# SPICE 1: Basics of Circuit Simulation

ECE 3410, Utah State University

## What is SPICE?

- **S**imulation **P**ackage with **I**ntegrated **C**ircuit **E**mphasis is a numerical simulation tool originally developed by the University of California, Berkeley in 1973, recognized as an IEEE Milestone in the development of electronics.
- Today, circuit designers rely heavily on simulators based on SPICE (or very similar to it). Familiarity with SPICE is essential for electronic engineers.
- Circuits are described in plain-text documents using the **SPICE netlist** language, which is a type of **hardware description language**. In this class we will learn the structure and syntax of the netlist files.
- Many graphical schematic entry tools exist to translate a circuit diagram into a netlist file. In this class we will work directly with the netlist files.

## SPICE "flavors"

Berkeley SPICE is a free open-source product. Over the years it was forked into numerous versions, called "flavors", that mostly work the same way as the original SPICE simulator. Most SPICE flavors use the original SPICE source code as a foundation.

Some popular commercial SPICE flavors:

- HSPICE – Linux, owned by Synposys, regarded as a gold standard.
- PSPICE – Windows, owned by Cadence.
- LTspice – Windows/Mac, owned by Analog Devices, specialized for simulating AD products.

Some popular open-source SPICE flavors:

- **ngspice** – Linux/Mac/Windows, actively developed, recognized as the unofficial successor to Berkeley SPICE. **We will use this one**.
- WinSpice – Windows port of Berkeley SPICE.

## Obtaining the Assignments

We assume you will use a Linux computer with both `git` and `ngspice` installed.
Launch a **terminal** window and clone your spice assignment repository:

```
git clone https://[username:password]@left.engr.usu.edu/git/username/spice_assignments
```

This command should download assignment files to your computer in a folder called `spice_assignments`. Now navigate to the first assignment:

```
cd spice_assignments/spice1
```

and do a directory listing:

```
ls
```

## SPICE project structure

In this class we will separate our files into three categories:

- **netlists** describe circuit components and their connections.
- **testbenches** contain commands for simulations, measurements, plots, etc.
- **models** list the physical constants and parameters that govern the physical behavior of electronic devices like diodes and transistors.

## File and Directory Organization

Each lab assignment has a specific file structure designed to keep things organized:

```
spice/
  +- assignment_name/
      +- netlists/           for netlist files
      +- tests/              for testbench files
      +- models/             for model files
      +- plots/              for saving plot outputs
      +- data/               for saving output data
      +- figures/            for images, schematics, etc
      +- report/             for your lab report
      - .spiceinit           ngspice settings (hidden file)
      - assignment_name.html instructions
```

## SPICE language files

Every SPICE file shall have:

- A `.sp` file extension.
- A title comment in the first line, beginning with an asterisk `*`

## SPICE netlist syntax

A netlist contains one or more **component** instances (**V**oltage sources, **R**esistors, etc), their node labels, and their parameters.

```
* Circuit 1

Vin nin gnd DC 5V

R1 nin  nout 100k
R2 nout gnd  10k
```

```
* Circuit 1

Vin nin  gnd  DC 5V

R1  nin  nout 100k
R2  nout gnd  10k
```

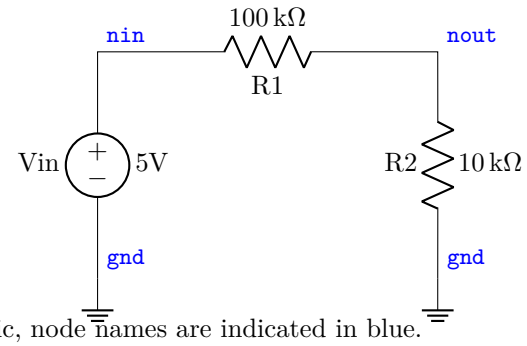



Figure 1: Schematic, node names are indicated in blue.

## Components

A **component instance** is a line beginning with an instance name. The instance name begins with the type letter, for example a voltage source must start with the letter `V`. For other devices, instance names begin with familiar letters:

| Letter | Component Type |
|---|---|
| V | Independent Voltage Source |
| I | Independent Current Source |
| R | Resistor |
| C | Capacitor |
| L | Inductor |
| D | Diode |
| M | MOSFET |
| Q | BJT |

Every instance must begin with a **unique** name, followed by a **node list**, and the line finishes with parameters for the component.

## Node Labels

When a component is instantiated, node labels are implicitly declared in the component's node list. Any nodes with the same label are connected together. **SPICE is NOT case sensitive**, so node labels are not distinguished by upper or lower case.

A netlist must assign a unique label to each node in the circuit. A node label can be any non-reserved alphanumeric string. My preferred convention is to start every node label with the letter `n`, to keep node labels distinct from component names and other identifiers.

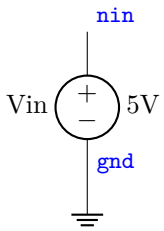
**GROUND node:** every netlist must indicate a ground node, labeled as the number `0` or the name `gnd`.

## Component: DC Voltage Source

For example, the voltage source named `Vin` is **instantiated** by the line

```
Vin nin gnd DC 5V
```

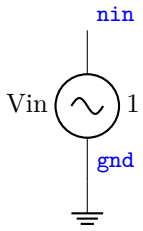
The component name starts with `V` because it is an independent voltage source, its node list is `nin` (positive terminal) and `0` (negative terminal), and its parameters are `DC 5V`.




## Component: AC Voltage Source

A voltage source can also be prepared for AC signal analysis (i.e. Bode plots). In most instances we set the AC magnitude to 1.

```
Vin nin gnd AC 1
```



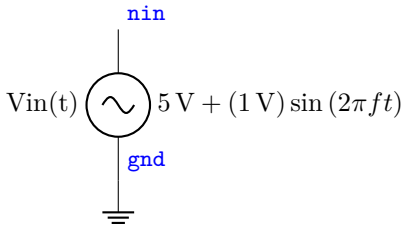

## Component: Sinusoidal Voltage Source

A voltage source can also be prepared for **transient simulation** by specifying

```
Vname npos nneg SIN(offset amplitude frequency)
```

Example:

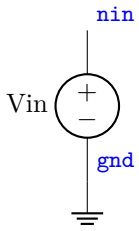
```
Vin nin gnd SIN(5V 1V 1kHz)
```




## Exercise 1: AC+DC+SIN Voltage Source

We can combine DC. AC and sinusoidal parameters in the same declaration, to support multiple types of simulation. In this example, a single long line is "continued" by using the `+` symbol on successive lines. **Edit `circuit1.sp` to match the lines shown below.**

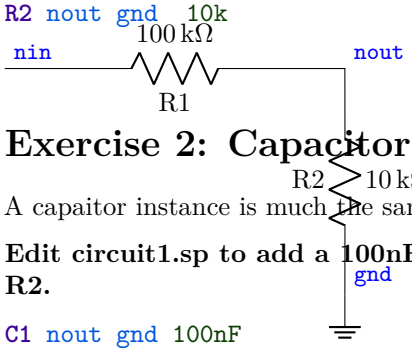
```
Vin nin gnd DC 5V
+ AC 1
+ SIN(5V 1V 1kHz)
```




## Component: Resistor

The next two lines place the resistors.

```
R1 nin  nout 100k
R2 nout gnd  10k
```

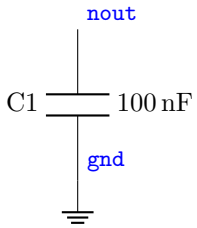



## Exercise 2: Capacitor

A capaitor instance is much the same as a resistor.

**Edit circuit1.sp to add a 100nF capacitor C1 in parallel with resistor R2.**

```
C1 nout gnd 100nF
```




## About Units

SPICE allows SI units with engineering prefixes, e.g. `kHz` for "kilo Hertz", `ms` for "milli seconds", and so on. Since the SPICE language is not case sensitive, **there is no difference between M and m**, so both of those mean "milli". To express "Mega", you have to type out `MEG`.

| prefix | meaning | scale | prefix | meaning | scale |
|---|---|---|---|---|---|
| f | femto | $10^{-15}$ | | | |
| p | pico | $10^{-12}$ | T | tera | $10^{12}$ |
| n | nano | $10^{-9}$ | G | giga | $10^{9}$ |

| prefix | meaning | scale | prefix | meaning | scale |
|---|---|---|---|---|---|
| u | micro | $10^{-6}$ | Meg | mega | $10^{6}$ |
| m | milli | $10^{-3}$ | k | kilo | $10^{3}$ |

After the scale letter, you can specify physical units (do not include any space between the scale and the unit). You can write `MegOhm` for "mega Ohms", `uF` for "micro Farads" and so on. Physical units are not mandatory, but they can help readability.

## Exercise 3: Running `ngspice`

To simulate `circuit1.sp`, run the `ngspice` application from within the terminal. **Important:** always run `ngspice` from the assignment directory. Since this is assignment "spice1", you should position your terminal in the `spice1/` directory before launching `ngspice`. This is because `ngspice` application settings are defined in a hidden file within the `spice1/` directory, and the application won't find those settings if you launch it elsewhere.

This will launch the `ngspice` interactive interpreter:

```
ngspice netlists/circuit1.sp | tee -a log.txt
```

**Make sure to include `| tee -a log.txt` so that your session will be recorded in a text file for grading.** The `|` or "pipe" symbol is in the upper right part of the keyboard, obtained by typing shift-backslash, i.e. SHIFT+\.

In the next few slides we will describe some of the common simulation commands. Please run each of these commands from within the `ngspice` interpreter.

## Command: `listing`

At the `ngspice` prompt, you can run the `listing` command to see the current circuit being simulated.

```
ngspice 12 -> listing
      * circuit 1

  2 : .global gnd
  3 : vin nin 0 dc 5v ac 1 sin(1 1 1k)
  4 : r1 nin nout 100k
  5 : r2 nout 0 10k
  6 : c1 nout 0 100nf
  7 : .end
```

You may notice that `ngspice` adds or modifies some lines when loading them into memory, like:

- `.global gnd` and `.end` lines are inserted
- The `gnd` node is relabeled as `0` within the netlist.
- All text is **converted to lower case**. Although the SPICE language is not case-sensitive, the `ngspice` interpeter is, so all component and node references must be made lower-case.

## Simulation Command: `op`

To run an **operating point simulation**, use the `op` command. This computes the circuit's DC solution.

```
ngspice 6 -> op
Doing analysis at TEMP = 27.000000 and TNOM = 27.000000


No. of Data Rows : 1
```

## Command: `display`

To see the circuit's nodes and branches, use the `display` command:

```
ngspice 7 -> display
Here are the vectors currently active:

Title: * circuit 1
Name: op2 (Operating Point)
Date: Mon Dec 19 10:49:29  2022

    nin                 : voltage, real, 1 long [default scale]
    nout                : voltage, real, 1 long
    vin#branch          : current, real, 1 long
```

Usually branch currents are saved only for voltage sources, in this case the only voltage source is `vin`.

## Command: `print all`

To see the actual values of the node voltages and branch currents, you can use the `print` command:

```
ngspice 14 -> print all
nin = 5.000000e+00
nout = 4.545455e-01
vin#branch = -4.54545e-05
```

Note that branch currents are usually reported with negative values, since they flow "up" through the voltage source.

## Command: `print v(node)`

To see the values for one or more specific nodes or branches, you can specify them when calling the `print` command.

```
ngspice 15 -> print v(nin) v(nout)
v(nin) = 5.000000e+00
v(nout) = 4.545455e-01
```

## Command: `print i(vsource)`

You can print the branch current through a voltage source:

```
ngspice 26 -> print -i(vin)
-i(vin) = 4.545455e-05
```

Mathematical expressions are also permitted, in this case a minus sign is applied so that the current is reported as positive.

## Simulation Command: `dc`

One of the most powerful capabilities of `SPICE` is the **DC Sweep**. In essence, this command repeats the `op` simulation while adjusting the value of some component parameter. The syntax is:

```
dc name start stop step
```

For example, we can step `Vin` from 0V to 5V in steps of 1V:

```
ngspice 28 -> dc vin 0 5 1
Doing analysis at TEMP = 27.000000 and TNOM = 27.000000


No. of Data Rows : 6
```

## Print results for DC Sweep

After a DC sweep, the `print` command produces a table of results:

```
ngspice 19 -> print v(nin) v(nout)

* circuit 1

DC transfer characteristic  Wed Dec 21 22:24:48  2022
```

```
--------------------------------------------------------------------------------
Index   v-sweep         v(nin)          v(nout)
--------------------------------------------------------------------------------
0       0.000000e+00    0.000000e+00    0.000000e+00
1       5.000000e-01    5.000000e-01    4.545455e-02
2       1.000000e+00    1.000000e+00    9.090909e-02
3       1.500000e+00    1.500000e+00    1.363636e-01
4       2.000000e+00    2.000000e+00    1.818182e-01
5       2.500000e+00    2.500000e+00    2.272727e-01
6       3.000000e+00    3.000000e+00    2.727273e-01
7       3.500000e+00    3.500000e+00    3.181818e-01
8       4.000000e+00    4.000000e+00    3.636364e-01
9       4.500000e+00    4.500000e+00    4.090909e-01
10      5.000000e+00    5.000000e+00    4.545455e-01
```

## Simulation: DC Sweep of Resistor Value

The `dc` command can be used to sweep any component parameter. Here we try varying a resistor:

```
ngspice 32 -> dc r1 1k 10k 1k
Doing analysis at TEMP = 27.000000 and TNOM = 27.000000


No. of Data Rows : 10
```

## Print results for resistor sweep

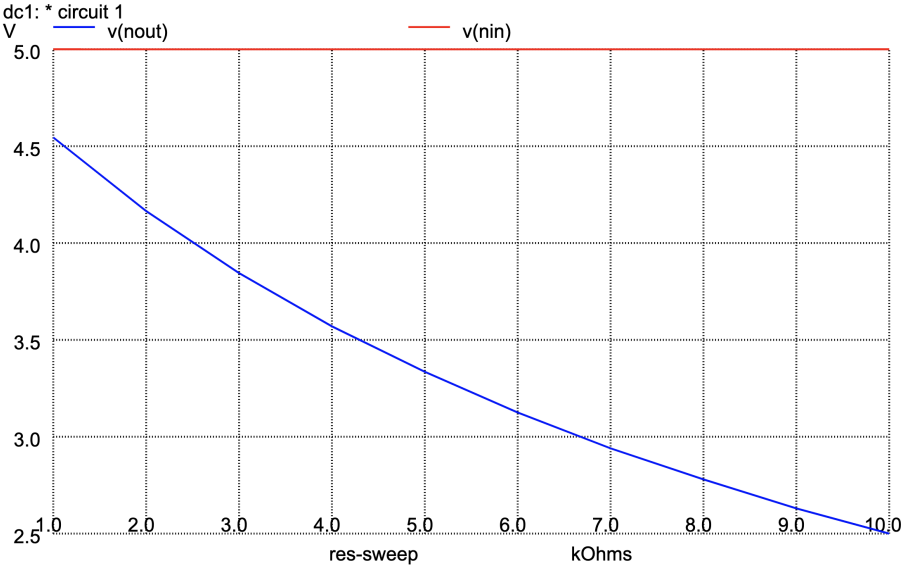
```
ngspice 34 -> print v(nin) v(nout)
                                  * circuit 1
                                  DC transfer characteristic  Mon Dec 19 11:09:53  2022
--------------------------------------------------------------------------------
Index   v-sweep         v(nin)          v(nout)
--------------------------------------------------------------------------------
0       0.000000e+00    0.000000e+00    0.000000e+00
1       5.000000e-01    5.000000e-01    4.545455e-02
2       1.000000e+00    1.000000e+00    9.090909e-02
3       1.500000e+00    1.500000e+00    1.363636e-01
4       2.000000e+00    2.000000e+00    1.818182e-01
5       2.500000e+00    2.500000e+00    2.272727e-01
6       3.000000e+00    3.000000e+00    2.727273e-01
7       3.500000e+00    3.500000e+00    3.181818e-01
8       4.000000e+00    4.000000e+00    3.636364e-01
9       4.500000e+00    4.500000e+00    4.090909e-01
10      5.000000e+00    5.000000e+00    4.545455e-01
```

## Command: `plot` (for desktop interfaces)

SPICE is also capable of graphical plotting. In place of the `print` command, you can say `plot`.

```
ngspice 35 -> plot v(nin) v(nout)
```

You will see a plot window appear:

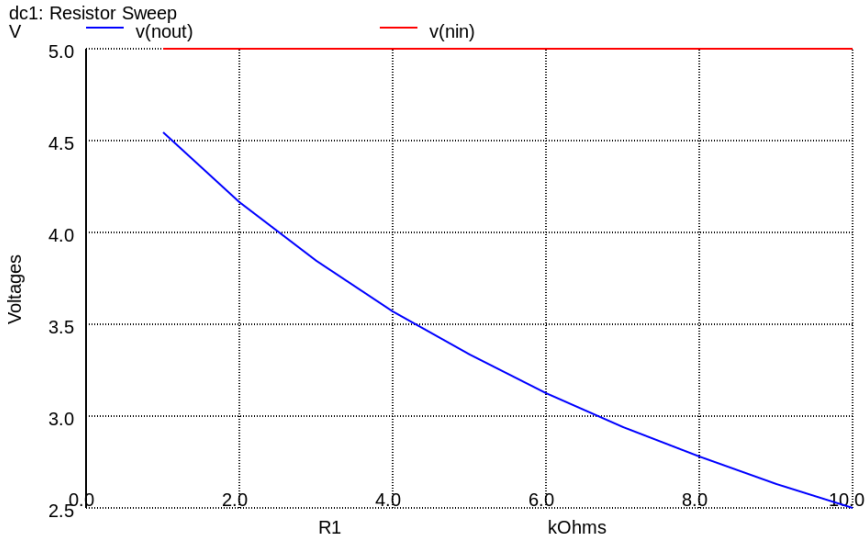



**NOTE:** the `plot` command will not work if you are accessing `ngspice` on a remote server. For that you can use the `hardcopy` command, discussed later.

## `plot` title and axis labels

You can add more plot settings after the `plot` command, for example:

```
plot v(nin) v(nout) title 'Resistor Sweep' xlabel 'R1' ylabel 'Voltages'
```




## Command: `hardcopy` (for terminal interfaces and reports)

The `hardcopy` command saves a plot in a graphics file. The command syntax is mostly the same as `plot`, except you must provide a filename, like this:

```
ngspice 2 -> hardcopy plots/circuit1_dc.svg v(nin) v(nout)

The file "plots/circuit1_dc.svg" has the Scalable Vector Graphics format.
```

Our default graphics format is SVG. As a rule, save plots in the `plots/` subdirectory and give them descriptive names.

## Using `hardcopy` for remote access

- If you use a remote terminal connection to run SPICE simulations, you can make plots using `hardcopy` and view them in a web browser.
- Our private servers are configured to allow user web pages in the `~/public_html` directory.
- Use the `ln` command to make a link to your simulation directory, something like this:
  `ln -s ~/spice_assignments/spice1/ ~/public_html/spice1`
- Then direct your web browser to the appropriate server, for example `http://genesys.bluezone.usu.edu/~username/spice1`

- You will need to be using the university network or VPN to access the servers.

## Simulation Command: `ac`

The `ac` command is used to simulate frequency response and make Bode plots.

```
ac lin|dec|oct pointsPerInterval startFrequency endFrequency
```

The first argument specifies the type of frequency stepping:

- `lin` : linear frequency scale
- `oct` : logarithmic, by octaves (an octave is a doubling of frequency)
- `dec` : logarithmic, by decades (a decade is 10x increase in frequency)

The most typical choice is `dec`, with 10 points per decade. For our circuit, let's simulate from 1Hz up to 10kHz:

```
ac dec 10 1Hz 10kHz
```

## Frequency Response: `vdb` and `phase`

After running an AC simulation, you can report the magnitude and phase response:

```
print vdb(nout) phase(nout)
```

This command prints numerical values, with `vout` in dB and the phase in radians. To see the phase in degrees, you need to scale it:

```
print  vdb(nout)  (180/PI)*phase(nout)
```

The `vdb` and `phase` functions also work with the `plot` and `hardcopy` commands:

```
plot vdb(nout)
plot (180/PI)*phase(nout)
```

## Simulation Command: `tran`

To run a transient (time domain) simulation, use the `tran` command:

```
tran [stop time] [time step]
```

Since our input signal is periodic with a 1ms period, it makes sense to simulate for a few periods, say 10ms, with a time step equal to some fraction of the period, say 100us.

```
tran 10ms 100us
plot v(nin) v(nout)
```

## Command: `quit`

Close the `ngspice` application using the `quit` command, then open the `circuit1.sp` netlist in a text editor of your choice.

## Exercise 4: Create an AC Testbench

Now create a file called `tests/circuit1_ac.sp` and enter these lines:

```
* AC Testbench for circuit1

.include netlists/circuit1.sp

.control
  ac dec 10 1 10Meg
  hardcopy plots/circuit1_ac_magnitude.svg vdb(nout)
  hardcopy plots/circuit1_ac_phase.svg (180/PI)*phase(nout)
  plot vdb(nout)
  plot (180/PI)*phase(nout)
.endc
```

This testbench shows a few new things: * The `.input` command is used to load lines from another file. In this case the file is the netlist, `circuit1.sp`. * The `.control` and `.endc` keywords indicate a script to automate the interactive commands. * The `ac` command is used to run an AC simulation. * The `vdb()` function is used to compute the Bode magnitude response. * The `phase()` function is used to compute the Bode phase response.

## Run the AC Testbench

To run the testbench, load it as the netlist in NGspice:

```
ngspice tests/circuit1_ac.sp | tee -a log.txt
```

View the results and VERIFY that the signal plots look like they should.

## Exercise 5: AC Measurements

The SPICE `measure` command supports a variety of automated measurements. After an AC simulation, you can measure things like the **peak magnitude** and the **cutoff frequency**.

Add the lines below after the `ac` command in your AC testbench, then run the simulation. **Verify**, using hand calculations, that the results are correct for this circuit.

```
meas ac vomax MAX vdb(nout)
let vdrop=vomax-3
meas ac f3db WHEN vdb(nout)=vdrop CROSS=1
```

The following slides dissect the meaning of these lines.

## Variables in NGSpice

All SPICE simulators support **variables**. In NGSpice, a variable is declared and defined by using the `let` command:

```
let <name>=<expression>
```

Once a variable is defined, it can be referenced in expressions the same way you would use a variable in a typical programming language.

## `MEAS` commands

The general syntax of a `MEAS` command is:

```
meas <ac|dc|tran> <varName> <measurement> <arguments>
```

- The `meas` can be applied after `ac`, `dc`, or `tran` simulations, and you have to indicate which type.
- The measurement result is saved in a variable, replace `<varName>` with the desired variable name.
- After the variable name, you specify which type of measurement you want. In our example we performed a `MAX` measurement.
- The `arguments` differ for each type of measurement. For the `MAX` measurement we specify a signal, `vdb(nout)`.
- Expressions are supported, for instance we could specify `2*vdb(nout)` as the signal.

## Common AC Measurements

- Magnitude at a specific frequency: use the `FIND/AT=` measurement:

```
let freq=1Meg
meas ac vmag FIND vdb(nout) AT=freq
```

- Frequency at a specific magnitude: use the `WHEN` measurement:

```
let vmag=-23.8
meas ac freq WHEN vdb(nout)=vmag
```

- Signal crossings: use the `WHEN/CROSS=` measurement:

```
* Find the FIRST crossing:
meas ac fmin WHEN vdb(nout)=vmag CROSS=1
```

This syntax is useful when a signal will cross multiple times, as in a band-pass filter circuit.

## Exercise 6: Create and Run a Transient Testbench

Now create a file called `tests/circuit1_tran.sp`, and enter the lines shown below, then run it and VERIFY that the result is correct.

```
* Transient Testbench for circuit1

.include netlists/circuit1.sp

.control
  tran 100u 10m
  hardcopy plots/circuit1_transient.svg v(nin) v(nout)
  plot v(nin) v(nout)
.endc
```

Run the simulation:

```
ngspice tests/circuit1_tran.sp | tee -a log.txt
```

## Exercise 7: Transient Measurements

Now let's measure the signal amplitudes and use them to calculate the **signal gain** at the simulated frequency. The lines below measure the peak-to-peak amplitude of the input, then for the output, then calculate their ratio to obtain the gain. The measurement is limited to the time-window specified by the `from` and `to` arguments.

Add these lines after the transient simulation in your testbench:

```
meas tran vipp PP v(nin) from=0s to=2ms
meas tran vopp PP v(nout) from=0s to=2ms

let gain=vopp/vipp
print gain
```

## Common Transient Measurements

- Peak-to-Peak amplitude: use `PP/from=/to=`

```
meas tran <varname> PP <signal> from=<startTime> to=<endTime>
```

- Average value: use `AVG/from=/to=`

```
meas tran <varname> PP <signal> from=<startTime> to=<endTime>
```

- Slope: use `DERIVATIVE/AT=`

```
meas tran <varname> DERIVATIVE <signal> AT=<time>
```

## Exercise 8: Create and Run a DC Testbench

Now create a file called `tests/circuit1_dc.sp`, and enter the lines shown below, then run it and VERIFY that the result is correct.

```
* DC Testbench for circuit1

.include netlists/circuit1.sp

.control
  dc vin 0 5 0.5
  hardcopy plots/circuit1_dc.svg v(nout)
  plot v(nout)
.endc
```

Run the simulation:

```
ngspice tests/circuit1_dc.sp | tee -a log.txt
```

## Exercise 9: DC Measurements

The lines below measure the maximum and minimum output voltages obtained by the DC sweep. The results are printed using the `echo` command. Add these lines and run the simulation.

```
meas dc vmax MAX v(nout)
meas dc vmin MIN v(nout)

echo "Vout ranges from $&vmin up to $&vmax ."
```

## Using `echo` to Print Numerical Data

The SPICE `print` command is used to print **numerical** data, as in:

```
print vmin vmax
```

The `echo` command is used to print **strings**, like

```
echo "This is a string."
```

You can **convert variables to strings** using the `$&` prefix:

```
echo "This is a string with vmax=$&vmax"
```

## Summary of Exercises

1. AC+DC+SIN Voltage Source (edit `netlists/circuit1.sp`)
2. Capacitor (edit `netlists/circuit1.sp`)
3. Running `ngspice` (use `tee` to create `log.txt` and try the commands indicated in these slides)
4. Create an AC Testbench (create `tests/circuit1_ac.sp`)
5. AC Measurements (add measurements to `tests/circuit1_ac.sp`)
6. Create and Run a Transient Testbench (create `tests/circuit1_tran.sp`)
7. Transient Measurements (add measurements to `tests/circuit1_tran.sp`)
8. Create and Run a DC Testbench (create `tests/circuit1_dc.sp`)
9. DC Measurements (add measurements to `tests/circuit1_dc.sp`)

## Turning in your work

The preferred way to turn in your work is to use `git`. From the Linux terminal:

```
git add *
git commit -a -m "Submitting SPICE 1 assignment"
git push origin master
```

Alternatively you can upload a ZIP file to Canvas containing all your assignment files.