UNIVERSITY OF CALIFORNIA AT BERKELEY
College of Engineering
Department of Electrical Engineering and Computer Sciences

HSPICE Tutorial

Contents

1 Introduction

This tutorial was written specifically for the EE105 course at UC Berkeley. It outlines how to run HSPICE in Windows and UNIX, then delves into the details of creating netlists, simulating them, and using WaveView to plot the results. The appendix to this document serves as a syntax reference that may be useful throughout the lab experiments.

HSPICE is just a program that takes in a netlist (a simple text file containing a circuit description and analysis options) and outputs the analysis it has performed on the circuit. An HSPICE netlist typically has an `.sp` extension, e.g. `circuit.sp`. Although HSPICE produces many output files, the only one that you

will need to examine for the labs is the `.lis` extension file, e.g. `circuit.lis`. This file contains all of the important results from the HSPICE analysis—operating points, measurement results, error messages, etc. Typically after simulating a circuit, it is best to check this file first to ensure there were no errors in your netlist.

The other files that HSPICE generates are used by WaveView Analyzer, or WaveView for short. WaveView is a program that allows you to graphically plot the results of the HSPICE analysis. However, you will need to explicitly command HSPICE to generate the extra files that WaveView needs; we will show you how to do this later in the tutorial. Generally, when running WaveView, you will need to open the `.sp` file of the circuit you analyzed. WaveView will then automatically find the analysis files that it needs and allow you to plot the results.

You can create netlists in any text editor that you like. Notepad++ and Wordpad are common options under Windows. Gedit, Emacs, Vim, and Pico are common options under UNIX. In this tutorial, we are going to assume that you are familiar enough with one of these text editors to use it proficiently under your operating system of choice. If you are working in the UNIX environment and are not familiar with any text editor, we recommend using gedit for now as it is the simplest to use.

2 Windows vs. UNIX

There are two versions of HSPICE that you will encounter at UC Berkeley: the Windows version and the UNIX version. Although any netlists you create will work on both versions, the difference in use of HSPICE in these different operating systems . Both the UNIX and Windows versions are available in the lab. Only the UNIX version can be used from home (there is no remote desktop support for HSPICE at this time). This section of the tutorial will show you how to run HSPICE in all of these environments (Windows or UNIX, at home or in lab).

2.1 Windows

If you're working in the lab, you can skip the "Connecting From Home" portion of this section. Just login to a computer and jump directly into the section on "Running HSPICE".

2.1.1 Connecting From Home

In order to use HSPICE from home in Windows/Mac/Linux, you will need to use SSH; see the UNIX subsection below.

2.1.2 Running HSPICE using the EE105 lab computers in 125 Cory

We're going to go through a sample HSPICE simulation and analysis in WaveView in order to teach you how to run these programs.

1. Start Notepad. You can do this either by clicking *Start* → *Run*, typing "Notepad", and hitting Enter. Or, you can click *Start* → *Programs* → *Accessories* → *Notepad*.
2. Copy the text in Figure 1 into Notepad exactly as it is written (this is what we call a "netlist").
3. Save this file into a folder called "Tutorial". Name it `rccircuit.sp`. *Note: To ensure the filename has a .sp extension, be sure to select "All Files" in the "Save as type" listbox. You can put the filename in quotations if you want to be extra careful.*
4. Save this file into a folder called "Tutorial". Name it `rccircuit.sp`. *Note: To ensure the filename has a .sp extension, be sure to select "All Files" in the "Save as type" listbox. You can put the filename in quotations if you want to be extra careful.*
5. Click *Start* → *Programs* → *HSPICE V-2004.03-SP1* → *Hspui V-2004.03-SP1*. This is called the "HSPICE UI" (HSPUI for short), a graphical user interface to HSPICE. Although you can run HSPICE and Awaves individually, HPSPUI puts HSPICE, Awaves, your netlist, and the HSPICE output (the `.lis` file mentioned earlier) in a convenient panel for easy access.

6. Click *Open* and select `rccircuit.sp` using the file browser. You'll see that HSPUI automatically picks out the title and picks an output file called `rccircuit.lis`, also in the Tutorial directory.

7. Click *Simulate* to simulate the circuit. HSPUI will open HSPICE for you and run your netlist through HSPICE, producing `rccircuit.lis` and other output files used by Awaves.

8. Click *Edit LL* to view the output from HSPICE. This file contains information about the analysis done by HSPICE. Typically, you would look here for measurement results, operating points, and error messages. You should see no error messages at this time. If you do, check to make sure you copied the netlist exactly as shown in Figure 1. Once you've verified there are no error messages, close the output file.

9. Click *Avanwaves* to start Awaves. You'll see that HSPUI opens your circuit in Awaves and you can immediately begin plotting data. We'll cover Awaves through some examples later, but you can see the results of your analysis here by selecting *Transient: ee105 spice tutorial example 1 - simple rc circuit* in the Results Browser and double-clicking `vs` and `vo` to plot them in the graph. You may have been able to guess from the netlist, but you'll see that `vs` is a step function input and `vo` is the output taken across a capacitor when the step function is applied as the input, a curve described by $v_o(t) = v_s\left(1 - e^{-t/RC}\right)$.

10. Always remember that if you want to save your data permanently, you **must copy it to the U: drive**. Your data will be deleted as soon as you logout otherwise. To access the U: drive, you click *Start* $\rightarrow$ *Run*, type "U:" in the text box, and hit Enter. Alternatively, you can double-click *My Computer*, then double-click *U:* to achieve the same result.

11. If you're working from home, you may also want to copy your files to your home computer. This is useful for printing your netlists, output, and graphs produced in Awaves. If you double-click *My Computer*, you should see all of you local hard drives listed (assuming you following the directions in "Connecting From Home"). You can copy and paste files from the remote desktop to those drives to save them to your local machine.

2.2 UNIX

In order to use HSPICE under UNIX at UC Berkeley, you'll need to login to a UNIX server. This is done using a protocol called SSH. If you are running Linux, MacOS, or UNIX at home, you should have everything you need to jump to the section entitled "Running SSH". If you are running Windows, you will probably need to install additional software. If you know you already have an X Server and SSH software, then you can also continue to "Running SSH". If not, continue reading.

2.2.1 Software

In order to work remoted on a UNIX machine, you will need an SSH client and an X Server. There are many, many options out there, but here we will recommend a couple different pieces of software to use.

SSH Clients The most convenient software to use for ssh is PuTTY for shell functionality and WinSCP for upload and download functionality. This tutorial will assume you are using PuTTY. Here are links to this software:

- PuTTY (click the `putty.exe` link)
- WinSCP (download version 4.0.2 beta)

X Servers We recommend using the Xming X server. In addition to downloading the X server itself, you will need to install a font package. You can also install Exceed, which is a commercial X Server program free for UC Berkeley students. If you use Exceed, you will not need the font package. Here are links to this software:

- Xming X Server and Font Package (download Xming and Xming-fonts) **Recommended**
- Exceed

Installing the software is relatively straightforward. If you have problems, be sure to read the documentation associated with the software you download.

2.2.2 Running PuTTY

1. Start the X Server that you downloaded.
   - For Xming, click *Start* → *All Programs* → *Xming* → *Xming*. It will run in the background.
   - For Exceed, click *Start* → *All Programs* → *Hummingbird Connectivity 2006* → *Exceed*.
2. Start the SSH client that you downloaded. For PuTTY, double-click the file you downloaded, `putty.exe`.
3. Login to a UNIX server. In either program, if it is your first time logging in, you will see a security warning message when you try to connect to the UNIX server. Click "Yes" in either program to bypass this message.
   - For PuTTY, enter `c199.eecs.berkeley.edu` for the "Host Name". Then, on the left panel, go to Connection → SSH → X11 and enable X11 forwarding. Click back on Session in the left panel and type a name into the "Saved Sessions" text box, and click *Save*. Then, double-click the session you created in the list of sessions. It will prompt you for your username, so enter the username for your class account (i.e. `ee105-xx`, where `xx` are two letters). Your named account should also work here if you have one. Hit Enter, then type your password and hit Enter again. You should now be at a UNIX shell.

   By default, the shell provided may be `csh`. We highly recommend typing `bash` to start the Bash shell, since it is more convenient to work with.

2.2.3 Running HSPICE

```
EE105 SPICE Tutorial Example 1 - Simple RC Circuit
vs vs gnd PWL(0s 0V 5ms 0V 5.001ms 5V 10ms 5V)
r1 vs vo  1k
c1 vo gnd 1uF
.tran 0.01ms 10ms
.option post=2
.end
```

Figure 1: A simple RC circuit netlist

1. Make a new directory called `Tutorial`. You can do this by typing `mkdir Tutorial` at the Bash shell. Change to this directory by typing `cd Tutorial`.
2. Using your preferred text editor (e.g. gedit, Emacs, Vim, or Pico), copy the netlist in Figure 1 to a file called `rccircuit.sp`.
3. Run HSPICE by typing

   ```
   $ hspice rccircuit.sp > rccircuit.lis
   ```

   This command will send the output from HSPICE to a file called `rccircuit.lis`. If you exclude the second half of the command, the output will be shown in the shell itself. This can be inconvenient for referencing and debugging, so it is common to send the output to a file.

4. Open `rccircuit.lis` in your preferred text editor or pagination utility (e.g. `more` or `less`). This file contains information about the analysis done by HSPICE. Typically, you would look here for measurement results, operating points, and error messages. You should see no error messages at this time. If you do, check to make sure you copied the netlist exactly as shown in Figure 1. Once you've verified there are no error messages, close the output file.

5. Run WaveView by typing

   ```
   $ wv rccircuit.tr0 &
   ```

   This will open your transient results (.tr0) in WaveView. We'll cover WaveView in detail later, but you can see the results of your analysis here by selecting *DO:rccircuit.tr0* in the Output View browser and double-clicking `vs` and `vo` to plot them in the graph. You may have been able to guess from the netlist, but you'll see that `vs` is a step function input and `vo` is the output taken across a capacitor when the step function is applied as the input, a curve described by $v_o(t) = v_s\left(1 - e^{-t/RC}\right)$.

6. If you're working from home, you may want to copy your files to your home computer. This is useful for printing your netlists, output, and graphs produced in WaveView.

   - For SSH Secure Shell, click *Window* → *New File Transfer in Current Directory*. This will open a new interface that allows you to drag and drop files from the UNIX server to your computer and vice versa. Simply drag the file you want to download from the right side of the screen and drop it somewhere convenient on the left side of the screen.
   - For PuTTY, you will have to use WinSCP to copy your files to a local directory. Run WinSCP by clicking *Start* → *All Programs* → *WinSCP* → *WinSCP*. Create a new session and type in the host name, username, and password as you did when logging in with PuTTY. Save the session by clicking *Save...*. Click *OK* when it prompts you about saving your password, then type a name in the text box and click *OK* again. Double-click the session you just created and you will see a split interface that will allow you to drag and drop files to and from the UNIX server. Simply drag the file you wish to download from the right side of the screen to the left side in order to download it to your computer.

3 Simulating Circuits in HSPICE and WaveView

3.1 HSPICE Netlist Syntax

At this point you should be comfortable running HSPICE and WaveView under your current environment. Now we can begin talking about the mechanics of writing a netlist. The first line of a netlist must always be a comment—it will be ignored by HSPICE no matter what you put on it. Typically, we put a brief description of the netlist on the first line.

Next, we define the models that we're going to use. At this point you haven't learned about MOSFET, BJT, or diode models, but you do know enough to know that these devices have many parameters that can change their behavior, unlike resistors, capacitors, inductors, and sources, which are specified by a single value. If your circuit does not contain any transistors or diodes, then it isn't necessary to define any models.

After defining device models, we write the description of the circuit topology itself. Each type of circuit element has its own command (e.g. resistors and capacitors are defined by unique commands). The way we describe the circuit topology is by giving each node (a node being the point connecting any two or more elements) in the circuit a unique name and then connecting circuit elements between these nodes. Note that you don't have to specify these elements in any particular order (though it is often useful to specify them in an order that is logical to you in case you need to change the netlist in the future).

After describing the circuit topology, we enter analysis commands describing how we want HSPICE to analyze the circuit. We can also define options that HSPICE uses when simulating the circuit. Athe end of this document we've provided a syntax reference containing all of the circuit element and analysis commands you'll need for this course. You may want to print that part of the tutorial (or open it in a separate window) for reference before you continue to the next section.

The last line of any HSPICE netlist must be the command `.end`. This tells HSPICE where to stop reading the netlist.

Note that the organization of a netlist in the order described here is somewhat arbitrary (aside from the first and last lines). You can define a model before you use it in a document, and you can specify your analysis before you describe the circuit topology. What we've described is just one convenient way to organize your netlist so it is easy to read and understand. It's important to note that SPICE is not a programming language like you may be familiar with, so you shouldn't think of it like a programming language. An example of this is that HSPICE is completely case insensitive. That means `vs`, `VS`, `Vs`, and `vS` are all the same to HSPICE.

3.2 Examples

The specifics of writing netlists are going to be taught through examples. We'll start with simple examples and gradually progress to more complicated circuits to allow you to gradually build your proficiency with SPICE.

3.2.1 Transient analysis of a simple RC circuit

Let's start with the simple RC circuit you analyzed earlier in the tutorial. The same netlist with line numbers is shown in Figure 2.

```
1  EE105 SPICE Tutorial Example 1 - Simple RC Circuit
2  vs vs gnd PWL(0s 0V 5ms 0V 5.001ms 5V 10ms 5V)
3  r1 vs vo  1k
4  c1 vo gnd 1uF
5  .tran 0.01ms 10ms
6  .option post=2 nomod
7  .end
```

Figure 2: A simple RC circuit netlist with line numbers

First, note lines 1 and 7. Line 1 is a comment, as always. You can specify a comment at any time using the asterisk (`*`). Any text following the asterisk will be a comment. Line 7 is the `.end` command, another requirement for any netlist.

Lines 2–4 contain the description of the circuit topology. In this case, we have three nodes: `vs`, `vo`, and `gnd`. Note that elements and nodes can have the same name, so Line 2 is perfectly legal even though it is connecting the voltage source called `vs` to the circuit node called `vs`. In some circuits, you may need to refer to a voltage source or circuit node later in an analysis statement, but context will always make it clear which you are referring to so a source and node having the same name will never cause ambiguities.

In these lines you can also see the usage of suffixes to denote powers of 10. From 10^{-15} to 10^{12}, these are: `f` (femto), `p` (pico) , `n` (nano), `u` (micro), `m` (milli), `k` (kilo), `meg` (mega), `g` (giga), `t` (tera). HSPICE often reports values following by an `x`—this is short-hand for 10^6, and you can use it interchangeably with `meg`.

The node `gnd` is a standard name used to reference ground. You can also use the number `0` to refer to the ground node (some people like to number their nodes instead of naming them, in which case `0` may be a more natural fit). Ground is a special node in the circuit, so only use `gnd` or `0` to refer to the ground node. The other nodes have arbitrary names, but in this case I've named them intuitively: `vs` is the voltage of the source, and `vo` is the voltage across the capacitor, which I consider the output in this case. Figure 3 shows the circuit diagram corresponding to this netlist (with nodes labeled as they are in the netlist).

If you take another look at Line 3, you'll see that when defining the source `vs` I put the node `vs` first and the node `gnd` second. This is an important distinction when dealing with elements that have positive and negative terminals. For a resistor, such as the one in Line 3, I can list the nodes in any order without changing the circuit's behavior. Also note that the value of the source is a complicated `PWL` expression. Refer to the syntax reference at the end of this guide and you'll see that this is actually a non-ideal step function (with a linear rise over 1 µs).

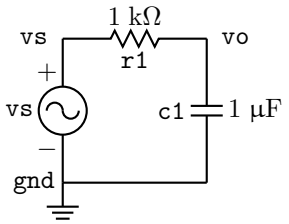

Figure 3: Circuit diagram corresponding to the netlist in Figure 2

Line 5 is our first analysis statement. It tells HSPICE to do a transient (i.e. measure values as a function of time) from time $t = 0$ ms to time $t = 10$ ms in increments of 0.01 ms. When you simulated this circuit earlier, you should've seen that the plot of `vo` is that of a capacitor charging due to a step function input. This analysis statement is what allowed you to plot that time-based analysis of node `vo`.

Line 6 is a list of options that you can pass to HSPICE. In this case, I've used two options: `post=2` and `nomod`. The first, `post=2`, tells HSPICE to generate extra files for WaveView to use in plotting data. If you do not specify `post=2` (actually, you can also specify `post=1` or `post=3`—the differences aren't important for our purposes), you will not be able to plot your data in WaveView. The second option, `nomod`, just tells HSPICE not to print model information in the output file. Although this is unimportant for this example (since we haven't included any models), it can be useful when you are dealing with real device models that can be half a page long.

Try simulating this circuit and plotting `vo` in WaveView like you did earlier in the tutorial. Now try to measure how long it takes for `vo` to go from 0V to $0.63(5) = 3.15$ V once `vs` steps from 0 V to 5 V. If you remember your RC circuits from EE40, you'll realize this should be the time constant of the circuit $t = RC = 1$ ms. Measuring this on the graph in WaveView, I get almost exactly 1 ms.

3.2.2 Operating point analysis for a diode

Another very useful type of analysis is operating point analysis. You probably recall having to calculate operating points in EE40 by finding the intersection between a diode's IV curve and the load line created by a resistor. Figure 4 shows a simple circuit diagram for which we'd like to calculate the operating point. Figure 5 shows the corresponding netlist (once again with line numbers). I've labeled the nodes and elements in the diagram consistently with how they're named in the netlist.

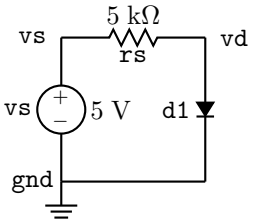

Figure 4: A diode circuit for operating point analysis

```
1  EE105 SPICE Tutorial Example 2 - Simple Diode Circuit
2  .model tut_diode d (is=1e-14 vj=0.6 rs=10)
3  vs vs gnd 5V
4  rs vs vd  5k
5  d1 vd gnd tut_diode
6  .op
7  .end
```

Figure 5: The netlist corresponding to the diode circuit in Figure 4

Let's take a look at the new features of this netlist. First, we've defined a model on Line 2 called `tut_diode`. Models are necessary whenever you are using diodes or transistors in your netlists. The model defines certain parameters related to the device you specify. In this class we'll typically give you the parameters you need to define for your SPICE models, and our models will typically be very simple. Real device models can have dozens of parameters specified in excruciating detail. Note that a model name cannot start with a number in HSPICE (for example, `2N4401` would be an invalid name, but `QN4401` would be a valid model name).

Line 5 shows how you can define a diode. You should recall from EE40 that a diode is asymmetric, so we have to specify the positive terminal first and the negative terminal second. You also have to specify which model to use, and in this case I'm using the model we just defined in Line 2.

The last new command shown here is on Line 6, the `.op` command. This command stands for "operating point" and tells HSPICE to calculate the voltage at every node and the current through every branch in the circuit. Note that we've excluded the `.option post=2` command in this netlist because we will not need to use WaveView for this analysis.

Simulate this circuit and open the SPICE output file. Scroll through this file until you see a line that says "operating point information". Below that, you should see a list of all of the nodes in the circuit (in this case, we just have two nodes) and the voltages at those nodes, plus a list of all the elements in the circuit and the voltages and currents associated with those elements. We can read off the operating point we're interested in: the voltage at `vd` is 655.8104 mV.

Let's compare this to the hand-calculated operating point for the same circuit using $I_S = 10^{-14}$ A for the diode. We have:

$$I_S e^{V_D/V_T} = \frac{V_S}{R_S} - \frac{1}{R_S} V_D$$

Solving this equation gives $V_D = 654.889$ mV, a percent error of 0.14 % compared to the value given by HSPICE.

3.2.3 DC analysis of a diode

Another type of analysis we can do is DC analysis. DC analysis allows us to measure values in our circuit as we sweep a voltage. For example, let's look again at our diode circuit. Instead of looking at `vd` for `vs` at just 5 V, maybe we want to sweep `vs` from 0 V to 5 V and see how `vd` changes as a result. Let's look at Figure 6 to see how we might do this (I've used the same node names as shown in Figure 4).

```
1  EE105 SPICE Tutorial Example 3 - Simple Diode Circuit
2  .model tut_diode d (is=1e-14 vj=0.6 rs=10)
3  vs vs gnd 5V
4  rs vs vd  5k
5  d1 vd gnd tut_diode
6  .dc vs 0V 5V 0.01V
7  .option post=2
8  .end
```

Figure 6: Netlist for DC analysis of the diode circuit in Figure 4

Let's see how this netlist has changed to allow us to do a DC analysis. First, note the Line 6, instead of being an operating point analysis `.op`, is now a DC analysis `.dc`. In this case, we are sweeping `vs` from 0 V to 5 V in steps of 0.01 V. Looking at Line 3, you may think this conflicts with the value I've already put there; however, when you do a DC analysis of a source, it ignores whatever value you gave the source earlier.

The second change I made was in adding Line 7, since we are going to be using WaveView to look at a plot of `vd` versus `vs`. Simulate the circuit and open it in WaveView. Before we continue, I'm going to briefly describe the WaveView interface that we've been using so far.

When you first open WaveView with your circuit, it will automatically open the data from input file that you give it. The various traces will be displayed in the Output Browser. This will be the first place to look

when you want to plot something you've measured. There are many extraneous quantities you'll probably never want to plot (for example, you can plot the voltage at the ground node), but usually the quantity you'll want to plot will be located somewhere in the Output Browser.

3.2.4 AC analysis of a high-pass filter

Let's take a look at a basic RC high-pass filter, which you learned about in EE40. Figure 7 shows the circuit diagram we'll be simulating, and Figure 8 shows the associated netlist.

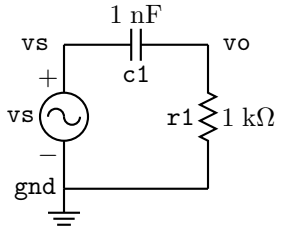

Figure 7: A simple RC high-pass filter

```
1  EE105 SPICE Tutorial Example 4 - Simple RC High-Pass Filter
2  vs vs gnd ac 1V
3  c1 vs vo  1nF
4  r1 vo gnd 1k
5  .ac dec 500 100 1G
6  .option post=2
7  .end
```

Figure 8: The netlist corresponding to the high-pass filter in Figure 7

First let's look at how the netlist has changed once again. Look at Line 2. Although this looks extremely similar to all of our other voltage definitions, it has one extra parameter specified before the value: `ac`. This tells HSPICE that the AC analysis should be done by varying the frequency of this source. If you try to do an AC analysis of a circuit without any AC sources specified, HSPICE will give you an error.

Now look at the new analysis statement on Line 5. The `.ac` command tells HSPICE to do an AC analysis by sweeping the frequency of the AC source as you specify. The first parameter passed to this command, in our case `dec`, tells HSPICE whether to do a linear (`lin`) sweep, a sweep by octaves (`oct`), or a sweep by decades (`dec`). In this class, we'll be using `dec` for all of our AC analyses. The second parameter tells HSPICE how many sample points to take per decade. The third parameter is the starting frequency of the sweep, and the fourth parameter is the ending frequency of the sweep. In this case, we're taking 500 points per decade from $f = 100$ Hz to $f = 1$ GHz.

Simulate this circuit and open the *.ac0 circuit it produces in WaveView. You'll notice your *.ac0 file in the Output View browser. Expand it and click on toplevel and then find the vo signal in the panel below.

Let's start with the phase Bode plot. In *Tools* → *Equation Builder* you can find the tools you need to build a Bode Plot. To plot the phase, right click on the `vo` signal in the Output View Browser. Then select `phase(v)` from the `RF Built-in Functions` box.

Follow a similar process to plot the magnitude in decibels (look under the math functions; for voltage ratios always use db20() and for power ratios use db10()).

As a side note, normally when you make a Bode plot you'd need to take the magnitude of the output over the input in dB. In general this is true, but in our case we set the magnitude of our AC source to 1 V, so dividing by a magnitude of 1 V would not have changed the results. If you were to use a different magnitude, you'd have to use the same method described above to divide the magnitude of the output by the magnitude of the input before computing the expression's value in dB.

3.2.5 Transfer function analysis

Let's try to find the voltage gain (output over input) for the circuit in Figure 9. Since this a simple resistive network, you can hand-calculate the answer and compare it to the result you get in HSPICE. I've used more convential notation in the diagram this time and haven't labeled the nodes. At this point, you should be able to recognize what is going on without those labels. Figure 10 shows the netlist for this circuit diagram.

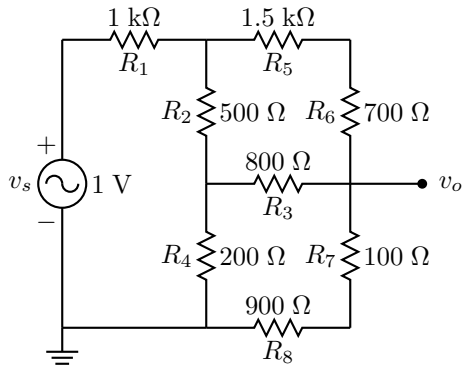

Figure 9: A complicated resistor network

```
 1  EE105 SPICE Tutorial Example 5 - Complicated Resistor Network
 2  vs vs gnd 1V
 3  r1 vs 1   1k
 4  r2 1  2   500
 5  r3 2  vo  800
 6  r4 2  gnd 200
 7  r5 1  4   1.5k
 8  r6 4  vo  700
 9  r7 vo 5   100
10  r8 5  gnd 900
11  .tf v(vo) vs
12  .end
```

Figure 10: SPICE netlist for a resistor network

The only new feature here is the analysis statement on Line 11, which is a `.tf` statement, short for transfer function. This calculates the DC gain, input resistance, and output resistance between the voltage node specified first (the output) and the voltage source specified second (the input). We don't need the `.option post=2` statement since we will not be using WaveView for this analysis.

Simulate the circuit in HSPICE and open the output file. Look for a line that says "small-signal transfer characteristics". Underneath that, you should see values for `v(vo)/vs`, or the gain from `vs` to `vo`, the input resistance at `vs`, and the output resistance at `vo`. In this case, the gain is 109.8628 m $\approx$ 0.110. If you were to calculate the voltage at node v_o by hand, you would get the same result.

3.2.6 Pole-zero analysis

HSPICE can also find poles and zeroes for you. Let's take another look at the circuit in Figure 7. This time, we'll use the netlist in Figure 11.

As you can see, the `.pz` statement is used much like the `.tf` statement in that you first specify an output node, then an input source. After you simulate the circuit in HSPICE, open the output file and look for a line that says "pole/zero analysis". You'll notice that just below that line is a list of all of the poles and zeroes from the input source to the output node. In this case, we have just one zero, since this is a high-pass filter. You can confirm the result by hand-calculating the transfer function.

```
1  EE105 SPICE Tutorial Example 6 - Pole-zero analysis of a high-pass filter
2  vs vs gnd ac 1V
3  c1 vs vo  1nF
4  r1 vo gnd 1k
5  .pz v(vo) vs
6  .end
```

Figure 11: A netlist for finding the poles and zeroes in the circuit in Figure 7

3.2.7 The `.measure` command

The `.measure` command is used to measure almost any parameter in your circuit under some specified conditions. For example, it can be used to measure rise time, fall time, delay, maxima and minima, etc. Here we will cover just one use of the `.measure` command.

TRIG/TARG One use of the `.measure` command involves specifying a trigger and a target (TRIG and TARG, respectively). A trigger tells the command when to start measuring, and the target tells the command when to stop measuring. The command will then report the value you want to measure at the trigger condition, target condition, and the difference between its values under those conditions. This is best illustrated through an example. Consider the RC circuit in Figure 3. Let's say we want to measure the rise time of this RC circuit. First, we need to define the rise time metric—let's say it's the time it takes for v_o to go from 10 % to 90 % of its final value (i.e. from 0.5 V to 4.5 V).

Now, let's take a look at the netlist that will perform this measurement, shown in Figure 12. Pay close attention to line 6, which contains the `.measure` command. The structure of the command is as follows (note that the plus signs allow you to continue a command on the next line and are not necessary):

```
.measure <ac|dc|tran> <name>
+ trig <node> val=<value> <rise|fall|cross>=<value>
+ targ <node> val=<value> <rise|fall|cross>=<value>
```

The first argument specifies what analysis statement to associate the measurement with. For example, if measuring time the first argument should be `tran`. If measuring frequency, it should be `ac`, and if voltage, `dc`. The second argument is simply a name for your measurement. In the example for computing rise time, I use the name `trise` for my measurement. Now you must specify the trigger and target, which have identical structure. For each, you must specify what node you want to trigger/target on, the value it needs to equal at the time of the trigger/target, and how many rises/falls/crosses have occurred prior to the node reaching that value. You should understand the concept of rising and falling, i.e. a signal going from low to high or high to low. A cross is simply the sum of rises and falls.

Let's look at our example again. I am triggering when `v(vo)`, the voltage at the output node, is equal to 0.5 V, or 10 % of its final value. I want it to measure this value on the first rise, so I specify `rise=1`. I am targeting when `v(vo)` is equal to 4.5 V, or 90 % of its final value. Again, I want the value on the first rise, so I again specify that `rise=1`. That's all there is to it. If your simulate this netlist in HSPICE and look at the output, under a line labeled "transient analysis" you should see values for `trise`, `trig`, and `targ`, where `trise` is just `targ - trig`.

One issue to watch out for is that you cannot set your trigger to at zero. If you want to trigger at zero, you can approximate the measurement by using a very small, but non-zero, trigger value. For example, I could trigger on the value `1n`.

For more information on the `.measure` command, refer to Specifying User-Defined Analysis (.MEASURE).

3.3 Including another file

During the course of the semester, you'll often have to use models that we provide on the course website. Instead of copying and pasting the models, you can just include the model file in your circuit. To do this, place the model file in the same directory as your netlist (for example, `circuit.sp` and `2N4401.mod`). Then, in your netlist, include the following line:

```
1  EE105 SPICE Tutorial Example 7 - Computing rise time
2  vs vs gnd PWL(0s 0V 5ms 0V 5.001ms 5V 10ms 5V)
3  r1 vs vo  1k
4  c1 vo gnd 1uF
5  .tran 0.01ms 10ms
6  .measure tran trise trig v(vo) val=0.5 rise=1 targ v(vo) val=4.5 rise=1
7  .end
```

Figure 12: A netlist to compute the rise time of an RC circuit

```
.inc '<filename>'
```

In this case, we would use `2N4401.mod` for the filename.

4 Syntax Reference

Any bracketed labels must be replaced entirely (i.e. if you want a value of 5 V, you should replace `<value>` with `5V`).

- Independent voltage source
  `v<name> <+ terminal> <- terminal> <value>`
- Independent current source
  `i<name> <+ terminal> <- terminal> <value>`
- Voltage-controlled voltage source
  `E<name> <+ terminal> <- terminal> <+ control> <- control> <gain>`
- Current-controlled voltage source (`vcontrol` refers to the voltage source which the controlling current flows through)
  `H<name> <+ terminal> <- terminal> <vcontrol> <gain>`
- Voltage-controlled current source
  `G<name> <+ terminal> <- terminal> <+ control> <- control> <gain>`
- Current-controlled current source (`vcontrol` refers to the voltage source which the controlling current flows through)
  `F<name> <+ terminal> <- terminal> <vcontrol> <gain>`
- Sinusoidal source (used as a `<value>`)
  `sin(<offset> <amplitude> <frequency> <delay> <damping> <phase>)`
- Square wave source (used as a `<value>`)
  `pulse(<vmin> <vmax> <delay> <rise time> <fall time> <pulse width> <period>)`
- Piece-wise linear source (used as a `<value>`)
  `pwl(<t0> <v0> <t1> <v1> <t2> <v2> ...)`
- Resistor
  `r<name> <terminal 1> <terminal 2> <value>`
- Capacitor
  `c<name> <terminal 1> <terminal 2> <value>`
- Inductor
  `l<name> <terminal 1> <terminal 2> <value>`

- Model (type can be `nmos`, `pmos`, `NPN`, `PNP`, or `D` for diode)
  `.model <name> <type> (<parameter list>)`
- MOSFET (you can specify additional parameters, such as `W=<value> L=<value>`, in the parameter list)
  `m<name> <drain> <gate> <source> <body> <model> <parameter list>`
- BJT
  `q<name> <collector> <base> <emitter> <model> <parameter list>`
- Diode
  `d<name> <+ terminal> <- terminal> <model> <parameter list>`
- AC analysis (pick either `lin`, `dec`, or `oct`)
  `.ac <lin|dec|oct|> <number of samples> <freq start> <freq stop>`
- DC analysis
  `.dc <source> <start> <stop> <step>`
- Nested DC analysis (`source1` is swept, `source2` is stepped)
  `.dc <source1> <start1> <stop1> <step1> <source2> <start2> <stop2> <step2>`
- Transient analysis
  `.tran <t step> <t stop>`
- Transfer Function (TF) analysis
  `.tf v(<node>) <source>`
- Poles and Zeros (PZ) analysis
  `.pz v(<node>) <source>`