University of Ljubljana
Faculty of Electrical Engineering

Lectures Laboratory Sessions

Circuit Analysis and Optimization

International Course

Circuit topology, constitutive relations, and nodal analysis

1st Lecture W1, October

In this lecture we are going get acquainted with the concept of concentrated circuits, Kirchoff current and voltage law and constitutive relations of circuit elements. These equations form the mathematical model of a circcuit. The number of equations and unknowns can be greatly reduced if we introduce nodal voltages (which results in the nodal analysis approach to circuit equations). This also bring some restrictions that we are going to loosen up a bit in later lectures.

To make things more simple we focus on linear circuits for now. This makes it possible for us to write the equations in matrix form. By taking a long hard look at the coefficient matrix and the vector of right-hand values we observe simple patterns (element footprints) that enable us to construct the system of equations on the fly.

« READ LESS

Notation

Currents will be denoted by letter I, voltages by U, and potentials by V. Lowercase letters denote voltages and currents in the time domain, i.e. i(t) and u(t). Capital letters with uppercase indices denote operating point (DC) voltages and currents, i.e. U_{CE} and I_C. When subscripts are numbers we denote the DC voltages and currents with an addittional subscript Q (quiescent), i.e. U_{12Q} and

I_{5Q}.

When working in frequency domain we will be using sinunsoidal voltages with given magnitude and phase expressed with a complexor (i.e. complex value representing a sinusoidal signal). Such voltages and currents will be denoted by a capital letter with lowercase indices, i.e. U_{ce} and I_c. A sinusoidal signal of the form

$$x_{CD}(t) = A\cos(\omega t + \varphi)$$

corresponds to complexor

$$X_{cd} = A\mathrm{e}^{\mathrm{j}\varphi}$$

where j is the imaginary unit (i.e. j^2=-1). Note that a complexor does not contain any information on the frequency of a signal. One can convert a complexor back to time-domain by applying the following formula

$$x_{CD}(t) = \mathrm{Re}(X_{cd}\mathrm{e}^{\mathrm{j}\omega})$$

Kichoff's laws

Throughout this series of lectures we are going to assume that our circuits are small compared to the wavelength of electromagnetic radiation that corresponds to the highest frequency ocurring in our circuit's response. This is a reasonable assumption for a large class of circuits. Under this assumption we can introduce nodes in our circuit. A node is a point where elements are connected to each other.

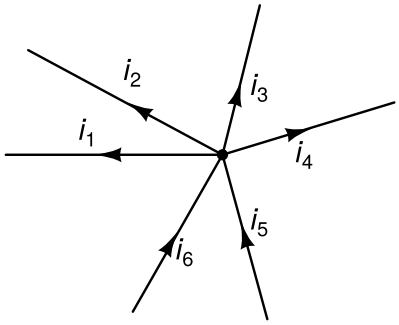


Fig. 1: Kirchoff's current law. The sum of currents flowing into/from a node must be zero.

A node must remain electically neutral (i.e. cannot accumulate charge) when currents flows into a node and out from a node. This requirement arises from charge conservation and is formally stated in Kirchoff's current law (KCL). For a node depicted in Fig. 1 KCL can be written as

$$i_1 + i_2 + i_3 + i_4 - i_5 - i_6 = 0$$

By convention we use the positive sign for currents flowing out from a node, while for currents flowing into a node we use the negative sign. It doesn't matter if, say, i_1 actually flows into the node. In such case if we solve for i_1 we are going to get a negative value which indicates that the current flows in the opposite direction as denoted in the figure.

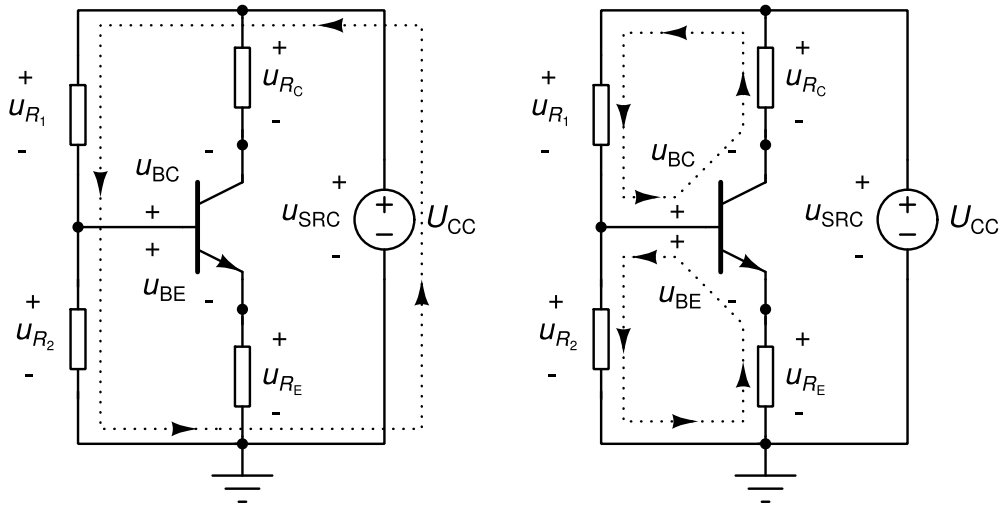


Fig. 2: Kirchoff's voltage law. Three independent paths in a circuit with 4 nodes.

Among all nodes in a circuit we designate one to be the reference node. In schematic we usually denote this by connecting a ground symbol to that node. The voltage between two arbitrary nodes is referred to as branch voltage. Of course we are not interesten in all n(n-1) branch voltages that can be constructed between n nodes. We are interested only in a subset of m branch voltages that comprises the branch voltages of the circuit's elements.

Suppose a circuit comprises n nodes. At this point a question arises: how many KCL equations one must write to formulate the charge conservation law for a circuit? The answer is straightforward - n. But not all of these equations are linearly independent, i.e. some of them can be expressed as linear combinations of others. Actually only n-1 KCL equations are linearly independent. Due to this we omit writing down the KCL equation for the reference node.

If our assumption regarding the size of the circuit compared to the wavelenth of electromagnetic radiation holds then we can assume that the electric field has no curl. Now choose any closed path through the circuit. Adding up branch voltages along a closed path must always result in zero. This requirement is also referred to as the Kirchoff's voltage law (KVL). Take, for instance, the circuit in Fig. 2. We can define 3 closed paths in this circuit. Adding up the branch voltages along any of the paths should result in 0. For the first path (Fig. 2, left) we can write

$$u_{R_1} + u_{R_2} - u_{SRC} = 0$$

We denoted branch voltages where we visited the + sign before the - sign as positive, and the remaining branch voltages as negative. Again when we solve for a particular branch voltage and get a negative result this means that the actual voltage has its + sign where the - sign is drawn in the schematic.

We can write two more KVL equations for this circuit (cf. Fig. 2, right).

$$u_{R_1} + u_{BC} - u_{R_C} = 0$$
$$u_{R_2} - u_{R_E} - u_{BE} = 0$$

Again we can ask ourselves the question: how many KVL equations do I need to write? The answer is not as simple as with KCL equations. But if we assume we have m branch voltages then we need to write m-n+1 KVL equations. And not just any KVL equations. The closed paths used for writing down these equations must be independent. This means that no closed path must be a combination of other closed paths. In Fig. 2 (right) there are two closed path. If we assume another

closed path that starts at the + sign of u_{RC}, goes over u_{R1}, u_{R2}, u_{RE}, u_{BE}, u_{BC}, and u_{RC} we get the following KVL equation

$$u_{R_1} + u_{R_2} - u_{R_E} - u_{BE} + u_{BC} - u_{R_C} = 0$$

we can see that this equation is actually the sum of the previous two equations. This is because the closed path that was used for writing down this equation is a linear combination of the closed paths in Fig 2 (right).

At this point we are faced with two problems

1. How to choose the closed paths for writing down the KVL equations_
2. There are many branches in a typical circuit resulting in many branch voltages. Can we reduce the number of variables by any means?

To answer these questions let us introduce node potentials. A node potential (or nodal voltage) is defined as the voltage between a node and the reference node. We denote nodal voltages by letter V. The nodal voltage of the reference node is by definition 0. If we express all branch voltages with nodal voltages we automatically satisfy KVL for any closed path.

Think about it! A branch voltage for one branch (between nodes 1 and 2) along a path can be expressed as V_1-V_2. The next branch (between nodes 2 and 3) can be expressed as V_2-V_3. When we add up these two branch voltages in a KVL equation the two V_2 terms will cancel out. As we finish adding up all the branch voltages along a closed path all nodal voltages will simply cancel out and the KVL equation will be satisfied. This means that if we introduce nodal voltages and express all branch voltages with them we no longer need to write down KVL equations.

This is, of course, great news. Not only we no longer have to choose m-n+1 independent closed paths in the circuit. We just got rid of m-n+1 euations!

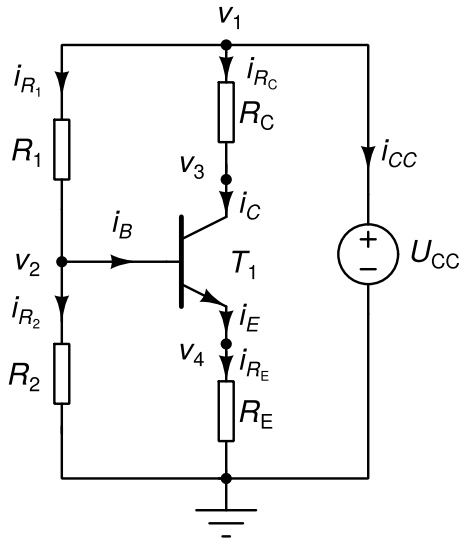


Fig. 3: Nodal voltages and branch currents for the circuit in Fig. 2.

As an excercise let us write down the n-1=4 KCL equations for the circuit in Fig. 3. This gives us n-1=4 equations we later refer to as group A.

$$
\begin{aligned}
i_{R_1} + i_{R_C} + i_{CC} &= 0 \\
-i_{R_1} + i_{R_2} + i_B &= 0 \\
-i_{R_C} + i_C &= 0 \\
-i_E + i_{R_E} &= 0
\end{aligned}
$$

Now let us also express m=7 branch voltages with n-1 nodal voltages. These equations will be our "surrogate" for the KVL equations. Note that the nodal voltage of the reference node is 0. This gives us m=7 equations we later refer to as group B.

$$
\begin{aligned}
u_{R_1} &= v_1 - v_2 \\
u_{R_2} &= v_2 \\
u_{BE} &= v_2 - v_4 \\
u_{BC} &= v_2 - v_3 \\
u_{R_C} &= v_1 - v_3 \\
u_{R_E} &= v_4 \\
u_{SRC} &= v_1
\end{aligned}
$$

Unfortunately the KCL and the "surrogate" KVL equations are not enough for solving the circuit. We have m=8 branch voltages, n-1=4 nodal voltages, and b=8 branch currents appearing in KCL equations. All in all we have m+n-1+b=20 unknowns. On the other hand we have only n-1 KCL equations and m branch voltages expressed with nodal voltages (all in all this is only n-1+m=12 equations). We still need b=8 equations to uniquely determine the solution. To get them we must express the branch currents with the remaining unknowns in a manner different to the one we applied so far.

What is missing are the so-called constitutive relations of circuit elements. These relations connect branch currents to branch voltages. We refer to these equations later as group C. Let us write down the simple ones first. Resistors are subject to Ohm's law. This gives us 4 equations for 4 resistors.

$$
\begin{aligned}
i_{R_1} &= R_1^{-1} u_{R_1} \\
i_{R_2} &= R_2^{-1} u_{R_2} \\
i_{R_C} &= R_C^{-1} u_{R_C} \\
i_{R_E} &= R_E^{-1} u_{R_E}
\end{aligned}
$$

The voltage source sets the branch voltage of the branch where it is placed. This results in another equation in group C.

$$
u_{SRC} = U_{CC}
$$

Finally, the transitor gives us 3 equations. We are not going to dig into transistor models so we are expressing the colelctor and the emitter current simply as two nonlinear functions of u_{BE} and u_{BC}. Due to charge conservation the base current must be equal to the difference between the emitter current and the collector current. This yields a total of 3 addittional equations in group C.

$$i_E = g_E(u_{BE}, u_{BC})$$
$$i_C = g_C(u_{BE}, u_{BC})$$
$$i_B = i_E - i_C$$

The n-1+m+b equations from groups A, B, and C are also referred to as the circuit tableau. We can immediately reduce the number of equations in the circuit tableau if we substitute the equations expressing branch voltages with nodal voltages (group B) into the constitutive relations of elements (group C). This leaves us with n-1+b equations. The set of unknowns is reduced to n-1 nodal voltages and b branch currents.

A further reduction can be achieved if we substitute the resulting constitutive relations of elements into KCL equations (group A). This is possible if we can express the branch currents explicitly with nodal voltages. For the example in Fig. 3 this is not possible. The problem arises because we cannot express the current flowing through the voltage source with branch voltages. To achieve that we would need to solve the circuit tableau which we are trying to simplify without solving it first.

If we leave the voltage sources aside for a moment and focus on circuits without voltage sources (and some other similarly pesky elements) we can actually express branch currents explicitly with nodal voltages. Take for instance the circuit in Fig. 4 with n=4 nodes.

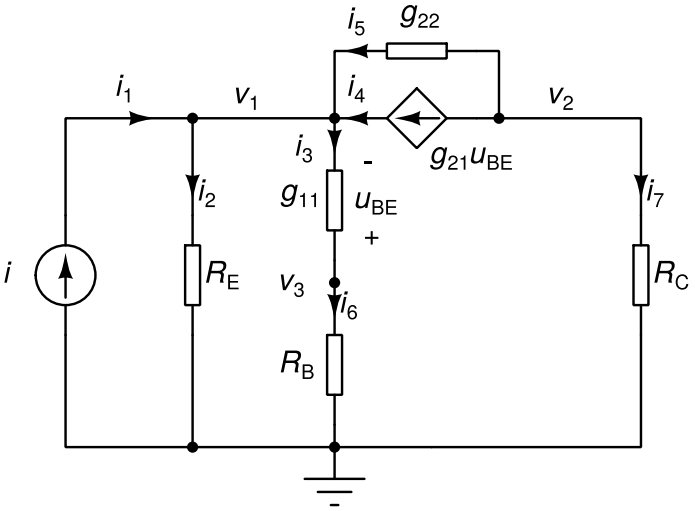


Fig. 4: Nodal equations can be written for circuits where all branch currents can be explicitly expressed with nodal voltages.

Let us first write down the n-1=3 KCL equations.

$$-i_1 + i_2 + i_3 - i_4 - i_5 = 0$$
$$i_4 + i_5 - i_7 = 0$$
$$-i_3 + i_6 = 0$$

Now let us express the b=7 branch currents with nodal voltages.

$$
\begin{aligned}
i_1 &= i \\
i_2 &= R_E^{-1} v_1 \\
i_3 &= g_{11}(v_1 - v_3) \\
i_4 &= g_{21} u_{BE} = g_{21}(v_3 - v_1) \\
i_5 &= g_{22}(v_2 - v_1) \\
i_6 &= R_B^{-1} v_3 \\
i_7 &= R_C^{-1} v_2
\end{aligned}
$$

Substituting these branch currents into KCL equations yields the final set of n-1=3 equations with n-1=3 unknowns (v_1, v_2, and v_3).

$$
\begin{aligned}
(R_E^{-1} + g_{11} + g_{21} + g_{22}) v_1 - g_{22} v_2 - (g_{11} + g_{21}) v_3 &= i \\
(g_{21} + g_{22}) v_1 - (R_C^{-1} + g_{22}) v_2 + g_{21} v_3 &= 0 \\
-g_{11} v_1 + (R_B^{-1} + g_{11}) v_3 &= 0
\end{aligned}
$$

This method of writing down circuit equations based on KCL and explicitly expressed branch currents is also referred to as nodal analysis. The term originates from the fact that the circuit equations are basically the KCL equations that correspond to non-reference nodes. The equations are also referred to as the nodal equations. Solving them results in nodal voltages. Branch voltages can easily be expressed with nodal voltages. It is also fairly easy to express branch currents. Remember we had to explicitly express them with nodal voltages before substituting them in KCL equations.

The equations describing the circuit in Fig. 4 are linear. This is reflected in the fact that all unknowns in the equations appear in linear terms (i.e. as additive terms with a coefficient and unknown raised to the power of 1). In the remainder of this lecture we limit ourselves to circuits with linear equations that contain independent current sources, linear resistors, and linear voltage-controlled current sources. Linear equations can be written in matrix form which is more concise. For the circuit in Fig. 4 the equations in matrix form are

$$
\begin{bmatrix}
R_E^{-1} + g_{11} + g_{21} + g_{22} & -g_{22} & -(g_{11} + g_{21}) \\
-(g_{21} + g_{22}) & R_C^{-1} + g_{22} & g_{21} \\
-g_{11} & 0 & R_B^{-1} + g_{11}
\end{bmatrix}
\begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}
=
\begin{bmatrix} i \\ 0 \\ 0 \end{bmatrix}
$$

Rows of the coefficient matrix and the right-hand side (RHS) vector correspond to n-1=3 KCL equations for all nodes except the reference node. Columns of the matrix and rows of the vector of unknowns correspond to n-1 unknowns (i.e. nodal voltages).

If we take a good look at this system of equations written in matrix form we can see that individual circuit elements contribute to the matrix and the RHS according to a pattern which we are going to refer to as element footprint.

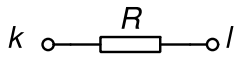


Fig. 5: Resistor.

Let us, for instance, take a resistor with resistance R connected between nodes k and l (Fig. 5). Its element footprint consists of 4 entries in the coefficient matrix because, generally, a resistor contributes current to two nodes. Due to this the entries are at the crossings of the k-th and l-th row (corresponding to k-th and l-th KCL equation) with k-th and l-th column (corresponding to

nodal voltages of k-th and l-th node). Because the coefficient matrix consists of conductances the entries will be $\pm R^{-1}$.

$$\begin{array}{c|ccccccc} & v_1 & \cdots & v_k & \cdots & v_l & \cdots & v_{n-1} \\ \hline \mathrm{KCL}_1 & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \mathrm{KCL}_k & \cdot & \cdots & +R^{-1} & \cdots & -R^{-1} & \cdots & \cdot \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \mathrm{KCL}_l & \cdot & \cdots & -R^{-1} & \cdots & +R^{-1} & \cdots & \cdot \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \mathrm{KCL}_{n-1} & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot \end{array}$$

If one of the pins of the resistor is connected to the reference node the element footprint simplifies to

$$\begin{array}{c|ccccc} & v_1 & \cdots & v_k & \cdots & v_{n-1} \\ \hline \mathrm{KCL}_1 & \cdot & \cdots & \cdot & \cdots & \cdot \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \mathrm{KCL}_k & \cdot & \cdots & R^{-1} & \cdots & \cdot \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \mathrm{KCL}_{n-1} & \cdot & \cdots & \cdot & \cdots & \cdot \end{array}$$

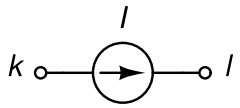


Fig. 6: Independent current source.

The element footprint of an independent current source affects only the RHS vector. Suppose an independent current source draws current I from node k and pushes it to node l (Fig. 6).

$$\begin{array}{c|c} & \mathrm{RHS} \\ \hline \mathrm{KCL}_1 & \cdot \\ \vdots & \vdots \\ \mathrm{KCL}_k & -I \\ \vdots & \vdots \\ \mathrm{KCL}_l & +I \\ \vdots & \vdots \\ \mathrm{KCL}_{n-1} & \cdot \end{array}$$

If a current source pushes current into the ground node the KCL_l contribution is missing. If it draws current from the ground node the KCL_k contribution is missing.

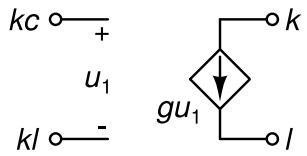


Fig. 7: Voltage-controlled current source.

Finally, suppose we have an voltage controlled current source (VCCS) drawing current from node k

and pushing it to node l with controlling voltage obtained as voltage between nodes kc and lc (controlling voltage is v_{kc}-v_{lc}). Let g denote its transconductance (Fig. 7). Such a controlled source contributes to the KCL equations of nodes k and l at columns corresponding to the control voltage (i.e. columns kc and lc). Its element footprint is

$$\begin{array}{c|ccccccc} & v_1 & \cdots & v_{kc} & \cdots & v_{lc} & \cdots & v_{n-1} \\ \hline \mathrm{KCL}_1 & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \mathrm{KCL}_k & \cdot & \cdots & +g & \cdots & -g & \cdots & \cdot \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \mathrm{KCL}_l & \cdot & \cdots & -g & \cdots & +g & \cdots & \cdot \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \mathrm{KCL}_{n-1} & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot \end{array}$$

If one of the source terminals is connected to the reference node, the corresponding row (either k or l) is missing from the footprint. If one of the nodes defining the controlling voltage is the reference node, the corresponding column (either kc or lc) is missing from the footprint.

Modified nodal analysis

2nd Lecture W2, October

Nodal analysis has one great disadvantage. It cannot handle elements for which constitutive relations express branch voltages with branch currents (e.g. independent voltage source). In this lecture we are going to introduce modified nodal analysis. If we cannot explicitly express a branch current with branch voltages in some constitutive relation we simply keep that branch current as an unknown. To make sure the system of equations is fully determined we must add an additional equation for every branch current we decide to keep. This additional equation is the corresponding element's constitutive relation.

Now we can handle independent voltage sources, linear controlled voltage sources, and linear current controlled sources. Modified nodal analysis is the approach used in most circuit simulators today. With everything we learned up to now it is fairly easy to handle arbitrary linear elements in our equations. We demonstrate this with several examples: ideal transformer, ideal opamp with negative feedback, and inverting amplifier built with an opamp.

« READ LESS

The problem and its solution

In previous lecture we learned how to systematically write down the circuit equations with a small subset of all possible unknowns. We deemed this approach nodal analysis. The main shortcoming

of nodal analysis is that it can't handle independent voltage sources, or any other elements where branch currents cannot be expressed with branch voltages in a straightforward manner.

To sidestep this shortcoming most simulators use the following approach. Instead of trying to express the branch current of a voltage source we simply keep the branch current as an unknown. This increases the number of unknowns by one for every independent voltage source in the circuit. Of course, due to additional unknowns we must also supply additional equations. These additional equations are obtained from the constitutive relations of independent voltage sources. Suppose the voltage source with voltage U is connected with its + pin to node 1 and - pin to node 2. The additional equation is then

$$v_1 - v_2 = U$$

This approach to writing circuit equations is referred to as modified nodal analysis (MNA). Take, for instance, the circuit in Fig. 1. It has n=4 nodes and one independent voltage source. Let us apply MNA to this circuit and write down its equations.

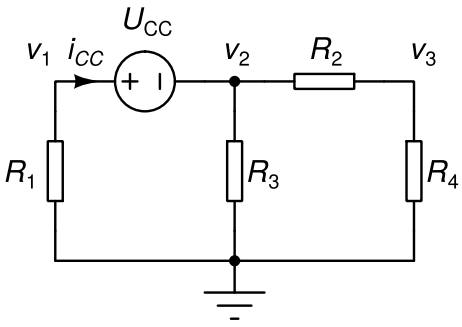


Fig. 1: A simple linear circuit with an independent voltage source.

Writing down the first n-1=3 equations based on KCL is straightforward. Note that we keep the current flowing through an independent voltage source (i_{CC}) as an unknown.

$$\begin{aligned} R_1^{-1} v_1 + i_{CC} &= 0 \\ R_2^{-1}(v_2 - v_3) + R_3^{-1} v_2 - i_{CC} &= 0 \\ R_2^{-1}(v_3 - v_2) + R_4^{-1} v_3 &= 0 \end{aligned}$$

The additional equation based on the constitutive relation of the voltage source is

$$v_1 - v_2 = U_{CC}$$

Rewriting these four equations in matrix form yields

$$\begin{bmatrix} R_1^{-1} & 0 & 0 & 1 \\ 0 & R_2^{-1} + R_3^{-1} & -R_2^{-1} & -1 \\ 0 & -R_2^{-1} & R_2^{-1} + R_4^{-1} & 0 \\ 1 & -1 & 0 & 0 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ i_{CC} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ U_{CC} \end{bmatrix}$$

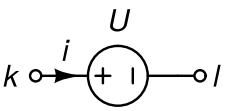


Fig. 2: Independent voltage source.

Looking at the obtained matrix we can construct the element footprint of an independent voltage source (Fig. 2). Let the unknowns be ordered in such manner that node voltages come before

branch currents introduced via MNA. Suppose the source is connected between nodes k (+) and l (-). Let i denote the unknown introduced by MNA (i.e. the current flowing through the voltage source into its + pin). A voltage source then contributes the following element footprint to the coefficient matrix

$$\begin{array}{c} \\ \mathrm{KCL}_1 \\ \vdots \\ \mathrm{KCL}_k \\ \vdots \\ \mathrm{KCL}_l \\ \vdots \\ \mathrm{KCL}_{n-1} \\ \vdots \\ \mathrm{VSRC} \\ \vdots \end{array} \begin{array}{c} \begin{array}{ccccccccc} v_1 & \cdots & v_k & \cdots & v_l & \cdots & v_{n-1} & \cdots & i \end{array} \\ \begin{bmatrix} \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & 1 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & -1 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \cdot & \cdots & 1 & \cdots & -1 & \cdots & \cdot & \cdots & \cdot \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \end{bmatrix} \end{array} \begin{array}{c} \\ \mathrm{KCL}_1 \\ \vdots \\ \mathrm{KCL}_k \\ \vdots \\ \mathrm{KCL}_l \\ \vdots \\ \mathrm{KCL}_{n-1} \\ \vdots \\ \mathrm{VSRC} \\ \vdots \end{array} \begin{array}{c} \mathrm{RHS} \\ \begin{bmatrix} \cdot \\ \vdots \\ \cdot \\ \vdots \\ \cdot \\ \vdots \\ \cdot \\ \vdots \\ U \\ \vdots \end{bmatrix} \end{array}$$

The element footprint of an independent voltage source contributed to the RHS vector is

If one of the pins of an independent voltage source is connected to the ground the corresponding rows and columns of the coefficient matrix and RHS vector are omitted from the footprint.

Example of a nonlinear circuit

Now let us revise the first example of the previous lecture (Fig. 3) and construct its modified nodal equations.

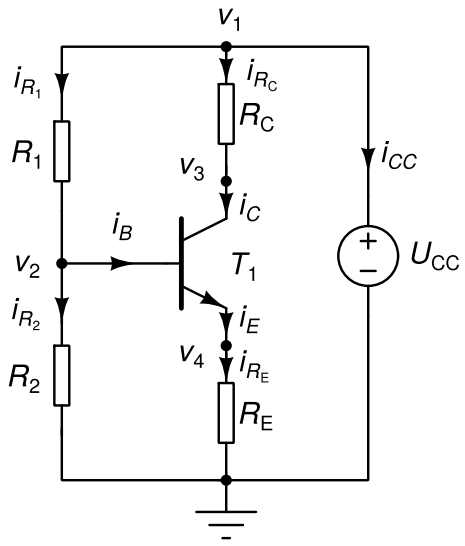


Fig. 3: Circuit from first lecture, revisited.

We start with KCL equations.

$$\begin{aligned} i_{R_1} + i_{R_C} + i_{CC} &= 0 \\ -i_{R_1} + i_{R_2} + i_B &= 0 \\ -i_{R_C} + i_C &= 0 \\ -i_E + i_{R_E} &= 0 \end{aligned}$$

But this time we are going to keep i_{CC} as an unknown in the system of equations. To keep the system of equations fully determined we add one more equation - the constitutive relation of voltage source U_{CC}.

$$v_1 = U_{CC}$$

Finally, we substitute the constitutive relations of resistors and the bipolar transistor ad we arrive at the following system of equations

$$\begin{aligned} R_1^{-1}(v_1 - v_2) + R_C^{-1}(v_1 - v_3) + i_{CC} &= 0 \\ -R_1^{-1}(v_1 - v_2) + R_2^{-1}v_2 + g_E(v_2 - v_4, v_2 - v_3) - g_C(v_2 - v_4, v_2 - v_3) &= 0 \\ -R_C^{-1}(v_1 - v_3) + g_C(v_2 - v_4, v_2 - v_3) &= 0 \\ -g_E(v_2 - v_4, v_2 - v_3) + R_E^{-1}v_4 &= 0 \\ v_1 &= U_{CC} \end{aligned}$$

We cannot write this system of equations in matrix form because three of the equations (second, third, and fourth) are nonlinear - they contain nonlinear functions g_C and g_E.

Other linear controlled sources

In previous chapter we learned how to include voltage-controlled current sources in circuit equations. With MNA we can handle other types controlled sources. Let us limit ourselves for now to linear controlled sources.

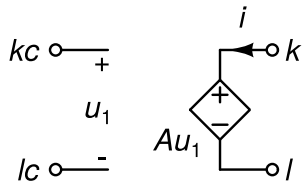


Fig. 4: Voltage-controlled voltage source.

A voltage-controlled voltage source (VCVS) connected between nodes k (+) and l (-), controlled by voltage between nodes kc (+) and lc (-) with gain A (Fig. 4) has the following constitutive relation

$$v_k - v_l - A(v_{kc} - v_{lc}) = 0$$

Such a source contributes to KCL equations for nodes k and l. Let i denote the unknown corresponding to the current flowing through such source (from node k to node l). The element footprint of a VCVS contributed to the coefficient matrix is

$$\begin{array}{c|ccccccccccccc} & v_1 & \cdots & v_k & \cdots & v_l & \cdots & v_{kc} & \cdots & v_{lc} & \cdots & v_{n-1} & \cdots & i & \cdots \\ \hline \vdots & \vdots & & \vdots & & \vdots & & \vdots & & \vdots & & \vdots & & \vdots & \\ \mathrm{KCL}_k & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & 1 & \cdots \\ \vdots & \vdots & & \vdots & & \vdots & & \vdots & & \vdots & & \vdots & & \vdots & \\ \mathrm{KCL}_l & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & -1 & \cdots \\ \vdots & \vdots & & \vdots & & \vdots & & \vdots & & \vdots & & \vdots & & \vdots & \\ \mathrm{KCL}_{n-1} & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots \\ \vdots & \vdots & & \vdots & & \vdots & & \vdots & & \vdots & & \vdots & & \vdots & \\ \mathrm{VCVS} & \cdot & \cdots & 1 & \cdots & -1 & \cdots & -A & \cdots & A & \cdots & \cdot & \cdots & \cdot & \cdots \\ \vdots & \vdots & & \vdots & & \vdots & & \vdots & & \vdots & & \vdots & & \vdots & \end{array}$$

If any of the output pins is connected to the ground the corresponding row and column (k or l) is omitted from the footprint. Similarly if any of the controlling pins is connected to the ground the corresponding column (kc or lc) is omitted from the footprint. VCVS does not contribute to the RHS vector.

We can also handle current-controlled sources. The controlling current must be one of the unknowns in the system of equations (i.e. in our case a current flowing through a voltage source). If no such unknown is available we can add one by inserting a zero-voltage independent voltage source is series with the required branch current.

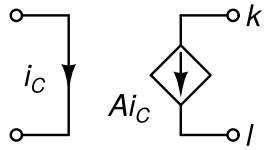


Fig. 5: Current-controlled current source.

Suppose we have a current-controlled current source (CCCS) connected between nodes k and l with gain A (Fig. 5). Let i_C denote the controlling current. A CCCS contributes a term of the form Ai to KCL equations for nodes k and l. Unlike VCVS it does not add an equation to the system because it does not introduce an additional unknown. CCCS does not contribute to the RHS vector. The element footprint of a CCCS contribution to the coefficient matrix is

$$\begin{array}{c} \\ \mathrm{KCL}_1 \\ \vdots \\ \mathrm{KCL}_k \\ \vdots \\ \mathrm{KCL}_l \\ \vdots \\ \mathrm{KCL}_{n-1} \\ \vdots \end{array} \begin{array}{c} v_1 \\ \cdots \\ \cdot \\ \vdots \\ \cdot \\ \vdots \\ \cdot \\ \vdots \\ \cdot \\ \vdots \end{array} \begin{array}{c} \cdots \\ \ddots \\ v_{n-1} \\ \ddots \\ \cdot \\ \ddots \\ \cdot \\ \ddots \\ \cdot \\ \ddots \end{array} \begin{array}{c} \cdots \\ \ddots \\ \cdot \\ \ddots \\ A \\ \ddots \\ -A \\ \ddots \\ \cdot \\ \ddots \end{array} \begin{array}{c} i_C \\ \cdots \\ \cdot \\ \ddots \\ \cdot \\ \ddots \\ \cdot \\ \ddots \\ \cdot \\ \ddots \end{array} \begin{array}{c} \cdots \\ \ddots \\ \cdot \\ \ddots \\ \cdot \\ \ddots \\ \cdot \\ \ddots \\ \cdot \\ \ddots \end{array}$$

If a CCCS is connected with one of its pins to the ground the corresponding row (either k or l) is omitted from the footprint.

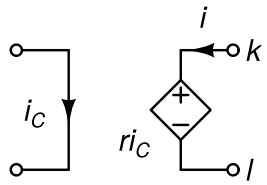


Fig. 6: Current-controlled voltage source.

A current-controlled voltage source (CCVS) is a bit more complicated. Suppose it is connected between nodes k (+) and l (-), is controlled by current i_C (which in turn must be an unknown in the system of equations), and its transimpedance is r (Fig. 6). A CCVS introduces an additional unknown into the system because it is a voltage source. Let i denote this unknown. The constitutive relation of a CCVS is

$$v_k - v_l - ri_C = 0$$

A CCVS does not contribute to the RHS vector. Its element footprint in the coefficient matrix is

$$\begin{array}{c} \\ \mathrm{KCL}_1 \\ \vdots \\ \mathrm{KCL}_k \\ \vdots \\ \mathrm{KCL}_l \\ \vdots \\ \mathrm{KCL}_{n-1} \\ \vdots \\ \mathrm{CCVS} \\ \vdots \end{array} \begin{array}{c} v_1 \\ \cdots \\ \cdot \\ \vdots \\ \cdot \\ \vdots \\ \cdot \\ \vdots \\ \cdot \\ \vdots \\ \cdot \\ \vdots \end{array} \begin{array}{c} \cdots \\ \ddots \\ v_k \\ \ddots \\ \cdot \\ \ddots \\ \cdot \\ \ddots \\ \cdot \\ \ddots \\ 1 \\ \ddots \end{array} \begin{array}{c} \cdots \\ \ddots \\ v_l \\ \ddots \\ \cdot \\ \ddots \\ \cdot \\ \ddots \\ \cdot \\ \ddots \\ -1 \\ \ddots \end{array} \begin{array}{c} \cdots \\ \ddots \\ v_{n-1} \\ \ddots \\ \cdot \\ \ddots \\ \cdot \\ \ddots \\ \cdot \\ \ddots \\ \cdot \\ \ddots \end{array} \begin{array}{c} \cdots \\ \ddots \\ i_C \\ \ddots \\ \cdot \\ \ddots \\ \cdot \\ \ddots \\ -r \\ \ddots \\ \cdot \\ \ddots \end{array} \begin{array}{c} \cdots \\ \ddots \\ i \\ \ddots \\ \cdot \\ \ddots \\ \cdot \\ \ddots \\ \cdot \\ \ddots \\ \cdot \\ \ddots \end{array} \begin{array}{c} \cdots \\ \cdots \\ \ddots \\ \ddots \\ \ddots \\ \ddots \\ \ddots \\ \ddots \\ \ddots \\ \ddots \\ \ddots \end{array}$$

If a CCVS is connected with one of its pins to the ground the corresponding row (either k or l) and column (either v_k or v_l) is omitted from the footprint.

Inverting amplifier built with a real opamp

In this example we are going to write down the equations of an inverting amplifier which is obtained if we add two resistors to an opamp (Fig. 7, left). The opamp has finite gain (A). Therefore we can model it as a linear VCVS (Fig. 7, right).

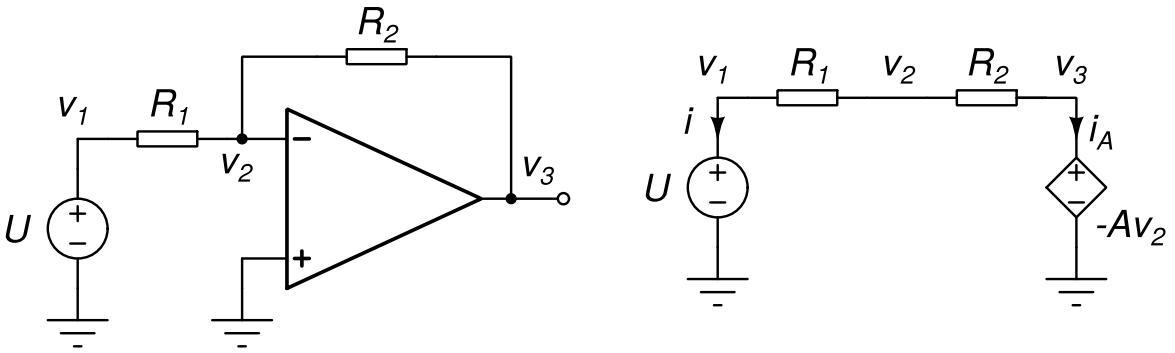


Fig. 7: Inverting amplifier (left) and its model (right).

The circuit has 4 nodes and tho voltage sources. Two additional unknowns are added to the system of equations (i and i_A) which comprises 3 KCL equations and the constitutive relations of the independent voltage source and the VCVS.

$$\begin{aligned} R_1^{-1}(v_1 - v_2) + i &= 0 \\ R_1^{-1}(v_2 - v_1) + R_2^{-1}v_2(v_2 - v_3) &= 0 \\ R_2^{-1}(v_3 - v_2) + i_A &= 0 \\ v_1 &= U \\ v_3 &= -Av_2 \end{aligned}$$

After rearranging these equations we can write them in matrix form.

$$\begin{bmatrix} R_1^{-1} & -R_1^{-1} & 0 & 1 & 0 \\ -R_1^{-1} & R_1^{-1} + R_2^{-1} & -R_2^{-1} & 0 & 0 \\ 0 & -R_2^{-1} & R_2^{-1} & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & A & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ i \\ i_A \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ U \\ 0 \end{bmatrix}$$

Ideal opamp with negative feedback

An ideal opamp with negative feedback (Fig. 8, left) is connected to three nodes. Nodes k and l represent the non-inverting and the inverting input, while node m is its output. The current flowing into the input terminals is zero. The output of an opamp behaves as a controlled voltage source connected between node m and ground (Fig. 8, right).

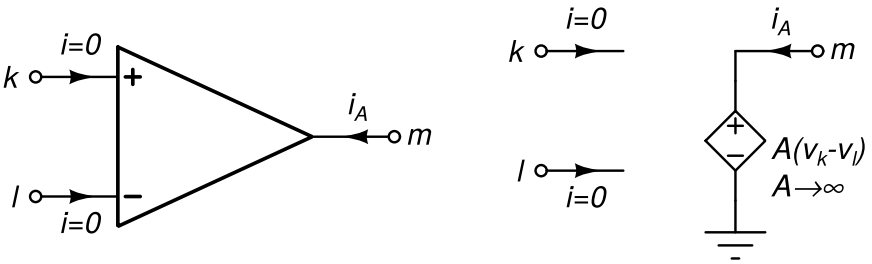


Fig. 8: Ideal opamp with negative feedback (left) and its model (right).

The controlling voltage is the voltage between the non-inverting and the inverting input. In an indeal opamp the gain is infinite. When such an opamp is used in a linear circuit with a negative feedback loop the opamp produces an output voltage that forces the controlling voltage to zero because this is the only way for satisfying the constitutive relation of the opamp without an infinite voltage at its output. Because its output is a controlled voltage source an unknown representing its current is added to the system of equations. The constitutive relation of an ideal opamp with

negative feedback is

$$v_k - v_l = 0$$

An ideal opamp with negative feedback does not contribute to the RHS vector. In the matrix of coefficients it contributes to a single equation (its constitutive relation). Its element footprint in the matrix of coefficients is

$$\begin{array}{c|ccccccccc} & v_1 & \cdots & v_k & \cdots & v_l & \cdots & v_{n-1} & \cdots & i_A & \cdots \\ \hline \mathrm{KCL}_1 & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots \\ \mathrm{KCL}_k & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots \\ \mathrm{KCL}_l & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots \\ \mathrm{KCL}_{n-1} & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots \\ \mathrm{OPAMP} & \cdot & \cdots & 1 & \cdots & -1 & \cdots & \cdot & \cdots & \cdot & \cdots \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots \end{array}$$

Ideal transformer

An ideal transformer (Fig. 9, left) has four pins where k1 and l1 represent the terminals of the primary coil and k2 and l2 represent the terminals of the secondary coil. It is obtained from a real transformer when the magnetic coupling is ideal and inductances of the coils are infinite.

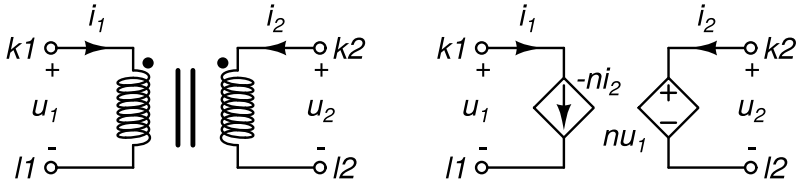


Fig. 9: Ideal transformer (left) and its model (right).

It can be described with two equations:

$$u_2 = nu_1$$
$$i_1 = -ni_2$$

where n is the ratio of secondary vs. primary coil windings. We can express the first equation with node voltages

$$nv_{k1} - nv_{l1} - v_{k2} + v_{l2} = 0$$

Based on these equations we can construct a model (Fig. 9, right) which is the basis for writing down the element footprint of an ideal transformer. The model comprises one VCVS modelling the secondary coil and one CCCS modelling the primary coil. Because we have a voltage source in the model an ideal transformer introduces a new unknown in the system of equations (i_2). Due to this we need an additional equation in the system. We obtain it from the first equation describing the transformer (which is actually the constitutive relation of a VCVS). The second equation is the constitutive relation of the CCCS. An ideal transformer does not contribute to the RHS vector. The element footprint in the coefficient matrix is

$$
\begin{array}{c|ccccccccccccc}
 & v_1 & \cdots & v_{k1} & \cdots & v_{l1} & \cdots & v_{k2} & \cdots & v_{l2} & \cdots & v_{n-1} & \cdots & i_2 & \cdots \\
 & & & & & & & & & & & & & & \cdots \\
\hline
\mathrm{KCL}_1 & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots \\
\vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots \\
\mathrm{KCL}_{k1} & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & -n & \cdots \\
\vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots \\
\mathrm{KCL}_{l1} & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & n & \cdots \\
\vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots \\
\mathrm{KCL}_{k2} & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & 1 & \cdots \\
\vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots \\
\mathrm{KCL}_{l2} & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & -1 & \cdots \\
\vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots \\
\mathrm{KCL}_{n-1} & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots \\
\vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots \\
\mathrm{XFORM} & \cdot & \cdots & n & \cdots & -n & \cdots & -1 & \cdots & 1 & \cdots & \cdot & \cdots & \cdot & \cdots \\
\vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots
\end{array}
$$

If the primary winding is connected with one of its pins to the ground the corresponding row (either k1 or l1) and column (either v_{k1} or v_{l1}) is omitted from the footprint. Similarly, if the secondary winding is connected with one of its pins to the ground the corresponding row (either k2 or l2) and column (either v_{k2} or v_{l2}) is omitted from the footprint.

Solving systems of linear equations

3rd Lecture W3, October

Solving systems of linear equations is nothing new. Several approaches were developed in the past. For starters we take a look at Gaussian elimination. We examine its computational cost and show how it can fail. To improve the robustness of Gaussian elimination we introduce pivoting. Gaussian elimination leads to many unnecessary operations when it is used for solving multiple systems of equations with the same coefficient matrix (which is common in circuit simulation). To reduce the number of operations we introduce LU-decomposition followed by backward and forward substitution.

« READ LESS

Gaussian elimination

We are going to explain Gaussian elimination on an example. Suppose we have the following linear system comprising three equations.

$$\begin{aligned} x_1 + 2x_2 + 3x_3 &= -7 \\ 2x_1 + 3x_2 - 5x_3 &= 9 \\ -6x_1 - 8x_2 + x_3 &= 22 \end{aligned}$$

We can solve this system by using the first equation to eliminate x_1 from the remaining equations. This yields a new system of two equations with two unknowns. By applying the procedure recursively we end up with one equation and one unknown. The procedure is referred to as Gaussian elimination. In our example we start it by multiplying the first equation with -2/1 (i.e. the negative of the coefficient corresponding to x_1 in the second equation divided by the coefficient corresponding to x_1 in the first equation) and adding it to the second equation. This eliminates x_1 from the second equation resulting in

$$\begin{aligned} x_1 + 2x_2 + 3x_3 &= -7 \\ -x_2 - 11x_3 &= 23 \\ -6x_1 - 8x_2 + x_3 &= 22 \end{aligned}$$

Next we multiply the first equation by 6 and add it to the third equation.

$$\begin{aligned} x_1 + 2x_2 + 3x_3 &= -7 \\ -x_2 - 11x_3 &= 23 \\ 4x_2 + 19x_3 &= -20 \end{aligned}$$

At this point we eliminated x_1 from the second and third equation. These two equations contain only two unknowns. Now we repeat the procedure and eliminate x_2 from the third equation by multiplying the second equation with 4 and adding it to the third equation.

$$\begin{aligned} x_1 + 2x_2 + 3x_3 &= -7 \\ -x_2 - 11x_3 &= 23 \\ -25x_3 &= 72 \end{aligned}$$

A more convenient way for representing a system of equations is to use the form of an extended matrix where rows correspond to the equations, the first 3 columns correspond to the coefficients in front of the unknowns, and the fourth column corresponds to the right-hand sides of the equations.

$$\begin{bmatrix} 1 & 2 & 3 & -7 \\ 0 & -1 & -11 & 23 \\ 0 & 0 & -25 & 72 \end{bmatrix}$$

The matrix obtained after Gaussian elimination has zeros below the main diagonal and is also referred to as the row echelon form. Suppose our system of equations has n equations. Gaussian elimination requires n(n-1)/2 divisions, (n-1)n(n+1)/3 multiplications, and (n-1)n(n+1)/3 additions. Roughly speaking the computational burden of a Gaussian elimination grows proportionally to n^3.

From row echelon form we can quickly obtain the solution of the linear system by applying back-substitution. First we solve the last equation which has only one unknown (x_3). This results in x_3=-72/25. We substitute this into the second equation to eliminate x_3 which leaves us with only one unknown (x_2).

$$-x_2 + 792/25 = 23$$

From here we obtain x_2=217/25. Finally, we substitute x_2 and x_3 into the first equation which yields.

$$x_1 + 434/25 - 216/25 = -7$$

resulting in x_1=-393/25. Now we have the solution of the system of equations.

Assuming we have n equations back-substitution requires n(n-1)/2 multiplications, n(n-1)/2 subtractions, and n divisions. The computational complexity of back-substitution grows proportionally to n^2.

Gaussian elimination can handle systems of several thousands of equations. Larger systems of equations can be solved with iterative methods. Also note that large systems require a lot of memory for storing the matrix of coefficients. Take for instance n=10000. Matrices of this size have 10^8 elements which (if double precision is used) require 800MB of memory. On the other hand solving such a large system requires roughly $n^3/3=0.33\ 10^{12}$ multiplications. With a processor running at 3.3GHz and performing one multiplication per clock cycle the multiplications alone require over 100 seconds of CPU time.

Numerical error and partial pivoting

Rounding errors can significantly affect the result obtained with Gaussian elimination and back-substitution. Take for instance the system of equations represented by the following extended matrix

$$\begin{bmatrix} 0.143 & 0.357 & 2.01 & -5.17 \\ -1.31 & 0.911 & 1.99 & -5.46 \\ 11.2 & -4.30 & -0.605 & 4.42 \end{bmatrix}$$

The solution of this system is x_1=1, x_2=2, and x_3=-3. Due to finite precision the result of every operation is rounded. For double floating point precision this rounding takes place at the 16-th significant decimal digit. To illustrate the problem we perform Gaussian elimination and round every obtained matrix entry to 3 significant digits. First we eliminate nonzero entries below the first diagonal element.

$$\begin{bmatrix} 0.143 & 0.357 & 2.01 & -5.17 \\ 0 & 4.18 & 20.4 & -52.8 \\ 0 & -32.3 & 158 & 409 \end{bmatrix}$$

Next we eliminate the nonzero entry below the second diagonal element.

$$\begin{bmatrix} 0.143 & 0.357 & 2.01 & -5.17 \\ 0 & 4.18 & 20.4 & -52.8 \\ 0 & 0 & 316 & 1.00 \end{bmatrix}$$

From here we obtain x_3 with the first step of back-substitution which yields x_3=0.00316. This is way off from what we expected to get (-3). What went wrong? Obviously numerical error accumulated throughout Gaussian elimination and ruined the result.

When we are eliminating element a_{ki} in k-th row by adding scaled i-th row to k-th row the scaling factor is determined by $-a_{ki}/a_{ii}$. When the absolute value of this factor is large the added entries of scaled i-th row "drown" the much smaller entries of k-th row in numerical noise. We can avoid this problem if we keep the absolute value a_{ki}/a_{ii} as low as possible by making sure the absolute value of a_{ii} is as large as possible.

This is possible if we introduce row swapping. Swapping two rows is equivalent to swapping two equations. Clearly the final result is not affected. If we swap the i-th row with the one that has the largest absolute entry in the i-th column between a_{ii} and a_{ni} we make sure the scaling factor will be minimized. The entry that replaces a_{ii} is also referred to as the pivot. The aforementioned procedure is called partial pivoting.

Let us apply Gaussian elimination with partial pivoting to see if we do any better. Again we are going to round obtained matrix entries to 3 significant digits. The pivot below and including a_{11} is found in the third row. Therefore we swap the first and the third row and start Gaussian elimination on the following extended matrix

$$\begin{bmatrix} 11.2 & -4.30 & -0.605 & 4.42 \\ -1.31 & 0.911 & 1.99 & -5.46 \\ 0.143 & 0.357 & 2.01 & -5.17 \end{bmatrix}$$

After eliminating entries below the first diagonal element we get.

$$\begin{bmatrix} 11.2 & -4.30 & -0.605 & 4.42 \\ 0 & 0.408 & 1.92 & -4.94 \\ 0 & 0.412 & 2.02 & -5.23 \end{bmatrix}$$

Before eliminating entries below the second diagonal element we swap the second and the third row so that a_{32} becomes the new pivot in the second column.

$$\begin{bmatrix} 11.2 & -4.30 & -0.605 & 4.42 \\ 0 & 0.412 & 2.02 & -5.23 \\ 0 & 0.408 & 1.92 & -4.94 \end{bmatrix}$$

The next step completes the Gaussian elimination and yields

$$\begin{bmatrix} 11.2 & -4.30 & -0.605 & 4.42 \\ 0 & 0.412 & 2.02 & -5.23 \\ 0 & 0 & -0.0804 & 0.239 \end{bmatrix}$$

The first step of back-substitution now gives us x_3=-2.97 which is close to the correct value (-3) and much better than we did when no pivoting was used.

Note that there also exists a more elaborate procedure called complete pivoting where the pivot is chosen between matrix entries in the rectangle between a_{ii} and a_{nn}. In this case we not only swap rows, but also columns. Swapping columns corresponds to swapping unknowns. Therefore the values of the unknowns obtained after the back-substitution must be swapped in the same manner as columns were swapped during complete pivoting to obtain the correct result.

LU decomposition

Often we need to solve multiple systems of equations with identical coefficient matrices which differ only in the value of the RHS vector. In such cases we use matrix decomposition. A matrix can be decomposed in a product of two matrices in many ways. A commonly used decomposition is the LU decomposition where we express matrix A as

$$A = LU$$

where L is a lower triangular matrix with all zeros above its main diagonal and U is an upper

triangular matrix with all zeros below its main diagonal. The diagonal entries of matrix L are all equal to 1. Once the LU decomposition of a matrix is known solving a linear system is fairly cheap in computational sense. Suppose we are solving a linear system expressed in matrix form as Ax=b. We can write

$$Ax = LUx = Lz$$

where z is a vector. To solve the linear system we first solve Lz=b. Because L is lower triangular we can apply a procedure referred to as forward substitution which is similar to back-substitution, except that now we start by expressing the first component of z from the first equation. We substitute this into the second equation which gives us the second component of z. By repeating this procedure we obtain all components of z in the same number of operations as are required for one back-substitution (i.e. the computational complexity grows proportionally to n^2). In fact we don't even have to perform division because the diagonal entries of L are all equal to 1.

Because z=Ux by definition and U is upper triangular we can solve for x by applying back-substitution to Ux=z. The computational complexity of this step also grows with n^2. To conclude, the computational complexity of solving a linear system of equations when the LU decomposition is known in advance is proportional to n^2. This is, of course, much cheaper than performing a Gaussian elimination.

Take for instance the following linear system Ax=b where the LU decomposition of A is

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & -5 \\ -6 & -8 & 1 \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -6 & -4 & 1 \end{bmatrix}}_{L} \underbrace{\begin{bmatrix} 1 & 2 & 3 \\ 0 & -1 & -11 \\ 0 & 0 & -25 \end{bmatrix}}_{U}$$

and

$$b = \begin{bmatrix} -7 \\ 9 \\ 22 \end{bmatrix}$$

We start by solving Lz=b which yields

$$\begin{aligned} z_1 &= b_1 = -7 \\ z_2 &= b_2 - 2z_1 = 23 \\ z_3 &= b_3 + 6z_1 + 4z_2 = 72 \end{aligned}$$

and

$$z = \begin{bmatrix} -7 \\ 23 \\ 72 \end{bmatrix}$$

Finally, we solve Ux=z.

$$\begin{aligned} x_3 &= (-25)^{-1} z_3 = -72/25 \\ x_2 &= (-1)^{-1}(z_2 + 11x_3) = 217/25 \\ x_1 &= 1^{-1}(z_1 - 2x_2 - 3x_3) = -393/25 \end{aligned}$$

The result is

$$x = \begin{bmatrix} -393/25 \\ 217/25 \\ -72/25 \end{bmatrix}$$

LU decomposition algorithm

The LU decomposition of a matrix can be computed as a byproduct of Gaussian elimination. The upper triangular part of the matrix (including the diagonal entries) obtained after Gaussian elimination is the U matrix. The L matrix is composed of ones on the main diagonal while subdiagonal entries (l_{ij}) are the negatives of the factors we used for multiplying the row containing the pivot (a_{ii}) when we were eliminating element a_{ij}. To demonstrate the algorithm let us perform Gaussian elimination side by side with LU decomposition. Note that this time we are going to work with matrix A instead of the extended matrix so there will only be n columns.

Take, for instance, the following matrix A. At this point we know little of L and U.

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & -5 \\ -6 & -8 & 1 \end{bmatrix} \quad L = \begin{bmatrix} 1 & 0 & 0 \\ ? & 1 & 0 \\ ? & ? & 1 \end{bmatrix} \quad U = \begin{bmatrix} ? & ? & ? \\ 0 & ? & ? \\ 0 & 0 & ? \end{bmatrix}$$

To eliminate the entries in the first column below the main diagonal we multiply the first row with -2/1 and add it to the second row. This eliminates the entry in the first column of the second row. The entry in the first column of the second row of matrix L is equal to the negative of the multiplier, i.e. 2/1. Similarly we multiply the first row with -(-6)/1 and add it to the third row to eliminate the entry in the first column of the third row. The element of L in the first column of the third row is therefore -6/1=-6. After we eliminate all subdiagonal entries in the first column we can copy the diagonal element of the first row along with the ones lying to its right into matrix U.

$$\begin{bmatrix} 1 & 2 & 3 \\ 0 & -1 & -11 \\ 0 & 4 & 19 \end{bmatrix} \quad L = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -6 & ? & 1 \end{bmatrix} \quad U = \begin{bmatrix} 1 & 2 & 3 \\ 0 & ? & ? \\ 0 & 0 & ? \end{bmatrix}$$

Next, we eliminate the subdiagonal entries in the second column. There is only one such entry. We eliminate it by multiplying the second row with -4/(-1)=4 and add the result to the third row. The corresponding element of matrix L is therefore -4. Because at this point the elimination of subdiagonal entries in the second column is complete we can copy the diagonal element of the resulting matrix along with all elements to its right into matrix U.

$$\begin{bmatrix} 1 & 2 & 3 \\ 0 & -1 & -11 \\ 0 & 0 & -25 \end{bmatrix} \quad L = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -6 & -4 & 1 \end{bmatrix} \quad U = \begin{bmatrix} 1 & 2 & 3 \\ 0 & -1 & -11 \\ 0 & 0 & ? \end{bmatrix}$$

Finally, there are no subdiagonal elements to eliminate in the last (third) column. Therefore we copy the diagonal element from the third row into matrix U. The LU decomposition is complete.

$$\begin{bmatrix} 1 & 2 & 3 \\ 0 & -1 & -11 \\ 0 & 0 & -25 \end{bmatrix} \quad L = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -6 & -4 & 1 \end{bmatrix} \quad U = \begin{bmatrix} 1 & 2 & 3 \\ 0 & -1 & -11 \\ 0 & 0 & -25 \end{bmatrix}$$

Note how the matrix we obtained after Gaussian elimination is actually the U matrix.

In our last example we didn't use partial pivoting. In order to avoid large numerical errors partial pivoting is necessary. When performing LU decomposition with partial pivoting we must record the row exchanges that took place during LU decomposition. When solving for z from LU=b' we use b' instead of b where b' is obtained by exchanging the elements of b in the same way we exchanged rows during LU decomposition.

Sparse matrices, solving large systems of linear equations

4th Lecture W1, November

Sparse matrices are matrices where most entries are zero. Matrices of equations corresponding to real-world circuits are sparse. This makes it possible to analyze large circuits without prohibitively large memory requirements. But there is a catch. Performing LU-decomposition of sparse matrices must make sure that as few as possible new nonzero entries are created during decomposition (fill-in). Unfortunately one cannot have both - a small fill-in and small numerical error. This is because avoiding fill-in dictates the choice of matrix pivots which now cannot be chosen in a way that would result in minimal numerical error.

« READ LESS

Advantages

Suppose we have a large circuit with many nodes (say n=1000). Assuming that the number of voltage sources is small the coefficient matrix has approximately $n^2=10^6$ elements. Storing these elements as double precision real numbers requires requires 8 bytes for every matrix entry. Therefore just storing the matrix requires 8MB of memory. Of course, today this is not that much of an issue. But, on the other hand, circuits with several 10000 or even 100000 nodes are often simulated, (e.g. in integrated circuit design when a full-chip verification is performed). Such large circuits with n=100000 would require 80GB of memory for storing the coefficient matrix. This is even on today's scale a lot of memory.

Memory requirements grow proportionally with n^2. This means that doubling the number of rows quadruples the number of elements. If we consider that according to Moore's law the number of transistors in an integrated circuits doubles every 18 months (albeit we are not sure how long this trend is going to continue) we see that we must wait 36 months (or 3 years) so that the memory capacity can accommodate a circuit twice as large as the circuit which pushes the limits of matrix storage today.

But storage is not the only problem. Note how the number of operations required for solving a linear system of equations (LU decomposition + forward substitution + back-substitution) grows proportionally with n^3. As n grows we can expect to hit the barrier imposed by the computational time growing beyond anything reasonable even sooner that we are going to run out of memory.

The question arises: can we circumvent these limitations in any way? The answer is yes. But this is possible only if our matrices have a particular property: most of the matrix elements must be zero. Fortunately this is the case with coefficient matrices of circuits. In almost all circuits every node is connected to a small subset of remaining nodes (either electrically or via controlled sources). Therefore every KCL equation depends on a small subset of all unknowns in the system of equations. Consequently most matrix entries are zero. Such matrices are also referred to as sparse matrices. The term dense matrix is used for matrices for which all elements are considered to be nonzero (and thus stored in computer's memory regardless of their value).

When a matrix is sparse we don't have to store all its elements. It suffices to store only the nonzero entries. This brings along certain overhead as we must store not only the value of a nonzero matrix element, but also its position (indices i and j). If we assume this overhead consumes 8 bytes (4 bytes per integer) we see that the break even point for storage requirements is reached for matrices where at least half of the matrix elements are equal to zero. Most sparse matrices that arise from real-world circuits are significantly more sparse.

Sparse matrices have an additional advantage. Performing LU decomposition on a sparse matrix can be much cheaper if performed in a certain way. The number of operations for such an LU decomposition is proportional to the number of nonzero matrix entries.

LU decomposition and fill-in

When performing LU decomposition the result is often stored in the initial input matrix. The subdiagonal elements of the resulting matrix store the subdiagonal elements of matrix L. Elements on the diagonal of the resulting matrix and elements above it correspond to elements of matrix U. Since the diagonal elements of L are always 1 we don't have to store them. Now let us visualize the steps of LU decomposition performed on a sparse matrix. Let 'a', 'l', and 'u' denote nonzero entries of the coefficient matrix, L matrix, and U matrix. Zero entries are denoted by '0'. We start with a sample sparse matrix

$$\begin{bmatrix} a & a & a & a \\ a & a & 0 & 0 \\ a & 0 & a & 0 \\ a & 0 & 0 & a \end{bmatrix}$$

After the first step of Gaussian elimination we have

$$\begin{bmatrix} u & u & u & u \\ l & x & x & x \\ l & x & x & x \\ l & x & x & x \end{bmatrix}$$

We denote by 'x' an intermediate result. Such intermediate results occur in columns where the row used for eliminating subdiagonal elements has a nonzero element. Some intermediate results create new nonzero entries in the matrix. Note that even if an intermediate result is zero we treat it as a nonzero element. The second step of elimination takes care of subdiagonal elements below the second diagonal element and yields

$$\begin{bmatrix} u & u & u & u \\ l & u & u & u \\ l & l & x & x \\ l & l & x & x \end{bmatrix}$$

Finally, the third step (which is the last one) results in

$$\begin{bmatrix} u & u & u & u \\ l & u & u & u \\ l & l & u & u \\ l & l & l & u \end{bmatrix}$$

Note that there is no need to perform the fourth step because there are no subdiagonal elements below the fourth diagonal element. Despite initial matrix being sparse the final result is a dense matrix which means that we gained nothing in terms of memory consumption. It is reasonable to expect that LU decomposition will change some zero entries in the matrix to nonzero entries. Every such additional nonzero entry is referred to as fill-in. To keep fill-in as small as possible the pivots for Gaussian elimination must be chosen accordingly. But before we explore pivoting in sparse LU decomposition let us first introduce a more sophisticated pivoting approach.

Complete pivoting

The pivoting described in previous lecture is also referred to as partial pivoting where for the i-th step of LU decomposition the pivot is chosen among the i-1 subdiagonal entries.

A more sophisticated pivoting approaches chooses the pivot for the i-th step of LU decomposition in a submatrix including all rows below the i-th diagonal entry and all columns to the right including the i-th column. This approach is deemed complete pivoting. If the pivot is chosen outside i-th column one has to permute the columns of the matrix. Permuting the columns corresponds to permuting the unknowns. Just like row permutations, the column permutations must also be stored. This information is needed when we are solving the system of equations. After the backward substitution is completed the resulting vector must be permuted in the opposite order as we permuted the matrix columns to produce the correct vector of unknowns.

Choosing the pivot in sparse LU decomposition

Because the matrix is sparse we don't expect many pivot candidates in the column below the diagonal entry. Due to this partial pivoting is not the best choice for sparse matrices and complete pivoting is usually applied. Let us revisit the last example, but this time let us reverse the order of rows and columns (i.e. we permute rows and columns). We start LU decomposition with the following matrix

$$\begin{bmatrix} a & 0 & 0 & a \\ 0 & a & 0 & a \\ 0 & 0 & a & a \\ a & a & a & a \end{bmatrix}$$

The first step of LU decomposition elmininated the subdiagonal entries in the first column and yields

$$\begin{bmatrix} u & 0 & 0 & u \\ 0 & a & 0 & x \\ 0 & 0 & a & x \\ l & a & a & x \end{bmatrix}$$

The second step results in

$$\begin{bmatrix} u & 0 & 0 & u \\ l & u & 0 & u \\ l & 0 & a & x \\ l & l & a & x \end{bmatrix}$$

The last step yields the final result

$$\begin{bmatrix} u & 0 & 0 & u \\ l & u & 0 & u \\ l & 0 & u & u \\ l & l & l & u \end{bmatrix}$$

This time the final result is sparse. Even better, we have no fill-in! The choice of pivot not only affects the numerical error, but also the fill-in created during LU decomposition. Just like in LU decomposition of dense matrices pivoting is generally performed for every LU decomposition step. To choose the optimal pivot in therms of fill-in a simulated LU decomposition run is performed where only the fill-in is considered and no computation takes place.

The pivot for one step of LU decomposition is selected by simulating the fill-ins of all pivot candidates in the first step of LU decomposition. The candidate with the smallest fill-in is then chosen and the corresponding rows and columns are exchanged. If two or more candidates produce the same fill-in the candidate with the largest absolute value in the initial matrix is chosen. Note that due to this fill-ins cannot be chosen for pivots. The fill-ins produced by the LU decomposition step are added to the sparse matrix. (i.e. entries for the new nonzero elements are added with no specific value set at this point).

The proposed approach has two weak spots: fill-ins are not considered as pivot candidates, and secondly, the magnitudes of values in the initial matrix are used for choosing pivots. LU decomposition steps can make some fill-in large (which obviously should make it the pivot, but it cannot be chosen for pivot). The magnitude of the chosen pivot can also be reduced by previous LU decomposition steps too much so that it no longer represents a feasible choice for a pivot.

Note how we didn't consider the magnitudes of pivot candidates (except when a tie in terms of fill-in takes place between two or more candidates). Consequently such a pivoting generally produces bad results in terms of numerical error. To avoid numerical error SPICE has two parameters that reduce the set of pivot candidates. These two parameters are relative pivot tolerance (pivrel) and absolute pivot tolerance (pivtol). The absolute value of a pivot candidate in i-th step of LU decomposition must be greater than pivtol and greater than pivrel times the largest subdiagonal element in the i-th column. The values of pivtol and pivrel are by default set to 10^{-12} and 10^{-3}, respectively.

What about matrix inversion?

Multiple linear systems of equations with a common coefficient matrix can also be solved by first inverting the matrix and then multiplying every right-hand side vector with the inverted matrix. This approach has a major disadvantage. If the matrix is sparse its inverse is usually not. The LU decomposition, however, can be sparse if the pivots are chosen in the right way.

Direct solvers and iterative solvers.

The approach we presented (LU decomposition with forward and backward substitution) is a member of the large family of approaches deemed as direct methods. The second large family of

methods comprises iterative methods. These methods repeatedly apply a (usually simple) algorithm to the system of equations. With each iteration a new approximate solution is obtained which is more accurate than the previous one. After a sufficient number of iterations the quality of the solution becomes sufficient and we can stop iterating.

Nonlinear elements and the Newton-Raphson algorithm

5th Lecture W2, November

When we introduce nonlinear elements we can no longer write equations in matrix form. Instead they are now written as a list of nonlinear equations. We take a look at the nonlinear models of selected semiconductor elements (diodes, transistors). If the equations are twice continuously differentiable we can numerically solve them with the Newton-Raphson algorithm. The algorithm iteratively approaches the solution by linearizing the equations and solving the resulting linear system to produce a new approximation to the solution of the original nonlinear system. The linearized system can again be constructed by means of the element footprints approach.

We discuss the stopping conditions for the Newton-Raphson algorithm. The algorithm can also fail to converge to a solution. Several approaches to finding a solution in case of non convergence are discussed. Strong nonlinearities in the element characteristics can also be the cause for non convergence. Approaches for dealing with such elements are presented.

« READ LESS

Introducing nonlinear elements

Until now all circuit elements were linear which made it possible for us to write down the system of linear equations directly from the circuit schematic using element footprints. When teh elements are nonlinear their constitutive relations become nonlinear equations. Take for instance a semiconductor diode. Its current (i_D) is expressed with the voltage across its terminals (u_D) by the following relation

$$i_\mathrm{D} = I_\mathrm{S}(\mathrm{e}^{u_\mathrm{D}/V_\mathrm{T}} - 1)$$

where I_S is the diode's saturation current and V{sub>T is the thermal voltage. Assuming all constitutive relations (with the exception of voltage sources) are in the form where device currents are expressed with branch voltages we can substitute them in the KCL equations (or add them to the system of equations if we are dealing with a voltage source). The system of equations becomes nonlinear. As such it can be formulated as

$$g_1(\mathbf{x}) = 0$$
$$\dots$$
$$g_n(\mathbf{x}) = 0$$

where $\mathbf{x}$ is the vector of unkowns an g_i is the i-th nonlinear function defining the i-th nonlinear equation. Often we use a shorthand notation by introducing a vector-valued function $\mathbf{g}$ which yields a vector with n components for every argument $\mathbf{x}$.

$$\mathbf{g}(\mathbf{x}) = \mathbf{0}$$

Solving such systems of equations can be done efficiently by means of Newton-Raphson algorithm.

The Newton-Raphson algorithm for 1-dimensional problems

We illustrate the Newton-Raphson algorithm on a n=1 dimensional example. Suppose we are trying to solve

$$\mathrm{e}^x = 2 - x$$

If we rewrite this equation as g(x)=0 we see that

$$g(x) = \mathrm{e}^x - 2 + x$$

The Newton-Raphson algorithm is an iterative one. With every iteration it improves the solution. Suppose we start with initial approximate solution $x^{(i)}$. Then the algorithm computes the new approximate solution $x^{(i+1)}$ as

$$x^{(i+1)} = x^{(i)} - \frac{g(x^{(i)})}{g'(x^{(i)})}$$

where g' is the derivative of g with respect to x. Suppose our initial approximate solution is x(0)=0. Then the following sequence of approximate solutions is produced by the algorithm:
0.500000000000000
0.443851671995364
0.442854703829747
0.442854401002417
0.442854401002389
...

We see the algorithm converges rapidly. In only 5 iterations the result stabilizes at 12 significant digits (i.e. differs in 13th digit between fourth and fifth iteration). The solution of the equation written with 15 significant digits is 0.442854401002389. We see the Newton-Raphson algorithm solved the equation to double precision (15 digits) in only 5 iterations.

What makes the Newton-Raphson algorithm so efficient? Mathematically it can be shown that the algorithm converges quadratically in the neighborhood of a solution. Quadratic convergence means that the error (i,.e. the difference between the approximate solution $x^{(i)}$ and the exact solution x^*) can be expressed as

$$|x^{(i+1)} - x^*| \leq M|x^{(i)} - x^*|^2$$

for i that is large enough. Roughly speaking this means that the number of exact digits doubles with every iteration of the algorithm as i becomes large enough. This is rue if the initial approximate solution is close o the exact solution of the problem. The algorithm can fail in several ways.

- If the derivative of g becomes zero (i.e. at a stationary point) the algorithm fails due to division by zero.

- The algorithm can be trapped in a cycle where the same approximate solutions are visited over and over again.
- The algorithm can fail to converge if the derivative of g is not continuous.
- If the solution of the equation is a multiple root (i.e. for $g(x)=(x-1)^2$ the solution $x=1$ has multiplicity 2) the algorithm converges slowly in the neighborhood of the solution.

For optimal performance the function g must be twice continuously differentiable, The initial approximate solution must be close to the real solution, and the first derivative of g must not be equal to zero in an interval containing the initial approximate solution $x^{(0)}$ and the exact solution.

When to stop?

The Newton-Raphson algorithm improves the approximate solution with every iteration. At some point the approximate solution becomes good enough. How do we know when to stop? Simulators usually stop the algorithm when the following condition is satisfied

$$|x^{(i+1)} - x^{(i)}| \leq e_r \max(|x^{(i+1)}|, |x^{(i)}|) + e_a$$

Here e_r and e_a are the relative and the absolute tolerance, respectively. The stopping criterion is based on the assumption that when two consecutive approximate solutions are close enough to each other they are also close enough to the exact solution. In SPICE the relative tolerance (reltol simulator parameter) is 10^{-3}. The absolute tolerance depends on the type of the unknown. If the unknown is a voltage 10^{-6} is used (specified by the vntol simulator parameter). For currents the absolute tolerance is 10^{-12} (specified by the abstol simulator parameter).

Generalizing the algorithm for n>1

To help us understand the algorithm for higher dimensional problems, let us find a geometric interpretation for the Newton-Raphson formula by first rewriting it as

$$g(x^{(i)}) + g'(x^{(i)})(x^{(i+1)} - x^{(i)}) = 0$$

The left-hand side is a linear function of $x^{(i+1)}$. In fact it is the linearization of g(x) in the neighborhood of $x^{(i)}$. The whole equation requires this linearization to be zero. When the linearization is performed close to the exact solution then it is almost equal to g(x) and solving the linearized equation produces a good approximation to the exact solution.

Now what is different when we have n unkowns? We can linearize n equations by computing the derivatives of the n left-hand sides. Let k denote the index of an unknown. One equation ($g(\mathbf{x})=0$) is linearized as

$$g(\mathbf{x^{(i)}}) + \sum_{k=1}^{n} \left.\frac{\partial g}{\partial x_k}\right|_{x=x^{(i)}} (x_k^{(i+1)} - x_k^{(i)}) = 0$$

The linearized equation defines a plane in n-dimensional space. Therefore linearizing n nonlinear equations gives us n planes in n-dimensional space. The intersection of these n planes is the new approximate solution $\mathbf{x}^{(i+1)}$. The intersection of n planes can be obtained by solving the corresponding system of linear equations (i.e. obtained by linearizing the nonlinear system of equations at the previous approximate solution). We already know how to do this (by means of Gaussian elimination or LU decomposition, forward, and backward substitution ...).

We can write the linearized system of equations in matrix form as

$$\mathbf{g}(\mathbf{x}^{(i)}) + \mathbf{G}(\mathbf{x}^{(i)})(\mathbf{x}^{(i+1)} - \mathbf{x}^{(i)}) = 0$$

where matrix $\mathbf{G}$ is the Jacobian of the system at $\mathbf{x}^{(i)}$ and is defined as

$$\mathbf{G}(\mathbf{x}^{(i)}) = \begin{bmatrix} \frac{\partial g_1}{\partial x_1} & \frac{\partial g_1}{\partial x_2} & \cdots & \frac{\partial g_1}{\partial x_n} \\ \frac{\partial g_2}{\partial x_1} & \frac{\partial g_2}{\partial x_2} & \cdots & \frac{\partial g_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial g_n}{\partial x_1} & \frac{\partial g_n}{\partial x_2} & \cdots & \frac{\partial g_n}{\partial x_n} \end{bmatrix}_{\mathbf{x}=\mathbf{x}^{(i)}}$$

The Newton-Raphson algorithm solves the following linear system to obtain the next approximate solution $\mathbf{x}^{(i+1)}$

$$\mathbf{G}(\mathbf{x}^{(i)})\mathbf{x}^{(i+1)} = \mathbf{G}(\mathbf{x}^{(\mathbf{i})})\mathbf{x}^{(\mathbf{i})} - \mathbf{g}(\mathbf{x}^{(\mathbf{i})})$$

The first term corresponds to the linearized system of equations, but does not include the constant term of the linearization. The second term is the value obtained from the nonlinear system of equations.

The stopping condition is applied to every component of $\mathbf{x}$ independently. The algorithm stops when all components satisfy the stopping condition. In SPICE OPUS the stopping condition is slightly more elaborate. Three requirements must be met in order for the Newton-Raphson algorithm to stop.

- The difference between candidate solutions from iterations i+1 and i must be within the given relative and absolute tolerances,
- The difference between candidate solutions from iterations i and i-1 must be within the given relative and absolute tolerances, and
- The last three approximate solutions must form a triangle in the n-dimensional space where the angle at the second point is smaller than 90°.

The last two requirements prevent the algorithm from stopping when it is in fact oscillating around a solution. Because this last check can slow down convergence users of SPICE OPUS can turn it off by setting the noconviter simulator parameter.

Usually the initial approximate solution is a vector of all zeros. In SPICE one can override this default initial approximate solution with the use of the .nodeset netlist directive.

In practice the algorithm has a limited number of iterations for satisfying the stopping condition. This number is set by the itl1 parameter in SPICE (100 by default). If the algorithm fails to satisfy the stopping condition after itl1 iterations the analysis is considered as failed and an error is reported.

Element footprints revisited

Let us illustrate the construction of the linearized system of equations that are used in one iteration of the Newton-Raphson algorithm. Let us start with a simple example: a semiconductor diode (Fig. 1).

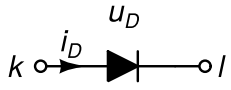


Fig. 1: A semiconductor diode connected between nodes k and l.

The constitutive relation of a semiconductor diode expresses the diode current as a nonlinear

function of the diode voltage.

$$i_D = I_\mathrm{S}(\mathrm{e}^{u_D/V_\mathrm{T}} - 1)$$

The diode contributes its current to the KCL equations of nodes k and l (i.e. $g_k(\mathbf{x})=0$ and $g_l(\mathbf{x})=0$).

$$g_k(\mathbf{x}) = \cdots + i_D + \cdots$$
$$g_l(\mathbf{x}) = \cdots - i_D + \cdots$$

The diode branch voltage can be expressed with node potentials as

$$u_D = v_k - v_l$$

To obtain the diode's contribution to the Jacobian matrix we must compute the derivative of the diode current with respect to the node potentials. First, let us compute the derivative with respect to the diode voltage.

$$\frac{\partial i_D}{\partial u_D} = \frac{I_\mathrm{S}}{V_\mathrm{T}}\mathrm{e}^{u_D/V_\mathrm{T}} = g_D$$

Here g_D denotes the differential conductance of the diode (not to be mistaken with nonlinear functions g_k and g_l which originate from the KCL). The diode's contribution depends on two unknowns: v_k and v_l. The derivatives of the diode current with respect to these two unknowns are

$$\frac{\partial i_D}{\partial v_k} = \frac{\partial i_D}{\partial u_D}\frac{\partial u_D}{\partial v_k} = g_D$$
$$\frac{\partial i_D}{\partial v_l} = \frac{\partial i_D}{\partial u_D}\frac{\partial u_D}{\partial v_l} = -g_D$$

When we linearize the two nonlinear KCL equations for nodes k and l the differential conductance of the diode is added to the Jacobian of the system into rows k and l (corresponding to the two KCL equations) and columns k and l (corresponding to node potentials of nodes k and l). Let us assume for now there are no voltage sources in the circuit so we don't have to apply MNA. Note that rows of the Jacobian correspond to KCL equations and columns correspond to unknowns (node potentials). Let i denote the iteration of the Newton-Raphson algorithm. The element footprint of a semiconductor diode in the Jacobian matrix is then

$$\begin{array}{c|ccccccc}
 & v_1 & \cdots & v_k & \cdots & v_l & \cdots & v_{n-1} \\
\hline
\mathrm{KCL}_1 & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot \\
\vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\
\mathrm{KCL}_k & \cdot & \cdots & +g_D^{(i)} & \cdots & -g_D^{(i)} & \cdots & \cdot \\
\vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\
\mathrm{KCL}_l & \cdot & \cdots & -g_D^{(i)} & \cdots & +g_D^{(i)} & \cdots & \cdot \\
\vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\
\mathrm{KCL}_{n-1} & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot
\end{array}$$

Note that $g_D^{(i)}$ is computed from $v_k^{(i)}$ and $v_l^{(i)}$. What bout the right-hand side of the system of linearized equations? A diode will contribute to the k-th and l-th row of the RHS vector. The contributions of the diode to the k-th and l-th nonlinear equation are

$$[\mathbf{g}(\mathbf{x^{(i)}})]_k = -[\mathbf{g}(\mathbf{x^{(i)}})]_l = I_\mathrm{S}(\mathrm{e}^{(v_k^{(i)}-v_l^{(i)})/V_\mathrm{T}} - 1)$$

For the diode's contribution in the k-th row we get

$$[\mathbf{G}(\mathbf{x^{(i)}})\mathbf{x^{(i)}} - \mathbf{g}(\mathbf{x^{(i)}})]_k = g_{\mathrm{D}}^{(i)}(v_k^{(i)} - v_l^{(i)}) - I_{\mathrm{S}}(\mathrm{e}^{(v_k^{(i)} - v_l^{(i)})/V_{\mathrm{T}}} - 1)$$

The first term comes from the linearized system of equations and the second one is the contribution to the nonlinear system of equations. The contribution to row l of the RHS vector is

$$[\mathbf{G}(\mathbf{x^{(i)}})\mathbf{x^{(i)}} - \mathbf{g}(\mathbf{x^{(i)}})]_l = -g_{\mathrm{D}}^{(i)}(v_k^{(i)} - v_l^{(i)}) + I_{\mathrm{S}}(\mathrm{e}^{(v_k^{(i)} - v_l^{(i)})/V_{\mathrm{T}}} - 1)$$

Now we can revisit the linear resistor connected to nodes k and l, but this time we treat it like a nonlinear element with constitutive relation

$$i = R^{-1}u$$

We see that the differential conductance is R^{-1} and does not depend on the current approximate solution. The contribution to the Jacobian matrix is therefore the same as the contribution to the coefficient matrix of a linear circuit and does not change between iterations of the Newton-Raphson algorithm. The contributions to the k-th row of the RHS vector is

$$[\mathbf{G}(\mathbf{x^{(i)}})\mathbf{x^{(i)}} - \mathbf{g}(\mathbf{x^{(i)}})]_k = R^{-1}(v_k^{(i)} - v_l^{(i)}) - i^{(i)} = R^{-1}(v_k^{(i)} - v_l^{(i)}) - R^{-1}(v_k^{(i)} - v_l^{(i)}) = 0$$

Similarly, the contribution to the l-th row of the RHS vector is also 0. This is due to the linearized contribution of a linear resistor being identical to the "nonlinear" contribution (the two terms cancel each other out). We see that the element footprint of a linear resistor does not change between iterations of the Newton-Raphson algorithm. In fact linear elements contribute to the Jacobian matrix in the same way as they do to the coefficient matrix of a linear circuit.

From the two examples we see that the Jacobian matrix of a nonlinear circuit in one iteration of the Newton-Raphson algorithm has the same role as the coefficient matrix of a linear circuit. The major difference between solving linear and nonlinear circuits is that the former requires solving only one linear system of equations while the latter requires solving multiple systems of linear equations, where every linear system corresponds to the linearized circuit in one iteration of the Newton-Raphson algorithm.

Element footprint of a nonlinear element with multiple pins

We are going to illustrate the construction of an element footprint for elements with multiple pins on a NMOS transistor operating in saturation region (Fig. 2). MOS transistors are 4-pin elements. To keep things simple we assume the bulk pin is connected to the source pin.

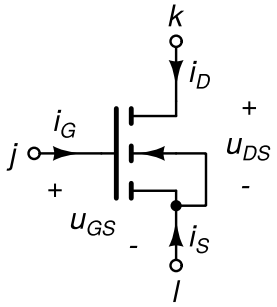


Fig. 2: A NMOS transistor.

A NMOS transistor is operating in the saturation region when the following two conditions are satisfied.

$$u_{GS} \geq U_T$$
$$u_{DS} \geq u_{GS} - U_T$$

Where U_T is the threshold voltage of the NMOS transistor. The currents flowing into the pins in the saturation region are given by

$$i_G = 0$$
$$i_D = K(u_{GS} - U_T)^2(1 + \lambda u_{DS})$$
$$i_S = -i_G - i_D$$

We can express branch voltages u_{GS} and u_{DS} with node potentials as

$$u_{GS} = v_j - v_l$$
$$u_{DS} = v_k - v_l$$

To construct the element footprint in the Jacobiam matrix we first compute the partial derivatives of currents with respect to branch potentials.

$$\frac{\partial i_G}{\partial u_{GS}} = \frac{\partial i_G}{\partial u_{DS}} = 0$$
$$\frac{\partial i_D}{\partial u_{GS}} = g_{21} = 2K(u_{GS} - U_T)(1 + \lambda u_{DS})$$
$$\frac{\partial i_D}{\partial u_{DS}} = g_{22} = K\lambda(u_{GS} - U_T)^2$$

next, we express the partial derivatives of branch currents with respect to node potentials. There are 9 partial derivatives we need to compute. These derivatives can be expressed with g_{21} and g_{22}.

$$\frac{\partial i_G}{\partial v_j} = \frac{\partial i_G}{\partial v_k} = \frac{\partial i_G}{\partial v_l} = 0$$
$$\frac{\partial i_D}{\partial v_j} = \frac{\partial i_D}{\partial u_{GS}}\frac{\partial u_{GS}}{\partial v_j} + \frac{\partial i_D}{\partial u_{DS}}\frac{\partial u_{DS}}{\partial v_j} = g_{21}$$
$$\frac{\partial i_D}{\partial v_k} = \frac{\partial i_D}{\partial u_{GS}}\frac{\partial u_{GS}}{\partial v_k} + \frac{\partial i_D}{\partial u_{DS}}\frac{\partial u_{DS}}{\partial v_k} = g_{22}$$
$$\frac{\partial i_D}{\partial v_l} = \frac{\partial i_D}{\partial u_{GS}}\frac{\partial u_{GS}}{\partial v_l} + \frac{\partial i_D}{\partial u_{DS}}\frac{\partial u_{DS}}{\partial v_l} = -g_{21} - g_{22}$$
$$\frac{\partial i_S}{\partial v_j} = -\frac{\partial i_G}{\partial v_j} - \frac{\partial i_D}{\partial v_j} = -g_{21}$$
$$\frac{\partial i_S}{\partial v_k} = -\frac{\partial i_G}{\partial v_k} - \frac{\partial i_D}{\partial v_k} = -g_{22}$$
$$\frac{\partial i_S}{\partial v_l} = -\frac{\partial i_G}{\partial v_l} - \frac{\partial i_D}{\partial v_l} = g_{21} + g_{22}$$

Now we can construct the element footprint in the Jacobian matrix. A NMOS transistor contributes to KCL equations of nodes k and l in columns corresponding to node potentials v_j, v_k, and v_l. There is no contribution to the KCL equation of node j because i_G=0.

$$\begin{array}{c|ccccccccc} & v_1 & \cdots & v_j & \cdots & v_k & \cdots & v_l & \cdots & v_{n-1} \\ \hline \mathrm{KCL}_1 & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \mathrm{KCL}_k & \cdot & \cdots & g_{21}^{(i)} & \cdots & g_{22}^{(i)} & \cdots & -g_{21}^{(i)} - g_{22}^{(i)} & \cdots & \cdot \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \mathrm{KCL}_l & \cdot & \cdots & -g_{21}^{(i)} & \cdots & -g_{22}^{(i)} & \cdots & g_{21}^{(i)} + g_{22}^{(i)} & \cdots & \cdot \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \mathrm{KCL}_{n-1} & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot \end{array}$$

A NMOS does not contribute to the j-th row of the RHS vector due to i_G=0. The contribution to the k-th row is

$$g_{21}^{(i)} v_j^{(i)} + g_{22}^{(i)} v_k^{(i)} - (g_{21}^{(i)} + g_{22}^{(i)}) v_l^{(i)} - K((v_j^{(i)} - v_l^{(i)}) - U_T)^2 (1 + \lambda (v_k^{(i)} - v_l^{(i)}))$$

The contribution to the l-th row of the RHS vector is

$$-g_{21}^{(i)} v_j^{(i)} - g_{22}^{(i)} v_k^{(i)} + (g_{21}^{(i)} + g_{22}^{(i)}) v_l^{(i)} + K((v_j^{(i)} - v_l^{(i)}) - U_T)^2 (1 + \lambda (v_k^{(i)} - v_l^{(i)}))$$

Convergence problems and how to solve them

The Newton-Raphson algorthm can exhibit convergence problems (slow convergence or even no convergence), particularly for circuits with strong nonlinearities. Simulators use various tricks to achieve convergence.

Junction voltage limiting. p-n junctions in diodes and transistors exhibit an exponential i(u) characteristic. This can cause very large values of currents (ans consequently large values of the nonlinear left-hand side g(x) in the KCL equations). The values can even exceed the maximum value allowed by double floating point precision. Once that happens IEEE floating point infinite values or even NaN (not a number) values can occur in the candidate solution. Especially NaN values spread like a "virus". Any (binary or unary) operation performed on a NaN value results in a NaN so the NaNs quickly spread across the whole solution vector and make the result completely useless.

To avoid this the independent variable in the exponential function (the branch voltages across p-n junctions) are limited to interval [-voltagelimit, voltagelimit] where voltagelimit is a simulator parameter (10^{30}V by default). If the branch voltage is smaller than -voltagelimit it is truncated to -voltagelimit. Similarly, if the branch voltage is greater than voltagelimit it is truncated to voltagelimit. The obtained value is then used for computing the p-n junction current and its derivative with resect to the node potentials. This procedure not only helps avoid infinite and NaN values, but also speeds up the convergence of the Newton-Raphson algorithm.

Damped Newton-Raphson algorithm. The Newton-Raphson algorithm can produce an oscillating sequence of candidate solutions. This behavior can be eliminated to great extent if the step taken by the algorithm is shortened. To simplify the presentation of this approach we assume a 1-dimensional problem (n=1). Let e_r and e_a denote the relative and the absolute tolerance used in the stopping condition. Let $x^{(i)}$ and $x^{(i+1)}$ denote the previous and the new approximate solution. We define the step tolerance as

$$\epsilon = e_r \max(|x^{(i+1)}|, |x^{(i)}|) + e_a$$

After every Newton-Raphson iteration the stopping condition is checked. If the condition is not satisfied the step is truncated according to the following formula

$$x_{\text{truncated}}^{(i+1)} = \begin{cases} x^{(i+1)} & x^{(i+1)} \in [x^{(i)} - \epsilon/s, x^{(i)} + \epsilon/s] \\ x^{(i)} - \epsilon/s & x^{(i+1)} < x^{(i)} - \epsilon/s \\ x^{(i)} + \epsilon/s & x^{(i+1)} > x^{(i)} + \epsilon/s \end{cases}$$

In the next Newton-Raphson iteration the truncated solution is used as the previous solution. Parameter s is the truncation factor specified by the sollim simulator parameter in SPICE OPUS (10 by default). The truncated algorithm makes a slow but sure progress. Due to this slow progress it often runs out of available iterations. Therefore the iteration limit in SPICE OPUS is increased to itl1 * sollimiter. The value of the sollimiter simulator parameter is 10 by default.

The damped Newton-Raphson algorithm is used only when the simulator detects convergence problems. By default the original Newton-Raphson algorithm is used.

Adding shunt resistors to the circuit. If we connect resistors from every node to the ground we effectively add diagonal entries equal to the inverse of the added resistance to the Jacobian. Therefore if the resistance is small enough the diagonal part begins to dominate the Jacobian. In practice many convergence problems are reduced if resistors with sufficiently small resistance (shunts) are added. If the resistance is not too small shunt resistors do not significantly alter the circuit's behavior. By default shunting is turned off. Shunting is turned on by specifying the shunt resistance with the rshunt simulator parameter.

Homotopy-based approaches

- GMIN stepping - source stepping - source lifting and cmin stepping

Operating point analysis and DC sweep

With the knowledge we gained up to this point we can handle circuits with arbitrary linear and nonlinear resistive elements. The main property of resistive elements is that we can express the current flowing into the pins of the element (voltage) as a (non)linear function of the node potentials corresponding to nodes to which the element is connected. The function may not contain any derivatives or integrals with respect to time. The derivatives and integrals with respect to time are required for describing reactive elements (like capacitors and inductors), but we haven't reached that point, yet.

Now let us assume we describe our reactive elements by means of derivatives with respect to time. We can always do that (i.e. convert an integral into a derivative) by choosing an appropriate independent variable in the formulation of the element's constitutive relation. Now suppose all derivative terms are equal to zero. This is the case when the voltages and the currents in the circuit no longer change. For stable circuits excited only by DC voltage and current sources we reach this state if we wait for a sufficient amount of time. We refer to this state as the circuit's operating point. For computing the circuit's operating point we need to consider only the resistive elements in the circuit. Therefore the above described algorithm can be used for finding the operating point of the circuit.

Often we are interested in how the operating point of a circuit changes if we change the DC value of the circuit's excitation voltages/currents. Such analysis is also referred to as the DC operating point sweep or simply DC analysis. A DC analysis is much faster than the equivalent sequence of operating point analyses because the solution of the last sweep point is used as the initial iterate in

the Newton-Raphson algorithm solving the next sweep point. This provides a good initial guess for the Newton-Raphson algorithm which consequently requires only a few iterations to satisfy the stopping condition. Therefore SPICE provides a separate simulator parameter (itl2) for setting the limit on the number of Newton-Raphson iterations available for solving one point in a DC sweep. By default its value is set to 50.

DC small signal analysis

6th Lecture W3, November

One can interpret the elements of the coefficient matrix as conductances, resistances, and controlled sources. This interpretation results in the linearized circuit model. The linearized circuit model can be used for obtaining small signal properties of the circuit like gain, input impedance, and output impedance. If the signals are composed of a large DC component and a small perturbation we can treat the circuit as linear if we consider only the perturbations. We draw parallels between linear electronics and small signal DC analysis. We also relate small signal DC analysis results to the results of operating point sweeps.

« READ LESS

Linearized circuit

Suppose we are interested in how much a circuit's operating point will change if we slightly change the circuit's excitation. In previous chapter we formulated the nonlinear equations describing the circuit. Now let us split the nonlinear function in two parts. The first part is a function of $\mathbf{x}$ while the second part is independent of $\mathbf{x}$ and thus represents the independent sources in the circuit.

$$\mathbf{f}(\mathbf{x}) = \mathbf{g}(\mathbf{x}) - \mathbf{y}$$

Here $\mathbf{g}$ is a vector-valued function and $\mathbf{y}$ is the right-hand side of the nonlinear equations. The system of equations can now be written as

$$\mathbf{g}(\mathbf{x}) = \mathbf{y}$$

A small perturbation of $\mathbf{y}$ results in a small perturbation of $\mathbf{g}$ which can be formulated as

$$\mathbf{g}(\mathbf{x} + \Delta\mathbf{x}) = \mathbf{y} + \Delta\mathbf{y}$$

By taking the first two terms of the Taylor series for $\mathbf{g}$ and neglecting the rest we can write the system of equations as

$$\mathbf{g}(\mathbf{x}) + \mathbf{G}(\mathbf{x})\Delta\mathbf{x} = \mathbf{y} + \Delta\mathbf{y}$$

where $\mathbf{G}$ is the Jacobian (i.e. the matrix of first derivatives) of $\mathbf{g}$. Because $\mathbf{g}$ and $\mathbf{f}$ differ only by a

constant term it is also equal to the Jacobian of the circuit equations ($\mathbf{F}$). The Jacobian depends on the solution of the unperturbed circuit ($\mathbf{x}$). Note that we can neglect higher order terms in the Taylor series because we assume the perturbation is small. Because the intial circuit equation (the one without perturbations) must still be satisfied the first and the second term cancel each other out and we are left with

$$\mathbf{G}(\mathbf{x})\Delta\mathbf{x} = \Delta\mathbf{y}$$

We obtained a linear system of equations which corresponds to some linear circuit. By solving it we can express the perturbation of the circuit's operating point with the perturbation of the circuit's excitation. The process of formulating this system of linear equations is also referred to as circuit linearization.

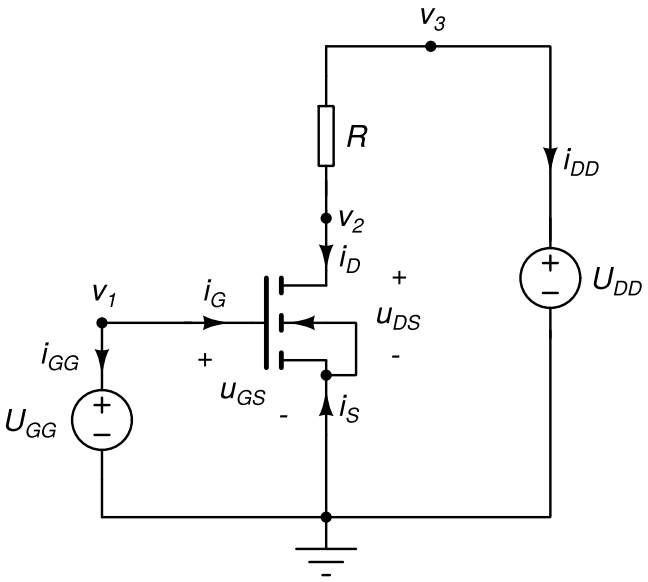


Fig. 1: A simple circuit with an MOS transistor.

For instance, take a simple circuit with a NMOS transistor depicted in Fig. 1. Let us write the equations of this circuit and then lninearize them. The circuit has 4 nodes and 2 voltage sources. The list of unknowns includes the node potentials of nodes 1, 2, and 3, and the currents flowing into the two independent voltage sources. We assume the MOS transistor is operating in the saturation region. Its currents can be expressed as

$$\begin{aligned} i_G &= 0 \\ i_D &= K(u_{GS} - U_T)^2(1 + \lambda u_{DS}) \\ i_S &= -i_G - i_D \end{aligned}$$

The three KCL equations and the two constitutive equations of the independent voltage sources are

$$\begin{aligned} i_{GG} &= 0 \\ K(v_1 - U_T)^2(1 + \lambda v_2) + R^{-1}(v_2 - v_3) &= 0 \\ R^{-1}(v_3 - v_2) + i_{DD} &= 0 \\ v_1 &= U_{GG} \\ v_3 &= U_{DD} \end{aligned}$$

After solving this system of equations we obtain the operating point of the circuit (specified by V_1,

V_2, V_3, I_{GG}, and I_{DD}). The operating point depends on the values of the two independent voltage sources V_{GG} and V_{DD}. Now suppose we perturb the first voltage source by ΔU_{GG} while the second one is left unperturbed (ΔU_{DD}=0). The operating point of the circuit changes to V_1+ΔV_1, V_2+ΔV_2, V_3+ΔV_3, I_{GG}+ΔI_{GG}, and I_{DD}+ΔI_{DD}. We can compute the perturbations of the solution (ΔV_1, ΔV_2, ΔV_3, ΔI_{GG}, and ΔI_{DD}) from the linearized system of equations

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ g_{11} & g_{22}+R^{-1} & R^{-1} & 1 & 0 \\ 0 & -R^{-1} & R^{-1} & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta V_1 \\ \Delta V_2 \\ \Delta V_3 \\ \Delta I_{GG} \\ \Delta I_{DD} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \Delta U_{GG} \\ 0 \end{bmatrix}$$

Where g_{21} and g_{22} are expressed as (see an example in the previous lecture)

$$g_{21} = \left.\frac{\partial i_D}{\partial u_{GS}}\right|_{u_{GS}=V_1, u_{DS}=V_2} = 2K(V_1 - U_T)(1+\lambda V_2)$$

$$g_{22} = \left.\frac{\partial i_D}{\partial u_{DS}}\right|_{u_{GS}=V_1, u_{DS}=V_2} = K\lambda(V_1 - U_T)^2$$

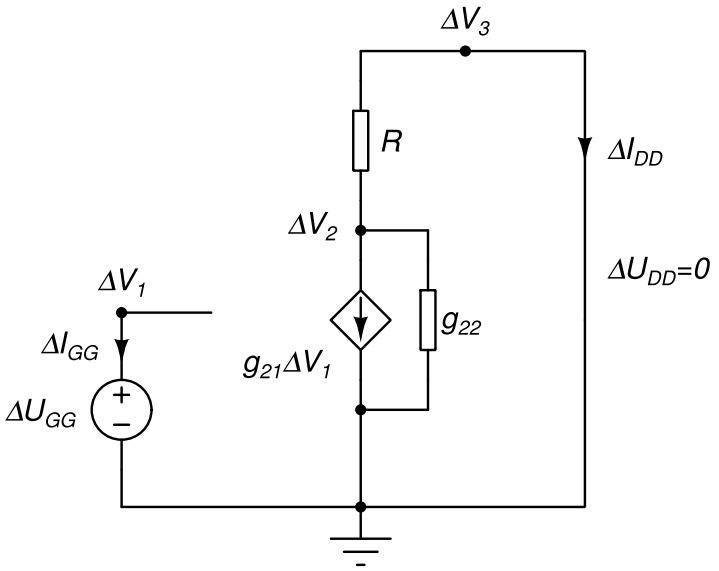


Fig. 2: Linearized circuit obtained from the circuit in Fig. 1.

From the obtained linear system of equations we can reconstruct a linear circuit depicted in Fig. 2. This circuit is also referred to as the linearized circuit or small signal model of the circuit. Conductance g_{22} and transconductance g_{21} represent the linearized model of the NMOS transistor. Linearized models of linear elements are identical to respective linear elements (e.g. resistor in Fig. 2). Independent sources that are not perturbed are set to zero (turned off) in the linearized circuit (indepenmdent current sources are removed, independent voltage sources are replaced with short circuits). The value of perturbed independent sources in the linearized circuit is equal to the respective perturbation (e.g. see U_{GG} in Fig. 2).

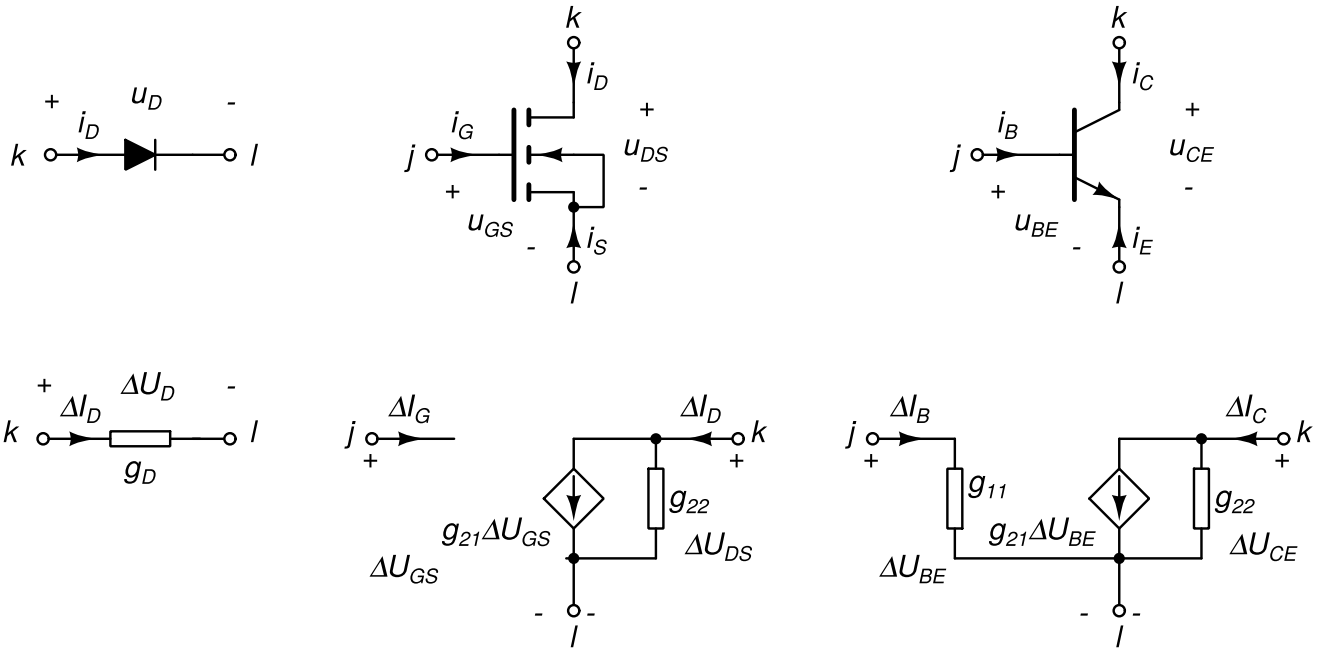


Fig. 2: Linearized models (bottom) of nonlinear elements (top) for diode (left), NMOS transistor (center), and NPN bipolar transistor (right).

Linearized models of nonlinear elements can be built in advance. The linearized circuit can be obtained by replacing nonlinear elements with their linearized models and by handling independent sources as mentioned in the previous paragrah. Fig. 3 depicts the linearized model of a diode, a NMOS transistor, and a NPN bipolar junction transistor. Parameter g_D of the linearized model of a diode at operating point U_D can be computed as

$$g_D = \left.\frac{\partial i_D}{\partial u_D}\right|_{u_D=U_D}$$

Parameters of the linearized model of a NMOS transistor are computed at operating point defined by U_{GS} and U_{DS} as

$$g_{21} = \left.\frac{\partial i_D}{\partial u_{GS}}\right|_{u_{GS}=U_{GS},u_{DS}=U_{DS}}$$
$$g_{22} = \left.\frac{\partial i_D}{\partial u_{DS}}\right|_{u_{GS}=U_{GS},u_{DS}=U_{DS}}$$

Parameters of the linearized model of a NPN bipolar transistor are computed at operating point defined by U_{BE} and U_{CE} as

$$g_{11} = \left.\frac{\partial i_B}{\partial u_{BE}}\right|_{u_{BE}=U_{BE},u_{CE}=U_{CE}}$$
$$g_{21} = \left.\frac{\partial i_C}{\partial u_{BE}}\right|_{u_{BE}=U_{BE},u_{CE}=U_{CE}}$$
$$g_{22} = \left.\frac{\partial i_C}{\partial u_{CE}}\right|_{u_{BE}=U_{BE},u_{CE}=U_{CE}}$$

Analysis of a linearized circuit reveals many important characteristics like current and voltage gain, input and output impedances, transconductance, etc. Take for instance the example in Fig. 2 which is in fact an amplifier. The input signal is generated by U_{GG} and the output signal is observed as the

node potential V_2. If we compute ΔV_2 and from there the quotient $\Delta V_2/\Delta U_{GG}$ we obtain the gain of the amplifier for small signals (i.e. small perturbations of the operating point).

DC small signal analysis in SPICE OPUS

In SPICE OPUS the DC small signal analysis is deemed TF analysis. One has to specify two things at its invocation: the output signal (e.g. node potential, voltage between two nodes, or current of a voltage source) and the independent source (voltage or current) that provides the small signal perturbation. The analysis then computes three things. The small signal gain (also referred to as the transfer function) whose type depends on the choice of the input and output. There are four possibilities: voltage gain, current gain, transconductance, and transimpedance. The analysis also computed input impedance at the nodes of the circuit where the independent source providing the excitation is located and the output impedance at the circuit's output. The latter is computed correctly only if the output is a node potential or a voltage between two nodes.

Most of the time in TF analysis is spent for computing the circuit's operating point. Once it is computed the Jacobian matrix describing the linearized circuit and its LU decomposition are already available. The simulator only constructs a right-hand side vector according to the specified input excitation and performs a forward and backward substitution to obtain the solution. From this solution the transfer function and teh input impedance are computed. For computing teh output impedance the simulatro constructs another right-hand side vector and performs another forward+backeard substitution. The output impedance is computed from the obtained result.

Small-signal analysis in the frequency domain

7th Lecture W1, December

We introduce the modelling of linear reactive elements (linear capacitors, inductors, and coupled inductors). We extend the notion of small-signal analysis to sinusoidal signals represented by complex numbers. The absolute value of such representation corresponds to the magnitude of the sinusoidal signal while the argument corresponds to its phase. We assume all signals in the circuit share the same frequency. Due to reactive elements the solution of the circuit depends on this frequency.

We extend linear reactive elements to nonlinear ones. We demonstrate the modelling of nonlinear capacitances on an example - semiconductor diode. Finally, we show how nonlinear elements are handled in small-signal frequency-domain analysis.

« READ LESS

Complex representation of sinusoidal signals

In circuit analysis we are often interested in the circuit's response when all excitations are

sinusoidal signals of the same frequency. Beside sinusoidal excitations we also allow DC excitations. The circuit's response to such stimulus is composed of a transient response and a periodic signal. In stable circuits the transient response eventually dies off and we are left with a periodic signal superimposed on a DC component. If the magnitudes of the sinusoidal excitation signals are small the circuit's response (excluding the DC component) is sinusoidal even if the circuit is nonlinear. We refer to the sinusoidal part of this response as small-signal sinusoidal response. The magnitudes and phases of the response signals provide us with many valuable insights into circuit's behavior.

In unstable circuits we cannot observe the small-signal sinusoidal response because the transient response never dies off. We can, however, compute it with a simulator. Again, the computed response can provide many valuable insights into circuit's behavior (like frequency response, stability, etc.).

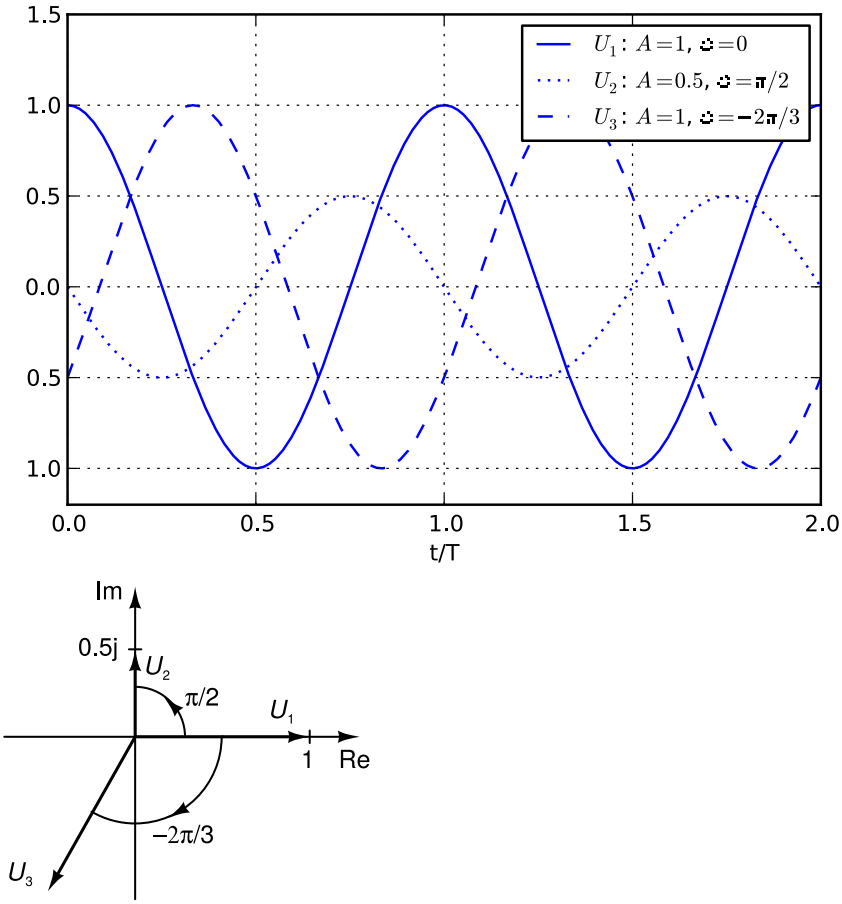


Fig. 1: Three sinusoidal signals in time domain (left) and their complex representations (right).

Let us assume a sinusoidal signal with magnitude A and phase **Φ**.

$$x(t) = A\cos(\omega t + \phi)$$

We can express it in terms of complex numbers as

$$x(t) = A\cos(\omega t + \phi) = \mathrm{Re}\left(A\mathrm{e}^{\mathrm{j}\phi}\mathrm{e}^{\mathrm{j}\omega}\right) = \mathrm{Re}\left(X\mathrm{e}^{\mathrm{j}\omega}\right)$$

where j denotes the imaginary unit. We can see that, assuming the frequency **ω** is known, the signal is uniquely defined by complex number X. The absolute value of X is equal to the magnitude of the signal while the argument of X is equal to the phase of the signal. Fig. 1 depicts 3 sinusoidal signals and the corres[ponding complex numbers as vectors in a complex plane. We refer to complex numbers that represent sinusoidal signals as complexors.

Capacitors and inductors in the frequency domain

By assuming all unknowns are sinusoidal signals of same frequency we effectively moved our analysis to the frequency domain. Unknowns become complex numbers representing magnitudes and phases of sinusoidal signals. Now what do we gain by doing this? Take for instance the constitutive relation of a linear capacitor (Fig. 2, left).

Fig. 2: Linear capacitor (left) and inductor (right).

$$i = C\frac{du}{dt}$$

After applying the Fourier transformation, which transposes us into frequency domain, this relation changes into

$$I = \mathrm{j}\omega CU$$

where I and U are complexors representing the current and the voltage of a capacitor. Similarly, for an inductor (Fig. 2, right) with constitutive relation

$$u = L\frac{di}{dt}$$

we have

$$U = \mathrm{j}\omega LI$$

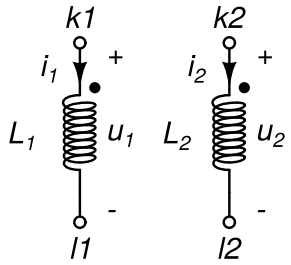


Fig. 3: Coupled inductors. When a current is flowing into the pin marked with a dot it is considered positive.

Another element to consider is the coupling between two inductors. Suppose inductors L_1 and L_2 are coupled with a coupling factor k<1. The constitutive relations of the two inductors are extended with an additional term representing the magnetic coupling.

$$u_1 = L_1\frac{di_1}{dt} + M_{12}\frac{di_2}{dt}$$
$$u_2 = M_{12}\frac{di_1}{dt} + L_2\frac{di_2}{dt}$$

where

$$M_{12} = k\sqrt{L_1 L_2}$$

In the frequency domain the constitutive relations become

$$U_1 = \mathrm{j}\omega L_1 I_1 + \mathrm{j}\omega M_{12} I_2$$
$$U_2 = \mathrm{j}\omega M_{12} I_1 + \mathrm{j}\omega L_2 I_2$$

Frequency-domain analysis of linear circuits

Assuming all signals in the circuit are sinusoidal greatly simplifies circuit analysis. The constitutive relations of capacitors and inductors become algebraic equations. Instead of solving a system of differential equations we are confronted with a system of linear equations. The unknowns and the coefficients are now complex. Capacitors and inductors are described in a manner equivalent to ordinary resistors, i.e. the current is proportional to the voltage. The coefficient of proportionality is complex and depends on the frequency (**ω**). With this in mind we can immediately derive the element footprint of a capacitor (Fig. 2, left) in the coefficient matrix.

$$\begin{array}{c|ccccccc} & v_1 & \cdots & v_k & \cdots & v_l & \cdots & v_{n-1} \\ \hline \mathrm{KCL}_1 & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \mathrm{KCL}_k & \cdot & \cdots & +\mathrm{j}\omega C & \cdots & -\mathrm{j}\omega C & \cdots & \cdot \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \mathrm{KCL}_l & \cdot & \cdots & -\mathrm{j}\omega C & \cdots & +\mathrm{j}\omega C & \cdots & \cdot \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \mathrm{KCL}_{n-1} & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot \end{array}$$

Handling of inductors is somewhat more complicated. For a single inductor we could easily express the current with the voltage. But for coupled inductors this would require inverting a matrix. Furthermore, expressing the current explicitly with voltage in time-domain would require solving a system of differential equations. Therefore most simulators choose to take a different path. Instead of explicitly expressing the device current the introduce a new unknown for every inductor - its current. This way inductors are handled in a manner similar to voltage sources. The constitutive relation of an inductor becomes the additional equation that makes the system fully determined.

$$V_k - V_l - \mathrm{j}\omega L I = 0$$

The element footprint of an inductor (Fig. 2, right) is

$$\begin{array}{c|ccccccccc} & v_1 & \cdots & v_k & \cdots & v_l & \cdots & v_{n-1} & \cdots & i & \cdots \\ \hline \mathrm{KCL}_1 & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots \\ \mathrm{KCL}_k & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & 1 & \cdots \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots \\ \mathrm{KCL}_l & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & -1 & \cdots \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots \\ \mathrm{KCL}_{n-1} & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots \\ \mathrm{IND} & \cdot & \cdots & 1 & \cdots & -1 & \cdots & \cdot & \cdots & -\mathrm{j}\omega L & \cdots \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots \end{array}$$

For coupled inductors (Fig. 3) the coupling appears in the constitutive relations of the two inductors, but not in the KCL part of the circuit equations.

$$\begin{aligned} V_{k1} - V_{l1} - \mathrm{j}\omega L_1 I_1 - \mathrm{j}\omega M_{12} I_2 = 0 \\ V_{k2} - V_{l2} - \mathrm{j}\omega M_{12} I_1 - \mathrm{j}\omega L_2 I_2 = 0 \end{aligned}$$

The element footprint of a pair of coupled inductors is

	$\cdots$	v_{k1}	$\cdots$	v_{l1}	$\cdots$	v_{k2}	$\cdots$	v_{l2}	$\cdots$	$\cdots$	i_1	$\cdots$	i_1
$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$
KCL_{k1}	$\cdots$	$\cdot$	$\cdots$	$\cdot$	$\cdots$	$\cdot$	$\cdots$	$\cdot$	$\cdots$	$\cdots$	1	$\cdots$	$\cdot$
$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$
KCL_{l1}	$\cdots$	$\cdot$	$\cdots$	$\cdot$	$\cdots$	$\cdot$	$\cdots$	$\cdot$	$\cdots$	$\cdots$	-1	$\cdots$	$\cdot$
$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$
KCL_{k2}	$\cdots$	$\cdot$	$\cdots$	$\cdot$	$\cdots$	$\cdot$	$\cdots$	$\cdot$	$\cdots$	$\cdots$	$\cdot$	$\cdots$	1
$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$
KCL_{l2}	$\cdots$	$\cdot$	$\cdots$	$\cdot$	$\cdots$	$\cdot$	$\cdots$	$\cdot$	$\cdots$	$\cdots$	$\cdot$	$\cdots$	-1
$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$
IND_1	$\cdots$	1	$\cdots$	-1	$\cdots$	$\cdot$	$\cdots$	$\cdot$	$\cdots$	$\cdots$	$-\mathrm{j}\omega L_1$	$\cdots$	$-\mathrm{j}\omega M_{12}$
$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$
IND_2	$\cdots$	$\cdot$	$\cdots$	$\cdot$	$\cdots$	1	$\cdots$	-1	$\cdots$	$\cdots$	$-\mathrm{j}\omega M_{12}$	$\cdots$	$-\mathrm{j}\omega L_2$
$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$

Coupled inductors do not contribute to the right-hand side vector. We can see that the element footprint is identical to the element footprint of the two inductors with the addition of the $\mathrm{j}\omega M_{12}$ term to both constitutive relations.

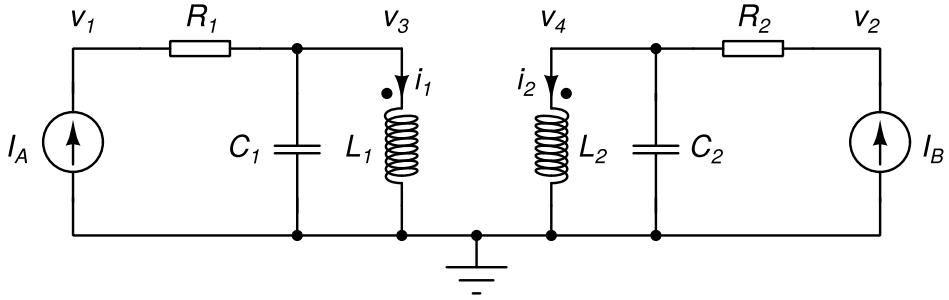


Fig. 4: Model of non-ideal transformer. The coupling factor between the two inductors is given by k.

Let us illustrate frequency-domain analysis of a linear circuit with an example (Fig. 4). The circuit is a model of a non-ideal transformer with imperfect magnetic coupling k<1, winding resistance, and winding capacitance. Two input signals are generated by the two current sources. The circuit has 5 nodes. Due to this the list of unknowns contains 4 node potentials. Additionally, two currents are introduced into the list of unknowns by the two inductors in the circuit. With our knowledge of element footprints we can write the system of equations by inspection.

$$\begin{bmatrix} R_1^{-1} & 0 & -R_1^{-1} & 0 & 0 & 0 \\ 0 & R_2^{-1} & 0 & -R_2^{-1} & 0 & 0 \\ -R_1^{-1} & 0 & R_1^{-1}+\mathrm{j}\omega C_1 & 0 & 1 & 0 \\ 0 & -R_2^{-1} & 0 & R_2^{-1}+\mathrm{j}\omega C_2 & 0 & 1 \\ 0 & 0 & 1 & 0 & -\mathrm{j}\omega L_1 & -\mathrm{j}\omega M_{12} \\ 0 & 0 & 0 & 1 & -\mathrm{j}\omega M_{12} & -\mathrm{j}\omega L_2 \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ V_4 \\ I_1 \\ I_2 \end{bmatrix} = \begin{bmatrix} I_A \\ I_B \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Nonlinear capacitors and inductors

All capacitors and inductors in this chapter were linear. Now we are going to introduce nonlinear capacitors and inductors. The former ones model charge storage in semiconductor devices, while the latter ones can be used for modelling coilswith nonlinear cores.

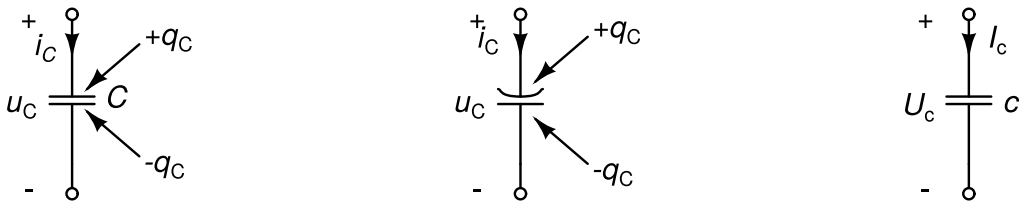


Fig. 5: A linear capacitor (left), a nonlinear capacitor (center), and the linearized model of a nonlinear capacitor (right).

A capacitor stores charge (Fig. 5, left). The plate connected to the node with the higher potential holds positive charge (q), while the opposite plate holds equally large negative charge (-q). For linear capacitors the charge is proportional to the voltage across the capacitor.

$$q_C = Cu_C$$

For nonlinear capacitors this relation is nonlinear. With respect to modelling there are two kinds of nonlinear capacitors. The ones where the charge can be expressed as a univariate function of the voltage (voltage-dependent capacitors) and the ones where the voltage can be expressed as a univariate function of charge (charge-dependent capacitors). Because in most real-world cases the nonlinear function is a bijective map (and thus its inverse exists) bot approaches are feasible for most real-world nonlinear capacitors. We are going to focus on voltage-dependent nonlinear capacitors because this approach fits well with the modified nodal analysis. For a voltage-dependent nonlinear capacitor (Fig. 5, center) we can write

$$q_C = q_C(u_C)$$

For both linear and nonlinear capacitors the charge conservation must be honored and therefore we can express the current flowing through a capacitor as

$$i_C = \frac{dq_C(u_C)}{dt}$$

If we assume the sotred charge depends only on the voltage (but not on time itself) we can write

$$i_C = \frac{dq_C(u_C)}{du_C}\frac{du_C}{dt} = c(u_C)\frac{du_C}{dt}$$

Here $c(u_C)$ denotes the differential capacitance which is voltage-dependent. For a linear capacitor the differential capacitance is equal to the capacitor's total capacitance (i.e. stored charge divided by the voltage). For a nonlinear capacitor it does not make sense to define the total capacitance because the charge is not proportional to the voltage.

Now suppose we slightly perturb the voltage of a nonlinear capacitor from U_C to $U_C+\Delta u_C$. How much does the charge change?

$$q_C(U_C) + \Delta q_C = q_C(U_C) + \left.\frac{dq_C}{du_C}\right|_{u_C=U_C} \Delta u_C$$

Because the first term on the left-hand side cancels out the first term on the right-hand side we have

$$\Delta q_C = \left.\frac{dq_C}{du_C}\right|_{u_C=U_C} \Delta u_C = c(U_C)\Delta u_C$$

We can see that a nonlinear capacitor behaves as a linear capacitor if we consider onlz the small changes in voltage and charge. Now suppose the voltage is composed of a DC component U_C and a small sinusoidal component given by complexor U_c. Because for small perturbations the

nonlinear capacitor behaves as a linear capacitor with capacitance equal to the differential capacitance at U_C the small-signal model of a nonlinear capacitor is a linear capacitor in Fig. 5 (right). A small sinusoidal voltage results in a small sinusoidal current flowing through a nonlinear capacitor which can be expressed as

$$I_c = \mathrm{j}\omega c(U_C)U_c$$

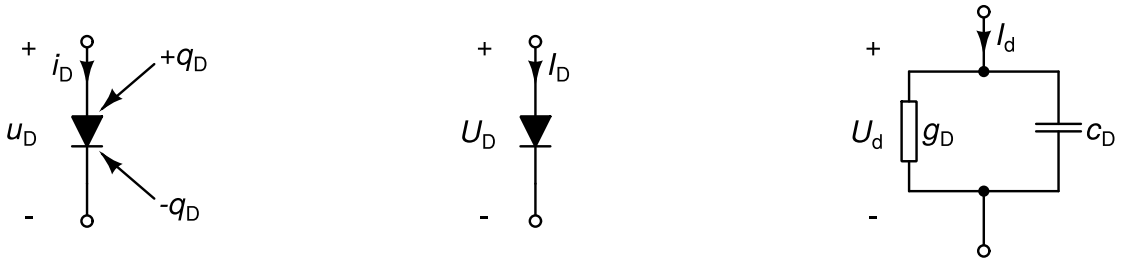


Fig. 6: A semiconductor diode (left), its operating point (center), and its linearized model (right).

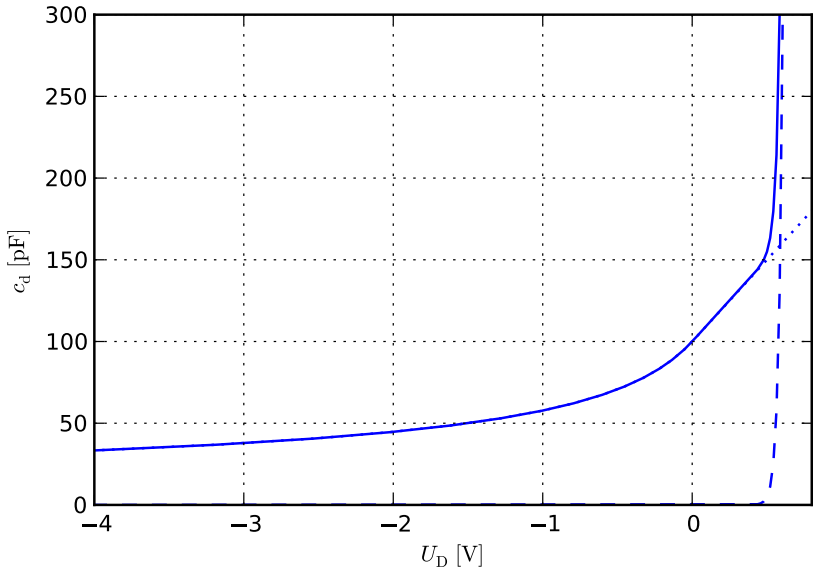
An example of a nonlinear capacitor is the capacitance of a semiconductor diode (Fig. 6, left). Its operating point (Fig. 6, center) is given by voltage U_D which drives a DC current I_D through the diode. The differential conductance of a diode (g_D) depends on the operating point and was computed in one of the previous lectures. The differential capacitance of a diode also depends on the voltage. The differential capacitance of a diode consists of three components: depletion capacitance which is dominant for reverse polarization (U_D<0), diffusion capacitance which dominates when the diode starts to conduct significant currents, and linear capacitance due to overlap effects. The depletion capacitance can be expressed with the voltage as

$$c_{dep} = \begin{cases} C_0 \left(1 - \frac{U_D}{V_J}\right)^{-M} & U_D \leq 0 \\ C_0 \left(1 + M\frac{U_D}{V_J}\right) & U_D > 0 \end{cases}$$

where C_0, M, and V_J are diode model parameters. The diffusion capacitance, on the other hand, depends on the resistive current flowing through the diode.

$$c_d = g_D\tau = \frac{I_S}{V_T}\mathrm{e}^{\frac{U_D}{V_T}}\tau$$

I_S and τ are diode model parameters, and V_T is the thermal voltage.



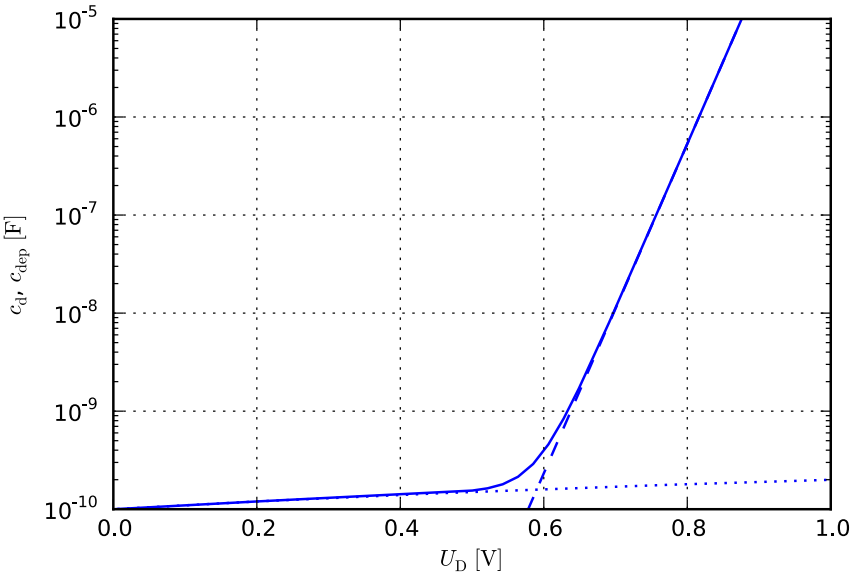


Fig. 7: Differential capacitance (full line), depletion capacitance (dotted line), and diffusion capacitance (dashed line) of a semiconductor diode with respect to operating point voltage in linear (top) and logarithmic (bottom) scale.

The differential capacitance of a diode (Fig. 7) is

$$c_D = \left.\frac{q_D}{u_D}\right|_{u_D=U_D} = c_{dep} + c_d + C_{ovl}$$

where the last term represents the overlap capacitance (which in turn is independent of the operating point).

Nonlinear inductances can be handled in a similar manner. Two kinds of nonlinear inductors exist with respect to the modelling approach. For current-dependent inductors the magnetic flux (**Φ**) can be expressed as a univariate function of the current. For flux-dependent inductors the current can be expressed as a univariate function of the magnetic flux. Again, in most practical cases both approaches can be applied as the mapping between flux and current is a bijective one. Here we are going to introduce the former one, where the magnetic flux is a nonlinear function of the current.

$$\phi_L = \phi_L(i_L)$$

The voltage across an inductor can then be expressed as

$$u_L = \frac{d\phi_L(i_L)}{dt}$$

Assuming the flux depends only on current, but not on time itself we can write

$$u_L = \frac{d\phi_L(i_L)}{di_L}\frac{di_L}{dt} = l(i_L)\frac{di_L}{dt}$$

By introducing differential inductance l (which in turn depends on the operating point) we arrive at the small-signal model of a nonlinear inductor which is a linear inductor with inductance equal to the differential inductance l(I_L) where I_L is the operating point current flowing through the inductor. We leave the rest of the derivation to the interested user, as nonlinear inductors are not common in modern integrated circuits.

Within this framework for a nonlinear inductor we can handle a linear inductor by writing

$$\phi_L = Li_L$$

Equations of nonlinear circuits revisited

Modified nodal analysis of circuits that comprise nonlinear capacitors results in a nonlinear system of equations of the form

$$\mathbf{g}(\mathbf{x}) + \tfrac{d}{dt}\mathbf{q}(\mathbf{x}) = \mathbf{y}$$

Here nonlinear vector-valued function $\mathbf{g}$ and vector $\mathbf{y}$ represent the resistive part of the circuit and its excitations, while $\mathbf{q}$ is a vector valued function expressing the total charge stored by the capacitors connected to a particular node in the circuit. Note that every component of $\mathbf{q}$ is a nonlinear function of the circuit's unknowns. In this way the stored charge can depend on an arbitrary node potential which results in nonlinear transcapacitances where the stored charge does not depend solely on the voltage between the capacitor's pins but also on other voltages in the circuit.

Within the framework of this equation we can also handle nonlinear inductors for which the magnetic flux is expressed as a function of the inductor's current. The inductor's current must be one of the circuit's unknowns. This current appears as a term in the corresponding KCL equations. The constitutive relation of the nonlinear inductor appears as an additional nonlinear equation in vector valued function $\mathbf{q}$.

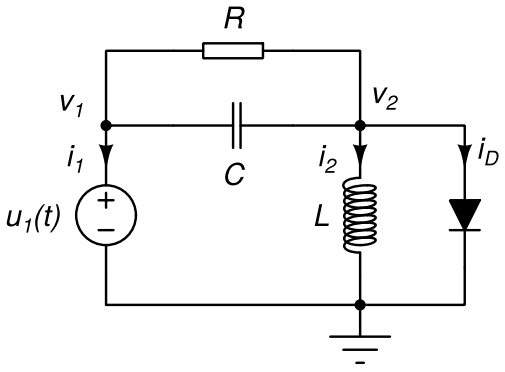


Fig. 8: A nonlinear circuit.

Let us illustrate this by writing down the nonlinear equations of a circuit which contains nonlinear resistive and capacitive elements in Fig. 8. The diode current consists of two components: a resistive one and a capacitive one.

$$i_D = I_S(\mathrm{e}^{v_2/V_T} - 1) + \tfrac{d}{dt}q_D(v_2)$$

The circuit has 3 nodes which result in w KCL equations. The independent volatge source and the inductor add two more unknowns to the system of equations (i_1 and i_2). With the knowledge we gather up to now the two KCL equations can be written down by inspection.

$$R^{-1}(v_1 - v_2) + \frac{d}{dt}\left(C(v_1 - v_2)\right) + i_1 = 0$$
$$R^{-1}(v_2 - v_1) + \frac{d}{dt}\left(C(v_2 - v_1)\right) + I_S(\mathrm{e}^{v_2/V_T} - 1) + \frac{d}{dt}q_D(v_2) + i_2 = 0$$

Two more equations are obtained from the constitutive relations of the independent voltage source and inductor.

$$v_1 = u_1(t)$$
$$v_2 - \frac{d}{dt}(Li_2) = 0$$

After gathering the resitive terms (terms that do not include derivatives with respect to time) and the reactive terms (the ones that include derivatives with respect to time) in two vectors we obtain the following system of equations.

$$\underbrace{\begin{bmatrix} R^{-1}v_1 - R^{-1}v_2 + i_1 \\ -R^{-1}v_1 + R^{-1}v_2 + I_S(\mathrm{e}^{v_2/V_T} - 1) + i_2 \\ v_1 \\ v_2 \end{bmatrix}}_{\mathbf{g}} + \frac{d}{dt}\underbrace{\begin{bmatrix} Cv_1 - Cv_2 \\ -Cv_1 + Cv_2 + q_D(v_2) \\ 0 \\ -Li_2 \end{bmatrix}}_{\mathbf{q}} = \underbrace{\begin{bmatrix} 0 \\ 0 \\ u_1(t) \\ 0 \end{bmatrix}}_{\mathbf{y}}$$

The first term on the left-hand side corresponds to the nonlinear resistive part of the circuit (vector-valued function $\mathbf{g}$) while the second term represents the nonlinear reactive part of the circuit (vector-valued function $\mathbf{q}$). The right-hand side represents the circuit's excitation (vector $\mathbf{y}$). Note that the last component of $\mathbf{q}$ represents the negative of the inductor's magnetic flux because we are modelling inductors by adding new unknowns (inductor currents) and expressing their constitutive relations in the same manner as we did with capacitors.

Small-signal frequency-domain analysis of nonlinear circuits

Suppose the circuit's excitations are composed of a DC component and a small sinusoidal component. Therefore we can write

$$\mathbf{g} = \mathbf{y}_{DC} + Re\left(\mathbf{y}_{AC}\mathrm{e}^{\mathrm{j}\omega t}\right)$$

Complex vector $\mathbf{y}_{AC}$ comprises complexors representing small sinusoidal excitations superimposed on the DC excitations specified by vector $\mathbf{y}_{DC}$. Because the sinusoidal excitations are small we can linearize the circuit and assume its response is of the form

$$\mathbf{x} = \mathbf{x}_{DC} + Re\left(\mathbf{x}_{AC}\mathrm{e}^{\mathrm{j}\omega t}\right)$$

where the magnitudes of components in complex vector $\mathbf{x}_{AC}$ are small. We already know how to linearize the resistive part of the circuit.

$$\mathbf{g}\left(\mathbf{x}_{DC} + Re\left(\mathbf{x}_{AC}\mathrm{e}^{\mathrm{j}\omega t}\right)\right) = \mathbf{g}(\mathbf{x}_{DC}) + \mathbf{G}(\mathbf{x}_{DC})Re\left(\mathbf{x}_{AC}\mathrm{e}^{\mathrm{j}\omega t}\right)) = \mathbf{g}(\mathbf{x}_{DC}) + Re\left(\mathbf{G}(\mathbf{x}_{DC})\mathbf{x}_{AC}\right)$$

Matrix $\mathbf{G}$ is the Jacobian of the vector valued function $\mathbf{g}$.

$$\mathbf{G}(\mathbf{x}_{DC}) = \left.\frac{d\mathbf{g}}{d\mathbf{x}}\right|_{\mathbf{x}=\mathbf{x}_{DC}} = \begin{bmatrix} \frac{\partial g_1}{\partial x_1} & \frac{\partial g_1}{\partial x_2} & \cdots & \frac{\partial g_1}{\partial x_n} \\ \frac{\partial g_2}{\partial x_1} & \frac{\partial g_2}{\partial x_2} & \cdots & \frac{\partial g_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial g_n}{\partial x_1} & \frac{\partial g_n}{\partial x_2} & \cdots & \frac{\partial g_n}{\partial x_n} \end{bmatrix}_{\mathbf{x}=\mathbf{x}_{DC}}$$

Now let us linearize the reactive part.

$$\mathbf{q}\left(\mathbf{x}_{DC} + Re\left(\mathbf{x}_{AC}\mathrm{e}^{\mathrm{j}\omega t}\right)\right) = \mathbf{q}(\mathbf{x}_{DC}) + \mathbf{C}(\mathbf{x}_{DC})Re\left(\mathbf{x}_{AC}\mathrm{e}^{\mathrm{j}\omega t}\right))\mathbf{q}(\mathbf{x}_{DC}) + Re\left(\mathbf{C}(\mathbf{x}_{DC})\mathbf{x}_{AC}\mathrm{e}^{\mathrm{j}\omega}\right.$$

Matrix $\mathbf{C}$ is the Jacobian of the vector valued function $\mathbf{q}$.

$$\mathbf{C}(\mathbf{x}_{DC}) = \left.\frac{d\mathbf{q}}{d\mathbf{x}}\right|_{\mathbf{x}=\mathbf{x}_{DC}} = \left[\begin{array}{ccc} \frac{\partial q_1}{\partial x_1} & \frac{\partial q_1}{\partial x_2} & \cdots & \frac{\partial q_1}{\partial x_n} \\ \frac{\partial q_2}{\partial x_1} & \frac{\partial q_2}{\partial x_2} & \cdots & \frac{\partial q_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial q_n}{\partial x_1} & \frac{\partial q_n}{\partial x_2} & \cdots & \frac{\partial q_n}{\partial x_n} \end{array}\right]_{\mathbf{x}=\mathbf{x}_{DC}}$$

It contains the differential capacitances of reactive elements. Computing the derivative of with respect to time eliminates the first term yielding

$$\frac{d}{dt}\mathbf{q}\left(\mathbf{x}_{DC} + Re\left(\mathbf{x}_{AC}\mathrm{e}^{\mathrm{j}\omega t}\right)\right) = Re\left(\mathrm{j}\omega\mathbf{C}(\mathbf{x}_{DC})\mathbf{x}_{AC}\mathrm{e}^{\mathrm{j}\omega t}\right)$$

By taking into account both linearizations the system of equations becomes

$$\mathbf{g}(\mathbf{x}_{DC}) + Re\left(\mathbf{G}(\mathbf{x}_{DC})\mathbf{x}_{AC}\mathrm{e}^{\mathrm{j}\omega t}\right)\right) + Re\left(\mathrm{j}\omega\mathbf{C}(\mathbf{x}_{DC})\mathbf{x}_{AC}\mathrm{e}^{\mathrm{j}\omega t}\right)\right) = \mathbf{y}_{DC} + Re\left(\mathbf{y}_{AC}\mathrm{e}^{\mathrm{j}\omega t}\right)$$

Because $\mathbf{g}(\mathbf{x}_{DC})=\mathbf{y}_{DC}$ must be satisfied (i.e. $\mathbf{x}_{DC}$ is the circuit's operating point) we can simplify the equation to

$$Re\left(\mathbf{G}(\mathbf{x}_{DC})\mathbf{x}_{AC}\mathrm{e}^{\mathrm{j}\omega t}\right)\right) + Re\left(\mathrm{j}\omega\mathbf{C}(\mathbf{x}_{DC})\mathbf{x}_{AC}\mathrm{e}^{\mathrm{j}\omega t}\right)\right) = Re\left(\mathbf{y}_{AC}\mathrm{e}^{\mathrm{j}\omega t}\right)$$

which finally yields

$$\left(\mathbf{G}(\mathbf{x}_{DC}) + \mathrm{j}\omega\mathbf{C}(\mathbf{x}_{DC})\right)\mathbf{x}_{AC} = \mathbf{y}_{AC}$$

For linear circuits the expression in parentheses is actually the matrix of coefficients of the circuit's linear system of equations obtained via modified nodal analysis.

To illustrate the small signal analysis of nonlinear circuits let us write down the system of equations for circuit in Fig. 8. First, let us obtain the equations by computing the Jacobians of the resistive and the reactive part. The former is

$$\mathbf{G}(\mathbf{x}_{DC}) = \left.\frac{\partial\mathbf{g}}{\partial\mathbf{x}}\right|_{\mathbf{x}=\mathbf{x}_{DC}} = \left[\begin{array}{cccc} R^{-1} & -R^{-1} & 1 & 0 \\ -R^{-1} & R^{-1}+g_D & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{array}\right]$$

Because g_D is the differential conductance of the diode, which is a nonlinear element, it depends on the operating point. Let V_1, V_2, I_1, and I_2 denote the operating point of the circuit. Then g_D can be computed as

$$g_D = \left.\frac{\partial i_D}{\partial v_2}\right|_{v_2=V_2} = \frac{I_S}{V_T}\mathrm{e}^{V_2/V_T}$$

The Jacobian of the reactive part of the circuit is

$$\mathbf{C}(\mathbf{x}_{DC}) = \left.\frac{\partial\mathbf{q}}{\partial\mathbf{x}}\right|_{\mathbf{x}=\mathbf{x}_{DC}} = \left[\begin{array}{cccc} C & -C & 0 & 0 \\ -C & C+c_D & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -L \end{array}\right]$$

The differential capacitance of the diode depends on the operating point and can be obtained from

$$c_D = \left.\frac{\partial q_D}{\partial v_2}\right|_{v_2=V_2}$$

We assume the excitation provided by the voltage source consists of a DC component U_1 and a small sinusoidal signal represented by complexor U_{1AC}. Similarly, all signals representing the response also consist of a DC component (i.e. the operating point of the circuit given by V_1, V_2, I_1, and I_2) and a small sinusoidal component represented by a complexor (V_{1AC}, V_{2AC}, I_{1AC}, and I_{2AC}).

The operating point is determined by solving the nonlinear system of equations where the term representing the reactive part of the circuit is removed. This means that in operating point analysis capacitors are removed and inductors are replaced with short circuits.

$$\underbrace{\begin{bmatrix} R^{-1}V_1 - R^{-1}V_2 + I_1 \\ -R^{-1}V_1 + R^{-1}V_2 + I_S(\mathrm{e}^{V_2/V_T} - 1) + I_2 \\ V_1 \\ V_2 \end{bmatrix}}_{\mathbf{g}} = \underbrace{\begin{bmatrix} 0 \\ 0 \\ U_1 \\ 0 \end{bmatrix}}_{\mathbf{y}}$$

After the operating point is obtained the differential capacitances of nonlinear elements can be computed and matrix $\mathbf{C}$ built. Note that the entries in matrix $\mathbf{C}$ belonging to linear elements do not depend on the operating point. Now we can build the system of equations for small signal analysis.

$$\begin{bmatrix} R^{-1} + \mathrm{j}\omega C & -R^{-1} - \mathrm{j}\omega C & 1 & 0 \\ -R^{-1} - \mathrm{j}\omega C & R^{-1} + g_D + \mathrm{j}\omega(C + c_D) & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -\mathrm{j}\omega L \end{bmatrix} \begin{bmatrix} V_{1AC} \\ V_{2AC} \\ I_{1AC} \\ I_{2AC} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ U_{1AC} \\ 0 \end{bmatrix}$$

From tegh resulting system of equations we can see that the matrix and the right-hand side can be assembled with the element footprint approach. Linear elements are handled in the same manner as we described in the beginning of this lecture. The footprints of nonlinear elements can be derived from their linearized models (e.g. see Fig. 6 (right) for the linearized model of a semiconductor diode). The right-hand side contributions of the independent sources generating nonzero sinusoidal excitations are built in the same manner as the contributions of those sources in operating point analysis, except that now their value is equal to the complexor that represents the sinusoidal signal.

Small-signal frequency-domain analysis in SPICE OPUS

In SPICE the small-signal frequency-domain analysis is called AC analysis. At invocation the frequency range and step must be specified. The magnitudes and optional phases of sinusoidal excitations are specified in the circuit description by passing the ac parameter to independent sources. Note that the ac value is used only in small-signal frequency-domain analysis.

The simulator first computes the operating point of the circuit. In this computation all reactive elements are disabled (capacitors areremoved and inductors replaced with short circuits). The operating point is used for computing the Jacobians of the resistive and the reactive part of the circuit. The approach via element footprints is used for quickly assembling these two matrices. The right-hand size of the system of equations for small-signal frequency-domain analysis is assembled based on the values of the ac parameter passed to the independent sources (again with the element footprint approach). The system of equations is then solved for all frequencies specified by the range and the step parameters at analysis invocation. The linearized models are computed only once because the equations differ between two frequency points only by the value of ω.

AC analysis is quite fast. It requires only one LU decomposition per frequency point. For a moderate number of frequency points a large part of the time spent by the analysis is used for computing the operating point.

Small-signal noise modeling and analysis

8th Lecture W2, December

We briefly introduce the relevant aspects of noise modeling and analysis in linear circuits. Various types of noise appearing in electronic circuits are presented and characterized. Noise models of selected circuit elements are presented. We introduce small-signal noise analysis as a special case of AC analysis where signals are represented with power spectrum densities. We conclude with the computation of output and equivalent input noise contributions.

« READ LESS

Noise as a signal

One way to classify signals is according to their energy and power. Let x(t) denote a signal. Signals with finite energy satisfy

$$\int_{-\infty}^{\infty} |x(t)|^2 dt < \infty$$

As time approaches positive or negative infinity such signals must approach zero, Due to this periodic signals do not belong to this class of signals. Instead they often satisfy a less strict requirement, i.e. they have finite power.

$$\frac{1}{T} \int_{0}^{T} |x(t)|^2 dt < \infty$$

where T denotes the period of the signal. Noise signals are generated by random processes. We are going to focus on noise signals with finte power. First of all, noise signals are random signals. The same noise source can generate infinitely many realizations of the same noise signal. Therefore observing the dependence of the signal on time does not deliver much useful information. Let x(t) denote a realization of a noise signal. All realiyations of a particular noise signal have some common properties. Mathematically these properties can be formulated with teh correlation function. The correlation function of two signals x(t) and y(t) is defined as

$$c_{xy}(t, \tau) = E\left[x(t) y(t+\tau)\right]$$

where E[.] denotes expectation, i.e. the mean value computed across all possible realizations of the two sigfnals. If x(t) and y(t) are generated by two stationary random processes the correlation function depends only on τ. If the two processes are also ergodin (i.e. its statistical properties can be obtained from a sufficiently long realization of the noise signal) then averaging over all realizations can be replaced with averaging over time.

$$c_{xy}(\tau) = \lim_{T\to\infty} \frac{1}{T} \int_{-T/2}^{T/2} x(t)y(t+\tau)dt$$

Most noise signals we meet in practice are ergodic. If we choose y(t)=x(t) the correlation function is also referred to as the autocorrelation function.

$$c_{xx}(\tau) = \lim_{T\to\infty} \frac{1}{T} \int_{-T/2}^{T/2} x(t)x(t+\tau)dt$$

The fourier transform of the autocorrelation function is also referred to as the power spectrum density.

$$S_{xx}(f) = \mathcal{F}\left(c_{xx}(t)\right) = \int_{-\infty}^{\infty} c_{xx}(t)\mathrm{e}^{-\mathrm{j}2\pi ft}dt$$

where f denotes the frequency. The integral of the power spectrum density is equal to the signal's power (Parseval's theorem).

$$\int_{-\infty}^{\infty} S_{xx}(f)df = c_{xx}(0) = \lim_{T\to\infty} \frac{1}{T} \int_{-T/2}^{T/2} |x(t)|^2 dt$$

The autocorrelation function is symmetric ($c_{xx}(-\tau)=c_{xx}(\tau)$). Therefore the power spectrum density is an even function (i.e. $S_{xx}(-f)=S_{xx}(f)$) so it is sufficient to know its values for nonnegative frequencies. With this in mind we introduce one-sided power spectrum density.

$$S_{xx}^{+}(f) = 2S_{xx}(f)$$

Parseval's theorem for one-sided power spectrum density can be written as

$$\int_{0}^{\infty} S_{xx}^{+}(f)df = c_{xx}(0) = \lim_{T\to\infty} \frac{1}{T} \int_{-T/2}^{T/2} |x(t)|^2 dt$$

We introduced this one-sided power spectrum density because it is used for noise characterization in circuit simulators. The power spectrum density also has a physical meaning. Suppose we pass a noise signal through an ideal bandpass filter with pass-band between f_1 and f_2, and measure the RMS value of the output signal (RMS). The filter eliminates all frequencies outside the pass band. The following relation between the measured RMS value and the power spectrum density holds

$$RMS^2 = \int_{f_1}^{f_2} S_{xx}^{+} df$$

Common types of noise and their spectra

Thermal noise (Johnson-Nyquist noise)

This type of noise arises due to the chaotic movement of electrons in the conductor. Every resistor generates thermal noise. It can be modelled with a current source i(t) in parallel with the resistor where the noise originates (Fig. 1, right). Note that the polarity of the source is not important because the power spectrum density does not change if we reverse the current.

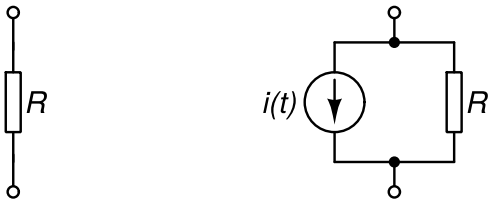


Fig. 1: Ideal resistor (left) and a resistor that generates thermal noise (right).

The current source produces thermal noise with the following one-sided power spectrum density

$$S_{ii}^{+}(f) = \frac{4Rhf}{R\left(\mathrm{e}^{\frac{hf}{kT}} - 1\right)}$$

where h is the Planck constant, k is the Boltzmann constant, T is the absolute temperature, and R is the resistance of the resistor, Because at room temperature the exponent in the exponential term is small for frequencies below 6THz we can simplify the formula to

$$S_{ii}^{+}(f) = \frac{4kT}{R}$$

We can see the power spectrum density does not depend on frequency across a very wide range. Therefore we refer to this noise as "white".

Shot noise

This type of noise occurs because the electric current consists of a flow of discrete charges (electrons). Its power spectrum density does not depend on temperature or frequency. In a diode this type of noise is modelled by a current source in parallel with the p-n junction (Fig. 2, right, represented by current source $i_s(t)$).

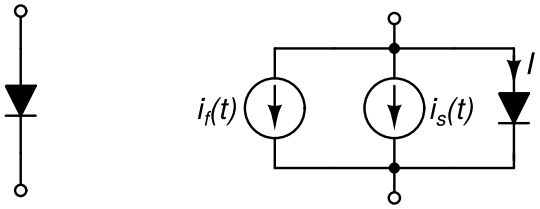


Fig. 2: Ideal diode (left) and a diode with current sources representing flicker noise i_f and shot noise i_s (right).

The power spectrum density of of the current source representing shot noise is

$$S_{ii}^{+}(f) = 2qI$$

where q is the electron charge and I is the current flowing across the p-n junction. Shot noise is also "white".

Flicker noise

The power spectrum density of flicker noise is inversely proportional to the frequency. This type of noise is also referred to as 1/f noise or pink noise. In a semiconductor diode flicker noise is modelled by a current source in parallel with the p-n junction (Fig. 2, right, represented by current source $i_f(t)$). The power spectrum density of the current source is

$$S_{ii}^{+}(f) = \frac{K_f I^{A_f}}{f}$$

where K_f and A_f are two constants that characterize the flicker noise, and I is the current flowing through the p-n juncion. Noise signals with power spectrum densities proportional to 1/f are also deemed pink noise.

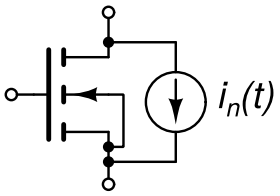


Fig. 3: A current source for modelling channel thermal noise and flicker noise in a MOS transistor.

We can model noise by introducing noise sources in arbitrary semiconductor devices. Take, for instance, an MOS transistor (Fig. 3). The channel thermal noise and flicker noise can be modelled with a single current source connected between the drain and the source terminal. We are not going to introduce the expression for the power spectrum density of this source because it exceeds the scope of this lecture. Let us only note that the power spectrum density depends on the currents and voltages at the terminals of a MOS transistor.

Modelling the noise generated by circuit elements

All resistances and semiconductor devices are sources of noise. The noise generated by circuit elements is modelled by current sources. For shot and flicker noise the power spectrum density of the noise current source depends on the current flowing through the device. One semiconductor element has several such noise sources. A diode, for instance, has three: shot noise and flicker noise generated by the current flowing across the p-n junction, and thermal noise of the contact resistance. A bipolar transitor has 5 noise sources: shot noise and flicker noise due to the device current, and thermal noise of the contact resistances of the emitter, base, and the collector contact.

Small-signal transfer function

Before we proceed to computing the output noise of a circuit we must introduce the notion of a small-signal transfer function. Suppose there is an independent source in the circuit that generates a small-signal sinusoidal excitation characterized by complexor X. Let us assume this complexor does not depend on the frequency. The excitation results in small sinusoidal responses observed all over the circuit. Let Y denote the complexor representing one such response observed at a selected point in the circuit. The small-signal transfer function from the independent source to the selected point where the response is observed is defined as

$$H(f) = \frac{Y(f)}{X}$$

Note that the transfer function depends on the frequency. It can be computed from the results of the small-signal frquency-domain analysis (i.e. AC analysis in SPICE) by simply dividing the observed response with the complexor representing the excitation. The unit of the transfer function depends on the type of the observed response (voltage or current) and the type of the excitation (independent voltage or current source). If the circuit is excited by a voltage source and the observed response is a current then the units of the transfer function are A/V. If, on the other hand, the response is also a voltage, the transfer function is a dimensoinless quantity (since it is a ratio of two voltages).

Noise in linear systems

Nonlinear circuits behave as linear if we consider only the small signal sinusoidal excitations and the corresponding response. The steady-state response of a linear system excited by an independent sinusoidal source is sinusoidal and can be represented by a complexor (Y). This

complexor can be computed from the complexor (X) representing the excitation and the small-signal transfer function as

$$Y = H(f)X$$

where f denotes the frequency of the excitation. If the input signal is a small noise signal with power spectrum density $S_{xx}^{+}(f)$ then the output of the linear system is also a noise signal with power spectrum density $S_{yy}^{+}(f)$. The output power spectrum density can be computed as

$$S_{yy}^{+}(f) = |H(f)|^2 S_{xx}^{+}(f)$$

Now suppose a linear system is excited by two independent sources X_1 and X_2. If we observe the response of the system to X_1 while X_2 is disabled (i.e. set to zero) the response can be expressed as

$$Y_1 = H_1(f)X_1$$

where H_1 is the transfer function from the first independent source to the output of the system. Similarly, if X_1 is disabled and the system is excited only by X_2 the response is

$$Y_2 = H_2(f)X_2$$

where H_2 is the transfer function from the second independent source to the output of the system. Now suppose the system is excited simultaneously by both independent sources. The response of the system can then be expressed as

$$Y = H_1(f)X_1 + H_2(f)X_2 = Y_1 + Y_2$$

This property is also referred to as superposition. With this tool in our hands we could compute the time-domain response of a circuit excited by individual noise sources and then obtain the response of the circuit excited by all noise sources simultaneously by adding up these partial responses. If, however, we are interested in the power spectrum density of the response we need one more piece of the puzzle.

So how do we treat noise signals that are obtained by summing two noise signals $x_1(t)$ and $x_2(t)$? Things are quite simple if the two signals are uncorrelated, i.e. the correlation function satisfies

$$c_{12}(\tau) = \lim_{T\to\infty} \int_{-T/2}^{T/2} x_1(t)x_2(t+\tau)dt = 0$$

In real-world circuits the noise sources representing various types of noise generated by a circuit element are uncorrelated. Similarly, noise sources modelling the noise generated by two circuit elements are also uncorrelated. Let $S_{11}^{+}(f)$ and $S_{22}^{+}(f)$ denote the power spectrum densities of $x_1(t)$ and $x_2(t)$, respectively. Then the power spectrum density of $y(t)=x_1(t)+x_2(t)$ is

$$S_{yy}^{+}(f) = S_{11}^{+}(f) + S_{22}^{+}(f)$$

Uncorrelated noise signals remain uncorrelated even after they are transformed by a linear system. If we put together all we have learned so far: for a linear system is excited by two uncorrelated noise sources with power spectrum densities $S_{11}^{+}(f)$ and $S_{22}^{+}(f)$ the power spectrum density of the output noise can be expressed as

$$S_{yy}^{+}(f) = |H_1(f)|^2 S_{11}^{+}(f) + |H_2(f)|^2 S_{22}^{+}(f)$$

where H_1 and H_2 are transfer functions from the two noise sources to the circuit's output. This

formula can be generalized to an arbitrary number of noise sources and is the basis for small-signal noise analysis in circuit simulators.

Small-signal noise analysis

We illustrate the small-signal noise analysis with an example. Take, for instance, the nonlinear circuit from previous chapter (Fig. 4). This time we added the noise sources modelling the noise generated by the two resistors and the MOS transistor.

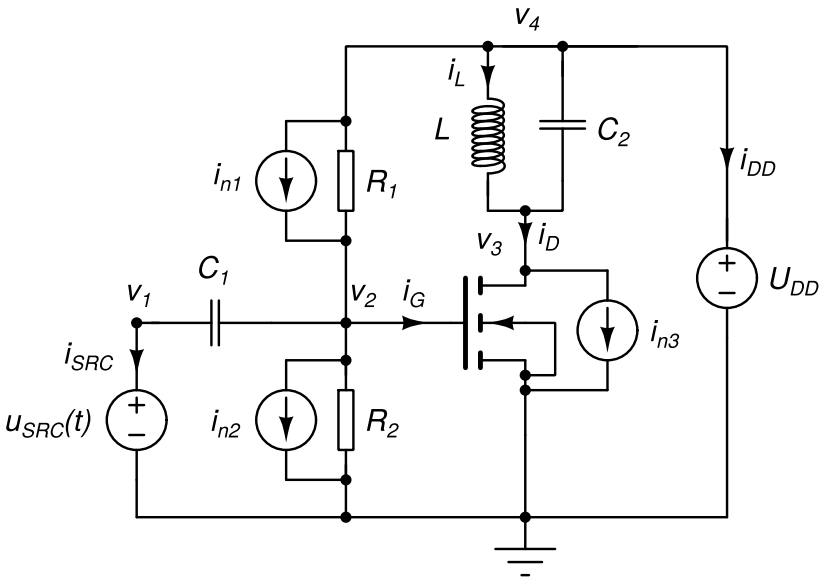


Fig. 4: A nonlinear circuit with noise sources (i_{n1}, i_{n2}, and i_{n3}).

The first step of small-signal noise analysis is the computation of the circuit's operating point. In this computation all noise sources are disabled (set to 0). The obtained operating point is used for computing the power density spectrum of the noise generated by the MOS transistor and represented by i_{n3}. Two noise sources are independent of the operating point (thermal noise generated by the two resistors represented by i_{n1} and i_{n2}). The operating point is also used for computing the linearized models of nonlinear elements (i.e. MOS transistor). After the linearization is complete the frequency-domain system of equations for the linearized circuit is assembled. The coefficient matrix ($\mathbf{Z}$) for the circuit in Fig. 4 was constructed in the previous lecture,

$$\mathbf{Z}(\mathbf{x}_{DC}) = \begin{bmatrix} \mathrm{j}\omega C_1 & -\mathrm{j}\omega C_1 & 0 & 0 \\ -\mathrm{j}\omega C_1 & R_1^{-1} + R_2^{-1} + \mathrm{j}\omega(C_1 + c_{gd} + c_{gs}) & -\mathrm{j}\omega c_{gd} & -R_1^{-1} \\ 0 & g_{21} - \mathrm{j}\omega c_{gd} & g_{22} + \mathrm{j}\omega(C_2 + c_{gd}) & -\mathrm{j}\omega C_2 \\ 0 & -R_1^{-1} & -\mathrm{j}\omega C_2 & R_1^{-1} + \mathrm{j}\omega C_2 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & 1 \end{bmatrix}$$

The output noise comprises contributions from all three noise sources. To compute these contributions we first need to compute the transfer functions from these noise sources to the output signal. For that purpose the small-signal analysis described in the previous chapter is used. The procedure we are going to describe computes the power spectrum density of output noise at a single frequency.

To compute the transfer function from i_{n1} to v_3 we must construct the corresponding system of equations for a circuit where the only sinusoidal excitation is comming from i_{n1}. The coefficient matrix of this system does not depend on the noise source because independent sources

contribute only to the right-hand side of the system. The only thing we need to construct is the right-hand side, which is easy, as we already know the matrix footprint of an independent current source. If we set the complexor representing the AC current generated by i_{n1} to 1 the complexor of the response observed at v_3 will be identical to the transfer function we want to compute. This may seem strange because in previous chapter we were discussing how the excitations for small-signal frequency-domain analysis must be small signals. On the other hand, small-signal analysis computes the solution of a linear circuit. For linear circuits arbitrary magnitudes can be used for excitations. By increasing the excitation of a linear circuit the output signal increases by the same factor. Of course, in real world circuits such large excitations don't result is small sinusoidal perturbations of the operating point. But then again, real world circuits are not linear and cannot handle arbitrary magnitudes of excitation. With this in mind we obtain the following system of equations

$$\mathbf{Z}(\mathbf{x}_{DC}) \begin{bmatrix} V_{1AC} \\ V_{2AC} \\ V_{3AC} \\ V_{4AC} \\ I_{SRCAC} \\ I_{DDAC} \\ I_{LAC} \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ -1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

By solving for V_{3AC} we obtain the desired transfer function denoted by H_1. Similarly for the transfer function from i_{n2} to v_3 the system of equations is

$$\mathbf{Z}(\mathbf{x}_{DC}) \begin{bmatrix} V_{1AC} \\ V_{2AC} \\ V_{3AC} \\ V_{4AC} \\ I_{SRCAC} \\ I_{DDAC} \\ I_{LAC} \end{bmatrix} = \begin{bmatrix} 0 \\ -1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

By solving for V_{3AC} we obtain the transfer function denoted by H_2. Finally, to obtain the transfer function from i_{n3} to v_3 we must solve

$$\mathbf{Z}(\mathbf{x}_{DC}) \begin{bmatrix} V_{1AC} \\ V_{2AC} \\ V_{3AC} \\ V_{4AC} \\ I_{SRCAC} \\ I_{DDAC} \\ I_{LAC} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

By solving for V_{3AC} we obtain the transfer function denoted by H_3. Because matrix $\mathbf{Z}$ depends on the frequency (ω) we can compute the values of all required transfer functions at a single frequency with only one LU decomposition which is common to all transfer functions at given ω. Computing the value of a transfer function for one ω requires only one forward and one backward substitution.

Now suppose S_{11}^+, S_{22}^+, and S_{33}^+ denote the power spectrum densities of i_{n1}, i_{n2}, and i_{n3} at the chosen frequency for which the transfer functions H_1, H_2, and H_3 were computed. The output

power spectrum density (S_{out}^+) observed at the circuit's output (node potential v_2) is then

$$S_{out}^+ = |H_1|^2 S_{11}^+ + |H_2|^2 S_{22}^+ + |H_3|^2 S_{33}^+$$

The units of the power spectrum density are V^2/Hz if the observed output signal is a voltage. If the output signal is a current the power spectrum density is in A^2/Hz.

Equivalent input noise

Sometimes we are interested how much noise (i.e. power density spectrum) must be injected at the circuit's input to recreate the power density noise spectrum at the circut's output assuming all noise sources in the circuit are disabled. This equivalent input noise power density spectrum represents the aggregation of all noise sources at the circuit's input.

To compute it, we need to define the circuit's input (i.e. the independent source that produces the input signal). Suppose for the circuit in Fig. 4 this is $u_{SRC}(t)$. First, we need to compute the transfer function from this source to the circuit's output (let us assume this is again v_3) defined as V_{3AC}/U_{SRCAC}. We can obtain this transfer function by setting U_{SRCAC} to 1 while disabling all other independent sources. This produces the following system of equations (which depends on frequency **ω**).

$$\mathbf{Z}(\mathbf{x}_{DC}) \begin{bmatrix} V_{1AC} \\ V_{2AC} \\ V_{3AC} \\ V_{4AC} \\ I_{SRCAC} \\ I_{DDAC} \\ I_{LAC} \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

After solving this system the resulting V_{3AC} corresponds to the transfer function which we denote by H. Because power spectrum densities are transformed by multiplying the input power spectrum density with the squared absolute value of the transfer function the equivalent input noise power spectrum density is obtained as

$$S_{in}^+ = S_{out}^+ / |H|^2$$

The computation of H and S_{out}^+ needs to be repeated for every frequency for which we would like to compute the equivalent input noise power spectrum density. If the input signal source is a voltage source the equivalent input noise power spectrum density is in V^2/Hz. If the input source is a current source the result is in A^2/Hz.

Small-signal noise analysis in SPICE OPUS

When performing noise analysis one has to specify the output signal (which can be any node potential or current appearing in the system of equations as an unknown), the input voltage source or current source (for computing the equivalent input noise), and the frequency range (in the same manner as for AC analysis). The simulator produces two groups of results. The first one comprises output and equivalent input noise power spectrum densities along with output noise power spectrum density contributions corresponding to individual noise sources. The second group of results comprises integrals of noise spectra over the whole simulated frequency range.

Time-domain analysis

9th Lecture W3, December

To simulate the circuit in time-domain we first divide the time scale in discrete equidistant points. A nonlinear circuit is solved at every timepoint. Reactive elements are handled by expressing the time derivatives in their constitutive relations with approximations based on circuit solution at past timestep and the one we are currently computing (implicit integration). Several integration algorithms are presented. The local truncation error (LTE) is introduced and we show how it can be kept low by selecting an appropriate timestep. We replace fixed timestep with a variable one to obtain the time-domain analysis approach used in modern circuit simualtors. Variable timestep complicates the integration algorithm because its coefficients must be recomputed for every new timestep. We show how this is achieved with selected numerical integration algorithms.

« READ LESS

Approximating the time scale with discrete steps

Suppose we want to simulate the behavior of a circuit from t_1 to t_2. Clearly, we cannot solve the circuit for all timepoints. Therefore we divide the timescale in discrete steps. Let t_k denote the timepoint where the latest solution of the circuit's equations was computed. The solution we are trying to compute corresponds to timepoint t_{k+1}. Let $h_k=t_{k+1}-t_k$ denote the k-th timestep between t_k and t_{k+1}. In circuit simulation the timestep is usually not constant because the dynamics of the circuit's behavior can vary greatly from time to time. When the circuit's response changes quickly the timestep is appropriately small. When the circuit "sleeps" and nothing interesting is happening with voltages and currents the timestep can be much greater.

Numerical integration of differential equations

As we learned in previous chapters the system of equations describing the circuit's behavior in time domain can be formulated as

$$\mathbf{g}(\mathbf{x}) + \frac{d}{dt}\mathbf{q}(\mathbf{x}) = \mathbf{y}$$

where vector-valued function $\mathbf{g}$ represents the resistive part of the circuit, vector-valued function $\mathbf{q}$ represents the reactive part of the circuit, vector $\mathbf{y}$ represents the independent sources (excitations), and vector $\mathbf{x}$ represents the unknowns (node potentials and branch currents introduced by the modified nodal approach to circuit equations). This is a nonlinear ordinary differential equation (ODE). Until now we managed to avoid the derivative term. In operating point analysis and DC small-signal analysis it simply vanished. In AC small-signal analysis we transformed the equation to frequency domain which changed the derivative term to simple multiplication with $j\omega$. But now we must face the music and deal with the derivative term.

The usuall approach is to approximate the derivative of a quantity with a weighted sum of the quantity and its derivatives at past timepoints t_k, t_{k-1}, ... (where they are already known). This

procedure is referred to as numerical integration of ODEs. In fact, we go even one step further. In this weighted sum we also allow the value of the quantity at the timepoint for which we are trying to solve the circuit (t_{k+1}). If we do the latter the numerical integration is referred to as implicit numerical integration, opposite to explicit numerical integration where we avoid using quatities that are not computed yet.

For the sake of simplicity we introduce the following notation. Let $\dot{q}$ denote the derivative of q with respect to time. Subscripts denote timepoint indices, i.e. q_k corresponds to timepoint t_k. A very simple implicit integration algorithm is the backward Euler algorithm where the derivative is expressed with the last computed value and the one that is about to be computed.

$$\dot{q}_{k+1} = \frac{1}{h_k}(q_{k+1} - q_k)$$

Another simple algorithm is the trapezoidal algorithm which originates in the reasoning that the finite difference approximation to the derivative is assumed to be equal to the average of the derivative at the endpoints of the interval.

$$\frac{q_{k+1}-q_k}{h_k} = \frac{\dot{q}_{k+1}+\dot{q}_k}{2}$$

From here we can express the trapezoidal formula

$$\dot{q}_{k+1} = -\dot{q}_k + \frac{2}{h_k}(q_{k+1} - q_k)$$

Nonlinear reactive elements and their matrix footprints

If, for instance, the backward Euler formula is applied to the circuit equations we obtain the following algebraic equation for the circuit's response at t_{k+1}.

$$\mathbf{g}(\mathbf{x}_{k+1}) + \frac{1}{h_k}\left(\mathbf{q}(\mathbf{x}_{k+1}) - \mathbf{q}(\mathbf{x}_k)\right) = \mathbf{y_{k+1}}$$

In this equation $\mathbf{q}(\mathbf{x}_k)$ and $\mathbf{y}_{k+1}$ are known (the former one can be computed from the solution at the previous timepoint and the latter one is the value of circuit's excitations at the timepoint where we are solving the circuit).

$$\mathbf{g}(\mathbf{x}_{k+1}) + \frac{1}{h_k}\mathbf{q}(\mathbf{x}_{k+1}) = \frac{1}{h_k}\mathbf{q}(\mathbf{x}_k) + \mathbf{y_{k+1}}$$

This is a nonlinear equation which can be solved by applying the Newton-Raphson algorithm. From this equation we can quickly deduce the matrix footprint of a linear capacitor. We start with its constitutive relation where the charge is expressed with the voltage as q = C v. In circuit equations the capacitor contributes a term of the form $\dot{q}$. When we are solving at timepoint t_{k+1} we must express $\dot{q}_{k+1}$ with past values of the circuit's response and q_{k+1} (afterall we are using implicit integration). Suppose we use the backward Euler formula which yields

$$\dot{q}_{k+1} = \frac{1}{h_k}\left(q_{k+1} - q_k\right) = \frac{C}{h_k}v_{k+1} - \frac{C}{h_k}v_k$$

Fig. 1: Linear capacitor (left) and its model used for computing the transient response at one timestep (right).

Here v_{k+1} is the unknown and v_k is the circuit's solution at the last computed timepoint. Because $\dot{q}$ is the capacitor current we can interpret the obtained equation as a parallel connection of a resistor with resistance $R=h_k/C$ and independent current source $I=-C\ v_k/h_k$ (Fig. 1). From this we can conclude that the matrix footprint fo a capacitor connected between nodes a and b for solving the

circuit at timepoint t_{k+1} is

$$\begin{array}{c|ccccccc} & v_1 & \cdots & v_a & \cdots & v_b & \cdots & v_{n-1} \\ \hline \mathrm{KCL}_1 & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \mathrm{KCL}_a & \cdot & \cdots & \frac{h_k}{C} & \cdots & -\frac{h_k}{C} & \cdots & \cdot \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \mathrm{KCL}_b & \cdot & \cdots & -\frac{h_k}{C} & \cdots & \frac{h_k}{C} & \cdots & \cdot \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \mathrm{KCL}_{n-1} & \cdot & \cdots & \cdot & \cdots & \cdot & \cdots & \cdot \end{array}$$

Similarly, the contribution to the right-hand side (originating from the current source) is

$$\begin{array}{c|c} & \mathrm{RHS} \\ \hline \mathrm{KCL}_1 & \cdot \\ \vdots & \vdots \\ \mathrm{KCL}_a & \frac{Cv_k}{h_k} \\ \vdots & \vdots \\ \mathrm{KCL}_b & -\frac{Cv_k}{h_k} \\ \vdots & \vdots \\ \mathrm{KCL}_{n-1} & \cdot \end{array}$$

Now suppose we use the trapezoidal integration formula. The capacitor current ($\dot{q}$) is now expressed as

$$\dot{q}_{k+1} = -\dot{q}_k + \frac{2}{h_k}\left(q_{k+1} - q_k\right) = \frac{2C}{h_k}v_{k+1} - \frac{2C}{h_k}v_k - \dot{q}_k$$

One migh ask at this point: where do we get $\dot{q}_k$ from. Fortunately the derivative is computed using the integration formula at every past timestep for which the circuit's equations were solved. We just need to store it for later use. This leaves us in a bit of a dilemma - what to do when we are solving the circuit's equations at the first timepoint? Well, in that case we resort to integration formulae which do not require the knowledge of the derivative at past timepoints (e.g. Backward Euler formula). We can switch to a more advanced integration formula (like trapezoidal integration) at the second timepoint.

Generalized approach to numerical integration formulae

There exist many integration formulae. In numerical mathematics their purpose is to express the value of a quantity q at timepoint t_{k+1} with the values of the quantity and its first derivative $\dot{q}$ at past timepoints and t_{k+1} (the latter is used only in implicit formulae). The following ansatz can be used for expressing an arbitrary integration formula.

$$q_{k+1} = \sum_{i=0}^{p} a_i q_{k-i} + h_k \sum_{i=-1}^{r} b_i \dot{q}_{k-i}$$

Suppose q is a function of time. For most functions an integration formula is only approximate in the sense that q_{k+1} is only approximately equal to the right-hand side of the ansatz. An exception to this are polynomials of an order not exceeding n. An integration formula has order n if it is exact for polynomials of order up to and including n. By assuming q(t) is a polynomial of time of the form αt^j we can derive coefficients a_i and b_i.

Suppose we want do derive an implicit integration formula of order n=1 (**backward Euler formula**). Because a polynomial of first order has only two coefficients the integration formula can have only two nonzero coefficients. Because we chose to construct an implicit formula, b_{-1} will be nonzero. As the second nonzero coefficient we choose a_0. The ansatz for the integration formula is therefore

$$q_{k+1} = a_0 q_k + h_k b_{-1} \dot{q}_{k+1}$$

For a polynomial q(t)=α we have

$$q_{k-i} = q(t_{k-i}) = \alpha$$
$$\dot{q}_{k-i} = \dot{q}(t_{k-i}) = 0$$

Substituting this in our ansatz yields the first equation for computing the coefficients b_{-1} and a_0.

$$\alpha = a_0 \alpha + h_k b_{-1} \cdot 0$$

For a polynomial q(t)=αt we can write

$$q_{k-i} = q(t_{k-i}) = \alpha t_{k-i}$$
$$\dot{q}_{k-i} = \dot{q}(t_{k-i}) = \alpha$$

By substituting this in the ansatz we get the second equation for computing coefficients a_{-1} and a_0.

$$\alpha t_{k+1} = a_0 \alpha t_k + h_k b_{-1} \alpha$$

Equations can be simplified if we assume $t_k=0$ which implies $t_{k+1}=h_k$. The system of equations is now

$$a_0 = 1$$
$$b_{-1} = 1$$

This system is already solved. The obtained numerical integration formula is therefore

$$q_{k+1} = q_k + h_k \dot{q}_{k+1}$$

By expressing $\dot{q}_{k+1}$ we get the backward Euler formula we introduced earlier.

$$\dot{q}_{k+1} = \frac{1}{h_k} (q_{k+1} - q_k)$$

Trapezoidal formula is a second order implicit formula (n=2). We obtain it if we assume the nonzero coefficients are a_0, a_1, and b_{-1}. The ansatz for the formula is

$$q_{k+1} = a_0 q_k + h_k b_0 \dot{q}_k + h_k b_{-1} \dot{q}_{k+1}$$

By substituting q(t)=α and q(t)=αt in the ansatz we get the first two equations for computing the coefficients.

$$\alpha = a_0 \alpha + h_k b_0 \cdot 0 + h_k b_{-1} \cdot 0$$
$$\alpha t_{k+1} = a_0 \alpha t_k + h_k b_0 \alpha + h_k b_{-1} \alpha$$

The third equation is obtained by substituting $q(t)=\alpha t^2$ in the ansatz. For this purpose we first compute

$$q_{k-i} = q(t_{k-i}) = \alpha t_{k-i}^2$$
$$\dot{q}_{k-i} = \dot{q}(t_{k-i}) = 2\alpha t_{k-i}$$

By substituting this in the ansatz we obtain the third equation.

$$\alpha t_{k+1}^2 = a_0 \alpha t_k^2 + h_k b_0 2\alpha t_k + h_k b_{-1} 2\alpha t_{k+1}$$

Again, we assume t_k=0 and t_{k+1}=h_k which greatly simplifies the equations and yields the following linear system.

$$\begin{aligned} a_0 &= 1 \\ b_0 + b_{-1} &= 1 \\ 2b_{-1} &= 1 \end{aligned}$$

Solving the system yields a_0=1, b_0=1/2, and b_{-1}=1/2. After substituting these values in the ansatz we get

$$q_{k+1} = q_k + \tfrac{h_k}{2}\dot{q}_k + \tfrac{h_k}{2}\dot{q}_{k+1}$$

from where the trapezoidal formula follows after solving for $\dot{q}_{k+1}$.

$$\dot{q}_{k+1} = -\dot{q}_k + \tfrac{2}{h_k}\left(q_{k+1} - q_k\right)$$

Adams-Moulton integration formulae

An Adams-Moulton integration formula of order n is obtained if we choose b_{-1}, ..., b_{n-1}, and a_0 to be the nonzero coefficients in the general integration formula ansatz. The backward Euler formula is in fact the Adams-Moulton formula of order n=1. The trapezoidal formula is the Adams-Moulton formula of order n=2.

Gear integration formulae

These formulae are also referred to as bacward differentiation formulae (BDF). The BDF formula of order n is obtained if a_0, ..., a_{n-1} and b_{-1} are choosen as the nonzero coefficients in the general integration formula ansatz. The BDF formula of order n=1 is in fact the backward Euler formula.

Single-step vs. multistep

If the integration formula involves only values at t_k and t_{k+1} then it is a single-step formula. Such a formula depends only on the last step length (h_k). The backward Euler formula and the trapezoidal formula are single step integration formulae. Because their coefficients depend only on h_k, they can be computed in advance.

Multistep formulae involve values at more that the previously mentioned two timepoints. If the step is constant (i.e. h_k=h_{k-1}=h_{k-2}=...) their coefficients can be computed in advance. Unfortunately in circuit simulators the step varies with the circuit's dynamics. Therefore the coefficients of multistep methods must be recomputed at every timestep. For a formula of order n this involves solving a linear system of n+1 equations (i.e. like we did for the backward Euler and the trapezoidal formula).

How simulators apply numerical integration

A quantity (i.e. capacitor charge, inductor flux) must be numerically integrated to get rid of its

derivative with respect to time and replace it with a weighted sum of past values of the quantity and its derivative. As we already know, the value of the quantity that is about to be computed (at t_{k+1}) also takes part in this weighted sum if the numerical integration formula is an implicit one. For this purpose the simulator stores past values of the quantity and its derivative with respect to time. The length of this storage depends on the type and maximal order of the integration algorithm used by the simulator.

If a multistep algorithm is used the simulator recomputes the integration formula coefficients before the Newton-Raphson algorithm starts solving the circuit for one timepoint. Because the last timestep (h_k) is part of the integration formula ansatz the coefficients don't need to be recomputed if the simulator decides to abandon a solution at a particular timepoint, reduce the timestep, and repeat the Newton-Raphson algorithm for this shorter timestep (this happens when the obtained solution is not accurate enough; we will discuss it later when we introduce local truncation error).

What about explicit integration?

There also exist various explicit integration algorithms. For instance, if we select the nonzero coefficients to be b_0, ..., b_{n-1} and a_0 we obtain the **Adams-Bashforth integration formula of order n**. The forward Euler formula is in fact the Adams-Bashforth formula of order n=1.

$$q_{k+1} = q_k + h_k \dot{q}_k$$

Explicit integration methods tend to produce unstable results (the circuit response explodes) if the timestep (h_k) is greater than the time constant of the circuit's response. Consequently the timestep must be kept small, even when the circuit's response does not change much. This is particularly problematic in "stiff" circuits with multiple time constants, of which one is small and one is large. Such circuits can be analyzed with explicit integration only when the timestep is smaller than the shortest time constant of the circuit. Because every step contributes some numerical error (we will later name it local truncation error) many steps must be computed accumulating a lot of error which in turn can become greater than the response itself.

As a side note let us mention that if we choose as nonzero coefficients a_0, ..., a_n we obtain an **n-th order polynomial extrapolation formula** for computing the value of q at t_{k+1}. The obtained formula is equivalent to constructing an n-th order polynomial interpolation that matches the quantity at timepoints t_{k-n}, ..., t_0 and computing its value at t_{k+1}.

Local truncation error

In the process of deriving an integration algorithm formula of n-th order we assumed that the response (q(t)) is a polynomial of order not exceeding n). Now suppose the response is a polynomial of order exceeding n. As any sufficiently smooth function can be expressed in the form of a Taylor series (i.e. as a polynomial of infinite order) we can expect that the result produced by the integration algorithm will differ from the actual response q(t). Even if the values of q and $\dot{q}$ are known exactly at past timepoints t_k, t_{k-1}, ..., we can expect that the value computed by the integration algorithm at t_{k+1} will differ from $q(t_{k+1})$. This difference is referred to as the local truncation error (LTE). LTE is expressed in terms of q (i.e. for capacitors this is the stored charge and for inductors the stored magnetic flux).

Suppose the exact values of q are given by a Taylor series in the neighbourhood of t_k as

$$q(t) = \sum_{j=0}^{\infty} \frac{q^{(j)}(t_k)}{j!}(t - t_k)^j$$

where $q^{(j)}$ denotes the j-th derivative of q with respect to time. The first derivative of the exact response is

$$\dot{q}(t) = \sum_{j=1}^{\infty} \frac{q^{(j)}(t_k)}{j!} j(t - t_k)^{j-1}$$

The integration algorithm computes an approximation of $q(t_{k+1})$ denoted by q_{k+1}. We substitute $q(t_k)$ and $\dot{q}\ (t_k)$ for q_k and $\dot{q}_{\ k}$ in the integration algorithm ansatz to obtain this approximate value of $q(t_{k+1})$ computed by the integration algorithm.

$$q_{k+1} = \sum_{i=0}^{p} a_i \sum_{j=0}^{\infty} \frac{q^{(j)}(t_k)}{j!}(t_{k-i} - t_k)^j + h_k \sum_{i=-1}^{r} b_i \sum_{j=1}^{\infty} \frac{q^{(j)}(t_k)}{j!} j(t_{k-i} - t_k)^{j-1}$$

We separate the term corresponding to j=0.

$$q_{k+1} = \sum_{i=0}^{p} a_i + \sum_{j=1}^{\infty} \left(\sum_{i=0}^{p} a_i \frac{q^{(j)}(t_k)}{j!}(t_{k-i} - t_k)^j + h_k j \sum_{i=-1}^{r} b_i \frac{q^{(j)}(t_k)}{j!}(t_{k-i} - t_k)^{j-1} \right)$$

and express the sum over j as a series comprising terms with powers of h_k.

$$q_{k+1} = q(t_k) \sum_{i=0}^{p} a_i + \sum_{j=1}^{\infty} \frac{q^{(j)}(t_k)}{j!} \left(\sum_{i=0}^{p} a_i (\frac{t_{k-i} - t_k}{h_k})^j + \sum_{i=-1}^{r} b_i j (\frac{t_{k-i} - t_k}{h_k})^{j-1} \right) h_k^j$$

On the other hand, the exact value of the response at t_{k+1} can be expressed with the Taylor series

$$q(t_{k+1}) = \sum_{j=0}^{\infty} \frac{q^{(j)}(t_k)}{j!}(t_{k+1} - t_k)^j = \sum_{j=0}^{\infty} \frac{q^{(j)}(t_k)}{j!} h_k^j = q(t_k) + \sum_{j=1}^{\infty} \frac{q^{(j)}(t_k)}{j!} h_k^j$$

The LTE is defined as the difference betweeen q_{k+1} and $q(t_{k+1})$. Due to the way we expressed these two terms LTE can be expressed as a series comprising terms with powers of h_k.

$$\epsilon_{k+1} = q_{k+1} - q(t_{k+1}) = \sum_{j=0}^{\infty} C_j q^{(j)}(t_k) h_k^j = C_0 q(t_k) + \sum_{j=1}^{\infty} C_j q^{(j)}(t_k) h_k^j$$

By comparing the last three expressions we get

$$C_0 = -1 + \sum_{i=0}^{p} a_i$$

$$C_j = \frac{1}{j!} \left(-1 + \sum_{i=0}^{p} a_i \left(\frac{t_{k-i} - t_k}{h_k} \right)^j + j \sum_{i=-1}^{r} b_i \left(\frac{t_{k-i} - t_k}{h_k} \right)^{j-1} \right)$$

When we were deriving the equations for computing coefficients a_i and b_i we required $q_{k+1}=q(t_{k+1})$ for q(t) that was a polynomial of order not exceeding n. This is possible only if coefficients C_i are zero for all i=1,2,...,n. In fact, the n equations given by the last two expressions are exactly those that are used for computing coefficients a_i and b_i. For an integration formula of order n the first nozero coefficient in the series expressing the LTE is therefore C_{n+1} which is also referred to as the error coefficient. The series expressing LTE has infinitely many terms. For the purpose of estimating the LTE we keep only the term with the lowest power of h_k.

$$\epsilon_{n+1} \approx C_{n+1} q^{(n+1)}(t_k) h_k^{n+1}$$

Let us compute **the error coefficient for the backward Euler formula** which has n=1 (this implies j=n+1=2). Because the only nonzero coefficients in the formula are a_0 and b_{-1} we have p=0 and r=-1. The error coefficient is therefore

$$C_{n+1} = C_2 = \frac{1}{2!} \left(-1 + a_0 \left(\frac{t_k - t_k}{h_k} \right)^2 + 2b_{-1} \left(\frac{t_{k+1} - t_k}{h_k} \right)^1 \right) = \frac{1}{2}(-1 + 2b_{-1}) = \frac{1}{2}$$

The error coefficient for the trapezoidal formula (n=2, p=0, r=0) is obtained as

$$C_{n+1} = C_3 = \frac{1}{3!}\left(-1 + a_0\left(\frac{t_k - t_k}{h_k}\right)^3 + 3b_{-1}\left(\frac{t_{k+1} - t_k}{h_k}\right)^2 + 3b_0\left(\frac{t_k - t_k}{h_k}\right)^2\right) = \frac{1}{6}(-1 + 3b_{-1}) =$$

Whenever the coefficients of the integration formula are recomputed the error coefficient must also be recomputed. One piece is still missing before we can actually compute the LTE - the n+1-th derivative of q at t_k. We can estimate it by applying polynomial interpolation of order n+1 to the latest n+2 values of q. Because LTE is estimated after the Newton-Raphson algorithm solves the circuit at t_{k+1}, the values of q used in the interpolation are q_{k-n}, ..., q_{k+1}. The n+1-th derivative of obtained polynomial p(t) is constant and does not depend on time. A convenient way for computing it is to use divided differences. Suppose we have n+1 points (x_0 $y(x_0)$), ..., ($x_{n>}$, $y(x_n)$). The Newton interpolation polynomial of order n that interpolates these points can be expressed as

$$N_n(x) = \sum_{i=0}^{n} \alpha_i \eta_i(x)$$

where basis polynomials are defined as

$$\eta_i(x) = \prod_{j=0}^{i-1}(x - x_j)$$

Note that $\eta_0(x)=1$. The coefficients α_i are defined recursively with divided differences as

$$\begin{aligned} \alpha_i &= y[x_0, x_1, ..., x_i] \\ y[x_i] &= y(x_i) \\ y[x_0, x_1, ..., x_i] &= \frac{y[x_1, x_2, ..., x_i] - y[x_0, x_1, ..., x_{i-1}]}{x_i - x_0} \end{aligned}$$

The n-th derivative of $N_n(x)$ can be expressed as

$$\frac{d^n}{dt^n} N_n(x) = n!\alpha_n = n! y[x_0, x_1, ..., x_n]$$

Assuming values q_{k-n}, ..., q_k, q_{k+1} correspond to timepoints t_{k-n}, ..., t_k, t_{k+1} the LTE can be expressed as

$$\epsilon_{n+1} \approx (n+1)! C_{n+1} h_k^{n+1} q[t_{k-n}, t_{k-n+1}, ..., t_{k+1}]$$

Timestep control

The initial approximate solution used by the Newton-Raphson algorithm for solving the circuit at t_{k+1} is the circuit's solution at t_k. If certain conditions are satisfied the solution at timepoint $t_{k+1}$is accepted and the simulator proceeds with computing the response for the next timepoint t_{k+2}. But this is not always the case. The computed timepoint is rejected if

- the Newton-Raphson algorithm fails to converge
- the Newton-Raphson algorithm converges slowly which indicates that the initial approximate solution is far from the obtained solution

If timepoint t_{k+1} is rejected the simulator reduces the timestep h_k and repeats the Newton algorithm at timepoint $t_{k+1}=t_k+h_k$.

Sometimes the circuit's excitations contain discontinuities in their value or derivatives. Timepoints t_b where such discontinuities occur are referred to as **breakpoints**. If a breakpoint lies on the time interval $t_k < t < t_{k+1}$ the timestep is shortened so that $t_{k+1}=t_b$. In case of a breakpoint the simulator

behaves in the same manner as when a timepoint is rejected.

When a timepoint is accepted the new timestep is computed depending on the LTE estimate in such manner that the LTE is kept bounded. This also means that if LTE is small the timestep is increased.

Choosing the order of the integration formula

For the first timepoint an integration formula of order n=1 is used (backward Euler formula) which does not require the values of the derivative of q at past timepoints. Whenever a timepoint is rejected and at every breakpointthe order of the integration formula used in the next run of the Newton-Raphosn algorithm is reduced to n=1.

Whenever a timepoint is accepted the order of the integration formula used for the next timepoint can be increased by 1. The order of the integration formula is limited. In SPICE OPUS when an Adams-Moulton formula is used the maximal allowed order is n=2. For Gear formulae the maximal allowed order is 6.

Predictor-corrector approach to numerical integration

The initial approximate solution used by the Newton-Raphson algorithm for solving the circuit at t_{k+1} is the circuit's solution at t_k. This is assumed to be a good starting point for which we expect the Newton-Raphson algorithm to require only a few iterations to reach convergence. If, however, we have a means for obtaining a better initial approximate solution we can expect even faster convergence.

The method for computing the initial approximate solution is also referred to as the **predictor**. The method used for performing numerical integration is also referred to as the **corrector**.

In SPICE OPUS when an Adams-Moulton formula of order n is used as the corrector the Adams-Bashforth explicit integration formula of order n is used as the predictor, but instead of applying it to components of **q** we apply it to components of the vector of uknowns (**x**). Recall that the nonzero coefficients are a_0 and b_0, ..., b_{n-1}. The prediected solution is then expressed as

$$\mathbf{x}_{k+1}^{\text{pred}} = \mathbf{x}_k + \sum_{i=0}^{n-1} b_i \dot{\mathbf{x}}_{k-i}$$

Because usually the derivative of the circuit's response with respect to time is not available we have to compute it numerically with finite differences (i.e. divided differences of first order).

$$\dot{\mathbf{x}}_{k-i} \approx \frac{\mathbf{x}_{k-i} - \mathbf{x}_{k-i-1}}{t_{k-i} - t_{k-i-1}} = \frac{\mathbf{x}_{k-i} - \mathbf{x}_{k-i-1}}{h_{k-i-1}}$$

When SPICE OPUS uses a Gear formula of order n as the corrector simple polynomial extrapolation of order n is used as the predictor. Note that in our general approach to integration formulae we have to choose a_0, ..., a_n to obtain the coefficients of polynomial extrapolation of order n. The predicted value is then computed as

$$\mathbf{x}_{k+1}^{\text{pred}} = \sum_{i=0}^{n} a_i \mathbf{x}_{k-i}$$

When predictor-corrector numerical integration is used the LTE can be computed in a fairly simple way. The LTE of the predictor can be expressed as

$$\epsilon_{k+1}^{\text{pred}} = \mathbf{x}_{k+1}^{\text{pred}} - \mathbf{x}(t_{k+1}) = C_{n+1}^{\text{pred}} \mathbf{x}^{(n+1)}(t_k) h_k^{n+1}$$

where $\mathbf{x}(t_{k+1})$ denotes the exact response of the circuit. The LTE of the corrector, on the other hand is given by

$$\epsilon_{k+1} = \mathbf{x}_{k+1} - \mathbf{x}(t_{k+1}) = C_{n+1}\mathbf{x}^{(n+1)}(t_k)h_k^{n+1}$$

The difference between the value obtained from the predictor and the value obtained from the corrector (i.e. the result of the Newton-Raphson algorithm) is

$$\mathbf{x}_{k+1} - \mathbf{x}_{k+1}^{\text{pred}} = \epsilon_{k+1} - \epsilon_{k+1}^{\text{pred}} = (C_{n+1} - C_{n+1}^{\text{pred}})\mathbf{x}^{(n+1)}(t_k)h_k^{n+1}$$

By comparing the last two expressions we can see that they differ only by a constant factor. Therefore the LTE can be expressed as

$$\epsilon_{k+1} = \frac{C_{n+1}}{C_{n+1} - C_{n+1}^{\text{pred}}}\left(\mathbf{x}_{k+1} - \mathbf{x}_{k+1}^{\text{pred}}\right)$$

The obtained expression for computing the LTE does not require us to use divided differences and is therefore much simpler to compute.

Circuit optimization

10th Lecture W1, January

We introduce optimization algorithms for finding the minimum of a function of many variables. A short overview of available algorithms is presented. We show how design requirements for a circuit can be formally defined. A designer tunes these requirements by changing parameters of selected elements (design parameters). Constraints are imposed on the design parameters due to the nature of the circuit. These constraints can significantly reduce the number of design parameters. To automate the design process we introduce the cost function which is then minimized by an optimization algorithm to find circuits that satisfy design requirements. A live demonstration of the approach is given.

READ MORE »

About the course

This is a compulsory course in the 1. semester of the Master's degree curriculum "Electronics". The aim is to introduce students to the theoretical background of analog circuit simulation. The course also involves laboratory work in the advanced field of circuit simulation and optimization with SPICE OPUS.

Staff

Lecturer:

prof. dr. Árpád Bürmen

Teaching Assistant:

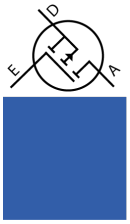
as. Žiga Rojec

Required knowledge

Basics of Electromagnetics Physics Mathematics I, II, III Basics of Programming

Lectures in PDF

Lectures - Circuit Analysis and Optimization.pdf



Page by Rojec, 2018.