PSpice Tutorials

PSLect01 PSLect02 PSLect03 PSLect04 PSLect05
PSLect06 PSLect07 PSLect08

Contents

4th Tutorial: Transient Analysis

5th Tutorial: Steady State AC Analysis

6th Tutorial: Mutual Inductances

7th Tutorial: Frequency Response

8th Tutorial: Special Sources

This site is maintained by William E. Dillon, Ph.D., P.E. To send comments, queries or suggestions click here.

Last Modified: undefined

1st Tutorial on PSpice

Introduction

The following information concerns the text-edited version of MicroSim PSpice, version 8. It is offered here solely for the purpose of helping undergraduate students complete their classroom assignments in computer-aided circuit analysis at the University of Texas at Arlington. No other use of these notes is supported by the University of Texas at Arlington.

File Types Used and Created by PSpice

The basic input file for PSpice is a text (ASCII) file that has the file type "CIR." In the beginning, this will be created by hand as the primary method of getting the circuit we want modeled into the PSpice program. Later, when we use the schematic capture program, it will create the *.CIR file for us, along with several auxiliary file types. Do not use a word processor to create these *.CIR files unless you "Save as" text or as ASCII. You can use Notepad to edit these files, but the best editor for this purpose is the one that is provided by MicroSim, called "TextEdit."

The output file always generated by PSpice is a text (ASCII) file that has the file type "OUT." I.e., if you submit a data file to PSpice named "MYCIRKUT.CIR," it will create an output file named "MYCIRKUT.OUT." This output file is created even if your run is unsuccessful due to input errors. The cause for failure is reported in the *.OUT file, so this is a good place to start looking when you need to debug your simulation model. You examine the *.OUT file with the TextEdit or Notepad programs. When everything works properly, you will find the output results in this file if you are running a DC analysis. If you are running a transient analysis or a frequency sweep analysis, there will be too much data for the *.OUT file. In these cases, we add a command to the *.CIR file that tells PSpice to save the numerical data in a *.DAT file.

The aforementioned *.DAT file is by default a binary (i.e., non-ASCII) file that requires a MicroSim application called PROBE for you to see the data. PROBE is installed with PSpice from the CD-ROM. If you want, you can change the default storage format to ASCII. This is not recommended because it requires more disk space to store the data in ASCII code. Later, we will describe the procedure for invoking PROBE and creating the *.DAT file. A companion file to the *.DAT file is the *.PRB file which holds initializing information for the PROBE program.

Another common method used by experienced PSpice users is the use of *.INC (include) files. These enable us to store frequently used subcircuits that have not yet been added to a library. Then we access these *.INC files with a single command line in the *.CIR file. Very convenient.

Other files used with PSpice are *.LIB files where the details of complex parts are saved; we may discuss this later, but it is unlikely that we will engage in LIB file alterations until you are taking advanced courses.

When we begin using the schematic capture program that is bundled with PSpice, we will encounter some additional file types. These are the *.SCH (the schematic data, itself), *.ALS (alias files) and *.NET (network connection files).

Some Facts and Rules about PSpice

- PSpice is not case sensitive. This means that names such as *Vbus*, *VBUS*, *vbus* and even *vBuS* are equivalent in the program.
- All element names must be unique. Therefore, you can't have two resistors that are both named "Rbias," for example.
- The first line in the data file is used as a title. It is printed at the top of each page of output. You should use this line to store your name, the assignment, the class and any other information appropriate for a title page. PSpice will ignore this line as circuit data. Do not place any actual circuit information in the first line.
- There must be a node designated "0." (Zero) This is the reference node against which all voltages are calculated.
- Each node must have at least two elements attached to it.
- The last line in any data file must be ".END" (a period followed by the word "end.")
- All lines that are not blank (except for the title line) must have a character in column 1, the leftmost position on the line.
    - Use "*" (an asterisk) in column 1 in order to create a comment line.
    - Use "+" (plus sign) in column 1 in order to continue the previous line (for better readability of very long lines).
    - Use "." (period) in column 1 followed by the rest of the "dot command" to pass special instructions to the program.
    - Use the designated letter for a part in column 1 followed by the rest of the name for that part (no spaces in the part name).
- Use "whitespace" (spaces or tabs) to separate data fields on a line.
- Use ";" (semicolon) to terminate data on a line if you wish to add commentary information on that same line.

The above basic information is essential to using PSpice. Learn and understand these issues now to facilitate your use of the program.

References

1. *Spice: A Guide to Circuit Simulation and Analysis Using PSpice*; Tuinenga, Paul W.; © 1992, 1988 by Prentice-Hall, Inc.; ISBN: 0-13-747270; (my favorite)
2. *Computer-Aided Analysis Using SPICE*; Banzhaf, Walter; © 1989 by Prentice-Hall, Inc.; ISBN: 0-13-162579-9; (another good reference)

3. *SPICE for Circuits and Electronics Using PSpice*; Rashid, Muhammad H.; © 1990 by Prentice-Hall, Inc.; ISBN: 0-13-834672; (supports electronics well)
4. *SPICE for Power Electronics and Electric Power*; Rashid, Muhammad H.; © 1993 by Prentice-Hall, Inc.; ISBN: 0-13-030420; (best for power electronics)

Node Designations in PSpice

The original SPICE program developed decades ago at U. C. Berkeley, accepted data only on BCD punch cards. That's why it was not case sensitive; developers have preserved this lack of case sensitivity for backward compatibility. In the original SPICE program, users were expected to designate nodes by number. Most users used small integers, and the numbers did not have to be contiguous. Today, most spice programs accept ordinary text for node designations. If you want to declare a node as "Pbus," you can. The only restriction seems to be that you can't embed spaces in a node name. Use the underscore ("_") character to simulate spaces.

Out of habit, most users of PSpice still use small integers as node designations. This often improves the readability of a PSpice source file or output file. In general, you should avoid extremely long textual names for node designations. Naming a node "Arlington_Junior_Chamber_of_Commerce" makes your files look choppy and hard to read. Also, you will sometimes have to *type* that long cumbersome name when you are performing analysis on the output data file. My suggestion is to use small integers as node designators for most cases. However, use short descriptive names for nodes whenever clarity is improved. "T1_col," when used to designate the collector node of transistor, T1, carries a lot more meaning than "37."

Large and Small Numbers in PSpice

PSpice is a computer program used mostly by engineers and scientists. Accordingly, it was created with the ability to recognize the typical metric units for numbers. Unfortunately, PSpice cannot recognize Greek fonts or even upper vs. lower case. Thus our usual understanding and use of the standard metric prefixes has to be modified. For example, in everyday usage, "M" indicates "mega" (10^6) and "m" stands for milli (10^{-3}). Clearly, this would be ambiguous in PSpice, since it is not case sensitive. Thus, in PSpice, a factor of 10^6 is indicated by "MEG" or "meg." "M" or "m" is reserved for 10^{-3}. Another quirk of PSpice is the designation for 10^{-6}. In most publications, the Greek letter, μ, is used for this multiple. Since there can be no Greek fonts (or any other special font designations) in PSpice, the early developers of PSpice borrowed a trick from those who used typewriters. Before the IBM Selectric typewriter was introduced, most writers of technical papers had to improvise for Greek letters. Since the Latin letter "u" (at least in lower case) sort of resembled the lower case Greek μ, it was widely used as a substitute for μ. Hence, either "U" or "u" stands for 10^{-6} in PSpice. Without further background explanations, these are the metric prefix designations used in PSpice:

Number	**Prefix**	***Common Name***
10^{12} -	"T" or "t"	*tera*

10^{9}	-	"G" or "g"	*giga*
10^{6}	-	"MEG" or "meg"	*mega*
10^{3}	-	"K" or "k"	*kilo*
10^{-3}	-	"M" or "m"	*milli*
10^{-6}	-	"U" or "u"	*micro*
10^{-9}	-	"N" or "n"	*nano*
10^{-12}	-	"P" or "p"	*pico*
10^{-15}	-	"F" or "f"	*femto*

An alternative to this type of notation, which is in fact, the default for PSpice output data, is "textual scientific notation." This notation is written by typing an "E" followed by a signed or unsigned integer indicating the power of ten. Some examples of this notation are shown below:

- 656,000 = 6.56E5
- -0.0000135 = -1.35E-5
- 8,460,000 = 8.46E6

The Most Basic Parts

Here, we present the simplest circuit elements. Knowing how to model these ideal, linear circuit elements is an essential start to modeling more complex circuits. In each case, we will only present the most fundamental version of the part at this time. Later we will show you more sophisticated uses of the part models.

Ideal Independent Voltage Sources

We begin with the DC version of the ideal independent voltage source. This is the default form of this class of part. The beginning letter of the part name for all versions of the ideal independent voltage source is "*V*." This is the character that must be placed in column 1 of the line in the text file that is used to enter this part. The name is followed by the positive node designation, then the negative node designation, then an optional tag: "DC" followed by the value of the voltage. The tag "DC" (or "dc" if you prefer) is optional because it is the default. Later, when we begin modeling AC circuits and voltage sources that produce pulses and other interesting waveforms, we will be required to designate the type of source or it will default back to DC.

One of the interesting uses of ideal independent voltage sources is that of an *ammeter*. We can take advantage of the fact that PSpice saves and reports the value of current entering the positive terminal of an independent voltage source. If we do not actually require a voltage source to be in the branch where we want to measure the current, we simply set the voltage source to a zero value. It still calculates the current in the branch. In fact, we *require* an independent voltage source in a branch where that branch's current is the controlling current for a current-controlled dependent source.

Examples:

```
*name +node -node  type  value   comment
Va    4     2      DC      16.0V; "V" after "16.0" is optional
vs    qe    qc     dc      24m  ; "QE" is +node & "qc" is -node
VWX   23    14             18k  ; "dc" not really needed
vwx   14    23     DC   -1.8E4 ; same as above
Vdep  15    27     DC     0V    ; V-source used as ammeter
```

Resistors

Although PSpice allows for sophisticated temperature-dependent resistor models, we will begin with the simple, constant-value resistor. The first letter of the name for a resistor must be "R." The name is followed by the positive node, then the negative node and then the value in ohms or some multiple of ohms. The value of resistance will normally be positive. Negative values are allowed in order to permit an alternative model of an energy source. A value of zero, however, will produce an error. Later, we will introduce special resistor models that will permit additional analysis methods to be used.

The resistor is not an active device, so the polarity of its connection has no effect on the values of the voltages and currents reported in the solution. However, the current through a resistor is reported as that which flows from the node on the left to the node on the right in the source code line in which the resistor is entered. Thus .PRINT statements and PROBE queries that report resistor current may show negative values of current depending on the order in which you list the resistor's two nodes in the *.CIR file. If you want to see the resistor's current as a positive value, reverse the order of the nodes on the resistor's line in the *.CIR file and re-run the analysis. Nothing else will be affected and both solutions can be correct.

Examples:

```
*name +node -node value comment
Rabc  31     0    14k ; reported current from 31 to 0
Rabc   0    31    14k ; reported current changes sign
rshnt 12    15    99m ; 0.099 ohm resistor
Rbig  19    41  10MEG ; 10 meg-ohm resistor
```

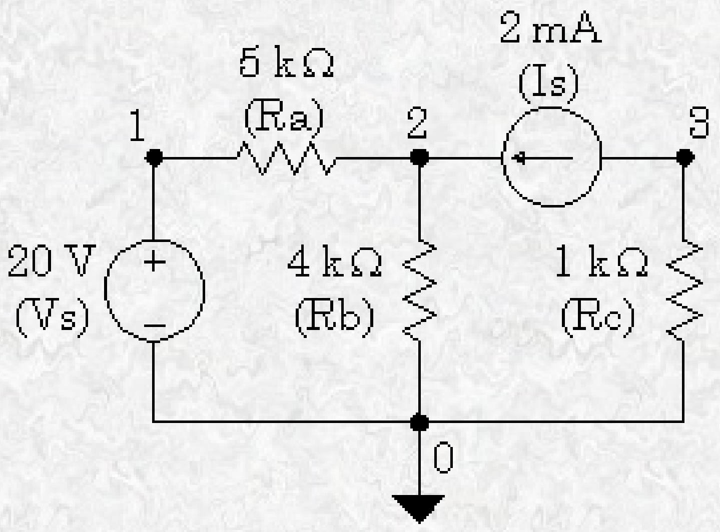
Ideal Independent Current Sources

The name of an ideal independent current source begins with the letter "I" in column 1 of the data file. As with the independent voltage source, we begin by introducing only the DC form of this part, but several other forms exist. Since the current source, is an active element, it matters greatly how it is connected. Designated current flows into the node written on the left, through the current source, out the node written on the right. As with the independent voltage source, the default type is DC. Remember that the so-called +node on a current source may have a negative voltage with respect to the so-called -node. This is due to the fact that the circuit external to the current source determines its voltage.

Examples:

```
*name +node -node type   value comment
Icap  11     0    DC     35m  ; 35mA flows from node 11 to 0
ix    79    24            1.7 ; "DC" not needed
I12   43    29    DC  1.5E-4  ;
I12   29    43    dc  -150uA  ; same as above
```

Circuit Example 1



```
Example_1 EXMPL01.CIR
Vs   1   0   DC   20.0V ; note the node placements
Ra   1   2   5.0k
Rb   2   0   4.0k
Rc   3   0   1.0k
Is   3   2   DC   2.0mA ; note the node placements
.END
```

The output file EXMPL01.OUT is below. This has been edited to remove extra lines.

```
Example_1 EXMPL01.CIR                      <== Title Line
Vs 1 0 20.0V ; note the node placements
Ra 1 2 5.0k
Rb 2 0 4.0k
Rc 3 0 1.0k
Is 3 2 2.0mA ; note the node placements
```

```
Example_1 EXMPL01.CIR                      <== Title Line

NODE VOLTAGE NODE VOLTAGE NODE VOLTAGE NODE VOLTAGE

( 1) 20.0000 ( 2) 13.3330 ( 3) -2.0000    <== Results

VOLTAGE SOURCE CURRENTS

NAME CURRENT

Vs -1.333E-03       <== Current entering node 1 of Vs

TOTAL POWER DISSIPATION 2.67E-02 WATTS

JOB CONCLUDED

TOTAL JOB TIME .26
```

This was the bare-bones minimum problem we could ask of PSpice. Note that we obtained the node voltages which is sufficient information to calculate the resistor currents. However, there is another command that we can use to get even that done by PSpice.

Use of the .PRINT Command

One of the many "dot commands" in PSpice is the .PRINT command. It has many uses, but we will concentrate here on using it for printing DC voltages and currents. The .PRINT command can be repeated as often as necessary in an analysis. You can list as many items on a line as you wish.

However, we must keep in mind that the .PRINT command was designed to work with a DC or an AC sweep. This is a method of varying a parameter over a range of values so that we get a batch of cases solved all at once. Often, we do not actually want to run a sweep over many values of a parameter. We can circumvent the sweep by setting its range so that it can only run one value. Usually, a DC sweep is made by changing the values of a source; although we will later learn to sweep over other circuit parameters. For now, let's look at the syntax for a DC sweep command with the default linear type range.

```
.DC Sweep_Variable Starting_Value Stopping_Value Increment
```

For our example problem, we choose the voltage source and set the sweep variable range so that it cannot run more than one value:

```
.DC Vs 20.0  20.0 1.0
```

Since the starting value equals the stopping value, the analysis will only run for one case, i.e., for Vs at 20 volts. Remember that the only reason we are running the DC sweep statement is to *enable* the .PRINT command. The .PRINT command will not work unless there is a sweep going on. Note: What you enter in the .DC statement *overrides* any voltage value you may have placed in the part listing for the source.

Printing DC Voltages

In addition to printing the node voltages in which you type the letter "V" with the node number in parentheses, you can print the voltage between any pair of nodes; ergo, V(m,n) prints the voltage from node "m" to node "n."

```
.PRINT DC V(1) V(2) V(3)  ;  prints the node voltages
.PRINT DC V(1,2)          ;  prints the voltage across Ra
.PRINT DC V(3,2)          ;  prints the voltage across Is
```

Printing DC Currents

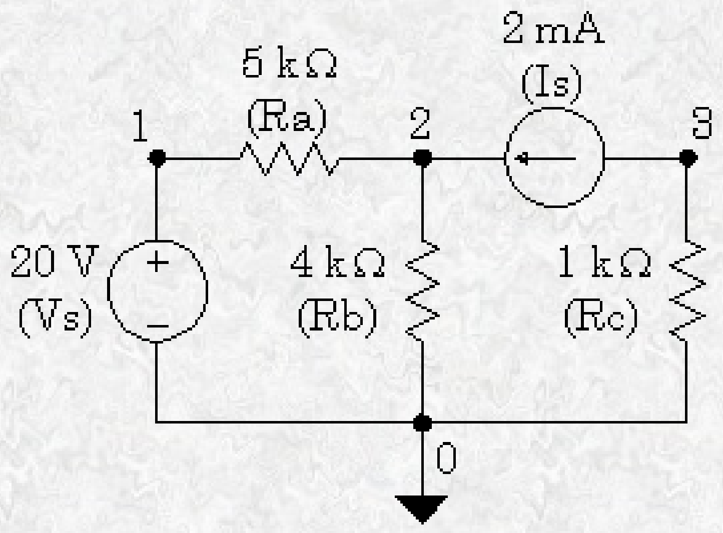
To print currents, you type the letter "I" with the element name in parentheses. Note that the reported current is that which flows into the element from the node listed on the left in the *.CIR file, through the element, and out the node listed on the right in the *.CIR file. If you want to change the sign of the reported current in a resistor, then swap the two nodes for that resistor.

```
.PRINT DC I(Ra)          ; prints the currents from + to - of Ra
.PRINT DC I(Rb) I(Rc)    ; prints the currents through Rb and Rc
```

Print Commands can be Combined

```
.PRINT DC V(1,2) I(Ra)   ; voltage and current for Ra
.PRINT DC V(2,0) I(Rb)   ; V(2,0) same as V(2)
.PRINT DC V(3,0) I(Rc)   ; V(3,0) same as V(3)
```

Use .PRINT with Previous Example



```
Example_2 EXMPL02.CIR
Vs    1    0    DC    20.0V ; note the node placements
Ra    1    2    5.0k
Rb    2    0    4.0k
Rc    3    0    1.0k
Is    3    2    DC    2.0mA ; note the node placements
.DC Vs 20 20 1            ; this enables the .print commands
.PRINT DC V(1,2) I(Ra)
.PRINT DC V(2) I(Rb)
.PRINT DC V(3) I(Rc)
.END
```

The output file EXMPL02.OUT is below.   This has been edited to remove extra lines.

```
Example_2 EXMPL02.CIR
Vs 1 0 20.0V ; note the node placements
Ra 1 2 5.0k
Rb 2 0 4.0k
Rc 3 0 1.0k
Is 3 2 2.0mA ; note the node placements
.DC Vs 20 20 1 ; this enables the Print commands
.PRINT DC V(1,2) I(Ra)
.PRINT DC V(2) I(Rb)
.PRINT DC V(3) I(Rc)

Example_2 EXMPL02.CIR

**** DC TRANSFER CURVES TEMPERATURE = 27.000 DEG C
```

```
Vs          V(1,2)    I(Ra)

2.000E+01 6.667E+00 1.333E-03 <== data for Ra

Example_2 EXMPL02.CIR

**** DC TRANSFER CURVES TEMPERATURE = 27.000 DEG C

Vs          V(2)      I(Rb)

2.000E+01 1.333E+01 3.333E-03 <== data for Rb

Example_2 EXMPL02.CIR

**** DC TRANSFER CURVES TEMPERATURE = 27.000 DEG C

Vs          V(3)       I(Rc)

2.000E+01 -2.000E+00 -2.000E-03 <== data for Rc

JOB CONCLUDED

TOTAL JOB TIME .13
```

With a little bit of effort, we can get PSpice to do most of the work, most of the time. Note that using .PRINT has suppressed the default printing of all the node voltages. This is not a problem in our case because we printed all three node voltages anyway. Be sure that you include everything you need in the .PRINT statements.

Back to Main Page

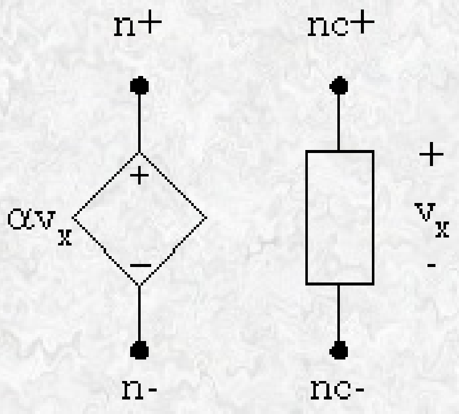
Last Modified: undefined

2nd Tutorial on PSpice

Simple Dependent Sources

We now extend our circuit parts list by adding the most basic dependent sources. The four dependent sources we now encounter are simple multiples of the controlling voltage or current. It is possible to model dependent sources that are complex nonlinear functions of several controlling voltages and/or currents. However, we will now concentrate on the basic linear dependent sources.

Voltage Controlled Dependent Voltage Source

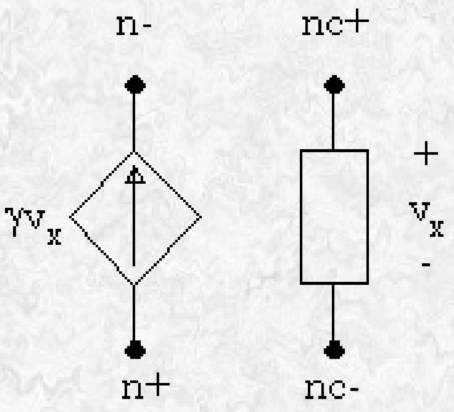


In the above figure, we find the dependent source whose positive terminal is designated as "n+" and whose negative terminal is designated as "n-." The controlling voltage is a branch voltage at some other circuit location. In this case, the positive terminal of the controlling branch is designated as "nc+" while the negative terminal is designated as "NC" The "gain" of the dependent voltage source is α, a dimensionless quantity. For example, if v_x happened to be 16.0 volts while α = 4, then node "n+" would be at 64.0 volts higher potential than node "n-."

The first letter of the part name for the voltage-controlled dependent voltage source is "E." This is the letter that must appear in column 1 of the *.CIR file describing the circuit. Some examples of the voltage-controlled dependent voltage source PSpice entries follow.

```
*Name   n+  n-   NC+   NC  gain
Ebar    17   8   42    18    24.0; gain is 24
efix     3   1   11     0   20.0
efix     3   1    0    11  -20.0; same as above
efix     1   3   11     0  -20.0; same as above
efix     1   3    0    11   20.0; same as above
Ellen   12   0   20    41  16.0
```

Voltage Controlled Dependent Current Source

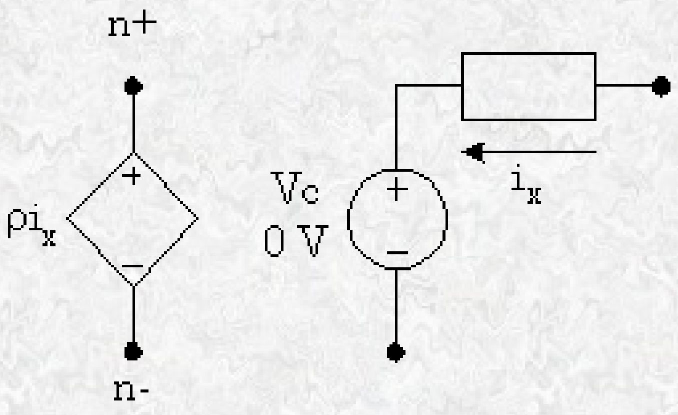


In the above voltage-controlled dependent current source a current equal to γ times v_x flows from node "n+" through the source and out node "n-." γ is called the transconductance and has the dimensions of siemens (inverse ohms). For example, if the controlling branch voltage, v_x , equals 6.0 volts and the transconductance, γ, is 0.25 siemens, the current produced by the dependent source is 1.5 amps.

The first letter of the part name for the voltage-controlled dependent current source is "G." Some examples of how this part is coded into the *.CIR file are shown below.

```
*Name n+   n-   NC+   NC   transconductance
Glab  23   17     8    3    2.5
G1    12    9     1     0   4E-2
Grad  19   40     6   99    0.65
Grad  19   40    99    6    -0.65 ; same as above
Grad  40   19    99    6    0.65 ; etc.
```

Current Controlled Dependent Voltage Source

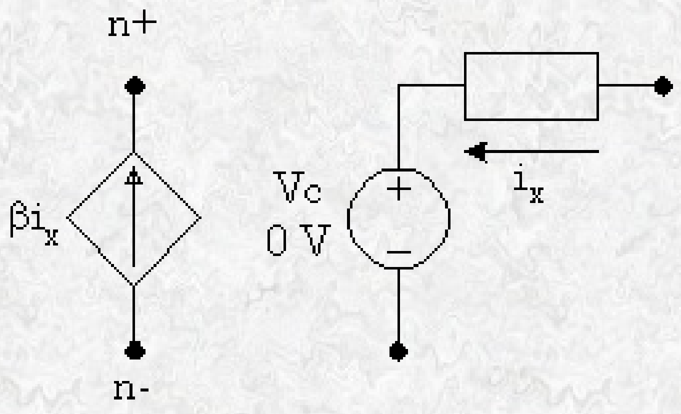


The current-controlled dependent voltage source as shown above, produces a voltage proportional to the current, i_x, in a different branch of the network. The transresistance, ρ, in ohms is multiplied by i_x in amps to produce the dependent source voltage in volts. Unlike the two previous examples, we cannot simply designate the controlling branch by its nodes. Since there could be multiple branches carrying very different currents between any pair of nodes, we must explicitly identify the branch of the controlling current. Eventually, we will be able to do this with any type of element. However, the only reliable method of doing this at present is to use an independent voltage source as an ammeter to report the current of the controlling branch to the dependent source. Usually, this means you must insert a zero-valued independent voltage source in series with the branch containing the controlling current so that the controlling current enters the positive terminal of the independent voltage source. However, if there happens to be an independent voltage source that monitors the controlling current you can use it. If necessary, use a minus sign to get the right polarity.

The first letter of the part name for the current-controlled dependent voltage source is "H." Some examples follow for this device.

```
*Name n+  n-  Vmonitor  transresistance
Hvx   20  12  Vhx         50.0
Vhx   80  76  DC          0V ; controls Hvx

Hab   10   0  V20          75.0
V20   15   5  DC          0V ; controls Hab

HAL   20  99  Vuse         10.0
Vuse   3   5  DC         20V ; actual voltage source
```

Current Controlled Dependent Current Source



The current-controlled dependent current source produces a current proportional to the controlling current, i_x, flowing in a different branch. The current gain, β, is dimensionless. Designating the control scheme is similar to setting up the current-controlled dependent voltage source previously discussed. We must use a voltage source connected in series with the controlling element so that the controlling current enters the positive terminal of the independent voltage source used as an ammeter. If no voltage source is needed for its voltage, we use a zero-valued voltage source as shown in the figure.

The first letter in the part name for this dependent source is "F." The syntax for entering this part in *.CIR files is shown in several examples below.

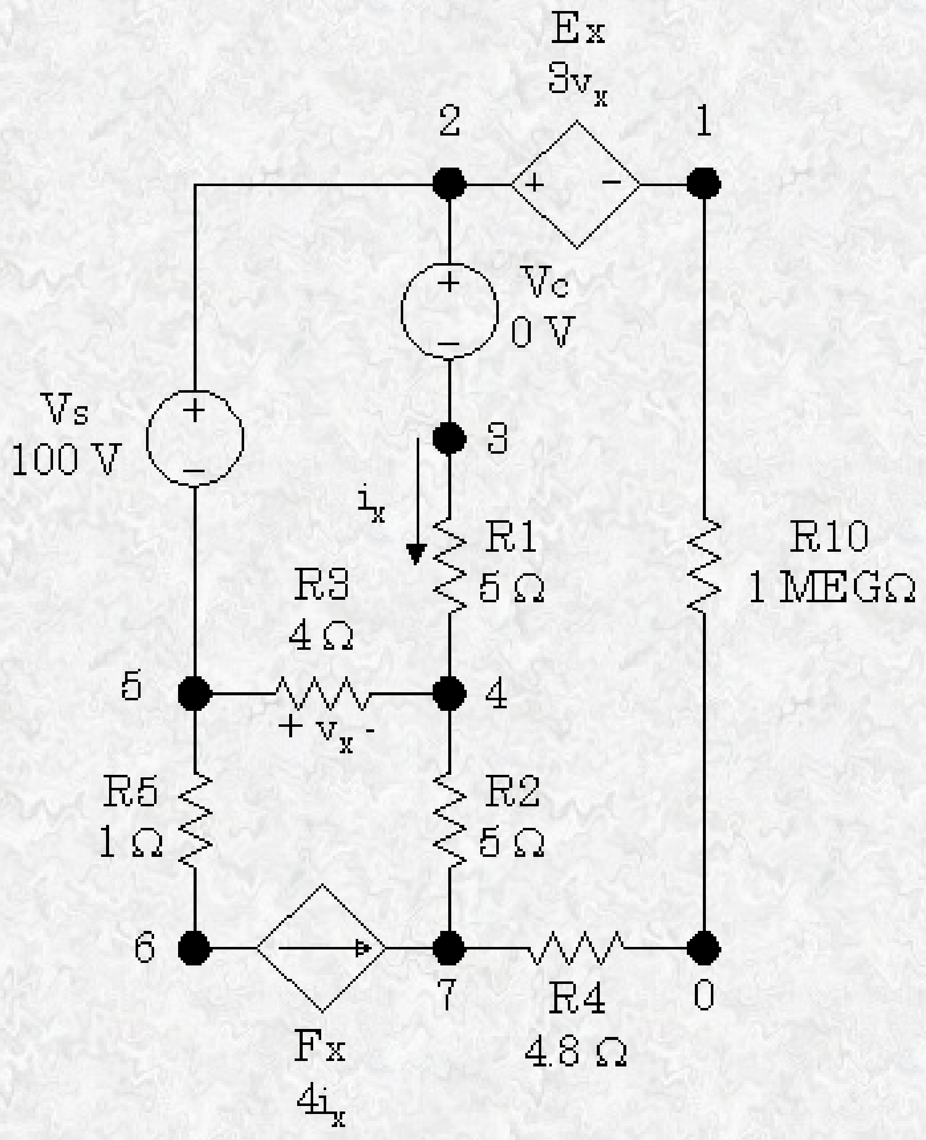
```
*Name   n-   n+   Vmonitor    Gain
Ftrn    81   19   Vctl         50.0
Vclt    23   12   DC           0V ; controls Ftrn

Fcur    63   48   Vx           20.0
Vx      33   71   DC           0V ; controls Fcur

F3       2     0   V1           15.0
V1       3    1   DC           0V ; controls F3
```

Using PSpice to find Thévenin Equivalent Circuit

In addition to performing general purpose circuit analysis, PSpice can be used to determine the Thévenin resistance and open circuit voltage of a circuit. This can be of great advantage if the circuit is complex, with several dependent sources, or if the circuit cannot be reduced by successive source transformations. The PSpice "dot command" that makes this easy, is ".TF," where "TF" indicates "transfer function." The transfer function is intended to find the ratio between a source voltage or current, and a resulting voltage difference or branch current. This is useful in characterizing circuits. In addition to reporting the calculated transfer function ratio and input resistance at the source, PSpice reports the *output resistance* at the terminal pair of interest. The voltage across the terminal pair of interest is the Thévenin voltage and the output resistance is the Thévenin resistance. At this point we will ignore the transfer function ratio and the input resistance at the source. In fact, we do not care which source is chosen as long as we only want the Thévenin equivalent circuit parameters. An example of the syntax for the .TF command is shown below.

```
*command output_variable input_source
.TF      V(4)             Vs
```

The above command will report the ratio between source Vs and node voltage V(4). If we wanted the Thévenin circuit from nodes 4 to 0, the output resistance reported would be our Thévenin resistance, and the voltage V(4) would be the Thévenin (open circuit) voltage. The input source can be a voltage or a current source, and the output variable can be a node voltage, branch voltage or a device current. Now we examine a specific example.



In this example, we want the Thévenin equivalent circuit from nodes 1 to 0. The 1 Megohm resistor is placed in the circuit because PSpice requires at least two connections to each node. This resistor is large enough that it will not have an effect on the calculations. Note the use of voltage source Vc which has the purpose of monitoring the control current, i_x, used for the current-controlled dependent current source, Fx. The input lines in the *.CIR file are shown below.

```
Thevenin Example No. 1
Vs   2   5   DC     100V
Vc   2   3   DC       0V; controls Fx
Fx   6   7   Vc      4.0; gain = 4
*   n+  n-  NC+ NC gain
Ex   2   1   5  4   3.0; gain = 3
R1   3   4   5.0
R2   4   7   5.0
R3   5   4   4.0
R4   7   0   4.8
```

```
R5   5   6   1.0
R10  1   0   1MEG; satisfies PSpice
*   out_var  input_source
.TF  V(1,0)  Vs
.END
```

Portions of the output file produced by this case will now be listed.

```
Thevenin Example No. 1

**** CIRCUIT DESCRIPTION

Vs 2 5 DC 100V
Vc 2 3 DC 0V; controls Fx
Fx 6 7 Vc 4.0; gain = 4.0
Ex 2 1 5 4 3.0; gain = 3.0
R1 3 4 5.0
R2 4 7 5.0
R3 5 4 4.0
R4 7 0 4.8
R5 5 6 1.0
Rab 1 0 1MEG
.TF V(1,0) Vs

Thevenin Example No. 1

**** SMALL SIGNAL BIAS SOLUTION TEMPERATURE = 27.000 DEG C

NODE VOLTAGE NODE VOLTAGE NODE VOLTAGE NODE VOLTAGE

( 1) 180.0000 ( 2) -60.0010 ( 3) -60.0010 ( 4) -80.0010

( 5) -160.0000 ( 6) -176.0000 ( 7)-864.0E-06

 VOLTAGE SOURCE CURRENTS

NAME CURRENT

Vs -4.000E+00

Vc  4.000E+00

TOTAL POWER DISSIPATION 4.00E+02 WATTS
```

```
 **** SMALL-SIGNAL CHARACTERISTICS

 V(1,0)/Vs = 1.800E+00
```
<== Transfer function

```
INPUT RESISTANCE AT Vs = 2.500E+01

OUTPUT RESISTANCE AT V(1,0) = 5.000E+00
```
<== Thévenin resistance

```
 JOB CONCLUDED

TOTAL JOB TIME .01
```

We conclude that the Thévenin resistance is 5 ohms and the open circuit voltage is 180 volts. Use of the .TF function allows us to get the answers in a single job. The alternative to using the .TF function would be to run one case with a large resistor across the terminal pair of interest (if necessary) to get the open circuit voltage; and then run a second case with a zero-valued voltage source across the terminal pair to get the short circuit current. Then divide the short circuit current into the open circuit voltage to get the Thévenin resistance. We prefer the ".TF" method for obtaining Thévenin equivalent circuits.

Back to Main Page

Last Modified: undefined

3rd Tutorial on PSpice

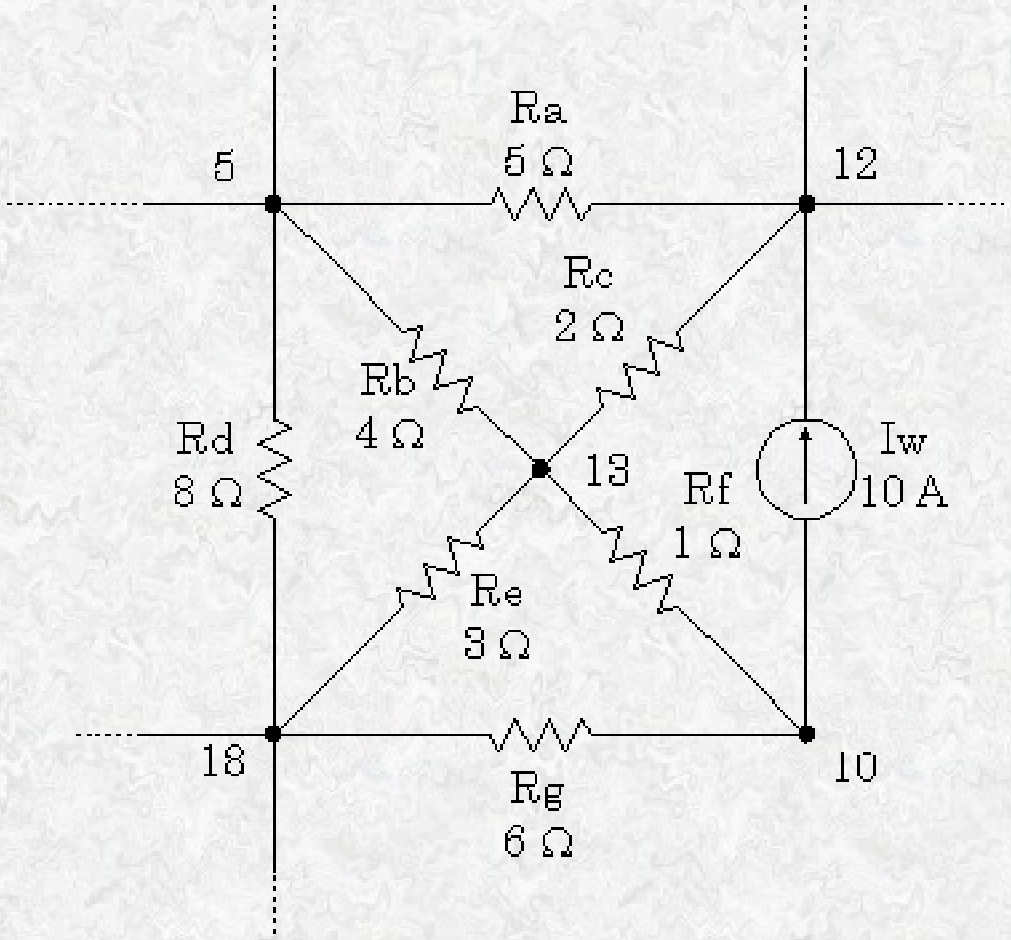
Simple Subcircuits in PSpice

One of the more useful concepts in PSpice is the use of *subcircuits* to group elements into clusters in order to replicate the clusters without having to re-enter all the elements each time. This is very useful for several reasons. First is the labor savings of replacing many lines of circuit data with a single subcircuit call.   Second, the use of a subcircuit usually improves clarity by removing confusing clutter. The user can suppress printing unwanted details internal to a subcircuit, thus making the output easier to understand. If desired, the user can place often-used subcircuits into an *include* file so that the main source file for the problem is kept simple. Then the definition of the subcircuit is out of sight entirely.

Coding a Subcircuit

Each subcircuit used in a study must have a unique name. This is true of any other circuit element. Also, there must be a list of at least two nodes that can be connected to elements external to the subcircuit. A subcircuit can have many external node connections, if needed. Later, we will find that parameters can be passed to a subcircuit in order to allow unique behavior and responses from an instance of a subcircuit.

The initial line of a subcircuit section must begin with ".SUBCKT," followed by the name and then the external node list. After that, optional features (not to be discussed yet) can be added. The best method of understanding the use of a subcircuit is by example. Below, we find a cluster of components that can be combined into a subcircuit.



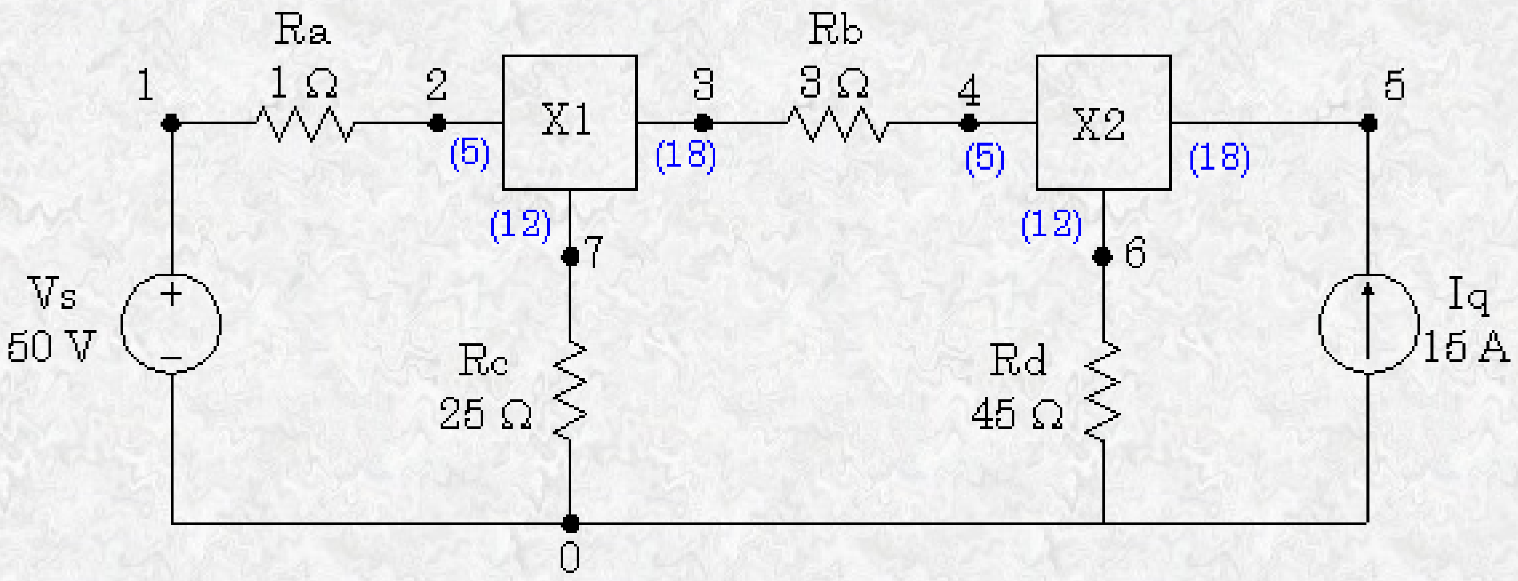
Note that nodes 5, 12 and 18 have external connections. Therefore, they must be included in the node list in the subcircuit definition. Nodes 10 and 13 do not have external connections and need not be (indeed *should* not be) included in this node list. They are internal nodes and will be used to help define the subcircuit.   Now, we can code the above subcircuit as follows. Note that the code could be embedded into the rest of the code for the main circuit or could be placed in a separate *include* file.

```
*       name         nodelist
.SUBCKT Example_1   5   12   18
Iw   10   12   DC   10A
Ra    5   12   2.0
Rb    5   13   5.0
Rc   12   13   2.0
Rd    5   18   8.0
Re   13   18   3.0
Rf   10   13   1.0
Rg   10   18   6.0
.ENDS
```

Note that the subcircuit section must be terminated with a ".ENDS" command.

Invoking a Subcircuit

All subcircuit calls are made by declaring a part with a unique name beginning with "X," followed by the node list and then the subcircuit name. The node list in the calling statement must have the same number of nodes as the node list in the subcircuit definition. To demonstrate the use of the calling statement, we present the following main circuit which contains two instances of the above subcircuit. X1 and X2 are the two instances of the subcircuit "Example_1." For added clarity, the subcircuit's defined external nodes are shown in parentheses. Note that these nodes are mapped into the main circuit by *different names*.



The code for the above circuit with the subcircuit included follows:

```
Subcircuit Example No. 1
*       name        nodelist
.SUBCKT Example_1   5   12   18
Iw    10    12    DC    10A
Ra     5    12    2.0
Rb     5    13    5.0
Rc    12    13    2.0
Rd     5    18    8.0
Re    13    18    3.0
Rf    10    13    1.0
Rg    10    18    6.0
.ENDS
Vs     1     0    DC    50V
Ra     1     2    1.0   ; different from Ra above
Rb     3     4    3.0   ; different from Rb above
Rc     7     0   25.0   ; different from Rc above
Rd     6     0   45.0   ; different from Rd above
```

```
*     nodelist       name
X1    2    7    3    Example_1
X2    4    6    5    Example_1
.END
```

Scope of Element Names and Nodes in a Subcircuit

Scope of names and nodes is local to a subcircuit. In the main circuit of which the above subcircuit is a part, there is a node 5 and there are resistors with the names of "Ra," "Rb," "Rc," and "Rd," and PSpice can keep these apparent duplications separated. If the subcircuit were invoked as "X1," for example, PSpice would consider the subcircuit parts as "X1.Ra," "X1.Rb" and so on. Additionally, the internal node numbers would be treated as "X1.5," "X2.5," "X2.13" and so forth. Thus PSpice maintains uniqueness of element names and node numbers.

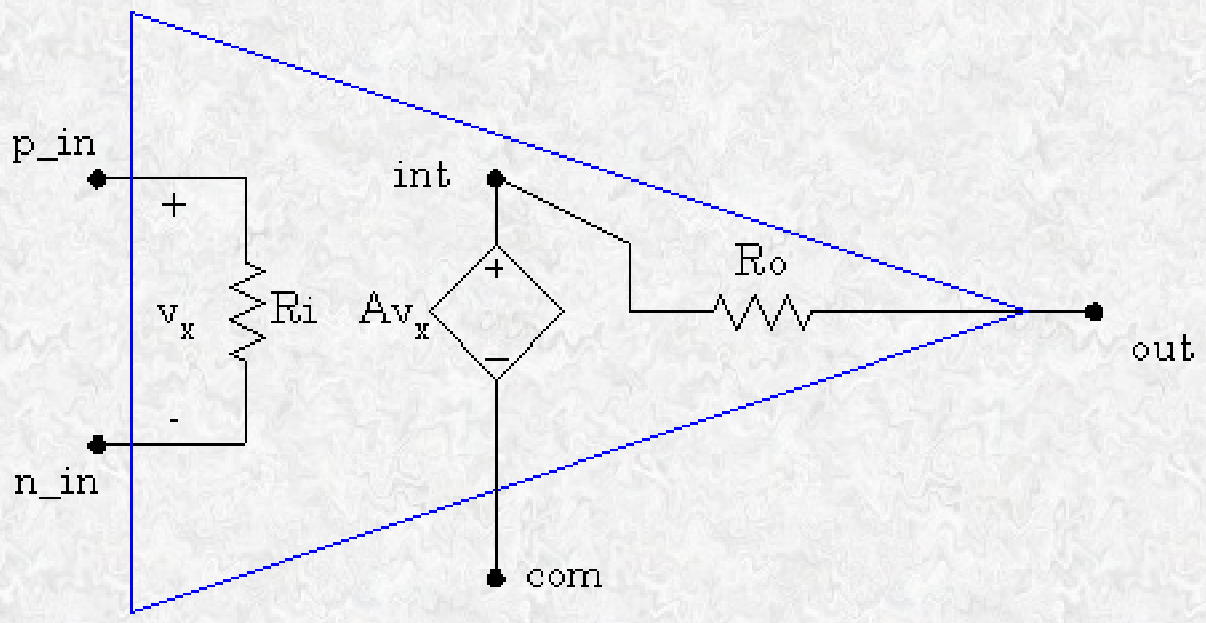
Nesting of Subcircuits

Subcircuit *calls* may be nested as long as they are not circular. In other words, you can have a part name starting with "X" within a .SUBCKT/.ENDS block provided that the "X" part definition does not call on that block for its own definition.

However, subcircuit *definitions* may *not* be nested. I.e., you can't have one .SUBCKT/.ENDS block nested within another.

An Op-Amp Example

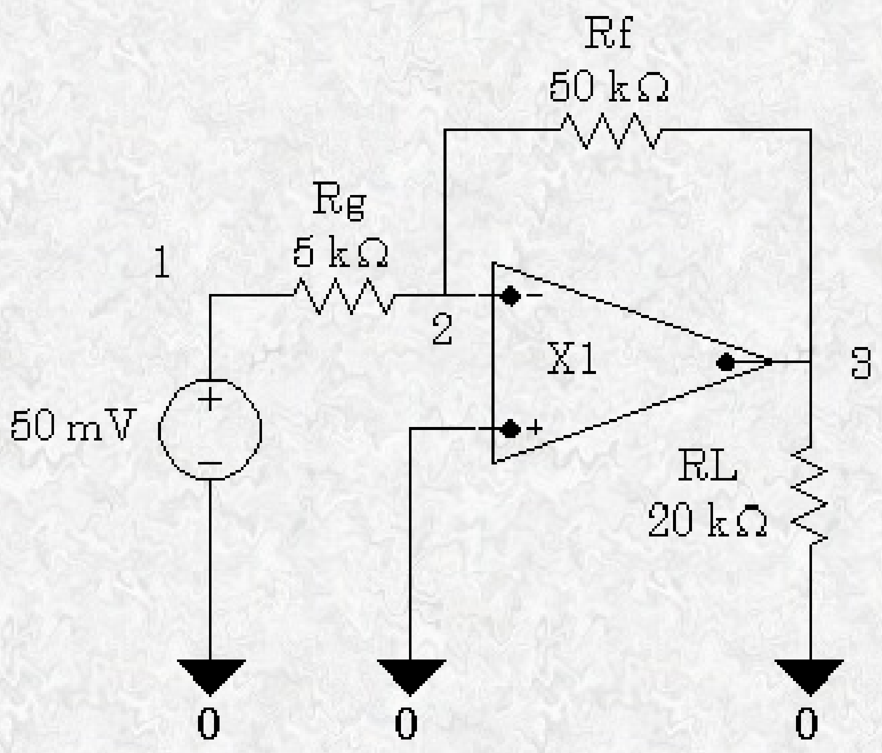
At this stage of knowledge about PSpice, we can model a simple op-amp as a subcircuit. Alas, we will not be able to show its saturation characteristics until we explain the use of the "TABLE" feature of PSpice. However, we can do a credible job of modeling an op-amp as long as it isn't allowed to saturate. The figure below illustrates this simple model of an op-amp.



In an ideal op-amp, Ri, the input resistance, and A, the open-loop gain, are infinite.   Also, Ro, the output resistance, is zero. Here, we will use "typical" values of a practical op-amp. Let A = 100,000, Ri = 500 kΩ, and Ro = 50 Ω. After all, PSpice doesn't accept infinity as a number and resistors cannot be set to zero. We will also use *text* for node designations here. Also, note the internal node "int" that is not included in the node list. The key part to the op-amp is the voltage-controlled dependent voltage source designated as part "Ex" in the listing. For a more complete description of this part, see Tutorial No. 2, where dependent sources are introduced. Code to define the subcircuit follows.

```
.SUBCKT OpAmp p_in n_in com out
Ex    int    com    p_in    n_in    1e5
Ri    p_in   n_in   500k
Ro    int    out    50.0
.ENDS
```

The main circuit with which we will test this op-amp subcircuit follows. We will use a simple inverting amplifier circuit for which we can verify the results by inspection.



```
Subcircuit Example No. 2 - Inverting OpAmp
.SUBCKT OpAmp p_in n_in com out
Ex    int    com    p_in    n_in    1e5
Ri    p_in   n_in   500k
Ro    int    out    50.0
.ENDS
Vg    1      0      DC       50mV
Rg    1      2      5k
Rf    2      3      50k
RL    3      0      20k
X1    0      2      0       3     OpAmp
.END
```

The output file (edited to remove excess lines, etc.,) is as follows:

```
Subcircuit Example No. 2 - Inverting OpAmp
**** CIRCUIT DESCRIPTION


.SUBCKT OpAmp p_in n_in com out
Ex int com p_in n_in 1e5
RI p_in n_in 500k
Ro int out 50.0
.ENDS
Vg 1 0 DC 50mV
Rg 1 2 5k
Rf 2 3 50k
```

```
RL 3 0 20k
X1 0 2 0 3 OpAmp

Subcircuit Example No. 2 - Inverting OpAmp

**** SMALL SIGNAL BIAS SOLUTION TEMPERATURE = 27.000 DEG C

NODE VOLTAGE NODE VOLTAGE NODE VOLTAGE NODE VOLTAGE

 ( 1) .0500 ( 2) 5.017E-06 ( 3) -.4999 (X1.int) -.5017

VOLTAGE SOURCE CURRENTS

NAME CURRENT

Vg -9.999E-06

TOTAL POWER DISSIPATION 5.00E-07 WATTS

JOB CONCLUDED

TOTAL JOB TIME .19
```

Discussion of Results

Had this op-amp been ideal, the closed loop gain would have been -10   -(Rf / Rg). Then the output voltage would have been -0.5V. Instead, we calculated -0.4999V. Also, an ideal op-amp would have produced a voltage of zero at the negative input. Instead, we see about 5 microvolts. These discrepancies are due to the more realistic model of the op-amp. Yet the difference is small.   It seems that the concept of using the ideal op-amp model does not lead to excessive error in this case.

The advantage of using the subcircuit in PSpice is now apparent.   We could have replicated the subcircuit many times, using only one line of code per replication. Later, we will add to our knowledge of the features of subcircuits.

Back to Main Page

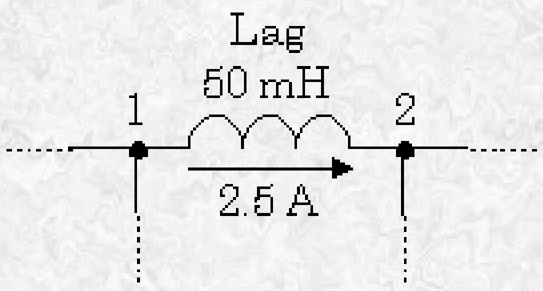
Last Modified: undefined

4th Tutorial on PSpice

Linear Inductors in PSpice

The next passive element we add to our parts list is the linear inductor. This part name begins with the letter, L, in column 1 of the source listing. Be aware that PSpice enables this part to access a nonlinear model description. That will be explained in a later tutorial. For now, our inductor model is a linear device incapable of saturation.

The inductor stores energy in its magnetic field. This makes it necessary to be able to specify its initial current in a simulation. Although we can include inductors in DC circuit simulations, there is usually little advantage in doing so because the inductor behaves as a short circuit under steady-state DC excitation. In steady-state AC simulations the inductor behaves as an imaginary impedance. We do not specify initial current in an inductor in either of those steady-state conditions. However, when simulating transient operations, we often need to specify this initial current.



The figure shown above shows the circuit symbol for an inductor with node designations of "1" and "2," an initial current of 2.5 A, and a value of 50 mH. An appropriate code listing for entering this element into a PSpice circuit file is:

```
*name nodelist    L_val
Lag    1    2     50m     IC=2.5
```

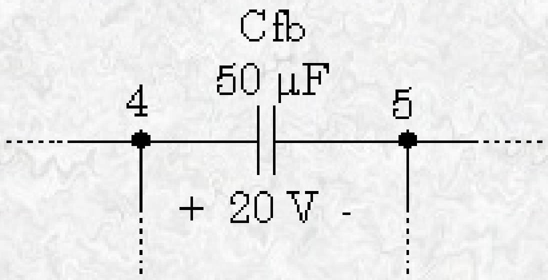
Note that the initial current is assumed to flow from the first node in the node list through the inductor towards the second node in the node list. If there is a need to change the direction of this initial current, either reverse the order of the nodes in the node list or place a minus sign in front of the value of the initial current. For better readability, the above line could be written as:

```
Lag    1     2     50mH   IC=2.5A
```

The "H" for henrys and the "A" for amps will be ignored by PSpice.

Linear Capacitors in PSpice

The capacitor is the second energy storing circuit component we add to our parts list. We will assume that the capacitor is ideal in the sense of being linear and lossless. Since it can store energy, PSpice provides a method for specifying the initial voltage across the capacitor. This is useful for simulations of transient behavior of circuits with capacitors. The figure shown below illustrates a capacitor with node designations and an initial voltage of 20 from node 4 to node 5. The part name for a capacitor must start with the letter, C.



An appropriate code listing to represent this capacitor in a PSpice listing is:

```
*name nodelist   C_val
Cfb   4    5     50u    IC=20
```

The capacitance of the above element is 50µF. This can be represented as "50u" in PSpice. (See PSpice Tutorial No. 1 for a description of the system for metric prefixes in PSpice.) Note that the polarity of the initial voltage (as shown) is such that the positive side is the first node in the list with the negative side on the second node in the list. To reverse the polarity of the initial voltage for the simulation, either reverse the order of the nodes in the node list or place a minus sign in front of the value in the "IC=" phrase. For better clarity, the above capacitor could be coded as:

```
Cfb   4    5      50uF   IC=20V
```

PSpice would ignore the "F" for farads and the "V" for volts.

Transient Analysis Using PSpice

One of the most interesting aspects of circuit analysis is the study of natural and step responses of circuits and the responses of circuits to time-varying sources. To perform these analyses we introduce another group of "dot" commands.

Use of the .TRAN command

This is the command that passes the user's parameters for performing the transient analysis on a circuit to the PSpice program. There are four time parameters and an instruction to use the initial condition rather than calculated bias point values for starting conditions. First, we show a sample .TRAN statement and then we will describe its parameters.

```
*       prt_stp   t_max   prt_dly    max_stp
.TRAN   20us      20ms    8ms         10us      UIC
```

In the above statement, the "20us" value labeled "prt_stp" *(print step)* is the frequency with which data is saved. In this case, the system variables are stored each 20µs of simulation time. The actual time steps used by PSpice may be different from this. The second parameter, "20ms," labeled as "t_max" *(final time)* is the value of time at which the simulation will be ended. Since PSpice starts at t = 0, there will be a total of 20ms time span of simulation for the circuit. The third parameter, "8ms," labeled as "prt_dly" *(print delay)* is the print delay time. In some cases, we do not want to store the data for the entire time span of the simulation. In our sample statement shown above, we ignore the data from the first 8ms of simulation and then store the data for the last 12ms. Most of the time, this parameter is set to zero or not used. The fourth parameter, "10us," labeled as "max_stp" *(max step)* is the maximum time step size PSpice is allowed to take during the simulation. Since PSpice automatically adjusts its time step size during the simulation, it may increase the step size to a value greater than desirable for displaying the data. When the variables are changing rapidly, PSpice shortens the step size, and when the variables change more slowly, it increases the step size. Use of this parameter is optional. The last parameter in our list is "UIC." It is an acronym for "Use Initial Conditions." Unless you include this parameter, PSpice will ignore the initial conditions you set for your inductors and capacitors and will use its own calculated bias point information instead. Note that the use of the letter "s" after the numbers in the .TRAN statement is optional. PSpice assumes these values are seconds and actually ignores the "s." However, it is recommended that you use units until you are extremely familiar with all of these commands and definitions.

Now, we will examine some more .TRAN examples.

```
.TRAN   10ns   500us
```

In the above example, PSpice will save data at each 10ns interval of the simulation starting at t = 0 until the final time of 500µs. I.e., there is no print delay and the user has given full control of the calculation step size to PSpice. In addition, PSpice will calculate its own initial conditions for any inductors and capacitors, ignoring any initial conditions set by the user.

```
.TRAN   50m   2.5   0    10m   UIC
```

In the above statement, PSpice collects the data at each 50ms time interval starting from zero up to 2.5s. A zero was required as as a placeholder for the print delay parameter since the maximum step size of 10ms was specified. PSpice will use the designated initial conditions of capacitor voltage and inductor current. Notice that the units were left off the numbers in this statement. Only the prefixes which size the values are needed.

Use of the .PROBE command

In addition to specifying the time parameters for a transient solution of a circuit problem, we need to specify how the data is to be saved. In most cases, this simply means that we include a line in the *.CIR file consisting of ".PROBE." This instructs PSpice to create a data file and store the data it calculates. If we create a circuit listing named "CIRCUIT1.CIR" containing a ".TRAN" statement and a ".PROBE" statement, PSpice will create a file named "CIRCUIT1.DAT" holding the data as well as the usual "CIRCUIT1.OUT" file with basic information about the circuit. By default, the data file created by PSpice is a binary data file; i.e., you can't read it with a text editor. This is the most efficient way of saving the data. However, there is an optional parameter (/CSDF) for the .PROBE statement that causes PSpice to save the data in a Common Simulation Data Format that is a text format that allows you to look at the raw data with a text editor. However, it will take up more space and PROBE doesn't load it for graphing. You will need to make a second run without the /CSDF parameter if you want to plot the data.

Also by default, .PROBE causes all the circuit variables to be saved, including all the variables inside each instance of each subcircuit. In some cases, this can amount to a lot of data. If you simulate a large complex circuit with many parts and need to save data at short time intervals over a very long time span, you can easily create gigabyte-size "DAT" files. To avoid this, you can specify the values you want to save. If the ".PROBE" command is issued without any parameters, everything is saved. If you specify the quantities you want saved, *only* those quantities will be saved. We will now examine some .PROBE statements.

```
.PROBE
```

All the above statement does is the enable PSpice to save everything in a binary DAT file.

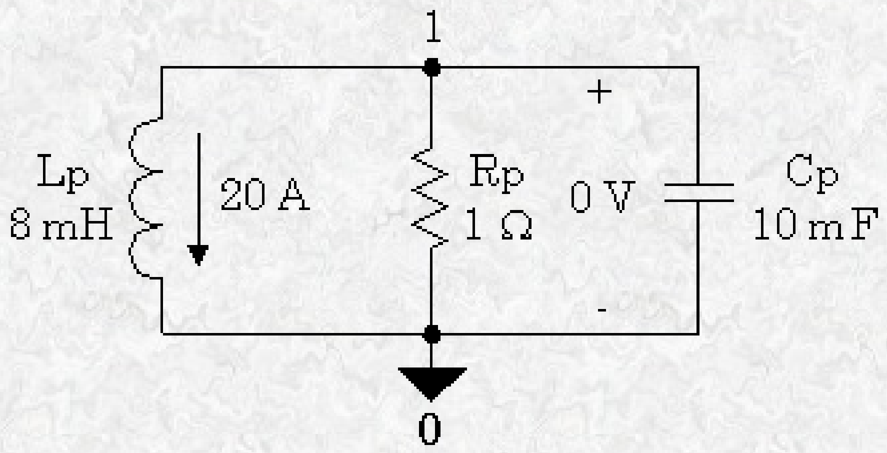
```
.PROBE/CSDF
```

The above statement enables PSpice to save everything in a CSDF file that can be opened (and edited) with a text editor. You can also read and understand the values. Unfortunately, PROBE cannot make plots from the data in this form.

```
.PROBE V(5,23) I(Rx) I(L4)
```

The above statement tells PSpice to save only the voltage drop between nodes 5 and 23, the current through resistor, Rx, and the current through inductor, L4, all in binary format. No other data will be saved.

Example of Transient Circuit Analysis



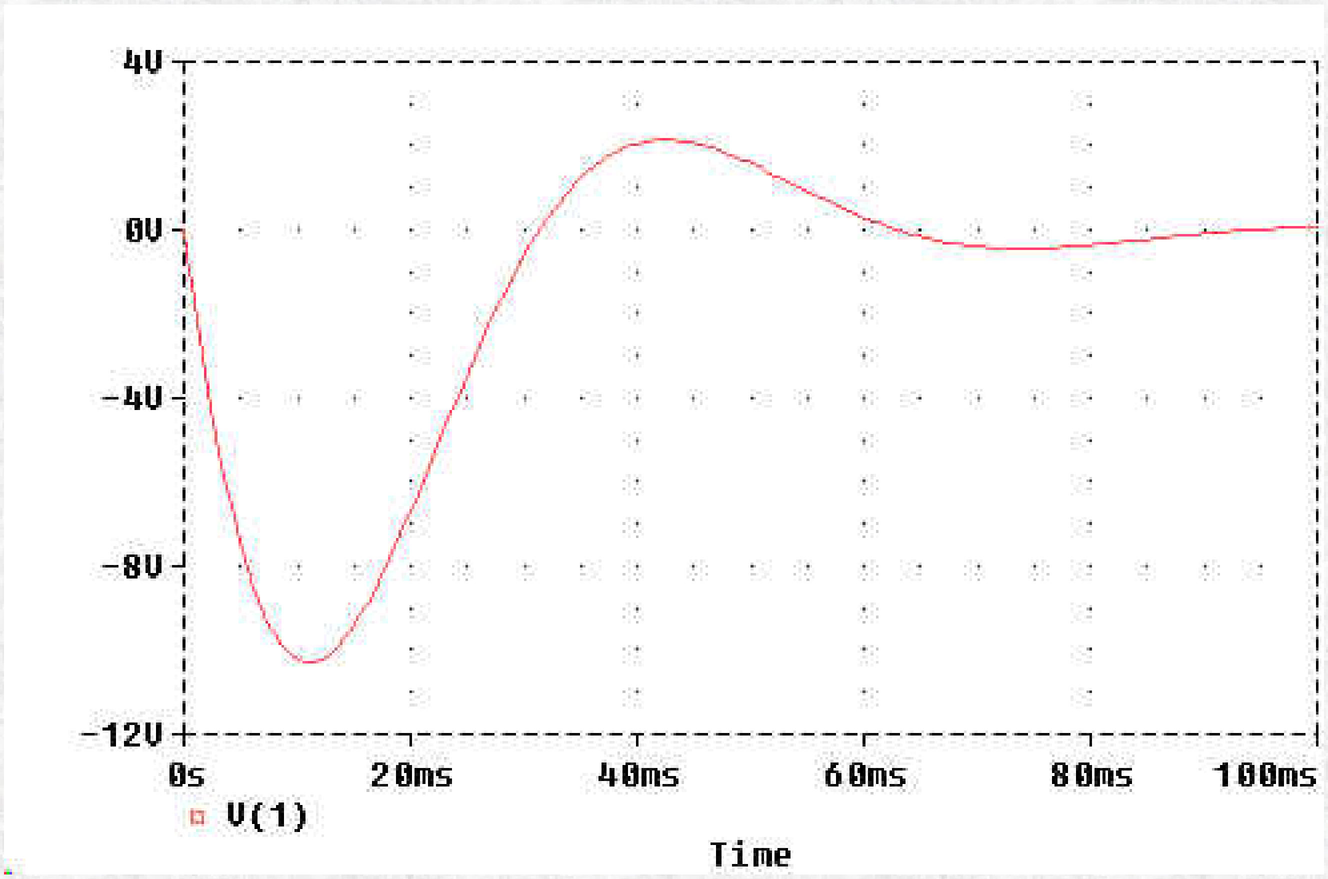
The complete listing for the "RLCNAT01.CIR" file is as follows:

```
Natural Response of a parallel RLC circuit
Rp   0   1   1.0
Lp   1   0   8mH   IC=20A
Cp   1   0   10mF  IC=0V
.TRAN 500us 100ms 0s 500us UIC
.PROBE
.END
```

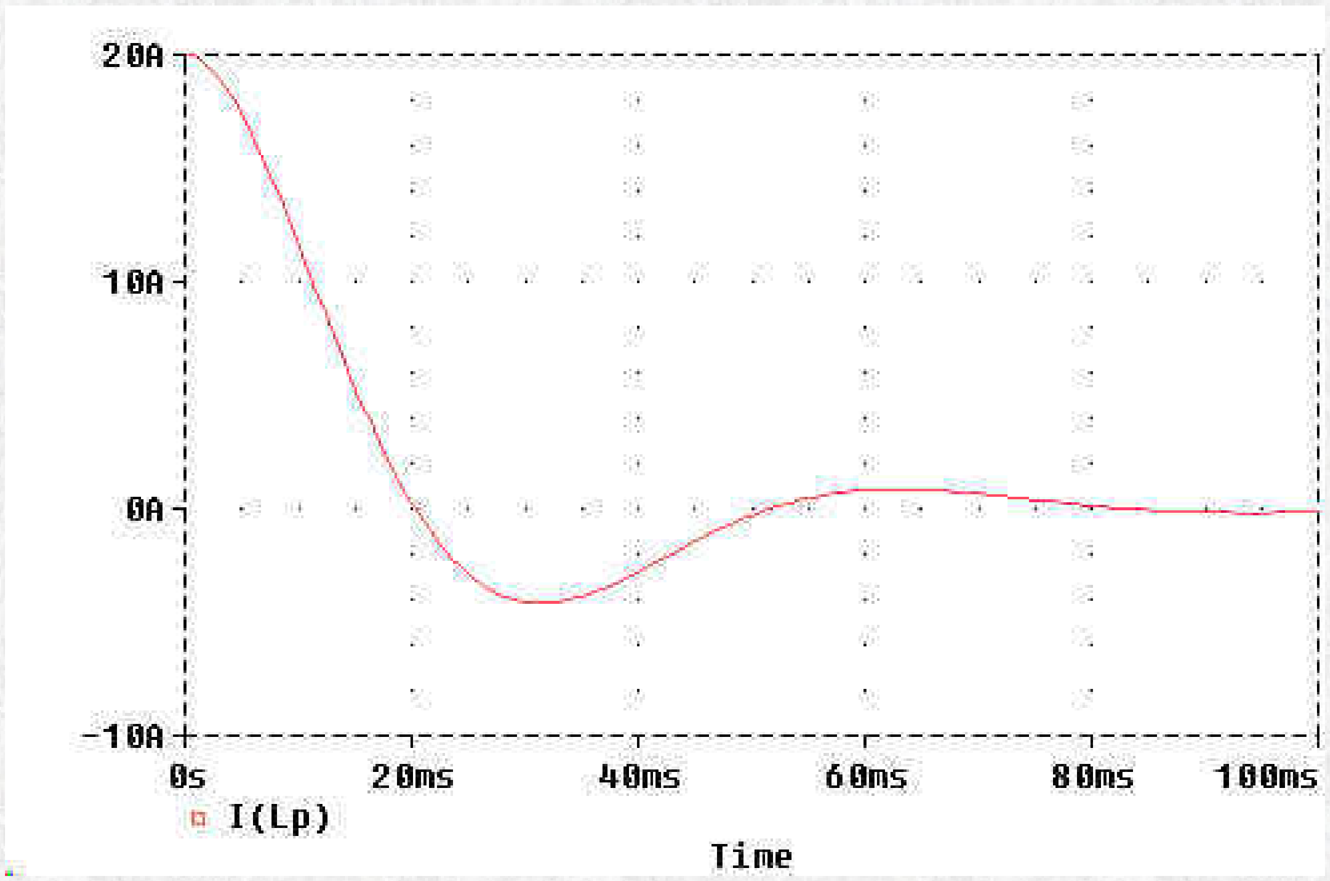
In the above example, the eight millihenry inductor, Lp, has an initial current of 20 amps flowing from node 1 through the inductor to node 0. The 10 millifarad capacitor, Cp, has an initial voltage of 0 volts. Both the print step size and the maximum step size are set to 500µs and the final time is 100ms. There is no print delay, and PSpice is instructed to use the initial conditions provided. The "RLCNAT01.OUT" file is listed below. There is little information in it because the text file can show very little of the transient behavior of the circuit.

```
**** 07/17/98 18:47:40 ******* NT PSpice 8.0 (July 1997)

Natural response of a parallel RLC circuit
RP 0 1 1.0
LP 1 0 8mH IC=20A
Cp 1 0 10mF IC=0V
.TRAN 500us 100ms 0s 500us UIC
.PROBE

JOB CONCLUDED

TOTAL JOB TIME .16
```

For meaningful information about the transient response we need to use another program that is bundled with PSpice. This program is named PROBE. The Probe program graphs the data that was saved in the "RLCNAT01.DAT" file. To invoke this program you left-click on "Run Probe" in the PSpice *File* menu. Probe will automatically open the DAT file you have just created. You can also launch Probe from the Start menu of Windows, but you will then need to go to Probe's File menu and open the DAT file you want to see. After you have Probe running with the proper DAT file open, choose "Add" in the Probe *Trace* menu. You will see a list of circuit variables that can be displayed. Choose V(1), the voltage at node 1, and then click "OK." You should see the following trace in Probe.



In Probe, click on the V(1) at the lower left corner (not here, you need to be running *Probe*) and then hit the "Delete" key. Then go back to the *Trace* menu in Probe and choose "Add" again. This time choose I(LP) and click "OK." You should see the following trace of the inductor current:



Actually, you will see a negative of the above traces. In order to get the white background as you see above, you will need to modify the "INI" file for PSpice.   That will be the topic of another tutorial.

Back to Main Page

Last Modified: undefined

5th Tutorial on PSpice

Steady-State AC Analysis in PSpice

In addition to DC circuit analysis and transient analysis, PSpice can be used to work steady-state phasor problems. To see the results of this analysis in the .OUT file, we will want to use a new form of the .PRINT command. In the first tutorial, we learned that the .PRINT DC command would not work unless we enabled it with the .DC command. This was the DC sweep command although we only allowed it to sweep a single value of voltage. We have a somewhat similar situation when we need to print AC values; i.e., we will use the .AC command to *enable* the .PRINT AC command to print our phasor voltages and currents.

AC Voltage and Current Sources`

Up to now, all our voltage and current sources were DC. We learned the syntax of the DC source in the first tutorial. The syntax for an AC source is very similar. The AC source is assumed to be a *cosine* waveform at a specified phase angle. Its frequency must be defined in a separate ".AC" command that defines the frequency for *all* the sources in the circuit. The unique information for the individual source is: the name, which must start with "V" or "I," the node numbers, the magnitude of the source, and its phase angle. Some examples follow.

```
*name nodelist type value  phase(deg)
Vac    4   1    AC   120V    30
Vba    2   5    AC   240                 ; phase angle 0 degrees
Ix     3   6    AC   10.0A  -45          ; phase angle -45 degrees
Isv   12   9    AC   25mA                ; 25 milliamps @ 0 degrees
```

Notice that the type, AC, *must* be specified, because the default is DC. If the phase angle is not specified it will be assumed as zero degrees. The units of the phase angle will be in degrees. As before, the "V" after the voltage value is optional, as is the "A" after the current value in a current source. The polarity of the AC voltage source is determined as if the voltage were a cosine function of ωt at t = 0. Then the node on the left is the positive node and the node on the right is the negative node. Similarly, the polarity of the AC current source is determined as if the current were a cosine function of ωt at t = 0. Then positive current flows into the source from the node on the left, passes through the source, and leaves the source from the node on the right.

A note of caution is needed here. By now, some of you may have discovered the "SIN" type of source by reading some of the supplementary material. The SIN is one of several useful source types (also EXP, PULSE, PWL & SFFM to name a few) that are used for *transient* analysis. Do not attempt to use *SIN* for steady-state (phasor) AC analysis nor for frequency sweeps. The SIN type is a time-based function for time-based analysis, whereas the AC type is used in frequency-based modeling. Since phasor analysis uses frequency-based models of circuit elements, always use the AC type as described in this tutorial for phasor analysis of circuits.

Use of the .PRINT AC Command

Before the .PRINT command will work, it must be enabled by the .AC command. The .AC command was designed to make a sweep of many frequencies for a given circuit. This is called a *frequency response* and will be discussed in a later tutorial. Three types of ranges are possible for the frequency sweep: LIN, DEC and OCT. At this time we only want a single frequency to be used so it does not matter which one we choose. We will pick the LIN (linear) range to designate our single frequency. Some examples of the .AC statement follow.

```
   * type #points  start stop
.AC  LIN  1        60Hz  60Hz;  <== what we want now.
.AC  LIN  11       100   200;   <== a linear range sweep
.AC  DEC  20       1Hz   10kHz; <== a logarithmic range sweep
```

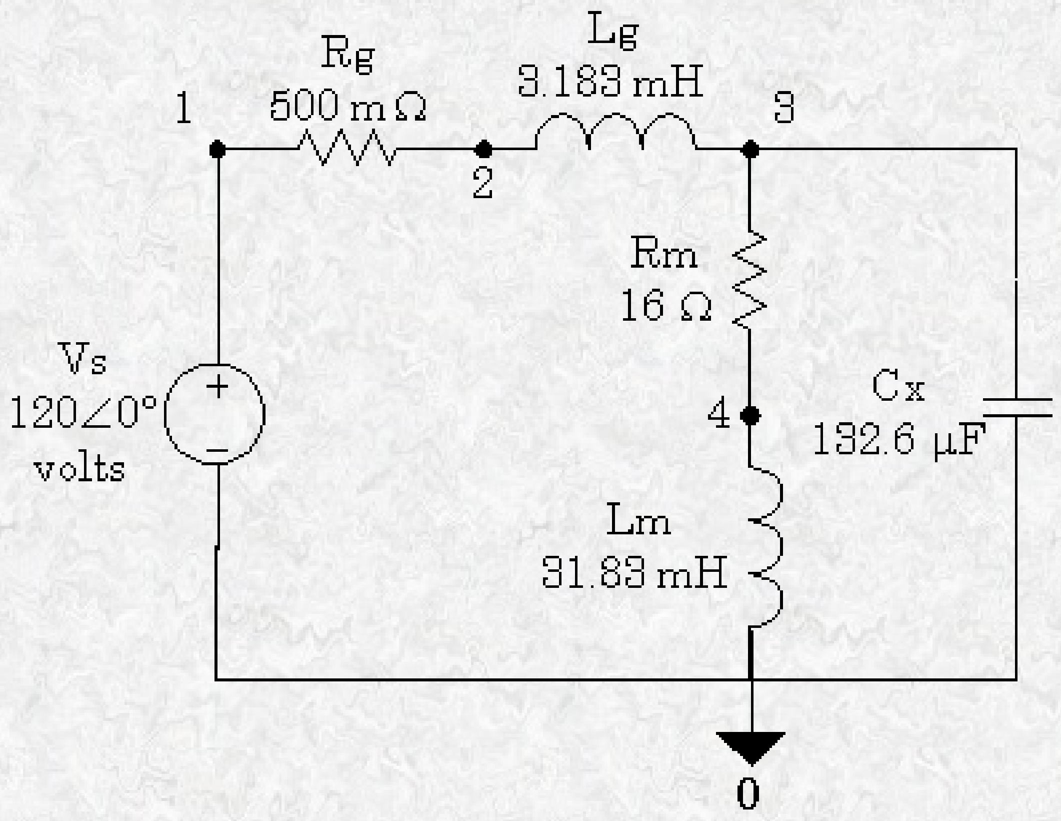
The first statement above performs a single analysis using the frequency of 60 Hz. Placing the units "Hz" after the value is optional. The second statement would perform a frequency sweep using frequencies of 100Hz, 110Hz, 120Hz, 130Hz, 140Hz, 150Hz, 160Hz, 170Hz, 180Hz, 190Hz and 200Hz. This will not be used here. The third statement performs a logarithmic range sweep using 20 points per decade over a range of four decades. This will be useful later for studying frequency response of circuits.

Finally, we can discuss the actual .PRINT AC command. Printing the components of phasor values (complex numbers) requires some options. There are four expressions needed for this: magnitude, phase (angle), real part, and imaginary part. In addition, we can print voltages or currents. For instance, to print the magnitude of a voltage between nodes 2 and 3, we would specify "VM(2,3)." The phase angle of this same voltage would be "VP(2,3)" and would be printed in degrees. If we need the current magnitude through resistor Rload, we would specify "IM(Rload)." The real part of the voltage on node 7 would be specified "VR(7)" and its imaginary part, "VI(7)." As with the .PRINT DC command, there is no limit on the number of times it can be used in a listing; nor is there a limit on how many print requests can be on a single line. Some complete examples follow:

```
.PRINT AC VM(30,9) VP(30,9); magnitude & angle of voltage
.PRINT AC IR(Rx) II(Rx); real & imag. parts Rx current
.PRINT AC VM(17) VP(17) VR(17) VI(17); the whole works on node 17
```

Example Circuit

We will analyze the following circuit at a frequency of 60 Hz.



```
60 Hz AC Circuit
Vs  1  0  AC  120V  0
Rg  1  2  0.5
Lg  2  3  3.183mH
Rm  3  4  16.0
Lm  4  0  31.83mH
Cx  3  0  132.8uF
.AC LIN 1 60 60
.PRINT AC VM(3) VP(3) IM(Rm) IP(Rm) IM(Cx) IP(Cx)
.END
```

In the above listing, the .AC command sets up the analysis for a single solution at 60 Hz. The .PRINT AC command tells PSpice to report on the voltage magnitude and phase angle at node 3, and the current magnitude and phase angle for the current through resistor Rm and the current magnitude and phase angle through capacitor Cx. The resulting output file (edited to delete clutter) follows:

```
60 Hz AC Circuit

**** CIRCUIT DESCRIPTION

Vs 1 0 AC 120 0
Rg 1 2 0.5
Lg 2 3 3.183mH
Rm 3 4 16.0
```

```
Lm 4 0 31.83mH
Cx 3 0 132.6uF
.AC LIN 1 60 60
.PRINT AC VM(3) VP(3) IM(Rm) IP(Rm) IM(Cx) IP(Cx)

60 Hz AC Circuit

**** SMALL SIGNAL BIAS SOLUTION

TEMPERATURE = 27.000 DEG C

NODE VOLTAGE NODE VOLTAGE NODE VOLTAGE
( 1) 0.0000  ( 2) 0.0000  ( 3) 0.0000

NODE VOLTAGE ( 4) 0.0000
VOLTAGE SOURCE CURRENTS

NAME CURRENT
Vs 0.000E+00

TOTAL POWER DISSIPATION 0.00E+00 WATTS

60 Hz AC Circuit

**** AC ANALYSIS TEMPERATURE = 27.000 DEG C

FREQ      VM(3)      VP(3)      IM(Rm)     IP(Rm)
6.000E+01 1.203E+02 -3.332E+00 6.014E+00 -4.020E+01

FREQ      IM(Cx)     IP(Cx)
6.000E+01 6.013E+00  8.667E+01

JOB CONCLUDED TOTAL JOB TIME .26
```

Notice that the small signal bias solution yields zero for the voltages. This is the DC part of the solution which is zero in this case because there was no DC excitation. The AC analysis has been printed in blue. The voltage at node 3 is 120.3∠-3.332° volts and the current through the capacitor is 6.014∠86.67° amps. Theory predicts that the current through a capacitor leads the voltage across the capacitor by 90°, which it does.

Summary of AC Phasor Circuit Analysis Using PSpice

- Use *AC* as the type for all independent sources
- Specify phase angle of sources if other than zero degrees

- There must be a ".*AC*" command to specify the frequency to be used for all sources
- Use a *.PRINT AC* command to specify which voltages and currents are to be listed in the output file
- **M** indicates *magnitude*, **P** indicates *phase angle*, **R** indicates *real part* and **I** indicates *imaginary part*, when these letters follow **V** (for voltage) or **I** (for current).

Back to Main Page

Last Modified: undefined

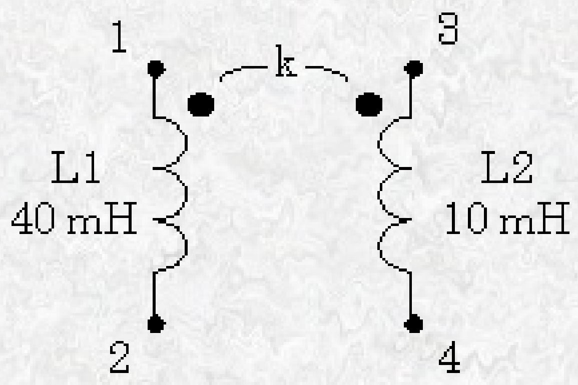
.

6th Tutorial on PSpice

Mutual Inductances in PSpice

Users of PSpice often need to model inductors that are magnetically coupled. This may occur in steady-state power system simulations, or in power electronics transient circuit simulations where linear or nonlinear transformer models are used. In some cases it is necessary to model weakly coupled inductors. This tutorial will address the issues of modeling magnetic coupling in these circumstances.

Basic Linear Coupled Inductors

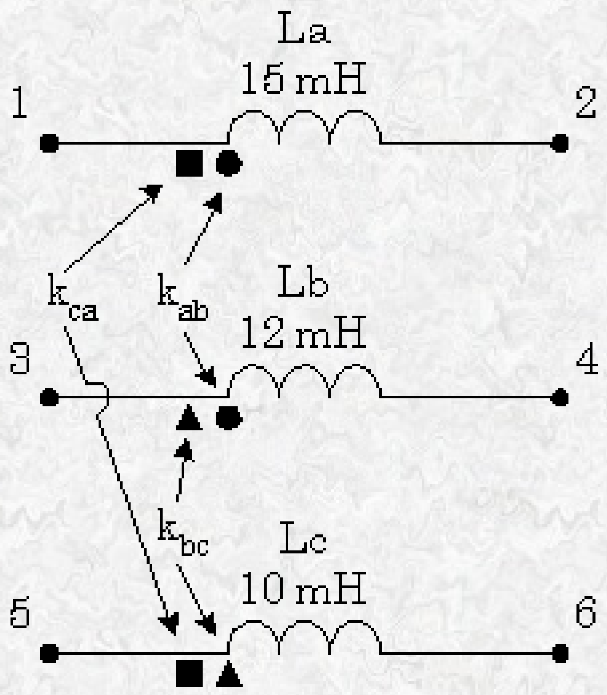


In the above figure two inductors are coupled by a coefficient of coupling, k. Their nodes are designated by small integers, and polarity marks have been added. The polarity information is passed to PSpice by the order of the nodes. If the coefficient of coupling is k = 0.8, a valid PSpice coding could be:

```
*name node1 node2 inductance (comment line)
L1    1     2     40mH
L2    3     4     10mH
*name ind1  ind2  k (comment line)
K12   L1    L2    0.8
```

Note that the polarity marks in the figure are beside nodes 1 and 3 of the inductors. In the listing, these are entered as the leftmost nodes. An equivalent polarity relationship could be indicated by reversing both nodes on both inductors. The coupling of the coils is entered by including a new part that must begin with the letter, K. The "K" part name is followed by a list of the coupled inductors, then by the value of the coefficient of coupling. The coefficient of coupling must occupy the range, $0 \leq k \leq 1$.

Multiple Couplings with Different Values

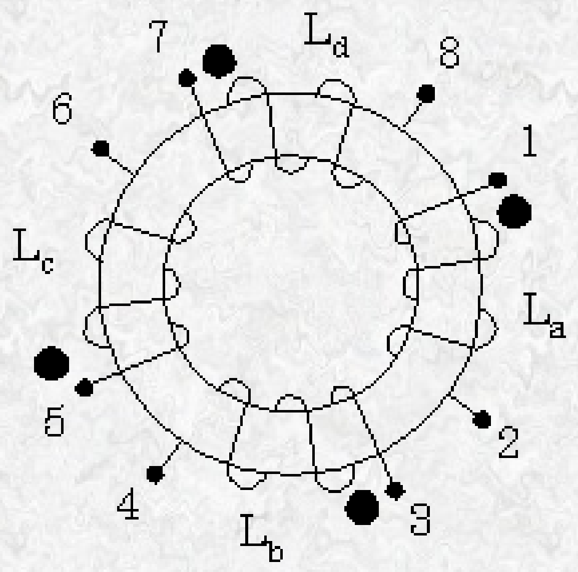


In the above figure, each inductor has mutual coupling with more than one other conductor, but at different values; i.e., the coefficient of coupling is different for at least one of the pairs. In this case, PSpice requires a separate "K" part for each coefficient of coupling as in the following code:

```
La      1      2      15mH
Lb      3      4      12mH
Lc      5      6      10mH
Kab     La     Lb     0.08
Kbc     Lb     Lc     0.075
Kca     Lc     La     0.04
```

Note that the polarities of the inductors, and therefore the sense of the mutual coupling is accounted for by the order of the nodes entered for the self inductance parts. In this case, different symbols have been used in the figure to assure that it is understood which pairs are coupled and in what sense. In general, if there are n coils, there will be ½ n(n-1) "K" parts needed.

Multiple Couplings with Same Values

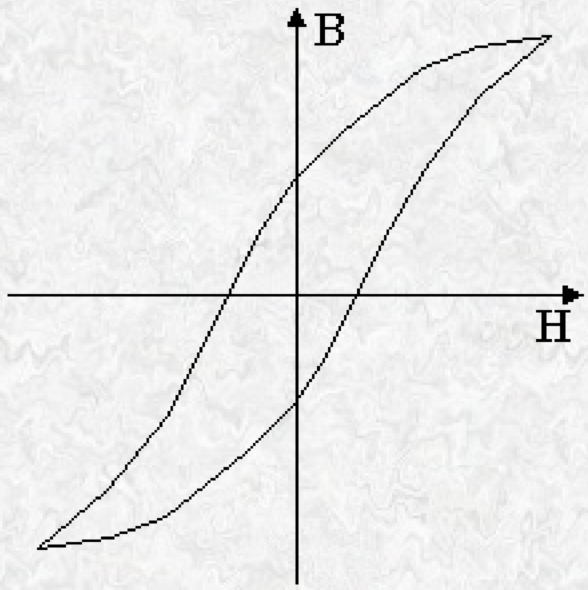


In the above example, we assume that all inductors share identical coefficients of coupling. This is a reasonable assumption when coil symmetry exists and all coils are wound on a common core. Under these conditions, PSpice allows a single "K" part to describe all the coupling.

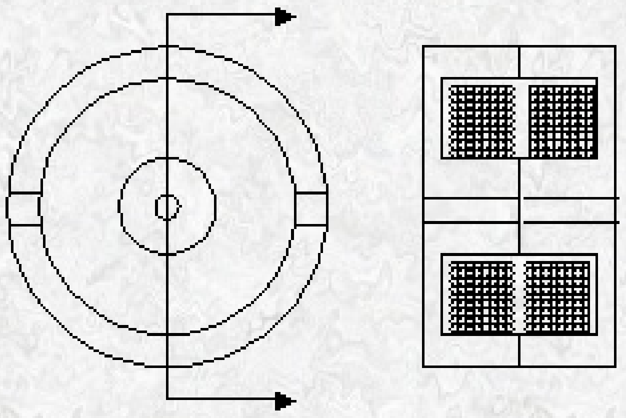
```
La     1    2    25uH
Lb     3    4    50uH
Lc     5    6   100uH
Ld     7    8   200uH
Kall  La   Lb   Lc   Ld   0.98
```

Again, the polarity information is entered by the order of the nodes for the self inductances. Since all the coupling coefficients were the same, only one "K" part was needed instead of six.

Nonlinear Core Model



Occasionally, we need to model some specific magnetic core material properties such as magnetic saturation and loss factors. This requires an additional statement in PSpice. The above figure shows a typical hysteresis loop of magnetic flux density, B, plotted as a function of magnetic field intensity, H. The effects of this can be modeled in PSpice for a specific sample of the core material with dimensions of the core and its nonlinear magnetic properties.



The above figure shows a typical ferrite "pot" core where the view on the left is the perspective one sees looking into an open core half, while the view on the right is a section view of a complete pot core transformer with two windings.

When specifying a nonlinear core, an additional PSpice statement is needed. This is the ".MODEL" statement. MODEL statements are required for most of the more complex parts used in PSpice. We will explain this by presenting an example and then dissecting its parts.

```
L1  1  2  25; <== 25 turns in winding
L2  3  4  50; <== 50 turns in winding
Knl L1 L2 0.98 my_core; <== "my_core" is a model name
.MODEL my_core Core(MS=400K A=45 C=0.4 K=25
+ AREA=1.38 PATH=4.52)
```

Several things are different about this example. First, the values after the node numbers in the "L1" and "L2" statements are the numbers of turns in the windings, instead of the inductance in henrys. This interpretation is triggered in PSpice whenever the inductor part name is mentioned in a "K" part listing that references a .MODEL statement. The next thing that is different is the model statement. All model statements have a similar form so that learning about this one will help you understand other model statements you will encounter later. The last item on the "K" part listing was the model *name* we are declaring. It is "my_core" in this instance. A model name does not have to start with any particular letter; i.e., you can use about any name you want as long as you avoid illegal characters such as embedded spaces. The .MODEL statement defines the model we are creating. Hence, the second item in the model statement is the model name. The third item in the model statement is the model *type*. In our case, this is "Core." There are many pre-defined model types in PSpice. The model type sets the rules on how to interpret the model parameters. In the "Core" model type, the parameters are: MS, the magnetic saturation in gauss; A, the thermal energy parameter in amp/meter; C, the domain flexing parameter (dimensionless); K, the domain anisotropy parameter in amp/meter; AREA, the cross-sectional flux area in cm^2; and PATH, the magnetic path length in cm. Since the format within the model statement is "parameter_name = value," the order of the parameters is not important. Since all the parameters have default values, they need not be entered if you are satisfied with the default. The "Core" model type has another parameter, GAP, which is the air gap length. It has a default value of zero, so we did not need to specify it in our example.

There appears to be a great deal of tedious detail in the above procedure. However, we normally find the details of a core model already provided in a parts library, just as we would for a standard type of diode or transistor. Our purpose here is to explain the usage, not the derivation of the models.

Uncoupled, Nonlinear Inductor

It is possible to use the nonlinear core model to create a single nonlinear inductor. We still need the "K" part and the .MODEL statement. An example follows:

```
Lsat 1 2 35; <== 35 turns
Ksat LSAT 0.999 my_new_model
.MODEL my_new_model Core(MS=500K A=40 C=0.42 K=26
+ AREA=1.25 PATH=5.63)
```

Notice that only one inductor is in the "K" part list. This use of the core model for a single, uncoupled inductor is not supported in some versions of SPICE.

Back to Main Page

Last Modified: undefined

7th Tutorial on PSpice

Frequency Sweeps in PSpice

As promised in PSpice Tutorial No. 5, we now discuss frequency sweeps over a range of frequencies. The purpose of this type of analysis is to study the frequency response of different kinds of circuits. Since frequency sweeps produce a lot of data that needs to be graphed to be clearly understood, we will reintroduce *Probe*, the graphing program that is bundled with PSpice. We last used *Probe* in PSpice Tutorial No. 4, when we discussed transient analysis.

Specifying AC Sources

AC voltage and current sources are specified as they were described in PSpice Tutorial No. 5; i.e., we use the *AC* designation. To review AC source designations, click here.

To specify the frequency range we need the .AC command. When we discussed single-frequency phasor analysis in PSpice Tutorial No. 5, we alluded to the LIN, OCT and DEC types of frequency ranges. At that time we only wanted one frequency, so which range type we used didn't matter much. We will discuss the different range types here.

LIN Range Type

The LIN range type is linear. It divides up the range between the minimum and maximum user-specified frequencies into evenly spaced intervals. This is best used to view details over a narrow bandwidth. The first parameter after the keyword LIN is the number of points to calculate. This is followed by the lowest frequency value in Hz, then the highest frequency value in Hz. As with all the range types, the unit "Hz" is optional.

```
.AC LIN 101 2k 4k; 101 points from 2 kHz to 4 kHz
.AC LIN 11  800 1000; 11 pts from 800 Hz to 1 kHz
```

OCT Range Type

The OCT range is logarithmic to the base two. Thus each octave has the same number of points calculated. This is somewhat useful for designing electronic equipment for musical applications. However, the resulting graphs are very similar in appearance to sweeps made with the DEC range. The first parameter after the keyword OCT is the number of points per octave to calculate. This is followed by the lowest frequency value in Hz, then the highest frequency value in Hz.

```
.AC OCT 20 440Hz 1.76kHz; 20 points/octave over 2 octaves
.AC OCT 40 110Hz 880Hz; 40 points/octave over 3 octaves
```

DEC Range Type

The DEC range is logarithmic to the base ten. Thus each decade has the same number of points calculated. This is the most commonly used range for making *Bode plots* of a frequency response. The first parameter after the keyword DEC is the number of points per decade to calculate. This is followed by the lowest frequency value in Hz, then the highest frequency value in Hz.

```
.AC DEC 50 1kHz 100kHz; 50 points/decade over 2 decades
.AC DEC 25 100k 100MEG; 25 points/decade over 3 decades
```

Probe Needed

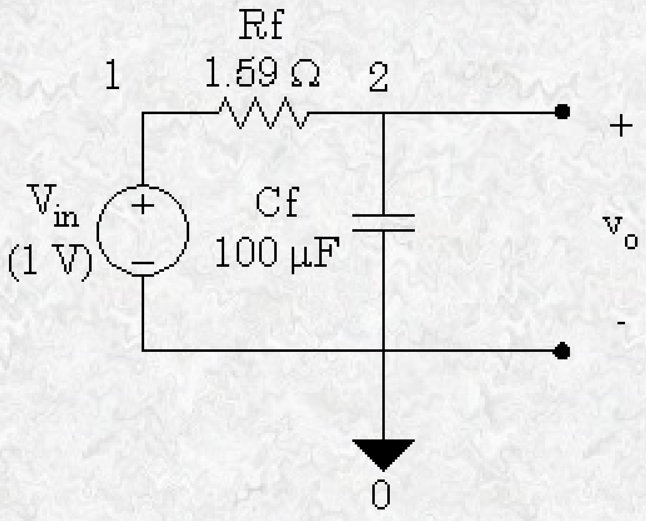
In PSpice Tutorial No. 4, we introduced PROBE since the transient analyses we were using then required storing lots of data and the ability to plot it. We have the same situation here. The major difference is that the independent variable used by PROBE in a .TRAN analysis is *time*; whereas the independent variable used in a frequency sweep is *frequency*. Also, when PROBE stores data in a transient (.TRAN) analysis, the dependent variables are instantaneous voltages and currents; whereas in a frequency sweep these dependent variables are real and imaginary components of phasor voltages and currents.

To review details of the .PROBE command, click here.

.PRINT AC?

Normally, we do not use the .PRINT AC command when we run a frequency sweep on a circuit because PROBE does a pretty good job of supplying graphical interpretations of the data. However, the .PRINT AC command can be used to store any voltage or current data in tabular form in the OUT file. This is readable and can be copied and pasted into other programs for whatever additional processing your imagination can contrive. This can be done simultaneously with the .PROBE command, whereas your alternative (if you *must* have a text-readable version of the data) is to make a separate run with the /CSDF parameter after the .PROBE command.

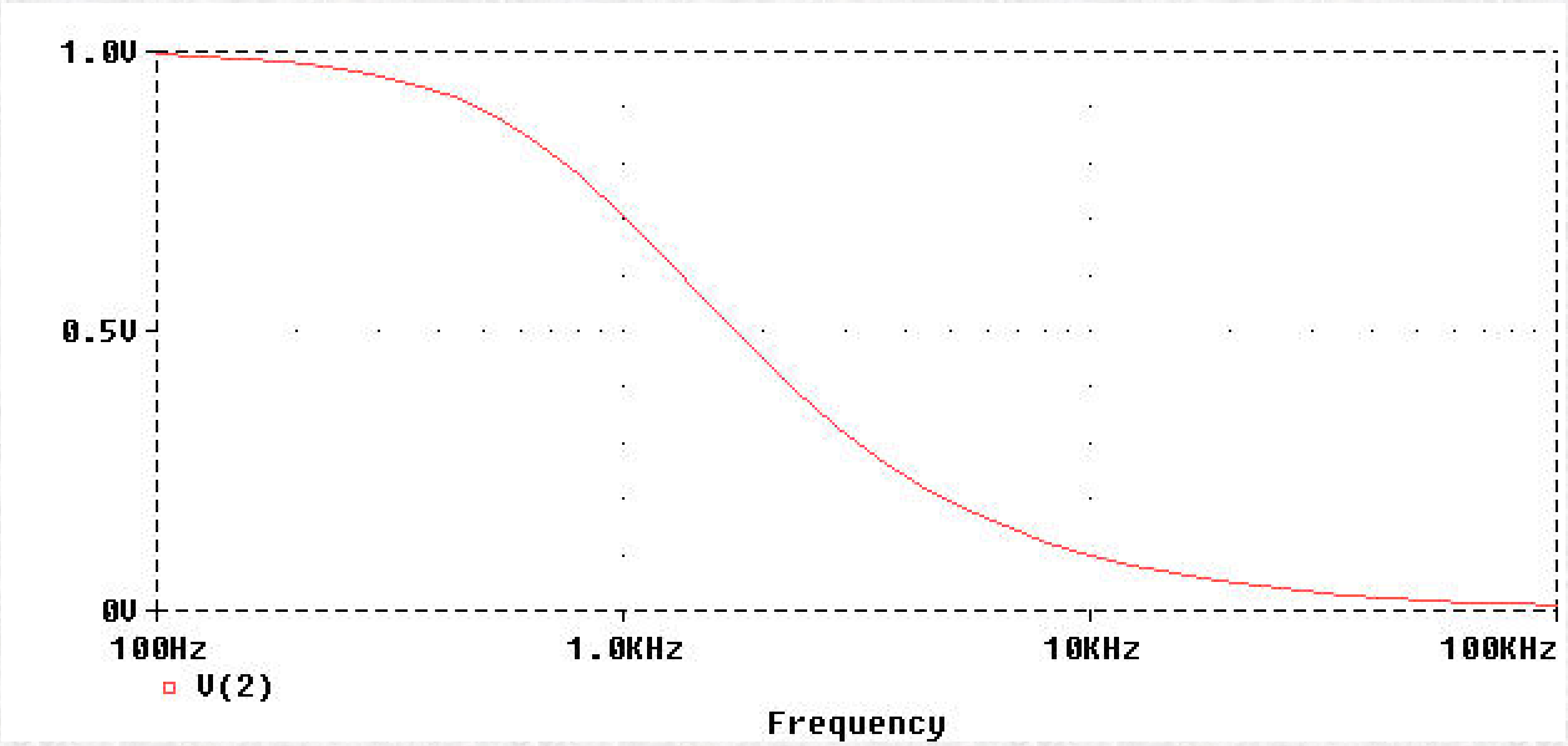
Examples of Frequency Sweeps



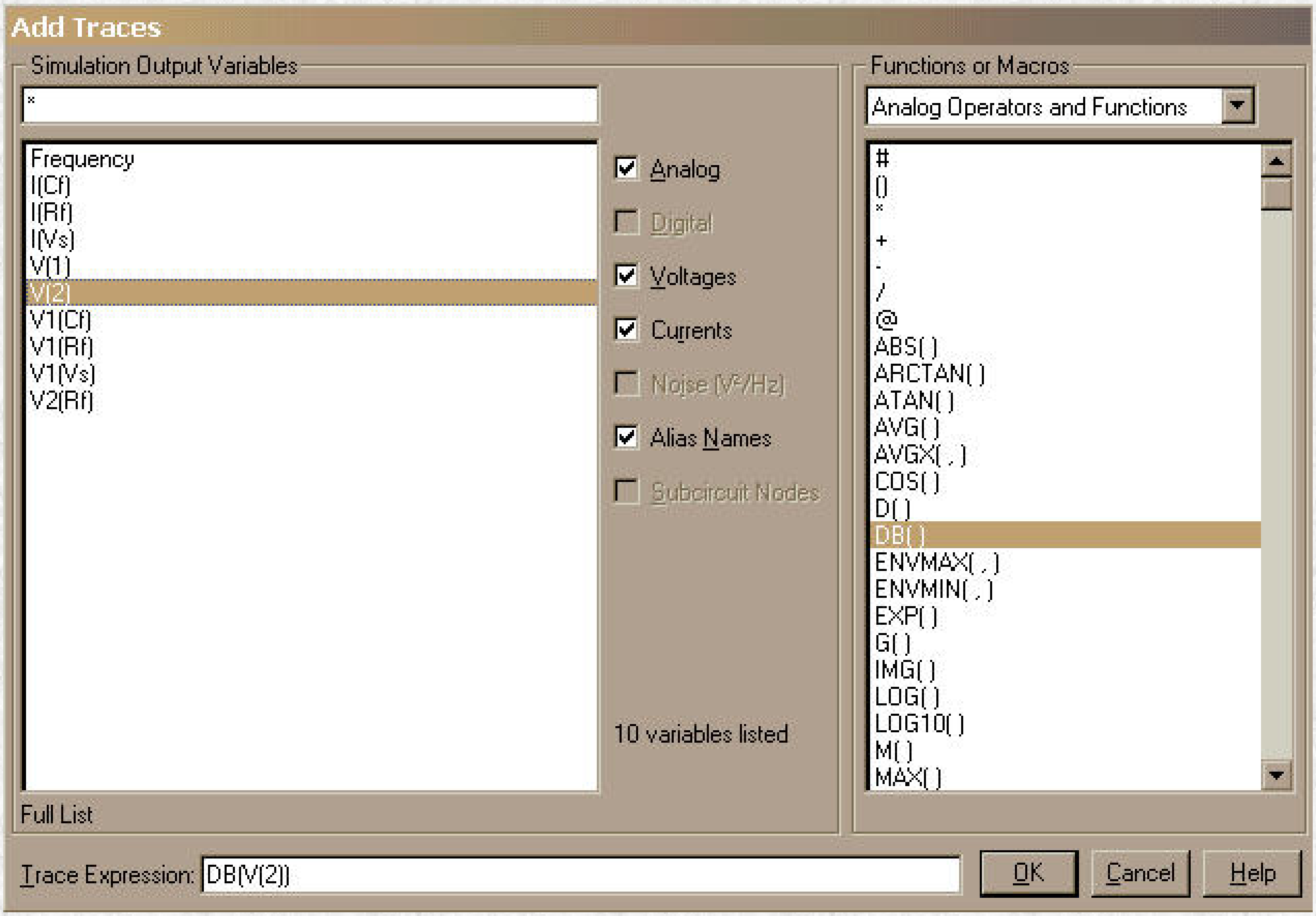
The above circuit is a first-order low-pass filter. Since we want the gain of this filter, it is convenient to make the input voltage 1 volt so the output voltage in numerically equivalent to the gain. However, the post-processer within PROBE is fully capable of performing arithmetic such as dividing the input voltage into the output voltage. The source code for the CIR file follows.

```
First-order low-pass RC filter
Vin 1  0 AC 1.0V
Rf  1  2 1.59
Cf  2  0
.AC DEC 20 100Hz 100kHz
.PROBE
.END
```

After running this in PSpice, we start PROBE, choose "Add" from the "Trace" menu and plot the output voltage. PROBE provides the following graph.



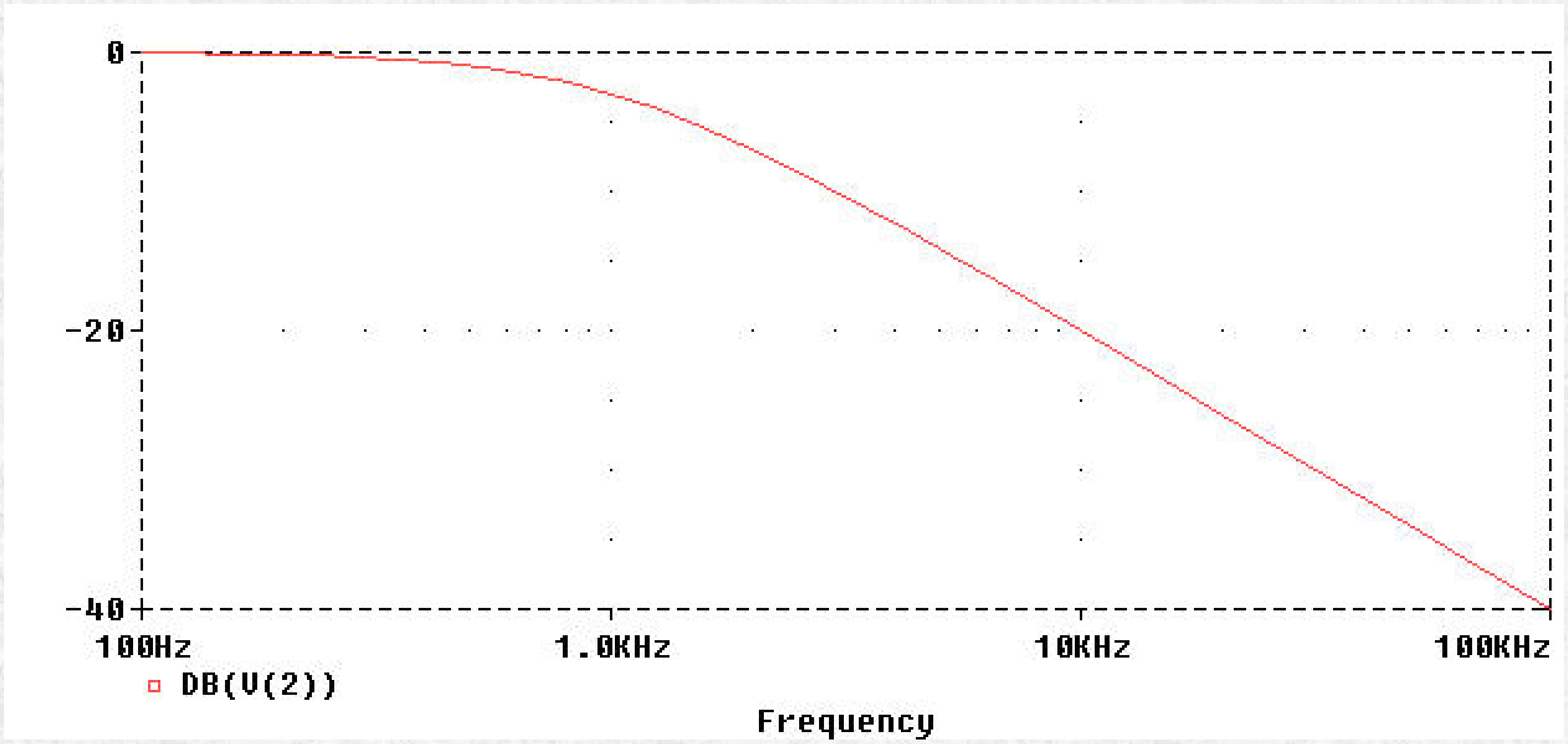
Another option is to have PROBE plot the gain in decibels. To do this choose "Add" from the "Trace" menu in PROBE. Then select the "DB" function in the right-hand column and choose "V(2)" from the left-hand column.



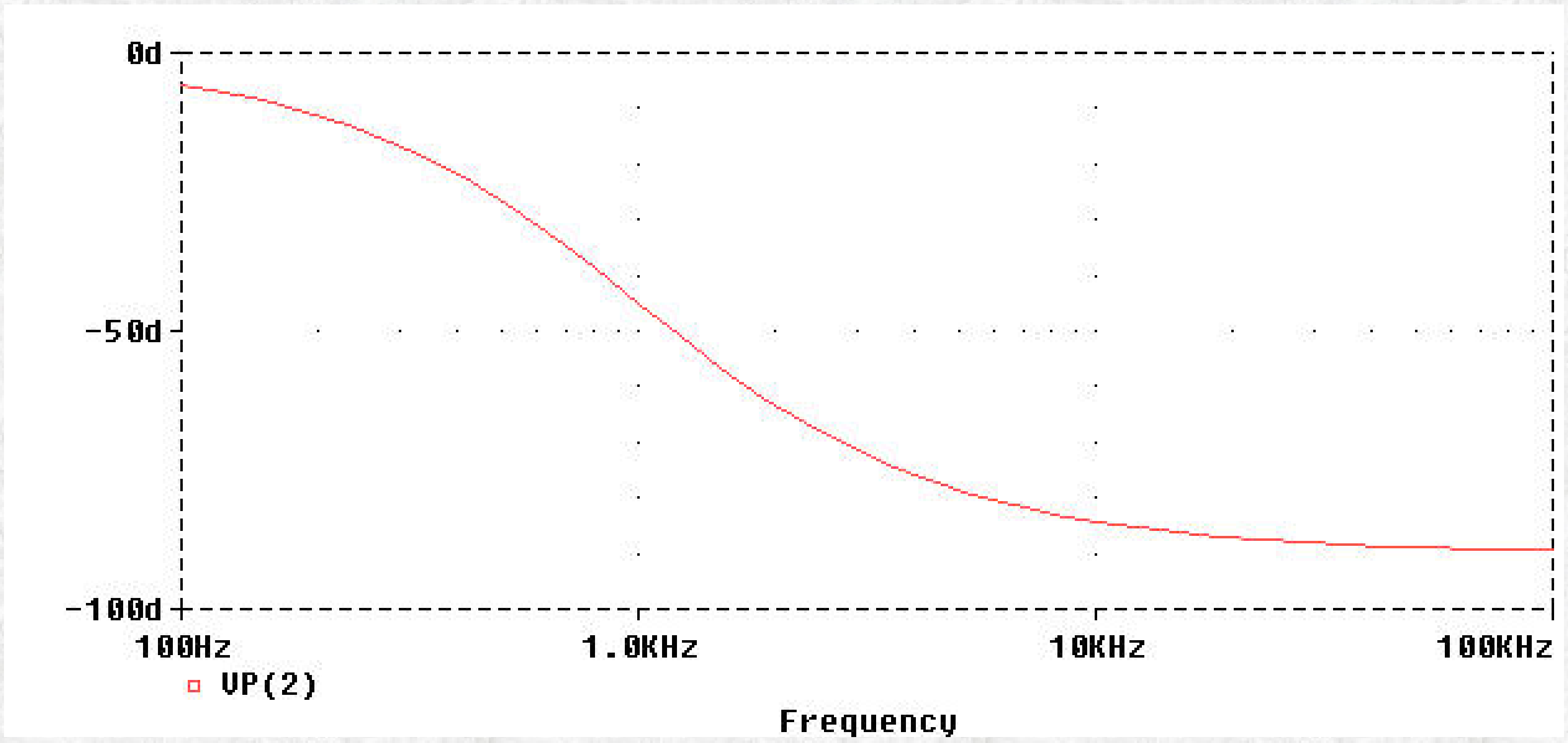
Trace Expression: DB(V(2))

Analog [✓], Digital [ ], Voltages [✓], Currents [✓], Noise (V²/Hz) [ ], Alias Names [✓], Subcircuit Nodes [ ]

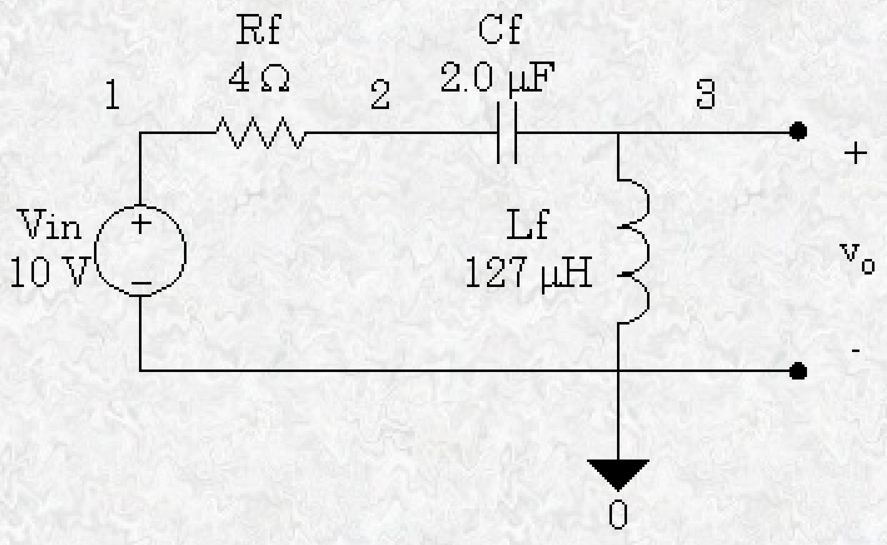
After selecting "OK," you should see the following trace.



Notice that the gain is -3db at a frequency of 1 kHz (the half-power frequency) and declines at 20 dB/decade thereafter. The remaining demonstration for this example is to have PROBE plot the *phase shift* of the low-pass filter as a function of frequency. We simply specify "VP(2)" from the "Add Trace" dialog box. Notice that this is the same format used in the .PRINT AC command in PSPICE. PROBE automatically shows the angles in degrees.

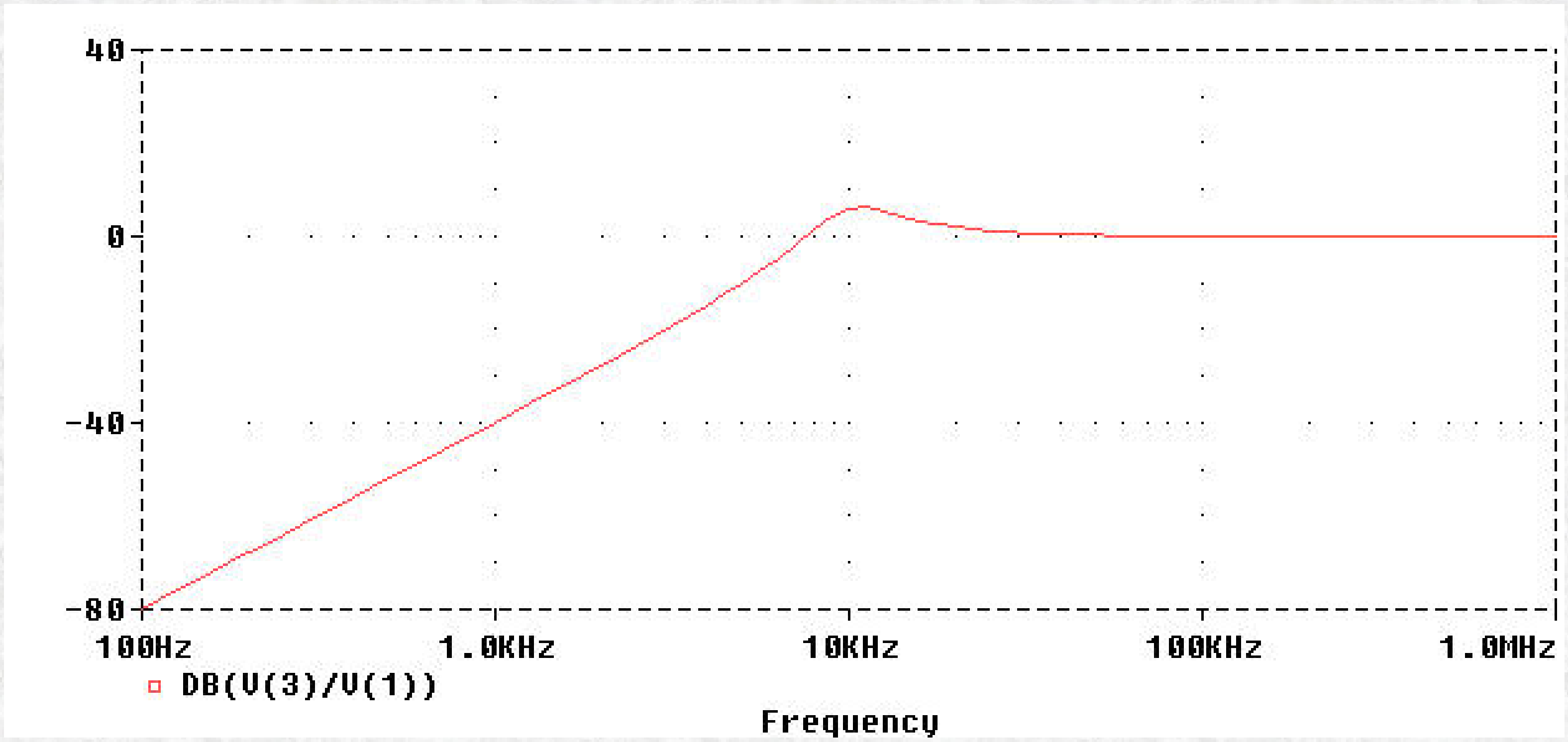


We will now examine a second-order high-pass filter as our second example.

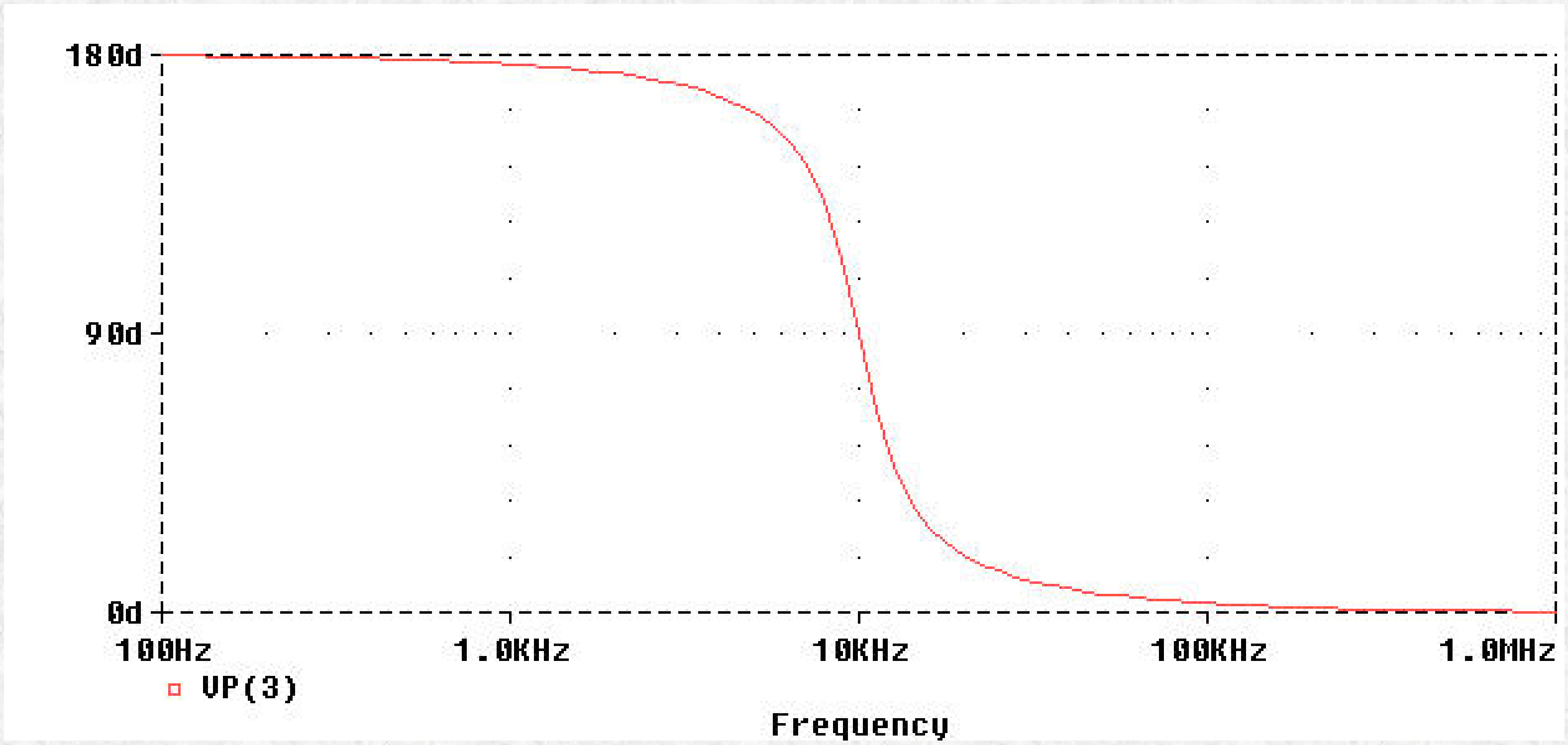


```
Second-Order High-Pass Filter
Vin 1 0 AC 10V
Rf 1 2 4.0
CF 2 3 2.0uF
Lf 3 0 127uH
.AC DEC 20 100Hz 1MEG
.PROBE
.END
```

This time we did not use 1V for the input voltage. Therefore, we will need to have PROBE actually divide the input into the output to get the gain. We show this gain in decibels.



Notice that the gain below the resonant frequency of 10 kHz slopes upward at 40 dB/decade. When we plot the phase shift of this filter, we only need to specify the phase angle of the output voltage since the input voltage was specified at 0 degrees.



Modifying PROBE Display

In PSpice Tutorial No. 4 we promised to show in a future tutorial how to make PROBE display a white background (as you have been seeing in the above examples) instead of a black one . We now fulfill this promise. However, we must make clear that we are modifying a file needed by PSpice and all of its associated programs for successful operation. Therefore, we strongly suggest that you make a back-up copy of this file before you modify it. If this file is lost or destroyed, you may have to reinstall PSpice on your computer. Also, making these modifications requires that you have WRITE as well as READ privileges in the WINDOWS folder of the computer you are using; i.e., don't try this on the school's lab workstations.

The file to be modified depends on which version of PSpice that you have. If you are running version 7 or 8 of the old MicroSim evaluation release, the file name will be "msim_evl.ini" whereas the new OrCAD release 9 evaluation version of PSpice uses the file name "pspiceev.ini" to specify the same start-up information. In either case the file will be found in the "WINDOWS" folder of your computer if you are running Windows 95 or Windows 98. It will be in the WINNT folder if you are running Windows NT or Windows 2000. Be sure to open this file with an *ASCII*-type text editor such as NOTEPAD or the DOS EDIT program. A word processor such as MS Word™ or WordPerfect™ may ruin the file unless you are very careful to save it as a DOS or TXT type file. The safest editor to use is the text editor that comes with PSpice. This is called TEXTEDIT in MicroSim releases 7 and 8. In the new OrCAD release the text editor is run from PSpiceAD itself. In either case, search for the following sequence of statements starting with "[PROBE DISPLAY COLORS]":

```
; (Original version)
.
[PROBE DISPLAY COLORS]
```

```
NUMTRACECOLORS=12
BACKGROUND=BLACK
FOREGROUND=WHITE
TRACE_1=BRIGHTGREEN
TRACE_2=BRIGHTRED
TRACE_3=BRIGHTBLUE
TRACE_4=BRIGHTYELLOW
TRACE_5=BRIGHTMAGENTA
TRACE_6=BRIGHTCYAN
TRACE_7=MUSTARD
TRACE_8=PINK
TRACE_9=LIGHTGREEN
TRACE_10=DARKPINK
TRACE_11=LIGHTBLUE
TRACE_12=PURPLE
.
.
```

There may be minor variations in some of these statements depending on your version; for instance, most MicroSim evaluation releases only define 6 trace colors. The most important two lines to change are the *background* and the *foreground* colors. You need to make the background color BRIGHTWHITE and the foreground color BLACK. If you specify just WHITE for the background color, it will actually be a light gray. This does not look very good when you paste a graph into a document for a report. The following listing is what I use in my own file:

```
; (Modified version)
.
[PROBE DISPLAY COLORS]
NUMTRACECOLORS=12
BACKGROUND=BRIGHTWHITE
FOREGROUND=BLACK
TRACE_1=BRIGHTRED
TRACE_2=BRIGHTBLUE
TRACE_3=BRIGHTGREEN
TRACE_4=BRIGHTMAGENTA
TRACE_5=BRIGHTCYAN
TRACE_6=PURPLE
TRACE_7=MUSTARD
TRACE_8=PINK
TRACE_9=LIGHTGREEN
TRACE_10=DARKPINK
TRACE_11=LIGHTBLUE
TRACE_12=BRIGHTYELLOW
.
.
```

You will find other editable features in this file such as schematic colors. Just be sure to back up the file before you make any changes.

Back to Main Page

Last Modified: undefined

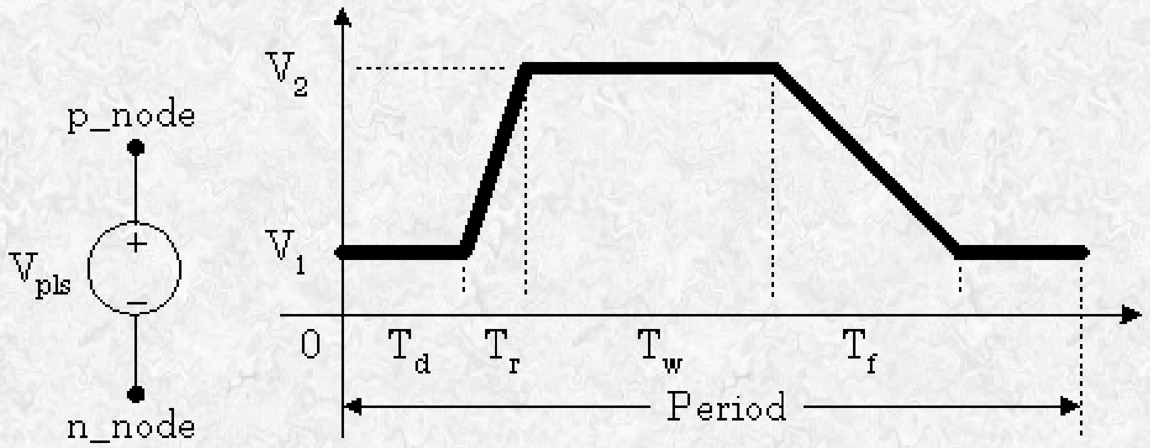
8th Tutorial on PSpice

Special Sources in PSpice

Up to now, these tutorials have discussed only the most basic types of sources. These include the independent DC and AC voltage and current sources, and the simple voltage or current controlled dependent voltage and current sources. At this time, we will introduce three new independent source types and two new dependent source types. This by no means completes the list of possible sources in PSpice, but these new sources will add a great deal of capability to our circuit modeling efforts.

PULSE Sources

This type of source can be either a voltage or a current source. We often use it as a stimulus for transient response simulation of a circuit. It should never be used in a frequency response study because the model assumes it is in the time domain. The designation of the pulse source starts as any other independent source; i.e., the part name must begin with the letter V (for voltage) or I (for current). This is followed by the node names. Then, instead of "DC" or "AC," we use the keyword "PULSE" followed by the necessary parameter list. Items in the parameter list may be separated by spaces or commas. An example of a pulse type of voltage source follows:

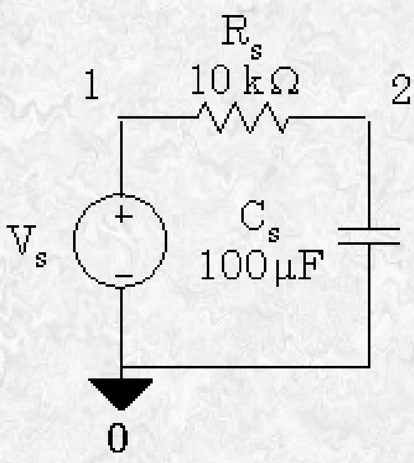


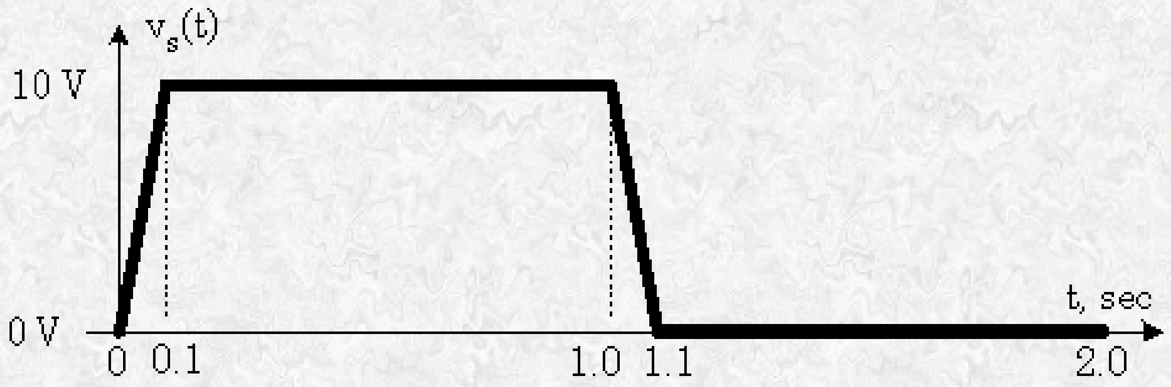
The parameters for the pulse (to be entered in the order given) are:

- **V_1** is the value when the pulse is not "on." This can be zero or negative as required. For a pulsed current source, the units would be "amps" instead of "volts."
- **V_2** is the value when the pulse is fully turned on. This can also be zero or negative. (Obviously, V_1 and V_2 should not be equal.) Again, the units would be "amps" if this were a current pulse.
- **T_d** is the time delay. The default units are seconds. The time delay may be zero, but not negative.
- **T_r** is the rise time of the pulse. PSpice allows this value to be zero, but zero rise time may cause convergence problems in some transient analysis simulations; i.e., there is a very good reason for the existence of this parameter. The default units are seconds.
- **T_f** is the fall time in seconds of the pulse. See note on T_r before setting this to zero.
- **T_w** is the pulse width. This is the time in seconds that the pulse is fully on.

- **Period** is the total time in seconds of the pulse. Be aware that the pulse repeats if the simulation time exceeds the period.

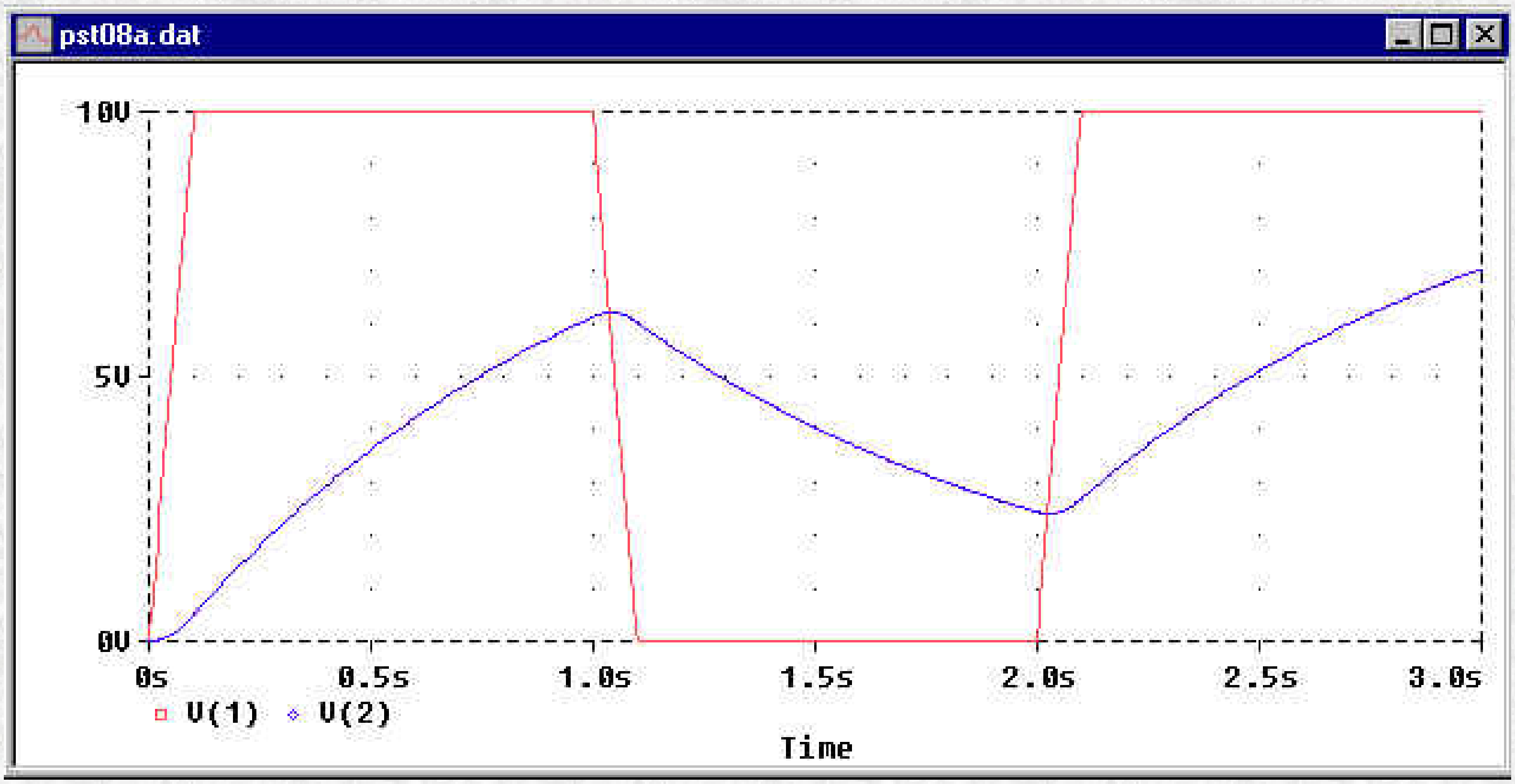
A case study now follows for a simple circuit with a pulsed source:





```
Transient response of a low-pass filter
*            V1 V2  Td Tr    Tf    Tw    Per.
Vs 1 0 PULSE(0V 10V 0s 100ms 100ms 900ms 2s)
Rs 1 2 10k
Cs 2 0 100uF IC=0V
.TRAN 5ms 3s 0s 5ms UIC
.PROBE
.END
```

Discussion: V_1 is set to zero for this case and the pulse is at the 10-volt level (V_2) for T_w = 900 ms. Note that the simulation time (3s) is greater than the period (2s). The red trace shown below represents the pulse from the voltage source while the blue trace represents the response voltage across the capacitor.



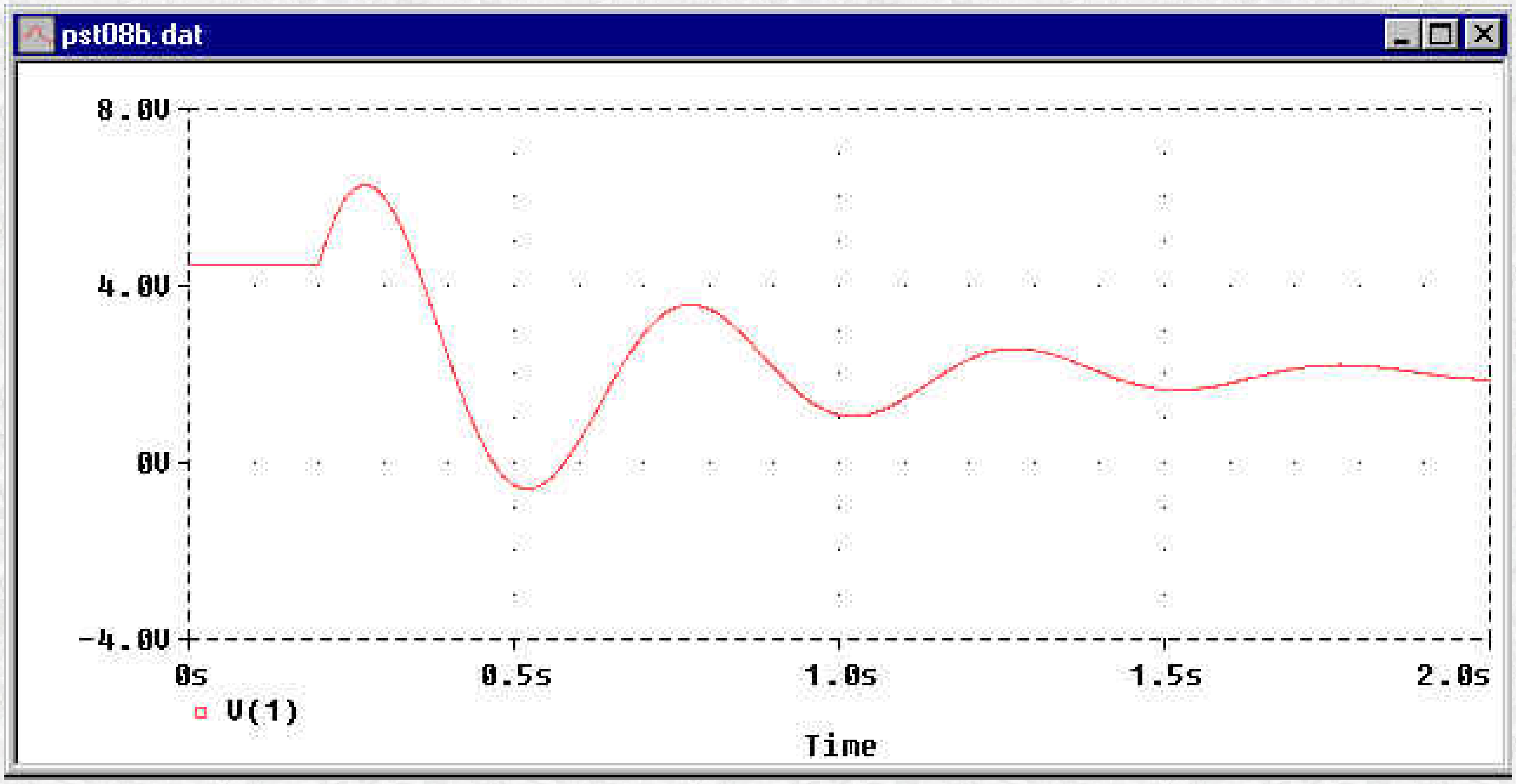
[Back to Main Page](#)

SIN Sources

The SIN type of source is actually a damped sine with time delay, phase shift and a DC offset. Usually, we only want a simple sine wave to model an AC power source in a *transient* analysis simulation. For the record, here is the whole definition with all six parameters explained. The following represents a voltage source, but the first two parameters could readily be changed to currents in amps to make this a current source. N. B.: Do not use this type of source for a phasor or frequency sweep analysis.

- $\mathbf{V_o}$ is the DC offset value. It should be set to zero if you need a pure sinusoid.
- $\mathbf{V_a}$ is the undamped amplitude of the sinusoid; i.e., the peak value measured from zero if there were no DC offset value.
- $\mathbf{f_r}$ is the frequency in Hz of the sinusoid.
- $\mathbf{T_d}$ is the time delay in seconds. Set this to zero for the normal sinusoid.
- $\mathbf{D_f}$ is the damping factor in s^{-1}. Also set this to zero for the normal sinusoid.
- θ is the phase advance in degrees. Set this to 90 if you need a cosine waveform.

Below is a sample waveform where: V_o = 2V, V_a = 5V, F_r = 2Hz, T_d = 200ms, D_f = $2s^{-1}$ and θ = 30°.



Here is the circuit listing that produced the above waveform:

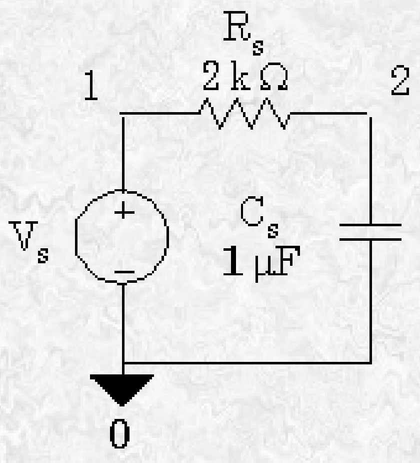
```
Example of a SIN source
*           Vo Va Fr  Td    Df  θ
Vs 1 0 SIN(2V 5V 2Hz 200ms 2Hz 30d)
RS 1 0 1MEG
.TRAN 1ms 2s 0s 1ms UIC
.PROBE
.END
```

The way PSpice uses the parameters is:

$$v(t) = V_o + V_a \sin\left[2\pi\left(f_r\left(t - T_d\right) + \frac{\theta}{360^\circ}\right)\right] e^{-(t-T_d)D_f}, \quad for \quad t \geq T_d$$

$$v(t) = V_o + V_a \sin\left(\frac{\pi\theta}{180^\circ}\right), \qquad for \ \ t < T_d$$

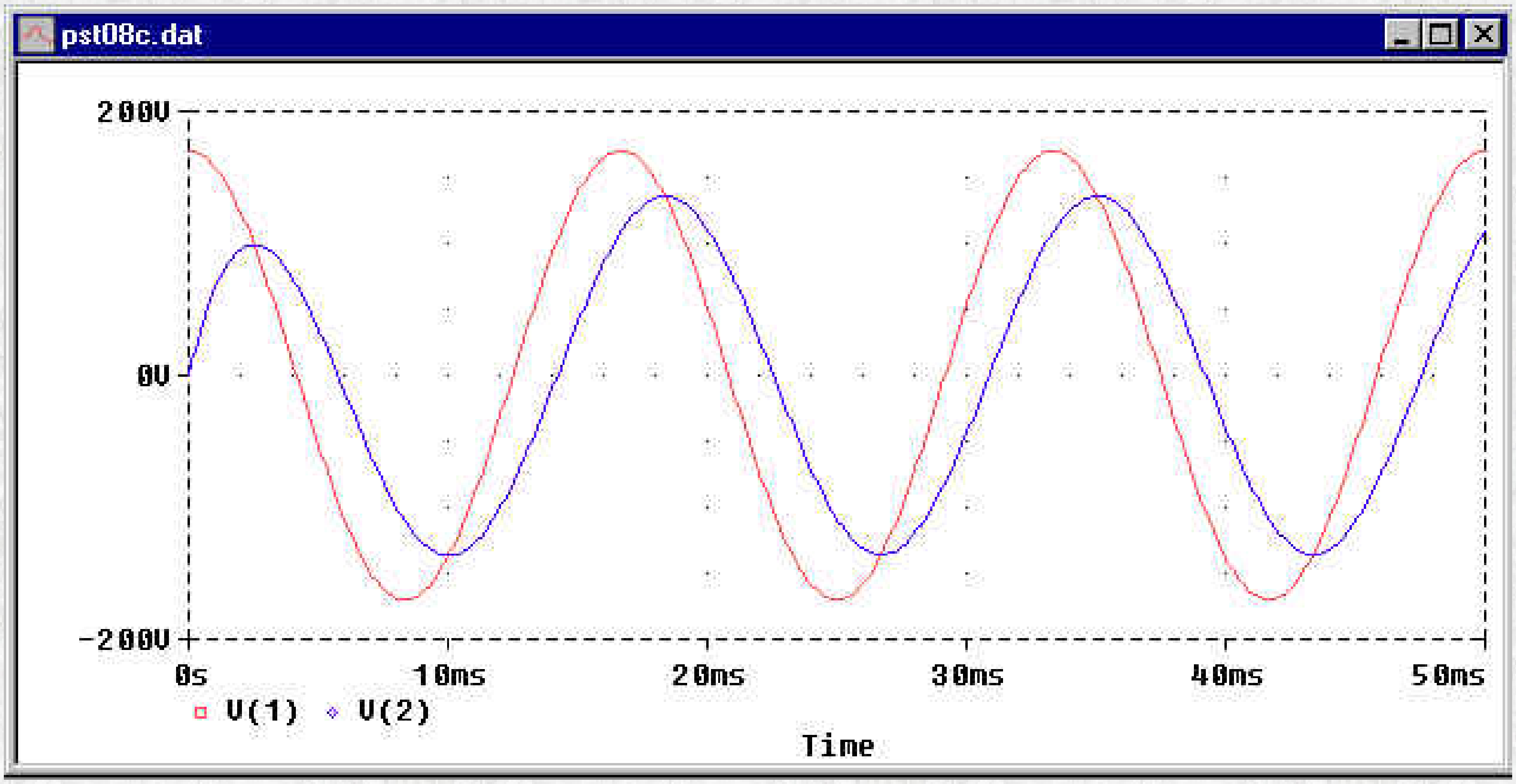
Now that we got that out of the way, let's see an example of a plain sinusoid as it may be used in a power system transient response simulation.



Let the above circuit commence with a cosine waveform starting at t = 0. There is no stored energy in the capacitor. The input data for this circuit would be:

```
Transient Response of a Sinusoid
*          Vo Va   Fr   Td Df  θ
Vs 1 0 SIN(0V 170V 60Hz 0s 0Hz 90d)
RS 1 2 2k
Cs 2 0 1uF IC=0V
.TRAN 100us 50ms 0s 100us UIC
.PROBE
.END
```

Note that the normal usage of this source type is to set V_o, T_d and D_f to zero. Since a cosine was required here, we set the phase advance to 90 degrees. If we were willing to use a sine instead of a cosine, the last three parameters would have been equal to zero and could have been omitted. The Probe output for this case is shown below. The red waveform is the cosine source voltage and the blue waveform is the capacitor response voltage.



Back to Main Page

PWL Source

The PWL source is a PieceWise Linear function that you can use to create a waveform consisting of straight line segments drawn by linear interpolation between points that you define. Since you can use as many points as you want, you can create a very complex waveform. This source type can be a voltage source or a current source. Like all the other independent sources, the part name must start with the letter "V" for a voltage source and the letter "I" for a current source. The syntax for this source type is flexible and has several optional parameters. The required parameters are two-dimensional points consisting of a time value and a voltage (or current) value. There can be many of these data pairs, but the time values must be in ascending order, and the intervals between time values need not be regular. The two optional parameters are "DC" and "AC." The use of an AC parameter with this source is very dubious since it is intended for use with transient analyses, and any AC value would be ignored. However, if you want to change the analysis type and use an AC source the AC parameter would be the only thing used.

Let's examine a few examples of this source type:

```
*  +n -n  dc=10  ac=1     point 2 point 3 point 4
Vx 12 24 DC 10V AC 1V PWL(1ms 12V 3ms 15V 8ms 4V)
```

In the above example, the AC parameter will be ignored in a transient analysis. The DC parameter will be paired with time = 0s to create the first data point. If, for some peculiar reason, you run an AC sweep with this source, it would be a simple 1V AC source at the frequencies designated. An equivalent usage of this source that is more clear will now be presented.

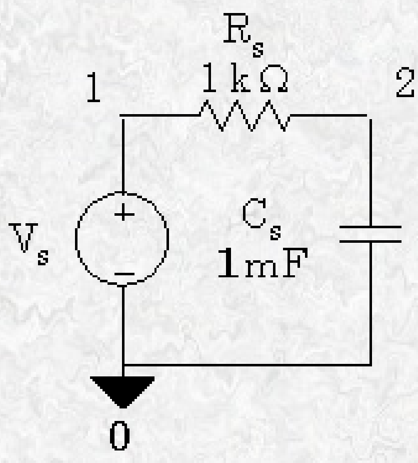
```
*  +n -n     point 1 point 2 point 3 point 4
Vx 12 24 PWL(0ms 10V 1ms 12V 3ms 15V 8ms 4V)
```

Note that we have discarded the unneeded AC parameter and obviated the need for the DC parameter by entering the starting point in the PWL list. Additional flexibility exists in the various ways PSpice will accept parameter lists. We can use commas, spaces or tabs as we wish; and the parentheses enclosing the parameter lists are not required. Here are a few examples that are equivalent to the first two.

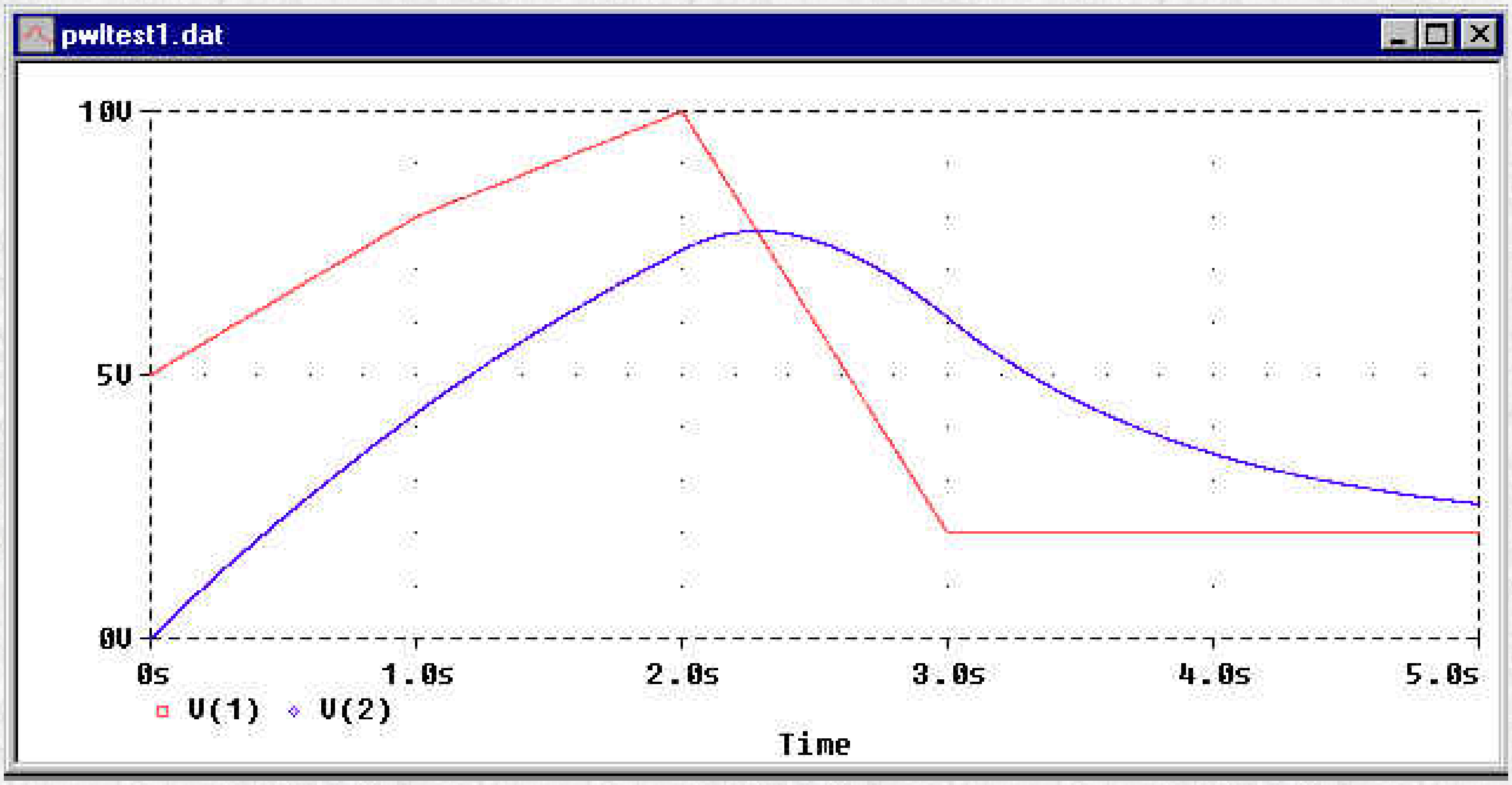
```
Vx 12 24 PWL 0ms,10V 1ms,12V 3ms,15V 8ms,4V; <== no ()
Vx 12 24 PWL(0ms,10V,1ms,12V,3ms,15V,8ms,4V)
Vx 12 24 PWL(0ms,10V 1ms,12V 3ms,15V 8ms,4V)
```

If the time span of the transient analysis exceeds the time value of the last point, the source behaves as a DC source set at the voltage (or current) value of the last point for the rest of the simulation. It does <u>not</u> repeat its cycle as does the Pulse source.

Finally, we will create an example using a simple PWL source:



```
PWL Example
Vs 1 0 PWL(0s,5V 1s,8V 2s,10V 3s,2v)
RS 1 2 1.0k
Cs 2 0 1mF IC=0V
.TRAN 1ms 5s 0s 1ms UIC
.PROBE
.END
```



In the above figure, the red trace is the source waveform we created and the blue waveform is the response voltage across the capacitor.

Although there are several more independent source types, the exponential and SFFM sources to mention a couple, we will postpone their explanation until a future tutorial. At this point, we will benefit more from the understanding of two new *dependent* source types: the Laplace dependent source and the Table dependent source.

Back to Main Page

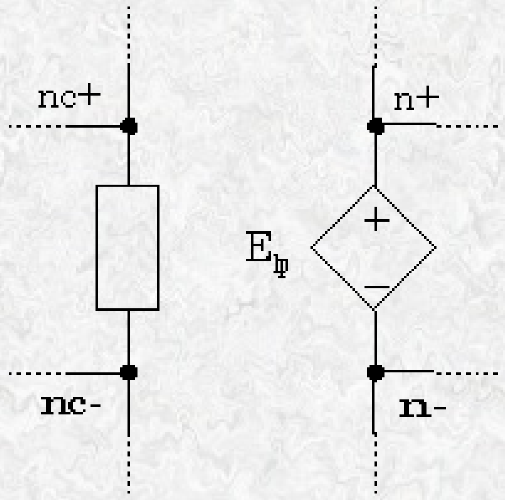
Laplace Source Type

The Laplace source is a transfer function actuator. It takes a mathematical expression of circuit voltages or currents as its input and produces its output on the basis of the transfer function defined in the Laplace domain. This can replace the detailed modeling of the circuit which accomplishes the same thing. This source is usually used for frequency response filter modeling, but also works in the time domain. A Laplace source can be an expression-controlled voltage source or an expression-controlled current source. The expression can be a function of the voltage between a pair of nodes or the current flowing through a voltage source. Usually, we just use a node voltage as the input expression.

Suppose we have a transfer function, H(s), defined in the Laplace domain as:

$$H(s) = \frac{1000}{s + 1000}$$

This could represent a low-pass filter with a cutoff frequency of 1000 radians/sec (or 159 Hz). The elements used to model this in a circuit application could appear as follows:

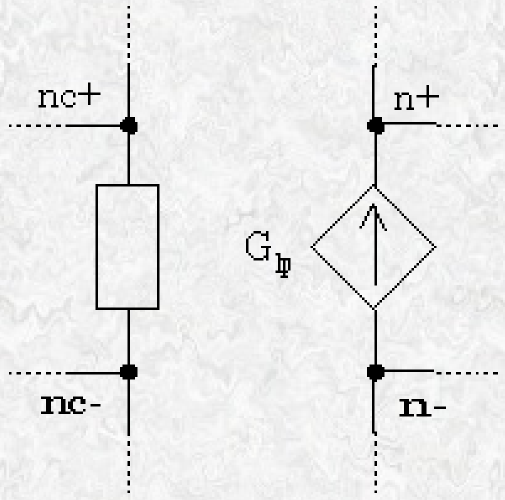


In the above circuit fragment, the input is the voltage drop from node nc+ to nc- and the output appears as the dependent source voltage at element E_{lp}. The input code that will reside in the circuit file would be:

```
*                  input      transfer function
Elp n+ n- LAPLACE {V(NC+,NC)}={1000/(s+1000)}
```

An interesting quirk of this source is that there *must* be a space immediately to the right of the keyword "LAPLACE" and before the left curly brace that starts the input expression. Another quirk is that the transfer function enclosed in curly braces must reside on one line. Also notice how the transfer function is encoded.

The current source version of this source could be:



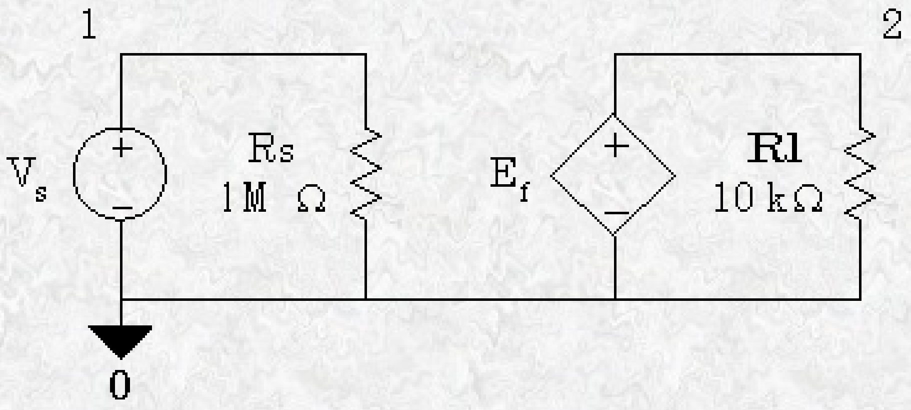
The circuit file data for the above fragment would be:

```
*                  input      transfer function
Glp n- n+ LAPLACE {V(NC+,NC)}={1000/(s+1000)}
```

Now, we will construct an actual example of the Laplace source used as a filter tester. The transfer function is for a two-pole low-pass Butterworth filter with a corner frequency of 300 Hz.

$$H(s) = \frac{3.553 \cdot 10^6}{s^2 + 2666s + 3.553 \cdot 10^6}$$

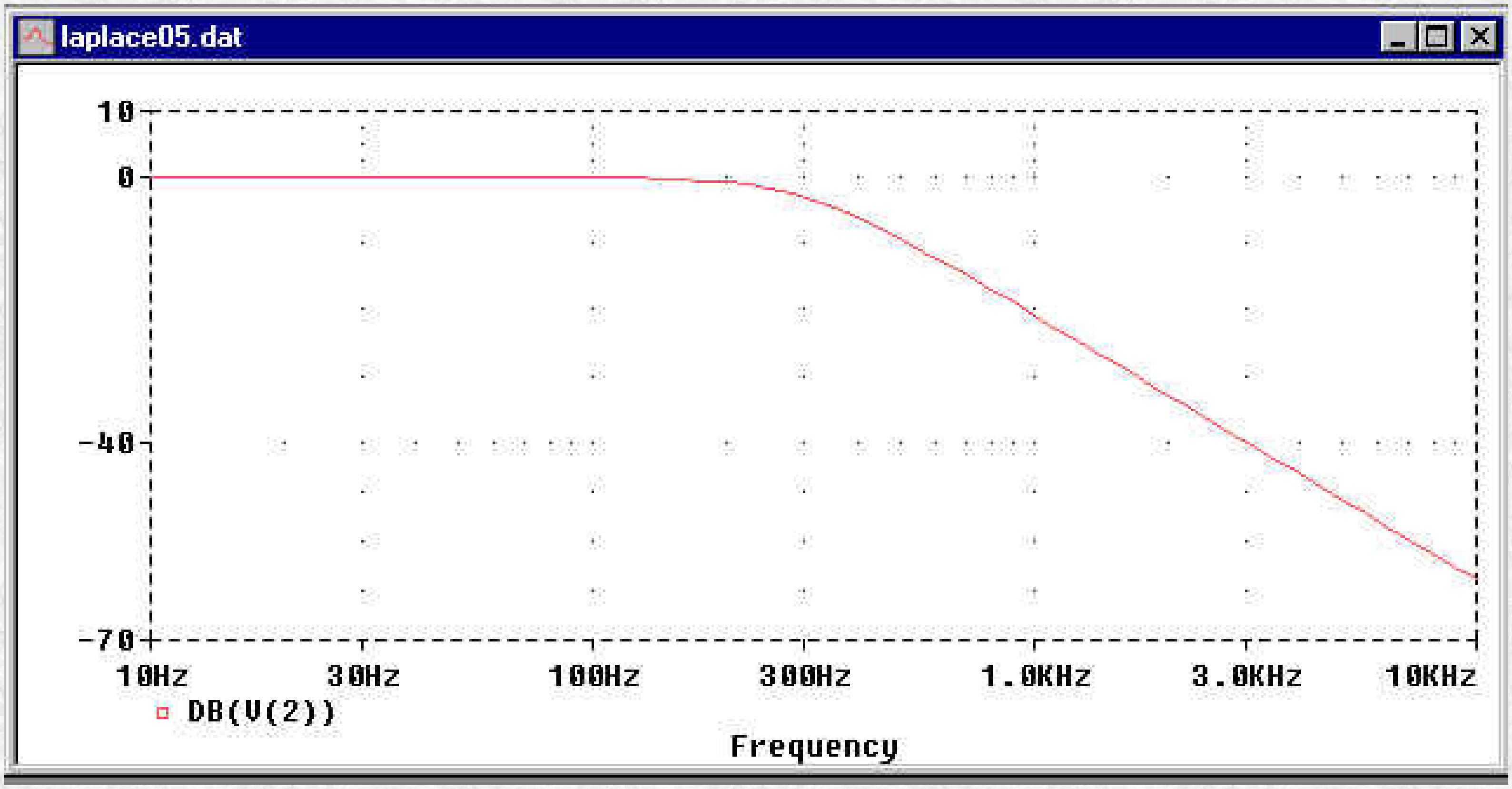
A minimal circuit to test this filter is shown below.



The circuit file code to model the above circuit is shown below. We will let the frequency sweep span three decades. Note the method for encoding the transfer function as an inline statement. This is not unlike a line of programming code.

```
Laplace Example
Vs 1 0 AC 1V
RS 1 0 1MEG
Ef 2 0 Laplace {V(1)}={3.553E6/(s^2+2666*s+3.553E6)}
Rl 2 0 10k
.AC DEC 20 10Hz 10kHz
.PROBE
.END
```

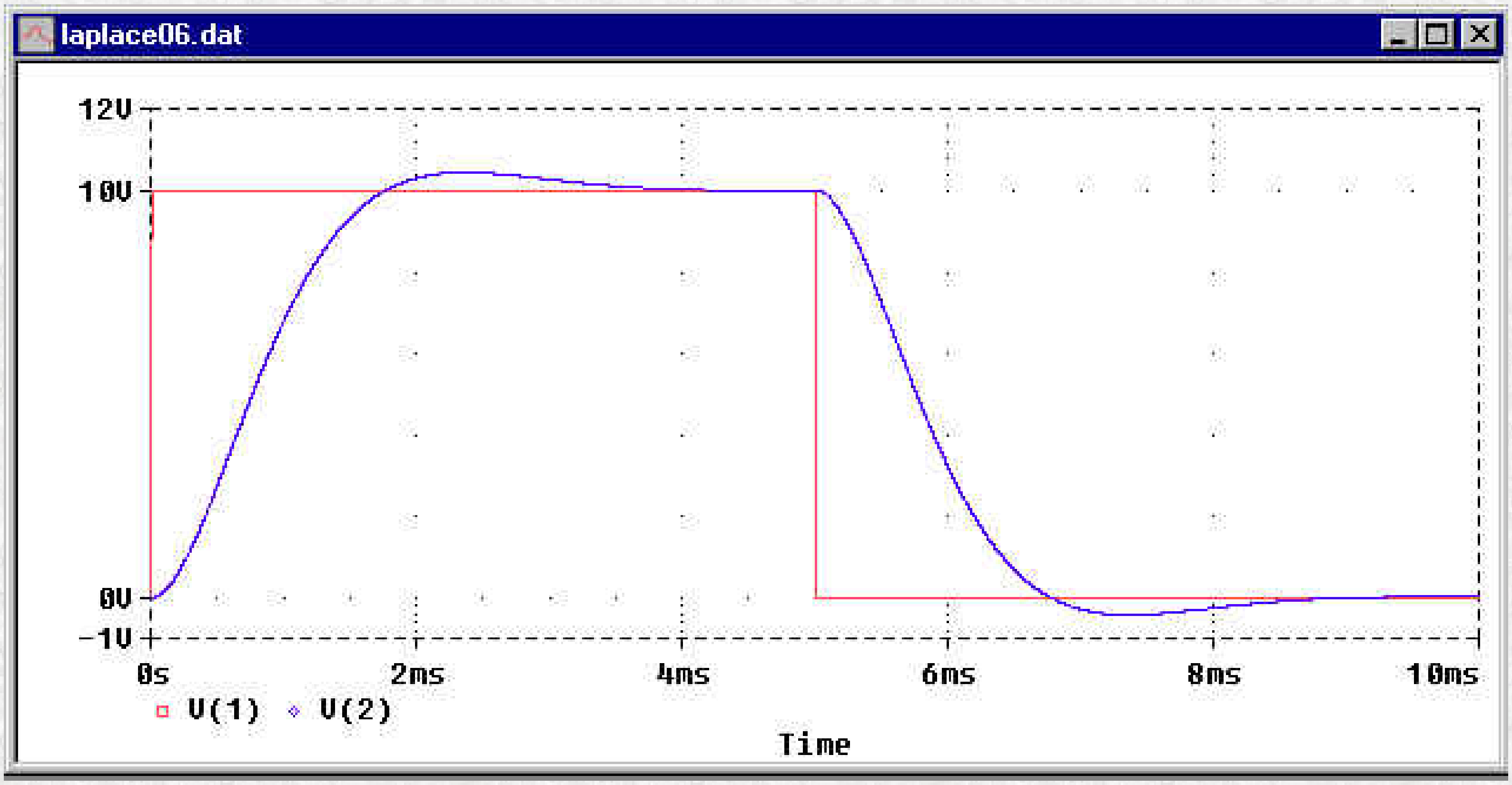
Now, we will examine the Bode plot for this filter as produced by PROBE.



To demonstrate that the Laplace source can work well in the time domain, we excite the same filter with a pulse instead of a frequency sweep. The circuit file code for this is:

```
Laplace Transient Example
Vs 1 0 PULSE(0V 10V 0s 10us 10us 4.99ms 10ms)
RS 1 0 1MEG
Ef 2 0 Laplace {V(1,0)}={3.556E6/(s^2+2666*s+3.553E6)}
Rl 2 0 10k
.TRAN 1us 10ms 0s 1us
.PROBE
.END
```

And the results from PROBE are:



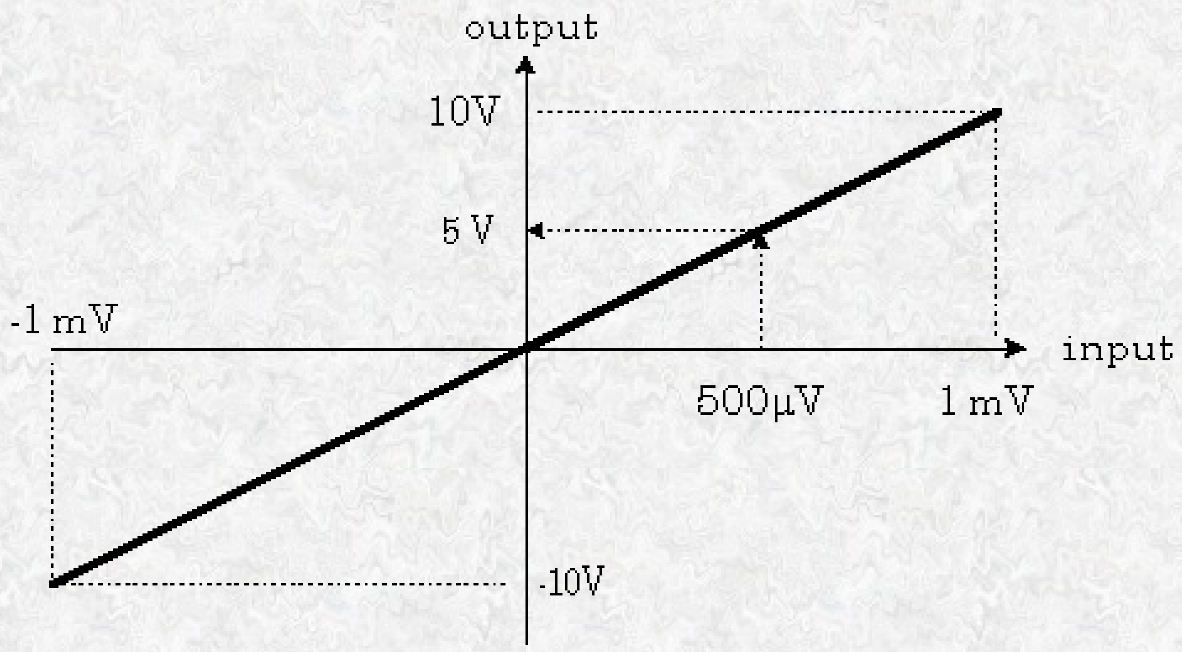
where the red trace is the input pulse and the blue trace is the filter output response.

The Laplace source allows us to model complex electromechanical systems or control systems in terms of the block diagrams where the transfer functions are known. There is no need to define the complex circuitry within the block. However, it should be noted that the Laplace source makes heavy use of resources. There is a price for the convenience.

Back to Main Page

Table Source Type

This source type is one of the most flexible and powerful modeling methods offered by PSpice. It is a dependent source because its output is dependent on the voltages or currents used in the input expression. Instead of a mathematical function, a lookup from a table of points is executed from the input expression. Linear interpolation is used whenever the input expression's value lies between two table values. Consider the following graph:

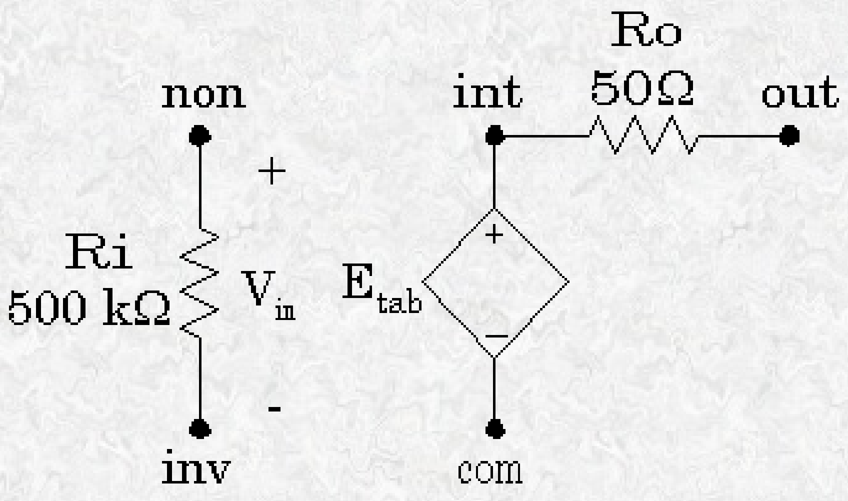


Only two points are needed to specify this graph: (-1mV, -10V) and (1mv, 10V). Any input value between -1mV and +1mV has a corresponding value from the graph. For example, an input value of 500µV would produce an output voltage of 5V. The way PSpice interprets the table data, an input value outside the defined range will simply return the closest output value; i.e., an input value of 2mV would still return 10V for the output. If we examine this graph, we find a ratio of 10^4 between output and input. Thus our graph describes a gain of 10^4 between the range of -1mV to +1mV of input. We could use this to define an op-amp with an open loop gain of 10^4 that saturates at 10 volts. Here is how we could encode this Table type voltage source in a PSpice circuit file:

```
*    n+ n-       input    in    out   in  out
Etab 2   0 TABLE {V(1)}=(-1mV,-10V) (1mv,10V)
```

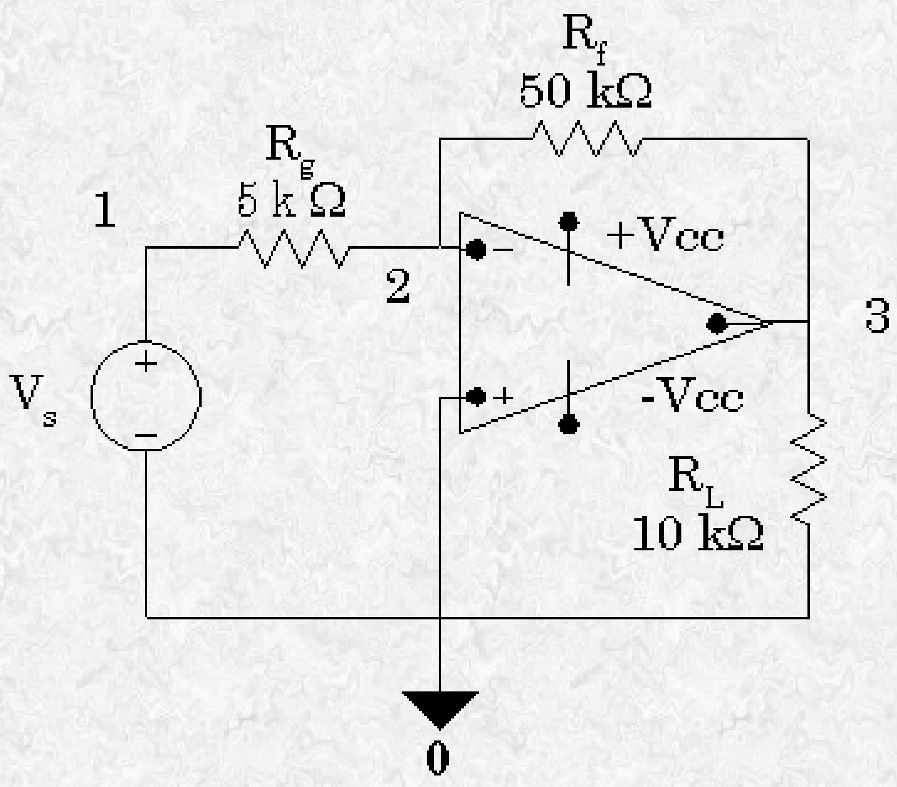
where the voltage at node 1 is the input. Note that the table data pairs require an input value, then an output value. Grouping the data pairs with parentheses and commas as shown above is strictly for human convenience. PSpice allows a great deal of stylistic freedom in accepting tabular data. One quirk you must remember, however, is that a space is required between the keyword "TABLE" and the left curly brace that starts the input expression. This is similar to the Laplace source syntax requirements. You may use continuation lines for the input/output data pairs. Since linear interpolation is used between data pairs, these data pairs must be ordered so that the input values are in ascending order.

Now, we will define an opamp subcircuit using this source with a 500 kΩ input resistance and a 50 Ω output resistance:



```
.SUBCKT OpAmpSat non inv out com
Ri non inv 500k
Ro int out 50.0
Et int com TABLE {V(non,inv)}=(-1mV,-10V) (1mV,10V)
.ENDS
```

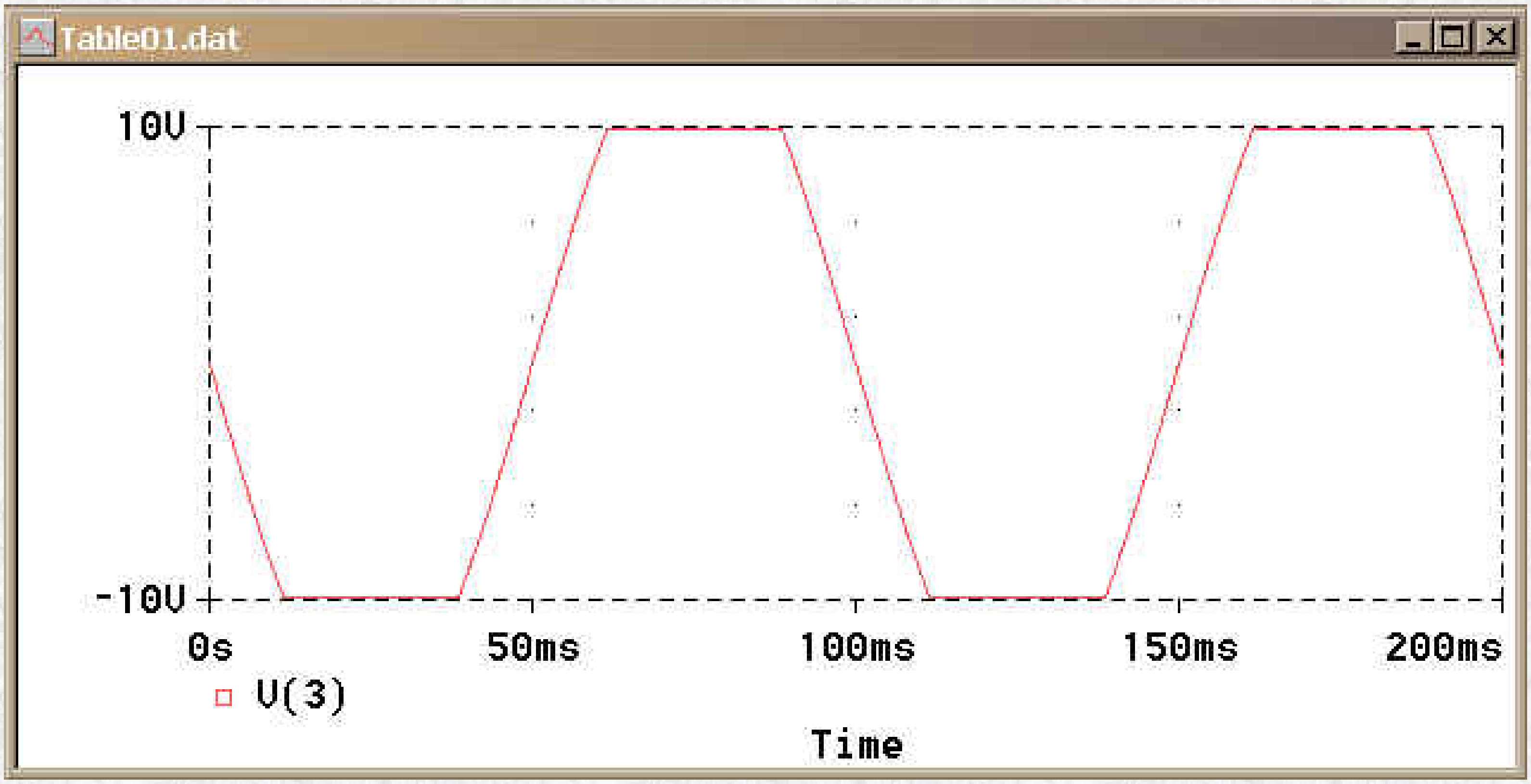
Due to the saturation effects enabled by the TABLE paradigm, this subcircuit will behave as an opamp whose +Vcc and -Vcc values are +10V and -10V respectively. Let's try it out in an inverting amplifier circuit. We will overdrive the input with a SIN source to see the saturation effects.



```
Saturated Opamp Example
Vs 1 0 SIN(0V 1.5V 10Hz); last 3 params = 0
```

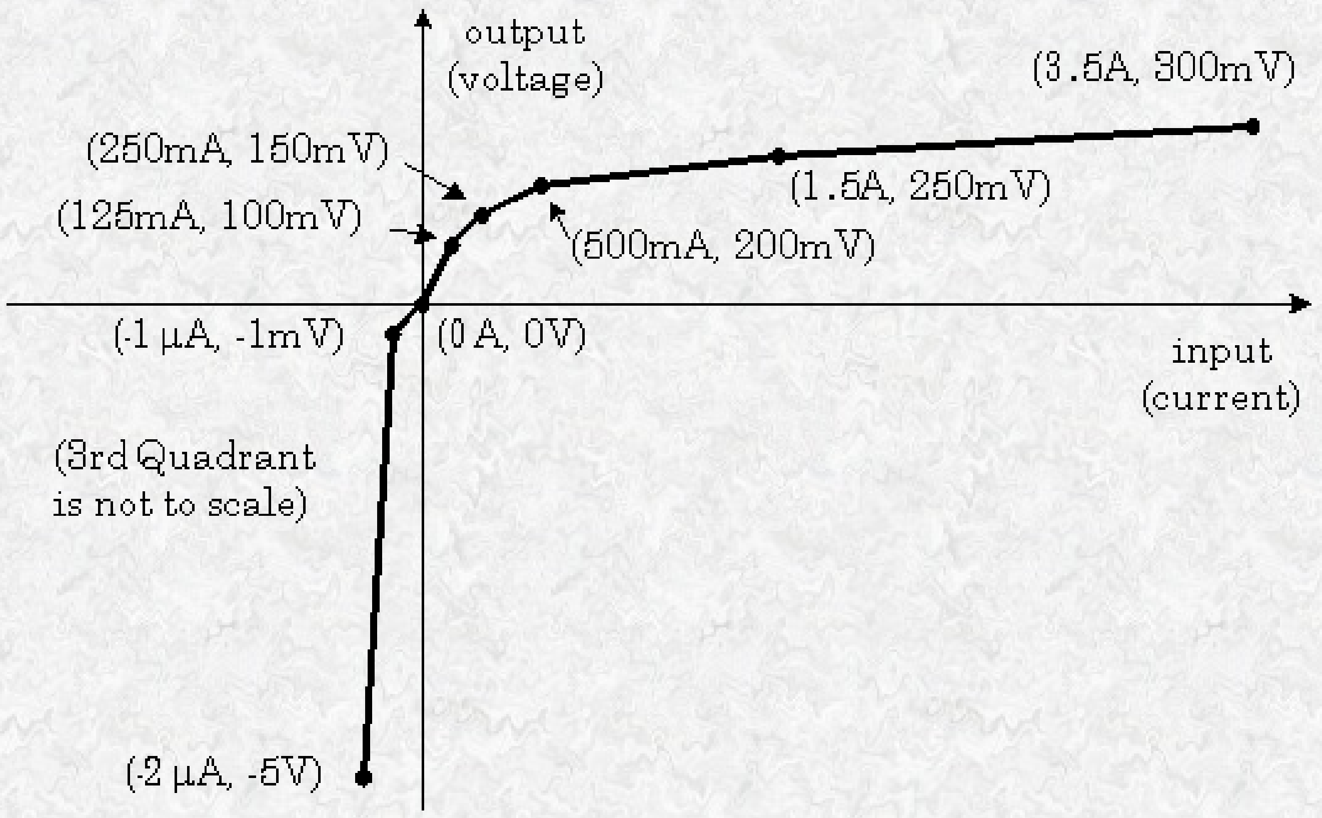
```
Rg 1 2 5k
Rf 2 3 50k
RL 3 0 10k
Xp 0 2 3 0 OpAmpSat; must include above subckt def.
.TRAN 100us 200ms 0s 100us
.PROBE
.END
```

The output voltage from PROBE is shown below:

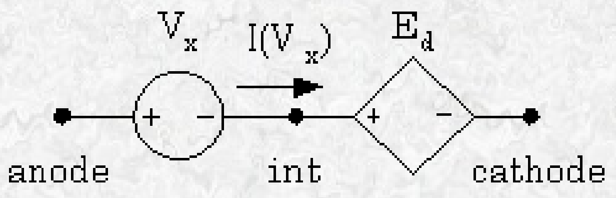


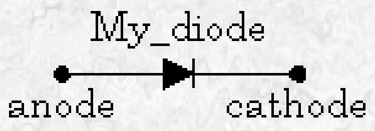
The closed loop gain of this inverting amplifier circuit is -10. Since the peak input voltage is 1.5V, the output peaks would be 15V were it not for the saturation effect. This opamp circuit uses far less resources than the somewhat more accurate library models included with PSpice.

A more sophisticated example of an input/output table follows:



The above graph is an approximate representation of the v-i characteristic of a diode. We can create a diode model from a Table type dependent voltage source whose input is the current through the diode branch. To do this, we will need to resort to the old trick of using a zero-value DC voltage source to measure the current. The elements would be connected as follows:

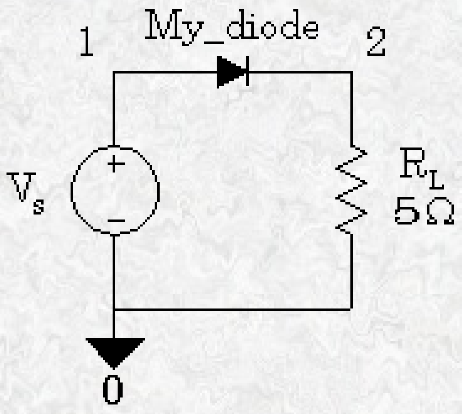




We'll create a subcircuit for our diode model:

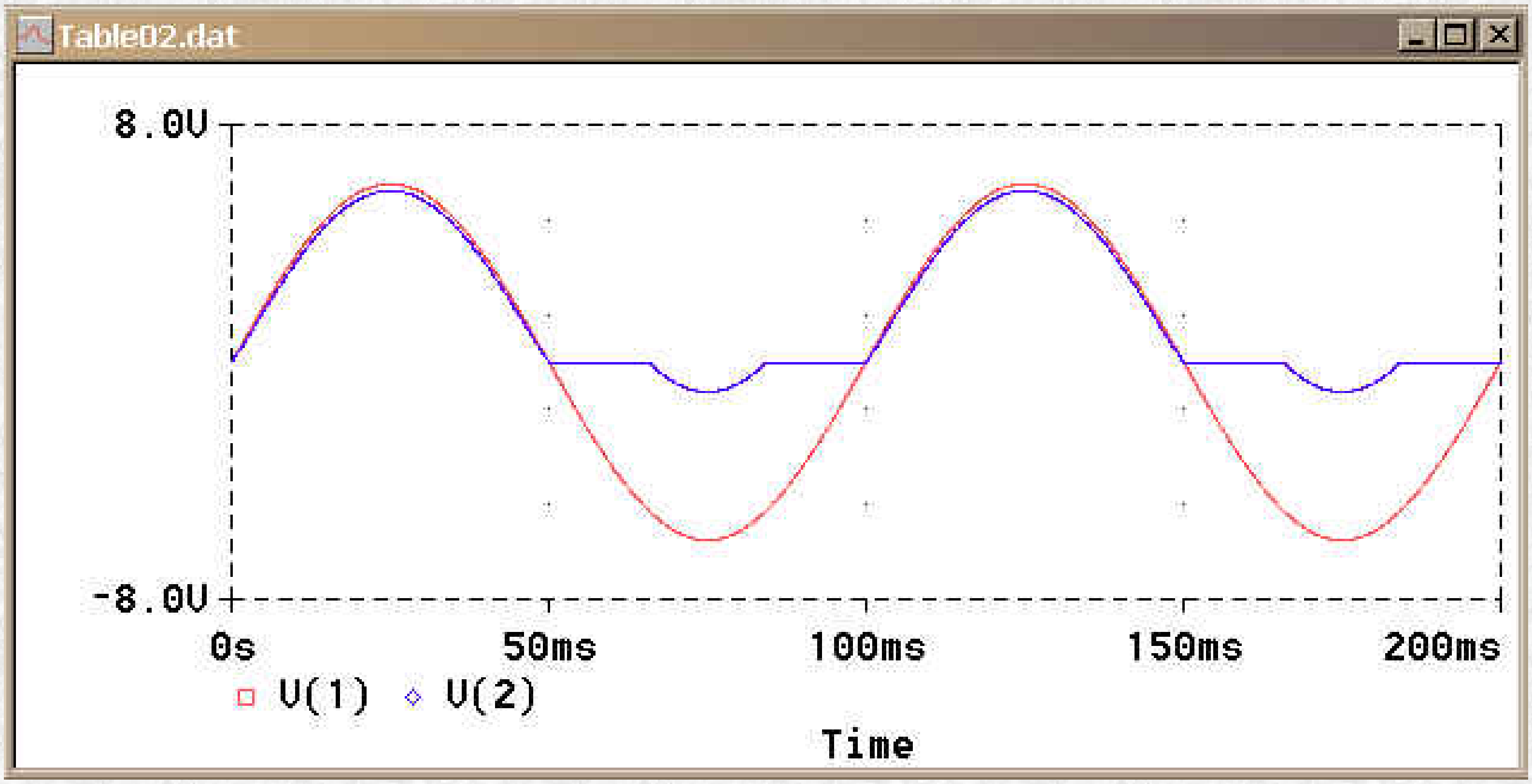
```
.SUBCKT My_diode anode cathode
Vx anode int DC 0V; use this to measure current
Ed int cathode TABLE {I(Vx)}=(-2uA,-5V) (-1uA,-1mV)
+ (0A,0V) (125mA,100mV) (250mA,150mV) (500mA,200mV)
+ (1.5A,250mV) (3.5A,300mV)
.ENDS
```

Now let's test our diode subcircuit in a simple half-wave rectifier circuit.



```
Diode Simulation with Table
Vs 1 0 SIN(0V 6V 10Hz)
Rl 2 0 5.0
Xd 1 2 My_diode; must include above SUBCKT
.TRAN 100us 200ms 0s 100us
.PROBE
.END
```

The PROBE plot for this simulation follows.



The red trace represents the 6-volt peak-value sine wave and the blue trace represents the voltage across the load resistor. The voltage difference between the two traces during the positive half-cycles is the forward voltage drop across the diode. Since the largest negative voltage given in the table was only 5 volts, there is a Zener breakdown when the diode's reverse voltage exceeds 5 V. If you do not wish to model the Zener diode breakdown effect, simply set the output value of the first data pair in the table to a large enough negative value. Bear in mind, that the diode models included in the PSpice library are more sophisticated than this, but they require more resources.

Although these two TABLE source examples did not utilize the current source capability, you can easily create Table-type current sources by replacing "E" by "G" in the part name and remembering that the output values in the data pairs have become currents.

There are almost limitless possibilities for modeling electronic parts with these sources.

Back to Main Page

Last Modified: undefined