SEQUEL: using the GUI

Mahesh B. Patil
Department of Electrical Engineering
Indian Institute of Technology, Bombay
Mumbai-400076, India
e-mail: mbpatil@ee.iitb.ac.in

Contents

1 Introduction

The purpose of the SEQUEL GUI is to make it easier for the user to enter the circuit schematic, specify analysis options, and view the simulation results. Using the GUI involves the following steps:

(a) entering the circuit schematic

(b) preparing the solve sections

(c) generating the circuit file

(d) running SEQUEL on the circuit file

(e) viewing the results

When the GUI is started, one gets the display shown schematically in Fig. 1. Of the various

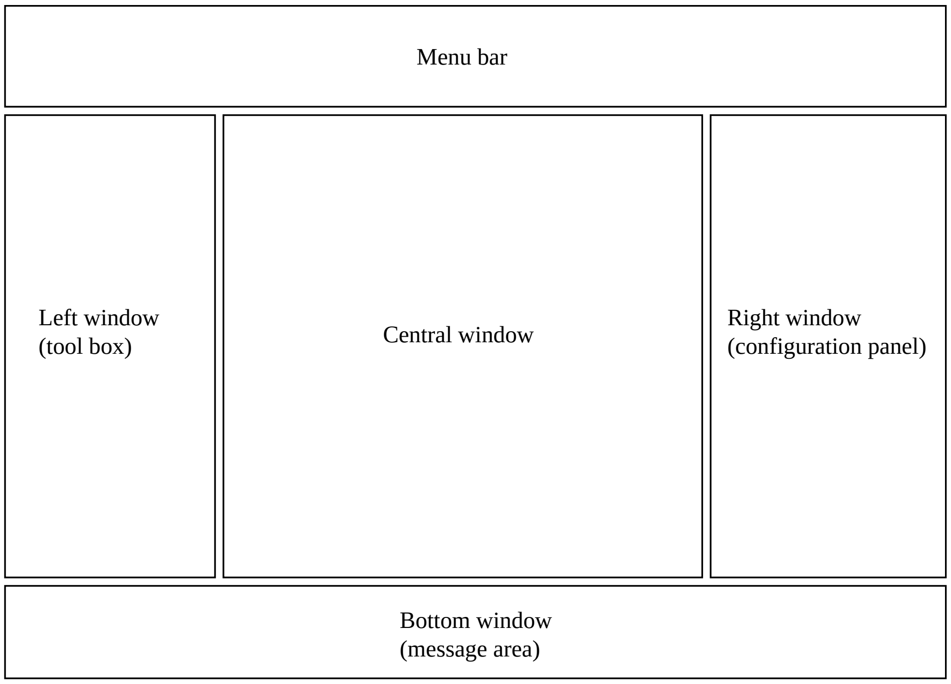


Figure 1: Schematic diagram of the GUI window.

windows, the menu bar remains fixed, but the contents of the other windows may change, depending on the context. For example, when the "circuit editor" menu is selected, a canvas for drawing the circuit schematic appears in the central window. When "graphs" is selected, plotting options appear in the central window, and so on.

The best way to get started with the GUI is to run an existing example and view the results, by following the steps below:

(i) Click[1] on the "open project" menu. A new window will appear, showing the directory structure and files in the current directory, Select the directory[2] `electronics_gnrl`. The SEQUEL projects available in this directory (each with extension `.sqproj`) will appear in a seprate window. Click on the project `rc1.sqproj`.

(ii) Click on the "circuit editor" tab to view the circuit schematic. You may single-click on any of the components to view its parameters in the right window, by clicking on the tab "property editor" there. You could also view the list of output variables that have been selected for this project, by clicking on the "output variables" tab in the right window.

(iii) Click on the "solve blocks" tab to view the details of the analyses which have been requested for this circuit. In the "output blocks" of the solve sections, you will find a list of output variables, a subset of those you would have seen in (ii) above. On execution, SEQUEL would make these variables available in the file specified in the output block. The output file is written to the directory `/SequelGUI2_Release/SGUI/Output`.

(iv) Click on the "circuit file" tab and select "generate CF." The circuit file corresponding to the circuit schematic and solve sections seen in (ii) and (iii) will appear in the central window. The circuit block in the circuit file (between `begin_circuit` and `end_circuit`) corresponds to the circuit schematic, and the solve blocks (between `begin_solve` and `end_solve`) to the solve sections. The circuit file may also be written to the hard disk by clicking on the "save CF" button; however, it is not required for running SEQUEL.

(v) Run SEQUEL on the circuit file generated in (iv) by clicking on the "run solver" button of the menu bar[3]. The messages that the solver may generate while executing the circuit file are displayed in the bottom window and can be seen by clicking on the "solver output" tab of the bottom window. The messages include matrix size, iteration number (for transient analysis), etc., and error messages, if any, from the solver. When execution is completed, you will see the message, "SEQUEL: program completed." In this particular example (`rc1.sqproj`), which is a small circuit, the execution is very quick, and you will see the program completion message instantaneously.

When the program is completed, the output data requested by the user are written to the output files, as described in (iii).

[1]By "click," we would generally mean "left-click," unless specified otherwise.

[2]The exact path for this directory will depend on where the SEQUEL GUI has been installed. This comment also applies to other directories mentioned in this document.

[3]The "run solver" button generates the circuit file and then invokes the simulation engine. In that sense, the "generate CF" step described above is not really required separately.

(vi) Now click on the "graphs" button. The central window will turn into a three-column format. In the left column, you will see the names of the output files requested in the solve blocks (see (iii) above). When you select the desired file, the names of the variables contained in that file will appear in the second and third columns. One of the variables from the second column can be selected as the x-axis, and one or more from the third column as the y-axis variable(s). Clicking on the "graph it" button would plot the requested graph. If another plot is desired, click on the "back" button, and repeat the above procedure.

2 A Tutorial Example

Having seen the basic functioning of the GUI, we will now construct a SEQUEL project from scratch. In particular, we will simulate the circuit shown in Fig. 2, and plot i, V_1, V_2 versus time.

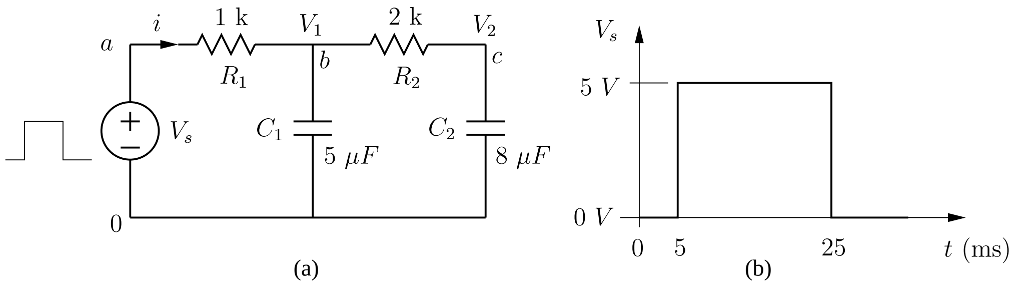


Figure 2: RC circuit for the tutorial example.

2.1 Creating the circuit block

SEQUEL allows the following types of elements:

(i) electrical compound elements (`ece`)

(ii) digital compound elements (`dce`)

(iii) general compound elements (`gce`)

(iv) thermal compound elements (`tce`)

(v) general mixed elements (`gme`)

In the circuit of Fig. 2, only elements of type `ece` are involved, viz., resistors (`r.ece`), capacitors (`c.ece`), and a pulse voltage source (`vpulse.ece`). Click on the circuit editor tab. The central window will turn into a canvas for drawing the circuit schematic, and a list of the available types of elements will be displayed in the left window. Expand the electrical element list (`ece`) and click on `r.ece`. Take the cursor to some location on the canvas and double-click. A resistor will appear in the canvas.

Select the "property editor" option in the right window. Now, go back to the canvas and single-click on the resistor. The resistor will get highlighted, and a list of properties of `r.ece` will appear in the right window. Change its name[4] to `r1` and its resistance (which appears as a real parameter in the list) to `1k`.

Following the above procedure, create the other required elements, and edit their properties as desired. For the voltage pulse, we can let the pulse start at $t_1 = 5$ ms and end at $t_1 = 25$ ms. The relevant parameters are therefore `t_1=5m`, `t_2=25m`, `v_1=0`, and `v_2=5`. The parameters `delt_1` and `delt_2` represent the rise/fall time of the pulse and can be set to a suitable (small) number, say, `10u`.

Rotate the capacitors clockwise through 90^o. This can be done by selecting the element and typing `r` or by setting "rotation" to 90^o in the property editor window.

Place the elements suitably by either (a) clicking and dragging or (b) clicking, releasing the mouse button, and moving the selected element with the arrow keys. Note that, if an element is selected (i.e., highlighted), it can be de-selected by simply left-clicking on an empty area of the canvas.

We are now ready to make the required connections. To connect `node 1` to `node 2`, click on `node 1`, release the mouse button, take the cursor to `node 2` or, if necessary, to a grid point where a bend in the wire is desirable. Continue this process until you arrive at node 2. The GUI will automatically exit the wiring mode when you click on `node 2`. If you are in the middle of making a connection and would like to abandon the present wire (without completing the connection), press F4, and the GUI will exit the wiring mode.

After completing the connections, we need to inform SEQUEL about the node that needs to be designated as the "reference node." For this purpose, bring the element `ground.ece` on the canvas and connect it to the desired wire or element node.

As with element names, the GUI generates a default name for each node. The program would run with the default names. However, for convenience, the user may want to assign node names which are more meaningful. This can be done in two ways:

[4]The GUI generates a default name for each element, and that is good enough for running SEQUEL. However, for easy identification, it may be desirable to assign familiar names to some of the elements.

(i) Click on the wire (node) and edit its name in the right window (make sure that the "property editor" tab is selected in the right window).

(ii) Click on an element to which the node is connected and edit the node name for that element in the right window.

If a node name is changed in the above manner, the node assignments for *all* elements sharing that node are updated simultaneously.
For the circuit in Fig. 2, we can assign the node names `a`, `b`, `c`, `0` as shown in the figure. A convenient way to figure out the name (default or assigned) of a node or an element is to keep the cursor (*without* clicking a mouse button) on the concerned node (or element) for two seconds or so. The name will then appear in a box near the cursor[5]. You can check this out for your circuit in the canvas.
The final step in completing the circuit description is definition of the output variables. In our example, we want to define the current through R_1, and node voltages at `b` and `c` as output variables. This can be done as follows[6].

(i) Click on the "output variables" tab in the right window and select "add variable." The GUI is now in the "add output variable" mode. Click on the resistor `r1`. A menu will appear, showing `i1`, `v1`, etc. Select `i1`. The GUI will automatically exit the "add output variable" mode.

(ii) Enter the "add output variable" mode and click on node (wire) `b`.

(iii) Enter the "add output variable" mode and click on node (wire) `c`.

As you execute the above steps, you will notice that the GUI has generated the following lines in the right window:

```
var1=i1_of_r1
var2=nodev_of_b
var3=nodev_of_c
```

The names `var1`, `var2`, `var3` have been generated by the GUI as default names. You could change these to more meaningful names by double-clicking on the concerned line and typing the desired name. For example, we could use `i1`, `v_b`, `v_c`, respectively, instead of `var1`, `var2`, `var3`.

[5]If it is an element, the element type will also appear in the box.

[6]There is also an "expert" add variable mode which one can enter by clicking on the "Output variables" tab while keeping the `cntrl` key pressed. In that case, the user can keep adding output variables one after another and finally, click on the "Add variable" button once again to exit the expert add variable mode. Exiting the mode is rather important!

2.2 Creating the solve blocks

We have now completed the circuit to be simulated. Next, we must inform the simulator about the analyses that should be performed on the circuit. That is the purpose served by the solve sections.

For the circuit of Fig. 2, there are two analyses that we would like to perform:

(a) Start-up analysis to obtain the solution at $t = 0^-$.

(b) Transient analysis to obtain the solution for $t > 0$.

Click on the "solve blocks" button. The central window can now be used for creating and editing solve blocks, as follows:

(a) Click on the "add solve block" button. This will display a solve block. Make sure that the "property editor" tab is selected in the right window. Click on the title bar of the solve block. In the right window, you can now select the type of the solve block (`Start_Up`) in the drop-down menu that appears after clicking on the bar showing "DC." Next, we need to select an option for the initial solution to be used for this block. Do this by dragging `initial_sol` from the left window into the solve block. The default value for `initial_sol` is `initialize`, and it is appropriate for this solve block.

(b) Add another solve block and change its type to `transient`. Drag `initial_sol` from the left to the current solve block in the central window, click on it, and select `previous` in the right window[7]. This will add the assignment `initial_sol=previous` in the solve block. Drag `back_euler` and set it to `yes`, by clicking on it in the right window and then toggling the square box there. Drag `t_start` and set it to `0` in the right window. Using the same procedure, make the following assignments:

```
t_end=100m
delt_const=10u
```

The above choices will instruct SEQUEL to perform transient analysis from $t_{\text{start}} = 0$ to $t_{\text{end}} = 100$ ms, with a constant $\Delta t = 10\ \mu$s, using the backward Euler method. We now need to specify the names of the output files and the output variables to be written to each of the output files. Say, we decide to write the node voltages `v_b` and `v_c` to a file called `rc_1.dat` and the current `i1` to another file called `rc_2.dat`. This can be specified as follows.

[7]Many of the options for solve blocks are specified in this manner, viz., by dragging a keyword from the left window to the central window, clicking on it, and then selecting an appropriate option in the right window.

Drag “output block” from the left window to the central window, and click on it. The right window will show the various options that may be set for the output block. Set `filename` to `rc_1.dat`. Click on the “output variables” field. This will display a list of the output variables (which we defined while creating the circuit block). Select `v_b` and `v_c`. Similarly, create another output block with `filename=rc_2.dat` and select the output variable `i1` for that block.

2.3 Executing the program and viewing plots

The circuit file can now be created by clicking on the “circuit file” tab and then on the “generate CF” button in the central window. The circuit file corresponding to the circuit schematic and solve blocks you have created will appear in the central window. While it is instructive to look at the circuit file, it is not necessary for the *user* to generate it since the GUI will automatically generate it when the “run solver” option is used.
Run SEQUEL on the circuit file thus generated, by clicking on the “run solver” button. Click on the “solver output” tab in the bottom window and make sure that the message, “SEQUEL: program completed” appears there. Click on the “graphs” tab, and view the plots of `v_b` and `v_c` with respect to time, following the procedure explained in Sec. 1.
The Graph interface allows quantities to be plotted with respect to different y-axes. This feature is useful when two quantities of different orders of magnitude are to be plotted together, e.g., a voltage varying from $0\ V$ to $5\ V$, and a current varying from $-1 \times 10^{-3}\ A$ to $2 \times 10^{-3}\ A$. The user can choose the left y-axis for the voltage and the right y-axis for the current in that case by simply selecting the appropriate box for the y-axis.
Exercise: Add the node voltage `v_a` to the output file `rc_1.dat` (i.e., the first output block), generate the circuit file, run SEQUEL, and plot `v_a`, `v_b`, `v_c` with respect to time on the same plot.

3 Creating the circuit block: general tips

The previous exercise of creating an example from scratch has made the reader familiar with some of the operations and features allowed by the GUI. Now, we describe these in more detail, for general reference. We will start with the circuit block and assume in the following that the GUI is already in the “circuit editor” mode.

3.1 Global parameters

There are times when one would like to assign the same number to parameters of different elements and maybe even to some parameters in the solve sections. In such cases, one can define a “global parameter” (integer or real), and assign that to the parameter of interest rather than assigning hard numbers. As an example, consider the Op Amp summer shown in Fig. 3, which is designed to have the same gain for all three inputs. Therefore, if we change,

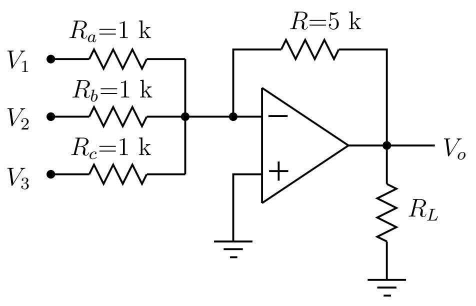


Figure 3: An Op Amp summer circuit.

say, R_a from 1 k to 2 k, we would like R_b and R_c also to change similarly. This can be achieved by using a global parameter, as follows.
Right click on an empty location in the canvas, select “add global variables,” and type `r1` as its name. Click on an empty location in the canvas (where there are no elements or wires). Select the “property editor” tab in the right window. The right window will display a list of global parameters (which currently has only one entry, `r1`). Assign `1k` to `r1` in the right window. Click on the resistor R_a in your circuit, and edit the real parameter `r` (its resistance). Click on the right half of this field, and a drop-down menu will appear. Choose `r1`. This makes the assignment `r=r1`. Make the same assignment for R_b and R_c as well. If the global parameter `r1` is now changed to 2 k, this will apply to each of R_a, R_b, R_c through the global parameter assignment we have defined.

3.2 Expression evaluation

It is often convenient to define a parameter of an element in terms of the parameter(s) of other elements. As an example, consider the R-$2R$ ladder circuit shown in Fig. 4.
In this circuit, we would like to define R_{1a} = R_{1b} = R_{1c} = R, say, 5 Ω, and R_{2a} = R_{2b} = R_{2c} = R_2 = $2R$. This can be achieved by using the following two statements:

```
greal r=5
greal 2r=$((r)*(2))
```

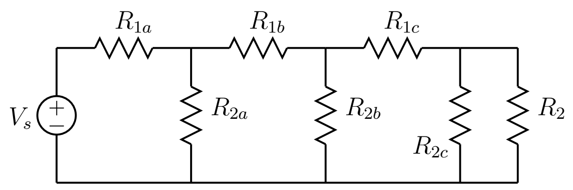


Figure 4: R-$2R$ ladder circuit.

and then assigning `r` to the resistance of R_{1a}, R_{1b}, R_{1c}, and `2r` to R_{2a}, R_{2b}, R_{2c}, R_2. The above statements define the global parameter r to be 5 Ω, and `2r` to be 10 Ω.
Two forms of expression evaluation are allowed:

(a) `greal_name=$(string1(string2))` where `greal_name` is the name of a global real parameter. `string1` can be one of `exp`, `sin`, or `cos`. `string2` can be a real number or the name of a previously defined global real parameter. For example,
```
greal evt=$(exp(vt))
```

(b) `greal_name=$((string1)string2(string3))` where `greal_name` is the name of a global real parameter. `string1` (`string3` can be a real number or the name of a previously defined global real parameter. `string2` is used to denote an operation and can be one of `+`, `-`, `*`, `/`, `**`. For example,
```
greal 2r=$((2)*(r))
greal y1=$((x1)**(1.5))
```

In both forms of expression evaluation, the `$` sign and brackets are required by the syntax rules. Multiple operations are not allowed in a single `greal` statement and must be carried out with a series of statements. For example,
```
greal k=1.3806226e-23
greal q=1.6021918e-19
greal tmpr=300
greal kT=$((k)*(tmpr))
greal vt=$((kT)/(q))
```

3.3 Choosing elements

Given the circuit to be simulated, how do we decide which component to choose from the element list? There are various ways to do this.

(a) Selecting an element from the element list, if the element name is known: As an example, suppose we want to get `vsrcac.ece` on the canvas. There are two ways of doing this:

   (i) Expand the electrical element list. Click on `vsrcac.ece`. That item will get highlighted. Release the mouse button. Move the cursor to the desired position on the canvas and double-click.

   (ii) Type `vsrcac` in the search box in the left window. The GUI will present all elements whose names match the string you have entered. Click on `vsrcac.ece`, move the cursor to the desired position on the canvas, and double-click.

(b) If the element name is not known, go through the existing SEQUEL projects. Check if there are circuits similar to the one you have in mind. If so, find out the name of the element that will serve your need.

(c) Go through the on-line documentation about the elements by right-clicking on an element in the expanded list and selecting help. Figure out which element is most suitable for your purpose. In many cases, the element names themselves may be descriptive enough, e.g., the elements `r.ece`, `c.ece`, `l.ece` can be used for a resistor, capacitor, and inductor, respectively.

(d) Copy/paste from another project: If there is an element which has already been assigned values that are suitable for your project, you can use copy/paste[8], in one of the following ways.

   (i) Select the element and press `ctrl c`. This will copy the object to the "clipboard." You can view the contents of the clipboard by selecting the "clipboard" tab of the right window. Take the cursor to the location where you want to paste the copied element and press `ctrl v`.

   (ii) Keep the `ctrl` key pressed and, using the mouse, draw a rectangle around the elements to be copied. Press `ctrl c`. This will copy the elements to the clipboard. Take the cursor to the location where you want to paste the copied elements and press `ctrl v`. The elements will appear there.

   The clipboard may also be used to cut/paste elements, by simply using `ctrl x` instead of `ctrl c`. This feature is generally useful if an element or a group of elements needs to be moved from one location to another within a project.

   The clipboard remembers objects copied/cut in the past (even if the GUI is closed and started again), and *any* object on the clipboard can be selected for pasting by

[8]Copy/paste works in an intra-project as well as inter-project manner. It also works between two separate GUI applications.

simply clicking on it. A clipboard object can be removed by right-clicking on it and then selecting "remove." It is a good idea to clear the entire clipboard once in a while, if the elements copied earlier are not going to be useful to you any more.

One needs to be careful if an element, which has been copied from another project, has global parameters assigned to it. The same global parameters should be defined for the current project as well, following the procedure described in Sec. 3.1.

3.4 Navigating in the canvas

The following facilities may be used to navigate around the canvas in the circuit editor mode.

(a) Zooming in and out can be done by clicking on the "zoom in" or "zoom out" button. Alternatively, you can use the scroll wheel on your mouse for zooming. There are two other useful zooming options:

   (i) The "zoom fit" button, which appears with two arrows, is used to adjust the grid resolution so that the circuit fits exactly on the canvas. It is also useful when you have "lost" your circuit (i.e., it is somewhere in the canvas, but that part of the canvas is out of view). Pressing the "zoom fit" button will make the circuit visible.

   (ii) The "zoom one" option, which can be accessed from the "view" drop-down menu, is used to make the grid spacing equal to the default unit value.

(b) The whole canvas can be moved left/right or up/down by using the scrolling bars on the right and at the bottom of the canvas. Alternatively, one may left-click on an empty location on the canvas, hold the mouse button down, and move the canvas with the mouse.

3.5 Moving elements

Elements can be moved from one location to another on the canvas in the following ways.

(a) Left-click on the element and move the cursor to the desired position (with the key pressed).

(b) Right-click on the element, and select "Move element." A dialog box will appear in which you can specify the number of grid points by which the element should be moved in the x and y directions. The action is completed when you click on the `OK` button.

(c) Select the element and use the arrow keys to move it.

For the above options to work, the element must be “free”, i.e., it should not have any wires connected to it.

(d) Use the cut/paste option described earlier to delete the element from its previous position and paste it at the desired position. In this case, if the element is connected to others, all wires attached to it will be removed automatically.

3.6 Rotating and flipping of elements

Click on the desired element. It will be highlighted. In the right window, click on the “property editor” tab. For rotation, you will find a drop-down menu for rotating the element through 0, 90, 180, and 270 degrees (clock-wise). Alternatively, clicking on the element and typing `r` may be used to rotate the element through 90 degrees (clock-wise).
For flipping, you will see two option in the right window: “horizontal” and “vertical” for flipping it horizontally and vertically, respectively.

3.7 Labeling an element

Select the element by clicking on it. Type the desired element in the “caption” field of the property editor. It may be noted that names of elements are optional; the GUI will automatically generate a default name for each element added to the circuit.

3.8 Adding text on the canvas

For the purpose of documentation or presentation, it is useful to add some text (e.g., labels and values of components) to the circuit schematic. This can be done by clicking on the text button, which puts the GUI in the “enter text” mode. On clicking at an appropriate location on the canvas, a text box will appear which will allow you to choose the font and font size, and type text. The desired text will appear on the screen. Bring the cursor close to the text, a semi-transparent rectangle will appear. You can click anywhere on this rectangle and move it around. After you have placed it at the appropriate location, it is a good idea to resize the rectangle so that it does not come in the way for your further work. It is also possible to make the text disappear from the canvas by clicking on the “toggle text” button.
Text added by the user is only meant for making the schematic more meaningful for the user; it is ignored by SEQUEL.

3.9 Exporting the schematic

The circuit schematic, along with the text added by the user, if any, can be exported to a format such as `jpg` or `pdf`. Select “snapshot” from the “view” drop-down menu, select the appropriate format, select the directory where you intend to write the file, and type the name of the file. Finally, click on “save” to complete the process.

3.10 Wiring

To connect[9] node 1 to node 2, click on node 1. The GUI will enter the wiring mode, indicated by the highlighted wiring icon. Move the cursor, with the mouse button released, to another location on the canvas. Clicking there would create an “anchor” at that location. Proceeding in this manner, complete the wire by finally clicking on node 2. The wiring mode will automatically terminate.

A wire may similarly be connected between an element node and some point on another existing wire. Doing this would make the new wire the same as the old wire (i.e., the two will acquire the same node name).

If you are in the middle of making a connection and would like to abandon the present wire (without completing the connection), press F4, and the GUI will exit the wiring mode.

Important: Placing a node of an element on top of a node of another element does NOT make a connection. It is best to place the elements slightly apart and then connect them with a wire. If it is desirable to have two elements with overlapping nodes, make sure (by manually editing the “nodes” section of each element) that the overlapping nodes are assigned the same node name.

3.11 Deleting a wire (node)

Select the wire by clicking on it. It will get highlighted. Now press the delete key. Note that deleting a wire will also delete the node assignments for the elements which were connected to that node, and the GUI will automatically generate new node names for the concerned element nodes.

3.12 Naming a wire (node)

There are two ways of naming a wire:

[9]Note that only nodes of the same type (electrical, digital, general, temperature, or thermal power) can be connected.

(a) Select the wire by clicking on it. Click on the “property editor” tab of the right window and edit the name of the node there.

(b) Select an element (by clicking on it) which is connected to the concerned wire. Click on the “property editor” tab of the right window. The properties of the element will appear in the right window with the concerned node name as one of the properties. Edit the name there.

It should be noted that any change in the name of a node applies to *all* elements connected to it and also to output variables (if any) which contain the name of that node.

3.13 Using “connectors” for wiring

In the element list, you will find “connector” elements, `connector_e.ece`, `connector_d.ece`, `connector_g.ece`, `connector_t.ece`, and `connector_p.ece`. These are “dummy” elements and are ignored by SEQUEL, but they are useful in the following situations:

(a) Long wires: Wiring of long wires can be broken up into smaller segments by using connectors. For this purpose, instead of connecting node 1 directly to node 2, we can connect node 1 to a connector and then the connector to node 2.

(b) Distinguishing between crossing and connecting wires: If two wires are crossing each other, they may or may not represent the same node, depending on how they were wired. In such cases, a connector can be used to indicate a physical connection between two crossing wires. This makes the circuit diagram more readable.

3.14 Use of tool tip

It is not always necessary to click on a node or element to know its name. You can place the cursor, without clicking a mouse button, on the desired node or element for a short time (a second or so), and its name will be displayed by the GUI in a box right there.

3.15 Reference node

In a circuit containing electrical elements, it is required by SEQUEL that one of the nodes be declared as the reference node. This is done by bringing `ground.ece` into the canvas and connecting the desired node to it, as shown in Fig. 5.

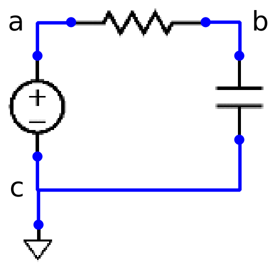


Figure 5: Declaration of node `c` as the reference node by connecting `ground.ece`.

3.16 Use of "dummy" ground

In some circuits, many element nodes may need to be connected to the reference node, something that may become cumbersome. In such cases, we can simply edit the node name as a property of the element rather than physically connecting it to the reference node. As an example, consider Fig. 6. The source marked `vcc` has two nodes, `p` and `n`. We can make the

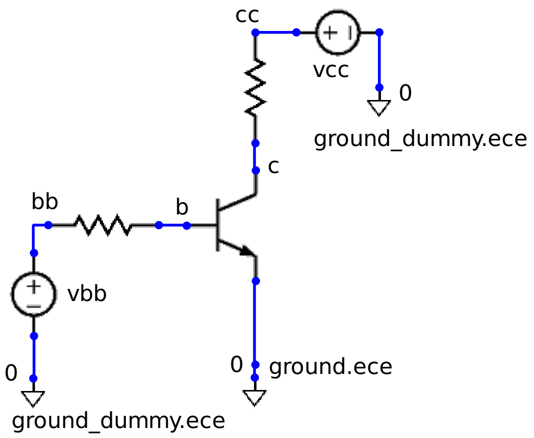


Figure 6: Use of `dummy_ground.ece` to indicate connection to reference node.

assignment `n=0`, `0` being the name of the reference node, in the property editor for this element. Although this is good enough for the purpose of running SEQUEL, we would not like to leave this node simply hanging around, and it would be nice to visually indicate that it is connected to the reference node. This can be achieved by bringing `ground_dummy.ece` (which has the same symbols as `ground.ece`) into the canvas and connecting it to that node, as shown in the figure. Similarly, another `ground_dummy.ece` has been connected to the source named `vbb`.

3.17 Expanding the working space

For larger circuits, it is convenient to have a larger canvas. This can be done to a limited extent by reducing the size of the left, right, and bottom windows by clicking on the respective boundary and re-sizing. These windows may also be closed by clicking on the "close" (cross) icon in the window bar. When necessary, each of these windows can be displayed again by selecting the "panels menu" in the "view" drop-down menu and then selecting message area, tool box, or configuraion panel.

3.18 Selection of output variables

Output variables are used for the purpose of writing the simulation results to output files. There are two steps involved in this process: (a) defining the variables of interest, and (b) indicating to the simulator which of the variables defined in (a) should be written to the output filess. Of these, step (a) step is taken care of in the circuit editor window and step (b) in the solve blocks window.

In the circuit editor, we define *all* output variables that we are likely to be interested in. When an output variable is defined, the GUI creates an assignment of the type,

`outvar_name=string1_of_string2`,

where `outvar_name` is the name of the output variable, and the expression on the right is generated automatically by the GUI, depending on the user's selection. For example, if the user has selected the node voltage of a node called `vcc`, then the expression,

`var1=nodev_of_vcc`

is generated. The name `var1` (it could be `var2`, `var3`, etc.) is a default name generated by the GUI, and it could be edited by the user by double-clicking on `var1` and typing the desired name in its place.

To select an output variable, click on the "output variables" tab in the right window and then on "add variable." The cursor will turn into a plus ("cross-hair"), indicating that the GUI is now in the "add output variable" mode. When an output variable is selected, the "add output variable" mode will automatically terminate, and the cursor will return to its normal shape[10]. The various types of output variables can be selected as follows:

[10]There is also an "expert" add variable mode which one can enter by clicking on the "Output variables" tab while keeping the `cntrl` key pressed. In that case, the user can keep adding output variables one after another and finally, click on the "Add variable" button once again to exit the expert add variable mode. Exiting the mode is rather important!

(a) Node voltage: Click on the desired node (wire) of type electrical,

(b) Node voltage (AC): Click on the desired node (wire) of type electrical, say node `xyz`. You will see a new entry with `nodev_of_xyz` in the right window. Right-click on it and tick the box for AC. The string will now turn into `nodev_ac_of_xyz`, as required by the SEQUEL syntax rules.

(c) General variable: Click on the desired node of type general.

(d) General variable (AC): Click on the desired node (wire) of type general, say node `xyz`. You will see a new entry with `var_of_xyz` in the right window. Right-click on it and tick the box for AC. The string will now turn into `var_ac_of_xyz`, as required by the SEQUEL syntax rules.

(e) Digital variable: Click on the desired node of type digital.

(f) An output variable related to an element: For most of the elements, one or more quantities *internal* to that element are available as output variables. These are listed in the element document. For example, for `r.ece`, you will find that the quantities, `v1` and `i1` (the branch voltage and branch current, respectively) are available as output variables. These can be selected as output variables at the circuit level, by clicking on an element of type `r.ece` in the circuit and then selecting `v1` or `i1`.

4 Solve sections: general comments

Once the circuit block is completed, we need to specify which type of analysis is to be carried out by the program. Depending on the circuit, only certain analyses are meaningful, and only those should be specified. For example, in a circuit involving digital elements, it would not make sense to perform AC analysis. Similarly, in machine drives, which involve components such as a PI controller, transient simulation would be of interest, and it would not be meaningful to carry out DC analysis or AC analysis.
Following is a brief description of some of the analyses[11] provided by SEQUEL.

4.1 DC analysis

DC analysis is relevant when all variables of interest are expected to be constant with respect to time. If there are elements involving time derivatives (e.g., inductors and capacitors), the

[11]SEQUEL provides other analyses, viz., noise and sensitivity; however, we will not discuss these here.

time derivatives are set to zero. Note that DC analysis also covers situations in which we are interested in the variation of a certain DC quantity (such as a voltage or a current) with respect to a circuit parameter (such as a supply voltage or a resistance).
Examples: bias point computation in a BJT amplifier, transfer (output versus input) characteristic of a BJT or CMOS inverter.

4.2 Start-up analysis

Start-up analysis is relevant when there are elements involving time derivatives, and we want to know the solution at "$t = 0$."
Consider the RC circuit shown in Fig. 7. In DC analysis, we would set the derivative dV_c/dt

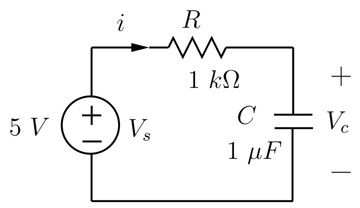


Figure 7: An example of start-up analysis.

to zero, thereby making the current ($= C\ dV_c/dt$) equal to zero. There is therefore no voltage drop across the resistor, and the entire source voltage ($5\ V$) appears across the capacitor.
The "start-up" situation is very different. Here, we have a "start-up" parameter (let us call it V_0) for the capacitor, which specifies the initial voltage on the capacitor at $t = 0^-$. If $V_0 = 2\ V$, for example, then the conditions at $t = 0^-$ are $V_c = 2\ V$ and i = 3 mA, which are clearly quite different than the DC solution.
A start-up analysis is generally used to provide the starting point for a transient analysis solve block. In the RC circuit of Fig. 7, for example, the result of transient simulation (i.e., the solution for $t > 0$) would depend on the start-up solution of the circuit (with a specified value of V_0), and a start-up analysis should therefore be carried out before transient analysis.
In many other situations of interest, however, the values of the various start-up parameters (such as capacitor voltages and inductor currents) cannot be set *a priori*, and therefore, start-up simulation is not particularly relevant. In these cases, we may as well start our transient analysis solve block without a prior start-up analysis. This can be done by setting `initial_sol` to `initialize`. As an example, consider the response of a machine drive to a step change in the speed reference (see Fig. 8). In this case, we could perform transient analysis (without a preceding start-up analysis) for a suitable time interval in order to reach the steady state and then apply the step in the reference speed.

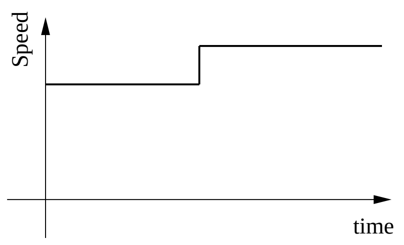


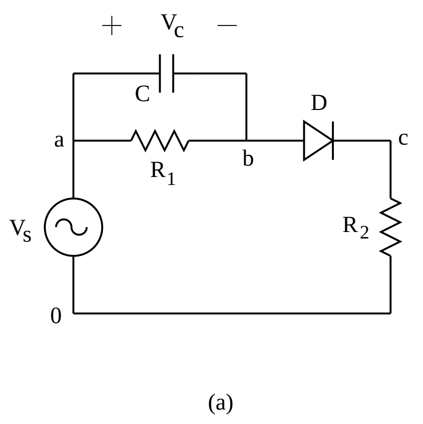
Figure 8: An example of a speed command for a drive.

4.3 Transient analysis

Transient analysis is required when the variables of interest are changing with time. The starting point for a transient analysis solve block could be obtained from a previous solve block (by setting `initial_sol` to `previous` or `file`) or by using an internally generated initial condition (by setting `initial_sol` to `initialize`). In transient analysis, the circuit equations are solved at several time points, and the quantities of interest are stored in a file at each time point (or at fixed output time intervals specified by the user).
Methods available for transient analysis may be broadly divided into two categories:
(a) methods using a constant time step and (b) methods using a variable (automatically computed) time step. The basic objective of the auto step methods is to use a small time step only when it is required and use a larger time step otherwise, thus saving computation time.

4.4 Steady-state waveform (SSW) analysis

Steady-state waveform (SSW) analysis may be considered as a special case of transient analysis. SSW analysis is useful when we are interested only in the periodic steady-state response of the circuit, i.e., only in *one* period in the steady state.
Consider the circuit in Fig. 9 (a), with $f = 50$ Hz, i.e., with period $T = 20$ ms. To reach the steady-state condition for this circuit, we could have started from scratch (say, with an initial condition $V_c = 0\ V$) and solved the circuit equations for several cycles until all quantities settled to a periodic steady state. This approach is inefficient in terms of computing time, since we may need to simulate for hundreds or thousands of cycles before steady state is reached. In reality, we may not care for the results for these hundreds of cycles but only for what happens once the steady state is reached. The SSW option makes it possible to get the steady-state information *without* going through the long initial transient.
The basic idea behind the SSW method is shown in Fig. 9 (b), which shows the results for the state variable V_c from $t = 0$ to $t = T$. If we start with $V_c(0) = 0\ V$, numerical integration of the circuit equations results in $V_c(T) = 0.12\ V$ (the green curve). $V_c(0) = -0.4\ V$ results in



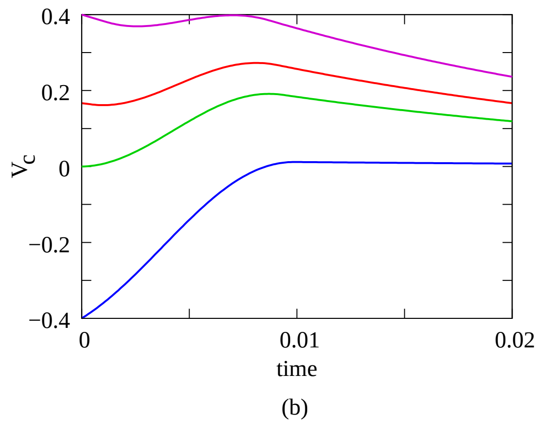


Figure 9: Illustration of SSW analysis: (a) circuit example, (b) results for V_c.

$V_c(T) = 0\ V$ (the blue curve), and so on. If $V_c(0)$ is chosen in such a way that $V_c(T)$ ends up being *equal* to $V_c(0)$, we have got the desired solution, because that is exactly the condition required for periodic steady state. If there are other state variables in the circuit (such as inductor currents and capacitor voltages), *all* of them should satisfy the condition of periodicity, i.e., $X_i(0) = X_i(T)$, where X_i is a state variable. The purpose of SSW analysis is to obtain the set $X_1(0)$, $X_2(0)$, $\cdots$ such that, after solving the circuit equations for one period, we end up with $X_1(T) = X_1(0)$, $X_2(T) = X_2(0)$, etc. This is achieved by employing the Newton-Raphson (N-R) method [1]-[5] as shown by the "SSW N-R loop" in Fig. 10. Note that, if the circuit is nonlinear, the transient analysis block would also involve a separate N-R loop (at each time point). The "inner" N-R iterations (in the transient analysis block) and the "outer" N-R iterations (in the SSW loop) are governed by different method parameters, as we shall see later.

4.5 AC analysis

AC analysis should be performed when one is interested in the frequency response of a circuit or system. AC analysis involves computations with complex numbers (phasors) and may be categorized as follows:

(i) Bias-independent components: Consider the RLC circuit shown in Fig. 11. Here, for a given frequency ω, the impedances due to R, L, and C can be computed as R, $j\omega L$, and $-j/\omega C$, respectively, and they are independent of any DC bias that the source may have. In such cases, we can perform AC analysis directly, i.e., without a prior DC analysis.

(ii) Bias-dependent components: Consider the common-emitter amplifier shown in

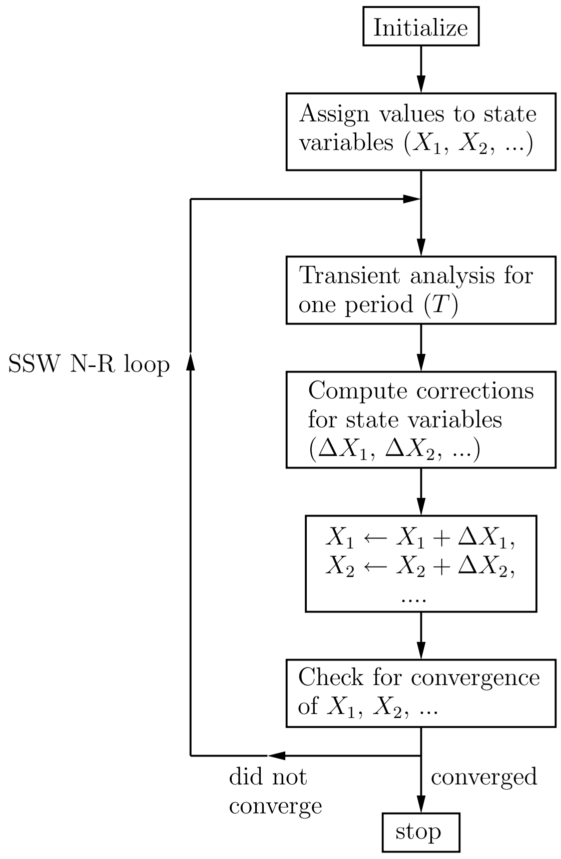

Initialize
Assign values to state variables (X_1, X_2, ...)
Transient analysis for one period (T)
Compute corrections for state variables (ΔX_1, ΔX_2, ...)
$X_1 \leftarrow X_1 + \Delta X_1$, $X_2 \leftarrow X_2 + \Delta X_2$, ...
Check for convergence of X_1, X_2, ...
SSW N-R loop
did not converge
converged
stop

Figure 10: Flow chart for SSW analysis.

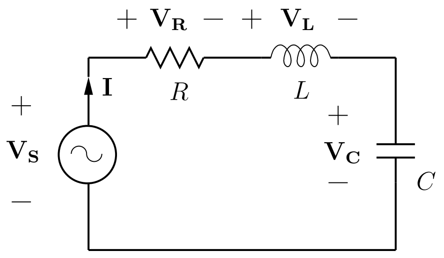


Figure 11: A series RLC circuit.

Fig. 12 (a). There is a nonlinear element here, viz., the BJT, whose small-signal (AC) behaviour depends on the bias point, i.e., the DC solution. In particular, if the π equivalent circuit is used to model the BJT in the small-signal situation (see Fig. 12 (b)), then $g_m = I_C/V_T$, and $r_\pi = \beta/g_m$ depend on the DC collector current I_C. Clearly, it would not make sense here to perform AC analysis directly; the AC solve block must be preceded by a DC solve block so that the bias-dependent quantities in the small-signal equivalent circuit get computed by the simulator.

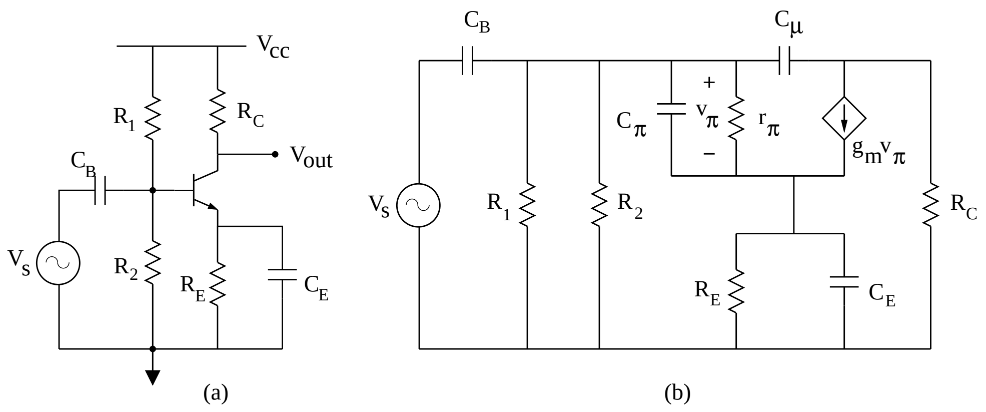


Figure 12: Common-emitter amplifier: (a) circuit schematic, (b) small-signal equivalent circuit.

5 Solve sections: N-R parameters

Nonlinear equations can arise in different types of analyses: DC, start-up, transient, and SSW. SEQUEL, like most other circuit simulators, uses the Newton-Raphson (N-R) method to solve nonlinear equations. The N-R method is controlled by the following parameters:

5.1 Maximum number of iterations

* `itmax_newton` (optional, default=150) is used to specify the maximum number of N-R iterations. If the N-R process does not converge in `itmax_newton` iterations, then (a) In DC or start-up analysis, the program will stop and display an error message. (b) In transient or SSW analysis, the program may stop or continue by repeating the N-R process with a smaller time step, depending on the method indicated by the user (to be discussed).

5.2 Convergence criteria

The N-R process is said to converge when certain convergence criteria are satisfied. SEQUEL allows the following norms (criteria) to be specified.

* `norm_2` (optional, default=`1e-10`) corresponds to

$$\epsilon_{\mathrm{RHS}(2)} = \frac{1}{N} \sqrt{\sum_{i=1}^{N} f_i^2}\,. \tag{1}$$

The system of equations being solved is $f_i = 0$; thus, the magnitude of f_i represents the *deviation* from zero and is therefore a measure of error.

* `delxmax_kcl` (optional, default=`1e-9`) corresponds to

$$\epsilon_{\text{KCL}} \;=\; \text{max over all nodes}\left(\sum_k i_k\right) . \tag{2}$$

  In other words, at *each* node, we compute the sum of all currents entering that node and then take the maximum over all nodes. ϵ_{KCL} should ideally be zero. `delxmax_kcl` is the deviation (in Amperes) from zero that we are willing to tolerate. Note that `delxmax_kcl` is relevant only if there are electrical elements in the circuit.

* `delxmax_volt` (optional, default=`1e-4`) corresponds to

$$\epsilon_{\Delta V} \;=\; \text{max over all nodes}\,(\Delta V_i) \;, \tag{3}$$

  where ΔV_i (at node i) is the change in the node voltage from one N-R iteration to another. As the N-R process converges, all ΔV values should approach zero. $\epsilon_{\Delta V}$ is the deviation (in Volts) from zero that we are willing to tolerate. Note that `delxmax_volt` is relevant only if there are electrical elements in the circuit.

* `delxmax_tmpr` (optional, default=`0.1`) corresponds to

$$\epsilon_{\Delta T} \;=\; \text{max over all temperature nodes}\,(\Delta T_i) \;, \tag{4}$$

  where ΔT_i (at temperature node i) is the change in the temperature (in ^oC or ^oK) from one N-R iteration to another. As the N-R process converges, all ΔT values should approach zero. $\epsilon_{\Delta T}$ is the deviation from zero that we are willing to tolerate. Note that `delxmax_tmpr` is relevant only if there are thermal elements in the circuit.

* `delxmax_pwr` (optional, default=`1e-7`) corresponds to

$$\epsilon_{\Delta P} \;=\; \text{max over all power nodes}\,(\Delta P_i) \;, \tag{5}$$

  where ΔP_i (at power node i) is the change in the thermal power (in W) from one N-R iteration to another. As the N-R process converges, all ΔP values should approach zero. $\epsilon_{\Delta P}$ is the deviation from zero that we are willing to tolerate. Note that `delxmax_pwr` is relevant only if there are thermal elements in the circuit.

Which of the convergence criteria should be checked by the simulator? This is decided by additional statements, as follows.

* `chk_rhs2=yes (no)` for checking (not checking) `norm_2` (default=`yes`)

* `chk_delx_kcl=yes (no)` for checking (not checking) `delxmax_kcl` (default=`no`)

* chk_delx_volt=yes (no) for checking (not checking) delxmax_volt (default=no)

* chk_delx_tmpr=yes (no) for checking (not checking) delxmax_tmpr (default=no)

* chk_delx_pwr=yes (no) for checking (not checking) delxmax_pwr (default=no)

As an example, suppose the user has specified that the norm_2 criterion should be checked by SEQUEL (by setting the flag chk_rhs2=yes), and norm_2 has been specified to be 1e-9. In this case, the program will compute the quantity $\epsilon_{\mathrm{RHS}(2)}$ (see Eq. 1) in every N-R iteration. If it is smaller than 10^{-9}, the N-R process is said to converge; if not, the program will try another N-R iteration, up to a maximum of itmax_newton iterations.
Sometimes, it is useful to write one or more of the norms described above to the console[12]. In particular, if the N-R process fails to converge, the norm information can help in solving the convergence problem. The following flags may be used to select the norms to be written to the console.

* write_rhs2=yes (no) for writing (not writing) $\epsilon_{\mathrm{RHS}(2)}$ as defined in Eq. 1 (default=no)

* write_delx_kcl=yes (no) for writing (not writing) ϵ_{KCL} as defined in Eq. 2 (default=no)

* write_delx_volt=yes (no) for writing (not writing) $\epsilon_{\Delta V}$ as defined in Eq. 3 (default=no)

* write_delx_tmpr=yes (no) for writing (not writing) $\epsilon_{\Delta T}$ as defined in Eq. 4 (default=no)

* write_delx_pwr=yes (no) for writing (not writing) $\epsilon_{\Delta P}$ as defined in Eq. 5 (default=no)

Since SEQUEL is a general-purpose program that allows different types of variables such as electrical, general, and thermal variables, norm_2 specifies an *overall* limit on $\frac{1}{N}\sqrt{\sum_{i=1}^{N} f_i^2}$, i.e., the computation involves functions f_i of *all* types (electrical, general, and thermal).
It should be remembered that the minimum 2-norm achievable depends on the "scale" of the problem. Consider the circuit shown in Fig. 13 in which R is a normal resistor and R1 is a nonlinear element satisfying,

$$i = k_1\, i + k_2\, i^2 + k_3\, i^3\,. \tag{6}$$

Let us consider two sets of parameters (with $V_0 = 4\ V$ held constant),

[12]These will appear in the bottom window of the GUI, when the "solver output" tab is selected.

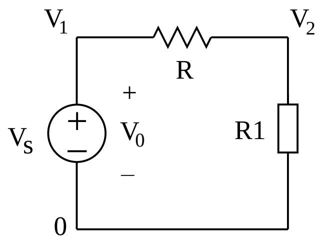


Figure 13: A nonlinear circuit example.

(a) $R = 1\ \Omega$, $k_1 = k_2 = k_3 = 1$, and

(b) $R = 1\ \mathrm{k}\Omega$, $k_1 = k_2 = k_3 = 10^{-3}$,

where k_1, k_2, k_3 are in A/V, A/V^2, A/V^3, respectively. The second case is simply a scaled version of the first, and the solution (in terms of node voltages) is identical in both cases, viz., $V_2 = 1\ V$. The N-R method gives the following results for case (a).

```
 1 norm_rhs_2= 0.444444444          v2=0.000000000E+00
 2 norm_rhs_2= 1.51111111           v2=0.200000000E+01
 3 norm_rhs_2= 0.370467194          v2=0.135849057E+01
 4 norm_rhs_2= 0.0562189282         v2=0.106553526E+01
 5 norm_rhs_2= 0.00219068161        v2=0.100265975E+01
 6 norm_rhs_2= 3.76580074E-06       v2=0.100000458E+01
 7 norm_rhs_2= 1.1187525E-11        v2=0.100000000E+01
 8 norm_rhs_2= 0.                   v2=0.100000000E+01
```

For case (b), it gives,

```
 1 norm_rhs_2= 0.444444444          v2=0.000000000E+00
 2 norm_rhs_2= 0.00151111111        v2=0.200000000E+01
 3 norm_rhs_2= 0.000370467194       v2=0.135849057E+01
 4 norm_rhs_2= 5.62189282E-05       v2=0.106553526E+01
 5 norm_rhs_2= 2.19068161E-06       v2=0.100265975E+01
 6 norm_rhs_2= 3.76580074E-09       v2=0.100000458E+01
 7 norm_rhs_2= 1.11874622E-14       v2=0.100000000E+01
 8 norm_rhs_2= 1.41534746E-17       v2=0.100000000E+01
 9 norm_rhs_2= 1.41534746E-17       v2=0.100000000E+01
10 norm_rhs_2= 1.41534746E-17       v2=0.100000000E+01
```

By the fifth iteration, the solution is already accurate to the third decimal place in both cases (see the `v2` column). However, in (a), the 2-norm actually reaches zero in the 8th iteration while it remains finite and tends to saturate[13] at about 1.4×10^{-17}. If we specify the `norm_2` criterion and specify the value of `norm_2` as 1×10^{-18}, convergence is achievable for case (a) but not for case (b). However, this accuracy is clearly more than what we need, since the solution (`v2`) is already fairly accurate by the fifth iteration. The message is that one should not panic if the simulator says, "convergence not reached," but simply try to loosen the tolerance a little and try again.

We have seen that the difference between successive values of `v2` becomes smaller as the N-R process converges. We could use this difference as a convergence criterion. This can be done by setting `chk_delx_volt=yes`. The maximum acceptable value (`delxmax_volt`) can be assigned a suitable small number, say `0.1m`.

What is the best choice for the convergence criteria? In a purely electrical circuit, `delxmax_kcl` and `delxmax_volt` can be set meaningfully[14]. For example, if we know that the currents in the circuit are in the mA range, we could set `delxmax_kcl=1n` (1 nA).

However, if the circuit has different types of elements (such as an electrical drive which has a motor, a PI controller, etc.), the most appropriate choice would be `norm_2`, which takes into account *all* types of functions. Although the default value (`1e-10`) works well in many cases, selecting a value for `norm_2` can sometimes be a little tricky. If the N-R process fails to converge with a certain value of `norm_2`, it is useful to write the 2-norm to the console (by setting `write_rhs2=yes`) and decide on an appropriate value. Often, it helps to look at the existing examples. Chances are, you will find an example which is similar (in terms of the elements and magnitudes involved) to the one you are trying to simulate, and the convergence criteria set in that example may work for your example as well.

Finally, note that, for linear circuits, there is no question of convergence since the solution is obtained in a single matrix inversion step, without using the N-R process.

[13]The 2-norm often saturates at some point due to the finite precision of the computer. If our computer had infinite precision, we could have made the norm arbitrarily small, by executing a larger number of N-R iterations. The same is true about other norms.

[14]Note that it is possible to set *multiple* convergence criteria. If we set, for example, `chk_delx_kcl=yes` and `chk_delx_volt=yes`, the program will use both ϵ_{KCL} (Eq. 2) and $\epsilon_{\Delta V}$ (Eq. 3) to decide if convergence has been attained.

5.3 Damping of N-R iterations

The most useful and celebrated aspect of the N-R method is its quadratic convergence rate[15]. However, it is not always *robust*: If the initial guess is not "close" to the solution, the N-R process may fail to converge. Damping of the N-R process can be used in such cases to aid convergence.

The basic idea of damping is simple. Instead of updating the solution vector as,

$$\mathbf{x}^{(n+1)} \leftarrow \mathbf{x}^{(n)} + \Delta\mathbf{x}^{(n)}\,, \tag{7}$$

we update it as,

$$\mathbf{x}^{(n+1)} \leftarrow \mathbf{x}^{(n)} + k\,\Delta\mathbf{x}^{(n)}\,, \tag{8}$$

where k is a "damping factor" between 0 and 1. In other words, instead of using the entire correction $\Delta x^{(n)}$, we use only a part of it, viz., $k\,\Delta x^{(n)}$.

Figs. 14 and 15 illustrate how damping can be used to achieve convergence when the standard method fails. Fig. 14 shows that the standard N-R method does not converge with the initial guess, $x^{(0)} = 1.5$. On the other hand, with damping, the N-R method converges for the *same* initial guess (Fig. 15).

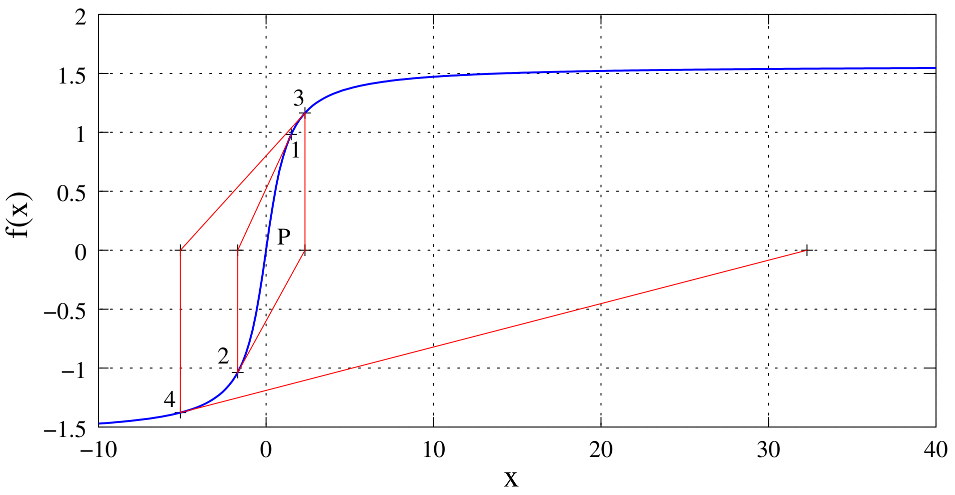


Figure 14: Application of the N-R method to $f(x) = \tan^{-1} x = 0$, with $x = 1.5$ as the initial guess. Note that the N-R method does not converge in this case.

Though damping improves the changes of convergence, it slows down the *rate* of convergence. Damping should therefore be used only when the standard N-R method fails. Also, even if damping is found to help, it could be applied only for the first few N-R iterations. This would

[15] "Quadratic" convergence rate means that the error goes down quadratically. For example, if the error is of the order of 10^{-3} in the present iteration, it would be of the order of 10^{-6} in the next iteration.

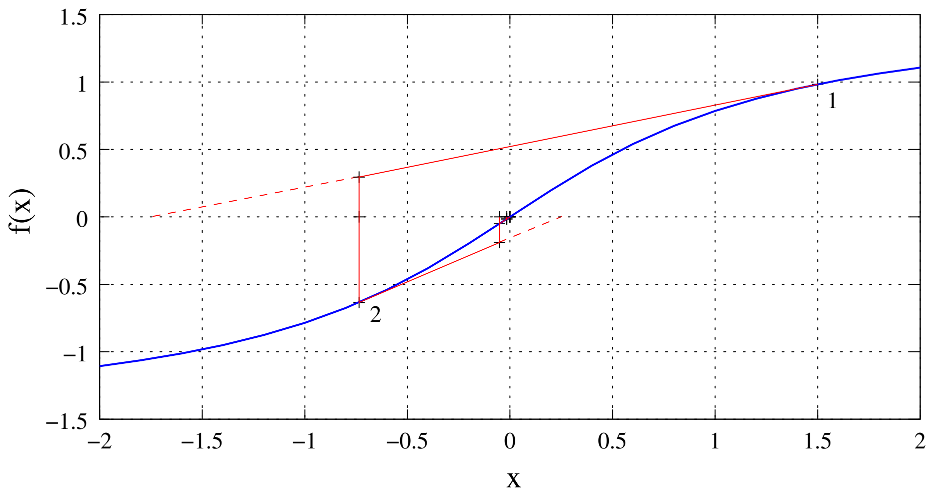


Figure 15: Application of the N-R method to $f(x) = \tan^{-1} x = 0$, with $x = 1.5$ as the initial guess and a damping factor $k = 0.7$.

help to get closer to the solution, and once that happens, the standard N-R method can take over successfully. With this scenario in mind, the following parameters can be specified in SEQUEL solve blocks for using the damped N-R method.

* `dmp=yes (no)` for N-R method with (without) damping (default=`no`)
* `dmp_k` (optional, default=`0.8`): value of k in Eq. 8.
* `dmp_newt_max` (optional, default=`10`): number of N-R iterations for which damping is to be applied. After `dmp_newt_max` iterations, the standard N-R method is used, the total number of iterations (damped+standard) being specified by `itmax_newton`.

  The parameters `dmp_k` and `dmp_newt_max` are relevant only if `dmp` is set to `yes`.

5.4 `gmin` stepping:

An effective way to attain convergence of the N-R iterations is the so-called "`gmin` stepping." It is based on the fact that the N-R method is more likely to converge when the initial guess is close to the solution. Consider the BJT amplifier shown in Fig. 16.[16] The model equations for a BJT involve $\exp(V_{BE}/V_T)$ and $\exp(V_{BC}/V_T)$, which are highly nonlinear terms. As a result, the N-R method may not converge, particularly when a poor initial guess is used. This is a situation in which `gmin` stepping can be useful.

[16]For simplicity, the coupling capacitors are not shown. For a DC bias solution, they are open circuits anyway.

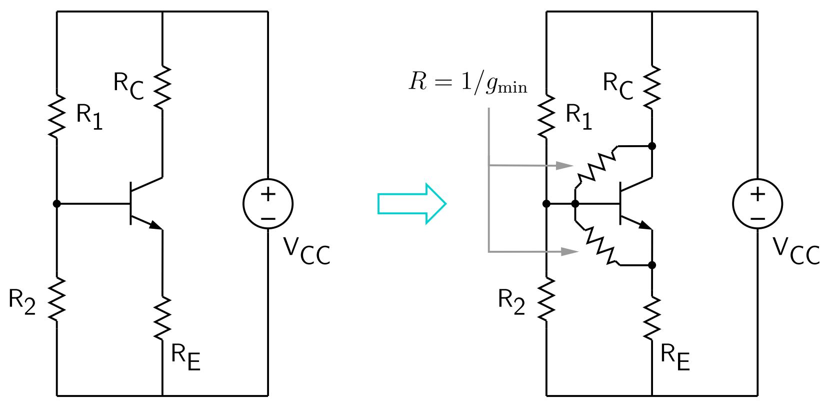


Figure 16: Example showing the use of `gmin` stepping.

In `gmin` stepping, resistors (with $R = 1/g_{\rm min}$; hence, the name `gmin` stepping) are added as shown in the figure. If the resistances are small, the BJT is effectively bypassed, and the circuit becomes a linear circuit leading to quick convergence of the N-R iterations. The solution thus obtained is used as the initial guess for a new problem in which the resistance values (for the newly added resistors) are increased from the previous values. This process is repeated until the bypass resistances become very large (e.g., $10^{12}\,\Omega$) and become effectively open circuits, and the solution for the original problem is thus obtained.
The following parameters govern `gmin` stepping in SEQUEL:

* `gmin_step=yes (no)` if gmin stepping should (not) be used. (default=`no`)
* `gmin_start` specifies the starting value of `gmin` (default=$1\,\Omega^{-1}$)
* `gmin_end` specifies the ending value of `gmin` (default=$1 \times 10^{-12}\,\Omega^{-1}$)
* `gmin_npoints` specifies the number of `gmin` steps to be used in going (logarithmically) from `gmin_start` to `gmin_end`. (default=50)

The parameters `gmin_dmp`, `gmin_dmp_k`, `gmin_dmp_newt_max`, `gmin_itmax_newton` have the same meaning as `dmp`, `dmp_k`, `dmp_newt_max`, `itmax_newton`, respectively, that we have come across earlier, except that they are used only during the `gmin` stepping procedure.

6 Circuits involving digital elements

Digital (logical) elements such as gates and flip flops are handled by SEQUEL in a manner that is fundamentally different than that for analog elements. For a circuit involving analog elements, the task is that of assembling the circuit equations and solving them either in one

matrix inversion step (if the problem is linear) or in several N-R iterations (if the problem is nonlinear). The outcome of this exercise, i.e., the solution, is a vector of real numbers representing the node voltages, brach currents, etc.
For a digital circuit, on the other hand, we are not interested in the actual (analog) value of a voltage; we are only interested in knowing whether it is high ("1") or low ("0"). This calls for a completely different – and computationally much cheaper – approach.

6.1 Transient analysis

The most effective way (and the one implemented in SEQUEL) for transient analysis of circuits involving digital elements is the so-called "event-driven" simulation, in which we look at the inputs of each digital element, and schedule an "event" (i.e., $0 \rightarrow 1$ or $1 \rightarrow 0$) at its output only if a change is expected to happen as a result of a change in the input(s). The time interval between a change at the input(s) of an element and an event at its output node(s) is defined by the delay values for the concerned element. Transient analysis of a digital circuit is carried out by scheduling events and processing them at the scheduled times.
As an example, consider the circuit in Fig. 17 [6]. Initially, A=0, B=0, and C=1. For $t > 0$, the inputs vary as shown in Fig. 18. At $t = 0$, D=E=1, and F=0. At $t = t_1$, one of the

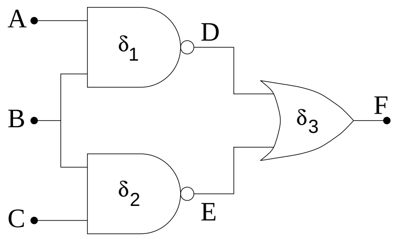


Figure 17: A simple digital circuit. The delay of each gate is also marked.

inputs (A) has changed. The following questions must now be asked: (i) Which gates will be affected by this "event"? (ii) Does it actually cause any of the variables to change? If so, at what time should that change ("event") be scheduled?
It turns out that the event at t_1 does not cause a change in the output of the gate (node D), as its other input is still 0. In fact, in this example, no events need to be scheduled up to t_5. Because of the transition at t_5, node E must change from 1 to 0. This change or "event" must be scheduled at $t = t_5 + \delta_2$, where δ_2 is the delay of the gate. When the event at node E is executed or "processed", the OR gate must be examined for a possible change in its output, as node E serves as an input node for the OR gate.
It is clear that the treatment of digital elements is completely different than that of analog elements, and it is computationally much simpler, almost trivial. Two major differences

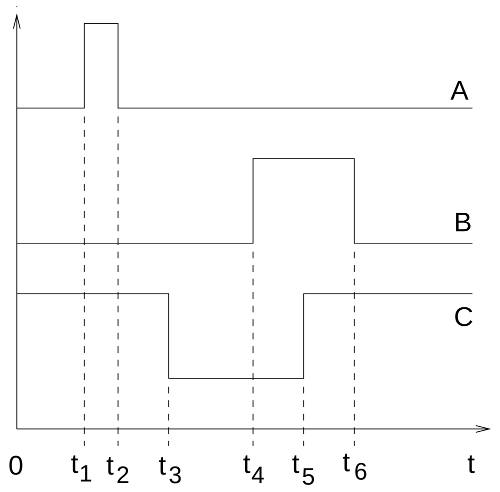


Figure 18: Inputs applied to the digital circuit of Fig. 17

.

between simulation of analog and digital elements emerge from the above discussion:
(a) When analog elements are involved, a system of equations needs to be solved. With digital elements, only *assignment* of logical values needs to be performed. Thus, for the same "complexity" (say, the same number of nodes), the CPU time required per time step would be much larger for analog elements. (b) Simulation of digital elements is "event-driven" [7],[8], i.e., a given digital element needs to be treated only if one or more of its inputs have changed.

6.2 DC and start-up analysis

The situation for DC (or start-up) is different. Here, since we are looking for a solution at just *one* time point, there is no need for scheduling or processing events. Instead, what is required is to obtain the values of all digital variables such that they are *consistent* with each other. Consider the example in Fig. 19. As a starting point, say, we assume the output of each gate

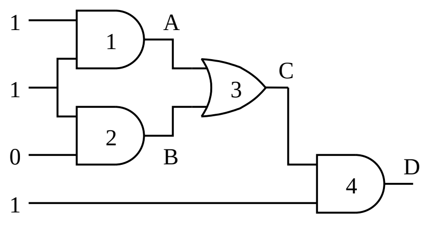


Figure 19: A digital circuit with constant (DC) inputs.

(i.e., nodes `A`, `B`, `C`, `D`) to be `0`. With these conditions, we make the following "passes."

`Pass 1:` We conclude that nodes `A` and `B` must be `1` and `0`, respectively. Note that this *revised* information is not available for the OR gate in this pass.

`Pass 2:` We conclude that node `C` must be `1`.

`Pass 2:` We conclude that node `D` must be `1`.

Thus, at the end of three passes, we have obtained a solution that is consistent with the logical behaviour of all elements. This process is governed by the following parameter.

* `n_pass_dgtl` (optional, default=20): maximum number of passes to be made in obtaining DC or start-up solution for circuits involving digital elements.

In circuits with both analog and digital elements (i.e., "mixed-signal" circuits), the values of the digital variables must be consistent with respect to the digital *and* analog elements. Consider the circuit[17] shown in Fig. 20. Before we start transient analysis of this circuit, we

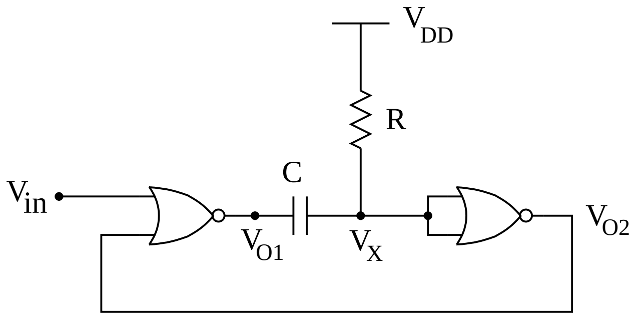


Figure 20: A mixed-signal circuit.

would like to get the start-up solution. The value of V_{o1}, for example, must be such that it satisfies the logical behaviour of the NOR gates *and* the analog solution for the RC circuit. The same is true about other variables.
For DC or start-up analysis, the flow chart shown in Fig. 21 has been implemented in SEQUEL. We now have inner passes for obtaining the solution for the digital variables ($\mathbf{X}$) and outer passes to ensure that the analog variables ($\mathbf{x}$) are also consistent with the digital variables ($\mathbf{X}$). The parameter `n_pass_mixed` governs the outer passes:

* `n_pass_mixed` (optional, default=20): maximum number of outer passes (see Fig. 21) to be made in obtaining DC or start-up solution for circuits involving digital and analog elements. Note that the inner passes in this case are still governed by the parameter `n_pass_dgtl` which was seen earlier.

[17]The A-to-D and D-to-A elements required to convert the signals into the appropriate type are not shown in the figure.

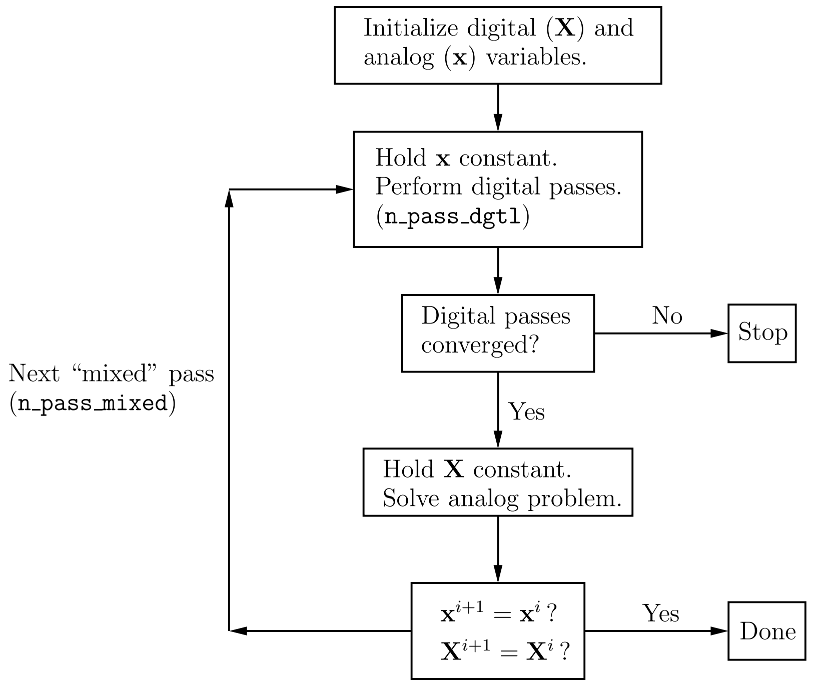

Initialize digital ($\mathbf{X}$) and analog ($\mathbf{x}$) variables.
Hold $\mathbf{x}$ constant. Perform digital passes. (n_pass_dgtl)
Digital passes converged?
No
Stop
Yes
Hold $\mathbf{X}$ constant. Solve analog problem.
$\mathbf{x}^{i+1} = \mathbf{x}^i$? $\mathbf{X}^{i+1} = \mathbf{X}^i$?
Yes
Done
Next "mixed" pass (n_pass_mixed)

Figure 21: Flow chart for mixed-signal analysis in DC or start-up condition.

7 Solve block syntax

The purpose of the solve blocks is to convey several things to the simulator, such as (a) which analysis should be performed for the circuit, (b) Which method should be used, (c) the values of the method parameters, (d) which variables should be stored, etc. In this section, we will discuss these details, starting with the method card, followed by other statements.

7.1 The method card

A large number of parameters are available for the method card, and it is bound to look a little forbidding to a beginner. However, only a subset of these would generally apply to a specific analysis. Further, their default values may often be good enough, so they would not require to be assigned by the user. In the following, we will list the method parameters for each type of analysis, and describe their purpose.

7.1.1 DC/start-up analysis

* N-R parameters: as described in Sec. 5. Note that these are relevant for nonlinear problems only.

* `n_pass_dgtl` and `n_pass_mixed`: as described in Sec. 6.2. Note that these are relevant only if there are digital elements in the circuit.

7.1.2 AC analysis

There are no `method` parameters for AC analysis, since the AC problem is always linear[18], and therefore, there is only one method to obtain the solution, i.e., solving a matrix equation in complex variables. We do need to specify the frequency (frequencies) for which the analysis is requested. This is done by the `set_freq` or `vary_freq` statements, to be discussed later.

7.1.3 Transient analysis

In transient analysis, we have to worry about two (more or less independent) problems:

(i) Solution of equations at a given time point: This is similar to the DC case, and the N-R parameters of Sec. 5 apply here as well, if the problem is nonlinear.

(ii) Discretization of time: This has to do with how the time interval of interest is divided into smaller intervals and which method is used to discretize the time derivative terms in the problem. The following `method` options can be used in this context.

  * `back_euler=yes (no)` for using (not using) the backward Euler method with a fixed time step (default=`no`)
  * `back_euler_auto=yes (no)` for using (not using) the backward Euler method with auto time step (default=`no`)
  * `trapezoidal=yes (no)` for using (not using) the trapezoidal method with a fixed time step (default=`no`)
  * `trapezoidal_auto=yes (no)` for using (not using) the trapezoidal method with auto time step (default=`no`)
  * `gear2=yes (no)` for using (not using) the second-order Gear method with a fixed time step (default=`no`)
  * `trbdf2=yes (no)` for using (not using) the Trapezoidal/BDF2 method (default=`no`)

  If the user does not specify any method for transient analysis, the trapezoidal method is assumed.

  Method-independent parameters: These parameters are applicable for *all* methods.

[18]As we have seen in Sec. 4,even a nonlinear circuit is linearized for AC (small-signal) analysis.

* `t_start` (required, no default): starting time for transient analysis
* `t_end` (required, no default): ending time for transient analysis
* `delt_const` (required, no default): In the constant time step methods (`back_euler`, `trapezoidal`, and `gear2`), `delt_const` specifies the constant time step Δt. In the auto time step methods (`back_euler_auto`, `trapezoidal_auto`, and `trbdf2`), `delt_const` defines the starting Δt, the subsequent time steps being automatically computed by the simulator.
* `delt_min` (optional, default=0.0002×`delt_const`) specifies the minimum time step that the simulator will take. Note that this parameter is relevant even in the constant time-step methods, since certain elements may force time points at intervals smaller than `delt_const`, and `delt_min` then defines the minimum time step that the simulator will take. An example is shown in Fig. 22.

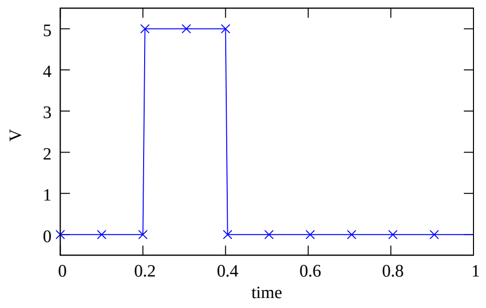


Figure 22: Waveform obtained with `clock_1.gce`. The following parameters have been used: `delt_const=0.1`, `delt_min=0.0001`. Note the extra time points (which have been forced by `clock_1.gce`) near $t=0.2$ and $t=0.4$.

* `delt_max` (optional, default=10×`delt_const`) specifies the maximum time step that the simulator can take.
* `itmax_trns` (optional, default=50000) specifies the maximum number of time steps that the simulator is allowed to take. It is mainly a "safety feature" to ensure that the program does not run for ever if, for example, too large a value for `t_end` or too small a value for `delt_const` has been specified.

Method-specific parameters: In the auto-step methods, there are additional parameters that control the time step, as described in the following.

(a) `back_euler_auto` method: If the problems is nonlinear, the time step can be computed on the basis of convergence of the N-R method, as illustrated in the flow

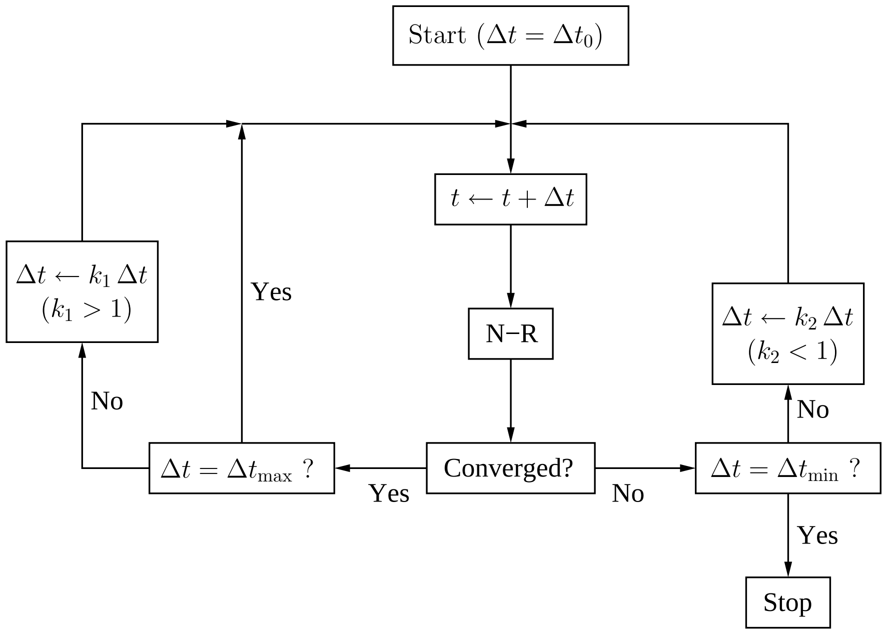

Start ($\Delta t = \Delta t_0$)
$t \leftarrow t + \Delta t$
N−R
Converged?
$\Delta t = \Delta t_{\max}$?
$\Delta t = \Delta t_{\min}$?
$\Delta t \leftarrow k_1\,\Delta t$ ($k_1 > 1$)
$\Delta t \leftarrow k_2\,\Delta t$ ($k_2 < 1$)
Yes
No
Yes
No
No
Yes
Stop

Figure 23: Flow chart for auto step methods based on convergence of N-R iterations.

chart of Fig. 23. For a given value of Δt, if the N-R method converges (in `itmax_newton` iterations), we accept the solution and make the next value of Δt larger than the present value, up to a maximum of `delt_max`. If the N-R method does not converge, we reduce the time step and try again. If there is no convergence with the smallest allowed time step (`delt_min`), the program stops. The following parameters control the time step.

* `fctr_stepred` (optional, default=`0.05`): factor (k_2 in Fig. 23) by which the time step is reduced if the N-R process does not converge.
* `fctr_stepinc` (optional, default=`1.5`): factor (k_1 in Fig. 23) by which the time step is increased if the N-R process converges.
* `itmax_stepred` (optional, default=`20`): Maximum number of times (at a *given* time point) the step size may be reduced if the N-R process does not converge.

(b) `trapezoidal_auto` method: This method uses the same flow chart (see Fig. 23) and parameters as the `back_euler_auto` method.

(c) `trbdf2` method: If the problem is linear, it is solved directly, and not by employing an iterative method. Therefore, a convergence-based method, such as that shown in Fig. 23, is ruled out. In such cases, we can use an approach based on an estimate of the local truncation error (LTE). In the TR-BDF2 scheme [9], the time step Δt_{i+1} (i.e., the interval between t_i and t_{i+1}) is divided into two parts:
(a) $\Delta t_1 = \gamma\ \Delta t_{i+1}$ ($\gamma < 1$) in which the trapezoidal method is applied. We will call

this the TR step. (b) $\Delta t_2 = (1-\gamma)\ \Delta t_{i+1}$ in which the second-order Backward Difference (BDF2) scheme is applied. After we execute one TR and one BDF2 step (i.e., we are now at time t_{i+1}), an estimate of the local truncation error is made, and based on the relative magnitude of this error with the user-specified tolerance value, the current time step is either accepted or rejected.

If the error is found to be too large, the previous time step is rejected, the TR-BDF2 step is carried out with a smaller time step, and the error is calculated again. The time step may again be accepted or rejected, depending on the error. The basic idea is much the same as that depicted in Fig. 23 except that the estimate of LTE is used (rather than convergence of the N-R process) for deciding whether to accept or reject the given time step. More details of the method may be found in [9]. The following parameters are applicable for the `trbdf2` method.

* `trbdf2_tolr` (optional, default=`1e-5`): "tolerance" for the TR-BDF2 method (ϵ_A in Eq. (43) of Ref. [9])
* `trbdf2_gamma` (optional, default=`0.586`): the value of γ in the above discussion. The default value has been obtained by solving Eq. (33) of Ref. [9].

For convenience, the various transient analysis parameters are presented in Fig. 24.

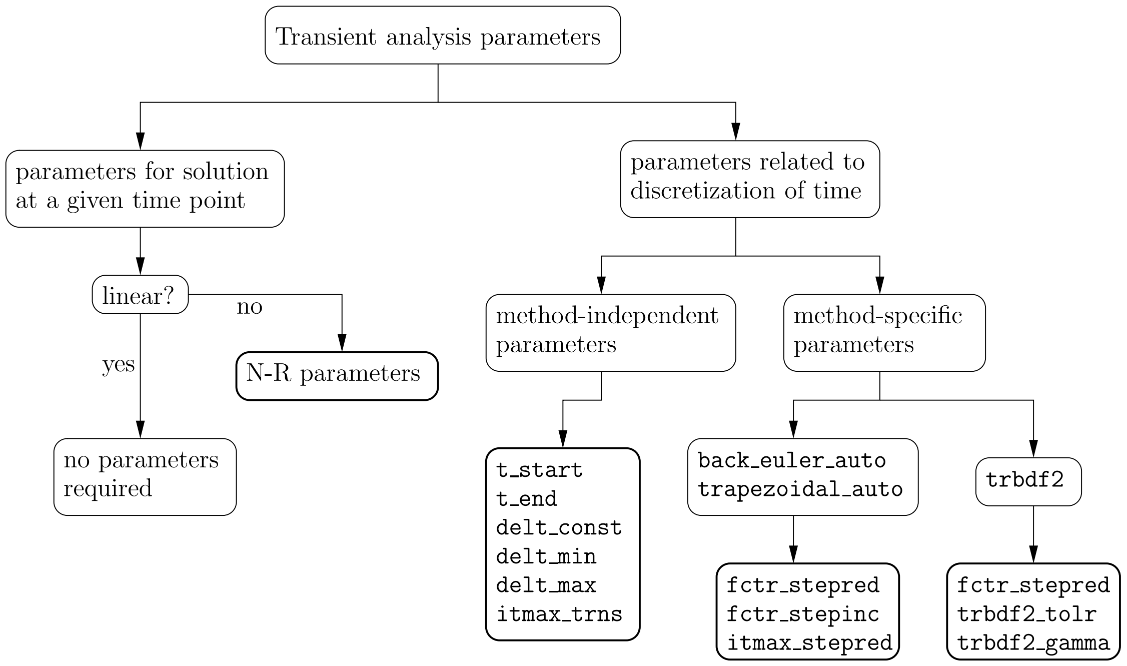


Figure 24: Transient analysis parameters.

7.1.4 SSW analysis

As seen in Sec. 4.4, SSW analysis involves outer SSW N-R iterations and transient analysis in *each* outer iteration. Therefore, all parameters discussed in Sec. 7.1.3 for transient analysis are applicable for SSW analysis except that the choice of methods is restricted to[19] `back_euler`, `trapezoidal`, `back_euler_auto`, and `trapezoidal_auto`. Also, the parameter `t_end`, the ending time in transient analysis, is not relevant for SSW analysis, since we are interested in just one cycle (of duration T).
In addition to the transient analysis parameters, the following parameters are used for SSW analysis.

* `ssw_period`: period (T) of the waveforms

* `ssw_frequency`: This is another way of specifying the period. If `ssw_frequency` (f_0) is specified, `ssw_period` is computed internally as $T = 1/f_0$.

  Note that one of `ssw_period` and `ssw_frequency` must be specified.

* `ssw_period_mult` (optional, default=1): This parameter (an integer) is used to specify how many cycles are to be considered. Making `ssw_period_mult` different than 1 does not make sense from the computational point of view, since the solution will simply repeat at intervals of T, and there is no new information in the subsequent intervals. However, `ssw_period_mult`$>$ 1 is useful when the user wishes to generate plots that bring out the periodic nature of the waveforms explicitly.

The parameters `ssw_itmax_newton`, `ssw_norm`, `ssw_dmp`, `ssw_dmp_k`, and `ssw_dmp_newt_max` are similar to the ones seen in Sec. 5.3 except that they apply to the SSW N-R iterations (the outer loop in Fig. 10), as described below.

* `ssw_itmax_newton` (optional, default=20) is used to specify the maximum number of N-R iterations in the SSW N-R loop. If the N-R process does not converge in `ssw_itmax_newton` iterations, the program will print an error message and stop.

* `ssw_norm`: value of tolerance to be used for comparison with $\epsilon^{\rm SSW}_{\rm RHS(2)}$ to decide whether the SSW N-R iterations have converged. $\epsilon^{\rm SSW}_{\rm RHS(2)}$ is given by

$$\epsilon^{\rm SSW}_{\rm RHS(2)} = \frac{1}{N'} \sqrt{\sum_{i=1}^{N'} f_i^2}\,, \tag{9}$$

  where N' is the number of state variable equations.

[19]Since the time interval of interest for SSW is much smaller than a typical transient analysis interval, there is generally no particular need for using the auto step methods, and `back_euler` or `trapezoidal` works fine.

* `ssw_dmp=yes (no)` to indicate that damping should (should not) be used for the SSW N-R iterations (optional, default=`no`).

* `ssw_dmp_k` (optional, default=`0.8`): value of k (see Eq. 8) for the SSW N-R loop.

* `ssw_dmp_newt_max` (optional, default=`10`): number of SSW N-R iterations for which damping is to be applied. After `ssw_dmp_newt_max` iterations, the standard N-R method is used, the total number of iterations (damped+standard) being specified by `ssw_itmax_newton`.

  The parameters `ssw_dmp_k` and `ssw_dmp_newt_max` are relevant only if `ssw_dmp` is set to `yes`.

7.2 Other solve block statements

Apart from the method statements, a solve section can have a few other statements which may be used depending on the analysis type of the solve block.

7.2.1 `initial_sol` statement

The purpose of this statement is to specify how to generate the initial solution, i.e., the "starting point" for the analysis. The following options are available for this statement.

(i) `initial_sol initialize`
This indicates that the initial solution is to be internally generated by the simulator. This is the default option.

(ii) `initial_sol file filename=string`
This indicates that the initial solution is to be read from a file whose name is given by `string`.

(iii) `initial_sol previous`
This indicates that the previous solution (obtained in the last solve block) is to be used as the initial solution.

Initial solution is relevant for DC, start-up, transient, and SSW analyses. In particular, if the problem is nonlinear, then the initial solution provides the starting point for the N-R process[20]. For AC analysis, an initial solution is not required.

[20]Note that, for transient and SSW analysis, the initial solution provides the starting point for the N-R process only for the *first* time point. Subsequently, the solution obtained at t_i is used as the starting point for t_{i+1}.

7.2.2 Output block syntax

The purpose of an output block is to inform the simulator about the variables to be stored in the output files, the names of the output files, etc. One or more output blocks are allowed within each solve block. The following statements are allowed in an output block:

(i) `begin_output`
This indicates the beginning of an output block.

(ii) `filename=string1 limit_lines=string2 append=string3`
This statement specifies the name of the output file to be created, and the related options.

`string1` specifies the name of the file. For example, `filename=rc_circuit.dat`.

An upper limit on the number of lines in the file can be specified by `string2`, e.g., `limit_lines=200000`. This is a "safety feature" to guard against inadvertent creation of huge output files; if the number of lines stored in the output file exceeds `limit_lines`, the program will write an error message and stop. The default value for `limit_lines` is 100000.

The keyword `append` indicates whether the file is to be opened in the "append" mode. `string3` can be `yes` or `no` (default=`no`). The `append` facility is useful if the user wishes to split a simulation into different solve blocks but write the data to the same output file, for the purpose of plotting it all together. For example, consider the frequency response of a notch filter. We may want to split the simulation in three parts: (a) $f < f_0$, (b) $f \simeq f_0$, and (c) $f > f_0$. This would allow us to use a relatively coarse frequency resolution in the solve blocks corresponding to (a) and (c) and fine resolution in (b). However, we would like to write the output quantities of interest (versus frequency) to the *same* output file for the purpose of viewing. This can be achieved by using `append=yes` in the solve blocks corresponding to (b) and (c).

(iii) `variables: list`
This indicates the information to be written to the output file. The following options may be used in the `list`:

* names of output variables defined in the circuit block[21]

[21] For an AC variable, say `v1`, we need to specify whether the magnitude or phase of `v1` is to be stored. This can be done by writing `mag_of_v1` or `phase_of_v1` in the list.

* `solution` to indicate the entire solution. This is usually done to store the solution for the purpose of using it as a starting point in another solve block. If `solution` is specified, no other strings are allowed in the `variables` statement.

Variables of digital and analog types are not allowed to be written to the same output file.

(iv) `control: string1=string2` (etc.)
This statement, which is optional, is used for two purposes:

(a) to control the output time points at which output is written to files in a transient analysis solve block[22]. The following combinations for `string1` and `string2` are relevant in this case.

  * `string1: fixed_interval` to specify the interval between successive output time points.
    `string2:` a real number
  * `string1: out_tstart` to indicate that output is not to be written before the specified time.
    `string2:` a real number
  * `string1: out_tend` to indicate that output is not to be written after the specified time.
    `string2:` a real number

(b) to control the way the phase of an AC (complex) variable should be written to the output file. This is meant essentially to avoid "jumps" from -180^o to $+180^o$ in plots of phase versus frequency. Such jumps, although not technically incorrect, can be distracting when one is interpreting a phase plot. To produce a continuous phase plot, we can make use of the fact that any angle θ is equivalent to $\theta \pm 360^o$. The following combinations for `string1` and `string2` are relevant in this case.

  * `string1: min_phase` to indicate the minimum value of phase to be written to the file.
    `string2:` a real number (degrees)
  * `string1: max_phase` to indicate the maximum value of phase to be written to the file.
    `string2:` a real number (degrees)
    Either `min_phase` or `max_phase` may be specified but not both of them together. By default, none of the two is specified.

[22]This should not be confused with the time points at which the circuit equations are actually solved.

* `string1: delta_phase` to indicate the maximum allowed difference between the phase values at successive frequency points.
  `string2:` a real number (degrees, default=200)

  Generally, the default values work well; however, if there are jumps in the phase plot which are not desired, the user could set the above parameters suitably in order to avoid the jumps.

(v) `end_output`
This indicates the end of the output block.

7.2.3 `set_parm` statement

`set_parm string1_of_string2=string3`
This statement assigns a specific value to an integer or real parameter of an element, overriding the definition in the circuit block. It is an optional statement. More than one `set_parm` statements are allowed in a solve block. Here, `string1` is the name of the parameter, `string2` is the name of the element, and `string3` is an integer or a real number, depending on the type of the parameter involved.

7.2.4 `set_stparm` statement

`set_stparm string1_of_string2=real_number`
This statement assigns a specific value to a start-up parameter of an element of type `ece`, `gce`, or `tce`, overriding the definition in the circuit block. It is an optional statement. More than one `set_stparm` statements are allowed in a solve block. Here, `string1` is the name of the start-up parameter and `string2` is the name of the element.

7.2.5 `vary_parm` statement

The purpose of this statement is to vary a real parameter of an element. It is an optional statement. More than one `vary_parm` statements are allowed, and in that case, they work like nested `do` or `for` statements. The parameter appearing in the last `vary_parm` statement is varied first, followed by the parameter in the last but one `vary_parm` statement, and so on. The `vary_parm` statement is applicable for DC, start-up, and AC solve blocks. It can be of one of the three forms given below.

(i)
```
vary_parm string1_of_string2 from real_number_1 to real_number_2
+   type=linear n_points=integer
```

This indicates that the parameter named `string1` of the element named `string2` is to be varied linearly from `real_number_1` to `real_number_2`, the total number of points being given by `n_points`. Note that, the `+` in the first column (here and elsewhere) merely indicates a continuation of the statement in SEQUEL syntax.

(ii)
```
vary_parm string1_of_string2 from real_number_1 to real_number_2
+   type=log n_points=integer
```
This indicates that the parameter named `string1` of the element named `string2` is to be varied logarithmically from `real_number_1` to `real_number_2`, the total number of points being given by `n_points`. Note that the real numbers in this case must be positive.

(iii)
```
vary_parm string1_of_string2 type=table x1 x2 ..
```
This indicates that the parameter named `string1` of the element named `string2` is to be varied. The first real number (`x1`) appearing after `type=table` is to be assigned as the first value of the parameter, and so on.

There is another way of using the `vary_parm` statement in which `string2` is replaced by the keyword `glbl`, thus indicating that all element real parameters which have been mapped to the global real parameter `string1` are to be varied.

7.2.6 `vary_parm_sync` statement

The `vary_parm_sync` statement (valid in DC and AC analyses) is useful when parameters for different elements are to be varied "in sync" with each other. As an example, consider a circuit with several resistors, `r1`, `r2`, etc. If we want to vary some of them simultaneously, then the following statements can be used:

```
vary_parm_sync r_of_r1 from 2 to 10 type=linear n_points=5
vary_parm_sync r_of_r2 from 1 to 5  type=linear n_points=5
```

The above statements will cause the analysis to be repeated five times, with (R_1,R_2)=(2,1), (4,2), (6,3), (8,4), and (10,5). The syntax for the `vary_parm_sync` statement is the same as that for the `vary_parm` statement.

7.2.7 `set_freq` statement

```
set_freq=real_number
```

This statement is used to set the frequency in AC analysis when the analysis is to be performed for a single frequency.

7.2.8 `vary_freq` statement

The purpose of this statement is to vary the frequency in a specified manner in AC analysis. The `vary_freq` statement can be of one of the three forms given below.

(i)
```
vary_freq from real_number_1 to real_number_2
+   type=linear n_points=integer
```
This indicates that the frequency is to be varied linearly from `real_number_1` to `real_number_2`, the total number of points being given by `n_points`.

(ii)
```
vary_freq from real_number_1 to real_number_2
+   type=log n_points=integer
```
This indicates that the frequency is to be varied logarithmically from `real_number_1` to `real_number_2`, the total number of points being given by `n_points`.

(iii)
```
vary_freq type=table x1 x2 ..
```
This indicates that the first real number (`x1`) appearing after `type=table` is the first value of the frequency, and so on.

In an AC analysis solve block, there must be either one `set_freq` statement or one `vary_freq` statement (but not both).

7.2.9 `set_tmpr` statement

```
set_tmpr tmpr_name=real_number
```
This statement is used to (optionally) set values for temperatures. More than one `set_tmpr` statements are allowed. Here, `tmpr_name` is the name of a temperature node in the circuit. The `set_tmpr` statement can be used in AC analysis or analysis of purely digital circuits; it is ignored in other cases.

References

[1] M. B. Patil, M. C. Chandorkar, B. G. Fernandes, and K. Chatterjee, "Computation of steady-state response in power electronic circuits," *IETE J. Research*, vol. 48, no. 6, pp. 471-477, Nov. 2002.

[2] T. J. Aprille and T. N. Trick, "Steady-state analysis of nonlinear circuits with periodic inputs," *Proc. IEEE*, vol. 60, pp. 108-114, 1972.

[3] T. J. Aprille and T. N. Trick, "A computer algorithm to determine the steady-state response of nonlinear oscillators," *IEEE Trans. Circuit Theory,* vol. 19, pp. 354-360, 1972.

[4] F. R. Colon and T. N. Trick, "Fast periodic steady-state analysis for large-signal electronic circuits," *IEEE J. Solid-State Circuits*, vol. 8, pp. 260-269, 1973.

[5] T. N. Trick, F. R. Colon, and S. P. Fan, "Computation of capacitor voltage and inductor current sensitivities with respect to initial conditions for the steady-state analysis of nonlinear periodic circuits," *IEEE Trans. Circuits and Systems*, vol. 22, pp. 391-396, 1975.

[6] M. B. Patil, "A public-domain program for mixed-signal simulation," *IEEE Trans. Education*, pp. 187-193, May 2002.

[7] R. Raghuram, *Computer Simulation of Electronic Circuits*, Wiley Eastern, New Delhi, 1989.

[8] P. Antognetti, D. O. Pederson, and H. de Man, *Computer Design Aids for VLSI Circuits, NATO ASI Series*, Martinus Nijhoff Publishers, The Hague, 1984.

[9] R. E. Bank, W. M. Coughran, W. Fichtner, E. H. Grosse, D. J. Rose, and R. K. Smith, "Transient simulation of silicon devices and circuits," *IEEE Trans. Electron Devices.*, vol. 32, no. 10, pp. 1992-2006, 1992.