

The tikzquads Package

An Extension to CircuiTikZ

Version 1.0

Alceu Frigeri*

January 2024

Abstract

This package defines a few extra shapes (single / dual port boxes) designed to be used together with the *CircuiTikZ* package.

Contents

1	Introduction	2
1.1	CircuiTikZ	2
2	Auxiliary Shapes and Basic Keys	2
2.1	Auxiliary shapes	2
2.2	General Keys	3
3	Z, Y, G, H Quadripoles	3
3.1	The Base Quadripole Shape	3
3.1.1	Base Keys	3
3.2	Quad	5
3.2.1	Quad Keys	5
3.2.2	Examples of <i>fit to use</i>	5
3.3	Quad Z	6
3.3.1	Quad Z keys	6
3.4	Quad Y	7
3.4.1	Quad Y keys	7
3.5	Quad G	7
3.5.1	Quad G keys	8
3.6	Quad H	8
3.6.1	Quad H keys	9
4	Thevenin, Norton single port boxes	9
4.1	The Base Black Box Shape	9
4.1.1	Base Keys	9
4.2	Black Box	10
4.2.1	Black Box keys	11
4.2.2	Examples of <i>fit to use</i>	11
4.3	Thevenin	11
4.3.1	Thevenin keys	12
4.4	Norton	12
4.4.1	Norton keys	12
5	Pseudo-Graph Shape	12
5.1	Pseudo-Graph Keys	13
6	Examples of use	13

*<https://github.com/alceu-frigeri/tikzquads>

1 Introduction

In standard text books, both for Circuits Theory and Electronics, quite frequently, in the process of modelling sub-circuits, one ends representing them either:

- as a single port *black box*, or
- as a dual port *black box*

This package defines a few, parameterized shapes for each case:

- for single port *black boxes*:
 - Black Box
 - Thevenin
 - Norton
- for dual port *black boxes*:
 - Quad
 - Quad Z
 - Quad Y
 - Quad G
 - Quad H

Lastly, this package also defines a *Pseudo-Graph load line* shape, for those moments where a true graph, *pgfplots*, isn't needed.

1.1 CircuiTikZ

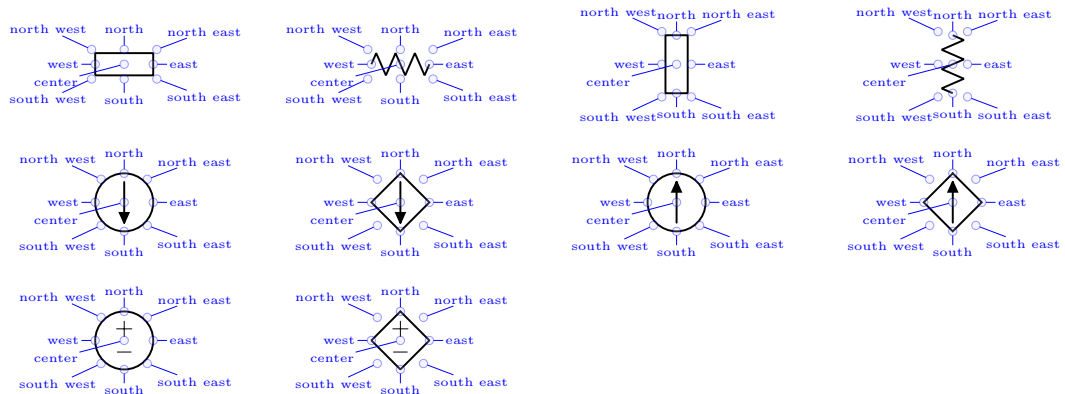
Unfortunately, some implementation details of these shapes don't follow the code structure adopted by *CircuiTikZ*, and some significant part of this package's code would have to be re-written if it were to be integrated directly in *CircuiTikZ*, and that's the main reason this is, for the time being, a separate package. After all, even though this doesn't follows *CircuiTikZ* code scheme, it does work nicely with it, as is.

2 Auxiliary Shapes and Basic Keys

Those shapes are not intended for end users.

2.1 Auxiliary shapes

A set of auxiliary shapes are defined, but not meant to be used otherwise, though their anchors might be relevant:



Note: The point being that, regardless of the sub-shape orientation, the intuitive geographical coordinates applies.

2.2 General Keys

These are the keys to fine tuning a shape:

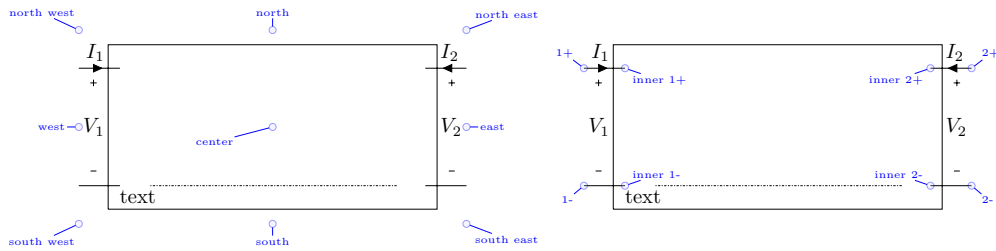
<code>outer sep</code>	Text outer separation, initial value: 1.5pt
<code>inner sep</code>	Text inner separation, initial value: 1pt
<code>thickness</code>	Components thickness (relative to the drawing thickness), initial value: 2
<code>tip len</code>	tip len (current source). initial value: 4pt
<code>tip type</code>	possible values: <i>triangle</i> and <i>bezier</i> . initial value: <i>triangle</i>
<code>minussign len</code>	Minus sign len (voltage source). initial value: $\backslash pgf@circ@Rlen/14$
<code>plussign len</code>	Plus sign len (voltage source). initial value: $1.1\backslash pgf@circ@Rlen/14$
<code>source radius</code>	The base radius. initial value: $0.3\backslash pgf@circ@Rlen$
<code>round sources</code>	Sources will be round ones
<code>control sources</code>	Sources will be control/diamond ones
<code>generic, european</code>	Impedances will be generic rectangles
<code>zigzag, american</code>	Impedances will be draw as zigzags

3 Z, Y, G, H Quadripoles

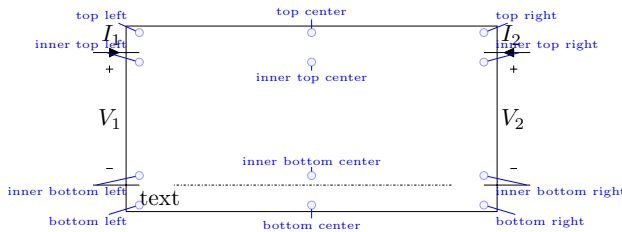
A set of configurable Quadripoles is defined, whereas quadripoles parameters (for instance Z_{11} , Z_{12} , Z_{21} and Z_{22}) are `<key-value>` parameters.

3.1 The Base Quadripole Shape

The base shape just draws a base box and sets some connection anchors: `1+`, `1-`, `inner 1+`, `inner 1-`, `2+`, `2-`, `inner 2+` and `inner 2-`, besides the geographic ones:



And also a set of (meant for) `text` anchors:



3.1.1 Base Keys

These applies to all Quad shapes:

<code>base width</code>	The 'box' width
<code>half base width</code>	Ditto, half width. Initial value: $2\backslash pgf@circ@Rlen$.
<code>base height</code>	The distance between <code>1+</code> and <code>1-</code> . The 'box' full height is equal to $2*(half\ base\ height + height\ ext + height\ ext+)$.
<code>half base height</code>	Ditto, half height. Initial value: $\backslash pgf@circ@Rlen/7$
<code>height ext</code>	Initial value: $2\backslash pgf@circ@Rlen/7$
<code>height ext+</code>	Initial value: 0
<code>inner ext</code>	distance between the 'box' and <code>inner1+/1-/2+/2-</code> . initial value: $\backslash pgf@circ@Rlen/7$
<code>outer ext</code>	distance between the 'box' and <code>1+/1-/2+/2-</code> . initial value: $5\backslash pgf@circ@Rlen/14$
<code>inner marks</code>	If set, the inner anchors will be marked.
<code>outer marks</code>	If set, the outer anchors will be marked.
<code>invert</code>	The shape will be inverted, more or less like 'x scale=-1'.
<code>alt, opt</code>	Case a Voltage source is zero, a series impedance will be draw vertically.
<code>outer x fit to</code>	For any Quad, this is the same as <code>outer x fit to*</code> .

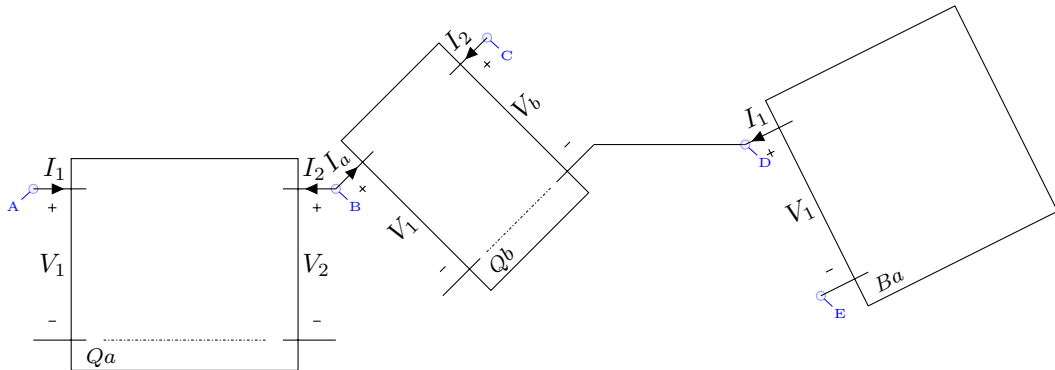
<code>outer x fit to*</code>	<code>outer x fit*={⟨CoordA⟩}{⟨CoordB⟩}</code> . The width will be set so that $\langle 1+ \rangle$ and $\langle 2+ \rangle$ (or $\langle 1- \rangle$ and $\langle 2- \rangle$, depending on the used anchor) will fit $\langle \text{CoordA} \rangle$ and $\langle \text{CoordB} \rangle$. This might result in a shape rotation.
<code>outer x fit to!</code>	<code>outer x fit!=\{⟨CoordA⟩\}\{⟨CoordB⟩\}</code> . The width will be set so that the distance between $\langle 1+ \rangle$ and $\langle 2+ \rangle$ (or $\langle 1- \rangle$ and $\langle 2- \rangle$, depending on the used anchor) will be the same as $\langle \text{CoordA} \rangle$ and $\langle \text{CoordB} \rangle$. This will never result in a shape rotation.
<code>inner x fit to</code>	For any Quad, this is the same as <code>inner x fit to*</code> .
<code>inner x fit to*</code>	<code>inner x fit*={⟨CoordA⟩}{⟨CoordB⟩}</code> . The width will be set so that $\langle \text{inner } 1+ \rangle$ and $\langle \text{inner } 2+ \rangle$ (or $\langle \text{inner } 1- \rangle$ and $\langle \text{inner } 2- \rangle$, depending on the used anchor) will fit $\langle \text{CoordA} \rangle$ and $\langle \text{CoordB} \rangle$. This might result in a shape rotation.
<code>inner x fit to!</code>	<code>inner x fit!=\{⟨CoordA⟩\}\{⟨CoordB⟩\}</code> . The width will be set so that the distance between $\langle \text{inner } 1+ \rangle$ and $\langle \text{inner } 2+ \rangle$ (or $\langle \text{inner } 1- \rangle$ and $\langle \text{inner } 2- \rangle$, depending on the used anchor) will be the same as $\langle \text{CoordA} \rangle$ and $\langle \text{CoordB} \rangle$. This will never result in a shape rotation.
<code>y fit to</code>	For any Quad, this is the same as <code>y fit to!</code> .
<code>y fit to*</code>	<code>y fit*={⟨CoordA⟩}{⟨CoordB⟩}</code> . The height will be set so that $1+$ and $1-$ will fit CoordA and CoordB . This might result in a shape rotation
<code>y fit to!</code>	<code>y fit!=\{⟨CoordA⟩\}\{⟨CoordB⟩\}</code> . The height will be set so that the distance between $\langle 1+ \rangle$ and $\langle 1- \rangle$ will be equal to the distance between $\langle \text{CoordA} \rangle$ and $\langle \text{CoordB} \rangle$. This will never result in a shape rotation.
<code>label top left</code>	It will place a label at the top left anchor
<code>label top center</code>	It will place a label at the top center anchor
<code>label top right</code>	It will place a label at the top right anchor
<code>label inner top left</code>	It will place a label at the inner top left anchor
<code>label inner top center</code>	It will place a label at the inner top center anchor
<code>label inner top right</code>	It will place a label at the inner top right anchor
<code>label bottom left</code>	It will place a label at the bottom left anchor
<code>label bottom center</code>	It will place a label at the bottom center anchor
<code>label bottom right</code>	It will place a label at the bottom right anchor
<code>label inner bottom left</code>	It will place a label at the inner bottom left anchor
<code>label inner bottom center</code>	It will place a label at the inner bottom center anchor
<code>label inner bottom right</code>	It will place a label at the inner bottom right anchor

A small example of the *fit* to keys:

```

1 \begin{tikzpicture}
2 \draw (0,0) \pincoord(A,blue,225) ++(4,0) \pincoord(B,blue,-45) ++(2,2) \pincoord(C) ;
3
4 \draw (A) node[Quad,anchor=1+,outer x fit to={A}{B}](Qa){\footnotesize$Qa$};
5 \draw (B) node[Quad,anchor=1+,outer x fit to={B}{C},I1=$I_a$,V2=$V_b$](Qb){\footnotesize$Qb$};
6
7 \draw (Qb.2-) -- ++(2,0) \pincoord(D) ++(1,-2) \pincoord(E);
8
9 \draw (D) node[Black Box,anchor=1+,y fit to={D}{E}](Ba){\footnotesize$Ba$};
10
11 \draw (Qa.1-) ++(0,-1);
12 \end{tikzpicture}

```



3.2 Quad

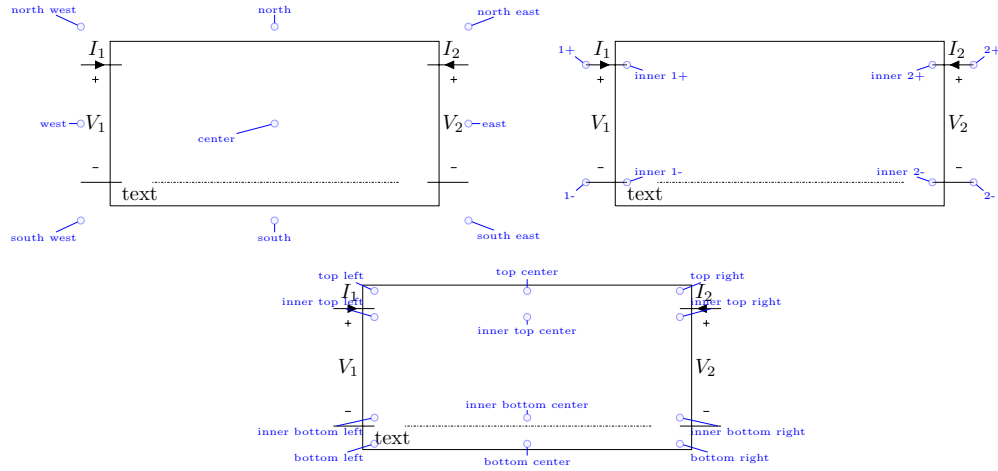
```

1 % Node use
2 node[Quad]{}
3
4 % To path use
5 (A) to[ToQuad] (B)

```

This is just the base shape, to be used in cases whereas one just want to emphasises part of a circuit (using, for instance, the *inner x fit to* key, or just mark a two port black box.

Note: There is also a *ToQuad* to be used in a *to[]* path, in which case the key *outer x fit to* style will be triggered with the starting and ending points of the *to[]* path.



3.2.1 Quad Keys

<i>name</i>	$\langle \text{node-name} \rangle$, when using a <i>to[]</i> path.
<i>I1</i>	Initial value: $\$I_1\$$
<i>I2</i>	Initial value: $\$I_2\$$
<i>V1</i>	Initial value: $\$V_1\$$
<i>V2</i>	Initial value: $\$V_2\$$

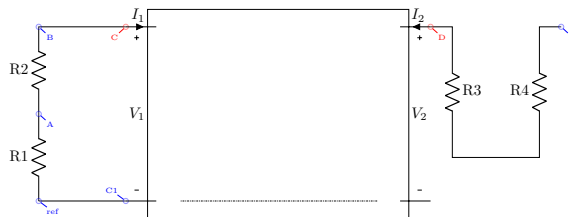
3.2.2 Examples of *fit to* use

Squeezing a Quadripole between two parts of a circuit (nodes C and D):

```

1 \begin{center}
2 \resizebox{0.5\textwidth}{!}{
3 \begin{tikzpicture}
4 \draw (0,0) \pincoord(ref) to[R=R1] ++(0,2) \pincoord(A) to[R=R2] ++(0,2) \pincoord(B)
5 -- ++(2,0) \pincoord(C,red,225) (C |- ref) \pincoord(C1,blue,135) -- (ref);
6 \draw (C) ++(7,0) \pincoord(D,red) -- ++(0.5,0) to[R=R3] ++(0,-3) -- ++(2,0) to[R=R4] ++(0,3) -- ++(0.5,0)
7 \pincoord(E);
8 \draw (C) node[Quad,anchor=1+,y fit to={C}{C1},outer x fit to={C}{D}] {};
9 \end{tikzpicture}
10 \end{center}

```

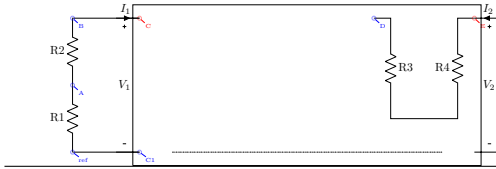


Fitting some circuit inside the Quadripole (nodes C and E):

```

1 \resizebox{0.4\textwidth}{!}{
2 \begin{tikzpicture}
3   \draw (0,0) \pincoord(ref) to[R=R1] ++(0,2) \pincoord(A) to[R=R2] ++(0,2) \pincoord(B)
4   -- ++(2,0) \pincoord(C,red) (C |- ref) \pincoord(C1) -- (ref);
5   \draw (C) ++(7,0) \pincoord(D) -- ++(0.5,0) to[R=R3] ++(0,-3) -- ++(2,0) to[R=R4] ++(0,3) -- ++(0.5,0)
6   \pincoord(E,red);
7   \draw (C) node[Quad,anchor=inner 1+,y fit to={C}{C1},inner x fit to={C}{E}]{};
8 \end{tikzpicture}}

```



3.3 Quad Z

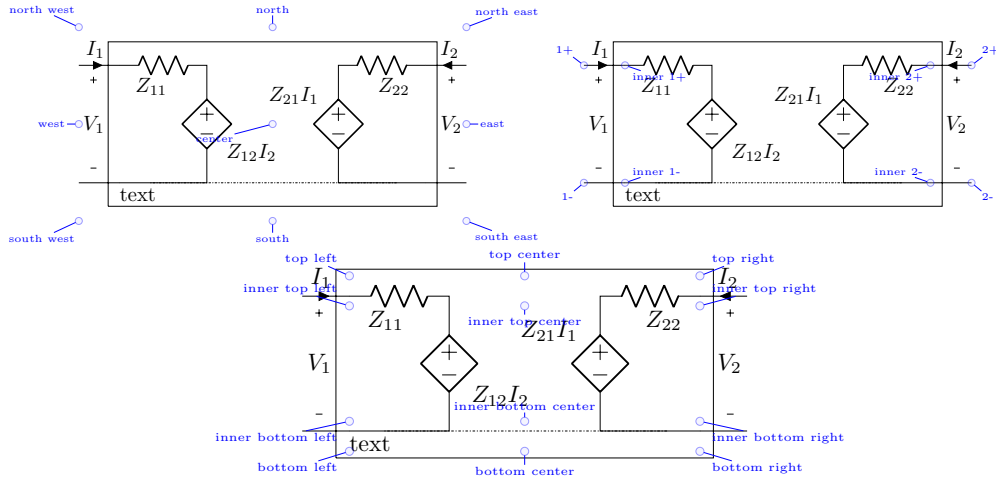
```

1 % Node use
2 node[Quad Z]{}
3
4 % To path use
5 (A) to[ToQuad Z] (B)

```

This shape, besides the base anchors (see 3) it has 4 internal nodes: `<node>-Z11`, `<node>-Z12`, `<node>-Z21` and `<node>-Z22` and each of those sub-nodes has geographic anchors as defined at 2.1.

Note: There is also a `ToQuad Z` to be used in a `to[]` path, in which case the key `outer x fit to` style will be triggered with the starting and ending points of the `to[]` path.



3.3.1 Quad Z keys

name	<code><node-name></code> , when using a <code>to[]</code> path.
<code>I1</code>	Initial value: I_1
<code>I2</code>	Initial value: I_2
<code>V1</code>	Initial value: V_1
<code>V2</code>	Initial value: V_2
<code>Z11</code>	Initial value: Z_{11}
<code>Z12</code>	Initial value: Z_{12}
<code>Z21</code>	Initial value: Z_{21}
<code>Z22</code>	Initial value: Z_{22}
<code>Z11 label pos</code>	changes the label position. Defaults to: {south west}{top left}
<code>Z12 label pos</code>	changes the label position. Defaults to: {south east}{top left}
<code>Z21 label pos</code>	changes the label position. Defaults to: {north west}{bottom right}
<code>Z22 label pos</code>	changes the label position. Defaults to: {south east}{top right}

Note: The label pos keys expects two anchor names (... label pos= {<anchor A>}{<anchor B>}). The first anchors refers the sub-shape node and the second anchor is the text one.

3.4 Quad Y

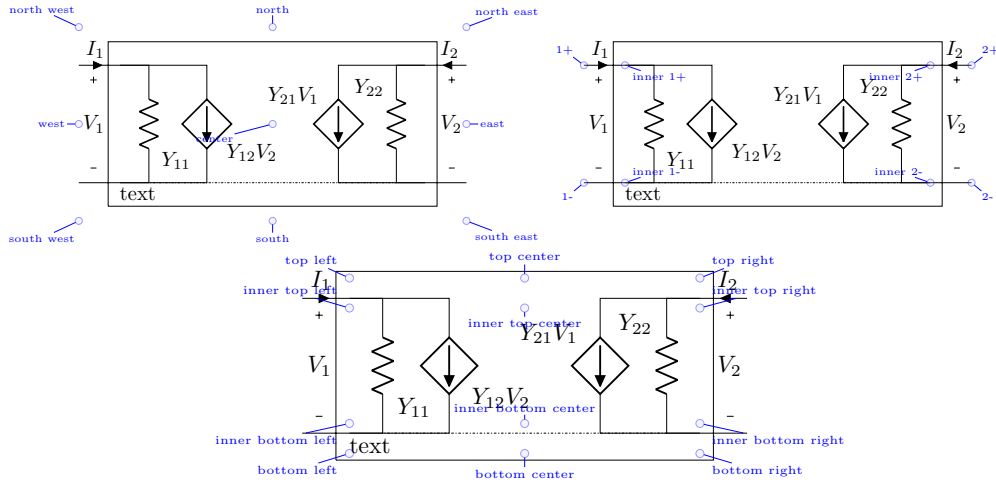
```

1 % Node use
2 node[Quad Y]{}
3
4 % To path use
5 (A) to[ToQuad Y] (B)

```

This shape, besides the base anchors (see 3) it has 4 internal nodes: `<node>-Y11`, `<node>-Y12`, `<node>-Y21` and `<node>-Y22` and each of those sub-nodes has geographic anchors as defined at 2.1.

Note: There is also a `ToQuad Y` to be used in a `to[]` path, in which case the key `outer x fit to` style will be triggered with the starting and ending points of the `to[]` path.



3.4.1 Quad Y keys

name	<code><node-name></code> , when using a <code>to[]</code> path.
<code>I1</code>	Initial value: $\$I_1\$$
<code>I2</code>	Initial value: $\$I_2\$$
<code>V1</code>	Initial value: $\$V_1\$$
<code>V2</code>	Initial value: $\$V_2\$$
<code>Y11</code>	Initial value: $\$Y_{11}\$$
<code>Y12</code>	Initial value: $\$Y_{12}\$$
<code>Y21</code>	Initial value: $\$Y_{21}\$$
<code>Y22</code>	Initial value: $\$Y_{22}\$$
<code>Y11 label pos</code>	changes the label position. Defaults to: <code>{south west}{top left}</code>
<code>Y12 label pos</code>	changes the label position. Defaults to: <code>{south east}{top left}</code>
<code>Y21 label pos</code>	changes the label position. Defaults to: <code>{north west}{bottom right}</code>
<code>Y22 label pos</code>	changes the label position. Defaults to: <code>{north west}{bottom right}</code>

Note: The label pos keys expects two anchor names (... label pos= `{<anchor A>}{<anchor B>}`). The first anchors refers the sub-shape node and the second anchor is the text one.

3.5 Quad G

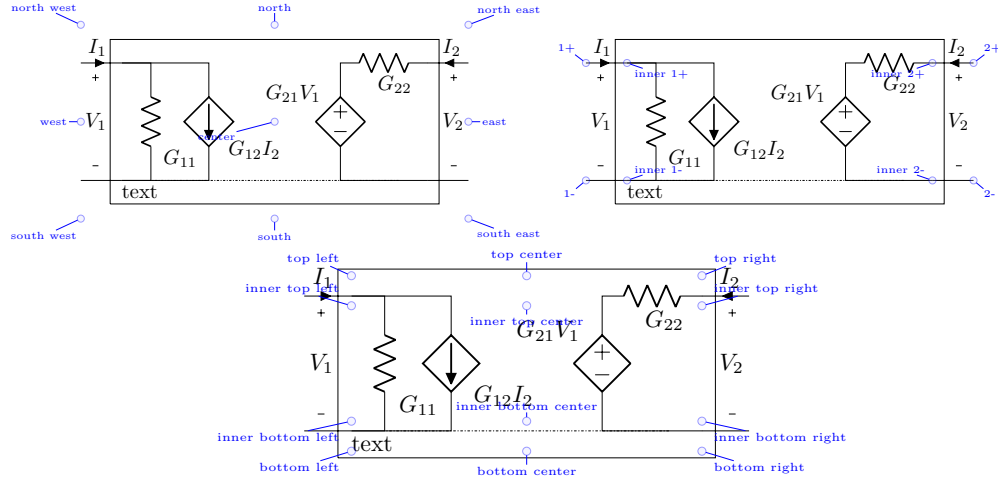
```

1 % Node use
2 node[Quad G]{}
3
4 % To path use
5 (A) to[ToQuad G] (B)

```

This shape, besides the base anchors (see 3) it has 4 internal nodes: `<node>-G11`, `<node>-G12`, `<node>-G21` and `<node>-G22` and each of those sub-nodes has geographic anchors as defined at 2.1.

Note: There is also a `ToQuad G` to be used in a `to[]` path, in which case the key `outer x fit to` style will be triggered with the starting and ending points of the `to[]` path.



3.5.1 Quad G keys

<i>name</i>	$\langle \text{node-name} \rangle$, when using a <code>to[]</code> path.
<i>I1</i>	Initial value: $\$I_1\$$
<i>I2</i>	Initial value: $\$I_2\$$
<i>V1</i>	Initial value: $\$V_1\$$
<i>V2</i>	Initial value: $\$V_2\$$
<i>G11</i>	Initial value: $\$G_11\$$
<i>G12</i>	Initial value: $\$G_12\$$
<i>G21</i>	Initial value: $\$G_21\$$
<i>G22</i>	Initial value: $\$G_22\$$
<i>G11 label pos</i>	changes the label position. Defaults to: {south west}{top left}
<i>G12 label pos</i>	changes the label position. Defaults to: {south east}{top left}
<i>G21 label pos</i>	changes the label position. Defaults to: {north west}{bottom right}
<i>G22 label pos</i>	changes the label position. Defaults to: {south east}{top right}

Note: The label pos keys expects two anchor names (\dots label pos= { $\langle \text{anchor A} \rangle$ } { $\langle \text{anchor B} \rangle$ }). The first anchors refers the sub-shape node and the second anchor is the text one.

3.6 Quad H

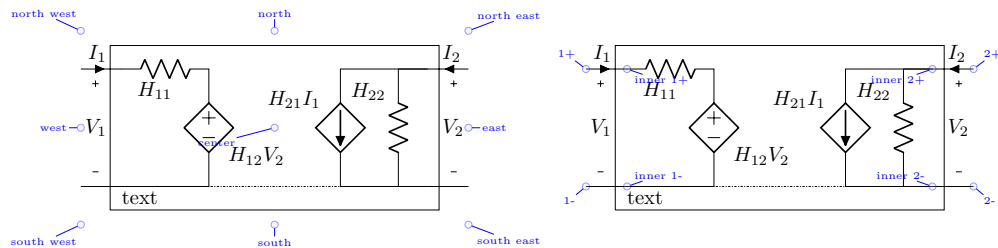
```

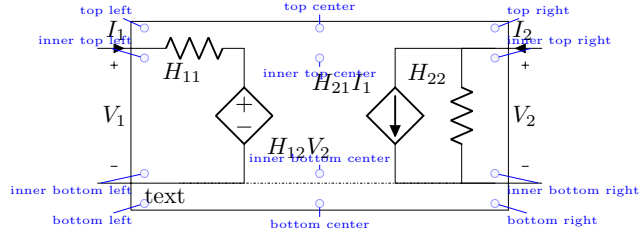
1 % Node use
2 node[Quad H] {}
3
4 % To path use
5 (A) to[ToQuad H] (B)

```

This shape, besides the base anchors (see 3) it has 4 internal nodes: $\langle \text{node} \rangle\text{-H11}$, $\langle \text{node} \rangle\text{-H12}$, $\langle \text{node} \rangle\text{-H21}$ and $\langle \text{node} \rangle\text{-H22}$ and each of those sub-nodes has geographic anchors as defined at 2.1.

Note: There is also a *ToQuad H* to be used in a `to[]` path, in which case the key *outer x fit to* style will be triggered with the starting and ending points of the `to[]` path.





3.6.1 Quad H keys

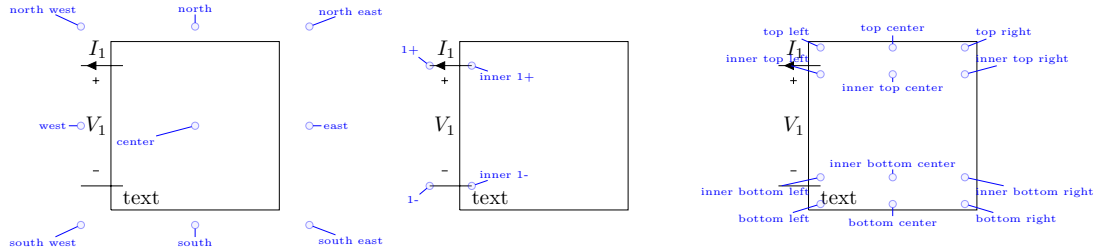
<i>name</i>	$\langle \text{node-name} \rangle$, when using a <code>to[]</code> path.
<i>I1</i>	Initial value: $\$I_1\$$
<i>I2</i>	Initial value: $\$I_2\$$
<i>V1</i>	Initial value: $\$V_1\$$
<i>V2</i>	Initial value: $\$V_2\$$
<i>H11</i>	Initial value: $\$H_{\{11\}}\$$
<i>H12</i>	Initial value: $\$H_{\{12\}}\$$
<i>H21</i>	Initial value: $\$H_{\{21\}}\$$
<i>H22</i>	Initial value: $\$H_{\{22\}}\$$
<i>H11 label pos</i>	changes the label position. Defaults to: $\{\text{south west}\}\{\text{top left}\}$
<i>H12 label pos</i>	changes the label position. Defaults to: $\{\text{south east}\}\{\text{top left}\}$
<i>H21 label pos</i>	changes the label position. Defaults to: $\{\text{north west}\}\{\text{bottom right}\}$
<i>H22 label pos</i>	changes the label position. Defaults to: $\{\text{north west}\}\{\text{bottom right}\}$

Note: The label pos keys expects two anchor names (\dots label pos = $\{\langle \text{anchor A} \rangle\} \{\langle \text{anchor B} \rangle\}$). The first anchors refers the sub-shape node and the second anchor is the text one.

4 Thevenin, Norton single port boxes

4.1 The Base Black Box Shape

The base shape just draws a base box and sets some connection anchors: $1+$, $1-$, *inner* $1+$, *inner* $1-$, besides the geographic and text ones:



4.1.1 Base Keys

These applies to all *Black Box* shapes:

<i>base width</i>	The 'box' width
<i>half base width</i>	Ditto, half width. Initial value: $2\backslash\text{pgf@circ@Rlen}$.
<i>base height</i>	The distance between $1+$ and $1-$. The 'box' full height is equal to $2*(\text{half base height} + \text{height ext} + \text{height ext+})$.
<i>half base height</i>	Ditto, half height. Initial value: $\backslash\text{pgf@circ@Rlen}/7$
<i>height ext</i>	Initial value: $2\backslash\text{pgf@circ@Rlen}/7$
<i>height ext+</i>	Initial value: 0
<i>inner ext</i>	distance between the 'box' and $\text{inner}1+/1-/2+/2-$. initial value: $\backslash\text{pgf@circ@Rlen}/7$
<i>outer ext</i>	distance between the 'box' and $1+/1-/2+/2-$. initial value: $5\backslash\text{pgf@circ@Rlen}/14$
<i>inner marks</i>	If set, the inner anchors will be marked.
<i>outer marks</i>	If set, the outer anchors will be marked.

<code>invert</code>	The shape will be inverted, more or less like 'x scale=-1'.
<code>alt, opt</code>	Case a Voltage source is zero, a series impedance will be draw vertically.
<code>outer x fit to</code>	For any Black Box, this is the same as <code>outer x fit to!</code> .
<code>outer x fit to*</code>	<code>outer x fit*={⟨CoordA⟩}{⟨CoordB⟩}</code> . The width will be set so that ⟨1+⟩ and ⟨2+⟩ (or ⟨1-⟩ and ⟨2-⟩, depending on the used anchor) will fit ⟨CoordA⟩ and ⟨CoordB⟩. This might result in a shape rotation.
<code>outer x fit to!</code>	<code>outer x fit!={⟨CoordA⟩}{⟨CoordB⟩}</code> . The width will be set so that the distance between ⟨1+⟩ and ⟨2+⟩ (or ⟨1-⟩ and ⟨2-⟩, depending on the used anchor) will be the same as ⟨CoordA⟩ and ⟨CoordB⟩. This will never result in a shape rotation.
<code>inner x fit to</code>	For any Black Box, this is the same as <code>inner x fit to!</code> .
<code>inner x fit to*</code>	<code>inner x fit*={⟨CoordA⟩}{⟨CoordB⟩}</code> . The width will be set so that ⟨inner 1+⟩ and ⟨inner 2+⟩ (or ⟨inner 1-⟩ and ⟨inner 2-⟩, depending on the used anchor) will fit ⟨CoordA⟩ and ⟨CoordB⟩. This might result in a shape rotation.
<code>inner x fit to!</code>	<code>inner x fit!={⟨CoordA⟩}{⟨CoordB⟩}</code> . The width will be set so that the distance between ⟨inner 1+⟩ and ⟨inner 2+⟩ (or ⟨inner 1-⟩ and ⟨inner 2-⟩, depending on the used anchor) will be the same as ⟨CoordA⟩ and ⟨CoordB⟩. This will never result in a shape rotation.
<code>y fit to</code>	For any Black Box, this is the same as <code>y fit to*</code> .
<code>y fit to*</code>	<code>y fit*={⟨CoordA⟩}{⟨CoordB⟩}</code> . The height will be set so that 1+ and 1- will fit CoordA and CoordB. This might result in a shape rotation
<code>y fit to!</code>	<code>y fit!={⟨CoordA⟩}{⟨CoordB⟩}</code> . The height will be set so that the distance between ⟨1+⟩ and ⟨1-⟩ will be equal to the distance between ⟨CoordA⟩ and ⟨CoordB⟩. This will never result in a shape rotation.
<code>label top left</code>	It will place a label at the top left anchor
<code>label top center</code>	It will place a label at the top center anchor
<code>label top right</code>	It will place a label at the top right anchor
<code>label inner top left</code>	It will place a label at the inner top left anchor
<code>label inner top center</code>	It will place a label at the inner top center anchor
<code>label inner top right</code>	It will place a label at the inner top right anchor
<code>label bottom left</code>	It will place a label at the bottom left anchor
<code>label bottom center</code>	It will place a label at the bottom center anchor
<code>label bottom right</code>	It will place a label at the bottom right anchor
<code>label inner bottom left</code>	It will place a label at the inner bottom left anchor
<code>label inner bottom center</code>	It will place a label at the inner bottom center anchor
<code>label inner bottom right</code>	It will place a label at the inner bottom right anchor

4.2 Black Box

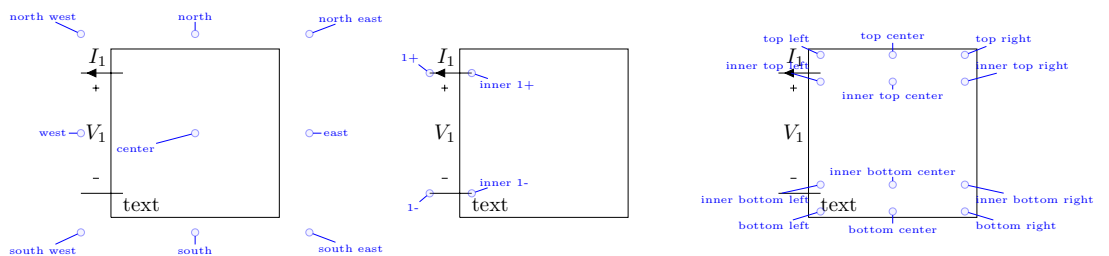
```

1 % Node use
2 node[Black Box]{}
3
4 % To path use
5 (A) to[ToBlack Box] (B)

```

This is just the base shape, to be used in cases whereas one just want to emphasises part of a circuit (using, for instance, the `inner x fit to` key, or just mark a single port black box.

Note: There is also a `ToBlack Box` to be used in a `to[]` path, in which case the key `y fit to` style will be triggered with the starting and ending points of the `to[]` path.



4.2.1 Black Box keys

<i>name</i>	$\langle \text{node-name} \rangle$, when using a <code>to[]</code> path.
<i>I1</i>	Initial value: $\$I_1\$$
<i>V1</i>	Initial value: $\$V_1\$$

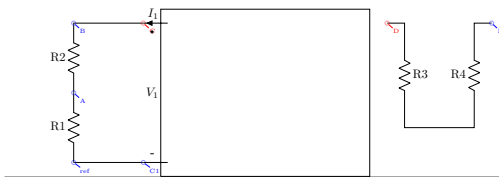
4.2.2 Examples of *fit* to use

Squeezing a Black Box between two parts of a circuit (nodes C and D):

```

1 \resizebox{0.4\textwidth}{!}{
2 \begin{tikzpicture}
3   \draw (0,0) \pincoord(ref) to[R=R1] ++(0,2) \pincoord(A) to[R=R2] ++(0,2) \pincoord(B)
4   -- ++(2,0) \pincoord(C,red) (C |- ref) \pincoord(C1) -- (ref);
5   \draw (C) ++(7,0) \pincoord(D,red) -- ++(0.5,0) to[R=R3] ++(0,-3) -- ++(2,0) to[R=R4] ++(0,3) -- ++(0.5,0)
6   \pincoord(E);
7   \draw (C) node[Black Box,anchor=1+,y fit to={C}{C1},outer x fit to={C}{D}]{ };
8 \end{tikzpicture}

```



Fitting some circuit inside the Black Box (nodes C and E):

```

1 \resizebox{0.4\textwidth}{!}{
2 \begin{tikzpicture}
3   \draw (0,0) \pincoord(ref) to[R=R1] ++(0,2) \pincoord(A) to[R=R2] ++(0,2) \pincoord(B)
4   -- ++(2,0) \pincoord(C,red) (C |- ref) \pincoord(C1) -- (ref);
5   \draw (C) ++(7,0) \pincoord(D,red) -- ++(0.5,0) to[R=R3] ++(0,-3) -- ++(2,0) to[R=R4] ++(0,3) -- ++(0.5,0)
6   \pincoord(E);
7   \draw (C) node[Black Box,anchor=inner 1+,y fit to={C}{C1},inner x fit to={C}{E}]{ };
8 \end{tikzpicture}

```



4.3 Thevenin

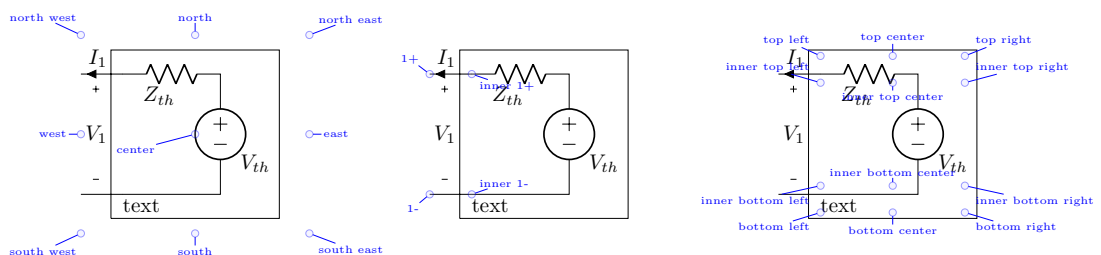
```

1 % Node use
2 node[Thevenin]{}
3
4 % To path use
5 (A) to[ToThevenin] (B)

```

This is the classical Thevenin circuit. Besides the base anchors (see 4.1) it has 2 internal nodes: `<node>-Zth` and `<node>-Vth` and each of those sub-nodes has geographic anchors as defined at 2.1.

Note: There is also a `ToThevenin` to be used in a `to[]` path, in which case the key `y fit to` style will be triggered with the starting and ending points of the `to[]` path.



4.3.1 Thevenin keys

<i>name</i>	$\langle \text{node-name} \rangle$, when using a <code>to[]</code> path.
<i>I1</i>	Initial value: $\$I_1\$$
<i>V1</i>	Initial value: $\$V_1\$$
<i>Zth</i>	Initial value: $\$Z_{\text{th}}\$$
<i>Vth</i>	Initial value: $\$V_{\text{th}}\$$
<i>Zth label pos</i>	changes the label position. Defaults to: <code>{south west}{top left}</code>
<i>Vth label pos</i>	changes the label position. Defaults to: <code>{south east}{top left}</code>

Note: The label pos keys expects two anchor names (... label pos= $\{\langle \text{anchor A} \rangle\} \{\langle \text{anchor B} \rangle\}$). The first anchors refers the sub-shape node and the second anchor is the text one.

4.4 Norton

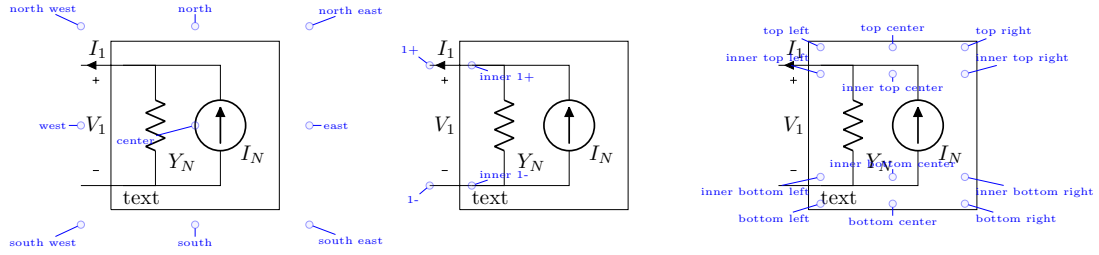
```

1 % Node use
2 node[Norton]{}
3
4 % To path use
5 (A) to[ToNorton] (B)

```

This is the classical Norton circuit. Besides the base anchors (see 4.1) it has 2 internal nodes: $\langle \text{node} \rangle\text{-}Y_n$ and $\langle \text{node} \rangle\text{-}I_n$ and each of those sub-nodes has geographic anchors as defined at 2.1.

Note: There is also a `ToNorton` to be used in a `to[]` path, in which case the key `y fit to` style will be triggered with the starting and ending points of the `to[]` path.



4.4.1 Norton keys

<i>name</i>	$\langle \text{node-name} \rangle$, when using a <code>to[]</code> path.
<i>I1</i>	Initial value: $\$I_1\$$
<i>V1</i>	Initial value: $\$V_1\$$
<i>Yn</i>	Initial value: $\$Y_{\text{N}}\$$
<i>In</i>	Initial value: $\$I_{\text{N}}\$$
<i>Yn label pos</i>	changes the label position. Defaults to: <code>{south west}{top left}</code>
<i>In label pos</i>	changes the label position. Defaults to: <code>{south east}{top left}</code>

Note: The label pos keys expects two anchor names (... label pos= $\{\langle \text{anchor A} \rangle\} \{\langle \text{anchor B} \rangle\}$). The first anchors refers the sub-shape node and the second anchor is the text one.

5 Pseudo-Graph Shape

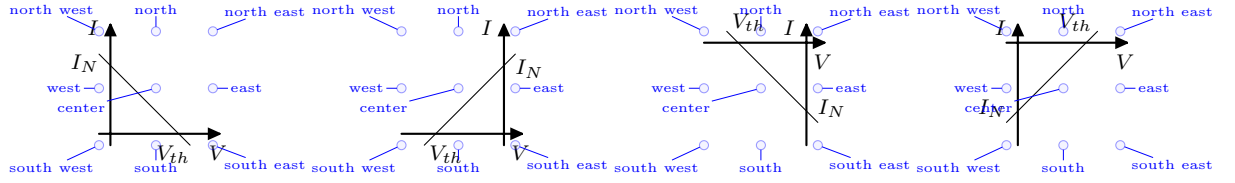
```

1 % Node use
2 node[PG load line]{}
3
4 node[PG linear load line]

```

Sometimes when representing a single port sub-circuit, one might use a X-Y graph, for which `gnuplot` and `pgfplots` are excellent choices, but a bit overkill if all you want is a crude representation of a linear load line.

This shape is just that, a X-Y graph mockup, that nicely fits inside a black box, and nothing else.



5.1 Pseudo-Graph Keys

These are the keys to fine tuning a shape:

<code>x axis</code>	X axis <i>name</i> . Initial value: V
<code>x val</code>	X axis <i>val</i> at the crossing point. Initial value: V_{th}
<code>y axis</code>	Y axis <i>name</i> . Initial value: I
<code>y val</code>	Y axis <i>val</i> at the crossing point. Initial value: I_N
<code>first quadrant</code>	First quadrant mock up. (which is also the default).
<code>second quadrant</code>	Second quadrant mock up.
<code>third quadrant</code>	Third quadrant mock up.
<code>fourth quadrant</code>	Fourth quadrant mock up.
<code>base width</code>	The <i>graph</i> width
<code>half base width</code>	Ditto, half width. Initial value: $0.5\pgf@circ@Rlen$.
<code>base height</code>	The <i>graph</i> height
<code>half base height</code>	Ditto, half height. Initial value: $0.5\pgf@circ@Rlen$.

Note: Besides these, one can also use the keys presented at 2.2.

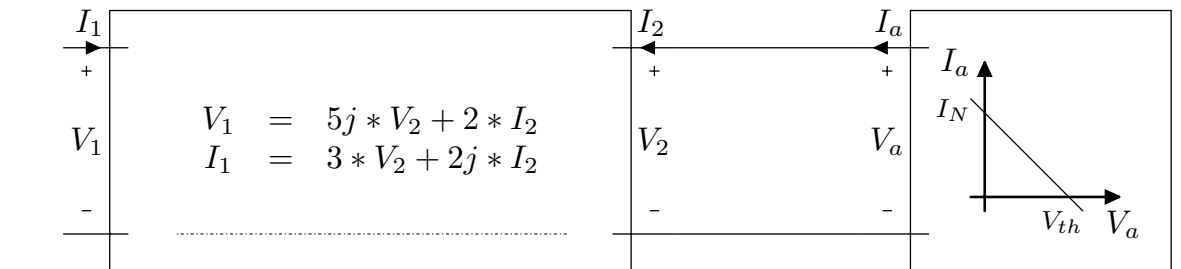
6 Examples of use

First of, a simple case of combining a generic Quad with equations and a generic Black Box with a Pseudo-Graph:

```

1 \resizebox{\textwidth}{!}{
2 \begin{tikzpicture}
3   \draw (0,0) \ncoord(ref) node[Quad,anchor=1+](Q1){
4     (Q1.2+) -- ++(1,0) \ncoord(X) -- ++(1,0) node[Black Box,anchor=1+,V1=$V_a$,I1=$I_a$](B1){}
5     (Q1.2-) -- (B1.1-)
6     (B1.center) node[PG linear load line,x axis=$V_a$,y axis=$I_a$]{}
7     (Q1.center) node{$\begin{matrix} V_1 & = & 5j * V_2 + 2 * I_2 \\ I_1 & = & 3 * V_2 + 2j * I_2 \end{matrix}$}
8     \end{matrix} $%
9   } ;
10 \end{tikzpicture}
11 }

```

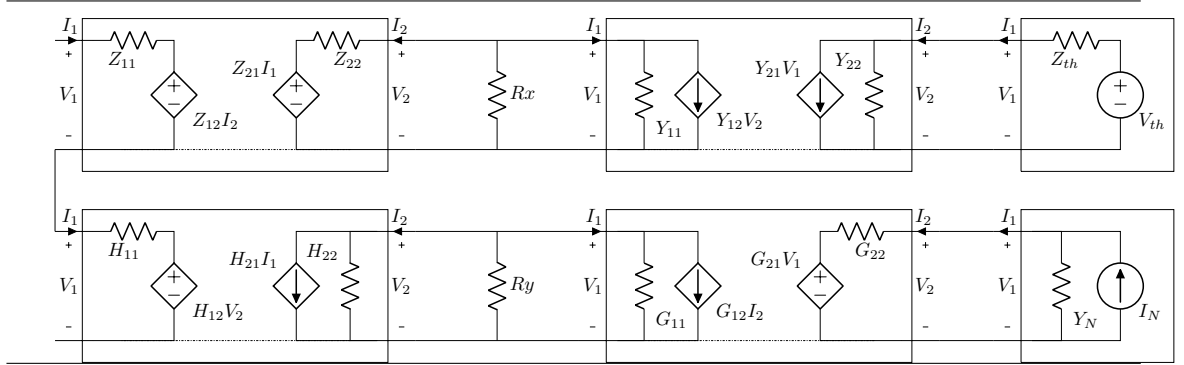


All default Quadripoles and Thevenin/Norton.

```

1 \resizebox{\textwidth}{!}{
2 \begin{tikzpicture}
3   \draw (0,0) \ncoord(ref) node[Quad Z,anchor=1+] (Qz1){}
4     (Qz1.2+) -- ++(1.5,0) \ncoord(X) -- ++(1.5,0) node[Quad Y,anchor=1+] (Qy1){}
5     (Qy1.2+) -- ++(1,0) node[Thevenin,anchor=1+] (th1){}
6     (Qz1.1-) -- ++(0,-1.5) node[Quad H,anchor=1+] (Qh1){}
7     (Qh1.2+) -- ++(1.5,0) \ncoord(Y) -- ++(1.5,0) node[Quad G,anchor=1+] (Qg1){}
8     (Qg1.2+) -- ++(1,0) node[Norton,anchor=1+] (nr1){}
9     (Qz1.2-) -- (Qy1.1-) (Qy1.2-) -- (th1.1-)
10    (Qh1.2-) -- (Qg1.1-) (Qg1.2-) -- (nr1.1-)
11  ;
12  \draw (X) to[R=$R_x$] (X |- Qz1.2-)
13        (Y) to[R=$R_y$] (Y |- Qh1.2-)
14  ;
15 \end{tikzpicture}
16 }

```

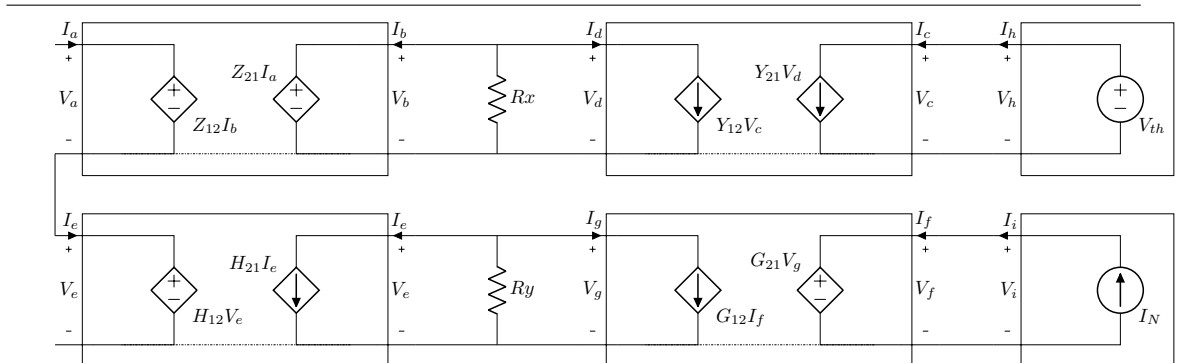


The same demo but with all parameter 11 and 22 zeroed, and changing the “control sources”

```

1 \resizebox{\textwidth}{!}{
2 \begin{tikzpicture}
3   \draw (0,0) \ncoord(ref) node[Quad Z,anchor=1+,Z11=0,Z22=0,I1=$I_a$,V1=$V_a$,I2=$I_b$,V2=$V_b$] (Qz1){}
4     (Qz1.2+) -- ++(1.5,0) \ncoord(X) -- ++(1.5,0) node[Quad Y,anchor=1+,Y11=0,Y22=0,I1=$I_d$,V1=$V_d$,I2=$I_c$,V
5     2=$V_c$] (Qy1){}
6     (Qy1.2+) -- ++(1,0) node[Thevenin,anchor=1+,Zth=0,I1=$I_h$,V1=$V_h$] (th1){}
7     (Qz1.1-) -- ++(0,-1.5) node[Quad H,anchor=1+,H11=0,H22=0,I1=$I_e$,V1=$V_e$,I2=$I_f$,V2=$V_e$] (Qh1){}
8     (Qh1.2+) -- ++(1.5,0) \ncoord(Y) -- ++(1.5,0) node[Quad G,anchor=1+,G11=0,G22=0,I1=$I_g$,V1=$V_g$,I2=$I_f$,V
9     2=$V_f$] (Qg1){}
10    (Qg1.2+) -- ++(1,0) node[Norton,anchor=1+,Yn=0,I1=$I_i$,V1=$V_i$] (nr1){}
11  ;
12  \draw (X) to[R=$R_x$] (X |- Qz1.2-)
13        (Y) to[R=$R_y$] (Y |- Qh1.2-)
14  ;
15 \end{tikzpicture}
16 }

```

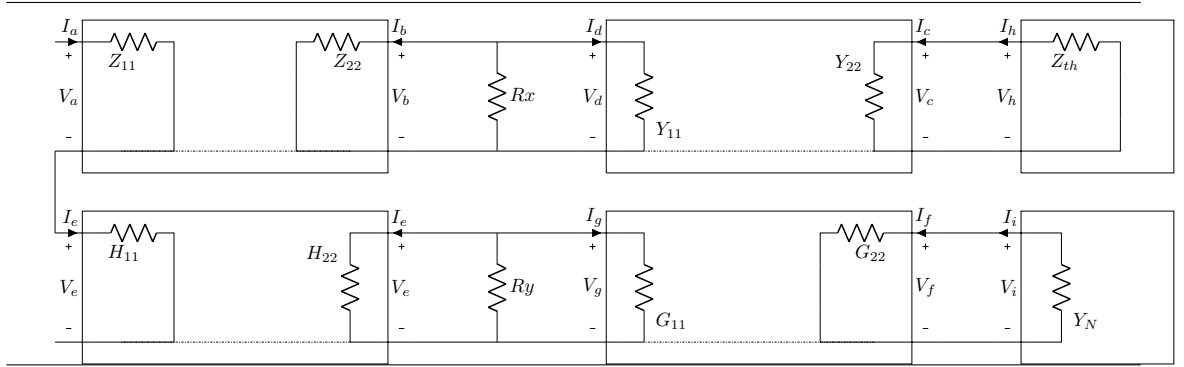


Now with the 12 and 21 parameters zeroed, normal form:

```

1 \resizebox{\textwidth}{!}{
2 \begin{tikzpicture}
3   \draw (0,0) \ncoord(ref) node[Quad Z,anchor=1+,Z12=0,Z21=0,I1=$I_a$,V1=$V_a$,I2=$I_b$,V2=$V_b$] (Qz1){}
4     (Qz1.2+) -- ++(1.5,0) \ncoord(X) -- ++(1.5,0) node[Quad Y,anchor=1+,Y12=0,Y21=0,I1=$I_d$,V1=$V_d$,I2=$I_c$,V
5       2=$V_c$] (Qy1){}
6     (Qy1.2+) -- ++(1,0) node[Thevenin,anchor=1+,Vth=0,I1=$I_h$,V1=$V_h$] (th1){}
7     (Qz1.1-) -- ++(0,-1.5) node[Quad H,anchor=1+,H12=0,H21=0,I1=$I_e$,V1=$V_e$,I2=$I_e$,V2=$V_e$] (Qh1){}
8     (Qh1.2+) -- ++(1.5,0) \ncoord(Y) -- ++(1.5,0) node[Quad G,anchor=1+,G12=0,G21=0,I1=$I_g$,V1=$V_g$,I2=$I_f$,V
9       2=$V_f$] (Qg1){}
10    (Qg1.2+) -- ++(1,0) node[Norton,anchor=1+,In=0,I1=$I_i$,V1=$V_i$] (nr1){}
11    (Qz1.2-) -- (Qy1.1-) (Qy1.2-) -- (th1.1-)
12    (Qh1.2-) -- (Qg1.1-) (Qg1.2-) -- (nr1.1-)
13    ;
14    \draw (X) to[R=$R_x$] (X |- Qz1.2-)
15          (Y) to[R=$R_y$] (Y |- Qh1.2-)
16    ;
17 \end{tikzpicture}
18 }

```

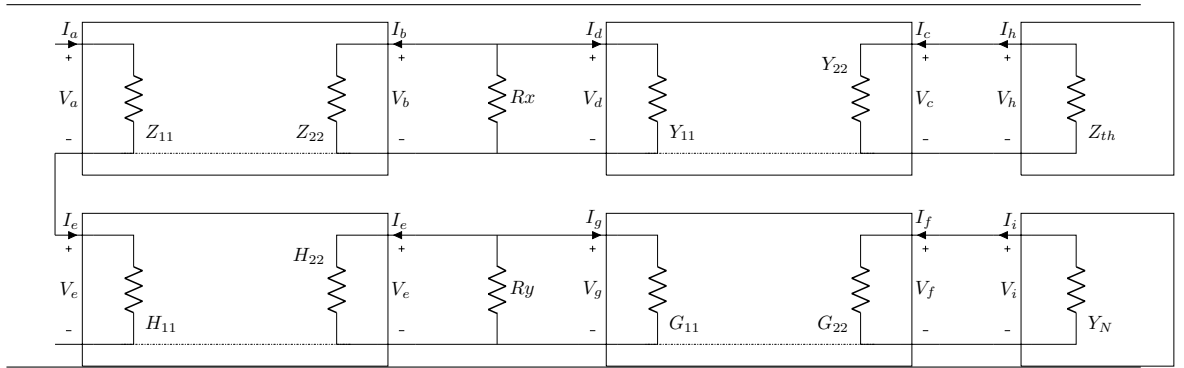


Same as last one, but with an alternate form:

```

1 \resizebox{\textwidth}{!}{
2 \begin{tikzpicture}
3   \draw (0,0) \ncoord(ref) node[Quad Z,anchor=1+,Z12=0,Z21=0,I1=$I_a$,V1=$V_a$,I2=$I_b$,V2=$V_b$] (Qz1){}
4     (Qz1.2+) -- ++(1.5,0) \ncoord(X) -- ++(1.5,0) node[Quad Y,anchor=1+,Y12=0,Y21=0,I1=$I_d$,V1=$V_d$,I2=$I_c$,V
5       2=$V_c$] (Qy1){}
6     (Qy1.2+) -- ++(1,0) node[Thevenin,anchor=1+,Vth=0,I1=$I_h$,V1=$V_h$] (th1){}
7     (Qz1.1-) -- ++(0,-1.5) node[Quad H,anchor=1+,H12=0,H21=0,I1=$I_e$,V1=$V_e$,I2=$I_e$,V2=$V_e$] (Qh1){}
8     (Qh1.2+) -- ++(1.5,0) \ncoord(Y) -- ++(1.5,0) node[Quad G,anchor=1+,G12=0,G21=0,I1=$I_g$,V1=$V_g$,I2=$I_f$,V
9       2=$V_f$] (Qg1){}
10    (Qg1.2+) -- ++(1,0) node[Norton,anchor=1+,In=0,I1=$I_i$,V1=$V_i$] (nr1){}
11    (Qz1.2-) -- (Qy1.1-) (Qy1.2-) -- (th1.1-)
12    (Qh1.2-) -- (Qg1.1-) (Qg1.2-) -- (nr1.1-)
13    ;
14    \draw (X) to[R=$R_x$] (X |- Qz1.2-)
15          (Y) to[R=$R_y$] (Y |- Qh1.2-)
16    ;
17 \end{tikzpicture}
18 }

```



Setting all parameters, some impedances as zig-zag, others as generic, per quadripole:

```

1 \resizebox{\textwidth}{!}{
2 \begin{tikzpicture}
3   \draw (0,0) \ncoord(ref) node[Quad Z,alt,round sources,european,anchor=1+,Z11=$Z_a$,Z22=$Z_b$,Z12=$Z_{re}$,Z21=$
4     Z_{fe}$,I1=$I_a$,V1=$V_a$,I2=$I_b$,V2=$V_b$] (Qz1){}
5     (Qz1.2+) -- ++(1.5,0) \ncoord(X) -- ++(1.5,0) node[Quad Y,alt,anchor=1+,Y11=$Y_a$,Y22=$Y_b$,Y12=$Y_{re}$,Y21=$
6     Y_{fe}$,I1=$I_d$,V1=$V_d$,I2=$I_c$,V2=$V_c$] (Qy1){}
7     (Qy1.2+) -- ++(1,0) node[Thevenin,alt,anchor=1+,Vth=$V_1$,Zth=$Z_a$,I1=$I_h$,V1=$V_h$] (th1){}
8     (Qz1.1-) -- ++(0,-1.5) node[Quad H,european,alt,anchor=1+,H11=$H_a$,H22=$H_b$,H12=$H_{re}$,H21=$H_{fe}$,I1=$I_
9     e$,V1=$V_e$,I2=$I_e$,V2=$V_e$] (Qh1){}
10    (Qh1.2+) -- ++(1.5,0) \ncoord(Y) -- ++(1.5,0) node[Quad G,alt,anchor=1+,G11=$G_a$,G22=$G_b$,G12=$G_{re}$,G21=$
11    G_{fe}$,I1=$I_g$,V1=$V_g$,I2=$I_f$,V2=$V_f$] (Qg1){}
12    (Qg1.2+) -- ++(1,0) node[Norton,alt,control sources,european,anchor=1+,In=$I_b$,Yn=$Y_b$,I1=$I_i$,V1=$V_i$] (nr
13    1){}
14    (Qz1.2-) -- (Qy1.1-) (Qy1.2-) -- (th1.1-)
15    (Qh1.2-) -- (Qg1.1-) (Qg1.2-) -- (nr1.1-)
16  ;
17  \draw (X) to[R=$R_x$] (X |- Qz1.2-)
18        (Y) to[R=$R_y$] (Y |- Qh1.2-)
19  ;
20 \end{tikzpicture}
21 }

```

