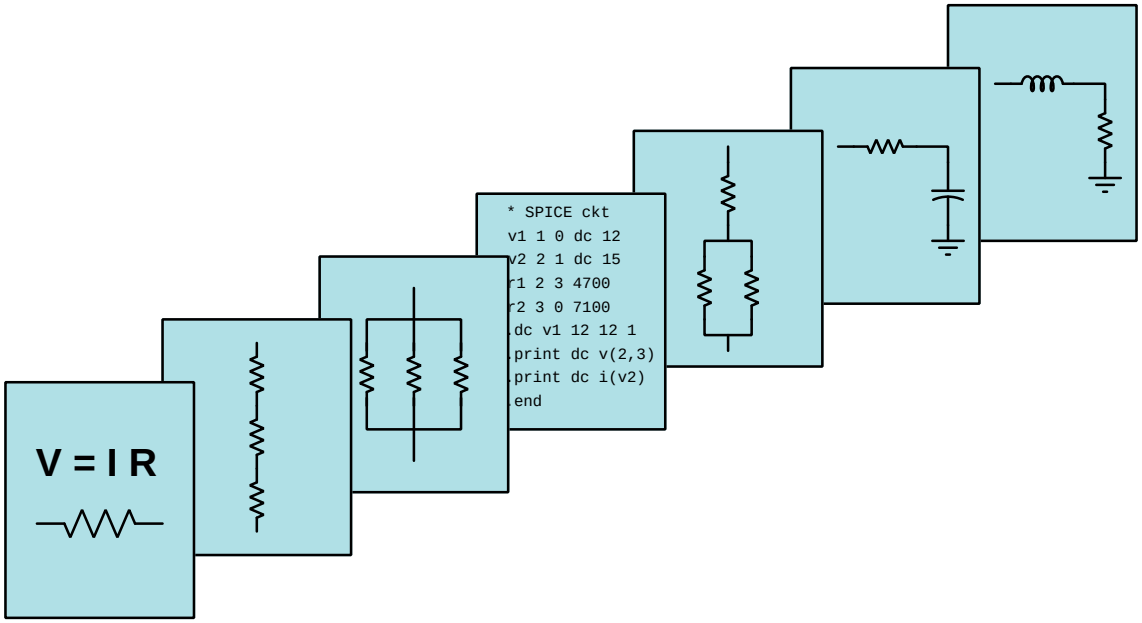
Modular Electronics Learning (ModEL) project



SPICE Modeling of Resistor Circuits

© 2016-2024 by Tony R. Kuphaldt – under the terms and conditions of the Creative Commons Attribution 4.0 International Public License

Last update = 6 September 2024

This is a copyrighted work, but licensed under the Creative Commons Attribution 4.0 International Public License. A copy of this license is found in the last Appendix of this document. Alternatively, you may visit `http://creativecommons.org/licenses/by/4.0/` or send a letter to Creative Commons: 171 Second Street, Suite 300, San Francisco, California, 94105, USA. The terms and conditions of this license allow for free copying, distribution, and/or modification of all licensed works by the general public.

Contents

1 Introduction **3**
- 1.1 Recommendations for students . . . 3
- 1.2 Challenging concepts related to SPICE circuit simulation . . . 5
- 1.3 Recommendations for instructors . . . 6

2 What is SPICE? **7**

3 Using SPICE **9**
- 3.1 Summary of steps . . . 11
- 3.2 Demonstration on Microsoft Windows . . . 12
  - 3.2.1 Invoking a text editor . . . 12
  - 3.2.2 Saving the deck . . . 13
  - 3.2.3 Invoking SPICE . . . 15
  - 3.2.4 Viewing the SPICE analysis . . . 16
- 3.3 Demonstration on Linux or CygWin . . . 17
  - 3.3.1 Invoking a text editor . . . 17
  - 3.3.2 Saving the deck . . . 18
  - 3.3.3 Invoking SPICE . . . 19
  - 3.3.4 Viewing the SPICE analysis . . . 20
- 3.4 Demonstration of NGSPICE interactive mode . . . 21
  - 3.4.1 Creating the netlist . . . 22
  - 3.4.2 Starting NGSPICE . . . 22
  - 3.4.3 Verifying the loaded netlist . . . 23
  - 3.4.4 Running the analysis . . . 23
  - 3.4.5 Printing a voltage . . . 24
  - 3.4.6 Plotting graphs using NGSPICE . . . 25
  - 3.4.7 Issuing multiple commands in a single line . . . 26
- 3.5 Idiosyncrasies of SPICE . . . 27
  - 3.5.1 Beginning and ending cards . . . 27
  - 3.5.2 Node zero . . . 27
  - 3.5.3 Current measurement . . . 27
  - 3.5.4 Open and short circuits . . . 28
  - 3.5.5 Multiple sources . . . 29
  - 3.5.6 Multiple inductors/capacitors . . . 30

4 SPICE component descriptions **31**
4.1 Independent voltage sources . . . 32
4.1.1 Example: DC source . . . 32
4.1.2 Example: "Dummy" source . . . 32
4.1.3 Example: AC source . . . 33
4.1.4 Example: Sinusoidal source . . . 35
4.1.5 Example: Pulse source . . . 36
4.2 Independent current sources . . . 37
4.2.1 Example: DC source . . . 37
4.2.2 Example: AC source . . . 37
4.3 Resistors . . . 38
4.3.1 Example . . . 38
4.4 Capacitors . . . 39
4.4.1 Example: Capacitor with initial charge . . . 39
4.4.2 Example: Uncharged capacitor . . . 39
4.5 Inductors . . . 40
4.5.1 Example: Inductor with initial charge . . . 40
4.5.2 Example: Uncharged inductor . . . 40
4.6 Transformers . . . 41
4.6.1 Example: 2:1 ratio step-down transformer . . . 41
4.7 Transmission lines . . . 42
4.7.1 Example: 50-Ohm transmission line with 10 nanosecond delay . . . 42
4.7.2 Example: half-wavelength (at 35 MHz) 300-Ohm transmission line . . . 42
4.8 Linear dependent sources . . . 43
4.8.1 Example: voltage-controlled voltage source . . . 44
4.8.2 Example: voltage-controlled current source . . . 44
4.9 Nonlinear dependent sources . . . 45
4.9.1 Example: multiplier . . . 45
4.10 Diodes . . . 46
4.10.1 Example: Generic diode . . . 46
4.10.2 Example: 1N4001 . . . 47
4.11 Bipolar Junction Transistors (BJTs) . . . 48
4.11.1 Example: Generic NPN transistor . . . 49
4.11.2 Example: 2N2907 . . . 49
4.12 Junction Field-Effect Transistors (JFETs) . . . 50
4.12.1 Example: Generic N-channel JFET . . . 51
4.13 Metal-Oxide Field-Effect Transistors (MOSFETs) . . . 52
4.13.1 Example: Generic N-channel depletion-type MOSFET . . . 53
4.13.2 Example: Generic N-channel enhancement-type MOSFET . . . 53
4.13.3 Example: Generic P-channel depletion-type MOSFET . . . 54
4.13.4 Example: Generic P-channel enhancement-type MOSFET . . . 54
4.14 Subcircuits . . . 55
4.14.1 Example: resistor subnetwork . . . 56
4.14.2 Example: solar cell array . . . 57

5 SPICE analysis descriptions **59**
5.1 DC voltage/current "sweep" analysis . . . 60
5.1.1 Example: sweep of voltage source . . . 60
5.1.2 Example: sweep of voltage and current sources . . . 60
5.2 AC frequency "sweep" analysis . . . 61
5.2.1 Example: linear frequency sweep . . . 61
5.2.2 Example: decade logarithmic frequency sweep . . . 61
5.2.3 Example: octave logarithmic frequency sweep . . . 61
5.3 Transient analysis . . . 62
5.3.1 Example: using initial conditions, beginning at time $t = 0$ . . . 62
5.3.2 Example: using initial conditions, beginning at non-zero time . . . 62
5.4 Fourier analysis . . . 63
5.4.1 Example: analysis of 60 Hz waveform . . . 63
5.5 Display option: print . . . 64
5.5.1 Example: printing a DC analysis . . . 64
5.5.2 Example: printing an AC analysis . . . 64
5.6 Display option: plot . . . 65
5.6.1 Example: plotting a DC analysis . . . 65
5.6.2 Example: plotting an AC analysis . . . 65
5.6.3 Example: logarithmically plotting an AC analysis . . . 66
5.6.4 Example: plotting a transient analysis . . . 66
5.6.5 Example: plotting parametric functions . . . 66
5.7 Display option: width . . . 67
5.7.1 Example . . . 67

6 Primitive circuit examples **69**
6.1 DC voltage source with .op analysis . . . 70
6.2 DC voltage source with single-point .dc sweep analysis . . . 74
6.3 DC current source with single-point .dc sweep analysis . . . 77
6.4 DC voltage source with multi-point .dc sweep analysis . . . 78
6.5 AC voltage source with single-point .ac sweep analysis . . . 81
6.6 AC voltage source with multi-point .ac sweep analysis . . . 82
6.7 Additive AC voltage sources with single-point .ac sweep analysis . . . 86
6.8 Transient analysis of discharging RC circuit . . . 88
6.9 Transient analysis of a steady sinusoidal voltage source . . . 93
6.10 Transient analysis of a sinusoidal capacitive circuit . . . 96
6.11 Transient analysis of a damped, offset sinusoidal voltage source . . . 99
6.12 Additive AC voltage sources with transient analysis . . . 102
6.13 Solar cell array simulation . . . 104
6.14 Solar panel array simulation . . . 107

7 Gallery **113**
7.1 Using gallery examples for practice . . . 114
7.2 Series resistor circuits . . . 115
7.2.1 One DC current source, three resistors . . . 116
7.2.2 One DC voltage source, three resistors . . . 117

7.2.3 One DC voltage source, four resistors . . . 118
7.2.4 One DC current source, five resistors . . . 119
7.2.5 One DC voltage source, ten resistors . . . 120
7.2.6 Three DC voltage sources, four resistors . . . 122
7.3 Parallel resistor circuits . . . 123
7.3.1 One DC voltage source, three resistors . . . 124
7.3.2 One DC current source, three resistors . . . 125
7.3.3 One DC voltage source, four resistors . . . 126
7.3.4 One DC current source, five resistors . . . 128
7.3.5 Three DC current sources, two resistors . . . 130
7.4 Series-parallel resistor circuits . . . 131
7.4.1 One DC voltage source, six resistors . . . 132
7.4.2 One DC current source, six resistors . . . 134
7.5 Simple multi-source resistor circuits . . . 136
7.5.1 One voltage, one current . . . 137
7.5.2 Two currents, one voltage . . . 138
7.5.3 Two voltages, one current . . . 139
7.5.4 Two currents, two voltages . . . 140
7.6 Resistive bridge circuits . . . 141

A Problem-Solving Strategies **143**

B Instructional philosophy **145**
B.1 First principles of learning . . . 146
B.2 Proven strategies for instructors . . . 147
B.3 Proven strategies for students . . . 149
B.4 Design of these learning modules . . . 150

C Tools used **153**

D Creative Commons License **157**

E References **165**

F Version history **167**

Index **167**

Chapter 1

Introduction

1.1 Recommendations for students

Many computer-based circuit simulation software applications exist today, most of these based on graphical "schematic capture" where one describes the circuit to the computer by drawing a diagram of it. SPICE, however, is a throwback to a much earlier era of computing when people interacted with the machine through text (and in some cases, punched paper cards!). This means SPICE is a type of *programming language* where you describe the circuit to be simulated by written rather than graphic means. The fact that even the most modern circuit simulation packages often use SPICE-compliant or SPICE-like instructions "behind the scenes" to specify details of circuit analysis is one reason to learn SPICE in the 21st century, but another reason is that it is a relatively simple form of coding, and all students of electronics are well-served by learning to code in multiple languages. More specifically, SPICE is what may be called a *concurrent* programming language whereby the instructions you write using plain-text characters are not executed in sequential order but rather serve to describe entities of a circuit and therefore may appear in any order! In this way SPICE actually shares more in common with hardware-oriented languages like Verilog or VHDL than with classic procedural languages such as assembly or C.

Important concepts related to SPICE include **netlists**, **text editing**, **cards**, **decks**, **nodes**, **dependent sources** versus **independent sources**, **passive sign convention**, component **models**, **subcircuits**, , , , , , and .

One of the beautiful aspects of learning a programming language is the natural means by which you have to learn by experimentation. All you need is access to a computer with the proper software installed, and you can not only run example programs offered to you, but also modify those programs and/or even create your own to test your own growing understanding of how to "speak" the language. It is akin to learning to speak a human language and then being able to immediately apply it in conversation with someone else fluent in it as a way to test and hone your knowledge. Therefore, no reader of this module should be satisfied by merely reading what is written in these pages – instead *try running SPICE on your own* to see firsthand how it functions!

Here are some good questions to ask of yourself while studying this subject:

- How does a SPICE netlist instruct the computer to know which component terminals connect to which?
- How could you identify series versus parallel component connections by inspection of a netlist?
- What are some essential elements of any SPICE netlist?
- How may open-circuit conditions be simulated in SPICE without creating errors?
- How may short-circuit conditions be simulated in SPICE without creating errors?
- Where must we be careful when inserting multiple voltage sources into a netlist?
- Where must we be careful when inserting multiple current sources into a netlist?
- Where must we be careful when inserting multiple inductors into a netlist?
- Where must we be careful when inserting multiple capacitors into a netlist?
- Which node is always Ground in a netlist?
- How are voltage measurement points specified in a SPICE analysis?
- What are some different ways in which we may measure (or infer) current in a SPICE circuit simulation?
- What is a SPICE *subcircuit* and where might one be useful?
- How might you alter one of the example simulations shown in the text, and then determine the behavior of that altered circuit?
- Devise your own question based on the text, suitable for posing to someone encountering this subject for the first time.

The following active-reading strategies are advised for this module's tutorial(s);

- **Limiting cases** – seeing opens and shorts as being limiting cases of resistance provides a means of simulating the same in a SPICE netlist, extremely large resistances simulating opens and extremely small resistances simulating shorts.
- **Experiment using simulations** – SPICE is a programming language, and learning to program a computer is never a spectator sport. Run SPICE for yourself to see how it works!

1.2 Challenging concepts related to SPICE circuit simulation

The following list cites concepts related to this module's topic that are easily misunderstood, along with suggestions for properly understanding them:

- **Voltage as a relative quantity** – this is arguably one of the most difficult concepts students first encounter when studying electricity: understanding that voltage is a *difference* in energy levels between two locations, and not something that can exist at any single point. In this respect, voltage is similar to *distance* which is always something existing *between* two specified points.
- **Voltage versus Current** – these are two completely distinct things, each capable of existing without the other. *Voltage* is the amount of potential energy lost or gained by electric charge carriers as they move from one place to another, whereas *current* is the actual motion of electric charge carriers.
- **Test sweeps** – SPICE excels at performing "sweep" analyses, which refers to the incrementing or decrementing of some circuit parameter, re-running the entire analysis with each new value of this parameter. Sweeping is essential for building up a graph or table of values showing trends related to that parameter.

The *Gallery* chapter contains many example SPICE netlists complete with analyses, which are excellent both for becoming familiar with operating SPICE (i.e. copying and pasting those given netlists in your text editor and running the simulations yourself) as well as starting points (i.e. templates) for creating your own circuit simulations.

1.3 Recommendations for instructors

This section lists realistic student learning outcomes supported by the content of the module as well as suggested means of assessing (measuring) student learning. The outcomes state what learners should be able to do, and the assessments are specific challenges to prove students have learned.

- **Outcome** – Demonstrate effective technical reading and writing

  Assessment – Students present their outlines of this module's instructional chapters (e.g. What is SPICE?, Using SPICE, Primitive circuit examples, Gallery, etc.) ideally as an entry to a larger Journal document chronicling their learning. These outlines should exhibit good-faith effort at summarizing major concepts explained in the text.

  Assessment – Students show how quantitative results were obtained by SPICE in the given examples, performing some of those analyses manually to compare your results with the simulations'.

- **Outcome** – Design a circuit, analyze it, and write a corresponding SPICE netlist

  Assessment – Sketch a schematic diagram of a circuit and analyze all circuit parameters using the principles and equations you have learned so far.

  Assessment – Write and run a netlist for that circuit to compare your hand-calculated results against the computer's.

- **Outcome** – Independent research

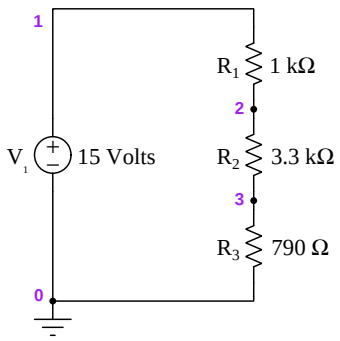
  Assessment – Identify some circuit simulators based on schematic capture rather than typed code. Some simulators are even web-based which means you may use them by navigating an internet browser application to the appropriate website (e.g. `EveryCircuit`).

Chapter 2

What is SPICE?

SPICE is a general-purpose computer simulator for electronic circuits, with several "freeware" versions available for academic use. Although its text-based user interface may seem clumsy and archaic at first, it is quite powerful, and also neatly side-steps the many problems students tend to experience with graphic-entry ("WYSIWYG")[1] circuit simulators.

Circuits are described to SPICE in the form of a *netlist*, which is a text-based code listing of each component within the circuit, the "nodes" connecting them to each other, and the types of analyses requested of SPICE to perform on that circuit. A simple three-resistor DC circuit is shown in schematic form below, along with the netlist you would write and input to SPICE instructing it to analyze this circuit for a single source voltage value of 15 Volts and displaying the voltage drop across each resistor as well as the current through the voltage source. Note the node numbers used to define the termination points of each component:




```
* SPICE netlist for a series DC circuit
v1 1 0 dc 15
r1 1 2 1000
r2 2 3 3300
r3 3 0 790
.dc v1 15 15 1
.print dc v(1,2) v(2,3) v(3) i(v1)
.end
```

SPICE netlists represent a specialized form of computer programming, with lines of text providing data and instructions for the computer to follow. As a programming language, SPICE is rather unique in that the order of the lines generally don't matter (other than the obligatory title line at the start and the "end" instruction at the very end). Rather than given step-by-step instructions like most programming languages, SPICE netlists describe a circuit's various components, how they interconnect, and the analyses desired for that circuit.

[1] "WYSIWYG" is an acronym meaning *What You See Is What You Get*.

Here is a line-by-line interpretation of the netlist describing the previous circuit:

```
* SPICE netlist for a series DC circuit This is the arbitrary "title" line
v1 1 0 dc 15 Defines a 15 Volt DC voltage source named "v1" between nodes 1 and 0
r1 1 2 1000 Defines a 1000 Ohm resistor between nodes 1 and 2
r2 2 3 3300 Defines a 3300 Ohm resistor between nodes 2 and 3
r3 3 0 790 Defines a 790 Ohm resistor between nodes 3 and 0
.dc v1 15 15 1 Requests a DC "sweep" analysis from 15 Volts to 15 Volts on source "v1"
.print dc v(1,2) v(2,3) v(3) Measures voltages between nodes 1-2, 2-3, and 3 to ground (0)
.end Tells SPICE to stop
```

Note the importance of the node numbers in the netlist: these node numbers are how SPICE knows where each component is in the circuit. Every electrically-common point in a circuit must bear the same node number; each electrically distinct point in the circuit requires a unique node number. Component types are designated by the first letter of each line (e.g. `v` for voltage source, `r` for resistor). All lines beginning with an asterisk (`*`) are comments, intended as notes for human readers of the netlist but ignored by SPICE. Lines beginning with a period (`.`) are special SPICE commands instructing it to do something with the component values given.

The purpose of this document is to give a general introduction to the use of SPICE, and also to showcase a "gallery" of examples where SPICE is used to perform analyses on circuits containing semiconductor components. This gallery will not only help new users of SPICE learn how to format netlists and interpret analyses, but they also serve as an extensive bank of practice problems for new students of electricity and electronics to use as they build and sharpen their circuit analysis skills.

This last point deserves some elaboration. The study of electricity and electronics, like all scientific studies, is aided by the solution of practice problems. Problem-solving is a skill like any other, and practice helps to build an arsenal of tools useful to any new problems you may encounter. Having answers provided for these problems allows students to check their work and to self-identify any misconceptions or bad problem-solving habits.

Traditional textbooks provide banks of practice problems for students to solve, and while these are useful, the availability of computer simulation software take this concept one enormous step further. Not only may the sample simulations given in this learning module serve as practice problems for students, but they will also serve as templates for creating one's own practice problems, so you will never be wanting for more practice opportunities. Like all computer simulations, SPICE rigorously applies known laws and principles of physics to predict how specific systems will behave, and as such it never makes mistakes[2].

SPICE has played a pivotal role in my own self-education on the topic of circuits, being an excellent tool to check my own analytical skills and to explore the properties of certain devices (e.g. transistors, transmission lines, transformers) with relative ease. The ultimate goal of any scientific education is to equip the student with mental tools to solve problems and to continue their own learning. SPICE is a software tool that can aid in the development of these mental tools, and should be used as such.

[2]Some simulations can and do miss certain details if they are not set up properly, but this is not the same kind of error as a human is prone to committing, and it is precisely this difference between computers and humans that makes computer simulations useful learning tools. Simply put, the computer will not repeat *your* mistakes.

Chapter 3

Using SPICE

SPICE is computer software designed to simulate the behavior of electric circuits. It was developed in the early days of digital computing when the dominant user interface was the *command prompt*, and the most common user file format was plain "ASCII" text. Graphic, mouse- or touchscreen-driven interfaces simply did not exist at that time except for prototype systems or highly specialized applications. Most general-purpose computing of that era occurred through the use of a keyboard and a monochrome text-based monitor, or worse yet in the form of punched paper cards and teletyped output.

This is why SPICE usage seems to resemble text-based code programming. The user (that means *you*) enters a description of the circuit to be analyzed and the analysis/display methods preferred by means of typed text, then the SPICE program is run and the output is viewed again in text form.

This may all seem terribly anachronistic in the 21st century, but there are some decided advantages to using SPICE with this legacy interface:

- These legacy versions of SPICE are free to use. No license fees, nothing.
- The relatively small (by modern standards) SPICE software runs very fast on modern computers.
- Circuits that are challenging to graphically draw are easy to "code"
- Errors resulting from the computer's incorrect interpretation[1] of a diagram are eliminated
- Learning to use SPICE in this manner is an excellent orientation to simple programming

The first step in setting up SPICE to analyze a circuit is to describe that circuit using SPICE's own language, and to type this description into a plain-text file on your computer. This file is often referred to as a *deck*, with each line of text in that file called a *card*. Taken together, those lines of text (a.k.a. *cards*) comprise a *netlist* describing the circuit and the intended analysis for SPICE. The

[1] I have struggled multiple times to get a WYSIWYG circuit analyzer program to simply analyze what I have drawn, due to "hidden" wire connections, or intended wire connections that don't exist – all "bugs" in the software stemming from the non-trivial challenge of getting a computer to recognize and correctly interpret a hand-made diagram. SPICE, with its "netlist" input and clear node-labeling protocol, neatly eliminates this whole problem.

anachronistic terms "card" and "deck" harken back to the days of punched-card input for computer systems, where each separate instruction for SPICE was a series of punched holes on its own paper card, the collection of cards representing one circuit being a "deck of cards" that would be fed into the card reader one at a time[2].

Once you have typed the netlist and saved it as a text file (a.k.a. *deck*) to your computer, you are ready to run a SPICE analysis on it. This is done by typing the word `spice` on a command-line interface[3] followed by a less-than symbol (<) and then followed by the name of the text file "deck", and finally terminated by pressing the Enter key. Here is an example, showing a screenshot of a SPICE analysis about to happen on a "deck" file named `test.cir` just prior to pressing the Enter key:

After entering this command, SPICE performs its analysis and by default dumps a report as plain text to the same command-line terminal. If you desire to save SPICE's text output to another plain-text file for posterity, or to be able to paste that analysis into another document, you may "redirect" SPICE to send its plain-text analysis to another file of your choosing by using the greater-than symbol (>) in the command. Here is an example showing how to instruct SPICE to read the contents of `test.cir` and place all the analysis text into another file named `test.out`:

```
spice < test.cir > test.out
```

[2]Interestingly, legacy versions of SPICE do not care what order these "cards" appear in the "deck" with the exception of the first line (the Title) and the last line (the `.end`) statement.

[3]Microsoft Windows operating systems provide a general-purpose command line interface terminal named `cmd`. Unix-based systems such as Apple's OS X and Linux typically offer a variety of terminal programs suitable for this purpose.

3.1 Summary of steps

To summarize, using SPICE in this legacy format requires the use of a *text editor* program (such as "Notepad" on Microsoft Windows operating systems) to write the netlist, and a command-line interface on your computer to invoke SPICE to perform its analysis. The sequence of steps may be listed:

1. Invoke a text editor program and type the netlist (instructions) describing the circuit and the type of analysis you wish SPICE to perform
2. Save this netlist as a plain-text file to your computer
3. Type `spice` < *deck_filename* on the command-line interface and then press Enter
4. View the analysis in the command-line interface

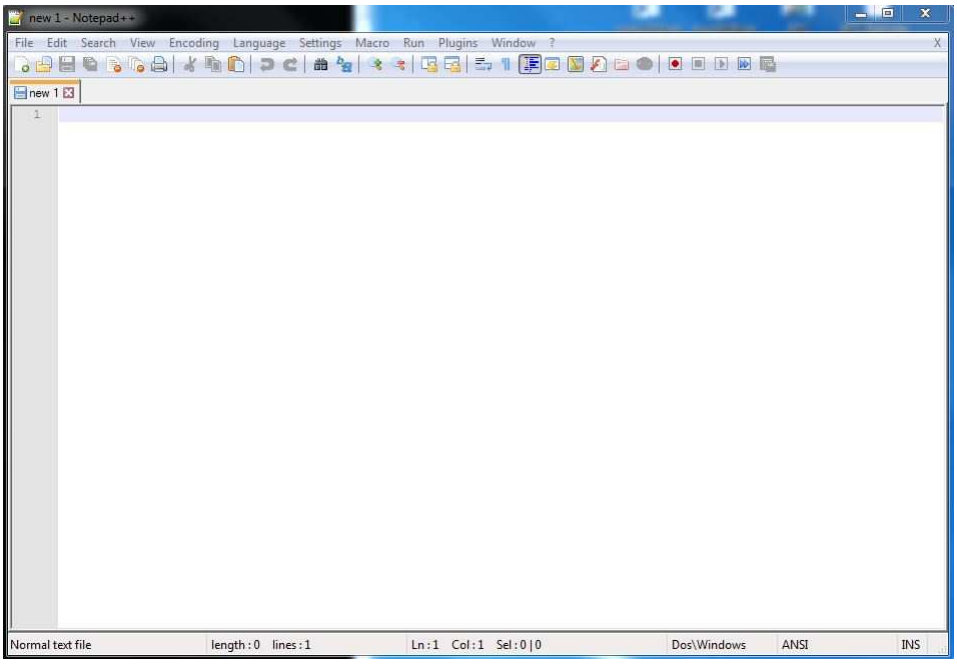
Most of SPICE's "learning curve" is in becoming familiar with the syntax of the netlist: exactly which letters, numbers, and sequences must be entered in order to properly describe the circuit and the type(s) of analysis desired. A few cardinal rules must be obeyed when creating or editing netlists:

- The first line (card) of the netlist is the Title. It cannot be omitted.
- The last line (card) of the netlist is the `.end` command. It cannot be omitted.
- Each and every unique connection point in the circuit must be assigned a number called a *node*. This is how you describe the "shape" of the circuit to SPICE: by describing which nodes each component connects to. Components sharing common node numbers are electrically common to each other; components with differing node numbers are electrically distinct from each other. There *must* be a node 0, and this is the "Ground" node of the circuit.

3.2 Demonstration on Microsoft Windows

Here I will demonstrate this four-step process as a series of screenshots from a computer running SPICE version 2G6 on the Microsoft Windows version 7 operating system.

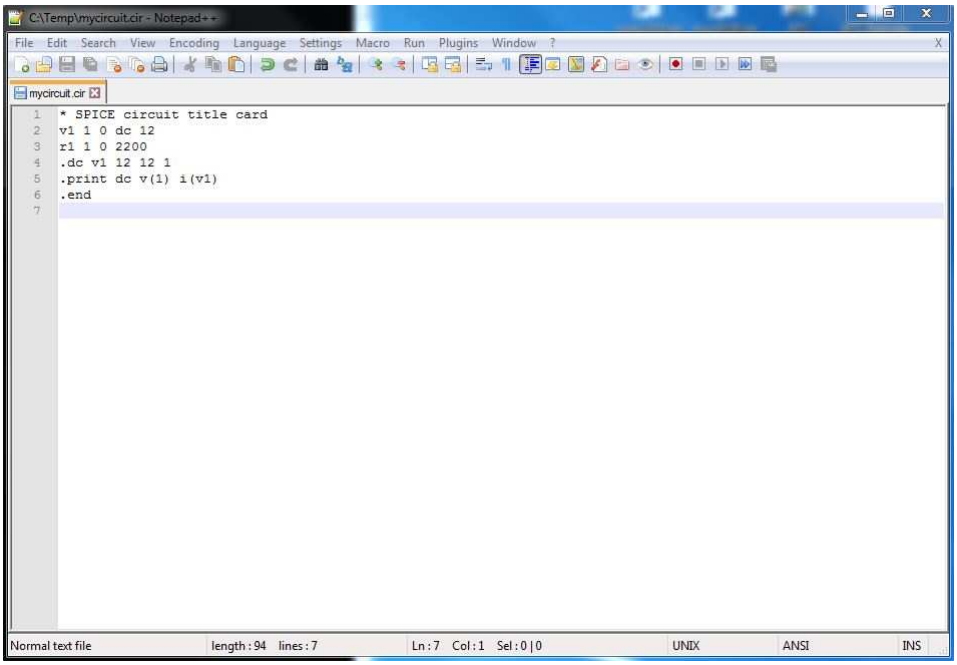
3.2.1 Invoking a text editor



Here, the screenshot shows an empty file, ready for me to type text. *Text editors* are similar to *word processors* except they lack provision for neatly formatting the text (e.g. no font selections, no paragraph formatting, etc.). Their sole purpose is to facilitate the creation and editing of plain-text files.

Text editors are the go-to application for practically all text-based computer programming. *Many* different text editors exist, each with their own features, and computer programmers tend to develop a fondness for one particular editor that best suits their programming needs. A "stock" text editor comes with every version of Microsoft Windows called `notepad`, but this editor is extremely limited in its capabilities, and I do not recommend it. The editor shown here (`notepad++`) is *vastly* superior and highly recommended.

3.2.2 Saving the deck



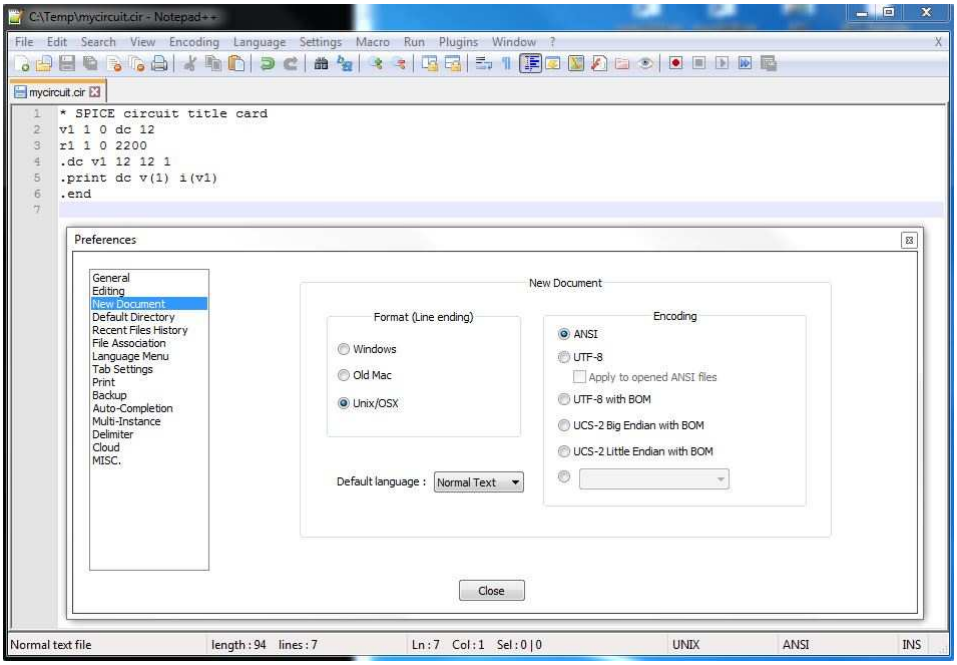
Here, I have chosen the filename `mycircuit.cir` for the deck. Note that you do not have to end every deck filename with the suffix "`.cir`" – this is simply my personal choice. Any filename consistent with the naming conventions of your computer's filesystem and containing no space characters is permissible. We avoid space characters because command-line interfaces typically interpret spaces as delimiting characters designed to separate commands and filenames from each other. If you must have a "space" in your filename, use an underscore character or a dash (_ or -). For example, `my-circuit` is acceptable, but `my circuit` is not.

Here is a line-by-line (a.k.a. *card-by-card*) interpretation of the netlist:

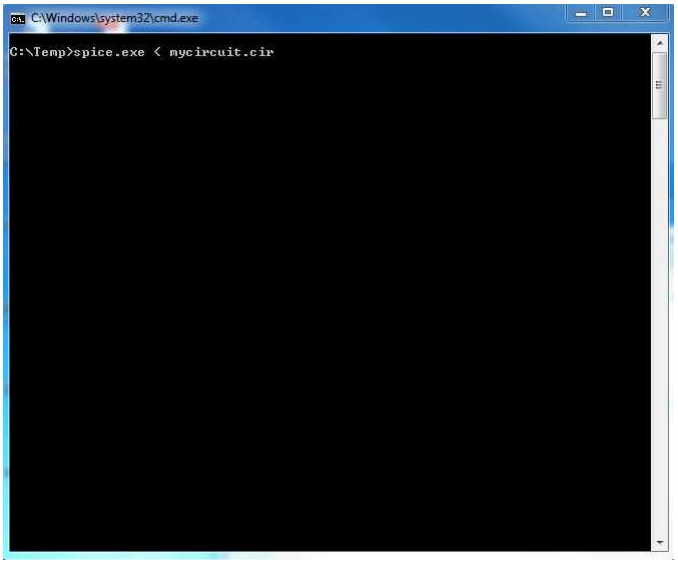
`* SPICE circuit title card` *This is the arbitrary "title" line*
`v1 1 0 dc 12` *Defines a 12 Volt DC voltage source named "v1" between nodes 1 and 0*
`r1 1 0 2200` *Defines a 2200 Ohm resistor between nodes 1 and 0*
`.dc v1 12 12 1` *Requests a DC "sweep" analysis from 12 Volts to 12 Volts on source "v1"*
`.print dc v(1) i(v1)` *Measures the voltage at node 1 and current through source "v1"*
`.end` *Tells SPICE to stop*

Note how components are "connected" to each other in the netlist by common node (connection-point) numbers. This is how SPICE is able to "picture" circuits without the use of graphical images. Each electrically distinct point in a circuit is given a unique number, and different components specified by the points they're connected to.

An idiosyncrasy of SPICE version 2G6 is that it must receive its "deck" file in Unix text format rather than Windows text format. Thankfully, `notepad++` provides a means to select which text format the deck file will be written in, as shown in this screenshot of the "Preferences" dialog window:



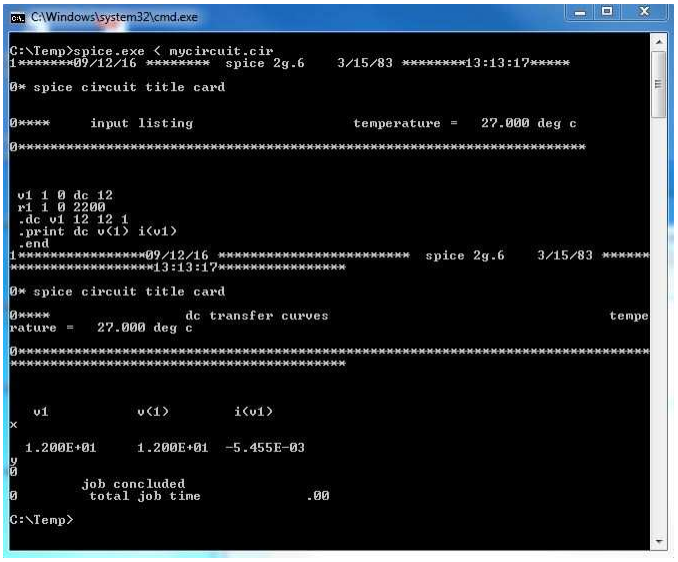
3.2.3 Invoking SPICE



This screenshot shows the command-line interface terminal (not the text editor), with the command ready to enter. This terminal may be access on any Windows operating system by invoking the `cmd` command.

Note that the exact command being entered here is `spice.exe < mycircuit.cir` because the SPICE executable file installed on this computer is named `spice.exe` rather than just `spice`.

3.2.4 Viewing the SPICE analysis



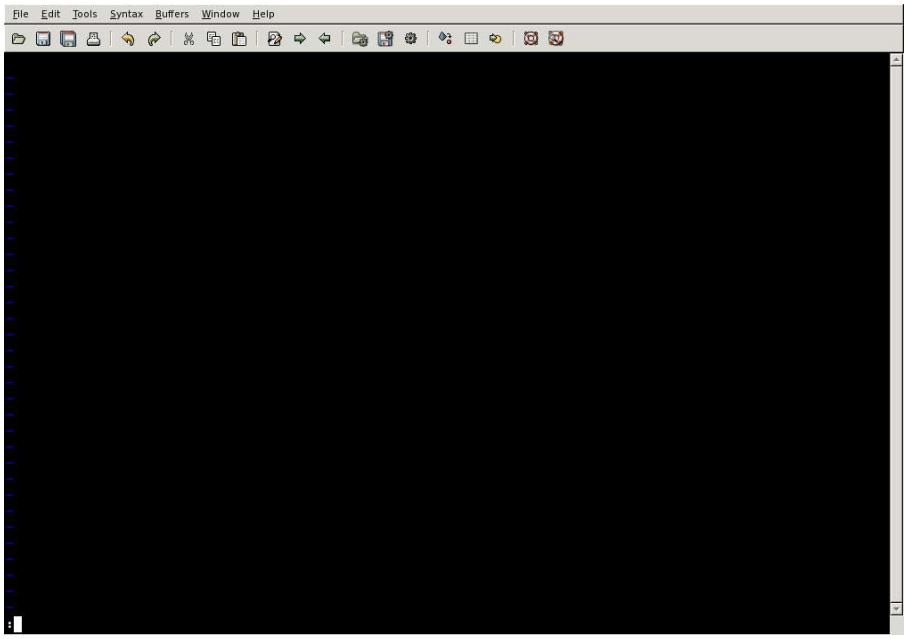
As you can see, SPICE is a very "chatty" program that displays a fair amount of extraneous text on the screen following an analysis. Ignoring all the headings and labels, you can see in this display a reiteration of the original netlist (deck) followed by the requested measurements of `v(1)` (i.e. voltage between node 1 and ground, which is node 0) and `i(v1)` (current through source `v1`).

This concludes a *very* brief orientation on using SPICE. For more information, I refer you to the following sections on SPICE component descriptions and analysis descriptions, as well as to the "Gallery" chapter which shows you tested SPICE netlists and output results so you may learn by example.

3.3 Demonstration on Linux or CygWin

Now, I will show you this same four-step process using SPICE version 2G6 running under the Linux operating system. The steps shown here are virtually identical within the "CygWin" emulator which adds a Unix-like command line environment to any Microsoft Windows operating system.

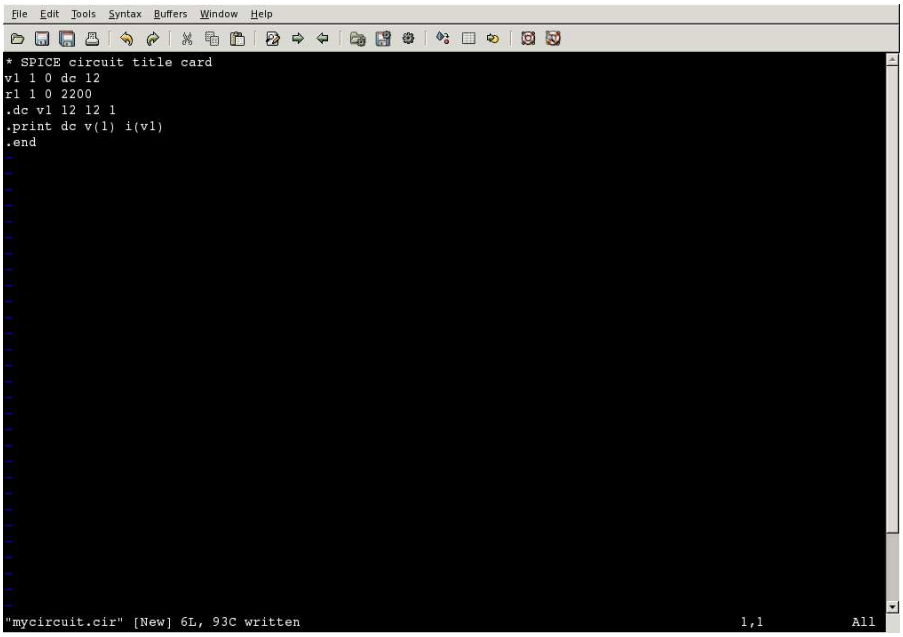
3.3.1 Invoking a text editor



Here, the screenshot shows an empty file, ready for me to type text. *Text editors* are similar to *word processors* except they lack provision for neatly formatting the text (e.g. no font selections, no paragraph formatting, etc.). Their sole purpose is to facilitate the creation and editing of plain-text files.

Text editors are the go-to application for practically all text-based computer programming. *Many* different text editors exist, each with their own features, and computer programmers tend to develop a fondness for one particular editor that best suits their programming needs. I personally prefer `gvim` (or `vim` in the CygWin emulator) and loathe Microsoft `notepad`, but text editors tend to be a highly personal choice.

3.3.2 Saving the deck



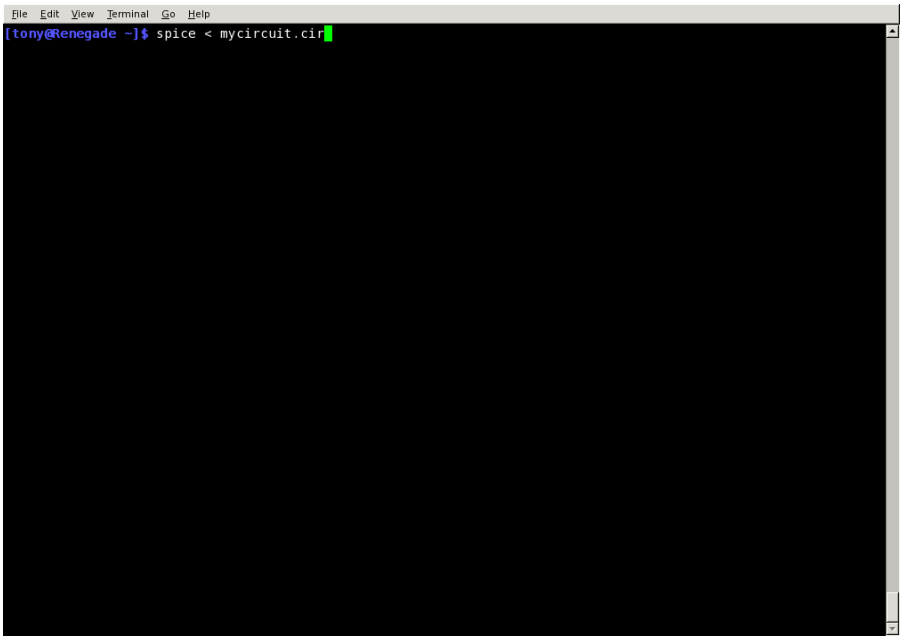
Here, I have chosen the filename `mycircuit.cir` for the deck. Note that you do not have to end every deck filename with the suffix "`.cir`" – this is simply my personal choice. Any filename consistent with the naming conventions of your computer's filesystem and containing no space characters is permissible. We avoid space characters because command-line interfaces typically interpret spaces as delimiting characters designed to separate commands and filenames from each other. If you must have a "space" in your filename, use an underscore character or a dash (_ or -). For example, `my-circuit` is acceptable, but `my circuit` is not.

Here is a line-by-line (a.k.a. *card*-by-*card*) interpretation of the netlist:

`* SPICE circuit title card` *This is the arbitrary "title" line*
`v1 1 0 dc 12` *Defines a 12 Volt DC voltage source named "v1" between nodes 1 and 0*
`r1 1 0 2200` *Defines a 2200 Ohm resistor between nodes 1 and 0*
`.dc v1 12 12 1` *Requests a DC "sweep" analysis from 12 Volts to 12 Volts on source "v1"*
`.print dc v(1) i(v1)` *Measures the voltage at node 1 and current through source "v1"*
`.end` *Tells SPICE to stop*

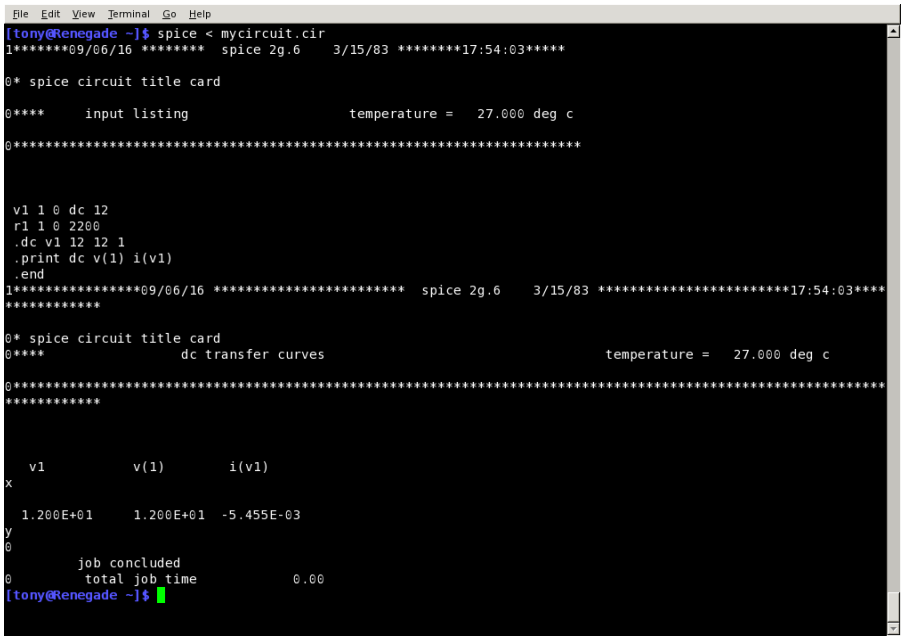
Note how components are "connected" to each other in the netlist by common node (connection-point) numbers. This is how SPICE is able to "picture" circuits with no graphics. Each and every electrically distinct point in a circuit is given a unique number, component placement being specified by these numbered connection points.

3.3.3 Invoking SPICE



This screenshot shows the command-line interface terminal (not the text editor), with the command ready to enter.

3.3.4 Viewing the SPICE analysis



As you can see, SPICE is a very "chatty" program that displays a fair amount of extraneous text on the screen following an analysis. Ignoring all the headings and labels, you can see in this display a reiteration of the original netlist (deck) followed by the requested measurements of `v(1)` (i.e. voltage between node 1 and ground, which is node 0) and `i(v1)` (current through source `v1`).

This concludes a *very* brief orientation on using SPICE. For more information, I refer you to the following sections on SPICE component descriptions and analysis descriptions, as well as to the "Gallery" chapter which shows you tested SPICE netlists and output results so you may learn by example.

3.4 Demonstration of NGSPICE interactive mode

A more modern (yet still free) version of SPICE is NGSPICE which provides all the old command-line functionality of legacy SPICE versions, plus a window-based *interactive* mode where you may call for certain analyses of a circuit without editing the netlist (deck file). This interactive display window contains a command-entry field as well as a display showing the results of those commands. The procedure for doing this, which I have tested on NGSPICE version 26 on a computer running the Microsoft Windows 7 operating system, is as follows:

1. Create the netlist using your favorite text editor and save it under a convenient filename in a convenient location. I will arbitrarily choose the filename `test.cir` for this demonstration.

2. Start the interactive version of NGSPICE and then enter the command `source test.cir` to instruct it to read the contents of that netlist file. Alternatively, instruct the Windows operating system to associate all `*.cir` files with NGSPICE by right-clicking on the file's icon and choosing the "Open with..." option, selecting `ngspice.exe` as the application you *always* wish to use for opening any `.cir` files. From then on, all netlist files ending in `.cir` will appear as icons bearing the NGSPICE logo, and double-clicking on it will start NGSPICE and automatically run the `source` instruction. *You should be able to skip this filename-association step for all future uses of NGSPICE.*

3. (Optional) – Type `listing` and press Enter to execute the "listing" command. NGSPICE will print the netlist in its display window, letting you verify that NGSPICE has loaded your intended netlist.

4. Enter the `run` command.

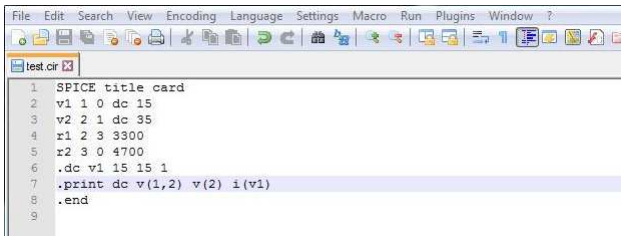
5. Enter various display commands such as `print` and `plot`. If you choose to plot results using the `plot` command, for example if you are testing a circuit's response over a range of time, or of source values swept over a specified interval, NGSPICE will open up an additional graphic window showing the results in color. When in doubt, `print all` and `plot all` are good display commands to try.

If you decide to alter the circuit after having run an analysis, simply close down NGSPICE, edit the netlist file using your text editor (re-saving that file), and then re-open NGSPICE by double-clicking on the netlist file icon again. Then, just issue the `run` command followed by any display commands you wish.

A very nice feature of this interactive mode is the ability to recall past commands without the need to re-type them. Simply press the "up" and down "arrow" keys to recall all past typed commands from history buffer, then press Enter to re-issue that command. This will save you much time and potential for keystroke errors!

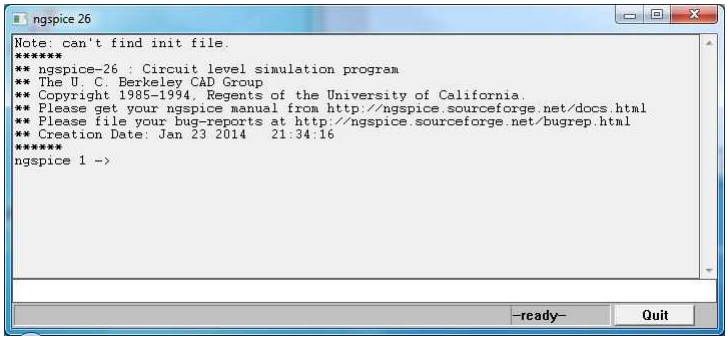
3.4.1 Creating the netlist

To begin our demonstration, we will first create a netlist using a text editor. In this case, the editor happens to be `notepad++` which is much more capable than the stock `notepad` that comes with Microsoft Windows. After writing the netlist, it will be saved under the filename `test.cir`:



3.4.2 Starting NGSPICE

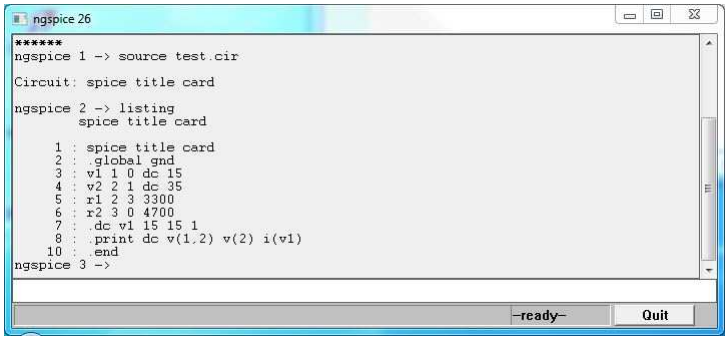
When NGSPICE is started with no command-line arguments (or simply started by double-clicking on the `ngspice.exe` filename), you will see this interactive window appear on your computer's screen. Commands are typed into the rectangular field at the bottom of the window, and after pressing Enter you will see the results of that command displayed in the upper portion of the window:



Please note that if you take the time to associate `.cir` files with NGSPICE in Microsoft Windows, this step (as well as the "Loading the netlist" step and "Verifying the loaded netlist" step) will be performed automatically for you whenever you double-click on the netlist file icon. You will know you have been successful in setting up this association if all files with filenames ending in `.cir` appear with the NGSPICE logo in their Windows icons.

3.4.3 Verifying the loaded netlist

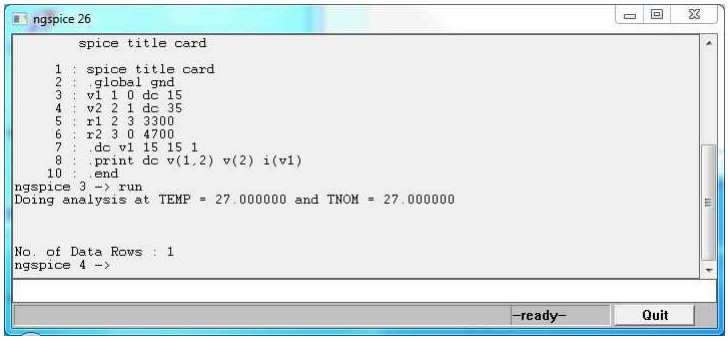
Now we will verify the successful loading of our netlist by issuing the `listing` command. Our netlist now appears in the display window:



This step is optional, but is a good habit. If you approach SPICE as a learning tool, using it to run simulated experiments on virtual circuits to help you master certain electrical concepts, you will find yourself making many changes to your netlists and re-running analyses on those revised circuits. The `listing` step gives you opportunity to verify each time that your edited netlist has been loaded into NGSPICE and is ready for analysis. It also shows, as in this case, if NGSPICE inserted any statements of its own (e.g. `.global gnd`).

3.4.4 Running the analysis

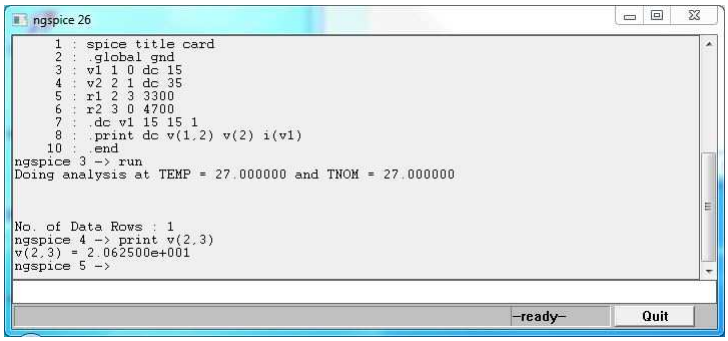
Before we may request any displayed results from NGSPICE, we must issue the `run` command:



An interesting feature of NGSPICE is that it ignores the `print` command within the netlist. Running the simulation merely performs the calculations, but provides no automatic output of results. In the "interactive" version of NGSPICE, it waits for you to enter specific commands before actually displaying any results to the screen. Even with `.print` or `.plot` instructions (cards) contained in the netlist, NGSPICE's interactive mode demands you type the display command yourself after it has "run" the simulation. To print the voltage between nodes 1 and 2, for example, you would enter the command `print v(1,2)`. NGSPICE relies on the analysis option in the netlist to discern whether this is a DC or AC analysis.

3.4.5 Printing a voltage

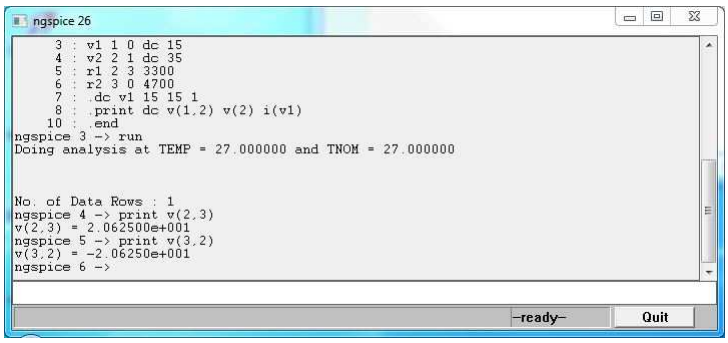
Now we are ready to make `print` or `plot` requests of NGSPICE. Here we request NGSPICE to print the voltage between nodes 2 and 3 with the `print v(2,3)` command:



Please note how NGSPICE does not require a preceding period symbol with the `print` command when it is typed at the interactive command line, nor does it require we specify `dc` analysis as part of the `print` command! In fact we could have omitted the entire `.print` card in the netlist, as NGSPICE's interactive mode only follows output instructions typed at the command line.

Note also how NGSPICE defaults to scientific notation, in this case the voltage displayed as `2.062500e+001` which means 2.062500×10^{1} Volts, or 20.62500 Volts.

If we issue the `print` command with the node numbers swapped, NGSPICE displays the DC voltage as though measuring with a DC voltmeter with the red test lead on node 3 and black test lead on node 2, showing a negative value instead of positive:



If we wish to see more than one voltage reported, we may specify this at the interactive command line in exactly the same manner as we would have specified within the netlist. For example, issuing the command `print v(1,2) v(2) i(v1)` would result in NGSPICE printing the voltage between nodes 1 and 2, followed by the voltage between nodes 2 and 0, followed by the current through voltage source v1. Alternatively, we can type `print all` in the NGSPICE command line and receive a report of all voltages (with reference to node 0 which is ground) rather than just the voltage(s) we specify.

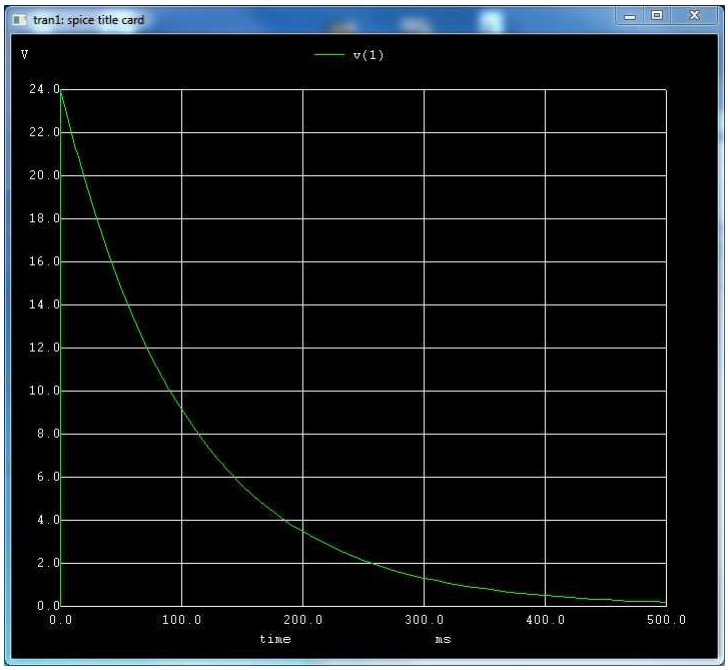
3.4.6 Plotting graphs using NGSPICE

Now we will explore the graphic plotting capabilities of NGSPICE. This is much more advanced than the text-based plots of legacy SPICE versions such as 2G6. Let's begin our example with the following netlist:

```
SPICE title card
c1 1 0 47e-6 ic=24
r1 1 0 2200
.tran 10m 500m uic
.plot tran v(1)
.end
```

This particular netlist defines a simple capacitor-resistor circuit with the 47 μF capacitor precharged to 24 Volts. The `.tran` card calls for a time-domain analysis from $t = 0$ to $t = 500$ milliseconds, in 10 millisecond steps, using the initial condition specified in the capacitor's card.

Once having loaded this netlist into NGSPICE and issuing the `run` command, we may type `plot v(1)` and get the following output in its own window. As with the previous (DC) analysis, there is no need to specify the analysis type in the `plot` instruction (i.e. `plot v(1)` rather than `plot tran v(1)`) because NGSPICE takes its analysis cue from the `.tran` card in the netlist. In fact, the `.plot tran v(1)` card in the netlist is entirely ignored (and could have been omitted!) when using NGSPICE in interactive mode:



As with the `print` command, we may issue a `plot all` instruction at the command line which

will prompt NGSPICE to plot *all* variables in the simulation. This may make for a cluttered result when done with complex circuits containing many nodes, but in simple cases such as this example it may be more convenient to simply enter `plot all` rather than specify each of the desired variables one by one.

3.4.7 Issuing multiple commands in a single line

A very convenient feature of NGSPICE's interactive command line is its support for multiple commands in a single typed line. This is useful if you wish to edit the netlist and re-run the simulation without re-starting NGSPICE. Instead of typing in `source test.cir` followed by `run` followed by any specific analysis commands such as `print` and/or `plot` commands, it is possible to type all these commands on a single line in the interactive window (separated by semicolons), after which the entire string of commands may be repeated by pressing the "up" arrow key followed by the "Enter" key.

Consider for example the following sequence of commands, typically entered one command at a time in the NGSPICE interactive environment to read, execute, and finally plot data from a netlist file named `test.cir`:

```
source test.cir

run

plot v(2) v(4,3)
```

These three commands may alternatively be entered at once on a single line, then recalled with a single press of the "up" arrow key, thus conserving many keystrokes and shortening the test/development cycle:

```
source test.cir ; run ; plot v(2) v(4,3)
```

3.5 Idiosyncrasies of SPICE

SPICE, particularly early versions such as 2G6, is intolerant of certain omissions and circuit topologies. Some of them are listed here.

3.5.1 Beginning and ending cards

Every SPICE deck must begin with a "title" line of plain text. I typically insert a comment (a line beginning with a `*` character) in the first line, but a comment is not strictly necessary. The deck cannot begin with a component or analysis description.

Similarly, every SPICE deck must finish with an `.end` card to instruct the program to stop its analysis. There should be no lines past the `.end` card, not even blank lines! Similarly, the `.end` line should not be terminated with either a linefeed nor a carriage return character; i.e. do *not* press the Enter key when typing in this last line! SPICE version 2G6 will still process the deck properly with a terminated `.end` card, but it generates an annoying *missing .end card* statement anyway.

3.5.2 Node zero

Every SPICE deck must contain reference to node zero (0). Node zero is not just the default "ground" reference point in a circuit, it is a *necessary* point of reference in order for SPICE to perform its mathematical analyses of the circuit. Furthermore, every other node in the circuit must have some DC path to node 0 (ground) or else SPICE will refuse to analyze the circuit, even if the specified analysis is not DC.

3.5.3 Current measurement

Early versions of SPICE lacked the ability to print or plot values for current through any component but a voltage source. Therefore, if you need to display the calculated current at any point in a circuit being analyzed by a legacy version of SPICE, you must insert special "dummy" voltage sources (set to zero volts each) in the circuit, and then instruct the `.print` or `.plot` analysis statements to display the current through those "dummy" sources.

Alternatively, you may insert low-valued "shunt" resistors in series with the point of interest through which you wish to measure current, and then have SPICE measure the voltage(s) across the resistor(s), just the same as you might do for any real circuit where you lack an ammeter suitable for measuring current. So long as these shunt resistor values are considerably less than the other resistances through which you need to measure current, their impact on the circuit will be negligible. Shunt resistor values of 1 Ω are common for many practical current-measurement applications in the milliAmpere range. For greater currents, shunt resistances of 1 mΩ or even 1 $\mu\Omega$ may be used.

Interestingly, the mathematical sign of the calculated current is negative when the voltage source in question is actually functioning as a source (i.e. conventional current flow out the + terminal and in the − terminal). In order to display a positive current value, current must be entering the voltage source's + terminal and exiting the − terminal, like a load.

3.5.4 Open and short circuits

The fundamental problem with open or shorted circuits is that either condition creates an *undefined* mathematical quantity that SPICE cannot compute. Opens create a condition of infinite (undefined) resistance. Shorts create a condition of zero resistance, which leads to infinite (undefined) current.

Since the insertion of a "short" or an "open" condition may be very useful for simulating a failed component, a common strategy in SPICE modeling is to use a resistor having a very low resistance (short) or a very high resistance (open). For example, a resistance with a value of `1e-9` Ohms is a short for all practical purposes. Similarly, a resistance with a value of `1e9` Ohms is tantamount to an open for most applications.

3.5.5 Multiple sources

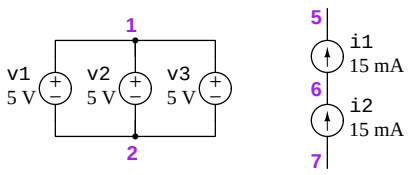
Multiple voltage and/or current sources are permitted in SPICE, but only in certain configurations. Connecting multiple voltage sources in parallel creates a *voltage source loop* error which will cause the analysis to abort, and connecting multiple current sources in series creates a *no dc path to ground* error that is similarly fatal, regardless of source values.

The problem may be understood by considering the internal impedance of each source, and how that impedance "appears" from the perspective of any other source(s). Ideal voltage sources have zero internal impedance, which is why SPICE balks at parallel-connected voltage sources: from the perspective of one voltage source, any other paralleled voltage source appears to be a direct short-circuit. Ideal current sources have infinite internal impedance, which is why SPICE balks at series-connected current sources: from the perspective of one current source, any others connected in series appear to create an open-circuit condition.

It is permissible, however, to connect multiple voltage sources in series with each other. Likewise, multiple current sources connected in parallel are also permissible.

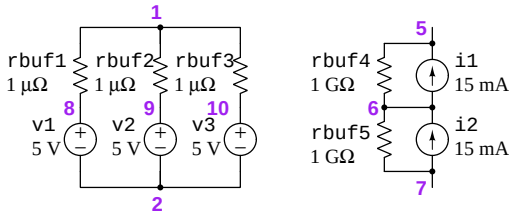
If paralleled voltage sources and/or series-connected current sources are necessary for your circuit design, you may "break up" the offending configuration by inserting *buffer components*. For example, inserting resistors of extremely low value in series with each voltage source before paralleling them will allow the analysis to proceed. A resistor of extremely high value inserted in parallel with a current source will provide the dc path around the other current source that SPICE is looking for, similarly allowing the analysis to proceed.

Impermissible source connections




```
v1 1 2 dc 5
v2 1 2 dc 5
v3 1 2 dc 5
*
i1 6 5 dc 15e-3
i2 7 6 dc 15e-3
```

Permissible source connections




```
v1 8 2 dc 5
v2 9 2 dc 5
v3 10 2 dc 5
rbuf1 1 8 1e-6
rbuf2 1 9 1e-6
rbuf3 1 10 1e-6
*
i1 6 5 dc 15e-3
i2 7 6 dc 15e-3
rbuf4 5 6 1e9
rbuf5 6 7 1e9
```

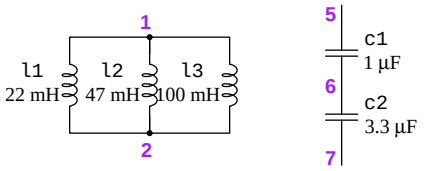
3.5.6 Multiple inductors/capacitors

Similar problems arise when connecting multiple inductors directly in parallel with each other, or multiple capacitors directly in series with each other. SPICE views an inductor as being a short (zero resistance) under DC conditions, and a capacitor as being an open (infinite resistance) under DC conditions. As with multiple sources, the problem may be understood by considering the DC resistance of each component as seen from the other components. Parallel-connected inductors are problematic because any attempt to analyze the behavior of one inductor is thwarted by the apparent short-circuit formed by the other inductor(s); series-connected capacitors are problematic because any attempt to analyze the behavior of one capacitor is thwarted by the apparent open-circuit formed by the other capacitor(s).

Likewise, SPICE cannot tolerate any voltage source being connected in parallel with any inductor, nor any current source being connected in series with any capacitor.

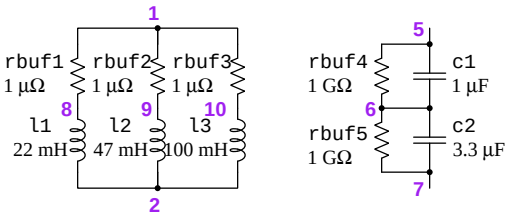
If any of these offending configurations are necessary for your circuit design, you may skirt the problem by inserting buffer components as previously described:

Impermissible inductor/capacitor connections




```
l1 1 2 22e-3
l2 1 2 47e-3
l3 1 2 100e-3
*
c1 5 6 1e-6
c2 6 7 3.3e-6
```

Permissible inductor/capacitor connections




```
l1 8 2 22e-3
l2 9 2 47e-3
l3 10 2 100e-3
rbuf1 1 8 1e-6
rbuf2 1 9 1e-6
rbuf3 1 10 1e-6
*
c1 5 6 1e-6
c2 6 7 3.3e-6
rbuf4 5 6 1e9
rbuf5 6 7 1e9
```

Chapter 4

SPICE component descriptions

To review, here are some of the "cardinal rules" for writing SPICE netlists:

- The first line (card) of the netlist is the Title. It cannot be omitted.
- The last line (card) of the netlist is the `.end` command. It cannot be omitted.
- Each and every unique connection point in the circuit must be assigned a number called a *node*. This is how you describe the "shape" of the circuit to SPICE: by describing which nodes each component connects to. Components sharing common node numbers are connected to each other. There *must* be a node 0, and this is the "Ground" node of the circuit.
- Comment lines (cards) must begin with an asterisk symbol (`*`). These are lines of text inserted into the netlist strictly for the benefit of human readers. SPICE skips over them.

Note that SPICE is case-insensitive. Any instances of capitalization in the following SPICE examples are included solely for clarity and readability, and do not matter when entered into a SPICE netlist.

4.1 Independent voltage sources

General format for DC and AC analyses:

[**V***name*] [+node_ID] [−node_ID] [DC/AC *voltage*] [AC *voltage phaseangle*]

The first node specified is always the positive (+) terminal of the voltage source, while the second node is always the negative (−) terminal.

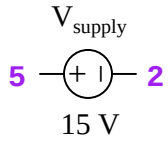
4.1.1 Example: DC source

- Name = *supply*
- Polarity = *+ on node 5 and − on node 2*
- Value = *15 Volts DC*

SPICE element description

```
Vsupply 5 2 dc 15
```

Schematic representation



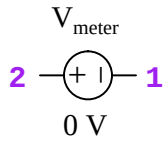
4.1.2 Example: "Dummy" source

- Name = *meter*
- Polarity = *+ on node 2 and − on node 1*
- Value = *0 Volts*

SPICE element description

```
Vmeter 2 1 0
```

Schematic representation



"Dummy" voltage sources are useful in SPICE netlists, to serve as points of measurement for current. Legacy versions of SPICE could not display a calculated current value for any element other than a voltage source, so a "dummy" voltage source with a value of 0 Volts served the purpose quite well.

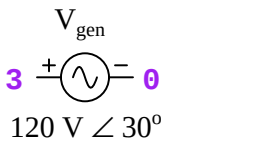
4.1.3 Example: AC source

- Name = *gen*
- Polarity = *+ on node 3 and − on node 0*
- Value = *120 Volts ∠ 30* degrees

SPICE element description

```
Vgen 3 0 ac 120 30
```

Schematic representation



AC sources specified in this manner are assumed to be sinusoidal in waveform with a constant magnitude and phase shift, and the analysis performed by SPICE will be in the frequency domain.

In addition to steady-state DC and AC sources, SPICE also supports multiple types of time-dependent sources, useful for simulating waveforms and pulse trains in the time domain.

General format for time-domain analyses:

[**V***name*] [+node_ID] [−node_ID] [*waveform_type*] ([*option1 option2*] [*option3* $\cdots$])

The first node specified is always the positive (+) terminal of the voltage source, while the second node is always the negative (−) terminal. Five different waveform types are supported, each with their own options. The waveform types are Sinusoidal (`sin`), Pulse (`pulse`), Exponential (`exp`), Piecewise Linear (`pwl`), and Frequency-Modulated Sinusoidal (`sffm`).

No provision for phase angle exists for AC time-domain sources. Instead, one must make creative use of the start delay time option. A positive start delay time means that the waveform's start becomes delayed, representing a negative phase shift angle. Conversely, a negative start delay time means the waveform has an "early" start, or a positive phase shift angle. Unfortunately, SPICE version 2G6 does not support negative start delay time values.

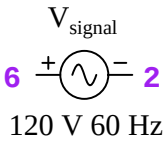
4.1.4 Example: Sinusoidal source

- Name = *signal*
- Polarity = *+ on node 6 and − on node 2*
- Wave type = *sinusoidal*
- DC offset = *10 Volts*
- Peak amplitude = *120 Volts*
- Frequency = *60 Hz*
- Start delay = *8 milliseconds*
- Damping factor = *0 seconds*$^{-1}$ (i.e. an undamped waveform)

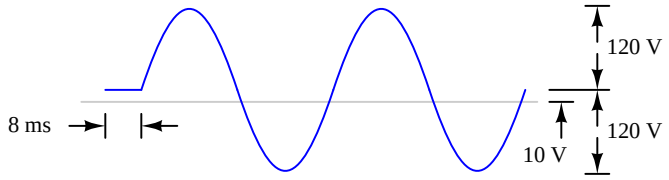
SPICE element description

```
Vsignal 6 2 sin(10 120 60 8m 0)
```

Schematic representation



Waveform



As opposed to a simple AC voltage sources which is assumed to be sinusoidal in waveform and constant in magnitude (i.e. no growth or decay over time), this source is explicitly sinusoidal, and the analysis performed by SPICE will be in the time domain.

Note that the damping factor assumes a decaying signal with a positive value. In order to generate an exponentially growing waveform you will need to use a *negative* value for the damping factor. In other words, the damping factor is equal to $-\sigma$.

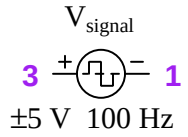
4.1.5 Example: Pulse source

- Name = *signal*
- Polarity = *+ on node 3 and − on node 1*
- Wave type = *pulse*
- Initial value = −5 *Volts*
- Pulsed value = 5 *Volts*
- Start delay = *3 milliseconds*
- Rise time = *1 millisecond*
- Fall time = *2 millisecond*
- Pulse width = *5 milliseconds*
- Period = *10 milliseconds*

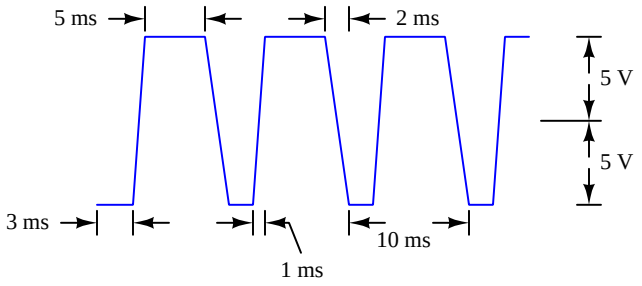
SPICE element description

```
Vsignal 3 1 pulse (-5 5 3m 1m 2m 5m 10m)
```

Schematic representation



Waveform



Like other time-dependent sources, a `pulse` source requires analysis in the time domain using the `.tran` analysis option.

4.2 Independent current sources

General format for DC and AC analyses:

[**I***name*] [sink_node_ID] [source_node_ID] [DC/AC *current*] [AC *current phaseangle*]

The first node specified is always the terminal where current (conventional flow) enters the current source, while the second node is always the terminal where current exits the source.

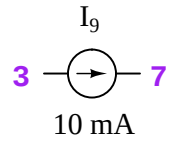
4.2.1 Example: DC source

- Name = *9*
- Polarity = *current enters node 3 and exits node 7*
- Value = *10 milliAmperes*

SPICE element description

```
I9 3 7 dc 10e-3
```

Schematic representation



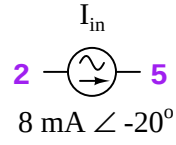
4.2.2 Example: AC source

- Name = *in*
- Polarity = *current enters node 2 and exits node 5*
- Value = *8 milliAmperes ∠ -20* degrees

SPICE element description

```
Iin 2 5 ac 8m -20
```

Schematic representation



4.3 Resistors

General format:

[**R***name*] [node_ID] [node_ID] [*value*]

As with all SPICE elements, resistor values may be specified using plain decimal notation, power-of-ten notation, or metric-prefix notation as alternatively shown in the example.

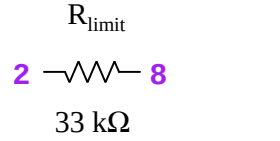
4.3.1 Example

- Name = *limit*
- Nodes = *connected between nodes 2 and 8*
- Value = *33 kilo-Ohms* Ω

SPICE element description

```
Rlimit 2 8 33000
(or)
Rlimit 2 8 33e3
(or)
Rlimit 2 8 33k
```

Schematic representation



4.4 Capacitors

General format:

[**C***name*] [+node_ID] [−node_ID] [*value*] [**IC=** *initial_voltage*]

An important caveat when using capacitors in a SPICE netlist is that SPICE cannot tolerate multiple capacitors connected directly in series. If your circuit design requires this configuration, you will need to insert a "buffer" component (e.g. a resistor with an extremely high resistance value) in parallel with one or more of the capacitors so that the offending components are no longer directly in series with each other.

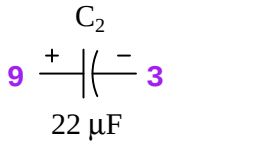
4.4.1 Example: Capacitor with initial charge

- Name = *2*
- Polarity = *+ on node 9 and − on node 3, initially charged to 3.5 Volts*
- Value = *22 micro-Farads*

SPICE element description

```
C2 9 3 22e-6 ic=3.5
```

Schematic representation



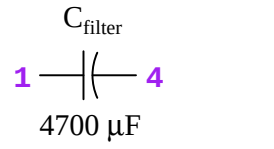
4.4.2 Example: Uncharged capacitor

- Name = *filter*
- Nodes = *connected between nodes 1 and 4, no initial voltage*
- Value = *4700 micro-Farads*

SPICE element description

```
Cfilter 1 4 4700e-6
```

Schematic representation



4.5 Inductors

General format:

[**L***name*] [sink_node_ID] [source_node_ID] [*value*] [**IC=** *initial_current*]

An important caveat when using inductors in a SPICE netlist is that SPICE cannot tolerate multiple inductors connected directly in parallel, nor can it abide any voltage source connected directly in parallel with an inductor. If your circuit design requires either of these configurations, you will need to insert a "buffer" component (e.g. a resistor with a negligible resistance value) between the inductors or inductor/voltage source so that the offending components are no longer directly in parallel with each other.

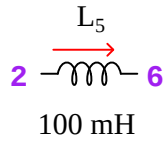
4.5.1 Example: Inductor with initial charge

- Name = *5*
- Polarity = *current enters node 2 and exits node 6, initially carrying 25 milliAmperes*
- Value = *100 milliHenrys*

SPICE element description

```
L5 2 6 100e-3 ic=25e-3
```

Schematic representation



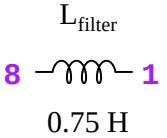
4.5.2 Example: Uncharged inductor

- Name = *filter*
- Nodes = *connected between nodes 8 and 1, no initial current*
- Value = *0.75 Henrys*

SPICE element description

```
Lfilter 8 1 0.75
```

Schematic representation



4.6 Transformers

Transformers do not exist as independent entities in SPICE. Instead, one must specify the windings of a transformer as separate inductors, then specify a *coupling* factor k magnetically linking those inductors. The relationship between mutual inductance (M), k, and the two inductor values (L_1 and L_2) are given by the formula $M = k\sqrt{L_1 L_2}$.

General format:

[**K***name*] [**L***name_1*] [**L***name_2*] [*value*]

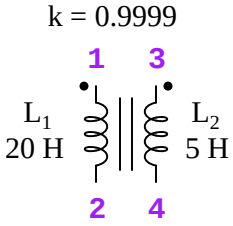
4.6.1 Example: 2:1 ratio step-down transformer

- Name = *xfmr1*
- Nodes = *primary inductor L1 connected between nodes 1 and 2, secondary inductor L2 connected between nodes 3 and 4*
- Value = *0.9999*

SPICE element description

```
L1 1 2 20
L2 3 4 5
Kxfmr1 L1 L2 0.9999
```

Schematic representation



Note how the ratio of primary to secondary inductance is equal to the square of the turns ratio, because inductance is proportional to the square of the winding turns. Thus, in order to achieve a 2:1 turns ratio, we need a 4:1 inductance ratio.

4.7 Transmission lines

Transmission lines are four-terminal devices, much like dependent sources. Rather than specifying a physical length for the transmission line, we may *either* specify delay time (i.e. the time it takes for the signal to propagate along the entire length of the line) *or* the signal frequency and corresponding electrical length of the line (in units of wavelengths). These two methods of specifying line length are exclusive to each other. If using the frequency option, the electrical line length defaults to 0.25 ($\frac{\lambda}{4}$) unless specified otherwise.

General format:

[**T***name*] [node_ID] [node_ID] [node_ID] [node_ID] [**Z0=** *characteristic impedance*] [**TD=** *delay time*] [**F=** *frequency* and **NL=** *electrical length*]

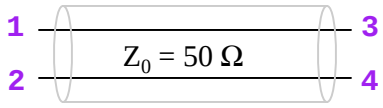
4.7.1 Example: 50-Ohm transmission line with 10 nanosecond delay

- Name = *cable*
- Nodes = *one end of cable has nodes 1 and 2, other end of cable has nodes 3 and 4*
- Delay = *10e-9 seconds*

SPICE element description

```
Tcable 1 2 3 4 z0=50 td=10e-9
```

Schematic representation



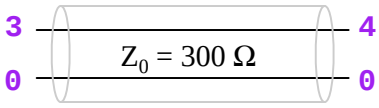
4.7.2 Example: half-wavelength (at 35 MHz) 300-Ohm transmission line

- Name = *feedline*
- Nodes = *one end of cable has nodes 3 and 0, other end of cable has nodes 4 and 0*
- Frequency = *35 MHz*
- Electrical length = *0.5 wavelengths*

SPICE element description

```
Tcable 3 0 4 0 z0=300 f=35e6 nl=0.5
```

Schematic representation



4.8 Linear dependent sources

While independent sources output a prescribed voltage or current, dependent sources output a voltage or current as a function of some other voltage or current signal in the circuit. While there is no real-world equivalent component for a dependent source, dependent sources are useful for modeling a variety of phenomena and are often used as portions of a SPICE model for some real-world component.

SPICE offers dependent sources in four different types:

- Voltage-controlled voltage source (`E`)
- Voltage-controlled current source (`G`)
- Current-controlled voltage source (`H`)
- Current-controlled current source (`F`)

Voltage-controlled sources (types `E` and `G`) have four nodes specified in their SPICE card: two nodes for the input terminals of the controlling voltage, and two nodes for the output terminals. Current-controlled sources (types `H` and `F`) only specify the two output terminal nodes, the controlling current being specified by the name of some independent voltage source used as a current sensor.

General formats:

[**E***name*] [+output_node_ID] [−output_node_ID] [+input_node_ID] [−input_node_ID] [*gain*]

[**G***name*] [sink_node_ID] [source_node_ID] [+input_node_ID] [−input_node_ID] [*gain*]

[**H***name*] [+output_node_ID] [−output_node_ID] [**V***name*] [*gain*]

[**F***name*] [sink_node_ID] [source_node_ID] [**V***name*] [*gain*]

In each case, the *gain* value is the ratio of output to input, regardless of units (e.g. Volts output per Volt input for a type `E` source; Amperes output per Volt input for a type `G` source, etc.).

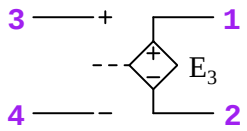
4.8.1 Example: voltage-controlled voltage source

- Name = *3*
- Polarity = *input on nodes 3 (+) and 4 (−) ; output on nodes 1 (+) and 2 (−)*
- Gain = *0.5 Volts output per Volt input*

Spice element description

```
E3 1 2 3 4 0.5
```

Schematic representation



$V_{out} = (0.5)(V_{in})$

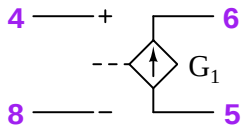
4.8.2 Example: voltage-controlled current source

- Name = *1*
- Polarity = *input on nodes 4 (+) and 8 (−) ; output current enters node 5 and exits node 6*
- Gain = *2.3 Amperes output per Volt input*

Spice element description

```
G1 5 6 4 8 2.3
```

Schematic representation



$I_{out} = (2.3)(V_{in})$

4.9 Nonlinear dependent sources

While independent sources output a prescribed voltage or current, dependent sources output a voltage or current as a function of some other voltage or current signal in the circuit. A *nonlinear* source is able to implement a mathematical function relating one or more inputs to its output. While there is no real-world equivalent component for a dependent source, dependent sources are useful for modeling a variety of phenomena and are often used as portions of a SPICE model for some real-world component.

Note: these sources are not available in legacy versions of SPICE, but are supported in NGSPICE.

General format:

[**B***name*] [+output_node_ID] [−output_node_ID] [i= *expression*] [v= *expression*]

Supported mathematical functions include the arithmetic basic operations of addition (`+`), subtraction (`-`), multiplication (`*`), division (`/`), and powers (`^`) as well as many other functions. Exponential (`exp`) and logarithmic (both `ln` and `log`), trigonometric , hyperbolic, and other functions are supported too.

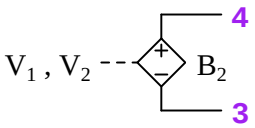
4.9.1 Example: multiplier

- Name = *2*
- Polarity = *output on nodes 4 (+) and 3 (−)*
- Function = *voltage v(1) times voltage v(2)*

Spice element description

```
B2 4 3 v=(v(1)*v(2))
```

Schematic representation



$V_{out} = (V_1)(V_2)$

4.10 Diodes

Semiconductor diodes, like other semiconductor components in SPICE, must be accompanied by a `.model` card somewhere in the netlist specifying the model name and parameters.

General format:

[**D***name*] [anode_node_ID] [cathode_node_ID] [*model_name*] [**IC=** *initial_voltage*]
[**.MODEL**] [*model_name*] [D] [**Parameter1=** *value*] [**Parameter2=** *value*] [$\cdots$]

The following is a list of parameters, any of which may be included in the `.model` card:

- IS = Reverse saturation current (default = 1×10^{-14} Amperes)
- N = Emission coefficient (default = 1)
- RS = Ohmic resistance (default = 0 Ohms)
- TT = Transit time (default = 0 seconds)
- CJO = Zero-bias junction capacitance (default = 0 Farads)
- VJ = Junction potential (default = 1 Volt)
- M = Grading coefficient (default = 0.5)
- EG = Activation energy (default = 1.11 electron-Volts, representing Silicon)
- BV = Breakdown voltage (default = infinite Volts)
- IBV = Current at breakdown voltage (default = 1×10^{-3} Amperes)

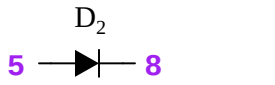
4.10.1 Example: Generic diode

- Name = *2*
- Polarity = *anode on node 5 and cathode on node 8*
- Model = *acme*, generic diode

SPICE element description

```
D2 5 8 acme
.MODEL acme d
```

Schematic representation



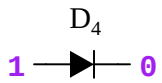
4.10.2 Example: 1N4001

- Name = *4*
- Polarity = *anode on node 1 and cathode on node 0*
- Model = *1N4001* rectifying diode

SPICE element description

```
D4 1 0 1N4001
.MODEL 1N4001 d
+ IS=1.2e-8 RS=0.04
+ N=1.83 EG=0.6
+ BV=50 IBV=5e-8
+ CJO=1e-11 VJ=0.65
+ M=0.5 TT=1e-9
```

Schematic representation



4.11 Bipolar Junction Transistors (BJTs)

BJTs, like other semiconductor components in SPICE, must be accompanied by a `.model` card somewhere in the netlist specifying the model name and parameters. This is where the type of BJT (NPN or PNP) is specified.

General format:

[**Q***name*] [collector_node_ID] [base_node_ID] [emitter_node_ID] [*model_name*]
[**.MODEL**] [*model_name*] [NPN *or* PNP] [**Parameter1=** *value*] [**Parameter2=** *value*] [$\cdots$]

The following is a list of parameters, any of which may be included in the `.model` card:

- IS = Reverse saturation current (default = 1×10^{-14} Amperes)
- BF = Forward current gain (default = 100)
- BR = Reverse current gain (default = 1)
- NF = Forward emission coefficient (default = 1)
- NR = Reverse emission coefficient (default = 1)
- VAF = Forward Early voltage (default = infinite Volts)
- VAR = Reverse Early voltage (default = infinite Volts)
- RC = Collector ohmic resistance (default = 0 Ohms)
- RB = Base ohmic resistance (default = 0 Ohms)
- RE = Emitter ohmic resistance (default = 0 Ohms)
- TF = Forward transit time (default = 0 seconds)
- TR = Reverse transit time (default = 0 seconds)
- CJE = Zero-bias B-E junction capacitance (default = 0 Farads)
- CJC = Zero-bias B-C junction capacitance (default = 0 Farads)
- VJE = B-E junction potential (default = 0.75 Volt)
- VJC = B-C junction potential (default = 0.75 Volt)
- MJE = B-E grading coefficient (default = 0.33)
- MJC = B-C grading coefficient (default = 0.33)

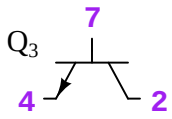
4.11.1 Example: Generic NPN transistor

- Name = *3*
- Terminals = *collector on node 2 and base on node 7 and emitter on node 4*
- Model = *mynpn*, generic NPN

SPICE element description

```
Q3 2 7 4 mynpn
.MODEL mynpn npn
```

Schematic representation



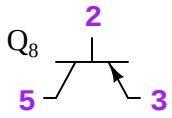
4.11.2 Example: 2N2907

- Name = *8*
- Terminals = *collector on node 5 and base on node 2 and emitter on node 3*
- Model = *2N2907* small-signal PNP

SPICE element description

```
Q8 5 2 3 2N2907
.MODEL 2N2907 pnp
+ IS=3e-12 BF=395
+ NF=1.15 VAF=42
+ VJE=0.69 VJC=0.41
```

Schematic representation



4.12 Junction Field-Effect Transistors (JFETs)

JFETs, like other semiconductor components in SPICE, must be accompanied by a `.model` card somewhere in the netlist specifying the model name and parameters. The channel type is specified in the `.model` card: NJF for N-channel and PJF for P-channel.

General format:

[**J***name*] [drain_node_ID] [gate_node_ID] [source_node_ID] [*model_name*]
[**.MODEL**] [*model_name*] [NJF *or* PJF] [**Parameter1=** *value*] [**Parameter2=** *value*] [$\cdots$]

The following is a list of parameters, any of which may be included in the `.model` card:

- VTO = Threshold pinch-off voltage (default = -2.0 Volts)
- BETA = Transconductance (default = 10^{-4} A/V^2)
- LAMBDA = Channel length modulation (default = 0 V^{-1})
- RD = Drain ohmic resistance (default = 0 Ohms)
- RS = Source ohmic resistance (default = 0 Ohms)
- CGS = Zero-bias G-S junction capacitance (default = 0 Farads)
- CGD = Zero-bias G-D junction capacitance (default = 0 Farads)
- IS = Gate junction saturation current (default = 1×10^{-14} Amperes)
- PB = Gate junction potential (default = 1.0 Volt)

The mathematical sign of the threshold pinch-off voltage (`VTO`) is important. Since JFETs are normally-on devices, and the gate-channel PN junction must be reverse-biased in order to turn the transistor off, the value for `VTO` is negative. Interestingly, this negative sign is true regardless of channel type (i.e. N-channel or P-channel) although the actual polarity necessary for reverse-biasing that junction is different for the two channel types.

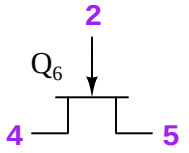
4.12.1 Example: Generic N-channel JFET

- Name = *6*
- Terminals = *drain on node 4 and gate on node 2 and source on node 5*
- Model = *nchannel*, generic NJF

SPICE element description

```
J6 4 2 5 nchannel
.MODEL nchannel njf
```

Schematic representation



4.13 Metal-Oxide Field-Effect Transistors (MOSFETs)

MOSFETs, like other semiconductor components in SPICE, must be accompanied by a `.model` card somewhere in the netlist specifying the model name and parameters. The channel type is specified in the `.model` card: NMOS for N-channel and PMOS for P-channel.

General format:

[**M***name*] [drain_node_ID] [gate_node_ID] [source_node_ID] [bulk_node_ID] [*model_name*]

[**.MODEL**] [*model_name*] [NMOS *or* PMOS] [**Parameter1=** *value*] [**Parameter2=** *value*] [$\cdots$]

Note: the "bulk" terminal describes the semiconductor substrate upon which the MOSFET is built, and is optional. The following is a list of parameters, any of which may be included in the `.model` card:

- VTO = Threshold pinch-off voltage (default = 0 Volts)
- KP = Transconductance (default = 2×10^{-5} A/V^2)
- GAMMA = Bulk threshold (default = 0 V$^{\frac{1}{2}}$)
- LAMBDA = Channel length modulation (default = 0 V^{-1})
- RD = Drain ohmic resistance (default = 0 Ohms)
- RS = Source ohmic resistance (default = 0 Ohms)
- RSH = Drain and source diffusion sheet resistance (default = 0 Ohms)
- CBD = Zero-bias B-D junction capacitance (default = 0 Farads)
- CBS = Zero-bias B-S junction capacitance (default = 0 Farads)
- CJ = Zero-bias bulk junction bottom capacitance (default = 0 Farads)
- MJ = Bulk junction grading coefficient (default = 0.5)
- CJSW = Zero-bias bulk junction sidewall capacitance (default = 0 Farads)
- MJSW = Bulk junction sidewall grading coefficient (default = 0.33)
- IS = Bulk junction saturation current (default = 1×10^{-14} Amperes)
- PB = Bulk junction potential (default = 1.0 Volt)
- CGDO = G-D overlap capacitance per unit channel width (default = 0 Farads/meter)
- CGSO = G-S overlap capacitance per unit channel width (default = 0 Farads/meter)

- CGBO = G-B overlap capacitance per unit channel width (default = 0 Farads/meter)
- TOX = Oxide thickness (default = infinite meters)
- LD = Lateral diffusion (default = 0 meters)

The same device card is used for enhancement-mode as well as depletion-mode MOSFETs, the only difference being the value assigned to the threshold pinch-off voltage (`VTO`). Enhancement-mode MOSFETs are normally-off devices and thus are "pinched off" at zero gate voltage, requiring `VTO` to be a positive value for NMOS devices and a negative value for PMOS devices. Depletion-mode MOSFETs are normally-on devices and need a "reverse-bias" gate voltage to turn off, requiring `VTO` to be a negative value for NMOS devices and a positive value for PMOS devices.

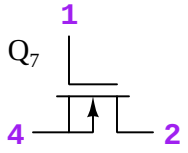
4.13.1 Example: Generic N-channel depletion-type MOSFET

- Name = *7*
- Terminals = *drain on node 2 and gate on node 1 and source on node 4 (bulk connected to source)*
- Model = *ndmos*, generic depletion-type NMOS

SPICE element description

```
M7 2 1 4 4 ndmos
.MODEL ndmos VTO=-3
```

Schematic representation



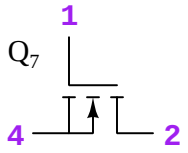
4.13.2 Example: Generic N-channel enhancement-type MOSFET

- Name = *7*
- Terminals = *drain on node 2 and gate on node 1 and source on node 4 (bulk connected to source)*
- Model = *nemos*, generic enhancement-type NMOS

SPICE element description

```
M7 2 1 4 4 nemos
.MODEL nemos VTO=2
```

Schematic representation



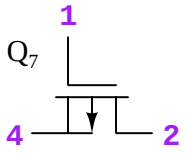
4.13.3 Example: Generic P-channel depletion-type MOSFET

- Name = *7*
- Terminals = *drain on node 2 and gate on node 1 and source on node 4 (bulk connected to source)*
- Model = *pdmos*, generic depletion-type PMOS

SPICE element description

```
M7 2 1 4 4 pdmos
.MODEL pdmos VTO=3
```

Schematic representation



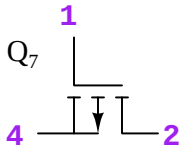
4.13.4 Example: Generic P-channel enhancement-type MOSFET

- Name = *7*
- Terminals = *drain on node 2 and gate on node 1 and source on node 4 (bulk connected to source)*
- Model = *pemos*, generic enhancement-type PMOS

SPICE element description

```
M7 2 1 4 4 pemos
.MODEL pemos VTO=-2
```

Schematic representation



4.14 Subcircuits

Computer programming languages provide the means to perform *subroutines*, which consist of collections of code (instructions) separate from the main body of the program, but which may be "called" from the main program as many times as needed. This is used in applications where a particular algorithm or set of instructions must be repeatedly executed: rather than have those instructions appear repeated throughout the program, they are entered as a subroutine and that subroutine is called by name as many times as necessary.

SPICE has the ability to do the equivalent when simulating electric circuits. A collection of components that would otherwise need to be repeatedly entered in the netlist can be defined once as a *subcircuit*, and then that subcircuit may be called by name as often as desired elsewhere in the netlist. Subcircuits are useful when the overall circuit is comprised of multiple identical sub-networks, because it makes the netlist shorter and easier to understand to a human reader.

[**X**] [*subcircuit_name*] [node_1] [node_2] [$\cdots$]

[**.SUBCKT***name*] [node_1] [node_2] $\cdots$]

All node numbers with the exception of node 0 used within the `.SUBCKT` card are "local" to that card and may be re-used for other purposes elsewhere in the netlist.

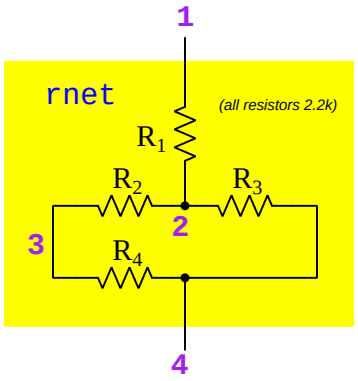
4.14.1 Example: resistor subnetwork

- Name = *rnet*
- Terminals = *first node is upper, second is lower*

SPICE subcircuit description

```
.SUBCKT rnet 1 4
R1 1 2 2200
R2 2 3 2200
R3 2 4 2200
R4 3 4 2200
.ENDS rnet
```

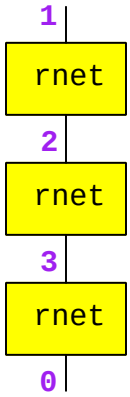
Schematic representation



SPICE subcircuit usage

```
X1 1 2 rnet
X2 2 3 rnet
X3 3 0 rnet
```

Schematic representation



4.14.2 Example: solar cell array

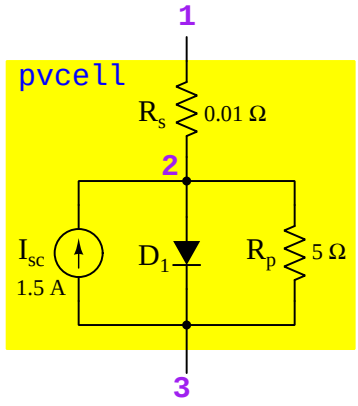
The simulation of a solar cell array is an excellent use of both subcircuits and semiconductor models in SPICE. Each individual solar cell outputs about the same amount of voltage as the forward drop of a silicon rectifying diode, because a solar cell *is* a silicon PN junction. The larger the cell's junction contact area, the greater its short-circuit current capacity. We may simulate each cell's short-circuit (maximum) current using a current source (I_{sc}) and its PN-junction characteristics using a diode element (D_1) with added resistors representing substrate and inter-cell connection resistances:

- Name = *pvcell*
- Terminals = *first node is positive, second is negative*

SPICE subcircuit description

```
.SUBCKT pvcell 1 3
RS 1 2 0.01
ISC 3 2 1.5
D1 2 3 pnjunc
RP 2 3 5
.MODEL pnjunc d
.ENDS pvcell
```

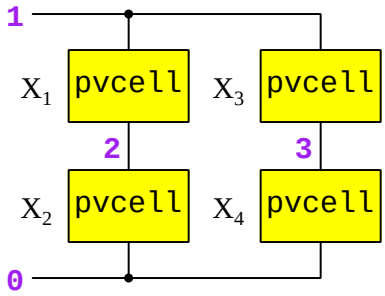
Schematic representation



SPICE subcircuit usage

```
X1 1 2 pvcell
X2 2 0 pvcell
X3 1 3 pvcell
X4 3 0 pvcell
```

Schematic representation



Chapter 5

SPICE analysis descriptions

To review, here are some of the "cardinal rules" for writing SPICE netlists:

- The first line (card) of the netlist is the Title. It cannot be omitted.
- The last line (card) of the netlist is the `.end` command. It cannot be omitted.
- Each and every unique connection point in the circuit must be assigned a number called a *node*. This is how you describe the "shape" of the circuit to SPICE: by describing which nodes each component connect to. Component terminals sharing common node numbers are electrically common (i.e. directly connected to each other). There *must* be a node 0, and this is the "Ground" node of the circuit.
- Comment lines (cards) must begin with an asterisk symbol (`*`). These are lines of text inserted into the netlist strictly for the benefit of human readers. SPICE skips over them.

Note that SPICE is case-insensitive. Any instances of capital letters in the following SPICE analysis descriptions are included solely for clarity and readability.

5.1 DC voltage/current "sweep" analysis

General format:

[**.dc**] [*source_name*] [*start_value*] [*end_value*] [*increment*]

The starting and ending values for the sweep will override any static values specified in the source's line (card).

5.1.1 Example: sweep of voltage source

- Source = *voltage source "v1"*
- Starting value = *2 Volts*
- Ending value = *12 Volts*
- Increment value = *5 Volts*

SPICE analysis description: `.dc v1 2 12 5`

5.1.2 Example: sweep of voltage and current sources

- Source1 = *voltage source "vce"*

  Starting value = *0 Volts*

  Ending value = *20 Volts*

  Increment value = *0.5 Volt*

- Source2 = *current source "ib"*

  Starting value = *0 milliAmperes*

  Ending value = *0.6 milliAmperes*

  Increment value = *0.1 milliAmperes*

SPICE analysis description: `.dc vce 0 20 0.5 ib 0 0.6e-3 0.1e-3`

Multi-source sweeps are useful when plotting "families" of characteristic curves for semiconductor devices, where each curve plotted is at a different controlling signal value. In the example shown, V_{CE} is swept from 0 to 20 Volts at an I_B value of 0 milliAmperes, then swept again from 0 to 20 Volts at I_B = 0.1 mA, then swept again from 0 to 20 Volts at I_B = 0.2 mA, etc.

5.2 AC frequency "sweep" analysis

General format:

[**.ac**] [*linearity*] [*points*] [*start_frequency*] [*end_frequency*]

5.2.1 Example: linear frequency sweep

- Linearity type = *linear*
- Number of points = *100*
- Starting frequency = *400 Hz*
- Ending frequency = *900 Hz*

SPICE analysis description: `.ac lin 100 400 900`

5.2.2 Example: decade logarithmic frequency sweep

- Linearity type = *decade*
- Number of points = *5 per decade*
- Starting frequency = *1 kHz*
- Ending frequency = *100 kHz*

SPICE analysis description: `.ac dec 5 1k 100k`

5.2.3 Example: octave logarithmic frequency sweep

- Linearity type = *octave*
- Number of points = *12 per octave*
- Starting frequency = *440 Hz*
- Ending frequency = *3.52 kHz*

SPICE analysis description: `.ac oct 12 440 3.52e3`

Note: this particular analysis covers four octaves of the frequencies of the equal-tempered Western musical scale, beginning with 440 Hz (concert tuning pitch "A").

5.3 Transient analysis

General format:

[**.tran**] [*interval_time*] [*end_time*] [*start_time*] [**uic**] [*end_frequency*]

This instructs SPICE to perform a time-domain analysis of the circuit, from time $t = 0$ to the specified end time, over intervals of specified length. The starting time is used only for analyses where the printed or plotted results do not begin at $t = 0$; if unspecified it is assumed that the starting time is zero. The `UIC` parameter is optional, instructing SPICE to honor any initial conditions (`IC=`) specified in the component descriptions.

5.3.1 Example: using initial conditions, beginning at time $t = 0$

- Time interval = *10 milliseconds*
- Ending time = *500 milliseconds*

SPICE analysis description: `.tran 10m 500m uic`

5.3.2 Example: using initial conditions, beginning at non-zero time

- Time interval = *30 microseconds*
- Ending time = *200 microseconds*
- Print/plot starting time = *100 microseconds*

SPICE analysis description: `.tran 30u 200u 100u uic`

5.4 Fourier analysis

General format:

[**.four**] [*fundamental_frequency*] [*variable*] [*variable*] [*variable* ···]

Fourier analysis provides a printed-text summary of the first ten harmonic frequencies of the specified variable(s). This analysis must always be used in conjunction with a *transient* (`.tran`) analysis in the same netlist, since the Fourier transform takes a time-domain function (i.e. transient data in SPICE) and converts it into a frequency-domain function.

5.4.1 Example: analysis of 60 Hz waveform

- Fundamental frequency = *60 Hz*
- Variable = *Voltage between nodes 3 and 1*

SPICE analysis description: `.four 60 v(3,1)`

5.5 Display option: print

General format:

[**.print**] [*analysis_type*] [*variable*] [*variable*] [*variable* ···]

In order for this display option to function, the netlist (deck) must also contain an analysis line (card).

5.5.1 Example: printing a DC analysis

- Analysis type = *DC* (netlist must contain a `.dc` analysis line)
- First variable = *Voltage between node 4 and Ground*
- Second variable = *Voltage between nodes 6 and 3*
- Third variable = *Current through voltage source "vsupply"*

SPICE analysis description: `.print dc v(4) v(6,3) i(vsupply)`

5.5.2 Example: printing an AC analysis

- Analysis type = *AC* (netlist must contain an `.ac` analysis line)
- First variable = *Voltage magnitude between node 4 and Ground*
- Second variable = *Voltage phase angle between node 4 and Ground*
- Third variable = *Current magnitude through voltage source "vsupply"*
- Fourth variable = *Current phase angle through voltage source "vsupply"*

SPICE analysis description: `.print ac vm(4) vp(4) im(vsupply) ip(vsupply)`

Note that SPICE version 2 defaults to *degrees* as the unit for phase angles, while SPICE version 3 defaults to *radians*. If degrees are desired while running newer versions of SPICE, you may include the following cards in the netlist deck to control units:

```
.control
set units=degrees
.endc
```

5.6 Display option: plot

General format:

[**.plot**] [*analysis_type*] [*variable*] [*variable*] [*variable* ···]

The `.plot` display accepts all the same parameters as the `.print` display, but instead of merely printing a list of numbers it plots a graph of the variables over the domain of the analysis type. In order for this display option to function, the netlist (deck) must also contain an analysis line (card).

Since the purpose of the `.plot` display is to produce a *graph*, the analysis line really needs to "sweep" over multiple points.

5.6.1 Example: plotting a DC analysis

- Analysis type = *DC* (netlist must contain a `.dc` analysis line)
- First variable = *Voltage between nodes 4 and 1*

SPICE analysis description: `.plot dc v(4,1)`

5.6.2 Example: plotting an AC analysis

- Analysis type = *AC* (netlist must contain an `.ac` analysis line)
- First variable = *Voltage magnitude between nodes 4 and 1*
- Second variable = *Voltage phase angle between nodes 4 and 1*

SPICE analysis description: `.plot ac vm(4,1) vp(4,1)`

Note that SPICE version 2 defaults to *degrees* as the unit for phase angles, while SPICE version 3 defaults to *radians*. If degrees are desired while running newer versions of SPICE, you may include the following cards in the netlist deck to control units:

```
.control
set units=degrees
.endc
```

It is important to note that an AC plot is in the frequency domain; that is to say, the x axis of the plot will represent frequency while the y axis will represent the variable(s) specified in the `.plot` line. This is useful for generating Bode plots. An AC analysis will *not* plot waveforms as a function of time. That feature is delivered by `plot` only when the netlist specifies a *transient* analysis rather than an *AC* analysis.

5.6.3 Example: logarithmically plotting an AC analysis

- Analysis type = *AC* (netlist must contain an `.ac` analysis line)
- Variable = *Voltage magnitude in decibels between nodes 3 and 7*

SPICE analysis description: `.plot ac vdb(3,7)`

5.6.4 Example: plotting a transient analysis

- Analysis type = *transient* (netlist must contain a `.tran` analysis line)
- First variable = *Voltage between nodes 7 and 3*

SPICE analysis description: `.plot tran v(7,3)`

This plot will show the waveform as a function of time, with the plot's x axis expressed in units of time.

5.6.5 Example: plotting parametric functions

- Analysis type = *transient* (netlist must contain a `.tran` analysis line)
- First variable = *Voltage between nodes 1 and 0*
- First variable = *Voltage between nodes 5 and 8*

SPICE analysis description: `.plot tran v(1) vs v(5,8)`

Not all versions of SPICE support the "versus" option, but in those that do, this is useful for plotting one circuit parameter against another, rather than plot circuit parameter(s) over time. Use this option to create Lissajous figures, plotting one sinusoidal voltage against another at different frequencies and/or phase shifts.

5.7 Display option: width

General format:

[**.width**] [**out=** *characters*]

When SPICE outputs plain-text to the computer terminal, it assumes a 120-character display width, which is fitting for legacy teletype machines, but actually exceeds the standard electronic terminal display width of 80 characters. For this reason, especially when using the `.plot` output option, it is advisable to also use the `.width` option set to 80 characters to limit the width of the text display.

5.7.1 Example

- Display width = *80 characters*

SPICE analysis description: `.width out=80`

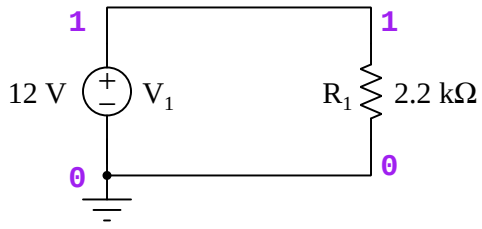
Chapter 6

Primitive circuit examples

"Primitive" circuits consist of the minimum number of components to demonstrate a concept. In most cases they consist of one source and one load directly connected together. They are presented here as a form of "Hello World" example to show how SPICE netlists and analyses are formatted.

6.1 DC voltage source with .op analysis

Circuit schematic diagram (with node numbers listed). Note how the voltage source and the resistor are connected between the same pair of nodes: 1 and 0. This tells SPICE they are parallel to each other, because the upper terminals of each component are electrically common (node 1) and the lower terminals of each component are also electrically common (node 0). Note also how ground is defined in SPICE as node number *zero*, which must be included in every netlist:



SPICE netlist:

```
* SPICE circuit
v1 1 0 dc 12
r1 1 0 2200
.op
.end
```

SPICE version 2G6 analysis (edited for brevity):

```
node   voltage
 (1)   12.0000

   voltage source currents

   name       current
   v1       -5.455E-03

   total power dissipation   6.55E-02  watts
```

Just to show you what the raw output of SPICE version 2G6 looks like for this analysis, here it is in all its unedited glory (note that I had to shorten some of the lines that exceeded the page width):

```
1*******09/02/16 ********  spice 2g.6    3/15/83 ********12:04:38*****

0* spice circuit

0****     input listing                    temperature =   27.000 deg c

0******************************************************************************



 v1 1 0 dc 12
 r1 1 0 2200
 .op
 .end
1******09/02/16 *********  spice 2g.6 3/15/83 ********12:04:38**********

0* spice circuit
0****  small signal bias solution           temperature =   27.000 deg c

0******************************************************************************


  node   voltage


 (  1)   12.0000



     voltage source currents

     name       current


     v1       -5.455E-03


     total power dissipation   6.55E-02  watts
0
         job concluded
0         total job time            0.00
```

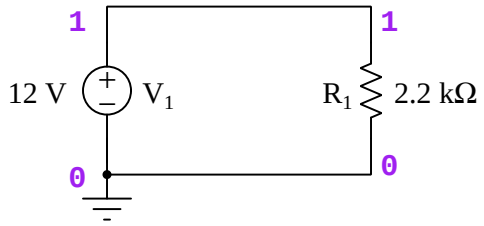
NGSPICE version 26 is more verbose than the legacy SPICE version 2G6 running the "Operating Point" (`.op`) analysis, requiring more than a page to display:

```
Circuit: * spice circuit

Doing analysis at TEMP = 27.000000 and TNOM = 27.000000


No. of Data Rows : 1
Node                                  Voltage
----                                  -------
-----------
V(1)                              1.200000e+01

Source Current
-------------

v1#branch                         -5.45455e-03

 Resistor models (Simple linear resistor)
      model                       R

        rsh                       0
     narrow                       0
      short                       0
        tc1                       0
        tc2                       0
       defw                   1e-05
          l                   1e-05
         kf                       0
         af                       0
          r                       0
     bv_max                   1e+99

 Resistor: Simple linear resistor
     device                      r1
      model                       R
 resistance                    2200
         ac                    2200
      dtemp                       0
     bv_max                   1e+99
      noisy                       1
          i              0.00545455
          p               0.0654545

 Vsource: Independent voltage source
```

```
     device                    v1
         dc                    12
      acmag                     0
      pulse         -
       sine         -
        sin         -
        exp         -
        pwl         -
       sffm         -
         am         -
    trnoise         -
   trrandom         -
          i           -0.00545455
          p             0.0654545

CPU time since last call: 0.032 seconds.

Total CPU time: 0.032 seconds.

Total DRAM available = 489.546875 MB.
DRAM currently available = 152.234375 MB.
Total ngspice program size = 2.191406 MB.
Resident set size = 593.000 kB.
Shared ngspice pages = 464.000 kB.
Text (code) pages = 1.087891 MB.
Stack = 0 bytes.
Library pages = 164.000 kB.
```

6.2 DC voltage source with single-point .dc sweep analysis

Circuit schematic diagram (with node numbers listed). Note how ground is defined in SPICE as node number *zero*. In this example we will maintain the voltage source at a constant value of 12 Volts DC. The "sweep" analysis is intended to cover a specified range of voltage values, but we are exploiting that analysis option here to perform an analysis at only a single voltage value:



SPICE netlist:

```
* SPICE circuit
v1 1 0 dc 12
r1 1 0 2200
.dc v1 12 12 1
.print dc v(1) i(v1)
.end
```

SPICE version 2G6 analysis (edited for brevity):

```
  v1           v(1)         i(v1)
1.200E+01     1.200E+01  -5.455E-03
```

SPICE version 2G6 is slightly less verbose when performing the `.dc` analysis than when performing the `.op` analysis. Here is the "raw" output of SPICE version 2G6 for your perusal:

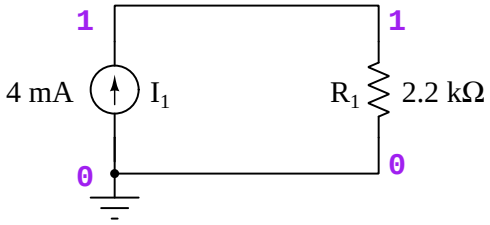
```
1*******09/02/16 ********  spice 2g.6    3/15/83 ********12:22:56*****

0* spice circuit

0****     input listing                    temperature =   27.000 deg c

0******************************************************************************



 v1 1 0 dc 12
 r1 1 0 2200
 .dc v1 12 12 1
 .print dc v(1) i(v1)
 .end
1******09/02/16 *********  spice 2g.6  3/15/83 *********12:22:56********

0* spice circuit
0****    dc transfer curves      temperature =   27.000 deg c

0******************************************************************************



   v1            v(1)        i(v1)
x

  1.200E+01     1.200E+01  -5.455E-03
y
0
         job concluded
0         total job time            0.00
```

NGSPICE version 26 is substantially less verbose when performing the `.dc` analysis. Here is the "raw" output of NGSPICE version 26 for your perusal:

```
Circuit: * spice circuit

Doing analysis at TEMP = 27.000000 and TNOM = 27.000000


No. of Data Rows : 1
                          * spice circuit
                          DC transfer characteristic  Fri Sep  2 12:28:41  2016
--------------------------------------------------------------------------------
Index   v-sweep         v(1)            v1#branch
--------------------------------------------------------------------------------
0       1.200000e+01    1.200000e+01    -5.45455e-03

CPU time since last call: 0.032 seconds.

Total CPU time: 0.032 seconds.

Total DRAM available = 489.546875 MB.
DRAM currently available = 152.128906 MB.
Total ngspice program size = 2.191406 MB.
Resident set size = 595.000 kB.
Shared ngspice pages = 466.000 kB.
Text (code) pages = 1.087891 MB.
Stack = 0 bytes.
Library pages = 164.000 kB.
```

6.3 DC current source with single-point .dc sweep analysis

Circuit schematic diagram (with node numbers listed). Note how ground is defined in SPICE as node number *zero*. In this example we will maintain the current source at a constant value of 4 milliAmperes DC. The "sweep" analysis is intended to cover a specified range of current values, but we are exploiting that analysis option here to perform an analysis at only a single current value:



SPICE netlist:

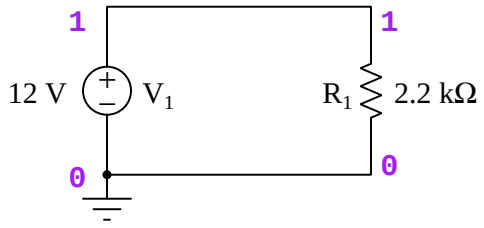
```
* SPICE circuit
i1 0 1 dc 4e-3
r1 1 0 2200
.dc i1 4e-3 4e-3 1
.print dc v(1)
.end
```

SPICE version 2G6 analysis (edited for brevity):

```
   i1            v(1)
 4.000E-03     8.800E+00
```

6.4 DC voltage source with multi-point .dc sweep analysis

Circuit schematic diagram (with node numbers listed). Note how ground is defined in SPICE as node number *zero*. In this example we will "sweep" the voltage source from 2 to 12 Volts DC in increments of 1 Volt:



First, showing the "printed" output option:

SPICE netlist:

```
* SPICE circuit
v1 1 0 dc 12
r1 1 0 2200
.dc v1 2 12 1
.print dc v(1) i(v1)
.end
```

SPICE version 2G6 analysis (edited for brevity):

```
 v1            v(1)         i(v1)
2.000E+00     2.000E+00  -9.091E-04
3.000E+00     3.000E+00  -1.364E-03
4.000E+00     4.000E+00  -1.818E-03
5.000E+00     5.000E+00  -2.273E-03
6.000E+00     6.000E+00  -2.727E-03
7.000E+00     7.000E+00  -3.182E-03
8.000E+00     8.000E+00  -3.636E-03
9.000E+00     9.000E+00  -4.091E-03
1.000E+01     1.000E+01  -4.545E-03
1.100E+01     1.100E+01  -5.000E-03
1.200E+01     1.200E+01  -5.455E-03
```

Next, showing the "plotted" output option[1]:

SPICE netlist:

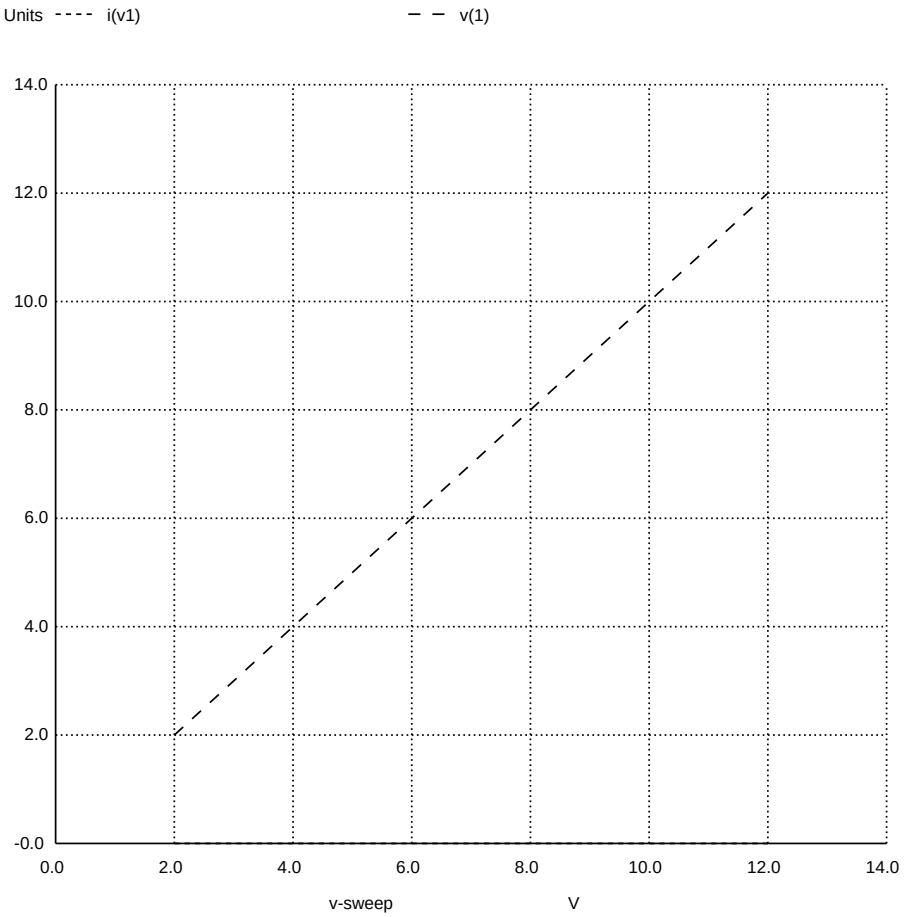
```
* SPICE circuit
v1 1 0 dc 12
r1 1 0 2200
.dc v1 2 12 1
.plot dc v(1) i(v1)
.width out=80
.end
```

SPICE version 2G6 analysis (edited for brevity):

```
legend:
 *: v(1)
 +: i(v1)

   v1          v(1)

(*)------------  0.000E+00     5.000E+00     1.000E+01     1.500E+01  2.000E+01
                 - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(+)------------ -6.000E-03    -4.000E-03    -2.000E-03     0.000E+00  2.000E-03
                 - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 2.000E+00  2.000E+00 .     *       .             .       +     .            .
 3.000E+00  3.000E+00 .       *     .             .   +         .            .
 4.000E+00  4.000E+00 .          *  .             .+            .            .
 5.000E+00  5.000E+00 .             *           + .             .            .
 6.000E+00  6.000E+00 .             .  *     +    .             .            .
 7.000E+00  7.000E+00 .             .     x       .             .            .
 8.000E+00  8.000E+00 .             .  +    *     .             .            .
 9.000E+00  9.000E+00 .            +.          *  .             .            .
 1.000E+01  1.000E+01 .         +   .             *             .            .
 1.100E+01  1.100E+01 .      +      .             .  *          .            .
 1.200E+01  1.200E+01 .   +         .             .     *       .            .
                 - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

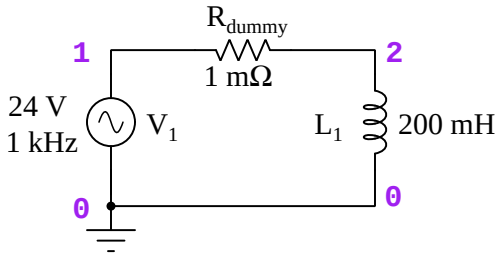
[1]SPICE assumes a 120-character width display, and so to fit the text output on a standard terminal the `.width` option must be invoked to set the display width at 80 characters

NGSPICE version 26 analysis (using window-based interactive mode on Linux/X-Windows):



6.5 AC voltage source with single-point .ac sweep analysis

Circuit schematic diagram (with node numbers listed). Note how the series connection between the "dummy" resistor and the inductor is specified by three nodes: 1, 2, and 0. The resistor connects between nodes 1 and 2, while the inductor connects between nodes 2 and 0. The one node common to both components (node 2) makes those two components share a common connection. Note as always how ground is defined in SPICE as node number *zero*. In this example we will "sweep" the source frequency from 1 kHz to 1 kHz with just one interval. The "dummy" resistance of 1 milli-Ohm is necessary in order to avoid a direct "loop" between the voltage source and the inductor, which SPICE cannot abide:



SPICE netlist:

```
* SPICE circuit
v1 1 0 ac 24
rdummy 1 2 1m
l1 2 0 200m
.ac lin 1 1k 1k
.print ac vm(1) vp(1) im(v1) ip(v1)
.end
```

SPICE version 2G6 analysis (edited for brevity):

```
    freq        vm(1)        vp(1)        im(v1)       ip(v1)
1.000E+03    2.400E+01    0.000E+00    1.910E-02    9.000E+01
```

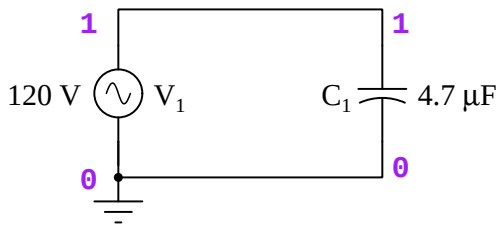
Early versions of SPICE such as 2G6 assumed phase angle unit of degrees, but many later versions assume *radians* when displaying phase angles. In order to specifically instruct modern versions of SPICE to display angles in degrees, the following "control" cards must be added to the netlist:

```
.control
set units=degrees
.endc
```

These cards may be added to the netlist "deck" anywhere between the first line (`* SPICE circuit`) and the last (`.end`).

6.6 AC voltage source with multi-point .ac sweep analysis

Circuit schematic diagram (with node numbers listed). Note how ground is defined in SPICE as node number *zero*. In this example we will "sweep" the source frequency from 60 Hz to 6 kHz on a decade logarithmic scale with 5 points per decade:



First, showing the "printed" output option:

SPICE netlist:

```
* SPICE circuit
v1 1 0 ac 120
c1 1 0 4.7u
.ac dec 5 60 6k
.print ac vm(1) vp(1) im(v1) ip(v1)
.end
```

SPICE version 2G6 analysis (edited for brevity):

```
    freq        vm(1)        vp(1)        im(v1)       ip(v1)
  6.000E+01     1.200E+02    0.000E+00    2.126E-01   -9.000E+01
  9.509E+01     1.200E+02    0.000E+00    3.370E-01   -9.000E+01
  1.507E+02     1.200E+02    0.000E+00    5.341E-01   -9.000E+01
  2.389E+02     1.200E+02    0.000E+00    8.465E-01   -9.000E+01
  3.786E+02     1.200E+02    0.000E+00    1.342E+00   -9.000E+01
  6.000E+02     1.200E+02    0.000E+00    2.126E+00   -9.000E+01
  9.509E+02     1.200E+02    0.000E+00    3.370E+00   -9.000E+01
  1.507E+03     1.200E+02    0.000E+00    5.341E+00   -9.000E+01
  2.389E+03     1.200E+02    0.000E+00    8.465E+00   -9.000E+01
  3.786E+03     1.200E+02    0.000E+00    1.342E+01   -9.000E+01
  6.000E+03     1.200E+02    0.000E+00    2.126E+01   -9.000E+01
```

Next, showing the "plotted" output option[2]:

SPICE netlist:

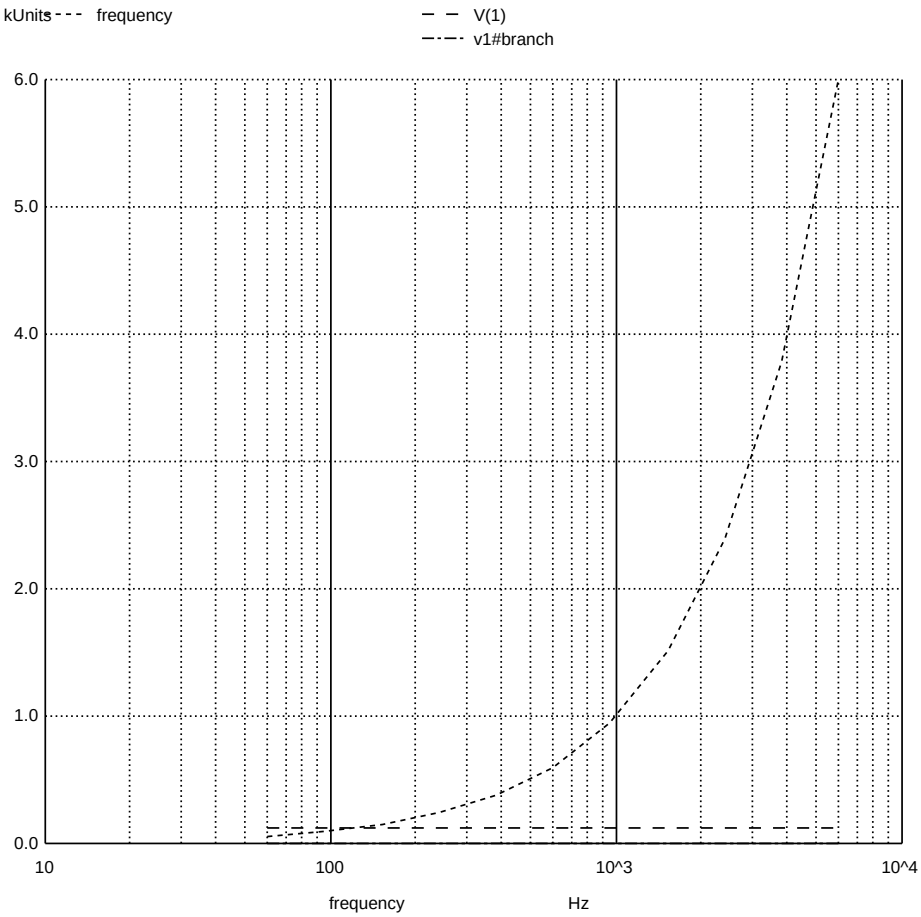
```
* SPICE circuit
v1 1 0 ac 120
c1 1 0 4.7u
.ac dec 5 60 6k
.plot ac vm(1) vp(1) im(v1) ip(v1)
.width out=80
.end
```

[2]SPICE assumes a 120-character width display, and so to fit the text output on a standard terminal the `.width` option must be invoked to set the display width at 80 characters

SPICE version 2G6 analysis (edited for brevity):

```
legend:
 *: vm(1)
 +: vp(1)
 =: im(v1)
 $: ip(v1)

     freq      vm(1)
(*)------------  1.000E+01     1.000E+02     1.000E+03     1.000E+04  1.000E+05
                 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(+)------------ -1.000E-12    -5.000E-13     0.000E+00     5.000E-13  1.000E-12
                 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(=)------------  1.000E-01     1.000E+00     1.000E+01     1.000E+02  1.000E+03
                 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
($)------------ -1.500E+02    -1.000E+02    -5.000E+01     0.000E+00  5.000E+01
                 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 6.000E+01  1.200E+02 .    =        .* $          +             .             .
 9.509E+01  1.200E+02 .      =      .* $          +             .             .
 1.507E+02  1.200E+02 .         =   .* $          +             .             .
 2.389E+02  1.200E+02 .            =.* $          +             .             .
 3.786E+02  1.200E+02 .             .*=$          +             .             .
 6.000E+02  1.200E+02 .             .* $ =        +             .             .
 9.509E+02  1.200E+02 .             .* $   =      +             .             .
 1.507E+03  1.200E+02 .             .* $      =   +             .             .
 2.389E+03  1.200E+02 .             .* $         =+             .             .
 3.786E+03  1.200E+02 .             .* $          + =           .             .
 6.000E+03  1.200E+02 .             .* $          +    =        .             .
                 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

NGSPICE version 26 analysis (using window-based interactive mode on Linux/X-Windows):



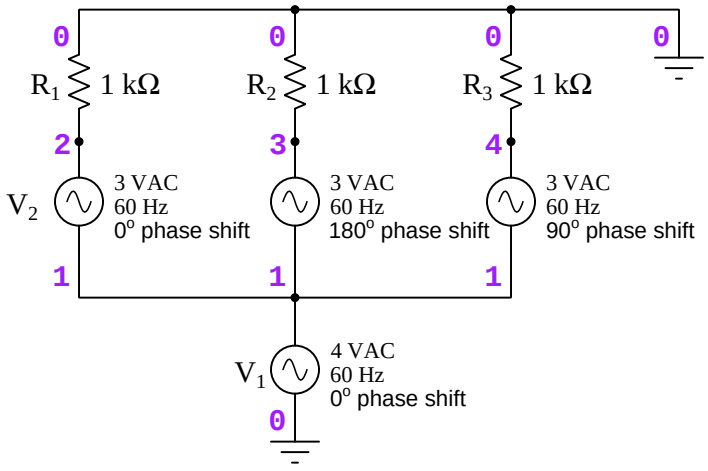
It should be noted that early versions of SPICE such as 2G6 assumed phase angle unit of degrees, but many later versions assume *radians* when displaying phase angles. In order to specifically instruct modern versions of SPICE to display angles in degrees, the following "control" cards must be added to the netlist:

```
.control
set units=degrees
.endc
```

These cards may be added to the netlist "deck" anywhere between the first line (`* SPICE circuit`) and the last (`.end`).

6.7 Additive AC voltage sources with single-point .ac sweep analysis

Circuit schematic diagram (with node numbers listed). Note the use of node numbers to specify the circuit's shape. Node 0 is the ground point, while nodes shared between different components specify direct connections between those components (i.e. all identically-numbered nodes are electrically common to each other). Nodes with different numbers are electrically distinct from each other:



SPICE netlist:

```
* SPICE circuit
v1 1 0 ac 4 0
v2 2 1 ac 3 0
v3 3 1 ac 3 180
v4 4 1 ac 3 90
r1 2 0 1k
r2 3 0 1k
r3 4 0 1k
.ac lin 1 60 60
.print ac vm(2) vm(3) vm(4)
.print ac vp(2) vp(3) vp(4)
.end
```

SPICE version 2G6 analysis (edited for brevity):

```
    freq        vm(2)       vm(3)       vm(4)
6.000E+01     7.000E+00   1.000E+00   5.000E+00

    freq        vp(2)       vp(3)       vp(4)
6.000E+01     0.000E+00   2.105E-14   3.687E+01
```

The voltage at node 2 (with respect to ground, 0) is the simple sum of 4 Volts and 3 Volts because V_1 and V_2 are perfectly in sync with each other, analogous to a pair of DC voltage sources connected in series with polarities additive. This also explains the zero phase shift of node 2's voltage: the sum of those two sources is in-phase with our reference source V_1 (at 0 degrees phase angle).

The voltage at node 3 is only 1 Volt (the difference between 4 Volts and 3 Volts) because V_1 and V_3 are perfectly *opposed* to one another, analogous to a pair of DC voltage sources connected in series with polarities subtractive. The phase shift of this voltage is practically zero, the only reason for the miniscule non-zero phase result being imperfections in SPICE's AC analysis.

The voltage at node 4 has no DC analogue because here the two sources V_1 and V_4 are neither directly in sync with each other nor perfectly opposed to each other – instead, these two sources are 90 degrees out of phase with each other. Their phasor sum is 5 Volts for the same reason that a right triangle with opposite and adjacent side lengths of 4 and 3 will have a hypotenuse length of 5. Note that the phase angle of this voltage is neither 0 degrees nor 90 degrees, but instead is 36.87 degrees: the same angle one would expect between the 4-length and 5-length sides of a right triangle (i.e. $\cos^{-1}\left(\frac{4}{5}\right) = 36.87^o$).

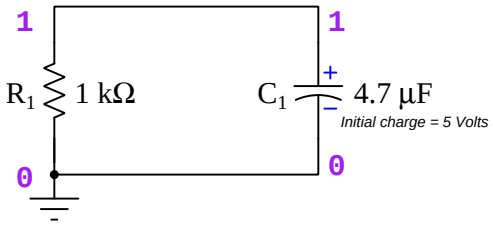
It should be noted that early versions of SPICE such as 2G6 assumed phase angle unit of degrees, but many later versions assume *radians* when displaying phase angles. In order to specifically instruct modern versions of SPICE to display angles in degrees, the following "control" cards must be added to the netlist:

```
.control
set units=degrees
.endc
```

These cards may be added to the netlist "deck" anywhere between the first line (`* SPICE circuit`) and the last (`.end`).

6.8 Transient analysis of discharging RC circuit

Circuit schematic diagram (with node numbers listed). Note how ground is defined in SPICE as node number *zero*. In this example we will pre-charge the capacitor to 5 Volts and analyze its discharge over time. The time-span for analysis is from $t = 0$ to $t = 250$ milliseconds, in 10 millisecond intervals:



First, showing the "printed" output option:

SPICE netlist:

```
* SPICE circuit
r1 1 0 1k
c1 1 0 47u ic=5
.tran 10m 250m uic
.print tran v(1)
.end
```

SPICE version 2G6 analysis (edited for brevity):

```
   time       v(1)
0.000E+00     5.000E+00
1.000E-02     4.046E+00
2.000E-02     3.270E+00
3.000E-02     2.643E+00
4.000E-02     2.136E+00
5.000E-02     1.726E+00
6.000E-02     1.395E+00
7.000E-02     1.127E+00
8.000E-02     9.112E-01
9.000E-02     7.364E-01
1.000E-01     5.951E-01
1.100E-01     4.810E-01
1.200E-01     3.887E-01
1.300E-01     3.142E-01
1.400E-01     2.539E-01
1.500E-01     2.052E-01
1.600E-01     1.658E-01
1.700E-01     1.340E-01
1.800E-01     1.083E-01
1.900E-01     8.754E-02
2.000E-01     7.075E-02
2.100E-01     5.718E-02
2.200E-01     4.621E-02
2.300E-01     3.735E-02
2.400E-01     3.018E-02
2.500E-01     2.437E-02
```

Next, showing the "plotted" output option[3]:

SPICE netlist:

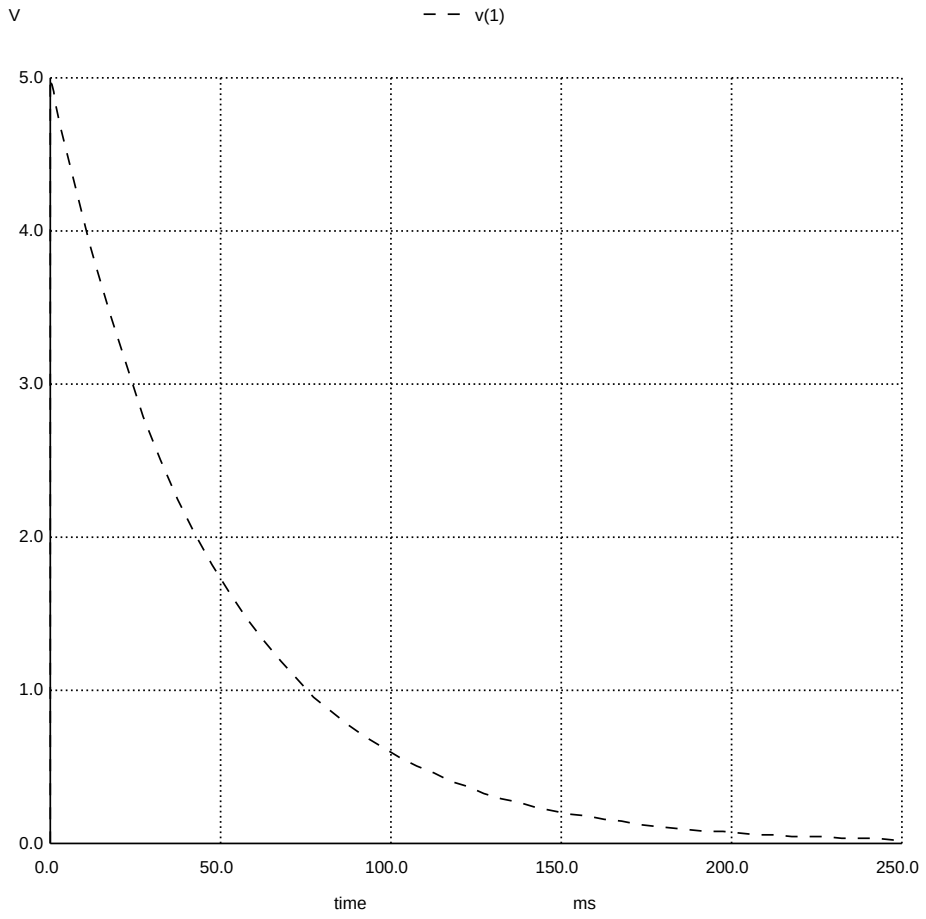
```
* SPICE circuit
r1 1 0 1k
c1 1 0 47u ic=5
.tran 10m 250m uic
.plot tran v(1)
.width out=80
.end
```

[3]SPICE assumes a 120-character width display, and so to fit the text output on a standard terminal the `.width` option must be invoked to set the display width at 80 characters

SPICE version 2G6 analysis (edited for brevity):

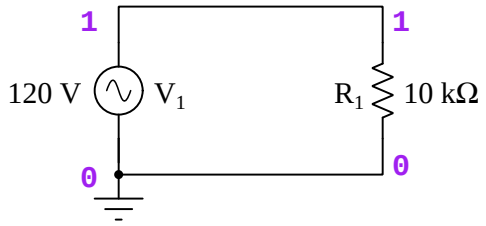
```
    time      v(1)
                0.000E+00     2.000E+00     4.000E+00     6.000E+00  8.000E+00
                - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
0.000E+00  5.000E+00 .             .             .      *      .             .
1.000E-02  4.046E+00 .             .             *             .             .
2.000E-02  3.270E+00 .             .        *    .             .             .
3.000E-02  2.643E+00 .             .   *         .             .             .
4.000E-02  2.136E+00 .             .*            .             .             .
5.000E-02  1.726E+00 .           * .             .             .             .
6.000E-02  1.395E+00 .         *   .             .             .             .
7.000E-02  1.127E+00 .       *     .             .             .             .
8.000E-02  9.112E-01 .     *       .             .             .             .
9.000E-02  7.364E-01 .    *        .             .             .             .
1.000E-01  5.951E-01 .   *         .             .             .             .
1.100E-01  4.810E-01 .  *          .             .             .             .
1.200E-01  3.887E-01 .  *          .             .             .             .
1.300E-01  3.142E-01 . *           .             .             .             .
1.400E-01  2.539E-01 . *           .             .             .             .
1.500E-01  2.052E-01 .*            .             .             .             .
1.600E-01  1.658E-01 .*            .             .             .             .
1.700E-01  1.340E-01 .*            .             .             .             .
1.800E-01  1.083E-01 .*            .             .             .             .
1.900E-01  8.754E-02 .*            .             .             .             .
2.000E-01  7.075E-02 *             .             .             .             .
2.100E-01  5.718E-02 *             .             .             .             .
2.200E-01  4.621E-02 *             .             .             .             .
2.300E-01  3.735E-02 *             .             .             .             .
2.400E-01  3.018E-02 *             .             .             .             .
2.500E-01  2.437E-02 *             .             .             .             .
                - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

NGSPICE version 26 analysis (using window-based interactive mode on Linux/X-Windows):



6.9 Transient analysis of a steady sinusoidal voltage source

Circuit schematic diagram (with node numbers listed). Note how ground is defined in SPICE as node number *zero*. In this example we will define a voltage source with a sinusoidal waveform, no DC offset, 120 Volt peak amplitude, 60 Hz frequency, no start delay, and no decay (i.e. an undamped waveform). The time-span for analysis is from $t = 0$ to $t = 34$ milliseconds, in 1 millisecond intervals:



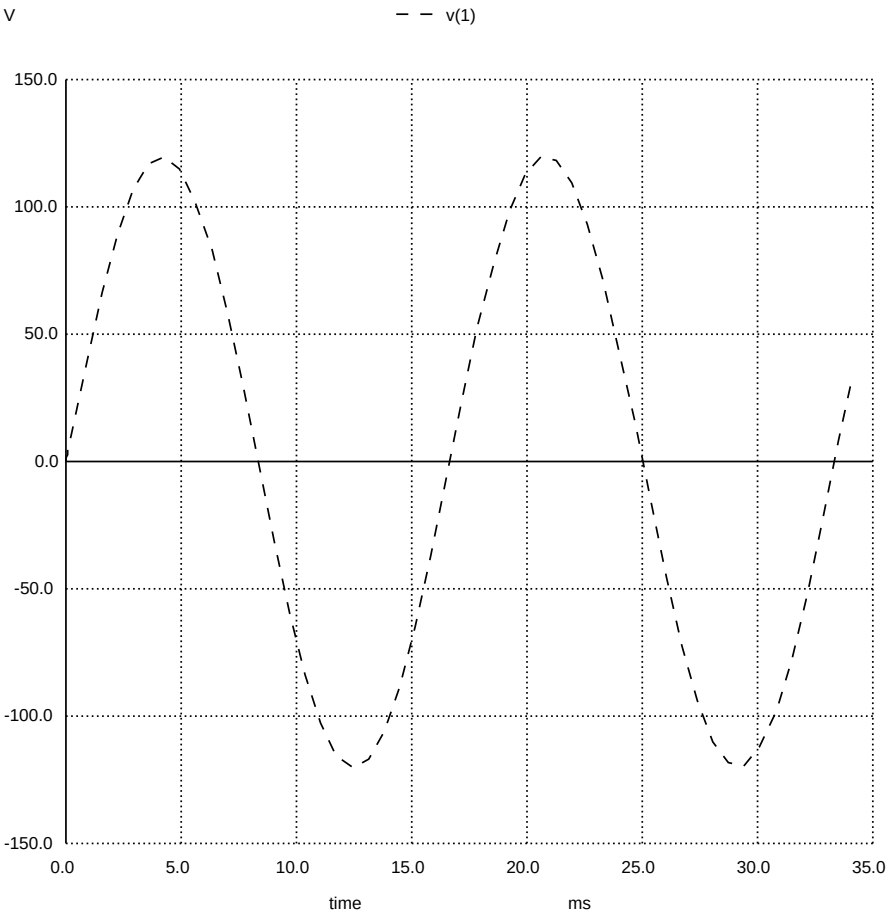
SPICE netlist:

```
* SPICE circuit
v1 1 0 sin (0 120 60 0 0)
r1 1 0 10e3
.tran 1m 34m
.plot tran v(1)
.width out=80
.end
```

SPICE version 2G6 analysis (edited for brevity):

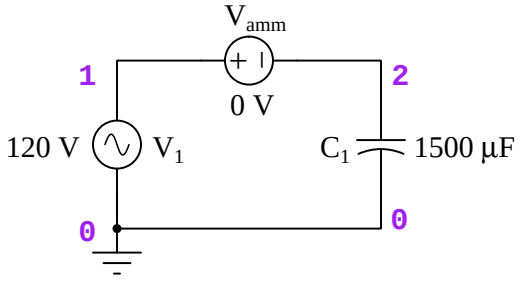
```
    time      v(1)
               -2.000E+02    -1.000E+02     0.000E+00     1.000E+02  2.000E+02
                    - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
0.000E+00  0.000E+00 .              .              *              .             .
1.000E-03  4.387E+01 .              .              .     *        .             .
2.000E-03  8.104E+01 .              .              .          *   .             .
3.000E-03  1.071E+02 .              .              .             .*             .
4.000E-03  1.181E+02 .              .              .             .  *           .
5.000E-03  1.125E+02 .              .              .             . *            .
6.000E-03  9.108E+01 .              .              .            *.              .
7.000E-03  5.690E+01 .              .              .       *     .              .
8.000E-03  1.473E+01 .              .              . *           .              .
9.000E-03 -2.951E+01 .              .         *    .             .              .
1.000E-02 -6.961E+01 .              .   *          .             .              .
1.100E-02 -9.993E+01 .              *              .             .              .
1.200E-02 -1.162E+02 .            * .              .             .              .
1.300E-02 -1.162E+02 .            * .              .             .              .
1.400E-02 -9.982E+01 .              *              .             .              .
1.500E-02 -6.945E+01 .              .   *          .             .              .
1.600E-02 -2.933E+01 .              .         *    .             .              .
1.700E-02  1.492E+01 .              .              . *           .              .
1.800E-02  5.707E+01 .              .              .       *     .              .
1.900E-02  9.120E+01 .              .              .            *.              .
2.000E-02  1.125E+02 .              .              .             . *            .
2.100E-02  1.180E+02 .              .              .             .  *           .
2.200E-02  1.070E+02 .              .              .             .*             .
2.300E-02  8.091E+01 .              .              .          *  .              .
2.400E-02  4.346E+01 .              .              .     *        .              .
2.500E-02 -9.465E-02 .              .              *             .              .
2.600E-02 -4.363E+01 .              .        *     .             .              .
2.700E-02 -8.104E+01 .              .  *           .             .              .
2.800E-02 -1.071E+02 .            *.               .             .              .
2.900E-02 -1.181E+02 .          *  .               .             .              .
3.000E-02 -1.125E+02 .           * .               .             .              .
3.100E-02 -9.108E+01 .             .*              .             .              .
3.200E-02 -5.690E+01 .              .     *        .             .              .
3.300E-02 -1.473E+01 .              .           * .             .              .
3.400E-02  2.951E+01 .              .             .   *         .              .
                    - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

NGSPICE version 26 analysis (using window-based interactive mode on Linux/X-Windows):



6.10 Transient analysis of a sinusoidal capacitive circuit

Circuit schematic diagram (with node numbers listed). Note how ground is defined in SPICE as node number *zero*. In this example we will define a voltage source with a sinusoidal waveform, no DC offset, 120 Volt peak amplitude, 60 Hz frequency, no start delay, and no decay (i.e. an undamped waveform). The time-span for analysis is from $t = 0$ to $t = 30$ milliseconds, in 1 millisecond intervals:



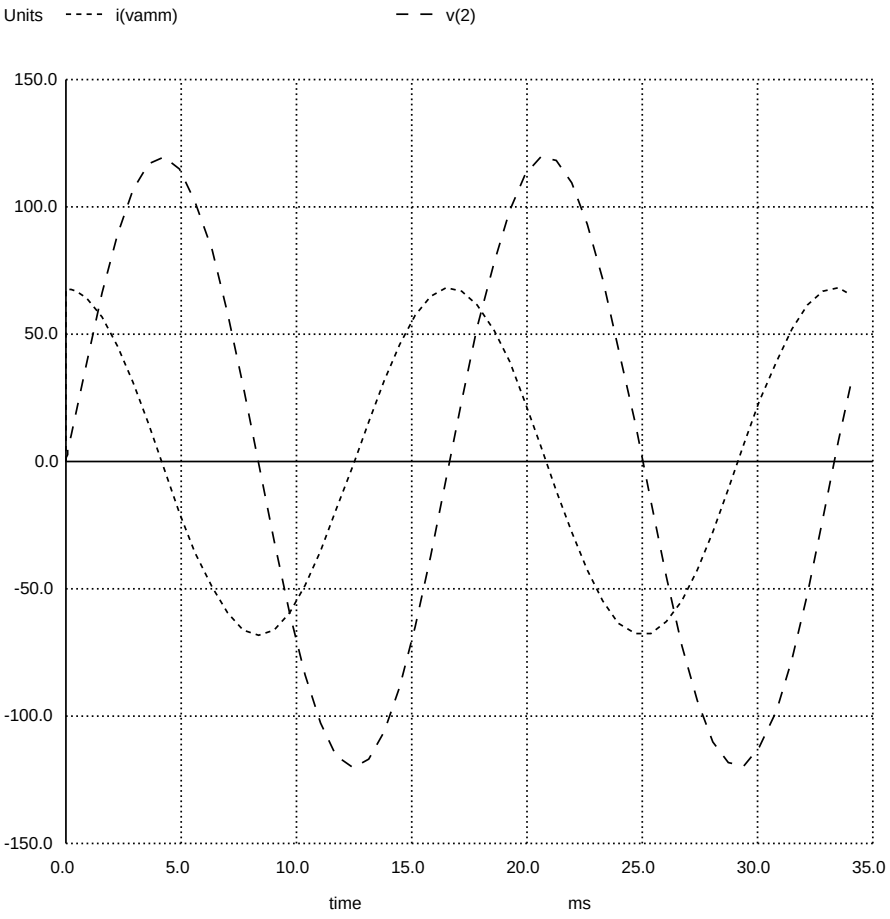
SPICE netlist:

```
* SPICE circuit
v1 1 0 sin (0 120 60 0 0)
c1 2 0 1500u
vamm 1 2 0
.tran 1m 30m
.plot tran v(1) i(vamm)
.width out=80
.end
```

SPICE version 2G6 analysis (edited for brevity):

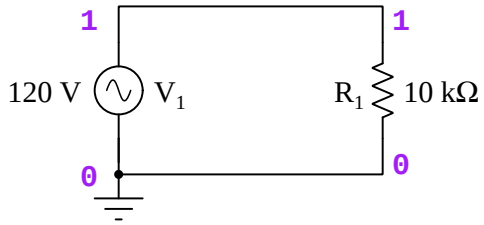
```
legend:
 *: v(1)
 +: i(vamm)

     time      v(1)

(*)------------ -2.000E+02    -1.000E+02     0.000E+00     1.000E+02  2.000E+02
                 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(+)------------ -1.000E+02    -5.000E+01     0.000E+00     5.000E+01  1.000E+02
                 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 0.000E+00  0.000E+00 .             .             x             .            .
 1.000E-03  4.390E+01 .             .             .     *       .   +        .
 2.000E-03  8.203E+01 .             .             .          *  +            .
 3.000E-03  1.080E+02 .             .             .       +     .*           .
 4.000E-03  1.190E+02 .             .             .+            .  *         .
 5.000E-03  1.140E+02 .             .       +     .             . *          .
 6.000E-03  9.197E+01 .             . +           .            *.            .
 7.000E-03  5.747E+01 .          +  .             .       *     .            .
 8.000E-03  1.503E+01 .        +    .             . *           .            .
 9.000E-03 -2.971E+01 .         +   .         *   .             .            .
 1.000E-02 -7.010E+01 .            +.   *         .             .            .
 1.100E-02 -1.012E+02 .            *   +         .             .            .
 1.200E-02 -1.173E+02 .          * .         +   .             .            .
 1.300E-02 -1.172E+02 .          * .             .   +         .            .
 1.400E-02 -1.012E+02 .            *             .         +   .            .
 1.500E-02 -7.015E+01 .            .   *         .             .+           .
 1.600E-02 -2.967E+01 .            .         *   .             .   +        .
 1.700E-02  1.501E+01 .            .             . *           .    +       .
 1.800E-02  5.753E+01 .            .             .       *     .  +         .
 1.900E-02  9.189E+01 .            .             .           +*.            .
 2.000E-02  1.140E+02 .            .             .     +       . *          .
 2.100E-02  1.191E+02 .            .           +.              .  *         .
 2.200E-02  1.079E+02 .            .     +       .             .*           .
 2.300E-02  8.205E+01 .            +             .         *   .            .
 2.400E-02  4.393E+01 .         +  .             .     *       .            .
 2.500E-02  1.243E-02 .        +   .             *             .            .
 2.600E-02 -4.411E+01 .         +  .       *     .             .            .
 2.700E-02 -8.174E+01 .            +  *          .             .            .
 2.800E-02 -1.079E+02 .           *.     +       .             .            .
 2.900E-02 -1.196E+02 .          *  .           +.             .            .
 3.000E-02 -1.141E+02 .          * .             .     +       .            .
                 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

NGSPICE version 26 analysis (using window-based interactive mode on Linux/X-Windows):



6.11 Transient analysis of a damped, offset sinusoidal voltage source

Circuit schematic diagram (with node numbers listed). Note how ground is defined in SPICE as node number *zero*. In this example we will define a voltage source with a sinusoidal waveform, an 80 Volt DC offset, 120 Volt peak amplitude, 60 Hz frequency, no start delay, and a damping factor of 40 s^{-1}. The time-span for analysis is from $t = 0$ to $t = 34$ milliseconds, in 1 millisecond intervals:



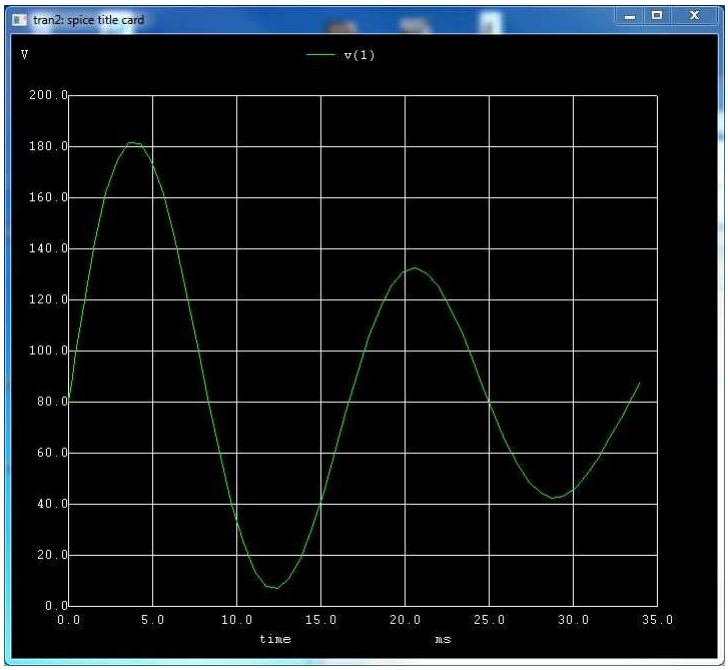
SPICE netlist:

```
* SPICE circuit
v1 1 0 sin (80 120 60 0 40)
r1 1 0 10e3
.tran 1m 34m
.plot tran v(1)
.width out=80
.end
```

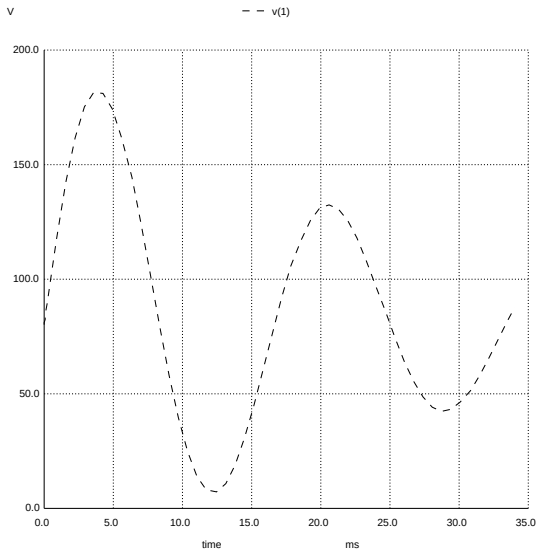
SPICE version 2G6 analysis (edited for brevity):

```
    time      v(1)
                0.000E+00     5.000E+01     1.000E+02     1.500E+02  2.000E+02
                - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
0.000E+00  8.000E+01 .             .       *     .             .            .
1.000E-03  1.221E+02 .             .             .     *       .            .
2.000E-03  1.552E+02 .             .             .             .*           .
3.000E-03  1.759E+02 .             .             .             .      *     .
4.000E-03  1.812E+02 .             .             .             .        *   .
5.000E-03  1.733E+02 .             .             .             .      *     .
6.000E-03  1.522E+02 .             .             .             .*           .
7.000E-03  1.237E+02 .             .             .      *      .            .
8.000E-03  9.098E+01 .             .          *  .             .            .
9.000E-03  5.922E+01 .             .  *          .             .            .
1.000E-02  3.321E+01 .        *    .             .             .            .
1.100E-02  1.497E+01 .   *         .             .             .            .
1.200E-02  7.632E+00 . *           .             .             .            .
1.300E-02  1.023E+01 .  *          .             .             .            .
1.400E-02  2.246E+01 .     *       .             .             .            .
1.500E-02  4.146E+01 .           * .             .             .            .
1.600E-02  6.426E+01 .             .   *         .             .            .
1.700E-02  8.750E+01 .             .         *   .             .            .
1.800E-02  1.079E+02 .             .             . *           .            .
1.900E-02  1.229E+02 .             .             .     *       .            .
2.000E-02  1.311E+02 .             .             .        *    .            .
2.100E-02  1.313E+02 .             .             .        *    .            .
2.200E-02  1.250E+02 .             .             .      *      .            .
2.300E-02  1.125E+02 .             .             .   *         .            .
2.400E-02  9.691E+01 .             .            *.             .            .
2.500E-02  8.008E+01 .             .       *     .             .            .
2.600E-02  6.442E+01 .             .   *         .             .            .
2.700E-02  5.238E+01 .             .*            .             .            .
2.800E-02  4.469E+01 .           *.              .             .            .
2.900E-02  4.274E+01 .           * .             .             .            .
3.000E-02  4.577E+01 .            *.             .             .            .
3.100E-02  5.339E+01 .             .*            .             .            .
3.200E-02  6.399E+01 .             .   *         .             .            .
3.300E-02  7.596E+01 .             .      *      .             .            .
3.400E-02  8.766E+01 .             .          *  .             .            .
                - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

NGSPICE version 26 analysis (using window-based interactive mode on Microsoft Windows 7):

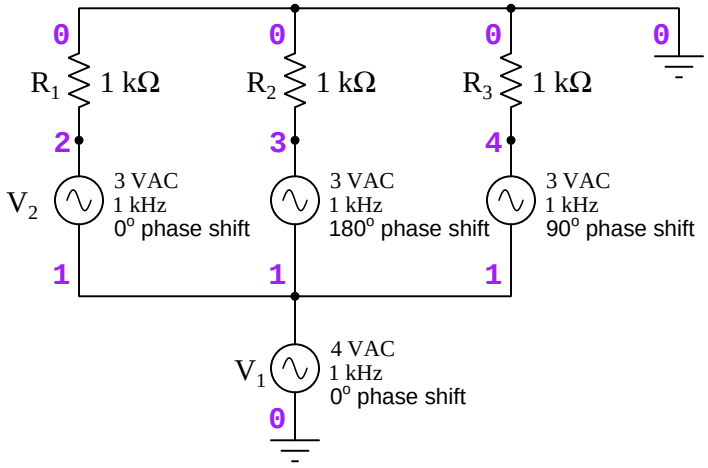


NGSPICE version 26 analysis (using window-based interactive mode on Linux/X-Windows):



6.12 Additive AC voltage sources with transient analysis

Circuit schematic diagram (with node numbers listed):



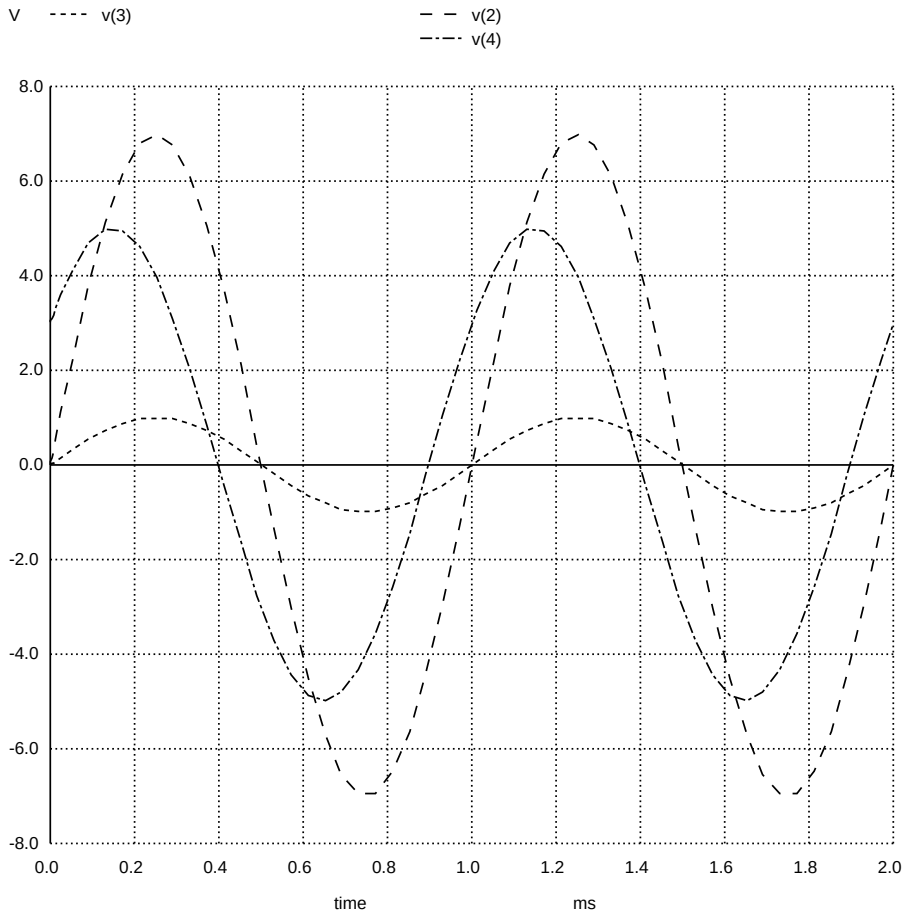
SPICE netlist:

```
* SPICE circuit
v1 1 0 sin (0 4 1k 0 0)
v2 2 1 sin (0 3 1k 0 0)
v3 3 1 sin (0 3 1k -0.5m 0)
v4 4 1 sin (0 3 1k -0.25m 0)
r1 2 0 1k
r2 3 0 1k
r3 4 0 1k
.tran 0.05m 4m
.plot tran v(1) v(2,1) v(3,1) v(4,1)
.plot tran v(2) v(3) v(4)
.end
```

Here, the phase shift of each sinusoidal voltage source must be specified in terms of *start delay time*. A 1000 Hertz waveform has a period of 1 millisecond, and therefore 180 degrees of phase shift is equivalent to a start delay of 0.5 milliseconds (i.e. one-half of the 360^o period) and 90 degrees of phase shift is equivalent to a start delay of 0.25 milliseconds (i.e. one-quarter of the 360^o period). The start delay times are shown here as negative quantities because we want the phase shifts to be *leading* (e.g. the V_3 sinusoidal source with a start delay of -0.5 milliseconds is already at its 180 degree mark by the time our reference source V_1 begins at 0 degrees.).

An important note here is that SPICE version 2G6 does not handle negative start delay time values, but NGSPICE version 26 does. For this reason, a SPICE version 2G6 analysis is omitted.

NGSPICE version 26 analysis (using window-based interactive mode on Linux/X-Windows):



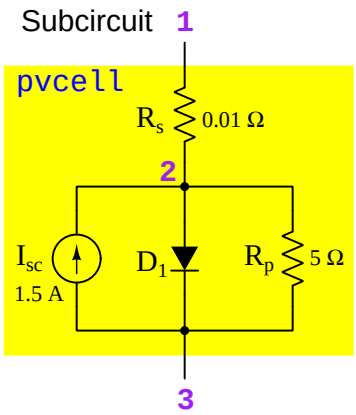
The voltage at node 2 (with respect to ground, 0) is the simple sum of 4 Volts and 3 Volts because V_1 and V_2 are perfectly in sync with each other, analogous to a pair of DC voltage sources connected in series with polarities additive. The voltage at node 3 is only 1 Volt (the difference between 4 Volts and 3 Volts) because V_1 and V_3 are perfectly *opposed* to one another, analogous to a pair of DC voltage sources connected in series with polarities subtractive.

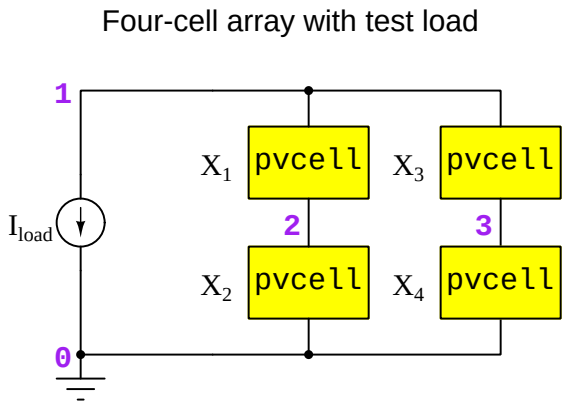
The voltage at node 4 has no DC analogue because here the two sources V_1 and V_4 are neither directly in sync with each other nor perfectly opposed to each other – instead, these two sources are 90 degrees out of phase with each other. Their phasor sum is 5 Volts for the same reason that a right triangle with opposite and adjacent side lengths of 4 and 3 will have a hypotenuse length of 5.

Note how the voltage at node 4 does not peak at the same times as the voltages at nodes 2 or 3 because its phase shift is not 0 degrees as with the others.

6.13 Solar cell array simulation

This simulation uses both subcircuits and semiconductor models to represent a solar (photovoltaic) panel comprised of four individual cells. Each cell is modeled by the `pvcell` subcircuit, while the four-cell series-parallel array calls on that subcircuit four times. In order to plot the array's voltage-current characteristic curve, we use an external current source as a controlled load (I_{load}) and "sweep" the load current from zero to the full short-circuit rating of the array. In this case, with four cells connected in series-parallel fashion we would expect an open-circuit voltage approximately equal to two diode forward voltage drops and a short-circuit current approximately equal to two cells' parallel-total current:





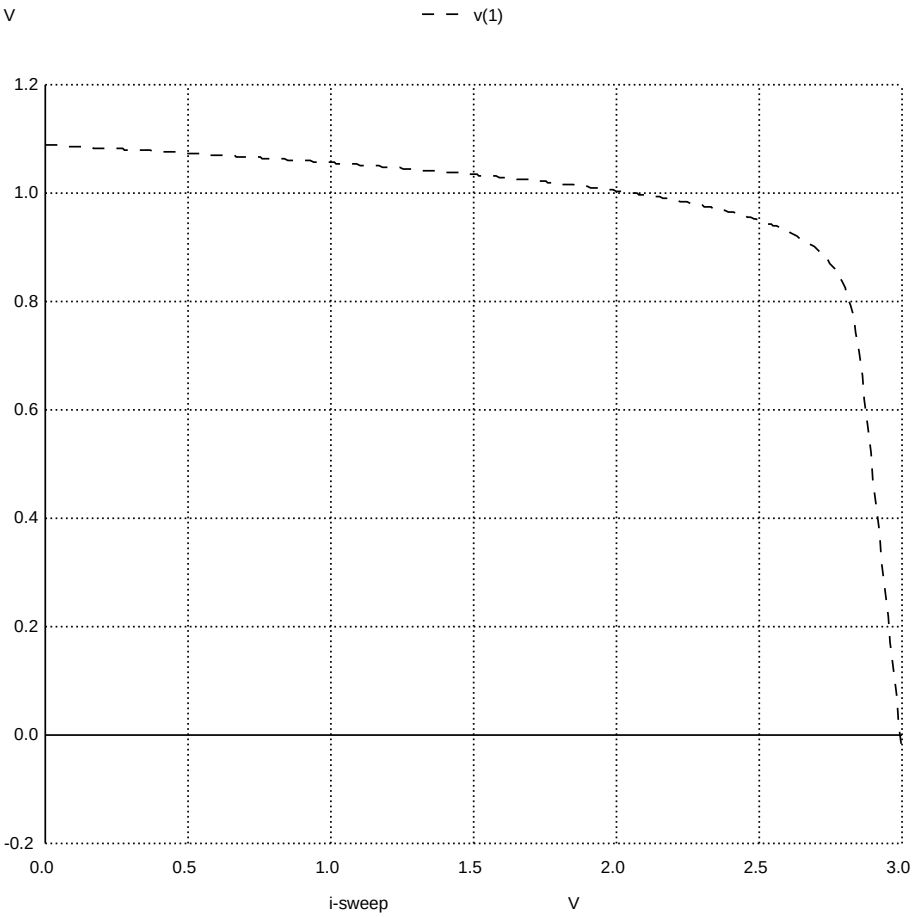
SPICE netlist:[4]

```
* SPICE circuit
x1 1 2 pvcell
x2 2 0 pvcell
x3 1 3 pvcell
x4 3 0 pvcell
iload 1 0
.subckt pvcell 1 3
rs 1 2 0.01
isc 3 2 1.5
d1 2 3 pnjunc
rp 2 3 5
.model pnjunc d is=1e-9 vj=0.65
.ends pvcell
.dc iload 0 3 0.01
.plot dc v(1)
```

[4] `.plot dc` statement intended to display the array's voltage as a function of load current.

```
.end
```

NGSPICE version 26 analysis (using window-based interactive mode on Linux/X-Windows):



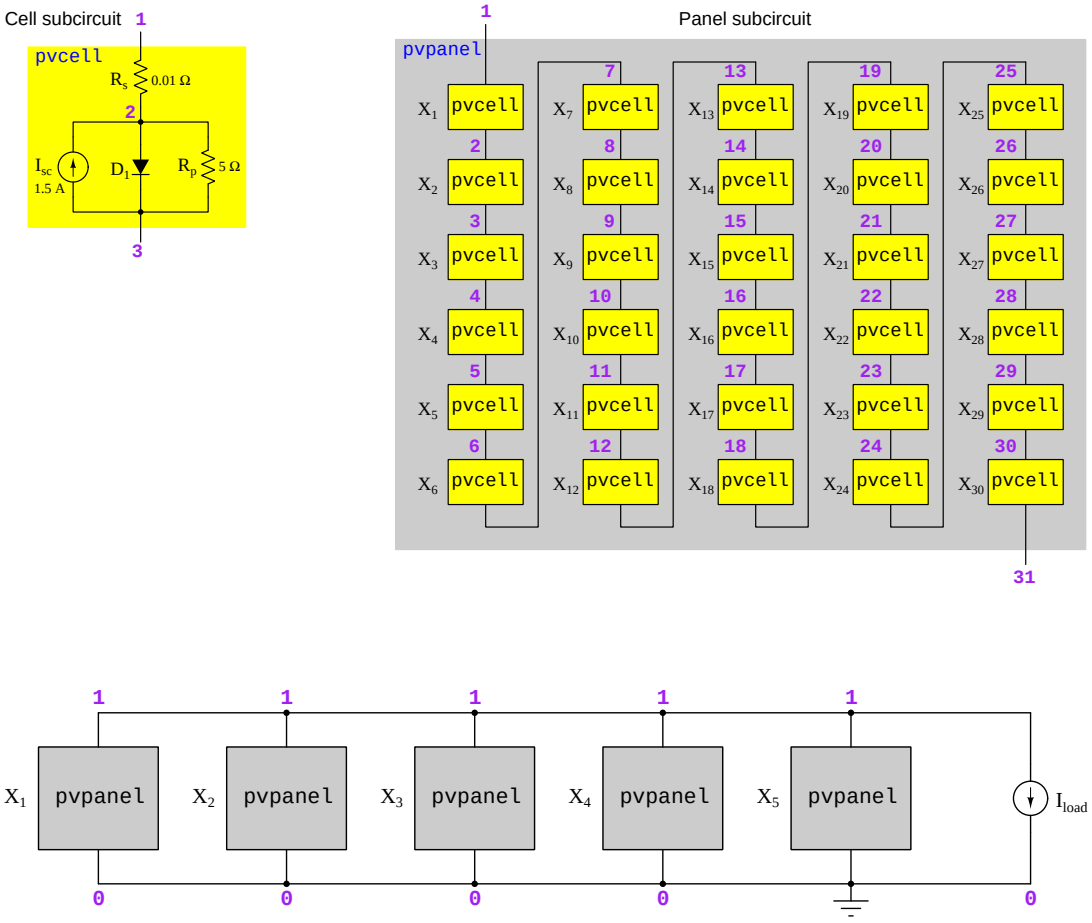
In this analysis we see the open-circuit condition represented at the far left of the graph (maximum voltage, zero current) and the short-circuit condition represented at the far right (zero voltage[5], maximum current).

The amount of open-circuit voltage for each cell is primarily a function of the cell subcircuit's diode model (`vj` parameter defining its nominal forward-voltage drop, and `is` parameter defining its saturation current). The amount of short-circuit current for each cell is primarily a function of the cell's current source (I_{sc}). The slope of the voltage-current curve toward the right-hand side is primarily a function of the cell's internal parallel resistance (R_p).

[5] We actually see the voltage dip slightly below zero at a load current of 3 Amperes, because at this point I_{load} is actually behaving as a *source* rather than a load. This demonstrates that the array's short-circuit capability is actually slightly less than 3 Amperes.

6.14 Solar panel array simulation

This simulation uses nested subcircuits, the lowest-level subcircuit representing a single solar (photovoltaic) cell and the next-level subcircuit representing a single panel comprised of 30 of those cells. The top-level simulation uses five of those panels in parallel to power a test load.



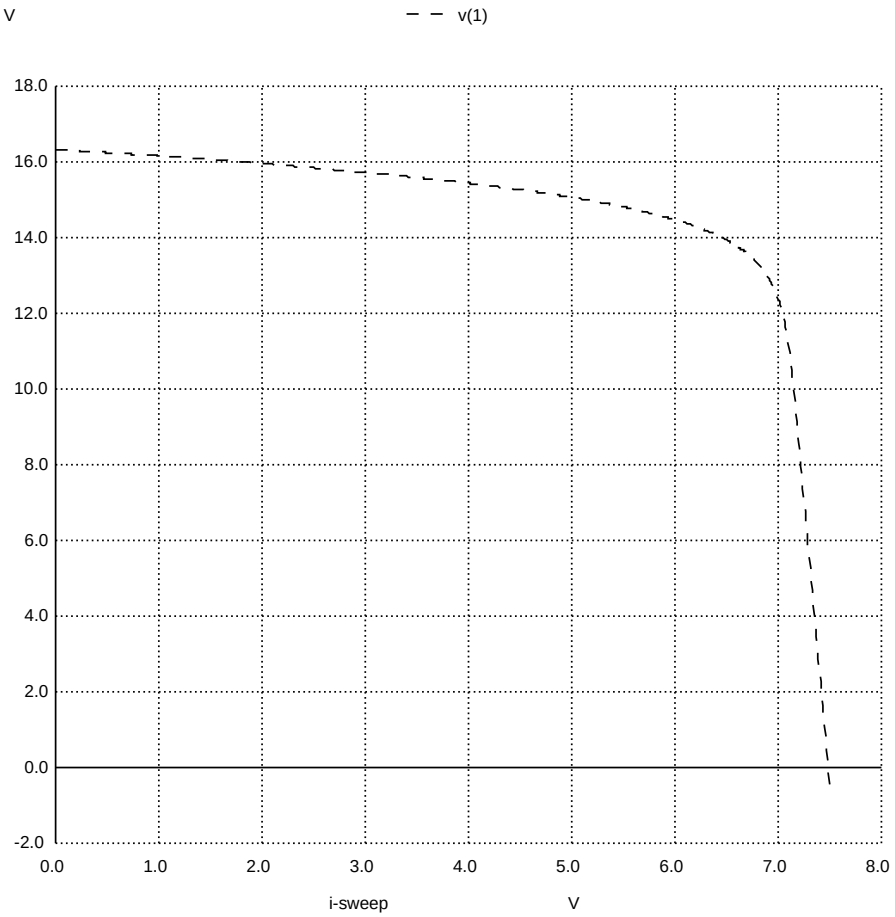
SPICE netlist:[6]

```
* SPICE circuit
x1 1 0 pvpanel
x2 1 0 pvpanel
x3 1 0 pvpanel
x4 1 0 pvpanel
x5 1 0 pvpanel
iload 1 0
*
* Panel subcircuit
.subckt pvpanel 1 31
x1 1 2 pvcell
x2 2 3 pvcell
x3 3 4 pvcell
x4 4 5 pvcell
x5 5 6 pvcell
x6 6 7 pvcell
x7 7 8 pvcell
x8 8 9 pvcell
x9 9 10 pvcell
x10 10 11 pvcell
x11 11 12 pvcell
x12 12 13 pvcell
x13 13 14 pvcell
x14 14 15 pvcell
x15 15 16 pvcell
x16 16 17 pvcell
x17 17 18 pvcell
x18 18 19 pvcell
x19 19 20 pvcell
x20 20 21 pvcell
x21 21 22 pvcell
x22 22 23 pvcell
x23 23 24 pvcell
x24 24 25 pvcell
x25 25 26 pvcell
x26 26 27 pvcell
x27 27 28 pvcell
x28 28 29 pvcell
x29 29 30 pvcell
```

[6] `.plot dc` statement intended to display the array's voltage as a function of load current. Also note the addition comments (`*`) inserted to visually separate and clearly label the subcircuits, so as to distinguish them from the rest of the netlist code.

```
x30 30 31 pvcell
.ends pvpanel
*
* Cell subcircuit
.subckt pvcell 1 3
rs 1 2 0.01
Isc 3 2 1.5
d1 2 3 pnjunc
rp 2 3 5
.model pnjunc d is=1e-9 vj=0.65
.ends pvcell
*
.dc iload 0 7.5 0.01
.plot dc v(1)
.end
```

NGSPICE version 26 analysis (using window-based interactive mode on Linux/X-Windows):



In this analysis we see the open-circuit condition represented at the far left of the graph (maximum voltage, zero current) and the short-circuit condition represented at the far right (zero voltage[7], maximum current).

The amount of open-circuit voltage for each cell is primarily a function of the cell subcircuit's diode model (`vj` parameter defining its nominal forward-voltage drop, and `is` parameter defining its saturation current). The amount of short-circuit current for each cell is primarily a function of the cell's current source (I_{sc}). The slope of the voltage-current curve toward the right-hand side is primarily a function of the cell's internal parallel resistance (R_p).

One of the tremendous advantages of using subcircuits in SPICE netlists is also seen when using

[7] We actually see the voltage dip slightly below zero at a load current of 7.5 Amperes, because at this point I_{load} is actually behaving as a *source* rather than a load. This demonstrates that the array's short-circuit capability is actually slightly less than 7.5 Amperes.

subroutines in a computer programming language: by relegating the specific details of each solar cell to its own sub-set of the code, we may very easily make adjustments to that sub-set and watch the effects of those adjustments on the larger system. Imagine if we had to build a SPICE netlist without subcircuits, representing all 150 current sources, 150 diodes, 150 series resistors, and 150 parallel resistors! Not only would the netlist be enormous, but it would also be extremely tedious to alter any parameter(s) common to every cell.

Chapter 7

Gallery

All SPICE netlists showcased in this "Gallery" have been tested on two different versions of SPICE: SPICE version 2G6, and a modern variant of SPICE called NGSPICE (version 26 or newer). In most cases only one of these SPICE versions is used to generate the output.

When an analysis has been noted as "edited for brevity" it means that I have deleted unnecessary blank lines, unnecessary spaces, unnecessary characters, and all the statistical information typically output by SPICE in order to maximize the readability of the SPICE text output. This editing is especially important for those who wish to use the Gallery examples as practice problems (with answers provided by SPICE) for developing manual circuit analysis skills and do not care about the particulars of SPICE.

7.1 Using gallery examples for practice

Perhaps the most obvious use of the example netlists and analyses presented in this "Gallery" chapter is to learn how to write well-formed netlists for SPICE, and how to interpret the output generated by SPICE simulations. In other words, you may use these examples to help you learn how to use SPICE as a circuit simulation tool specific types of circuits.

However, this is not the only use of these examples. Students in quantitative fields of study such as electricity and electronics benefit from having sets of worked example problems to hone and test their burgeoning analytical skills. Textbooks commonly contain practice problems, followed by answers (at least for *some* of these problems!) located in an appendix against which students may self-check their own results. The SPICE example analyses presented in this "Gallery" chapter may serve as the same: sample problems complete with their own answer keys. Even more, once the basic syntax and netlist format of SPICE is mastered, students may use SPICE to generate answer keys for circuit problems of their own making, thus freeing them from the limitations of their textbooks' problem sets.

Here are some suggestions for using the Gallery examples as practice problems:

1. Take the given information presented in the circuit schematic diagrams (also in the corresponding SPICE netlists) and use your analytical skills to calculate those same voltages and/or currents that SPICE is tasked with computing. When you think you have a complete analysis, turn to the SPICE analysis to check if your answers match.

2. Take the circuit schematic diagram and the results of the SPICE analysis as given parameters, and then use your analytical skills to "work backwards" to one or more of the specified component values in the SPICE netlist. For example, taking the schematic diagram and component voltage drops, and then calculating source voltage/current from that.

3. Modify component values within a SPICE netlist and re-run the analysis to make a new quantitative problem with different given conditions.

4. Use SPICE to predict the effects of *qualitative* changes by either increasing or decreasing any one component's value and re-running the simulation, challenging yourself beforehand to predict all the effects of that qualitative change before letting SPICE tell you what will happen.

5. Use SPICE to predict the effects of *component faults* (e.g. opens and shorts) by altering resistance values to become very large (open) or very small (short), or by inserting high resistances in series with a component to simulate an "open" fault and low resistances in parallel to simulate a "short" fault. As with qualitative component value changes, you will challenging yourself beforehand to predict all the effects of that fault before letting SPICE tell you what will happen.

7.2 Series resistor circuits

Series components by definition share a single pathway for current. Because of this, series-connected components share the same (continuous) current value at any given moment in time.

Series connections are specified in SPICE by successive components sharing one node number (making one terminal of a component electrically common with one terminal of the next component). For example, a partial netlist specifying three 1 kΩ resistors connected in series with each other might look like this:

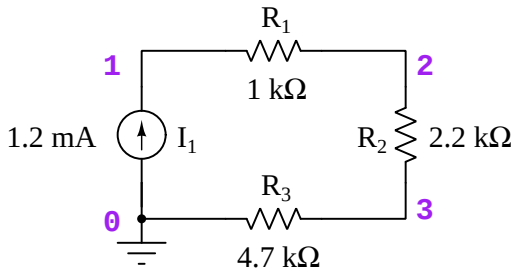
Partial SPICE netlist:

```
* Three series-connected resistors
r1 5 3 1000
r2 3 8 1000
r3 8 1 1000
```

Beginning at node 5, we encounter resistor R_1, which then connects to resistor R_2 at node 3, which then connects to resistor R_3 at node 8, the series chain ending at node 1.

7.2.1 One DC current source, three resistors

Circuit schematic diagram (with node numbers listed):



SPICE netlist:[1]

```
* SPICE circuit
i1 0 1 dc 1.2e-3
r1 2 1 1000
r2 2 3 2200
r3 3 0 4700
.dc i1 1.2e-3 1.2e-3 1
.print dc v(1,2) v(2,3) v(3)
.print dc v(1)
.end
```

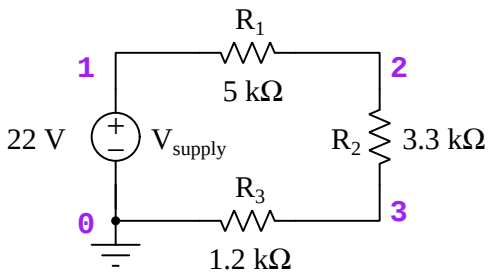
SPICE version 2G6 analysis (edited for brevity):

```
 i1           v(1,2)       v(2,3)       v(3)
1.200E-03     1.200E+00    2.640E+00    5.640E+00

 i1           v(1)
1.200E-03     9.480E+00
```

[1] `.print dc` statements intended to display each resistor's voltage in order, followed by source current.

7.2.2 One DC voltage source, three resistors

Circuit schematic diagram (with node numbers listed):



SPICE netlist:[2]

```
* SPICE circuit
vsupply 1 0 dc 22
r1 2 1 5000
r2 2 3 3300
r3 3 0 1200
.dc vsupply 22 22 1
.print dc v(1,2) v(2,3) v(3)
.print dc i(vsupply)
.end
```

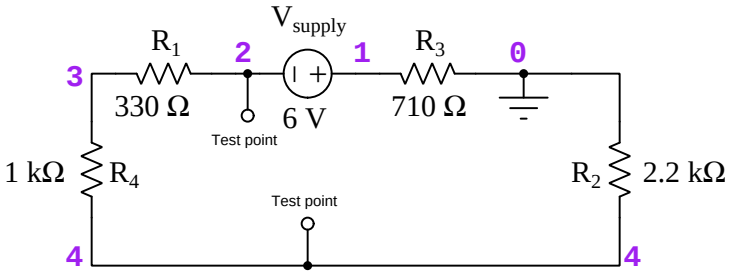
SPICE version 2G6 analysis (edited for brevity):

```
 vsupply      v(1,2)      v(2,3)      v(3)
2.200E+01    1.158E+01   7.642E+00   2.779E+00

 vsupply      i(vsupply)
2.200E+01   -2.316E-03
```

[2] `.print dc` statements intended to display each resistor's voltage in order, followed by source current.

7.2.3 One DC voltage source, four resistors

Circuit schematic diagram (with node numbers listed):



SPICE netlist:[3]

```
* SPICE circuit
vsupply 1 2 dc 6
r1 2 3 330
r2 0 4 2200
r3 1 0 710
r4 3 4 1000
.dc vsupply 6 6 1
.print dc v(3,2) v(0,4) v(1)
.print dc v(4,3) v(2) v(4)
.print dc i(vsupply)
.end
```

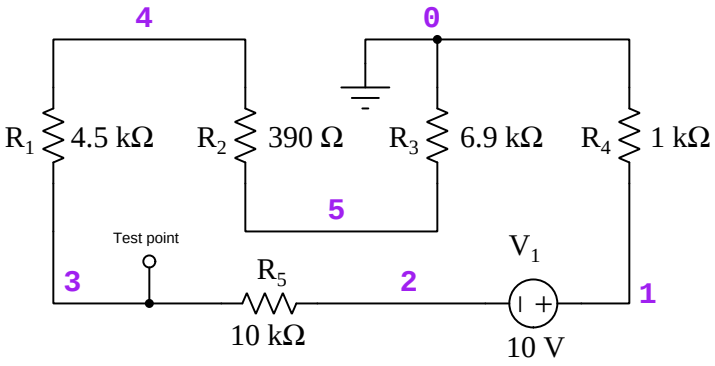
SPICE version 2G6 analysis (edited for brevity):

```
 vsupply      v(3,2)      v(0,4)      v(1)
6.000E+00    4.670E-01   3.113E+00   1.005E+00

 vsupply      v(4,3)      v(2)        v(4)
6.000E+00    1.415E+00  -4.995E+00  -3.113E+00

 vsupply      i(vsupply)
6.000E+00   -1.415E-03
```

[3] `.print dc` statements intended to display each resistor's voltage in order, followed by the test point voltages (with respect to ground) at nodes 2 and then 4, followed by source current.

7.2.4 One DC current source, five resistors

Circuit schematic diagram (with node numbers listed):



SPICE netlist:[4]

```
* SPICE circuit
v1 1 2 dc 10
r1 3 4 4.5e3
r2 4 5 390
r3 5 0 6.9e3
r4 0 1 1e3
r5 2 3 10e3
.dc v1 10 10 1
.print dc v(4,3) v(5,4) v(5) v(1)
.print dc v(3,2) v(3) i(v1)
.end
```

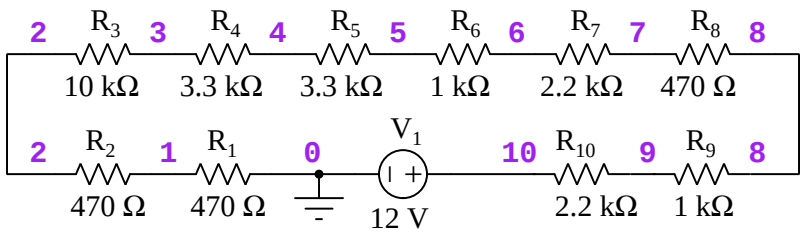
SPICE version 2G6 analysis (edited for brevity):

```
 v1           v(4,3)       v(5,4)       v(5)       v(1)
1.000E+01     1.975E+00    1.711E-01   -3.028E+00    4.388E-01

 v1           v(3,2)       v(3)         i(v1)
1.000E+01     4.388E+00   -5.173E+00   -4.388E-04
```

[4] `.print dc` statements intended to display each resistor's voltage in order, followed by the test point voltage (with respect to ground) at node 3, followed by source current.

7.2.5 One DC voltage source, ten resistors

Circuit schematic diagram (with node numbers listed):



SPICE netlist:[5]

```
* SPICE circuit
v1 10 0 dc 12
r1 0 1 470
r2 1 2 470
r3 2 3 10k
r4 3 4 3.3k
r5 4 5 3.3k
r6 5 6 1k
r7 6 7 2.2k
r8 7 8 470
r9 8 9 1k
r10 9 10 2.2k
.dc v1 12 12 1
.print dc v(1) v(2,1) v(3,2) v(4,3) v(5,4)
.print dc v(6,5) v(7,6) v(8,7) v(9,8) v(10,9)
.print dc i(v1)
.end
```

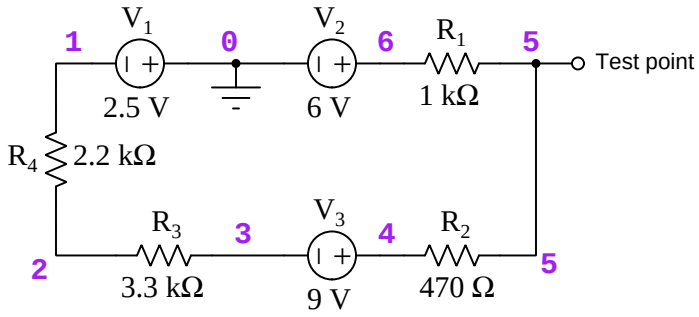
[5] `.print dc` statements intended to display each resistor's voltage in order, followed by source current.

SPICE version 2G6 analysis (edited for brevity):

```
 v1           v(1)         v(2,1)       v(3,2)       v(4,3)       v(5,4)
1.200E+01     2.311E-01    2.311E-01    4.916E+00    1.622E+00    1.622E+00

 v1           v(6,5)       v(7,6)       v(8,7)       v(9,8)       v(10,9)
1.200E+01     4.916E-01    1.082E+00    2.311E-01    4.916E-01    1.082E+00

 v1           i(v1)
1.200E+01    -4.916E-04
```

7.2.6 Three DC voltage sources, four resistors

Circuit schematic diagram (with node numbers listed):



SPICE netlist:[6]

```
* SPICE circuit
v1 0 1 dc 2.5
v2 6 0 dc 6
v3 4 3 dc 9
r1 6 5 1k
r2 5 4 470
r3 3 2 3.3k
r4 1 2 2.2k
.dc v1 2.5 2.5 1
.print dc v(5,6) v(4,5) v(2,3) v(1,2)
.print dc v(5) i(v1)
.end
```

SPICE version 2G6 analysis (edited for brevity):

```
 v1             v(5,6)       v(4,5)       v(2,3)       v(1,2)
2.500E+00      7.174E-02    3.372E-02    2.367E-01    1.578E-01

 v1             v(5)         i(v1)
2.500E+00      6.072E+00    7.174E-05
```

[6] `.print dc` statements intended to display each resistor's voltage in order, followed by the test point voltage (with respect to ground) at node 5, followed by source current.

7.3 Parallel resistor circuits

Parallel components by definition share one pair of electrically distinct points. Because of this, parallel-connected components share the same (continuous) voltage value at any given moment in time.

Parallel connections are specified in SPICE by each component bearing the same pair of node numbers. For example, a partial netlist specifying three 1 kΩ resistors connected in parallel with each other might look like this:

Partial SPICE netlist:

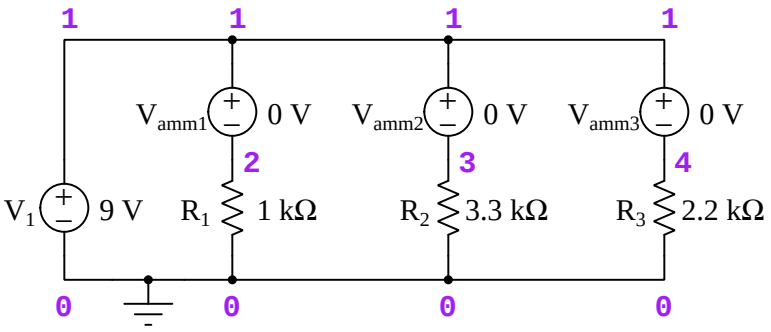
```
* Three parallel-connected resistors
r1 7 2 1000
r2 7 2 1000
r3 7 2 1000
```

Beginning at node 7, we encounter the three resistors R_1, R_2, and R_3. All three resistors end at the final node 2.

Legacy versions of SPICE do not support the display of current values through any other component type than a voltage source, and so in order to analyze parallel circuit branch currents we must insert "dummy" voltage sources with 0 Volt values to serve as "ammeters". These "dummy" voltage sources, impressing zero voltage into the circuit, do not affect the circuit's operation at all.

7.3.1 One DC voltage source, three resistors

Circuit schematic diagram (with node numbers listed):



SPICE netlist:[7]

```
* SPICE circuit
v1 1 0 dc 9
r1 2 0 1000
r2 3 0 3300
r3 4 0 2200
vamm1 1 2 0
vamm2 1 3 0
vamm3 1 4 0
.dc v1 9 9 1
.print dc i(vamm1) i(vamm2) i(vamm3)
.end
```

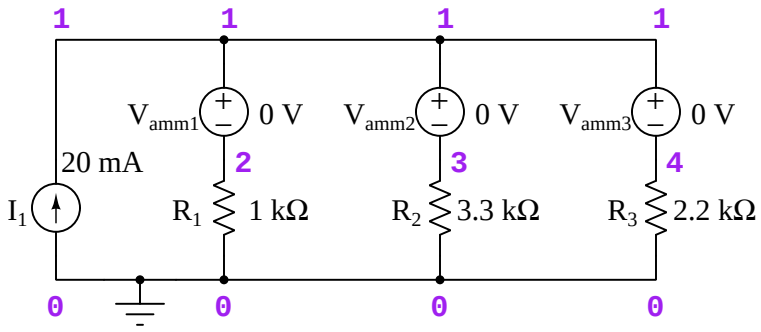
SPICE version 2G6 analysis (edited for brevity):

```
  v1           i(vamm1)    i(vamm2)    i(vamm3)
9.000E+00      9.000E-03   2.727E-03   4.091E-03
```

[7] Legacy versions of SPICE are only able to display current through a voltage source, and so this netlist contains three "dummy" voltage sources of 0 Volts each to serve as ammeters. The `.print dc` statement is intended to display each resistor's current in order.

7.3.2 One DC current source, three resistors

Circuit schematic diagram (with node numbers listed):



SPICE netlist:[8]

```
* SPICE circuit
i1 0 1 dc 20e-3
r1 2 0 1000
r2 3 0 3300
r3 4 0 2200
vamm1 1 2 0
vamm2 1 3 0
vamm3 1 4 0
.dc i1 20e-3 20e-3 1
.print dc i(vamm1) i(vamm2) i(vamm3)
.end
```

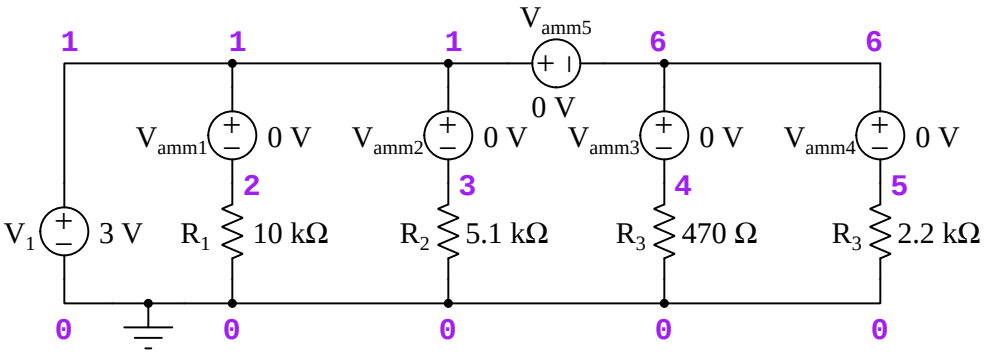
SPICE version 2G6 analysis (edited for brevity):

```
  i1            i(vamm1)     i(vamm2)     i(vamm3)
2.000E-02      1.138E-02    3.448E-03    5.172E-03
```

[8] Legacy versions of SPICE are only able to display current through a voltage source, and so this netlist contains three "dummy" voltage sources of 0 Volts each to serve as ammeters. The `.print dc` statement is intended to display each resistor's current in order.

7.3.3 One DC voltage source, four resistors

Circuit schematic diagram (with node numbers listed):



SPICE netlist:[9]

```
* SPICE circuit
v1 1 0 dc 3
r1 2 0 10e3
r2 3 0 5.1e3
r3 4 0 470
r4 5 0 2.2e3
vamm1 1 2 0
vamm2 1 3 0
vamm3 6 4 0
vamm4 6 5 0
vamm5 1 6 0
.dc v1 3 3 1
.print dc i(vamm1) i(vamm2) i(vamm3) i(vamm4)
.print dc i(vamm5) i(v1)
.end
```

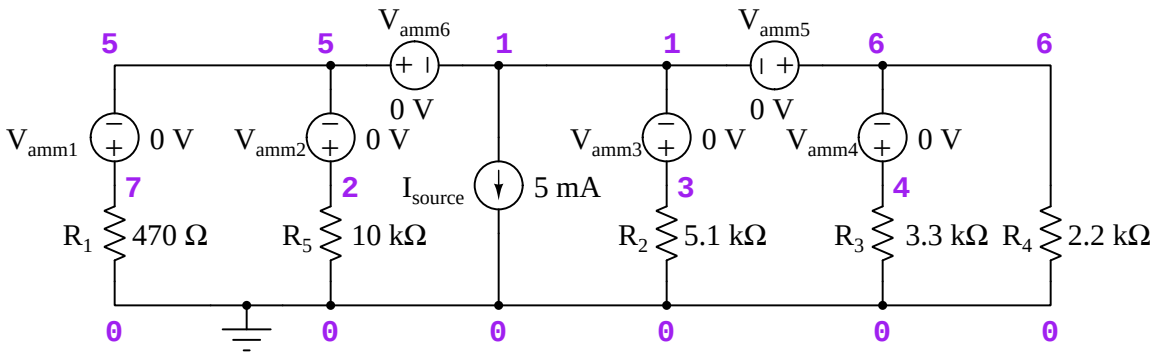
[9] Legacy versions of SPICE are only able to display current through a voltage source, and so this netlist contains five "dummy" voltage sources of 0 Volts each to serve as ammeters. The `.print dc` statement is intended to display each resistor's current in order, followed by the current of the last two branches, followed by source current.

SPICE version 2G6 analysis (edited for brevity):

```
 v1           i(vamm1)    i(vamm2)    i(vamm3)    i(vamm4)
3.000E+00     3.000E-04   5.882E-04   6.383E-03   1.364E-03

 v1           i(vamm5)    i(v1)
3.000E+00     7.747E-03  -8.635E-03
```

7.3.4 One DC current source, five resistors

Circuit schematic diagram (with node numbers listed):



SPICE netlist:[10]

```
* SPICE circuit
isource 1 0 dc 5m
r1 7 0 470
r2 3 0 5.1e3
r3 4 0 3.3k
r4 6 0 2.2e3
r5 2 0 10k
vamm1 7 5 0
vamm2 2 5 0
vamm3 3 1 0
vamm4 4 6 0
vamm5 6 1 0
vamm6 5 1 0
.dc isource 5m 5m 1
.print dc i(vamm1) i(vamm2) i(vamm3) i(vamm4)
.print dc i(vamm5) i(vamm6) v(1)
.end
```

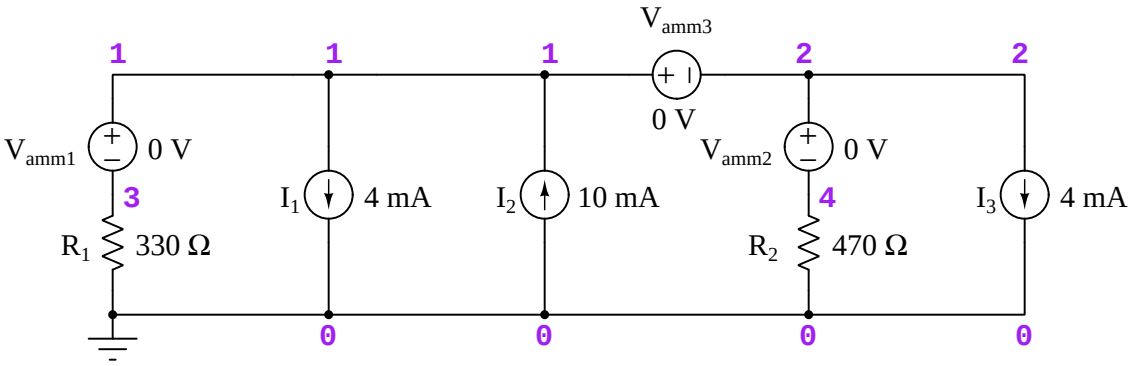
[10] Legacy versions of SPICE are only able to display current through a voltage source, and so this netlist contains six "dummy" voltage sources of 0 Volts each to serve as ammeters. The `.print dc` statement is intended to display each ammeter's current in order, followed by source voltage.

SPICE version 2G6 analysis (edited for brevity):

```
 i1           i(vamm1)    i(vamm2)    i(vamm3)    i(vamm4)
5.000E-03     3.344E-03   1.572E-04   3.082E-04   4.763E-04

 i1           i(vamm5)    i(vamm6)    v(1)
5.000E-03     1.191E-03   3.501E-03  -1.572E+00
```

7.3.5 Three DC current sources, two resistors

Circuit schematic diagram (with node numbers listed):



SPICE netlist:[11]

```
* SPICE circuit
i1 1 0 dc 4m
i2 0 1 dc 10m
i3 2 0 dc 4m
r1 3 0 330
r2 4 0 470
vamm1 1 3 0
vamm2 2 4 0
vamm3 1 2 0
.dc i1 4m 4m 1
.print dc i(vamm1) i(vamm2) i(vamm3)
.end
```

SPICE version 2G6 analysis (edited for brevity):

```
  i1           i(vamm1)    i(vamm2)    i(vamm3)
4.000E-03     1.175E-03   8.250E-04   4.825E-03
```

[11] Legacy versions of SPICE are only able to display current through a voltage source, and so this netlist contains three "dummy" voltage sources of 0 Volts each to serve as ammeters. The `.print dc` statement is intended to display each ammeter's current in order.

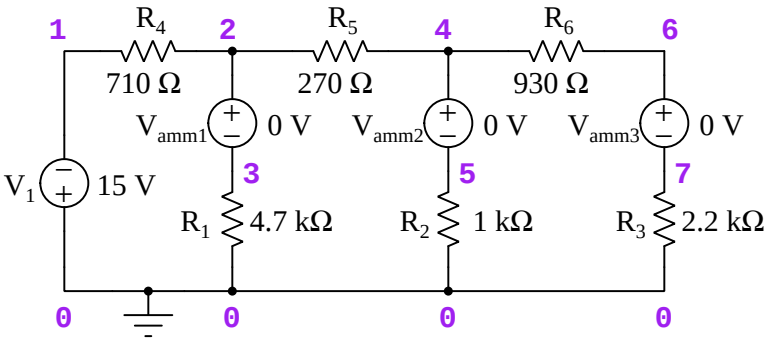
7.4 Series-parallel resistor circuits

Series-parallel circuits contain both series-connected and parallel-connected sub-networks.

Legacy versions of SPICE do not support the display of current values through any other component type than a voltage source, and so in order to analyze parallel circuit branch currents we must insert "dummy" voltage sources with 0 Volt values to serve as "ammeters". These "dummy" voltage sources, impressing zero voltage into the circuit, do not affect the circuit's operation at all.

7.4.1 One DC voltage source, six resistors

Circuit schematic diagram (with node numbers listed):



SPICE netlist:[12]

```
* SPICE circuit
v1 0 1 dc 15
r1 3 0 4700
r2 5 0 1000
r3 7 0 2200
r4 1 2 710
r5 2 4 270
r6 4 6 930
vamm1 2 3 0
vamm2 4 5 0
vamm3 6 7 0
.dc v1 15 15 1
.print dc v(3) v(5) v(7)
.print dc v(1,2) v(2,4) v(4,6)
.print dc i(vamm1) i(vamm2) i(vamm3)
.print dc i(v1)
.end
```

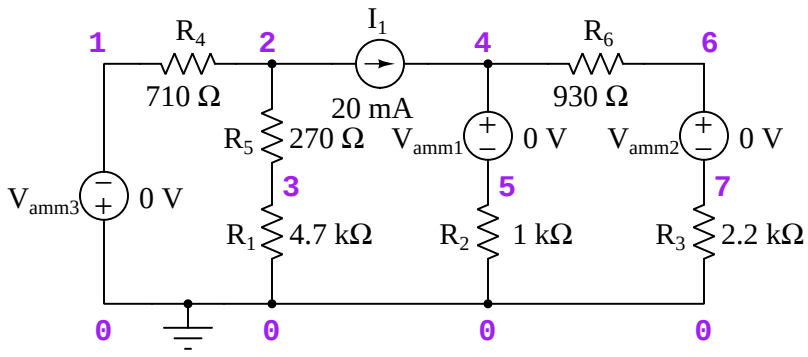
[12] Legacy versions of SPICE are only able to display current through a voltage source, and so this netlist contains three "dummy" voltage sources of 0 Volts each to serve as ammeters. The `.print dc` statement is intended to display each resistor's current in order.

SPICE version 2G6 analysis (edited for brevity):

```
 v1           v(3)        v(5)        v(7)
1.500E+01    -8.144E+00  -6.005E+00  -4.221E+00

 v1           v(1,2)      v(2,4)      v(4,6)
1.500E+01    -6.856E+00  -2.139E+00  -1.784E+00

 v1           i(vamm1)    i(vamm2)    i(vamm3)
1.500E+01    -1.733E-03  -6.005E-03  -1.918E-03

 v1           i(v1)
1.500E+01    -9.656E-03
```

7.4.2 One DC current source, six resistors

Circuit schematic diagram (with node numbers listed):



SPICE netlist:[13]

```
* SPICE circuit
i1 2 4 dc 20m
r1 3 0 4700
r2 5 0 1000
r3 7 0 2200
r4 1 2 710
r5 2 3 270
r6 4 6 930
vamm1 4 5 0
vamm2 6 7 0
vamm3 0 1 0
.dc i1 20m 20m 1
.print dc v(3) v(5) v(7)
.print dc v(1,2) v(3,2) v(4,6)
.print dc i(vamm1) i(vamm2) i(vamm3)
.print dc v(4,2)
.end
```

[13] Legacy versions of SPICE are only able to display current through a voltage source, and so this netlist contains three "dummy" voltage sources of 0 Volts each to serve as ammeters. The `.print dc` statement is intended to display each resistor's current in order.

SPICE version 2G6 analysis (edited for brevity):

```
 i1           v(3)        v(5)        v(7)
2.000E-02    -1.175E+01   1.516E+01   1.065E+01

 i1           v(1,2)      v(3,2)      v(4,6)
2.000E-02     1.242E+01   6.750E-01   4.504E+00

 i1           i(vamm1)    i(vamm2)    i(vamm3)
2.000E-02     1.516E-02   4.843E-03   1.750E-02

 i1           v(4,2)
2.000E-02     2.758E+01
```

7.5 Simple multi-source resistor circuits

The following circuits contain multiple voltage and/or current sources, but are considered "simple" in that no network theorems or analytical techniques involving simultaneous systems of linear equations are necessary to solve for all voltages, currents, and powers. In fact, each of these circuits is completely solvable using nothing but the following principles:

- Ohm's Law
- Kirchhoff's Voltage Law
- Kirchhoff's Current Law
- Properties of series circuits (e.g. equal currents)
- Properties of parallel circuits (e.g. equal voltages)

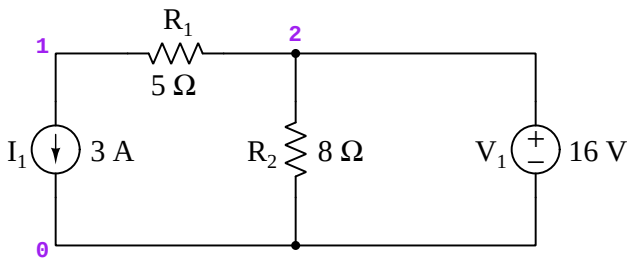
For this reason, I like to assign such circuits to students as a means of challenging and strengthening their understanding of these foundational principles. The following SPICE examples may be used as "practice problems" with answers given (by SPICE) for students learning to apply these all-important electrical principles.

An often-confusing convention in SPICE netlists is the specification of terminal nodes for voltage sources versus current sources. For a voltage source, the first node is the + terminal and the second node is the − terminal (e.g. `v1 3 8 dc 10` specifies a 10 Volt DC source with + on node 3 and − on node 8). For a current source, the first node is the incoming terminal and the second node is the outgoing terminal (e.g. `i1 2 6 dc 3` specifies a 3 Ampere DC source with current entering into the terminal connected to node 2 and exiting out of the terminal connected to node 6). Bear this in mind while studying these mixed-source networks!

Also bear in mind the fact that a component typically called a "source" may in fact function as an electrical load if being overpowered by some other source in a circuit. The final determination of the component's role as either source or load cannot be made until both voltage polarity and current direction have been identified for that component. All we can be sure of for a voltage "source" is that it will maintain the specified voltage across its terminals, and all we can be sure of for a current "source" is that it will maintain the specified current through itself.

7.5.1 One voltage, one current

Circuit schematic diagram (with node numbers listed):



SPICE netlist:

```
* SPICE circuit
i1 1 0 dc 3
v1 2 0 dc 16
r1 1 2 5
r2 2 0 8
.dc i1 3 3 1
.print dc v(1) v(1,2) v(2)
.print dc i(v1)
.end
```

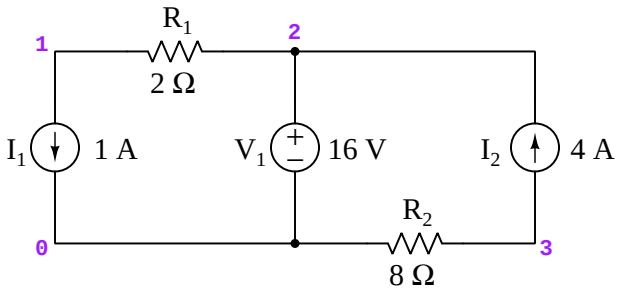
SPICE version 2G6 analysis (edited for brevity):

```
   i1            v(1)         v(1,2)       v(2)
3.000E+00     1.000E+00  -1.500E+01    1.600E+01

   i1            i(v1)
3.000E+00    -5.000E+00
```

R_1 drops 15 Volts (+ on node 2 and − on node 1 which explains why `v(1,2)` registers as a negative voltage) and R_2 drops 16 Volts (+ on node 2 and − on node 0). Voltage source V_1 passes 5 Amperes of current in an upward direction (the −5 value output by SPICE follows "passive sign convention" where a negative current means that component is acting as an electrical source). Current source I_1 drops 1 Volt (+ on node 1 and − on node 0), which means it is actually functioning as an electrical *load* in this circuit, being "overpowered" by source V_1.

7.5.2 Two currents, one voltage

Circuit schematic diagram (with node numbers listed):



SPICE netlist:

```
* SPICE circuit
i1 1 0 dc 1
i2 3 2 dc 4
v1 2 0 dc 16
r1 1 2 2
r2 3 0 8
.dc i1 1 1 1
.print dc v(1) v(1,2) v(2,3) v(3)
.print dc i(v1)
.end
```

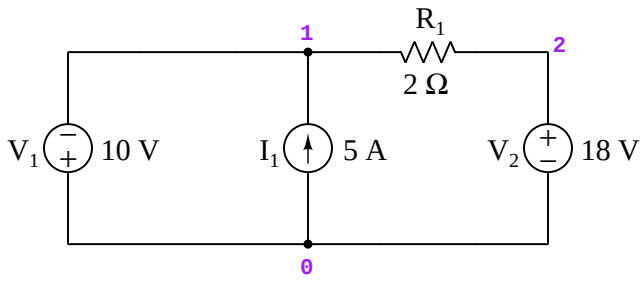
SPICE version 2G6 analysis (edited for brevity):

```
   i1            v(1)         v(1,2)       v(2,3)       v(3)
1.000E+00     1.400E+01   -2.000E+00    4.800E+01   -3.200E+01

   i1            i(v1)
1.000E+00     3.000E+00
```

I_1 drops 14 Volts and acts as a *load*. R_1 drops 2 Volts. R_2 drops 32 Volts. I_2 drops 48 Volts and acts as a *source*. V_1 passes 3 Amperes and acts as a *load*.

7.5.3 Two voltages, one current

Circuit schematic diagram (with node numbers listed):



SPICE netlist:

```
* SPICE circuit
v1 0 1 dc 10
v2 2 0 dc 18
i1 0 1 dc 5
r1 1 2 2
.dc i1 5 5 1
.print dc v(1) v(1,2) v(2)
.print dc i(v1)
.end
```

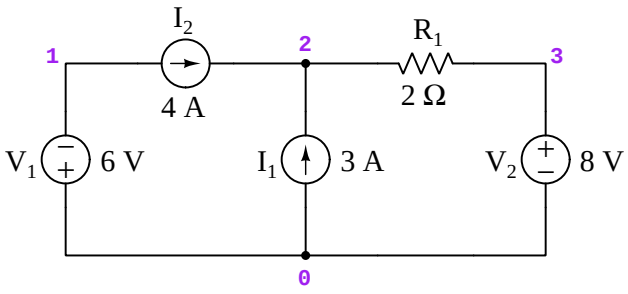
SPICE version 2G6 analysis (edited for brevity):

```
   i1           v(1)         v(1,2)       v(2)
5.000E+00    -1.000E+01   -2.800E+01    1.800E+01

   i1           i(v1)        i(v2)
5.000E+00    -1.900E+01   -1.400E+01
```

V_1 passes 19 Amperes of current and V_2 passes 14 Amperes of current, both acting as *sources*. I_1 drops 10 Volts and acts as a *load*. R_2 drops 28 Volts.

7.5.4 Two currents, two voltages

Circuit schematic diagram (with node numbers listed):



SPICE netlist:

```
* SPICE circuit
v1 0 1 dc 6
v2 3 0 dc 8
i1 0 2 dc 3
i2 1 2 dc 4
r1 2 3 2
.dc v1 6 6 1
.print dc v(2,1) v(2) v(2,3)
.print dc i(v1) i(v2)
.end
```

SPICE version 2G6 analysis (edited for brevity):

```
   v1           v(2,1)       v(2)         v(2,3)
6.000E+00     2.800E+01    2.200E+01    1.400E+01

   v1           i(v1)        i(v2)
6.000E+00     4.000E+00    7.000E+00
```

V_1 passes 4 Amperes and V_2 passes 7 Amperes, both acting as *loads*. I_1 drops 22 Volts and I_2 drops 28 Volts, both acting as *sources*. R_1 drops 14 Volts.

7.6 Resistive bridge circuits

Legacy versions of SPICE do not support the display of current values through any other component type than a voltage source, and so in order to analyze parallel circuit branch currents we must insert "dummy" voltage sources with 0 Volt values to serve as "ammeters". These "dummy" voltage sources, impressing zero voltage into the circuit, do not affect the circuit's operation at all.

Appendix A

Problem-Solving Strategies

The ability to solve complex problems is arguably one of the most valuable skills one can possess, and this skill is particularly important in any science-based discipline.

- Study principles, not procedures. Don't be satisfied with merely knowing how to compute solutions – learn *why* those solutions work.
- Identify what it is you need to solve, identify all relevant data, identify all units of measurement, identify any general principles or formulae linking the given information to the solution, and then identify any "missing pieces" to a solution. Annotate all diagrams with this data.
- Sketch a diagram to help visualize the problem. When building a real system, always devise a plan for that system and analyze its function *before* constructing it.
- Follow the units of measurement and meaning of every calculation. If you are ever performing mathematical calculations as part of a problem-solving procedure, and you find yourself unable to apply each and every intermediate result to some aspect of the problem, it means you don't understand what you are doing. Properly done, every mathematical result should have practical meaning for the problem, and not just be an abstract number. You should be able to identify the proper units of measurement for each and every calculated result, and show where that result fits into the problem.
- Perform "thought experiments" to explore the effects of different conditions for theoretical problems. When troubleshooting real systems, perform *diagnostic tests* rather than visually inspecting for faults, the best diagnostic test being the one giving you the most information about the nature and/or location of the fault with the fewest steps.
- Simplify the problem until the solution becomes obvious, and then use that obvious case as a model to follow in solving the more complex version of the problem.
- Check for exceptions to see if your solution is incorrect or incomplete. A good solution will work for *all* known conditions and criteria. A good example of this is the process of testing scientific hypotheses: the task of a scientist is not to find support for a new idea, but rather to *challenge* that new idea to see if it holds up under a battery of tests. The philosophical

principle of *reductio ad absurdum* (i.e. disproving a general idea by finding a specific case where it fails) is useful here.

- Work "backward" from a hypothetical solution to a new set of given conditions.
- Add quantities to problems that are qualitative in nature, because sometimes a little math helps illuminate the scenario.
- Sketch graphs illustrating how variables relate to each other. These may be quantitative (i.e. with realistic number values) or qualitative (i.e. simply showing increases and decreases).
- Treat quantitative problems as qualitative in order to discern the relative magnitudes and/or directions of change of the relevant variables. For example, try determining what happens if a certain variable were to increase or decrease before attempting to precisely calculate quantities: how will each of the dependent variables respond, by increasing, decreasing, or remaining the same as before?
- Consider limiting cases. This works especially well for qualitative problems where you need to determine which direction a variable will change. Take the given condition and magnify that condition to an extreme degree as a way of simplifying the direction of the system's response.
- Check your work. This means regularly testing your conclusions to see if they make sense. This does *not* mean repeating the same steps originally used to obtain the conclusion(s), but rather to use some other means to check validity. Simply repeating procedures often leads to *repeating the same errors* if any were made, which is why alternative paths are better.

Appendix B

Instructional philosophy

B.1 First principles of learning

- **Anyone can learn anything** given appropriate time, effort, resources, challenges, encouragement, and expectations. Dedicating time and investing effort are the student's responsibility; providing resources, challenges, and encouragement are the teacher's responsibility; high expectations are a responsibility shared by both student and teacher.

- **Transfer is not automatic.** The human mind has a natural tendency to compartmentalize information, which means the process of taking knowledge learned in one context and applying it to another usually does not come easy and therefore should never be taken for granted.

- **Learning is iterative.** The human mind rarely learns anything perfectly on the first attempt. Anticipate mistakes and plan for multiple tries to achieve full understanding, using the lessons of those mistakes as feedback to guide future attempts.

- **Information is absorbed, but understanding is created.** Facts and procedures may be memorized easily enough by repeated exposure, but the ability to reliably apply principles to novel scenarios only comes through intense personal effort. This effort is fundamentally creative in nature: casting new concepts in one's own words, running mental experiments to perceive patterns and trends, etc. These acts of making knowledge "one's own" need not be perfect in order to be effective, as the value lies in the activity and not necessarily the finished product. Depth of understanding and permanence of memory are both directly related to mental effort during the learning process – i.e. *easy learning is shallow and fleeting.*

- **Education trumps training.** There is no such thing as an entirely isolated subject, as all fields of knowledge are connected. Training is narrowly-focused and task-oriented. Education is broad-based and principle-oriented. When preparing for a life-long technical career, education beats training every time.

- **Character matters.** Poor habits are more destructive than deficits of knowledge or skill. This is especially true in collective endeavors, where a team's ability to function depends on trust between its members. Simply put, no one wants an untrustworthy person on their team. An essential component of education then, is character development.

- **People learn to be responsible by bearing responsibility.** An irresponsible person is someone who has never *had* to be responsible for anything that mattered enough to them. Just as anyone can learn anything, anyone can become responsible if the personal cost of irresponsibility becomes high enough.

- **What gets measured, gets done.** Accurate and relevant assessment of learning is key to ensuring all students learn. Therefore, it is imperative to measure what matters.

- **Failure is nothing to fear.** Every human being fails, and fails in multiple ways at multiple times. Eventual success only happens when we don't stop trying.

B.2 Proven strategies for instructors

- Assume every student is capable of learning anything they desire given the proper conditions. Treat them as capable adults by granting real responsibility and avoiding artificial incentives such as merit or demerit points.
- Create a consistent culture of high expectations across the entire program of study. Demonstrate and encourage patience, persistence, and a healthy sense of self-skepticism. Anticipate and de-stigmatize error. Teach respect for the capabilities of others as well as respect for one's own fallibility.
- Replace lecture with "inverted" instruction, where students first encounter new concepts through reading and then spend class time in Socratic dialogue with the instructor exploring those concepts and solving problems individually. There is a world of difference between observing someone solve a problem versus actually solving a problem yourself, and so the point of this form of instruction is to place students in a position where they *cannot* passively observe.
- Require students to read extensively, write about what they learn, and dialogue with you and their peers to sharpen their understanding. Apply Francis Bacon's advice that "reading maketh a full man; conference a ready man; and writing an exact man". These are complementary activities helping students expand their confidence and abilities.
- Use artificial intelligence (AI) to challenge student understanding rather than merely provide information. Find productive ways for AI to critique students' clarity of thought and of expression, for example by employing AI as a Socratic-style interlocutor or as a reviewer of students' journals. Properly applied, AI has the ability to expand student access to critical review well outside the bounds of their instructor's reach.
- Build frequent and rapid feedback into the learning process so that students know at all times how well they are learning, to identify problems early and fix them before they grow. Model the intellectual habit of self-assessing and self-correcting your own understanding (i.e. a cognitive *feedback loop*), encouraging students to do the same.
- Use "mastery" as the standard for every assessment, which means the exam or experiment or project must be done with 100% competence in order to pass. Provide students with multiple opportunity for re-tries (different versions of the assessment every time).
- Require students to devise their own hypotheses and procedures on all experiments, so that the process is truly a scientific one. Have students assess their proposed experimental procedures for risk and devise mitigations for those risks. Let nothing be pre-designed about students' experiments other than a stated task (i.e. what principle the experiment shall test) at the start and a set of demonstrable knowledge and skill objectives at the end.
- Have students build as much of their lab equipment as possible: building power sources, building test assemblies[1], and building complete working systems (no kits!). In order to provide

[1] In the program I teach, every student builds their own "Development Board" consisting of a metal chassis with DIN rail, terminal blocks, and an AC-DC power supply of their own making which functions as a portable lab environment they can use at school as well as take home.

this same "ground-up" experience for every new student, this means either previous students take their creations with them, or the systems get disassembled in preparation for the new students, or the systems grow and evolve with each new student group.

- Incorporate external accountability for you and for your students, continuously improving the curriculum and your instructional methods based on proven results. Have students regularly network with active professionals through participation in advisory committee meetings, service projects, tours, jobshadows, internships, etc. Practical suggestions include requiring students to design and build projects for external clients (e.g. community groups, businesses, different departments within the institution), and also requiring students attend all technical advisory committee meetings and to dialogue with the industry representatives at those meetings.
- Repeatedly explore difficult-to-learn concepts across multiple courses, so that students have multiple opportunities to build their understanding.
- Relate all new concepts, whenever possible, to previous concepts and to relevant physical laws. Challenge each and every student, every day, to *reason* from concept to concept and to explain the logical connections between. Challenge students to verify their conclusions by multiple approaches (e.g. double-checking their work using different methods). Ask *"Why?"* often.
- Maintain detailed records on each student's performance and share these records privately with them. These records should include academic performance as well as professionally relevant behavioral tendencies.
- Hold mandatory "check-in" meetings between all program faculty and each new student during their first term. Offer these to all other students as an option, except for any students continuing to manifest unprofessional behaviors, poor academic performance, or who have some other need for a face-to-face meeting with faculty.
- Address problems while they are small, before they grow larger. This is equally true for tutoring technical concepts as it is for helping students build professional habits.
- Build rigorous quality control into the curriculum to ensure every student masters every important concept, and that the mastery is retained over time. This includes (1) review questions added to every exam to re-assess knowledge taught in previous terms, (2) cumulative exams at the end of every term to re-assess all important concepts back to the very beginning of the program, and (3) review assessments in practical (hands-on) coursework to ensure critically-important skills were indeed taught and are still retained. What you will find by doing this is that it actually boosts retention of students by ensuring that important knowledge gets taught and is retained over long spans of time. In the absence of such quality control, student learning and retention tends to be spotty and this contributes to drop-out and failure rates later in their education.
- Finally, *never rush learning.* Education is not a race. Give your students ample time to digest complex ideas, as you continually remind yourself of just how long it took you to achieve mastery! Long-term retention and the consistently correct application of concepts are always the result of *focused effort over long periods of time* which means there are no shortcuts to learning.

B.3 Proven strategies for students

The single most important piece of advice I have for any student of any subject is to take responsibility for your own development in all areas of life including mental development. Expecting others in your life to entirely guide your own development is a recipe for disappointment. This is just as true for students enrolled in formal learning institutions as it is for auto-didacts pursuing learning entirely on their own. Learning to think in new ways is key to being able to gainfully use information, to make informed decisions about your life, and to best serve those you care about. With this in mind, I offer the following advice to students:

- **Approach all learning as valuable**. No matter what course you take, no matter who you learn from, no matter the subject, there is something useful in every learning experience. If you don't see the value of every new experience, you are not looking closely enough!
- **Continually challenge yourself**. Let other people take shortcuts and find easy answers to easy problems. The purpose of education is to stretch your mind, in order to shape it into a more powerful tool. This doesn't come by taking the path of least resistance. An excellent analogy for an empowering education is productive physical exercise: becoming stronger, more flexible, and more persistent only comes through intense personal effort.
- **Master the use of language**. This includes reading extensively, writing every day, listening closely, and speaking articulately. To a great extent language channels and empowers thought, so the better you are at wielding language the better you will be at grasping abstract concepts and articulating them not only for your benefit but for others as well.
- **Do not limit yourself to the resources given to you**. Read books that are not on the reading list. Run experiments that aren't assigned to you. Form study groups outside of class. Take an entrepreneurial approach to your own education, as though it were a business you were building for your future benefit.
- **Express and share what you learn**. Take every opportunity to teach what you have learned to others, as this will not only help them but will also strengthen your own understanding[2].
- Realize that **no one can give you understanding**, just as no one can give you physical fitness. These both must be *built*.
- **Above all, recognize that learning is hard work, and that a certain level of frustration is unavoidable**. There are times when you will struggle to grasp some of these concepts, and that struggle is a natural thing. Take heart that it will yield with persistent and varied[3] effort, and never give up! That concepts don't immediately come to you is not a sign of something wrong, but rather of something right: that you have found a worthy challenge!

[2] On a personal note, I was surprised to learn just how much my own understanding of electronics and related subjects was strengthened by becoming a teacher. When you are tasked every day with helping other people grasp complex topics, it catalyzes your own learning by giving you powerful incentives to study, to articulate your thoughts, and to reflect deeply on the process of learning.

[3] As the old saying goes, "Insanity is trying the same thing over and over again, expecting different results." If you find yourself stumped by something in the text, you should attempt a different approach. Alter the thought experiment, change the mathematical parameters, do whatever you can to see the problem in a slightly different light, and then the solution will often present itself more readily.

B.4 Design of these learning modules

"The unexamined circuit is not worth energizing" – Socrates (if he had taught electricity)

These learning modules, although useful for self-study, were designed to be used in a formal learning environment where a subject-matter expert challenges students to digest the content and exercise their critical thinking abilities in the answering of questions and in the construction and testing of working circuits. Every effort has been made to embed the following instructional and assessment philosophies within:

- The first goal of education is to enhance clear and independent thought, in order that every student reach their fullest potential in a highly complex and inter-dependent world. Robust reasoning is *always* more important than particulars of any subject matter, because its application is universal.
- Literacy is fundamental to independent learning and thought because text continues to be the most efficient way to communicate complex ideas over space and time. Those who cannot read with ease are limited in their ability to acquire knowledge and perspective.
- Articulate communication is fundamental to work that is complex and interdisciplinary.
- Faulty assumptions and poor reasoning are best corrected through challenge, not presentation. The rhetorical technique of *reductio ad absurdum* (disproving an assertion by exposing an absurdity) works well to discipline student's minds, not only to correct the problem at hand but also to learn how to detect and correct future errors.
- Important principles should be repeatedly explored and widely applied throughout a course of study, not only to reinforce their importance and help ensure their mastery, but also to showcase the interconnectedness and utility of knowledge.

These learning modules were expressly designed to be used in an "inverted" teaching environment[4] where students first read the introductory and tutorial chapters on their own, then individually attempt to answer the questions and construct working circuits according to the experiment and project guidelines. The instructor never lectures, but instead meets regularly with each individual student to review their progress, answer questions, identify misconceptions, and challenge the student to new depths of understanding through further questioning. Regular meetings between instructor and student should resemble a Socratic[5] dialogue, where questions serve as scalpels to dissect topics and expose assumptions. The student passes each module only after consistently demonstrating their ability to logically analyze and correctly apply all major concepts in each question or project/experiment. The instructor must be vigilant in probing each student's understanding to ensure they are truly *reasoning* and not just *memorizing*. This is why "Challenge" points appear throughout, as prompts for students to think deeper about topics and as starting points for instructor queries. Sometimes these challenge points require additional knowledge that hasn't been covered in the series to answer in full. This is okay, as the major purpose of the Challenges is to stimulate analysis and synthesis on the part of each student.

The instructor must possess enough mastery of the subject matter and awareness of students' reasoning to generate their own follow-up questions to practically any student response. Even completely correct answers given by the student should be challenged by the instructor for the purpose of having students practice articulating their thoughts and defending their reasoning. Conceptual errors committed by the student should be exposed and corrected not by direct instruction, but rather by reducing the errors to an absurdity[6] through well-chosen questions and thought experiments posed by the instructor. Becoming proficient at this style of instruction requires time and dedication, but the positive effects on critical thinking for both student and instructor are spectacular.

An inspection of these learning modules reveals certain unique characteristics. One of these is a bias toward thorough explanations in the tutorial chapters. Without a live instructor to explain concepts and applications to students, the text itself must fulfill this role. This philosophy results in lengthier explanations than what you might typically find in a textbook, each step of the reasoning process fully explained, including footnotes addressing common questions and concerns students raise while learning these concepts. Each tutorial seeks to not only explain each major concept in sufficient detail, but also to explain the logic of each concept and how each may be developed

[4] In a traditional teaching environment, students first encounter new information via *lecture* from an expert, and then independently apply that information via *homework*. In an "inverted" course of study, students first encounter new information via *homework*, and then independently apply that information under the scrutiny of an expert. The expert's role in lecture is to simply *explain*, but the expert's role in an inverted session is to *challenge*, *critique*, and if necessary *explain* where gaps in understanding still exist.

[5] Socrates is a figure in ancient Greek philosophy famous for his unflinching style of questioning. Although he authored no texts, he appears as a character in Plato's many writings. The essence of Socratic philosophy is to leave no question unexamined and no point of view unchallenged. While purists may argue a topic such as electric circuits is too narrow for a true Socratic-style dialogue, I would argue that the essential thought processes involved with scientific reasoning on *any* topic are not far removed from the Socratic ideal, and that students of electricity and electronics would do very well to challenge assumptions, pose thought experiments, identify fallacies, and otherwise employ the arsenal of critical thinking skills modeled by Socrates.

[6] This rhetorical technique is known by the Latin phrase *reductio ad absurdum*. The concept is to expose errors by counter-example, since only one solid counter-example is necessary to disprove a universal claim. As an example of this, consider the common misconception among beginning students of electricity that voltage cannot exist without current. One way to apply *reductio ad absurdum* to this statement is to ask how much current passes through a fully-charged battery connected to nothing (i.e. a clear example of voltage existing without current).

from "first principles". Again, this reflects the goal of developing clear and independent thought in students' minds, by showing how clear and logical thought was used to forge each concept. Students benefit from witnessing a model of clear thinking in action, and these tutorials strive to be just that.

Another feature of these learning modules is that they do not treat topics in isolation. Rather, important concepts are introduced early in the series, and appear repeatedly as stepping-stones toward other concepts in subsequent modules. This helps to avoid the "compartmentalization" of knowledge, demonstrating the inter-connectedness of concepts and simultaneously reinforcing them. Each module is fairly complete in itself, reserving the beginning of its tutorial to a review of foundational concepts.

To high standards of education,

Tony R. Kuphaldt

Appendix C

Tools used

I am indebted to the developers of many open-source software applications in the creation of these learning modules. The following is a list of these applications with some commentary on each.

You will notice a theme common to many of these applications: a bias toward *code*. Although I am by no means an expert programmer in any computer language, I understand and appreciate the flexibility offered by code-based applications where the user (you) enters commands into a plain ASCII text file, which the software then reads and processes to create the final output. Code-based computer applications are by their very nature *extensible*, while WYSIWYG (What You See Is What You Get) applications are generally limited to whatever user interface the developer makes for you.

The `GNU/Linux` computer operating system

> There is so much to be said about Linus Torvalds' `Linux` and Richard Stallman's `GNU` project. First, to credit just these two individuals is to fail to do justice to the *mob* of passionate volunteers who contributed to make this amazing software a reality. I first learned of `Linux` back in 1996, and have been using this operating system on my personal computers almost exclusively since then. It is *free*, it is completely *configurable*, and it permits the continued use of highly efficient `Unix` applications and scripting languages (e.g. shell scripts, Makefiles, `sed`, `awk`) developed over many decades. `Linux` not only provided me with a powerful computing platform, but its open design served to inspire my life's work of creating open-source educational resources.

Bram Moolenaar's `Vim` text editor

> Writing code for any code-based computer application requires a *text editor*, which may be thought of as a word processor strictly limited to outputting plain-ASCII text files. Many good text editors exist, and one's choice of text editor seems to be a deeply personal matter within the programming world. I prefer `Vim` because it operates very similarly to `vi` which is ubiquitous on `Unix`/`Linux` operating systems, and because it may be entirely operated via keyboard (i.e. no mouse required) which makes it fast to use.

Donald Knuth's TEX typesetting system

Developed in the late 1970's and early 1980's by computer scientist extraordinaire Donald Knuth to typeset his multi-volume magnum opus *The Art of Computer Programming*, this software allows the production of formatted text for screen-viewing or paper printing, all by writing plain-text code to describe how the formatted text is supposed to appear. TEX is not just a markup language for documents, but it is also a Turing-complete programming language in and of itself, allowing useful algorithms to be created to control the production of documents. Simply put, TEX *is a programmer's approach to word processing*. Since TEX is controlled by code written in a plain-text file, this means anyone may read that plain-text file to see exactly how the document was created. This openness afforded by the code-based nature of TEX makes it relatively easy to learn how other people have created their own TEX documents. By contrast, examining a beautiful document created in a conventional WYSIWYG word processor such as Microsoft `Word` suggests nothing to the reader about *how* that document was created, or what the user might do to create something similar. As Mr. Knuth himself once quipped, conventional word processing applications should be called WYSIAYG (What You See Is *All* You Get).

Leslie Lamport's LATEX extensions to TEX

Like all true programming languages, TEX is inherently extensible. So, years after the release of TEX to the public, Leslie Lamport decided to create a massive extension allowing easier compilation of book-length documents. The result was LATEX, which is the markup language used to create all ModEL module documents. You could say that TEX is to LATEX as `C` is to `C++`. This means it is permissible to use any and all TEX commands within LATEX source code, and it all still works. Some of the features offered by LATEX that would be challenging to implement in TEX include automatic index and table-of-content creation.

Tim Edwards' `Xcircuit` drafting program

This wonderful program is what I use to create all the schematic diagrams and illustrations (but not photographic images or mathematical plots) throughout the ModEL project. It natively outputs PostScript format which is a true vector graphic format (this is why the images do not pixellate when you zoom in for a closer view), and it is so simple to use that I have never had to read the manual! Object libraries are easy to create for `Xcircuit`, being plain-text files using PostScript programming conventions. Over the years I have collected a large set of object libraries useful for drawing electrical and electronic schematics, pictorial diagrams, and other technical illustrations.

`Gimp` graphic image manipulation program

Essentially an open-source clone of Adobe's `PhotoShop`, I use `Gimp` to resize, crop, and convert file formats for all of the photographic images appearing in the ModEL modules. Although `Gimp` does offer its own scripting language (called `Script-Fu`), I have never had occasion to use it. Thus, my utilization of `Gimp` to merely crop, resize, and convert graphic images is akin to using a sword to slice bread.

`SPICE` circuit simulation program

`SPICE` is to circuit analysis as TEX is to document creation: it is a form of markup language designed to describe a certain object to be processed in plain-ASCII text. When the plain-text "source file" is compiled by the software, it outputs the final result. More modern circuit analysis tools certainly exist, but I prefer `SPICE` for the following reasons: it is *free*, it is *fast*, it is *reliable*, and it is a fantastic tool for *teaching* students of electricity and electronics how to write simple code. I happen to use rather old versions of `SPICE`, version 2g6 being my "go to" application when I only require text-based output. `NGSPICE` (version 26), which is based on Berkeley `SPICE` version 3f5, is used when I require graphical output for such things as time-domain waveforms and Bode plots. In all `SPICE` example netlists I strive to use coding conventions compatible with all `SPICE` versions.

Andrew D. Hwang's `ePiX` mathematical visualization programming library

This amazing project is a `C++` library you may link to any `C/C++` code for the purpose of generating PostScript graphic images of mathematical functions. As a completely free and open-source project, it does all the plotting I would otherwise use a Computer Algebra System (CAS) such as `Mathematica` or `Maple` to do. It should be said that `ePiX` is *not* a Computer Algebra System like `Mathematica` or `Maple`, but merely a mathematical *visualization* tool. In other words, it won't determine integrals for you (you'll have to implement that in your own `C/C++` code!), but it can graph the results, and it does so beautifully. What I really admire about `ePiX` is that it is a `C++` programming library, which means it builds on the existing power and toolset available with that programming language. Mr. Hwang could have probably developed his own stand-alone application for mathematical plotting, but by creating a `C++` library to do the same thing he accomplished something much greater.

`gnuplot` mathematical visualization software

Another open-source tool for mathematical visualization is `gnuplot`. Interestingly, this tool is *not* part of Richard Stallman's GNU project, its name being a coincidence. For this reason the authors prefer "gnu" *not* be capitalized at all to avoid confusion. This is a much "lighter-weight" alternative to a spreadsheet for plotting tabular data, and the fact that it easily outputs directly to an X11 console or a file in a number of different graphical formats (including PostScript) is very helpful. I typically set my `gnuplot` output format to default (X11 on my Linux PC) for quick viewing while I'm developing a visualization, then switch to PostScript file export once the visual is ready to include in the document(s) I'm writing. As with my use of `Gimp` to do rudimentary image editing, my use of `gnuplot` only scratches the surface of its capabilities, but the important points are that it's *free* and that it *works well*.

`Python` programming language

Both Python and C++ find extensive use in these modules as instructional aids and exercises, but I'm listing Python here as a *tool* for myself because I use it almost daily as a *calculator*. If you open a Python interpreter console and type `from math import *` you can type mathematical expressions and have it return results just as you would on a hand calculator. Complex-number (i.e. *phasor*) arithmetic is similarly supported if you include the complex-math library (`from cmath import *`). Examples of this are shown in the Programming References chapter (if included) in each module. Of course, being a fully-featured programming language, Python also supports conditionals, loops, and other structures useful for calculation of quantities. Also, running in a console environment where all entries and returned values show as text in a chronologically-ordered list makes it easy to copy-and-paste those calculations to document exactly how they were performed.

Appendix D

Creative Commons License

Creative Commons Attribution 4.0 International Public License

By exercising the Licensed Rights (defined below), You accept and agree to be bound by the terms and conditions of this Creative Commons Attribution 4.0 International Public License ("Public License"). To the extent this Public License may be interpreted as a contract, You are granted the Licensed Rights in consideration of Your acceptance of these terms and conditions, and the Licensor grants You such rights in consideration of benefits the Licensor receives from making the Licensed Material available under these terms and conditions.

Section 1 – Definitions.

a. **Adapted Material** means material subject to Copyright and Similar Rights that is derived from or based upon the Licensed Material and in which the Licensed Material is translated, altered, arranged, transformed, or otherwise modified in a manner requiring permission under the Copyright and Similar Rights held by the Licensor. For purposes of this Public License, where the Licensed Material is a musical work, performance, or sound recording, Adapted Material is always produced where the Licensed Material is synched in timed relation with a moving image.

b. **Adapter's License** means the license You apply to Your Copyright and Similar Rights in Your contributions to Adapted Material in accordance with the terms and conditions of this Public License.

c. **Copyright and Similar Rights** means copyright and/or similar rights closely related to copyright including, without limitation, performance, broadcast, sound recording, and Sui Generis Database Rights, without regard to how the rights are labeled or categorized. For purposes of this Public License, the rights specified in Section 2(b)(1)-(2) are not Copyright and Similar Rights.

d. **Effective Technological Measures** means those measures that, in the absence of proper authority, may not be circumvented under laws fulfilling obligations under Article 11 of the WIPO Copyright Treaty adopted on December 20, 1996, and/or similar international agreements.

e. **Exceptions and Limitations** means fair use, fair dealing, and/or any other exception or

limitation to Copyright and Similar Rights that applies to Your use of the Licensed Material.

f. **Licensed Material** means the artistic or literary work, database, or other material to which the Licensor applied this Public License.

g. **Licensed Rights** means the rights granted to You subject to the terms and conditions of this Public License, which are limited to all Copyright and Similar Rights that apply to Your use of the Licensed Material and that the Licensor has authority to license.

h. **Licensor** means the individual(s) or entity(ies) granting rights under this Public License.

i. **Share** means to provide material to the public by any means or process that requires permission under the Licensed Rights, such as reproduction, public display, public performance, distribution, dissemination, communication, or importation, and to make material available to the public including in ways that members of the public may access the material from a place and at a time individually chosen by them.

j. **Sui Generis Database Rights** means rights other than copyright resulting from Directive 96/9/EC of the European Parliament and of the Council of 11 March 1996 on the legal protection of databases, as amended and/or succeeded, as well as other essentially equivalent rights anywhere in the world.

k. **You** means the individual or entity exercising the Licensed Rights under this Public License. **Your** has a corresponding meaning.

Section 2 – Scope.

a. License grant.

1. Subject to the terms and conditions of this Public License, the Licensor hereby grants You a worldwide, royalty-free, non-sublicensable, non-exclusive, irrevocable license to exercise the Licensed Rights in the Licensed Material to:

A. reproduce and Share the Licensed Material, in whole or in part; and

B. produce, reproduce, and Share Adapted Material.

2. Exceptions and Limitations. For the avoidance of doubt, where Exceptions and Limitations apply to Your use, this Public License does not apply, and You do not need to comply with its terms and conditions.

3. Term. The term of this Public License is specified in Section 6(a).

4. Media and formats; technical modifications allowed. The Licensor authorizes You to exercise the Licensed Rights in all media and formats whether now known or hereafter created, and to make technical modifications necessary to do so. The Licensor waives and/or agrees not to assert any right or authority to forbid You from making technical modifications necessary to exercise the Licensed Rights, including technical modifications necessary to circumvent Effective Technological Measures.

For purposes of this Public License, simply making modifications authorized by this Section 2(a)(4) never produces Adapted Material.

5. Downstream recipients.

A. Offer from the Licensor – Licensed Material. Every recipient of the Licensed Material automatically receives an offer from the Licensor to exercise the Licensed Rights under the terms and conditions of this Public License.

B. No downstream restrictions. You may not offer or impose any additional or different terms or conditions on, or apply any Effective Technological Measures to, the Licensed Material if doing so restricts exercise of the Licensed Rights by any recipient of the Licensed Material.

6. No endorsement. Nothing in this Public License constitutes or may be construed as permission to assert or imply that You are, or that Your use of the Licensed Material is, connected with, or sponsored, endorsed, or granted official status by, the Licensor or others designated to receive attribution as provided in Section 3(a)(1)(A)(i).

b. Other rights.

1. Moral rights, such as the right of integrity, are not licensed under this Public License, nor are publicity, privacy, and/or other similar personality rights; however, to the extent possible, the Licensor waives and/or agrees not to assert any such rights held by the Licensor to the limited extent necessary to allow You to exercise the Licensed Rights, but not otherwise.

2. Patent and trademark rights are not licensed under this Public License.

3. To the extent possible, the Licensor waives any right to collect royalties from You for the exercise of the Licensed Rights, whether directly or through a collecting society under any voluntary or waivable statutory or compulsory licensing scheme. In all other cases the Licensor expressly reserves any right to collect such royalties.

Section 3 – License Conditions.

Your exercise of the Licensed Rights is expressly made subject to the following conditions.

a. Attribution.

1. If You Share the Licensed Material (including in modified form), You must:

A. retain the following if it is supplied by the Licensor with the Licensed Material:

i. identification of the creator(s) of the Licensed Material and any others designated to receive attribution, in any reasonable manner requested by the Licensor (including by pseudonym if designated);

ii. a copyright notice;

iii. a notice that refers to this Public License;

iv. a notice that refers to the disclaimer of warranties;

v. a URI or hyperlink to the Licensed Material to the extent reasonably practicable;

B. indicate if You modified the Licensed Material and retain an indication of any previous modifications; and

C. indicate the Licensed Material is licensed under this Public License, and include the text of, or the URI or hyperlink to, this Public License.

2. You may satisfy the conditions in Section 3(a)(1) in any reasonable manner based on the medium, means, and context in which You Share the Licensed Material. For example, it may be reasonable to satisfy the conditions by providing a URI or hyperlink to a resource that includes the required information.

3. If requested by the Licensor, You must remove any of the information required by Section 3(a)(1)(A) to the extent reasonably practicable.

4. If You Share Adapted Material You produce, the Adapter's License You apply must not prevent recipients of the Adapted Material from complying with this Public License.

Section 4 – Sui Generis Database Rights.

Where the Licensed Rights include Sui Generis Database Rights that apply to Your use of the Licensed Material:

a. for the avoidance of doubt, Section 2(a)(1) grants You the right to extract, reuse, reproduce, and Share all or a substantial portion of the contents of the database;

b. if You include all or a substantial portion of the database contents in a database in which You have Sui Generis Database Rights, then the database in which You have Sui Generis Database Rights (but not its individual contents) is Adapted Material; and

c. You must comply with the conditions in Section 3(a) if You Share all or a substantial portion of the contents of the database.

For the avoidance of doubt, this Section 4 supplements and does not replace Your obligations under this Public License where the Licensed Rights include other Copyright and Similar Rights.

Section 5 – Disclaimer of Warranties and Limitation of Liability.

a. Unless otherwise separately undertaken by the Licensor, to the extent possible, the Licensor offers the Licensed Material as-is and as-available, and makes no representations or warranties of any kind concerning the Licensed Material, whether express, implied, statutory, or other. This includes, without limitation, warranties of title, merchantability, fitness for a particular purpose, non-infringement, absence of latent or other defects, accuracy, or the presence or absence of errors,

whether or not known or discoverable. Where disclaimers of warranties are not allowed in full or in part, this disclaimer may not apply to You.

b. To the extent possible, in no event will the Licensor be liable to You on any legal theory (including, without limitation, negligence) or otherwise for any direct, special, indirect, incidental, consequential, punitive, exemplary, or other losses, costs, expenses, or damages arising out of this Public License or use of the Licensed Material, even if the Licensor has been advised of the possibility of such losses, costs, expenses, or damages. Where a limitation of liability is not allowed in full or in part, this limitation may not apply to You.

c. The disclaimer of warranties and limitation of liability provided above shall be interpreted in a manner that, to the extent possible, most closely approximates an absolute disclaimer and waiver of all liability.

Section 6 – Term and Termination.

a. This Public License applies for the term of the Copyright and Similar Rights licensed here. However, if You fail to comply with this Public License, then Your rights under this Public License terminate automatically.

b. Where Your right to use the Licensed Material has terminated under Section 6(a), it reinstates:

1. automatically as of the date the violation is cured, provided it is cured within 30 days of Your discovery of the violation; or

2. upon express reinstatement by the Licensor.

For the avoidance of doubt, this Section 6(b) does not affect any right the Licensor may have to seek remedies for Your violations of this Public License.

c. For the avoidance of doubt, the Licensor may also offer the Licensed Material under separate terms or conditions or stop distributing the Licensed Material at any time; however, doing so will not terminate this Public License.

d. Sections 1, 5, 6, 7, and 8 survive termination of this Public License.

Section 7 – Other Terms and Conditions.

a. The Licensor shall not be bound by any additional or different terms or conditions communicated by You unless expressly agreed.

b. Any arrangements, understandings, or agreements regarding the Licensed Material not stated herein are separate from and independent of the terms and conditions of this Public License.

Section 8 – Interpretation.

a. For the avoidance of doubt, this Public License does not, and shall not be interpreted to, reduce, limit, restrict, or impose conditions on any use of the Licensed Material that could lawfully

be made without permission under this Public License.

b. To the extent possible, if any provision of this Public License is deemed unenforceable, it shall be automatically reformed to the minimum extent necessary to make it enforceable. If the provision cannot be reformed, it shall be severed from this Public License without affecting the enforceability of the remaining terms and conditions.

c. No term or condition of this Public License will be waived and no failure to comply consented to unless expressly agreed to by the Licensor.

d. Nothing in this Public License constitutes or may be interpreted as a limitation upon, or waiver of, any privileges and immunities that apply to the Licensor or You, including from the legal processes of any jurisdiction or authority.

Creative Commons is not a party to its public licenses. Notwithstanding, Creative Commons may elect to apply one of its public licenses to material it publishes and in those instances will be considered the "Licensor." Except for the limited purpose of indicating that material is shared under a Creative Commons public license or as otherwise permitted by the Creative Commons policies published at `creativecommons.org/policies`, Creative Commons does not authorize the use of the trademark "Creative Commons" or any other trademark or logo of Creative Commons without its prior written consent including, without limitation, in connection with any unauthorized modifications to any of its public licenses or any other arrangements, understandings, or agreements concerning use of licensed material. For the avoidance of doubt, this paragraph does not form part of the public licenses.

Creative Commons may be contacted at `creativecommons.org`.

Appendix E

References

Vladimirescu, Andrei, *The Spice Book*, John Wiley & Sons, New York, 1994.

Appendix F

Version history

This is a list showing all significant additions, corrections, and other edits made to this learning module. Each entry is referenced by calendar date in reverse chronological order (newest version first), which appears on the front cover of every learning module for easy reference. Any contributors to this open-source document are listed here as well.

6 September 2024 – divided the Introduction chapter into sections, one with recommendations for students, one with a listing of challenging concepts, and one with recommendations for instructors. Also re-named the "Introduction" chapter to be called "What is SPICE?".

3 September 2020 – added zero voltage values to all the "vamm" dummy voltage sources, as NGSPICE complains about voltage sources having no specified value, while SPICE 2G6 accepts this just fine.

4 June 2019 – added multi-source resistor circuits to the Gallery, under their own section.

7 April 2019 – added introductory section to the Gallery chapter discussing the use of SPICE as a tool for generating answers for practice problem sets.

11 February 2019 – added forced page breaks between examples in the Gallery chapter, because many of the SPICE netlists were getting cut off.

July 2017 – added references to *electrically common* and *electrically distinct* points when describing the function of node numbers in a SPICE netlist. Also, corrected a node-numbering error on one schematic diagram and improved the node numbering on other diagrams.

October 2016 – created changelog for future use.

September 2016 – document first published.

Index

.end, 31, 59

Adding quantities to a qualitative problem, 144
Annotating diagrams, 143

Bode plot, 65
Buffer components, 29

Card, 31, 59
Checking for exceptions, 144
Checking your work, 144
Code, computer, 153
Comments, 31, 59
Components, buffer, 29

Dimensional analysis, 143

Edwards, Tim, 154
Electrically common points, 11, 59, 70, 86, 115
Electrically distinct points, 11, 13, 18, 86, 123

Frequency-domain plot, 65

Graph values to solve a problem, 144
Ground, 27, 31, 59

How to teach with these modules, 151
Hwang, Andrew D., 155

Identify given data, 143
Identify relevant principles, 143
Intermediate results, 143
Inverted instruction, 151

Knuth, Donald, 154

Lamport, Leslie, 154
Limiting cases, 4, 144
Lissajous figure, 66

Load, 27

Moolenaar, Bram, 153

Netlist, 31, 59
Node, 31, 59

Open, 28
Open-source, 153

Parametric plot, 66
Problem-solving: annotate diagrams, 143
Problem-solving: check for exceptions, 144
Problem-solving: checking work, 144
Problem-solving: dimensional analysis, 143
Problem-solving: graph values, 144
Problem-solving: identify given data, 143
Problem-solving: identify relevant principles, 143
Problem-solving: interpret intermediate results, 143
Problem-solving: limiting cases, 4, 144
Problem-solving: qualitative to quantitative, 144
Problem-solving: quantitative to qualitative, 144
Problem-solving: reductio ad absurdum, 144
Problem-solving: simplify the system, 143
Problem-solving: thought experiment, 143
Problem-solving: track units of measurement, 143
Problem-solving: visually represent the system, 143
Problem-solving: work in reverse, 144

Qualitatively approaching a quantitative problem, 144

Reductio ad absurdum, 144, 150, 151

Short, 28

Simplifying a system, 143
Socrates, 150
Socratic dialogue, 151
Source, 27
SPICE, 7
Stallman, Richard, 153

Thought experiment, 143
Time-domain plot, 66
Title, 31, 59
Torvalds, Linus, 153

Units of measurement, 143

Visualizing a system, 143

Work in reverse to solve a problem, 144
WYSIWYG, 7, 9, 153, 154