Csim - a Simple HP48 Circuit Simulator for Educational Purposes

Per Stenius*
Circuit Theory Laboratory
CT-11

January 1992

Abstract

The report presents the theory required to develop and program a simple circuit simulator capable of DC, AC and transient analysis based on modified nodal analysis. Together with the theoretical presentation, the code for a circuit simulating program is developed in order to show how theory is translated into algorithms. The programming language is RPL, which is commonly used by Hewlett-Packard handheld calculators. The program listing presented in this report applies directly to the HP48 scientific calculator.

Indexing terms: Circuit simulation.

ISBN 951-22-0940-3
ISSN 0784-5979
TKK OFFSET

*The author is with the Helsinki University of Technology, Faculty of Electrical Engineering, Circuit Theory Laboratory, Otakaari 5A, SF-02150 Espoo, Finland.

Contents

1 Introduction

Circuit simulation is often considered to be very complicated and hard to understand. This is also often the case, especially for large circuits with nonlinear components. However, the basics of circuit simulation are not complicated, and in fact anyone understanding ordinary nodal formulation and Laplace-transform theory can easily develop a simple circuit simulator. This is what we want to show in this tutorial, by presenting the theory needed for such a circuit simulator, capable of doing `DC`, `AC` and transient (`TRAN`) analysis. A program capable of this, *Csim*, programmed for the *HP 48SX* handheld calculator is also presented. The reason that this specific calculator has been chosen, is that it has the matrix handling tools required. Developing code for circuit simulation purposes also requires access to matrix handling tools. Since in our case these are provided from the beginning we are able to focus on the problems involved with circuit simulation, without spending too much time on writing mathematical tools that suit our needs.

The *Csim* program is kept small, due to the memory and processing capability restrictions of the *HP 48SX*. Therefore certain restrictions on the simulating capabilities also exist. *Csim* is programmed using the *RPL* (reverse polish) language, uses 10 kB of memory and is capable of

- performing `AC` and `DC` analysis.
- performing transient analysis (`TRAN` analysis).
- simulating linear circuits composed of lumped components with constant values. Lossless transmission lines are provided for `AC` analysis.
- simulating independent sources with time (in `TRAN` analysis) or frequency (in `AC` analysis) dependency. In `TRAN` analysis any functional dependency of node voltages or branch currents in the circuit can be simulated, if this dependency is allowed to have the delay of one time step. This can also be used in `DC` analysis, if an iterative solving-method together with e.g. the Newton-Raphson algorithm is used. Any functional dependency can be specified for an independent source, as long as the function parameters have some meaningful values. This includes both current (`J`) and voltage (`E`) sources.
- plotting the results obtained from `AC` and `TRAN` analysis. Also, any user specified functional expression of the results in these analyses can be plotted. The plotting capability can be expanded to include `DC` analysis with circuit parameter sweeping (i.e. analyzing the circuit at different parameter values).

Although there is no restriction on how many nodes and branches the circuit to be simulated may have, the performance of the *HP 48SX* restricts the practical size of the circuit. The method used for setting up the simulation equations (i.e. component matrices and source vectors) is modified nodal formulation (MNF) [5]. The components supported in the current version of *Csim*, 2.61, are

- `R, G, L, M, C`, i.e. resistors, conductors, inductors, transformers and capacitors with constant values.

- `m`, i.e. a mutual inductance between two components. This is not a component, instead it specifies a dependency between two components that have been defined.

- `Y, Z`, i.e. impedances and admittances with constant complex values. For `AC` analysis at a single frequency only.

- `T`, i.e. lossless transmission lines. For `AC` analysis at a single frequency only.

- `J, E`, i.e. ideal independent current and voltage sources with functional values.

- `S, O`, i.e. short circuits and ideal operational amplifiers.

- `r, p, g, u, a, b`, i.e. current-controlled voltage sources (CCVS; `r` and `p`), voltage-controlled current sources (VCCS; `g`), voltage-controlled voltage sources (VCVS; `u`) and current-controlled current sources (CCCS; `a` and `b`).

- `y, z`, i.e. two-ports with y- or z-parameter representation. For `AC` analysis at a single frequency only.

Since we use modified nodal formulation, `E, L, M, S, O, r, u, a` and `b` all require one (or two) specified branches in addition to the nodes specified. This means that the currents of these branches are solved for and included in the result vector.

In the following sections we will discuss modified nodal formulation, the different analyses, and some underlying theory in detail. Some aspects of programming will also be discussed. The "component stamps" used in MNF are reproduced in appendix A. The derivation of the formulas for truncation errors involved in the `TRAN`

analysis methods used by *Csim* is presented in appendix B, and finally, a program listing is presented in appendix C. Together with the program listing a short manual is also provided. The theory for this tutorial has been obtained mostly from [5], but also from material used in the courses [3], [4]:

- `Ele-55.141`, Circuit Analysis 1,
- `Ele-55.142`, Circuit Analysis 2,
- `Ele-55.165`, Computer-Aided Circuit Design,

held at the Helsinki University of Technology, Faculty of Electrical Engineering.

1.1 Notation

In this presentation we will use uppercase letters in `DC` and `AC` analysis. In `TRAN` analysis all vectors are in lowercase, e.g. $\underline{w}$. Also, when indexing is used, it denotes the value of e.g. a vector at that time point. E.g. $\underline{w_1}$ refers to the value of the source vector after one time step. For time derivatives we use

$$\dot{x} = \frac{\partial x}{\partial t}, \tag{1}$$

$$\dot{x_0} = \frac{\partial x}{\partial t}|_{t=t_0} \tag{2}$$

Sometimes the notation

$$x^{'} = \frac{\partial x}{\partial t} \tag{3}$$

is also used, especially for higher order derivatives for which it is more convenient. Superscripts, such as in I^k, are used to denote the solution of the k'th iteration round, when iterative solving is required.

2 Modified nodal formulation

2.1 The set of equations

Regardless of the analysis in question, the circuit simulating problem is described by the matrix equation (which is equivalent to a set of equations)

$$\underline{\underline{T}}\,\underline{X} = \underline{W}. \tag{4}$$

This is exactly what e.g. ordinary nodal formulation [4] is based on. In that case we have the matrix equation

$$\underline{\underline{Y}}\,\underline{U} = \underline{I}. \tag{5}$$

The drawback of ordinary nodal formulation is that e.g. ideal voltage sources cannot be represented. This is one of the reasons we here decided to use modified nodal formulation, that does not have this restriction. Another solution would have been using gyrator transformation [2], which allows any circuit to be represented by voltage-controlled current sources only. In our case, matrix $\underline{\underline{T}}$ contains the contributions of the components in the circuit, and the vector $\underline{W}$ those of the independent sources. The unknowns, which in the modified nodal formulation can be both node voltages and branch currents, are represented by the vector $\underline{X}$. This means that the above set of equations can actally be separated into two different kinds of sets:

$$\underline{\underline{Y}}\,\underline{X}_U + \underline{\underline{A}}_I\,\underline{X}_I = \underline{W}_J, \tag{6}$$

$$\underline{\underline{A}}_U\,\underline{X}_U + \underline{\underline{Z}}\,\underline{X}_I = \underline{W}_E. \tag{7}$$

Here we denote

$$\underline{W} = \underline{W}_J + \underline{W}_E, \tag{8}$$

$$\underline{X} = \underline{X}_U + \underline{X}_I. \tag{9}$$

The components that have an admittance description (e.g. capacitors) belong to $\underline{\underline{Y}}$, and those that have an impedance description (e.g. inductors) belong to $\underline{\underline{Z}}$. Independent current sources belong to $\underline{W}_J$ whereas the independent voltage sources belong to $\underline{W}_E$. The matrices $\underline{\underline{A}}_I$, $\underline{\underline{A}}_U$ contain only $-1, 0, 1$-valued entries and sums of these and represent the Kirchhoff's voltage and current laws. Finally, the two vectors $\underline{X}_I$ and $\underline{X}_U$ include the unknown branch currents (for components that have an impedance description, or ideal independent voltage sources) and node voltages (for components that have an admittance description, or ideal independent current sources). The terms admittance description and impedance description will be further explained in section 2.2, where the component equations are presented. This should also make the notation used more obvious.

2.2 Component equations

Consider first the case of a simple ideal capacitor. In `DC` analysis there is no current through the capacitor, i.e. it represents an open circuit. In `AC` analysis, the current through the capacitor (with capacitance C farads) is

$$I_C = j\omega C \cdot U_C. \tag{10}$$

Thus, the higher the frequency, the larger the absolute value of the admittance of the capacitor. At $\omega = 0$ *rad/s* the absolute value of the admittance is zero. Now, let us compare this to the case of an ideal inductor. In `DC` analysis, the inductor represents a short circuit. If any `DC` voltage would be applied across an ideal short-circuit, the current through it would become infinite. If we would formulate the inductor in `AC` analysis the same way as we did with the capacitor, we would have

$$I_L = \frac{1}{j\omega L} \cdot U_L. \tag{11}$$

But when the radial frequency, ω, approaches zero (i.e. the `DC` condition), then the absolute value of the admittance of the inductor approaches infinity. Clearly, this is not a desirable situation if the same set of equations for the circuit is to be used in both `DC` and `AC` analyses. Thus, we write the equation for the inductor as

$$U_L = j\omega L \cdot I_L. \tag{12}$$

As ω approaches zero, the impedance of the inductor also approaches zero, i.e. that of a short circuit. We say that the inductor is a component with impedance description. Similarly, the capacitor is a component with admittance description. We use impedance or admittance description depending on which gives noninfinite values for real values of ω.

Since the current through the inductor is unknown, as well as the voltage across it, we need another equation in order to solve for both unknowns. The equations (Kirchhoff's voltage and current laws, KCL and KVL [4]) for the inductor are

$$U_{Lj} - U_{Lj'} - j\omega L \cdot I_L = 0, \tag{13}$$

$$I_{Lj} = I_L, \;\; I_{Lj'} = -I_L, \tag{14}$$

where I_L refers to the current through the inductor from node j to node j'. This is shown in figure 1. By doing this we obtain one single matrix equation which can be used in both `DC` as well as `AC` analysis, without any of the matrix entries becoming infinite. This same matrix equation can also be used in `TRAN` analysis, as will be seen further on. The code used for the simulator thus only requires a setup routine, that creates the matrices required, and a simple subroutine that uses these matrices for each specific analysis type.

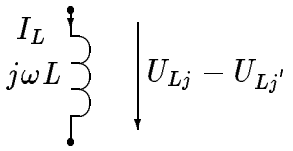


Figure 1: Ideal inductor.

What is the case of independent current and voltage sources? Clearly, for an independent current source we know the current through it, whereas the voltage across it is determined by the rest of the circuit. For an independent voltage source the situation is the opposite. It is assumed that the reader is familiar with ordinary nodal formulation [4], [5], thus the case of an independent current source is not discussed further. However, for the ideal voltage source E (let us say, between nodes j and $j^{'}$) we obtain the following equations

$$U_j - U_{j'} = E, \tag{15}$$

$$I_j = I_E, \;\; I_{j'} = -I_E. \tag{16}$$

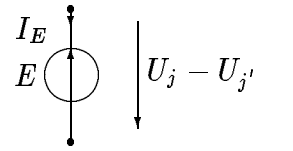


Figure 2: Ideal independent voltage source.

Here, U_j, $U_{j'}$ refer to the voltages of nodes j, $j^{'}$ and I_j to the current from node j to node $j^{'}$ (denoted as positive[1] as in the case of an inductor). This is presented in figure 2. Since the voltages of each node are included among the unknowns in ordinary nodal formulation, the first equation simply adds one element to the source vector, and a row (that has the entries 1 and -1) to the component matrix. The other equations add I_E to the vector of unknowns. This can be presented as (see also appendix A)

$$\begin{bmatrix} & & 1 \\ & & -1 \\ 1 & -1 & \end{bmatrix} \begin{bmatrix} U_j \\ U_{j'} \\ I_E \end{bmatrix} = \begin{bmatrix} \\ \\ E \end{bmatrix} + \begin{bmatrix} I_j \\ I_{j'} \\ \end{bmatrix} \tag{17}$$

As an example, consider the matrix equation generated by a simple circuit in which a resistor with conductance G mhos is connected to an ideal voltage source of

[1] This is on the left side of the equality sign in the equation system.

E volts. We have two nodes, one being ground and its potential defined as 0 volts. The two unknowns are the voltage across the resistor and the current through the ideal voltage source. We obtain

$$\begin{bmatrix} G & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} U_1 \\ I_E \end{bmatrix} = \begin{bmatrix} 0 \\ E \end{bmatrix} \tag{18}$$

which is equivalent to

$$G \cdot U_1 + 1 \cdot I_E = 0, \tag{19}$$
$$1 \cdot U_1 + 0 \cdot I_E = E. \tag{20}$$

As can be seen, the 1-valued entries in the matrix are dimensionless and represent additional terms in the equations so that the Kirchhoff's voltage and current laws are obeyed. To shed some light on the notation used in equations 6 and 7, we separate the above into two equation systems:

$$\begin{bmatrix} G & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} U_1 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ I_E \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{21}$$

$$\begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} U_1 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ I_E \end{bmatrix} = \begin{bmatrix} 0 \\ E \end{bmatrix} \tag{22}$$

where $\underline{\underline{Z}}$ contains no entries in this example (i.e. none of the components in this circuit uses impedance description). By comparing this to what was presented earlier, the meaning of the different matrices should be clear. This notation is used only so that the origin (or cause) of each entry is easy to see. For computational purposes we use $\underline{\underline{T}}$, $\underline{X}$ and $\underline{W}$.

2.3 Some programming aspects

The user interface *Csim* takes most of the program. The intention was to make *Csim* as easy and fast as possible to use. The circuit elements are entered on the "stack" of the *HP 48SX* in the following fashion:

```
{E 1 0 'IFTE(t MOD 2 > 1,-1,1)' 4}
{E 3 0 'IFTE(CV(2) > 0,-1,1)' 5}
{G 1 2 10}
{C 2 0 2}
```

This defines a simple RC-circuit with a periodic square-pulse voltage source, the second source being a digital inverter (See also appendix C for a manual).

The first task of the program is to find out how many nodes and branches there are in the circuit. This of course determines the dimension of the component matrix

and the source vector. It is assumed that the user has the nodes and the branches in some kind of order, and that they are denoted with integers in a sequence (i.e. $1, 2, 3, \ldots$). One of the nodes should be a reference node (i.e. ground), and denoted by 0. The number of a node or branch refers directly to a row/column in the matrix (or vector). When the dimension of the matrix equation is determined, the required matrix and vectors are created and loaded. Doing this, *Csim* uses "stamps" that are coded into the program (functions beginning with `put`, see appendix C). These determine what entries each component generates, and into which of the matrices. After the matrices have been loaded, the circuit is saved as a "list" into a variable named `CIR` from where it can easily be fetched for another analysis. Also, if the stack is empty when `Setup` is run, the contents of the `CIR` variable is automatically used to define the circuit. There is very little syntax checking done when running *Csim*, so the user should always make sure the circuit is properly connected and described. Except for the circuit description, all input is done interactively through the main program.

All of the steps mentioned above are done in the main program `Csim` and the `Setup` subroutine, mainly using the stack as memory storage. This makes the program look fairly complicated, when indeed it is not. It is only in the subroutines that local variables have been used. Whenever some task is done twice, a subroutine for that task has been written. Studying the code, it should not be too difficult for the user to modify it e.g. to add new components (stamps) or to create new algorithms for time-domain analysis. The user is also provided two functions, `CV(node)` and `CI(branch)` that fetch the previous solution for a node voltage or branch current. These can be used to define a source with any functional dependency of these values. However, no method to solve a nonlinear system of equations is provided with *Csim*, except for fixed point iteration (i.e. the system is solved over and over until the result supposedly converges). This method, `iterdc`, should be combined with e.g. the Newton-Raphson algorithm to yield results (some nonlinear circuits might converge with the use of fixed point iteration only).

If an analysis performes plotting, *Csim* executes the `GRAPH` command after the plot is drawn, leaving the user in the *Graphics Environment*. In order to use all the capabilities of plotting results, the user should be familiar with the *HP 48SX Graphics Environment*, which provides zooming, derivatives directly from the plot, polar plots and much more. For these capabilities, the user should refer to [1].

Since the *HP 48SX* has matrix calculation capabilities included, matrix inversion and multiplication are not problems to consider. However, to obtain an accurate solution to a matrix equation

$$\underline{\underline{A}}\,\underline{x} = \underline{b}$$

when $\underline{\underline{A}}$ is ill-conditioned, some iterative process may have to be applied. A program for this purpose is included in the *Csim* file, called `MTXSLV` (See also [1], *Advanced topics relating to matrices*). It can be used without major changes in the code.

3 DC and AC analysis

Both `DC` and `AC` analysis are solved by inverting the component matrix, which is then multiplied by the source vector to obtain the vector of unknowns. This is simple since the *HP 48SX* provides built-in functions for both inverting a matrix as well as multiplying a matrix (or vector) with another matrix (or vector). For advanced use a program for more accurate equation system solving, `MTXSLV`, is also provided with *Csim*.

The component matrix can be divided into three terms (matrices), which form $\underline{\underline{T}}$ as their sum. The first term, denoted $\underline{\underline{G}}$, contains all the real valued entries of $\underline{\underline{T}}$. Such are resistances, conductances and the $-1, 0, 1$-valued entries that represent Kirchhoff's current and voltage laws. The second term, denoted $\underline{\underline{C}}$, contains all values of $\underline{\underline{T}}$ that are multiplied by the constant s (or $j\omega$). These are generated by capacitors and inductors. Finally, the third term, denoted $\underline{\underline{C}}_c$, is used only in `AC` analysis at a single frequency point. It contains the constant valued complex entries of $\underline{\underline{T}}$. These are generated by components with constant admittance, impedance (`Y`, `Z`), two-ports presented with their y- or z-parameters (`y`, `z`) and transmission lines (`T`). Thus, we have

$$\underline{\underline{T}} = \underline{\underline{G}} + j\omega_c \cdot \underline{\underline{C}} + \underline{\underline{C}}_c. \tag{23}$$

In `DC` analysis, we use $\underline{\underline{T}}$ at $\omega = 0$ *rad/s*, i.e. $\underline{\underline{T}}_{DC} = \underline{\underline{G}}$ (since $\underline{\underline{C}}_c$ is not allowed to have any entries in `DC` analysis). The source vector, $\underline{W}$ is evaluated at $t = 0s$, $\omega = 0$ *rad/s*. The vector of unknown node voltages and branch currents is therefore

$$\underline{X}_{DC} = \underline{\underline{T}}_{DC}^{-1} \cdot \underline{W} = \underline{\underline{G}}^{-1} \cdot \underline{W}. \tag{24}$$

In `AC` analysis, $\underline{\underline{T}}$ is the sum of all three terms. The value of $j\omega \cdot \underline{\underline{C}}$ depends on ω. $\underline{W}$ is evaluated at $t = 0s$ (in case any dependency of time is specified) and ω *rad/s*. Therefore, we need to calculate the value of $j\omega \cdot \underline{\underline{C}}$ and $\underline{W}$ at each analysis point and perform the additions required to obtain $\underline{\underline{T}}$. This complex valued matrix is then inverted, and multiplied by $\underline{W}$ to give the vector of unknowns:

$$\underline{\underline{T}}_{AC}|_{\omega=\omega_c} = \underline{\underline{G}} + \underline{\underline{C}}_c + j\omega_c \cdot \underline{\underline{C}}, \tag{25}$$

$$\underline{\underline{T}}_{AC} = \underline{\underline{G}} + j\omega \cdot \underline{\underline{C}}, \tag{26}$$

$$\underline{X}_{AC} = \underline{\underline{T}}_{AC}^{-1} \cdot \underline{W}. \tag{27}$$

The first equation refers to the matrices created by *Csim* when analyzing at a certain radial frequency, ω_c. $\underline{\underline{C}}_c$ contains entries from components such as `T`, `Y`, `Z`, `y` and `z`, which are valid at ω_c only. These components should therefore only be used when performing `AC` analysis at one frequency point. The second equation refers to the general case, where the frequency dependency is known to be a real valued constant multiplied by $j\omega$.

Because of the time it takes to do the matrix inversion in `AC` analysis, the user should avoid to sweep ω through $\omega_{start} \ldots \omega_{stop}$. Instead, `AC` analysis at single points should be carried out when possible (by using the program `ac`).

3.1 Special components for AC analysis

Since we use modified nodal formulation, `DC` and `AC` analysis use exactly the same matrices for obtaining the solution. In fact, `DC` analysis (for linear circuits) is simply an `AC` analysis carried out at $\omega = 0$ *rad/s*. However, as we stated in the previous section, we have certain components the values of which are valid at a single frequency only, which are:

- `T`, a lossless transmission line.
- `Y`, a constant (complex) admittance.
- `Z`, a constant (complex) impedance.
- `y`, a two-port having a y-parameter representation.
- `z`, a two-port having a z-parameter representation.

Since the equivalent circuits that these components create have a meaning only at the specific frequency that their values are given at, the user must remember not to carry out `DC` (unless the component values are valid at $\omega = 0$ *rad/s*) or `TRAN` analysis, when any of these components appear in the circuit description. No error is returned if this is done, but the results are of course not valid. By checking that the matrix $\underline{\underline{C}}_c$ only contains $(0,0)$ valued entries, the user can make sure that other than `AC` analyses at $\omega = \omega_c$ are allowed.

For `Y` and `Z` the entries into $\underline{\underline{C}}_c$ are made as admittances, since

$$Y = Z^{-1}, \tag{28}$$

where both are complex numbers. The same approach is used for two-ports having a y- or z-parameter representation, i.e. in both cases the entries are made using the equivalent circuit for y-parameters, presented in figure 3. This can be done, since

$$\underline{\underline{y}} = \underline{\underline{z}}^{-1}. \tag{29}$$

Note that we actually must invert the complex matrix $\underline{\underline{z}}$ in order to obtain the equivalent y-parameters.

For the lossless transmission line (an expansion to lossy transmission lines can easily be done by minor changes in the code), its equivalent Π-circuit is used for

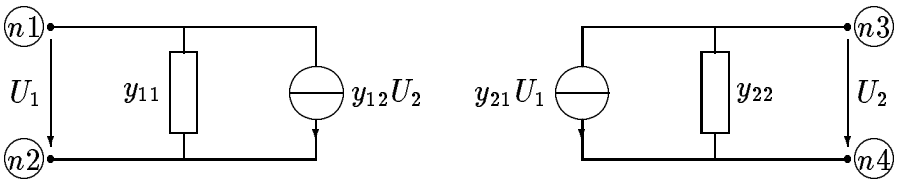


Figure 3: Equivalent circuit for y-parameter of a two-port.

matrix entries. The length of the line is given relative to the wavelength, obtained from the length of the transmission line and frequency by

$$l_\lambda = \frac{l}{\lambda} = \frac{lf}{c}, \tag{30}$$

where l is the length of the transmission line (in meters), f is the frequency (in Hertz) and c is the velocity of light in the transmission line medium (c_0 is often used, i.e. $\epsilon_r = 1$). The second parameter to be given is the characteristic impedance of the line, Z_0 (in ohms). The equivalent circuit used is presented in figure 4, which also presents the reason why the nodes 2 and 4 must be equal for `T` (see section on syntax in appendix C) .

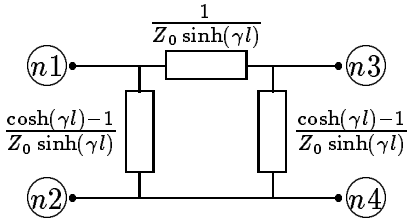


Figure 4: Equivalent circuit for lossless transmission line.

Using the notation of figure 4, we have

$$\gamma = \alpha + j\beta, \text{ where in our case} \tag{31}$$
$$\alpha = 0, \tag{32}$$
$$\beta = \frac{2\pi}{\lambda}, \text{ (TEM)}. \tag{33}$$

To obtain results in the time domain for the components presented above, equivalent circuits with memory should be introduced. This was considered by far too complicated for the purposes of *Csim*, and it would also make the analysis significantly slower. However, in `AC` analysis these components can be useful in many ways.

3.2 Nonlinear components

The fact that we do not consider nonlinear components when writing *Csim* makes the code substantially easier to develop. Another motivation for not considering nonlinear components is that this tutorial is intended for students, and thus is supposed to be easy to digest. However, for those interested in how nonlinear circuits could be treated, here is a brief introduction to that topic.

Nonlinear circuits introduce some new concepts to *Csim*. First we need an algorithm that can solve a matrix equation by iteration. This is simple, and indeed such an algorithm can be found in *Csim*, called `iterdc`. By using `iterdc` by itself, we apply what is called fixed point iteration. This means that we solve the matrix equation over and over, update the necessary entries at each iteration, and hope that the method will converge to a solution. Unfortunately, this rarely happens.

To obtain the means to solve nonlinear circuits in a general way, we need to apply some method that linearizes our problem using the derivatives of it. Such a method is e.g. the Newton-Raphson algorithm [3]. The problem is that it introduces a resistor (which can be considered to be a voltage-controlled current source), the value of which is not constant. *Csim* does not support this (using `CV()` or `CI()` does not help in this case; the delay of one time step cannot be accepted, as you will see from equation 36). In fact, since *Csim* only accepts constant valued components, some major changes in the code would be required to make nonlinear components (such as a diode) possible. Some way of updating the component matrix should be incorporated into the analysis algorithms.

Let us suppose we have some code written, that can update our component matrix in each iteration step. In that case we could consider e.g. a nonlinear conductance, for which we have

$$I = g(U_{ij}). \tag{34}$$

This we can approximate by writing

$$\Delta I = \frac{\partial g}{\partial U_{ij}} \Delta U_{ij}, \tag{35}$$

$$I^{k+1} = I^k + \frac{\partial g}{\partial U_{ij}}(U_{ij}^{k+1} - U_{ij}^k), \text{ which leads us to} \tag{36}$$

$$I^{k+1} = (I^k - \frac{\partial g}{\partial U_{ij}} U_{ij}^k) + \frac{\partial g}{\partial U_{ij}} U_{ij}^{k+1}. \tag{37}$$

The superscripts we used above refer to the iteration cycle in question. We expect the solution to improve for each iteration performed. Thus, the iteration uses the results obtained from the previous round (the first round uses the initial guesses provided by the user or simply the initial condition). There is an obvious parallelism between `DC` analysis and `TRAN` analysis when solving a nonlinear circuit; each time step taken

requires that the solution for that step is obtained by iteration. This means that a single `DC` analysis is equivalent to a `TRAN` analysis at a specific time point. The equation 37 gives us the equivalent circuit shown in figure 5.

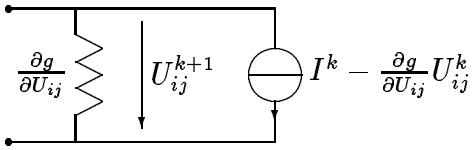


Figure 5: Equivalent linearized circuit for a nonlinear conductance.

We note that the equivalent circuit requires that we have a resistor the conductance of which is not constant. In fact, this is the only restriction imposed by *Csim*, since the current source could be defined using `J` and the functions `CV(), CI()`. The user should give the partial derivative of the conductance as a function to the simulating program, and this function should then be calculated at each analysis point (using the values of voltages or currents obtained as a solution in the previous iteration). The value obtained should then be used to update the component matrix.

4 Transient analysis

In this section we present a brief presentation on the theory and methods used to perform transient analysis with *Csim*. Most of the material in this section can be found in [3] and [5].

4.1 Simple methods for numerical integration

Consider a differential equation of the following kind:

$$\dot{x} = f(x,t), \tag{38}$$

in which, in our case, t denotes time. To solve for x we rewrite this as

$$x(t_1) = x(t_0) + \int_{t_0}^{t_1} f(x,t)dt. \tag{39}$$

Since we need to solve this using a computer, we use numerical integration with a time step Δt. We presume that the value of x at the starting point, say $t_0 = 0$, is known.

If we approximate the derivative of x at t_0, i.e. $f(x_0, t_0)$, with

$$\dot{x_0} \approx \frac{x_1 - x_0}{\Delta t}, \text{ where we have denoted} \tag{40}$$

$$\Delta t = t_{n+1} - t_n, \tag{41}$$

$$x_n = x(n\Delta t), \tag{42}$$

we obtain the *forward Euler formula*, which usually is written as

$$x_{n+1} = x_n + \Delta t \dot{x}_n. \tag{43}$$

In order to see how this can be used to solve the kind of equation as equation 38, consider the following example [5]:

$$\dot{x} = x + t^2, \text{ the exact solution of which is} \tag{44}$$

$$x = 3e^t - t^2 - 2t - 2. \tag{45}$$

We have as the initial conditions $t_0 = 0$, $x_0 = 1$ and $\dot{x}_0 = x_0 + t_0^2 = 1$. If we use a step size of $\Delta t = 0.025$, we obtain the value of x_1 from

$$x_1 = x_0 + \Delta t \dot{x}_0 = 1 + 0.025 \cdot 1 = 1.025. \tag{46}$$

The consequent values, x_n, are obtained similarly, i.e.

$$x_1 = 1.025, \tag{47}$$

$$\dot{x}_1 = 1.025 + (0.025)^2 = 1.02565, \tag{48}$$

$$x_2 = 1.025 + 0.025 \cdot 1.02565 = 1.05064, \text{ and so on.} \tag{49}$$

The calculation of a few more time steps and the comparison of the results to the exact solution are left as an exercise to the reader. However, the error will increase in magnitude for each time step taken, and it will be negative.

Another way of approximating the time derivative of x would be

$$\dot{x}_1 \approx \frac{x_1 - x_0}{\Delta t}, \text{ i.e.} \tag{50}$$

$$x_1 = x_0 + \Delta t \dot{x}_1. \tag{51}$$

By comparing this to equation 40, we note that the only difference is the index of the derivative. This method is called the *backward Euler formula*. It requires that we predict the value of $\dot{x}_1$ in order to obtain x_1. This can be done using the forward Euler formula as a predictor [5] and then applying iteration using the backward Euler formula. However, the solution used in *Csim* does not require this, as will be seen in the next section. This is due to the fact that we have restricted *Csim* to linear components. The error obtained using the backward Euler formula increases, in the problem presented, for each taken time step similarly to the forward Euler formula, but is positive. One might say that in this case the backward Euler formula *overshoots* the real solution, whereas the forward Euler formula *undershoots* it. Thus, one would expect a combination of these two methods to result in a smaller error. This is generally the case (see appendix B), and the method is called the *trapezoidal rule*, usually presented as

$$x_1 = x_0 + \frac{\Delta t}{2}(\dot{x}_1 + \dot{x}_0). \tag{52}$$

4.2 The algorithms used for transient analysis

Let us use the methods described in the previous section to solve the matrix equation

$$\underline{\dot{x}} = \underline{\underline{A}}\,\underline{x} + \underline{w}. \tag{53}$$

This implies that the vector of unknowns, $\underline{x}$, contains unknown functions of time, i.e.

$$\underline{x} = \begin{bmatrix} x_1(t) \\ x_2(t) \\ \vdots \\ x_n(t) \end{bmatrix} \tag{54}$$

Using the backward Euler formula, we have

$$
\begin{aligned}
\underline{x}_{n+1} &= \underline{x}_n + \Delta t \dot{\underline{x}}_{n+1} && (55)\\
&= \underline{x}_n + \Delta t(\underline{\underline{A}}\,\underline{x}_{n+1} + \underline{w}_{n+1}) && (56)\\
\Rightarrow & && (57)\\
(\underline{\underline{I}} - \Delta t\underline{\underline{A}})\underline{x}_{n+1} &= \underline{x}_n + \Delta t\underline{w}_{n+1} && (58)\\
\underline{x}_{n+1} &= (\underline{\underline{I}} - \Delta t\underline{\underline{A}})^{-1}(\underline{x}_n + \Delta t\underline{w}_{n+1}), && (59)
\end{aligned}
$$

where $\underline{\underline{I}}$ refers to the unity matrix. Note that no prediction is required if $\underline{w}$ is known, although we used the backward Euler formula. Similarly, using the trapezoidal rule, we obtain

$$
\underline{x}_{n+1} = (\underline{\underline{I}} - \frac{\Delta t}{2}\underline{\underline{A}})^{-1}[(\underline{\underline{I}} + \frac{\Delta t}{2}\underline{\underline{A}})\underline{x}_n + \frac{\Delta t}{2}(\underline{w}_{n+1} + \underline{w}_n)]. \tag{60}
$$

Now to discover how this relates to circuit theory, consider the equation

$$
(\underline{\underline{G}} + s\underline{\underline{C}})\,\underline{X} = \underline{W}. \tag{61}
$$

This is equivalent to the kind of equation we obtain when using modified nodal analysis. We know from Laplace transform theory [4] that multiplying with s in the complex plane is equivalent to taking the derivative in the time domain. We rewrite this as

$$
\begin{aligned}
\underline{\underline{G}}\,\underline{x} + \underline{\underline{C}}\,\dot{\underline{x}} &= \underline{w}, && (62)\\
\underline{\underline{C}}\,\dot{\underline{x}} &= \underline{w} - \underline{\underline{G}}\,\underline{x}. && (63)
\end{aligned}
$$

Using the backward Euler formula we obtain

$$
\begin{aligned}
\underline{\underline{C}}\,\underline{x}_{n+1} &= \underline{\underline{C}}\,\underline{x}_n + \Delta t\,\underline{\underline{C}}\,\dot{\underline{x}}_{n+1}\,, && (64)\\
\underline{\underline{C}}\,\underline{x}_{n+1} &= \underline{\underline{C}}\,\underline{x}_n + \Delta t\,(\underline{w}_{n+1} - \underline{\underline{G}}\,\underline{x}_{n+1}), && (65)\\
\underline{x}_{n+1} &= (\underline{\underline{C}} + \Delta t\,\underline{\underline{G}})^{-1}(\underline{\underline{C}}\,\underline{x}_n + \Delta t\underline{w}_{n+1}). && (66)
\end{aligned}
$$

Here we have

$$
\underline{x}_n = \begin{bmatrix} u_1(n\Delta t) \\ u_2(n\Delta t) \\ \vdots \\ i_1(n\Delta t) \\ i_2(n\Delta t) \\ \vdots \end{bmatrix} \tag{67}
$$

$$\underline{w}_n = \begin{bmatrix} j_1(n\Delta t) \\ j_2(n\Delta t) \\ \vdots \\ e_1(n\Delta t) \\ e_2(n\Delta t) \\ \vdots \end{bmatrix} \tag{68}$$

This means that by simply using addition, multiplication and inversion of matrices we can obtain a time domain solution for a circuit. Clearly, implementing this to the *HP 48SX* is not difficult, considering its matrix handling capabilities. Since $\underline{w}$ contains the contributions of the sources, its values at each time step are known. If we require $\underline{\underline{C}}$, $\underline{\underline{G}}$ and Δt to be constant valued, we are encountered with the inversion of a matrix only once before each analysis, and then the solution at each time step is obtained using addition and multiplication of matrices. This speeds up the solving procedure considerably, yet is sufficient for our purposes. However, it is essential to understand that this method must be done step-by-step, and requires the initial condition to be known. We have to start from the known solution (usually with $t = 0$ and all voltages and currents zero, or the `DC` solution) in order to proceed. Note that it is not necessary that $\underline{\underline{C}}$ contains entries for this method to work.

The derivation of the equation for solving in the time domain using the trapezoidal rule is left as an exercise for the reader. The result is presented below:

$$\underline{x}_{n+1} = (\underline{\underline{C}} + \frac{\Delta t}{2}\underline{\underline{G}})^{-1}[(\underline{\underline{C}} - \frac{\Delta t}{2}\underline{\underline{G}})\,\underline{x}_n + \frac{\Delta t}{2}(\underline{w}_{n+1} + \underline{w}_n)]. \tag{69}$$

Both the backward Euler formula and the trapezoidal rule are implemented in *Csim*. To see how, look at the code for the `tranBE` and `tranTR` subprograms, listed in appendix C.

5 Conclusions

The work required for programming this circuit simulator took about two weeks. However, most of the time was spent writing an user interface that would make *Csim* easy to use. In fact, the `TRAN` analysis algorithm was coded to *RPL* in just fifteen minutes!

Csim is used by numerous students interested in electrical engineering both at the Helsinki University of Technology as well as at other universities. The response from these users has mainly been positive, showing that this program does have a practical use. There are still, however, many ways in which *Csim* can be improved, making the code a nice "playground" for students who want to test their ideas. One possible way of developing *Csim* would be to rewrite the method of creating the matrix using the gyrator transform [2]. This method makes it possible to simulate any circuit just by using voltage-controlled current sources and ideal independent current sources, eliminating the need of "stamps" and branch currents.

Since this text does contain all theory used for the algorithms in *Csim*, we hope it has shown that the basics of circuit simulation are not all too difficult to understand. We hope that reading this text encourages more students to find out more about circuit simulation, since it provides many very interesting problems that can be solved in a great variety of ways.

References

[1] Hewlett Packard Co.: *HP 48SX Scientific Expandable, Owner's Manual, Volume I and II.* Hewlett Packard Co., Corvallis, 1990.

[2] Gaunholt,H., Heikkilä, P., Mannersalo, K., Porra, V., Valtonen, M.: "Gyrator Transformation - A Better Way for Modified Nodal Approach". *Proceedings of the 10'th European Conference on Circuit Theory and Design*, Vol. II, pp. 864-872, Copenhagen, July 1991.

[3] Mannersalo, K.: "Piirisynteesin numeeriset menetelmät", lecture notes, Helsinki University of Technology, Otaniemi 1991 (in Finnish).

[4] Valtonen, M: "Piirianalyysi 1 ja 2", lecture notes, Helsinki University of Technology, Otaniemi 1988 (in Finnish).

[5] Vlach, J., Singhal, K.: *Computer Methods for Circuit Analysis and Design.* Van Nostrand Reinhold Company Inc., New York, 1983.

Appendix A

The component stamps used in the program

A collection of stamps for components in modified nodal analysis can be found in [5]. Those used in *Csim* are reproduced here, together with the equivalent KCL and KVL equations:

J, an ideal independent current source of J A, between the nodes j and j' (current runs from j to j').

$$I_j = J \tag{70}$$
$$I_{j'} = -J \tag{71}$$

$\underline{W}$:	
j	$-J$
j'	J

E, an ideal independent voltage source of E V, between the nodes j and j' (j has the higher potential, branch m contains the current from j to j').

$$U_j - U_{j'} = E \tag{72}$$
$$I_j = I_E \tag{73}$$
$$I_{j'} = -I_E \tag{74}$$

$\underline{\underline{T}}$:	U_j	$U_{j'}$	I_m	$\underline{W}$:	
j			1		
j'			-1		
m	1	-1		m	E

Y, admittance of Y S (`C, G, T, y, Y, z, Z`).

$$I_j = Y(U_j - U_{j'}) \tag{75}$$
$$I_{j'} = -Y(U_j - U_{j'}) \tag{76}$$

$\underline{\underline{T}}$:	U_j	$U_{j'}$
j	Y	$-Y$
j'	$-Y$	Y

Z, impedance of Z Ω (`L, M, R, S`).

$$U_j - U_{j'} - ZI_m = 0 \tag{77}$$
$$I_j = -I_{j'} = I_m \tag{78}$$

$\underline{\underline{T}}$:	U_j	$U_{j'}$	I_m
j			1
j'			-1
m	1	-1	$-Z$

g, voltage-controlled current source, transfer conductance g S (`g, y, z`).

$$I_j = 0 \tag{79}$$
$$I_{j'} = 0 \tag{80}$$
$$I_k = g(U_j - U_{j'}) \tag{81}$$
$$I_{k'} = -g(U_j - U_{j'}) \tag{82}$$

$\underline{T}$:	U_j	$U_{j'}$
k	g	$-g$
k'	$-g$	g

u, voltage-controlled voltage source, amplification constant u.

$$u(U_{j'} - U_j) + U_k - U_{k'} = 0 \tag{83}$$
$$I_k = I_m \tag{84}$$
$$I_{k'} = -I_m \tag{85}$$

$\underline{T}$:	U_j	$U_{j'}$	U_k	$U_{k'}$	I_m
j					
j'					
k					1
k'					-1
m	$-u$	u	1	-1	

a, current-controlled current source, amplification constant a (`a, b`).

$$U_j - U_{j'} = 0 \tag{86}$$
$$I_j = -I_{j'} = I_m \tag{87}$$
$$I_k = -I_{k'} = aI_m \tag{88}$$

$\underline{T}$:	U_j	$U_{j'}$	U_k	$U_{k'}$	I_m
j					1
j'					-1
k					a
k'					$-a$
m	1	-1			

r, current-controlled voltage source, transfer resitance r (`r, p`).

$$U_j - U_{j'} = 0 \tag{89}$$
$$U_k - U_{k'} - rI_m = 0 \tag{90}$$
$$I_j = -I_{j'} = I_m \tag{91}$$
$$I_k = -I_{k'} = I_n \tag{92}$$

$\underline{\underline{T}}$:	U_j	$U_{j'}$	U_k	$U_{k'}$	I_m	I_n
j					1	
j'					-1	
k						1
k'						-1
m	1	-1				
n			1	-1	r	

O, ideal operational amplifier.

$$U_j - U_{j'} = 0 \tag{93}$$
$$I_k = -I_{k'} = I_m \tag{94}$$

$\underline{\underline{T}}$:	U_j	$U_{j'}$	U_k	$U_{k'}$	I_m
j					
j'					
k					1
k'					-1
m	1	-1			

M, transformer with inductances L_1, L_2 and mutual inductance M (`M, m`).

$$U_j - U_{j'} - sL_1 I_m - sM I_n = 0 \tag{95}$$
$$U_k - U_{k'} - sL_2 I_n - sM I_m = 0 \tag{96}$$
$$I_j = -I_{j'} = I_m \tag{97}$$
$$I_k = -I_{k'} = I_n \tag{98}$$

$\underline{\underline{T}}$:	U_j	$U_{j'}$	U_k	$U_{k'}$	I_m	I_n
j					1	
j'					-1	
k						1
k'						-1
m	1	-1			$-sL_1$	$-sM$
n			1	-1	$-sM$	$-sL_2$

Appendix B

Truncation errors involved with the methods used for transient analysis

We have considered both the backward Euler formula as well as the trapezoidal rule as methods for transient analysis. Which method is to be preferred and what are the errors involved with these methods? To answer this, let us consider a general formula for weighting the derivative at two points [5],

$$b_1\dot{x}_1 + b_0\dot{x}_0 = \frac{a_1x_1 + a_0x_0}{\Delta t}. \tag{99}$$

This formula includes both the forward and the backward Euler formula, as well as the trapezoidal rule. E.g. to obtain the backward Euler formula, choose $b_1 = 1$, $a_1 = 1 = -a_0$. Alternatively, by choosing $b_1 = b_0 = \frac{1}{2}$ and $a_1 = 1 = -a_0$, we obtain the trapezoidal rule.

Let us denote $t_1 = t_0 + \Delta t$ and write the previous formula as

$$a_1x(t_0 + \Delta t) + a_0x(t_0) - \Delta t[b_1\dot{x}(t_0 + \Delta t) + b_0\dot{x}(t_0)] = 0. \tag{100}$$

Consider now the formula for the Taylor polynomial of a function $f(x)$ at a:

$$f(x) = f(a) + \frac{f^{'}(a)}{1!}(x-a) + \frac{f^{''}(a)}{2!}(x-a)^2 + \ldots + \frac{f^n}{n!}(x-a)^n \tag{101}$$

If we write the Taylor polynomials of $x(t)$ and $\dot{x}(t)$ at t_0 and replace $x(t_0 + \Delta t)$ and $\dot{x}(t_0 + \Delta t)$ with them in the second formula, we obtain

$$a_1[x(t_0) + \frac{x^{'}(t_0)}{1!}(t_0 + \Delta t - t_0) + \frac{x^{''}(t_0)}{2!}(\Delta t)^2 + \ldots] + a_0x(t_0) -$$

$$\Delta t b_1[x^{'}(t_0) + \frac{x^{''}(t_0)}{1!}\Delta t + \frac{x^{'''}(t_0)}{2!}(\Delta t)^2 + \ldots] - \Delta t b_0 x^{'}(t_0) = 0. \tag{102}$$

Here, $\dot{x}$ and $x^{'}$ both refer to the time derivative of x, the latter notation is used for convenience. Rearranging terms, we finally obtain

$$x(t_0)(a_1 + a_0) + \Delta t\, x^{'}(t_0)(a_1 - b_1 - b_0) + (\Delta t)^2 x^{''}(t_0)(\frac{a_1}{2!} - b_1) +$$

$$(\Delta t)^3 x^{'''}(t_0)(\frac{a_1}{3!} - \frac{b_1}{2!}) + \ldots = 0. \tag{103}$$

This equation can be satisfied by letting the factors containing a_0, a_1, b_0 and b_1 equal zero. By doing so, we obtain conditions for these four parameters. A first order approximation gives

$$a_1 + a_0 = 0, \tag{104}$$

$$a_1 - b_1 - b_0 = 0. \tag{105}$$

Both forward and backward Euler formulas as well as the trapezoidal rule satisfy these conditions. A second order approximation gives the condition

$$\frac{a_1}{2} - b_1 = 0. \tag{106}$$

This is satisfied only by the trapezoidal rule. The truncation error can be defined as the value of the first term not satisfied by the chosen four parameters. For the trapezoidal rule this results in

$$-(\Delta t)^3 \frac{x'''(t_0)}{12}, \tag{107}$$

whereas for the backward Euler formula we obtain

$$-(\Delta t)^2 \frac{x''(t_0)}{2}. \tag{108}$$

Since Δt is assumed to be small it is easy to see that the trapezoidal rule is more accurate than either of the Euler formulas. However, if $x(t)$ is not well-behaved (i.e. x'' or x''' become very large at some points), or Δt is large, then this assumption does not necessarily hold.

Note that in this discussion nothing about the stability of these methods is mentioned. For a discussion on this, the reader is referred to section 9.3 in [5]. As a general comment it can be said that the trapezoidal rule is to be preferred with oscillating circuits[2]. In the case of the forward Euler formula, its instability is one of the reason it is not used at all for transient analysis. However, the method can easily be coded to *Csim*, if the user wants to try it for experimentational purposes (use the `tranBE` subroutine as a basis).

[2] Try e.g. an ideal LC resonator with an initial voltage across the capacitor. Compare the results obtained using the backward Euler formula (`Euler`= 1, see manual) and the trapezoidal rule (`Euler`= 0)

Appendix C

A short manual and the program listing

Manual

This manual explains the function of *Csim* and presents the syntax used for circuit description. It is written in ASCII-form in order to be easily distributed within the university computer networks.

```
Csim 2.61 SHORT DESCRIPTION AND MANUAL 12/17/91 (c) Per Stenius

RELEASE NOTE

This version differs from the previous (2.3, 2.5, 2.51, 2.6) in the
following:
Version 2.3:
- Lossless transmission line for AC analysis at a single freq. point. (T)
- The functions CV(node) and CI(branch) that fetch the voltage/current of
  given node/branch in the previous solution point. CV = ControlVoltage,
  can be used in other sources to define any functional dependency of a
  voltage/current in the circuit. Note that there is a delay of 'tstep'
  when using these functions.
- The program 'iterdc' for iterative dc solving to be used with nonlinear
  components.
- Additional examples of circuits in manual.
- New sample circuit.
Version 2.5, 2.51, (and previous versions):
- ABCD-matrix for two-port added.
- Minor changes in code for improved speed (during setup)
- BOTH NODES GND error removed, BOTH NODES SAME used instead.
Version 2.6 (and previous):
- ABCD, y and z matrix two-ports can be used in DC analysis if all matrix
  elements are real (i.e. the imaginary part must be zero)
- A->L converts array into list (for easier handling of matrices)

INTRODUCTION

This text describes a simple circuit simulator called Csim for the HP48.
It makes DC, AC and transient simulations and supplies the user with all
matrices used. The method used is modified nodal analysis, and thus
```

elements such as an inductor or ideal voltage source require that a current branch is also specified together with the nodes. As soon as a setup is done, single analyses can be made with the 'dc', 'ac' or 'tran' subprograms. All subprograms return the result as a vector, whereas Csim provides the user with a plot in transient and AC analysis. Note that for Csim to work correctly, the 'node' variable should be defined (either a node or branch number) when plotting a result. Note also that the RES variable (in PLOTR) should be 0. Finally, if SYM (in MODES) is not set Csim does not work correctly.

For a demo on how Csim works do the following:
- download the code (creates the directory CSIM in the current directory)
- enter the directory CSIM and enter the custom menu (the button marked CST between the PRG and VAR buttons on your HP48)
- press [Csim]:
  for: Setup? Y press: <enter>
  for: Analysis? (D,A,T) press: T <enter>
  for: Sweep range?
       :tstart:0
       :tstep:0
       :tstop:1   press: <backspc> 5 <enter> i.e. :tstop:5
  after which a time-domain plot is drawn (if you have something that you previously have plotted, press [CLRSC] before doing this).
  Press <ON> to exit the graph environment.
  Now, press 3 <left-shft> [node] <enter> and redo the above ([node] is a variable in the custom menu).
  Finally, press the CST button, <next> [CIR] [CIR->] to see the circuit description.

Here is an explanation on the custom menu of Csim (the custom menu is obtained by pressing the button marked CST on your HP48, and contains those variables and programs you need for using Csim):

Csim - the simulator program. Runs setup (if requested) and a single analysis. For ac, a single run directly from the custom menu provides a fast solution in one freq-point. When choosing T for transient analysis either tstep (time resolution) or tstop should be given. If both are given tstep is used and tstop ignored.

View - the StackView application, for easy check on components. Use ATTN (ON) to exit. You can also use the Interactive Stack the HP48 provides (see manual p.70).

node - the node or branch the value of which is wanted as a solution.
Used to GET the right value from the solution vector.

ymin, ymax
- define the picture y-axis (ac or tran analysis)

CLRSC - Clears the screen (simply the ERASE command)

outp - a program that takes a vector from the stack and returns one value.
it can be used to plot a result that is a function of the values
in the solution vector. To be edited by the user.
(default << node GET >>)

CIR-> - takes a list of lists, such as the one used to store circuit
descriptions (see also CIR, ->CIR) and puts the lists in it to
the stack (inverse to ->CIR). Usage: Press a variable containing
a circuit description (e.g. CIR) and press CIR->.

->CIR - takes the circuit description used by Csim and puts it to a list,
that can be stored in a variable.

CIR - when 'Setup' is run, the stack containing the circuit is stored in
this variable as a list. To use again, recall CIR and run 'CIR->'
(see above). Contains a sample circuit as default. Any circuit
description can be stored in a variable as a list of components
(which also are lists).

dc - single DC analysis. Requires that setup is done (the matrices are
ready). Takes no argument from stack and returns a solution vector.

ac - single AC analysis. Requires that setup is done (the matrices are
ready) and that 'w' is specified (rads/s angular freq). Takes no
argument from stack and returns a solution vector. Note: Complex
values! When A is chosen in Csim, the actual program stored by STEQ
is 'acplot', which executes 'ac' and then 'outp'. The program
'outp' should take a vector from stack and return a single number.
E.g. << node GET ABS >> would return the absolute value of a node
voltage (or branch current) that is to be plotted.

A->L - converts an array into a list.

tran - single transient analysis. Takes one time step and returns a

solution vector. The method used is trapezoidal rule. When T is chosen in Csim, the actual program stored by STEQ is 'tranBE' (or 'tranTR'). These return the result vector and call 'outp'. The program 'outp' should take a vector from stack and return a single number. E.g. << node GET >> would return the value of a node voltage (or branch current) that is to be be plotted.

Setup - setup routine for the simulator. Creates and loads the matrices needed and stores the stack as a list into CIR. Takes the circuit description from the stack as an argument (only component declarations are allowed on stack). Setup must be done once before analyzing, however, after that the matrices are ready to be used multiple times. This should be remembered e.g. when calculating a DC solution and thereafter starting a transient analysis from the obtained results. In this case running Setup a second time would zero the result vector. Setup also clears flag -3 (i.e. enables SYM), sets flag -17 and clears flag -18 (i.e. sets radians mode).

w - angular frequency (2*pi*f) rad/s.

G - conductance matrix. Contains all real valued entries, i.e. those caused by elements the values of which do not have an s or jw factor.

C - s-matrix. Contains all elements that have an s or jw factor.

Cc - constant valued complex matrix. Can be used in AC analysis only, at a single value of 'w' (angular frequency). Contains entries from Z, Y, z, y (See Section on syntax).

W - numerical values of the sources as a vector. This vector is updated in every analysis point.

Wlist - the functions representing each source as a list, from which the numerical values for 'W' are obtained.

Euler - specifies the method used in the tran analysis. If Euler = 1 then backward Euler ('tranBE') is used (faster but more inaccurate), if Euler = 0 then the trapezoidal rule is used, which is rather accurate but slower ('tranTR' and 'tran').

iterdc
- if CV() or CI() are used in DC analysis, iterative solution is

```
        required in order to obtain the correct solution. 'iterdc' can
        be used for this after setup has been done. Note that very few
        nonlinear circuits can actually be solved by iteration only.
        Usually some linearization method must also be used, e.g. the
        Newton-Raphson algorithm. Giving good initial guesses for node
        voltages and branch currents in the X vector also helps.

THE SYNTAX USED TO DESCRIBE A CIRCUIT

The syntax by which the components are entered is (NOTE! Each circuit
needs a ground node and its number is always 0 (zero)):

{R node1 node2 numval branch} - resistance [ohm]

{G node1 node2 numval}        - conductance [mho]

{C node1 node2 numval}        - capacitance [F]

{L node1 node2 numval branch} - inductance [H]

{Y node1 node2 complexnumval} - admittance with a constant complex value
                                (re,im). In DC analysis each entry must
                                be real (im = 0).

{Z node1 node2 complexnumval} - impedance with a constant complex value
                                (re,im) In DC analysis each entry must
                                be real (im = 0).

{J node1 node2 funcval}       - indep. current source

{E node1 node2 funcval branch}
                              - indep. voltage source

{S node1 node2 branch}        - short circuit (the current is fetched by
                                branch GET). Can be used to define a current
                                branch for dependent sources.

{O in+ in- out+ out- outbranch}
                              - ideal opamp (out- should be ground,
                                outbranch returns the output current)

{M l1node1 l1node2 l2node1 l2node2 l1val l2val mval l1branch l2branch}
                              - transformer i.e. two inductors (l1, l2) with
```

```
                                mutual inductance (mval). The values required
                                are the four nodes, the value of l1, l2, mval
                                [H] and the branches of l1 and l2. The dots
                                for m are at l1node1 and l2node1.

{m l1branch l2branch mval}    - mutual inductance of mval [H]. As M, but can
                                be used to define e.g. three inductances
                                that all have mutual inductances. To do
                                this, define the 3 L:s and then 3 m:s be-
                                tween them. Note that m takes the branches
                                of the L:s. Make sure you specify the in-
                                ductors the right way (the branch of L runs
                                from n1 to n2). Note also that m is not a
                                component, it merely states a dependency
                                between two L:s that should be defined se-
                                parately. No checking is done that l1branch
                                and l2branch actually belong to L:s.

{T node1 node2 node3 node4 llval Zoval}
                              - lossless transmission line (to be used in AC
                                analysis only) of length ll (in wavelengths)
                                and with the characteristic impedance Zo.
                                Note that nodes 2 and 4 must have the same
                                value (equivalent pi-circuit used).

{g node1 node2 node3 node4 numval}
                              - voltage-controlled current source i.e.
                                transconductance. The source current is
                                from node3 to node4 and the controlling
                                voltage from node1 to node2.

{r node1 node2 node3 node4 numval branch1 branch2}
                              - current-controlled voltage source. Defines
                                a short circuit between node1 and node2
                                and a controlled voltage source between
                                node3 and node4 (node3 being the positive
                                node). The controlling current runs through
                                branch1 and the current of the source is
                                fetched from branch2. Branch1 must not be
                                a previously defined branch.

{p node3 node4 numval branch1 branch2}
                              - same as r but does not define a short
```

```
                                circuit between node1 and node2. Instead
                                branch1 must be a predefined branch (e.g.
                                that of a resistor or inductor).

{a node1 node2 node3 node4 numval branch}
                              - current-controlled current source. Defines
                                a short circuit between node1 and node2
                                and a controlled current source the current
                                of which runs from node3 to node4. The
                                controlling current runs through branch
                                which must not be previously defined.

{b node3 node4 numval branch} - same as a but does not define a short
                                circuit between node1 and node2. Instead
                                branch must be a predefined branch (e.g.
                                that of a resistor or inductor). Compare
                                with p.

{u node1 node2 node3 node4 numval branch}
                              - voltage-controlled voltage source. The
                                current through the source is fetched
                                from branch.

{y node1p1 node2p1 node1p2 node2p2 y11 y12 y21 y22}
                              - a two-port with y-parameters that are
                                constant complex values (re,im). In DC
                                analysis each entry must be real (im =
                                0).

{z node1p1 node2p1 node1p2 node2p2 z11 z12 z21 z22}
                              - a two-port with z-parameters that are
                                constant complex values (re,im). In DC
                                analysis each entry must be real (im =
                                0).
{A node1p1 node2p1 node1p2 node2p2 A B C D}
                              - a two-port with ABCD-parameters that
                                are constant complex values (re,im).
                                In DC analysis each entry must be real
                                (im = 0).
FUNCTIONS

CV(node)                      - returns previously calculated value of
                                the voltage of node. In code << X node
```

GET >>

CI(branch) - returns previously calculated value of the current of branch. In code << X branch GET >>, the same as CV().

In the above, nodes and branches are integer numbers. The ground node is represented by 0. All nodes and branches should have a unique number and they should be given in order e.g. nodes 0,1,2,3 and branches 4,5,6. These numbers refer DIRECTLY to the position in the matrices/vectors. Thus the 4'th element in the result vector would be the current through branch 4 and the first element is the voltage of node 1. To help remembering the syntax, you could e.g. have a variable y with the following contents:

```
y
{y n1p1 n2p1 n1p2 n2p2 y11 y12 y21 y22}
```

NOTE: The two-port y, z and ABCD parameters are valid in DC analysis ONLY IF ALL THE ENTRIES in the corresponding matrix are real.

The following components form equivalent circuits:

```
{S 1 2 5}
{p 3 4 100 5 6} is the same as

{r 1 2 3 4 100 5 6},

{S 1 2 5}
{b 3 4 100 5} is the same as

{a 1 2 3 4 100 5},

{L 1 2 0.1 5}
{L 3 4 0.2 6}
{m 5 6 0.05} is the same as

{M 1 2 3 4 0.1 0.2 0.05 5 6}.
```

Finally, an example of usage for transient analysis:
(This is how your stack should look)

```
{E 1 0 '10*SIN(10*t)' 4}
{C 1 2 0.01}
```

```
{L 2 3 1 5}
{R 3 0 10 6}
```

This defines a RLC-circuit with nodes 1,2,3 (and ground) and current branches 4,5,6. The current through branch 4 is equal to the current through the ideal voltage source E, the current through branch 5 equals the current through the inductor L, and the current through branch 6 equals the current through the resistor R. The values of the components (which ALWAYS must be numerical) are 10 ohms, 1 henry and 0.01 farads. The voltage source has a time-dependent value (used in transient analysis). If 'node' is set to 3 the voltage over the resistor R is plotted in transient analysis. On the other hand, 'outp' could be written as << DUP 1 GET SWAP 2 GET - >> to return the voltage between nodes 1 and 2 as a result. Note that sources (E,J) may have functional values (should be suitable for the analysis requested! Time dependency for transient analysis and 'w' dependency (angular freq) for AC). When running Csim, the stack may ONLY contain component declarations!

Press the CST button and then [View]. Now press ATTN (low-left corner, i.e. ON). Press [Csim], press <enter> on "Setup? Y", press D (or T), <enter> on analysis.

An example for AC analysis (using A in Csim):

```
{E 2 0 1 3}
{G 2 1 1}
{C 1 0 1}
```

Set 'node' equal to 1 and write 'outp' equal to << node GET ABS >>. Select A on analysis and choose wstart 0 and wstop 10 (ymin = 0, ymax = 1).

Another example for AC analysis (using 'w' = 1 and 'ac'):

```
{J 0 1 10}
{G 3 0 1}
{G 2 0 1}
{Z 1 0 (0,-1)}
{M 3 1 2 1 2 1 0.5 4 5}
```

```
Setup
1 left-shft w ac
```

The currents through the transformer are the 4'th and 5'th elements in the

result vector. Here's an example involving a transistor for which we have the y-parameters y11 = 0.001, y12 = -j0.0001, y21 = 0.1 and y22 = 0.0001. Its base is at node 1, emitter at gnd (node 0) and collector at node 2.

```
{J 0 1 1}
{Z 1 0 1E3}       @ resistance of 1 kohms
{Z 2 0 1E3}
{Z 1 2 (0,-1000)} @ capacitance of -j1000 ohms
{y 1 0 2 0 1E-3 (0,-1E-4) 0.1 1E-4}
```

Setup
ac 2 GET ABS (returns |Uo/Jin| = 884.035 V/A)

For the use of the lossless transmission line we have the following example (analysis can be made at a single frequency point only, at which ll is valid):

```
{J 0 1 1}
{Z 2 0 (75,-69)}
{T 1 0 2 0 0.583 50}
```

Setup ac 1 GET

returns (25.2092, -34.5800) which is the input impedance (J = 1) of a lossless transmission line of the length 0.583 wavelengths (at some frequency) and with the characteristic impedance of 50 ohms, terminated with a load of (75,-69) ohms.

DC analysis:

```
{J 0 1 1}
{R 1 0 1E3 3}
{p 2 0 10 3 4}
```

The voltage of node 2 (the 2'nd element in the result vector) should be 10V. The current through the controlled voltage source should be 0A (the 4'th element in the result vector).

For iterative DC solving of circuits, consider these two examples:

```
{E 1 0 1 3}
{G 1 0 1}
{G 1 2 1}
```

```
{J 0 2 'SQ(CV(1))'}
```

and

```
{J 0 1 1}
{G 1 0 1}
{G 1 2 1}
{E 2 0 'SQ(CV(1))' 3}
```

Setup iterdc

The first circuit converges after only two iterations, but the second one requires several hundred to reach the exact solution ([[1] [1] [0]] in the second case). Good initial guesses may help, and also the use of e.g. the Newton-Raphson algorithm (see Vlach-Singhal; Computer Methods for Circuit Analysis and Design). Note that 'iterdc' should actually also be used at every time step in transient analysis to avoid the delay when using CV() and CI().

A simple small signal model for a transistor (B-1 C-2 E-3):

```
{R 1 3 1E3 4}
{G 2 3 0.0001}
{b 2 3 100 4}
```

A voltage source used as an digital inverter; if the voltage of node 2 is higher than 2.5 volts, then the voltage of node 3 is 0 volts, otherwise 5 volts (note that there is a delay of one 'tstep').

```
{E 3 0 'IFTE(CV(2)>2.5,0,5)' 4}
```

Including nonlinear components as such would make solving the matrix equation system a tedious process. It would also make the analysis MUCH slower. However, it could be done.

ERROR MESSAGES

These are the only error messages in this program. Note that there are not many error checking routines provided, so the user should be careful when entering the circuit description. For any strange behaviour or false results, please email me directly and explain what occurred.

SYNTAX ERROR - an error occurred in 'Setup' while Csim was loading the

matrices. Check the circuit description and the component that is first on stack. See also section on syntax.

NEGATIVE NODE NO. - a negative number was given as a node number. Check the the first component on stack.

BOTH NODES SAME - a component was specified having two nodes that were the same value. To override this, use a short-circuit (S) between these nodes.

ZERO VALUE OR BRANCH - a component with a value of zero was given or its branch number was zero (which is reserved for the ground node).

n2 MUST EQUAL n4 IN T - you have entered a transmission line in the circuit with nodes 2 and 4 not equal. This is not allowed since the equivalent pi-circuit to the transmission line is actually used.

Im{Cc} \=/ 0 IN DC - you have specified components with complex values in DC analysis.

A good way to avoid errors is to proceed systematically, e.g. in the following way:

1) Choose one reference node to be ground (GND) and set its node number to 0.

2) Assign the rest of the nodes a number each, in numerical order: 1,2,3,...

3) Find all the components that require a branch and assign each required branch a number starting from the highest node number plus one.

!) The node and the branch numbers should follow eachother in numerical order, with no 'gaps' in between (e.g. 0,1,2 are nodes and 3,4 are branches).

4) Enter the circuit description component by component. Note that the stack should only contain component descriptions! Check your stack with 'View'.

5) Decide what results you need and edit 'outp' if necessary. Also, set 'node' to the correct value.

6) Run 'Setup' once and start analyzing! Remember to run 'Setup' whenever you change your circuit or want to start from zero. Sometimes all you need to do is to edit your X vector.

FREQUENTLY ASKED QUESTIONS

Some people have had trouble with the branch currents (the direction...) so here's more on that:

In all components (J,E,L etc.) the current is defined to be FROM the first node TO the second node. Thus no matter which way you put your source (E) The value of the branch current remains the same (i.e. in the direction of the U of the source) with respective to the source. In the sample RLC circuit, when the first node of E and L is the same, the currents should be the opposite. When the second node of E is the first node of L, the currents are the same.

For E the situation looks like this:

```
           --- U=E -->

 node1 +  o->-( E )---o - node2

            I
```

for L:

```
           --- U1,2 ->

 node1    o->- Ind ---o   node2

            I
```

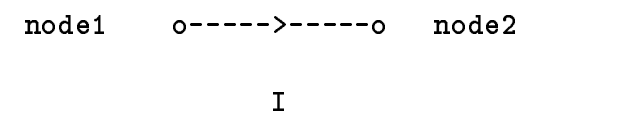
for J:

```
 node1    o->-( J )---o   node2

            I=J
```

for S:

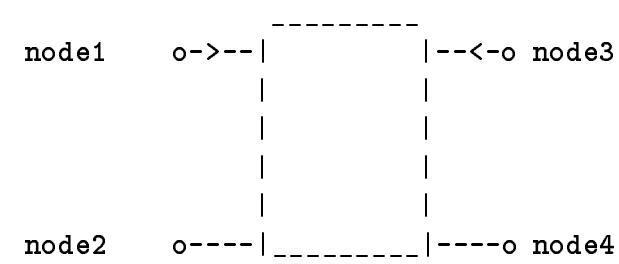
```
           --- U=0 -->
```




```
                I
```


This could be defined the other way around too, but in this case it isn't.

For two-ports, the node numbers are defined as



Note that the direction of the currents is always towards the two-port. This should be remembered when defining e.g. ideal opamps, controlled sources and two-ports with y- or z-parameter representation.

>I can't get the transient analysis to work properly unless I do
>an entire setup first. If I do a transient plot, and then repeat it, I
>get different results, unless I run setup.
>

What happens is that unless you run setup, the transient analysis continues from where it stopped (however, this time from the beginning of the screen). Thus the new beginning should match with the previous end. As you might have noticed, the transient analysis should always start from time=0. This is due to the fact that solving this problem is an iterative process.

>It would be nice to be able to specify initial values for capacitors and
>inductors.
>

This can be done. The X vector contains the starting values, and is zeroed at setup. However, nothing stops you from running a dc-analysis and then running a tran analysis without a setup in between, thereby giving the results from the dc-analysis as beginning values for the tran analysis. The X vector can also be manually edited. To do this, run setup, then press enter for analysis?, which stops the program. Edit the X, and run tran analysis without setup. The execution of setup can be avoided by answering

something else than 'Y' at 'Setup?'.

>I'm interested in any references you used, for algorithms for circuit
>solving, this is something I've never really looked into before.
>

A good place to start is to look at a book called

Computer Methods for Circuit Analysis and Design

by Jiri Vlach and Kilshore Singhal (Van Nostrand Reinhold Company 1983, ISBN 0-442-28108-0). Check out chapter 4 and 9. For transient analysis, (which is normally done by Laplace tranforms when working manually) the trapezoidal rule is pretty powerful. Also, for all the theory needed for Csim, I have written a report called

A Tutorial on Developing a Simple Circuit Simulating Program

which I can email (ps-file format) to anybody interested upon request. It is about 30 pages + 30 pages including this manual and the commented source code for Csim.

FINAL REMARK

Csim takes 10208.5 bytes when loaded, and its checksum is #45412d.

This simulator is not necessarily completely bug-free, so please report to me for any strange behaviour. Note also that the transient analysis methods are not necessarily stable for all values of time steps. Try another time range or time step if this happens. If you get the system error

```
   INV Error:
   Infinite Result
```

in any analysis mode, this usually indicates that the component matrix cannot be inverted. This error can sometimes be avoided by assigning the nodes the values 0...n and the branches the values n+1...m. If this does not help, please send me the circuit description you used.

I am happy to provide any further information on this program. Please send also some comments on its appearance, suggestions on improvement etc. Note that very little syntax checking is done (e.g. no node or branch number checks!). I hope this short manual is sufficient, if not please ask me directly via email.

```
copyright Per Stenius, Helsinki University of Technology.
          email perre@aplac.hut.fi or pstenius@otax.tky.hut.fi
```

Source code

This listing is the program file on a personal computer. It can be downloaded to the *HP 48SX* using the *KERMIT* file transfer protocol.

```
%%HP:T(3)A(R)F(.);
DIR
@ ------------------------------------------------------
@ Title   : CSIM (a simple circuit simulator for the HP48)
@ Version : 2.61
@ Author  : Per Stenius
@ LastEdit: 16.12.91
@ Copyright Per Stenius (1991)
@ ------------------------------------------------------

CST
{Csim View node ymin ymax CLRSC
outp CIR\-> CIR \->CIR Setup dc
w ac A\->L t tstep tran
X G C Cc W Wlist
Euler iterdc}

Csim
\<<
  "  Csim_HP-48 2.61


 (c) Per Stenius 1991" CLLCD 2 DISP
  1 WAIT CLLCD
  "Setup?" "Y" INPUT
  IF
  "Y" SAME
  THEN
    "Wait..." CLLCD 1 DISP
    IF
    DEPTH 0 ==
    THEN
      CIR CIR\->
    END
    Setup
  END
  "Analysis? (D, A, T)" "" INPUT
```

```
  \-> analysis
  \<<
    CASE

      analysis "D" SAME
      THEN
        dc
      END

      analysis "A" SAME
      THEN
        "Sweep range?" {":wstart:
:wstop:" { 1 0 } V } INPUT
        OBJ\-> \-> wstart wstop
        \<<
          wstop wstart - 130 /
          'wstep' STO
          wstart 'w' STO
          'acplot' STEQ
          wstart wstop XRNG
          ymin ymax YRNG
          'w' INDEP
          DRAX                @ Add ERASE to clear PICT
          {(0,0) "jw" "f(jw)"} AXES LABEL
          DRAW GRAPH
        \>>
      END

      analysis "T" SAME
      THEN
        "Sweep range?" {":tstart:0
:tstep:0
:tstop:1" { 3 0 } V } INPUT
        OBJ\->
        \-> tstart ttstep tstop
        \<<
          IF
          ttstep 0 ==
          THEN
            tstop tstart - 130 /
            'tstep' STO
          ELSE
            ttstep 'tstep' STO
```

```
          END
          tstart tstep 130 * XRNG
          ymin ymax YRNG
          't' INDEP
          DRAX                @ Add ERASE to clear PICT
          {(0,0) "t" "f(t)"} AXES LABEL
          IF
          Euler NOT
          THEN
            tstep 2 / 'tstep' STO
            'tranTR' STEQ
          ELSE
            'tranBE' STEQ
          END
          G tstep * C + INV
          'iChG' STO
          DRAW GRAPH
        \>>
      END
    END
  \>>
\>>

outp                          @ Enables user defined
\<< node GET                  @ calculations
\>>

dc
\<<                           @ DC analysis
 Wlist\->W W checkCc
 Gdc / DUP 'X' STO            @ The result vector is
\>>                           @ returned to the stack

checkCc
\<<
 Cc C\->R DUP
 IF
 CNRM NOT SWAP RNRM NOT AND
 THEN
   G + 'Gdc' STO
 ELSE
   "Im{Cc} \=/ 0 IN DC" DOERR
 END
```

```
\>>

iterdc                          @ Iterative DC analysis, max 100
\<< 0 \-> i                     @ iterations
  \<<
    DO
      X dc
    UNTIL
    ==
    'i' INCR 100 > OR
    END
    IF
    i 100 >
    THEN
      "100 ITERATIONS
CHECK CONVERGENCE" 1 DISP 1 FREEZE
    ELSE
      dc
    END
  \>>
\>>

ac
\<<
  Wlist\->W W G C w * R\->C   @ AC analysis
  Cc + / DUP 'X' STO          @ The result vector is
\>>                           @ returned to the stack

tran
\<<                           @ Trapezoidal approx.
 iChG
 W Wlist\->W W + tstep 2 / *
 C G tstep 2 / * - X * + *
 DUP 'X' STO
 t tstep + 't' STO
\>>

acplot
\<<
  ac outp                     @ outp is always called last
  wstep w + 'w' STO           @ in a plotting program
\>>
```

```
tranBE
\<<
 iChG                          @ Inverse Euler approx.
 Wlist\->W W tstep *           @ Returns the next result to stack
 C X * + *                     @ Used as default when plotting
 DUP 'X' STO outp              @ outp is always called last
\>>

tranTR
\<<                            @ Trapezoidal approx.
 iChG
 W Wlist\->W W + tstep *
 C G tstep * - X * + *
 DUP 'X' STO outp              @ outp is always called last
\>>

Wlist\->W                      @ Functional values -> numerical
\<<
 Wlist LIST\-> 1 SWAP
 START
    \->NUM
    dim ROLL
 NEXT
 dim 1 getpos \->ARRY
 'W' STO
\>>

Setup
\<<
  -3 CF                        @ Set symbolic mode
  -17 SF -18 CF                @ and radian mode
  0 't' STO
  0 'ndim' STO
  0 'bdim' STO
  DEPTH 1 SWAP
  START
     1 GETI
     \-> cmptype
     \<<
       IF
       cmptype 'm' SAME NOT  @ Not a component!
       THEN
         cmptype
```

```
      incbdim GETI
      incndim GETI
      incndim
      IF
      cmptype 'O' SAME    @ Components with 4 nodes
      cmptype 'M' SAME OR @ New two-ports: add type here!
      cmptype 'T' SAME OR
      cmptype 'g' SAME OR
      cmptype 'r' SAME OR
      cmptype 'a' SAME OR
      cmptype 'u' SAME OR
      cmptype 'y' SAME OR
      cmptype 'z' SAME OR
      cmptype 'A' SAME OR
      THEN
        GETI incndim
        GETI incndim
      END
    END
    DROP DEPTH ROLL
  \>>
NEXT
ndim bdim + 'dim' STO
[[ 0 ]] dim DUP getpos RDM DUP
'G' STO 'C' STO
[[ (0,0) ]] dim DUP getpos RDM
'Cc' STO
[[ 0 ]] dim 1 getpos RDM
DUP 'X' STO 'W' STO
1 dim
START
  0
NEXT
dim \->LIST 'Wlist' STO
DEPTH 1 SWAP
START
  IFERR
    DUP 1 GET
    loadmatrix
    DEPTH ROLL
  THEN
    "SYNTAX ERROR" DOERR
  END
```

```
  NEXT
  DEPTH \->LIST 'CIR' STO
\>>

loadmatrix
\<< \-> cmptype
  \<<
    DUP 2 GET
    2 PICK 3 GET              @ cmp n1 n2
    CASE
       cmptype 'J' SAME       @ Ideal current source
       THEN
          getval
          putJ
       END

       cmptype 'E' SAME       @ Ideal voltage source
       THEN
          getval
          getbranch
          putE
       END

       cmptype 'G' SAME       @ Conductor and capacitor
       cmptype 'C' SAME OR
       THEN
          getval
          cmptype putGC
       END

       cmptype 'R' SAME
       cmptype 'L' SAME OR    @ Resistor and inductor
       THEN
          getval
          getbranch
          IF
          cmptype 'R' SAME
          THEN
            'G'
            putRL
          ELSE
            putL
          END
```

```
END

cmptype 'Z' SAME       @ Constant valued impedance
THEN
   getval INV
   putY
END

cmptype 'Y' SAME       @ Constant valued admittance
THEN
   getval
   putY
END

cmptype 'S' SAME       @ Short-circuit
THEN
   getval              @ n1 n2 branch
   putS
END

cmptype 'O' SAME       @ Ideal opamp
THEN
   getn34
   5 PICK 6 GET        @ n1 n2 n3 n4 branch
   putO
END

cmptype 'M' SAME       @ Transformer
THEN
   getn34vb
   7 PICK 8 GET
   8 PICK 9 GET
   9 PICK 10 GET       @ n1 n2 n3 n4 l1 l2 m b1 b2
   putM
END

cmptype 'T' SAME       @ Lossless transmission line
THEN
   getn34vb            @ n1 n2 n3 n4 ll Zo
   putT
END

cmptype 'm' SAME       @ Mutual inductance
```

```
THEN
   getval
   putm                @ b1 b2 val
END

cmptype 'g' SAME       @ VCCS
THEN
   getn34
   5 PICK 6 GET        @ n1 n2 n3 n4 val
   putg
END

cmptype 'r' SAME       @ CCVS
THEN
   getn34vb
   7 PICK 8 GET        @ n1 n2 n3 n4 val b1 b2
   putr
END

cmptype 'p' SAME       @ CCVS version 2
THEN
   getvb1b2
   putp                @ n3 n4 val b1 b2
END

cmptype 'a' SAME       @ CCCS
THEN
   getn34vb            @ n1 n2 n3 n4 val branch
   puta
END

cmptype 'b' SAME       @ CCVS version 2
THEN
   getn34
   putb                @ n3 n4 val b
END

cmptype 'u' SAME       @ VCVS
THEN
   getn34vb            @ n1 n2 n3 n4 val branch
   putu
END
```

```
      cmptype 'z' SAME       @ z-parameters (two-port)
      THEN
         getn34v1234         @ n1 n2 n3 n4 y11 y12 y21 y22
         {2 2} \->ARRY INV
         ARRY\-> DROP
         puty
      END

      cmptype 'y' SAME       @ y-parameters (two-port)
      THEN
         getn34v1234         @ n1 n2 n3 n4 y11 y12 y21 y22
         puty
      END

      cmptype 'A' SAME       @ ABCD-parameters (two-port)
      THEN
         getn34v1234         @ n1 n2 n3 n4 A B C D
         ABCDtoy
         puty
      END
                             @ Add new components here!
    END
  \>>
\>>

ABCDtoy
\<< \-> A B C D
  \<<
    D B /
    C D A * B / -
    B INV NEG
    A B /
  \>>
\>>

putGC                        @ Routines to load component
\<< \-> n1 n2 value type     @ stamp into matrix (or vector)
  \<<                        @ Add new stamps here!
    value n2 n1 checknodes
    type RCL
    n1 n2 value puty2
    type STO
  \>>
```

```
\>>

putRL
\<< \-> n1 n2 value branch matr
  \<<
    branch n2 n1 checknodes   @ Enables short-circuits
    n1 n2 branch putL2
    matr RCL
    branch DUP value NEG putmatrix
    matr STO
  \>>
\>>

putJ
\<< \-> n1 n2 value
  \<<
    value n2 n1 checknodes
    Wlist DUP
    IF n1 0 >
    THEN
       n1 GET value - n1 SWAP
       PUT DUP
    END
    IF n2 0 >
    THEN
       n2 GET value + n2 SWAP PUT
    ELSE
       DROP
    END
    'Wlist' STO
  \>>
\>>

putE
\<< \-> n1 n2 value branch
  \<<
    value n2 n1 checknodes
    n1 n2 0 branch putL
    Wlist DUP
    branch GET value +
    branch SWAP PUT
    'Wlist' STO
  \>>
```

```
\>>

putM
\<< \-> n1 n2 n3 n4 l1 l2 m b1 b2
  \<<
    n1 n2 l1 b1 putL
    n3 n4 l2 b2 putL
    b1 b2 m putm
  \>>
\>>

putm
\<< \-> b1 b2 m
  \<<
    m b1 b2 checknodes
    C
    b1 b2 m NEG putmatrix
    b2 b1 m NEG putmatrix
    'C' STO
  \>>
\>>

putS
\<< \-> n1 n2 b
  \<<
    n1 n2 0 b putL
  \>>
\>>

putT
\<< \-> n1 n2 n3 n4 ll Zo
  \<<
    ll 2 \135 * * \->NUM \-> gamma
    \<<
      ll n1 n3 checknodes
      IF
      n2 n4 \139
      THEN
        "n2 MUST EQUAL n4 IN T" DOERR
      ELSE
        'INV(i*Zo*SIN(gamma))' \->NUM
        Cc
        n1 n3 4 PICK puty2
```

```
        SWAP
        'COS(gamma)-1' \->NUM *
        SWAP
        n1 n2 4 PICK puty2
        n3 n4 4 ROLL puty2
        'Cc' STO
      END
    \>>
  \>>
\>>

putg
\<< \-> n1 n2 n3 n4 value
  \<<
    value n2 n1 checknodes
    value n3 n4 checknodes
    G
    n1 n2 n3 n4 value putg2
    'G' STO
  \>>
\>>

putr
\<< \-> n1 n2 n3 n4 val b1 b2
  \<<
    b1 n2 n1 checknodes
    n1 n2 b1 putS             @ Short circuit
    n3 n4 val b1 b2 putp
  \>>
\>>

putp
\<< \-> n3 n4 val b1 b2
  \<<
    val n3 n4 checknodes
    G
    b2 n3 1 putmatrix
    b2 n4 -1 putmatrix
    b2 b1 val NEG putmatrix
    n3 b2 1 putmatrix
    n4 b2 -1 putmatrix
    'G' STO
  \>>
```

```
\>>

putu
\<< \-> n1 n2 n3 n4 value branch
  \<<
    value n2 n1 checknodes
    branch n3 n4 checknodes
    G
    branch n1 value NEG putmatrix
    branch n2 value putmatrix
    branch n3 1 putmatrix
    branch n4 -1 putmatrix
    n3 branch 1 putmatrix
    n4 branch -1 putmatrix
    'G' STO
  \>>
\>>

puta
\<< \-> n1 n2 n3 n4 val branch
  \<<
    val n2 n1 checknodes
    n1 n2 branch putS         @ Short circuit
    n3 n4 val branch putb
  \>>
\>>

putb
\<< \-> n3 n4 val branch
  \<<
    val n3 n4 checknodes
    G
    n3 branch val putmatrix
    n4 branch val NEG putmatrix
    'G' STO
  \>>
\>>

put0
\<< \-> n1 n2 n3 n4 branch
  \<<
    1 n2 n1 checknodes
    1 n3 n4 checknodes
```

```
    G
    branch n1 1 putmatrix
    branch n2 -1 putmatrix
    n3 branch 1 putmatrix
    n4 branch -1 putmatrix
    'G' STO
  \>>
\>>

putY
\<< \-> n1 n2 value
  \<<
    value n2 n1 checknodes
    Cc
    n1 n2 value puty2
    'Cc' STO
  \>>
\>>

puty
\<< \-> n1 n2 n3 n4 y11 y12 y21 y22
  \<<
    y11 n2 n1 checknodes
    y22 n2 n1 checknodes
    Cc
    n1 n2 y11 puty2
    n3 n4 y22 puty2
    n1 n2 n3 n4 y21 putg2
    n3 n4 n1 n2 y12 putg2
    'Cc' STO
  \>>
\>>

putL
\<<
 'C' putRL
\>>

putL2
\<< \-> n1 n2 branch
  \<<
    G
    n1 branch 1 putmatrix
```

```
    n2 branch -1 putmatrix
    branch n1 1 putmatrix
    branch n2 -1 putmatrix
    'G' STO
  \>>
\>>

putg2
\<< \-> n1 n2 n3 n4 value
  \<<
    n3 n1 value putmatrix
    n4 n2 value putmatrix
    n3 n2 value NEG putmatrix
    n4 n1 value NEG putmatrix
  \>>
\>>

puty2
\<< \-> n1 n2 value
  \<<
    n1 n1 value putmatrix
    n2 n2 value putmatrix
    n1 n2 value NEG putmatrix
    n2 n1 value NEG putmatrix
  \>>
\>>

putmatrix
\<< \-> row col val
  \<<
    IF
    row col AND
    THEN
       row col getpos
       DUP2                  @ matrix in level two
       GET val +
       PUT
    END
  \>>
\>>

incbdim                      @ Increase matrix dimension
\<< \-> cmptype              @ (branch)
```

```
  \<<
    IF
    cmptype 'E' SAME
    cmptype 'R' SAME OR
    cmptype 'L' SAME OR
    cmptype 'S' SAME OR
    cmptype 'O' SAME OR
    cmptype 'u' SAME OR
    cmptype 'a' SAME OR
    cmptype 'p' SAME OR
    THEN
       bdim 1 + 'bdim' STO
    ELSE
       IF
       cmptype 'M' SAME
       cmptype 'r' SAME OR
       THEN
          bdim 2 + 'bdim' STO
       END
    END
  \>>
\>>

incndim                          @ Increase matrix dimension
\<< \-> x                        @ (node)
  \<<
    IF
    x ndim >
    THEN
       x 'ndim' STO
    END
  \>>
\>>

checknodes
\<< \-> value n2 n1
  \<<
    CASE
       n1 0 <
       n2 0 < OR
       THEN
          "NEGATIVE NODE NO." DOERR
       END
```

```
       n1 n2 ==
       THEN
          "BOTH NODES SAME" DOERR
       END

       value 0 SAME
       THEN
          "ZERO VALUE OR BRANCH" DOERR
       END
    END
  \>>
\>>

getn34
\<<
 3 PICK 4 GET
 4 PICK 5 GET
\>>

getval
\<<
 3 PICK 4 GET
\>>

getvb1b2
\<<
 getn34
 5 PICK 6 GET
\>>

getn34vb
\<<
 getvb1b2
 6 PICK 7 GET
\>>

getn34v1234
\<<
 getn34vb
 7 PICK 8 GET
 8 PICK 9 GET
\>>
```

```
getbranch
\<<
 4 PICK 5 GET
\>>

getpos
\<<
 2 \->LIST
\>>

View                            @ Stack-View application
\<<
 PICT RCL \-> pict
  \<<
    PICT PURGE 1
    DEPTH 1 - 10 MIN
    DUP
    IF
      8 >
    THEN # 6d 1
    ELSE # 8d 2
    END \-> rowht tsize
    \<<
      FOR I
        PICT # 0d 65 I rowht * -
        2 \->LIST I \->STR ": "
        + I 3 + PICK \->STR +
        tsize \->GROB GOR
      NEXT
      { } PVIEW pict
      PICT STO
    \>>
  \>>
\>>

CV
\<< \-> node
  \<<
    X node GET
  \>>
\>>
```

```
CI
\<< \-> branch
  \<<
    X branch GET
  \>>
\>>

CLRSC                          @ Clears the screen
\<<
 ERASE
\>>

A\->L
\<<
 ARRY\-> DROP 4 \->LIST
\>>

CIR\->
\<<
 LIST\-> DROP
\>>

\->CIR
\<<
 DEPTH \->LIST
\>>

MTXSLV                         @ Solves a matrix equation Ax=B
\<< \-> B A                    @ Increased accuracy (iteration)
  \<<
    B A / B A
    3 PICK RSD A / +
  \>>
\>>

@ ------------------------------------------------------------
@               Default values and a sample circuit

CIR                            @ Sample circuit
{
 {E 1 0 'IFTE(t MOD 2 > 1,-1,1)' 4}
 {E 3 0 'IFTE(CV(2) > 0,-1,1)' 5}
 {G 1 2 10}
```

```
 {C 2 0 2}
}

ymin
-1

ymax
1

node
2

w                               @ Angular frequency (omega)
0

t                               @ Time
0

Euler
1
```