User Guide v-1

Circuit Solver User Guide

SemiVi LLC

Zollikon, Switzerland.

May 10, 2025

Contents

Chapter 1

Introduction

The circuit solver provided with the OESoft package performs various analyses of electronic circuit. It takes text files of circuit netlists in 'spice' format, and parses them to create a circuit matrix. The solver includes Newton's method based nonlinear solver for solving electronic circuits with nonlinear components. After a circuit is solved, all the solution quantities, i.e. all the node voltages and currents of each of the analysis (AC/DC/transient) are stored in a '.csv' file.

1.1 Features

The circuit solver has the following features.

- The circuit solver can perform a DC ramp, AC analysis at a fixed DC bias, and a transient analysis.
- A Newton's method based nonlinear solver is included. Since most of the electronic components exhibit nonlinear characteristics, a nonlinear solver is used in their circuit analyses.
- Parameters and functions can be defined and used in the main circuit or in a subcircuit. They are parsed using 'exprtk' parser.
- Values of various components can be parameterized, increasing reuse-ability of the circuit file.

1.2 Installation

OESoft currently supports software installation on various linux distributions. The software installer is available in debian package (*.deb file) and in RPM format (*.rpm file).

Note, that *if you have downloaded mkl version of the Circuit Solver*, the following package needs to be installed manually by you before installing the circuit solver from the installer package.

- Intel math kernel libraries (released in 2020 or later), which include distributions of open-mp, pardiso, etc. specific for intel processors.

The circuit solver sources mkl functions from the above installation. These functions can offer speed-up in the calculations on Intel processors. The mkl package can be downloaded from Intel website.

If the circuit solver without mkl acceleration is downloaded, then installation of the above package is not necessary.

Once `Intel math kernel libraries` are installed, download the installer on the local machine. The installer file named `CircuitSolver_amd64.deb` will appear in the `Downloads` directory. Go to the directory using `cd` command. Use the following command to install the `CircuitSolver` from the installer.

```
>> sudo apt install ./CircuitSolver_amd64.deb
```

Alternately, one may use `dpkg` to install the software and use `apt` to install missing dependencies as follows.

```
>> sudo dpkg -i ./CircuitSolver_amd64.deb
>> sudo apt install -f
```

You need to have `root` access to install the software on your machine.

1.3 Licensing

Two types of licenses can be purchased for OESoft circuit solver.

- Node-locked license: Locked to the user's machine. The circuit solver software can run only on this machine. Both, unlimited and limited node-locked licenses of the circuit solver can be purchased from OESoft.

- Server license: A server running on one of the machines in the client organization. All the other machines contact the server to fetch the software licenses. The server license is locked to the specific machine in the client organization. Only limited server licenses of the circuit solver can be purchased.

Unlimited node-locked licenses enable unlimited number of *simultaneous* executions of the circuit solver on the client machine. With *limited node-locked licenses*, only the specified number of *simultaneous* executions of the circuit solver can be performed on the machine. The node-locked license limits the usage of the circuit solver only to the user's machine.

With *limited server licenses*, only the specified number of *simultaneous* executions of the circuit solver can be performed on all the machines in the client organization together. A server license enables usage of maximum the computational capability at the client organization.

1.3.1 Purchasing the licenses

The clients can place order on OESoft website or by contacting our sales-persons. Along with the name and the organization details, the clients need to provide –

- For the node-locked licenses: Ethernet mac address of the client machine on which the software will run.
  OR

- For the server licenses: Ethernet mac address of the server machine at the client organization.

Ethernet mac address of the client machine can be obtained by running the following command on the client machine.

```
>> ifconfig eth0
```

The above command outputs various `eth0` config settings. Search for the keyword `ether` and write down the mac-id following `ether`. The mac-id mush have the following format – `xx:xx:xx:xx:xx:xx`, where `x` stands for a number (0–9) or a letter (a–f). You need to send this mac id to us to receive the licenses. The same procedure can be followed on the server machine to receive its mac-id.

Sometimes, `>> ifconfig eth0` does not give any output. In that case, please run the following command.

```
>> ifconfig -a
```

Search for the interface which begins with `enp` or `eno`. Kindly provide mac address of that interface. If interfaces which begin with both `enp` and `eno` are present, priority is given to `enp`.

We will process the request and send the license files on a secured email. The license files need to be activated on the same machines using the license key which is emailed separately. The activation procedure is described below.

1.3.2 Node-locked license activation

If you purchased one or more node-locked licenses, you will receive the following license file by secured email.

1. NodeLockedLicense_<id>_<Info>.lic, where <id> stands for license id and <Info> stands for customer identification in short.

Execute the following commands with admin access rights–

```
>> sudo cp NodeLockedLicense*.lic \
   /var/local/oesoft/licenses/0NodeLockedLicense.lic
>> sudo chmod 666 /var/local/oesoft/licenses/0NodeLockedLicense.lic
```

Any additional node-locked license purchased from us can be activated in the same way as above. Every additional activated license file can be stored at `/var/local/oesoft/licenses` by the name `<i>NodeLockedLicense.lic`, where $i = (0, 1, 2, ...49)$. The circuit solver will read the license files and lock the first available license.

1.3.3 Server license activation

If you purchased one or more server licenses, you will receive the following license files by secured email.

1. ServerLicense_<id>_<Info>.lic

Copy the above files to the server machine and execute the following commands with admin access rights–

```
>> sudo cp ServerLicense*.lic \
   /usr/share/oesoft/licenses/0ServerLicense.lic
```

Do not forget to add the following line to `.bashrc` file of each of the users.

```
export OESOFT_LICENSE_SERVER='<port>@<server>'
```

Any additional server license purchased from us can be stored in the same way as above. Every additional activated license file can be stored at `/usr/share/oesoft/licenses` by the name `<i>ActiveLicense.lic`, where $i = (0, 1, 2, ...49)$. The circuit solver will read the license files and lock the first available license.

1.3.4 Testing

User-guides of all the software provided by OESoft are stored at the location `/opt/oesoft/userguides/`.

Tutorials of all the software provided by OESoft are stored at the location `/opt/oesoft/tutorials/`.

Copy any one of the tutorial directory from `/opt/oesoft/tutorials/CircuitSol` to your working directory and run the spice circuit file in it using the following command.

```
>> CircuitSolver <filename>.cir
```

If licenses are activated, then the above command should not give any license error.

Chapter 2

Linear Components

Linear components can be separated into passive components, active components, and dependent sources. They are described below.

2.1 Passive Components

2.1.1 Resistor

A new line starting with the letter `R` defines a resistor. The first string is the component name and the next two strings are names of the nodes connected by the resistor. For example, a resistor of value r Ohms is instantiated between the nodes $n1$ and $n2$ by the following line.

```
Rb      n1      n2     r
```

It connects voltages between the two nodes by the following relationship.

$$V(n2) - V(n1) = I_{n1 \to n2} \cdot r \tag{2.1}$$

where $I_{n1 \to n2}$ is DC/AC/transient current flowing from node $n1$ to $n2$, $V(n1)$ and $V(n2)$ are DC/AC/transient voltages at $n1$ and $n2$.

Temperature response of the resistor can be defined using the following equation.

$$R(T) = R + CT1 \cdot (T - Tnom) + CT2 \cdot (T - Tnom)^2 \tag{2.2}$$

The parameters `CT1`, `CT2`, and `TEMP` can be defined together with the resistor definition as follows.

```
Rb      n1    n2    r CT1=0.01 TEMP=300.
```

2.1.2 Inductor

A new line starting with the letter `L` defines an inductor. The first string is the component name and the next two strings are names of the nodes connected by the inductor. For example, an inductor of value l Henri is instantiated between the nodes $n1$ and $n2$ by the following line.

```
La      n1    n2    l
```

In AC analysis, the inductor connects voltages between the two nodes by the following relationship.

$$V(n2) - V(n1) = I_{n1 \to n2} \cdot j\omega l \tag{2.3}$$

where $I_{n1 \to n2}$ is phasor current flowing from node $n1$ to $n2$, $V(n1)$ and $V(n2)$ are phasor voltages at $n1$ and $n2$.

In transient analysis, the inductor connects voltages between the two nodes by the following relationship.

$$V(n2) - V(n1) = l \cdot \frac{dI_{n1 \to n2}}{dt} \tag{2.4}$$

where $I_{n1 \to n2}$ is transient current flowing from node $n1$ to $n2$, $V(n1)$ and $V(n2)$ are phasor voltages at $n1$ and $n2$.

In DC analysis, the inductor is replaced by a 1nOhm resistance.

2.1.3 Mutual inductance

Each new line starting with the letter `K` creates a mutual inductance between the inductances whose names are listed after the first word. First word after the inductor names is regarded as a mutual coupling coefficient $m < 1.0$. More that two inductors can be listed as coupled inductances. In this case, the coupling coefficient between each pair of inductances is set to m.

If mutual inductance between any two inductors is defined on multiple lines, the previously defined values are overwritten by the latest value.

Mutual inductance can be used to model a transformer, or an electric motor, etc.

2.1.4 Capacitor

A new line starting with the letter `C` defines a capacitor. The first string is the component name and the next two strings are names of the nodes connected by the capacitor. For example, a capacitor of value c Henri is instantiated between the nodes $n1$ and $n2$ by the following line.

```
Ca      n1     n2    c
```

In AC analysis, the capacitor connects voltages between the two nodes by the following relationship.

$$V(n2) - V(n1) = I_{n1 \to n2} \cdot \frac{1}{j\omega c} \tag{2.5}$$

where $I_{n1 \to n2}$ is phasor current flowing from node $n1$ to $n2$, $V(n1)$ and $V(n2)$ are phasor voltages at $n1$ and $n2$.

In transient analysis, the capacitor connects voltages between the two nodes by the following relationship.

$$I_{n1 \to n2} = c \cdot \frac{d(V(n2) - V(n1))}{dt} \tag{2.6}$$

where $I_{n1 \to n2}$ is transient current flowing from node $n1$ to $n2$, $V(n1)$ and $V(n2)$ are phasor voltages at $n1$ and $n2$.

In DC analysis, the capacitor is replaced by a 10^20 Ohm resistance.

2.2 Active components

2.2.1 Voltage source

A new line starting with the letter `V` defines a voltage source. The first string is the component name and the next two strings are names

of the nodes connected by the voltage source. The strings after than define the AC, DC, and transient voltage. For example, a voltage source is instantiated between the nodes $n1$ (positive node) and $n2$ by the following line.

```
Vinput n1 n2 DC 0.0 AC 1.0 90 SIN(0.8 0.025 100.)
```

The number after `DC` sets DC voltage which is used in DC analysis. The two numbers after `AC` set phasor voltage and phase (in degrees) of the voltage source, which is used in AC analysis. The string after that specifies the type of the transient waveform together with the parameters in parentheses `(...)`.

2.2.2 Current source

A new line starting with the letter `I` defines a current source. The first string is the component name and the next two strings are names of the nodes connected by the voltage. The strings after than define the AC, DC, and transient voltage. For example, a current source is instantiated from node $n1$ to $n2$ by the following line.

```
Iinput n1 n2 DC 0.0 AC 1.0 90 SIN(0.8 0.025 100.)
```

The number after `DC` sets DC current which is used in DC analysis. The two numbers after `AC` set phasor current and phase (in degrees) of the current source, which is used in AC analysis. The string after that specifies the type of the transient waveform together with the parameters in parentheses `(...)`.

2.2.3 Transient waveform shapes

Following transient waveform types are supported.

- `SIN(p1 p2 p3 ...)` Sinusoidal pulse is used. Parameters listed in the parentheses have the following meaning.
  1. `p1` DC bias (Volts),
  2. `p2` AC peak voltage (Volts),

3. `p3` frequency (Hertz),
4. `p4` time delay (sec),
5. `p5` exponential decay factor in the prefactor (1/sec),
6. `p6` initial phase (degrees),

- `PWL(...)` Piece-wise linear waveform. List of time and voltage at that time are specified in parantheses. For example, `PWL(0. 0. 1. 1. 2. 1. 3. 0.)` specifies voltage waveform which where $V(0) = 0$, $V(1) = 1$, $V(2) = 1$, and $V(3) = 0$.
- `PULSE(...)` Rectangular pulse is used. Parameters listed in the parantheses have the following meaning.

1. `p1` voltage at logical zero,
2. `p2` voltage at logical one,
3. `p3` delay time,
4. `p4` rise time,
5. `p5` fall time,
6. `p6` pulse width of logical one,
7. `p6` pulse period.

2.3 Dependent sources

2.3.1 Voltage-controlled voltage source

A new line starting with the letter `E` defines a Voltage-controlled Voltage Source (VCVS). The first string is the component name. The next two strings are names of the positive and negative nodes of controlled-voltage source. The next two strings are the names of positive and negative controlling voltage sources in the VCVS. For example, a VCVS of voltage gain v between the nodes $N+$ and $N-$ with the controlling source between $NC+$ and $NC-$ is instantiated by the following line.

```
Eb N+ N- NC+ NC- v
```

It connects voltages between the four nodes by the following relationship.

$$V(N+) - V(N-) = v \cdot [V(NC+) - V(NC-)] \tag{2.7}$$

where $V(N+)$, $V(N-)$, $V(NC+)$, and $V(NC-)$ are DC/AC/transient voltages at respective nodes.

Nonlinear VCVS

A VCVS with nonlinear dependence on the input voltage is defined by specifying higher degree polynomial coefficients as follows.

```
Eb N+ N- NC+ NC- v1 v2 v3 ...
```

The resulting components links voltages between the four nodes as follows.

$$V_{in} = [V(NC+) - V(NC-)]\, V(N+) - V(N-) \quad = v1 \cdot V_{in} + v2 \cdot V_{in}^2 + v3 \cdot V_{in}^3 + \tag{2.8}$$

The nonlinear VCVS is solved by the nonlinear circuit solver using Newton's method.

2.3.2 Voltage-controlled Current source

A new line starting with the letter `G` defines a Voltage-controlled Current Source (VCCS). The first string is the component name. The next two strings are names of the nodes of the controlled-current source. The next two strings are the names of positive and negative controlling voltage sources in the VCCS. For example, a VCCS of transconductance g between the nodes $N+$ and $N-$ with the controlling source between $NC+$ and $NC-$ is instantiated by the following line.

```
Gb N+ N- NC+ NC- g
```

It connects voltages between the four nodes by the following relationship.

$$I_{N+\rightarrow N-} = v \cdot [V(NC+) - V(NC-)] \tag{2.9}$$

where $I_{N+\rightarrow N-}$ is the current from $N+$ to $N-$, while $V(NC+)$, and $V(NC-)$ are DC/AC/transient voltages at respective nodes.

Nonlinear VCCS

A VCCS with nonlinear dependence on the input voltage is defined by specifying higher degree polynomial coefficients as follows.

```
Gb N+ N- NC+ NC- g1 g2 g3 ...
```

The resulting components links voltages between the four nodes as follows.

$$V_{in} = [V(NC+) - V(NC-)]\, I_{N+\rightarrow N-} \quad = g1 \cdot V_{in} + g2 \cdot V_{in}^2 + g3 \cdot V_{in}^3 + \cdots \tag{2.10}$$

The nonlinear VCCS is solved by the nonlinear circuit solver using Newton's method.

2.3.3 Current-controlled voltage source

A new line starting with the letter `H` defines a Current-controlled Voltage Source (CCVS). The first string is the component name. The next two strings are names of the positive and negative nodes of controlled-voltage source. The next string is name of the voltage source from which the current is measured. For example, a CCVS of trans-resistance h between the nodes $N+$ and $N-$ with $Vname$ is voltage source name through which controlling current is measured is instantiated by the following line.

```
Hc N+ N- Vname h
```

It connects voltages between the four nodes by the following relationship.

$$V(N+) - V(N-) = h \cdot [I(\text{Vname})] \tag{2.11}$$

where $V(N+)$ and $V(N-)$ are DC/AC/transient voltages at respective nodes and $I(\text{Vname})$ is current through the voltage source `Vname`.

Nonlinear CCVS

A CCVS with nonlinear dependence on the input current is defined by specifying higher degree polynomial coefficients as follows.

```
Hc N+ N- Vname h1 h2 h3 ...
```

The resulting components links voltages between the four nodes as follows.

$$V(N+) - V(N-) = h1 \cdot I(\text{Vname}) + h2 \cdot I(\text{Vname})^2 + h3 \cdot I(\text{Vname})^3 + \cdots \tag{2.12}$$

The nonlinear CCVS is solved by the nonlinear circuit solver using Newton's method.

2.3.4 Current-controlled current source

A new line starting with the letter `F` defines a Current-controlled Current Source (CCCS). The first string is the component name. The next two strings are names of the nodes of controlled current source. The next string is name of the voltage source from which the controlling current is measured.

For example, a CCCS of trans-resistance f between the nodes $N+$ and $N-$ with $Vname$ as voltage source name through which controlling current is measured is instantiated by the following line.

```
Fc N+ N- Vname f
```

It connects voltages between the four nodes by the following relationship.

$$I_{N+\rightarrow N-} = h \cdot [I(\text{Vname})] \tag{2.13}$$

where $I_{N+\rightarrow N-}$ is the current from $N+$ to $N-$ while $I(\text{Vname})$ is current through voltage source with name $Vname$.

Nonlinear CCCS

A CCCS with nonlinear dependence on the input current is defined by specifying higher degree polynomial coefficients as follows.

```
Fc N+ N- Vname f1 f2  f3 ...
```

The resulting components links voltages between the four nodes as follows.

$$I_{N+\rightarrow N-} = f1 \cdot I(\text{Vname}) + f2 \cdot I(\text{Vname})^2 + f3 \cdot I(\text{Vname})^3 + \cdots \tag{2.14}$$

The nonlinear CCVS is solved by the nonlinear circuit solver using Newton's method.

2.4 Switches

2.4.1 Voltage-controlled switch

A new line starting with the letter `S` defines a switch. The first string is the component name. The next two strings are names of the nodes between which the switch is connected. The next two strings are the names of positive and negative controlling voltage sources in the switch. For example, a switch between the nodes $N+$ and $N-$ with the controlling source between $NC+$ and $NC-$ is instantiated by the following line.

```
.model Sw Switch(ROFF=1E12 RON=1E-2 VT=0.5 VH=0)
Sb N+ N- NC+ NC- Sw
```

When the switch is off, a resistor of resistance `ROFF` is connected between the nodes, $N+$ and $N-$. In the on-state of the, the switch resistance takes the value of `RON`.

The switch has three modes of operation depending on `VH`.

If `VH` is zero, switching happens when difference $Vin = V(NC+) - V(NC-)$ is equal to `VT`.

If `VH` is positive, the switch shows hysteresis. That is, the switch turns on when Vin rises above $VT + VH$, and turns off when Vin falls below $VT - VH$.

If `VH` is negative, the switch resistance smoothly transitions between `RON` and `ROFF`. The transition occurs between Vin of $VT + VH$ and $VT - VH$. The transition follows a linear fit to the logarithm of the switch's conduction.

2.4.2 Current-controlled switch

A new line starting with the letter `W` defines a switch. The first string is the component name. The next two strings are names of the nodes between which the switch is connected. The next string is name of the voltage source from which the controlling current is measured. For example, a switch between the nodes $N+$ and $N-$ with $Vname$ as voltage source name through which controlling current is measured, is instantiated by the following line.

```
.model Sw Switch(ROFF=1E12 RON=1E-2 IT=0.5 IH=0)
Wb N+ N- Vname Sw
```

When the switch is off, a resistor of resistance `ROFF` is connected between the nodes, $N+$ and $N-$. In the on-state of the, the switch resistance takes the value of `RON`.

The switch has three modes of operation depending on `IH`.

If `IH` is zero, switching happens when difference $I(Vname)$ is equal to `IT`.

If `IH` is positive, the switch shows hysteresis. That is, the switch turns on when $I(Vname)$ rises above $IT + IH$, and turns off when $I(Vname)$ falls below $IT - IH$.

If `IH` is negative, the switch resistance smoothly transitions between `RON` and `ROFF`. The transition occurs between $I(Vname)$ of $IT + IH$ and $IT - IH$. The transition follows a linear fit to the logarithm of the switch's conduction.

2.5 Transmission line models

2.5.1 RC Network

A new line starting with the letter `U` defines a 4-terminal component which models a RC network as a lumped model consisting of user-specified number of RC segments. The first string is the component name. The next three strings are names of the nodes on the left and the right side of the TL (see Fig. 2.2). The next string specifies the RC model.

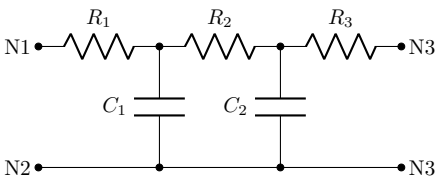


Figure 2.1: Equivalent circuit of the RC-network component having two RC-segments. Note, there are always $N + 1$ resistors and N capacitors in the network.

For example, a RC network between the nodes N1, N2, and N3 is instantiated by the following line.

```
.model RC1 TL (N=2 R=1 C=1)
Uname N1 N2 N3 RC1
```

Note, that for a given `N` segment RC network, there are always $N+1$ resistors and N capacitors in the network, to make it symmetric with respect to nodes `N1` and `N2`.

Lengths of RC segments in the network are in geometric progression, with longest segment being at the center and shortest segments at the `N1` and `N2` ends. `K` is set as the constant of geometric progression. With this arrangement, frequency response of the network remains the same independent of `N`, for a *sufficiently* large value of `N`.

All the input parameters of the RC network are listed in Table 2.1. If `ISPERL` is greater than zero, then capacitors in the RC network are modeled as diodes with depletion capacitance of `CPERL` per meter and leakage current of `ISPERL` per meter. The diodes are connected such that anodes of all the diodes are connected to `N3`.

Note: For a RC network, number of lumped RC-segments must be specified while defining the component. They cannot be changed afterwards.

Table 2.1: Input parameters of the RC network

Parameter	Description	Default	Unit
N	Number of lumped RC-segments	6	-
RPERL	Resistance per meter	1	Ω/m
CPERL	Capacitance per meter	1	F/m
L	Length of the RC network	1	meter
K	Propagation Constant	2	-
FMAX	Maximum Frequency of interest	1G	Hz
TC1	Linear temperature coefficient of resistor	0	-
TC2	Quadratic temperature coefficient of resistor	0	-
TEMP	Temperature	300	K
ISPERL	Diode leakage current per meter	0	A/m

2.5.2 Lossless TL

A new line starting with the letter `T` defines a 4-terminal component which mimics a *lossless* TL as a distributed network. The first string is the component name. The next four strings are names of the nodes on the left and the right side of the TL (see Fig. 2.2). The next string specifies the TL model.

For example, a TL between the nodes N1, N2, N3, and N4 is instantiated by the following line.

```
.model Tl1 TL (Z0=1 TD=1)
.model Tl2 TL (Z0=1 F=1 NL=1)
Tname N1 N2 N3 N4 Tl1
```

Input parameters of the lossless TL component are listed in Table 2.2. Either the parameter `TD` must be specified or `F` and `ND` must be specified. If `F` and `ND` are set, then `TD` is calculated as $\frac{NL}{F}$.

Implementation of the distributed TL model is shown in Fig. 2.2.

Table 2.2: Input parameters of the lossless TL

Parameter	Description	Default	Unit
Z0	input impedence = $\frac{L}{C}$	1	Ω
TD	delay	0	sec
F	frequency	1	Hz
ND	normalized length of TL	0	-

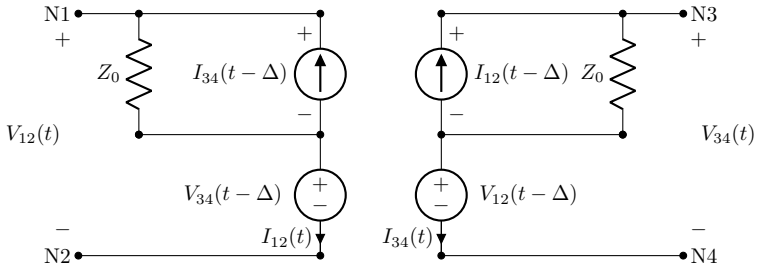


Figure 2.2: Equivalent circuit of the lossless transmission line. The parameters, input impendence ($Z0$) and delay (Δ) are specified by the user.

Table 2.3: Input parameters of the lossy TL

Parameter	Description	Default	Unit
L	Inductance per meter	1	H/m
C	Capacitance per meter	1	F/m
R	Resistance per meter	1	Ω/m
LEN	Length of TL	0	meter

2.5.3 Lossy Transmission line

A new line starting with the letter O defines a 4-terminal component which models a *lossy* TL as a distributed network. The first string is the component name. The next four strings are names of the nodes on the left and the right side of the TL. The next string specifies the TL model.

For example, a TL between the nodes N1, N2, N3, and N4 is instantiated by the following line.

```
.model Tl1 TL (L=1 C=1 R=1 LEN=1)
Oname N1 N2 N3 N4 Tl1
```

Input parameters of the lossy TL are listed in Table 2.3.

Note: Alternating Current (AC) response of lossy TL component is not modeled. In AC analysis, a lossy TL is treated as a lossless TL.

Chapter 3

Transfer-function or state-space models

Frequency response of a linear time invariant (LTI) system is given by the transfer-function. Modeling the temporal response of the system requires state-space representation of the system. This chapter describes modeling of such a system when transfer-function or state-space representation of the system is known.

3.1 Transfer-function model

A LTI system which takes a time-varying scalar signal $x(t)$ as input and outputs time-varying signal $y(t)$, is modeled by the transfer function as follows,

$$\begin{aligned} H(s) &= \frac{Y(s)}{X(s)} \\ &= \frac{n_0 + n_1 \cdot s + n_2 \cdot s^2 + \cdots}{m_0 + m_1 \cdot s + m_2 \cdot s^2 + \cdots} \\ &= \frac{\sum_0^N n_{\mathrm{i}} \cdot s^{\mathrm{i}}}{\sum_0^M m_{\mathrm{i}} \cdot s^{\mathrm{i}}} \end{aligned} \tag{3.1}$$

Here, m_i, n_i are real numbers and $s = j\omega = j2\pi f$ is frequency of the input signal. M and N are positive integers, with M is the order of the system.

3.2 State-space model

Any generic LTI system with time-varying input ($x(t)$) and time-varying output ($y(t)$), is modeled by the following set of differential equations.

$$\begin{aligned}
\dot{q_1} &= a_{1,1}q_1 + \cdots + a_{1,M}q_M + b_1 x(t) \\
&\cdots \\
\dot{q_i} &= a_{i,1}q_1 + \cdots + a_{i,M}q_M + b_i x(t) \\
&\cdots \\
\dot{q_M} &= a_{M,1}q_1 + \cdots + a_{M,M}q_M + b_M x(t) \\
y(t) &= c_1 q_1 + \cdots + c_M q_M + D \cdot x(t)
\end{aligned} \tag{3.2}$$

In the above, $a_{i,i}, b_i$, c_i, and D are real numbers, $\dot{q_i} = \frac{dq_i}{dt}$ represents time derivative of state variable $q(t)$.

The above set of differential equations can be written in matrix notation as follows.

$$\frac{d\vec{q}}{dt} = \dot{\vec{q}} = A_{M\times M} \cdot \vec{q} + B_{M\times 1} \cdot x(t) \tag{3.3}$$

$$y(t) = C_{1\times M} \cdot \vec{q} + D \cdot x(t) \tag{3.4}$$

Here, M is system order, A is a $M \times M$ square matrix, B and C are vectors of length M, and D is a scalar. $\vec{q}$ is a vector of length M of state-variables of the system.

Note: The transfer-function model described in Sec. 3.1 and state-space model described in Sec. 3.2 are two ways of describing the same LTI system. That is, specifying the system with any one of the these two models is sufficient.

3.3 Circuit model of the system

The LTI system described above is modeled in the `circuitsolver` by controlled sources. The voltage/current at the controlling source acts

as an input signal ($x(t)$) whereas the output signal $y(t)$ is the output V/I source.

Thus, if the LTI system is modeled as a VCVS (see Chapter 3, Sec. 2.3.1), input signal $x(t)$ is the voltage difference between input pins `N3` and `N4`, while signal $y(t)$ is a time-varying *voltage* source between output pins `N1` and `N2`.

For a VCCS, input signal $x(t)$ is the voltage difference between input pins `N3` and `N4`, while signal $y(t)$ is a time-varying *current* source connected between output pins `N1` and `N2`.

For a CCVS, input signal $x(t)$ is the current through the controlling voltage source, while signal $y(t)$ is a time-varying *voltage* source connected between output pins `N1` and `N2`.

For a CCCS, input signal $x(t)$ is the current through the controlling voltage source, while signal $y(t)$ is a time-varying *current* source connected between output pins `N1` and `N2`.

3.3.1 Transfer-function definition

A VCVS system, whose response is given by the transfer-function given by Eq. 3.5,

$$H(s) = \frac{Y(s)}{Xs} = \frac{n_0 + n_1 \cdot s + n_2 \cdot s^2}{m_0 + m_1 \cdot s + m_2 \cdot s^2 + m_3 \cdot s^3} \tag{3.5}$$

can be specified as follows.

```
Eb N1 N2 N3 N4 TransFunc Mmat=(m0 m1 m2 m3) Nmat=(n0 n1 n2)
```

The above line defines a VCVS component which links voltages between the four nodes by the transfer function, such that $V(N1) - V(N2) = y(t)$ and $V(N3) - V(N4) = x(t)$. The keyword `TransFunc` denotes that transfer-function of the system is provided. Note, that order of the system is defined by the number of entries in `Mmat`. Number of entries in `Nmat` is less than that in `Mmat`. Unspecified higher order entries of `Nmat` are set to 0.

3.3.2 State-space definition

If, instead of the transfer-function, state-space model of the VCVS system is known and given by Eq. 3.6.

$$\begin{aligned} \dot{q_1} &= a_{1,1}q_1 + a_{1,2}q_2 + b_1x(t) \\ \dot{q_2} &= a_{2,1}q_1 + a_{2,2}q_2 + b_2x(t) \\ y(t) &= c_1q_1 + c_2q_2 + D \cdot x(t) \end{aligned} \tag{3.6}$$

The VCVS system can be modeled by the following line.

```
Eb N1 N2 N3 N4 StateSpace Amat=(a11 a12 a21 a22) Bmat=(b1 b2) Cmat=(
```

The keyword, `StateSpace` denotes that state-space model of the system is provided. Notice, that parameter names `Amat` lists entries of matrix `A` in *row major* format. The parameter names `Bmat` and `Cmat` list entries of the respective vectors, while `D` lists the scalar.

Entry at a specific row and column in the matrix/vector can be specified by using the parameter name in the following format- <Matrix>_r_c. For example, to set entry in the 2nd row, 1st column of `A` matrix, specify it with the parameter name `A_2_1`. Notice the use of `A` instead of `Amat` as prefix. Also, notice that row and column ids start from one (instead of zero). Entry `a21` of matrix `A` will be overwritten by the value of the parameter `A_2_1`. The parameter `A_2_1` can take parametric entries as well. For example, if `par1` is defined in the subcircuit, then `A_2_1={par1}` is a valid assignment.

Chapter 4

Non-linear Components

Circuits with nonlinear components (e.g. diodes) cannot be solved by a direction matrix equation ($A \cdot x = b$) solver. In the case of nonlinear devices, the circuit matrix is a function of the solution variable (i.e. $A(x) \cdot x = b(x)$). Such a system needs to be solved by a nonlinear solver method such as 'Newton method'.

The circuit solver employs a damped Newton's method proposed by Bank-Rose in solving the nonlinear equation. For that purpose, derivatives of the node equation w.r.t. solution variables are calculated and used in the circuit matrix. Available nonlinear components are described below together with the node equations corresponding to each of the nonlinear component.

4.1 Diode component

A new line starting with the letter `D` defines a diode. The first string is the component name and the next two strings are names of node connected to the anode and cathode. The next string is the name of the diode model. The next three numbers are, respectively, diode area, initial DC voltage, and junction temperature.

For example, a diode of area scaling factor a is instantiated between anode $N+$ and cathode $N-$ by the following lines. Note, that the model 'Da1N4004' with various hyper-parameters of the diode is also

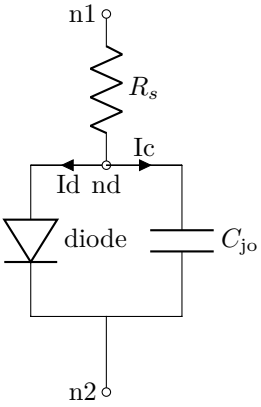


Figure 4.1: Equivalent circuit of the diode component

written.

```
.model Da1N4004 D ( IS=18.8n RS=0 BV=400 IBV=5.00u
+ CJO=1.E-9 M=0.333 N=2 TT=1E-6)
Di N+ N- Da1N4004 a
```

A resistor in series with and a capacitor in parallel to the 'ideal' diode are present in the above model. Together with them, equivalent circuit of the diode is shown in Fig. 4.1.

Parameters accepted by the 'diode' component are listed in Table 4.1 together with their meaning, default value, and the unit.

The following equations are solved in the 'ideal' diode.

$$I_{n1\to n2} = Id + Ic \tag{4.1}$$

$$Vd = V(ns) - V(n2) \tag{4.2}$$

$$V(n1) - V(n2) = RS \cdot (Id + Ic) + Vd \tag{4.3}$$

$$Id = area \cdot (I_{\mathrm{fwd}} - I_{\mathrm{rev}}) \tag{4.4}$$

$$I_{\mathrm{fwd}} = IS \cdot \left(\exp\left(\frac{Vd}{\eta V_T}\right) - 1 \right) + G_{min} \cdot (Vd) \tag{4.5}$$

Table 4.1: Input parameters of the diode component

Parameter	Description	Default	Unit
IS	saturation current	10^{-14}	A/m^2
RS	series resistance	0	Ω
N	emission coefficient	1	-
BV	breakdown voltage	10^{10}	V
IBV	current at breakdown voltage	10^{-5}	A/m^2
CJO	zero-bias junction capacitance	0	F/m^2
VJ	junction potential	0.8	V
M	grading coefficient	0.5	-
TT	transit time	0	sec
FC	coefficient in forward-bias depletion capacitance	0.5	-
EG	band-gap of the diode material	1.1	eV
TEMP	junction temperature	300.	K
XTI	IS temperature exponent	3.0	-
TBV1	BV temperature coefficient (linear)	0	1/K
TBV2	BV temperature coefficient (quadratic)	0	$1/K^2$
TRS1	RS temperature coefficient (linear)	0	1/K
TRS2	RS temperature coefficient (quadratic)	0	$1/K^2$

$$I_{\text{rev}} = IBV \cdot \left(\exp\left(\frac{-(Vd + BV)}{\eta V_T}\right) - 1 \right) - \frac{BV}{V_T} \quad (4.6)$$

Junction capacitance parallel to the diode is modeled as follows.

$$C_{\text{jo}} = area \cdot (Cd + Cj) \quad (4.7)$$

$$Cj = CJO \cdot \left(1 - \frac{Vd}{VJ}\right)^{-M} \quad (4.8)$$

$$Cd = TT \cdot \frac{IS}{\eta V_T} \cdot \exp\left(\frac{Vd}{\eta V_T}\right) \quad (4.9)$$

where $I_{n1 \to n2}$ is DC/AC/transient current flowing from node $n1$ to $n2$, $V(n1)$ and $V(n2)$ are DC/AC/transient voltages at $n1$ and $n2$. For convergence purpose, a resistor with the resistance of $1/G_{min}$ is connected in parallel with the diode.

Diode characteristics are temperature dependent. Temperature dependence if `IS`, `BV`, and `RS` is modeled using the following equations.

$$IS(T) = IS \cdot \exp\left(\left(\frac{T}{TNOM} - 1\right) \frac{EG}{N \cdot \phi_T} \right) \cdot \left(\frac{T}{TNOM}\right)^{XTI/N} \quad (4.10)$$

$$BV(T) = BV + TBV1 \cdot (T - TNOM) + TBV1 \cdot (T - TNOM)^2 \quad (4.11)$$

$$RS(T) = RS + TRS1 \cdot (T - TNOM) + TRS1 \cdot (T - TNOM)^2 \quad (4.12)$$

$$(4.13)$$

Temperature of the diode can be set by the parameter `TEMP`. If this parameter is not explicitly set, temperature dependence is disabled.

4.2 BJT component

A new line starting with the letter `Q` defines a Bipolar Junction Transistor (BJT). The first string is the component name and the next three strings are names of the nodes connected to the collector, base, and emitter of the BJT. The next string is the name of the BJT model. The next number is the BJT area factor.

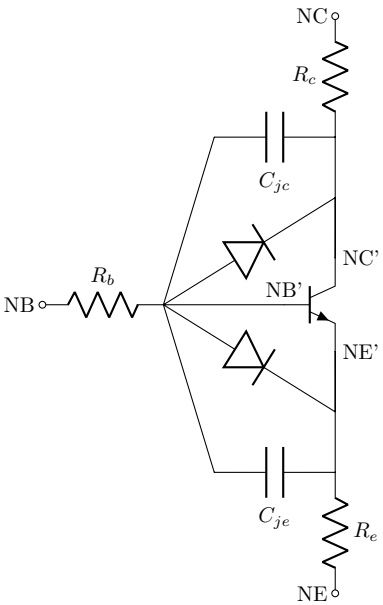


Figure 4.2: Equivalent circuit of the BJT component

For example, a BJT of area scaling factor a is instantiated with collector NC, base NB, and emitter NE in the following lines. Note, that the model '2N2222' of an npn BJT with various hyper-parameters is also written.

```
.model 2N2222 NPN (ISE=1E-14 ISC=1E-14 BF=100 BR=3)
Q1 NC NB  NE 2N2222 a
```

Resistors RC, RB, and RE are added in series with each of the collector, base, and emitter, respectively. Junction capacitors CJC and CJE are added in parallel to the 'ideal' diodes present at base-collector and base-emitter junctions, respectively. Together with them, equivalent circuit of the BJT is shown in Fig. 4.2.

Parameters accepted by the 'BJT' component are listed in Table 4.2 together with their meaning, default value, and the unit.

The ideal BJT in Fig. 4.2 is modeled by the Ebers-Mol equations given below.

$$I_{NB'\to NE'} = area \cdot IS \cdot \left(\exp\left(\frac{V(NB) - V(NE)}{\eta_F V_T}\right) - 1\right) - \alpha_{\mathrm{R}} \cdot I_{NB'\to NC'} \tag{4.14a}$$

$$I_{NB'\to NC'} = area \cdot IS \cdot \left(\exp\left(\frac{V(NB) - V(NC)}{\eta_R V_T}\right) - 1\right) - \alpha_{\mathrm{F}} \cdot I_{NB'\to NE'} \tag{4.14b}$$

Note, that the above equations are valid for 'NPN' BJT. For a PNP BJT, the following equations are solved.

$$I_{NE'\to NB'} = area \cdot IS \cdot \left(\exp\left(\frac{V(NE) - V(NB)}{\eta_F V_T}\right) - 1\right) - \alpha_{\mathrm{R}} \cdot I_{NC'\to NB'} \tag{4.15a}$$

$$I_{NC'\to NB'} = area \cdot IS \cdot \left(\exp\left(\frac{V(NC) - V(NB)}{\eta_R V_T}\right) - 1\right) - \alpha_{\mathrm{F}} \cdot I_{NE'\to NB'} \tag{4.15b}$$

Ideal diodes in Fig. 4.2 represent leakage current at B-C and B-E junctions. This leakage current does not take part in BJT action. Currents through these diodes are modeled by the following equations.

$$Vbe = V(NB') - V(NE') \tag{4.16}$$

$$I_{\mathrm{de}} = area \cdot ISE \cdot \left(\exp\left(\frac{Vbe}{\eta_E V_T}\right) - 1\right) + G_{min} \cdot (Vbe) \tag{4.17}$$

$$Vbc = V(NB') - V(NC') \tag{4.18}$$

$$I_{\mathrm{dc}} = area \cdot ISC \cdot \left(\exp\left(\frac{Vbc}{\eta_C V_T}\right) - 1\right) + G_{min} \cdot (Vbc) \tag{4.19}$$

Note, that the above equations are valid for 'NPN' BJT. Polarities of the diodes are reverse in a 'PNP' BJT.

Table 4.2: Input parameters of the BJT component

Parameter	Description	Default	Unit
IS	transport saturation current	10^{-14}	A/m^2
ISE	B-E leakage saturation current	0	A/m^2
ISC	B-C leakage saturation current	0	A/m^2
BF	ideal forward beta	100	A
BR	ideal reverse beta	100	A
NF	forward emission coefficient	1	A
NR	reverse emission coefficient	1	A
RE	emitter series resistance	0	Ω
RC	collector series resistance	0	Ω
RB	base series resistance	0	Ω
NE	B-E leakage emission coefficient	1.5	-
NC	B-C leakage emission coefficient	1.5	-
VJE	B-E junction potential	0.8	V
VJC	B-C junction potential	0.8	V
CJE	B-E zero-bias depletion capacitance	0	F/m^2
CJC	B-C zero-bias depletion capacitance	0	F/m^2
TF	forward transit time	0	sec
TR	reverse transit time	0	sec
MJE	B-E junction grading coeff.	0.5	-
MJC	B-C junction grading coeff.	0.5	-
FC	coefficient in forward-bias depletion capacitance	0.5	-
EG	band-gap of the diode material	1.1	eV
TEMP	junction temperature	300.	K
XTI	ISE and ISC temperature exponent	3.0	-
XTB	BF and BR temperature exponent	0	-
TRE1	RE temperature coefficient (linear)	0	1/K
TRE2	RE temperature coefficient (quadratic)	0	$1/K^2$
TRB1	RB temperature coefficient (linear)	0	1/K
TRB2	RB temperature coefficient (quadratic)	0	$1/K^2$
TRC1	RE temperature coefficient (linear)	0	1/K
TRC2	RE temperature coefficient (quadratic)	0	$1/K^2$

The semiconductor junction capacitances C_{je} and C_{jc} are modeled as follows.

$$C_{\text{je}} = area \cdot (Cde + Cje') \tag{4.20}$$

$$Cje' = CJE \cdot \left(1 - \frac{Vd}{VJE}\right)^{-MJE} \tag{4.21}$$

$$Cde = TF \cdot \frac{ISE}{\eta_E V_T} \cdot \exp\left(\frac{Vd}{\eta_E V_T}\right) \tag{4.22}$$

$$C_{\text{jc}} = area \cdot (Cdc + Cjc') \tag{4.23}$$

$$Cjc' = CJC \cdot \left(1 - \frac{Vd}{VJC}\right)^{-MJC} \tag{4.24}$$

$$Cdc = TF \cdot \frac{ISC}{\eta_C V_T} \cdot \exp\left(\frac{Vd}{\eta_C V_T}\right) \tag{4.25}$$

Temperature dependence of various BJT parameters has been modeled using the following equations. These equations are used when the BJT temperature is set by setting `TEMP` parameter.

$$IS(T) = IS \cdot \exp\left(\left(\frac{T}{TNOM} - 1\right)\frac{EG}{N \cdot \phi_T}\right) \cdot \left(\frac{T}{TNOM}\right)^{XTI} \tag{4.26}$$

$$ISE(T) = ISE \cdot \exp\left(\left(\frac{T}{TNOM} - 1\right)\frac{EG}{NE \cdot \phi_T}\right) \cdot \left(\frac{T}{TNOM}\right)^{XTI/NE} \tag{4.27}$$

$$ISC(T) = ISC \cdot \exp\left(\left(\frac{T}{TNOM} - 1\right)\frac{EG}{NC \cdot \phi_T}\right) \cdot \left(\frac{T}{TNOM}\right)^{XTI/NC} \tag{4.28}$$

$$BF(T) = BF \cdot \left(\frac{T}{TNOM}\right)^{XTB} \tag{4.29}$$

$$BR(T) = BR \cdot \left(\frac{T}{TNOM}\right)^{XTB} \tag{4.30}$$

$$RE(T) = RE + TRE1 \cdot (T - TNOM) + TRE2 \cdot (T - TNOM)^2 \tag{4.31}$$

$$RB(T) = RB + TRB1 \cdot (T - TNOM) + TRB2 \cdot (T - TNOM)^2 \tag{4.32}$$

$$RC(T) = RC + TRC1 \cdot (T - TNOM) + TRC2 \cdot (T - TNOM)^2 \tag{4.33}$$

4.3 JFET component

A new line starting with the letter `J` defines a Junction Field Effect Transistor (JFET). The first string is the component name and the next four strings are names of the nodes connected to the drain, gate, and source of the JFET. The next string is the name of the JFET model. The next number is the JFET area factor.

For example, a JFET of area scaling factor *area* is instantiated with drain ND, gate NG, and source NS in the following lines. Note, that the model 'njf_depl' of an n-channel JFET with various hyper-parameters is also written.

```
.model njf_depl NJF (KP=200u VTO=0.6 PHI=0.6 GAMMA=0)
J1 ND NG NS njf_depl
```

Resistors `RD`, and `RS` are added in series with each of the drain, and source, respectively. Junction capacitors `CGS` and `CGD` are added in parallel to the 'ideal' diodes present at gate-source and gate-drain junctions, respectively. Together with them, equivalent circuit of the JFET is shown in Fig. 4.3.

Parameters accepted by the 'JFET' component are listed in Table 4.4 together with their meaning, default value, and the unit.

Following equations model an ideal n-channel JFET.

$$V_{ds} = V(ND') - V(NS') \tag{4.34}$$

$$V_{ov} = V(NG') - V(NS') - VTO \tag{4.35}$$

$$I_{ND' \to NS'} = Id + Iii \tag{4.36}$$

$$I_{ND' \to NG'} = Iii \tag{4.37}$$

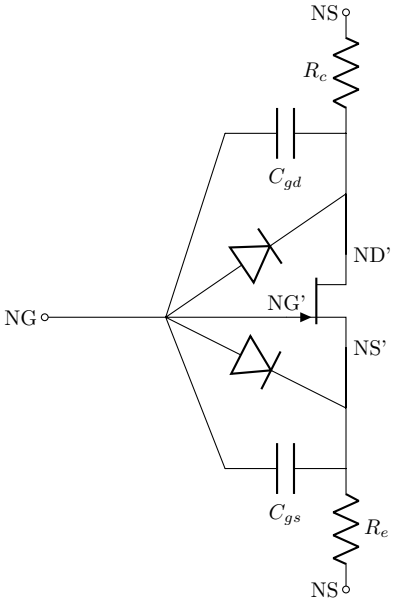


Figure 4.3: Equivalent circuit of the n-channel JFET component

Table 4.3: Input parameters of the JFET component

Parameter	Description	Default	Unit
VTO	threshold voltage	$-1.$	
BETA	trans-conductance parameter	0	$A/V^2/r$
LAMBDA	channel-length modulation	0	$A/V^2/r$
ALPHA	G-D junction ionization coefficient	0	V^{-1}
VK	G-D junction ionization voltage	0	V
ISAT	leakage current in sub-threshold region	10^{-14}	A
IS	G-S, G-D diode saturation current	1	A/m^2
RS	source series resistance	0	Ω
RD	drain series resistance	0	Ω
N	G-S, G-D diode emission coefficient	1.5	-
PB	G-S, G-D junction potential	0.8	V
CGD	G-D zero-bias depletion capacitance	0	F/m^2
CGS	G-S zero-bias depletion capacitance	0	F/m^2
TT	transit time	0	sec
MJ	G-D, G-S junction grading coeff.	0.5	-
FC	coefficient in forward-bias depletion capacitance	0.5	-
TEMP	JFET temperature	300	K
XTI	IS diode temperature exponent	3.	-
BETATCE	trans-conductance temperature coeff.	0	1/K
VTOTC	threshold voltage temperature coeff.	0	1/K
TR1	RD and RS linear temperature coeff	0	1/K
TR2	RD and RS quadratic temperature coeff	0	$1/K^2$

$$Id = \begin{cases} IS \cdot \exp\left(\frac{V_{ov}}{V_T}\right) & \text{if } V_{ov} < 0 \\ BETA \cdot V_{ds} \cdot (V_{ov} - \frac{1}{2} V_{ds}) & \text{if } V_{ov} > V_{ds} \\ \frac{1}{2} BETA \cdot V_{ov}^2 \cdot (1 + \lambda (V_{ds} - V_{ov})) & \text{if } V_{ov} < V_{ds} \end{cases} \tag{4.38}$$

$$Ii = \begin{cases} Id \cdot ALPHA \cdot (V_{ds} - V_{ov}) \cdot \exp\left(-\frac{VK}{V_{ds} - V_{ov}}\right) & \text{if } 0 < V_{ov} < V_{ds} \\ 0 & \text{otherwise} \end{cases} \tag{4.39}$$

(4.40)

Additionally, ideal diodes in Fig. 4.3 represent leakage current at Gate-Drain and Gate-Source junctions. Currents through these diodes are modeled by the following equations.

$$Vgs = V(NG') - V(NS') \tag{4.41}$$

$$I_{\text{gs}} = area \cdot IS \cdot \left(\exp\left(\frac{Vgs}{\eta V_T}\right) - 1\right) + G_{min} \cdot (Vgs) \tag{4.42}$$

$$Vgd = V(NG') - V(ND') \tag{4.43}$$

$$I_{\text{gd}} = area \cdot IS \cdot \left(\exp\left(\frac{Vgd}{\eta V_T}\right) - 1\right) + G_{min} \cdot (Vgd) \tag{4.44}$$

Note, that the above equations are valid for n-channel JFET. Polarities of the diodes are reverse in a p-channel JFET.

The semiconductor junction capacitances C_{gd} and C_{gs} are modeled as follows.

$$C_{\text{gd}} = area \cdot (Cgd1 + Cgd2) \tag{4.45}$$

$$Cgd1 = CGD \cdot \left(1 - \frac{Vd}{PB}\right)^{-M} \tag{4.46}$$

$$Cgd2 = TF \cdot \frac{IS}{\eta V_T} \cdot \exp\left(\frac{Vgd}{\eta V_T}\right) \tag{4.47}$$

$$C_{\text{gs}} = area \cdot (Cgs1 + Cgs2) \tag{4.48}$$

$$Cgs1 = CGS \cdot \left(1 - \frac{Vd}{PB}\right)^{-M} \tag{4.49}$$

$$Cgs2 = TF \cdot \frac{IS}{\eta V_T} \cdot \exp\left(\frac{Vgs}{\eta V_T}\right) \tag{4.50}$$

Temperature dependence of various JFET parameters has been modeled using the following equations. These equations are used when the JFET temperature is set by setting `TEMP` parameter.

$$VTO(T) = VTO + VTOTC \cdot (T - TNOM) \tag{4.51}$$

$$BETA(T) = BETA \cdot 1.01^{BETATCE \cdot (T - TNOM)} \tag{4.52}$$

$$IS(T) = IS \cdot \exp\left(\left(\frac{T}{TNOM} - 1\right)\frac{EG}{N \cdot \phi_T}\right) \cdot \left(\frac{T}{TNOM}\right)^{XTI} \tag{4.53}$$

$$RS(T) = RS + TR1 \cdot (T - TNOM) + TR2 \cdot (T - TNOM)^2 \tag{4.54}$$

$$RD(T) = RD + TR1 \cdot (T - TNOM) + TR2 \cdot (T - TNOM)^2 \tag{4.55}$$

In inverted mode, that is, $V_{\mathrm{ds}} < 0$, source and drain in the normal mode are switched.

4.4 MOSFET component

A new line starting with the letter `M` defines a Metal-Oxide Semiconductor Field Effect Transistor (MOSFET). The first string is the component name and the next four strings are names of the nodes connected to the drain, gate, source, and substrate of the MOSFET. The next string is the name of the MOSFET model. The next number is the MOSFET area factor.

For example, a MOSFET of area scaling factor a is instantiated with drain ND, gate NG, source NS, and substrate NB in the following lines. Note, that the model 'nmos_depl' of an n-channel MOSFET with various hyperparameters is also written.

```
.model nmos_depl NMOS (KP=200u VTO=0.6 PHI=0.6 GAMMA=0)
```

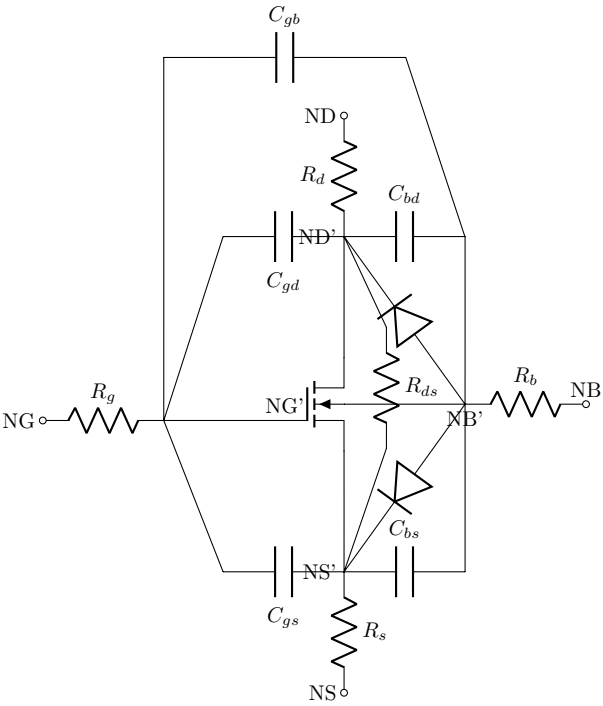


Figure 4.4: Equivalent circuit of the n-channel MOSFET component

```
Q1 ND NG NS NB nmos_depl
```

Resistors `RG`, `RD`, `RS`, and `RB` are added in series with each of the gate, drain, source, and substrate, respectively. Junction capacitors `CBS` and `CBD` are added in parallel to the 'ideal' diodes present at substrate-source and substrate-drain junctions, respectively. Together with them, equivalent circuit of the MOSFET is shown in Fig. 4.4.

Parameters accepted by the 'MOSFET' component are listed in Table 4.4 together with their meaning, default value, and the unit.

Following equations model an ideal n-channel MOSFET.

$$V_{bs} = V(NB') - V(NS') \tag{4.56}$$

Table 4.4: Input parameters of the MOSFET component

Parameter	Description	Default	Unit
L	channel length	1.	m
W	channel width	1.	m
AS	Source diffusion area	0.	m^2
AD	Drain diffusion area	0.	m^2
VTO	zero-bias threshold voltage	1.	V/K
KP	trans-conductance coefficient	0	$A/V^2/r$
LAMBDA	channel-length modulation	0	$A/V^2/r$
GAMMA	bulk threshold parameter	0	V
PHI	surface potential	0.6	V
ISAT	sub-threshold leakage current	10^{-14}	A/m^2
IS	G-S, G-D diode saturation current	1	A/m^2
RG	gate series resistance	0	Ω
RS	source series resistance	0	Ω
RD	drain series resistance	0	Ω
RDS	S-D leakage resistance	10^9	Ω
RB	bulk series resistance	0	Ω
RSH	D, S diffusion series resistance	0	Ω/m^2
N	B-S, B-D diode emission coefficient	1.5	-
PB	B-S, B-D junction potential	0.8	V
CBD	B-D zero-bias depletion capacitance	0	F
CBS	B-S zero-bias depletion capacitance	0	F
CJ	bulk planar 0-bias depletion capacitance	0	F/m^2
TT	B-D, B-S diodes transit time	0	sec
MJ	B-D, B-S junction grading coeff.	0.5	-
FC	coefficient in forward-bias depletion capacitance	0.5	-
TEMP	MOSFET temperature	300	K
XTI	IS diode temperature exponent	3.0	-
TCV	threshold voltage temperature coeff.	0	1/K
BEX	mobility temperature exponent.	0	1/K
TR1	RD and RS linear temperature coeff	0	1/K
TR2	RD and RS quadratic temperature coeff	0	$1/K^2$
TRG	RG linear temperature coeff	0	1/K
TRB	RB linear temperature coeff	0	1/K

$$V_{ds} = V(ND') - V(NS') \tag{4.57}$$

$$V_{th} = \begin{cases} Vt0 + \gamma\left(\sqrt{2\phi - V_{bs}} - \sqrt{2\phi}\right) & \text{if } V_{bs} < 2\phi. \\ Vt0 & \text{otherwise.} \end{cases} \tag{4.58}$$

$$V_{ov} = V(NG') - V(NS') - V_{th} \tag{4.59}$$

$$I_{NS\to ND} = \begin{cases} IS \cdot \exp\left(\frac{V_{ov}}{V_T}\right) & \text{if } V_{ov} < 0 \\ KP \cdot \frac{W}{L} \cdot V_{ds} \cdot (V_{ov} - \frac{1}{2}V_{ds}) & \text{if } V_{ov} > V_{ds} \\ \frac{1}{2}KP \cdot \frac{W}{L} \cdot V_{ov}^2 \cdot (1 + \lambda(V_{ds} - V_{ov})) & \text{if } V_{ov} < V_{ds} \end{cases} \tag{4.60}$$

Additionally, ideal diodes in Fig. 4.4 represent leakage current at Substrate-Drain and Substrate-Source junctions. Currents through these diodes are modeled by the following equations.

$$Vbs = V(NB') - V(NS') \tag{4.61}$$

$$I_{\text{gs}} = area \cdot IS \cdot \left(\exp\left(\frac{Vgs}{\eta V_T}\right) - 1\right) + G_{min} \cdot (Vbs) \tag{4.62}$$

$$Vbd = V(NB') - V(ND') \tag{4.63}$$

$$I_{\text{gd}} = area \cdot IS \cdot \left(\exp\left(\frac{Vgd}{\eta V_T}\right) - 1\right) + G_{min} \cdot (Vbd) \tag{4.64}$$

Note, that the above equations are valid for 'n-channel' MOSFET. Polarities of the diodes are reverse in a 'p-channel' MOSFET.

Capacitances at substrate-drain and substrate-source junction C_{bd} and C_{gs} are modeled as follows.

$$C_{\text{bd}} = area \cdot (Cbd1 + Cbd2) \tag{4.65}$$

$$CBD' = CBD + CJ \cdot AD \tag{4.66}$$

$$Cbd1 = CBD \cdot \left(1 - \frac{Vbd}{PB}\right)^{-MJ} \tag{4.67}$$

$$Cbd2 = TT \cdot \frac{IS}{\eta V_T} \cdot \exp\left(\frac{Vbd}{\eta V_T}\right) \tag{4.68}$$

$$C_{\text{bs}} = area \cdot (Cgs1 + Cgs2) \tag{4.69}$$

$$CBS' = CBS + CJ \cdot AS \tag{4.70}$$

$$Cbs1 = CBS' \cdot \left(1 - \frac{Vbs}{PB}\right)^{-MJ} \tag{4.71}$$

$$Cbs2 = TT \cdot \frac{IS}{\eta V_T} \cdot \exp\left(\frac{Vbs}{\eta V_T}\right) \tag{4.72}$$

Capacitances between gate-drain and gate-source nodes are MOS capacitances. They are calculated as follows.

$$C_{\text{gs}} = CGSO \cdot W \tag{4.73}$$

$$C_{\text{gd}} = CGSO \cdot W \tag{4.74}$$

$$C_{\text{gb}} = CGBO \cdot L \tag{4.75}$$

Source and drain series resistances are calculated as follows.

$$R_d = RD + RSH \cdot \frac{AD}{W^2} \tag{4.76}$$

$$R_s = RS + RSH \cdot \frac{AS}{W^2} \tag{4.77}$$

Temperature dependence of various MOSFET parameters has been modeled using the following equations. These equations are used when the MOSFET temperature is set by setting `TEMP` parameter.

$$VTO(T) = VTO + TCV \cdot (T - TNOM) \tag{4.78}$$

$$KP(T) = KP \cdot \left(\frac{T}{TNOM}\right)^{BEX} \tag{4.79}$$

$$IS(T) = IS \cdot \exp\left(\left(\frac{T}{TNOM} - 1\right)\frac{EG}{N \cdot \phi_T}\right) \cdot \left(\frac{T}{TNOM}\right)^{XTI/N} \tag{4.80}$$

$$RS(T) = RS + TR1 \cdot (T - TNOM) + TR2 \cdot (T - TNOM)^2 \tag{4.81}$$

$$RD(T) = RD + TR1 \cdot (T - TNOM) + TR2 \cdot (T - TNOM)^2 \tag{4.82}$$

$$RB(T) = RB + TRB \cdot (T - TNOM) \tag{4.83}$$

$$RG(T) = RG + TRG \cdot (T - TNOM) \tag{4.84}$$

In inverted mode, that is, $V_{\mathrm{ds}} < 0$, source and drain in the normal mode are switched.

4.5 MESFET component

A new line starting with the letter `Z` defines a Metal-Semiconductor Field Effect Transistor (MESFET). The first string is the component name and the next four strings are names of the nodes connected to the drain, gate, and source of the MESFET. The next string is the name of the MESFET model. The next number is the MESFET area factor.

For example, a MESFET of area scaling factor *area* is instantiated with drain ND, gate NG, and source NS in the following lines. Note, that the model 'mesfet_depl' of an n-channel MESFET with various hyper-parameters is also written.

```
.model mesfet_depl MESFET (KP=200u VTO=0.6 PHI=0.6 GAMMA=0)
Z1 ND NG NS mesfet_depl
```

Resistors `RD`, `RG`, and `RS` are added in series with each of the drain, gate, and source, respectively. Junction capacitors `CGS` and `CGD` are added in parallel to the 'ideal' diodes present at gate-source and gate-drain junctions, respectively. Together with them, equivalent circuit of the MESFET is shown in Fig. 4.5.

Parameters accepted by the 'MESFET' component are listed in Table 4.5 together with their meaning, default value, and the unit.

Following equations model an ideal n-channel MESFET.

$$V_{ds} = V(ND') - V(NS') \tag{4.85}$$

$$V_{ov} = V(NG') - V(NS') - VTO \tag{4.86}$$

$$I_{ND' \to NS'} = Id \tag{4.87}$$

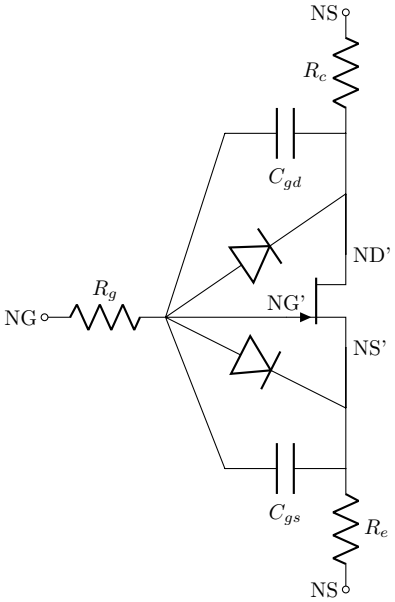


Figure 4.5: Equivalent circuit of the MESFET component

Table 4.5: Input parameters of the MESFET component

Parameter	Description	Default	Uni
VTO	threshold voltage	$-1.$	
BETA	trans-conductance parameter	0	$A/V^2/$
ALPHA	saturation voltage parameter	2	$A/V^2/$
LAMBDA	channel-length modulation	0	$A/V^2/$
VBD	G-D, G-S junction breakdown voltage	100	V
ISAT	leakage current in sub-threshold region	10^{-14}	A
IS	G-S, G-D diode saturation current	1	A/m
RS	source series resistance	0	Ω
RG	gate series resistance	0	Ω
RD	drain series resistance	0	Ω
N	G-S, G-D diode emission coefficient	1.5	-
B	Gdoping tail extension parameter	2	-
PB	G-S, G-D junction potential	0.8	V
CDS	D-S capacitance	0	F/m
CGD	G-D zero-bias depletion capacitance	0	F/m
CGS	G-S zero-bias depletion capacitance	0	F/m
TT	transit time	0	sec
MJ	G-D, G-S junction grading coeff.	0.5	-
FC	coefficient in forward-bias depletion capacitance	0.5	-
TEMP	MESFET temperature	300	K
XTI	IS diode temperature exponent	3.	-
BETATCE	trans-conductance temperature coeff.	0	1/K
VTOTC	threshold voltage temperature coeff.	0	1/K
TRD1	RD linear temperature coeff	0	1/K
TRG1	RG linear temperature coeff	0	1/K
TRS1	RS linear temperature coeff	0	1/K

$$Id = \begin{cases} IS \cdot \exp\left(\frac{V_{ov}}{V_T}\right) & \text{if } V_{ov} < 0 or \alpha < 0 \\ \frac{\beta \cdot V_{ov}^2}{1+B \cdot Vov} \cdot (1 + \lambda V_{ds}) \cdot (1 - (1 - \frac{\alpha V_{ds}}{3})^3) & \text{if } 0 < V_{ds} < 3/\alpha \\ \frac{\beta \cdot V_{ov}^2}{1+B \cdot Vov} \cdot (1 + \lambda V_{ds}) & \text{if } V_{ds} > 3/\alpha \end{cases} \tag{4.88}$$

(4.89)

Additionally, ideal diodes in Fig. 4.3 represent leakage current at Gate-Drain and Gate-Source junctions. Currents through these diodes are modeled by the following equations.

$$Vgs = V(NG') - V(NS') \tag{4.90}$$

$$I_{\text{gs}} = area \cdot IS \cdot \left(\exp\left(\frac{Vgs}{\eta V_T}\right) - 1\right) + G_{min} \cdot (Vgs) \tag{4.91}$$

$$Vgd = V(NG') - V(ND') \tag{4.92}$$

$$I_{\text{gd}} = area \cdot IS \cdot \left(\exp\left(\frac{Vgd}{\eta V_T}\right) - 1\right) + G_{min} \cdot (Vgd) \tag{4.93}$$

Note, that the above equations are valid for n-channel MESFET. Currently, the simulator supports *only* n-channel MESFET.

The semiconductor junction capacitances C_{gd} and C_{gs} are modeled as follows.

$$C_{\text{gd}} = area \cdot (Cgd1 + Cgd2) \tag{4.94}$$

$$Cgd1 = CGD \cdot \left(1 - \frac{Vd}{PB}\right)^{-M} \tag{4.95}$$

$$Cgd2 = TT \cdot \frac{IS}{\eta V_T} \cdot \exp\left(\frac{Vgd}{\eta V_T}\right) \tag{4.96}$$

$$C_{\text{gs}} = area \cdot (Cgs1 + Cgs2) \tag{4.97}$$

$$Cgs1 = CGS \cdot \left(1 - \frac{Vd}{PB}\right)^{-M} \tag{4.98}$$

$$Cgs2 = TT \cdot \frac{IS}{\eta V_T} \cdot \exp\left(\frac{Vgs}{\eta V_T}\right) \tag{4.99}$$

Temperature dependence of various MESFET parameters has been modeled using the following equations. These equations are used when the MESFET temperature is set by setting `TEMP` parameter.

$$VTO(T) = VTO + VTOTC \cdot (T - TNOM) \quad (4.100)$$

$$BETA(T) = \beta \cdot 1.01^{BETATCE \cdot (T - TNOM)} \quad (4.101)$$

$$IS(T) = IS \cdot \exp\left((\frac{T}{TNOM} - 1)\frac{EG}{N \cdot \phi_T} \right) \cdot (\frac{T}{TNOM})^{XTI} \quad (4.102)$$

$$RS(T) = RS + TRS1 \cdot (T - TNOM) \quad (4.103)$$

$$RD(T) = RD + TRD1 \cdot (T - TNOM) \quad (4.104)$$

$$RG(T) = RG + TRG1 \cdot (T - TNOM) \quad (4.105)$$

In inverted mode, that is, $V_{\rm ds} < 0$, source and drain in the normal mode are switched.

Chapter 5

Subcircuits and Parametric Equations

Blocks of circuit which are repeatedly used in the main circuit can be defined as subcircuits and reused. Subcircuits can be parameterically defined thereby increasing their re-usability. An example subcircuit is defined below.

```
.model Da1N4004 D ( IS=18.8n RS=0)

.subckt Example   N1   N2   N3 PARAMS: res1=2.0 res2=12.0
.PARAM res3={res2*2}
.PARAM res4={res1*5}
Rk    N1    N2   res3
Rl    N3   N2   res4
Rm    N3    N1   {2.0+I(Vprob)}
Vprob N3    N4  DC 0
Di    N4    N1  Da1N4004 1E-1
.ends

...
X3  n1 n2 n3 Example PARAMS: res1=9.0 res2=5.0
...
```

```
.end
```

In the above example, the subcircuit definition begins with the first keyword `.subckt` which is followed by the subcircuit name `Example`. The name is followed by the names of the input nodes of the subcircuit. Subcircuits can optionally take parameters which are listed after the keyword `PARAMS:` which is written after listing all the input nodes of the subcircuit. Each parameters has a default value listed with it. For example, `res1` is set to a default value of 2.0. The subcircuit ends with first keyword `.ends`.

Any subcircuit can access ground node by using the name `0` reserved for ground node. Any subcircuit can access any model defined outside the subcircuit definition. A subcircuit may not be defined inside the definition of another subcircuit. That is, all subcircuits must be defined in the main circuit file.

A subcircuit is instantiated in the main circuit or in another subcircuit by using the character `X` at the beginning of the subcircuit name (here it is `X3`). It is followed by the node names from the main circuit which are connected to the subciruit output nodes. The nodes are followed by the subcircuit name (here it is `Example`). User-defined values of subcircuit input parameters are supplied to this instance of the subcircuit by writing `PARAMS:` followed by the input parameter names and their values.

Internal parameters can be defined in a subcircuit by using `.PARAM` at the beginning of the new line. This is followed by `parname={<expr>}` *without* spaces. Here `parname` is a new parameter name and `<expr>` is a mathematical expression enclosed in curly brackets. All the expressions enclosed in `{...}` are evaluated by 'exprtk' expression parser.

Component 'values' can also be parameterized. In the above example, resistance of `Rk` is equal to `res3` which is defined as a parameter. Resistance of `Rm` depends on a solution variable `I(Vprob)`.

5.1 Node Voltages and Currents in Parameter Expressions

Resistance of `Rm` is set to `2.0+I(Vprob)`, where `I(Vprob)` is current through the voltage source `Vprob`. It can be accessed in the mathematical expression using the expression `I(Vprob)`. Similarly, potential difference between two nodes in the subcircuit can be used in the mathematical expression by `V(N3,N2)` where `N3` and `N2` are *internal* names of the nodes in the subcircuit. Voltage at a single node `N3` can be specified in the expression by using `V(N3,0)`.

After every AC/DC/transient solution, these solution variables are updated. However, derivatives of the equations of the components whose values depend on solution are *not* defined. Therefore, using such expressions may lead to non-convergence.

5.1.1 External Subcircuit Library

A library file '*.lib' containing definitions of subcircuits can be imported using `.LIB` command as follows.

```
.LIB filename.lib
```

Chapter 6

Thermal Circuit Solver

Heat is generated in the electronic components during operation. It is conducted to the heat-sink via a number of layers consisting of metals and packaging materials which have a finite heat conductivity and heat capacity. Temperature difference between the component and the heat-sink enables heat conduction. Temperature of the component is determined by the rate of heat generation and conductance of the intermediate layers. On the other hand, electrical characteristics of the electronic component depends on its temperature.

Heat flow equation can be converted to an equivalent electric circuit by drawing equivalency between temperature and voltage, electrical current and heat, electrical capacitance and heat capacity (see Table 6.1). Once an equivalent thermal circuit of the chip packaging is obtained, it can be solved with the circuit solver together with the electrical circuit to determine steady-state/transient voltages, currents, and temperature in the circuit.

6.1 Thermal components

Since the *same* equations are solved in the analysis of both the thermal and the electrical circuit, all the electrical components can be used as thermal components. In reality, only the following electrical components have thermal analogue.

Table 6.1: Equivalence between various thermal and electrical quantities in a thermal circuit

Thermal quantity	Unit	Equivalent electrical quantity	Unit
Heat current	Watt	Current	Ampere
Temperature difference	Kelvin	Potential difference	Volts
Thermal resistance	K/W	Electrical resistance	Ohm
Thermal capacity	Joules/K	Electrical capacitance	Farad

- Resistor: Thermal resistance of the chip package of a specific geometry.
- Capacitor: Heat capacity of the chip package of the specific size.
- Voltage source: A heat-sink which is always kept at a constant temperature relative to `TNOM` can be modeled by a voltage source.
- Current source: A component generating heat at a constant rate can be modeled by a current source.

6.1.1 Resistance

A thermal resistor is modeled in exactly the same way as described in Subsec. 2.1.1. A resistor with resistance value r_{th} models temperature between the two nodes specified by `n1` and `n2` using the following relationship.

$$T(n2) - T(n1) = P_{\text{th}} \cdot r_{\text{th}} \tag{6.1}$$

Here, P_{th} is heat current flowing from `n1` to `n2` through the resistance per unit time.

6.1.2 Capacitor

A thermal capacitor is modeled in exactly the same way as described in Subsec. 2.1.4. A capacitor with value C_{th} models temperature between the two nodes specified by `n1` and `n2` using the following relationship.

$$P_{\text{th}} = C_{\text{th}} \cdot \frac{d(T(n2) - T(n1))}{dt} \tag{6.2}$$

Here, P_{th} is heat current flowing `n1` to `n2` through the capacitor per unit time.

6.1.3 Voltage source

A heat-source/sink is modeled in exactly the same way as described in Subsec. 2.2.1. A heat-source/sink with value $\Delta T = T_{\text{heat-sink}} - T_{\text{nom}}$ creates a fixed temperature difference of ΔT between the two nodes specified by `n1` and `n2`.

6.1.4 Current source

A heat-generator is modeled in exactly the same way as described in Subsec. 2.2.2. A heat-generator with value P_{th0} creates a fixed heat current flowing from `n1` to `n2`. Heat generation in electrical components is modeled by adding a heat-generator corresponding to each of the components. This is described in the next section.

6.2 Heat generation in electrical components

In the circuit simulation, only the heat generation in the following components is modeled.

- Resistor,
- Diode,
- BJT,
- JFET,
- MOSFET

Heat generated in the above listed components enters the thermal circuit *only* when the appropriate node in the thermal circuit is specified as `THPORT` argument while defining the above components.

Heat generated in the parasitic resistances/diodes in the spice circuit of the electronic component (for ex. in BJT, see Fig. 4.2) is added to the heat generated in the main component (e.g. BJT).

Heat generated, if any, in all the other components is ignored. For example, heat generated in a voltage/current source or a controlled source is ignored.

6.3 Setting Component Temperature

During device operation, temperature of the component increases till heat generation rate in the component equals the rate of heat conduction away from the component. Increased temperature can alter electrical characteristics of the component. The altered electrical characteristics change heat generation rate.

While solving electro-thermal circuit, temperature at the thermal node specified by `THPORT` is used to calculate temperature dependence of electrical characteristics of the component. Heat generation rate is calculated from the electrical characteristics.

The following criteria are used in setting a component temperature.

- If *both* `TEMP` and `THPORT` parameters are specified with the component definition, then temperature at the node given by `THPORT` node is set as component temperature.

- If the node specified by `THPORT` is not connected to the thermal circuit, then it is assumed to be connected to thermal `GND`. That is, component temperature is set to `TNOM`.

- If *only* `TEMP` is specified, then component temperature is fixed to the value given by `TEMP` throughout the simulation.

- If neither of the `TEMP` and `THPORT` are specified, then the component temperature is fixed at `TNOM`.

6.4 Example circuit

An example circuit netlist in spice format containing both electrical and thermal circuits is given below.

```
.DC Vs 0.0 50.0 0.1
.AC DEC 1 1. 1M
.TRAN 0.0001 10. 0. 0.1 1E-12 1.1 1.1

.OPTIONS ITLDC=100 ABSTOL=1E-8 TRANTOL=1E-8 ALPHA=0.51
+             TRPZ=1 ITLINNER=10 TNOM=300

.model Da1N4004 D(IS=18.8u RS=1E-2. BV=400 IBV=5.00u
+             CJO=1.E-9 M=0.333 N=2 TT=1E-6 XTI=3.0)

**** Electrical circuit ****
Vs    1    0   DC   1. AC 1. 0 SIN(0. 230. 1.)
Rm    1    5   10. THPORT=rmth TC1=10. TC2=1.
Di    5    6  Da1N4004 1. TEMP=350. THPORT=dith
RL    5    6  100.
Rn    6    0   10.

**** Thermal circuit ****
Rth2 rmth   dith   0.1
Rth dith   8   0.1
Vth 8      0   DC  0.
.end
```

In the above file, both electrical and thermal circuits are specified in the same circuit file, but are separate. They are linked to each other only by the nodes specified by `THPORT` on the lines which define the resistor `Rm` and the diode `Di`. Heat generated in `Rm` and `Di` flows through the thermal circuit to the heat-source/sink `Vth` which is at $\Delta T = 8\,\mathrm{deg}\,C$. That is, absolute temperature of heat-sink is,

$$T_{\mathrm{Vth}} = 8 + T_{\mathrm{nom}} = 8 + 300 = 308\mathrm{K} \tag{6.3}$$

.

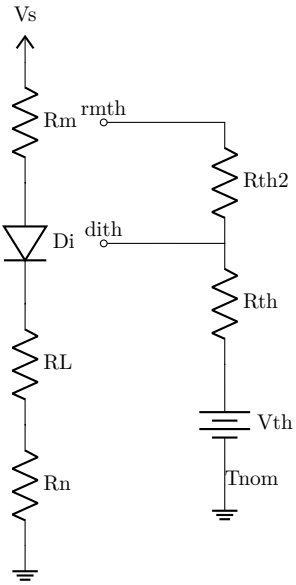


Figure 6.1: The example circuit containing thermal and electrical components. Note, the circuit on the left is an electrical circuit containing the diode, whereas the one on the right is a thermal circuit which is linked to the electrical circuit.

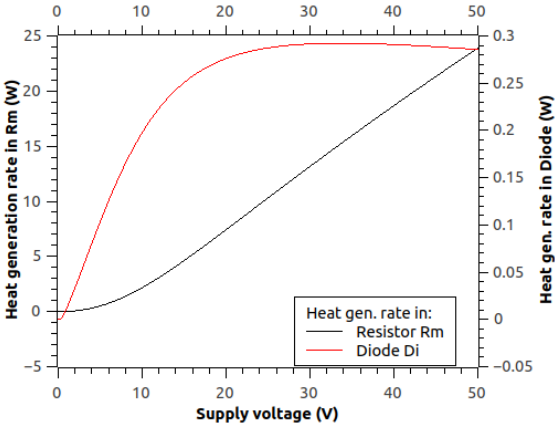
Note, that although `TEMP` keyword is specified with the component `Di`, it is disregarded. Temperature of `Di` is set by the following equation.

$$T_{\mathrm{Di}} = \Delta T(dith) + T_{\mathrm{nom}} \tag{6.4}$$

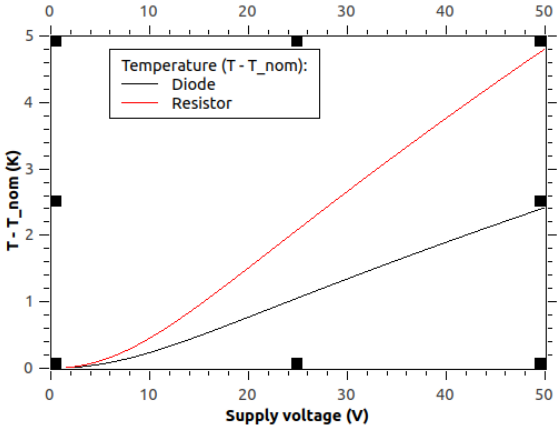
where, $\Delta T(dith)$ is obtained by solving the circuit.

The example circuit netlist given above can be pictorially represented as shown in Fig. 6.1. The above circuit can be solved using the following command.

```
>> CircuitSolver Dithckt.cir
```



(a) Heat generation rate



(b) Temperature

Figure 6.2: (a) Heat generation in the diode and the resistor, (b) Temperature of the two components.

Heat generated in the diode and the resistor during the DC analysis is shown in Fig. 6.2(a). Note, the different scales on the two Y axes. Much larger heat is generated in the external resistor `Rm` compared to the diode. The diode and the resistor temperature are plotted in Fig. 6.2(b). The diode temperature is high despite the low heat generation in the diode. Due to the placement of the diode and the resistor on the board (see Fig. 6.1), resistor heat flows through the diode package resulting in higher heat current through the diode package. This increases the diode temperature.

Chapter 7

Python Interface

The scripting language `Python` comes with various optimization, machine learning, and other libraries. These libraries can be used for calibration of the equivalent circuit models of various components, and also for optimization of the given system. The Circuit Solver provides a python library to enable calling the solver from a python file. This can enable the use of 'python-specific' libraries together with the circuit solver for various calibration and optimization tasks.

The following tasks can be performed using the python interface.

- Loading a circuit netlist from a Spice circuit file *.cir
- Adding `AC`, `DC`, and `Transient` solve commands to the solver before solving the circuit.
- Reading and editing a component parameter value.
- Reading voltages/currents at any specific node which are generating in any specific solve command.

At the moment, the solver does not enable editing circuit netlist from the python interface. The netlist can only be edited in the spice netlist file and reloaded to python script.

7.1 Example python script

An example python script to show usage of various procedures is given below.

```
import circuitsolver as cs
import numpy as np

p = cs.circuit ()

p.readSpiceCircuitFile ("CircuitTrial.cir")

print (p.setComponentParamVal (Component="X1 R3", Value=1E2))

print (p.getComponentParamVal (Component="X1 R2"))

p.printCircuitNetlist ()

p.clearAnalyses ()

dc1 = p.addDCAnalysis (source="Vinput", start=0., end=1.0, incr=0.1)

tr1 = p.addTransientAnalysis (tstep=0.0001, tstop=0.01,
          tstart=0., dtmax=0.001, dtmin=1E-12, dtincr=1.3, dtdecr=2.)

p.setNodeCurrentToSave (Node="Vout", Component="R1")

p.setNodeCurrentToSave (Node="X1 Vs", Component="R3")

p.solve ()

print (p.getVsourceId (VsourceAddr="Vinput"))

print (p.getVsourceTransCurrentNumpyArray (VsourceAddr="Vinput",
                                           analysisId=dc1))

print (p.getNodeTransCurrentNumpyArray(Node="X1 Vs",
                     Component="R3", analysisId=dc1))
```

```
print (p.getDCSolutionNumpyArray (analysisId=dc1))
```

The line `p = cs.circuit ()` creates an instance of the 'circuit' object. The next line reads in an ascii file 'CircuitTrial.cir' which contains netlist of the circuit spice format as given below.

```
.DC Vinput 0.0 1.01 0.1
.AC DEC 1 1. 1M
.TRAN 0.0001 0.01 0. 0.001 1E-12 1.3 2.

.subckt Rec2 Vin Vout
R1 Vin Vout 10.
R2 Vs  0    10.
R3 Vs Vout 1M
.ends

.subckt Rect Vin Vout
R1 Vin Vout 10.
R2 Vs  0    10.
R3 Vs Vout 1M
X1 Vin 0 Rec2
.ends

Vinput Vin 0 DC 0.0 AC 1.0 90
+        SIN(0.8V 0.025V 100.)
X1 Vin Vout1 Rect
X2 Vout1 Vout Rect
W5 Vin Vout 0 TempW
R1 Vout 0 1000.
.end
```

All the functions used in the above script file are listed below.

7.2 Read Circuit File

- `circuitsolver.circuit` : An instance of the `circuit` class is created and returned. Note, that all the functions listed below are called onto the instance of the circuit class.

- `readSpiceCircuitFile` : Reads an ascii file containing circuit netlist in spice format. Input: Spice circuit file *.cir Output: Returns 1 if the file is successfully read. Throws exception, otherwise.

- `printCircuitNetlist` : Lists all the components in the main circuit as well as sub-circuits, in the following format. `R1 : <R> : <n1>, <n2>` In the above line, `R1` is the component name, `<R>` is the resistance, `<n1>, <n2>` are the Ids of the nets connecting the resistor. Note, that the net Ids are different from the net names specified in the *.cir file. Input: None. Output: Prints all components.

- `clearCircuit` : Deletes all components of the circuit. It is important to clear all components loaded to the current instance of the circuit before loading the next components by using 'readSpiceCircuitFile'. Input: None.

7.3 Edit Parameters

Following functions can be used to retrieve a specific parameter value for any of the components given the parameter name and the component address.

- `getComponentParamVal` : Returns value of the given parameter of the specified component. Inputs:
  - `Component` : Component address followed by its name.
  - `Param` : Parameter name.

  Output: Stored parameter value.

- `setComponentParamVal` : Sets value of the given parameter of the specified component. Inputs:
  - `Component` : Component address followed by component name. For example, a component named `X1 R3` corresponds to a resistor named `R3` in the subcircuit `X1`.
  - `Value` : New parameter value.
  - `Param` : Parameter name.

Output: Returns 'true' if the parameter value is set.

- `setParameterValues` : Input a python map which stores circuit numeric parameter name and corresponding values. The numeric parameters of the circuit solver are set to the specified values.
- `setFunctionalModel` : Links a python object of the user defined `functionalmodel` class to the specified generic component (whose name starts with `W`). More information about the functional models and their usage is provided in Chapter 8.

7.4 Add Analyses

Note, that the analyses are added in the same order as the following commands are called in the python script file.

- `addDCAnalysis` : Adds DC analysis to the existing list of analyses, when the following inputs are provided. Inputs:
  - `source` : Name of the voltage source.
  - `start` : Voltage bias of the source at the beginning.
  - `end` : Voltage bias at the end.
  - `incr`: Increment in steps.
  - `source2` : Name of the second voltage source in the DC analysis of two sources on the 2D grid.
  - `start2` : Voltage bias at the beginning.
  - `end2` : Voltage bias at the end.
  - `incr2`: Increment in steps.

  Output: Returns an integer which is the 'analysis id'. This analysis id is required when retrieving the solution.
- `addACAnalysis` : Adds AC analysis to the existing list of analyses, when the following inputs are provided. Inputs:
  - `LineSpacing` : 'LIN' for linear spacing of points, 'DEC' for points spacing per decade, 'OCT' per octave.

  - `points` : Number of points per decade/octave.
  - `start` : Start frequency for the frequency sweep.
  - `end` : End frequency for the frequency sweep.

  Output: Returns an integer which is the 'analysis id'. This analysis id is required when retrieving the solution.

- `addTransientAnalysis` Adds transient analysis to the existing list of analyses, when the following inputs are provided. Inputs:

  - `tstop` : End time of the transient simulation.
  - `tstep` : Time step at the beginning of the analysis.
  - `tstart` : Start time of the transient simulation.
  - `dtmax` : Maximum allowed time step. Time step is capped at this value.
  - `dtmin` : Minimum allowed time step. If time step goes below this value, simulations are stopped.
  - `dtincr` : If the simulation at current step is successful, then time step is incremented by this factor.
  - `dtdecr` : If the simulation at current step is unsuccessful, then time step is decremented by this factor.

  Output: Returns an integer which is the 'analysis id'. This analysis id is required when retrieving the solution.

- `clearAnalyses` : Deletes all the analyses stored in the circuit object.

7.5 Read Solution Data

Following functions return various solutions of the circuit analyses or node quantities.

- `getGlobalNodeId` : Returns global id of the node given the address of the node and its name appended at the back. Voltage at the node obtained by solving the circuit is stored in the solution vector at the element id equal to the global id of the node (when the vector element numbering begins with one).

- `getVsourceId` : Returns voltage source id given a string with the address of the voltage source and its name appended at the back. Current flowing through this voltage source is stored in the solution vector at the element id equal to the voltage source id (when the vector element numbering begins with one).
- `get[DC | AC | Transient]SolutionNumpyArray` : When the analysis id is specified, this function returns a 2D numpy array of the solution at each node at each bias point or frequency or time. First column of the numpy array is the DC bias or frequency or time. Next columns store the solution at each of the nodes.
- `getNode[DC | AC | Transient]VoltageNumpyArray` : When the *node address* and the analysis id is specified, it returns a numpy array containing bias point or frequency or time dependent voltage at the given node.
- `getVsource[DC | AC | Transient]CurrentNumpyArray` : When the *voltage source address* and the analysis id is specified, it returns a numpy array containing bias point or frequency or time dependent current flowing through the given node. Since currents through voltage sources are calculated and stored while solving the circuit, it is not necessary to explicitly use `setNodeCurrentToSave` to store the currents.
- `getNode[DC | AC | Transient]CurrentNumpyArray` : When the node address, the component name, and the analysis id is specified, it returns a numpy array containing bias point or frequency or time dependent current *entering* the given component at the given node. In the example python file, the node address `X1 Vs` corresponds to a node named `Vs` in subcircuit `X1`. Among all the components connected to `Vs` in `X1`, current entering the resistor `R3` at each step in the analysis id stored in the variable `dc1` is returned as a numpy array.

  Note, that in order to return current using this function, the node current entering the device must be stored at bias points. See `setNodeCurrentToSave` function on how to do it.
- `setNodeCurrentToSave` : Specify the node address and the component name before solving the circuit. Current entering

the given component and at the given node is stored in the solver to be retrieved later. In the example python file, the node address `X1 Vs` corresponds to a node named `Vs` in subcircuit `X1`. Among all the components connected to `Vs` in `X1`, current entering the resistor `R3` is stored at every bias-point, AC frequency, or time-step. It can be retrieved at the end of the simulation by using the function `getNodeTransCurrentNumpyArray`.

Note, that current entering a subcircuit pin (begins with `X`) or a generic chip (begins with `W`) cannot be stored.

Chapter 8

Functional Modeling of The Driver Chip

Modern power electronic circuits are often controlled by an IC chip often called as the driver. This IC chip performs more than just sending the driver signals for the given type of power device. Various other logic functions involved in synchronizing with the input sinusoidal signal, over-current protection, receiving temperature sensor, etc. are also included in the driver chip. Such functionality of the driver chip can be modeled by using functional model blocks.

8.1 Defining the functional model

To define a functional model, a python class which inherits the base class "functionalmodel" is defined by the user. A member function named "updateOutputPinVoltages" is defined in this newly defined class. Functional model of the chip can be defined inside this member function as described below. Once the class is defined, a new instance of the class is created corresponding to each instance of the specific driver chip present in the circuit. An example functional model is defined in the python code snippet below.

```
import circuitsolver as cs
```

```
import numpy as np

p = cs.circuit ()

class SimpleDriver (cs.functionalmodel):
  def updateOutputPinVoltages (self, isOutputPin, inputV, time):
    numPins = len(isOutputPin)
    outV = []
    for i in range(numPins):
      if isOutputPin[i]:
        if time > 1.0: # transient voltage ramp
          outV.append (1.0)
        else if time == -1: # DC voltage ramp
          outV.append (1.0)
        else:
          outV.append (0.0)
    return outV

driv1 = SimpleDriver ()

p.readSpiceCircuitFile ("RCckt.cir")

p.setFunctionalModel ("W1", driv1)

p.solve ()
```

The above python script reads an example circuit from the file 'RCckt.cir', which is also shown below.

```
.TRAN 0.01 5. 0. 0.01 1E-12 1.3 1.3

.OPTIONS ITLDC=100 ABSTOL=1E-8 TRANTOL=1E-8 ALPHA=0.51 TRPZ=1

W1    1    0     DriChp OUTPINS=(0)
Rm    1    out   1000
Cn    out  0     1E-3
.end
```

The steps to be followed to use a functional model in the circuit simulations are described below.

8.1.1 Defining the model class

A class named `SimpleDriver` inherited from `cs.functionalmodel` class is defined in the above snippet. This functional model class defines a driver chip of one output pin. The driver sets the voltage of 1V at the output pin at $t = 1$ sec.

Note, the structure of the class `SimpleDriver`. The class has a function named `updateOutputPinVoltages`. This function takes the following input arguments.

- `self` : Reference to the class object.
- `isOutputPin` : A list of boolean variables which has the length equal to the number of output pins. Each variable is set `true` if the corresponding pin is an output pin and `false` otherwise.
- `inputV` : A list of real numbers which has the length equal to the number of output pins. Each variable specifies voltage at the corresponding pin in the last DC or transient step.
- `time` : A real number which is set to the time at the current time step in transient simulations. In DC bias ramp, time is set to -1.

The above function `updateOutputPinVoltages` must return a list of real numbers which has the length equal to the *number of output pins*. Voltage at each of the output pins must be listed in the same order as they are present in `inputV`.

8.1.2 Creating the driver instance

In order to use the functional model of the driver, an instance of the class `SimpleDriver` is created in the python script as follows.

```
driv1 = SimpleDriver ()
```

Note, that a new instance must be created for every driver chip of the same type present in the circuit.

8.1.3 Creating the driver in the netlist file

A driver chip is instantiated inside the netlist file 'RCckt.cir' using a line which begins with the letter `W` as follows.

```
W1    1    0     DriChp OUTPINS=(1)
```

In the above line, the first word specifies the driver instance name (here `W1`).The output pins of the chip are listed as comma-separated list in the brackets after the argument `OUTPINS=`. The second-last word corresponds to the generic name of the driver chip (here `DriChp`).

In this case, the chip has only two pins. The first pin is connected to the net '1' in the netlist while the second pin is connected to the ground ('0'). Out of the two pins, only first pin is output pin. Therefore, `OUTPINS=(1)` is set at the end of the line.

8.1.4 Linking the functional model to the driver

Every instance of the driver chip in the circuit netlist must be associated with an instance of user-defined functional model. To link an instance of the functional model to the driver chip, a python procedure `setFunctionalModel` must be called on the circuit instance as follows.

```
p.setFunctionalModel ("W1", driv1)
```

The above procedure associates the driver chip in the circuit file named `W1` with the functional model class instance `driv1`. In this way, at every time step during transient/DC simulation, the procedure `updateOutputPinVoltages` is called. Voltages at the output pins are read from the python list returned by the procedure and the voltage sources connected to the output pins are set.

When performing an AC frequency sweep, the output pins are connected to the AC voltage sources with zero AC bias.

Once the above steps are listed in the python script file, the `solve` procedure can be called on the `circuit` instance to solve the circuit.

8.2 Python model library file

A python file containing the functional model can be imported to the circuit file (*.cir) using the line below.

```
.pylib filename.py
```

Here `filename` is the name of the file in which the functional model class is defined. The circuitsolver reads the circuit file, imports all the functional python models, and links them to the appropriate components.

Chapter 9

Behavioural Model Interface

Modern nonlinear electronic devices such as memoristor, etc. sometimes cannot be modeled using the existing compact models (of diodes, BJTs, etc.). Behavioural model interface of the circuitsolver enables the users to develop codes of their own behavioural models of these devices and perform circuit simulations by deploying these compact models.

To facilitate quick model development and deployment, 'python' language is used to create the compact models.

9.1 Defining the behavioural model

To define a behavioural model, a python class which inherits the base class "behaviouralmodel" must be defined by the user. Various member functions need to be defined by the user in this class. Their functions are described below.

9.1.1 Behavioural model class definition

A python class named 'MyDiode' inheriting the base class 'behaviouralmodel' can be defined as follows.

`class MyDiode (cs.behaviouralmodel)`

Static member variables of the class i.e. the variables shared by all the class instances, are defined after the above line. Following member functions *must be* defined by the user.

Constructor

`def __init__(self):` is a constructor of the class. It is called when a new class instance is created. Member variables which are limited to the instance are defined in constructor.

Note, constructor of the base class 'behavioural model' must be called in this constructor as follows – `cs.behaviouralmodel.__init__(self)`.

isNonLinear

This function must return `True` if the device exhibits nonlinear behaviour.

useNumericDifferentiation

If this function returns `True`, then numeric differentiation is used for calculating derivatives in the nonlinear solver. If `True`, central difference scheme is used to calculate the derivatives numerically. Numeric differentiation often leads to non-convergence of the nonlinear solver.

setParameter

This function receives `name` and `value` as input arguments. It can be used to set parameter values, when the parameter is specified in the spice circuit file.

getPinCurrents

This function receives `inputV` and `time` as input arguments. `inputV` is a python list of voltages at the device pins. Hence, its length is equal to the number of pins. `time` is equal to the ramp time, if the simulations are in transient ramp. Otherwise, it is equal to -1.

This function calculates pin currents from pin voltages of the device. It returns the pin currents as a python list of numbers. Length of the returned list must be equal to that of `inputV`. As a convention, current entering the pin is positive.

This function is called when `useNumericDifferentiation` returns `True`.

getDerivativesAndPinCurrents

This function receives `inputV` and `time` as input arguments. `inputV` is a python list of voltages at the device pins. Hence, its length is equal to the number of pins. `time` is equal to the ramp time, if the simulations are in transient ramp. Otherwise, it is equal to -1.

This function calculates pin currents from pin voltages of the device. It also calculates the derivative of the current of i^{th} pin with respect to the voltage of j^{th} pin for all pairs $(i, j) : i, j \in 1, .., n$. Thus, we have a $n \times n$ matrix of elements $\frac{dI}{dV}|_{i,j}$, as follows.

$$J = \begin{bmatrix} \frac{dI_1}{dV_1} & \frac{dI_1}{dV_2} & \cdots & \frac{dI_1}{dV_n} \\ \frac{dI_2}{dV_1} & \frac{dI_2}{dV_2} & \cdots & \frac{dI_2}{dV_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{dI_n}{dV_1} & \frac{dI_n}{dV_2} & \cdots & \frac{dI_n}{dV_n} \end{bmatrix} \tag{9.1}$$

$$I = [I_1, I_2, \ldots, I_n] \tag{9.2}$$

The above matrix is flattened to create a python list. Pin currents are appended to the list. Thus, for a n-pin device, the list length is $(n + 1) \cdot n$, as shown below.

$$L = [\frac{dI_1}{dV_1}, \frac{dI_1}{dV_2}, \ldots, \frac{dI_1}{dV_n}, \frac{dI_2}{dV_1}, \ldots, \frac{dI_n}{dV_1}, \frac{dI_n}{dV_2}, \ldots, \frac{dI_n}{dV_n}, I_1, I_2, \ldots, I_n] \tag{9.3}$$

After computing the derivatives, the user must create a list composed of the derivatives and the currents in exactly the above order and return it.

This function is called when `useNumericDifferentiation` returns `False`.

9.1.2 Use of auto-differentiation

Various auto-differentiation packages are available. They can be used to calculate derivatives. The example below uses one such package `autograd` to compute derivatives using autodifferentiation.

9.1.3 Example code

Behavioural model of a diode defined using the 'behaviouralmodel' interface is shown below.

```
import numpy as np
import autograd as ad
from autograd.variable import Variable
import circuitsolver as cs
import math

p = cs.circuit ()

class MyDiode (cs.behaviouralmodel):
  name = "ABCD" # static member variable

  def __init__(self):
    cs.behaviouralmodel.__init__(self)
    self.Is = 1E-6 # member variables
    self.VT = 0.025
    self.N = 2.0 # ideality factor

  def isNonlinear (self):
    return True;

  def useNumericDifferentiation (self):
    return False;

  def setParameter (self, name, value):
    if (name == "IS"):
      self.Is = value

  def getPinCurrents (self, inputV, time):
```

```
    numPins = len(inputV)
    if numPins != 2:
      return [0] * numPins

    Vd = inputV[0] - inputV[1]
    Id = self.Is * math.exp(Vd / self.VT / self.N)
           + 1E-12 * Vd
    outI = [Id, -Id]
    return outI;

  def getDerivativesAndPinCurrents (self, inputV, time):
    numPins = len(inputV)
    if numPins != 2:
      return [0] * numPins * (numPins + 1)

    bigV = Variable (inputV)
    va, vc = bigV[0], bigV[1]
    # Anode current
    if inputV[0] - inputV[1] < 1:
      Ia = self.Is * ad.exp( (va - vc) / self.VT / self.N)
                     + 1E-12 * (va - vc)
    else:
      Ia = self.Is * math.exp( 1.0 / self.VT / self.N)
                      * (va - vc)

    # cathode current
    Ic = - Ia
    # computing gradients
    Ia.compute_gradients()
    Ic.compute_gradients()

    outI = np.append(Ia.data, Ic.data)
    outdI = np.append(Ia.gradient, Ic.gradient)
    out = np.append (outdI, outI)
    return out;

dio1 = MyDiode ()
p.readSpiceCircuitFile ("RCckt.cir")
```

```
p.setBehaviouralModel ("WB0", dio1)
p.solve ()
```

The above python script reads an example circuit from the file 'RCckt.cir', which is also shown below.

```
.TRAN 0.01 5. 0. 0.01 1E-12 1.3 1.3

.OPTIONS ITLDC=100 ABSTOL=1E-8 TRANTOL=1E-8 ALPHA=0.51 TRPZ=1

R0 2 3 1000
V0 3 0 DC 10
WB0 2 0 MyDiode IS=0.0001
.end
```

9.1.4 Linking the behavioural model to the driver

Every instance of the behavioural model class component in the circuit netlist must be associated with an instance of user-defined model. To link an instance of the behavioural model class to the component, a python procedure `setBehaviouralModel` must be called on the circuit instance as follows.

```
p.setBehaviouralModel ("W1", dio1)
```

The above procedure associates the component in the circuit file named `WB1` with the behavioural model class instance `dio1`. In this way, at every time step during transient/DC simulation, the procedure `getDerivativesAndPinCurrents` or `getPinCurrents` is called. Currents (and derivatives, if applicable) at the pins are read from the python list returned by the procedure.

Once the above steps are listed in the python script file, the `solve` procedure can be called on the `circuit` instance to solve the circuit.

9.2 Python model library file

A python file containing the behavioural model can be imported to the circuit file (*.cir) using the line below.

```
.pylib filename.py
```

Here `filename` is the name of the file in which the behavioural model class is defined. The circuitsolver reads the circuit file, imports all the behavioural python models, and links them to the appropriate components.

Chapter 10

Circuit Analyses

A main circuit is defined in a '*.cir' file. Various analyses are also specified in the circuit file by specific commands, typically at the beginning of the file. The analyses supported by the solver are described below.

10.1 DC analysis

A DC analysis is specified by the first word `.DC` which is followed by the name of the voltage source to be ramped up in the DC analysis. This is followed by three numbers, respectively, initial voltage, final voltage, and increment value. For example, first line of the following command sets a DC ramp of voltage source `Vs` from 0V to 2V in the steps of 0.1V.

```
.DC Vs 0.0 2.0 0.1
.DC Vin 0.0 1.0 0.1
```

When multiple DC ramp commands are specified, they are executed sequentially. In the above example, *after* `Vs` is ramped to 2V, `Vin` is ramped from 0V to 1V in the steps of 0.1V keeping `Vs` fixed at 2V.

If nonlinear components exist in the circuit, circuit solution (node voltages and currents through the voltage source) at each bias point is calculated using a nonlinear damped Newton's solver proposed by Bank

and Rose. At each bias point, convergence is achieved when norm of the residual is less than a predefined tolerance (ϵ_{abs}). The tolerance can be set by the option `ABSTOL`. Maximum number of Newton's iterations for the nonlinear solver are set by `ITLDC` and maximum inner iterations in Bank-Rose solver are set by `ITLINNER`. If `ITLINNER` is set to zero, then undamped Newton's solver is called.

10.2 AC analysis

An AC analysis is added to the set of analyses by the first word `.AC` and has one of the following format.

- `.AC DEC NF FS FE`
- `.AC OCT NF FS FE`
- `.AC LIN NF FS FE`

`DEC` means decade variation, and NF is the number of points per decade. `OCT` specifies octave variation, and NF is the number of points per octave. `LIN` stands for linear variation, and NP is the number of points. `FS` is the starting frequency, and `FE` is the final frequency. For example, the following commands set an AC analysis of the circuit at frequencies from 1Hz to 1kHz with 10 frequency points per decade followed by frequencies from 1kHz to 1MHz with 10 frequency points per decade.

```
.AC DEC 10 1. 1K
.AC DEC 10 1K 1M
```

Note, that DC bias at each voltage source can be set by specifying DC ramp command before AC analysis. Each voltage source or current source can act as an AC voltage or current source, if the AC component is set to a non-zero value. If not, the voltage source acts as a short and a current source acts as an open circuit.

10.3 Transient analysis

A transient analysis is set by the first word `.TRAN` and has the following format.

```
.TRAN TSt TE TS dTMax dTMin dTIn dTDe
```

In the above command, `TSt` is the initial time step, `TE` is end time, `TS` is start time, `dTMax` is the maximum time step, `dTMin` is the minimum time step. If the calculations at current time step converge, current step is multiplied by an increment factor given by `dTIn` and time is incremented. If the calculations don't converge, current time step is multiplied by decrement factor `dTDe` and new time-point is calculated by adding the modified step to the previously converged time-point. This convergence check is required only if nonlinear Newton's solver is used for calculations due to the presence of nonlinear components in the circuit.

Note, that a DC analysis is performed before starting the transient analysis. Also, only the voltage or current sources in which a transient waveform is specified are varied in a transient analysis. All the other sources are set constant at the DC bias value.

If nonlinear components exist in the circuit, circuit solution at each time-step is calculated using the nonlinear damped Newton's solver (Bank-Rose solver). At each bias point, convergence is achieved when norm of the residual is less than a predefined tolerance. The tolerance can be set by `ABSTOL`. Local Truncation Error (LTE) is calculated at each time-step as follows.

$$\mathrm{LTE}(t) = \sqrt{\frac{1}{N}\sum_{n}^{N}\left(\frac{r_n(t) - r_n(t-dt)}{(s_n(t) - s_n(t-dt)) \cdot \epsilon_{\mathrm{trans}} + \epsilon_{\mathrm{abs}}}\right)^2} \tag{10.1}$$

Here, $r_n(t)$ and $r_n(t-dt)$ are current and previous residuals at the node n. $s_n(t)$ and $s_n(t-dt)$ are current and previous solutions (node voltages) at node n. ϵ_{abs} and $\epsilon_{\mathrm{trans}}$ can be set by `ABSTOL` and `TRANTOL`, respectively. Maximum number of Newton's iterations for the nonlinear solver are set by `ITLTR`.

10.3.1 Time-stepping

Two time-stepping methods, namely, Backward-Euler (BE) and trapezoidal interpolation, have been implemented. The following temporal equation is solved in the transient circuit solver.

$$\frac{\partial \vec{s}}{\partial t} = C(s) \cdot \vec{s} \tag{10.2}$$

where, $C(s)$ is the circuit matrix and $\vec{s}$ is the solution at any given time.

The above equation is discretized as follows.

$$\frac{\vec{s}[t_i] - \vec{s}[t_{i-1}]}{dt} = \alpha \cdot C(\vec{s}[t_i]) \cdot \vec{s}[t_i] + (1-\alpha) \cdot C(\vec{s}[t_{i-1}]) \cdot \vec{s}[t_{i-1}] \tag{10.3}$$

Solution $\vec{s}[t_i]$ is calculated from the known variables using the above equation. Note, that in nonlinear circuits $C(\vec{s}[t_i])$ on Right Hand Side (RHS) is a nonlinear function of the solution at t_i. This necessitates use of a non-linear iterative method. When $\alpha = 1$, the above time-stepping method is called BE method. In a trapezoidal method $0 < \alpha < 1$.

When `TRPZ` is set to zero, BE method is called, else trapezoidal method is called. α can be set by the parameter `ALPHA`.

10.4 Solver settings

Solver settings can be modified in the circuit file by starting the line with the word `.OPTIONS` and listing all the parameters after it. An example is given below.

```
.OPTIONS ITLDC=100 ABSTOL=1E-8 TRANTOL=1E-8
```

A list of all the solver parameters which can be set using `.OPTIONS` is given below.

- `ITLDC` : Maximum Newton iterations in non-linear DC solver.
- `ITLTR` : Maximum Newton iterations in non-linear transient solver.

- `ITUNDAMPED` : Maximum inner iterations (>= 0) in undamped Newton non-linear solver. In case of no convergence after undamped iterations, damped Newton iterations are performed. (default value is 5).

- `ITLINNER` : Maximum inner iterations (>= 1) in Newton non-linear solver. First undamped iterations are performed, which are followed by Bank-Rose damping. Default value is 10.

- `NEWTONST` : Scaling factor of the update-step in undamped Newton non-linear solver (<= 1). Default value is 1.

- `ABSTOL` : Absolute tolerance which is used as a stopping criterion in nonlinear Newton solver.

- `TRANTOL` : Tolerance multiplier in the calculation of LTE.

- `RELTOL` : Relative tolerance

- `GMIN` : Minimum conductance.

- `TNOM` : Nominal temperature at which the component parameters are calibrated.

- `TEMP` : Ficed operating temperature of the entire circuit.

- `VNTOL` :

- `TRPZ` : If zero, Backward Euler method is used for time-stepping, else trapezoidal method is used.

- `ALPHA` : In trapezoidal method, weight of the Jacobian of current step is specified by this parameter.

- `TNOM` : Default temperature of all the devices as well as temperature of the `GROUND` node in thermal simulations.

- `PARDISO` : If greater than zero, *Intel mkl* libraries and *Intel Pardiso* solver is used to solve linear matrix equation ($A \cdot x = b$). Else, *SuperLU* solver is used.

- `ILS` : If greater than zero, an iterative linear solver (ILS) based on GMRES method is used to solve linear matrix equation. Incomplete LUT (ILUT) is used as a preconditioner.
- `ITILS` : If ILS solver is being used, then this sets maximum iterations performed in GMRES method.
- `ITREILS` : If ILS solver is being used, then this specifies iterations after which reset is performed.
- `TOLGMRES` : If ILS solver is being used, then this specifies solver tolerance in GMRES method.
- `TOLPRECOND` : If ILS solver is being used, then this specifies 'drop-tolerance' of the ILUT method. If the matrix entry is less than the tolerance value then it is ignored.
- `FILLPRECOND` : If ILS solver is being used, the this specifies fill-factor in the ILUT method.

Chapter 11

Circuit Netlist File Format

The circuit solver parses a circuit netlist file 'exampleSubckt.cir' and performs the listed analyses by using the following command.
`>> CircuitSolver exampleSubckt.cir`

11.1 Circuit File structure

A sample structure file which creates a 2D pn-diode is provided below.

```
.DC Vs 0.0 2.0 0.1
.AC DEC 10 1. 1M
.TRAN 0.001 5. 0. 0.1 1E-12 1.3 1.3

.OPTIONS ITLDC=100 ABSTOL=1E-8 TRANTOL=1E-8
+ ALPHA=0.51 TRPZ=1

.model Da1N4004 D ( IS=18.8n RS=0 )

* Defining subcircuit Example_2
```

```
.subckt Example_2   5   12   18 PARAMS: res1=2.0 res2=12.0
Rk    5    12   res1
Rl    18   12   res2
Rm    18    5   {2.0+I(Vprob)}
Vprob 18    19  DC 0
Di    19    5  Da1N4004 1E-1
.ends

* Defining subcircuit Example_1
.subckt Example_1   5   12   18 PARAMS: res1=2.0 res2=12.0
.PARAM res3={res2*2}
.PARAM res4={res3*5}
Rk    5    12   res1
Rl    18   12   res4
Rm    18    5   2.0
X3    5 12 18 Example_2 PARAMS: res1=9.0 res2=res2
.ends

* Defining main circuit
Vs    1    0   DC   1. AC 1. 0 SIN(0V 2.V 1.)
Ra    1    2   1.0
Rb    3    4   3.0
Rc    7    0  25.0
Rd    6    0  45.0
X1    2    7    3   Example_1 PARAMS: res1=5.0 res2=18.0
.end
```

The above circuit file uses many of the features described in the previous chapters. Apart from these features, following additional information may be useful for writing your own circuit file.

- Each line beginning with `*` is a comment line. It is *not* parsed by the solver.
- A long text line can be split on the multiple lines by adding `+` at the beginning of each of the next line.
- Each subcircuit must begin with a line `.subckt` and end with `ends`

- `.end` specifies end of the circuit file. Any component after `.end` is ignored.
- Each line is split into a list of string by `whitespace` characters (space or tab). Component names, node names, values, and parameters must be separated by whitespace characters.
- A parameter and its expression must not have any whitespace character between them.
- Text in curly brackets `{...}` is parsed by 'exprtk' parser.

11.2 Output files

The above circuit file performs AC, DC, and transient analyses as mentioned at the beginning of the file. Each of the analyses stores the results in a separate 'csv' file. These comma separated file can be viewed using a text editor or a csv viewer program.

Chapter 12

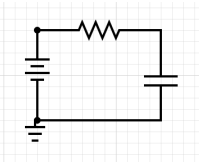
Linear Circuit simulation

Examples of circuits with various linear components are presented in this chapter together with the circuit files. The simulation results can be visualized using a plotting tool `qtiplot`.

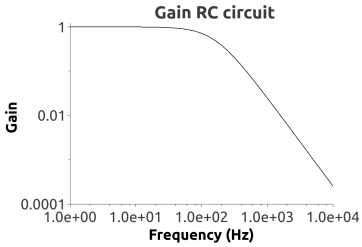
12.1 R-C circuits

A RC circuit shown in Fig. 12.1(a) is simulated using the spice file below.

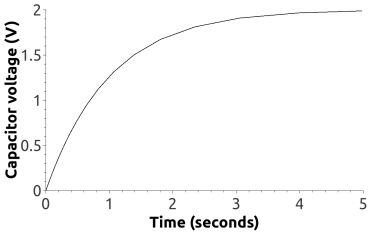
```
*.DC Vs 0.0 2.0 0.1
*.AC DEC 10 1. 1M
.TRAN 0.001 5. 0. 0.1 1E-12 1.3 1.3

.OPTIONS ITLDC=100 ABSTOL=1E-8 TRANTOL=1E-8 ALPHA=0.51 TRPZ=1

Vs    1    0   DC   1. AC 1. 0 PWL(0. 2. 1. 2. 2. 2.)
Rm    1    out   1000
Cn    out  0     1E-3
.end
```



(a) RC circuit



(b) AC voltage



(c) Transient voltage

Figure 12.1: RC circuit of time constant 1 sec in (a) is simulated to calculate magnitude of voltage across the capacitor in (b) AC analysis with $C = 1\mu$F, and in (c) Transient analysis with $C = 1$mF.

Note, that each of the analyses were 'activated' successively, such that *only* one analysis is active at a time. Resulting voltage across the capacitor ($C = 1\mu$F) is plotted vs. frequency in Fig. 12.1(b) for AC analysis and is plotted vs. time in Fig. 12.1(c) for transient analysis ($C = 1$mF).

12.2 L-R circuits

A RC circuit shown in Fig. 12.2(a) is simulated using the spice file below.

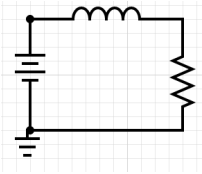
```
*.DC Vs 0.0 2.0 0.1
*.AC DEC 10 1. 1M
.TRAN 0.001 5. 0. 0.1 1E-12 1.3 1.3

.OPTIONS ITLDC=100 ABSTOL=1E-8 TRANTOL=1E-8 ALPHA=0.51 TRPZ=1

Vs    1    0   DC   1. AC 1. 0 PWL(0. 2. 1. 2. 2. 2.)
Rm    out    0   1E3
Ln    1  out     1E3
.end
```

Note, that each of the analyses were 'activated' successively, such that *only* one analysis is active at a time. Resulting voltage across the resistor is plotted vs. frequency in Fig. 12.2(b) for AC analysis ($L = 1$H) and is plotted vs. time in Fig. 12.2(c) for transient analysis ($L = 1$kH).

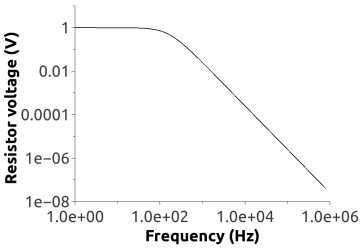
12.3 R-L-C circuits

A RLC circuit shown in Fig. 12.3(a) is simulated using the spice file below.

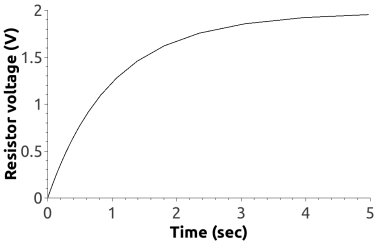
```
*.DC Vs 0.0 2.0 0.1
*.AC DEC 10 1. 1M
.TRAN 0.0001 10. 0. 0.1 1E-12 1.3 1.3

.OPTIONS ITLDC=100 ABSTOL=1E-8 TRANTOL=1E-8 ALPHA=0.75 TRPZ=1
```



(a) LR circuit

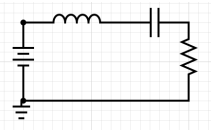


(b) AC voltage

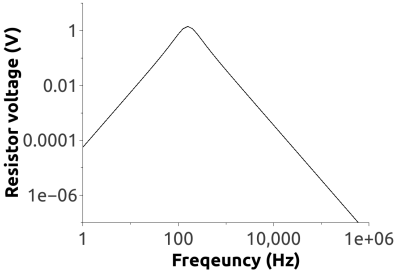


(c) Transient voltage

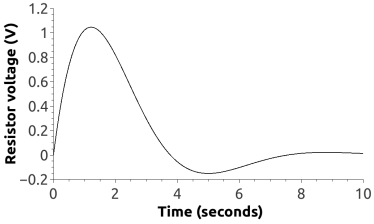
Figure 12.2: LR circuit of time constant 1 sec in (a) is simulated to calculate magnitude of voltage across the capacitor in (b) AC analysis with $L = 1$H, and in (c) Transient analysis with $L = 1$kH.



(a) RLC circuit



(b) AC voltage



(c) Transient voltage

Figure 12.3: RLC circuit of time constant 1 sec in (a) is simulated to calculate magnitude of voltage across the capacitor in (b) AC analysis with $L = 1\text{H}$, $C = 1\mu\text{F}$, and in (c) Transient analysis with $L = 1\text{kH}$ and $C = 1\text{mF}$.

```
Vs    1    0   DC   1. AC 1. 0 PWL(0. 2. 1. 2. 20. 2.)
Ln    1    2    1E3
Cm    2  out    1E-3
Rm    out  0    1E3
.end
```

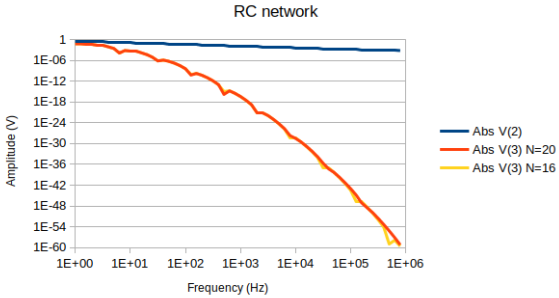
Note, that each of the analyses were 'activated' successively, such that *only* one analysis is active at a time. Resulting voltage across the resistor is plotted vs. frequency in Fig. 12.3(b) for AC analysis ($L = 1$H, $C = 1\mu$F) and is plotted vs. time in Fig. 12.3(c) for transient analysis ($L = 1$kH and $C = 1$mF). Note, that `ALPHA` was changed to 0.75 to avoid spurious oscillations arising from Trapezoidal transients.

12.4 RC-networks

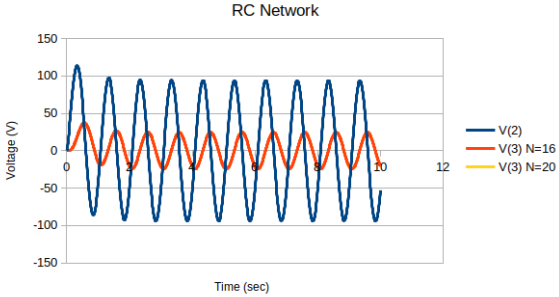
A circuit composed of a RC-network component is simulated using the spice file below.

```
.TRAN 0.01 10. 0. 0.01 1E-12 1.1 1.1
*.AC DEC 10 1. 1M

.OPTIONS ITLDC=100 ABSTOL=1E-8 TRANTOL=1E-8 ALPHA=0.51 TRPZ=1 ITLINNI

.model  RC1 RC ( RPERL=1 CPERL=1 L=1 N=16)
Vs    1    0   DC   0. AC 1. 0 SIN(0. 320. 1.)
R1    1    2   1
U1    2    3   0   RC1
R2    3    0   1
.end
```

Each of the AC and transient analyses were 'activated' successively, such that *only* one analysis is active at a time. Resulting voltages at the input pin (2) and output pin (3) is plotted vs. frequency in Fig. 12.4(a) for AC analysis and are plotted vs. time in Fig. 12.4(b) for transient analysis. A comparison of the AC and transient response for 16 and 20 lumped segments (`N`) confirms that for *sufficiently* large `N`, choice of `N` does not alter the transfer characteristics of the RC network.



(a) AC voltage



(b) Transient voltage

Figure 12.4: A circuit consisting or a lumped RC network component is simulated to calculate magnitude of voltage at the input and output pins (2 and 3, respectively) in (b) AC analysis, and in (c) Transient analysis.

12.5 Lossless TL

A circuit composed of a lossless TL is simulated using the spice file below.

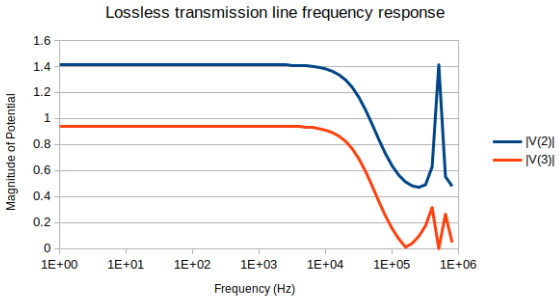
```
.TRAN 0.01 10. 0. 0.01 1E-12 1.1 1.1
*.AC DEC 10 1. 1M

.OPTIONS ITLDC=100 ABSTOL=1E-8 TRANTOL=1E-8 ALPHA=0.51 TRPZ=1 ITLINNI

.model  TL2s TL ( TD=1.5 Z0=1)

Vs    1    0   DC   0. AC 1. 0 SIN(0. 320. 1.)
R1    1    2   1
T1    2    0   3   0 TL2s
R2    3    0   1
.end
```

Each of the AC and transient analyses were 'activated' successively, such that *only* one analysis is active at a time. Resulting voltages at the input pin (2) and output pin (3) is plotted vs. frequency in Fig. 12.5(a) for AC analysis and is plotted vs. time in Fig. 12.5(b) for transient analysis. *Note, that for AC analysis, delay of 1μsec is used whereas for transient analysis delay of 1.5 sec is used.*

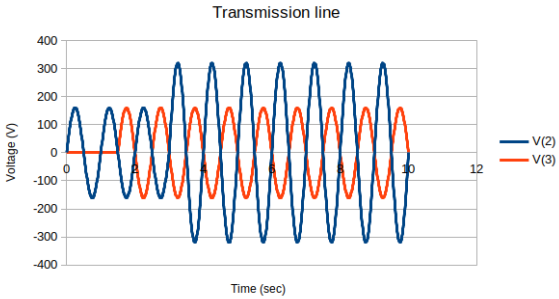
12.6 Lossy TL

A circuit composed of a lossy TL component is simulated using the spice file below.

```
.TRAN 0.01 10. 0. 0.01 1E-12 1.1 1.1
*.AC DEC 10 1. 1M

.OPTIONS ITLDC=100 ABSTOL=1E-8 TRANTOL=1E-8 ALPHA=0.51 TRPZ=1 ITLINNI

.model  TL2s TL (LEN=2 R=1 L=1 C=1)

Vs    1    0   DC   0. AC 1. 0 SIN(0. 320. 1.)
```



(a) AC voltage



(b) Transient voltage

Figure 12.5: A circuit consisting of distributed lossless TL component is simulated to calculate magnitude of voltage at the input and output pins (2 and 3, respectively) in (b) AC analysis, and in (c) Transient analysis. Note, that for AC analysis, delay of 1μsec is used whereas for transient analysis delay of 1.5 sec is used.

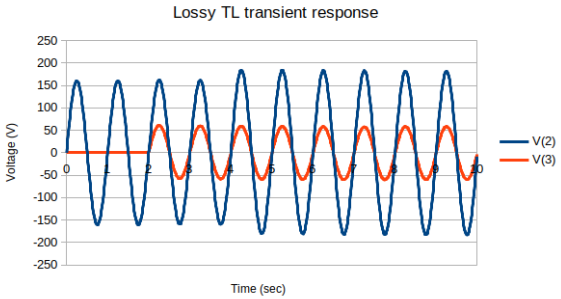


Figure 12.6: A circuit consisting of distributed lossless TL component is simulated to calculate voltage at the input and output pins (2 and 3, respectively) in transient analysis.

```
R1      1     2     1
O1      2     0     3     0 TL2s
R2      3     0     1
.end
```

Each of the AC and transient analyses were 'activated' successively, such that *only* one analysis is active at a time. Resulting voltages at the input pin (2) and output pin (3) is plotted vs. time in Fig. 12.5(b) for transient analysis. *Note, that for AC analysis, delay of 1µsec is used whereas for transient analysis delay of 1.5 sec is used.*

Chapter 13

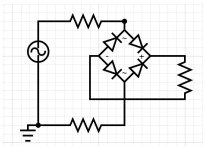
Nonlinear Circuit simulation

Examples of circuits with various nonlinear components such as diodes, BJTs, and MOSFETs are presented in this chapter together with the circuit files. The simulation results can be visualized using a plotting tool `qtiplot`.

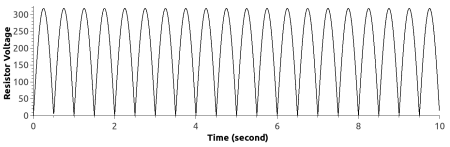
13.1 Full-bridge rectifier

A full-bridge rectifier circuit shown in Fig. 13.1(a) is simulated using the spice file below.

```
*.DC Vs 0.0 0.1 0.1
*.AC DEC 10 1. 1M
.TRAN 0.0001 10. 0. 0.01 1E-12 1.1 1.1

.OPTIONS ITLDC=100 ABSTOL=1E-8 TRANTOL=1E-8 ALPHA=0.51 TRPZ=1 ITLINNEI

.model Da1N4004 D ( IS=18.8 RS=0 BV=400 IBV=5.00u CJO=30 M=0.333 N=2)

Vs    1    0   DC   0. AC 1. 0 SIN(0. 320. 1.)
R1    1    2    1E-1
```



(a) Full-Bridge-Circuit



(b) Transient voltage

Figure 13.1: Full-bridge circuit in (a) is simulated to calculate voltage across the load resistor when a sinusoidal transient with peak voltage of 320V is applied.

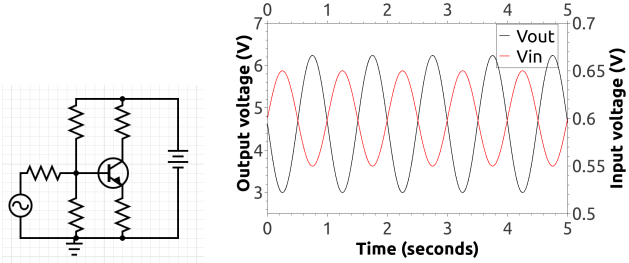
```
R2    3    0    1E-1
RL    4    5    1E3
Di1    4    2  Da1N4004 1.0
Di2    4    3  Da1N4004 1.0
Di3    2    5  Da1N4004 1.0
Di4    3    5  Da1N4004 1.0
.end
```

Note, that each of the analyses were 'activated' successively, such that *only* one analysis is active at a time. Resulting voltage across the load resistor is plotted vs. time in Fig. 13.1(b) in transient analysis.

13.2 BJT amplifier

A BJT amplifier circuit shown in Fig. 13.2(a) is simulated using the spice file below.

```
.DC Vsupply 0.0 12.001 0.2
.DC Vinput 0.0 0.601 0.1
```



(a) BJT amplifier circuit

(b) Output and input transients

Figure 13.2: BJT amplifier circuit in (a) is simulated to calculate output voltage at the collector terminal of the BJT when the sinusoidal input voltage of 50mV is applied. (b) The sinusoidal voltage at the output is plotted vs. time on the left Y axis together with the input voltage at the input (plotted on the right Y axis). Notice the range on both the left and the right Y axes. Also, notice that the amplifier introduces 180deg shift.

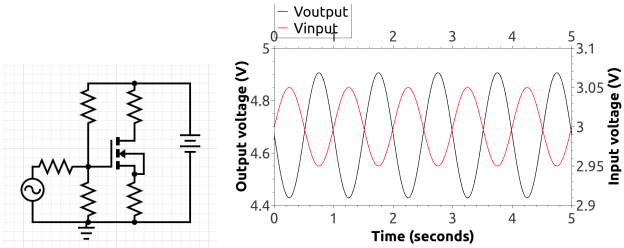
```
.TRAN 0.1m 10. 0. 0.005 1E-12 1.3 2.

.OPTIONS ITLDC=100 ABSTOL=1E-12 TRANTOL=1E-12 ALPHA=0.51 TRPZ=1

.model 2N2222 NPN (ISE=1E-10 ISC=1E-10 BF=200 BR=3)

.subckt bjtAmplifier Vc Vin Vout PARAMS: Rc=1K Rbc=22K Re=560 Rbe=6.8
R1 Vc Vout Rc
R2 Ve  0    Re
R3 Vb  0    Rbe
R4 Vc Vb   Rbc
Rin Vin Vb  10
Q1 Vout Vb  Ve 2N2222
.ends

Vsupply Vc 0 DC 2.0
Vinput Vin 0 DC 0.0 AC 1.0 0 SIN(0.6 0.05 1.)
X1 Vc Vin Vout bjtAmplifier PARAMS: Rc=10K Rbc=100K Rbe=1K Re=270
.end
```

Note, that the DC analyses were used to bias the circuit after which the transient analysis was performed. Resulting voltage at the collector terminal of the BJT is plotted vs. time in Fig. 13.2(b) in transient analysis.

13.3 MOSFET amplifier

A MOSFET amplifier circuit shown in Fig. 13.3(a) is simulated using the spice file below.

```
.DC Vsupply 0.0 12.001 0.2
.DC Vinput 0.0 3. 0.1

.TRAN 0.1m 10. 0. 0.005 1E-12 1.3 2.

.OPTIONS ITLDC=100 ABSTOL=1E-12 TRANTOL=1E-12 ALPHA=0.51 TRPZ=1

.model nmos_depl NMOS (KP=200u VTO=0.6 PHI=0.6 GAMMA=0)
```



(a) MOSFET amplifier circuit

(b) Output and input transients

Figure 13.3: MOSFET amplifier circuit in (a) is simulated to calculate output voltage at the drain terminal of the MOSFET when the sinusoidal input voltage of 50mV is applied. (b) The sinusoidal voltage at the output is plotted vs. time on the left Y axis together with the input voltage at the input (plotted on the right Y axis). Notice the range on both the left and the right Y axes. Also, notice that the amplifier introduces 180deg shift.

```
.subckt mosAmplifier Vd Vin Vout PARAMS: Rc=1K Rbc=22K Re=560 Rbe=6.8
R1 Vd Vout Rc
R2 Vs  0    Re
R3 Vg  0    Rbe
R4 Vd Vg   Rbc
Rin Vin Vg  10
M1 Vout Vg Vs Vs nmos_depl
.ends

Vsupply Vc 0 DC 2.0
Vinput Vin 0 DC 0.0 AC 1.0 0 SIN(3. 0.05 1.)
X1 Vc Vin Vout mosAmplifier PARAMS: Rc=10K Rbc=100K Rbe=1K Re=270
.end
```

Note, that the DC analyses were used to bias the circuit after which the transient analysis was performed. Resulting voltage at the collector terminal of the BJT is plotted vs. time in Fig. 13.3(b) in transient analysis.

Chapter 14

Circuit Optimization

Compact models of semiconductor devices should be able to reproduce the experimental data under different biasing conditions. This is achieved by fitting or calibration of the model parameters to the experimental data. This is achieved by minimizing the difference between the measure and simulated I-V data. The circuitsolver provides built-in minimizer to perform calibration of the compact model to the experimental data.

In order to get the highest performance or the lowest power loss, electronic circuits containing nonlinear electronic devices as well as linear components (R,L,C, etc.) are optimized by changing the device scaling factor and values of the linear components. This can also be achieved by the built-in minimizer.

Various procedures which can be used to perform the calibration or optimization are described in this chapter.

14.1 Optimizer functions

14.1.1 Initialize

An optimizer object runs optimization algorithm on a predefined circuit to get the optimum values of the component parameters subject to a specified set of targets. Therefore, the optimizer object requires a

circuit object for initialization. An optimizer object can be instantiated as follows.

```
import circuitsolver as cs
p = cs.circuit()
p.readSpiceCircuitFile("RCckt.cir")
f = cs.optimizer(p)
```

resetOptimizationTargets

Deletes all the optimization targets defined by the user.

resetOptimizationParams

Delets all the fitting/optimization parameters defined by the user.

14.1.2 Setup optimizer

The following functions setup optimization settings, targets, and parameters.

setOptimizationAlgorithm

`f.setOptimizationAlgorithm(...)` sets optimization algorithm as well as other parameters such as termination criteria, etc. It takes the following arguments.

1. `Algorithm` : An optimization algorithm is set. One of the following algorithms can be set. –
   - 'Conjugate-Gradient' : Conjugate gradient optimization
   - 'NM-Simplex' : Nelder-Mead simplex based optimization
2. `MaxIter` : Maximum number of optimization iterations to be performed.
3. `GradientErrorTol` : When norm of the gradient is lower than this value, then optimization terminates.
4. `RelativeErrorTol` : When the norm of the vector of consecutive values of parameters are less than this value, then optimization terminates.

addOptimizationParameter

`f.addOptimizationParameter(...)` adds a new circuit component parameter to be optimized. It takes the following arguments.

1. `Component` sets the address of the component whose parameter is to be optimized.
2. `Param` sets the parameter name to be optimized.
3. `StartValue` sets parameter value at the start of the optimization.
4. `MinValue` sets minimum allowed parameter value.
5. `MaxValue` sets maximum allowed parameter value.

addExperimentalDataAndExpression

`f.addExperimentalDataAndExpression(...)` adds a new optimization target for the optimization procedure. The target value is calculated for the specified analysis at the end of the circuit simulation. All the target values are added to get the total value. If the file containing the experimental data is provided, then the optimization performs fitting of the parameter to the experimental data. Otherwise, the target value is minimized. The function takes the following arguments.

1. `AnalysisType` specifies one of the following analysis types – 'DC', 'AC', and 'TRAN'.
2. `AnalysisId` specifies the analysis id. Note, it is returned when a new analysis is added to the `circuit`.
3. `FileName` sets the csv file name in which the experimental data is listed.
4. `OptimExpression` specifies the mathematical expression to be evaluated at each data point, e.g. "(V(0)-V(1))*V(2)".
5. `Scaling` sets the scaling factor to be multiplied to the target before summing all the targets.

6. `IntegrationStart` sets the start time/voltage/frequency of integration for target calculation.
7. `IntegrationEnd` sets the end time/voltage/frequency of integration.
8. `MinValue` sets minimum permissible value of the target, below which penalty is added to the target.
9. `MaxValue` sets maximum permissible value of the target, above which penalty is added to the target.
10. `PenaltyFactor` sets multiplication factor for penalty calculation.

addExperimentalDataAndNodeVoltages

`f.addExperimentalDataAndNodeVoltages(...)` adds a new optimization target for the optimization procedure. This function takes same arguments as the function Ref. **??**, except `OptimExpression`. The target is calculated from node voltages instead of a mathematical expression. The node voltages are specified by the following arguments *in addition* to the arguments listed in Ref. **??**.

- `VNodePlus` specifies address of the node whose voltage is added to the target.
- `VNodeMinus` specifies address of the node whose voltage is subtracted from the target.

addExperimentalDataAndCurrent

`f.addExperimentalDataAndCurrent(...)` adds a new optimization target for the optimization procedure. This function takes same arguments as the function Ref. **??**, except `OptimExpression`. The target is calculated from a device current instead of a mathematical expression. The device current specified by the following arguments *in addition* to the arguments listed in Ref. **??**.

- `INode` address of the node.
- `ComponentName` component connected to the node. Current *entering* this component is added to the target.

addExperimentalDataAndPower

`f.addExperimentalDataAndPower(...)` adds a new optimization target for the optimization procedure. This function takes same arguments as the function Ref. **??**, except `OptimExpression`. The target is calculated from a device current and node voltages instead of a mathematical expression. They specified by the following arguments *in addition* to the arguments listed in Ref. **??**.

Power to be added to the target is calculated as follows.

$$P = (V^+ - V^-) \cdot I_{\mathrm{n,comp}} \tag{14.1}$$

- `VNodePlus` specifies address of the node whose voltage is added (V^+).
- `VNodeMinus` specifies address of the node whose voltage is subtracted (V^-).
- `INode` address of the node 'n' of $I_{\mathrm{n,comp}}$.
- `ComponentName` component 'comp' connected to the node. Current *entering* this component is added to the target.

14.1.3 Optimize

optimize

`f.optimize()` begins optimization procedure. Optimization of the parameters registered till now is carried out. The targets added till now are added together to form a single optimization target.

pause

`f.pause()` pauses the optimization procedure, if it is currently running. If the optimization procedure is already paused, it does nothing.

resume

`f.resume()` resumes the optimization procedure, if it is paused before. If the optimization procedure is running, it does nothing.

stop

`f.stop()` stops running or paused optimization procedure.

getOptimizedValues

`f.getOptimizedValues()` returns parameter values at the end of the optimization as a python list of numbers. Order of the parameter values in the list is same as the order in which the parameters are added.

14.2 Example Code

The following example python code describes optimization of a resistive divider circuit to maximize power in the load resistor.

```
import circuitsolver as cs
# define circuit
p = cs.circuit()
p.readSpiceCircuitFile ("CircuitTrial.cir")

# define optimizer
f = cs.optimizer(p)
f.setOptimizationAlgorithm(Algorithm = "NM-Simplex",
      MaxIter=100,GradientErrorTol=1E-10,
      RelativeErrorTol=1E-5)
f.addOptimizationParameter(Component="R0",
      Param="Value", StartValue=10,
      MinValue=1, MaxValue=200)
f.addExperimentalDataAndPower(AnalysisType="DC",
      AnalysisId=0, FileName="", VNodePlus="2",
      VNodeMinus="0", INode="2", ComponentName="R0",
      Scaling=-1, IntegrationStart=0,
      IntegrationEnd=10, MinValue=-10000,
      MaxValue=10000, PenaltyFactor=10)
# perform optimization
f.optimize()
# read output
```

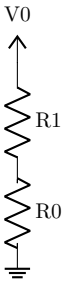


Figure 14.1: The resistive divider circuit defined in the file 'CircuitTrial.cir'.

```
out = f.getOptimizedValues()
```

The above code reads a resistive divider circuit from the file 'CircuitTrial.cir' into the `circuit` object `p`. Contents of the file are given below.

```
.DC V0 0 10 1
R0 2 0 10
R1 2 1 100
V0 1 0 DC 10 AC 10 1
.end
```

Circuit defined in the above file is shown in Fig. 14.1. In the circuit, resistance of `R1` is fixed while that of `R0` is adjustable. The optimization task is to maximize power dissipated in `R0` by adjusting the resistance r ($\in (1, 200)$). The example code solves this task as follows.

The line `p.readSpice...` reads the circuit file and creates the circuit shown in Fig. 14.1. A `DC` analysis with analysis-id '0' is also defined in the file.

The line `cs.optimizer(p)` creates a new optimizer object which would optimize the `circuit` object `p`.

The next line `f.setOptimizationAlgorithm(...)` specifies 'Nelder-Mead simplex' algorithm as an optimization algorithm and also sets various other optimization parameters described before.

The line `f.addOptimizationParameter(...)` adds the resistance of `R0` as an optimization parameter. Note that resistance is set by the parameter `Value`. Thus, at least one optimization parameter is added.

The line `f.addExperimentalDataAndPower(...)` adds the optimization target. It specifies, that power dissipated in the resistor $((V(2) - V(0)) * I_{2,\mathrm{R0}})$ is the optimization target. The target is calculated by integrating power in the ramp from `IntegrationStart` to `IntegrationEnd` in the `DC` analysis with id `0`. Note, that the optimization algorithm minimizes the target. However, the scaling factor -1 causes 'maximization' of the power dissipated in resistor `R0`.

The line `f.optimize()` performs optimization, and `f.getOptimizedValues()` returns the optimized values of all the parameters as a python list.

Chapter 15

Circuit Drawing Application

Circuit drawing, simulations, and curve plotting can also be done with a graphical-user-interface (gui) of the circuit solver. The gui can be opened using the following line -

```
circuitdraw &
```

The above command opens a gui window which consists of a 'Menu-bar', a 'Tool-bar', a circuit drawing board, and a side-pane. The 'Menu-bar' contains various functions as drop-down menus, while the 'Tool-bar' contains some of the more common functions as short-cut buttons. The 'side-pane' contains a stack of different widgets. Each of the widgets contains an information on the circuit, such as analyses, parameters, etc. or it assists the user in curve-plotting or optimization. The components added to the circuit appear on the circuit drawing board. The components can be connected to each other by drawing a 'wire' joining them. The `circuitdraw` tool saves the current state of the circuit into a file with extension '*.cktdr'. This file can be opened later in the `circuitdraw` tool to load the state of the circuit.

Various functionalities available as a single-click buttons on the gui window are described in Fig. 15.1. Each of the above functions has been described in detail below.

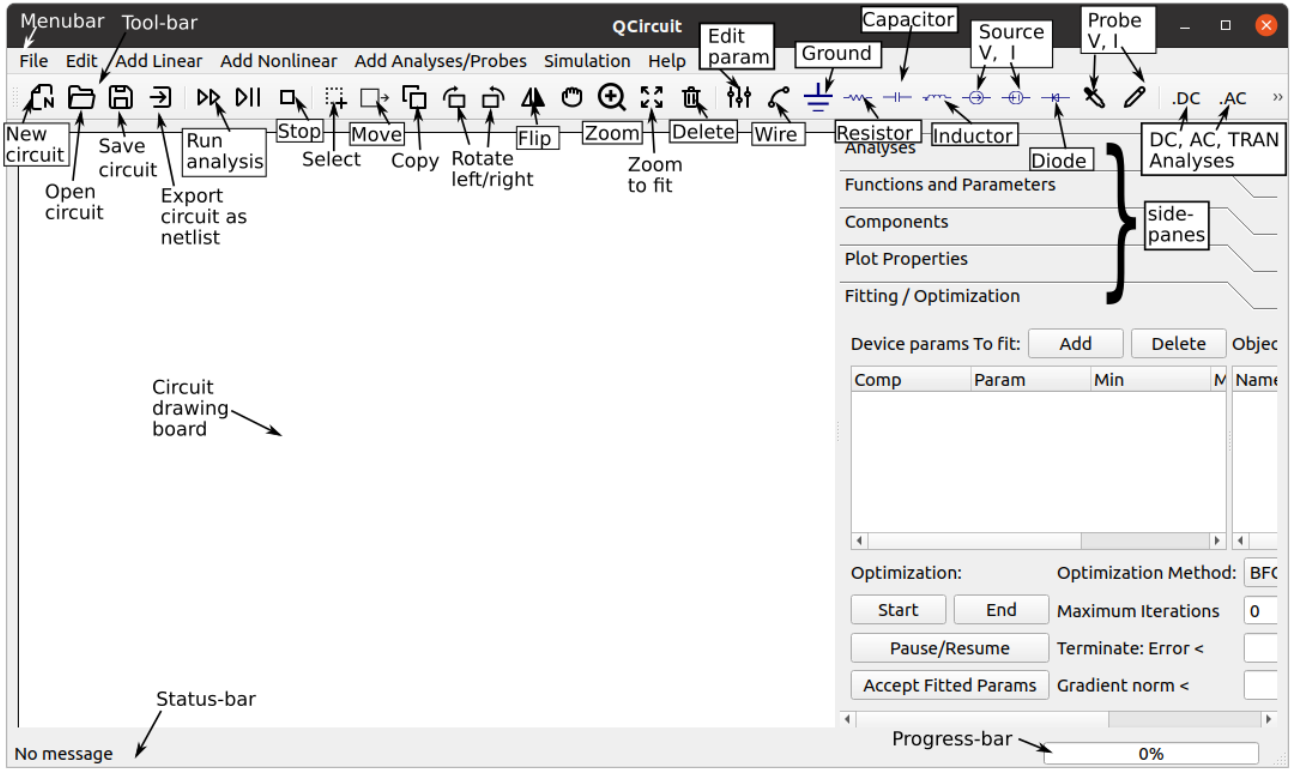


Figure 15.1: Graphical-user-interface of the circuit drawing tool.

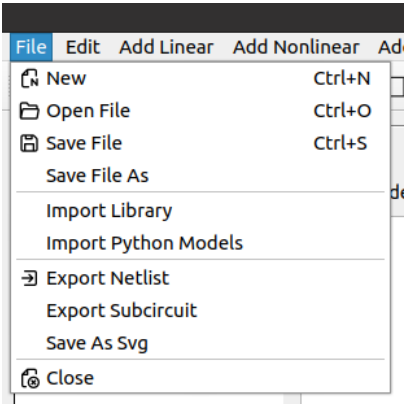


Figure 15.2: File menu.

15.1 File-menu

'File' menu on the menu-bar shows the following 'items' on click (see Fig. 15.2).

15.1.1 New

When clicked, it opens a new `circuitdraw` window keeping the current circuit unmodified. *If the current circuit window is empty, then it does nothing.*

15.1.2 Open File

When clicked, it opens a dialog box to select the '*.cktdr' file to be opened. The selected '*.cktdr' file is opened in a new `circuitdraw` window. *If the current window is empty, then the circuit file is loaded to the current window instead.*

15.1.3 Save File

When clicked, it opens a dialog box to input file-name and location of the '*.cktdr' to be saved. Following information is stored in the '*.cktdr' file with the given name and at the given location.

- Circuit drawing and connectivity
- Modified component parameters
- Analyses
- User-defined functions and parameters
- User-defined I/V probes
- User-defined optimization parameters, targets, and settings,
- Loaded library files.

15.1.4 Save File As

When clicked, it opens a dialog box to input file-name and location of the '*.cktdr' to be saved. Current state of the `circuitdraw` is stored.

15.1.5 Import Library

When clicked, it opens a dialog box to select the spice library ('*.lib' file) to be loaded. The loaded file is read and all the models/subcircuits are loaded to the current window.

15.1.6 Import Python Models

When clicked, it opens a dialog box to select the python file containing functional or behavioural models ('*.py' file). The loaded file is read and all the valid python functional and behavioural models are loaded to the current window.

15.1.7 Export Netlist

When clicked, it opens a dialog box to input file-name and location of the spice '*.cir' file. Spice netlist of the circuit, including the imported spice libraries, analyses, and user-defined functions/parameters are stored in the file. This file can be used to run spice simulations using the `circuitsolver`.

15.1.8 Export Subcircuit

When clicked, it opens a dialog box to input file-name and location of the spice '*.lib' file. Spice netlist of the circuit, including user-defined functions/parameters are stored in the file as a *sub-circuit*. Pins defined in the circuit are set as the output pins of the stored sub-circuit.

15.1.9 Save As Svg

When clicked, it opens a dialog box to input file-name and location of the svg file '*.svg'. The circuit diagram is stored in SVG format in this file.

15.2 Edit-menu

'Edit' menu on the menu-bar shows the following 'items' on click (see Fig. 15.3).

Various actions can be 'toggled'. This means, if the given action is active, clicking on the action will deactivate it. If the given action is inactive, clicking on the action will activate it.

Various actions can be 'toggled'. That is, if the action is active,

15.2.1 Select

When clicked, 'select' action is toggled. When 'Select' is active, Draw any arbitrary rectangle in the circuit drawing window by press → drag → release left mouse button. All the components and the wires in the rectangle are selected.

Alternately, individual components can be selected by clicking on them while 'Select' action is active.

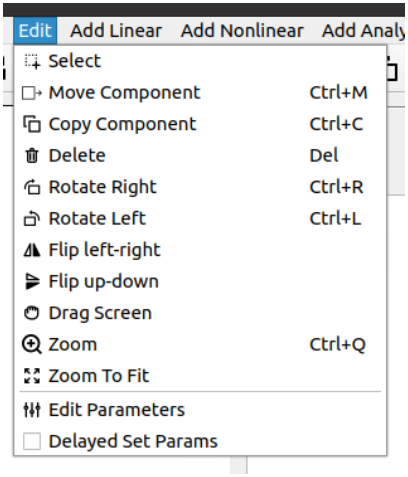


Figure 15.3: Edit menu.

The selected components and wires are painted in *red*.

15.2.2 Move Component

When clicked, 'Move Component' action is toggled. When 'Move Component' is active, press left mouse button on the component to be moved, drag the cursor to the desired location, and relase it. The component will be moved to the new location. Note, that the wires connected to the component will not be moved. Hence, this operation changes connectivity.

If one or more components have been previously selected by Select, then they are moved by the above action.

15.2.3 Copy Component

When clicked, 'Copy Component' action is toggled. When 'Copy Component' is active, press left mouse button On the component to be copied, drag the cursor to the desired location, and relase it. A

new component same as the clicked one will be created and moved to the new location. Note, that the wires connected to the component will not be copied/moved.

If one or more components have been previously selected by Select, then they are copied by the above action.

15.2.4 Delete

When clicked, 'Delete' action is toggled. When 'Delete' is active, left mouse click on any of the component or wire will delete it.

If one or more components have been previously selected by Select, then they are deleted when 'Delete' is clicked.

15.2.5 Rotate Right/Left

When clicked, 'Rotate Right' or 'Rotate Left' action is toggled. When 'Rotate' is active, left mouse click on any of the component will rotate it *clockwise* or *anticlockwise*, respectively.

15.2.6 Flip Up-down/Right-left

When clicked, 'Flip up-down' or 'Flip left-right' action is toggled. When 'Flip' is active, left mouse click on any of the component will flip it *vertically* or *laterally*, respectively.

15.2.7 Drag Screen

When clicked, 'Drag-screen' action is toggled. When 'Drag-screen' is active, drag the circuit drawing screen by press → drag → release left mouse button anywhere on the screen.

15.2.8 Zoom

When clicked, 'Zoom' action is toggled. When 'Zoom' is active, draw any arbitrary rectangle in the circuit drawing window by press → drag → release left mouse button. The circuit drawing area will be zoomed to the rectangular area. Alternately, rolling mouse-wheel can zoom-in or zoom-out the circuit diagram, where the mouse-pointer is present.

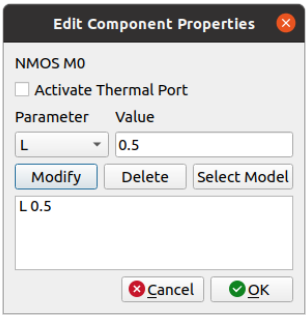


Figure 15.4: 'Edit Parameters' window.

15.2.9 Zoom to fit

When clicked, the circuit diagram is zoomed to fit the entire circuit drawing. Note, that this is not a toggle-able action.

15.2.10 Edit Parameters

When clicked, 'Edit Parameters' action is activated. Left mouse click on any of the component will open the component properties window (see Fig. 15.4) for that component. In this window, component parameter values can be modified, a new model is assigned to the component, or thermal port of the component is activated.

15.2.11 Delayed Set Params

15.3 Add Linear Menu

'Add Linear' menu lists linear devices such as R, L, C, I/V-sources, controlled sources, etc.

15.3.1 Wire

When clicked, 'Wire' is toggled. When active, press left mouse button on a pin of the device, drag the cursor to another pin of the

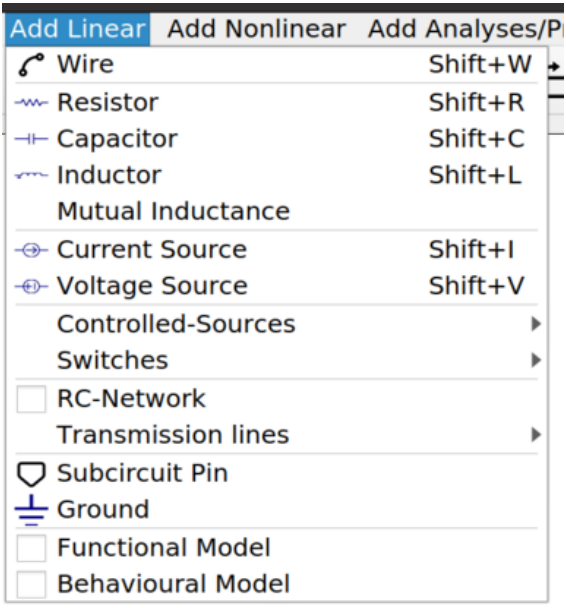


Figure 15.5: 'Add Linear' menu.

other/same device, and release the button. This action will create a wire from a pin to another.

15.3.2 Resistor, Inductor, Capacitor

When clicked, 'Resistor', 'Inductor', and 'Capacitor' actions are toggled. When active, click left mouse button on the circuit drawing area to place the respective component there.

15.3.3 Mutual Inductor

Use 'Select' action to select two or more 'Inductor' components in the circuit. After selecting, click on 'Mutual Inductance' button. A window will popup asking for mutual inductance k between the selected inductors. This will add a 'Mutual Inductance' to the circuit.

15.3.4 I/V sources and ground

When clicked, I/V sources or ground buttons are toggled. When the actions are active, clicking on the circuit drawing area adds the respective component to the circuit.

15.3.5 Controlled sources

When clicked, a drop-down menu appears, which contains toggle-able items of CCCS, CCVS, VCCS, and VCVS. When the actions are active, clicking on the circuit drawing area adds the respective component to the circuit.

15.3.6 Switches

When clicked, a drop-down menu appears, which contains toggle-able items of Current- and Voltage-controlled switches. When the actions are active, clicking on the circuit drawing area adds the respective component to the circuit.

Figure 15.6: 'Functional model' selection window.

15.3.7 Transmission lines

When clicked, a drop-down menu appears, which contains toggleable items of lossless and lossy transmission-lines. When the actions are active, clicking on the circuit drawing area adds the respective component to the circuit.

15.3.8 RC-Network

When clicked, 'RC-Network' action are toggled. When active, clicking left mouse button on the circuit drawing area to place the respective component there.

15.3.9 Subcircuit Pin

When clicked, 'Subcircuit Pin' action is toggled. When active, clicking on the circuit drawing area adds pin to it. When File → Export Subcircuit is clicked, a subcircuit with all the 'pins' as output pins is exported to a *.lib file.

15.3.10 Functional model

When clicked and activated, a new window (Fig. 15.6) opens which asks users to select the functional model to be added from the drop-down model list. Output pins of the functional model are also requested. Once selected, the new functional model is added to the circuit drawing area wherever the user clicks.

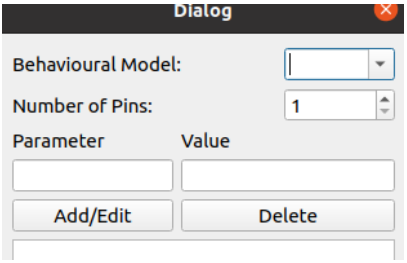


Figure 15.7: 'Behavioural model' selection window.

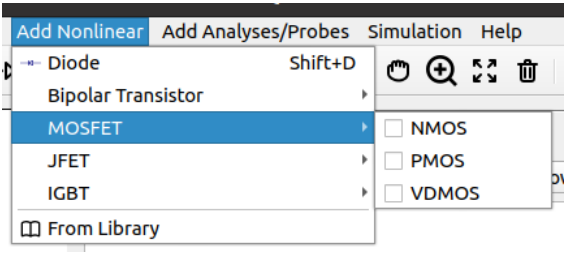


Figure 15.8: 'Add Nonlinear' menu.

15.3.11 Behavioural model

When clicked, a new window opens (Fig. 15.7) which asks users to select the behavioural model to be added from the drop-down model list. Parameters of the behavioural model can also be input. Once selected, the new behavioural model is added to the circuit drawing area wherever the user clicks.

15.4 Add Nonlinear Menu

'Add Nonlinear' menu lists nonlinear devices.

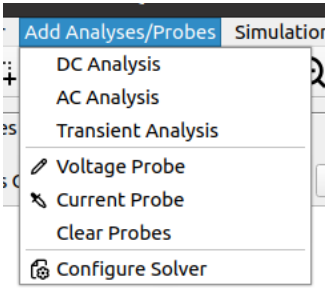


Figure 15.9: 'Add Analysis/Probe' menu.

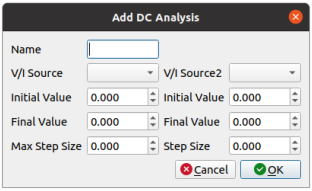


Figure 15.10: 'DC Analysis' dialog box.

15.4.1 Nonlinear components

Clicking the components in the component list toggles the actions. When the action is action is active, a left click in the circuit drawing area adds the respective component to the circuit.

15.5 Add Analyses/Probes Menu

'Add Analyses/Probes' menu lists actions which add AC, Direct Current (DC), transient analyses, and V/I probes. An action to configure the solver is also present.

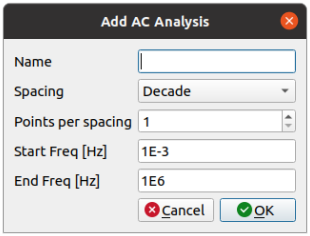


Figure 15.11: 'AC Analysis' dialog box.

15.5.1 DC Analyses

Clicking 'DC Analyses' opens a new dialog box to configure DC analysis. The dialog box is shown in Fig. 15.10. The user-specified analysis name is used elsewhere to refer to the given DC analysis.

V/I source can be selected from the already added sources which are also seen in the drop-down list. For analysis on a 2D grid of bias points (e.g. MOSFET transfer plots for different V_{ds}), second source is also selected. Else, the second source is kept empty. Initial and final values of DC ramp, together with the step-size are also set.

15.5.2 AC Analyses

Clicking 'AC Analyses' opens a new dialog box to configure AC analysis. The dialog box is shown in Fig. 15.11. The user-specified analysis name is used elsewhere to refer to the given AC analysis. Frequeny spacing, number of points per spacing, and start/end frequencies must be specified.

15.5.3 Transient Analyses

Clicking 'Transient Analyses' opens a new dialog box to configure transient analysis. The dialog box is shown in Fig. 15.12. The user-specified analysis name is used elsewhere to refer to the given transient analysis. Additional parameters such as start/end times, max/min time-steps, initial time-step are specified here.

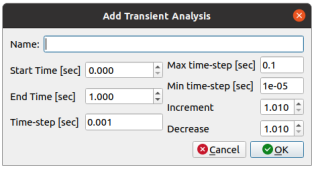


Figure 15.12: 'Transient Analysis' dialog box.

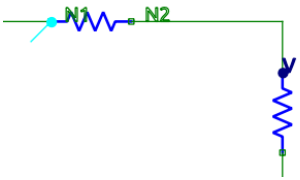


Figure 15.13: Diagram showing current and voltage probes placed at the pins.

If the simulations converge, time-step is scaled up by 'increment' factor, else it is scaled down by 'Decrease' factor.

15.5.4 Voltage/Current probes

On click, 'Voltage Probe' or 'Current Probe' actions are toggled. When 'Voltage Probe' is active, a left-click on any of the component pins adds voltage probe at that pin. When 'Current Probe' is active, a left-click on any of the component pin adds a current probe at the pin. This current probe measures current entering that particular component. Voltage or current of these pins can be plotted after simulations are over. Also, in order to use node I/V as targets of optimization, a probe must be place at the pins before the simulation.

Voltage and current probes placed at the pin are marked by a dark blue circle with 'V' sign and a cyan probe line, respectively, as shown in Fig. 15.13.

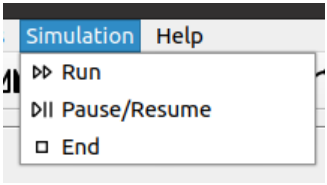


Figure 15.14: 'Simulation' menu.

15.5.5 Clear Probes

On click, this action clears any of the probes present in the circuit.

15.5.6 Configure Solver

On click, 'Configure Solver' action opens a window in which various solver configurations can be viewed and modified.

15.6 Simulation Menu

The simulation menu lists actions which 'Run', 'Pause/Resume', and 'Stop'.

15.6.1 Run

On click, this action begins simulation of the opened circuit. If spice netlist of the circuit is not yet exported, it is exported and then simulations begin.

15.6.2 Pause/Resume

On click, this action pauses currently running simulations. If the simulations are already paused, it resumes the simulations.

15.6.3 Stop

On click, this action stops running simulations.

Various actions listed above can also be run from the tool-bar.

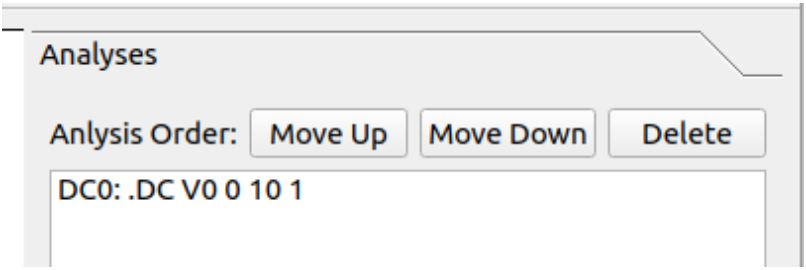


Figure 15.15: 'Analyses' tab in the side-pane.

15.7 Side-panes

Following tabs are present on the side-pane of the circuit drawing area.

15.7.1 Analyses

'Analyses' tab lists all the analyses added to the circuit (see Fig. 15.17). The analyses entries are listed as follows.

`<name><id>: <spice command>`

Here, `<name>` means user-defined analysis name, `<id>` is analysis-id, and `<spice command>` is the line corresponding to the analysis in the spice simulator.

Multiple analyses can be added one after another. They will be listed in the side-pane. The click-button Move Up moves up the selected analysis in the list. The button Move Down moves the analysis down, whereas the button Delete deletes the analysis.

The analyses will be run according to the order in which they are listed in 'Analyses' side-pane.

15.7.2 Functions and Parameters

'Functions and Parameters' tab lists all the user-defined parameters and functions added to the circuit. The click-button New Parameter and New Function add a new parameter and a new function, respectively, to the circuit. The button Validate checks if the selected function

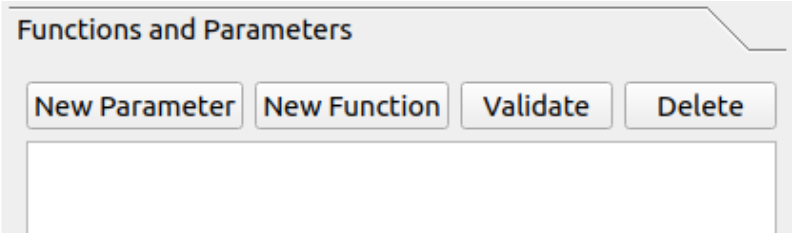


Figure 15.16: 'Functions and parameters' tab in the side-pane.

expression is a valid expression, and whether all the parameters used in the function are already defined. The button Delete deletes the selected parameter or function.

15.7.3 Components

'Components' tab lists all the components added. Note, that components list is updated only after the circuit is exported to the netlist circuit.

15.7.4 Plot Properties

'Plot Properties' tab provides an interface to plot current and voltages at the I/V-probes. `Select Analysis` selects the analysis whose data is plotted. Data to be plotted on left and right Y-axis are listed in left and right legends. Double clicking the legend can edit legend properties. Min/max span of X, left-Y, and right-Y axis, axis labels can be specified in corresponding text boxes. Create Plot creates a new plot.

15.7.5 Fitting/Optimization

'Fitting/Optimization' tab (see Fig. 15.18) provides an interface to run optimization of the given circuit (see Fig. 15.17).

Left window in the tab displays a list of fitting-parameters added by the user. To add one or more fitting paremeters, left-click on the adjacent toggle-able Add button. It will toggle state. When Add

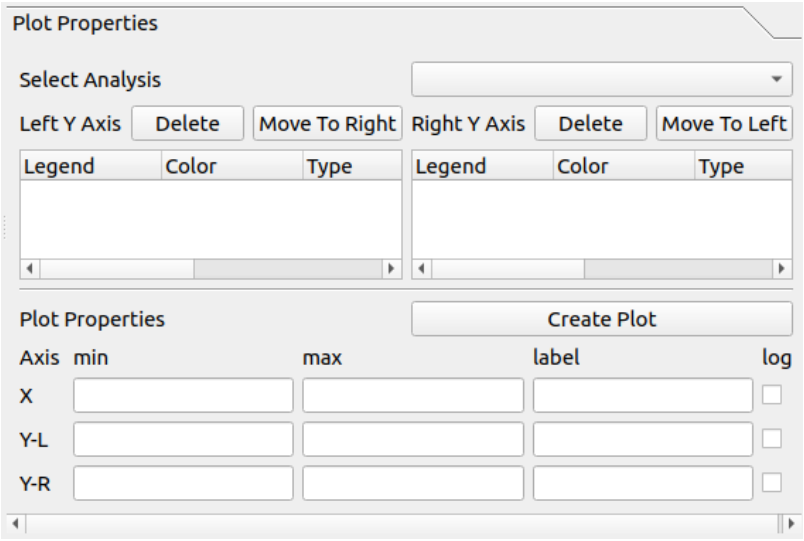


Figure 15.17: 'Plot Properties' tab in the side-pane.

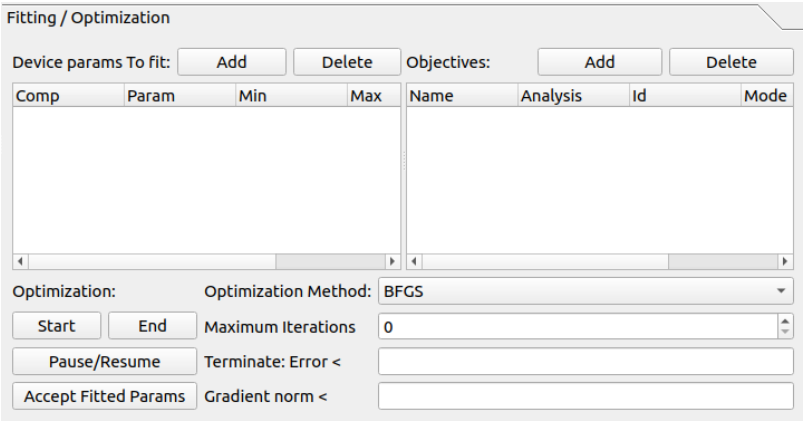


Figure 15.18: 'Fitting/Optimization' tab in the side-pane.

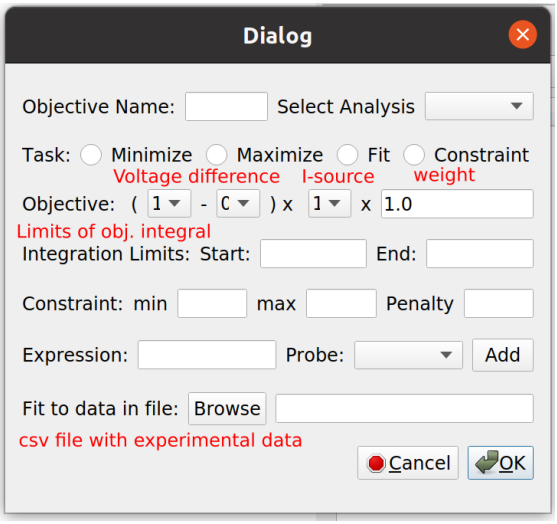


Figure 15.19: 'Fit Objective' window to add an objective.

is active, click on any of the components in the circuit to select the component whose parameter is to be fitted/optimized. A dialog box requesting the component parameter and its min, max, current values opens up. Clicking Ok adds the selected parameter of the component as a fitting parameter. Clicking Delete deletes *highlighted* parameter from the fitting-parameters list.

Right window in the tab displays a list of user-defined fitting/optimization objectives. To add an objective, left-click on the adjacent Add button. A dialog box as shown in Fig. 15.19 opens up. In this box, you can select objective name, analysis, expression to be used for fitting, etc. In the case of fitting, a csv file containing experimental data corresponding to the objective should also be provided.

Input parameters for the fitting procedure including fitting algorithm are specified in the bottom right corner. For description of the parameters, refer to Chapter 14.

The buttons Start, End, and Pause/Resume are used to start the optimization, end it, or pause a running optimization/resume a stopped one.

Appendix A

Notation and Acronyms

Acronyms

AC	Alternating Current
BE	Backward-Euler
BJT	Bipolar Junction Transistor
CCCS	Current-controlled Current Source
CCVS	Current-controlled Voltage Source
MOSFET	Metal-Oxide Semiconductor Field Effect Transistor
DC	Direct Current
JFET	Junction Field Effect Transistor
LTE	Local Truncation Error
MESFET	Metal-Semiconductor Field Effect Transistor

RHS	Right Hand Side
TL	Transmission Line
VCCS	Voltage-controlled Current Source
VCVS	Voltage-controlled Voltage Source

Bibliography