu.ca

Svetlana Vujicic
svujicic@cs.sfu.ca

Web page:
http://www.ensc.sfu.ca/people/grad/bvujicic/ENSC895/Webpage.htm
Table of Contents

1. Table of contents ........................................................................................................ 2
2. Abstract ...................................................................................................................... 3
3. INTRODUCTION ......................................................................................................... 4
4. Technical background ................................................................................................. 5
   Homotopy .................................................................................................................... 5
   Mathematical preliminaries .......................................................................................... 6
   Parser ......................................................................................................................... 9
5. Implementation ........................................................................................................... 10
   Parser ......................................................................................................................... 11
   Matlab code ............................................................................................................... 11
6. Simulations and Results ............................................................................................ 14
   Schmitt Trigger circuit ............................................................................................... 14
   Chua’s circuit ............................................................................................................. 16
   Flip-Flop circuit ......................................................................................................... 20
7. Conclusions and Future work .................................................................................... 21
8. REFERENCES ............................................................................................................. 23

Appendix A: Code Listing
Appendix B: Orchard circuit
1. Abstract

DC operating points of a nonlinear circuit give the information regarding the circuit’s behavior and stability. Some circuits are designed to operate at a single operating point. For some other circuits, it is desirable to have more than one operating point. Therefore, it is very important that the correct operating point(s) of a circuit is (are) found.

There are different methods to find DC operating points of a nonlinear circuit. One of the methods is using artificial parameter homotopy. In this method, a parameter is embedded into the equations describing the circuit. Embedding the parameter makes it easier to solve the equations.

This report describes a project that has been done to find the DC operating points of the nonlinear circuits using homotopy methods implemented in the software package MATLAB.

This report gives a brief introduction to Homotopy methods, mathematical preliminaries for solving the nonlinear equations and a C++ utility Parser. Then, the project implementation details in Matlab are described followed by the description of different circuits used, results and future work.
2. Introduction

The most important step in the analysis of a nonlinear circuit is calculating the DC operating points. There are different methods to find DC operating points of a nonlinear circuit. One of the most commonly used methods is the Newton-Raphson method. This method has many drawbacks. One of them is the convergence problem. Another disadvantage is that, the user has to guess a starting point that is very close to the solution. If a circuit has more than one operating point, then this method can find only one operating point that is closest to the starting point. Therefore, the user has to guess a starting point more than once to find all the operating points.

Finding operating points using homotopy methods provides a solution to the problems described above. Aims of this project were to simulate different nonlinear circuits using Pspice, to find DC operating points of different circuits using homotopy methods implemented in Matlab and to compare the results from Matlab with the results obtained from circuit simulator Hspice.

The circuits used in this project are:

- Schmitt Trigger circuit
- Chua’s circuit with nine DC operating points
- Flip - Flop circuit.
3. Technical background

3.1 Homotopy

Homotopy methods are more efficient in finding DC operating points of a nonlinear circuit than the conventionally used methods. Homotopy methods converge properly and can find multiple solutions (if there are multiple solutions).

Consider that the nonlinear equations to be solved are of the form

\[ F(x) = 0, \text{ where } F: \mathbb{R}^n \rightarrow \mathbb{R}^n, \ x : \mathbb{R}^n \rightarrow \mathbb{R}^n \]

Homotopy method introduces an additional parameter called the homotopy parameter or continuation parameter (\( \beta \)) to the system of equations to be solved. This results in a system of equations of the form

\[ H(x, \beta) = 0, \text{ where } H: \mathbb{R}^{n+1} \rightarrow \mathbb{R}^n, \ x : \mathbb{R}^n \rightarrow \mathbb{R}^n \]

This system of equations has a one higher dimension than that of the original system. Now the system becomes easier to solve than the original case. [3]

The homotopy parameter, \( \beta \) is varied from 0 to 1. The parameter, \( \beta \) is embedded in such a way that when \( \beta = 0 \), the system

\[ H(x,0) = 0 \]

is easier to solve.

When \( \beta \) reaches 1, \( H ( x, 1) \) formulates back into the original system.
\[ H(x,1) = F(x). \]
Starting from \( H(x, 0) = 0 \), a set of points \((x, ?)\) is found such that, \( H(x, ?) = 0 \) until \( ? = 1 \). These set of points are connected and they trace a path known as zero curve [3], [6].

In order to prevent the “fold back” of the curve, \( ? \) is made a function of the arc length \( s \). This method is known as the arc length continuation [6]. Now the system of equations becomes \( H(x(s), ?(s)) = 0 \).

The points in the trajectory, \( y(s) = [?(s) \ x(s)]^T \) are obtained by differentiating \( H(x(s), ?(s)) \) with respect to \( s \) on \( y(s) \) [5].

\[
\frac{d}{ds} H(x(s), ?(s)) \frac{??H}{???} \frac{??x}{???} \frac{??}{??s} \frac{??}{???} \frac{d?}{ds} ? \quad \frac{??y}{???} \quad \frac{??}{??s} \quad \frac{??P(y)y}{??} \quad 0
\]

The above equation needs to be solved for \( y' \) has rank \( n \), then \( y' \) is unique in direction. The norm of \( y' \) has to be 1. Now \( y' \) has to be integrated with the initial condition \( y_0=[0 \ x_0]^T \), where \( x_0 \) is the solution when \( ? = 0 \) [5].

3.2 Mathematical preliminaries

Nonlinear equations are solved usually using Newton-Rhapson methods. Nevertheless, these methods can give us only one solution to problem, and we need to provide initial guess that is close to solution. Spice like simulators uses these techniques. Moreover, these algorithms don’t converge always. Parameter embedding methods (homotopy methods) are more promising since they have the ability to converge to solution if they are implemented properly and they designate more than one solution for nonlinear problems. Homotopy methods continuously modulate easy problem for whose solution is known into solution of hard problem. In homotopy we introduce one additional parameter lambda whose value vary between 0 and 1. For value of lambda equal to 0 we have well known problem, and for value of lambda equal to 1 we have solution of complex problem. In order to ensure convergence (we need to ensure that lambda reaches one) the homotopy function is parameterized with respect to the arc.
length of this trajectory. On this way, the solution of easy problem is gradually modulated to the solution of the difficult problem by tracing path. There exist three basic path tracking algorithms for homotopy method: ordinary differential equation based, normal flow, and quasi Newton augmented Jacobian matrix. In our project we implemented ordinary differential equation tracking algorithm. This solver use variable-step predictor-corrector method. MATLAB code odepcvs implements variable-step predictor-corrector ordinary differential equation (ODE) solver. Variable step is better than fixed step-size for two reasons: changing step size is necessary to reach points on y(s) where lambda is equal to 1 and it is desirable to get largest step size for fixed error tolerance. Variable –step predictor corrector method in our project employs Gear’s implementation [9]. This approach takes previous derivative evaluation and creates a polynomial spline. The spline is a piecewise polynomial function that can have a locally very simple form, yet at the same time be globally flexible and smooth[mathworld.wolfram.com]. When step size is modified, previous derivative is recalculated based on the spline. Complexity of this homotopy method arises when we need to locate the exact points on y(s) where lambda is equal to 1. This is solved by predicting the appropriate step-size so that targeted points are reached. Prediction is exercised by employing cubic spline \( \lambda(s) = as^3 + bs^2 + cs + d \) using the last four computed points and calculated the roots of \( \lambda(s) - 1 = 0 \). Between each pair of evaluated points there is a cubic curve. To assure that curves are jointed together flowly, the first and second derivative at the end of one curve must be equal to the first and second derivative in beginning of following one. In essence, evaluating the natural cubic spline involves solving a system of simultaneous equations to provide condition for above mentioned to take place. Substantially, we evaluate smallest root \( r \) that is greater than current value of \( s \) and we predict the next step size from difference \( h = r - s \). Since tol(2) provides bound for lambda being close to 1 before accepting solution, if lambda skip this bound, algorithm will record value of x at that point as solution. User can determine order of predictor-corrector method. The order of ODE is the order of the highest derivative of y regarding to x appearing in it. The order of predictor corrector method is defined based on local truncation error (LTE). Namely, predictor-corrector methods proceed by extrapolating polynomial fit to the derivative from previous point to the new point (predictor step), then using this to interpolate the derivative (the corrector step) [9]. Since we have used in our project first-order Adams Bashforth method as predictor and first –order Adams-Moulton as corrector, let us to explain term of local truncation error.
on example of these methods. In order to understand these methods we are going to start of more general methods: Forward and Backward Euler methods. If we denote by \( t_n \) \( n \)th time-step and \( y_n \) is value of function in that moment Forward Euler methods computes \( y_{n+1} \) as:

\[
y_{n+1} \approx y_n + hf(y_n, t_n)
\]

where \( h \) is step size given by:

\[
h = t_{n+1} - t_n
\]

and we assumed that \( h \) is constant for sake of simplicity.

If we expand \( y \) in neighborhood of \( t = t_n \) in Taylor series:

\[
y(t_n + h) = y(t_n) + h \frac{dy}{dt}igr|_{t_n} + \frac{h^2}{2!} O(h^2) + \cdots
\]

We can conclude that error is produced due to truncation of the Taylor series, and at that point we introduce term of local truncation error (LTE) of given method. In above example the LTE is \( O(h^2) \) and it refers to us as first order method. Generally speaking, method which have LTE \( O(h^{k+1}) \) is kth order method. As we can see higher order methods give us lower truncation error respectively lower global error, which is defined as absolute value of difference between true solution and computed solution. If we talk about convergent numerical methods we can say that numerically computed solution approaches to the true solution when the step size approaches to zero. Since, we don’t know the true solution we can assume that solution that we can get with small step size is true solution. Forward Euler method is explicit method since \( y_{n+1} \) is given in terms of well known values such as \( y_n \) and \( f(y_n, t_n) \). We will see latter that in case of the implicit equations this term \( y_{n+1} \) depends also of \( f(y_{n+1}, t_{n+1}) \) which is not known and we will define that as implicit method. Explicit methods are easier to implement but deficiency is limitation of time step to ensure numerical stability. Implicit methods are harder to implement since \( y_{n+1} \) is given in term of implicit equations but we can satisfy more stringent stability requirements.

Corresponding method is Backward Euler method and it is based on the next Taylor series:

\[
y_{n+1} \approx y(t_n) + h \frac{dy}{dt}igr|_{t_{n+1}} + \frac{h^2}{2!} O(h^2), \quad \text{and}
\]

\[
y_{n+1} \approx y_n + hf(y_{n+1}, t_{n+1})
\]

Whole idea behind predictor corrector methods is to use combination of explicit and implicit techniques to provide better convergence characteristics. In practice we can find often combinations of Forward Euler as predictor and Adams Moulton 2 as corrector. Adams methods are based on the idea of approximating the integrand with polynomial within the interval \( (t_n, t_{n+1}) \), using a kth order polynomial results in k+1 th order
method [9]. Implicit type of Adams methods is Adams-Moulton and explicit type is Adams-Bashforth method. The second order of these methods use linear interpolation and it is given as follow:

\[ y_{n+1} - y_n = \frac{h}{2} (f(y_{n+1}, t_{n+1}) - f(y_n, t_n)) \]  

Adams-Moulton of second order

\[ y_{n+1} - y_n = \frac{h}{2} (3f(y_n, t_n) - f(y_{n-1}, t_{n-1})) \]  

Adams-Bashforth of second order

In our project are applied first order of these methods what is simply the Forward Euler and Backward Euler methods and that Adams Bashforth of the first order as predictor and Adams Moulton of the first order as corrector. Adams-Moulton of order \( p \) is more accurate than Adams-Bashforth of the same order, so it can use larger step size. These methods are more complex than Runge-Kutta methods since it deals with variable step size. Namely, these methods attempt to select at each step optimal step size to satisfy certain accuracy. If we use higher order of these methods we will achieve higher accuracy, and we could go up to order four in our example. Following figure shows interpolation of \( p \)th order of Adams-Bashforth and Adams-Moulton methods.

3.3 Parser

DC operating points of a nonlinear circuit are the solutions of nonlinear equations describing the circuit. For transistor circuits, the number of equations describing the circuit would be twice the number of the transistors. Writing equations manually and solving them might be easier for circuits with one or two transistors. However, for circuits with larger number of transistors, writing circuit equations and solving them manually
is a very hard task. The effort of writing equations describing nonlinear circuits can be reduced by using the software program Parser.

Parser is a C++ utility written by Edward Chan [4]. Parser requires an input file, which contains circuit descriptions in the Spice netlist format. Parser can generate Nodal or Modified Nodal equations. The user can choose the type of equations required. In addition, we can also specify reference node otherwise parser will take node with the largest number of connections as reference node. The output of Parser is a text file with extension .Pout. The output file contains the nodal or modified nodal equations (as selected by the user) and the Jacobians of the equations. It also contains the transistor parameters and the values of the resistors used in the circuit description.

Parser requires that the nodes in the circuit should be represented by unsigned integers in the input file. The input file should contain a .model statement, because the parsing begins by constructing necessary model objects from the .model statements. Parser reads the node descriptions and stores the different connections to each node in the memory. The equations for nodes are generated by looking at the node connections stored in the memory. [4]

4. Implementation

The spice netlist file was used as the input to the Parser after certain modifications. The equations generated by the Parser were solved using MATLAB.

We used the same homotopy function for all circuits. The homotopy function used is:

\[ H(x, ?) = (1- ?) \times g_{\text{leak}}(x-a) + ? \times F(x), \]

where \( x \) is the matrix containing node voltages, \( g_{\text{leak}} \) is the leakage conductance added to the nodes.

When \( ? = 0 \), the equation reduces to \( H(x, 0) = g_{\text{leak}}(x-a) \). The solution to this equation, (i.e. the starting point of the homotopy trajectory) is \( x = a \), where \( a \) is a random vector depending on the number of equations.

When \( ? = 1 \), the equation reduces to \( H(x, 1) = F(x) \). Values of the node voltages when the homotopy path crosses \( ? = 1 \) are the solutions to equation \( F(x) = 0 \).
4.1 Parser

We used Parser to obtain equations and Jacobians for the circuits.

Some modifications were made to the Parser code. Parser stores the node details in memory and the memory were not released properly. So it was unable to run Parser more than once. The code was modified so that the memory allocation and release were done properly.

The input to the Parser was the spice netlist obtained from the Pspice Schematics (except for Chua’s circuit). The netlist had to be modified in order to make it in the format recognized by the Parser. In the netlist obtained from Pspice, the nodes were represented by numbers and characters (e.g.: $N_{0002}$). The characters were removed and the node numbers were given as integers. In addition, .model statement with saturation current, forward and reverse current gains and temperature was added to the Parser input file. The extra white spaces were removed from the netlist because Parser generates error when more than one white space is present between the component name and values in the input file.

The output of the Parser was double checked before using it in the Matlab code. Some equations for Chua’s circuit obtained from the Parser were wrong. As a result the Jacobians generated were also wrong. These errors were fixed before giving the Parser output to Matlab. For some circuits, the Parser missed some Jacobians. The missing Jacobians were added in order for the Matlab code to function properly.

4.2 Matlab code:

The ordinary differential equation solver which uses above mentioned numerical technique consists of five Matlab files: Init.m, Jac.m, Pchomotopy.m, Deval.m and Eval.m.

Init.m:
Init file is needed to determine the starting point of homotopy path. Instead of using a random vector, we can start with some stated value. If that initial guess is good, algorithm will converge faster.

Pchomotopy.m:
Pchomotopy solves nonlinear system of equations using ODE variable-order and variable-step corrector predictor integration technique. This is the
main function. It contains numerical integration routine and routine to find values when lambda equals 1. PC stands for predictor corrector. Inputs of this function are J – string containing name of user supplied homotopy Jacobian, i.e. name of function that evaluates the Jacobian P(y), x0-solution of homotopy when embedded parameter is equal to 0, s-final- final value of arc length s, k- order of corrector predictor methods and two dimension tolerance vector tol which determine relative bound to local truncation error of numerical integration, tol(1)(default value is 1.e-6) and bound for lambda being close to 1 before accepting solution, tol(2) whose default value is 1.e-6.Pchomotopy returns column vector ,sout ,of evaluated arc lengths, vector lambdaout, of values of lambda along trajectory, vector of calculated solutions along trajectory(xout), vector of calculated solutions of homotopy when lambda is equal to one xsolout, one column vector per solution and number of the homotopy Jacobian evaluations NJaceval.

Deval.m:
Deval evaluates derivative vector y’ from homotopy Jacobian and it is called by Pchomotopy. Inputs to this function are J-the Jacobian evaluation function, y- current calculated point on the trajectory, y’ old -the previous derivative evaluation, y’ -derivative evaluation and bad- a Boolean which indicates whether derivative calculation was successful. Derivative evaluation is achieved by calculating the following linear equation:

$$P(y) \cdot y' = 0,$$

Where $P(y) \in \mathbb{R}^{n \times (n+1)}$

Since $P(y)$ is not square we can’t use standard linear solvers and we could employ QR-factorization [9].

$$P(y) = QR$$

This gives us a new aspect to the problem. Now the problem reduces to

$$R \cdot y' = 0$$

$$R = [U \: r]$$

U is an upper triangular matrix U $\in \mathbb{R}^{n \times n}$ and r is a (n+1) dimensional vector. Thereafter we can reformulate problem as:

$$\begin{bmatrix}
y'(1) \\
y'(2) \\
\vdots \\
y'(n) \\
y'(n+1)
\end{bmatrix}
= \begin{bmatrix}
U_1 \\
U_2 \\
\vdots \\
U_n \\
r
\end{bmatrix}
\begin{bmatrix}
y(1) \\
y(2) \\
\vdots \\
y(n) \\
y'(n+1)
\end{bmatrix}$$

Since we are going to normalize vector y’ we can set y’ (n+1) = 1 which will give us the standard linear equation. The derivative y’ is normalized
and its sign is chosen such that the inner product of y’ and its previous evaluation is greater than zero (i.e. the angle between the two evaluation vectors is less than 90 degrees). If a derivative evaluation is taken on point that is not on trajectory y(s), Jacobian P(y) could be ill-conditioned or rank-deficient [9]

**Jac.m:**

Jac evaluates homotopy Jacobian function. Inputs for this function are embedded homotopy parameter and node voltages. This file needs to be modified according to the number of circuit equations.

**Eval.m:**

Eval is needed for checking.

In our project we have used variable step predictor corrector methods since these methods have advantage over Runge-Kutta method that integrates by predicting future derivatives. In essence, we use explicit Adams-Bashforth method to predict the next value of y(s) and then implicit Adams-Moulton to correct that value. Specific mathematical formulation of p order for these methods is as follow:

\[ y_{\text{n+1}}^{[0]} = y_{\text{n}}^{[0]} + h f(y_{\text{n}}) \]

\[ y_{\text{n+1}}^{[p]} = y_{\text{n}}^{[p]} + h^p f(y_{\text{n}}) \]

where \( p \) is the order of the method, and \( h \) is the step size.

As we can see from above formulas these methods use past derivatives and they don’t expect any of future derivatives in evaluation. If we allow \( p \geq 1 \) we can predict Local Truncation Error (LTE) by using so called Milne’s trick

\[ e_{\text{n+1}} = \frac{C_{p+1}}{C_{p+1}^*} (y_{\text{n+1}}^{[1]} - y_{\text{n+1}}^{[0]}) \]

Where \( C_{p+1} \) and \( C_{p+1}^* \) are the LTE constants of p order of Adams-Moulton and Adams-Bashforth methods. Variable step is achieved by taking previous derivative evaluations and making polynomial spline. When the step size is modified, previous derivatives are recalculated based on spline. (Gear’s implementation [9])
5. Simulations and Results

In this section, the different circuits used for simulation and simulation results are explained.

5.1 Schmitt Trigger circuit

The circuit diagram for the Schmitt trigger is given below.

Figure 3. Schmitt Trigger circuit

Nodal equations for this circuit were obtained from Parser. A leakage conductance was added to each node in the circuit. The value of leakage conductance added was varied with lambda.

The resulting trajectories are shown in the figure below. It is a plot of the node voltages versus lambda.
The Schmitt trigger circuit is known to possess three operating points. From the trajectories, it can be seen that each trajectory crosses the $\lambda=1$ line three times thus indicating the three operating points.

The results from the Matlab simulation were compared with the output of Hspice.

<table>
<thead>
<tr>
<th>voltages</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>V(1)</td>
<td>0.71026</td>
<td>1.7659</td>
<td>8.6308</td>
</tr>
<tr>
<td>V(2)</td>
<td>0.67246</td>
<td>0.69174</td>
<td>0.96455</td>
</tr>
<tr>
<td>V(3)</td>
<td>10</td>
<td>7.2396</td>
<td>1.0382</td>
</tr>
<tr>
<td>V(4)</td>
<td>0.71024</td>
<td>1.4899</td>
<td>1.7962</td>
</tr>
<tr>
<td>V(5)</td>
<td>1.5</td>
<td>1.5</td>
<td>1.5</td>
</tr>
<tr>
<td>V(6)</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>voltages</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>V(1)</td>
<td>0.7128394</td>
<td>1.7694</td>
<td>8.632</td>
</tr>
<tr>
<td>V(2)</td>
<td>0.6753985</td>
<td>0.694778</td>
<td>0.96460</td>
</tr>
<tr>
<td>V(3)</td>
<td>10</td>
<td>7.2084</td>
<td>1.038</td>
</tr>
<tr>
<td>V(4)</td>
<td>0.7128395</td>
<td>1.4902</td>
<td>1.7931</td>
</tr>
<tr>
<td>V(5)</td>
<td>1.5</td>
<td>1.5</td>
<td>1.5</td>
</tr>
<tr>
<td>V(6)</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

Results from Matlab  
Results from Hspice

Figure 5. Tables showing the values of node voltages obtained from Matlab and Hspice simulations

It can be seen that the two results are consistent.
5.2. Chua’s circuit

Another circuit we have used in this project is Chua’s circuit. It’s a theoretical circuit developed by Chua. The circuit consists of four transistors. This circuit is known to have nine operating points. The circuit diagram is shown below.

The same homotopy function was used for Schmitt trigger circuit and Chua’s circuit.

The trajectories of node voltages and source currents are plotted against lambda.
Figure 7. Node voltages plotted against lambda.

Figure 8. A plot of current through voltage sources against lambda.
One of the node voltages has been zoomed to give a clear view of the trajectory. We can see that the trajectory crosses the $\lambda = 1$ line nine times indicating the nine operating points.

![Figure 9: Voltage at node 8 plotted against lambda](image)

Due to the number of equations and operating points, the implementation in Matlab took a longer time for this circuit compared with the time taken for other circuits.

The results from Matlab were compared with the results obtained from Hspice. As can be seen from the tables given below, the two results are consistent.
Matlab simulation results

<table>
<thead>
<tr>
<th>Voltagess</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>V(1)</td>
<td>1.7718</td>
<td>1.7823</td>
<td>1.7688</td>
<td>1.8195</td>
<td>1.8457</td>
<td>1.7823</td>
<td>1.767</td>
<td>1.8109</td>
<td>1.7279</td>
</tr>
<tr>
<td>V(6)</td>
<td>0.36887</td>
<td>1.8035</td>
<td>1.8216</td>
<td>1.8606</td>
<td>1.8727</td>
<td>0.37063</td>
<td>0.36811</td>
<td>1.8347</td>
<td>1.7832</td>
</tr>
<tr>
<td>V(7)</td>
<td>0.36887</td>
<td>0.37062</td>
<td>0.36838</td>
<td>1.8441</td>
<td>1.8727</td>
<td>1.8035</td>
<td>1.826</td>
<td>1.8577</td>
<td>1.7898</td>
</tr>
<tr>
<td>V(8)</td>
<td>1.3881</td>
<td>1.4298</td>
<td>1.4409</td>
<td>1.4811</td>
<td>1.4961</td>
<td>1.3982</td>
<td>1.3835</td>
<td>1.4597</td>
<td>1.4029</td>
</tr>
<tr>
<td>V(9)</td>
<td>1.3881</td>
<td>1.3982</td>
<td>1.3852</td>
<td>1.4687</td>
<td>1.4961</td>
<td>1.4298</td>
<td>1.4446</td>
<td>1.4772</td>
<td>1.4086</td>
</tr>
<tr>
<td>V(11)</td>
<td>0.89338</td>
<td>-0.55715</td>
<td>-0.93496</td>
<td>-2.4107</td>
<td>-2.3094</td>
<td>-0.55716</td>
<td>-0.98389</td>
<td>-2.444</td>
<td>-2.7632</td>
</tr>
<tr>
<td>V(12)</td>
<td>10.693</td>
<td>10.523</td>
<td>10.478</td>
<td>2.8934</td>
<td>2.5757</td>
<td>3.5799</td>
<td>1.5422</td>
<td>1.5844</td>
<td>1.5025</td>
</tr>
<tr>
<td>V(13)</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>V(14)</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
</tbody>
</table>

Hspice simulation results

<table>
<thead>
<tr>
<th>voltages</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>V(1)</td>
<td>1.7705</td>
<td>1.7811</td>
<td>1.7658</td>
<td>1.819</td>
<td>1.8451</td>
<td>1.7811</td>
<td>1.767</td>
<td>1.8104</td>
<td>1.7277</td>
</tr>
<tr>
<td>V(6)</td>
<td>0.36882</td>
<td>1.8021</td>
<td>0.36806</td>
<td>1.8599</td>
<td>1.872</td>
<td>0.37059</td>
<td>1.8201</td>
<td>1.8341</td>
<td>1.7827</td>
</tr>
<tr>
<td>V(7)</td>
<td>0.36882</td>
<td>0.37059</td>
<td>1.8246</td>
<td>1.8436</td>
<td>1.872</td>
<td>1.8021</td>
<td>0.36834</td>
<td>1.8569</td>
<td>1.7892</td>
</tr>
<tr>
<td>V(8)</td>
<td>1.3882</td>
<td>1.4298</td>
<td>1.3837</td>
<td>1.4818</td>
<td>1.4968</td>
<td>1.3984</td>
<td>1.4409</td>
<td>1.4605</td>
<td>1.4038</td>
</tr>
<tr>
<td>V(9)</td>
<td>1.3882</td>
<td>1.3984</td>
<td>1.4446</td>
<td>1.4695</td>
<td>1.4968</td>
<td>1.4298</td>
<td>1.3854</td>
<td>1.4779</td>
<td>1.4095</td>
</tr>
<tr>
<td>V(10)</td>
<td>10.4579</td>
<td>3.3502</td>
<td>10.2371</td>
<td>1.5947</td>
<td>2.3378</td>
<td>10.2874</td>
<td>1.54</td>
<td>2.7592</td>
<td>1.4986</td>
</tr>
<tr>
<td>V(11)</td>
<td>0.89254</td>
<td>-0.55678</td>
<td>-0.98478</td>
<td>-2.4118</td>
<td>-2.3108</td>
<td>-0.55678</td>
<td>-0.93584</td>
<td>-2.445</td>
<td>-2.7631</td>
</tr>
<tr>
<td>V(12)</td>
<td>10.6932</td>
<td>10.5227</td>
<td>1.5417</td>
<td>2.8894</td>
<td>2.5731</td>
<td>3.5855</td>
<td>10.4781</td>
<td>1.5847</td>
<td>1.503</td>
</tr>
<tr>
<td>V(13)</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>V(14)</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
</tbody>
</table>

Figure 10. Table showing Matlab and Hspice values
5.3. Flip-Flop circuit

The third circuit we used for our simulation is the Flip – Flop circuit. This is a practical circuit and possesses three operating points. The circuit diagram is shown below.

![Flip-Flop circuit diagram](image1)

The resistor values were varied to bias the transistors properly. The nodal equations were written using Parser. The homotopy paths for different node voltages are shown below.

![Node voltages vs. lambda](image2)
We can see that the trajectories clearly indicate that the circuit possesses three operating points.

We obtained consistent results for operating points using Hspice.

<table>
<thead>
<tr>
<th>Node voltages</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>V(1)</td>
<td>0.52757</td>
<td>0.47974</td>
<td>0.006819</td>
</tr>
<tr>
<td>V(2)</td>
<td>0.0084418</td>
<td>0.47325</td>
<td>0.5324</td>
</tr>
<tr>
<td>V(3)</td>
<td>0.0084414</td>
<td>0.47325</td>
<td>0.5324</td>
</tr>
<tr>
<td>V(4)</td>
<td>0.52757</td>
<td>0.47974</td>
<td>0.006819</td>
</tr>
<tr>
<td>V(5)</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
</tbody>
</table>

Node voltages obtained using Matlab

<table>
<thead>
<tr>
<th>Node voltages</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>V(1)</td>
<td>0.522017</td>
<td>0.46887</td>
<td>0.007541</td>
</tr>
<tr>
<td>V(2)</td>
<td>0.0075409</td>
<td>0.46887</td>
<td>0.522018</td>
</tr>
<tr>
<td>V(3)</td>
<td>0.0075409</td>
<td>0.46887</td>
<td>0.522017</td>
</tr>
<tr>
<td>V(4)</td>
<td>0.52201</td>
<td>0.46887</td>
<td>0.007541</td>
</tr>
<tr>
<td>V(5)</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
</tbody>
</table>

Node voltages from Hspice

Figure13. Table showing the values obtained from Matlab and Hspice

6. Conclusions and Future work

(a) In this project, we simulated circuits in Pspice and found out their operating points using Homotopy methods implemented in Matlab. The results were compared with those obtained from Hspice and consistency verification was confirmed.

(b) The equations and Jacobians obtained from Parser can be used in Matlab after certain modifications. They have to be checked for errors before using in homotopy methods.

(c) We used a simple homotopy function to calculate the operating point for the three circuits. The same homotopy function proved to be sufficient for calculating the operating points of all the circuits used in this project.

As the number of equations and operating points increase, the homotopy methods become slower. This was very evident while simulating Chua’s circuit (the circuit has 18 equations and 9 operating points).
In this report, the Matlab implementation of homotopy methods was described. The operating points of the three circuits used were successfully calculated using homotopy methods implemented in Matlab. We used variable-step predictor corrector method as the numerical integration technique in this project. The predictor corrector methods are not a good choice for high-precision applications. For such applications Bulirsh-Stoer method is better. For low precision, adaptive step size applications, Runge-Kutta methods are used [13]. The above mentioned numerical methods can be implemented in Matlab and the performance of the different techniques could be compared.

There is a lot of room for improvements in Parser. At present Parser can give only nodal or modified nodal equations. It can be modified to give equations of different forms (e.g.: $T F(v) + Gv + C = 0$). Parser can also be modified to accept input file without much modifications from the Pspice Schematics format. Some corrections need to be applied to ensure that Parser always generates valid output. Various homotopy equations could be implemented in Matlab and results could be compared. The convergence of homotopy methods could be improved if a good starting point is provided instead of a random value. So some algorithms for finding good starting points could be developed.
REFERENCES


