# Chapter 2
# DC Analysis

## 2.1 Introduction

Many of the analyses available for SPICE and Spectre compute operating points. For example, SPICE's `.op` analysis computes and outputs information about the operating point such as the voltage, current, power, etc. at each component. The `.dc` analysis computes the operating point as a function of some independent variable. The linear small signals analyses, such as `.ac` and `.noise`, first compute the operating point and then linearize the circuit about that operating point before computing the small-signal behavior of the circuit. Finally, the transient analysis computes an operating point for the initial state of the circuit. Clearly the ability to reliably compute an accurate operating point is very important.

In general operating points are simply snap-shots of some solution trajectory. In DC analysis, the operating points are also assumed to be equilibrium points. Equilibrium points are constant-valued operating points. In other words, equilibrium points are solutions that do not change with time. A circuit cannot reach an equilibrium point if the stimulus is still changing, so the first step of a DC analysis is to configure the independent sources so they are constant. In addition, since all waveforms are constant-valued at equilibrium points, $dv/dt = 0$ and $di/dt = 0$ and so capacitors act as open circuits and inductors act as short circuits. Therein lies the basic algorithm for computing an equilibrium point.

1. Configure all independent sources to be constant valued.
2. Replace all capacitors with open circuits.
3. Replace all inductors with short circuits.

Solving the equations that describe the resulting system gives an equilibrium point. This system of equations is nonlinear and algebraic (no time derivatives or integrals). Solving large nonlinear systems of algebraic equations is a difficult challenge. The only way to solve general nonlinear equations is to use iterative methods such as Newton's method. However, even these methods are not guaranteed to work. In fact, there is no practical algorithm that always works. When an iterative method fails, it does so by not converging to a solution. Convergence issues are the primary concern when using DC analysis because convergence is problematic for circuit simulators, especially on large circuits. The accuracy of the solution is a secondary issue, because once the simulator converges, the results are rarely inaccurate. In this chapter, the focus is mainly on convergence, though accuracy issues are also discussed.

## 2.2 DC Analysis Theory

As stated in Section 1.3.2 on page 10,

$$i(v(t)) + \frac{d}{dt}q(v(t)) + u(t) = 0 \tag{2.1}$$

$$v(0) = a \tag{2.2}$$

is solved to find the transient behavior of the circuit. Also of interest is the DC solution, or equilibrium point, which is defined as a solution to (2.1) that does not vary with time (the input $u(t)$ is assumed to be constant valued). The DC equations are formulated from (2.1) by assuming that $\frac{d}{dt}v(t) = 0$ for all $t$, and so $\frac{d}{dt}q(v(t)) = 0$ (this constraint replaces the initial condition constraint of (2.2)).

$$i(v_{dc}) + u_{dc} = 0 \tag{2.3}$$

Circuit simulators solve this equation to compute the DC solution or operating point. However, it must be stressed that the solution

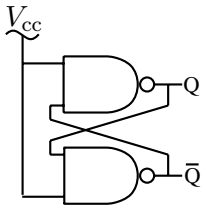


**Figure 2.1:** A latch, a circuit with 3 equilibrium points, one of which is unstable.

computed is not necessarily unique, nor is it required to be stable. For example, consider the latch shown in Figure 2.1. This circuit has three equilibrium points, $Q = V_{cc}$, $Q = 0$, and $Q = \frac{1}{2}V_{cc}$. The first two solutions are stable. In other words, if the circuit is at one of these solutions, and is perturbed slightly, it eventually returns to the same solution. The last equilibrium point is unstable, meaning any perturbation causes the voltages to drift away from this solution and eventually ends up at one of the other two solutions. The simulator is just as likely to find an unstable solution as it is to find a stable one.

It is important to understand that:

1. Circuits sometimes have more than one DC solution.
2. The DC solution computed by the circuit simulator may be unstable.

The fact that circuit simulators do not really distinguish between stable and unstable solutions allows a circuit simulator to compute a DC solution for oscillators, which do not generally have stable DC equilibrium points.

### 2.2.1 Solving Nonlinear Equations

The set of equations that result from DC analysis (2.3) and on every step of a transient analysis (4.12) are nonlinear algebraic systems and so in general cannot be solved directly. These equations can be solved by the Newton-Raphson algorithm (also referred to as Newton's method), which converts the solution of a nonlinear equation into the solution of a sequence of linear equations. Newton-Raphson starts with an initial guess. It then linearizes the circuit about that guess, and solves the linear circuit. The circuit is then re-linearized about the new point and the procedure repeats until the process converges.

Newton's method solves equations of the form

$$f(\hat{v}) = 0 \tag{2.4}$$

for $\hat{v}$ by starting with an initial guess called $v^{(0)}$ and repeatedly solving the Newton-Raphson iteration equation

$$J(v^{(k)})(v^{(k+1)} - v^{(k)}) = -f(v^{(k)}) \tag{2.5}$$

or

$$v^{(k+1)} = v^{(k)} - J^{-1}(v^{(k)})f(v^{(k)}) \tag{2.6}$$

for $v^{(k+1)}$ (the value of $v$ on the $k+1^{\text{th}}$ iteration) until some convergence criteria are met. $J(v) = \frac{d}{dv}f(v)$ is called the Jacobian of $f$ at $v$. Since both $f(v)$ and $v$ are $N$-dimensional vectors, $J(v)$ is an $N \times N$ matrix. It represents the circuit linearized about $v$. This process is explicitly stated in Algorithm 2.1 and is illustrated graphically with a simple scalar example in Figure 2.2.

The sequence generated by (2.6) is guaranteed to converge to $\hat{v}$ if $f$ is continuously differentiable, if the solution is isolated (this concept is discussed in Section 2.2.3.1 on page 25), and if $v^{(0)}$ is sufficiently close to $\hat{v}$. In circuit simulation, none of these three conditions are guaranteed, and so neither is convergence. Failure to converge is probably the biggest complaint designers have with circuit simulators.

Newton-Raphson has the very desirable property of quadratic convergence, meaning that once it is close to the solution, it reduces the error by squaring it on each iteration. The end result is that once

**Step 0:** Initialize
set $k \leftarrow 0$
choose $v^{(0)}$

**Step 1:** Linearize about $v^{(k)}$
$J_f(v^{(k)}) = \partial f(v^{(k)})/\partial v$
where $J_f$ is the Jacobian of $f$.

**Step 2:** Solve the linearized system
$v^{(k+1)} \leftarrow v^{(k)} - J_f^{-1}(v^{(k)})f(v^{(k)})$

**Step 3:** Iterate
set $k \leftarrow k + 1$
if not converged, go to step 1.

**Algorithm 2.1:** Newton-Raphson algorithm for finding $\hat{v}$ such that $f(\hat{v}) = 0$:

Newton-Raphson is close to the solution, the solution is found very accurately with only a few more iterations.

## 2.2.2 Convergence Criteria

Newton-Raphson is a method that takes an initial guess of the solution of a system of nonlinear equations, and refines it making it more and more accurate on each iteration. However, in its pure form, Newton's method never terminates. A way of deciding when the iteration should be terminated is needed.

### 2.2.2.1 Absolute Convergence Criteria

The Newton-Raphson iteration is considered to have converged, and therefore can be terminated, only after the approximate solution satisfies two convergence criteria. These two convergence criteria are given in simplified form first, with a more practical form given later.

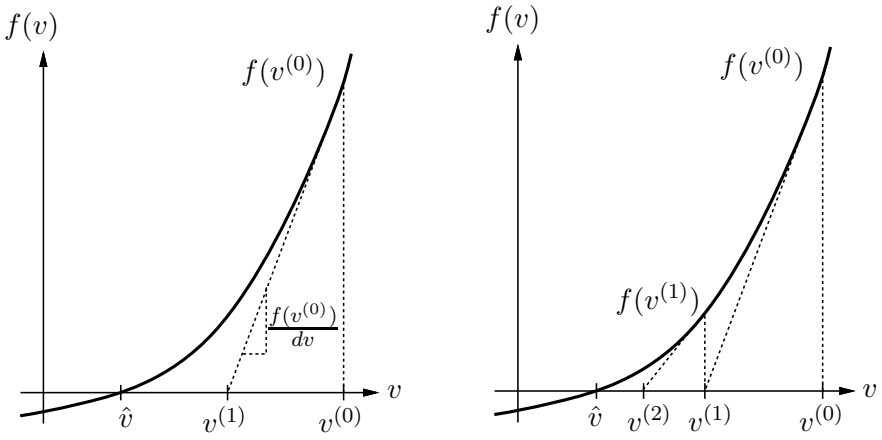


**Figure 2.2:** Newton's method applied to find the value $\hat{v}$ such that $f(\hat{v}) = 0$ (the value of $v$ where the curve crosses the horizontal axis). The process starts by guessing the value $v^{(0)}$. The function is linearized about $v^{(0)}$ and solved for the next guess $v^{(1)}$. If all goes well, $v^{(1)}$ is closer to the solution $\hat{v}$ than was $v^{(0)}$, and $v^{(k)} \to \hat{v}$ as $k \to \infty$. The process terminates when $v^{(k)}$ is sufficiently close to $\hat{v}$.

The first criterion specifies that KCL should be satisfied to a given degree,

$$|f_n(v^{(k)})| < \epsilon_f \tag{2.7}$$

where $\epsilon_f$ is some small positive number. The second tries to control the error in the solution by asserting that the difference between the last two iterations must be small,

$$|v_n^{(k)} - v_n^{(k-1)}| < \epsilon_x \tag{2.8}$$

where $\epsilon_x$ is some small positive number.

A simulator considers $v^{(k)}$ a solution if (2.7) and (2.8) are both satisfied. It is necessary to assure that both conditions are satisfied to be certain that the solution computed by Newton's method is correct. However, most simulators (in particular, those that are descendants

of SPICE) only check condition (2.8). This occasionally results in a condition called false convergence, which occurs when the iteration is terminated prematurely with (2.4) far from being satisfied because progress on one iteration is slow (and therefore $v^{(k)} - v^{(k-1)}$ is small and so (2.8) is satisfied). These simulators try to avoid this problem by using heuristics (generally that $|f_n(v^{(k)}) - f_n(v^{(k-1)})| < \epsilon_f$), but these sometimes fail. The reason they fail is that neither condition actually verifies that the equations are being solved (that KCL is being honored). Instead they are satisfied when the current iteration is close to the previous one. Thus, if the rate of convergence becomes slow, these conditions are satisfied without the iteration being close to the solution. The progress on an iteration in both (2.7) and (2.8) would be slow if the Jacobian is wrong because of an error in the implementation of a model. On the other hand, conditions (2.7) and (2.8), which are used in Spectre, do not exhibit false convergence. Condition (2.7) assures that KCL is approximately satisfied, and (2.8) bounds the error in the solution.

**Why the Residue Criterion is Needed** In general, the residue criterion (2.7) is important when the impedance at a node is small. For example, consider a strongly forward-biased *pn*-junction. Even small changes in voltage across the junction result in large changes in the current through the junction. In such circuits, the residue criterion is more important that the update criterion (2.8) for maintaining the accuracy of the solution.

**Why the Update Criterion is Needed** The update criterion (2.8) is important when the impedance at a node is large. Consider a node that is isolated from others by a reverse-biased *pn*-junction. There is a large range of voltages that result in the current through the junction being less than the absolute current tolerance. In this situation, the update criterion is more important than the residue criterion for maintaining the accuracy of the solution.

#### 2.2.2.2 Relative Convergence Criteria

While conceptually simple, the convergence conditions given by (2.7) and (2.8) are not used as given in practice because the criteria do not tolerate changes in scale well. Consider condition (2.8), a better criterion is:

**Newton Update Convergence Criterion**
*The solution updates are said to have converged if*

$$|v_n^{(k)} - v_n^{(k-1)}| < \texttt{reltol}\, v_{n\max} + \texttt{vntol} \qquad (2.9)$$

*where typically* $v_{n\max} = \max(|v_n^{(k)}|, |v_n^{(k-1)}|)$.

By default, `reltol` is 0.001 and `vntol` (called `vabstol` in Spectre) is $1\mu$V.

`reltol` is called the relative convergence tolerance because it specifies how small the update must be relative to the node voltage. `reltol` allows you to simulate high voltage circuits and low voltage circuits without adjusting the convergence criteria. `vntol` is referred to as an absolute tolerance. It becomes important when the solution on a particular node is near zero. In this case, just using the `reltol` criterion would force the update to be microscopic before convergence was allowed. In some cases the required update would be smaller than the computer round-off error, in which case convergence would never occur. `vntol` prevents these problems from occurring by causing any update smaller that `vntol` to be accepted.

Criterion (2.9) overcomes several problems that plague (2.8). First, (2.8) does not automatically scale itself to the problem, and so $\epsilon_x$ would have to be manually adjusted to fit the problem. Second, if the solution at node $n$ is large and at node $m$ is small, then $\epsilon_x$ must be chosen fairly large so that convergence is not precluded at node $n$, which results in convergence being checked very loosely at node $m$.

The residue convergence criterion of Equation (2.7) can also be improved.