CIRCUIT ANALYSIS COMPUTING

"Conventional circuit simulation" usually means simulation of small to medium-sized analog circuits. The most widely known and used circuit simulation program is SPICE (simulation program with integrated circuit emphasis). This program was first written at the University of California at Berkeley by Laurence Nagel in 1975. Research in the area of circuit simulation is ongoing at many universities and industrial sites. Commercial versions of SPICE or related programs are available on a wide variety of computing platforms, from small personal computers to large mainframes. A list of some commercial simulator vendors can be found at the end of this article. The main focus of this article is the simulators themselves and the numerical methods employed in them. A few examples are also given to illustrate some uses of the simulators.

PURPOSE OF SIMULATION

Computer-aided simulation is a powerful aid during the design or analysis of electronic circuits and semiconductor devices. While the main emphasis here is on analog circuits, the same simulation techniques may, of course, be applied to digital circuits (which are, after all, composed of analog circuits). The main limitation will be the size of these circuits because the techniques presented here provide a very detailed analysis of the circuit in question and, therefore, would be too costly in terms of computer resources to analyze a large digital system.

It is possible to simulate virtually any type of circuit using a program like SPICE. The programs have built-in elements for resistors, capacitors, inductors, dependent and independent voltage and current sources, diodes, metal oxide semiconductor field-effect transistors (MOSFETs), junction field-effect transistors (JFETs), bipolar junction transistors (BJT), transmission lines, transformers, and even transformers with saturating cores in some versions. Found in commercial versions are libraries of standard components that have all necessary parameters prefitted to typical specifications. These libraries include items such as discrete transistors, op amps, phase-locked loops, voltage regulators, logic integrated circuits, and saturating transformer cores.

Computer-aided circuit simulation is now considered an essential step in the design of integrated circuits, because without simulation the number of "trial runs" necessary to produce a working integrated circuit (IC) would greatly increase the cost of the IC. Simulation provides other advantages, however:

- The ability to measure "inaccessible" voltages and currents. Because a mathematical model is used, all voltages and currents are available.
- No loading problems are associated with placing a voltmeter or oscilloscope in the middle of the circuit, with measuring difficult one-shot waveforms or probing a microscopic die.
- Mathematically ideal elements are available. Creating an ideal voltage or current source is trivial with a simulator, but impossible in the laboratory. In addition, all component values are exact and no parasitic elements exist.
- It is easy to change the values of components or the configuration of the circuit. Unsoldering leads or redesigning IC masks are unnecessary.

Unfortunately, computer-aided simulation has its own set of problems.

- Real circuits are distributed systems, not the "lumped element models" that are assumed by simulators. Real circuits, therefore, have resistive, capacitive, and inductive parasitic elements present besides the intended components. In high-speed circuits these parasitic elements are often the dominant performance-limiting elements in the circuit and must be painstakingly modeled.
- Suitable predefined numerical models have not yet been developed for certain types of devices or electrical phenomena. The software user may be required, therefore, to create his or her own models out of other models that are available in the simulator. (An example is the solid-state thyristor, which may be created from an npn and pnp bipolar transistor.)
- The numerical methods used may place constraints on the form of the model equations used.
- There are small errors associated with the solution of the equations. These errors, which are usually referred to as truncation errors, are the result of discretizing the underlying differential equations using a finite number of time steps.

CIRCUITS AND NET LISTS

Before we can consider simulation of a circuit, we must first consider how to represent a circuit in a way that the computer can understand. This is most easily done using a netlist. Figure 1 shows the circuit for a simple differential pair. The cir-

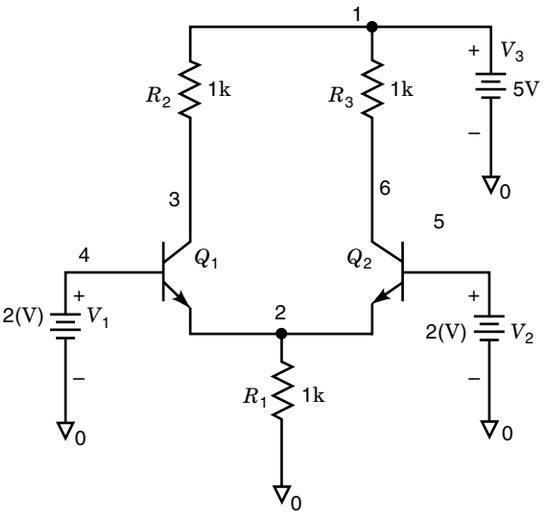

Figure 1. Circuit for differential pair.

```
V1 4 0 2V
V2 5 0 2V
V3 1 0 5V
R1 2 0 1k
R2 3 1 1K
R3 6 1 1K
Q1 3 4 2 m2n2222
Q2 6 5 2 m2n2222
.model m2n2222 NPN IS=1e-12 BF=100 BR=5 TF=100pS
```

Figure 2. Netlist for differential pair.

cuit nodes are formed wherever two or more branches meet. This particular circuit has seven nodes, which are numbered zero to six. The ground or datum node is traditionally numbered as zero.

The netlist of a circuit provides description of the topography of a circuit. Most circuit simulation programs take the netlist as the starting point for the simulation. The netlist is really just a list of the branches (or elements) that make up the circuit along with the nodes to which they are connected. Normally the elements may be entered in any order and each has a unique name, a list of nodes, and either a value or model identifier. For the differential amplifier of Fig. 1, the netlist is shown in Fig. 2. The format used here corresponds to that used by SPICE. The first three lines define the three voltage sources. The letter V at the beginning tells SPICE that this is a voltage source element. The list of nodes (two in this case) is next followed by the value in volts. In SPICE for two terminal elements, the positive node is listed before the negative node. The syntax for the resistor is similar to that of the voltage source; the starting letter R in the names of the resistors tells SPICE that these are resistors. SPICE also understands that the abbreviation k after a value means 1000. For the two transistors Q_1 and Q_2 the starting letter Q indicates to SPICE a bipolar transistor. Q_1 and Q_2 each have three nodes, and in SPICE the convention for their ordering is collector, base, emitter. So for Q_1 the collector is connected to node 3, the base to node 4, and the emitter to node 2. The final entry `m2n2222` is a reference to the model for the bipolar transistor. (Note that both Q_1 and Q_2 reference the same model.) The `.model` statement at the end of the listing defines this model. The model type is npn and a list of `parameter=value` entries follows. These entries define the numerical values of constants in the mathematical models that are used for the bipolar transistor.

From the schematic of a small circuit it is a trivial process to generate the netlist. Simply assign each node on the schematic a unique name (or integer value) and then create a list of the elements in the circuit similar to Fig. 2. Most commercial circuit simulation packages come with "schematic capture" software, which allows the designer to draw the circuit by placing and connecting the elements with the mouse. The inverse process of creating a pleasing schematic from the netlist is much more difficult (and is the essence of the place and route problem).

FORMULATION OF THE CIRCUIT EQUATIONS

In circuit simulators like SPICE, the circuits are represented by a system of ordinary differential equations. These equations are then solved using several different numerical techniques. The equations are constructed using Kirchhoff's voltage and current laws. The first system of equations pertains to the currents flowing into each node. One equation is written for each node in the circuit (except for the ground node), so the following equation is really a system of N equations for the N nodes in the circuit. The subscript i denotes the node index.

$$0 = \boldsymbol{F}_i = \boldsymbol{G}_i(\boldsymbol{V}) + \frac{\partial \boldsymbol{Q}_i(\boldsymbol{V})}{\partial t} + \boldsymbol{W}_i(t) \tag{1}$$

Variable $\boldsymbol{V}$ is an N-dimensional vector that represents the voltages at the nodes. Variable $\boldsymbol{Q}$ is another vector that represents the electrical charge (in coulombs) at each node. The term $\boldsymbol{W}$ represents any independent current sources that may be attached to the nodes and has units of amps. The function $G(\boldsymbol{V})$ represents the currents that flow into the nodes as a result of the voltages $\boldsymbol{V}$.

For example, for the circuit of Fig. 3, which has two nodes, we need to write two equations. At node 1:

$$0 = (V_1 - V_2)/R_1 + \frac{d(C_1 V_1)}{dt} + I_1$$

We can clearly identify $\boldsymbol{G}_1(\boldsymbol{V})$ as $(V_1 - V_2)/R$; the term $\boldsymbol{Q}_1(\boldsymbol{V})$ is $C_1 V_1$ and $W(t)$ is simply I_1. Likewise, at node 2 we can write

$$0 = (V_2 - V_1)/R_1 + V_2/R_2 + g_m V_1$$

In this equation only $\boldsymbol{G}_2(\boldsymbol{V})$ appears. In this example $\boldsymbol{G}$ and $\boldsymbol{Q}$ were simple linear terms; however, in general they will be nonlinear functions of the voltage vector $\boldsymbol{V}$.

These equations are quite easy to assemble in an automated fashion. Observe that each element that is attached to a node makes a contribution to either $\boldsymbol{G}$, $\boldsymbol{Q}$, or $\boldsymbol{W}$ for that node. To assemble the equation, we can use the following procedure:

- For each element I{
  - For each terminal j of the element I{
    - Determine the node k to which j is attached.
    - if ($k > 0$) {
      - Compute the current I_j at terminal j and sum into G
        $G_k = G_k + I_j$
      - Compute the charge at terminal j and sum into Q_k

}}}

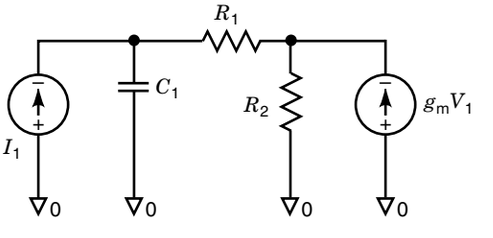

Figure 3. Example circuit for nodal analysis.

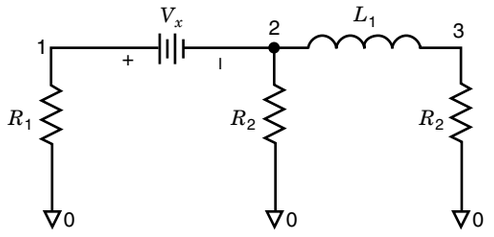

Figure 4. Circuit for modified nodal analysis.

Fortunately, it is easy to determine the node that each terminal is attached to from the netlist. The procedure outlined is nodal analysis of a circuit.

MODIFIED NODAL ANALYSIS

Normal nodal analysis, which sums the currents flowing into each node, cannot be used to represent ideal voltage sources or inductors. This is so because the branch current in these elements cannot be expressed as a function of the branch voltage. To resolve this problem, loop equations are written around each inductor or voltage source. Figure 4 shows an example of this procedure. In this figure, the unknowns to be solved for are the voltage V_1 at node 1, the voltage V_2 at node 2, the voltage V_3 at node 3, and the current flowing through voltage source V_1, which we shall call $I(V_x)$, and the current flowing in the inductor L_1, which we shall call $I(L_1)$. The system of equations is

$$\begin{aligned}
0 &= V_1/R_1 + I(V_x) \\
0 &= V_2/R_2 - I(V_x) + I(L_1) \\
0 &= V_3/R_3 - I(L_1) \\
0 &= V_1 - V_x + V_2 \\
0 &= V_2 + \frac{d(L_1 I(L_1))}{dt} - V_3
\end{aligned}$$

Modified nodal analysis effectively creates as new system of equations that augments the original system produced by nodal analysis. The second system pertains to the currents $\boldsymbol{I}$ and magnetic flux $\boldsymbol{\Phi}$ flowing in any voltage sources or inductors that may be present. These equations result from the application of the Kirchhoff voltage law equations around any voltage sources or inductors in the circuit.

$$0 = \mathbf{F}_i = \mathbf{H}_i(I) + \frac{d\boldsymbol{\Phi}_i(\mathbf{I})}{dt} + \mathbf{E}_i(t)$$

$\boldsymbol{E}(t)$ represents any independent voltages sources. In our examples, $\boldsymbol{E}(t)$ corresponds to the independent source V_x and the magnetic flux $\boldsymbol{\Phi}$ corresponds to the term $L_1 I(L_1)$.

The use of modified nodal analysis does have the disadvantage of requiring that an additional equation be included for each inductor or voltage source but has the advantage that ideal voltage sources can be used. The total number of equations to be solved is therefore the number of nodes plus the number of voltage sources and inductors. Modified nodal analysis has an additional advantage since it provides a method of determining currents flowing in certain branches via the insertion of zero voltage sources that function as amp meters. The equations for modified nodal analysis can be assembled on an element-by-element basis just as with nodal analysis. The only difference is that with modified nodal analysis, we need to add new equations and variables for each voltage source or inductor.

ACTIVE DEVICE MODELS

Most circuits of interest contain active devices like transistors or diodes that act as amplifiers. These devices are normally described by a set of nonlinear equations and can become very complex. For our discussion we shall consider a simple model for the bipolar transistor, the Ebers-Moll model. This model is one of the first that was developed, and while it is too simple for practical application it is useful for discussion.

A schematic of the Ebers-Moll model is shown in Fig. 5. The model contains three nonlinear voltage-dependent current sources, I_c, I_{bf}, and I_{br} and two nonlinear capacitances, C_{be} and C_{bc}. The currents flowing in the three current sources are given by the following equations:

$$\begin{aligned}
I_c &= I_s(\exp(V_{be}/V_t) - \exp(V_{ce}/V_t)) \\
I_{bf} &= \frac{I_s}{B_f}(\exp(V_{be}/V_t) - 1) \\
I_{br} &= \frac{I_s}{B_r}(\exp(V_{bc}/V_t) - 1)
\end{aligned}$$

The voltages V_{be} and V_{bc} are the voltages between base and emitter and the base and collector, respectively. I_s, B_f, and B_r are three user-defined parameters that govern the dc operation of the BJT. V_t is the thermal voltage, or kT/q, which has the numerical value of approximately 0.26 V at room temperature. Observe that in the normal forward active mode, where $V_{be} > 0$ and $V_{ce} < 0$, I_{br} and the second term in I_c vanish and the current gain of the BJT, which is defined as I_c/I_b, becomes numerically equal to B_f. Likewise, in the reverse mode, where $V_{ce} > 0$ and $V_{be} < 0$, the reverse gain (I_e/I_b) is equal to B_r.

The Ebers–Moll model has a number of shortcomings. Observe that once we enter the forward mode, the current gain is a constant B_f, which does not depend on the collector current or base collector voltage. Real transistors do not behave this way. In real transistors as V_{ce} becomes more negative, the base collector depletion region consumes more of the base, producing a narrower base with and higher current gain (the early effect). As a result, a plot of I_c versus V_{ce} shows positive

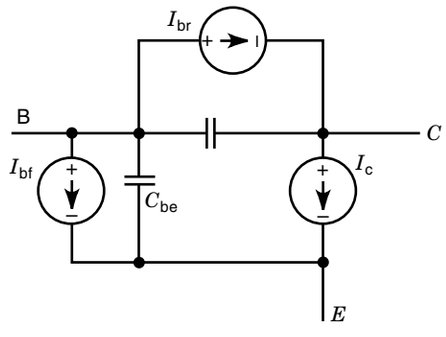

Figure 5. The Ebers–Moll model for the bipolar transistor.

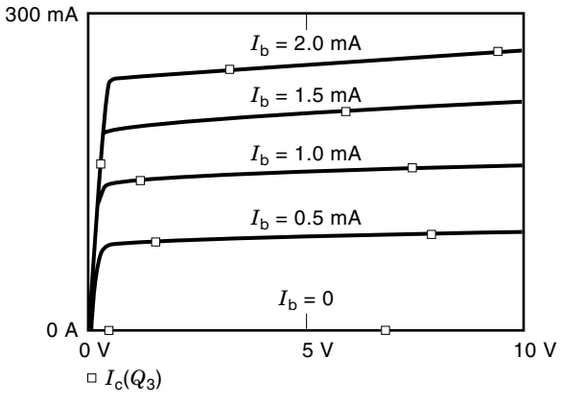

Figure 6. Collector curves for 2N2222 generated using the Gummel-Poon model.

slope in a real device (the collector has finite output resistance; see Fig. 6). In the Ebers–Moll model, however, the slope of the I_c versus V_{ce} curve is zero and therefore the collector resistance is infinite. Real devices also suffer gain degradation at low emitter injection (due to recombination) and at high injection due to base pushout, resulting in gain reduction at both low and high collector current. Notice how the collector curves become compressed as the base current increases in Fig. 6. The Gummel–Poon model was created to address these problems and produces more accurate BJT current–voltage (*IV*) characteristics.

The two capacitances in Fig. 5 contribute charge to the emitter, base, and collector, and this charge is given by the following equations:

$$Q_{be} = \tau_f I_s(\exp(V_{be}/V_t) - 1) + C_{je}\int_0^{V_{be}} (1 - V/V_{je})^{-m_e}\, dv$$

$$Q_{bc} = \tau_r I_s(\exp(V_{bc}/V_t) - 1) + C_{jc}\int_0^{V_{bc}} (1 - V/V_{jc})^{-m_c}\, dv$$

Q_{be} contributes positive charge to the base and negative charge to the emitter. Q_{bc} contributes positive charge to the base and negative charge to the collector. The first term in each charge expression is due to charge injected into the base from the emitter for Q_{be} and from the collector into the base for Q_{bc}. Observe that the exponential terms in the charge terms are identical to the term in I_c. This is so because the injected charge is proportional to the current flowing into the transistor. The terms τ_f and τ_r are the forward and reverse transit times and correspond to the amount of time it takes the electrons (or holes) to cross the base. The second terms in the charge expression (the term with the integral) correspond to the charge in the depletion region of the base-emitter junction for Q_{be} and in the base-collector junction for Q_{bc}. Recall that the depletion width in a *pn* junction is a function of the applied voltage. The terms V_{je} and V_{jc} are the "built-in" potentials with units of volts for the base-emitter and base-collector junctions. The terms m_c and m_e are the grading coefficients for the two junctions and are related to how rapidly the material changes from *n*-type to *p*-type across the junction.

This "simple" model has eleven constants—I_s, B_f, B_r, C_{je}, C_{jc}, M_e, M_c, V_{je}, V_{jc}, T_f, and T_r—that must be specified by the user. Typically these constants would be extracted from measured IV and capacitance–voltage (CV) data taken from real transistors using a fitting or optimization procedure. The Gummel–Poon model, on the other hand, has more than 40 parameters that must be adjusted to get a good fit to data in all regions of operation.

Models for MOS devices are even more complicated than the bipolar models. Modeling the MOSFET is more difficult than the bipolar transistor because it is often necessary to use a different equation for each of the four regions of operation (off, subthreshold, linear, saturation) and the drain current and capacitance are functions of three voltages (V_{ds}, V_{bs}, and V_{gs}) rather than just two (V_{be} and V_{ce}), as in the case of the BJT. If the equations are to be accurate and result in good convergence, the IV characteristics and capacitances must be continuous and it is best if their first derivatives are continuous as well. Furthermore, many MOS models contain the width (W) and length (L) of the MOSFET channel as parameters, and for the best utility the model should remain accurate for many values of W and L. This property is referred to as "scalability."

Many commercial simulators contain other type of models besides the traditional R, L, C, MOS, and BJT devices. Some simulators contain "behavioral" models that are useful for systems design or integration tasks; examples of these are integrators, multipliers, summation, and LaPlace operator blocks. Simulators may also contain prefitted models for commercially available operational amplifiers and logic chips. Some simulators allow "mixed mode" simulation, which is a combination of logic simulation (which normally allows only a few discrete voltage states) with conventional circuit simulation.

TYPES OF ANALYSIS

For analog circuits there are three commonly used methods of analysis: dc, ac, and transient analysis. The dc analysis is used to examine the steady-state operation of a circuit (that is, what the circuit voltages and currents would be if all inputs were held constant for an infinite time). The ac analysis (or sinusoidal steady state) examines circuit performance in the frequency domain using phasor analysis. Transient analysis is analysis in the time domain and is the most computationally intensive of the three.

DC (STEADY-STATE) ANALYSIS

The dc analysis calculates the state of a circuit with fixed (non-time-varying) inputs after an infinite period of time. Such analysis is useful to determine the operating point (*Q*-point) of a circuit, power consumption, regulation and output voltage of power supplies, transfer functions, noise margin and fanout in logic gates, and many other types of analysis. In addition, dc analysis is used to find the starting point for ac and transient analysis. To perform the analysis, the simulator assembles the circuit equations as usual but removes the time-dependent terms from the equations (sets them to zero). This procedure is equivalent to replacing all the capacitors with open circuits and replacing all the inductors with short circuits.

Now we need a method to solve the system of equations. Unfortunately, since the circuit elements will be nonlinear in

most cases, a system of transcendental equations will normally result and it is therefore impossible to solve the system analytically. We therefore resort to a numerical procedure, and the method that has met with the most success is Newton's method or one of its derivatives.

Newton's Method

Newton's method is actually quite simple. The problem is to solve the system of equations $\boldsymbol{F}(\boldsymbol{X}) = 0$ for $\boldsymbol{X}$, where both $\boldsymbol{F}$ and $\boldsymbol{X}$ are vectors of dimension N. $\boldsymbol{F}$ is the system of equations from modified nodal analysis and $\boldsymbol{X}$ is the vector of voltages and current to be solved for. Newton's method states that given an initial guess for $\boldsymbol{X}^i$ we can obtain a better guess $\boldsymbol{X}^{i+1}$ from the equation

$$\boldsymbol{X}^{i+1} = \boldsymbol{X}^i - [\boldsymbol{J}(\boldsymbol{X}^i)]^{-1}\boldsymbol{F}(\boldsymbol{X}^i) \tag{2}$$

Note that all terms on the right side of the equation are functions only of the vector $\boldsymbol{X}^i$. The term $\boldsymbol{J}(\boldsymbol{X})$ is an N by N square matrix of partial derivatives of $\boldsymbol{F}$ called the Jacobian. Each term in $\boldsymbol{J}$ is given by

$$\boldsymbol{J}_{i,j} = \frac{\partial F_i(X)}{\partial X_j}$$

The Jacobian matrix for the circuit is assembled at the same time as the circuit equations. Normally analytic derivatives are used; however, some simulators use numeric derivatives via divided differences instead. This is referred to as the secant method.

The -1 in Eq. (2) indicates that it is necessary to invert the Jacobian matrix before multiplying by the vector $\boldsymbol{F}$. Of course, we do not need actually to invert $\boldsymbol{J}$ to solve the problem; we only need to solve the linear problem $\boldsymbol{F} = \boldsymbol{YJ}$ for the vector $\boldsymbol{Y}$ and then calculate $\boldsymbol{X}^{i+1} = \boldsymbol{X}^i - \boldsymbol{Y}$. A direct method such as lower-upper triangular (LU) decomposition is usually employed to solve the linear system. More details on the structure of $\boldsymbol{J}$ and the inversion process are given later in the section on the Jacobian matrix structure. The equation assembly procedure including the Jacobian matrix becomes

- For each element n{
  - For each terminal j of the element n{
    - Determine the node k to which j is attached.
    - if ($k > 0$){
      - Compute the current at j and sum into G_k
      - For each terminal I of the element n{
        - Compute the derivative for the element:
          $$g_{j,I} = \frac{\partial I_j}{\partial V_I}$$
        - Find the node m which is attached to terminal I
        - Sum in the Jacobian contribution:
          $$J_{k,m} = J_{k,m} + g_{j,I}$$

}}}}

For the small circuit of Fig. 3, analyzed in steady state (without the capacitor), the Jacobian entries are

$$J_{1,1} = 1/R_1 \qquad J_{1,2} = -1/R_1$$
$$J_{2,1} = -1/R_1 + g_\mathrm{m} \qquad J_{2,2} = 1/R_1 + 1/R_2$$

For a passive circuit (i.e., a circuit without gain), the Jacobian will be symmetric and for any row the diagonal entry will be greater than the sum of all the other entries.

Newton's method converges quadratically provided that the initial guess $\boldsymbol{X}^i$ is sufficiently close to the true solution. Quadratically it implies that if the distance between $\boldsymbol{X}^i$ and the true solution is d, then the distance between $\boldsymbol{X}^{i+1}$ and the true solution will be d^2. Of course, this assumes that d is small to start with. Still, programs like SPICE may require 50 or more iterations to achieve convergence because often the initial guess is poor and quadratic convergence is not obtained until the last few iterations. There are additional complications, such as the fact that the model equations can become invalid for certain voltages. For example, the BJT model will "explode" if a junction is forward biased by more than 1 V since $\exp(1/V_\mathrm{t}) = 5e16$. Special limiting or damping methods must be used to keep the voltages and currents to within reasonable limits. (See the section on Convergence Issues.)

EXAMPLE SIMULATION

Most circuit simulators allow the user to ramp one or more voltage sources and plot the voltage at any node or the current in certain branches. Returning to the differential pair of Fig. 1, we can perform a dc analysis by simply adding a `.dc` statement (see Fig. 7). The format for the dc statement is:

```
.dc Vname start stop step
```

where $\mathrm{V_{name}}$ is the source to be swept and the start, stop, and step parameters control the sweep. For the circuit to be valid for dc analysis:

- A dc path to ground must exist from every node in the circuit
- No loops of voltage sources may exist
- No cuts of current sources may exist
- Each node must be connected to at least 2 elements

A plot of the differential output voltage (between the two collectors) and the voltage at the two emitters is shown in

```
V1 4 0 2V
V2 5 0 2V
V3 1 0 5V
R1 2 0 1k
R2 3 1 1K
R3 6 1 1K
Q1 3 4 2 m2n2222
Q2 6 5 2 m2n2222
.model m2n2222 NPN IS=1e-12 BF=100 BR=5 TF=100pS
.dc V1  1.0 4.0 0.01
```

Figure 7. Input file for dc sweep of `V1`.

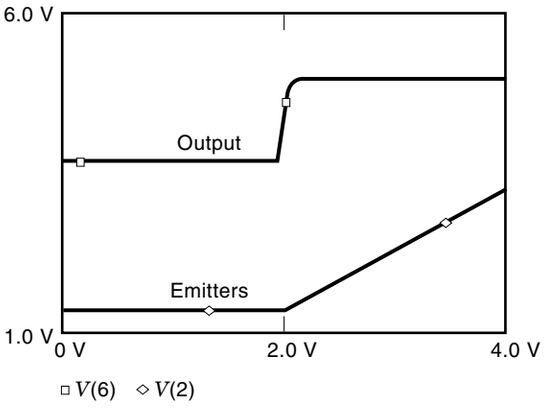

Figure 8. Output from dc analysis.

Fig. 8. Observe that the output voltage is zero when the differential pair is "balanced" with 2.0 V on both inputs. The output saturates at both high and low values for `V1`, illustrating the nonlinear nature of the analysis. This simulation was run using the PSPICE package from MicroSim corporation. The simulation runs in a few seconds on a type 486 PC.

AC ANALYSIS

The ac analysis is performed in the frequency domain under the assumption that all signals are represented as a dc component V_{dc} plus a small sinusoidal component V_{ac}.

$$V = V_{dc} + V_{ac}\exp(j\omega t)$$

Here $j = \sqrt{-1}$, ω is the radial frequency ($2\pi f$), and V_{ac} is a complex number. Expanding Eq. (1) about the dc bias point V_{dc} (also referred to as the Q-point), we obtain

$$\boldsymbol{F}(\boldsymbol{V}) = \boldsymbol{F}(\boldsymbol{V}_{dc}) + \boldsymbol{W}_{dc} + \boldsymbol{W}_{ac} + \frac{\partial \boldsymbol{G}(\boldsymbol{V}_{dc})}{\partial \boldsymbol{\nabla}_{dc}}\boldsymbol{V}_{ac} + \frac{\partial}{\partial t}\left(\frac{\partial \boldsymbol{Q}(\boldsymbol{V}_{dc})}{\partial \boldsymbol{V}_{dc}}\right)\boldsymbol{V}_{ac} + \alpha \boldsymbol{V}_{ac}^2$$

The series has an infinite number of terms; however, if $\boldsymbol{V}_{ac}$ is sufficiently small, all terms above first order can be neglected. The first two terms on the right-hand side are the dc solution and when taken together yield zero. The third term $\boldsymbol{W}_{ac}$ is the vector of independent ac current sources that drive the circuit. The partial derivative in the fourth term is the dc Jacobian element and the derivative of $\boldsymbol{Q}$ in parentheses is the capacitance at the node. When we substitute the exponential into the preceding equation each term will have an exponential term that can be canceled. The result of all these simplifications is the familiar result

$$0 = \boldsymbol{W}_{ac} + \boldsymbol{J}\boldsymbol{V}_{ac} + j\omega \boldsymbol{C}\boldsymbol{V}_{ac}$$

This equation contains only linear terms that are equal to the partial derivatives of the original problem evaluated at the Q-point. Therefore, before we can solve the ac problem, we must calculate the dc bias point. Rearranging terms slightly, we obtain

$$\boldsymbol{V}_{ac} = -(\boldsymbol{J} + j\omega \boldsymbol{C})^{-1}\boldsymbol{W}_{ac}$$

The solution at a given frequency can be obtained from a single matrix inversion. The matrix, however, is complex but normally the complex terms share a sparsity pattern similar to the real terms. (See the section on the Jacobian structure.) It is normally possible (in FORTRAN and C++) to create a suitable linear solver by taking the linear solver that is used to calculate the dc solution and substituting "complex" variables for "real" variables. Since there is no nonlinear iteration, there are no convergence problems and ac analysis is straightforward and foolproof. The same type of analysis can be applied to the equations for modified nodal analysis. The additional unknowns will, of course, be currents flowing through the voltage sources:

$$\boldsymbol{I}_{ac} = -(\boldsymbol{J} + j\omega \boldsymbol{L})^{-1}\boldsymbol{E}_{ac}$$

The only things that must be remembered with ac analysis are the following:

1. The ac solution is sensitive to the Q-point, so if an amplifier is biased near its saturated dc output level the ac gain will be smaller than if the amplifier were biased near the center of its range.
2. This is a linear analysis, and therefore "clipping" and slew rate effects are not modeled. For example, if a 1 V ac signal is applied to the input of a small signal amplifier with a gain of 100 and a power supply voltage of 5 V, ac analysis will predict an output voltage of 100 V. This is, of course, impossible since the output voltage cannot exceed the power supply voltage of 5 V. Transient analysis should be used to include these effects.

AC ANALYSIS EXAMPLE

In the following example we will analyze the differential pair using ac analysis to determine its frequency response. To perform this analysis in SPICE we need only specify which sources are the ac driving sources (by adding the magnitude of the ac signal at the end) and specify the frequency range on the `.AC` statement (Fig. 9). SPICE lets the user specify

```
V1 4 0 2V  AC 1
V2 5 0 2V
V3 1 0 5V
R1 2 0 1k
R2 3 1 1K
R3 6 1 1K
Q1 3 4 2 m2n2222
Q2 6 5 2 m2n2222
.model m2n2222 NPN IS=1e-12 BF=100 BR=5 TF=100pS
.AC DEC 10 1e3 1e9
```

Figure 9. Input file for ac analysis.

the range as linear or "decade," indicating that we desire a logarithmic frequency scale. The first number is the number of frequency points per decade, the second number is the starting frequency, and the third is the ending frequency.

Figure 10 shows the results of the analysis. The gain begins to roll off at about 30 MHz due to the parasitic capacitances within the transistor models. The input impedance (which is plotted in kilohms) begins to roll off at a much lower frequency. The reduction in input impedance is due to the increasing current that flows in the base-emitter capacitance as the current increases. SPICE does not have a method of calculating input impedance, so we have calculated it as $Z = V_{\text{in}}/I(V_{\text{in}})$, where $V_{\text{in}} = 1.0$, using the postprocessing capability of PSPICE. This analysis took about 2 s on a 486-type PC.

Noise Analysis

Noise is a problem in circuits that are designed for the amplification of small signals, like the radio frequency (RF) and intermediate frequency (IF) amplifiers of a receiver. Noise is the result of random fluctuations in the currents that flow in the circuit and is generated every circuit element. In circuit simulation, noise analysis is an extension of ac analysis. During noise analysis it is assumed that every circuit element contributes some small noise component, either as a voltage V_n in series with the element or as a current I_n across the element. Since the noise sources are small in comparison with the dc signal levels, ac small-signal analysis is an appropriate analysis method.

Different models have been developed for the noise sources. In a resistor thermal noise is the most important component. Thermal noise is due to the random motion of the electrons:

$$I_n^2 = \frac{4kT\Delta f}{R}$$

where T is the temperature, k is Boltzmann's constant, and Δf is the bandwidth of the circuit. In a semiconductor diode shot noise is important. Shot noise is related to the probability that an electron will surmount the semiconductor barrier energy and be transported across the junction:

$$I_n^2 = 2qI_d\Delta f$$

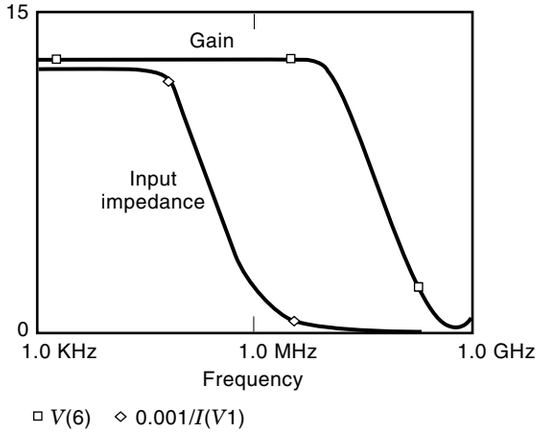

Figure 10. Gain and input impedance calculated by ac analysis.

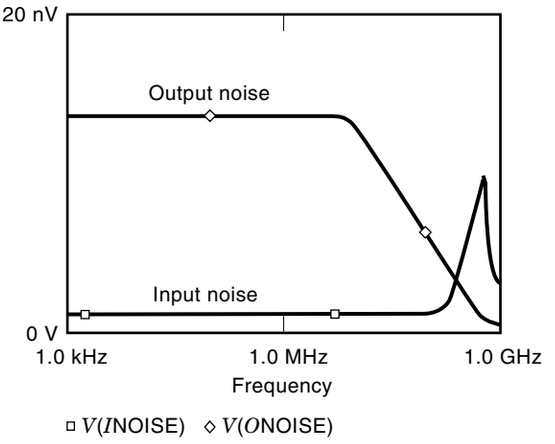

Figure 11. Noise referenced to output and input.

There are other types of noise that occur in diodes and transistors (examples are flicker and popcorn noise). Noise sources, in general, are frequency dependent.

Noise signals will be amplified or attenuated as they pass through different parts of the circuit. Normally, noise is referenced at an output point called the "summing node." This would normally be the output of the amplifier where we would actually measure the noise. The gain between the summing node and the current flowing in an element j in the circuit is defined as $A_j(f)$. Here f is the analysis frequency since this gain will normally be frequency dependent.

Noise signals are random and uncorrelated to each other, so their magnitudes must be root-mean-square summed rather than simply summed. Summing all noise sources in a circuit yields

$$I_n(f) = \sqrt{\sum_j A_j^2(f) I_j^2(f)}$$

It is also common to reference noise back to the amplifier input, and this is easily calculated by dividing the preceding expression by the amplifier gain. Specifying noise analysis in SPICE is simple. All the user needs to do is add a statement specifying the summing node and the input source. SPICE then calculates the noise at each as a function of frequency

```
.noise V(6) V1
```

See Fig. 11 for example output. Many circuit simulators will also list the noise contributions of each element as part of the output. This is particularly helpful in locating the source of noise problems.

TRANSIENT ANALYSIS

Transient analysis is the most powerful analysis capability because the transient response of a circuit is so difficult to calculate analytically. Transient analysis can be used for many types of analysis, such as switching speed, distortion, and checking the operation of circuits like logic gates, oscillators, phase-locked loops, or switching power supplies. Transient analysis is also the most CPU (central processing unit)

intensive and can require 100 or 1000 times the CPU time of dc or ac analysis.

Numerical Integration Methods

In transient analysis time is discretized into intervals called timesteps. Typically the timesteps are of unequal length, with the smallest steps being taken during intervals where the circuit voltages and current are changing most rapidly. The following procedure is used to discretize the time-dependent terms in Eq. 1.

Time derivatives are replaced by difference operators, the simplest of which is the forward difference operator (also known as the forward Euler method):

$$\frac{\partial Q(t_k)}{\partial t} = \frac{Q(t_{k+1}) - Q(t_k)}{h}$$

where the timestep h is given by

$$h = t_{k+1} - t_k$$

This equation is easily solved for the charge $Q(t_{k+1})$ at the next time point:

$$\boldsymbol{Q}(t_{k+1}) = \boldsymbol{Q}(t_k) - h(\boldsymbol{G}_i(\boldsymbol{V}(t_k)) + \boldsymbol{W}_i(t_k))$$

using only values from past time points. This means that it would be possible to solve the system simply by plugging in the updated values for $\boldsymbol{V}$ each time. This can be done without any matrix assembly or inversion and appears to be very efficient. (Note that for simple linear capacitors, $V = Q/C$ at each node, so it is easy to get V back from Q). However, this approach is undesirable for circuit simulation for two reasons: (1) The charge $\boldsymbol{Q}$, which is a "state variable" of the system, is not a convenient choice since some nodes may not have capacitors (or inductors) attached, in which case they will not have Q values; (2) a more serious problem is that forward (or explicit) time discretization methods like this one are unstable for "stiff" systems, and most circuit problems result in stiff systems. The term *stiff system* refers to a system that has greatly varying time constants.

To overcome the stiffness problem, we must use implicit time discretization methods, which in essence means that the $\boldsymbol{G}$ and $\boldsymbol{W}$ terms in the preceding equations must be evaluated at t_{k+1}. Since $\boldsymbol{G}$ is nonlinear we will need to use Newton's method once again.

The most popular implicit method is the trapezoidal method. The trapezoidal method has the advantage of only requiring information from one past time point, and furthermore it has the smallest error of any method requiring one past time point. The trapezoidal method states that if I_c is the current in a capacitor, then

$$I_c^{(t_{k+1})} = \frac{\partial Q}{\partial t} = 2\frac{Q(V_c(t_{k+1})) - Q(V_c(t_k))}{t_{k+1} - t_k} - I_c(t_k)$$

Therefore, we need only substitute the preceding equation into Eq. (1) to solve the transient problem.

Observe that we are solving for the voltages $V(t_{k+1})$, and all terms involving t_k are constant and will not be included in the Jacobian matrix. An equivalent electrical model for the capacitor is shown in Fig. 12. Therefore, the solution of the transient problem is, in effect, a series of dc solutions, where the values of some of the elements depend on voltages from the previous time points.

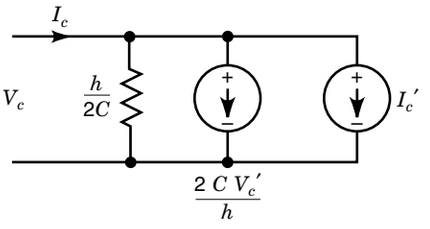

Figure 12. Electrical model for a capacitor. The two current sources are independent sources. The prime indicates values from preceding time point.

The procedure for transient analysis is as follows:

- Compute dc solution to provide initial conditions
- Repeat until (t>tstop){
  - Compute time step h based on LTE
  - Repeat until converged{
    - For each element n{
      - For each terminal j of the element n{
        - Determine the node k to which j is attached.
        - if (k>0){
          - Compute $Ic_j^{k+1}=\frac{2}{h}(Q_j(V(t_{k+1}))-Q_j(V(t_k)))$
          - Sum into current vector: $F_m=I_j+Ic_j^{k+1}-Ic_j^k+W_j(t_{k+1})$
          - Compute the current at terminal j and sum into G_k
          - For each terminal I of the element n {
            - Compute the derivative for the element: $g_{j,1}=\frac{\partial I_j(V(t_k))}{\partial V_1}+\frac{2}{h}\frac{\partial Q_j(V(t_k))}{\partial V_1}$
            - Find the node m which is attached to terminal I
            - Sum in the Jacobian contribution: $J_{k,m}=J_{k,m}+g_{j,1}$

        }}}
        Solve $X=JF$
        $V=V-X$
    }
    $t=t+h$
}

All modern circuit simulators feature automatic timestep control. This feature selects small timesteps during intervals where changes are occurring rapidly and large timesteps in intervals where there is little change. The most commonly used method of timestep selection is based on the local trun-

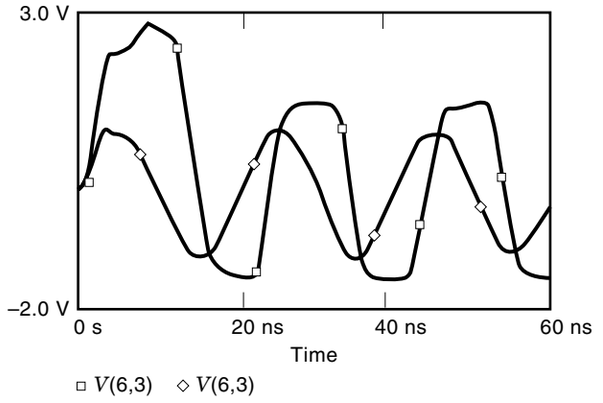

Figure 13. Transient response $V(6,3)$ of differential amplifier to sinusoidal input at $V(4,5)$.

cation error (LTE) for each timestep. For the trapezoidal rule, the LTE is given by

$$\epsilon = \frac{h}{12}\frac{d^3x}{dt^3}(\xi)$$

and represents the maximum error introduced by the trapezoidal method at each timestep. The trapezoidal method is a second-order method since the error is proportional to the timestep cubed. The LTE for a time point can only be computed after the solution at that point is known. LTE timestep control methods calculate the timestep for the next time point from the LTE at the preceding time point by assuming that the error ϵ at each step to be as close to a certain small fixed value (usually 0.1%) as possible.

$$h = \sqrt{12\epsilon \Big/ \frac{d^3x}{dt^3}}$$

(We are assuming that the required h does not change much from timestep to timestep.) If, after computing the solution at a new time point, we find that the LTE at that point is very large (say, five times the error criteria) the simulator will reduce the timestep (usually by 1/2) and go back and recompute the point. In addition, most simulators select time points so that they coincide with the edges of pulse-type waveforms.

TRANSIENT ANALYSIS EXAMPLES

As a simple example, we return to the differential pair and apply a sine wave differentially to the input (Fig. 13). The amplitude (2 V peak to peak) is selected drive the amplifier into saturation. In addition, the frequency (50 MHz) is set high enough to see phase shift effects. The output signal is therefore clipped due to the nonlinearities and shifted in phase due to the capacitive elements in the transistor models. The first cycle shows extra distortion since it takes time for the "zero-state" response to die out. This simulation runs in about a second on a type-486 computer.

As a final example, a phase-locked loop (PLL) circuit, which is used as a 4× frequency multiplier, is simulated.This circuit is shown in Fig. 14 and includes digital, analog, and behavioral components. This particular circuit is designed primarily to illustrate the capabilities of simulation. Phase-locked loops are difficult circuits to simulate since they can have greatly varying frequency components. The input to the circuit is a 1 MHz signal (at node 1) and the output is a 4 MHz signal at node 9. Phase-locked loops have three main components: a voltage controlled oscillatory (VCO), a phase detector, and a loop filter. In this circuit the VCO is made up of a summer, an integrator, and a ideal sine element, which simply computes the sine of its input. Resistors R_2 and R_3 form a voltage divider and set the voltage at node 8 to 2.5 V. The signal at node 9 of the VCO is therefore

$$V(9) = \sin\left(10^7 \times \int_0^t (V(5) + 2.5) \times dt\right)$$
$$= \sin(10^7 \times (V(5) + 2.5) \times t)$$

and $V(5)$ is the VCO control voltage. Note that PSPICE generates the integrator from a capacitor, a voltage-controlled current source (G) and a voltage-controlled voltage source (E).

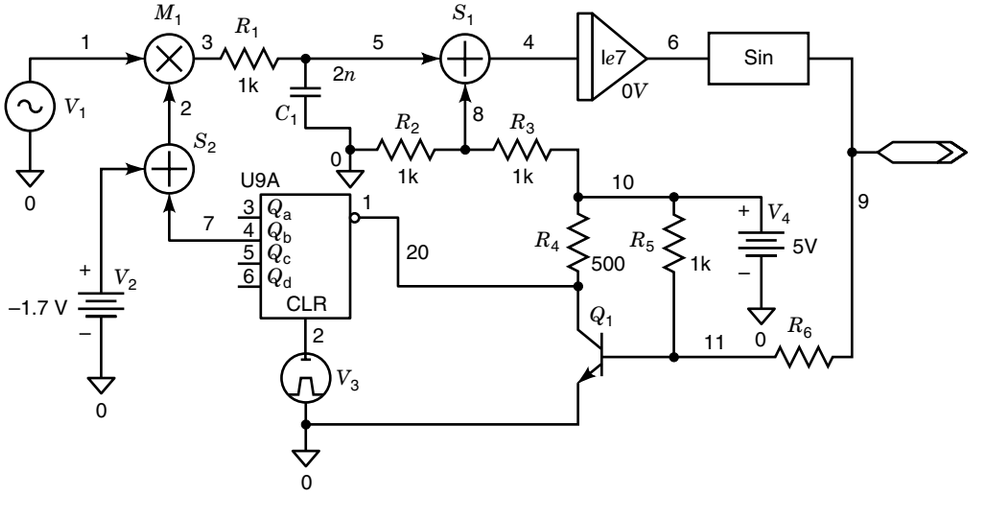

Figure 14. Phase-locked loop circuit.

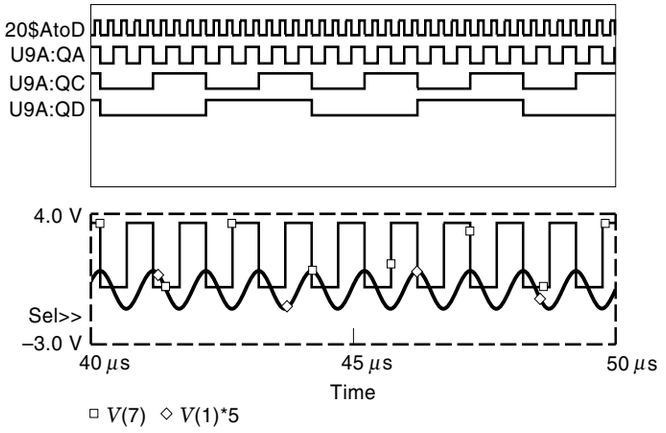

Figure 15. Outputs from the PLL simulation.

The network of Q_1, R_4, R_5, and R_6 is an overdriven amplifier that converts the sinusoidal signal at $V(9)$ into a square wave for driving U9A. Integrated circuit U9A is a 4-bit binary counter. Voltage source $V(3)$ applies a pulse at the start of the simulation to UA9's Clear (CLR) input to zero the counters. Since the output is taken at Q_b, the signal at $V(7)$ is at one fourth the frequency of $V(9)$. The "free running" frequency of the signal at $V(7)$ is therefore $2.5\epsilon 7/2\pi$ or 0.9947 MHz. The digital output at $V(7)$ has $V_h = 3.6$ *V* and $V_I = 0.2$ V. Summer S_2 is therefore used to shift the dc value of $V(7)$ back to zero volts for application to the phase detector. The phase detector is a simple ideal multiplier (M_1). The low-pass loop filter is made up of resistor R_1 and capacitor C_1.

The output from the PLL simulation is shown in Figs. 15 and 16. Figure 15 shows waveforms from the last 10 μs of the 50 μs simulation. The bottom figure shows the signal from U9A (the binary counter and the input signal at node 1). It can be seen that the two signals are locked in phase as they should be. The top figure shows the digital waveforms at input and outputs of U9A. It can be seen that the binary counter is operating correctly. The model for U9A is a digital block with "high," "low," and "undetermined" signal levels rather than analog signal levels (The U9A model contains no transistors.) Using digital models is much faster than using the analog equivalent (which in this case would require close to 100 transistors). The ability to mix analog and digital blocks is known as "mixed-mode" simulation. Special conversion operators are applied at the inputs and outputs of U9A to convert the analog input to a digital input and the digital output to an appropriate analog signal.

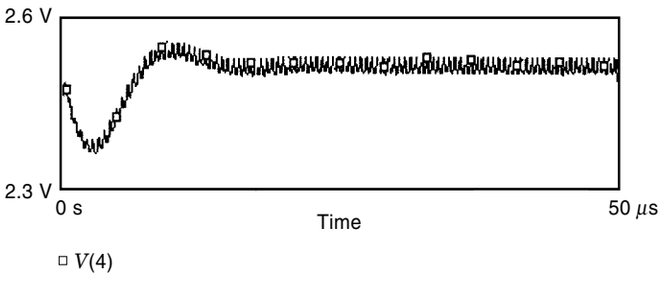

Figure 16. PLL output from loop filter.

Figure 16 shows the signal from output of Summer S_1. The PLL capture transient can be seen. Since the signals at $V(1)$ and $V(7)$ are initially out of phase and at slightly different frequencies, it takes some time for the PLL to lock onto the input signal. The "noise" on $V(4)$ is the high-frequency output from the multiplier ($V(1)*V(7)$). The PLL simulation requires about 5 min of CPU time on a type-486 computer.

CONVERGENCE ISSUES

Considering that a circuit can contain thousands of nodes requiring the solution of thousands of simultaneous nonlinear equations, it is amazing that Newton's method can converge at all. Unfortunately, there are many cases when Newton's method needs help and we will address a few methods here.

Many convergence problems are caused by an initial guess that lies far from the true solution. One way of improving the initial guess is through projection. Projection does just as its name implies and uses two previous solutions to calculate an initial guess. In time-dependent simulations we can use the following linear projection:

$$V^{i+1} = V^i + \frac{(V^i - V^{i-1})(t^{i+1} - t^i)}{t^i - t^{i-1}}$$

Higher-order methods can also be used, but linear projection is the safest. A variant method uses the Jacobian matrix and has the advantage that it can be used when only one past solution is available, but it is more work to program and compute:

$$V^{i+1} = V^i + J(V^i)^{-1} \frac{\partial F}{\partial t} (t^{i+1} - t^i)$$

Projection can also be used for calculating dc solutions by combining it with the continuation method, discussed next.

Computing the first dc solution to a fully powered is a difficult task. It would be much easier if we could start from some known solution and gradually move to the true solution. Techniques that do this are referred to as continuation methods. A common method is source stepping, which in effect ramps up the power supplies and gradually turns the circuit on. The method is as follows:

Source Stepping Algorithm

1. Modify all independent voltage and current sources so that their value is the original value multiplied by the continuation constant k^i. Here the superscript i is the continuation iteration counter.
2. Start with $k^0 = 0$ so that all independent voltage and current sources set to zero. The solution to this problem will be that all voltages and currents everywhere must also be zero.
3. Set $k^1 = 0.1$ or other small number and solve using the solution from step k^0 as the initial guess.
4. If the Newton sequence converges, set $i = i + 1$, increase k, and solve again using the solution from k^{i-1} as the initial guess. If the Newton sequence does not converge, reduce k to a smaller value (usually (k^i +

$k^{i-1})/2$) and try again. The projection techniques described previously can be applied simply by replacing t with k.

5. Repeat step 4 until $k = 1$, at which point all the dependent voltage and current sources will be at their correct values.

While it would appear that this method should be foolproof, it can be shown that source stepping will fail for certain circuits. Consider a circuit consisting of a voltage source driving an element with an IV characteristic similar to that of Fig. 17. The locus of solutions is the intersection of the vertical line in the figure with the curve. Convergence would typically fail at point A since the solution jumps abruptly to point B. The only way to proceed beyond point A is to convert the voltage source to a current source and continue on by stepping in current. This would, in effect, change the vertical line to a horizontal line, in which case there are no abrupt transitions.

There is no method of solving nonlinear equations that is always guaranteed to work. A particularly serious convergence problem is caused by bad conditioning of the Jacobian. A matrix is badly conditioned if its determinate is close to zero. Since the inverse of a matrix is related to the inverse of its determinate, if the Jacobian is badly conditioned small changes in $F(V)$ result in very large changes in V, which can make convergence difficult. Bad conditioning results from nodes that are isolated or nearly isolated. Consider the circuit in Fig. 18, which contains two resistors. The Jacobian matrix is

$$\begin{matrix} 1/R_1 & -1/R_1 \\ -1/R_1 & (1/R_1 + 1/R_2) \end{matrix}$$

As R_2 approaches infinity, the two rows become identical and (or, more precisely, linearly dependent) and the determinate that is $1/(R_1R_2)$ becomes zero. Another way of looking at the problem is that with infinite R_2, R_2 disappears and nodes 1 and 2 can be assigned any voltage (as long as both are the same) and a valid solution will result.

An example of a real circuit that exhibits bad conditioning is the CMOS logic gate shown in Fig. 18. Node 1 is attached to the gates of the second invertor and the drains of the first. In a dc simulation, the MOSFET gates that have only capacitance function as an open circuit. If the power supplies and

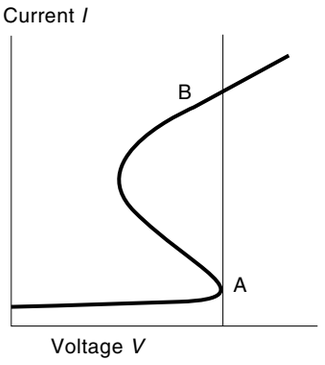

Figure 17. A case that will not converge.

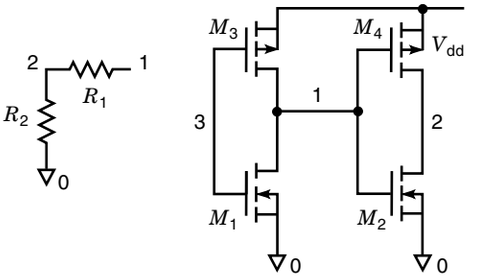

Figure 18. Two badly conditioned circuits.

all nodal voltages are set to zero (as we would do for source stepping), all the MOS devices would be biased with $V_{gs} = 0$ or $V_{gs} < V_{th}$ (i.e., in the off state). In many MOS models, if the device is off, the drain current and the drain conductance ($\partial I_d/\partial V_{ds}$) are zero. This causes node 1 to be isolated and causes bad conditioning, which can make it impossible to find a solution.

Observe that improving the initial guess will not help in the case of convergence failure due to bad conditioning. A possible solution might be to improve the accuracy of the linear solver by the use of pivoting or scaling schemes, but there is usually a CPU penalty involved. A common solution is to connect a small conductance from every node of the circuit to ground, thereby eliminating isolated nodes. However, if the conductance is made too large, it effectively alters the circuit, which can produce incorrect results (in SPICE this conductance is referred to as G_{min}). For the CMOS circuit, once the circuit is biased at full power, at least one of the MOSFETs will be in the conducting state and node 1 will no longer be isolated. This suggests that if we use the source-stepping algorithm, we could use a large conductance during the early phase when k is small and nodes may be isolated and gradually reduce G_{min} as k approaches 1. Note, however, that some CMOS circuits (for example, those containing a transmission gate) can have isolated nodes even with full power applied.

THE JACOBIAN MATRIX STRUCTURE AND LINEAR SOLUTION

The form of the Jacobian matrix of a circuit depends on how the nodes are connected by the circuit elements. Each element introduces a "dependency"—that is, if an element connects node i to node j then the current at node j depends on the voltage at node i and vice versa. Therefore, an element $J_{i,j}$ will be nonzero only when there is an element connected between nodes i and j.

The exception to this rule are the diagonal entries that indicate how the current at a node depends on its own voltage. In most cases, the diagonal entries will be nonzero. A circuit element with two terminals such as a resistor or capacitor will make four nonzero contributions to the Jacobian matrix. If the resistor is connected bewteen nodes i and j, it will contribute a conductance of $-G = -1/R$ to the Jacobian entries $J_{i,j}$ and $J_{j,i}$ and will contribute $G = 1/R$ to the diagonal entries $J_{i,i}$ and $J_{j,j}$. An element with three terminals, like a bipolar transistor, will contribute to nine entries in the Jacobian (three on the diagonal), and a MOSFET with four terminals will contribute to 16 Jacobian entries.

Most circuits do not have elements connecting every node to every other node. Typically, as circuits grow larger the ratio of connections at each node to the number of total nodes becomes smaller. As a result in most circuits the Jacobian matrix is very sparse, meaning that most of its entries are zero. Due to this sparseness, the number of nonzero elements normally depends linearly on the number of total nodes rather than quadratically, as would be expected for a full matrix. Circuit simulation programs employ a sparse matrix storage system that only stores the nonzero elements. Typical systems use three vectors. The first vector ***A*** contains the floating-point nonzero entries in the matrix. The second vector ***IA*** contains the integer row indices. The third vector ***JA*** contains pointers to the location of the start of each column in ***IA***. Consider the following example matrix:

$$\begin{matrix} 1.0 & 3.0 & 0 & 0 \\ 0 & 4.0 & 0 & 9.0 \\ 0 & 0 & 7.0 & 6.0 \\ 5.0 & 0 & 0 & 8.0 \end{matrix}$$

The vector ***A*** contains [1.0, 5.0, 3.0, 4.0, 7.0, 9.0, 6.0, 8.0]. The vector ***IA*** contains [1, 4, 1, 2, 3, 2, 3, 4], and ***JA*** contains [1, 3, 5, 6, 9]. There are other storage systems (for example, some programs store the matrix row wise rather than column wise; it is also common to store the diagonal entries in a separate floating-point array or force the matrix to be symmetric by padding with zeros and then only use pointers to the upper 1/2).

Solving the linear problem $Ax = b$ for a full matrix is normally performed using the LU decomposition followed by forward and back solve operations. The LU decomposition factors the matrix A into the product of two matrices L and U. L is lower triangular, meaning that all entries above the diagonal are zero. U is upper triangular meaning that all its entries below the diagonal are zero. Thus $Ax = b$ becomes $LUx = b$ or $Ly = b$ followed by $Ux = y$. Since L and U are triangular, these are easy to solve. The code for a full matrix is easy to program for the LU decomposition:

```
for(i=1 to n-1 ) {
      for(j=i+1 to n) {
            a(j,i)=a(j,i)/a(i,i)
      }
      for(j=i+1 to n) {
            for(k=i+1 to n) {
                  a(j,k)=a(j,k)-a(j,i)*a(i,k)
            }
      }
}
```

The preceding code is Dolittle's method and computes the LU decomposition "in place," meaning that the computes L and U are written over the original elements in A. The forward solve is

```
for(i=1 to n-1) {
      for (j=i+1 to n) {
            b(j)=b(j)-a(j,i)b(i)
      }
}
```

And the back solve is

```
for (i=n to 2){
      b(i)=b(i)/a(i,i)
      for (j=1 to i-1) {
            b(j)=b(j)-a(j,i)b(i)
      }
}
b(1)=b(1)/a(1,1)
```

The forward and back solve are also done "in place" so that when the process is finished, x resides in b. Observe that the decomposition step can result in the creation of nonzero entries where previously there were zeros [For example, if $a(i, j) = 0$ but $a(j, i)$ and $a(i, k)$ are not zero.] These newly generated elements can generate additional nonzero elements, causing the nonzeros to propagate. This process is referred to as creating "fill" and occurs only within the bandwidth of the matrix A (i.e., new nonzeros are generated only at locations between existing elements in A and the diagonal). The fill terms can increase the computer memory and CPU requirements. Fortunately, the bandwidth of the matrix and resulting fill can be reduced by proper reordering of the circuit equations. The number of nonzero entries in the Jacobian and result of the solution is independent of how the nodes are numbered (assuming ideal arithmetic), but the locations of the nonzero entries will depend on the node ordering. A number of algorithms exist for this renumbering process.

Double precision arithmetic is used throughout most circuit simulators. In some cases it may be desirable to use partial pivoting to improve the accuracy of the linear solution. Recent research has focused on the use of iterative linear solvers instead of the directed solver, which was outlined previously. The simplest of these is Gauss-Sidel, but modern versions often used preconditioned gradient or minimization techniques. Iterative solvers may be faster, more accurate, and use less memory than the direct solver but can introduce additional convergence problems of their own since now iteraction is involved in the linear as well as nonlinear solution. Iterative solvers are often more sensitive to bad conditioning of the Jacobian matrix than the LU decomposition.

FAST SIMULATION METHODS

As circuits get larger simulation times become longer. In addition, as integrated circuit feature sizes shrink, second-order effects become more important, and many circuit designers would like to be able to simulate large digital systems at the transistor level (requiring 10,000 to 100,000 nodes). Numerical studies in early versions of SPICE showed that the linear solution time could be reduced to 26% for relatively small circuits with careful coding. The remainder is used during the assembly of the matrix, primarily for model evaluation. The same studies found that the CPU time for the matrix solution was proportional to $n^{1.24}$, where n is the number of nodes. The matrix assembly time, on the other hand, should increase linearly with node count. Circuits have since grown much bigger, but the models (particularly for MOS devices) have also become more complicated.

Matrix assembly time can be reduced by a number of methods. One method is to simplify the models; however, ac-

curacy will be lost as well. A better way is to precompute the charge and current characteristics for the complicated models and store them in tables. During simulation the actual current and charges can be found from table lookup and interpolation, which can be done quickly and efficiently. However, there are some problems:

1. To ensure convergence of Newton's method, both the charge and current functions and their derivatives must be continuous. This rules out most simple interpolation schemes and means that something like a cubic spline must be used.
2. The tables can become large. A MOS device has four terminals, which means that all tables will be functions of three independent variables. In addition, the MOSFET requires four separate tables (I_d, Q_g, Q_d, Q_b). If we are lucky, we can account for some parametric variations (like channel width) by a simple multiplying factor. However, if there are more complex dependencies, as is the case with channel length, oxide thickness, temperature, or device type, we will need one complete set of tables for each device.

If the voltages applied to an element do not change from the past iteration to the present iteration, then there is no need to recompute the element currents, charges, and their derivatives. This method is referred to as bypass or taking advantage of latency and can result in large CPU time savings in logic circuits, particularly if coupled with a method that refactors only part of the Jacobian matrix. The tricky part is knowing when the changes in voltage can be ignored. Consider, for example, the input to a high gain op amp. Here ignoring a microvolt change at the input could result in a large error at the output. Use of sophisticated latency determining methods could also cut into the CPU time savings.

Another set of methods are the waveform relaxation techniques, which increase efficiency by temporarily ignoring couplings between nodes. The simplest version of the method is the Gauss–Seidel method which is as follows. Consider a circuit with n nodes that requires m time points for its solution. The circuit can be represented by the vector equation

$$F_i(V(t)) + \frac{dQ_i(V(t))}{dt} = 0$$

Using trapezoidal time integration gives a new function.

$$W_i(V(k)) = (F_i(V(k)) + F_i(V(k-1))) \cdot b + 2[Q_i(V(k)) - Q_i(V(k-1))] = 0$$

We need to find the $V(k)$ that makes W zero for all $k = 1$ to m time points at all $i = 1$ to n nodes. The normal method solves for all n nodes simultaneously at each time point before advancing k. Waveform relaxation solves for all m time points at a single node (calculates the waveform at that node) before advancing to the next node. An outer loop ensures that all the individual nodal waveforms are consistent with each other (global convergence):

```
Iterate on j global convergence is obtained{
    for node i=1 to n{
            for timepoint k=1 to m{
                    Iterate on l until con-
                     vergence is obtained{
V_i^j(k,j) = V_i^{j-1}(k,j)
        - W_i(V_1(k,j-1), V_2(k,j-1); V_i^{j-1}(k,j), V_n(k,j-1)) / ∂W/∂V_i(k,j)
}}}}
```

Observe that the innermost loop (on l) uses Newton's method to solve the nonlinear equation. However, only a single variable is solved for. In addition, W and its derivative are functions only of $(V_i^{j-1}(k, j))$; all the other terms are constant during iteration on l. Waveform relaxation is extremely efficient as long as the number of outer loops (on j) is small. The number of iterations of j will be small if the equations are solved in the correct order (that is, starting on nodes that are signal sources and following the direction of signal propagation through the circuit). This way the waveform at node $i + 1$ will depend strongly on the waveform at node i, but the waveform at node i will depend weakly on the signal at node $i + 1$. The method is particularly effective if signal propagation is unidirectional, as is sometimes the case in logic circuits. During practical implementation, the total simulation interval is divided into several subintervals and the subintervals are solved sequentially. This reduces the total number of time points that must be stored in central memory. Variants of the method solve small numbers of tightly coupled nodes as a group; such a group might include all the nodes in a transistor–transistor logic (TTL) gate or in a small feedback loop. Large feedback loops can be handled by making the simulation time for each subinterval less than the time required for a signal to propagate around the loop. The efficiency of this method can be further improved by using different timesteps (h) at different nodes yielding a multirate method. This way during a given interval, small time steps are used on active nodes, whereas long steps are used at inactive nodes. Waveforms at nodes may also be computed in parallel on a multiprocessor computer since waveforms at iteration j depend only on voltages at other nodes from iteration $j-1$ (the $i = 1$ to n loop can be parallelized).

COMMERCIALLY AVAILABLE SIMULATORS

The simulations in this article were performed with the evaluation version of PSPICE from Microsim. The following vendors market circuit simulation software. The different programs have strengths in different areas, and most vendors allow users to try their software in-house for an "evaluation period" before they buy.

SPICE2-SPICE3, University of California Berkeley, CA
PSPICE, Microsim Corporation, Irvine, CA
HSPICE, MetaSoftware, Campbell, CA
ISPICE, Intusoft, San Pedro, CA
SABER, Analogy, Beaverton, OR
SPECTRE, Cadence Design Systems, San Jose, CA

TIMEMILL, Epic Corporation, Sunnyvale, CA
ACCUSIM II, Mentor Graphics, Wilsonville, OR

BIBLIOGRAPHY

On General Circuit Simulation

J. A. Connelley and P. Choi, *Macromodeling with SPICE,* Engelwood Cliffs, NJ: Prentice-Hall, 1992.

P. Gray and R. Meyer, *Analysis and Design of Analog Integrated Circuits,* New York: Wiley, 1977.

K. Kundert, *The Designers Guide to SPICE and SPECTRE,* Kulwer Academic, 1995.

L. Nagel, *SPICE2: A Computer Program to Simulate Semiconductor Circuits,* Ph.D. Thesis, University of California, Berkeley, 1975.

A. E. Ruehli (ed.), *Circuit Analysis, Simulation and Design, Part 1,* North Holland: Elsevier Science Publishers, 1981.

P. W. Tuinenga, *SPICE, A Guide to Circuit Simulation and Analysis Using PSPICE,* Englewood Cliffs, NJ: Prentice-Hall, 1988.

J. Vlach and K. Singhal, *Computer Methods for Circuit Analysis and Design,* New York: Van Norstand Reinhold, 1983.

A. Vladimiresch, *The SPICE Book,* New York: Wiley, 1994.

On Modern Techniques

A. E. Ruehli (ed.), *Circuit Analysis, Simulation and Design, Part 2,* North Holland: Elsevier Science Publishers, 1981.

On Device Models

P. Antognetti and G. Massobrio, Semiconductor Modeling with SPICE2, 2 ed. New York: McGraw-Hill, 1993.

J. GREGORY ROLLINS
MTS, Antrim Design Systems