

Symbulator

A symbolic simulator of linear electric circuits

Symbulator 9 Manual

Python / SymPy

Roberto Perez-Franco

Contents

Part 1 — What Symbolator is	5
What it solves	5
What it will not do	5
Where it runs	6
Part 2 — The circuit description	7
The fifteen letters	7
Names	8
Values	8
Brackets	9
Separators	9
Part 3 — Running it, and reading the answer	10
The answer names	10
Signs	11
Asking about one thing	11
Part 4 — Symbols, Define and Expert Mode	13
Define	13
Evaluate	14
Solve	14
Expert Mode	15
Part 5 — Sources, conductances and shorthands	16
Current sources	16
Dependent sources	16
The parallel shorthand	17
Part 6 — Equivalents and the load question	18
Equivalent resistance	18

Thévenin and Norton	18
The load question	19
Part 7 — Operational amplifiers	21
What the op amp reports	21
When ideal is not enough	22
Part 8 — Capacitors, inductors and TR	23
TR	23
No source at all	24
Switched circuits, in two runs	24
Two things that make TR bearable	24
Part 9 — AC, phasors and power	25
The four power answers	25
Three-phase	27
Part 10 — Mutual inductance and transformers	28
m — mutual inductance	28
t — the ideal transformer	29
Part 11 — The s-domain, transfer functions and plots	31
FD	31
t2s and s2t	32
Limits	32
The four plots	32
Resonance	32
Part 12 — Two-ports	34
Finding the parameters	34
A two-port as an element	35
The gain mini-tool	35
Part 13 — The rest of the toolbox	37
By-Hand Equations	37

The Numerical Solver	37
The SPICE Translator	38
The schematic	38
The package	38
Part 14 — Symbolator in a notebook	39
Getting set up	39
A circuit is a string	39
The analyses	41
The Evaluate and Solve cards	43
Expert Mode	44
Cell magics	45
Plotting	45
Notebooks to start from	45
What stays in the app	46
Part 15 — Reference	47
Elements	47
Answer names	47
SI prefixes	48
Analyses	48
Tools and mini-tools	49
Plots	49
Settings	50
What a saved file remembers	50
Outside the tool	50
Who made this	51
The author	51
Acknowledgements	51
Licence	52
Getting in touch	52
Index	53

What Symbulator is

PART 1

A symbolic solver for linear circuits. What it does, what it refuses, and the nine things a circuits course covers that it will not.

Symbulator solves linear electric circuits symbolically. You describe the circuit as text, choose an analysis, and get every current, voltage and power in it — as exact expressions, not decimals, unless you ask otherwise.

It is not SPICE. There is no netlist to compile, no convergence to tune, no timestep. A circuit is a handful of lines, and the answer comes back as algebra you can read.

What it solves

Four analyses, on the same description:

DC	direct current, steady state
AC	sinusoidal steady state at one ω , in phasors
FD	the s-domain, initial conditions included
TR	transient, in the time domain

Fifteen element types, listed in The circuit description (page 7). Three tools that answer a question about a circuit rather than solving it outright — equivalent resistance, Thévenin/Norton and two-port parameters — and four mini-tools that work on values you type: *aa*, *pf*, *gain* and *pz*.

Every value may be a symbol. That is the point of the thing: leave r_1 as r_1 and the answer comes back in terms of r_1 .

What it will not do

Linear circuits only, lumped elements only. Nine topics from a first course in circuits fall outside it, and it is cheaper to say so here than to have you hunt:

- **Fourier series and transforms.** Not implemented.
- **Convolution.** Not implemented.
- **Energy.** You get power; integrating it is yours.
- **Nonlinear devices.** No diode curve, no transistor model. A transistor enters as its small-signal equivalent, which is linear, and works.
- **Transmission lines**, and anything distributed.

- **Noise, tolerance, Monte Carlo.** Not a statistical simulator.

Four more are supported but not automated — **superposition, source transformation, power-factor correction** and **pole-zero work**. Each is a thing you do to the circuit, then solve. No tool does it for you.

Where it runs

Three builds of one interface, and they behave identically:

Online	<code>symbolator.pythonanywhere.com</code> — nothing to install
In your browser, offline	<code>install.symbolator.com</code> — installs as an app
On your machine	<code>symbolator.com/9/local.zip</code> — a folder and a launcher

The solver is also a Python package: `pip install symbolator`.

The circuit description

PART 2

All fifteen element letters with their fields, the naming rules, node 0, brackets, and the SI prefix shorthand. The densest page here, and the one to come back to.

A circuit description is one element per line. The first character of the name picks the type; the fields that follow are positional and comma-separated.

Take the simplest circuit worth describing: a 12 V source driving a 1 kΩ and a 2 kΩ resistor in series. We name the source e1 and the resistors r1 and r2, take the source's lower end as node 0, call its upper end node 1 and the junction of the two resistors node 2, and write each element between the two nodes it joins:

Circuit Description

```
e1,1,0,12
r1,1,2,1'k
r2,2,0,2'k
```

Node 0 is ground, always, and every circuit needs one; the other node names are ours to choose.

The fifteen letters

name is always the first field. Everything else is positional.

	Element	Fields after the name
r	resistor	n1,n2,value
l	inductor	n1,n2,value, <i>initial current</i>
c	capacitor	n1,n2,value, <i>initial voltage</i>
e	voltage source	n1,n2,value
j	current source	n1,n2,value
o	ideal op amp	n+,n-,nout
m	mutual inductance	Lname1,Lname2,M OR Lname1,Lname2,k=0.5
s	short circuit	n1,n2
t	ideal transformer	n1,n2,turns1,turns2
z y h g a b	two-port block	n1,n2,[p11,p12,p21,p22]

Italic fields are optional. The initial conditions on **l** and **c** are read by FD and TR and ignored by DC and AC.

Note what **m** takes: the *names of two inductors*, not nodes. Node order on those inductors is the dot convention — the first node you name in each coupled element is its dotted end. Get it backwards and every current comes back wrong, looking right.

An **m** has no answers of its own. It is a statement about two other elements, and its effect shows up in their currents.



TIP

Sign and direction, once

A current is positive flowing from the first node toward the second. A voltage drop is the first node's potential minus the second's. Every answer in the book follows from those two sentences; if a sign surprises you, read the element's node order.

Names

Letters, digits and underscores. The first character sets the type, so `rload` is a resistor and `r_b` is a resistor.

A name must be identifier-safe. `r-x` looks reasonable and is refused, because `2*i_r-x` would silently read as `2*i_r` minus `x`.

Node names are yours: 1, 2, in, out, b, e, c. Only 0 is reserved.

Values

A value is a number, a symbol, or an expression in either. Here is the divider again with its source left as a symbol we call `vs`, and beside it a second source, `e2`, driving a $2\text{ k}\Omega$ resistor, whose value is five times the voltage at the divider's midpoint:

Circuit Description

```
e1,1,0,vs
r1,1,2,1'k
r2,2,0,1'k
e2,3,0,5*v2
r3,3,0,2'k
```

Because `vs` is a symbol, every answer comes back in terms of it. `e2` is a **dependent source**: its value names `v2`, the voltage at node 2, so it is five times whatever the rest of the circuit puts there. Any answer name may appear in any value. That is how all four dependent-source types are written, and why there is no separate letter for them.

An underscore in a symbol makes a subscript in the typeset answer, and nothing else. Underscores, and subscripts in your answers (in the Course) has it.

SI prefixes

An apostrophe and a letter multiply the value, exactly:

$4.7\text{ 'k} \cdot 10\text{ 'u} \cdot 560\text{ 'm} \cdot 1\text{ 'M}$

Eleven prefixes, peta down to atto; the full table is in “SI prefixes” (page 48). Two things to know here: **case matters** — 'm is milli and 'M is mega — and **a prefixed value is exact**, as is a plain integer. 8000 and 8'k are exact; 8000. and 8E3 are approximate, and that difference shows up the moment an answer is symbolic.

Brackets

Brackets mean exactly three things.

A parallel group, in a resistor's value only. $[r1, r2, r3]$ is those three in parallel. Three resistors of 1 k Ω , 2 k Ω and 2 k Ω side by side between node 1 and ground, written as one element we call r_p :

Circuit Description

```
rp,1,0,[1'k,2'k,2'k]
```

A pair of terminals, for a transformer or a two-port, when neither side sits on ground. $[top, bottom]$. Here a two-port block, $z1$, sits between a source and a load with neither of its ports touching ground; we call its four terminals **a**, **b**, **c** and **d**, top before bottom on each side:

Circuit Description

```
e,1,0,10
r1,1,a,50
r2,b,0,50
z1,[a,b],[c,d],[100,10,20,50]
r1,c,d,1'k
```

The two-node form $z1,1,2$ is the same element with both bottoms on 0, and a four-terminal block reports a current at each of its four terminals rather than two.

A parameter set, the four numbers of a two-port, or a transformer's two turns counts.

Anywhere else, a bracket is an error rather than a guess.

Separators

A line each is the readable form. A colon does the same job on one line — $e,1,0,12:r1,1,2,1'k$ — and both are accepted everywhere, including in a saved file.

Running it, and reading the answer

PART 3

One run returns every answer in the circuit. The names they come back under, what each one means, and the two spellings of any of them.

Paste the description, pick an analysis, press **Run**. There is nothing to select first: a run returns *every* answer in the circuit, not the one you asked about.

The divider of the last chapter, the 12 V source across 1 k Ω and 2 k Ω in series, described as before:

Circuit Description

e1,1,0,12

r1,1,2,1'k

r2,2,0,2'k

Set **Analysis** to *DC* — *direct current* and press **Run Symulator**. It comes back as two node voltages and three answers per element; three of them:

voltage at node 2

$$v_2 = 8 \text{ V}$$

current through r1

$$i_{r1} = \frac{1}{250} \text{ A}$$

power consumed by r1

$$p_{r1} = \frac{2}{125} \text{ W}$$

Exact, because 12, 1'k and 2'k are exact. 1/250 A is 4 mA; see “Settings” (page 50) for how to be shown the decimal instead.

The answer names

A node's voltage is *v* and the node's name. An element's answers are a letter and the element's name.

Name	Answer	Where
v2	voltage at node 2	every node
ir1	current through r1	every element
vr1	voltage drop across r1	every element
pr1	power consumed by r1 — in AC the average (real) power	every element
qr1	reactive power consumed	AC
sr1	complex power consumed	AC
zr1	impedance seen	AC, FD
rr1	resistance seen	DC

Resistance seen and *impedance seen* are the element's own voltage over its own current — useful on a source, where it is the resistance the source is driving.

Every name has two spellings. `ir1` and `i_r1` are the same current, `v2` and `v_2` the same voltage, and capitals make no difference. Use whichever you like, anywhere a name is accepted. Underscores, and subscripts in your answers (in the Course) says why both exist.

Signs

Two sentences, and every sign in the book follows from them:

- A **current** is positive flowing from the element's first node toward its second.
- A **voltage drop** is the first node's potential minus the second's.

So `e1,1,0,12` above reports `ie1 = -1/250` A: current flows *out* of the source's first node into the circuit, which is negative by that rule. Power follows the same rule, so a source delivering power *consumes* a negative one — and because that is the number nobody wants, a source's card shows the **power delivered** instead, as `-pe1`, labelled so. The answer `pe1` itself is unchanged: `p` is always power consumed, in **Evaluate**, in **Solve** and in every saved file, and the card merely shows its negative. In AC a source's card also carries the power factor of the power it delivers.

If a sign surprises you, read the element's node order before doubting the answer.

Asking about one thing

The whole answer set is the point, but three cards narrow it when you want that:

- **Evaluate** takes an expression over the answers — `vr1/ir1`, or `pr1+pr2` — and can carry **Conditions**. The usual functions are there, `sqrt`, `abs`, `exp`, the trigonometric ones, and for a complex answer `re(se)` and `im(se)` give its real and imaginary parts, `conj(se)` its conjugate and `arg(se)` its angle. `limit(s*vo, s, oo)` takes a limit, and a condition at infinity, such as `t = oo`, is read as one.
- **Solve** solves equations written over the answers.

- In TR, the results can be limited before they are computed, which saves real time. Limiting the results to save time (in the Course) has it.

Symbols, Define and Expert Mode

PART 4

Any value may be a symbol, and the answer comes back in terms of it. Define supplies values; Expert Mode adds equations, unknowns and conditions to the system.

This is the part that is not SPICE. Any value may be a symbol, and the answer is an expression rather than a number.

A divider with no values at all: a source and two resistors in series, every value left as a symbol. We name the symbols *vs*, *ra* and *rb*, and the elements *e*, *r1* and *r2*:

Circuit Description

```
e,1,0,vs  
r1,1,2,ra  
r2,2,0,rb
```

Run in DC, the voltage across the second resistor comes back as

voltage drop across r2

$$v_{r2} = \frac{rb\,vs}{ra + rb}$$

The divider rule, derived rather than recalled. No values were given and none were needed.

Define

Define gives values to symbols without editing the description. One per line, name = value:

Define

```
ra = 4'k  
rb = 6'k  
vs = 10
```

Run the same circuit again and the same answer now reads

voltage drop across r2

$$v_{r2} = 6\text{ V}$$

The description still says *ra*, so changing a value is one line and a re-run, not an edit to the circuit.

A symbol left undefined stays symbolic. You can define some and not others, and get a partly symbolic answer — which is usually what you want when one component is the unknown.

Evaluate

Evaluate takes expressions over the answers, one per line, after the solve:

```
Evaluate  
vr1/ir1  
pr1 + pr2
```

Conditions constrain the evaluation without changing the circuit — $r_a = r_b$, say, to see what the divider does when the two are equal.

Solve

Solve goes the other way. Give it an equation over the answers and an unknown, and it finds the value that satisfies it.

Suppose we want the resistor that puts 7 V across **r2**. We take r_b out of **Define**, so that it stays a symbol and the answers come back in terms of it, leave r_a and v_s defined, and run again. Then, in the **Solve** card, the equation and the unknown:

```
Equation(s) to solve in terms of the results  
vr2 = 7
```

```
Unknown(s) to solve for  
rb
```

Press **Solve equations**, and the card returns $r_b = 9333.33 \Omega$: the resistance that puts 7 V across **r2** when r_a is 4 k Ω and v_s is 10 V. A value the equation needs but the circuit does not know can also be given in the card's **Conditions** box, $r_a = 4 \text{ k}$ on a line of its own, instead of in Define.

WARNING

More than one answer is normal

An equation on a power or on a product of answers is quadratic in its unknown, so it can have two roots, and both may be physical. Symbulator returns every root and offers a picker under the outputs. It does not choose for you, and the first one shown is not more correct than the second — read the picker before quoting an answer.

Real solutions only is a tick on the same card. Leave it on unless a complex root is meaningful in your problem.

Expert Mode

Expert Mode adds to the system the solver assembles, rather than working on its answers afterwards. Three boxes:

- **equations** — extra relations, one per line or separated by and
- **unknowns** — extra symbols to solve for
- **conditions** — constraints applied to the result

Use it when the condition has to be part of the solve itself, so that *every* answer comes back with it applied: the whole circuit at the component value that meets the condition, not the component alone. Put the condition in *equations* and the component in *unknowns*, and the circuit is solved and the condition satisfied in one system. For the component's value on its own, the **Solve** card above is the shorter route, and it is the one this documentation's worked examples take.

Values in these boxes are parsed exactly as circuit values are, so 4.7'k works and a bare 4.7k does not.



TIP

See what was assembled

Tick **Show equations** in **Settings** and an **Equations** card appears with the system the solver actually built — the fastest way to find out why an Expert Mode run did not do what you expected.

Sources, conductances and shorthands

PART 5

Current sources, dependent sources of all four kinds, conductance as a value, and the parallel-resistor shorthand.

Current sources

`j` takes the same four fields as `e`, and its current flows **from the first node to the second, inside the element**. A source written `j,0,1` therefore pushes current into node 1.

A 3 mA source feeding two resistors in parallel, one of them given not as a resistance but as a conductance g . We write the source with its arrow pointing into node 1, `j,0,1`, name the resistors `r1` and `r2`, and give the first the value $1/g$, since there is no conductance element and a conductance is a resistance written as its reciprocal:

Circuit Description

```
j,0,1,3'm  
r1,1,0,1/g  
r2,1,0,2'k
```

Run in DC, the voltage across the pair comes back in terms of that conductance:

voltage drop across `r1`

$$v_{r1} = \frac{6}{2000g + 1}$$

The algebra took care of the reciprocal itself.

Dependent sources

There is no letter for them. An `e` or a `j` whose value names an answer is dependent, and that covers all four textbook types:

	Value looks like
VCVS	<code>e2,3,0,5*v2</code>
VCCS	<code>j2,3,0,g*v2</code>
CCVS	<code>e2,3,0,r*ir1</code>
CCCS	<code>j2,3,0,beta*irb</code>

The controlling quantity is just an answer name, so it may be any answer in the circuit, and it may appear inside a larger expression.

The parallel shorthand

In a **resistor's value only**, a bracketed list is that group in parallel. A 9 V source across three resistors of 1 k Ω , 2 k Ω and 2 k Ω side by side, the three written as one element we call `rp`:

Circuit Description

```
e,1,0,9
```

```
rp,1,0,[1'k,2'k,2'k]
```

Run in DC, the current through the group is

current through rp

$$i_{rp} = \frac{9}{500} \text{ A}$$

which is 18 mA: 1 k Ω || 2 k Ω || 2 k Ω is 500 Ω . The entries may be symbols, and the list may be any length.

It saves describing three elements and three nodes when the group is not the thing you are studying. When it *is* the thing you are studying, describe them separately so each reports its own current.

WARNING

Brackets are not general

[...] means a parallel group here, a terminal pair on a transformer or two-port, and a parameter set. Anywhere else it is an error rather than a guess — see The circuit description (page 7).

Equivalents and the load question

PART 6

Equivalent resistance, Thévenin and Norton at a pair of nodes, and the load answers that come with them.

Two of the four tools answer a question *about* a circuit instead of solving it. Both take a pair of nodes.

Equivalent resistance

Find equivalent → *Resistance / impedance*, with the two nodes.

Three resistors and no source: a $100\ \Omega$ in series with a $200\ \Omega$ and a $300\ \Omega$ in parallel, and the question is the resistance of the whole seen from its two ends. We name them r_1 to r_3 , call the free end node **1** and the junction node **2**, and put the far end on ground:

Circuit Description

$r_1, 1, 2, 100$

$r_2, 2, 0, 200$

$r_3, 2, 0, 300$

Open **Find equivalent**, choose *Resistance / impedance*, and give the two nodes **1** and **0**:

equivalent resistance

$$R_{eq} = 220\ \Omega$$

$100\ \Omega$ in series with $200 \parallel 300$. No source is needed and none is present: this is a question about the network, not about a circuit that runs.

If the circuit does contain independent sources, they are suppressed first — voltage sources shorted, current sources opened — which is the textbook procedure. Dependent sources are not suppressed, because they are part of the network's behaviour.

In AC the same tool returns an impedance, and the menu says so.

Thévenin and Norton

Find equivalent → *Thévenin / Norton*, with the two nodes. One run returns both forms, because they are the same two numbers.

A 20 V source and a divider of $50\ \Omega$ and $150\ \Omega$, and the question is what the circuit looks like from the divider's midpoint: a single source behind a single resistance. We describe the three elements, naming them e , r_1 and r_2 , and call the midpoint node **2**:

Circuit Description

e,1,0,20

r1,1,2,50

r2,2,0,150

Open **Find equivalent**, choose *Thévenin / Norton*, and give the two terminals **2** and **0**:

Thévenin voltage

$$v_{th} = 15 \text{ V}$$

equivalent resistance

$$R_{eq} = \frac{75}{2} \Omega$$

Norton current

$$i_{no} = \frac{2}{5} \text{ A}$$

The Norton current is reported in the direction it actually flows, from the first node to the second. Carry the sign through rather than assuming a positive value.

A fourth answer comes with those three, unasked:

maximum deliverable power

$$p_{max} = \frac{3}{2} \text{ W}$$

pmax is the maximum power transfer result – the power delivered when the load equals R_{eq} , which for this circuit is $15^2/(4 \times 37.5)$.

The load question

Under the two node boxes is a tick: **Are you running a problem with a load connected to this equivalent circuit?** It adds three more answers, each an expression in a symbol load:

ir1 current in the load

vr1 voltage drop in the load

pr1 power consumed in the load

Give load a value in **Define** and all three become numbers.

WARNING

The load formulas assume a load and nothing else

They hold only when the load is the one thing across those terminals. Connect anything more and none of them applies; the problem has to be run as a circuit again. When a load is not the only thing attached (in the Course) covers the button

that writes that circuit for you.

Operational amplifiers

PART 7

The `o` element is an ideal op amp — a nullor. What that buys you, what it hides, and how to model a real one when the ideal answer is not enough.

`o` takes three nodes, in this order:

```
o1,n+,n-,nout
```

No power rails, no supply, no gain figure. It is an **ideal** op amp: a nullor, which is the statement that the two inputs are at the same potential and draw no current, and that the output supplies whatever the rest of the circuit needs.

An inverting amplifier: a 2 V source into the op amp's inverting input through 10 k Ω , a 47 k Ω resistor feeding the output back to that input, and the non-inverting input on ground. We name the source `e` and the resistors `r1` and `r2`, call the summing junction **m** and the output **o**, and write the op amp's three nodes in the order the element wants, non-inverting input first:

Circuit Description

```
e,1,0,2
```

```
r1,1,m,10'k
```

```
r2,m,o,47'k
```

```
o1,0,m,o
```

Run in DC, the output is

voltage at node o

$$v_o = -\frac{47}{5} \text{ V}$$

That is -9.4 V from 2 V, a gain of -4.7, which is -47k/10k, arrived at from the nullor conditions rather than from the formula.

Note the node order — **o** is the non-inverting input and **m** the inverting one. Swap them and the feedback becomes positive, which is a different circuit and will usually refuse to solve.

What the op amp reports

An `o` reports a **current** and a **power**, and no voltage drop — its input terminals are at the same potential by definition, so a drop across it would not mean anything. The current is the one it sources or sinks at its output.

When ideal is not enough

There is no finite-gain op amp element. Model one as a dependent source, which is what a finite-gain op amp is. The same amplifier, with the `o` element replaced by a voltage source we call `ea`, driving the output node with 100,000 times the voltage between the two inputs — the non-inverting one, at ground, minus the inverting one at `m`:

Circuit Description

```
e,1,0,2  
r1,1,m,10'k  
r2,m,o,47'k  
ea,o,0,100000*(0-vm)
```

Run in DC, the output is now

voltage at node o

$$v_o = -\frac{9400000}{1000057} \text{ V}$$

which is -9.39946 V against the ideal -9.4 V : the two differ in the fifth figure, which is the honest way to find out whether the ideal model was good enough for your problem.

The same shape gives you finite input impedance (a resistor across the inputs) and non-zero output impedance (a resistor in series with `ea`). Once you are modelling it, the op amp is just circuit elements.



TIP

The ideal answer first

Run the nullor version, then the modelled one, and compare. If they agree to more figures than your problem needs, the ideal answer was the right one and the model was wasted work.

Capacitors, inductors and TR

PART 8

Storage elements, initial conditions in the fifth field, transient analysis, switched circuits as two runs, and plotting an answer against time.

c and l take the same four fields as a resistor, plus an optional fifth – the initial voltage on a capacitor, the initial current in an inductor.

```
c1,2,0,10'u no initial charge
c1,2,0,10'u,5 starting at 5 V
l1,2,0,2'm,0.3 starting at 300 mA
```

The fifth field is read by TR and FD only. DC and AC ignore it, which is correct: in DC a capacitor is an open circuit and the initial condition has nothing to say.

TR

Choose **TR** and the answers come back as functions of t .

A 10 V source charging a 1 μF capacitor through a 1 k Ω resistor, with nothing stored in the capacitor to begin with. In TR a source with a plain numerical value is a step that begins at $t = 0$, so this is the source switched on at that instant. We name the three elements e, r and c, and write the capacitor with no fifth field:

```
Circuit Description
e,1,0,10
r,1,2,1'k
c,2,0,1'u
```

Set **Analysis** to *TR – transient / time domain* and run:

voltage drop across c

$$v_c = 10 - 10e^{-1000t}$$

The familiar step response, $\tau = RC = 1$ ms, and derived rather than recalled. Answers are valid for $t \geq 0$.

Give the capacitor a starting voltage, 4 V in its fifth field, and run again:

```
Circuit Description
e,1,0,10
r,1,2,1'k
c,2,0,1'u,4
```

voltage drop across c

$$v_c = 10 - 6e^{-1000t}$$

Only the coefficient changed: it starts at 4 and climbs to 10, so the swing is 6 rather than 10.

No source at all

A natural response needs no source — the energy is already in the element. A capacitor charged to 10 V and left across a 1 kΩ resistor:

Circuit Description

r,1,0,1'k

c,1,0,1'u,10

Run in TR:

voltage drop across c

$$v_c = 10e^{-1000t}$$

Second-order circuits are no different: add an inductor and the answer comes back over-, critically or under-damped as the values dictate. You do not select the case; the algebra does.

Switched circuits, in two runs

There is no switch element and no $t < 0$. A switched problem is two runs, and this is the whole method:

1. **Run the $t < 0$ circuit in DC.** Read the capacitor voltages and inductor currents.
2. **Write the $t > 0$ circuit**, putting those numbers in the fifth field.
3. **Run it in TR.**

That is the same thing you would do by hand, and step 1 is where the initial conditions come from rather than being assumed.

Two things that make TR bearable

Limit the results. Every TR answer costs an inverse Laplace transform, so a circuit with a dozen elements pays for dozens of them. Ask for the one or two you want and the wait collapses. Limiting the results to save time (in the Course) has it.

Plot it. The Plot card takes *Plot a function of time (TR)*, an answer name, and a time range. The s-domain, transfer functions and plots (page 31) covers all four plot types.

AC, phasors and power

PART 9

Sinusoidal steady state at one ω , rectangular or polar, and the four power answers — including the one setting that changes an answer rather than its appearance.

Choose **AC** and give an ω . Elements keep their own units — henries and farads, not reactances — and the solver does the conversion.

A 10 V source driving a 50 Ω resistor and a 50 mH inductor in series, at 1000 rad/s. We name the three e, r and l, and write the inductor as its inductance:

Circuit Description

e,1,0,10

r,1,2,50

l,2,0,0.05

Set **Analysis** to *AC — alternating current*, put **1000** in the ω — **angular frequency** box, and run. At that frequency the inductor is 50 Ω of reactance, so the loop's impedance is 50 + 50j, and the answers say so:

current through r

$$i_r = 0.1 - 0.1j \text{ A}$$

voltage drop across l

$$v_l = 5.0 + 5.0j \text{ V}$$

You may also give reactances directly, as ohms, if that is how the problem is stated — a resistor-valued l is not a thing, so put the reactance in as an impedance and leave ω out of it.

Rectangular or polar is a display choice: tick **Show AC answers as polar phasors** in **Settings** and 0.1 - 0.1j reads 0.1414 $\angle$ -45°. Nothing about the answer changes. The **aa** mini-tool does the same conversion for one value you type.

The four power answers

In AC an element reports more than in DC:

-
- p average (real) power consumed, $P = \text{Re } \mathbf{S}$, in watts — ap is the same answer under the calculator's name
 - q reactive power consumed, $Q = \text{Im } \mathbf{S}$, in var
 - s complex power consumed, $\mathbf{S} = P + jQ$, in VA
 - z impedance seen
-

The three are P , Q and $\mathbf{S}$ as every book writes them, whichever way the RMS setting is: RMS changes their values, not their names. A source's card shows the delivered forms, -pe, -qe and -se, and the power factor of the power it delivers; the answers pe, qe and se are the consumed values all the same.

The same shape with a $30\ \Omega$ resistor and a 40 mH inductor, at 1000 rad/s again, so that the impedance is $30 + 40j$, a 3-4-5 triangle:

Circuit Description

```
e,1,0,10
r,1,2,30
l,2,0,0.04
```

Run in AC at $\omega = 1000$, the source's card reads

complex power delivered by e

$$-s_e = 0.6 + 0.8j \text{ VA}$$

the negative of s_e , since a source's card shows what it delivers. So $P = 0.6$ W, $Q = 0.8$ var, $|\mathbf{S}| = 1.0$ VA, and the card's last row gives the power factor of that delivered power, 0.6 lagging. The card's next two rows are those two parts, -pe = 0.6 W and -qe = 0.8 var, and $\text{abs}(s_e)$ in **Evaluate** is the apparent power.

An inductor or a capacitor reports only its complex power. That is not an omission: a pure reactance consumes no average power, and a zero printed every time would be noise; its reactive power is $\text{im}(s_l)$ in **Evaluate**.

The **pf** mini-tool takes one value, and which of two things it decides what comes back. Given a complex power such as s_e — or any complex value, an impedance, an expression with symbols in it — it returns the power factor of that value as given, $|P|/|S|$, with the word the value carries when it is a number and symbolic when it is not. Given the **name** of an element of the AC solve, e or r1, it returns the power factor with its lead/lag sense, and it works only when that element's voltage and current came out as numbers.

The two forms read different powers, and that is what makes the word right. A variable such as s_e , s_j or s_{r1} is read as it stands: the complex power *consumed*, which is what every element stores, source or load — and its word is that power's own, so s_e at a source says the opposite word to e, the consumed power being the opposite of the delivered one. A name is read by the element's kind, as the calculator read it: an impedance (r, l, c) on the power it *consumes*, so an inductive load reads lagging; a source (e, j) on the power it *delivers*, the current negated first, so a source reads the circuit it sees, and a source feeding an inductive load says lagging like the load. The value is the same either way; only the word depends on it, and a source read on s_e says the opposite word. Under a name's reading the tool says which power in words — *the power delivered by source e, the*

power consumed by impedance r1; under a value's it says nothing, the value being what you gave it.

 WARNING

RMS is the one setting that changes an answer

RMS phasors, in **Settings**, is not a display choice. Off means peak amplitude, the $\div 2$ convention. For the same phasor magnitudes, RMS reports twice the power that peak does — a 10 V source across $5\ \Omega$ gives 10 W with the tick off and 20 W with it on. With the tick on every current and voltage label on the cards reads *effective*, since the magnitudes are then RMS values, the textbook's effective values. Match the book you are working from before comparing numbers.

Three-phase

There is no three-phase mode, and none is needed. A three-phase circuit is an AC circuit with three sources whose values carry the phase — $(120\angle 0^\circ)$, $(120\angle -120^\circ)$, $(120\angle 120^\circ)$ — and Y or Δ is just how you wire the nodes. Balanced and unbalanced are the same description with different values.

Line and phase quantities are then read off the elements directly: there is no separate answer for them, because each is some element's own current or voltage.

Mutual inductance and transformers

PART 10

`m` couples two named inductors and the node order carries the dots. `t` is an ideal transformer, two-terminal or four.

`m` – mutual inductance

`m` names **two inductors**, not two nodes:

```
m1,l1,l2,0.01
```

Those are the names of two `l` elements already in the circuit, and the last field is M in henries. It may be a symbol, and it may carry an SI prefix. A book that gives the coupling as a coefficient rather than an inductance is written as it states it:

```
m1,l1,l2,k=0.5
```

and Symbulator works out $M = k\sqrt{L_1L_2}$ itself when it reads the description. k may be a symbol too, in which case M stays symbolic.

The two coils may instead be written as impedances in AC, `r` elements with positive imaginary values, coupled by a positive imaginary M in ohms, `m1,r1,r2,2j`; `k=` works there as well, giving $j k \sqrt{|Z_1||Z_2|}$.

The line is checked before anything runs. Both names must be elements of the circuit and of one kind, two inductors in henries or two impedances in ohms; one of each is refused, where it used to be accepted and answered wrongly. A numeric value on either coil or on the coupling must be real and positive in henries, or positive imaginary in ohms, with no resistive or capacitive part; a symbol passes. With everything numeric the coupling may not exceed $\sqrt{L_1L_2}$, which is a coefficient of 1, and a `k=` must lie between 0 and 1. And a pair written in ohms is an AC description: in DC, TR and FD it is refused, and the coils are to be written in henries.

An `m` has no answers of its own. It is a statement about two other elements; its effect turns up in their currents.

The dots are the node order

Symbulator cannot draw a dot, so **the first node you name in each coupled inductor is its dotted end**. That is the whole convention, and it is the one most likely to hand you a correct-looking wrong answer.

Two 30 mH coils coupled by 10 mH, the first across a 10 V source and the second across a $30\ \Omega$ load, at 1000 rad/s. We name the coils `l1` and `l2` and the load `r2`, and we write

each coil's dotted end first, node 1 for the primary and node 2 for the secondary:

```
Circuit Description
e,1,0,10
l1,1,0,0.03
l2,2,0,0.03
m1,l1,l2,0.01
r2,2,0,30
```

Run in AC at $\omega = 1000$, the current in the load is

current through r2

$$i_{r2} = 0.0621 - 0.0552j \text{ A}$$

Now write the second coil the other way round, `l2,0,2,0.03`, the same coil with its dot at the other end, and run again:

current through r2

$$i_{r2} = -0.0621 + 0.0552j \text{ A}$$

Exactly the negative. Same magnitude, opposite sign, no error message. Nothing will warn you.

So: read the schematic, decide which end of each coil carries the dot, and name that node first in **both** coupled elements.

t — the ideal transformer

```
t1,n1,n2,turns1,turns2
```

The two-node form names the top terminal of each side and grounds the other two. A 120 V source on the primary, a 5Ω load on the secondary, ten turns to one; we name the transformer `t1` and the load `r1`, and call the two tops node **1** and node **2**:

```
Circuit Description
e,1,0,120
t1,1,2,10,1
r1,2,0,5
```

Run in DC, the load's voltage and current are

voltage drop across r1

$$v_{r1} = 12 \text{ V}$$

current through r1

$$i_{r1} = \frac{12}{5} \text{ A}$$

Ten to one, so a tenth of the 120 V across the load.

An ideal transformer has no leakage, no magnetising current and no losses. For a real one, model it as coupled inductors with m instead — that is the same choice you make on paper.

Four terminals

When neither side sits on ground, name all four with bracketed pairs:

```
t1, [t1, b1], [tr, br], [10, 1]
```

[t1, b1] is the left port, top terminal first; [tr, br] the right. The turns must then be bracketed too. The two-node form is exactly this with both bottoms on 0.

A transformer reports **a current at each live terminal** rather than one current, since the two sides carry different ones. The names are the element's name followed by the node's: for t1 with ports at nodes **1** and **2**, they are it11 and it12.

The same bracketed-pair form works on two-port blocks; see Two-ports (page 34).

The s-domain, transfer functions and plots

PART 11

FD returns answers in s , which is where transfer functions come from. The $t2s$ and $s2t$ shortcuts, and all four plot types.

FD

Choose **FD** and the answers come back as functions of s , initial conditions included.

The RC circuit of Capacitors, inductors and TR (page 23) with its source left as a symbol we call v_s , a $1\text{ k}\Omega$ resistor and a $1\text{ }\mu\text{F}$ capacitor:

Circuit Description

e,1,0, v_s

r,1,2,1'k

c,2,0,1'u

Set **Analysis** to *FD – complex frequency domain* and run:

voltage drop across c

$$v_c = \frac{1000 v_s}{s + 1000}$$

Leave the source symbolic and the answer *is* the transfer function: divide by v_s and you have $H(s) = 1000/(s + 1000)$, a first-order low pass with its pole at -1000 . Nothing is labelled “transfer function” because nothing needs to be – it is the answer with the input left as a symbol.

The **pz** mini-tool reads its poles and zeros. Give it the ratio:

Value

v_c/v_s

It returns the poles, the roots of the denominator, and the zeros, the roots of the numerator: here one pole at **-1000** and no zero. A factor the numerator and the denominator share is cancelled first, and a repeated root is written once with its multiplicity, as in -4×2 .

t2s and s2t

Two shortcuts convert an expression between the domains:

```
t2s(10*exp(-1000*t))  
s2t(1000/(s + 1000))
```

Curly brackets are shorthand for the same thing inside a value, so a source may be written in t and used in an FD run.

Limits

Evaluate takes a limit over the answers as `limit(expression, variable, point)`. The initial- and final-value theorems are two of them:

```
limit(s*vo, s, oo)  
limit(s*vo, s, 0)
```

A line in **Conditions** at infinity is read as a limit too, so $\omega = \infty$ gives a gain at high frequency and, in TR, $t = \infty$ a final value.

The four plots

The Plot card offers four, and which ones are available depends on the analysis:

Plot	Analysis	What it wants
Plot a function of time	TR	an answer name, and a time range
Node plot of a variable	FD	an answer name, and a frequency range
Node plot of transfer function $H(s)$	—	$H(s)$ typed directly
Plot a variable against another	DC	two answer names, and a sweep range

The third needs no circuit at all: type $1000/(s+1000)$ and get its magnitude and phase. Useful when the transfer function came from somewhere else, or when you are checking a hand derivation.

The fourth is a **DC sweep** — one answer plotted against another as a value varies, which is how a load line or a maximum-power curve is drawn. Give it the two names and a range.

Resonance

There is no resonance tool. Resonance is the frequency that clears the reactance, so it is a **Solve** problem: take the impedance answer, set its imaginary part to zero, and solve

for ω . The same works for half-power points, which makes bandwidth and Q two more lines of algebra rather than a feature.



TIP

Filters are just circuits

There is nothing filter-shaped in Symbulator. Describe the circuit, run FD, and read the transfer function; run the Bode plot to see it. A passive filter and an active one differ only by having an o in them.

Two-ports

PART 12

Six parameter sets out of any network, the same six letters as circuit elements, and the gain tool.

Finding the parameters

Find equivalent → *Two-port parameters*, two nodes, and a kind:

- z** impedance
- y** admittance
- h** hybrid
- g** inverse hybrid
- a** transmission
- b** inverse transmission

A T of three resistors, $10\ \Omega$ and $30\ \Omega$ along the top and $20\ \Omega$ down the middle, seen as a two-port with one port at each end. We name the three r_a , r_b and r_c , call the two ends node **1** and node **3** and the middle node **2**, and put the bottom rail on ground:

Circuit Description

$r_a, 1, 2, 10$

$r_b, 2, 0, 20$

$r_c, 2, 3, 30$

Open **Find equivalent**, choose *Two-port parameters*, set the kind to **z**, and give the ports as **1** and **3**:

open-circuit input impedance

$$z_{11} = 30\ \Omega$$

open-circuit reverse transfer impedance

$$z_{12} = 20\ \Omega$$

open-circuit forward transfer impedance

$$z_{21} = 20\ \Omega$$

open-circuit output impedance

$$z_{22} = 50\ \Omega$$

10 + 20 and 30 + 20 on the diagonal, the shared 20 off it. $z_{12} = z_{21}$ because the network is reciprocal — passive and source-free. A network with a dependent source in it will not be, and that is a result rather than a mistake.

Run it again with **y** and you get the inverse matrix, as you should.

A two-port as an element

The same six letters are element types. Give the block its four parameters in brackets and it behaves as that two-port. A block known only by its **z** parameters, between a source with 50 Ω in series and a 1 k Ω load; we name the block **z1** and call its two ports **a** and **b**:

Circuit Description

```
e,1,0,10
r1,1,a,50
z1,a,b,[100,10,20,50]
r1,b,0,1'k
```

The order is [p11, p12, p21, p22]. Leave the brackets off and the parameters become symbols — z_{p11} , z_{p12} and so on — which is how you solve for the block a circuit would need.

A two-port reports a current at each port, named for the element and then the node — a block **z1** with ports at **a** and **b** reports $iz1a$ and $iz1b$. With four terminals named, it reports four.

Four terminals

z1, [tl, bl], [tr, br], [...] when neither port sits on ground — the same form as the transformer in Mutual inductance and transformers (page 28). A port whose far side floats gets its own reference automatically, and Symbulator says so in the notes rather than refusing.

The gain mini-tool

gain takes a two-port and a load and returns what a cascade designer actually wants:

G_{v}	voltage gain
G_{i}	current gain
G_{p}	power gain
Z_{in}	input impedance

It is a mini-tool, so it takes the four parameters and the load as values you type rather than needing a circuit.



TIP

Interconnections are wiring, not formulas

Series, parallel and cascade combinations have no tool. Wire two blocks together in one description and solve — the combination falls out, and you never have to remember which interconnection adds which matrix.

The rest of the toolbox

PART 13

By-Hand Equations, the Numerical Solver, the SPICE Translator, the schematic, and the solver as a Python package.

Five things the app does that are not a circuit solve. Four of them are documented nowhere else.

By-Hand Equations

Tick it and a card appears with the system a first course would have you write: **nodal**, with supernodes, or **mesh**, with supermeshes. It names the method it used and counts the supernodes or supermeshes it needed. Mesh currents turn clockwise. **Flip the mesh currents** turns them all the other way, arrows and signs together.

It is written independently of the classic solve and then compared with it. The classic solve stays the authority and is never fed by the by-hand one — the point is that two different derivations agree, which is worth something when you are checking your own work against the machine's.

Use it to find where your hand derivation went wrong, not to avoid doing one.

The Numerical Solver

A property of its own, at /eqsheet/, reached from the app's **Explore Numerically** card. It is a TK!Solver-style equation sheet rather than a circuit solver: a list of equations, each variable marked **Known** or **Unknown**, and a numerical solve over the lot.

Two ways in:

- **From a solve.** The card hands over the system and the results from a DC or numeric- ω AC run, already loaded.
- **Empty.** Open it with nothing and type your own equations. It does not need a circuit.

Every unknown carries a **Restriction** — *Unrestricted*, *Positive*, *Negative*, or *Range* with a from/to pair. This is not cosmetic: a restricted solve runs a bounded least-squares rather than the unrestricted root finder, and a system whose root lies outside its restriction says so instead of handing back a boundary value that looks like an answer.

Any identifier works as a variable name, Python keywords included — *is*, the natural name for a source current, is accepted.

It needs SciPy, which the offline builds bundle.

The SPICE Translator

Both directions, in a card of its own.

Symbulator to SPICE returns a netlist with a title line and `.end`, ready to paste into ngspice or LTspice. Anything it cannot translate comes out as a `*` comment and is reported — it never quietly drops an element. Dependent sources translate whenever their value is affine in node voltages.

SPICE to Symbulator takes the linear subset of a netlist. Elements and directives outside that subset are dropped and reported, never guessed at.

It is a prototype, and says so on the card. Read the warnings both ways.

The schematic

Symbulator draws the circuit you described, from the description alone. It is worth a glance before trusting any answer: the fastest way to catch a node typed wrong is to see the picture disagree with the circuit in your head.

It is a drawing of what you *wrote*, not of what you meant — which is exactly what makes it useful.

The package

The solver is on PyPI and does not need the app:

```
pip install symbulator
```

```
from symbulator import dc
r = dc("e,1,0,10:r1,1,2,1'k:r2,2,0,2'k")
r["v2"]
```

Every answer is a SymPy expression, so it substitutes, differentiates and lambdifies like any other. Symbulator in a notebook (page 39) is the chapter on using it from a Jupyter notebook: the cell magics, the Evaluate and Solve cards as functions, and the notebooks to start from.

Symbulator in a notebook

PART 14

The solver as a Python package, used from a Jupyter notebook: a circuit is a string, an answer is a SymPy expression, and the Evaluate and Solve cards are two functions. With the notebooks to start from.

The app and the package are one solver. What a card does to a circuit, a cell in a notebook does, and the answers are the same ones.

What a notebook adds is Python. An answer is a SymPy expression, so it substitutes, differentiates, plots, and goes on into the rest of a calculation.

Getting set up

On your own computer, in a terminal:

```
pip install symbulator[notebook]
```

That brings JupyterLab, NumPy and Matplotlib with the package. On Google Colab the package is not installed, and the notebooks below begin with a cell that installs it there and does nothing anywhere else:

```
import sys
if "google.colab" in sys.modules:
    %pip install -q symbulator matplotlib
```

A circuit is a string

The description is the one the **Input File** card takes, an element to a line or the elements joined by colons. The circuit description (page 7) has the grammar.

```
type
from symbulator import dc, draw

divider = """
e1,1,0,12
r1,1,2,1'k
r2,2,0,2'k
"""
draw(divider)
```

`draw` shows the schematic under the cell, from the description alone, as the app does. `dc` runs the analysis and returns every answer in the circuit:

```
type
res = dc(divider)
res["v2"]
```

```
returns
8
```

Answers have the app's names. Either spelling finds an answer, `ir1` or `i_r1`; the package stores the second.

```
type
res["ir1"], res["i_r1"], res["pr1"]
```

```
returns
(1/250, 1/250, 2/125)
```

The answers are exact because the values are, and a value may be a symbol, as in the app:

```
type
sym = dc("e,1,0,vs:r1,1,2,ra:r2,2,0,rb")
sym["vr2"]
```

```
returns
rb*vs/(ra + rb)
```

Being SymPy, it takes a value for each symbol with `subs`:

```
type
sym["vr2"].subs({"vs": 10, "ra": 4000, "rb": 6000})
```

```
returns
6
```

The Rounding setting is `rounded`, which returns a copy and leaves the exact answers alone:

```
type
res.rounded(4)["ir1"]
```

```
returns
0.004000
```

The analyses

Each choice in **Analysis** is a function, and each tool of **Find equivalent** is one too. Nodes are given as strings.

In the app	In a cell
<i>DC</i>	<code>dc(circuit)</code>
<i>AC, with ω</i>	<code>ac(circuit, omega=1000)</code>
<i>FD</i>	<code>fd(circuit)</code>
<i>TR</i>	<code>tr(circuit)</code>
<i>Thévenin / Norton</i>	<code>th(circuit, "2", "0")</code>
<i>Resistance / impedance</i>	<code>er(circuit, "1", "0")</code>
<i>Two-port parameters</i>	<code>port(circuit, "1", "3", "z")</code>

In AC, omega is a number, a symbol's name such as "w", or an expression. The equivalent tools take domain="ac" and an omega the same way. An AC answer is complex, and polar shows it as magnitude and angle:

```
type
from symbulator import ac, polar

r1 = "e,1,0,10:r,1,2,50:l,2,0,0.05"
ac(r1, omega=1000)["ir"]
```

```
returns
0.1 - 0.1*I
```

```
type
polar(ac(r1, omega=1000)["ir"])
```

```
returns
0.1414/-45.00°
```

A TR answer is a function of t, the symbol the answers use. Import it to use the same one, and at substitutes a time by name:

```
type
from symbulator import tr, t

rc = "e,1,0,10:r,1,2,1'k:c,2,0,1'u"
tr(rc)["v2"]
```

```
returns
10 - 10*exp(-1000*t)
```

```
type
tr(rc).at("v2", t=0.001)
```

```
returns
6.32120558828558
```

The capacitor's voltage is the voltage of node 2, since its other end is ground. That is how to read any voltage that a TR or FD result does not give.

What TR and FD do not give. A TR or FD result holds the node voltages and the element currents, and nothing worked out from them: no element's voltage drop and no element's power. DC and AC store both. TR and FD leave them out because every answer stored costs an inverse Laplace transform, and these are one subtraction and one product away from what is stored. The drop across an element is the difference of the node voltages at its two ends, ground counting as zero:

```
type
res = tr(rc)
res["v1"] - res["v2"]
```

```
returns
10*exp(-1000*t)
```

That is the voltage across r . The power it consumes is the drop times its current, the rule DC uses for every element:

```
type
(res["v1"] - res["v2"]) * res["ir"]
```

```
returns
exp(-2000*t)/10
```

A source follows the same rule, and reads negative while it delivers:

```
type
res["v1"] * res["ie"]
```

```
returns
-exp(-1000*t)/10
```

WARNING

Not in FD

In FD an answer is a transform, and the product of two transforms is not the transform of a product: $V(s)I(s)$ is the transform of a convolution, not of a power. Take a power in TR, or in AC, where it is stored.

A Thévenin equivalent comes back with its four answers as attributes:

```

type
from symbulator import th

eq = th("e,1,0,20:r1,1,2,50:r2,2,0,150", "2", "0")
eq.vth, eq.z, eq.ino, eq.pmax

```

```

returns
(15, 75/2, 2/5, 3/2)

```

The Evaluate and Solve cards

Once a circuit is solved, the app lets you write an expression in its answers, and solve an equation in them. Those are two functions, `evaluate` and `solve`, and they take the result of any analysis above.

```

type
from symbulator import evaluate, solve

res = dc("e1,1,0,vs:r1,1,2,1'k:r2,2,0,1'k")
evaluate(res, "v2/vs")

```

```

returns
1/2

```

Evaluate in the app has a Conditions box, and here it is conditions. An equation there gives a name a value while the expression is worked out:

```

type
evaluate(res, "v2", conditions=["vs = 10"])

```

```

returns
5

```

A voltage drop that the result does not hold is the difference of two node voltages, and it goes into an expression as one. The resistance of r , from its drop and its current, in TR:

```

type
evaluate(tr(rc), "(v1 - v2)/ir")

```

```

returns
1000

```

The tools' own answers work the same way. A Thévenin result also answers to the load's `ir1`, `v1` and `pr1`, in the variable `load`, and a matched load draws the maximum power:

```
type
evaluate(eq, "prl", conditions=["load = req"])
```

```
returns
3/2
```

Solve is `solve`. It substitutes the answers first, solves for whatever is left, and returns a list of solutions, each a dictionary:

```
type
solve(res, ["v2 = 6"], ["vs"])
```

```
returns
[{'vs': 12}]
```

`conditions` pins a symbol when it is an equation and filters the roots when it is a comparison, and `real_only=True` keeps real solutions only. A series circuit's resonance is where the source sees no reactance, with ω left as the symbol `w`:

```
type
res = ac("e,1,0,20:r,1,2,2:l,2,3,1'm:c,3,0,.4'u", "w")
solve(res, ["im(ze) = 0"], ["w"], conditions=["w > 0"], real_only=True)
```

```
returns
[{'w': 50000.00000000000}]
```

A value the **Define** card gives is a condition here, in `dc`, `ac`, `fd` or `tr` as much as in `evaluate`.

Expert Mode

The three boxes of **Expert Mode** are the three keywords `equations`, `unknowns` and `conditions`, on any of `dc`, `ac`, `fd` and `tr`:

```
type
design = dc("e1,1,0,12:r1,1,2,4'k:r2,2,0,r_b",
equations=["v_2 = 6"], unknowns=["r_b"])
design["r_b"]
```

```
returns
4000
```

Inside an equation, write an answer as the package stores it, with its underscore: `v_2`, `p_jd1`. The shorter spelling often works too, but a name such as `re` is SymPy's own function for the real part, and no rule of the package can tell the two apart. Symbols, Define and Expert Mode (page 13) has Expert Mode.

Cell magics

`%load_ext symbulator` adds a cell magic for each analysis: `%%dc`, `%%ac`, `%%fd` and `%%tr`. A cell that starts with one is a circuit, an element to a line, exactly as the input card takes it:

```
type
%load_ext symbulator
```

```
type
%%dc into=magic nodraw
e1,1,0,5
r1,1,2,1'k
r2,2,0,1'k
```

The cell draws the circuit and shows every answer, and `into` binds the result to a name:

```
type
magic["v2"]
```

```
returns
5/2
```

Options go on the magic's line: `omega=1000` for `%%ac`, `rms`, `variables=v_2,i_r1` to limit a transient, `into=` and `nodraw`.

Plotting

SymPy plots a transient as it is, with Matplotlib behind it:

```
type
import sympy as sp

sp.plot(tr(rc)["v2"], (t, 0, 0.005),
xlabel="t (s)", ylabel="v2 (V)");
```

`bode_samples` and `time_samples` return arrays for a frequency response or a time response, for Matplotlib to draw. The monograph's notebook below uses the first for the 1999 amplifier.

Notebooks to start from

Each is already executed, so you can read the answers before you run anything, and each opens in Google Colab. They are made from the app's own example files, and every answer in them has been compared with the app's.

Notebook	What is in it
Quick start	The package in a few cells, from a circuit to a plot.
Claude's sampler	The app's twelve showcase entries, one of each kind of analysis.
A Baker's Dozen	Thirteen solved examples that look like an afternoon's work by hand.
The Manual's circuits	Every circuit of this Manual, run, with its printed answer beneath.
The monograph's exemplars	The exemplar circuits of <i>The Internal Logic of Symbulator</i> .
Alexander & Sadiku and Nilsson & Riedel	The two textbook samplers.

The Course's own problems are there as well, one notebook to a lesson; the folder on GitHub has them all.

What stays in the app

The **Numerical Solver** is a page of the app and has no function in the package. SPICE has two, `to_spice` and `from_spice`, for the **SPICE Translator**'s two directions. The schematic is `draw`, and `to_svg` returns it as text.

WARNING

t is nonnegative

The symbol `t` is declared nonnegative, because a transient answer is only valid for $t \geq 0$. A limit taken through zero is the first place that surprises. Import it, `from symbulator import t`, `s`, and every answer's symbol is the one you hold.

Reference

PART 15

Every element, every answer name, what each tool returns, every setting, and what a saved file remembers. Lookup, not reading.

Elements

name first, then these, comma-separated. *Italic fields are optional.*

	Element	Fields	Notes
r	resistor	$n1, n2, \text{value}$	value may be $[r, r, \dots]$ in parallel
l	inductor	$n1, n2, \text{value}, i_o$	i_o read by FD and TR
c	capacitor	$n1, n2, \text{value}, v_o$	v_o read by FD and TR
e	voltage source	$n1, n2, \text{value}$	dependent if value names an answer
j	current source	$n1, n2, \text{value}$	current flows $n1 \rightarrow n2$ inside it
o	ideal op amp	$n+, n-, \text{nout}$	a nullor; reports i and p , no v
m	mutual inductance	$Lname1, Lname2, M \text{ or } \dots, k=0.5$	names two inductors, both in henries or both in $j\Omega$; no answers
s	short circuit	$n1, n2$	
t	transformer	$n1, n2, t1, t2$	or $[t1, b1], [tr, br], [t1, t2]$
z y h g a b	two-port	$n1, n2, [p11, p12, p21, p22]$	or bracketed pairs

Node 0 is ground. Names are letters, digits and underscores; the first character picks the type.

Answer names

Both spellings work everywhere: i_{r1} and i_r1 , v_2 and v_2 . Case is ignored.

		Where
v<node>	node voltage	all
i<el>	current, first node → second	all
v<el>	voltage drop, first minus second	all
p<el>	power consumed	all
ap<el>	average power	AC
s<el>	complex power, $P + jQ$	AC
z<el>	impedance seen	AC, FD
r<el>	resistance seen	DC
i<el><node>	terminal current	transformers, two-ports

SI prefixes

Apostrophe, then the letter. Exact, not decimal.

'P 10 ¹⁵	'T 10 ¹²	'G 10 ⁹	'M 10 ⁶	'k 'K 10 ³	
'm 10 ⁻³	'u 'μ 10 ⁻⁶	'n 10 ⁻⁹	'p 10 ⁻¹²	'f 10 ⁻¹⁵	'a 10 ⁻¹⁸

Case matters everywhere but kilo. 8000 and 8'k are exact; 8000. and 8E3 are approximate.

Analyses

DC	steady state	ignores initial conditions
AC	one ω , phasors	needs an ω
FD	s-domain	initial conditions included
TR	time domain	answers in t , valid for $t \geq 0$

Tools and mini-tools

Tool	Takes	Returns
Resistance / impedance	two nodes	req or zeq
Thévenin / Norton ...with the load tick	two nodes	vth, ino, req, pmax plus irl, vrl, prl
Two-port parameters	two nodes, a kind	four parameters of that kind
pf	a complex power, or an element's name	power factor with lead/lag: of the value as given, or of the power the element de- livers (a source) or consumes (an impedance)
gain	parameters and a load	G_v, G_i, G_p, Z_{in}
aa	a complex value	amplitude and an- gle
pz	a ratio of polynomials in s	poles and zeros, a common factor cancelled, a re- peated root with its multiplicity

Plots

	Analysis
Plot a function of time	TR
Bode plot of a variable	FD
Bode plot of transfer function $H(s)$	none needed
Plot a variable against another	DC

Settings

	Changes
Rounding — exact / exact+approx / approx to n / approx full	the display
Show units	the display
Use SI prefixes in answers	the display; forces off <i>exact</i>
Show AC answers as polar phasors	the display; AC only
Show equations	adds the Equations card
RMS phasors	the answer. AC only

Full detail in “Settings” (page 50).

What a saved file remembers

A `.cir` entry carries **32 fields**: the description, the analysis and its settings, Define, Evaluate and its conditions, Solve, Expert Mode’s three boxes, the plot’s five, the display settings, a note and an image. Saving and reloading returns you to the run, not just the circuit. Working with input files (in the Course) has the format.

Outside the tool

Fourier series and transforms · convolution · energy · nonlinear devices · transmission lines · noise and tolerance analysis.

Supported but not automated, because each is something you do *to* the circuit and then solve: superposition · source transformation · power-factor correction · magnitude and frequency scaling.

Who made this

Symbulator is the work of one person, shaped by the feedback of many. Created in 1999, MIT-licensed since 2026. See the Course for full credits.

The author

Symbulator and its documentation are the work of **Roberto Perez-Franco**, begun on **30 March 1999** as an engineering student at the Technological University of Panama. The first public release, version 0.92, went onto the Internet on 2 April 1999, and version 0.994 — the equivalent of version 1 — came out on 29 May 1999.

An early version won first place at the IEEE Student Paper Contest for Latin America in 2000; version 5 was his graduation thesis in 2001. Version 6 came in 2013 at MIT, versions 7 and 8 in 2023 in Melbourne, and this one in 2026.

Acknowledgements

Many people have made Symbulator better over the decades, through their suggestions and their corrections.

Versions 1 to 6 — José Vega (Panama), Tim Hutcheson (USA), Lars Frederiksen (Denmark), Joe Riel (USA), Arne Harstad (Norway), Erwin Baert (Belgium), Charles 'Chuck' Ware (USA), Doug Burkett (USA), Reinhard Willinski (Germany), Kamil Malinski (Austria), Jake Adams (USA), Daniele Martini (Italy), Rozgonyi Szabolcs (Hungary), Michael Rans (UK), Alex Astashyn (Russia), Al Charpentier (USA), Nevin McChesney (USA), Ivan Oro Yu (Panama), Pepe Iborra (Spain) and Dave Conklin (USA).

Versions 7 and 8 — Carlos Perez Ortega (Chile) and Qifan Wang (China).

Version 9 — Antony García (Panama).

My thanks to all of them. I truly am in their debt.

Claude

Porting Symbulator to Python and SymPy, building its interface and websites and rewriting its documentation would not have happened without Anthropic's AI assistant. Claude let me do in two weeks what I had not found time for in two decades.

The software it stands on

Python, with **SymPy** for the algebra and **NumPy** for the numerics. My thanks to their creators and maintainers, and it is why Symbulator is free and open-source from 2026.

Answers are checked during development against **ahkab**, an independent circuit simulator by Giuseppe Venturini.

Licence

MIT, since 2026. Port it anywhere you like, provided the copyright notice and the attribution travel with it.

Getting in touch

The author can be reached at help@symbulator.com

Index

- aa mini-tool, 25
- AC analysis, 25
- analyses (DC, AC, TR, FD), 5
- angular frequency (ω), 25
- answer names, 10
- average power, 26

- Bode plot of a transfer function $H(s)$, 32
- brackets, 9
- By-Hand Equations card, 37

- capacitor, 23
- case sensitivity, 11
- circuit description, 7
- Colab, 39
- complex power, 26
- Conditions (Evaluate), 14
- conductance, 16
- convolution, 5, 50
- coupling coefficient (k), 28
- curly-bracket shorthand, 32
- current source, 16

- DC sweep plot, 32
- Define card, 13
- dependent source, 16
- dot convention, 8, 28

- element types, 5, 7
- energy, 5, 50
- Equations card, 15
- equivalent resistance, 18
- Evaluate card, 14
- evaluate() and solve(), 43
- exact and approximate values, 9
- Expert Mode, 15

- filters, 33
- floating section (island), 35
- four-terminal form, 30
- Fourier series and transforms, 5, 50
- frequency domain, 31

- gain mini-tool, 35

- getting Symbulator (online, offline, local), 6
- ground node, 7

- impedance, 25
- inductor, 23
- initial conditions, 23
- initial- and final-value theorems, 32
- input files (.cir), 50
- interconnected two-ports, 36

- Jupyter notebook, 39

- Laplace transform, 32
- limit (Evaluate), 11, 32
- limiting the results (TR), 24
- load problems (irl, vrl, prl), 19

- maximum power transfer, 19
- mesh current direction, 37
- MIT licence, 52
- multiple solutions, 14
- mutual inductance, 28
- mutual inductance, checks on, 28

- names of elements and nodes, 8
- natural response, 24
- nodal and mesh analysis, 37
- noise and tolerance analysis, 6, 50
- nonlinear devices, 5, 50
- Norton equivalent, 19
- nullor, 21
- Numerical Solver, 37

- op amp model (finite gain), 22
- operational amplifier, 21

- parallel resistors, 17
- passive circuit, 18
- pf mini-tool, 26
- plots (Plotting Tools card), 24, 32
- polar phasors, 25
- poles and zeros, 6, 31
- power consumed and delivered, 11
- power factor, 26

power-factor correction, 6, 50
Python package (pip install
 symbolator), 6, 38, 39
pz mini-tool, 31

re and im (Evaluate), 11
reactances given in ohms, 25
reactive power, 26
real solutions only, 14
reference node, 7
reporting problems, 52
resistance seen by a source, 11
resonance, 33
Results card, 10
RMS, 27

s-domain, 31
schematic, 38
second-order circuits (damping), 24
separators (newline or colon), 9
SI prefixes, 9
sign conventions, 11

Solve card, 14
source transformation, 6, 50
SPICE Translator, 38
step response, 23
supernodes and supermeshes, 37
superposition, 6, 50
switched circuits (two runs), 24
symbolic circuit, 13

t2s and s2t, 32
three-phase circuits, 27
Thévenin equivalent, 18
transfer function, 31
transformer, 29
transient analysis (TR), 23
transmission lines, 5, 50
two-port, 35
two-port parameters (z , y , h , g , a , b), 34

underscore, 8

voltage drops and powers in TR and FD,
 42