Electric Circuit Analysis, KTH EI1120
N. Taylor

Topic 03: Systematic methods

Topic 02 was mainly about ways of manipulating a circuit diagram into a form that is easier for us to solve. That is a bit like rearranging a mathematical expression into a form that you know how to solve.

It is convenient if we don't always have to search for the right steps and "tricks" to get to the solution; there are a lot of simplifications and theorems that you would have to consider for some complicated circuits. Many practical circuits, such as power systems with thousands of nodes, or electronic circuits with thousands of devices, really *need* a systematic method if they are to be solved as whole systems.

Here in Topic 03, a more detailed title would be "Systematic conversion of circuit problems to solvable systems of equations". The word *systematic* suggests several interesting things: we expect a quite simple set of rules, convenient for programming a computer; we also can hope for generality, such that we will be able to find solvable equations for any solvable circuit by following our set of rules. A useful property of systematic methods is that when we know that any of a whole class of circuit diagrams can be converted to particular type of equation system, we can use the properties of these equation systems to form proofs of circuit theorems, such as the Thevenin or Norton equivalents that we come to in Topic 04.

On the bad side, finding a solvable system of equations does not necessarily mean the same thing as finding a simple and convenient set of equations! There may be cases where other methods give simpler calculations, or where a *mixture* of simplifications, systematic methods, and circuit theorems (Topics 02, 03, 04) is best. In some cases the non-systematic methods may also have an advantage by giving us a 'feeling' for the circuit while we handle it, so that we would be good at making intuitive judgements. Intuition is important in some design questions, where the number of possibilities of components and connections is too great for doing an unguided search for a good design.

We will study almost exclusively the systematic method known as *nodal analysis*. Nodal analysis is commonly the core of circuit simulation programs, from microelectronics to power systems. Its dual (complementary) method is *loop analysis*, which is often presented in textbooks in the more restricted form of *mesh analysis*. In this course we do not properly study these other methods, but only mention them to prevent the names from coming as a total surprise in some later studies or work. A little more comparison can be found in the Extra section of this Chapter. Instructions for using mesh analysis are found in most Circuits textbooks.

At this stage in the course, the circuits consist of two-terminal components that are voltage- and current sources (independent or dependent) and resistors. The components are connected in a particular way by nodes. Each component has a voltage and current, and it *partly* determines these by fixing one of them or fixing a ratio between them. Then the connections impose the demands given by Kirchhoff's laws, KCL and KVL.

Given a circuit like this, we could *define* the voltage and current for every component, without thinking whether we actually know the value. Then we could write an equation to describe what each component demands about its voltage and current: for example, a source fixes one of these quantities, and a resistance fixes the ratio. Then we could write KCL and KVL for every node and closed loop that we can find. The resulting equations, for a well defined circuit (without silly things like parallel ideal voltage sources), should then have a solution. By a lot of work, or by giving these equations to a computer, the values of all the voltages and currents could be found.

However, that method would contain a lot of unnecessary work. Some of the equations, such as KVL, could end up being applied in cases that are just linear combinations of each other, unless care is taken to choose just the minimal set of loops. And not all the variables really have to be solved in order to give us a useful solution; this is particularly clear in the quite common case where we are wanting to solve for just one quantity. The main difficulty with big circuits is that one quantity can depend on a lot of components in different places; there can be a lot of interdependence. If we can calculate just the node potentials, for example, then it is trivial to calculate any voltage, and thereby the currents in resistors; it is not really necessary to define and solve every single quantity in the circuit.

One aim of a systematic method should therefore be to produce equations that are *sufficient* to show all necessary information to get a solution, and are also *independent* so that we aren't doing any more work than necessary.

Nodal analysis uses node potentials as the unknowns. All the potentials are defined, and one is chosen as the reference of zero, marked by the 'ground' symbol. Then Kirchhoff's current law is applied at the nodes, producing an equation system in the unknown potentials.

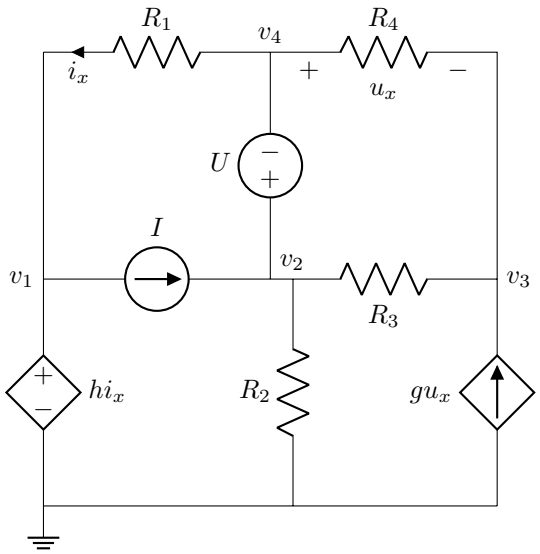
1 Nodal analysis

[More should be put here some day, about starting from smaller cases. We do that a bit in this topic's lecture. For now, in the Chapter we just start with following the rules.]

2 Forming the nodal equations

In the following, we start by showing a simple set of rules for converting circuits to equations. The resulting equations can be many, and even for the quite small circuits that we use as examples it could be tedious to solve the equations by hand. Then we see a more human-friendly approach, that makes some simplifications before and during writing the equations. Finally, it is shown that the simplifications from Topic 02 can be useful as steps before and/or after nodal analysis, when doing the solution by hand. The Extra section gives examples of using a computer to solve the equations.

We will start with the following circuit, as an example that contains independent and dependent voltage and current sources, along with several resistors.



First, we note that the following necessary steps have already been made, to define the node potentials:
* Define one node as the reference potential (earth).
* Define the other nodes' potentials as v_1, v_2, etc.

2.1 Extended ('simple') method

This is a simple way to *write* nodal equations. The number of equations can become quite large: for a circuit with N nodes, V voltage sources, and D controlling variables of dependent sources, this method would give $N - 1 + V + D$ equations and the same number of unknowns. In the above circuit, this is $5 - 1 + 2 + 2$, so we expect 8 equations.

I call this the 'simple' method because each of the equations can be directly written based on the part of the circuit it concerns: for example, we know that one equation is KCL at node n, and another equation is the relation of node potentials imposed by a particular voltage source. That should make it easy to write the equations reliably, easy to update the equations if the circuit is changed, and easy to program a computer to generate the equations from a circuit description.

Sometimes this type of method is called 'extended' because the unknown variables are not just the node potentials but also the currents in voltage-sources, and the controlling variables of dependent sources. If a computer is doing the solution, it's usually not problem to introduce the extra equations and unknowns, and one then gets solutions of these quantities too.

2.1.1 Procedure for Extended method

The following steps can be used to write an equation system corresponding to the circuit:

1. For every marked node *except* the ground node, write Kirchhoff's current law, KCL. Be very clear about which node each KCL equation relates to, and which direction of current you are considering; I usually equate all outgoing currents to zero. In order to write KCL in each node you need to express the current in each branch[1] connecting to that node:

> For resistors, the currents can be found from the node potentials between which the resistor is connected, e.g. $(v_2 - v_3)/R_3$.
>
> For current sources, the current is just the source's value; be careful about the direction. (If it's a dependent current source, the expression for current is not just a constant such as 'I', but an expression containing the controlling variable, e.g. gu_z.)
>
> For voltage sources (independent or dependent) we don't know the current directly. Instead, just *define* a new (unknown) current through the source, to use in KCL: for example, i_α, i_β, etc. Mark the current's name and direction clearly by the source, so you don't make an error with the direction in KCL.

2. For each voltage source, write a further equation that describes the constraint the source puts on the potentials of the two nodes that it connects between. For example, in the circuit above, $U = v_2 - v_4$. In this way, you use the *known* property of a voltage source (its voltage) to compensate for this source having introduced a new *unknown* quantity (its current), and you thereby ensure there are still as many equations as unknowns. This is done for independent or dependent voltage sources: the dependent sources will have a controlling variable in their value instead of a constant, e.g. $hi_x = v_1 - 0$.

3. If the controlling variable of any dependent source is unknown (like u_x and i_x in this example), you must write a further equation that expresses this variable in terms of existing known or unknown variables. For

[1] Branches are the parts joining nodes; in our case here, each component is a branch, but in other cases we might view a group of components as a single equivalent branch if we don't care about the individual voltages and currents between those components.

example, $u_x = v_4 - v_3$. Note how each controlling variable introduces a new unknown but also a new equation, into the equation system.

x. Finally: have a good look at your equations. Compare them with the diagram, to double-check the directions (and values) in each KCL. Identify the unknown variables: check that this number matches the number of equations.

2.1.2 Example: extended method

Now we perform the steps decsribed above, for the example circuit.

Write KCL for each node. Let's sum the currents *out* of each node. Nodes are numbered according to the marked potentials (v_1 etc.). Let's denote the currents in the voltage sources as i_α in the independent voltage source U, and i_β in the dependent voltage source hi_x, with both having their reference direction into the voltage-source + terminal.

$$\begin{aligned}
I + \frac{v_1 - v_4}{R_1} + i_\beta &= 0, \quad \text{KCL}_{(1,\text{out})} \\
-I + \frac{v_2}{R_2} + \frac{v_2 - v_3}{R_3} + i_\alpha &= 0, \quad \text{KCL}_{(2,\text{out})} \\
-gu_x + \frac{v_3 - v_2}{R_3} + \frac{v_3 - v_4}{R_4} &= 0, \quad \text{KCL}_{(3,\text{out})} \\
\frac{v_4 - v_1}{R_1} - i_\alpha + \frac{v_4 - v_3}{R_4} &= 0, \quad \text{KCL}_{(4,\text{out})}
\end{aligned}$$

The above 4 equations have 4 unknown node potentials v_1, v_2, v_3 and v_4.

But they also have an unknown current for each voltage source, i_α and i_β, and an unknown controlling variable for each controlling variable of a dependent source, i_x and u_x. More unknowns than equations is a problem, if we want a unique solution of all the node potentials.

The voltage sources which have caused the problem of unknown currents can also solve this problem. They provide some more information about the circuit, which we haven't yet used. Each voltage source fixes a relation between the potentials of the nodes that it connects to. These relations provide two new equations without any new unknowns.

$$\begin{aligned}
v_2 - v_4 &= U \quad \text{voltage source } U \\
v_1 - 0 &= hi_x \quad \text{voltage source } hi_x
\end{aligned}$$

By looking at how these controlling variables are defined in the circuit, we can express them in terms of other known and unknown variables that already are the equations. Here again, we get new equations (information) without new unknowns.

$$\begin{aligned}
u_x &= v_4 - v_3 \quad \text{definition of } u_x \\
i_x &= \frac{v_4 - v_1}{R_1} \quad \text{definition of } i_x
\end{aligned}$$

Now there are 8 equations and 8 unknowns. Because we used a suitable method for choosing which equations to write, the equations can be expected to be independent provided that the circuit is a sensible one that has a single solution. However, if the equations had been derived by "just searching around the circuit trying to write down some true statements" there would be a real risk of the equations having linear dependence. In that case, 8 equations might turn out to be only 7 truly different equations, and there would not be a unique solution to 8 unknowns. Unless you're sure you have another suitable way, then use a systematic method for choosing the equations!

Some simple checks can (and should) be made immediately after writing the KCL equations. For example, if we always intend to write outgoing currents, then at node n the sign of every v_n/R_x term should be positive, while the sign of every v_m/R_x term ($m \neq n$) should be negative. So $\frac{v_3 - v_2}{R_3}$ is valid at node 3 but not elsewhere. If we choose incoming currents the opposite is true. We can also check that the number of terms at each node matches the numbers of branches connecting to that node.

2.2 Supernode method

This method is a way to avoid caring about the currents in voltage sources. Its advantage is that the number of equations to write down and solve can be reduced. A disadvantage is that currents in voltage sources are not directly solved for, and more conscious thought is needed when writing the equations at the start.

Instead of writing KCL for every separate node, any group of nodes that are connected to each other by voltage sources are treated as a single region; KCL is written for currents going between this region and the rest of the circuit. In this way, the current in any voltage source is just an internal current within one of these groups of nodes, so it is not included in the KCL equation. Sometimes the grouped nodes are referred to as a *supernode*. A supernode is a set of nodes between which one can travel by going through just voltage sources; it can be *two or more* actual nodes.

The trouble with combining nodes like this is that less information is provided by the smaller number of KCL equations; there are fewer equations, so fewer unknowns can be solved. This reduced number of equations is compensated by defining unknown potentials for only *one* node in each supernode. After solving for this reduced number of potentials, the potentials of the unsolved nodes in each supernode can be found by looking at the one solved node potential and the values of the voltage sources that connect the member nodes in the supernode.

By the supernode method we get a simpler equation system, but arguably have to think harder when writing it. The equations provided by the extended method could be simplified to give the same result;

the difference in the methods is whether the equations are simplified before or after writing them down! For solving the equations by hand, the supernode method's early simplification will very likely be a better choice than writing and simplifying more equations. For on a computer by numeric linear-algebra functions, the equations need to be in a neat matrix form such as $Ax = b$; the supernode method can provide suitable equations for its reduced number of nodes. However, if you will put the nodal equations directly into a symbolic computer program, there is no need for a particular layout of the equations. It is then more sensible to use the extended method. The equations can be directly related to the circuit; they are easy to check, and if you change the circuit a bit, it will be obvious how to change the equations. The extended method will also directly provide solutions for the currents in the voltage sources, which in some cases is useful.

2.2.1 Procedure: supernode method

* Make sure a ground node is defined.

* Any set of nodes that are joined by voltage sources is defined as a *supernode*. So: identify the nodes and identify any groups of them that are supernodes.

* For each supernode, define the potential of just *one* of its constituent nodes (the nodes inside it). Write this on the diagram. Then write expressions for the potentials of the other nodes in the supernode in terms of this potential and the source values. For example, in the above question, we see that $v_2 = v_4 + U$, so only v_2 *or* v_4 should be defined as an unknown.

* For each dependent source, define its value in terms of the known quantities and the defined node potentials that you chose in the above step. In the example circuit, it is unusually hard work to do this for i_x, since v_1 depends on hi_x; but you can write this equation and rearrange to get i_x in terms of h, R_1, U and v_4 or v_2 (depending on whether you chose v_4 or v_2 as the defined value within the supernode created by the voltage source U).

* Write a KCL equation at each supernode and each other node (that is not part of a supernode) except for the ground node.
Do *not* try to write KCL for each separate node inside the supernode: that extra information is not needed, as we already know a relation between all the potentials within the supernode.
If a supernode contains the ground node, then you don't need to do KCL on *any* part of that supernode.
For a supernode that does *not* contain ground, write just one KCL equation for all currents out of the whole supernode.

* This should lead to a modest number of equations: in the circuit that we used as an example for the simple method, there would be two KCL equations in two unknown potentials. After solving this pair of equations for the unknown potentials, the other potentials can quickly be found.

2.2.2 Example: supernode method

The supernode method is now used, on the same example circuit.

There are two voltage sources.

The independent source, U, has connections to v_4 and v_2, such that $v_2 = v_4 + U$. A supernode is therefore formed from nodes 2 and 4 together. Only one of those node potentials should be used as a variable: let's choose v_4. The other is expressed in terms of this one: thus the potential at node 2 will be written v_4+U in the KCL equations.

The dependent source, hi_x, has one terminal connected to the ground node, and the other to node 1. Node 1 therefore becomes part of the ground supernode. The potential at node 1 still needs to be known, as nodes 1 and 2 are linked by a resistor, and the current in this will be used in KCL at node 2. The source tells us that $v_1 = 0+hi_x$; but this unfortunately involves an unknown variable, i_x, which we would like to eliminate now so that we can get the minimum number of equations, with just some node potentials as the unknowns. We know that $i_x = (v_4 - v_1)/R_1$, and so $v_1 = (v_4 - v_1)h/R_1$, which is rearranged to $v_1 = v_4h/(h + R_1)$ so that we can use $v_4h/(h + R_1)$ in place of v_1 in the KCL equations.

There are now just two places to apply KCL: one is node 3, and the other is the supernode that combines nodes 2 and 4. Remember that v_1 is part of the ground supernode, and we do not apply KCL to any part of this [super]node. Potentials v_4 and v_3 are the only ones that should appear in the KCL equations.

The two KCL equations, with outgoing currents, are

KCL(2&4):

$$\frac{v_4 - \frac{v_4 h}{h+R_1}}{R_1} + \frac{v_4+U}{R_2} + \frac{v_4+U-v_3}{R_3} + \frac{v_4 - v_3}{R_4} - I = 0$$

KCL(3):

$$\frac{v_3 - v_4}{R_4} + \frac{v_3 - (v_4+U)}{R_3} - g(v_4 - v_3) = 0$$

These can be written more neatly, to group coefficients of unknown node potentials v_3 and v_4,

$$v_3\left(-\tfrac{1}{R_3} - \tfrac{1}{R_4}\right) + v_4\left(\tfrac{1}{R_1} - \tfrac{h}{hR_1+R_1^2} + \tfrac{1}{R_2} + \tfrac{1}{R_3} + \tfrac{1}{R_4}\right) = I - U\left(\tfrac{1}{R_2} + \tfrac{1}{R_3}\right),$$

$$v_3\left(\tfrac{1}{R_4} + \tfrac{1}{R_3} + g\right) + v_4\left(-\tfrac{1}{R_4} - \tfrac{1}{R_3} - g\right) = U\tfrac{1}{R_3}.$$

These two simultaneous equations can be solved, but any manipulation is hindered by the quite long coefficients; if we had numeric values of components it would be much easier, as each coefficient would become a single number.

The eight equations from the extended method could have been put into this two-equation form by

successively substituting one equation into the others to eliminate variables. However, it often feels easier, even more reliable, to make the simplifications in the circuit. That's particularly true when there are several voltage sources, which would make a large number of equations in the extended method, but very few in the supernode method.

3 Combination with simplifications

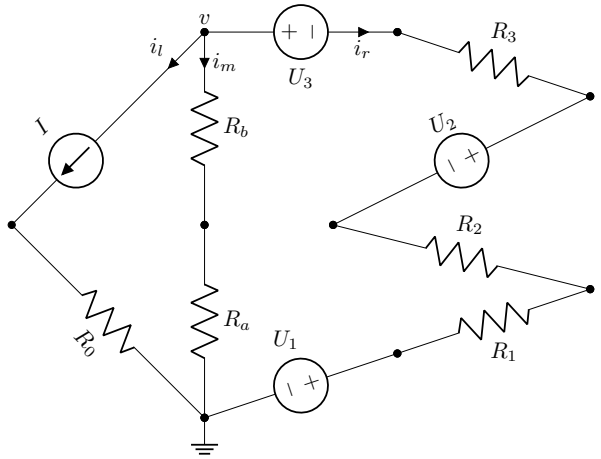
When using nodal analysis 'by hand' it can be useful to apply some simplifications before and possibly afterwards. This is particularly common when the aim is to find one quantity instead of all the node potentials. Simplifications that remove irrelevant components are surely a good idea, and other simplifications may also help in some cases.

3.1 Equivalent branches between nodes

Consider the following circuit, in which the aim is to find the potential v. Component values are assumed to be known, but marked currents (e.g. i_r) are not; they are just definitions.

If we just followed the rigid set of rules for nodal analysis from Section 2.1.1, this circuit might be analysed as having 8 nodes apart from the ground node. That might be good for a simple computer program where the programmer doesn't want lots of "if, then" cases. But an electrically-oriented person would probably prefer to do a little more thinking, to write an easier equation.

If we are solving the problem by hand, and only care about finding the potential v, then we don't really want to calculate potentials at the other 7 nodes: the good features of simplifications *and* nodal analysis can be combined.



Notice that the circuit can be seen as three 'branches' between v and the ground node. If we can describe the current in each branch as a function of the known values of components and the unknown v, then a single KCL equation can be written and solved for node v.

On the left, the branch behaves as a current source: the current is determined by the source, regardless of the irrelevant component R_0,

$$i_\mathrm{l} = I.$$

In the middle, the branch behaves as an equivalent resistor R_ab, where

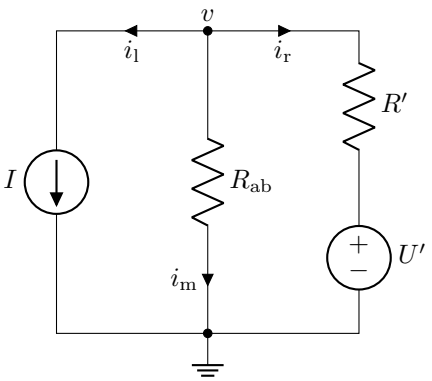
$$i_\mathrm{m} = \frac{v}{R_\mathrm{a} + R_\mathrm{b}} = \frac{v}{R_\mathrm{ab}}.$$

On the right, the branch behaves as a Thevenin source, with voltage $U' = U_1 + U_2 + U_3$ and resistance $R' = R_1 + R_2 + R_3$. The result is that

$$i_\mathrm{r} = \frac{v - U_1 - U_2 - U_3}{R_1 + R_2 + R_3} = \frac{v - U'}{R'}.$$

Note that in an equivalent branch *the order of the components does not matter*. All that matters is the relation between the current in the branch and the voltage across the total branch. It is therefore possible to put U_1, R_1, U_2 etc in any order in the branch, which makes it easier to see that all the voltage sources and all the resistors can be combined. That is obvious if you consider that the voltage across the branch is expressed in terms of the current by $U_1 + i_\mathrm{r}R_1 + i_\mathrm{r}R_2 + U_2 + i_\mathrm{r}R_3 + U_3$, and that addition can be done in any order.

The simplified branches are shown in the following diagram.



The nodal equation KCL(v,out) is

$$I + \frac{v}{R_\mathrm{ab}} + \frac{v - U'}{R'} = 0,$$

which can be rewritten in terms of the given values of the components,

$$I + \frac{v}{R_\mathrm{a} + R_\mathrm{b}} + \frac{v - U_1 - U_2 - U_3}{R_1 + R_2 + R_3} = 0.$$

The only unknown is v, which solves as

$$v = \frac{(U_1 + U_2 + U_3 - I\,(R_1 + R_2 + R_3))\;(R_\mathrm{a} + R_\mathrm{b})}{R_1 + R_2 + R_3 + R_\mathrm{a} + R_\mathrm{b}}.$$

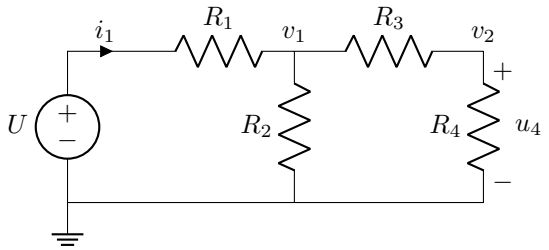
It is useful in such cases to do the algebra with the simplified variables such as U', then substitute the

known values such as $U_1 + U_2 + U_3$ in the very final expression to give the answer.

We could get the same result by defining potentials at all nodes, writing the full node equations from the extended method, and then substituting one equation into another until we end up with the above equation for v. The supernode method would have made this look a bit simpler, but if we applied it in exactly the standard way without simplifying even the left or middle branches, there would still be 4 potentials being solved.

3.2 Simplify before and after

This example is less obviously beneficial than the previous one. It was given in Topic 02, as an example where a solution can be found by repeatedly equivalencing components in series and parallel. Potentials have been defined for nodal analysis; the potential about the source U is clearly U, so it has not been given a further name.



Let's suppose we need to find i_1 and u_4. By the supernode method, which is generally a good way for manual calculation, we write KCL for the two nodes of unknown potentials v_1 and v_2; the other two nodes have known potentials of 0 and U.

$$\begin{aligned} \frac{v_1 - U}{R_1} + \frac{v_1}{R_2} + \frac{v_1 - v_2}{R_3} &= 0 \qquad \text{KCL(1,out)} \\ \frac{v_2 - v_1}{R_3} + \frac{v_2}{R_4} &= 0 \qquad \text{KCL(2,out).} \end{aligned}$$

The solution is

$$\begin{aligned} v_1 &= \frac{R_2(R_3 + R_4)U}{R_1R_2 + (R_1 + R_2)(R_3 + R_4)} \\ v_2 &= \frac{R_2R_4U}{R_1R_2 + (R_1 + R_2)(R_3 + R_4)}, \end{aligned}$$

from which u_4 is directly given as

$$u_4 = v_2,$$

and i_1 is found as

$$i_1 = \frac{U - v_1}{R_1} = \frac{(R_2 + R_3 + R_4)U}{R_1R_2 + (R_1 + R_2)(R_3 + R_4)}.$$

The above was a bit tedious to write, for solving just a couple of quantities in such a simple-looking circuit; it perhaps doesn't feel terribly much easier than the successive reduction of resistors that we could have used in Topic 02.

In order to try simplifications, we can look ahead and see that u_4 could be found from voltage division of v_1. Thus, only v_1 really needs to be solved; node v_2 can be ignored by combining R_3 and R_4 into an equivalent resistor. Writing the single KCL at node v_1,

$$\frac{v_1 - U}{R_1} + \frac{v_1}{R_2} + \frac{v_1}{R_3 + R_4} = 0,$$

which solves to

$$v_1 = \frac{R_2(R_3 + R_4)U}{R_1R_2 + (R_1 + R_2)(R_3 + R_4)}.$$

The current i_1 can be calculated as before, and the voltage u_4 (equal to potential v_2 can be found by voltage division as

$$u_4 = \frac{R_4}{R_3 + R_4}v_1,$$

which does indeed result in the same expression as in the first calculation,

$$v_2 = \frac{R_2R_4U}{R_1R_2 + (R_1 + R_2)(R_3 + R_4)}.$$

> As a joke, we could try mesh analysis, for which this type of circuit is a classic case of two loops. Define a 'mesh current' i_1 clockwise around the left loop, as already indicated on the diagram. Define a new mesh current i_2 clockwise around the right loop. The mesh equations are then
>
> $$\begin{aligned} U &= i_1(R_1 + R_2) - i_2R_2 \\ 0 &= -i_1R_2 + i_2(R_2 + R_3 + R_4). \end{aligned}$$
>
> After solution, we note that i_1 is directly solved, and $u_4 = i_2R_4$. The results are (as we would hope!) the same as by nodal analysis. But the process of solving still doesn't feel easier than the nodal analysis with two equations.

I would say that the method of simplification *and* node analysis was about 'as good as it gets' for this example.

4 – Extra –

4.1 Loop and Mesh analysis

Another form of systematic circuit analysis is *loop* analysis. It is in many ways the dual of node analysis: it defines currents in a set of closed loops around the circuit (instead of potentials at nodes), and obtains equations by KVL around these loops (instead of KCL at the nodes), by expressing resistor-voltages in terms of the unknown loop currents.

It is more common to find descriptions of *mesh analysis*, for hand calculations. Mesh analysis has the same solution principle as loop analysis, but it has a restrictive definition of how the loops are chosen. It requires that the circuit is *planar*, meaning that it can be represented in two dimensions without any parts crossing each other. The meshes are then the smallest loops in this representation of the circuit: i.e. any loop that contains another loop is not treated for KVL. The mesh method is therefore only suited to a subclass of circuits.

The loop method is more general, but defining the right number of indepdendent loops to analyse in a complicated circuit is likely to be hard (by hand) compared to using node analysis. Programming a nodal method is easy and direct, with input describing a list of components along with the nodes that each connects to. The nodal approach is used in basically all circuit calculation programs I'm aware of, for electronics or power. For the dual methods, one would have to work out the sufficient set of loops, or the circuit's meshes.

Node analysis deals with potentials and branch currents, which are all directly associated with the circuit. Loop currents are not necessarily actual quantities in a circuit, but can be subparts of the actual currents. An apt quotation in the old Thomas/Rosa textbook cited in Topic 01 is,

"(The) advantage of the nodal formulation results from the fact that the equations can be more directly correlated with the physical structure of the network than is possible with the mesh formulation."
Hendrik W. Bode, 1945.

Mesh analysis was commonly taught in circuit theory as an equal to node analysis. If one knew both methods, one would choose whether a particular circuit could more easily be solved by the mesh or node method. Typically, one would prefer to have a small number of equations, so a circuit with fewer meshes than nodes would typically be solved by mesh analysis.

The nodal equations typically have lots of reciprocals in the coefficients, such as $\frac{1}{R_1} + \frac{1}{R_2} \cdots$, while the mesh equations have neater coefficients like $R_1 + R_2 \cdots$. This could seem like a disadvantage of the nodal method; but it is not fundamental, and not very important. If one uses conductance instead of resistance, this situation is reversed; computers will happily handle the calculations in any case.

It's good to know what mesh analysis means. It might be an easier method in some problems. You might even prefer it. You are welcome to use it in solutions that allow you to choose your own method. But make sure first that you are good at doing node analysis, with all the types of components that we consider. Don't risk mixing the two, or learning two methods but not to a reliable 'expert' level. Being good at one method is better than being mediocre at several.

Mesh analysis has caused more exam troubles than I care to remember. It has its advantages in some cases, but there is no compelling need to learn it when one has nodal analysis. The small advantage of its being arguably a big quicker to solve by hand in some particular cases is outweighed by extra time taken from getting really *good* at nodal and other analysis, and on confusion between the two. There is a trend in circuits education towards cutting mesh analysis, as a step in focusing the curriculum!

4.2 Solutions by Computer

The following are some examples of how one can use numeric or symbolic equation solvers, or a specific *circuit* solver, to get or check answers.

It is *not* required material in this course.

It is hoped to be stimulating to a few people who might already be getting familiar with using computers for symbolic solutions and linear algebra. The use of computers for checking can be helpful to you even in this course, for confirming your solutions to homeworks and other problems. Familiarity with numerical and symbolic programs is likely to be useful in later courses and in later work.

4.2.1 Observations on Computer-assisted checking

I often use computers for deriving and double-checking results. For example, I try two solution methods and put in the same randomly set input variables to both, to check the solution is the same. I can't claim to be very familiar with using symbolic programs: I know Mathematica for differential equations and integrals, but most I do most other checks numerically.

My experience of Computer Algebra (symbolic) programs is that the results are often very messy. Even the `FullSimplify[]` (Mathematica) or `simplify()` (Matlab) is frequently not in a form that is good compared to hand calculation. But some of this is the user's fault: there are ways to give these programs further clues about what sort of simplification one wants.

For many calculations, the checking can be done purely numerically: for example, if you want to compare two equations that are functions of one variable, then plot them both and look for a difference; or plot the difference; or sum the absolute value of the differences, etc.

The Matlab element-wise operators `.*`, `./` and `.^` are useful for making an equation operate on a whole vector of input, to produce a vector of output. Typically the vector can be time-points or multiple possibilities for a voltage or component-value. But that is getting a little advanced for now.

4.2.2 Numerical result, from arbitrarily-expressed equations

The extended nodal analysis generates a lot of unknown variables and corresponding equations: one per node (except ground), another per voltage source, and another per controlling variable of a dependent source.

Even if we have numeric values for all the unknowns, we can't simply write these equations directly in a simple computer language that only uses the processor's ability to do addition, multiplication etc. Such languages take *known* numbers in an expression on the *right-hand side* of the `=` symbol, and assign the result to a single variable on the *left-hand side*. A typical example is `x = 2*a + b` with `a` and `b` already defined as numeric values.

An invalid example is `0 = 2*a + b - x`, which would be accused of having an invalid *lvalue* (value on the left-hand side). This could however be easily rearranged by us when writing the equation. The bigger trouble here is that our equations are simultaneous: each one can have several unknowns, so it cannot be rearranged to have just one unknown value, on the left. Multiple equations must be solved together.

If the nodal analysis did not include voltage sources, it is quite easy to use the KCL equations to write a matrix of numerical coefficients that can be solved by linear algebra functions available in many computer programs.

However, there now exist more general programs that can handle equations that are expressed in arbitrary form. This can be useful in allowing us to write the equations in exactly the way that we found from looking at the circuit. Then they can easily be double-checked and easily be modified if the circuit is modified.

In the following, a symbolic solver (from Matlab R2012b) is used to give a numeric result from the original equations that we derived for the circuit in Section 2.1.2. The main part is clearly the `solve()` function, which you can look up in the help. The equations are given as arguments with the == symbol, then the unknowns to solve for are listed.

```
syms   v1 v2 v3 v4 ia ib ux ix   real  % define as real symbolic variables
U = 20.0; I = 5.0; h = 4.0; g = 6.0;
R1 = 10.0; R2 = 12.0; R3 = 15.0; R4 = 9.0;
s = solve( ...
        0 == I + (v1-v4)/R1 + ib, ...
        0 == -I + v2/R2 + (v2-v3)/R3 + ia, ...
        0 == -g*ux + (v3-v2)/R3 + (v3-v4)/R4, ...
        0 == (v4-v1)/R1 - ia + (v4-v3)/R4, ...
        U == v2 - v4, ...
        h*ix == v1 - 0, ...
        ux == v4 - v3, ...
        ix == (v4-v1)/R1, ...
        v1, v2, v3, v4, ia, ib, ux, ix );
% the "structure" s now contains "fields" giving the solutions;
   s.v1,  s.v2,  s.v3,  s.v4
% because we're using the symbolic toolbox, the results such
% as s.v1 are still "perfect precision", treating the numerical
% values as exact: so they appear as fractions
   6800/1807    59940/1807    24190/1807    23800/1807
% to convert them into the normal numeric values on
% the computer (double-precision floating-point), we do
  double(s.v1), double(s.v2), double(s.v3), double(s.v4)
% this gives node potentials 1--4 as:
   3.7631    33.1710     13.3868     13.1710
% other variables can be shown similarly,
  double(s.ia)
    0.9168
```

This way of writing the equations relies on a quite new version of the Matlab symbolic toolbox; for example, it works in R2013b, but not R2009b.

4.2.3 Solve for symbolic result

The Matlab symbolic toolbox can be used without even giving numbers for the components' values; it then returns expressions defining the solved quantities such as v_1 and i_x, in terms of the component values such as R_1, U, and g. These expressions can be used to check our own hand-written ones, or as an alternative, or they can have numeric values substituted into them. Unfortunately, the expressions are often rather long and complicated, even if neater ones could be found.

The following is the same set of equations as before, but now in a different format that works in the older symbolic toolbox found in Matlab R2009b. The symbolic expressions in the code below are given as strings (i.e. text, within quotation marks `'...'`), which means that the variables do not all have to be declared with the `syms` command before calling `solve()`, Only single `=` symbols need to be used within the strings. The list of variables to solve for is given as a single string in the last argument to `solve()`. These older versions of Matlab interpret the symbolic variable `I` automatically as being the imaginary unit: $\sqrt{-1}$. We therefore have to write the current source as `I_`.[2]

This time we also change the situation by calling `solve()` *without* having given any numeric definitions of parameters. All variables that we are not asking it to solve for are assumed to be symbolic variables: in the terminology we've used in the course, these would be our known variables. If we wanted this type of solution in the previous example (with newer Matlab) the line `U = 20.0; ...` etc, could be replaced with `syms U I ...` etc., so as to define the component values as symbolic variables instead of numbers.

```
s = solve( ...
        '0 = I_ + (v1-v4)/R1 + ib', ...
        '0 = -I_ + v2/R2 + (v2-v3)/R3 + ia', ...
        '0 = -g*ux + (v3-v2)/R3 + (v3-v4)/R4', ...
        '0 = (v4-v1)/R1 - ia + (v4-v3)/R4', ...
        'U = v2 - v4', ...
        'h*ix = v1 - 0', ...
        'ux = v4 - v3', ...
        'ix = (v4-v1)/R1', ...
        'v1, v2, v3, v4, ia, ib, ux, ix' )
% looking inside one of the solutions shows the LONG expression:
s.v4
   -((R1 + h)*(R3*U + R4*U - I_*R2*R3 - I_*R2*R4 + R2*R4*U*g + R3*R4*U*g
- I_*R2*R3*R4*g))/((R1 + R2 + h)*(R3 + R4 + R3*R4*g))
% will the simplify() function give a reduction in length?
simplify(s.v4)
 [unfortunately not .. just as long!]
% now try putting in the numbers
U = 20.0; I_ = 5.0; h = 4.0; g = 6.0;
R1 = 10.0; R2 = 12.0; R3 = 15.0; R4 = 9.0;
% s.v1 still is a symbolic equation; the "subs" function substitutes
% the numeric values, assuming them to be exact,
subs(s.v1)
   6800/1807
% convert to normal double-precision computer-numbers
double(subs(s.v1)), double(subs(s.v2)), double(subs(s.v3)), double(subs(s.v4))
    3.7631    33.1710   13.3868    13.1710
```

The result for each solved variable, such as `s.v2`, is then an expression the defines the solution in terms of the parameters. This expression is symbolic, and can be used in further calculations in the symbolic toolbox.

[2] It's rather silly that Matlab has ever had `i` or `I` or `j` to mean an imaginary unit. In symbolic work this causes trouble by surprising us when we try to use it for something else ... no warning is given. In numeric work, someone might use these variables as the imaginary unit, but overwrite the variable with another value: a classic case is when using a_i notation for subscripts, `a(i)`. When using Matlab, always write the imaginary unit as `1i` or `1j`; as this starts with a number it can never be confused with a variable. Scilab manages nicely by having special things like π and `j` defined with a special prefix, e.g. `%i`, so they can never be redefined by treating them as normal variables in an assignment.

4.2.4 Or, for Mathematica users ...

All the same things as above can — of course — be done in Mathematica. Matlab was originally a numeric matrix-algebra interface, with a language for this focus; it has then grown into further directions. Mathematica was from its outset designed to be very versatile for handling a wide range of symbolic manipulations, useful for mathematicians in their proofs as well as for natural scientists and engineers in their calculations. Mathematica has therefore a significantly different syntax.

A symbolic solution and attempted simplification can be done in the way shown on the right. The result is not given any name, so the solutions for the 8 solved variables will be shown on the screen.

```
FullSimplify[  Solve[
          0 == Isrc + (v1-v4)/R1 + ib
      &&  0 == -Isrc + v2/R2 + (v2-v3)/R3 + ia
      &&  0 == -g*ux + (v3-v2)/R3 + (v3-v4)/R4
      &&  0 == (v4-v1)/R1 - ia + (v4-v3)/R4
      &&  Usrc == v2 - v4
      &&  h*ix == v1 - 0
      &&  ux == v4 - v3
      &&  ix == (v4-v1)/R1 ,
          { v1, v2, v3, v4, ia, ib, ux, ix }
]  ]
```

Alternatively, by putting `s :=` before the `FullSimplify` at the beginning, the whole solution could be assigned to the convenient name `s`, by which it could be called within later manipulations.

The symbolic results after the `FullSimplify` are formatted on the screen as shown on the right.

```
In[1]:=
FullSimplify[
 Solve[0 == Isrc + (v1 - v4) / R1 + ib && 0 == -Isrc + v2 / R2 + (v2 - v3) / R3 + ia &&
   0 == -g * ux + (v3 - v2) / R3 + (v3 - v4) / R4 && 0 == (v4 - v1) / R1 - ia + (v4 - v3) / R4 &&
   Usrc == v2 - v4 && h * ix == v1 - 0 && ux == v4 - v3 && ix == (v4 - v1) / R1,
  {v1, v2, v3, v4, ia, ib, ux, ix}]]
```

```
Out[1]= {{v1 → (h (Isrc R2 (R3 + R4 + g R3 R4) - (R3 + R4 + g R2 R4 + g R3 R4) Usrc)) / ((h + R1 + R2) (R3 + R4 + g R3 R4)),
  v2 → ((Isrc (h + R1) R2 (R3 + R4 + g R3 R4) + R2 (R3 - (-1 + g (h + R1)) R4 + g R3 R4) Usrc) / ((h + R1 + R2) (R3 + R4 + g R3 R4))), v3 →
  ((Isrc (h + R1) R2 (R3 + R4 + g R3 R4) - (-R2 R4 + (h + R1) (R3 + g (R2 + R3) R4)) Usrc) / ((h + R1 + R2) (R3 + R4 + g R3 R4))),
  v4 → ((h + R1) (Isrc R2 (R3 + R4 + g R3 R4) - (R3 + R4 + g R2 R4 + g R3 R4) Usrc)) / ((h + R1 + R2) (R3 + R4 + g R3 R4)),
  ia → (Isrc R2 (R3 + R4 + g R3 R4) - (h + R1 + R2 + R3 + R4 + g (R2 + R3) R4) Usrc) / ((h + R1 + R2) (R3 + R4 + g R3 R4)),
  ib → -(Isrc (h + R1) + ((R3 + R4 + g R2 R4 + g R3 R4) Usrc) / (R3 + R4 + g R3 R4)) / (h + R1 + R2),
  ux → -(R4 Usrc) / (R3 + R4 + g R3 R4), ix → (Isrc R2 - ((R3 + R4 + g R2 R4 + g R3 R4) Usrc) / (R3 + R4 + g R3 R4)) / (h + R1 + R2)}}
```

The final lines show the result for just v_1 (which evaluates to the expected 3.7631 if the component values used in the other examples are inserted). There are other long symbolic expressions for other potentials too.

If the solution has been given a name such as `s` then the `/.` operator can be used to substitute a set of further equations into this symbolic solution. By substituting numeric values of the known variables, the numeric values of the solution will be given.

```
snum = s /.  {Usrc -> 20.0, Isrc -> 5.0, h -> 4.0, g -> 6.0,
   R1 -> 10.0, R2 -> 12.0, R3 -> 15.0, R4 -> 9.0 }

  {v1 -> 3.76314, v2 -> 33.171, v3 -> 13.3868, v4 -> 13.171,
    ia -> 5.94079, ib -> -0.916805, ux -> -0.215827, ix -> 0.940786}}
```

The equations and numbers could instead be stated all in one go, for an immediate numeric output.

```
Solve[  0 == Isrc + (v1-v4)/R1 + ib
    &&  0 == -Isrc + v2/R2 + (v2-v3)/R3 + ia
    &&  0 == -g*ux + (v3-v2)/R3 + (v3-v4)/R4
    &&  0 == (v4-v1)/R1 - ia + (v4-v3)/R4
    &&  Usrc == v2 - v4
    &&  h*ix == v1 - 0
    &&  ux == v4 - v3
    &&  ix == (v4-v1)/R1 ,
       { v1, v2, v3, v4, ia, ib, ux, ix } ]   /.
   {   Usrc -> 20.0, Isrc -> 5.0, h -> 4.0, g -> 6.0,
       R1 -> 10.0, R2 -> 12.0, R3 -> 15.0, R4 -> 9.0  }
```

4.2.5 Numerically, on well-arranged equations

A common way of solving linear circuits is numerically, by putting in the known numbers to a matrix equation and then solving by elimination. The equation system that we get directly from nodal analysis needs to be arranged into a neat form of, for example, `A*x = b`, in order to be solved with simple numerical linear algebra functions.

With just resistors and independent current sources it would be easy to write the equations in this form. The addition of voltage sources and dependent sources necessitates a bit more thought. The supernode method is one useful way for setting up suitble equations even when voltage sources are included. Its equations use just a subset of the potentials as the unknowns.

We show the use of this method below, using the equations from the example of the supernode method in Section 2.1.2, and choosing the same arbitrary values for the components as have been used in the other computer-solution examples. Notice the choice of defining every variable as a number and name, then forming the matrix based on names. This makes it easy to change a number and re-run the program.

```
% define the known variables (component values)
U = 20.0; I = 5.0; h = 4.0; g = 6.0;
R1 = 10.0; R2 = 12.0; R3 = 15.0; R4 = 9.0;

% form the known arrays
A = [  -1/R3 - 1/R4,        1/R1 - h/(h*R1+R1^2) + 1/R2 + 1/R3 + 1/R4 ;
        1/R4 + 1/R3 + g,   -1/R4 - 1/R3 - g  ];
b = [  I - U*( 1/R2 + 1/R3 ) ;
        U/R3  ];

% these now have the following values:
%  A = [
%    -0.17778   0.33254
%     6.17778  -6.17778 ]
%  b = [ 2.0000
%        1.3333  ]

% solve (the "proper" way by elimination and back-substitution)
x = A\b;
% alternatively, "less efficient", by full matrix inversion,
%x = inv(A)*b;

% extract the two potentials from the solution vector
v3 = x(1);
v4 = x(2);

% calculate the other potentials in the supernodes
v1 = v4*h/(h+R1);
v2 = v4 + U;

% this gives
  v1,  v2,  v3,  v4
    3.7631    33.1710   13.3868    13.1710
```

Circuit-solver

One quicker way of checking (or making) a solution for more complicated circuits is a specific circuit-solver. There are many graphical-based solvers available nowadays.

SPICE (Simulation Program with Integrated-Circuit Emphasis) is very well known and very versatile, with a long history. Various interfaces and extra component-libraries exist for it, often proprietary. The basic program, however, works on text-based input and output. Its main strengths are in analysis of circuits that contain semiconductor devices such as transistors and diodes (highly nonlinear); however, it can also be used for our simpler cases of linear circuits.

This following text is a SPICE 'netlist' that specifies the circuit used as the example in Section 2 and in the above computer solutions.

```
EI1110_HT14_HW04
V1      2  4   DC   20.0
I1      1  2   DC    5.0
V41     5  1   DC    0
H1      1  0   V41   4.0
G1      0  3   4  3  6.0
R1      4  5        10.0
R2      2  0        12.0
R3      3  2        15.0
R4      4  3         9.0
.OP
.PRINT DC V(0) V(1) V(2) V(3) V(4)
.END
```

The first column shows the type of component and its name. The second and third columns show the nodes that the component connects between; some components in SPICE have more than just two terminals. The final column gives the component's value. Other columns may be used, such as for specifying the controlling variables of dependent sources.

An extra voltage source of zero voltage had to be added here in series with `R1` in order to define the current marked as i_x in the circuit diagram, which controls a dependent source! SPICE uses the extended nodal analysis that solves for voltage-source currents as unknowns; it therefore can use this method of a 'fake' voltage source as way to measure a current. A voltage can more easily be measured as simply a difference between node potentials.

Running the above in Spice (spice 2g.6, from 1983-03-15!) gives node potentials of:
`(1) 3.7631  (2) 33.1710  (3) 13.3868  (4) 13.1710`
Fortunately, this agrees with our earlier calculations... Of course, SPICE can solve much more than our simple nodal analysis, as its main purpose is solutions of circuits with hundreds of highly nonlinear multiterminal devices like transistors.

Command-line SPICE programs are freely available, and various graphical interfaces also exist. There's an online solver at `[NGspice]`. If the above example is entered as text, nothing will show in the plot (no plot is requested in the text), but you can see the text output, including the numbers, by clicking the `Raw output file` option below the plot.

This `[Spiceoverview]` is a good source for understanding the format and the different components. Each of the main lines specifies a component. The first letter shows what type of component, the columns 2 and 3 show the nodes that the + and − terminals connect to (even a resistor has + and −, so that its current and voltage can be easily defined). The voltage source `V41` was added in series with `R1` to allow the current in `R_1` to be measured as the input to the current-controlled source `H1`.