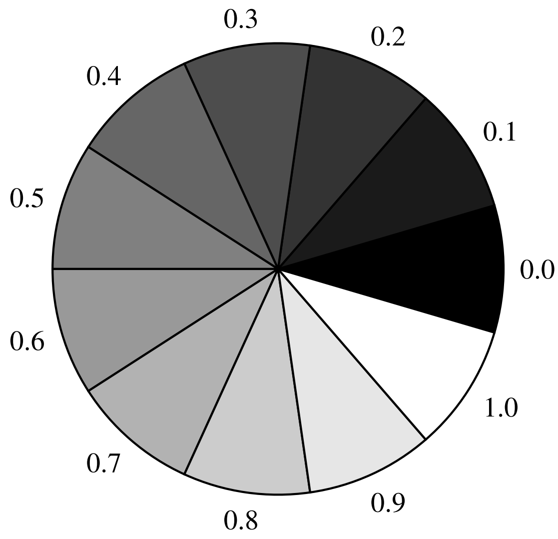
```
\def\zigzag{\psline(0,0)(.5,1)(1.5,-1)(2,0)}%
\psset{unit=.25,linewidth=1.5pt}
\multips(0,0)(2,0){8}{\zigzag}
```

multido

PSTricks is distributed with a much more general loop macro, called **\multido**. You must input the file multido.tex or multido.sty. See the documentation multido.doc for details. Here is a sample of what you can do:

```
\begin{pspicture}(-3.4,-3.4)(3.4,3.4)
  \newgray{mygray}{0}   % Initialize 'mygray' for benefit
  \psset{fillstyle=solid,fillcolor=mygray}   % of this line.
  \SpecialCoor
  \degrees[1.1]
  \multido{\n=0.0+.1}{11}{%
    \newgray{mygray}{\n}
    \rput{\n}{\pswedge{3}{-.05}{.05}}
    \uput{3.2}[\n](0,0){\small\n}}
\end{pspicture}
```



All of these loop macros can be nested.

26 Axes

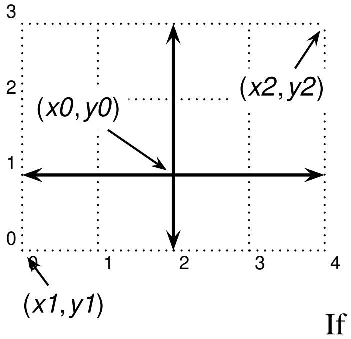
pst-plot

The axes command described in this section is defined in pst-plot.tex / pst-plot.sty, which you must input first. pst-plot.tex, in turn, will automatically input multido.tex, which is used for putting the labels on the axes.

The macro for making axes is:

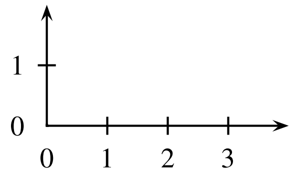
\psaxes*[*par*]{*arrows*}(*x0,y0*)(*x1,y1*)**(*x2,y2*)**

The coordinates must be Cartesian coordinates. They work the same way as with **\psgrid**. That is, if we imagine that the axes are enclosed in a rectangle, (*x1,y1*) and (*x2,y2*) are opposing corners of the rectangle. (I.e., the x-axis extends from *x1* to *x2* and the y-axis extends from *y1* to *y2*.) The axes intersect at (*x0,y0*). For example:



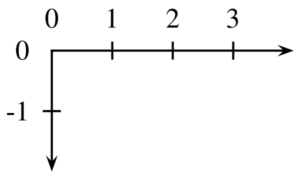

```
\psaxes[linewidth=1.2pt,labels=none,
   ticks=none]{<->}(2,1)(0,0)(4,3)
```

If (*x0,y0*) is omitted, then the origin is (*x1,y1*). If both (*x0,y0*) and (*x1,y1*) are omitted, (0,0) is used as the default. For example, when the axes enclose a single orthont, only (*x2,y2*) is needed:




```
\psaxes{->}(4,2)
```

Labels (numbers) are put next to the axes, on the same side as *x1* and *y1*. Thus, if we enclose a different orthont, the numbers end up in the right place:

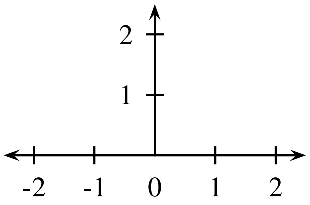



```
\psaxes{->}(4,-2)
```

Also, if you set the **arrows** parameter, the first arrow is used for the tips at *x1* and *y1*, while the second arrow is used for the tips at *x2* and *y2*. Thus, in the preceding examples, the arrowheads ended up in the right place too.[12]

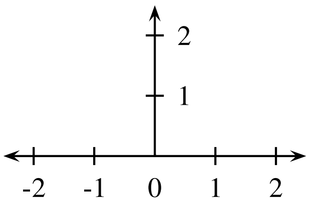
[12]Including a first arrow in these examples would have had no effect because arrows are never drawn at the origin.

When the axes don't just enclose an orthont, that is, when the origin is not at a corner, there is some discretion as to where the numbers should go. The rules for positioning the numbers and arrows described above still apply, and so you can position the numbers as you please by switching *y1* and *y2*, or *x1* and *x2*. For example, compare



\psaxes{<->}(0,0)(-2.5,0)(2.5,2.5)

with what we get when *x1* and *x2* are switched:



\psaxes{<->}(0,0)(2.5,0)(-2.5,2.5)

\psaxes puts the ticks and numbers on the axes at regular intervals, using the following parameters:

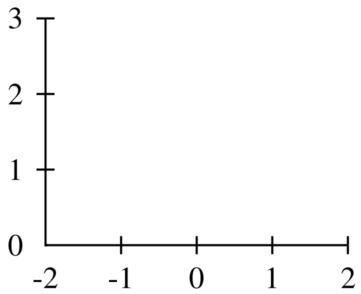
Horitontal	*Vertical*	*Dflt*	*Description*
Ox=num	**Oy=num**	0	Label at origin.
Dx=num	**Dy=num**	1	Label increment.
dx=dim	**oy=dim**	0pt	Dist btwn labels.

When **dx** is 0, Dx\psxunit is used instead, and similarly for **dy**. Hence, the default values of 0pt for **dx** and **dy** are not as peculiar as they seem.

You have to be very careful when setting **Ox**, **Dx**, **Oy** and **Dy** to non-integer values. multido.tex increments the labels using rudimentary fixed-point arithmetic, and it will come up with the wrong answer unless **Ox** and **Dx**, or **Oy** and **Dy**, have the same number of digits to the right of the decimal. The only exception is that **Ox** or **Oy** can always be an integer, even if **Dx** or **Dy** is not. (The converse does not work, however.)[13]

[13]For example, **Ox=1.0** and **Dx=1.4** is okay, as is **Ox=1** and **Dx=1.4**, but **Ox=1.4** and **Dx=1**, or **Ox=1.4** and **Dx=1.15**, is not okay. If you get this wrong, PSTricks won't complain, but you won't get the right labels either.

Note that **\psaxes**'s first coordinate argument determines the physical position of the origin, but it doesn't affect the label at the origin. E.g., if the origin is at (1,1), the origin is still labeled 0 along each axis, unless you explicitly change **Ox** and **Oy**. For example:




```
\psaxes[Ox=-2](-2,0)(2,3)
```

The ticks and labels use a few other parameters as well:

labels=*all/x/y/none* **Default: all**

To specify whether labels appear on both axes, the x-axis, the y-axis, or neither.

showorigin=*true/false* **Default: true**

If true, then labels are placed at the origin, as long as the label doesn't end up on one of the axes. If false, the labels are never placed at the origin.

ticks=*all/x/y/none* **Default: all**

To specify whether ticks appear on both axes, the x-axis, the y-axis, or neither.

tickstyle=*full/top/bottom* **Default: full**

For example, if **tickstyle=top**, then the ticks are only on the side of the axes away from the labels. If **tickstyle=bottom**, the ticks are on the same side as the labels. full gives ticks extending on both sides.

ticksize=*dim* **Default: 3pt**

Ticks extend *dim* above and/or below the axis.

The distance between ticks and labels is **\pslabelsep**, which you can change with the **labelsep** parameter.

The labels are set in the current font (ome of the examples above were preceded by \small so that the labels would be smaller). You can do fancy things with the labels by redefining the commands:

\psxlabel
\psylabel

E.g., if you want change the font of the horizontal labels, but not the vertical labels, try something like

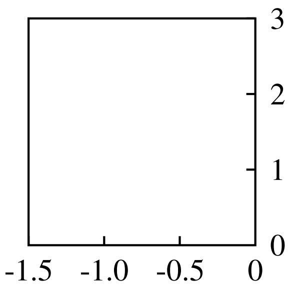
```
\def\psxlabel#1{\small #1}
```

You can choose to have a frame instead of axes, or no axes at all (but you still get the ticks and labels), with the parameter:

axesstyle=*axes/frame/none* **Default: axes**

The usual **linestyle**, **fillstyle** and related paremeters apply.

For example:




```
\psaxes[Dx=.5,dx=1,tickstyle=top,axesstyle=frame](-3,3)
```

The **\psaxes** macro is pretty flexible, but PSTricks contains some other tools for making axes from scratch. E.g., you can use **\psline** and **\psframe** to draw axes and frames, respectively, **\multido** to generate labels (see the documentation for multido.tex), and **\multips** to make ticks.

VI Text Tricks

27 Framed boxes

The macros for framing boxes take their argument, put it in an \hbox, and put a PostScript frame around it. (They are analogous to LaTeX's \fbox). Thus, they are composite objects rather than pure graphics objects. In addition to the graphics parameters for **\psframe**, these macros use the following parameters:

framesep=*dim* **Default: 3pt**

Distance between each side of a frame and the enclosed box.

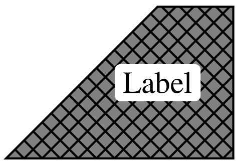
boxsep=*true/false* **Default: true**

When true, the box that is produced is the size of the frame or whatever that is drawn around the object. When false, the box that is produced is the size of whatever is inside, and so the frame is "transparent" to TeX. This parameter only applies to **\psframebox**, **\pscirclebox**, and **\psovalbox**.

Here are the three box-framing macros:

\psframebox*[*par*]**{*stuff*}**

A simple frame (perhaps with rounded corners) is drawn using **\psframe**. The * option is of particular interest. It generates a solid frame whose color is **fillcolor** (rather than **linecolor**, as with the closed graphics objects). Recall that the default value of **fillcolor** is white, and so this has the effect of blotting out whatever is behind the box. For example,




```
\pspolygon[fillcolor=gray,fillstyle=crosshatch*](0,0)(3,0)
   (3,2)(2,2)
\rput(2,1){\psframebox*[framearc=.3]{Label}}
```

\psdblframebox*[*par*]***{stuff}***

This draws a double frame. It is just a variant of **\psframebox**, defined by

```
\newpsobject{psdblframebox}{psframebox}{doublesep=\pslinewidth}
```

For example,

```
\psdblframebox[linewidth=1.5pt]{%
  \parbox[c]{6cm}{\raggedright A double frame is drawn
  with the gap between lines equal to {\tt doublesep}}}
```

A double frame is drawn with the gap between lines equal to doublesep

\psshadowbox*[*par*]***{stuff}***

This draws a single frame, with a shadow.

Great Idea!!

```
\psshadowbox{\bf Great Idea!!}
```

You can get the shadow with **\psframebox** just be setting the **shadowsize** parameter, but with **\psframebox** the dimensions of the box won't reflect the shadow (which may be what you want!).

\pscirclebox*[*par*]***{stuff}***

This draws a circle. With **boxsep=true**, the size of the box is close to but may be larger than the size of the circle. For example:

You are
here

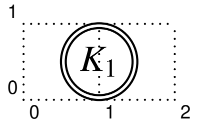
```
\pscirclebox{\begin{tabular}{c} You are \\ here \end{tabular}}
```

\cput*[*par*]***{angle}(x,y){stuff}***

This combines the functions of **\pscirclebox** and **\rput**. It is like

```
\rput{<angle>}(x0,y0){\string\pscirclebox*[<par>]{<stuff>}}
```

but it is more efficient. Unlike the **\rput** command, there is no argument for changing the reference point; it is always the center of the box. Instead, there is an optional argument for changing graphics parameters. For example




```
\cput[doubleline=true](1,.5){\large $K_1$}
```

\psovalbox*[*par*]**{*stuff*}**

This draws an ellipse. If you want an oval with square sides and rounded corners, then use **\psframebox** with a positive value for **rectarc** or **linearc** (depending on whether **cornersize** is relative or absolute). Here is an example that uses **boxsep=false**:

At the introductory price of $13.99, it pays to act now!

```
At the introductory price of
\psovalbox[boxsep=false,linecolor=darkgray]{\$13.99},
it pays to act now!
```

You can define variants of these box framing macros using the **\newpsobject** command.

If you want to control the final size of the frame, independently of the material inside, nest *stuff* in something like LaTeX's \makebox command.

28 Clipping

The command

\clipbox[*dim*]**{*stuff*}**

puts *stuff* in an \hbox and then clips around the boundary of the box, at a distance *dim* from the box (the default is 0pt).

The **\pspicture** environment also lets you clip the picture to the boundary.

The command

\psclip{*graphics*} ... **\endpsclip**

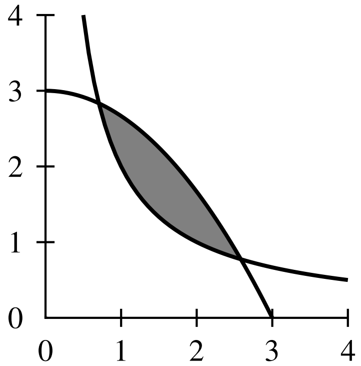
sets the clipping path to the path drawn by the graphics object(s), until the **\endpsclip** command is reached. **\psclip** and **\endpsclip** must be properly nested with respect to TeX grouping. Only pure graphics (those described in Part II and **\pscustom**) are permitted. An Overfull \hbox warning indicates that the *graphics* argument contains extraneous output or space. Note that the graphics objects otherwise act as usual, and the **\psclip** does not otherwise affect the surrounded text. Here is an example:

“One of the best new plays I have seen all year: cool, poetic, ironic …” proclaimed *The Guardian* upon the Lon-

```
\parbox{4.5cm}{%
  \psclip{\psccurve[linestyle=none](-3,-2)
    (0.3,-1.5)(2.3,-2)(4.3,-1.5)(6.3,-2)(8,-1.5)(8,2)(-3,2)}
  “One of the best new plays I have seen all year: cool, poetic,
  ironic   \ldots” proclaimed {\em The Guardian} upon the London
  premiere of this extraordinary play about a Czech director and
  his actress wife, confronting exile in America.\vspace{-1cm}
  \endpsclip}
```

If you don’t want the outline to be painted, you need to include **linestyle=none** in the parameter changes. You can actually include more than one graphics object in the argument, in which case the clipping path is set to the intersection of the paths.

\psclip can be a useful tool in picture environments. For example, here it is used to shade the region between two curves:




```
\psclip{%
  \pscustom[linestyle=none]{%
    \psplot{.5}{4}{2 x div}
    \lineto(4,4)}
  \pscustom[linestyle=none]{%
    \psplot{0}{3}{3 x x mul 3 div sub}
    \lineto(0,0)}}
\psframe*[linecolor=gray](0,0)(4,4)
\endpsclip
\psplot[linewidth=1.5pt]{.5}{4}{2 x div}
\psplot[linewidth=1.5pt]{0}{3}{3 x x mul 3 div sub}
\psaxes(4,4)
```

Driver notes: The clipping macros use **\pstverbscale** and **\pstVerb**. Don’t be surprised if PSTricks’s clipping does not work or causes problem—it is never robust. \endpsclip uses initclip. This can interfere with other clipping operations, and especially if the TEX document is converted to an Encapsulated PostScript file. The command **\AltClipMode** causes **\psclip** and **\endpsclip** to use gsave and grestore instead. This bothers some drivers, such as NeXTTeX’s TeXView, especially if **\psclip** and **\endpsclip** do not end up on the same page.

29 Rotation and scaling boxes

There are versions of the standard box rotation macros:

\rotateleft{*stuff*}

\rotateright{*stuff*}
\rotatedown{*stuff*}

stuff is put in an \hbox and then rotated or scaled, leaving the appropriate amount of spaces. Here are a few uninteresting examples:

Left Down Right

```
\Large\bf \rotateleft{Left} \rotatedown{Down} \rotateright{Right}
```

There are also two box scaling macros:

\scalebox{*num1* *num2*}{*stuff*}

If you give two numbers in the first argument, *num1* is used to scale horizontally and *num2* is used to scale vertically. If you give just one number, the box is scaled by the same in both directions. You can't scale by zero, but negative numbers are OK, and have the effect of flipping the box around the axis. You never know when you need to do something like this (\scalebox{-1 1}{this}).

\scaleboxto(*x*,*y*){*stuff*}

This time, the first argument is a (Cartesian) coordinate, and the box is scaled to have width *x* and height (plus depth) *y*. If one of the dimensions is 0, the box is scaled by the same amount in both directions. For example:

Big and long

```
\scaleboxto(4,2){Big and long}
```

PSTricks defines LR-box environments for all these box rotation and scaling commands:

```
\pslongbox{Rotateleft}{\rotateleft}
\pslongbox{Rotateright}{\rotateright}
\pslongbox{Rotatedown}{\rotatedown}
\pslongbox{Scalebox}{\scalebox}
\pslongbox{Scaleboxto}{\scaleboxto}
```

Here is an example where we **Rotatedown** for the answers to exercises:

Question: How do Democrats organize a firing squad?

Answer: First they get in a circle, …

```
Question: How do Democrats organize a firing squad?
  \begin{Rotatedown}
  \parbox{\hsize}{Answer: First they get in a circle, \ldots\hss}%
\end{Rotatedown}
```

See the documentation of fancybox.sty for tips on rotating a LaTeX table or figure environment, and other boxes.

VII Nodes and Node Connections

pst-node

All the commands described in this part are contained in the file pst-node.tex/pst-node.sty.

The node and node connection macros let you connect information and place labels, without knowing the exact position of what you are connecting or of where the lines should connect. These macros are useful for making graphs and trees, mathematical diagrams, linguistic syntax diagrams, and connecting ideas of any kind. They are the trickiest tricks in PSTricks!

Although you might use these macros in pictures, positioning and rotating them with **\rput**, you can actually use them anywhere. For example, I might do something like this in a guide about page styles:

```
\makeatletter
\gdef\ps@temp{\def\@oddhead{}\def\@evenhead{}
  \def\@oddfoot{\small\sf
    \ovalnode[boxsep=false]{A}{\rightmark}
    \nccurve[ncurv=.5,angleB=240,angleA=180,nodesep=6pt]{<-}{A}{B}
    \hfil\thepage}
  \let\@evenfoot\@oddfoot}
\makeatother
\thispagestyle{temp}
  With the {\tt myfooters} page style, the name of the current section
appears at the bottom of each \rnode{B}{page}.
```

With the myfooters page style, the name of the current section appears at the bottom of each page.

You can use nodes in math mode and in alignment environments as well. Here is an example of a commutative diagram:

$$
\begin{tikzcd}
& A \arrow[dl, "f"'] \arrow[d, "g"] \\
B \arrow[r, dotted, "h"] & C
\end{tikzcd}
$$

```
$
\begin{array}{c@{\hskip 1cm}c}
   & \rnode{a}{A}\\[2cm]
   \rnode{b}{B} & \rnode{c}{C}
\end{array}
\psset{nodesep=3pt}
\everypsbox{\scriptstyle}
\ncline{->}{a}{b}\Bput{f}
\ncline{->}{a}{c}\Aput{g}
\ncline[linestyle=dotted]{->}{b}{c}\Aput{h}
$
```

There are three components to the node macros:

Node definitions The node definitions let you assign a name and shape to an object. See Section 30.

Node connections The node connections connect two nodes, identified by their names. See Section 31.

Node labels The node label commands let you affix labels to the node connections. See Section 32.

30 Nodes

The *name* of a node must contain only letters and numbers, and must begin with a letter.

Warning: Bad node names can cause PostScript errors.

\rnode[*refpoint*]**{*name*}{*stuff*}**

This assigns the *name* to the node, which will have a rectangular shape for the purpose of making connections, with the "center" at the reference point (i.e., node connections will point to the reference point. **\rnode** was used in the two examples above.

\Rnode(x,y)**{*name*}{*stuff*}**

This is like **\rnode**, but the reference point is calculated differently. It is set to the middle of the box's baseline, plus (x,y). If you omit the (x,y) argument, command

\RnodeRef

is substituted. The default definition of **\RnodeRef** is 0,.7ex. E.g, the following are equivalent:

```
\Rnode(0,.6ex){stuff}
{\def\RnodeRef{0,.6ex}\Rnode{stuff}}
```

\Rnode is useful when aligning nodes by their baaelines, such as in commutative diagrams. With **\rnode** horizontal node connections might not be quite horizontal, because of differences in the size of letters.

\pnode$(x,y)${***name***}

This creates a zero dimensional node at the point (x,y) (default (0,0)).

\cnode*[*par*](x,y){***radius***}{***name***}

This draws a circle and assigns the *name* to it.

\circlenode*[*par*]{***name***}{***stuff***}

This is a variant of **\pscirclebox** that gives the node the shape of the circle.

\cnodeput*[*par*]{*angle*}(x,y){***name***}{***stuff***}

This is a variant of **\cput** that gives the node the shape of the circle.

\ovalnode*[*par*]{***name***}{***stuff***}

This is a variant of **\psovalbox** that gives the node the shape of the ellipse.

The reason that there is no **\framenode** command is that using **\psframebox** (or **\psshadowbox** or **\psdblframebox**) in the argument of **\rnode** gives the desired result.

31 Node connections

All the node connection commands begin with nc, and they all have the same syntax:

```
\<nodeconnection>[<par>]{<arrows>}{<nodeA>}{<nodeB>}
```

A line of some sort is drawn from *nodeA* to *nodeB*. Some of the node connection commands are a little confusing, but with a little experimentation you will figure them out, and you will be amazed at the things you can do.

The node and point connections can be used with **\pscustom**. The beginning of the node connection is attached to the current point by a straight line, as with **\psarc**.[14]

When we refer to the A and B nodes below, we are referring only to the order in which the names are given as arguments to the node connection macros.

When a node name cannot be found on the same page as the node connection command, you get either no node connection or a nonsense node connection. However, TEX will not report any errors.

The node connections use the following parameters:

nodesep=*dim* **Default: 0**

The border around the nodes added for the purpose of determining where to connect the lines.

offset=*dim* **Default: 0**

After the node connection point is calculated, it is shift up for *nodeA* and down for *nodeB* by *dim*, where “up” and “down” assume that the connecting line points to the right from the node.

arm=*dim* **Default: 10pt**

Some node connections start with a segment of length *dim* before turning somewhere.

angle=*angle* **Default: 0**

Some node connections let you specify the angle that the node connection should connect to the node.

arcangle=*angle* **Default: 8**

This applies only to **\ncarc**, and is described below.

ncurv=*num* **Default: .67**

This applies only to **\nccurve** and **\pccurve**, and is described below.

[14] See page 71 if you want to use the nodes as coordinates in other PSTricks macros.

loopsize=*dim* **Default: 1cm**

This applies only the **\ncloop** and **\pcloop**, and is described below.

You can set these parameters separately for the two nodes. Just add an A or B to the parameter name. E.g.

```
\psset{nodesepA=3pt, offsetA=5pt, offsetB=3pt, arm=1cm}
```

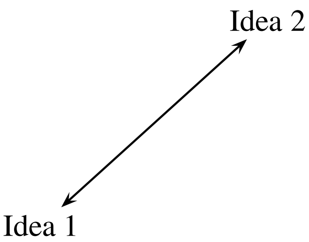
sets **nodesep** for the A node, but leaves the value for the B node unchanged, sets offset for the A and B nodes to different values, and sets arm for the A and B nodes to the same value.

Don't forget that by using the **border** parameter, you can create the impression that one node connection passes over another.

Here is a description of the individual node connection commands:

\ncline*[*par*]{*arrows*}**{*nodeA*}{*nodeB*}**

This draws a straight line between the nodes. Only the **offset** and **nodesep** parameters are used.



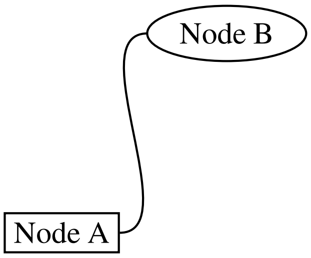

```
\rput[bl](0,0){\rnode{A}{Idea 1}}
\rput[tr](4,3){\rnode{B}{Idea 2}}
\ncline[nodesep=3pt]{<->}{A}{B}
```

\ncLine*[*par*]{*arrows*}**{*nodeA*}{*nodeB*}**

This is like **\ncline**, but the labels (with **\lput**, etc) are positioned as if the line began and ended at the center of the nodes. This is useful if you have multiple parallel lines and you want the labels to line up, even though the nodes are of varying size, e.g., in commutative diagrams.

\nccurve*[*par*]{*arrows*}**{*nodeA*}{*nodeB*}**

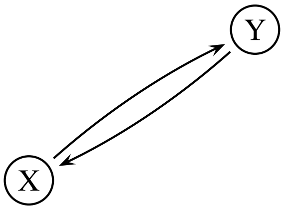
This draws a bezier curve between the nodes. It uses the **nodesep**, **offset**, **angle** and **ncurv** parameters.




```
\rput[bl](0,0){\rnode{A}{\psframebox{Node A}}}
\rput[tr](4,3){\ovalnode{B}{Node B}}
\nccurve[angleB=180]{A}{B}
```

\ncarc*[*par*]{*arrows*}***{nodeA}{nodeB}***

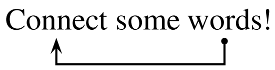
This is actually a variant of **\nccurve**. I.e., it also connects the nodes with a bezier curve, using the **nodesep**, **offset**, and **ncurv** parameters. However, the curve connects to node A at an angle **arcangleA** from the line between A and B, and connects to node B at an angle -**arcangleB** from the line between B and A. For small, equal values of **angleA** and **angleB** (e.g., the default value of 8) and with the default value of **ncurv**, the curve approximates an arc of a circle. **\ncarc** is a nice way to connect two nodes with two lines.




```
\cnodeput(0,0){A}{X}
\cnodeput(3,2){B}{Y}
\psset{nodesep=3pt}
\ncarc{->}{A}{B}
\ncarc{->}{B}{A}
```

\ncbar*[*par*]{*arrows*}***{nodeA}{nodeB}***

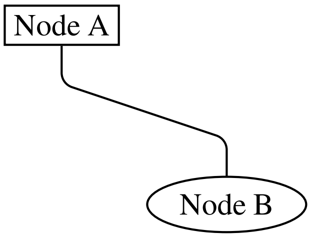
First, lines are drawn attaching to both nodes at an angle **angleA** and of lengths **armA** and **armB**. Then one of the arms is extended so that when the two are connected, the finished line contains 3 segments meeting at right angles. Generally, the whole line has three straight segments. The value of **linearc** is used for rounding the corners.




```
\rnode{A}{Connect} some \rnode{B}{words}!
\ncbar[nodesep=3pt,angle=-90]{<-**}{A}{B}
```

\ncdiag*[*par*]{*arrows*}***{nodeA}{nodeB}***

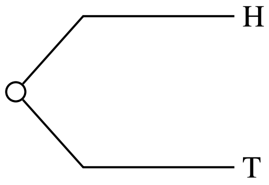
First, the arms are drawn using **angle** and **arm**. Then they are connected with a straight line. Generally, the whole line has three straight segments. The value of **linearc** is used for rounding the corners.




```
\rput[tl](0,3){\rnode{A}{\psframebox{Node A}}}
\rput[br](4,0){\ovalnode{B}{Node B}}
\ncdiag[angleA=-90,angleB=90,arm=.5,linearc=.2]{A}{B}
```

\ncdiagg*[*par*]{*arrows*}**{*nodeA*}{*nodeB*}**

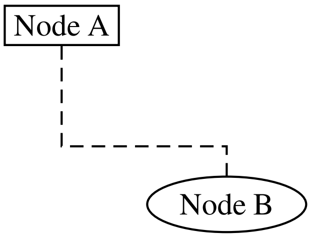
This is similar to **\ncdiag**, but only the arm for node A is drawn. The end of this arm is then connected directly to node B. The connection typically has two segments. The value of **linearc** is used for rounding the corners.




```
\cnode(0,0){4pt}{a}
\rput[l](3,1){\rnode{b}{H}}
\rput[l](3,-1){\rnode{c}{T}}
\ncdiagg[angleA=180,armA=2cm,nodesepA=3pt]{b}{a}
\ncdiagg[angleA=180,armA=2cm,nodesepA=3pt]{c}{a}
```

\ncangle*[*par*]{*arrows*}**{*nodeA*}{*nodeB*}**

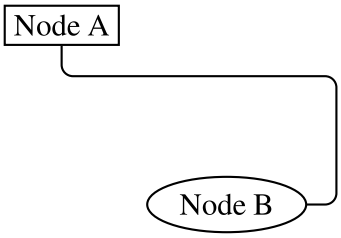
The node connection points are determined by **angleA** and **angleB** (and **nodesep** and **offset**). Then an arm is drawn for node B using **armB**. This arm is connected to node A by a right angle, that also meets node A at angle **angleA**. Generally, the whole line has three straight segments, but it can have fewer. The value of **linearc** is used for rounding the corners. Simple, right? Here is an example:




```
\rput[tl](0,3){\rnode{A}{\psframebox{Node  A}}}
\rput[br](4,0){\ovalnode{B}{Node  B}}
\ncangle[angleA=-90,angleB=90,arm=.4cm,
   linestyle=dashed]{A}{B}
```

\ncangles*[*par*]{*arrows*}**{*nodeA*}{*nodeB*}**

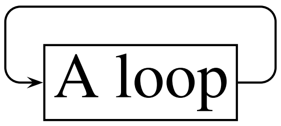
This is similar to **\ncangle**, but both **armA** and **armB** are used. The arms are connected by a right angle that meets arm A at a right angle as well. Generally there are four segments (hence one more angle than **\ncangle**, and hence the s in **\ncangles**). The value of **linearc** is used for rounding the corners. Compare this example with the previous one:




```
\rput[tl](0,3){\rnode{A}{\psframebox{Node  A}}}
\rput[br](4,0){\ovalnode{B}{Node  B}}
\ncangles[angleA=-90,arm=.4cm,linearc=.15]{A}{B}
```

\ncloop*[*par*]{*arrows*}***{nodeA}{nodeB}***

The first segment is **armA**, then it makes a 90 degree turn to the left, drawing a segment of length **loopsize**. The next segment is again at a right angle; it connects to **armB**. For example:




```
\rnode{a}{\psframebox{\Huge A loop}}
\ncloop[angleB=180,loopsize=1,arm=.5,linearc=.2]{->}{a}{a}
```

\nccircle*[*par*]{*arrows*}***{node}{radius}***

This draws a circle from a node to itself. It is the only node connection command of this sort. The circle starts at angle **angleA** and goes around the node counterclockwise, at a distance **nodesepA** from the node.

The node connection commands make interesting drawing tools as well, as an alternative to **\psline** for connecting two points. There are variants of the node connection commands for this purpose. Each begins with pc (for "point connection") rather than nc. E.g.,

```
\pcarc{<->}(3,4)(6,9)
```

gives the same result as

```
\pnode(3,4){A}\pnode(6,9){B}\pcarc{<->}{A}{B}
```

Only **\ncLine** and **\nccircle** do not have pc variants:

\pcline*[*par*]{*arrows*}***(x1,y1)(x2,y2)***

Like **\ncline**.

\pccurve*[*par*]{*arrows*}***(x1,y1)(x2,y2)***

Like **\nccurve**.

\pcarc*[*par*]{*arrows*}***(x1,y1)(x2,y2)***

Like **\ncarc**.

\pcbar*[*par*]{*arrows*}***(x1,y1)(x2,y2)***

Like **\ncbar**.

\pcdiag*[*par*]{*arrows*}***(x1,y1)(x2,y2)***

Like **\ncdiag**.

\pcangle*[*par*]{*arrows*}***(x1,y1)(x2,y2)***

Like **\ncangle**.

\pcloop*[*par*]{*arrows*}***(x1,y1)(x2,y2)***

Like **\ncloop**.

32 Attaching labels to node connections

Now we come to the commands for attaching labels to the node connections. The node label command must come right after the node connection to which the label is to be attached. You can attach more than one label to a node connection, and a label can include more nodes.

The node label commands must end up on the same TeX page as the node connection to which the label corresponds.

The coordinate argument in other PSTricks put commands is a single number in the node label commands: (*pos*). This number selects a point on the node connection, roughly according to the following scheme: Each node connection has potentially one or more segments, including the arms and connecting lines. A number *pos* between 0 and 1 picks a point on the first segment from node A to B, (fraction *pos* from the beginning to the end of the segment), a number between 1 and 2 picks a number on the second segment, and so on. Each node connection has its own default value of the positioning coordinate, which is used by some short versions of the label commands.

Here are the details for each node connection:

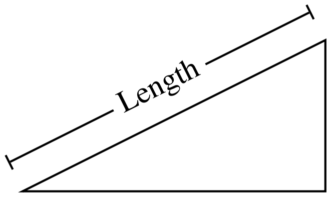
Connection	*Segments*	*Range*	*Default*
\ncline	1	$0 \le pos \le 1$	0.5
\nccurve	1	$0 \le pos \le 1$	0.5
\ncarc	1	$0 \le pos \le 1$	0.5
\ncbar	3	$0 \le pos \le 3$	1.5
\ncdiag	3	$0 \le pos \le 3$	1.5
\ncdiagg	2	$0 \le pos \le 2$	0.5
\ncangle	3	$0 \le pos \le 3$	1.5
\ncloop	5	$0 \le pos \le 4$	2.5
\nccircle	1	$0 \le pos \le 1$	0.5

There is another difference between the node label commands and other put commands. In addition to the various ways of specifying the angle of rotation for **\rput**, with the node label commands the angle can be of the form {:*angle*}. In this case, the angle is calculated after rotating the coordinate system so that the node connection at the position of the label points to the right (from nodes A to B). E.g., if the angle is {:U}, then the label runs parallel to the node connection.

Here are the node label commands:

\lput*[*refpoint*]{*rotation*}***(pos){stuff}***

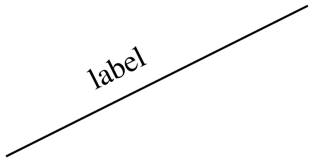
The l stands for "label". Here is an example illustrating the use of the optional star and :*angle* with **\lput**, as well as the use of the **offset** parameter with **\pcline**:




```
\pspolygon(0,0)(4,2)(4,0)
\pcline[offset=12pt]{|-|}(0,0)(4,2)
\lput*{:U}{Length}
```

(Remember that with the put commands, you can omit the coordinate if you include the angle of rotation. You are likely to use this feature with the node label commands.)

With **\lput** and **\rput**, you have a lot of control over the position of the label. E.g.,




```
\pcline(0,0)(4,2)
\lput{:U}{\rput[r]{N}(0,.4){label}}
```

puts the label upright on the page, with right side located .4 centimeters "above" the position .5 of the node connection (above if the node connection points to the right). However, the **\aput** and **\bput** commands described below handle the most common cases without **\rput**.[15]

[15]There is also an obsolete command **\Lput** for putting labels next to node connections. The syntax is

```
\Lput{<labelsep>}[<refpoint>]{<rotation>}(<pos>){<stuff>}
```

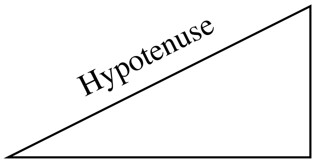
It is a combination of **\Rput** and **\lput**, equivalent to

```
\lput(<pos>){\Rput{<labelsep>}[<refpoint>]{<rotation>}(0,0){<stuff>}}
```

\Mput is a short version of **\Lput** with no {*rotation*} or (*pos*) argument. **\Lput** and **\Mput** remain part of PSTricks only for backwards compatibility.

\aput*[*labelsep*]{*angle*}***(pos){stuff}***

stuff is positioned distance **\pslabelsep** *above* the node connection, given the convention that node connections point to the right. \aput is a node-connection variant of **\uput**. For example:




```
\pspolygon(0,0)(4,2)(4,0)
\pcline[linestyle=none](0,0)(4,2)
\aput{:U}{Hypotenuse}
```

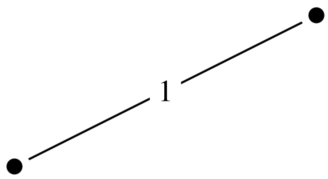
\bput*[*labelsep*]{*angle*}***(pos){stuff}***

This is like **\aput**, but *stuff* is positioned *below* the node connection.

It is fairly common to want to use the default position and rotation with these node connections, but you have to include at least one of these arguments. Therefore, PSTricks contains some variants:

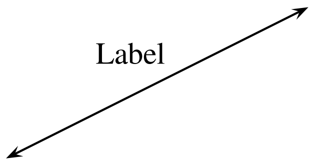
\mput*[*refpoint*]***{stuff}***
\Aput*[*labelsep*]***{stuff}***
\Bput*[*labelsep*]***{stuff}***

of **\lput**, **\aput** and **\bput**, respectively, that have no angle or positioning argument. For example:




```
\cnode*(0,0){3pt}{A}
\cnode*(4,2){3pt}{B}
\ncline[nodesep=3pt]{A}{B}
\mput*{1}
```

Here is another:




```
\pcline{<->}(0,0)(4,2)
\Aput{Label}
```

Now we can compare **\ncline** with **\ncLine**, and **\rnode** with **\Rnode**. First, here is a mathematical diagram with **\ncLine** and **\Rnode**:

```
\[
  \setlength{\arraycolsep}{1cm}
  \def\tX{\tilde{\tilde{X}}}
  \begin{array}{cc}
    \Rnode{a}{(X-A,N-A)}  &  \Rnode{b}{(\tX,a)}\\[1.5cm]
    \Rnode{c}{(X,N)}  &  \Rnode{d}{\LARGE(\tX,N)}\\[1.5cm]
  \end{array}
  \psset{nodesep=5pt,arrows=->}
  \everypsbox{\scriptstyle}
  \ncLine{a}{b}\Aput{a}
  \ncLine{a}{c}\Bput{r}
  \ncLine[linestyle=dashed]{c}{d}\Bput{b}
  \ncLine{b}{d}\Bput{s}
\]
```

$$\begin{tikzcd}
(X-A; N-A) \arrow[r, "a"] \arrow[d, "r"'] & (\tilde{\tilde{X}}; a) \arrow[d, "s"'] \\
(X; N) \arrow[r, dashed, "b"'] & (\tilde{\tilde{X}}; N)
\end{tikzcd}$$

Here is the same one, but with **\ncline** and **\rnode** instead:

$$\begin{tikzcd}
(X-A; N-A) \arrow[r, "a"] \arrow[d, "r"'] & (\tilde{\tilde{X}}; a) \arrow[d, "s"'] \\
(X; N) \arrow[r, dashed, "b"'] & (\tilde{\tilde{X}}; N)
\end{tikzcd}$$

Driver notes: The node macros use **\pstVerb** and **\pstverbscale**.

VIII Special Tricks

33 Coils and zigzags

pst-coil

The file pst-coil.tex/pst-coil.sty (and optionally the header file pst-coil.pro) defines the following graphics objects for coils and zigzags:

> **\pscoil***[*par*]{*arrows*}(*x0*,*y0*)**(*x1*,*y1*)**
> **\psCoil***[*par*]**{*angle1*}{*angle2*}**
> **\pszigzag***[*par*]{*arrows*}(*x0*,*y0*)**(*x1*,*y1*)**

These graphics objects use the following parameters:

coilwidth=*dim*	**Default: 1cm**
coilheight=*num*	**Default: 1**
coilarm=*dim*	**Default: .5cm**
coilaspect=*angle*	**Default: 45**
coilinc=*angle*	**Default: 10**

All coil and zigzag objects draw a coil or zigzag whose width (diameter) is **coilwidth**, and with the distance along the axes for each period (360 degrees) equal to

> **coilheight** x **coilwidth**.

Both **\pscoil** and **\psCoil** draw a "3D" coil, projected onto the xz-axes. The center of the 3D coil lies on the yz-plane at angle pcoilaspect to the z-axis. The coil is drawn with PostScript's lineto, joining points that lie at angle **coilinc** from each other along the coil. Hence, increasing **coilinc** makes the curve smoother but the printing slower. **\pszigzag** does not use the **coilaspect** and **coilinc** parameters.

\pscoiland **\pszigzag** connect (*x0*,*y0*) and (*x1*,*y1*), starting and ending with straight line segments of length **coilarmA** and **coilarmB**, resp. Setting **coilarm** is the same as setting **coilarmA** and **coilarmB**.

Here is an example of **\pscoil**: