Course Notes for

Numerical Simulation of Electronic Circuits: State-of-the-Art Techniques and Challenges

Peter Feldmann and Roland W. Freund

1. Circuit Equations

1.1. Kirchhoff's Laws

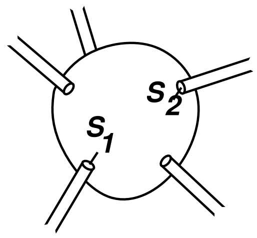
Kirchhoff's current law (KCL): The sum of all branch currents entering any closed surface is zero:

$$\sum_{k \in \text{loop}} I_k = 0.$$

The KCL results from the charge continuity equation

$$\int_S \overline{J} \cdot \overline{dS} = -\frac{\partial q}{\partial t}.$$

The charge continuity law states that the integral of the current density, $\overline{J}$, taken over any closed surface is equal to the negative rate of change of the electric charge contained in that closed surface, $-\frac{\partial q}{\partial t}$. In a lumped-element circuit approximation, the last term is either negligible or modeled by a capacitor, and therefore, can be set to zero.



We assume that current flows only through conductors, and hence

$$\int_S \overline{J} \cdot \overline{dS} = \int_{S_1} \overline{J} \cdot \overline{dS} + \int_{S_2} \overline{J} \cdot \overline{dS} + \cdots = I_1 + I_2 + \cdots = 0,$$

since the integral of the current density over the cross section of the conductor is exactly the current flowing through that conductor.

Note that the KCL states that the sum of currents is zero across any closed surface. In particular, a closed surface may contain only one circuit node. In this case, we have the more restricted form of Kirchhoff's current law which states that all currents entering a *nod/* add to zero.

Kirchhoff's voltage law (KVL): The sum of all branch voltages along any closed loop in a circuit adds to zero:

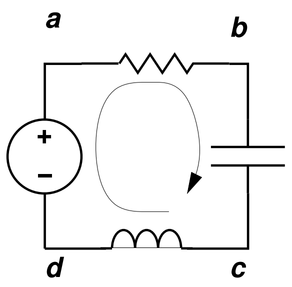
$$\sum_{k \in \text{loop}} V_k = 0.$$

The KVL results from Faraday's law

$$\oint \overline{E} \cdot \overline{dl} = -\frac{\partial \Psi}{\partial t}.$$

Faraday's law states that the integral of the electric field, $\overline{E}$, taken over any closed loop is equal to the negative rate of variation of magnetic flux through the loop (the term $\frac{-\partial \Psi}{\partial t}$). The last term is either assumed to be negligible or is already modeled by a lumped inductor or a mutual inductance in the loop.

Example: Consider the following circuit



Here we have

$$\oint \overline{E} \cdot \overline{dl} = \int_a^b \overline{E} \cdot \overline{dl} + \int_b^c \overline{E} \cdot \overline{dl} + \int_c^d \overline{E} \cdot \overline{dl} + \int_d^a \overline{E} \cdot \overline{dl} = 0,$$

which is in fact

$$V_{ab} + V_{bc} + V_{cd} + V_{da} = 0.$$

Another way of expressing the KVL is to assign voltage potentials to the nodes, e.g., V_a, V_b, V_c, and V_d. Since any loop comes back to the same node, and hence the same potential voltage, the sum of branch voltages, expressed as potential differences along the loop, will always add to zero. Indeed, for the above circuit, we have

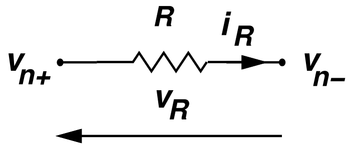
$$V_{ab} + V_{bc} + V_{cd} + V_{da} = (V_b - V_a) + (V_c - V_b) + (V_d - V_c) + (V_a - V_d) = 0.$$

Observe that node voltages are not uniquely defined. In fact, the relation continues to hold even if we add the same voltage to each of the node potentials. This ambiguity can be removed by choosing one of the nodes as reference (ground) and assign a potential (typically zero) to it. This way all node potentials become uniquely defined.

1.2. Widely-Used Circuit Elements and Their Equations

The two Kirchoff laws reflect only the topology of the circuit. In order to describe the circuit completely, we need to add laws reflecting the properties of the elements of the circuit. We call these laws *branch constitutive relations*(BCR). We now state the BCRs for a few widely-used circuit elements.

Resistors are characterized by a relationship between their current and voltage, Ohm's law:



In general, the relationship may be nonlinear and maybe implicit, e.g.,

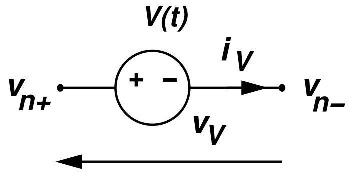
$$i_R = i(v_R) \qquad \text{or} \qquad f(i_R, v_R) = 0.$$

Linear resistors have a linear relationship between their voltage and current. The proportionality constant can be expressed as either a resistance, R, or a conductance, $G = 1/R$, and thus we have

$$i_R = \frac{v_R}{R} \qquad \text{or} \qquad i_R = G \cdot v_R,$$

respectively.

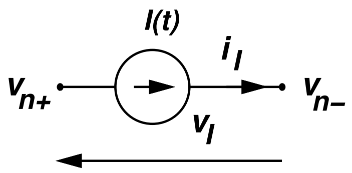
Independent Voltage Sources are ideal elements that maintain a given voltage across them and deliver any amount of current:



Note that the value of the voltage may be a function of time. The BCR of an independent voltage source is simply

$$v_V = V(t) \qquad \text{and} \qquad i_V = \text{any value}.$$

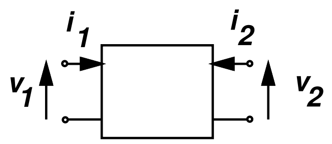
Independent Current Sources are ideal elements that generate a given current, which may be a function of time, independent of the voltage across them:



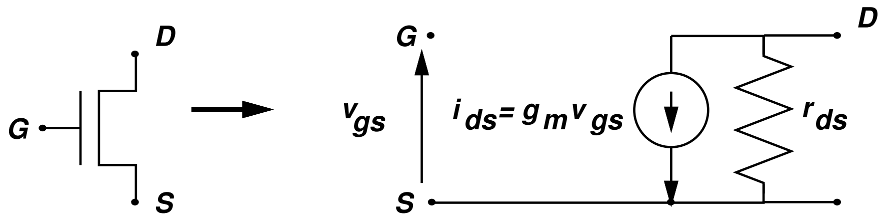
The BCR of an independent current source is

$$i_I = I \qquad \text{and} \qquad v_I = \text{any value}$$

Controlled sources are four-terminal elements that can also be regarded as two-ports:

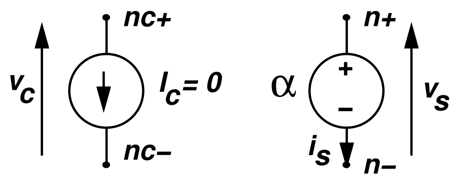


They are used mainly for modeling of more complicated devices. For example, below we have a very simple model of a MOS transistor:



Here are two examples of controlled sources:

- **Voltage-controlled voltage sources** are elements of the type

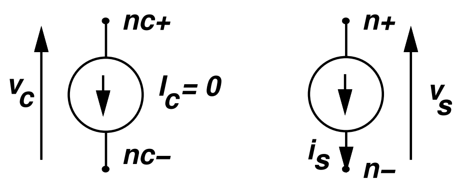


There BCR is

$$i_c = 0 \qquad \text{and} \qquad v_s = \alpha v_c.$$

The zero-valued current source acts a voltmeter.

- **Voltage-controlled current sources** are elements of the type

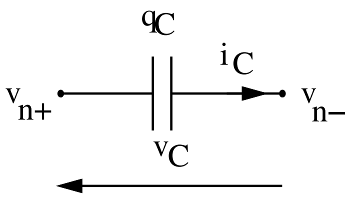


There BCR is

$$i_c = 0 \qquad \text{and} \qquad i_s = g_m v_c$$

So far we listed only devices that can be characterized by algebraic equations. In general, we refer to such devices as *resistive* or *memory-less*. Other devices, however, must be modeled with differential equations, and we refer to these as *reactive* elements. Next, we list some reactive elements.

Capacitors are characterized by a relationship between the electrical charge they store and the voltage across it:



Capacitor charge is related to the current flowing through it by $i_c = \dot{q}_c$. Here and in the sequel, $\dot{z}(t)$ always denotes the derivative of a time-dependent function $z(t)$ with respect to t.

The charge-voltage relationship for a capacitor may be nonlinear or even implicit:

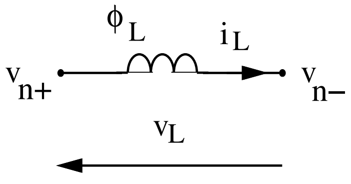
$$q_c = q(v_c) \qquad \text{or} \qquad g(q_c, v_c) = 0.$$

Here g is an implicit function. If the capacitor is linear, its BCR becomes

$$i_c = C\dot{v}_c,$$

where C is the *capacitance*.

Inductors are characterized by a relationship between the magnetic flux and its current:



The magnetic flux is related to the voltage across the inductor by $v_L = \dot{\phi}_L$.

The inductor law is

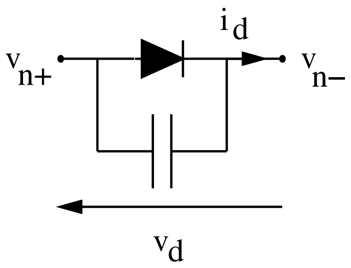
$$\phi_L = \phi(i_L) \qquad \text{or} \qquad h(\phi_L, i_L) = 0.$$

Here h is an implicit current-flux relationship. The BCR for the inductor is

$$v_L = L\dot{i}_L,$$

where L is the *inductance*.

Diodes are semiconductor devices which exhibit both resistive and reactive properties:



A diode exhibits the well-known exponential dependency of current on voltage. There is also a voltage-dependent charge q_d associated with the depletion region which acts as a nonlinear capacitance.

The BCR of a diode is

$$i_d = i_{DC}(v_d) + \frac{dq_d(v_d)}{dt},$$

with simple to very complicated expressions for the DC current i_{DC} and diode charge q_d. For example, assuming a very simple model, the DC current is given by

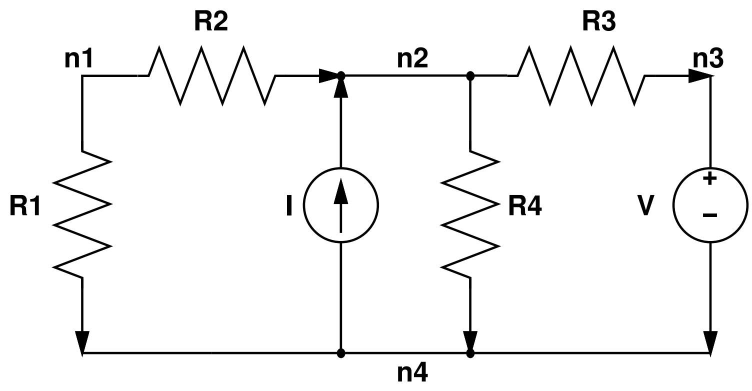
$$i_{DC} = I_{sat}\left(e^{\frac{qv_d}{kT}} - 1\right).$$

Transistors are multi(three or four)-terminal devices which also exhibit both reactive and resistive components. The equations describing transistors range from simple to very elaborate.

1.3. Systematic Formulation of Circuit Equations

The topology of a circuit can be described using the *adjacency matrix*. The rows of this matrix correspond to circuit elements and columns to circuit nodes. We attach an arbitrary direction to each circuit element. The row corresponding to an element has a $+1$ entry in the column corresponding to the "from" node and a -1 entry in the column corresponding to the "to" node. The rest of the row entries are zero.

As an example, consider the following circuit:



The associated adjacency matrix is given by

$$\begin{array}{c} \\ e1 \\ e2 \\ e3 \\ e4 \\ ev \\ ei \end{array} \begin{array}{c} \begin{array}{cccc} n1 & n2 & n3 & n4 \end{array} \\ \begin{bmatrix} +1 & 0 & 0 & -1 \\ +1 & -1 & 0 & 0 \\ 0 & +1 & -1 & 0 \\ 0 & +1 & 0 & -1 \\ 0 & 0 & +1 & -1 \\ 0 & -1 & 0 & +1 \end{bmatrix} \end{array} = \mathbf{A}_0.$$

Observe that the sum of all columns is always zero. Therefore, one of the columns of the matrix is redundant, i.e., its entries can be determined from the rest of the matrix. The product of the adjacency matrix with the vector of node voltages is in fact the vector of branch voltages:

$$\mathbf{A}_0 \begin{bmatrix} v_{n1} \\ v_{n2} \\ v_{n3} \\ v_{n4} \end{bmatrix} = \begin{bmatrix} v_{n1} - v_{n4} \\ v_{n1} - v_{n2} \\ v_{n2} - v_{n3} \\ v_{n2} - v_{n4} \\ v_{n3} - v_{n4} \\ v_{n4} - v_{n2} \end{bmatrix} = \begin{bmatrix} v_{R1} \\ v_{R2} \\ v_{R3} \\ v_{R4} \\ v_V \\ v_I \end{bmatrix}.$$

If we decide to treat v_{n4} as reference (ground) and set $v_{n4} = 0$, we can remove v_{n4} from the vector of unknown node voltages and also drop the column corresponding to $n4$ from $\mathbf{A}_0$. The result is the matrix

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & -1 & 0 \\ 0 & 1 & -1 \\ 0 & 1 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Using this matrix $\mathbf{A}$, we have the relation

$$\mathbf{A}\mathbf{v}_n = \mathbf{v}_b.$$

Here $\mathbf{v}_n$ is the vector of node voltages except the reference (ground) node and $\mathbf{v}_b$ is the vector of branch voltages. We already indicated that this is just another way of writing the KVL equations.

Next, consider the product of the transpose $\mathbf{A}^T$ matrix with the branch currents vector:

$$\mathbf{A}^T\mathbf{i}_b = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 1 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} i_1 \\ i_2 \\ i_3 \\ i_4 \\ i_V \\ i_I \end{bmatrix}.$$

Each row of the matrix $\mathbf{A}^T$ corresponds to a non-reference node, and it has a 1 entry corresponding to a branch leaving the node and a -1 entry corresponding to a branch entering the node. When the row is multiplied with the vector of currents, we get in fact the KCL equations:

$$\mathbf{A}^T\mathbf{i}_b = 0.$$

So far, we identified the following equations and unknowns for the system. The unknowns are:

- $\mathbf{v}_n$ — the $N-1$ node voltages, where N is the number of circuit nodes,
- $\mathbf{v}_b$ — the B branch voltages, where B is the number of circuit branches,
- $\mathbf{i}_b$ — the B branch currents,

a total of $2B+N-1$ unknowns.

The equations are:

- $\mathbf{A}\mathbf{v}_n = \mathbf{v}_b$ — the B KVL equations,
- $\mathbf{A}^T\mathbf{i}_b = 0$ — the $N-1$ KCL equations.

So far, we have $B+N-1$ equations, and hence we need B more in order to determine the $2B+N-1$ unknowns. Note that the equations so far reflect information only on the connectivity of the network. We have not yet used the relations between the currents and voltages of the same branch which characterize the types of circuit elements. These constitute the last group of equations: the branch constitutive relations (BCR).

For the example above, we have:

$$v_1 - R_1 i_1 = 0,$$

$$\begin{aligned} v_2 - R_2 i_2 &= 0, \\ v_3 - R_3 i_3 &= 0, \\ v_4 - R_4 i_4 &= 0, \\ v_V + 0 \cdot i_V &= V, \\ 0 \cdot v_I + i_I &= I. \end{aligned}$$

Writing these relations in matrix form, we get

$$\begin{bmatrix} 1 & & & & & \\ & 1 & & & & \\ & & 1 & & & \\ & & & 1 & & \\ & & & & 1 & \\ & & & & & 0 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_V \\ v_I \end{bmatrix} + \begin{bmatrix} -R_1 & & & & & \\ & -R_2 & & & & \\ & & -R_3 & & & \\ & & & -R_4 & & \\ & & & & 0 & \\ & & & & & 1 \end{bmatrix} \begin{bmatrix} i_1 \\ i_2 \\ i_3 \\ i_4 \\ i_V \\ i_I \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ V \\ I \end{bmatrix},$$

or, more compactly,

$$\mathbf{Y}\mathbf{v}_b + \mathbf{Z}\mathbf{i}_b = \mathbf{w}.$$

In the case of two-terminal elements, the matrices $\mathbf{Y}$ and $\mathbf{Z}$ are diagonal. Multi-terminal elements will introduce off-diagonal terms. We now have enough equations and unknowns to write a system of equations the solution of which are all circuit variables:

$$\begin{aligned} \mathbf{A}\mathbf{v}_n - \quad \mathbf{v}_b &= 0, \\ \mathbf{A}^T\mathbf{i}_b \qquad\qquad &= 0, \\ \mathbf{Z}\mathbf{i}_b \qquad + \mathbf{Y}\mathbf{v}_b &= \mathbf{w}. \end{aligned}$$

We have just written the so-called *Sparse Tableau* formulation of circuit equations. It has a relatively large dimension, $2B + N - 1$ equations and unknowns, but its matrix has a well-defined structure and is very sparse (most of the matrix elements are 0).

One can immediately obtain a more compact form just by eliminating the branch voltages $\mathbf{v}_b$ using the KVL equations $\mathbf{v}_b = \mathbf{A}\mathbf{v}_n$. The system then reduces to

$$\begin{aligned} -\mathbf{A}^T\mathbf{i}_b &= 0, \\ \mathbf{Y}\mathbf{A}\mathbf{v}_n + \mathbf{Z}\mathbf{i}_b &= \mathbf{w}. \end{aligned}$$

This system has dimension $B + N - 1$.

If we assume that all devices are voltage controlled, in other words their branch currents can always be expressed as a function of their branch voltages, we can reduce the system even further. Note that such an assumption excludes voltage sources, independent and controlled, and all current-controlled sources. Under this assumption the BCRs can be written as follows:

$$\mathbf{Y}\mathbf{v}_b + \mathbf{i}_b = \mathbf{w}.$$

Hence the branch currents can be expressed in terms of branch voltages, i.e.,

$$\mathbf{i}_b = \mathbf{w} - \mathbf{Y}\mathbf{v}_b,$$

or even in terms of node voltages:

$$\mathbf{i}_b = \mathbf{w} - \mathbf{Y}\mathbf{A}\mathbf{v}_n.$$

We can use this equation to eliminate the branch current variables and leave the system of equations only in terms of the nodal voltages:

$$-\mathbf{A}^T(\mathbf{w} - \mathbf{Y}\mathbf{A}\mathbf{v}_n) = 0.$$

We obtain a "compact" system of dimension $N-1$ which we call the *pure nodal* formulation:

$$\mathbf{A}^T\mathbf{Y}\mathbf{A}\mathbf{v}_n = \mathbf{A}^T\mathbf{w}.$$

This pure nodal formulation, however, is not very useful, since it cannot easily handle a wide category of devices including the independent-voltage source.

The example above is simple in the sense that all the BCRs are algebraic and linear. In general, the BCRs of a circuit form a system of functional equations

$$\mathcal{F}(\mathbf{i}_b, \mathbf{v}_b, \mathbf{q}_b, \phi_b, t) = 0,$$

where $\mathbf{i}_b$, $\mathbf{v}_b$, $\mathbf{q}_b$, and ϕ_b are all functions of time.

So far, we described two extreme ways to formulate circuit equations. The sparse tableau method is completely general, easy to understand and implement, but leads to a large system. The other extreme is the pure nodal method, which leads to probably the most compact system of equations, is almost as easy to implement, but unfortunately cannot handle many very useful devices. Real-life circuit simulators implement a hybrid method.

1.4. Formulation of Circuit Equations on an Element-by-Element Basis

One way to generate a complete set of circuit equations is to analyze the contribution of each circuit element separately, and ensure that for every unknown introduced, there will be a corresponding equation. This way, the total number of equations and unknowns is the same. By circuit elements we understand the nodes and the devices of the circuit.

We now derive the modified-nodal formulation contributions *stamps* for a few types of circuit elements.

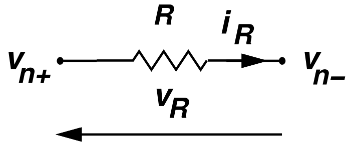
Nodes

The nodes n_i of the circuit introduce their node voltage, v_i, as the unknown. The associated equation is generated by the KCL for the node:

$$\mathrm{KCL}_{n_i}: \qquad 0 = \cdots .$$

The dots are to be filled by the various devices connected to node n_i.

The linear resistor



The resistor introduces two variables: its current i_R and its voltage v_R. It also contributes two equations: Kirchhoff's voltage law, and the resistor's BCR. Therefore, the contribution of the linear resistor to the system of circuit equations is as follows:

$$\begin{aligned} \mathrm{KCL}_{n+}: &\quad 0 = \cdots - i_R + \cdots , \\ \mathrm{KCL}_{n-}: &\quad 0 = \cdots + i_R + \cdots , \\ \mathrm{KVL}_{V}: &\quad 0 = v_R - v_{n+} + v_{n-} \\ \mathrm{BCR}_{R}: &\quad 0 = Ri_R - v_R. \end{aligned}$$

This method of contributing to the system of equations is equivalent to the sparse tableau formulation. If all devices introduce their currents and voltages as unknowns and their KVLs and BCRs as equations, in the end, we will have the complete set of sparse tableau equations.

Alternatively, note that the variables introduced by the resistor can be expressed in terms of only the node voltages:

$$i_R = v_R/R = (v_{n+} + v_{n-})/R.$$

Therefore, we can write the contribution of the resistor without introducing new equations and unknowns:

$$\begin{aligned} \mathrm{KCL}_{n+}: &\quad 0 = \cdots - (v_{n+} - v_{n-})/R + \cdots , \\ \mathrm{KCL}_{n-}: &\quad 0 = \cdots + (v_{n+} - v_{n-})/R + \cdots . \end{aligned}$$

At the individual resistor level, this method is equivalent to the nodal formulation.

In the following discussion of other devices, we will only describe the contributions that lead to the minimal number of equations.

The general two-terminal resistive device

A resistive device is characterized by a relation between its current and voltage, which in the most general case may be implicit, i.e., $f(i_B, v_B) = 0$. In this case, the current of the device, i_B, must be treated as an unknown and the device must define its own equation:

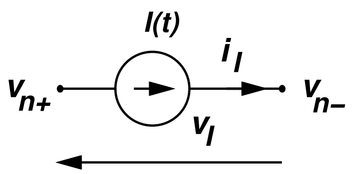
$$\begin{aligned}
\mathrm{KCL}_{n+}: &\quad 0 = \cdots - i_B + \cdots \ , \\
\mathrm{KCL}_{n-}: &\quad 0 = \cdots + i_B + \cdots \ , \\
\mathrm{BCR}_B: &\quad 0 = f(v_{n+} - v_{n-}, i_B).
\end{aligned}$$

In the special case that the current i_B can be expressed as a function of its voltage, $i_B = i(v_{n+} - v_{n-})$, the device becomes compatible with a nodal formulation:

$$\begin{aligned}
\mathrm{KCL}_{n+}: &\quad 0 = \cdots - i(v_{n+} - v_{n-}) + \cdots \ , \\
\mathrm{KCL}_{n-}: &\quad 0 = \cdots + i(v_{n+} - v_{n-}) + \cdots \ .
\end{aligned}$$

The linear resistor is a special case of such a formulation.

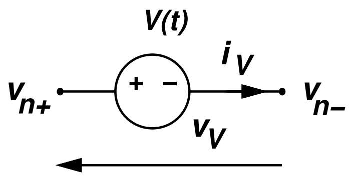
Independent current source



The current source does not introduce new equations or unknowns. It only affects the KCL equations of its two adjacent nodes:

$$\begin{aligned}
\mathrm{KCL}_{n+}: &\quad 0 = \cdots - I(t) + \cdots \ , \\
\mathrm{KCL}_{n-}: &\quad 0 = \cdots + I(t) + \cdots \ .
\end{aligned}$$

Independent voltage source

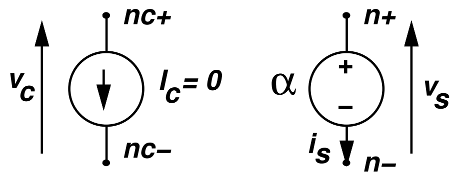


Unfortunately, it is not possible to describe the contribution of the voltage source solely in terms of nodal voltages and KCL equations. The reason is that the current through the voltage source does not depend on the node voltages, and the only way to determine its value is to solve the entire circuit. Therefore, we must introduce the voltage-source current, i_V, as an unknown. As a consequence, we must also introduce an equation. The total contribution is as follows:

$$\begin{aligned} \mathrm{KCL}_{n+}: &\quad 0 = \cdots - i_V + \cdots \ , \\ \mathrm{KCL}_{n-}: &\quad 0 = \cdots + i_V + \cdots \\ \mathrm{KVL}_{V}: &\quad 0 = V(t) - v_{n+} + v_{n-} \ . \end{aligned}$$

The voltage source must introduce a non-nodal unknown. This is the reason why voltage sources are not compatible with the pure nodal formulation.

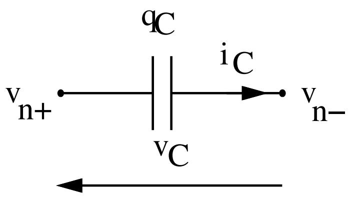
Voltage-controlled voltage source



Again, the controlled voltage source is not compatible with a pure nodal formulation, and therefore, it must introduce a non-nodal unknown, its current i_V. As a consequence, we also need an extra equation, its BCR:

$$\begin{aligned} \mathrm{KCL}_{n+}: &\quad 0 = \cdots - i_V + \cdots \ , \\ \mathrm{KCL}_{n-}: &\quad 0 = \cdots + i_V + \cdots \ , \\ \mathrm{BCR}: &\quad 0 = v_{n+} - v_{n-} - \alpha(v_{nc+} - v_{nc-}). \end{aligned}$$

Capacitors



The capacitor is characterized by a relationship of its charge and voltage. In the most general case, the capacitor law is given as an implicit equation:

$$\begin{aligned} g(q_C, v_C) &= 0, \\ i_C &= \dot{q}_c. \end{aligned}$$

In this case, we must add the capacitor charge as an unknown to the system of circuit equations. The contribution of an implicitly modeled capacitor is as follows:

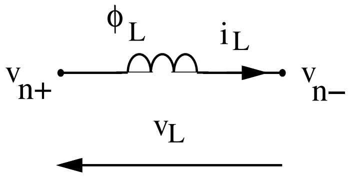
$$\begin{aligned} \mathrm{KCL}_{n+}: &\quad 0 = \cdots - \dot{q}_C + \cdots\ , \\ \mathrm{KCL}_{n-}: &\quad 0 = \cdots + \dot{q}_C + \cdots\ , \\ \mathrm{BCR}_C: &\quad 0 = g(q_C, v_{n+} - v_{n-}) \end{aligned}$$

If the capacitor can be modeled explicitly, i.e., $q_C = q(v_C)$, there is no need for new unknowns to be introduced by the capacitor:

$$\begin{aligned} \mathrm{KCL}_{n+}: &\quad 0 = \cdots - \dot{q}_C(v_{n+} - v_{n-}) + \cdots\ , \\ \mathrm{KCL}_{n-}: &\quad 0 = \cdots + \dot{q}_C(v_{n+} - v_{n-}) + \cdots\ . \end{aligned}$$

The linear capacitor is just a special case where $\dot{q}_C(v_{n+} - v_{n-}) = C(\dot{v}_{n+} - \dot{v}_{n-})$.

Inductors



In the most general case, an inductor is characterized by a relationship between its current and flux:

$$\begin{aligned} h(\phi_L, i_L) &= 0, \\ v_L &= \dot{\phi}_L. \end{aligned}$$

Here, we need to add two unknowns to the system of equations: the inductor current i_L and its flux ϕ_L. The contribution of the linear inductor to the circuit equations is as follows:

$$\begin{aligned} \mathrm{KCL}_{n+}: &\quad 0 = \cdots - i_L + \cdots\ , \\ \mathrm{KCL}_{n-}: &\quad 0 = \cdots + i_L + \cdots\ , \\ \mathrm{BCR1}_L: &\quad 0 = h(\phi_L, i_L), \\ \mathrm{BCR2}_L: &\quad 0 = v_{n+} - v_{n-} - \dot{\phi}_L. \end{aligned}$$

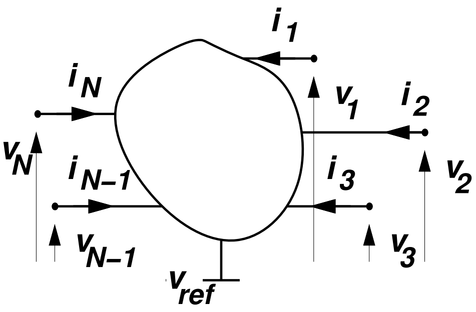
When we have an explicit model of the flux as a function of current, i.e., $\phi_L = \dot{\phi}(i_L)$, only one unknown needs to be introduced by the inductor:

$$\begin{aligned} \mathrm{KCL}_{n+}: &\quad 0 = \cdots - i_L + \cdots\ , \\ \mathrm{KCL}_{n-}: &\quad 0 = \cdots + i_L + \cdots\ , \\ \mathrm{BCR1}_L: &\quad 0 = v_{n+} - v_{n-} - \dot{\phi}(i_L). \end{aligned}$$

The linear inductor is just a special case where $\dot{\phi}(i_L) = L\dot{i_L}$.

Multi-terminal elements

In general, a multi-terminal element has $N+1$ terminals and N BCR equations that relate the N independent port voltages to the N independent terminal currents. One way (but not the only one) to chose the N ports is like in the figure below, where one of the nodes is chosen as reference (not to be confused to the ground node of the circuit).



The N independent BCRs have the following form:

$$\varphi_k(i_1, i_2, \ldots, i_N, v_1, v_2, \ldots, v_N) = 0, \quad k = 1, 2, \ldots, N.$$

The contribution to the circuit equations of the nonlinear multi-terminal element can be written as follows:

$$\begin{aligned}
\mathrm{KCL}_{n1}: &\quad 0 = \cdots - i_1 + \cdots \ , \\
\mathrm{KCL}_{n2}: &\quad 0 = \cdots - i_2 + \cdots \ , \\
&\quad \vdots \\
\mathrm{KCL}_{nN}: &\quad 0 = \cdots - i_N + \cdots \ , \\
\mathrm{KCL}_{nref}: &\quad 0 = \cdots + i_1 + i_2 + \cdots + i_N + \cdots \ , \\
\mathrm{BCR}_1: &\quad 0 = \varphi_1(i_1, i_2, \ldots, i_N, v_{n1} - v_{nref}, v_{n2} + v_{nref}, \cdots, v_{nN} + v_{nref}), \\
\mathrm{BCR}_2: &\quad 0 = \varphi_2(i_1, i_2, \ldots, i_N, v_{n1} - v_{nref}, v_{n2} + v_{nref}, \ldots, v_{nN} + v_{nref}), \\
\vdots &\quad \\
\mathrm{BCR}_3: &\quad 0 = \varphi_N(i_1, i_2, \ldots, i_N, v_{n1} - v_{nref}, v_{n2} + v_{nref}, \ldots, v_{nN} + v_{nref}).
\end{aligned}$$

In the case that the currents in the BCR equations can be expressed as functions of voltages, i.e.,

$$i_k = \psi_k(v_1, v_2, \ldots, v_N) = 0, \quad k = 1, 2, \ldots, N,$$

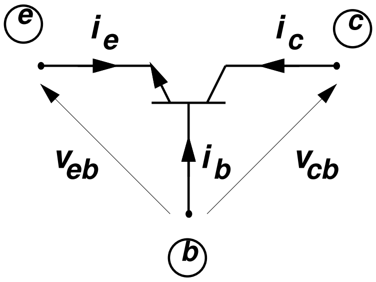
a nodal compatible formulation becomes possible, and the contribution of the multi-terminal element is as follows:

$$\mathrm{KCL}_{n1}: \qquad 0 = \cdots - \psi_1(v_{n1} - v_{nref}, \ldots, v_{nN} - vnref) + \cdots \ ,$$

$$
\begin{aligned}
\mathrm{KCL}_{n2}: &\quad 0 = \cdots - \psi_2(v_{n1} - v_{nref}, \ldots, v_{nN} - vnref) + \cdots\ , \\
&\vdots \\
\mathrm{KCL}_{nN}: &\quad 0 = \cdots - \psi_N(v_{n1} - v_{nref}, \ldots, v_{nN} - vnref) + \cdots \\
\mathrm{KCL}_{nref}: &\quad 0 = \cdots + \sum_{k=1}^{N} \psi_k(v_{n1} - v_{nref}, \ldots, v_{nN} - vnref) + \cdots\ .
\end{aligned}
$$

Of course, hybrid formulations are possible when only a part of the terminal currents can be expressed as functions of the port voltages.

Examples of multi-port elements are transistors, such as the *npn* bipolar transistor:



A simple low-frequency characterization of the bipolar transistor is given by the Ebers-Moll equations:

$$
\begin{aligned}
i_e &= -I_{ES}\left(e^{\frac{-v_{eb}}{v_T}} - 1\right) + \alpha_R I_{CS}\left(e^{\frac{-v_{cb}}{v_T}} - 1\right), \\
i_c &= \alpha_F I_{ES}\left(e^{\frac{-v_{eb}}{v_T}} - 1\right) - I_{CS}\left(e^{\frac{-v_{cb}}{v_T}} - 1\right).
\end{aligned}
$$

The associated contributions of a bipolar transistor using the Ebers-Moll model are as follows:

$$
\begin{aligned}
KCLe: &\quad 0 = \cdots + I_{ES}\left(e^{\frac{v_b - v_e}{v_T}} - 1\right) - \alpha_R I_{CS}\left(e^{\frac{v_b - v_c}{v_T}} - 1\right) + \cdots\ , \\
KCLc: &\quad 0 = \cdots - \alpha_F I_{ES}\left(e^{\frac{v_b - v_e}{v_T}} - 1\right) + I_{CS}\left(e^{\frac{v_b - v_c}{v_T}} - 1\right) + \cdots\ , \\
KCLb: &\quad 0 = \cdots - (1 - \alpha_F) I_{ES}\left(e^{\frac{v_b - v_e}{v_T}} - 1\right) - (1 - \alpha_R) I_{CS}\left(e^{\frac{v_b - v_c}{v_T}} - 1\right) + \cdots\ .
\end{aligned}
$$

More accurate models represent the bipolar transistor as a four-terminal element, where the fourth terminal represents the substrate. Other examples of nonlinear multi-terminal elements are the MOS transistors, which also can be modeled as either a three-terminal or a four-terminal device.

1.5. Various Forms of Circuit Equations

The result of adding together the contributions of all circuit elements is a system of first order, nonlinear, ordinary (as opposed to partial) algebraic-differential equations. The equations can be expressed in one of the forms below.

- The most general form, useful mostly for theoretical studies:

$$\mathbf{f}(\dot{\mathbf{x}}, \mathbf{x}, t) = 0$$

- The form that is the most natural for circuit simulation programs:

$$\dot{\mathbf{q}}(\mathbf{x}) + \mathbf{j}(\mathbf{x}) - \mathbf{e}(t) = 0$$

- The first term can be expanded using the chain rule resulting in

$$\mathbf{C}(\mathbf{x})\dot{\mathbf{x}} + \mathbf{j}(\mathbf{x}) - \mathbf{e}(t) = 0$$

  where $\mathbf{C}(\mathbf{x}) = \frac{\partial \mathbf{q}}{\partial \mathbf{x}}$. This is the form used in Spice. With this form, however, one must be careful with the integration method because it can lead to non-conservation of charge or fluxes. This issue will be discussed in more detail later.

- Finally, by introducing all (or at least the nonlinear) charges and fluxes as explicit unknowns in the system of equations we can formulate a system where the dependence on $\dot{\mathbf{x}}$ is linear

$$\mathbf{C}\dot{\mathbf{x}} + \mathbf{j}(\mathbf{x}) - \mathbf{e}(t) = 0$$

The matrices $\mathbf{C}(\mathbf{x})$ and $\mathbf{C}$ are in general singular, therefore, circuits are modeled by differential algebraic equations (DAE) and not just ordinary differential equations (ODE). In order to see why, recall the contribution of a nonlinear capacitor to the modified nodal circuit equations.

$$\begin{aligned} \mathrm{KCL}_{n+}: &\quad 0 = \cdots - \dot{q}_c + \cdots \\ \mathrm{KCL}_{n-}: &\quad 0 = \cdots + \dot{q}_c + \cdots \\ \mathrm{BCR}_C: &\quad 0 = g(q_c, v_c) \end{aligned}$$

The BCR contributes an all-zero row to the $\mathbf{C}(\mathbf{x})$ or $\mathbf{C}$ matrices. Also recall the linear inductor, with the following contribution to the modified nodal equations

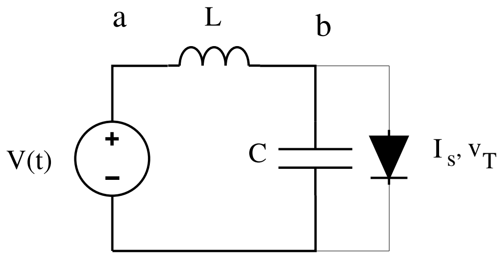
$$\begin{aligned} \mathrm{KCL}_{n+}: &\quad 0 = \cdots - i_L + \cdots \\ \mathrm{KCL}_{n-}: &\quad 0 = \cdots + i_L + \cdots \\ \mathrm{BCR}_L: &\quad 0 = L\dot{i}_L - v_{n+} + v_{n-}. \end{aligned}$$

Remark that there is no contribution from the inductor to the reactive part of the KCL equations. This may result again in an all-zero row in the $\mathbf{C}(\mathbf{x})$ or $\mathbf{C}$ matrices, when none of the other devices connected to one of these nodes contribute to the reactive part of the KCL equations.

If the matrix $\mathbf{C}(\mathbf{x})$ or $\mathbf{C}$ is singular, one cannot bring the system of equations to the canonical ODE form

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, t)$$

Example



1. Initially, we only have the two nodal KCL equations and two unknowns: v_a and v_b.

$$\begin{aligned} \mathrm{KCL}_a : &\quad 0 = \cdots \\ \mathrm{KCL}_b : &\quad 0 = \cdots \end{aligned}$$

2. The contribution of the voltage source is a new unknown i_V and a new equation

$$\begin{aligned} \mathrm{KCL}_a : &\quad 0 = -i_V + \cdots \\ \mathrm{KCL}_b : &\quad 0 = \cdots \\ \mathrm{BCR}_V : &\quad 0 = V(t) - v_a \end{aligned}$$

3. The contribution of the inductor is a new unknown i_L and another equation

$$\begin{aligned} \mathrm{KCL}_a : &\quad 0 = -i_V - i_L \cdots \\ \mathrm{KCL}_b : &\quad 0 = +i_L \cdots \\ \mathrm{BCR}_V : &\quad 0 = V(t) - v_a \\ \mathrm{BCR}_L : &\quad 0 = \frac{d}{dt}(L i_L) - v_b + v_a \end{aligned}$$

4. The capacitor does not add any new unknowns equation

$$\begin{aligned}
\mathrm{KCL}_a: &\quad 0 = -i_V - i_L \cdots \\
\mathrm{KCL}_b: &\quad 0 = +i_L - \frac{d}{dt}(Cv_b) \cdots \\
\mathrm{BCR}_V: &\quad 0 = V(t) - v_a \\
\mathrm{BCR}_L: &\quad 0 = \frac{d}{dt}(Li_L) - v_b + v_a
\end{aligned}$$

5. Finally, the diode contribution comes with no new unknowns and we obtain the circuit equations

$$\begin{aligned}
\mathrm{KCL}_a: &\quad 0 = -i_V - i_L \\
\mathrm{KCL}_b: &\quad 0 = +i_L - \frac{d}{dt}(Cv_b) - I_s\left(e^{\frac{v_b}{v_T}} - 1\right) \\
\mathrm{BCR}_V: &\quad 0 = V(t) - v_a \\
\mathrm{BCR}_L: &\quad 0 = \frac{d}{dt}(Li_L) - v_b + v_a
\end{aligned}$$

The circuit equations form a system of algebraic-differential equations, $\dot{\mathbf{q}}(\mathbf{x}) + \mathbf{j}(\mathbf{x}) - \mathbf{e}(t) = 0$ as discussed above

$$\begin{array}{llll}
 & -i_V - i_L & & = 0 \\
-C\dot{v}_b & +i_L - I_s\left(e^{\frac{v_b}{v_T}} - 1\right) & & = 0 \\
 & -v_a & +V(t) & = 0 \\
+L\dot{i}_L & -v_b + v_a & & = 0
\end{array}$$

Here

$$\mathbf{x} = \begin{bmatrix} v_a \\ v_b \\ i_V \\ i_L \end{bmatrix}, \quad \mathbf{q}(\mathbf{x}) = \begin{bmatrix} 0 \\ -Cv_b \\ 0 \\ -Li_L \end{bmatrix}, \quad \mathbf{j}(\mathbf{x}) = \begin{bmatrix} -i_V - i_L \\ i_L - I_s\left(e^{\frac{v_b}{v_T}} - 1\right) \\ -v_a \\ -v_b + v_a \end{bmatrix}, \quad \mathbf{e}(t) = \begin{bmatrix} 0 \\ 0 \\ -V(t) \\ 0 \end{bmatrix}.$$

2. Circuit-Simulation Problems

Next, we describe some typical circuit-simulation problems.

2.1. The DC Analysis Problem

An important type of circuit analysis concerns the state of DC equilibrium. In this case, all sources of excitation are constant in time (from here comes the name DC: direct current). Under certain circumstances the circuit will also reach a state of equilibrium where all circuit variables, voltages, currents, charges, etc. do not vary in time. This is not always possible; for example, oscillators have time-varying waveforms despite the fact that all excitation sources are constant. For the moment, however, assume that a DC equilibrium state exists. The circuit algebraic-differential equation, with the time-dependent excitation $\mathbf{e}(t)$ replaced by the constant excitation $\mathbf{e}_0$, is as follows:

$$\dot{\mathbf{q}}(\mathbf{x}) + \mathbf{j}(\mathbf{x}) - \mathbf{e}_0 = 0.$$

In the DC equilibrium, we have $\dot{\mathbf{q}}(= 0$, and hence the above equation reduces to the following system of nonlinear algebraic equations:

$$\mathbf{j}(\mathbf{x}_0) - \mathbf{e}_0 = 0.$$

Here, the unknowns $\mathbf{x}_0$ represent the DC equilibrium value of the circuit variables.

The DC analysis problem is difficult. The equations are strongly nonlinear and challenge even the most robust algorithms. When a solution is found, one must determine if it is stable, if it is unique, or if multiple solutions exist.

2.2. Small-Signal Analysis

Small-signal analysis is best suited to linear circuits and to nonlinear circuits operating in a linear regime. It is often the case that nonlinear circuits operate in such a linear regime. They are biased in such a way that, locally, the device characteristics are almost linear. Then, on top of the bias point, we apply a "small" time-varying excitation. If this excitation is "small" enough, the circuit behaves as if it were linear with respect to the "small" signal. We say that the circuit was linearized about the operating point.

Small-signal analysis is particularly interesting, first because many interesting circuits operate in this regime, and because the solution of circuit equations can be found exactly and efficiently without numerical integration of nonlinear differential-algebraic equations.

The analysis of such circuits is performed as follows. First, the circuit is solved with only the constant bias excitation $\mathbf{e}_0$. This is in fact just a DC analysis and consists of solving the equation

$$\mathbf{j}(\mathbf{x}_0) = \mathbf{e}_0.$$

Note that it is not guaranteed that such a solution exists and is stable. Circuits may exhibit autonomous oscillations. However, the class of circuits that we consider now will have a stable constant solution $\mathbf{x}_0$.

Consider now applying a time-varying "small-signal", $\mathbf{e}(\mathrm{t})$, on top of the constant biasing excitation $\mathbf{e}$. The system of differential equations becomes

$$\dot{\mathbf{q}}(\mathbf{x}_0 + \mathbf{x}(t)) + \mathbf{j}(\mathbf{x}_0 + \mathbf{x}(t)) = \mathbf{e}_0 + \mathbf{e}(t).$$

Let us expand the functions $\mathbf{q}$ and $\mathbf{j}$ in a Taylor series about the operating point $\mathbf{x}_0$. If the time-varying component of the signals are "small" enough, the first two terms of the Taylor expansion will accurately approximate the functions, and we get

$$\frac{d}{dt}\left[\mathbf{q}(\mathbf{x}_0) + \left.\frac{\partial \mathbf{q}}{\partial \mathbf{x}}\right|_{\mathbf{x}_0} \mathbf{x}(t)\right] + \mathbf{j}(\mathbf{x}_0) + \left.\frac{\partial \mathbf{j}}{\partial \mathbf{x}}\right|_{\mathbf{x}_0} \mathbf{x}(t) = \mathbf{e}_0 + \mathbf{e}(t).$$

We perform differentiation with respect to time and obtain

$$\frac{d}{dt}\mathbf{q}(\mathbf{x}_0) + \frac{d}{dt}\left.\frac{\partial \mathbf{q}}{\partial \mathbf{x}}\right|_{\mathbf{x}_0} \mathbf{x}(t) + \mathbf{j}(\mathbf{x}_0) + \left.\frac{\partial \mathbf{j}}{\partial \mathbf{x}}\right|_{\mathbf{x}_0} \mathbf{x}(t) = \mathbf{e}_0 + \mathbf{e}(t).$$

But $\mathbf{q}(\mathbf{x}_0)$ is constant, and therefore $\frac{d}{dt}\mathbf{q}(\mathbf{x}_0) = 0$. The Jacobian matrix $\left.\frac{\partial \mathbf{q}}{\partial \mathbf{x}}\right|_{\mathbf{x}_0}$ does not depend on time. Thus we have

$$\left.\frac{\partial \mathbf{q}}{\partial \mathbf{x}}\right|_{\mathbf{x}_0} \dot{\mathbf{x}}(t) + \mathbf{j}(\mathbf{x}_0) + \left.\frac{\partial \mathbf{j}}{\partial \mathbf{x}}\right|_{\mathbf{x}_0} \mathbf{x}(t) = \mathbf{e}_0 + \mathbf{e}(t).$$

From the operating-point equations it follows that $\mathbf{j}(\mathbf{x}_0) = \mathbf{e}_0$. Hence the above equation reduces to

$$\left.\frac{\partial \mathbf{q}}{\partial \mathbf{x}}\right|_{\mathbf{x}_0} \dot{\mathbf{x}}(t) + \left.\frac{\partial \mathbf{j}}{\partial \mathbf{x}}\right|_{\mathbf{x}_0} \mathbf{x}(t) = \mathbf{e}(t).$$

Using the notation

$$\mathbf{C} = \left.\frac{\partial \mathbf{q}}{\partial \mathbf{x}}\right|_{\mathbf{x}_0} \quad \text{and} \quad \mathbf{G} = \left.\frac{\partial \mathbf{j}}{\partial \mathbf{x}}\right|_{\mathbf{x}_0},$$

we can write the system of linear ordinary differential equations describing the "small-signal" activity of the circuit as follows:

$$\mathbf{C}\dot{\mathbf{x}}(t) + \mathbf{G}\mathbf{x}(t) = \mathbf{e}(t).$$

This system of equations is amenable to various analyses that are important and provide precious information and insight to circuit designers. These analyses are the *complex phasor analysis* (also called, for historical reasons, AC (from alternating current) analysis) and various forms of exact and approximate *pole-zero analysis*.

2.3. Transient Analysis

Designers want to predict the behavior of their circuits with device models and signals that describe as realistically as possible the environment for which they are designed. Therefore, in most cases, there is no escape but to perform a full time-domain (also called *transient*) simulation of the circuit. In other words, this means solving numerically the system of differential-algebraic equations

$$\dot{\mathbf{q}}(\mathbf{x}(t)) + \mathbf{j}(\mathbf{x}(t)) = \mathbf{e}(t).$$

In most cases, the initial state of the circuit is assumed known, $\mathbf{x}(t_0) = \mathbf{x}_0$, and we must solve an *initial-value* problem. There are important applications where we know that the circuit behaves periodically. In this case, we only know that the initial state of the circuit is equal to its state at a time equal to the period, $\mathbf{x}(t_0) = \mathbf{x}(t_0 + T)$. In this case, we have to solve a *two-point boundary-value* problem.

3. DC Analysis

As we have seen, the computation of the DC operating point of a circuit requires the solution of a systems of equation that are, in general, nonlinear. We now discuss the solution of such nonlinear equations.

3.1. Solution of Nonlinear Equations

The general problem can be stated as follows. Let

$$\mathbf{f} : \ \mathcal{D} \mapsto \mathbb{R}^n$$

be a given function of the form

$$\mathbf{f}(\mathbf{x}) = \begin{bmatrix} f_1(x_1, x_2, \ldots, x_n) \\ f_2(x_1, x_2, \ldots, x_n) \\ \vdots \\ f_n(x_1, x_2, \ldots, x_n) \end{bmatrix}, \quad \text{where} \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix},$$

on some subset $\mathcal{D} \subseteq \mathbb{R}^n$. Note that $\mathbf{x}$ is a vector of length n with components x_j, $j = 1, 2, \ldots, n$, and that $\mathbf{f}$ is a vector-valued function with real-valued component functions

$$f_j : \ \mathcal{D} \mapsto \mathbb{R}, \quad j = 1, 2, \ldots, n.$$

We wish to find a vector $\mathbf{x} \in \mathcal{D}$ that is a solution of the equation

$$\mathbf{f}(\mathbf{x}) = 0.$$

Such a vector $\mathbf{x}$ is called a *zero* of $\mathbf{f}$.

In general, at least some of the component functions are nonlinear, and the equation $\mathbf{f}(\mathbf{x}) = 0$ is called a *system of nonlinear equations*. We obtain linear systems

$$\mathbf{M}\mathbf{x} = \mathbf{b}$$

as a special case of $\mathbf{f}(\mathbf{x}) = 0$, by setting

$$\mathcal{D} = \mathbb{R}^n \quad \text{and} \quad \mathbf{f}(\mathbf{x}) = \mathbf{M}\mathbf{x} - \mathbf{b}.$$

Another important special case is $n = 1$, and then the system $\mathbf{f}(\mathbf{x}) = 0$ reduces to a single nonlinear equation, $f(x) = 0$, for one real unknown $x \in \mathcal{D} \subseteq \mathbb{R}$.

While a linear system $\mathbf{M}\mathbf{x} = \mathbf{b}$ always has a unique solution provided that the matrix $\mathbf{M}$ is nonsingular, a nonlinear system $\mathbf{f}(\mathbf{x}) = 0$ can have multiple solutions or need not have a solution at all.

3.2. Newton's Method

In general, there is no finite procedure to find a zero of a nonlinear function $\mathbf{f}$, and all algorithms are infinite procedures that generate a whole sequence of iterates:

$$\mathbf{x}^{(0)} \to \mathbf{x}^{(1)} \to \cdots \to \mathbf{x}^{(i)} \to \cdots .$$

Under suitable conditions, this sequence will converge to a zero of $\mathbf{f}$.

The standard algorithm for computing zeros of nonlinear functions is *Newton's method.*

We assume that all component functions f_j of $\mathbf{f}$ are differentiable with respect to all components x_k of $\mathbf{x}$. The corresponding derivatives

$$\frac{\partial f_j(\mathbf{x})}{\partial x_k}, \quad j,k = 1,2,\ldots,n,$$

are called the (first) *partial derivatives* of $\mathbf{f}$, and the $n \times n$ matrix

$$D\mathbf{f}(\mathbf{x}) = \left[\frac{\partial f_j(\mathbf{x})}{\partial x_k} \right]_{j,k=1,2,\ldots,n} = \begin{bmatrix} \frac{\partial f_1(\mathbf{x})}{\partial x_1} & \frac{\partial f_1(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial f_1(\mathbf{x})}{\partial x_n} \\ \frac{\partial f_2(\mathbf{x})}{\partial x_1} & \frac{\partial f_2(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial f_2(\mathbf{x})}{\partial x_n} \\ \vdots & \vdots & & \vdots \\ \frac{\partial f_n(\mathbf{x})}{\partial x_1} & \frac{\partial f_n(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial f_n(\mathbf{x})}{\partial x_n} \end{bmatrix}$$

is called the *Jacobian matrix* for $\mathbf{f}$ at the point $\mathbf{x}$. For the derivation of Newton's method, we also assume that the *second partial derivatives*

$$\frac{\partial^2 f_j(\mathbf{x})}{\partial x_l \partial x_k} = \frac{\partial}{\partial x_l}\left(\frac{\partial f_j(\mathbf{x})}{\partial x_k}\right), \quad j,k,l = 1,2,\ldots,n,$$

exist and are continuous.

Let $\mathbf{x}^{(i)}$ be an approximate zero of $\mathbf{f}$. Our goal is to compute a next improved approximate zero $\mathbf{x}^{(i+1)}$.

By Taylor's theorem for functions of several variables and since $\mathbf{f}$ is assumed to have continuous second partial derivatives, we have the expansion

$$\mathbf{f}(\mathbf{x}) = \mathbf{f}(\mathbf{x}^{(i)}) + D\mathbf{f}(\mathbf{x}^{(i)})(\mathbf{x} - \mathbf{x}^{(i)}) + \mathcal{O}\left(\|\mathbf{x} - \mathbf{x}^{(i)}\|_2^2\right)$$

about the point $x^{(i)}$. Here, the symbol $\mathcal{O}(h^2)$ means that there are numbers $C_i \geq 0$ and $h_i > 0$ such that

$$\|\mathcal{O}(h^2)\|_2 \leq C_i h^2 \quad \text{for all} \quad 0 \leq h \leq h_i.$$

In particular, if $\mathbf{x} \to \mathbf{x}^{(i)}$, then

$$\lim_{\mathbf{x} \to \mathbf{x}^{(i)}} \mathcal{O}\left(\|\mathbf{x} - \mathbf{x}^{(i)}\|_2^2\right) = 0.$$

By omitting the quadratic term $\mathcal{O}\left(\|\mathbf{x}-\mathbf{x}^{(i)}\|_2^2\right)$ in the above expansion, we obtain the linear approximation

$$\mathbf{l}(\mathbf{x}) = \mathbf{f}(\mathbf{x}^{(i)}) + D\mathbf{f}(\mathbf{x}^{(i)})(\mathbf{x}-\mathbf{x}^{(i)})$$

to the function $\mathbf{f}(\mathbf{x})$ at the point $\mathbf{x}^{(i)}$. Newton's method now simply consists of taking the zero of the linear function $\mathbf{l}(\mathbf{x})$ as the next iterate $\mathbf{x}^{(i+1)}$. Therefore, $\mathbf{x}^{(i+1)}$ is defined as the solution of

$$0 = \mathbf{l}(\mathbf{x}^{(i+1)}) = \mathbf{f}(\mathbf{x}^{(i)}) + D\mathbf{f}(\mathbf{x}^{(i)})(\mathbf{x}^{(i+1)}-\mathbf{x}^{(i)}).$$

For this equation to have a solution, we need to assume that the matrix $D\mathbf{f}(\mathbf{x}^{(i)})$ is nonsingular. If this is the case, then $\mathbf{x}^{(i+1)}$ is given by

$$\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)} + \mathbf{d}^{(i)}$$

where $\mathbf{d}^{(i)}$ is the solution of the linear system

$$D\mathbf{f}(\mathbf{x}^{(i)})\mathbf{d}^{(i)} = -\mathbf{f}(\mathbf{x}^{(i)}).$$

Therefore, Newton's method requires the solution of a linear system with coefficient matrix $D\mathbf{f}(x^{(i)})$ at each ith step. Any linear systems solver can be used for the solution of this system. The systems that arise in circuit simulation are typically large and sparse, and mostly, sparse Gaussian elimination is used for their solution.

If the matrix $D\mathbf{f}(\mathbf{x}^{(i)})$ happens to be singular, then Newton's method has to be stopped at the ith step. We stress that Gaussian elimination detects singularity of the coefficient matrix $\mathbf{M}$ of a linear system $\mathbf{M}\mathbf{x} = \mathbf{b}$. More precisely, $\mathbf{M}$ is singular if, and only if, Gaussian elimination encounters a submatrix $\widetilde{\mathbf{M}}^{(i)}$ whose elements are all zero.

We note that, for a single function

$$f:\ \mathcal{D} \mapsto \mathbb{R}, \quad \text{where} \quad \mathcal{D} \subseteq \mathbb{R},$$

of one variable x, we have $Df(x^{(i)}) = f'(x^{(i)})$, and the solution of

$$f'(x^{(i)})d^{(i)} = -f(x^{(i)}$$

is simply

$$d^{(i)} = -\frac{f(x^{(i)})}{f'(x^{(i)})}.$$

In particular, the iterates in Newton's method are given by

$$x^{(i+1)} = x^{(i)} - \frac{f(x^{(i)})}{f'(x^{(i)})}, \quad i = 0, 1, \dots .$$

Newton's method (in its simplest form) for the solution of systems of nonlinear equations can be summarized as follows.

Algorithm (Newton's method)

- `Choose an initial guess` $\mathbf{x}^{(0)} \in \mathbb{R}^n$ `for a zero` $\mathbf{x}$ `of the function` $\mathbf{f}$.
- `For` $i = 0, 1, \ldots,$ `do:`
  - `If` $\mathbf{x}^{(i)}$ `has converged, then stop.`
  - `If` $D\mathbf{f}(\mathbf{x}^{(i)})$ `is singular, then stop.`
  - `Solve the linear system`

    $$D\mathbf{f}(\mathbf{x}^{(i)})\mathbf{d}^{(i)} = -\mathbf{f}(\mathbf{x}^{(i)})$$

    `for` $\mathbf{d}^{(i)}$.
  - `Set`

    $$\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)} + \mathbf{d}^{(i)}.$$

Convergence Check

The test for convergence is usually based on the following three checks:

- Stop if $i = i_{\max}$, where $i_{\max}$ is the maximal number of Newton steps that the user allows. This check is a safeguard, since, in general, there is no guarantee that Newton's method will actually converge.
- Stop if

  $$\|\mathbf{f}(\mathbf{x}^{(i)})\|_2 \ \leq \ \texttt{FTOL}.$$

  Here, `FTOL` is some small tolerance.
- Stop if $i \geq 1$ and

  $$\|\mathbf{x}^{(i)} - \mathbf{x}^{(i-1)}\|_2 \ \leq \ \texttt{XTOL}.$$

  Here, `XTOL` is some small tolerance.

3.3. Convergence of Newton's Method

Let $\mathbf{x}^{(i)} \in \mathbb{R}^n$, $i = 1, 2, \ldots$, be a sequence of vectors. Assume that the sequence is convergent with limit

$$\mathbf{x} = \lim_{i \to \infty} \mathbf{x}^{(i)}.$$

If there is a constant $c_x < 1$ such that

$$\|\mathbf{x}^{(i+1)} - \mathbf{x}\|_2 \ \leq \ c_x \|\mathbf{x}^{(i)} - \mathbf{x}\|_2 \quad \text{for all} \quad i = 0, 1, \ldots,$$

then the sequence is said to be *linearly convergent.*

If there is a constant c_x such that

$$\|\mathbf{x}^{(i+1)} - \mathbf{x}\|_2 \; \leq \; c_x \|\mathbf{x}^{(i)} - \mathbf{x}\|_2^2 \quad \text{for all} \quad i = 0, 1, \ldots,$$

then the sequence is said to be *quadratically convergent.* Clearly, quadratic convergence is faster than linear convergence. Furthermore, if a sequence is quadratically convergent and $c_x \approx 1$, then the number of digits in which $\mathbf{x}^{(i)}$ and $\mathbf{x}$ agree roughly doubles from step i to step $i + 1$.

A sequence $\mathbf{x}^{(i)} \in \mathbb{R}^n$, $i = 1, 2, ...$, is said to converge *locally* if it converges to some vector $\mathbf{x} \in \mathbb{R}^n$ for all initial vectors $\mathbf{x}^{(0)}$ that are close to $\mathbf{x}$. Here, "close" means that there exists some $\epsilon > 0$ such that the sequence will converge for all $\mathbf{x}$ with $\|\mathbf{x}^{(0)} - \mathbf{x}\|_2 \leq \epsilon$.

Finally, a point $\mathbf{x} \in \mathcal{D}$, where $\mathcal{D} \subseteq \mathbb{R}^n$, is called an *interior point* of $\mathcal{D}$ if there exists some $\epsilon > 0$ such that

$$\mathbf{y} \in \mathcal{D} \quad \text{for all} \quad \mathbf{y} \in \mathbb{R}^n \quad \text{with} \quad \|\mathbf{y} - \mathbf{x}\|_2 \; \leq \; \epsilon.$$

Newton's method as stated in the above algorithm is locally quadratically convergent. In the following theorem, we give a precise statement of this convergence result.

Theorem. Let $\mathbf{f} : \; \mathcal{D} \mapsto \mathbb{R}^n$, where $\mathcal{D} \subseteq \mathbb{R}^n$, be a function with continuous first and second partial derivatives. Let $\mathbf{x} \in \mathcal{D}$ be a zero of $\mathbf{f}$, and assume that $\mathbf{x}$ is an interior point of $\mathcal{D}$ and that the Jacobian matrix $D\mathbf{f}(\mathbf{x})$ of $\mathbf{f}$ at the point $\mathbf{x}$ is nonsingular. Then, there exists an $\epsilon > 0$ such that for all initial points $\mathbf{x}^{(0)}$ with

$$\|\mathbf{x}^{(0)} - \mathbf{x}\|_2 \; \leq \; \epsilon,$$

the iterates

$$\mathbf{x}^{(i)}, \quad i = 0, 1, \ldots,$$

generated by Newton's method have the following properties:

- $\mathbf{x}^{(i)} \in \mathcal{D}$ for all $i = 0, 1, \ldots$;
- The Jacobian matrices $D\mathbf{f}(\mathbf{x}^{(i)})$ are nonsingular for all $i = 0, 1, \ldots$;
- The iterates converge quadratically to $\mathbf{x}$.

The condition that the Jacobian matrix $D\mathbf{f}(\mathbf{x})$ at the zero $\mathbf{x}$ be nonsingular is crucial to get quadratic convergence. If the matrix $D\mathbf{f}(\mathbf{x})$ is singular, but $D\mathbf{f}(\mathbf{y})$ is nonsingular for all $\mathbf{y} \neq \mathbf{x}$ close to $\mathbf{x}$, then Newton's method is still locally convergent. However, in this case, convergence is only linear.

3.4. Problems with Newton's Method

Newton's method is only locally convergent, and, in general, convergence cannot be guaranteed if the initial guess $\mathbf{x}^{(0)}$ is not close to a zero of the function.

For example, consider the function $f : \mathbb{R} \mapsto \mathbb{R}$ given by $f(x) = \arctan x$, which has exactly one zero, namely $x = 0$. Newton's method applied to this function yields the iteration

$$x^{(i+1)} = x^{(i)} - (1 + x^{(i)})^2 \arctan x^{(i)}.$$

One can show that if $x^{(0)}$ satisfies

$$\arctan \left| x^{(0)} \right| \; > \; \frac{2 \left| x^{(0)} \right|}{1 + (x^{(0)})^2},$$

then the Newton iterates diverge:

$$\lim_{i \to \infty} \left| x^{(i)} \right| \; = \; \infty.$$

A remedy for the nonconvergence problem is to combine Newton's method with a suitable line search strategy.

Newton's Method with Damping

As before, at each step of Newton's method, we again solve the linear

$$D\mathbf{f}(x^{(i)})\mathbf{d}^{(i)} = -\mathbf{f}(\mathbf{x}^{(i)})$$

to obtain the *Newton direction* $\mathbf{d}^{(i)}$. However, instead of simply adding $\mathbf{d}^{(i)}$ to the old iterate $\mathbf{x}^{(i)}$, we now incorporate *damping* and set

$$\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)} + \alpha_i \mathbf{d}^{(i)}$$

where $0 < \alpha_i \leq 1$ is a line-search parameter. It is chosen such that, at least, we have

$$\|\mathbf{f}(\mathbf{x}^{(i+1)})\|_2 \; < \; \|\mathbf{f}(\mathbf{x}^{(i)})\|_2.$$

This condition means that we are making progress towards our goal of finding a zero $\mathbf{x}$; note that any zero $\mathbf{x}$ of $\mathbf{f}$ satisfies

$$\|\mathbf{f}(\mathbf{x})\|_2 \; = \; 0.$$

It is always possible to find an α_i such that the above condition is satisfied. Setting

$$g(\alpha_i) = \|\mathbf{f}(\mathbf{x}^{(i)} + \alpha_i \mathbf{d}^{(i)})\|_2^2,$$

the above condition is equivalent to

$$g(0) > g(\alpha_i).$$

We have

$$\begin{aligned}\frac{dg}{d\alpha_i}(0) &= 2\left(\mathbf{f}(x^{(i)})\right)^T D\mathbf{f}(x^{(i)})\mathbf{d}^{(i)} \\ &= -2\left(\mathbf{f}(x^{(i)})\right)^T \mathbf{f}(x^{(i)}) \\ &= -2\|\mathbf{f}(x^{(i)})\|_2^2 \;<\; 0,\end{aligned}$$

and hence, $g(0) > g(\alpha_i)$ can always be achieved by choosing $\alpha_i > 0$ small enough.

Ideally, α_i would be chosen such that $\|\mathbf{f}(\mathbf{x}^{(i+1)})\|_2$ is as small as possible, i.e., α_i is the solution of the one-dimensional minimization problem

$$\min_{0<\alpha_i\leq 1} \|\mathbf{f}(\mathbf{x}^{(i)} + \alpha_i\mathbf{d}^{(i)})\|_2.$$

In general, solving this minimization problem exactly is computationally expensive, and instead approximate solutions are used.

A simple strategy is to try $\alpha_i = 2^{-j}$, $j = 0, 1, \ldots$, and to use the first value of j for which

$$\|\mathbf{f}(\mathbf{x}^{(i)} + 2^{-j}\mathbf{d}^{(i)})\|_2 \;<\; \|\mathbf{f}(\mathbf{x}^{(i)})\|_2.$$

This results in the following Newton's method with damping.

Algorithm (Newton's method with damping)

- `Choose an initial guess` $\mathbf{x}^{(0)} \in \mathbb{R}^n$ `for a zero` $\mathbf{x}$ `of the function` $\mathbf{f}$.
- `For` $i = 0, 1, \ldots,$ `do:`
    - `If` $\mathbf{x}^{(i)}$ `has converged, then stop.`
    - `If` $D\mathbf{f}(\mathbf{x}^{(i)})$ `is singular, then stop.`
    - `Solve the linear system`

      $$D\mathbf{f}(\mathbf{x}^{(i)})\mathbf{d}^{(i)} = -\mathbf{f}(\mathbf{x}^{(i)})$$

      `for` $\mathbf{d}^{(i)}$.
    - `Determine the smallest value of` $j = 0, 1, 2, \ldots,$ `for which`

      $$\|\mathbf{f}(\mathbf{x}^{(i)} + 2^{-j}\mathbf{d}^{(i)})\|_2 \;<\; \|\mathbf{f}(\mathbf{x}^{(i)})\|_2.$$

    - `Set`

      $$\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)} + 2^{-j}\mathbf{d}^{(i)}.$$

Device Models and Overflow

In general, the function $\mathbf{f} : \mathcal{D} \mapsto \mathbb{R}^n$ is only defined on a subset $\mathcal{D} \subset \mathbb{R}^n$, but not on all of $\mathbb{R}^n$. In Newton's method, it can happen that

$$\mathbf{x}^{(i)} \in \mathcal{D}, \quad \text{but} \quad \mathbf{x}^{(i+1)} \notin \mathcal{D},$$

and then $\mathbf{f}(\mathbf{x}^{(i+1)})$ would not even be defined. Again, this problem can be remedied by using damping and by adding the constraint

$$\mathbf{x}^{(i)} + \alpha_i \mathbf{d}^{(i)} \in \mathcal{D}$$

to the line-search procedure.

In the context of solving nonlinear circuit equations, the above constraint simply means that one stays within the region of validity of the device models. However, even if this is done, problems with overflow can occur. In particular, this is the case for devices with exponential characteristics, such as the diode. Here the device model is

$$I = I_{sat} \left(\exp \left(\frac{qV}{kT} \right) - 1 \right),$$

and two of the components of the ith Newton iterate $\mathbf{x}^{(i)}$ represent approximate values $V^{(i)}$ and $I^{(i)}$ for V and I. If the ith Newton step yields $V^{(i+1)} > V^{(i)}$, then—due to the rapid growth of the exponential function—overflow may occur when evaluating the diode equation at the $(i+1)$st Newton iterate. This problem can be remedied by suitable modifications of the device models, or by adding further constraints to the line search. Techniques of the latter type are referred to as *limiting methods*.

To apply Newton's method, we need to have that the function $\mathbf{f}$ has at least continuous first partial derivatives, so that the Jacobian matrix $D\mathbf{f}(\mathbf{x})$ exists and is continuous. Noncontinuity of the partial derivatives can lead to nonconvergence in Newton's method (even with damping). Therefore, it is crucial that only device models with continuous first derivatives are used.

Reducing the Computational Costs of Newton's Method

In each ith step of Newton's method, we need to compute

$$\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)} - \alpha_i \left(D\mathbf{f}(\mathbf{x}^{(i)}) \right)^{-1} \mathbf{f}(\mathbf{x}^{(i)}).$$

Here $0 < \alpha_i \leq 1$ is a suitably chosen step length, and $\alpha_i = 1$ if no damping is used. The computationally dominating parts of each Newton step are:

- The evaluation of the Jacobian matrix $D\mathbf{f}(x^{(i)})$;

- The solution of the linear system

$$Df(\mathbf{x}^{(i)})\mathbf{d}^{(i)} = -\mathbf{f}(\mathbf{x}^{(i)})$$

to obtain the Newton direction $\mathbf{d}^{(i)}$.

We now discuss various possibilities to reduce the computational costs for these two subproblems. First, we consider the solution of the linear system for the Newton direction.

Reducing the Costs of the Linear-Systems Solves

The sparsity structure of the Jacobian matrix $D\mathbf{f}(\mathbf{x}^{(i)})$ is the same throughout the Newton iteration, and only the numerical values of the entries of $D\mathbf{f}(\mathbf{x}^{(i)})$ change with i. This can be exploited when a sparse Gaussian elimination algorithm is used for the linear-systems solves.

A typical sparse Gaussian elimination algorithm has two phases: first a *symbolic* factorization is performed to determine a reordering of the unknowns and the equations, with the goal to minimize fill-in of original zero elements in an actual *numerical* factorization. Second, the numerical factorization is performed on the reordered system.

Since the sparsity structure does not change with i, in most iterations one can skip the symbolic factorization and only perform the numerical factorization. Indeed, one simply re-uses the reordering from the last step during which a symbolic factorization was performed.

A further reduction of work can be achieved by simply re-using the LU factorization of some old Jacobian $D\mathbf{f}(\mathbf{x}^{(i_{\mathrm{old}})})$, $i_{\mathrm{old}} < i$, for the next few, say l, steps. The resulting algorithm, however, is no longer Newton's method, and instead we perform the following iteration:

$$\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)} - \alpha_i \left(D\mathbf{f}(\mathbf{x}^{(i_{\mathrm{old}})})\right)^{-1} \mathbf{f}(\mathbf{x}^{(i)}), \quad i = i_{\mathrm{old}}, i_{\mathrm{old}}+1, \ldots, i_{\mathrm{old}}+l-1.$$

In circuit simulation, this process of "freezing" an old Jacobian for a number of steps is called the *bypass algorithm*. There, in addition, also function values that have not changed significantly are re-used in order to also reduce the costs of evaluating the nonlinear parts of $\mathbf{f}$. More precisely, the old value $f_j(\mathbf{x}^{(i-1)})$ of the kth component function f_j of $\mathbf{f}$ is used instead of the actual value $f_j(\mathbf{x}^{(i)})$ if the entries of $\mathbf{x}^{(i-1)}$ and $\mathbf{x}^{(i)}$ on which f_j depends are almost identical. (Note that, for the functions $\mathbf{f}$ arising in circuit simulation, each component f_j depends only on a few entries of $\mathbf{x}$.)

3.5. Computation of the Jacobian Matrix

We now discuss options for computing the entries of the Jacobian matrix

$$D\mathbf{f}(\mathbf{x}) = \left[\frac{\partial f_j(\mathbf{x})}{\partial x_k}\right]_{j,k=1,2,\ldots,n}.$$

Note that $D\mathbf{f}(\mathbf{x})$ is usually a sparse matrix. The reason is that each function f_j typically depends on only a few components of $\mathbf{x}$, and hence

$$\frac{\partial f_j(\mathbf{x})}{\partial x_k} = 0 \quad \text{for all vectors } \mathbf{x},$$

for all components x_k on $\mathbf{x}$ on which f_j does not depend. Clearly, we only have to compute the "nontrivial" partial derivatives for which

$$\frac{\partial f_j(\mathbf{x})}{\partial x_k} \neq 0 \quad \text{for at least some vectors } \mathbf{x}.$$

One possibility is to simply ask the user to provide all nontrivial partial derivatives of $\mathbf{f}$, along with the function values $\mathbf{f}(\mathbf{x})$, for all device models. However, this makes the models considerably more complicated, and it is also a potential source of error in that some of the provided derivatives may not be correct.

A more practical approach is to compute the derivatives numerically, and two such techniques are *numerical differentiation* and *automatic differentiation*. We now discuss numerical differentiation. Automatic differentiation will be described in more detail in Section 3.6. below.

3.5.1. Numerical Differentiation

The idea of numerical differentiation is to approximate derivatives by finite differences.

Partial derivatives can be defined as follows:

$$\frac{\partial f_j(\mathbf{x})}{\partial x_k} = \lim_{\substack{h \to 0 \\ h > 0}} \frac{f_j(\mathbf{x} + h\mathbf{e}_k) - f_j(\mathbf{x})}{h},$$

where

$$\mathbf{e}_k = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \begin{array}{l} \\ \\ \\ \leftarrow k \\ \\ \\ \\ \end{array}$$

is the kth unit vector in $\mathbb{R}^n$. This definition suggests to use

$$\frac{\partial f_j(\mathbf{x})}{\partial x_k} \approx \frac{f_j(\mathbf{x} + h\mathbf{e}_k) - f_j(\mathbf{x})}{h},$$

for some small $h > 0$, as an approximation. This approximation is called the *forward-difference* formula. Note that $f_j(\mathbf{x})$ is usually evaluated anyway, and so the forward-difference formula requires one additional function evaluation, $f_j(\mathbf{x} + h\mathbf{e}_k)$, for the approximation of each partial derivative.

Using Taylor's theorem, it immediately follows that the approximation error of the forward-difference formula is $\mathcal{O}(h)$. Hence, it is tempting to use a very small h in order to try to obtain a highly accurate approximation. On the other hand, for very small h, we have

$$f_j(\mathbf{x} + h\mathbf{e}_k) \approx f_j(\mathbf{x}),$$

and thus large roundoff error, due to catastrophic cancellation, may occur when computing the numerator of the forward-difference formula. Hence the total error of the forward-difference formula is given by

$$\text{total error} = \text{approximation error} + \text{roundoff error}.$$

Here, the approximation error decreases with decreasing $h > 0$, while the round-off error increases with decreasing $h > 0$, and thus reducing the approximation error and the roundoff error are conflicting goals. This is the dilemma of numerical differentiation. There are heuristics for choosing a value h that approximately minimizes the total error. However, the choice of a good value of h is quite tricky, and it involves estimates for higher derivatives.

Using Taylor's theorem, it is possible to construct higher-order finite difference formulas with approximation error of order $\mathcal{O}(h^p)$, $p = 2, 3, \ldots$. However, these formulas also require a higher number of function evaluations. For $p = 2$, the resulting approximation is

$$\frac{\partial f_j(\mathbf{x})}{\partial x_k} \approx \frac{f_j(\mathbf{x} + h\mathbf{e}_k) - f_j(\mathbf{x} - h\mathbf{e}_k)}{2h},$$

for some small $h > 0$. This approximation is called the *central-difference* formula. The approximation error of the central-difference formula is $\mathcal{O}(h^2)$, and thus it is more accurate than the forward-difference formula. On the other hand, it is also twice as expensive, as it requires two additional function evaluations for each partial derivatives, compared to one function evaluation for the forward-difference formula.

3.5.2. Construction of the Jacobian Matrix from the Stamps

Another possibility is to construct the Jacobian Matrix directly from the circuit stamps.

We now illustrate this technique, using the equation of the bipolar Ebers-Moll model of a bipolar transistor as an example.

Recall that the contribution to a nodal or modified nodal formulation of the bipolar Ebers-

Moll model is as follows:

$$
\begin{aligned}
KCLe:\quad & 0 = \cdots + I_{ES}\left(e^{\dfrac{v_b - v_e}{v_T}} - 1\right) - \alpha_R I_{CS}\left(e^{\dfrac{v_b - v_c}{v_T}} - 1\right) + \cdots\ , \\
KCLc:\quad & 0 = \cdots - \alpha_F I_{ES}\left(e^{\dfrac{v_b - v_e}{v_T}} - 1\right) + I_{CS}\left(e^{\dfrac{v_b - v_c}{v_T}} - 1\right) + \cdots\ , \\
KCLb:\quad & 0 = \cdots - (1-\alpha_F) I_{ES}\left(e^{\dfrac{v_b - v_e}{v_T}} - 1\right) - (1-\alpha_R) I_{CS}\left(e^{\dfrac{v_b - v_c}{v_T}} - 1\right) + \cdots\ .
\end{aligned}
$$

The equations $KCLe$, $KCLc$, $KCLb$, and the node voltages v_e, v_c, v_b, are three of the n equations $\mathbf{f} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{bmatrix}$ and the n unknowns $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$ respectively.

Recall that each step of Newton's method is based on the first order linearization

$$\mathbf{f}(\mathbf{x}^{(i)}) + D\mathbf{f}(\mathbf{x}^{(i)})\mathbf{d}^{(i)} = 0.$$

In consequence, the contribution of the bipolar transistor equations to the linearized system that needs to be solved is

$$
\begin{aligned}
KCLe:\quad & 0 = \cdots - i_e - \sum_{k=1}^{n} \frac{\partial i_e}{\partial x_k} d_k + \cdots\ , \\
KCLc:\quad & 0 = \cdots - i_c - \sum_{k=1}^{n} \frac{\partial i_c}{\partial x_k} d_k + \cdots\ , \\
KCLb:\quad & 0 = \cdots + i_e + \sum_{k=1}^{n} \frac{\partial i_e}{\partial x_k} d_k + i_c + \sum_{k=1}^{n} \frac{\partial i_c}{\partial x_k} d_k + \cdots\ .
\end{aligned}
$$

Here the currents and their derivatives are all evaluated at the current point in the Newton iteration. Note that the only unknowns x_k, $k = 1, 2, \ldots, k$ that appear in the transistor current expressions are the nodal voltages of the three transistor terminals, v_e, v_c, v_b, therefore only the partial derivatives with respect to them will be nonzero.

$$
\begin{aligned}
KCLe:\quad & 0 = \cdots - i_e - \frac{\partial i_e}{\partial v_e} d_e - \frac{\partial i_e}{\partial v_c} d_c - \frac{\partial i_e}{\partial v_b} d_b + \cdots\ , \\
KCLc:\quad & 0 = \cdots - i_c - \frac{\partial i_c}{\partial v_e} d_e - \frac{\partial i_c}{\partial v_c} d_c - \frac{\partial i_c}{\partial v_b} d_b + \cdots\ , \\
KCLb:\quad & 0 = \cdots + i_c + \frac{\partial i_c}{\partial v_e} d_e + \frac{\partial i_c}{\partial v_c} d_c + \frac{\partial i_c}{\partial v_b} d_b + i_e + \frac{\partial i_e}{\partial v_e} d_e + \frac{\partial i_e}{\partial v_c} d_c + \frac{\partial i_e}{\partial v_b} d_b + \cdots\ .
\end{aligned}
$$

Using the notation $g_{ee} = \dfrac{\partial i_e}{\partial v_e}$, $g_{ec} = \dfrac{\partial i_e}{\partial v_c}$, etc., the corresponding stamp is as follows:

$$\begin{bmatrix} \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & -g_{ee} & \cdots & -g_{ec} & \cdots & -g_{eb} & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & -g_{ce} & \cdots & -g_{cc} & \cdots & -g_{cb} & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & g_{ee}+g_{ce} & \cdots & g_{ec}+g_{cc} & \cdots & g_{eb}+g_{cb} & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \end{bmatrix} \begin{bmatrix} \cdots \\ d_e \\ \cdots \\ d_c \\ \cdots \\ d_b \\ \cdots \end{bmatrix} = \begin{bmatrix} \cdots \\ i_e \\ \cdots \\ i_c \\ \cdots \\ -i_e - i_c \\ \cdots \end{bmatrix}.$$

For a device that is not voltage controlled the procedure is similar:

$$\begin{array}{ll} KCLn+: & 0 = \cdots - i_w + \cdots \ , \\ KCLn-: & 0 = \cdots + i_w + \cdots \ , \\ BCR: & 0 = \phi(v_{n+} - v_{n-}, i_w). \end{array}$$

In this case, i_w is also an unknown and we need to consider the partial derivative with respect to it. The stamp is as follows:

$$\begin{bmatrix} \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & -1 & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & 1 & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \dfrac{\partial \phi}{\partial v_w} & \cdots & -\dfrac{\partial \phi}{\partial v_w} & \cdots & \dfrac{\partial \phi}{\partial i_w} & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \end{bmatrix} \begin{bmatrix} \cdots \\ d_{n+} \\ \cdots \\ d_{n+} \\ \cdots \\ d_{iw} \\ \cdots \end{bmatrix} = \begin{bmatrix} \cdots \\ i_e \\ \cdots \\ i_c \\ \cdots \\ -i_e - i_c \\ \cdots \end{bmatrix}.$$

Note that, in the derivation of the above stamp, we used the chain rule:

$$\frac{\partial \phi}{\partial v_{n+}} = \frac{\partial \phi}{\partial v_w}\frac{\partial v_w}{\partial v_{n+}} = \frac{\partial \phi}{\partial v_w} \quad \text{and} \quad \frac{\partial \phi}{\partial v_{n-}} = \frac{\partial \phi}{\partial v_w}\frac{\partial v_w}{\partial v_{n-}} = -\frac{\partial \phi}{\partial v_w}.$$

Finally, we can state an algorithm for solving nonlinear circuits using a Newton-based method.

Algorithm (The nonlinear DC simulation algorithm)

- `Choose an initial guess` $\mathrm{x}^{(0)}$ `for a zero of the function` $\mathbf{f}$. `The initial vector may be the zero vector, or some circuit intuition can be used to obtain a better initial solution. (Examples: bias all diodes to 0.6V, use a previous result if available, etc.)`

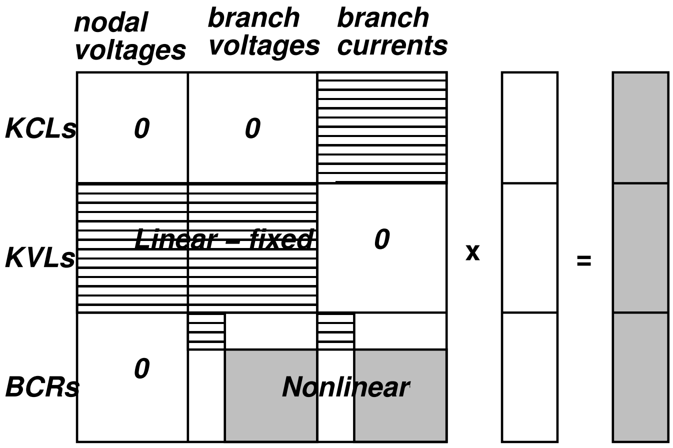
- `For` $i = 0, 1, \ldots,$ `do:`

- For all the devices in the circuit, evaluate the models at the current point, $\mathbf{x}^{(i)}$, and stamp the results into the Jacobian matrix $D\mathbf{f}(x^{(i)})$ and the right-hand side $-\mathbf{f}(\mathbf{x}^{(i)})$.
- If $D\mathbf{f}(x^{(i)})$ is singular, then stop.
- Solve the linear system

  $$D\mathbf{f}(x^{(i)})\mathbf{d}^{(i)} = -\mathbf{f}(\mathbf{x}^{(i)})$$

  for $\mathbf{d}^{(i)}$.
- Determine the next value $\mathbf{x}^{(i+1)}$, calculated from $\mathbf{x}^{(i)}$ and $\mathbf{d}^{(i)}$ according to the desired strategy.
- If $x^{(i+1)}$ has converged, then stop.

In a tableau formulation, one can take advantage of the particular structure of the matrix to save computations. This is illustrated in the following figure:



The portion of the matrix corresponding to linear equations remains the same for every iteration, and only the nonlinear part is changing. One way to take advantage of this fact is to partially factor the matrix. ASTAP, the IBM proprietary circuit simulator, which is based on the tableau formulation, used such a strategy.

3.6. Automatic Differentiation

Almost always, a function is given in the form of a numerical algorithm whose input is x and whose output is the corresponding function value $f(x)$. The basic idea of *automatic differentiation* is to differentiate all the arithmetic operations performed in the algorithm so that, at the end, one obtains the derivative of the function, in addition to the function value. The name "automatic differentiation" is not quite fortunate, and better names would be "algorithmic differentiation" or "computational differentiation". However, the technique was first called "automatic differentiation", and that name has been kept.

For simplicity, we only consider scalar functions,

$$f:\ \mathcal{D} \mapsto \mathbb{R}, \quad \mathcal{D} \subseteq \mathbb{R},$$

of one variable. We assume that f is given as an algorithm that, for any input $x \in \mathcal{D}$, generates the function value

$$y = f(x)$$

as output. Here, an algorithm consists of a sequence of basic operations ($+$, $-$, $\times$, $/$, $\sqrt{\ }$, log, exp, sin, cos, ...), and it may also contain conditional branches. For example, the function

$$y = f(x) = \cos\left(\exp(x) + \sqrt{x}\right)$$

is given by an algorithm consisting of the following four basic operations:

$$\begin{aligned} u &= \exp(x), \\ v &= \sqrt{x}, \\ w &= u + v, \\ y &= \cos(w). \end{aligned}$$

Conditional branches are needed for functions that are pieced together, such as the function

$$y = f(x) = \begin{cases} e^{-x^2} & \text{for} \quad x \geq 0, \\ 1 & \text{for} \quad x < 0. \end{cases}$$

Automatic differentiation extends the algorithm for computing $y = f(x)$ by differentiating the statements in the algorithm, using the chain rule and known derivatives of the basic operations, so that the algorithm also returns $f'(x)$, in addition of $f(x)$.

$y = f(x)$	$f'(x)$
$a + x$	1
$a - x$	-1
$a \times x$	a
a/x	$-y/x$
$\sqrt{x}$	$1/(2y)$
$\log(x)$	$1/x$
$\exp(x)$	y
$\sin(x)$	$\cos(x)$
$\cos(x)$	$-\sin(x)$
$\vdots$	$\vdots$

Differentiation of basic operations

For example, by differentiating the above algorithm for the function

$$y = f(x) = \cos\left(\exp(x) + \sqrt{x}\right),$$

we obtain the following extended algorithm:

$$\begin{aligned} u &= \exp(x) & u_x &= u \\ v &= \sqrt{x} & v_x &= \frac{1}{2v} \\ w &= u + v & w_x &= u_x + v_x \\ y &= \cos(w) & y_x &= -w_x \sin(w) \end{aligned}$$

Here, and in the sequel, we use the notation

$$f_x = \frac{df(x)}{dx}, \quad g_u = \frac{\partial g(u,v)}{\partial u}, \quad \text{and} \quad g_v = \frac{\partial g(u,v)}{\partial v}.$$

In this example, we have done the differentiation of the statements "from top to bottom", which is just the order they appear in the algorithm for evaluating $f(x)$. This is called the *forward* or *direct mode* of automatic differentiation. As we will see shortly, there is also a *reverse* or *adjoint* mode of automatic differentiation.

The process of differentiating the statements in an algorithms can be formalized, and there is software to do this differentiation and to produce code for the extended algorithm "automatically".

We now consider such a more formal description for the case of our example function $f(x) = \cos(\exp(x) + \sqrt{x})$. This function can be written as a decomposition of basic operations as follows:

$$y = f(x) = d\left(c\left(a(x), b(x)\right)\right),$$

where

$$a(x) = \exp(x), \quad b(x) = \sqrt{x}, \quad c(u,v) = u + v, \quad \text{and} \quad d(w) = \cos(w).$$

The evaluation of such a decomposition can be done by the following algorithm:

$$\begin{aligned} u &= a(x), \\ v &= b(x), \\ w &= c(u,v), \\ y &= d(w). \end{aligned}$$

Differentiating these statements gives the following relations (written in differential form):

$$\begin{aligned} du &= a'(x)dx, \\ dv &= b'(x)dx, \\ dw &= \left(\tfrac{\partial c(u,v)}{\partial u} a'(x) + \tfrac{\partial c(u,v)}{\partial v} b'(x)\right) dx, \\ dy &= d'(w) \left(\tfrac{\partial c(u,v)}{\partial u} a'(x) + \tfrac{\partial c(u,v)}{\partial v} b'(x)\right) dx. \end{aligned}$$

Therefore, the following statements generate $y_x = f'(x)$ by automatic differentiation in forward mode:

$$\begin{aligned} u_x &= a'(x)dx, \\ v_x &= b'(x)dx, \\ w_x &= \tfrac{\partial c(u,v)}{\partial u} u_x + \tfrac{\partial c(u,v)}{\partial v} v_x, \\ y_x &= d'(w)w_x. \end{aligned}$$

In reverse mode, we differentiate the statements from "bottom to top", which is just the reverse of the order they appear in the algorithm for evaluating $y = f(x)$. In the case of the function $y = f(x) = d(c(a(x), b(x)))$, this gives:

$$\begin{aligned} dy &= d'(w)dw, \\ dy &= \left(d'(w)\tfrac{\partial c(u,v)}{\partial u}\right) du + \left(d'(w)\tfrac{\partial c(u,v)}{\partial v} dv\right), \\ dy &= \left(d'(w)\tfrac{\partial c(u,v)}{\partial u}\right) du + \left(d'(w)\tfrac{\partial c(u,v)}{\partial v} b'(x)\right) dx, \\ dy &= \left(d'(w)\tfrac{\partial c(u,v)}{\partial u} a'(x) + d'(w)\tfrac{\partial c(u,v)}{\partial v} b'(x)\right) dx. \end{aligned}$$

Therefore, the following statements generate $y_x = f'(x)$ by automatic differentiation in reverse mode:

$$\begin{aligned} y_w &= d'(w), \\ y_u &= y_w \tfrac{\partial c(u,v)}{\partial u}, \\ y_v &= y_w \tfrac{\partial c(u,v)}{\partial v}, \\ y_x &= y_u a'(x) + y_v b'(x). \end{aligned}$$

The main advantage of the reverse mode is that it is more flexible. In particular, in reverse mode, it is possible to compute any linear combination of partial derivatives, without generating the partial derivatives themselves. For example, in reverse mode, one can compute matrix-vector products

$$\mathbf{z} = (D\mathbf{f}(\mathbf{x}))^T \cdot \mathbf{v}, \quad \text{where} \quad \mathbf{v} \in \mathbb{R}^n,$$

with the transpose of the Jacobian, without ever generating the Jacobian itself. This is not possible in forward mode.

On the other hand, in general, additional storage is required for the reverse mode, as intermediate quantities generated in the course of the function evaluation may need to be stored for later use when the differentiated statements are evaluated (in reverse order of the original statements).

Unlike numerical differentiation, there is no approximation error for automatic differentiation. In fact, in exact arithmetic, automatic differentiation would generate the exact derivatives. The only error that occurs is roundoff error due to the use of floating-point arithmetic.

Finally, we remark that automatic differentiation is not to be confused with *symbolic differentiation*, which is a different technique. Given an arithmetic formula for a function, symbolic differentiation produces an arithmetic formula for the derivative.

3.7. Homotopy Methods

We now return to nonlinear functions

$$\mathbf{f} : \mathcal{D} \mapsto \mathbb{R}^n,$$

where $\mathcal{D} \subseteq \mathbb{R}^n$, and our goal is again to find a zero $\mathbf{x}$ of $\mathbf{f}$. The basic idea of a *continuation method* is to embed the function $\mathbf{f}$ into a whole family (depending on a real parameter λ) of functions

$$\mathbf{H}(\cdot, \lambda) : \mathcal{D} \mapsto \mathbb{R}^n, \quad \text{where} \quad \lambda_{\min} \leq \lambda \leq \lambda_{\max},$$

such that it is "easy" to find a zero of

$$\mathbf{H}(\mathbf{x}, \lambda_{\min}) = 0,$$

while

$$\mathbf{H}(\mathbf{x}, \lambda_{\max}) = \mathbf{f}(\mathbf{x}) = 0$$

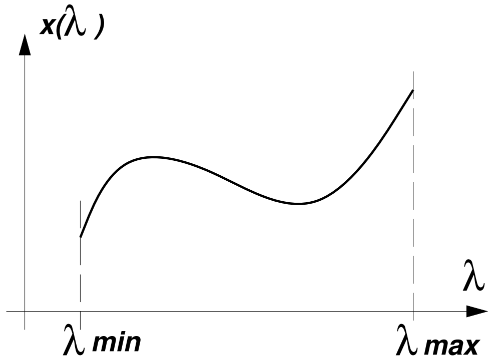
is the equation we really wish to solve. For each fixed λ, we denote by $\mathbf{x}(\lambda)$ a solution of the equation

$$\mathbf{H}(\mathbf{x}, \lambda) = 0.$$

Under suitable conditions, the points

$$\begin{bmatrix} \mathbf{x}(\lambda) \\ \lambda \end{bmatrix}, \quad \lambda_{\min} \leq \lambda \leq \lambda_{\max},$$

define a curve in $(n+1)$-dimensional space. The starting point $\mathbf{x}(\lambda_{\min})$ of this curve is the solution of the "easy" equation $\mathbf{H}(\mathbf{x}, \lambda_{\min}) = 0$, while the end point $\mathbf{x}(\lambda_{\max})$ is the desired solution of $\mathbf{f}(\mathbf{x}) = 0$. A continuation method tries to follow this curve from $\mathbf{x}(\lambda_{\min})$ to $\mathbf{x}(\lambda_{\max})$. This requires the solution of a sequence of nonlinear equations $\mathbf{H}(\mathbf{x}, \lambda) = 0$. The advantage of a continuation method over simply solving $\mathbf{f}(\mathbf{x}) = 0$ is that for each of the nonlinear equations we have a very good initial guess (namely, the previously computed $\mathbf{x}(\lambda_{\text{old}})$), and thus we can expect very fast convergence when solving $\mathbf{H}(\mathbf{x}, \lambda) = 0$. In the following figure, we show a typical curve for $n = 1$.



Examples for continuation functions $\mathbf{H}(\mathbf{x}, \lambda)$

1) Suppose $\mathbf{f}_0$ is a function such that it is easy to solve

$$\mathbf{f}_0(\mathbf{x}) = 0.$$

For instance, $\mathbf{f}_0$ could be the function that describes a simplified version of the nonlinear circuit whose DC operating point we want to compute by solving $\mathbf{f}(\mathbf{x}) = 0$. Then,

$$\mathbf{H}(\mathbf{x}, \lambda) = \lambda \mathbf{f}(\mathbf{x}) + (1 - \lambda)\mathbf{f}_0(\mathbf{x}), \quad 0 \le \lambda \le 1.$$

2) Let $\mathbf{x}^{(0)}$ be an initial guess for $\mathbf{f}(\mathbf{x}) = 0$. If $\mathbf{x}^{(0)}$ is not close to a zero of $\mathbf{f}$, then Newton's method may not converge. However, one can use $\mathbf{x}^{(0)}$ as the starting point for a continuation process by setting

$$\mathbf{H}(\mathbf{x}, \lambda) = \mathbf{f}(\mathbf{x}) + (\lambda - 1)\mathbf{f}(\mathbf{x}^{(0)}), \quad 0 \le \lambda \le 1.$$

Then, the equation

$$\mathbf{H}(\mathbf{x}, 0) = \mathbf{f}(\mathbf{x}) - \mathbf{f}(\mathbf{x}^{(0)}) = 0$$

has the trivial solution $\mathbf{x}^{(0)}$.

3) Often, λ can be chosen as an actual physical parameter, such as, e.g., an excitation voltage.

Next, we describe the structure of continuation algorithms. An outline of the basic method is as follows.

Algorithm (Basic continuation method)

- `Set` $\lambda = \lambda_{\min}$.
- `Choose an initial guess` $\mathbf{x}_{\text{old}}$ `for the solution of` $\mathbf{H}(\mathbf{x}, \lambda_{\min}) = 0$.
- `While` $\lambda \le \lambda_{\max}$ `do:`

- `Solve the nonlinear equation`
$$\mathbf{H}(\mathbf{x}, \lambda) = 0$$
`for x, using` $\mathbf{x}_{\text{old}}$ `as initial guess.`
- `Set`
$$\mathbf{x}_{\text{old}} = \mathbf{x}.$$
- `Choose a suitable step length` $\Delta\lambda > 0$ `and set`
$$\lambda = \lambda + \Delta\lambda.$$

Instead of simple using the old point $\mathbf{x}_{\text{old}}$ as initial guess, it is also possible to obtain a better *prediction* for the next point $\mathbf{x}$ by using tangential information. Set

$$\mathbf{x}'(\lambda) = \frac{d\mathbf{x}(\lambda)}{d\lambda}, \quad \mathbf{H}_\lambda(\mathbf{x}, \lambda) = \frac{\partial \mathbf{H}(\mathbf{x}, \lambda)}{\partial \lambda},$$

and denote by

$$D_x \mathbf{H}(\mathbf{x}, \lambda)$$

the Jacobian of $\mathbf{H}(\mathbf{x}, \lambda)$ with respect to $\mathbf{x}$. Then, by differentiating the identity

$$\mathbf{H}(\mathbf{x}(\lambda), \lambda) = 0, \quad \lambda_{\text{min}} \le \lambda \le \lambda_{\text{max}},$$

with respect to λ, we obtain

$$D_x \mathbf{H}(\mathbf{x}(\lambda), \lambda)\, \mathbf{x}'(\lambda) + \mathbf{H}_\lambda(\mathbf{x}(\lambda), \lambda) = 0.$$

Hence, by solving the linear system

$$D_x \mathbf{H}(\mathbf{x}(\lambda), \lambda)\, \mathbf{t} = -\mathbf{H}_\lambda(\mathbf{x}(\lambda), \lambda),$$

we can compute the tangent vector

$$\mathbf{x}'(\lambda) = \mathbf{t}$$

at the point $\mathbf{x}(\lambda)$. Using this vector, we obtain the predictor

$$\mathbf{x}(\lambda) + \Delta\lambda \mathbf{t}$$

for the next point $\mathbf{x}(\lambda + \Delta\lambda)$. This predictor is then "corrected" by solving

$$\mathbf{H}(\mathbf{x}, \lambda + \Delta\lambda) = 0$$

for $\mathbf{x}$, using $\mathbf{x}(\lambda) + \Delta\lambda \mathbf{t}$ as initial guess.

The resulting *predictor-corrector* method can be sketched as follows.

Algorithm (Predictor-corrector method)

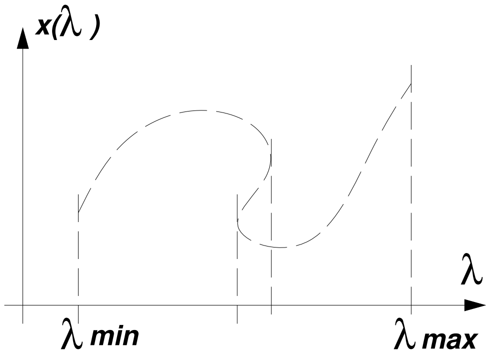
- `Set` $\lambda = \lambda_{\min}$.

- `Choose an initial guess` $\mathbf{x}_{\text{old}}$ `for the solution of` $\mathbf{H}(\mathbf{x}, \lambda_{\min}) = 0$.
- `While` $\lambda \leq \lambda_{\max}$ `do:`
  - `Solve the nonlinear equation`
  $$\mathbf{H}(\mathbf{x}, \lambda) = 0$$
  `for` $\mathbf{x}$, `using` $\mathbf{x}_{\text{old}}$ `as initial guess.`
  - `Solve the linear system`
  $$D_x \mathbf{H}(\mathbf{x}, \lambda)\, \mathbf{t} = -\mathbf{H}_\lambda(\mathbf{x}, \lambda),$$
  `for` $\mathbf{t}$.
  - `Choose a suitable step length` $\Delta\lambda > 0$ `and set`
  $$\begin{aligned} \lambda &= \lambda + \Delta\lambda, \\ \mathbf{x}_{\text{old}} &= \mathbf{x} + \Delta\lambda \mathbf{t}. \end{aligned}$$

The continuation methods described so far can only handle those problems for which $\mathbf{x}(\lambda)$ is a single-valued function of λ. However, this is not always the case. Indeed, the curve

$$\begin{bmatrix} \mathbf{x}(\lambda) \\ \lambda \end{bmatrix}, \quad \lambda_{\min} \leq \lambda \leq \lambda_{\max},$$

may have *turning points*, and then $\mathbf{x}(\lambda)$ becomes a multi-valued function of λ in between turning points. In the following curve, we show a curve with two turning points for the case $n = 1$.



Curves with turning points cannot be parametrized by λ, and in particular, continuation with respect to λ is no longer possible. The remedy is to use a different parametrization that always exists, even if the curve has turning points. Such a parametrization is so-called *arc length*, which is always guaranteed to exist if the curve is differentiable. Roughly speaking, arc length is the actual length of the curve one obtains by measuring along the curve. More precisely, arc-length parametrization of the curve is given by

$$\begin{bmatrix} \mathbf{x}(s) \\ \lambda(s) \end{bmatrix}, \quad 0 \le s \le s_0,$$

where $\mathbf{x}(s)$ and $\lambda(s)$ satisfy the arc-length condition

$$\|\dot{\mathbf{x}}(s)\|_2^2 + |\dot{\lambda}(s)|^2 = 1.$$

Here, $\dot{\mathbf{x}}(s)$ and $\dot{\lambda}(s)$ denote the derivatives of $\mathbf{x}(s)$ and $\lambda(s)$ with respect to s. Moreover, s_0 is the total length of the curve. Also, note that now both $\mathbf{x}(s)$ and $\lambda(s)$ are functions of the parameter s. Instead of solving $\mathbf{H}(\mathbf{x}, \lambda) = 0$ for $\mathbf{x}$, we now have to solve the *augmented* system

$$\begin{aligned} \mathbf{H}(\mathbf{x}(s), \lambda(s)) &= 0, \\ \|\dot{\mathbf{x}}(s)\|_2^2 + |\dot{\lambda}(s)|^2 &= 1, \end{aligned}$$

for $\mathbf{x}(s)$ and $\lambda(s)$. Note that this augmented equation involves a differential equation, namely the arc-length condition, and thus its solution is considerably more complicated than the solution of $\mathbf{H}(\mathbf{x}, \lambda) = 0$. The remedy to this is to use a finite-difference approximation of the arc-length condition, and then the augmented system is again a system of nonlinear equation, without any differential equation. The resulting parametrization is no longer arc-length s, but is described by so-called *pseudo* arc length σ, which in some sense approximates the true arc length s. The nonlinear system we now need to solve is of the form

$$\begin{aligned} \mathbf{H}(\mathbf{x}(\sigma), \lambda(\sigma)) &= 0, \\ N(\mathbf{x}(\sigma), \lambda(\sigma); \Delta\sigma) &= 0. \end{aligned}$$

Here, $N(\mathbf{x}, \lambda; \Delta\sigma) = 0$ is the approximation to the arc-length condition.

Suppose we already have a solution

$$\mathbf{x}_{\text{old}} = \mathbf{x}(\sigma_{\text{old}}) \quad \text{and} \quad \lambda_{\text{old}} = \lambda(\sigma_{\text{old}})$$

of the augmented system for the parameter value σ_{old}, and let

$$\sigma = \sigma_{\text{old}} + \Delta\sigma, \quad \text{where} \quad \Delta\sigma > 0,$$

be the next parameter value. Then two standard choices for the pseudo-arc-length function N are

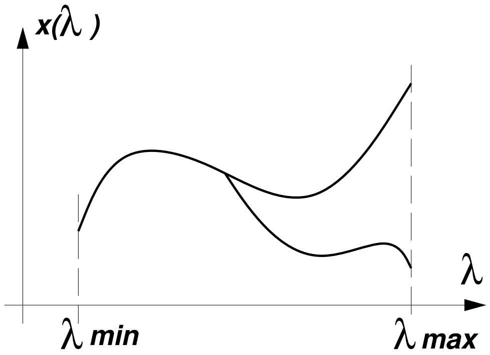
$$N_1(\mathbf{x}, \lambda; \Delta\sigma) = \|\mathbf{x} - \mathbf{x}_{\text{old}}\|_2^2 + |\lambda - \lambda_{\text{old}}|^2 - (\Delta\sigma)^2$$

and

$$N_2(\mathbf{x}, \lambda; \Delta\sigma) = \dot{\mathbf{x}}_{\text{old}}^T(\mathbf{x} - \mathbf{x}_{\text{old}}) + \dot{\lambda}_{\text{old}}(\lambda - \lambda_{\text{old}}) - \Delta\sigma.$$

It can be shown that, as long as $\Delta\sigma > 0$ is chosen small enough, a continuation method (using pseudo arc length σ) based on solving the augmented system (with $N = N_1$ or $N = N_2$) can be used to compute curves with turning points.

However, there is yet another difficulty that may arise, and this is a bifurcation point along the curve for circuits that have multiple operating points. Such a situation (for the case $n = 1$) is shown in the following figure.



It is possible to use continuation methods to detect bifurcation points and to follow the different branches of the curve originating from a bifurcation point. A detailed discussion of such techniques would be beyond the scope of this course.

4. Time-Domain Analysis

In this section, we discuss techniques for the time-domain analysis of circuits. First, we describe the basic problem, the simulation of the transient behavior of circuits. Then we discuss techniques for the numerical solution of the underlying systems of ODEs and DAEs.

4.1. Transient Analysis

The equations describing the transient behavior of a given circuit can be summarized as follows:

$$\dot{\mathbf{q}}(\mathbf{x}) + \mathbf{j}(\mathbf{x}) - \mathbf{e}(t) = 0.$$

For notational convenience, we combine the last two terms, and so the above equation becomes

$$\dot{\mathbf{q}}(\mathbf{x}) + \mathbf{f}(\mathbf{x}, t) = 0.$$

In general, this equation is not an ODE but an DAE, and therefore we cannot solve it for arbitrary initial conditions on $\mathbf{x}$. The initial conditions must satisfy the algebraic constraints. To show this, let us assume that we impose some arbitrary initial conditions, $\mathbf{x}(t_0) = \mathbf{x}_0$. Then, at $t = t_0$, we have

$$\dot{\mathbf{q}}(\mathbf{x}_0) + \mathbf{f}(\mathbf{x}_0, t_0) = 0.$$

We perform the differentiation w.r.t. t by using the chain rule

$$\left.\frac{\partial \mathbf{q}}{\partial \mathbf{x}}\right|_{\mathbf{x}=\mathbf{x}_0} \dot{\mathbf{x}}(t_0) + \mathbf{f}(\mathbf{x}_0, t_0) = 0.$$

This results in the linear system

$$\left.\frac{\partial \mathbf{q}}{\partial \mathbf{x}}\right|_{\mathbf{x}=\mathbf{x}_0} \dot{\mathbf{x}}(t_0) = -\mathbf{f}(\mathbf{x}_0, t_0)$$

for the unknowns $\dot{\mathbf{x}}(t_0)$. In general, this system will have no solution because the matrix $\left.\frac{\partial \mathbf{q}}{\partial \mathbf{x}}\right|_{\mathbf{x}=\mathbf{x}_0}$ is singular.

In order to find a set of acceptable initial conditions, we need to solve a nonlinear system of equations. There are several ways to specify these equations.

Equilibrium initial conditions

In this case, we assume that the state of the circuit for $t < t_0$ is DC steady state, i.e., all circuit variables are constant in time. Then

$$\dot{\mathbf{x}}(t_0) = 0,$$

and in order to find the initial conditions we need to solve the nonlinear system

$$\mathbf{f}(\mathbf{x}_0, t_0) = 0.$$

Note that this is exactly the system of equations $\mathbf{j}(\mathbf{x}_0) = \mathbf{e}(t_0)$ that needs to be solved to find a DC operating point before we perform a small-signal analysis, where the constant sources are set for their value at t_0. This is the nonlinear circuit obtained by replacing capacitors by open circuits and inductors by short circuits.

Non-equilibrium initial conditions

DC steady-state is not natural for all circuits. For example, an oscillator has no DC steady state unless it is in its meta-stable state. Often we want to start the simulation by specifying initial values on the reactive components. This amounts to solving a DC problem that results from a modified circuit topology. For example, in this DC problem, capacitors with specified initial voltages are replaced by voltage sources, and similarly, inductors with the initial current specified are replaced by current source. In this case, $\mathbf{x}_0$ will result from solving a modified system of nonlinear equations

$$\hat{\mathbf{f}}(\mathbf{x}_0, t_0) = 0.$$

These methods address the problem of specifying initial conditions for circuit DAEs. The other problem brought by DAEs, namely high index is in general ignored in circuit simulation. This aspect is still in research phase and there is suspicion that non-convergence in transient simulation is often caused by a high DAE index.

Integration of circuit equations

We can apply the various derivative approximation formulas directly to $\dot{\mathbf{q}}(\mathbf{x})$ instead of $\dot{\mathbf{x}}$. For example, the Backward Euler formula at time t_k with time step h is

$$\dot{\mathbf{q}}(\mathbf{x}_k) \approx \frac{\mathbf{q}(\mathbf{x}_k) - \mathbf{q}(\mathbf{x}_{k-1})}{h}.$$

This approximation yields the following nonlinear system of equations to be solved at time step t_k:

$$\frac{\mathbf{q}(\mathbf{x}_k) - \mathbf{q}(\mathbf{x}_{k-1})}{h} + \mathbf{f}(\mathbf{x}_k, t_k) = 0.$$

The unknowns here are the values of the circuit variables at time t_k, $\mathbf{x}_k$. The equations may also be written in the following form:

$$\frac{1}{h}\mathbf{q}(\mathbf{x}_k) + \mathbf{f}(\mathbf{x}_k, t_k) - \frac{\mathbf{q}(\mathbf{x}_{k-1})}{h} = 0.$$

The last term is a known quantity at time t_k.

The trapezoidal method is applied in a similar way. Here the derivative approximation is implied by the formula

$$\frac{\dot{\mathbf{q}}(\mathbf{x}_k) + \dot{\mathbf{q}}(\mathbf{x}_{k-1})}{2} \approx \frac{\mathbf{q}(\mathbf{x}_k) - \mathbf{q}(\mathbf{x}_{k-1})}{h}.$$

Using $\dot{\mathbf{q}}(\mathbf{x}_k) = -\mathbf{f}(\mathbf{x}_k, t_k)$ and $\dot{\mathbf{q}}(\mathbf{x}_{k-1}) = -\mathbf{f}(\mathbf{x}_{k-1}, t_{k-1})$ this approximation yields the following nonlinear system of equations to be solved at time step t_k:

$$\frac{\mathbf{q}(\mathbf{x}_k) - \mathbf{q}(\mathbf{x}_{k-1})}{h} + \frac{\mathbf{f}(\mathbf{x}_k, t_k) + \mathbf{f}(\mathbf{x}_{k-1}, t_{k-1})}{2} = 0.$$

The unknowns here are again the values of the circuit variables at time t_k, $\mathbf{x}_k$. The equations may also be written in the following form:

$$\frac{2}{h}\mathbf{q}(\mathbf{x}_k) + \mathbf{f}(\mathbf{x}_k, t_k) - \frac{2}{h}\mathbf{q}(\mathbf{x}_{k-1}) + \mathbf{f}(\mathbf{x}_{k-1}, t_{k-1}) = 0.$$

The last two terms are past values computed at time t_{k-1} and are known at time t_k.

We observe a general pattern emerging. The nonlinear equation at time t_k has the form

$$\alpha\mathbf{q}(\mathbf{x}_k) + \mathbf{f}(\mathbf{x}_k, t_k) + \beta = 0,$$

where α is a scalar and β is an n-vector. All linear multi-step integration methods will have such a form. The expressions for α and β depend on the chosen integration method. For example, $\alpha = 1/h$ for the backward Euler method and $\alpha = 2/h$ for the trapezoidal method.

In general, we use a Newton algorithm to solve the nonlinear system of equations. The Jacobian matrix of the equation system is given by

$$\mathbf{J} = \alpha \left.\frac{\partial \mathbf{q}}{\partial \mathbf{x}}\right|_{\mathbf{x}=\mathbf{x}_k} + \left.\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right|_{\mathbf{x}=\mathbf{x}_k}.$$

This representation suggests a modular interface with the circuit simulator. The circuit-evaluation routine need not be aware of the actual integration method. It only needs to be able to compute values for the expressions $\mathbf{q}(\mathbf{x})$, $\mathbf{f}(\mathbf{x})$, and their Jacobian matrices. The integration routine can then be implemented completely independent of the circuit problem, using only these quantities in an abstract way.

4.2. Solution of (Ordinary) Differential Equations

Equations that involve the values and the derivatives of an unknown function are called *differential equations*. The solution of a differential equation is a function, rather than just a number as for the systems of *algebraic equations* we have studied so far. Equations whose solutions are functions of several variables are called *partial* differential equations. Equations whose solutions are functions of a single variable are called *ordinary* differential equations. Mostly, the equations describing circuits are ordinary differential equations whose solutions are functions of one variable, namely the time t. Here, we will only consider ordinary differential equations and we always omit the adjective "ordinary".

4.2.1. The Problem

In its most general form, the problem of solving a *first-order* differential equation is as follows.

Given a function

$$\mathbf{F}: \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R} \mapsto \mathbb{R}^n$$

and initial values

$$\mathbf{x}_0 \in \mathbb{R}^n, \quad t_0 \in \mathbb{R},$$

find a vector-valued function

$$\mathbf{x}(t) \in \mathbb{R}^n, \quad t \geq t_0,$$

such that

$$\begin{aligned} \mathbf{F}(\mathbf{x}(t), \dot{\mathbf{x}}(t), t) &= 0, \quad t \geq t_0, \\ \mathbf{x}(t_0) &= \mathbf{x}_0. \end{aligned}$$

Here and in the sequel,

$$\dot{\mathbf{x}}(t) = \frac{d\mathbf{x}(t)}{dt}$$

denotes the first derivative of $\mathbf{x}(t)$. The above differential equation is said to be of first order, since it only involves the first derivative of the unknown function $\mathbf{x}(t)$. Of course, there are differential equation of higher order that involve also higher derivatives. However, such higher-order differential equations are less important in circuit simulation, and therefore, we will only consider first-order differential equations. Also, we remark that any higher-order differential equation can always be reformulated as a first-order differential equation.

From now on, we will use the following short form of the above formulation of a first-order differential equation:

$$\begin{aligned} \mathbf{F}(\mathbf{x}, \dot{\mathbf{x}}, t) &= 0, \\ \mathbf{x}(t_0) &= \mathbf{x}_0. \end{aligned}$$

Note that we are prescribing the initial value $\mathbf{x}(t_0)$ at time t_0, and therefore, the problem of solving the differential equation is called an *initial-value problem* (IVP).

An IVP in the above general formulation may have no solution at all, or it may have multiple solutions. For example, there is no real function $x(t)$ that would satisfy the differential equation (with $n = 1$)

$$x^2 + \dot{x}^2 + 1 = 0,$$

no matter what the initial-value is. On the other hand, the IVP (with $n = 1$)

$$\begin{aligned} x^2 - \dot{x}^2 &= 0, \\ x(0) &= 1, \end{aligned}$$

has exactly two solutions, $x(t) = \exp(t)$ and $x(t) = \exp(-t)$.

An important special case is differential equations for which the function $\mathbf{F}$ is such that

$$\mathbf{F}(\mathbf{x}, \dot{\mathbf{x}}, t) = \mathbf{f}(\mathbf{x}, t) - \dot{\mathbf{x}}.$$

In this case, the IVP can be rewritten in the following *explicit* form:

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{f}(\mathbf{x}, t), \\ \mathbf{x}(t_0) &= \mathbf{x}_0. \end{aligned}$$

If such an explicit formulation is *not* possible, then the differential equation is said to be in *implicit* form.

For differential equations in explicit form, continuity of the function $\mathbf{f}(\mathbf{x}, t)$ is sufficient to guarantee existence of a solution, at least for all $t > t_0$ that are close to t_0. However, the solution need not be unique. In order to guarantee existence of a solution for *all* $t \geq t_0$ and also uniqueness of the solution, one needs a somewhat stronger form of continuity of $\mathbf{f}$, namely *Lipschitz continuity*.

Theorem. Let $\mathbf{f}(\mathbf{x}, t)$ be continuous for all $\mathbf{x} \in \mathbb{R}^n$ and all $t \geq t_0$, and in addition, let $\mathbf{f}$ be Lipschitz continuous with respect to $\mathbf{x}$, i.e., there exists a constant L (independent of $\mathbf{x}$ and t) such that

$$\|\mathbf{f}(\mathbf{x}, t) - \mathbf{f}(\mathbf{y}, t)\| \ \leq \ L\|\mathbf{x} - \mathbf{y}\|$$

for all $\mathbf{x}$, $\mathbf{y} \in \mathbb{R}^n$ and all $t \geq t_0$. Then, for *any* $\mathbf{x}_0 \in \mathbb{R}^n$, there exists a unique function,

$$\mathbf{x}(t), \quad t \geq t_0,$$

such that

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{f}(\mathbf{x}, t), \\ \mathbf{x}(t_0) &= \mathbf{x}_0. \end{aligned}$$

We remark that Lipschitz continuity is essential in order to guarantee that a solution exists for all $t \geq t_0$. For example, consider the IVP (with $n = 1$),

$$\dot{x} = x^2, \quad x(0) = 1.$$

The solution of this IVP is the function

$$x(t) = \frac{1}{1 - t},$$

which has a pole at $x = 1$, and hence, the solution only exists for $0 \leq t < 1$. The reason is that the function $f(x) = x^2$ for this IVP is not Lipschitz continuous. In fact, we have the relation

$$\begin{aligned} \|f(x) - f(y)\| &= \|x^2 - y^2\| \\ &= \|(x + y)(x - y)\| \\ &= |x + y| \cdot \|x - y\|, \end{aligned}$$

and hence a Lipschitz constant L does not exist, since $|x + y| \to \infty$ as x, $y \to \infty$.

4.2.2. Linear Explicit Differential Equations

An important special case is *linear* explicit differential equations. Here, "linear" means that $\mathbf{f}(\mathbf{x}, t)$ is linear with respect to $\mathbf{x}$, i.e.,

$$\mathbf{f}(\mathbf{x}, t) = \mathbf{A}(t)\mathbf{x} + \mathbf{b}(t),$$

where $\mathbf{A}(t)$ is an $n \times n$ matrix and $\mathbf{b}(t)$ is a vector of length n, both with entries that may still depend on t.

A linear explicit IVP is thus of the following form:

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{A}(t)\mathbf{x} + \mathbf{b}(t), \\ \mathbf{x}(t_0) &= \mathbf{x}_0. \end{aligned}$$

The general solution set of this differential equation is as follows:

$$\mathbf{x}(t) = \mathbf{x}^{(0)} + \gamma_1 \mathbf{x}^{(1)}(t) + \gamma_2 \mathbf{x}^{(2)}(t) + \cdots \gamma_n \mathbf{x}^{(n)}(t), \quad \gamma_1, \gamma_2, \ldots, \gamma_n \in \mathbb{R},$$

where $\mathbf{x}^{(0)}$ is *any* solution of the *inhomogeneous* differential equation,

$$\dot{\mathbf{x}} = \mathbf{A}(t)\mathbf{x} + \mathbf{b}(t),$$

and $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(n)}$ are n linearly independent solutions of the *homogeneous* differential equation,

$$\dot{\mathbf{x}} = \mathbf{A}(t)\mathbf{x}.$$

The numbers $\gamma_1, \gamma_2, \ldots, \gamma_n$ in the above general solution are still arbitrary parameters. By choosing them appropriately, the initial-value condition, $\mathbf{x}(t_0) = \mathbf{x}_0$, can be satisfied. In fact, $\mathbf{x}(t_0) = \mathbf{x}_0$ if, and only if, the numbers $\gamma_1, \gamma_2, \ldots, \gamma_n$ fulfill the system of linear equations:

$$\left[\mathbf{x}^{(1)}(t_0) \quad \mathbf{x}^{(2)}(t_0) \quad \cdots \quad \mathbf{x}^{(n)}(t_0) \right] \begin{bmatrix} \gamma_1 \\ \gamma_2 \\ \vdots \\ \gamma_n \end{bmatrix} = \mathbf{x}_0 - \mathbf{x}^{(0)}(t_0).$$

Since $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(n)}$ are linearly independent, the coefficient matrix of this linear system is nonsingular, and therefore, for each initial value $\mathbf{x}_0$, there is a unique choice of $\gamma_1, \gamma_2, \ldots, \gamma_n$ so that the initial-value condition is satisfied.

4.3. Differential-Algebraic Equations

A differential equation in implicit form, $\mathbf{F}(\mathbf{x}, \dot{\mathbf{x}}, t) = 0$, can be converted into an explicit differential equation if the Jacobian matrix,

$$D_{\dot{\mathbf{x}}}\mathbf{F}(\mathbf{x}, \dot{\mathbf{x}}, t),$$

of $\mathbf{F}$ with respect to $\dot{\mathbf{x}}$ is nonsingular. In fact, if this matrix is nonsingular, then by the implicit-function theorem it follows that $\mathbf{F}(\mathbf{x}, \dot{\mathbf{x}}, t) = 0$ can be solved for $\dot{\mathbf{x}}$, i.e., we have

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, t)$$

with some suitable function $\mathbf{f}$. However, the above Jacobian matrix can indeed be singular, and then a formulation as an explicit differential equation is in general not possible. In particular, this is usually the case for the differential equations arising in circuit simulation.

First, consider the extreme case that

$$\mathbf{F}(\mathbf{x}, \dot{\mathbf{x}}, t) = \mathbf{f}(\mathbf{x}).$$

Note that the Jacobian matrix $D_{\dot{\mathbf{x}}}\mathbf{F}(\mathbf{x}, \dot{\mathbf{x}}, t) = 0$ for all $\mathbf{x}$ and t. In this case, the "differential equation", $\mathbf{F}(\mathbf{x}, \dot{\mathbf{x}}, t) = 0$, reduces to the system of algebraic equations,

$$\mathbf{f}(\mathbf{x}) = 0.$$

Recall that we already discussed Newton-type methods for the solution of such equations. The solution of such a system of algebraic equations is no longer a function, but simply a vector $\mathbf{x} \in \mathbb{R}^n$. In particular, a solution of the "IVP"

$$\mathbf{f}(\mathbf{x}) = 0, \quad \mathbf{x}(t_0) = \mathbf{x}_0,$$

only exists if the initial-value $\mathbf{x} = \mathbf{x}_0$ happens to be a zero of the function $\mathbf{f}$. Consequently, in this case, the IVP only has a solution for very particular initial values $\mathbf{x}_0$.

In general, the equation $\mathbf{F}(\mathbf{x}, \dot{\mathbf{x}}, t) = 0$ is in between the two extreme cases of an explicit differential equation and of a system of algebraic equations, and thus it is referred to as a *differential-algebraic equation* (DAE). In particular, the equations that arise in circuit simulation are usually DAEs.

Example: Often, a DAE can be separated into its differential and its algebraic equations by suitable reorderings of the components of the unknown function $\mathbf{x}$ and of the equations themselves. More precisely, this is the case if, after the reorderings, we have

$$\mathbf{x}(t) = \begin{bmatrix} \mathbf{v}(t) \\ \mathbf{w}(t) \end{bmatrix}$$

and the DAE reads as follows:

$$\begin{aligned} \dot{\mathbf{v}} &= \mathbf{f}(\mathbf{v}, \mathbf{w}, t) \quad \text{(the differential part)}, \\ 0 &= \mathbf{g}(\mathbf{v}, \mathbf{w}, t) \quad \text{(the algebraic differential part)}. \end{aligned}$$

DAEs are more difficult than explicit differential equations, for the following reasons:

- For a DAE, an IVP only has a solution for those initial values $\mathbf{x}_0$ that are *consistent* with the DAE. We have seen this effect already for the extreme case of a system of algebraic equations.
- DAEs can be categorized by their *index*. As we will see later, the efficiency of solution techniques will be affected when applied to DAEs with high index.

Next, we discuss the concept of the index for linear DAEs of the form

$$\begin{aligned}\mathbf{A}\dot{\mathbf{x}} &= \mathbf{B}\mathbf{x} + \mathbf{b}(t),\\ \mathbf{x}(t_0) &= \mathbf{x}_0,\end{aligned}$$

where $\mathbf{A}$ and $\mathbf{B}$ are constant $n \times n$ matrices. If the matrix A is nonsingular, then the DAE can immediately be reduced to an explicit differential equation by multiplying the equation from the left by $\mathbf{A}^{-1}$. Therefore, we now assume that $\mathbf{A}$ is singular, but with at least one eigenvalue different from 0, and then the above IVP is a true DAE. For simplicity, we also assume that the matrix $\mathbf{B}$ is nonsingular, and we remark that this assumption is not essential for the following discussion. Then the above IVP is equivalent to

$$\begin{aligned}\mathbf{B}^{-1}\mathbf{A}\dot{\mathbf{x}} &= \mathbf{x} + \mathbf{B}^{-1}\mathbf{b}(t),\\ \mathbf{x}(t_0) &= \mathbf{x}_0.\end{aligned}$$

Using the Jordan canonical form of $\mathbf{B}^{-1}\mathbf{A}$, it follows that there exists a nonsingular $n \times n$ matrix $\mathbf{T}$ such that

$$\mathbf{B}^{-1}\mathbf{A} = \mathbf{T}^{-1}\begin{bmatrix}\mathbf{C} & 0\\ 0 & \mathbf{N}\end{bmatrix}\mathbf{T},$$

where $\mathbf{C}$ is a nonsingular $m \times m$ matrix for some $1 \leq m < n$ and $\mathbf{N}$ is a singular nilpotent $(n-m) \times (n-m)$ matrix. Here, a square matrix $\mathbf{N}$ is called *nilpotent* if there exists an integer $k \geq 1$ such that

$$\mathbf{N}^k = 0, \quad \text{but} \quad \mathbf{N}^j \neq 0 \quad \text{for all} \quad j = 1, 2, \ldots, k-1.$$

The nilpotency index k of the matrix $\mathbf{N}$ is just the *index* of the above linear DAE. Next, we set

$$\mathbf{T}\mathbf{x}(t) = \begin{bmatrix}\mathbf{v}(t)\\ \mathbf{w}(t)\end{bmatrix} \quad \text{and} \quad \mathbf{T}\mathbf{B}^{-1}\mathbf{b}(t) = \begin{bmatrix}\mathbf{c}(t)\\ \mathbf{d}(t)\end{bmatrix},$$

where $\mathbf{v}(t)$ and $\mathbf{c}(t)$ are vector-valued functions with m components, and $\mathbf{w}(t)$ and $\mathbf{d}(t)$ are vector-valued functions with $n-m$ components. Furthermore, we transform the initial value $\mathbf{x}_0$ accordingly:

$$\mathbf{T}\mathbf{x}_0 = \begin{bmatrix}\mathbf{v}_0\\ \mathbf{w}_0\end{bmatrix}, \quad \text{where} \quad \mathbf{v}_0 \in \mathbb{R}^m,\ \mathbf{w}_0 \in \mathbb{R}^{n-m}.$$

With these new variables, the above DAE can be rewritten in the following decoupled form:

$$\begin{aligned}\mathbf{C}\dot{\mathbf{v}} &= \mathbf{v} + \mathbf{c}(t), \quad \mathbf{v}(t_0) = \mathbf{v}_0,\\ \mathbf{N}\dot{\mathbf{w}} &= \mathbf{w} + \mathbf{d}(t), \quad \mathbf{w}(t_0) = \mathbf{w}_0.\end{aligned}$$

Since the matrix $\mathbf{C}$ is nonsingular, the first of these two equations is equivalent to an explicit linear differential equation, and it has a unique solution for any continuous right-hand side function $\mathbf{c}(t)$ and any initial value $\mathbf{v}_0 \in \mathbb{R}^m$. However, the second equation only has a solution if the right-hand side function $\mathbf{d}(t)$ and the initial value $\mathbf{w}_0$ satisfy some additional conditions. This can be seen as follows. We denote by

$$\mathbf{w}^{(j)}(t) = \frac{d^j \mathbf{w}(t)}{dt^j} \quad \text{and} \quad \mathbf{d}^{(j)}(t) = \frac{d^j \mathbf{d}(t)}{dt^j}$$

the jth derivative of the functions $\mathbf{w}(t)$ and $\mathbf{d}(t)$, respectively. In particular, with this new notation, we can rewrite the equation $\mathbf{N}\dot{\mathbf{w}} = \mathbf{w} + \mathbf{d}(t)$ as follows:

$$\mathbf{w}(t) = \mathbf{N}\mathbf{w}^{(1)}(t) - \mathbf{d}(t).$$

By differentiating this relation $k-1$ times, we obtain that

$$\mathbf{w}^{(j)}(t) = \mathbf{N}\mathbf{w}^{(j+1)}(t) - \mathbf{d}^{(j)}(t), \quad j = 1, 2, \ldots, k-1.$$

Recall that k is the index of the DAE. Using these relations successively for $j = 1, 2, \ldots$, we deduce that

$$\begin{aligned}
\mathbf{w}(t) &= \mathbf{N}\mathbf{w}^{(1)}(t) - \mathbf{d}(t) \\
&= \mathbf{N}^2\mathbf{w}^{(2)}(t) - \mathbf{N}\mathbf{d}^{(1)}(t) - \mathbf{d}(t) \\
&= \mathbf{N}^3\mathbf{w}^{(3)}(t) - \mathbf{N}^2\mathbf{d}^{(2)}(t) - \mathbf{N}\mathbf{d}^{(1)}(t) - \mathbf{d}(t) \\
&\;\;\vdots \\
&= \mathbf{N}^k\mathbf{w}^{(k)}(t) - \mathbf{N}^{k-1}\mathbf{d}^{(k-1)}(t) - \mathbf{N}^{k-2}\mathbf{d}^{(k-2)}(t) - \cdots - \mathbf{N}\mathbf{d}^{(1)}(t) - \mathbf{d}(t).
\end{aligned}$$

Now, by the definition of the index, we have $\mathbf{N}^k = 0$, and thus the last equation reduces to

$$\mathbf{w}(t) = -\mathbf{N}^{k-1}\mathbf{d}^{(k-1)}(t) - \mathbf{N}^{k-2}\mathbf{d}^{(k-2)}(t) - \cdots - \mathbf{N}\mathbf{d}^{(1)}(t) - \mathbf{d}(t).$$

This relation shows that the part $\mathbf{w}(t)$ of the solution of the DAE is uniquely determined by the right-hand side function $\mathbf{d}(t)$. Furthermore, the function $\mathbf{d}(t)$ has to be $k-1$ times differentiable, and not just continuous, for the solution $\mathbf{w}(t)$ to exist. For DAE with indices $k > 1$, this is an additional restriction, which is solely sue to the DAE and which does not occur in the case of explicit differential equations. Finally, note that $\mathbf{w}(t)$ is completely independent of the initial value $\mathbf{w}_0$, and therefore, the DAE only has a solution if the right-hand side function $\mathbf{d}(t)$ and the initial value $\mathbf{w}_0$ satisfy the *consistency condition*

$$\mathbf{w}_0 = -\mathbf{N}^{k-1}\mathbf{d}^{(k-1)}(t_0) - \mathbf{N}^{k-2}\mathbf{d}^{(k-2)}(t_0) - \cdots - \mathbf{N}\mathbf{d}^{(1)}(t_0) - \mathbf{d}(t_0).$$

These additional conditions make the solution of DAEs more difficult than the solution of explicit differential equations. Also, note that often the index of a DAE is not known beforehand, and then it has to be determined in the course of the solution process.

4.4. Numerical Solution of Explicit IVPs

We now discuss techniques for the numerical solution of explicit IVPs of the form

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, t),$$
$$\mathbf{x}(t_0) = \mathbf{x}_0.$$

The simplest methods use a fixed time step $h > 0$, and they compute approximations to the solution of the IVP at the discrete time points

$$t_k = t_0 + hk, \quad k = 0, 1, \ldots .$$

Note that the exact solution at time t_k is $\mathbf{x}(t_k)$, and we denote by

$$\mathbf{x}_k \;\approx\; \mathbf{x}(t_k), \quad k = 0, 1, \ldots,$$

the approximations generated by the numerical method under consideration. Furthermore, we will use the notation

$$\mathbf{f}_k := \mathbf{f}(\mathbf{x}_k, t_k), \quad k = 0, 1, \ldots .$$

The simplest numerical method for the above IVP is *Euler's method* (also called *forward Euler*). Its motivation is as follows. For the exact solution $\mathbf{x}(t)$, we have

$$\dot{\mathbf{x}}(t_k) = \mathbf{f}(\mathbf{x}(t_k), t_k) \quad \text{for all} \quad k.$$

Approximating $\dot{\mathbf{x}}(t_k)$ by means of forward differences, we obtain that

$$\mathbf{x}(t_{k+1}) \;\approx\; \mathbf{x}(t_k) + h\mathbf{f}(\mathbf{x}(t_k), t_k).$$

This suggests to use the recursion

$$\mathbf{x}_{k+1} = \mathbf{x}_k + h\mathbf{f}_k, \quad k = 0, 1, \ldots,$$

to generate the approximate solutions $\mathbf{x}_k$. The numerical method based on this scheme is just Euler's method.

4.5. Linear Multistep Methods

Euler's method is a special case of the general family of linear multistep methods for the numerical solution of explicit IVPs. Let $s \geq 1$ denote an integer. A *linear s-step method* computes approximate solutions $\mathbf{x}_k \approx \mathbf{x}(t_k)$ at the time points $t_k = t_0 + hk$ by means of a recursion of the form

$$\mathbf{x}_{k+s} = \sum_{i=0}^{s-1} \mathbf{x}_{k+i}\alpha_i + h \sum_{i=0}^{s} \mathbf{f}_{k+i}\beta_i, \quad k = 0, 1, \ldots,$$

where $\alpha_0, \alpha_1, \ldots, \alpha_{s-1} \in \mathbb{R}$ and $\beta_0, \beta_1, \ldots, \beta_s \in \mathbb{R}$ are real parameters with

$$\alpha_0^2 + \beta_0^2 > 0.$$

This last condition simply means that the scheme is really an s-step method and not an s'-step method for some $s' < s$.

Clearly, Euler's method is an s-step method with $s = 1$, $\alpha_0 = 1$, $\beta_0 = 1$, and $\beta_1 = 0$. Other special cases are the so-called *backward Euler method* given by

$$\mathbf{x}_{k+1} = \mathbf{x}_k + h\mathbf{f}_{k+1}, \quad k = 0, 1, \ldots,$$

and the *trapezoidal method* given by

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \frac{h}{2}\left(\mathbf{f}_k + \mathbf{f}_{k+1}\right), \quad k = 0, 1, \ldots .$$

A linear s-step method is said to be *explicit* if $\beta_s = 0$ and *implicit* if $\beta_s \neq 0$. Hence Euler's method is explicit, while the backward Euler method and the trapezoidal method are implicit.

Practical Aspects

For the practical use of a linear s-step method, the following two points are important.

1. To start up the recursion, we need initial values for the first $s-1$ approximate solutions $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{s-1}$. Of these, only $\mathbf{x}_0 = \mathbf{x}(t_0)$ is given as part of the IVP. Therefore, if $s > 1$, we somehow need to generate values for $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{s-1}$. This is usually done by using an appropriate s'-step method. More precisely, to compute a value for $\mathbf{x}_i$, where $1 \leq i \leq s-1$, we can use any i-step method, since we already have values for $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{i-1}$. For example, one can always use Euler's method to obtain the initial values for any s-step method.

2. For an implicit s-step method, we need to solve the equation

   $$\mathbf{x}_{k+s} - h\beta_s \mathbf{f}(\mathbf{x}_{k+s}, t_{k+s}) = \sum_{i=0}^{s-1} \left(\mathbf{x}_{k+i}\alpha_i + h\beta_i \mathbf{f}_{k+i}\right)$$

   in order to obtain the next approximate solution $\mathbf{x}_{k+s}$. In general, this is a system of nonlinear equations, and we need to employ Newton's method or variants thereof for its solution. As we discussed earlier, it is crucial to have a good initial guess $\mathbf{x}_{k+s}^{(0)}$ for $\mathbf{x}_{k+s}$ in order to guarantee quadratic convergence of Newton's method. Such an initial guess can be generated by means of an explicit s-step method. More precisely, we set

   $$\mathbf{x}_{k+s}^{(0)} = \sum_{i=0}^{s-1} \left(\mathbf{x}_{k+i}\tilde{\alpha}_i + h\tilde{\beta}_i \mathbf{f}_{k+i}\right),$$

where the $\tilde{\alpha}_i$'s and the $\tilde{\beta}_i$'s denote the parameters of the explicit s-step method. Using $\mathbf{x}_{k+s}^{(0)}$ as an initial guess, we now apply Newton's method to the above equation resulting in the iterates

$$\mathbf{x}_{k+s}^{(j)}, \quad j = 0, 1, \dots .$$

Finally, when Newton's method has converged, we set $\mathbf{x}_{k+s} = \mathbf{x}_{k+s}^{(j)}$. This approach of combining an implicit s-step method with an explicit s-step method for generating initial values for Newton's method is called a *predictor-corrector method*. The explicit s-step method is used to compute the *prediction* $\mathbf{x}_{k+s}^{(0)}$ that is then corrected by means of Newton's method to obtain the final value $\mathbf{x}_{k+s}$ defined by the implicit s-step method. For example, the explicit Euler's method combined with the implicit trapezoidal method is a predictor-corrector method with $s = 1$.

4.6. Construction of Linear Multistep Methods

There are a number of possibilities to construct linear multistep methods. We now discuss two approaches, namely multistep methods derived from integration rules and from numerical differentiation formulas.

First, we consider integration. By integrating the differential equation, $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, t)$, we obtain

$$\begin{aligned}
\mathbf{x}(t_{k+s}) - \mathbf{x}(t_{k+s_0}) &= \int_{t_{k+s_0}}^{t_{k+s}} \dot{\mathbf{x}}(t)\, dt \\
&= \int_{t_{k+s_0}}^{t_{k+s}} \mathbf{f}((\mathbf{x}(t), t)\, dt,
\end{aligned}$$

where $0 \le s_0 < s$ are integers. Replacing $\mathbf{x}(t_{k+s})$ by the approximation $\mathbf{x}_{k+s}$, $\mathbf{x}(t_{k+s_0})$ by the approximation $\mathbf{x}_{k+s_0}$, and using an integration rule of the type

$$\begin{aligned}
\int_{t_{k+s_0}}^{t_{k+s}} \mathbf{f}((\mathbf{x}(t), t)\, dt &\approx h \sum_{i=0}^{s} \mathbf{f}(\mathbf{x}(t_{k+i}), t_{k+i}) \beta_i \\
&\approx h \sum_{i=0}^{s} \mathbf{f}_{k+i} \beta_i,
\end{aligned}$$

where $\beta_0, \beta_1, \dots, \beta_s \in \mathbb{R}$, we obtain the following s-step method for the solution of IVPs:

$$\mathbf{x}_{k+s} = \mathbf{x}_{k+s_0} + h \sum_{i=0}^{s} \mathbf{f}_{k+i} \beta_i, \quad k = 0, 1, \dots .$$

For example, when $s_0 = 0$, $s = 1$, and the trapezoidal rule is chosen as the integration rule, we just obtain the trapezoidal method for solving IVPs.

Next, we consider multistep methods based on numerical differentiation. Here, we replace the left-hand side of the relation

$$\dot{\mathbf{x}}(t_{k+s_0}) = \mathbf{f}(\mathbf{x}(t_{k+s_0}), t_{k+s_0})$$

by a numerical differentiation formula that consists of a linear combination of the approximate values $\mathbf{x}_{k+i} \approx \mathbf{x}(t_{k+i})$, $i = 0, 1, \ldots, s$, where $0 \leq s_0 \leq s$ are assumed to be integers. Any such linear combination can be represented in the form

$$\dot{\mathbf{x}}(t_{k+s_0}) \;\approx\; \frac{1}{h\beta_{s_0}} \left(\mathbf{x}_{k+s} - \sum_{i=0}^{s-1} \mathbf{x}_{k+i}\alpha_i \right).$$

Together with the approximation $\mathbf{f}(\mathbf{x}(t_{k+s_0}), t_{k+s_0}) \approx \mathbf{f}_{k+s_0}$ for the right-hand side in the above relation, this results in the formula

$$\frac{1}{h\beta_{s_0}} \left(\mathbf{x}_{k+s} - \sum_{i=0}^{s-1} \mathbf{x}_{k+i}\alpha_i \right) = \mathbf{f}_{k+s_0}.$$

Rewriting this relation, we obtain the following s-step method for the solution of IVPs:

$$\mathbf{x}_{k+s} = \sum_{i=0}^{s-1} \mathbf{x}_{k+i}\alpha_i + h\beta_{s_0}\mathbf{f}_{k+s_0}, \quad k = 0, 1, \ldots\,.$$

An important case is $s_0 = s$, with the coefficients β_s and $\alpha_0, \alpha_1, \ldots, \alpha_{s-1}$ chosen such that the numerical differentiation formula has the highest-possible order. The resulting s-step methods are called *backward-differentiation formulas* (BDFs). In the following table, we list the coefficients of the BDFs for $s = 1, 2, \ldots, 6$. Note that the BDF with $s = 1$ is just the backward Euler method.

s	β_s	α_0	α_1	α_2	α_3	α_4	α_5
1	1	1					
2	$\frac{2}{3}$	$\frac{-1}{3}$	$\frac{4}{3}$				
3	$\frac{6}{11}$	$\frac{2}{11}$	$\frac{-9}{11}$	$\frac{18}{11}$			
4	$\frac{12}{25}$	$\frac{-3}{25}$	$\frac{16}{25}$	$\frac{-36}{25}$	$\frac{48}{25}$		
5	$\frac{60}{137}$	$\frac{12}{137}$	$\frac{-75}{137}$	$\frac{200}{137}$	$\frac{-300}{137}$	$\frac{300}{137}$	
6	$\frac{60}{147}$	$\frac{-10}{147}$	$\frac{72}{147}$	$\frac{-225}{147}$	$\frac{400}{147}$	$\frac{-450}{147}$	$\frac{360}{147}$

Coefficients of BDFs for $s = 1, 2, \ldots, 6$

4.7. Local Truncation Error

So far, we have introduced a number of linear multistep methods. We now begin our discussion of properties of linear multistep methods.

A linear multistep method computes an approximate solution of an IVP, and clearly, it is desirable that the approximation error is as small as possible. The first important concept here is the *local truncation error* (LTE). Roughly speaking, the LTE measures the approximation error that is introduced when one step of the s-step method performed, assuming that all approximate solutions computed up to that point are exact. More precisely, assume that

$$\mathbf{x}_{k+i} = \mathbf{x}(t_{k+i}) \quad \text{for all} \quad i = 0, 1, \ldots, s-1.$$

Then, the LTE (at time t_{k+s}) measures how close $\mathbf{x}_{k+s}$ is to $\mathbf{x}(t_{k+s})$ in the following sense:

$$\mathrm{LTE} = \frac{\mathbf{x}(t_{k+s}) - \mathbf{x}_{k+s}}{h}.$$

Using the formula for the s-step method and setting $t = t_k$, it can be shown that

$$\mathrm{LTE} = \frac{1}{h}\left(\mathbf{x}(t+hs) - \sum_{i=0}^{s-1} \mathbf{x}(t+ih)\alpha_i\right) - \sum_{i=0}^{s} \dot{\mathbf{x}}(t+ih)\beta_i.$$

Note that, for each fixed t, the LTE is a function of the time step h, and we can look at its Taylor expansion about $h = 0$. This expansion is of the form

$$\mathrm{LTE} = c_p h^p \mathbf{x}^{(p+1)}(t) + \cdots = \mathcal{O}(h^p)$$

where $p \geq 0$ is an integer, $c_p \neq 0$ is a constant, and

$$\mathbf{x}^{(p+1)}(t) = \frac{d^{p+1}\mathbf{x}(t)}{dt^{p+1}}$$

denotes the $(p+1)st$ derivative of $\mathbf{x}(t)$. The integer p is called the *order* of the multistep method. Clearly, as the time step h goes to zero, the LTE should also go to zero. This is the case if, and only if, $p \geq 1$. A multistep method is said to be *consistent* if it has order $p \geq 1$.

Example: Consider the trapezoidal method, which is an s-step method with $s = 1$, $\alpha_0 = 1$, and $\beta_0 = \beta_1 = 1/2$. Hence, we have

$$\mathrm{LTE} = \frac{\mathbf{x}(t+h) - \mathbf{x}(t)}{h} - \frac{\dot{\mathbf{x}}(t) + \dot{\mathbf{x}}(t+h)}{2}.$$

Using the Taylor expansions of $\mathbf{x}(t+h)$ and $\dot{\mathbf{x}}(t+h)$ about $h = 0$, it follows that

$$\begin{aligned}\mathrm{LTE} &= \mathbf{x}^{(1)}(t) + \mathbf{x}^{(2)}(t)\tfrac{h}{2} + \mathbf{x}^{(3)}(t)\tfrac{h^2}{6} + \cdots - \tfrac{1}{2}\left(2\mathbf{x}^{(1)}(t) + \mathbf{x}^{(2)}(t)h + \mathbf{x}^{(3)}(t)\tfrac{h^2}{2} + \cdots\right) \\ &= -\tfrac{1}{12}h^2\mathbf{x}^{(3)}(t) + \cdots = \mathcal{O}(h^2).\end{aligned}$$

Therefore, the trapezoidal method has order $p = 2$, and it is consistent.

Similarly, one can show that Euler's method and the backward Euler method both have order $p = 1$. Furthermore, the family of BDF s-step methods have order $p = s$.

4.8. Convergence of Linear Multistep Methods

For a multistep method to be useful in practice, it should converge to the exact solution function $\mathbf{x}(t)$ of the IVP as we choose smaller and smaller time steps $h > 0$. More precisely, let $t > t_0$ be arbitrary but fixed, and consider time steps

$$h = h_K = \frac{t - t_0}{K}, \quad K = 1, 2, \dots .$$

Then, $t_K = t_0 + h_K K = t$, and

$$\mathbf{x}_K \;\approx\; \mathbf{x}(t), \quad K = 1, 2, \dots,$$

are approximations to the exact solution $\mathbf{x}(t)$ at time t. A multistep method is said to be *convergent* if

$$\lim_{K \to \infty} \mathbf{x}_K = \mathbf{x}(t)$$

for all $t > t_0$.

Clearly, a necessary condition for convergence is that the multistep method is consistent. However, consistence alone is not sufficient to guarantee convergence, and a second condition is needed. Next we discuss this second condition.

In general, associated with the s-step method

$$\mathbf{x}_{k+s} = \sum_{i=0}^{s-1} \mathbf{x}_{k+i}\alpha_i + h \sum_{i=0}^{s} \mathbf{f}_{k+i}\beta_i, \quad k = 0, 1, \dots,$$

is the polynomial

$$\psi(\mu) = \mu^s - \sum_{i=0}^{s-1} \mu^i \alpha_i.$$

Note that ψ has degree s and thus it has s zeros (if we count the zeros according to their multiplicities). The s-step method is said to be *stable* if for every zero λ of ψ we have $|\lambda| \leq 1$ and if $|\lambda| = 1$, then λ is a simple zero of ψ. Here, λ being a *simple zero* means that

$$\psi(\lambda) = 0 \quad \text{and} \quad \psi'(\lambda) \neq 0.$$

Consistence together with stability now guarantees that a linear multistep method converges. More precisely, we have the following result about convergence of linear multistep methods.

Theorem. A linear multistep method is convergent if, and only if, the method is consistent and stable. If it is convergent, then, for all fixed $t > t_0$,

$$\mathbf{x}_K = \mathbf{x}(t) + \mathcal{O}(h_K^p) \quad \text{for all} \quad h_K = \frac{t - t_0}{K}, \quad K = 0, 1, \ldots,$$

where p is the order of the linear multistep method.

Note that the last statement of this theorem means that the *convergence order* is identical to the order p defined by the LTE.

Examples: For the trapezoidal method, we have $\psi(\mu) = \mu - 1$. The polynomial ψ has only one zero $\lambda = 1$, and since $\psi'(1) = 1$, $\lambda = 1$ is a simple zero. Therefore, the trapezoidal method is stable. Since it is also consistent, the trapezoidal method is convergent by the above theorem.

For the BDF method with $s = 2$, we have

$$\psi(\mu) = \mu^2 - \frac{4}{3}\mu + \frac{1}{3} = \left(\mu - \frac{1}{3}\right)(\mu - 1).$$

Since both zeros $\lambda = 1/3$ and $\lambda = 1$ satisfy $|\lambda| \leq 1$ and $\psi'(1) = 2/3 \neq 0$, the method is stable, and thus convergent. However, not all BDF methods are stable. In fact, only the BDF methods for $s = 1, 2, \ldots, 6$ are stable. Since they are also consistent, the BDF methods for $s = 1, 2, \ldots, 6$ are convergent.

We now briefly return to DAEs in their most general form,

$$\begin{aligned} \mathbf{F}(\mathbf{x}, \dot{\mathbf{x}}, t) &= 0, \quad t \geq t_0, \\ \mathbf{x}(t_0) &= \mathbf{x}_0. \end{aligned}$$

The importance of the BDF methods (with $1 \leq s \leq 6$ of course) is that they can immediately be applied to such general DAEs. To this end, we simply need to replace the above DAE by the equations

$$\mathbf{F}(\mathbf{x}_{k+s}, \frac{1}{h\beta_s}\left(\mathbf{x}_{k+s} - \sum_{i=0}^{s-1} \mathbf{x}_{k+i}\alpha_i\right), t_{k+s}) = 0, \quad k = 0, 1, \ldots .$$

Each of this equations is then solved for x_{k+s}, using Newton's method.

For the special case of linear (in $\mathbf{x}$ and $\dot{\mathbf{x}}$) DAEs of the type

$$\mathbf{F}(\mathbf{x}, \dot{\mathbf{x}}, t) = \mathbf{A}(t)\dot{\mathbf{x}} - \mathbf{B}(t)\mathbf{x} - \mathbf{b}(t) = 0,$$

where $\mathbf{A}(t)$ and $\mathbf{B}(t)$ are time-dependent $n \times n$ matrices, and $\mathbf{b}(t)$ is a time-dependent vector of length n, one can show that, under mild assumptions, the BDF methods for $s = 1, 2, \ldots, 6$

converge with convergence order $p = s$. However, this result assumes that a constant time step $h > 0$ is used. In practice, one often uses variable time steps in order to speed up the overall computation. If variable time steps are used, then the convergence order is reduced if the DAE has index ≥ 3. More precisely, for the BDF s-step methods with index$-1 \leq s \leq 6$, one has

$$\mathbf{x}_K = \mathbf{x}(t) + \mathcal{O}(h_{\max}^{p'}),$$

where

$$p' = \min\{s, s - \text{index} + 2\}$$

and $h_{\max}$ denotes the maximal time step. In particular, we have a convergence-order reduction, i.e., $p' < p$, if index ≥ 3.

4.9. Stiff Differential Equations

We now return to explicit IVPs, and we discuss another important concept, namely stiffness of differential equations.

Often, the solution $\mathbf{x}(t)$ of an explicit IVP converges to a constant vector, the so-called steady state, $\mathbf{x}_\infty$, as $t \to \infty$, i.e.,

$$\lim_{t\to\infty} \mathbf{x}(t) = \mathbf{x}_\infty.$$

Roughly speaking, the IVP is called *stiff* if the function $\mathbf{x}(t) - \mathbf{x}_\infty$ is a combination of functions that decay at *very* different rates. This represents a problem in the following sense. When an explicit multistep method is used for the solution on a stiff IVP, then the *fastest* decaying function in such a combination of $\mathbf{x}(t) - \mathbf{x}_\infty$ is the one that dictates the choice of the time step $h > 0$, and it forces one to use a tiny time step. On the other hand, except for times t very close to the initial time t_0, the fastest decaying function has essentially no affect on the function $\mathbf{x}(t)$ for $t \gg t_0$. Still, its presence forces one to use a tiny time step for all $t > t_0$ when an explicit multistep method is employed. We now illustrate this phenomenon by means of an example.

Example: Let λ_1, $\lambda_2 < 0$ be negative constants, and consider the differential equation (with $n = 2$)

$$\dot{x}^{(1)} = \frac{\lambda_1 + \lambda_2}{2} x^{(1)} + \frac{\lambda_1 - \lambda_2}{2} x^{(2)},$$
$$\dot{x}^{(2)} = \frac{\lambda_1 - \lambda_2}{2} x^{(1)} + \frac{\lambda_1 + \lambda_2}{2} x^{(2)}.$$

Here, $x^{(1)}(t)$ and $x^{(2)}(t)$ denote the two component functions of the exact solution

$$\mathbf{x}(t) = \begin{bmatrix} x^{(1)}(t) \\ x^{(2)}(t) \end{bmatrix}.$$

It is easily verified that the exact solution of the above differential equation is given by

$$\begin{aligned}\dot{x}^{(1)} &= \gamma_1 \exp(\lambda_1 t) + \gamma_2 \exp(\lambda_2 t),\\ \dot{x}^{(2)} &= \gamma_1 \exp(\lambda_1 t) - \gamma_2 \exp(\lambda_2 t),\end{aligned}$$

where γ_1, γ_2 are constants that depend on the initial condition. Since $\lambda_1,\ \lambda_2 < 0$, we have

$$\mathbf{x}_\infty = \lim_{t\to\infty} \begin{bmatrix} x^{(1)}(t) \\ x^{(2)}(t) \end{bmatrix} = 0.$$

Solving the above differential equation numerically by means of the forward Euler method gives the approximate solutions

$$\begin{aligned}x_k^{(1)} &= \gamma_1(1 + h\lambda_1)^k + \gamma_2(1 + h\lambda_2)^k,\\ x_k^{(2)} &= \gamma_1(1 + h\lambda_1)^k - \gamma_2(1 + h\lambda_2)^k.\end{aligned}$$

The least one can expect from a reasonable solution technique is that, as $k \to \infty$, the approximate solution should also converge to the steady state $\mathbf{x}_\infty = 0$. However, for the forward Euler method, we have

$$\lim_{k\to\infty} \begin{bmatrix} x_k^{(1)} \\ x_k^{(2)} \end{bmatrix} = 0 \quad\Longleftrightarrow\quad |1 + h\lambda_1| < 1 \quad \text{and} \quad |1 + h\lambda_2| < 1.$$

The last two conditions are equivalent to

$$h < \frac{2}{|\lambda_1|} \quad \text{and} \quad h < \frac{2}{|\lambda_2|}.$$

Now assume that $\lambda_1 = -1$ and $\lambda_2 = -1000$. Then the second condition on h is the more restrictive one, and it forces one to use a time step h with $h < 0.002$. On the other hand, the corresponding terms $\exp(-1000t)$ in the exact solution very quickly go to zero as t increases, and therefore, they practically do not contribute anything. Nevertheless, the constant $\lambda_2 = -1000$ determines the tiny time step.

We now discuss the phenomenon of stiffness in more detail. For simplicity, we will restrict ourselves to linear IVPs of the form

$$\begin{aligned}\dot{\mathbf{x}} &= \mathbf{A}\mathbf{x},\\ \mathbf{x}(t_0) &= \mathbf{x}_0,\end{aligned}$$

where $\mathbf{A}$ is a given real $n \times n$ matrix. As we will see soon, each $n \times n$ matrix has n eigenvalues (some of which can be complex), $\lambda_1, \lambda_2, \ldots, \lambda_n$. Usually, all eigenvalues of the matrix $\mathbf{A}$ have negative real parts, i.e.,

$$\operatorname{Re} \lambda_j < 0, \quad j = 1, 2, \ldots, n,$$

and we then can assume that the eigenvalues are ordered such that

$$0 > \operatorname{Re} \lambda_1 \geq \operatorname{Re} \lambda_2 \geq \cdots \geq \operatorname{Re} \lambda_n.$$

The IVP is said to be *stiff* if

$$\operatorname{Re} \lambda_1 \gg \operatorname{Re} \lambda_n.$$

One-step methods (not necessarily linear ones) applied to linear IVPs of the above form produce approximate solutions $\mathbf{x}_k$ that usually satisfy recurrences of the form

$$\mathbf{x}_{k+1} = g(h\mathbf{A})\mathbf{x}_k, \quad k = 0, 1, \ldots,$$

where $h > 0$ again denotes the time step. Moreover, for an explicit one-step method, the function g is typically a polynomial, and for an implicit one-step method, g is a rational function. For example, Euler's method is given by

$$\mathbf{x}_{k+1} = \mathbf{x}_k + h\mathbf{A}\mathbf{x}_k = (I + h\mathbf{A})\,\mathbf{x}_k, \quad k = 0, 1, \ldots,$$

and thus the underlying function is the polynomial $g(z) = 1 + z$. Similarly, for the backward Euler method, one has $g(z) = 1/(1 - z)$.

Next consider the trapezoidal method. It is based on the relation

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \frac{h}{2}\left(\mathbf{A}\mathbf{x}_k + \mathbf{A}_k\mathbf{x}_{k+1}\right)$$

that can be rewritten as follows:

$$\left(I - \frac{h}{2}\mathbf{A}\right)\mathbf{x}_{k+1} = \left(I + \frac{h}{2}\mathbf{A}\right)\mathbf{x}_k.$$

It follows that

$$\mathbf{x}_{k+1} = g(h\mathbf{A})\mathbf{x}_k, \quad \text{where} \quad g(z) = \frac{2+z}{2-z}.$$

The importance of the function g underlying a one-step method for the solution of the above IVP is as follows. Assume that all eigenvalues of the matrix $\mathbf{A}$ have negative real parts. Then the steady state of the IVP is the zero vector $\mathbf{x}_\infty = 0$, and the approximations should reflect this behavior, i.e., one should have

$$\lim_{k \to \infty} \mathbf{x}_k = 0.$$

On the other hand, this condition should be satisfied for any time step $h > 0$, independent of the stiffness of the IVP. Now usually the eigenvalues of $\mathbf{A}$ are not known, but it is known that their real parts are all negative. Based on this knowledge, the condition $\lim_{k\to\infty} \mathbf{x}_k = 0$ is guaranteed if the function g satisfies

$$|g(z)| < 1 \quad \text{for all} \quad z \in \mathbb{C} \quad \text{with} \quad \operatorname{Re} z < 0.$$

A method whose associated function g fulfills this condition is called *absolutely stable* (A-stable). The condition can be equivalently stated as

$$\mathbb{C}_- := \{ z \in \mathbb{C} \mid \operatorname{Re} z < 0 \} \subseteq \mathcal{M},$$

where $\mathbb{C}_-$ is the left half of the complex plane and

$$\mathcal{M} := \{ z \in \mathbb{C} \mid |g(z)| < 1 \}$$

is called the *region of absolute stability* of the method given by g.

For example, for Euler's method, $g(z) = 1 + z$, and thus its region of absolute stability is

$$\mathcal{M} = \{ z \in \mathbb{C} \mid |1 + z| < 1 \},$$

which is the interior of a circle in the complex plane with center at $z = -1$ and radius 1. Consequently, Euler's method is not A-stable. For the backward Euler method,

$$\mathcal{M} = \{ z \in \mathbb{C} \mid 1 < |z - 1| \},$$

is the exterior of a circle in the complex plane with center at $z = 1$ and radius 1. Thus the backward Euler method is A-stable. For the trapezoidal method, one has

$$\mathcal{M} = \mathbb{C}_-,$$

and hence it is A-stable. Moreover, the trapezoidal method has the smallest-possible region of absolute stability, namely $\mathcal{M}$ is just the left half plane $\mathbb{C}_-$.

The importance of A-stable methods is that they are the only methods that can be used safely for the solution of stiff differential equations. Unfortunately, the class of A-stable methods is very limited, as the following theorem shows.

Theorem.

1. Only an implicit method can be A-stable.

2. There are no A-stable linear multistep methods of order $p > 2$.

3. The linear multistep method of order $p = 2$ with the smallest constant c_2 in the expansion
$$\text{LTE} = c_2 h^2 \mathbf{x}^{(3)}(t) + \cdots = \mathcal{O}(h^2)$$
of the local truncation error is the trapezoidal method for which
$$c_2 = -\frac{1}{12}.$$

In view of these negative results about the nonexistence of A-stable linear multistep methods of order $p > 2$, it is natural to relax the condition of A-stability in order to get a larger class of methods. One simple relaxation is the concept of A(α)-stability. For angles $0 < \alpha \leq 90°$, we denote by

$$\mathbb{C}_-(\alpha) := \left\{ z \in \mathbb{C} \,\middle|\, \operatorname{Re} z < 0 \text{ and } \left| \frac{\operatorname{Im} z}{\operatorname{Re} z} \right| < \tan \alpha \right\}$$

those points z in the left half plane $\mathbb{C}_-$ for which the angle between the line through 0 and z and the negative real axis is strictly less than α. Clearly,

$$\mathbb{C}_-(90°) = \mathbb{C}_-$$

is just the left half plane. A method is said to be A(α)-stable if $\mathbb{C}_-(\alpha)$ is contained in its region $\mathcal{M}$ of absolute stability, i.e.,

$$\mathbb{C}_-(\alpha) \subseteq \mathcal{M}.$$

For example, the BDF s-step methods are A(α)-stable for all $s = 1, 2, \ldots, 6$, and in the table below we list the corresponding angles $\alpha = \alpha(s)$. Recall that the BDF methods with $s = 1, 2, \ldots, 6$ are convergent.

s	$\alpha(s)$
1	90°
2	90°
3	88°
4	73°
5	51°
6	18°

Angles of A(α)-stability of BDFs for $s = 1, 2, \ldots, 6$

5. Complex Phasor Analysis

We have shown that linear and linearized circuits (i.e., nonlinear circuits for which the small-signal assumption applies) are modeled by the following system of linear differential equations:

$$\mathbf{C}\dot{\mathbf{x}} + \mathbf{G}\mathbf{x} - \mathbf{e}(t) = 0.$$

Here, $\mathbf{G}$ and $\mathbf{C}$ are constant $n \times n$ matrices.

We are often interested in the *sinusoidal, steady-state behavior* of circuits. In other words, the circuits are excited by one or more sinusoidal sources, and we are seeking the solution that is reached after the effects of the initial conditions have died out.

We further assume that all the sinusoidal excitation sources operate at the same frequency. This is not really a limitation. Linear circuits obey the law of superposition. Hence, when we have sources operating at different frequencies, we can always analyze their effect on the circuit one by one. Then, we simply add the results together.

Based on these assumptions, the excitation vector is of the form

$$\mathbf{e}(t) = \begin{bmatrix} a_1 \cos(\omega t + \phi_1) \\ a_2 \cos(\omega t + \phi_2) \\ \cdots \\ a_n \cos(\omega t + \phi_n) \end{bmatrix}.$$

Of course, for most practical circuits, the majority of the vector entries will be zero.

Consider now one individual entry in the excitation vector. It can be written as the real part of a complex number:

$$a \cos(\omega t + \phi) = \Re\left[a e^{i\omega t + \phi}\right] = \Re\left[\left(a e^{i\phi}\right) e^{\imath\omega t}\right] = \Re\left[A e^{\imath\omega t}\right].$$

The complex number $A = e^{i\phi}$ is called the *phasor*. When the frequency, $2\pi\omega$, is given, there is a one-to-one correspondence between any sinusoidal signal and a phasor. The correspondence is given by

$$A \leftrightarrow \Re\left[A e^{i\omega t}\right].$$

We can now write the excitation vector in terms of the phasors:

$$\mathbf{e}(t) = \Re\left[\begin{bmatrix} A_1 \\ A_2 \\ \cdots \\ A_n \end{bmatrix} e^{i\omega t}\right] = \Re\left[\mathbf{A} e^{i\omega t}\right].$$

Here $\mathbf{A}$ is a complex vector of length n.

The system of differential equations describing the circuit in this case can be written as follows:

$$\mathbf{C}\dot{\mathbf{x}} + \mathbf{G}\mathbf{x} - \Re\left[A e^{\imath\omega t}\right] = 0.$$

Consider now a companion system of differential equations that differs from the previous one only by its excitation:

$$\mathbf{C}\dot{\mathbf{y}} + \mathbf{G}\mathbf{y} - \Im\left[Ae^{\imath\omega t}\right] = 0.$$

From the two systems that involve only real variables, we now generate one new complex system that has our original one as the real part and the companion system as the imaginary part:

$$\mathbf{C}\dot{\mathbf{x}} + \mathbf{G}\mathbf{x} - \Im\left[Ae^{\imath\omega t}\right] + i\left[\mathbf{C}\dot{\mathbf{y}} + \mathbf{G}\mathbf{y} - \Im\left[Ae^{\imath\omega t}\right]\right] = 0.$$

With further manipulation, we obtain

$$\mathbf{C}\left(\dot{\mathbf{x}} + i\dot{\mathbf{y}}\right) + \mathbf{G}\left(\mathbf{x} + i\mathbf{y}\right) - Ae^{\imath\omega t} = 0$$

or

$$\mathbf{C}\dot{\mathbf{z}} + \mathbf{G}\mathbf{z} - Ae^{\imath\omega t} = 0.$$

This system has real coefficient matrices, $\mathbf{C}$ and $\mathbf{G}$, but a complex excitation vector. Therefore, in general, its solution $\mathbf{z}(t)$ will also be complex. Note, however, that if we manage to find a solution to this complex system, recovering the solution of the circuit is trivial. One simply takes the real part of the solution, $\mathbf{x}(t) = \Re[\mathbf{z}(t)]$. And indeed, finding the solution of the complex system turns out to be much simpler than the solution of our original system of equations. The excitation being a complex exponential, we seek solutions of the same form

$$\mathbf{z}(t) = \mathbf{Z}e^{i\omega t}.$$

Replacing $\mathbf{z}(t)$ in the complex system, we obtain

$$\mathbf{C}\frac{d}{dt}\left(\mathbf{Z}e^{i\omega t}\right) + \mathbf{G}\mathbf{Z}e^{i\omega t} - \mathbf{A}e^{\imath\omega t} = 0,$$

and by propagating the derivative, the equations become

$$i\omega\mathbf{C}\mathbf{Z}e^{i\omega t} + \mathbf{G}\mathbf{Z}e^{i\omega t} - \mathbf{A}e^{\imath\omega t} = 0.$$

We eliminate the factor $e^{i\omega t}$ and obtain the purely algebraic linear system of equations

$$i\omega\mathbf{C}\mathbf{Z} + \mathbf{G}\mathbf{Z} \;\; - \mathbf{A} = 0.$$

Its solution can be obtained by solving the complex system of linear equations

$$\left(\mathbf{G} + i\omega\mathbf{C}\right)\mathbf{Z} = \mathbf{A}.$$

The system matrix is now complex, $\mathbf{G}+i\omega\mathbf{C}$, however this fact causes no additional difficulty. Standard solution techniques for linear systems with real coefficients easily extend to the complex case. In particular, methods such as Gaussian elimination or LU-factorization apply identically to complex matrices. From the complex solution

$$\mathbf{Z} = \left(\mathbf{G} + i\omega\mathbf{C}\right)^{-1}\mathbf{A},$$

the circuit solution is immediately available:

$$\mathbf{x}(t) = \Re\left[\mathbf{Z}e^{\imath\omega t}\right].$$

It is often the case that designers want to study the response of their circuit over a range of frequencies. In this case, we need to solve the system for various values of ω. The coefficient matrix changes for each frequency, and therefore, it must be re-factored. However, the sparsity structure of the matrix remains the same, and hence the symbolic factorization phase need not be redone for each frequency.

6. Analysis of Linear and Linearized Circuits Using the Laplace Transform

So far, we studied the sinusoidal steady state of linear or linearized circuits excited by sinusoidal excitations. This analysis is performed using *phasors*. We would like to extend the analysis to circuits with arbitrary excitations. One option would be to use the ODE integration methods discussed in previous sections. For the case of linear time-invariant circuits, however, more accurate and efficient methods are available. Moreover, such methods provide more information about the analyzed circuit such as stability properties, analytic transfer functions, etc.

6.1. The Laplace Transform

A widely-used tool in the analysis of linear circuits is the Laplace transform. Recall that, in the case of studying the sinusoidal steady-state, we uniquely represented each sinusoidal signal by a complex number, the phasor:

$$A\cos(\omega t + \phi) \Longleftrightarrow Ae^{i\phi}.$$

Instead, we now attach to a signal, represented by a real function defined on the positive time axis, a complex function:

$$f : [0, \infty) \to \mathbb{R} \Longleftrightarrow F : \mathbb{C} \to \mathbb{C}.$$

The complex function represents the Laplace transform of f and is defined as follows:

$$F(s) = \mathcal{L}\{f\}(s) = \int_{0-}^{\infty} f(t)e^{-st}dt.$$

Here, $s = \sigma + i\omega \in \mathbb{C}$ is the complex frequency. Observe that, for sufficiently large values of σ, the integral is convergent even for functions $f(t)$ that grow exponentially. This property is important for the analysis of circuits that may be unstable and result in such exponentially growing waveforms.

For example, the function $f(t) = e^{at}$ has the Laplace transform

$$\mathcal{L}\{e^{at}\} = \int_{0-}^{\infty} e^{at}e^{-st}dt = \int_{0-}^{\infty} e^{(a-s)t}dt = \frac{1}{a-s}e^{(a-s)t}\Big|_{0-}^{\infty}.$$

When $a - \Re(s) < 0$, the value of the expression at infinity is zero and the integral is convergent, with the value

$$\frac{1}{s-a}.$$

Despite the fact that the integral is convergent only in a region of the complex plane, the above expression is well defined for any complex value of s except for $s = a$. A rigorous mathematical process called *analytic continuation* allows us to define the Laplace transform of e^{at} as follows:

$$\mathcal{L}\{e^{at}\} = \frac{1}{s-a}.$$

Note that this function is defined on $\mathbb{C} - \{a\}$, and not just on the region of convergence of the Laplace integral.

Important properties of the Laplace transform

- Uniqueness. The correspondence between the function and its Laplace transform is one-to-one, and knowing one, we are able to determine the other, by means of analytical techniques, tables, or numerical methods.
- Linearity:
$$\mathcal{L}\{\alpha f + \beta g\} = \alpha \mathcal{L}\{f\} + \beta \mathcal{L}\{g\}.$$
- Differentiation rule:
$$\mathcal{L}\{\frac{df}{dt}\} = s\mathcal{L}\{f\} - f(0-).$$
- Integration rule:
$$\mathcal{L}\{\int_0^t f(\tau)d\tau\} = \frac{1}{s}\mathcal{L}\{f\}.$$
- Convolution rule. We set $F(s) = \mathcal{L}\{f(t)\}$ and $G(s) = \mathcal{L}\{g(t)\}$. Then:
$$\mathcal{L}\{\int_0^t f(t-\tau)g(\tau)d\tau\} = F(s)G(s).$$

These properties are easily verified by using the definition of the Laplace transform and integration by parts.

The Laplace transform of widely-used functions

- The Dirac impulse $\delta(t)$. We first approximate the ideal impulse by a finite pulse of area 1 as follows:
$$p_\Delta(t) = \begin{cases} \frac{1}{\Delta} & \text{if} \quad 0 \le t < \Delta, \\ 0 & \text{otherwise.} \end{cases}$$
We compute the Laplace transform of this finite pulse and obtain
$$\mathcal{L}\{p_\Delta\} = \int_{0-}^{\infty} p_\Delta(t)e^{-st}dt = \int_{0-}^{\Delta} \frac{1}{\Delta}e^{-st}dt = -\frac{1}{s\Delta}\, e^{-st}\Big|_{0-}^{\Delta} = \frac{1-e^{-s\Delta}}{s\Delta}.$$
In the limit, as $\Delta \to 0$, we get
$$\mathcal{L}\{\delta(t)\} = 1.$$

- The step function defined by

$$u(t) = \begin{cases} 1 & \text{if} \quad t \geq 0, \\ 0 & \text{otherwise}, \end{cases}$$

  has the Laplace transform

$$\mathcal{L}\{u(t)\} = \frac{1}{s}.$$

  This expression suggests that the differentiation rule can be applied even when the function is not differentiable (differentiation is done in the sense of distributions).

- We have already shown that the Laplace transform of the exponential function is given by

$$\mathcal{L}\{e^{at}\} = \frac{1}{s-a}.$$

- The trigonometric functions sin and cos have the Laplace transforms

$$\mathcal{L}\{\sin \omega t\} = \frac{\omega^2}{s^2 + \omega^2}$$

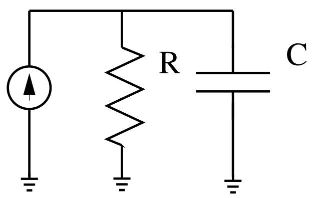
  and

$$\mathcal{L}\{\cos \omega t\} = \frac{s}{s^2 + \omega^2}.$$

Other Laplace transforms can be found in tables or using symbolic algebra packages.

Example

Consider the following simple RC circuit:



The equation describing this circuit is

$$i(t) - \frac{v(t)}{R} - C\frac{dv(t)}{dt} = 0.$$

We apply the Laplace transform to the entire equation, and by the linearity property and the differentiation rule, we obtain

$$I(s) - \frac{V(s)}{R} - C\left[sV(s) - v(0)\right] = 0.$$

The differential equation in time domain becomes an algebraic equation in the complex frequency variable, and we can easily solve it. Assume particular values for circuit elements say, $i(t) = sin(t)$, $R = 1$, $C = 1$. Then the equation becomes

$$\frac{1}{s^2+1} - V(s) - sV(s) + v(0) = 0.$$

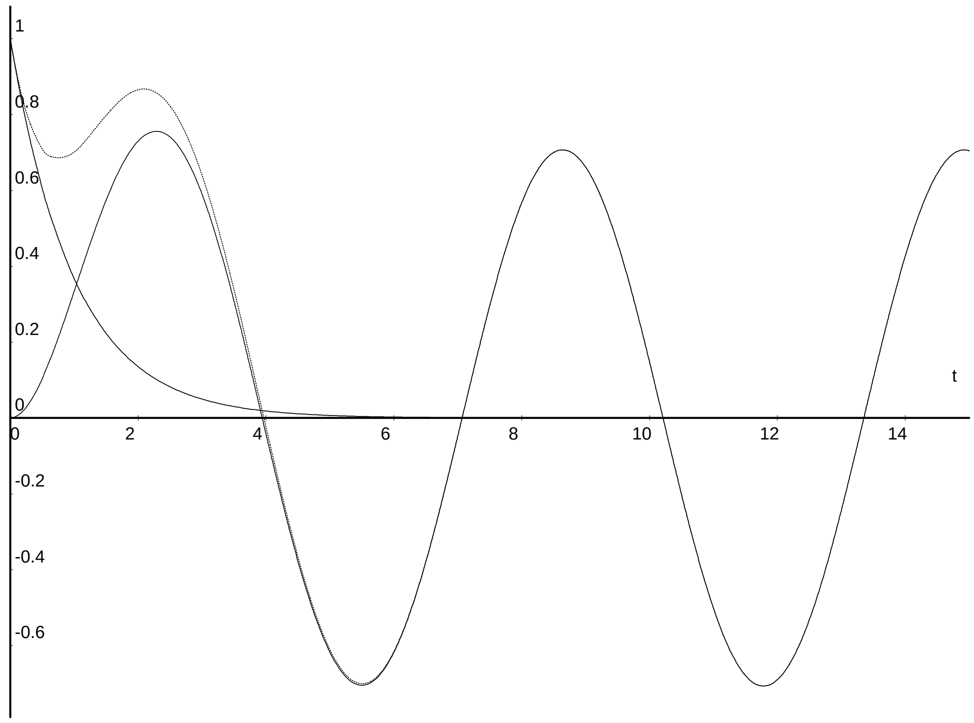
Solving for $V(s)$ gives

$$V(s) = \frac{\frac{1}{s^2+1} + v(0)}{1+s} = \frac{1}{(s^2+1)(1+s)} + \frac{v(0)}{1+s}.$$

The inverse Laplace transform of $V(s)$ represents the time-domain solution of the circuit:

$$v(t) = \frac{1}{2}e^{-t} - \frac{1}{2}\cos(t) + \frac{1}{2}sin(t) + v(0)e^{-t}.$$

We distinguish two components of the circuit solution $v(t)$. One is due solely to the initial condition and does nut depend on the excitation. The second part depends exclusively on the excitation and does not depend on the initial conditions. We call these components the *zero-input response* and the *zero-state response*, respectively. The figure below shows the circuit response and its two components. The zero-input response is just a decaying exponential. The zero-state response is a function that is zero at the origin and becomes a sinusoid in the limit.



6.2. Implementation in a Simulation Program

The following part of this section will describe methods to implement this type of analysis in a simulation program.

We start with the small-signal circuit equation of a linear circuit or of a circuit that was linearized about an operating point:

$$\mathbf{C}\dot{\mathbf{x}}(t) + \mathbf{G}\mathbf{x}(t) = \mathbf{b}(t).$$

Applying the Laplace transform to this differential equation, we obtain an algebraic equation

$$\mathbf{C}\left[s\mathbf{X}(s) - \mathbf{x}(0-)\right] + \mathbf{G}\mathbf{X}(s) = \mathbf{B}(s).$$

The solution of this equation is

$$\mathbf{X}(s) = \left(s\mathbf{C} + \mathbf{G}\right)^{-1}\mathbf{B}(s) + \left(s\mathbf{C} + \mathbf{G}\right)^{-1}\mathbf{C}\mathbf{x}(0-).$$

Just as in the above example, the solution has two components: the zero-input response and the zero-state response. The main difficulty in implementing this analysis in a simulator consists in the fact that the matrix $s\mathbf{C}+\mathbf{G}$ is symbolic in s, and therefore cannot be inverted numerically. Therefore, the expression needs to be analyzed further. To this send, we write

$$s\mathbf{C} + \mathbf{G} = \mathbf{G}(\mathbf{I} + s\mathbf{G}^{-1}\mathbf{C}).$$

Here, obviously, we assumed that the matrix $\mathbf{G}$ is nonsingular. While this assumption holds for most practical circuits, a procedure called *frequency shifting* extends this analysis even for the cases where $\mathbf{G}$ is singular. We use the notation $\mathbf{A} = -\mathbf{G}^{-1}\mathbf{C}$, and perform the eigendecomposition $\mathbf{A} = \mathbf{S}\mathbf{\Lambda}\mathbf{S}^{-1}$. Here, $\mathbf{\Lambda}$ is a diagonal matrix having the eigenvalues of $\mathbf{A}$ on the diagonal and $\mathbf{S}$ is the corresponding eigenvector matrix. Using this eigendecomposition, we get

$$\mathbf{G}(\mathbf{I} - s\mathbf{A}) = \mathbf{G}(\mathbf{I} - s\mathbf{S}\mathbf{\Lambda}\mathbf{S}^{-1}) = \mathbf{G}(\mathbf{S}\mathbf{S}^{-1} - s\mathbf{S}\mathbf{\Lambda}\mathbf{S}^{-1}) = \mathbf{G}\mathbf{S}(\mathbf{I} - s\mathbf{\Lambda})\mathbf{S}^{-1}.$$

We managed to factor the matrix $s\mathbf{C} + \mathbf{G}$ in a way that the part that depends on s is contained in the diagonal matrix

$$\mathbf{I} - s\mathbf{\Lambda} = \begin{bmatrix} 1 - s\lambda_1 & 0 & \cdots & 0 \\ 0 & 1 - s\lambda_2 & 0 & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & 0 & 1 - s\lambda_n \end{bmatrix}.$$

The inverse of the diagonal matrix is obvious and now we can also compute the inverse:

$$(s\mathbf{C} + \mathbf{G})^{-1} = \mathbf{S}\begin{bmatrix} \frac{1}{1-s\lambda_1} & 0 & \cdots & 0 \\ 0 & \frac{1}{1-s\lambda_2} & 0 & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & 0 & \frac{1}{1-s\lambda_n} \end{bmatrix}\mathbf{S}^{-1}\mathbf{G}^{-1}.$$

This inverse appears in both the zero-input response and the zero-state response of a network.

6.2.1. The Zero-Input Response

The zero-input response is the component of the linear circuit response that depends only on the initial state of the circuit, and it is given by

$$\mathbf{X}_{0i}(s) = (s\mathbf{C}+\mathbf{G})^{-1}\,\mathbf{C}\mathbf{x}(0-) = \mathbf{S}\begin{bmatrix} \frac{1}{1-s\lambda_1} & 0 & \cdots & 0 \\ 0 & \frac{1}{1-s\lambda_2} & 0 & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & 0 & \frac{1}{1-s\lambda_n} \end{bmatrix}\mathbf{S}^{-1}\mathbf{G}^{-1}\mathbf{C}\mathbf{x}(0-).$$

The product to the right of the symbolic matrix is in fact a column vector given by

$$\nu = \mathbf{S}^{-1}\mathbf{G}^{-1}\mathbf{C}\mathbf{x}(0-).$$

This vector can be computed by a succession of matrix-vector multiplications and linear system solves. The zero-input response becomes

$$\mathbf{X}_{0i}(s) = \mathbf{S}\begin{bmatrix} \frac{\nu_1}{1-s\lambda_1} \\ \frac{\nu_2}{1-s\lambda_2} \\ \cdots \\ \frac{\nu_n}{1-s\lambda_n} \end{bmatrix}.$$

Therefore, every component of the zero-input response is given by

$$\mathbf{X}_{0i}^{j}(s) = \sum_{k=1}^{n}\frac{s_{jk}\nu_k}{1-s\lambda_k}.$$

Using simple algebraic manipulation, this expression can be rewritten in the well-known pole-residue form

$$\mathbf{X}_{0i}^{j}(s) = r_\infty + \sum_{k=1}^{n'}\frac{r_k}{s-p_k}.$$

Here $p_k = 1/\lambda_k$ are the poles and r_k the corresponding residues. The term r_∞ results in the case that one or more of the eigenvalues λ_j is zero. Observe that the poles depend only on the network matrices **G** and **C** and are therefore a general property of the linear network. The residues, in contrast, are specific to the output of interest and to the initial state of the network.

The expression of the zero-input response can be easily converted back to the time domain using Laplace transform formulas.

$$\mathbf{x}_{0i}^{j}(t) = r_\infty\delta(t) + \sum_{k=1}^{n'} r_k e^{p_k t}$$

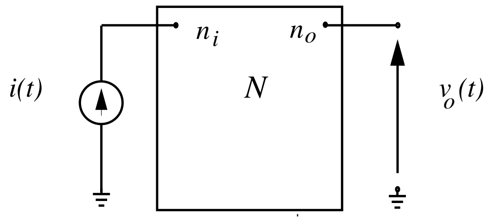
We distinguish several cases. If p_k is real than a positive value will lead to a growing exponential and a negative value to a decaying exponential. If p_k is complex it can be shown that, since we started from real matrices, $\bar{p_k}$, its complex conjugate will also be a pole of the system and moreover their corresponding residues will also be the complex conjugates of each other. Therefore we can treat them as a pair. Assume $r = a + ib$ and $p = \sigma + i\omega$. Then

$$re^{pt} + \bar{r}e^{\bar{p}t} = 2\Re[re^{pt}] = e^{\sigma t}[a\cos(\omega t) - b\sin(\omega t)]$$

Again, if σ, the real part of the pole is positive we have an oscillation with an exponentially increasing amplitude. Otherwise we will have a decaying oscillation. Recall that this is the response of the network with no external excitation. Outputs that are infinitely increasing are nonphysical and signify instability. In order to have a stable network, all the poles must belong to the left side of the complex plane.

6.2.2. The Zero-State Response

Consider a linear network excited by a source $i(t)$, as shown in the following figure:



Such a network is described by the equations

$$\mathbf{C}\dot{\mathbf{x}}(t) + \mathbf{G}\mathbf{x}(t) = \mathbf{b}i(t).$$

Here, $\mathbf{b}$ is a vector that has a one corresponding to the node n_i and has zero entries otherwise. The output v_o of the network is of the form

$$v_o(t) = \mathbf{l}^T\mathbf{x}(t),$$

where $\mathbf{l}$ is a vector that has a one corresponding to the node n_o and has zero entries otherwise. In Laplace domain, the equations become

$$\begin{aligned} \mathbf{C}[s\mathbf{X}(s) - \mathbf{x}(0-)] + \mathbf{G}\mathbf{X}(s) &= \mathbf{b}I(s) \\ V(s) &= \mathbf{l}^T\mathbf{X}(s). \end{aligned}$$

Solving these equations, we get

$$V(s) = \mathbf{l}^T(\mathbf{C}s + \mathbf{G})^{-1}\left[\mathbf{C}\mathbf{x}(0-) + \mathbf{b}I(s)\right].$$

We already analyzed the response due to the initial conditions, so we now turn to the response due to the input. This response is given by

$$V(s) = \mathbf{l}^T(\mathbf{C}s + \mathbf{G})^{-1}\mathbf{b}I(s).$$

We now set

$$H(s) = \mathbf{l}^T(\mathbf{C}s + \mathbf{G})^{-1}\mathbf{b}.$$

Note that $H(s)$ is a scalar function of s and that $H(s)$ is independent of the value of the excitation source. Moreover, if we know $H(s)$, we can predict the zero-state response to any excitation by simply multiplying this function with the Laplace transform of the excitation. The function $H(s)$ is called the *transfer function* of the network. In time domain, we can use the convolution property of the Laplace transform to obtain the time-domain response of the network. This yields

$$v_o(t) = \int_0^t h(\tau)i(t-\tau)d\tau,$$

where $h(t)$ is the inverse Laplace transform of $H(s)$, also called the *impulse response* of the network. It is clear that it is of great interest to be able to compute the transfer function. We have the same problem as before of finding the inverse of $\mathbf{C}$ s + $\mathbf{G}$. We use the expression derived above and obtain

$$H(s) = \mathbf{l}^T\mathbf{S}(\mathbf{I} - s\Lambda)^{-1}\mathbf{S}^{-1}\mathbf{G}^{-1}\mathbf{b}.$$

Setting $\mathbf{l}^T\mathbf{S} = \mu^T$ and $\mathbf{S}^{-1}\mathbf{G}^{-1}\mathbf{b} = \nu$, we obtain

$$H(s) = \sum_{k=0}^{n} \frac{\mu_k \nu_k}{1 - s\lambda_k}$$

or, equivalently, in the more familiar pole-residue notation

$$H(s) = r_\infty + \sum_{k=0}^{n'} \frac{r_k}{s - p_k}.$$

The expression of H(s) can therefore be obtained by performing an eigendecomposition of the matrix $\mathbf{A} = -\mathbf{G}^{-1}\mathbf{C}$ and a few simple linear algebra operations. The eigendecomposition is numerically difficult for systems of more than moderate size. For large systems, approximate methods must be used instead.

6.3. Numerical Eigendecomposition

6.3.1. The Eigenvalue Problem

With each $n \times n$ matrix $\mathbf{A}$, one can associate a set of n numbers, the so-called *eigenvalues* of $\mathbf{A}$. Eigenvalues play an important role in many applications. For example, as we have seen, the analysis of linear circuits in frequency domain involves *transfer functions* of the form

$$H(s) = \mathbf{l}^T(\mathbf{I} - s\mathbf{A})^{-1}\mathbf{r},$$

where $\mathbf{l}$ and $\mathbf{r}$ are vectors of length n, and $\mathbf{I}$ denotes the $n \times n$ identity matrix. The function H is rational, and its poles are given by

$$s_j = \frac{1}{\lambda_j}, \quad j = 1, 2, \ldots, n,$$

where the λ_j's are just the eigenvalues of $\mathbf{A}$.

We now list some properties of eigenvalues, and then we discuss algorithms for computing eigenvalues.

6.3.2. Properties of Eigenvalues

From now on, we always assume that $\mathbf{A}$ is an $n \times n$ matrix with real or complex entries. Furthermore, $\mathbf{I}$ always denotes the $n \times n$ identity matrix.

A number $\lambda \in \mathbb{C}$ is called an *eigenvalue* of $\mathbf{A}$ if the matrix

$$\mathbf{A} - \lambda \mathbf{I}$$

is singular.

Recall that an $n \times n$ matrix $\mathbf{M}$ is singular if, and only if, the linear system $\mathbf{M}\mathbf{x} = 0$ has a solution $\mathbf{x}$ with $\mathbf{x} \neq 0$. Therefore, the above definition of an eigenvalue is equivalent to the following statement: the number $\lambda \in \mathbb{C}$ is an eigenvalue of $\mathbf{A}$ if there exists a vector $\mathbf{x} \in \mathbb{C}^n$, $\mathbf{x} \neq 0$, such that

$$\mathbf{A}\mathbf{x} = \lambda \mathbf{x}.$$

The vector $\mathbf{x}$ is called an *eigenvector* of $\mathbf{A}$ associated with the eigenvalue λ. Similarly, $\lambda \in \mathbb{C}$ is an eigenvalue of $\mathbf{A}$ if there exists a *left eigenvector* $\mathbf{y} \in \mathbb{C}^n$, $\mathbf{y} \neq 0$, such that

$$\mathbf{y}^H \mathbf{A} = \lambda \mathbf{y}^H.$$

Here,

$$\mathbf{y}^H = [\,\overline{y_1} \quad \overline{y_2} \quad \cdots \quad \overline{y_n}\,]$$

denotes the conjugate transpose of the vector

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \in \mathbb{C}^n.$$

When there is need to distinguish $\mathbf{x}$ and $\mathbf{y}$, we refer to $\mathbf{x}$ as a *right* eigenvector. Whenever we use the term "eigenvector" alone, it always means a right eigenvector.

An eigenvalue λ is called *simple* if all eigenvectors $\mathbf{x}$ associated with λ are of the form

$$\mathbf{x} = \alpha \hat{\mathbf{x}}, \quad \alpha \in \mathbb{C},\ \alpha \neq 0,$$

where $\hat{\mathbf{x}} \in \mathbb{C}^n$, $\hat{\mathbf{x}} \neq 0$, is a fixed vector. In other words, an eigenvalue is simple if all eigenvectors associated with λ span a 1-dimensional subspace of $\mathbb{C}^n$. Properties of eigenvalues can often be formulated more easily when λ is assumed to be simple.

The eigenvalues of a matrix $\mathbf{A}$ are closely related to the *characteristic polynomial* p of $\mathbf{A}$ defined by

$$p(\lambda) = \det(\lambda I - \mathbf{A}).$$

Standard properties of the determinant imply that p is of the form

$$p(\lambda) = \lambda^n + \gamma_{n-1}\lambda^{n-1} + \cdots + \gamma_1 \lambda + \gamma_0,$$

where $\gamma_0, \gamma_1, \ldots, \gamma_{n-1} \in \mathbb{C}$ are suitable coefficients. Therefore, p is indeed a polynomial, and its degree is n. It turns out that the zeros of the characteristic polynomial p are just the eigenvalues of $\mathbf{A}$. Recall that a polynomial of degree n has exactly n zeros (some of which may be complex). Consequently, an $n \times n$ matrix $\mathbf{A}$ has n eigenvalues, say

$$\lambda_1, \lambda_2, \ldots, \lambda_n.$$

The set

$$\sigma(\mathbf{A}) = \{\, \lambda_1, \lambda_2, \ldots, \lambda_n \,\}$$

of all eigenvalues of $\mathbf{A}$ is called the *spectrum* of $\mathbf{A}$. We stress that, in general, not all of the n eigenvalues of $\mathbf{A}$ have to be different numbers. In fact, let L denote the number of distinct eigenvalues of $\mathbf{A}$. Then, we can always number the eigenvalues so that

$$\lambda_1, \lambda_2, \ldots, \lambda_L$$

are just the distinct eigenvalues of $\mathbf{A}$. Furthermore, the characteristic polynomial p has the following representation:

$$p(\lambda) = (\lambda - \lambda_1)^{m_1}(\lambda - \lambda_2)^{m_2} \cdots (\lambda - \lambda_L)^{m_L},$$

where $1 \leq m_1, m_2, \ldots, m_L$ are integers such that

$$m_1 + m_2 + \cdots + m_L = n.$$

For each $j = 1, 2, \ldots$, the integer m_j is called the *algebraic multiplicity* of the eigenvalue λ_j. Each eigenvalue λ_j, $j = 1, 2, \ldots, L$, also has a *geometric multiplicity*, g_j, which is defined as the dimension of the subspace of $\mathbb{C}^n$ spanned by all eigenvectors associated with λ_j. One can show that

$$1 \leq g_j \leq m_j \quad \text{for all} \quad j = 1, 2, \ldots, L.$$

Note that, in general, $g_j < m_j$ can occur, and hence the algebraic and the geometric multiplicities need not agree. Finally, note that $g_j = 1$ if, and only if, λ_j is a simple eigenvalue.

Example: The matrix

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

has the characteristic polynomial

$$p(\lambda) = \det \begin{bmatrix} \lambda & -1 & 0 \\ 0 & \lambda & -1 \\ 0 & 0 & \lambda \end{bmatrix} = \lambda^3.$$

Therefore, $\lambda_1 = 0$ is the only distinct eigenvalue, and its algebraic multiplicity is $m_1 = 3$. All the eigenvectors associated with λ_1 are given by

$$\mathbf{x} = \begin{bmatrix} \alpha \\ 0 \\ 0 \end{bmatrix}, \quad \alpha \in \mathbb{C}, \ \alpha \neq 0.$$

Therefore, λ_1 is a simple eigenvalue, and its geometric multiplicity is $g_1 = 1$.

It is tempting to use the above connection between the eigenvalues and the characteristic polynomial p to compute eigenvalues numerically by first generating the coefficients of p and then obtaining the eigenvalues as the zeros of p. However, such a procedure is numerically unstable, since the zeros of a polynomial are usually extremely sensitive to small changes of the coefficients of the polynomial. Therefore, the computation of eigenvalues via the characteristic polynomial is a bad idea. As we will see soon, there are much better algorithms that generate the eigenvalues directly from the matrix $\mathbf{A}$.

We remark that, for $n \geq 5$, it is not possible to compute the eigenvalues of a general $n \times n$ matrix $\mathbf{A}$ by means of a finite procedure, and hence, methods for computing eigenvalues have to be iterative procedures. If the eigenvalues of a matrix could be obtained by a finite procedure, then this would mean that the zeros of a polynomial could also be computed by means of a finite procedure. However, it is a mathematical fact that this is impossible for polynomials of degree $n \geq 5$.

6.4. Computation of Eigenvalues

There are two types of algorithms for eigenvalue computations:

- Algorithms for small and moderate-size matrices $\mathbf{A}$. The standard method of this type is the QR algorithm.

- Algorithms for large (and usually sparse) matrices $\mathbf{A}$.

Next, we present some basic material for eigenvalue computations. After that, we discuss the QR algorithm.

6.4.1. Similarity Transformations

Let $\mathbf{Q}$ be an arbitrary nonsingular $n \times n$ matrix. Then, the eigenvalue equation

$$\mathbf{A}\mathbf{x} = \lambda \mathbf{x}$$

is equivalent to the relation

$$\left(\mathbf{Q}\mathbf{A}\mathbf{Q}^{-1}\right)\mathbf{Q}\mathbf{x} = \lambda \mathbf{Q}\mathbf{x}.$$

Therefore, λ is an eigenvalue of $\mathbf{A}$ with eigenvector $\mathbf{x}$ if, and only if, λ is an eigenvalue of $\mathbf{Q}\mathbf{A}\mathbf{Q}^{-1}$ with eigenvector $\mathbf{Q}\mathbf{x}$.

The mapping

$$\mathbf{A} \longmapsto \mathbf{Q}\mathbf{A}\mathbf{Q}^{-1}$$

is called a *similarity transformation*. As we just saw, such a mapping leaves the eigenvalues unchanged and eigenvectors are multiplied by $\mathbf{Q}$.

The basic idea of the QR algorithm is to determine a "numerically well-behaved" matrix $\mathbf{Q}$ such that the eigenvalues of $\mathbf{Q}\mathbf{A}\mathbf{Q}^{-1}$ are obvious. This is certainly the case if

$$\mathbf{Q}\mathbf{A}\mathbf{Q}^{-1} = \mathbf{R} = \begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ 0 & r_{22} & \ddots & \vdots \\ \vdots & \ddots & \ddots & r_{n-1,n} \\ 0 & \cdots & 0 & r_{nn} \end{bmatrix}$$

is an upper triangular matrix, since the eigenvalues of $\mathbf{R}$ are simply its diagonal elements. The numerically best-behaved matrices that are available are unitary matrices. Hence the question is if there exists a unitary matrix $\mathbf{Q}$ such that $\mathbf{Q}\mathbf{A}\mathbf{Q}^H = \mathbf{R}$ is upper triangular. The answer is indeed "yes", as we see from the following theorem.

Theorem. Let $\mathbf{A}$ be an arbitrary $n \times n$ matrix. Then, there exist a unitary $n \times n$ matrix $\mathbf{Q}$ and an upper triangular $n \times n$ matrix $\mathbf{R}$ such that

$$\mathbf{Q}\mathbf{A}\mathbf{Q}^H = \mathbf{R} = \begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ 0 & r_{22} & \ddots & \vdots \\ \vdots & \ddots & \ddots & r_{n-1,n} \\ 0 & \cdots & 0 & r_{nn} \end{bmatrix}.$$

Moreover, the eigenvalues of $\mathbf{A}$ are just the diagonal elements of $\mathbf{R}$, i.e.,

$$\sigma(\mathbf{A}) = \{\, r_{11}, r_{22}, \ldots, r_{nn} \,\}.$$

6.4.2. The QR Algorithm

Given any $n \times n$ matrix $\mathbf{A}$, the QR algorithm is a computational procedure that generates a unitary $n \times n$ matrix $\mathbf{Q}$ and an upper triangular $n \times n$ matrix $\mathbf{R}$ such that $\mathbf{QAQ}^H = \mathbf{R}$. This is done in two steps.

In a first step, $\mathbf{A}$ is reduced to upper Hessenberg form. More precisely, a unitary $n \times n$ matrix $\mathbf{Q}_0$ is determined such that

$$\mathbf{Q}_0\mathbf{A}\mathbf{Q}_0^H = \mathbf{H} = \begin{bmatrix} \times & \times & \cdots & \cdots & \times \\ \times & \times & \ddots & & \vdots \\ 0 & \times & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \times \\ 0 & \cdots & 0 & \times & \times \end{bmatrix}.$$

A matrix $\mathbf{H}$ of this form is called an *upper Hessenberg matrix*. As usual, the symbol "$\times$" stands for a potentially nonzero entry of a matrix.

The reduction of $\mathbf{A}$ to upper Hessenberg form can be done by a sequence of elementary unitary matrices, namely either Givens rotation or Householder matrices. Givens rotations zero out one element at a time. Householder matrices create the necessary zeros in a whole column at once. Here, we will only discuss the use of Givens rotations.

An elementary *Givens rotation* is a 2×2 matrix of the form

$$\mathbf{G} = \begin{bmatrix} c & -e^{-i\phi}s \\ e^{i\phi}s & c \end{bmatrix}, \quad \text{where } c, s, \phi \in \mathbb{R} \text{ with } c^2 + s^2 = 1.$$

Moreover, $i = \sqrt{-1}$ is the imaginary unit. It is easy to verify that $\mathbf{G}$ is a unitary matrix. For any four numbers $a_{11}, a_{12}, a_{21}, a_{22} \in \mathbb{C}$, we can find numbers $c, s, \phi \in \mathbb{R}$ with $c^2 + s^2 = 1$ such that the associated Givens rotation satisfies

$$\mathbf{G}\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = \begin{bmatrix} \tilde{a}_{11} & \tilde{a}_{12} \\ 0 & \tilde{a}_{22} \end{bmatrix}.$$

Hence, Givens rotations can be used to zero out elements of a matrix. We remark that, if $a_{11}, a_{12}, a_{21}, a_{22} \in \mathbb{R}$, then one can always set $\phi = 0$, i.e., $\mathbf{G}$ is a real matrix.

We now illustrate the reduction to upper Hessenberg form by means of Givens rotations for the case of a 4×4 matrix,

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}.$$

The first Givens rotation, $\mathbf{G}_1$, is applied from the left to the second and third rows of $\mathbf{A}$ and chosen such that the element a_{31} is zeroed out. Simultaneously, $\mathbf{G}_1^H$ is applied from the right to the second and third columns of $\mathbf{A}$. The result is a matrix of the form

$$\operatorname{diag}(1, \mathbf{G}_1, 1)\, \mathbf{A}\, \operatorname{diag}(1, \mathbf{G}_1^H, 1) = \begin{bmatrix} \times & \star & \star & \times \\ \star & \star & \star & \star \\ 0 & \star & \star & \star \\ \times & \star & \star & \times \end{bmatrix}.$$

Here, "$\times$" denotes elements that were not changed by this transformation, and "$\star$" denotes elements that have changed. The second Givens rotation acts on rows 2 and 4, respectively columns 2 and 4, and it is chosen such that the element in the $(4, 1)$ position is zeroed out. Clearly, this does not destroy the zero in the $(3, 1)$ position that we just created. The result of this transformation is a matrix of the form

$$\begin{bmatrix} \times & \star & \times & \star \\ \star & \star & \star & \star \\ 0 & \star & \times & \star \\ 0 & \star & \star & \star \end{bmatrix}.$$

Finally, with a third Givens rotation, we zero out the element in the $(4, 2)$ position. The result is the final matrix

$$\mathbf{H} = \begin{bmatrix} \times & \times & \star & \star \\ \times & \times & \star & \star \\ 0 & \star & \star & \star \\ 0 & 0 & \star & \star \end{bmatrix}$$

in upper Hessenberg form.

For a general $n \times n$ matrix $\mathbf{A}$, one needs $n-2$ Givens rotations to zero out the elements in the $(3, 1), (4, 1), \ldots, (n, 1)$ positions, $n-3$ Givens rotations to zero out the elements in the $(4, 2), (5.2), \ldots, (n, 2)$ positions, and so fourth. The total number of Givens rotations is

$$(n-2) + (n-3) + \cdots + 2 + 1 = \frac{(n-2)(n-1)}{2}.$$

The second step of the QR algorithm is the QR iteration itself. Here, one generates a sequence of matrices $\mathbf{A}_k$, $k = 1, 2, \ldots$, as follows.

Algorithm (QR iteration)

1. `Set` $\mathbf{A}_1 = \mathbf{H}$`, where` $\mathbf{H}$ `is the upper Hessenberg matrix created in the first phase.`

2. `For` $k = 1, 2, \ldots$`, do:`

   - `Compute` $\mathbf{Q}_k$ `and` $\mathbf{R}_k$ `such that` $\mathbf{Q}_k$ `is unitary and`

     $$\mathbf{Q}_k \mathbf{A}_k = \mathbf{R}_k$$

     `is an upper triangular matrix.`

   - `Set`

     $$\mathbf{A}_{k+1} = \mathbf{R}_k \mathbf{Q}_k^H.$$

It is easy to verify that each of the matrices $\mathbf{A}_k$ generated by this algorithm is upper Hessenberg:

$$\mathbf{A}_k = \begin{bmatrix} \times & \times & \cdots & \cdots & \times \\ \times & \times & \ddots & & \vdots \\ 0 & \times & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \times \\ 0 & \cdots & 0 & \times & \times \end{bmatrix} \quad \text{for all} \quad k.$$

As a result, we can choose $\mathbf{Q}_k$ as a product of $n-1$ Givens rotations that are used to successively zero out the $n-1$ subdiagonal entries in positions $(2,1), (3,2), \ldots, (n, n-1)$.

Next, we note that, for each $k = 1, 2, \ldots$,

$$\begin{aligned}
\mathbf{A}_k &= \mathbf{R}_{k-1} \mathbf{Q}_{k-1}^H \\
&= \mathbf{Q}_{k-1} \mathbf{A}_{k-1} \mathbf{Q}_{k-1}^H \\
&= \mathbf{Q}_{k-1} \mathbf{Q}_{k-2} \mathbf{A}_{k-2} \mathbf{Q}_{k-2}^H \mathbf{Q}_{k-1}^H \\
&\ \vdots \\
&= \mathbf{Q}_{k-1} \mathbf{Q}_{k-2} \cdots \mathbf{Q}_1 \mathbf{A}_1 \mathbf{Q}_1^H \cdots \mathbf{Q}_{k-2}^H \mathbf{Q}_{k-1}^H \\
&= \mathbf{Q}_{k-1} \mathbf{Q}_{k-2} \cdots \mathbf{Q}_1 \mathbf{Q}_0 \mathbf{A} \mathbf{Q}_0^H \mathbf{Q}_1^H \cdots \mathbf{Q}_{k-2}^H \mathbf{Q}_{k-1}^H \\
&= \mathbf{Q}^{(k-1)} \mathbf{A} \left(\mathbf{Q}^{(k-1)} \right)^H,
\end{aligned}$$

where

$$\mathbf{Q}^{(k-1)} = \mathbf{Q}_{k-1} \mathbf{Q}_{k-2} \cdots \mathbf{Q}_1 \mathbf{Q}_0$$

is a unitary matrix. Hence, each $\mathbf{A}_k$ is similar to the original matrix $\mathbf{A}$, and thus $\mathbf{A}_k$ has the same eigenvalues as $\mathbf{A}$. On the other hand, the matrices $\mathbf{A}_k$ converge to an upper triangular matrix:

$$\lim_{k\to\infty} \mathbf{A}_k = \mathbf{R} = \begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ 0 & r_{22} & \ddots & \vdots \\ \vdots & \ddots & \ddots & r_{n-1,n} \\ 0 & \cdots & 0 & r_{nn} \end{bmatrix}.$$

Together, it follows that the diagonal elements $r_{11}, r_{22}, \ldots, r_{nn}$ are just the eigenvalues of $\mathbf{A}$.

Practical Aspects

For the practical use of the QR algorithm, the following two points are important.

1. To speed up the convergence, the QR iteration is combined with a shifting strategy. More precisely, at each step, a suitable shift $\sigma_k \in \mathbb{C}$ is chosen, and then one obtains $\mathbf{A}_{k+1}$ from $\mathbf{A}_k$ via the following relations:

$$\begin{aligned} \mathbf{Q}_k(\mathbf{A}_k - \sigma_k \mathbf{I}) &= \mathbf{R}_k, \\ \mathbf{A}_{k+1} &= \mathbf{R}_k \mathbf{Q}_k^H + \sigma_k \mathbf{I}. \end{aligned}$$

2. Since the upper Hessenberg matrices $\mathbf{A}_k$ converge to an upper triangular matrix, the subdiagonal elements of $\mathbf{A}_k$ converge to zero. As soon as one of the subdiagonal elements has converged to zero, then the diagonal element next to it is accepted as an eigenvalue of $\mathbf{A}$ and the corresponding row and column of $\mathbf{A}$ are deleted. This process is called *deflation*. Note that each deflation reduces the size of the matrix $\mathbf{A}_k$ by 1.

Complexity

If one is only interested in the eigenvalues of $\mathbf{A}$, then there is no need to actually compute and store the accumulated unitary matrix $\mathbf{Q}^{(k)}$. Moreover, the QR algorithm then requires approximately $16n^3$ additions and multiplications to obtain all eigenvalues of an $n \times n$ matrix $\mathbf{A}$.

If both eigenvalues and eigenvectors are to be computed, then the matrix $\mathbf{Q}^{(k)}$ has to be computed and stored. As a result, the work increases to approximately $30n^3$ additions and multiplications.

However, note that the QR algorithm itself does not generate the eigenvectors of $\mathbf{A}$. It only produces the unitary matrix

$$\mathbf{Q} = \lim_{k\to\infty} \mathbf{Q}^{(k)}$$

and an upper triangular matrix $\mathbf{R}$ such that

$$\mathbf{QAQ}^H = \mathbf{R}.$$

To obtain the eigenvector matrix $\mathbf{X}$ of $\mathbf{A}$ given by

$$\mathbf{AX} = \mathbf{X}\Lambda, \quad \Lambda = \mathrm{diag}(r_{11}, r_{22}, \ldots, r_{nn}),$$

one proceeds as follows. For each $j = 1, 2, \ldots, n$, we determine a vector of the form

$$\mathbf{z}_j = \begin{bmatrix} \times \\ \vdots \\ \times \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad \text{with 1 in the } k\text{th position,}$$

such that

$$(\mathbf{R} - r_{jj}\mathbf{I})\mathbf{z}_j = 0.$$

Then, we set

$$\mathbf{X} = \mathbf{Q}^H \left[\mathbf{z}_1 \quad \mathbf{z}_2 \quad \cdots \quad \mathbf{z}_n \right].$$