Symbulator

A symbolic simulator of linear electric circuits

Symbulator 9 Manual

Python / SymPy

Roberto Perez-Franco

Contents

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 three mini-tools that work on values you type: *aa*, *pf* and *gain*.

Every value may be a symbol. That is the point of the thing: leave `r1` as `r1` and the answer comes back in terms of `r1`.

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	`symbulator.pythonanywhere.com` — nothing to install
In your browser, offline	`install.symbulator.com` — installs as an app
On your machine	`symbulator.com/9/local.zip` — a folder and a launcher

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

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.

One element per line. The first character of the name picks the type; the fields that follow are positional and comma-separated.

Circuit Description

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

That is a 12 V source across a 1 kΩ and a 2 kΩ in series. **Node `0` is ground**, always, and every circuit needs one.

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` *,initial current*
c	capacitor	`n1,n2,value` *,initial voltage*
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`
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.

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
```

`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 (page **??**) has it.

SI prefixes

An apostrophe and a letter multiply the value, exactly:

`4.7'k · 10'u · 560'm · 1'M`

Eleven prefixes, peta down to atto; the full table is in "SI prefixes" (page 34). 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:

> 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]`:

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

Neither port of **z1** touches ground here. 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.

Circuit Description

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

In DC that comes back as two node voltages and three answers per element:

voltage at node 2

$$v_2 = 8\,\mathrm{V}$$

current through r1

$$i_{r1} = \frac{1}{250}\,\mathrm{A}$$

power consumed by r1

$$p_{r1} = \frac{2}{125}\,\mathrm{W}$$

Exact, because `12`, `1'k` and `2'k` are exact. 1/250 A is 4 mA; see “Settings” (page 35) 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**	every element
`apr1`	average power	AC
`sr1`	complex power	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 (page **??**) 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. A source delivering power reports negative power consumed.

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**.
- **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 (page **??**) 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.

Circuit Description

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

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
```

Same circuit, and `v_{r2}` now reads 6 V. The description still says `ra`, so changing the value is one line and re-running, 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 — `ra = rb`, 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.

Solve

```
vr2 = 7
```

with `rb` as the unknown. The answer is the resistance that puts 7 V across **r2**.

> ⚠ 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 thing you want is not an answer the circuit produces. The classic case is a component value the circuit must have for some condition to hold: put the condition in *equations*, the component in *unknowns*, and the circuit is solved and the condition satisfied in one system rather than two passes.

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.

Circuit Description

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

voltage drop across r1

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

Two things are happening in three lines. The source drives 3 mA into node 1; and **r1**'s value is `1/g`, so the answer comes back in terms of a conductance. There is no conductance element — a conductance is a resistance written as its reciprocal, and the algebra takes care of 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	`e2,3,0,5*v2`
VCCS	`j2,3,0,g*v2`
CCVS	`e2,3,0,r*ir1`
CCCS	`j2,3,0,beta*irb`

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:

Circuit Description

```
e,1,0,9
rp,1,0,[1'k,2'k,2'k]
```

current through rp

$$i_{rp} = \frac{9}{500}\,\mathrm{A}$$

1 kΩ ∥ 2 kΩ ∥ 2 kΩ is 500 Ω, so 18 mA. 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.

Circuit Description

```
r1,1,2,100
r2,2,0,200
r3,2,0,300
```

Across **1** and **0**:

equivalent resistance

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

100 Ω in series with 200 □ 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.

Circuit Description

```
e,1,0,20
r1,1,2,50
r2,2,0,150
```

Across **2** and **0**:

Thévenin voltage

$$v_{th} = 15\,\mathrm{V}$$

equivalent resistance

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

Norton current

$$i_{no} = \frac{2}{5}\,\mathrm{A}$$

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

$$p_{max} = \frac{3}{2}\,\mathrm{W}$$

`pmax` is the maximum power transfer result — the power delivered when the load equals `R_eq`, which for this circuit is 15²/(4×37.5).

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`:

`irl`	current in the load
`vrl`	voltage drop in the load
`prl`	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 (page **??**) 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.

Circuit Description

```
e,1,0,2
r1,1,m,10'k
r2,m,o,47'k
o1,0,m,o
```

The inverting amplifier. Node **m** is the summing junction, **o** the output.

voltage drop across r2

$$v_{r2} = \frac{47}{5}\,\mathrm{V}$$

So the output sits at −9.4 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 — **0** 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:

Circuit Description

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

`ea` is a VCVS of gain 10^5 driving the output node from the differential input — here `0 - vm`, the non-inverting input minus the inverting one. Solve it and the answer differs from the ideal one 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`.

Circuit Description

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

voltage drop across c

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

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 and only the coefficient changes:

Circuit Description

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

voltage drop across c

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

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:

Circuit Description

```
r,1,0,1'k
c,1,0,1'u,10
```

voltage drop across c

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

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 (page **??**) 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 27) 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.

Circuit Description

```
e,1,0,10
r,1,2,50
l,2,0,0.05
```

At ω = 1000 rad/s the inductor is 50 Ω of reactance, so the impedance is 50 + 50j:

current through r

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

voltage drop across l

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

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∠-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:

`ap`	average power, in watts — the real power
`s`	complex power, **S** = P + jQ, in VA
`p`	the power answer, labelled by the convention in force
`z`	impedance seen

Circuit Description

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

At $\omega = 1000$ that is 30 + 40j — a 3-4-5 triangle — and the source reports complex power −0.6 − 0.8j VA. So $P = 0.6$ W, $Q = 0.8$ var, $|S| = 1.0$ VA and the power factor is 0.6, lagging.

An inductor reports no average power, only complex. That is not an omission: a pure reactance consumes none, and a zero printed every time would be noise.

The **pf** mini-tool takes a complex power or an impedance and returns the power factor with its lead/lag sense.

> ⚠ WARNING
>
> **RMS is the one setting that changes an answer**
>
> **RMS phasors**, in **Settings**, is not a display choice. Off means peak amplitude, the ÷2 convention. For the same phasor magnitudes, RMS reports twice the power that peak does — a 10 V source across 5 Ω gives 10 W with the tick off and 20 W with it on, and the answer's label changes with it. 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`, `120*exp(-2j*pi/3)`, `120*exp(2j*pi/3)` — 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.

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.

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
```

At $\omega = 1000$ the current in **r2** comes back as

current through r2

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

Write the second coil as `l2,0,2` instead — the same coil, the other way round — and the answer is exactly its negative, `-0.0621 + 0.0552j`. 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:

Circuit Description

```
e,1,0,120
t1,1,2,10,1
rl,2,0,5
```

Ten to one, so 12 V across the load and

current through rl

$$i_{rl} = \frac{12}{5}\,\mathrm{A}$$

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,[tl,bl],[tr,br],[10,1]
```

`[tl,bl]` 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 29).

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.

Circuit Description

```
e,1,0,vs
r,1,2,1'k
c,2,0,1'u
```

voltage drop across c

$$v_c = \frac{1000\,vs}{s + 1000}$$

Leave the source symbolic and the answer *is* the transfer function: divide by `vs` 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.

Poles and zeros are then yours to take: factor the denominator, or hand the expression to SymPy, which is what the answer already is.

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.

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
Bode plot of a variable	FD	an answer name, and a frequency range
Bode 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

Circuit Description

```
ra,1,2,10
rb,2,0,20
rc,2,3,30
```

A T network. Ports at **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. `z12 = z21` 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:

Circuit Description

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

The order is `[p11,p12,p21,p22]`. Leave the brackets off and the parameters become symbols — `zp11`, `zp12` 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 25). 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.

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. In a notebook there are `%%dc`, `%%ac`, `%%fd` and `%%tr` cell magics that take a circuit written one element per line, the way the app's input card does.

> ⚠ WARNING
>
> **`t` is nonnegative**
>
> The symbol `t` is declared nonnegative, because a transient answer is only valid for $t \geq 0$. That surprises the first person who tries to take a limit through zero. `from symbulator import t, s` gets you the same symbols the answers use.

Reference

PART 14

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,value`	value may be `[r,r,…]` in parallel
l	inductor	`n1,n2,value`,*i_0*	i_0 read by FD and TR
c	capacitor	`n1,n2,value`,*v_0*	v_0 read by FD and TR
e	voltage source	`n1,n2,value`	dependent if value names an answer
j	current source	`n1,n2,value`	current flows n1 → n2 inside it
o	ideal op amp	`n+,n−,nout`	a nullor; reports i and p, no v
m	mutual inductance	`Lname1,Lname2,M`	names two inductors; no answers
s	short circuit	`n1,n2`	
t	transformer	`n1,n2,t1,t2`	or `[tl,bl],[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: `ir1` and `i_r1`, `v2` 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 ≥ 0

Tools and mini-tools

Tool	Takes	Returns
Resistance / impedance	two nodes	`req` or `zeq`
Thévenin / Norton	two nodes	`vth`, `ino`, `req`, `pmax`
...with the load tick		plus `irl`, `vrl`, `prl`
Two-port parameters	two nodes, a kind	four parameters of that kind
pf	complex power or impedance	power factor, with lead/lag
gain	parameters and a load	G□, G□, G□, Z□□
aa	a complex value	amplitude and angle

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 *exact*
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 35).

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 (page **??**) 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 · pole-zero work · 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.

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 simu-

lator 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