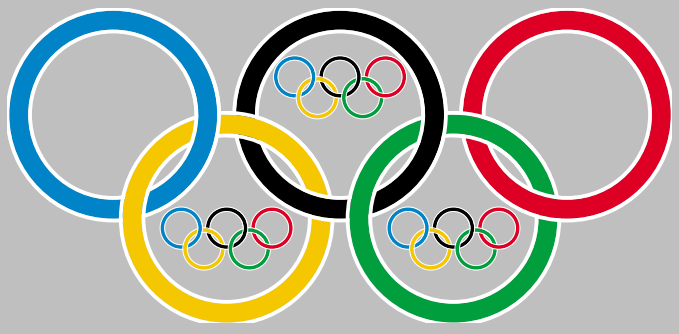
PSTricks

pstricks-add
additionals Macros for pstricks

v.3.85

January 30, 2018



Package author(s):
Dominique Rodriguez
Michael Sharpe
Herbert Voß

This version of pstricks-add needs pstricks.tex version >1.04 from June 2004, otherwise the additional macros may not work as expected. The ellipsis material and the option asolid (renamed to eofill) are now part of the new pstricks.tex package, available on CTAN. pstricks-add will for ever be an experimental and dynamical package, try it at your own risk.

- It is important to load pstricks-add as the **last** PSTricks related package, otherwise a lot of the macros won't work in the expected way.
- pstricks-add uses the extended version of the keyval package. So be sure that you have installed pst-xkey which is part of the xkeyval-package, and that all packages that use the old keyval interface are loaded **before** the xkeyval.
- the option tickstyle from pst-plot is no longer supported; use ticksize instead.
- the option xyLabel is no longer supported; use the option labelFontSize instead.
- if pstricks-add is loaded together with the package pst-func then InsideArrow of the \psbezier macro doesn't work!

Thanks to: Stefano Baroni; Martin Chicoine; Gerry Coombes; Ulrich Dirr; Christophe Fourey; Hubert Gäßlein; Jürgen Gilg; Denis Girou; Pablo Gonzáles; Peter Hutnick; Christophe Jorssen; Uwe Kern; Manuel Luque; Jens-Uwe Morawski; Tobias Nähring; Rolf Niepraschk; Alan Ristow; Christine Römer; Arnaud Schmittbuhl; John Smith; Timothy Van Zandt

Contents

1. \psGetSlope and \psGetDistance

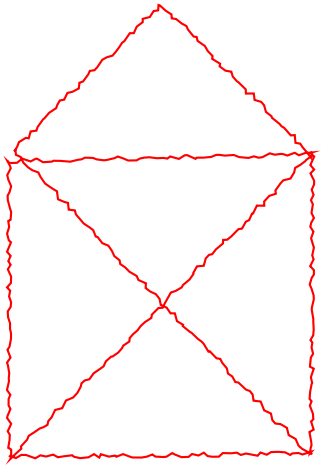
\psGetSlope(x_1,y_1)(x_2,y_2)\⟨*macro*⟩
\psGetDistance(x_1,y_1)(x_2,y_2)\⟨*macro*⟩

```
\psGetSlope(-2,1)(3,1)\SlopeVal \SlopeVal \quad
\psGetDistance(-2,1)(3,1)\DVal \DVal\\
\psGetSlope(-2,1)(-3,-1)\SlopeVal \SlopeVal\quad
\psGetDistance(-2,1)(-3,-1)\DVal \DVal\\
\psGetSlope(-2,0)(3,-1)\SlopeVal \SlopeVal\quad
\psGetDistance(-2,0)(3,-1)\DVal \DVal\\
\psGetSlope(-2111,-12)(3,1)\SlopeVal \SlopeVal\quad
%\psGetDistance(-2111,-12)(3,1)\DVal ==> Overflow!
```

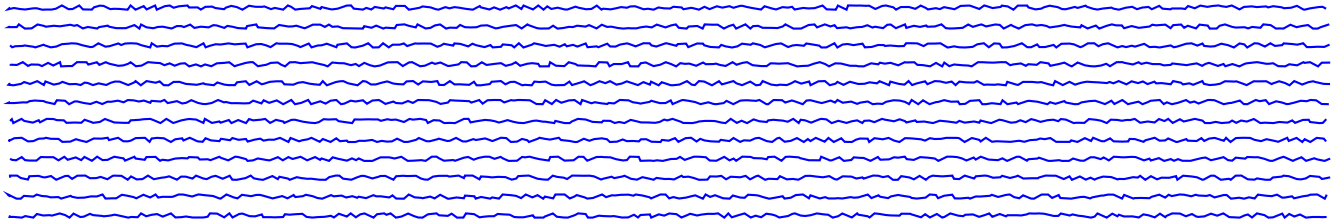
0.0 0.85759
2.0 0.55055
-0.2 1.10112
0.00615

2. "'Handmade"' lines :-)

\pslineByHand [Options] $(x_1,y_1)(x_2,y_2)(x_3,y_3)\ldots$

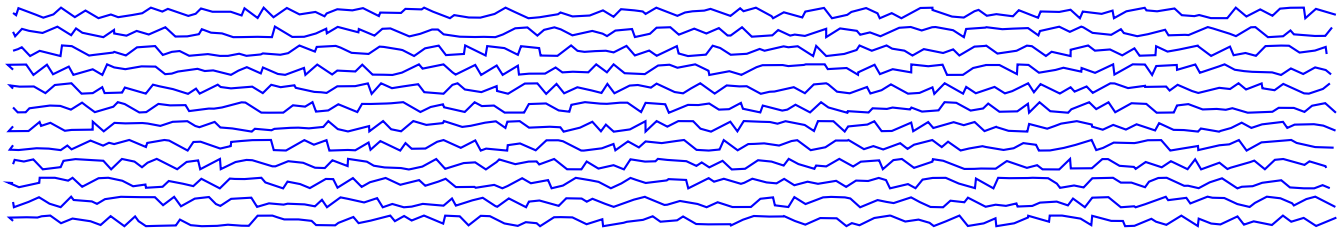


```
\begin{pspicture}(4,6)
\psset{unit=2cm}
  \pslineByHand[linecolor=red](0,0)(0,2)(2,2)(2,0)
      (0,0)(2,2)(1,3)(0,2)(2,0)
\end{pspicture}
```



```
\begin{pspicture}(\linewidth,3)
\multido{\rA=0.00+0.25}{12}{\pslineByHand[linecolor=blue](0,\rA)(\linewidth,\rA)}
\end{pspicture}
```

The amplitude and the width can be changed by the optional arguments `varsteptol` and `VarStepEpsilon`. Both are preset to `VarStepEpsilon=2,varsteptol=0.8`.

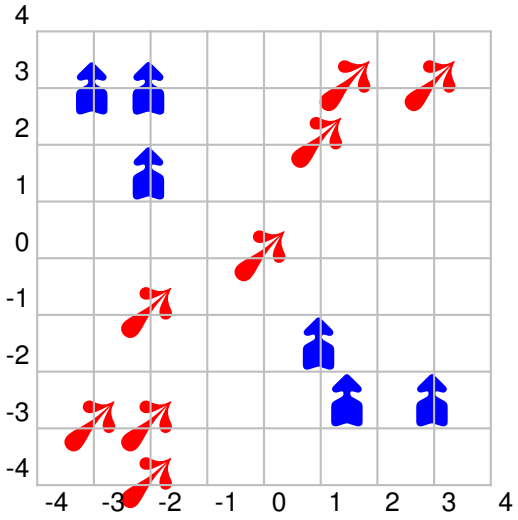


```
\begin{pspicture}(\linewidth,3)
\multido{\rA=0.00+0.25}{12}{%
  \pslineByHand[linecolor=blue,VarStepEpsilon=4,varsteptol=2](0,\rA)(\linewidth,\rA)}
\end{pspicture}
```

3. \rmultiput: a multiple \rput

PSTricks already has a \multirput, which puts a box n times with a difference of dx and dy relative to each other. It is not possible to put it with a different distance from one point to the next. This is possible with \rmultiput:

\rmultiput* [Options] {*any material*}$(x_1,y_1)(x_2,y_2)\ldots((x_n,y_n))$




```
\psset{unit=0.75}
\begin{pspicture}(-4,-4)(4,4)
\rmultiput[rot=45]{\red\psscalebox{3}{\ding{250}}}%
   (-2,-4)(-2,-3)(-3,-3)(-2,-1)(0,0)(1,2)(1.5,3)(3,3)
\rmultiput[rot=90,ref=lC]{\blue\psscalebox{2}{\ding
    {253}}}%
   (-2,2.5)(-2,2.5)(-3,2.5)(-2,1)(1,-2)(1.5,-3)(3,-3)
\psgrid[subgriddiv=0,gridcolor=lightgray]
\end{pspicture}
```

4. \psVector: Drawing relative vector lines

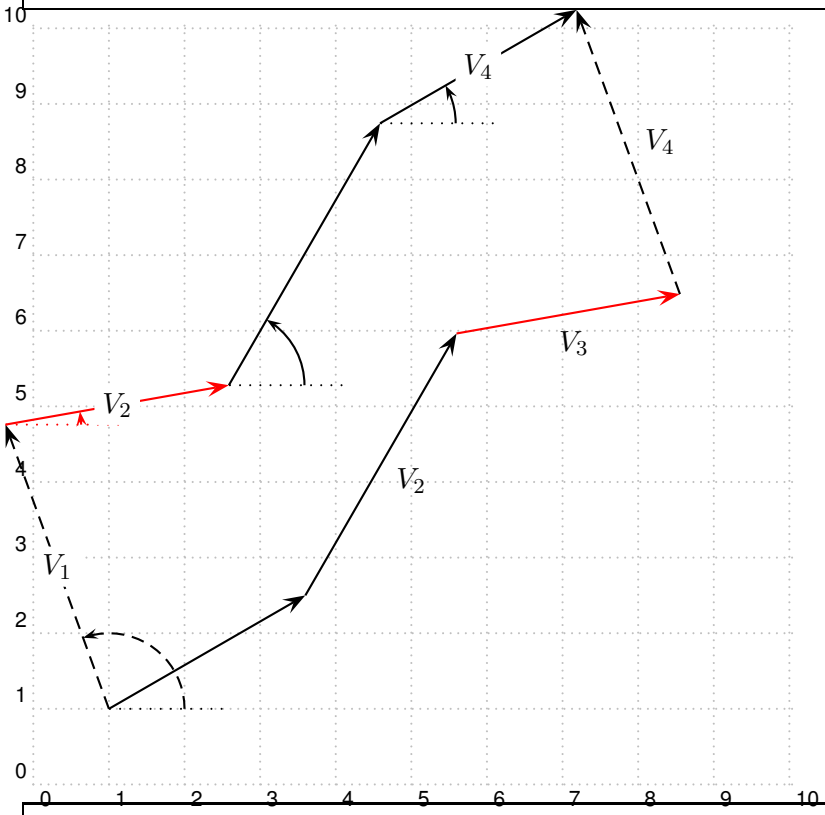
The new macros \psStartPoint and \psVector allow to draw a series of vectors which start point refers to the endpoint of the last drawn vector. The coordinates of the endpoint are *always* interpreted relative to the last the vector. The first vector refers to the coordinates set by \psStartPoint. With the boolean argument one can draw the horizontal angle of the vector.

```
\psVector [Options] <startpoint> (x1,y1)(x2,y2)...((xn,yn))
\psStartPoint [node basename] (x,y)
```

If the optional argument in angle braces is given then it will be the start point for the current vector and the next ones, until a new start point is defined or a new optional argument is used.

The style of the angle arc is saved in psMarkAngleStyle and the style for the horizontal line in psMarkAngleLineStyle and preset to

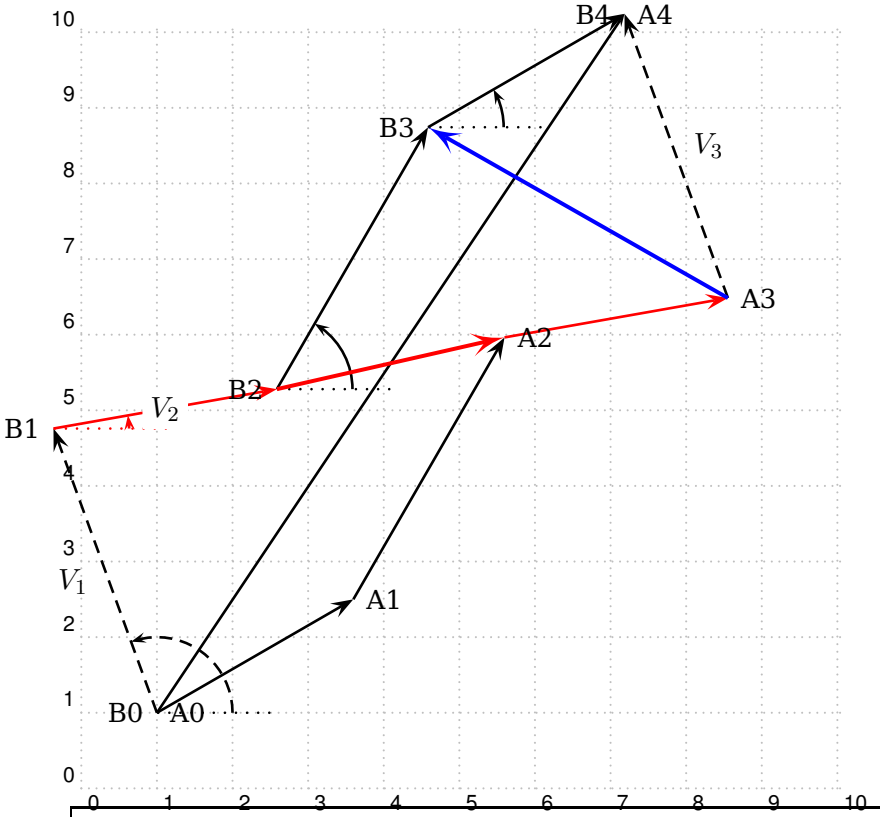
```
\newpsstyle{psMarkAngleStyle}{arrows=->,arrowsize=4pt}
\newpsstyle{psMarkAngleLineStyle}{linestyle=dotted}
```




```
\begin{pspicture}[showgrid](10,10)
\psVector<1,1>(3;30)(4;60)\nbput{$V_2$}
\psVector[linecolor=red](3;10)\nbput{$V_3$}
\psVector[linestyle=dashed](4;110)\nbput{$V_4$}
\psset{markAngle}
\psVector[linestyle=dashed]<1,1>(4;110)\ncput*{$V_1$}
\psVector[linecolor=red](3;10)\ncput*{$V_2$}
\psVector(4;60)(3;30)\ncput*{$V_4$}
\end{pspicture}
```

All end points of the vectors are saved in node names with the preset name Vector#, where # is the consecutive number of the nodes. Vector0 ist the starting point of the first \psVector. With the macro \psStartPoint one can set the starting point and with optional argument the name of

the nodes. Vector3 is the default node name of the endpoint of the third vector or the name of the starting point of the forth vector.



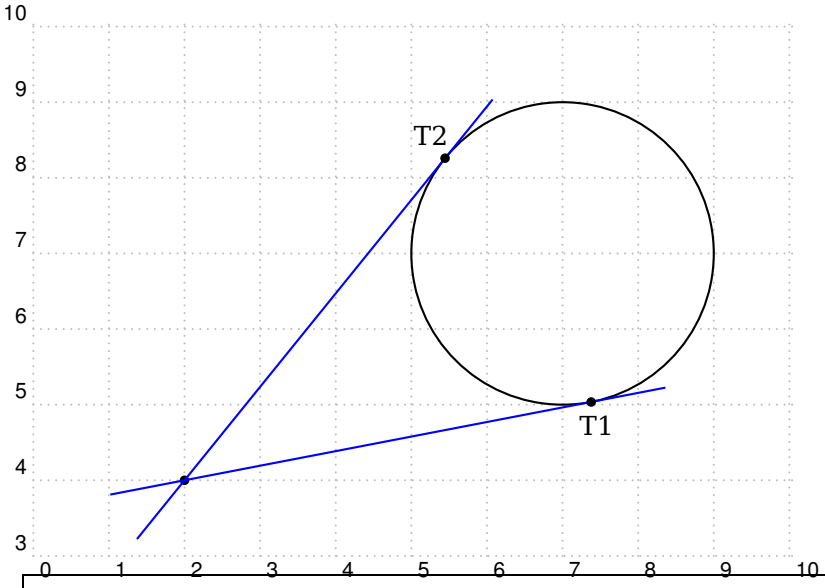

```
\begin{pspicture}[showgrid,linewidth=1pt](10,10.4)
 \psStartPoint[A](1,1)% nodes have the base name A
 \psVector(3;30)(4;60)\psVector[linecolor=red](3;10)
 \psVector[linestyle=dashed](4;110)\nbput{$V_3$}
 \psline{->}(A0)(A4)
 \psStartPoint[B](1,1)\psset{markAngle}% nodes have the base name B
 \psVector[linestyle=dashed](4;110)\naput{$V_1$}
 \psVector[linecolor=red](3;10)\ncput*{$V_2$}
 \psVector(4;60)(3;30)
 \psline[arrows=-D>,arrowscale=2,linewidth=1.5pt,linecolor=red](B2)(A2)
 \psline[arrows=-D>,arrowscale=2,linewidth=1.5pt,linecolor=blue](A3)(B3)
 \multido{\iA=0+1}{5}{\uput[0](A\iA){A\iA}\uput[180](B\iA){B\iA}}
 \end{pspicture}
```

5. \psCircleTangents: Calculating tangent lines of circles

The macro calculates the points on a circle where tangent lines from another point or another circle are drawn.

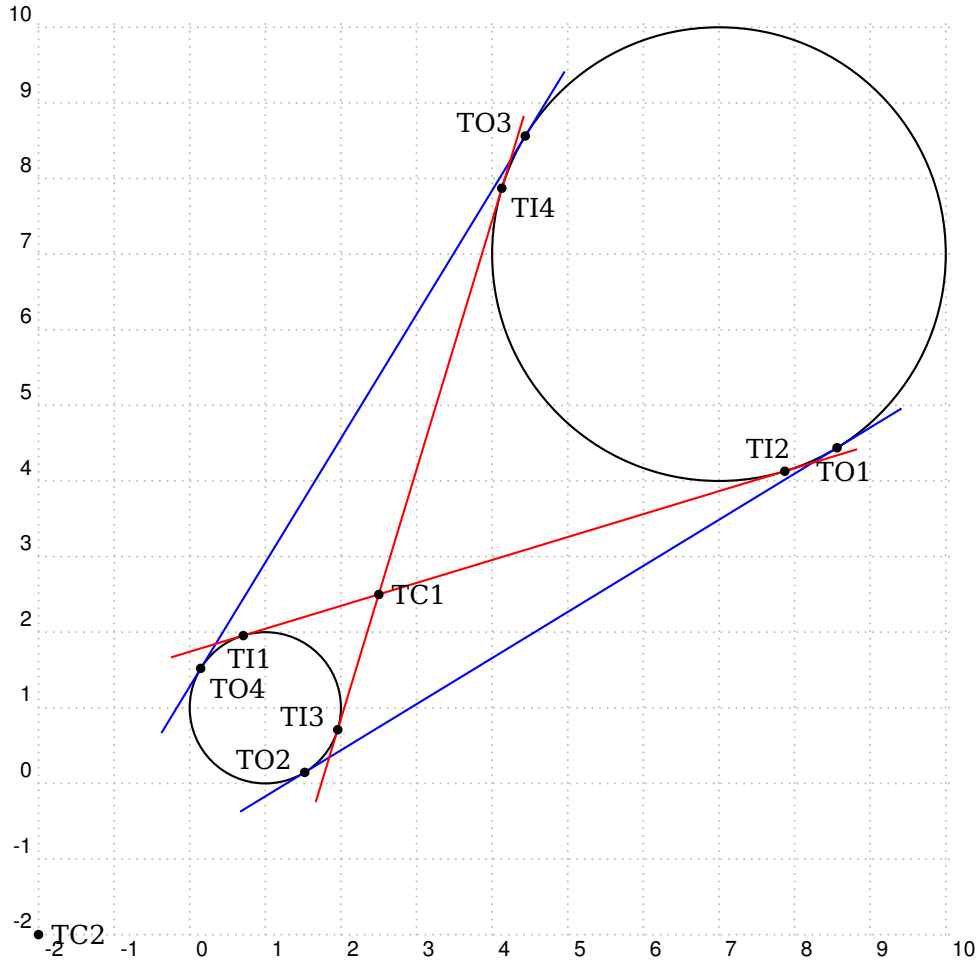
```
\psCircleTangents(x1,y1)(x2,y2){Radius}
\psCircleTangents(x1,y1){Radius}(x2,y2){Radius}
```

In the first case the coordinates of a point and the center and the radius of a circle must be given. The names of the calculates node names are `CircleT1` and `CircleT2`.




```
\begin{pspicture}[showgrid](0,3)(10,10)
\psdot(2,4)\pscircle(7,7){2}
\psCircleTangents(2,4)(7,7){2}
\pcline[nodesep=-1cm,linecolor=blue](2,4)(CircleT1)
\pcline[nodesep=-1cm,linecolor=blue](2,4)(CircleT2)
\psdots(CircleT1)(CircleT2)
\uput[-80](CircleT1){T1}\uput[115](CircleT2){T2}
\end{pspicture}
```

When using the other variant of the macro two circles must be given. The macro then defines ten nodes, named `CircleTC1` and `CircleTC2` for the two intersection points, `CircleTO1`, `CircleTO2`, `CircleTO3`, and `CircleTO4` for the four nodes of the outer tangent lines and `CircleTI1`, `CircleTI2`, `CircleTI3`, and `CircleTI4` for the four nodes of the inner tangent lines.



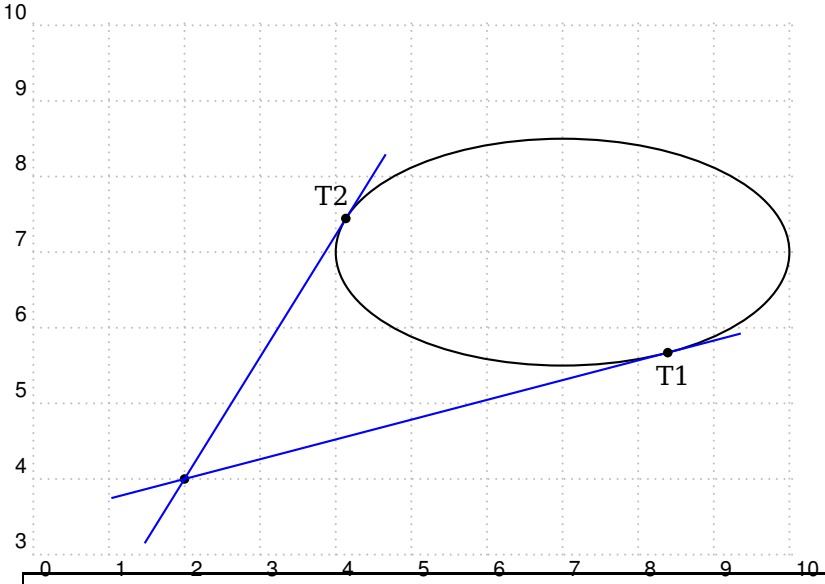

```
\begin{pspicture}[showgrid](-2,-2)(10,10)
\pscircle(1,1){1}\pscircle(7,7){3}
\psCircleTangents(1,1){1}(7,7){3}
\pcline[nodesep=-1cm,linecolor=blue](CircleTO1)(CircleTO2)
\pcline[nodesep=-1cm,linecolor=blue](CircleTO3)(CircleTO4)
\pcline[nodesep=-1cm,linecolor=red](CircleTI1)(CircleTI2)
\pcline[nodesep=-1cm,linecolor=red](CircleTI3)(CircleTI4)
\psdots(CircleTC1)\psdots(CircleTC2)%
  (CircleTO1)(CircleTO2)(CircleTO3)(CircleTO4)%
  (CircleTI1)(CircleTI2)(CircleTI3)(CircleTI4)%
\uput[0](CircleTC1){TC1}\uput[0](CircleTC2){TC2}
\uput[-80](CircleTI1){TI1}\uput[115](CircleTI2){TI2}
\uput[150](CircleTI3){TI3}\uput[-45](CircleTI4){TI4}
\uput[-80](CircleTO1){TO1}\uput[150](CircleTO2){TO2}
\uput[150](CircleTO3){TO3}\uput[-45](CircleTO4){TO4}
\end{pspicture}
```

6. \psEllipseTangents: Calculating tangent lines of an ellipse

The macro calculates the two points on an ellipse where tangent lines from an outside point are drawn.

\psEllipseTangents$(x_0,y_0)(a,b)(x_p,y_p)$

The first two pairs of coordinates are the same as the ones for the default ellipse. The names of the calculates node names are `EllipseT1` and `EllipseT2`.

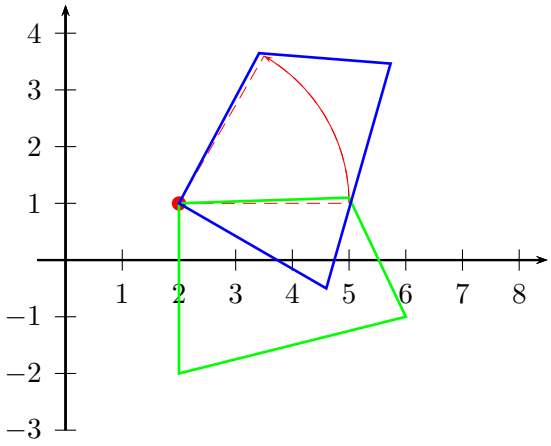



```
\begin{pspicture}[showgrid](0,3)(10,10)
\psdot(2,4)\psellipse(7,7)(3,1.5)
\psEllipseTangents(7,7)(3,1.5)(2,4)
\pcline[nodesep=-1cm,linecolor=blue](2,4)(EllipseT1)
\pcline[nodesep=-1cm,linecolor=blue](2,4)(EllipseT2)
\psdots(EllipseT1)(EllipseT2)
\uput[-80](EllipseT1){T1}\uput[115](EllipseT2){T2}
\end{pspicture}
```

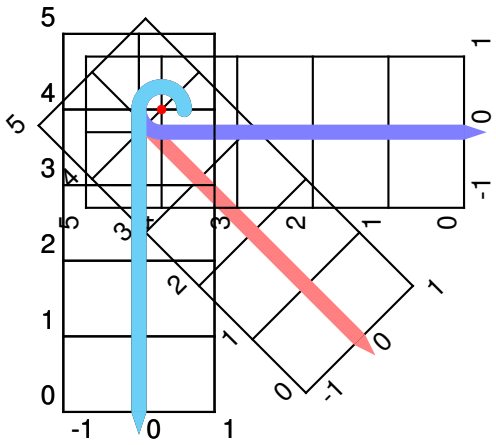
7. \psrotate: Rotating objects

\rput also has an optional argument for rotating objects, but it always depends on the \rput coordinates. With \psrotate the rotating center can be placed anywhere. The rotation is done with \pscustom, all optional arguments are only valid if they are part of the \pscustom macro.

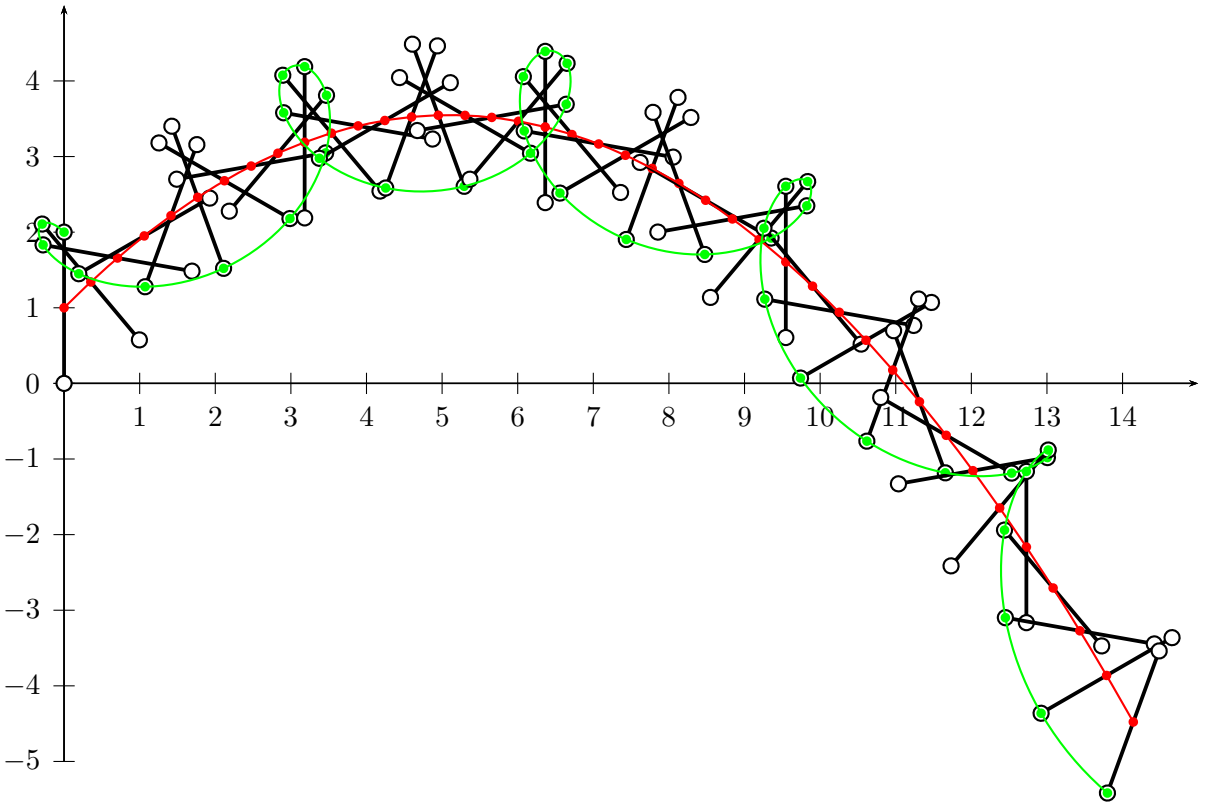
\psrotate [Options] (x, y) {*rot angle*} {*object*}




```
\psset{unit=0.75}
\begin{pspicture}(-0.5,-3.5)(8.5,4.5)
  \psaxes{->}(0,0)(-0.5,-3)(8.5,4.5)
  \psdots[linecolor=red,dotscale=1.5](2,1)
  \psarc[linecolor=red,linewidth=0.4pt,showpoints=
      true]
       {->}(2,1){3}{0}{60}
  \pspolygon[linecolor=green,linewidth=1pt](2,1)
      (5,1.1)(6,-1)(2,-2)
  \psrotate(2,1){60}{%
    \pspolygon[linecolor=blue,linewidth=1pt](2,1)
       (5,1.1)(6,-1)(2,-2)}
\end{pspicture}
```




```
\begin{pspicture}(-1,-1)(3,6)
\def\canne{% Idea by Manuel Luque
  \psgrid[subgriddiv=0](-1,0)(1,5)
  \pscustom[linewidth=2mm]{\psline(0,4)\psarcn(0.3,4)
      {0.3}{180}{360}}%
  \pscircle*(0.6,4){0.1}\pstriangle*(0,0)(0.2,-0.3)}
\def\Object{}
  \canne
  \psrotate(0.3,4){45}{\psset{linecolor=red!50}\canne}
  \psrotate(0.3,4){90}{\psset{linecolor=blue!50}\canne}
  \psrotate(0.3,4){360}{\psset{linecolor=cyan!50}\canne}
  \psdot[linecolor=red](0.3,4)
\end{pspicture}
```

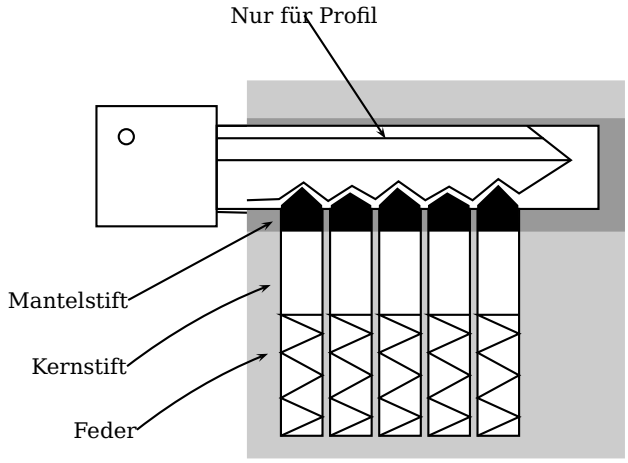



```
\begin{pspicture}(0,-6)(15,5)
\def\majorette{\psline[linewidth=0.5mm](0,2)% Idea by Manuel Luque
           \pscircle[fillstyle=solid]{0.1}
           \pscircle[fillstyle=solid](0,2){0.1}}
  \psaxes[linewidth=0.5pt]{->}(0,0)(0,-5)(15,5)
  \pstVerb{/V0 10 def /Alpha 45 def}% vitesse initiale, angle de lancement
  \multido{\nT=0.0+0.05,\iA=0+40}{41}{%
    \pstVerb{/nT \nT\space def}%
    \rput(!V0 Alpha cos mul nT mul -9.81 2 div nT dup mul mul V0 Alpha sin mul nT mul add){%
      \psrotate(0,1){\iA}{\majorette\psdot[linecolor=red](0,1)\psdot[linecolor=green](0,2)
          }}}
  \parametricplot[linecolor=red]{0}{2}{% trajectoire du milieu
    V0 Alpha cos mul t mul -9.81 2 div t dup mul mul V0 Alpha sin mul t mul add 1 add}
  \parametricplot[linecolor=green,plotpoints=360]{0}{2}{% d'une extremite
    V0 Alpha cos mul t mul 800 t mul sin sub % x(t)
    -9.81 2 div t dup mul mul V0 Alpha sin mul t mul add 1 add 800 t mul cos add }%y(t)
\end{pspicture}
```

8. \psComment: comments to a graphic

\psComment* [Options] {*arrows*} $(x_0,y_0)(x_1,y_1)${*Text*} [line macro] [put macro]

By default the macro uses the \ncline macro to draw a line from the first to the second point, it can be changed with the first additional optional argument. The label is put by default with \rput, which can be changed with the last optional argument. If this is used, then the line macro has also be defined, eg \psComment(A)(B){text}[\ncarc][\ncput} At least, leave the argument empty.




```
\SpecialCoor\newpsstyle{weiss}{fillstyle=solid,fillcolor=white}
\footnotesize\psset{unit=0.5cm,dimen=middle}
\begin{pspicture}(-12,-4)(6,10)
\psframe*[linecolor=black!20](-5,-3)(5,7) \psframe*[linecolor=black!40](-5,3)(5,6)
\pscircle(-8.19,5.51){0.2}
\psframe[fillcolor=white,fillstyle=solid](-5.8,3.6)(4.3,5.8)
\psframe(-8.98,3.14)(-5.8,6.32)
\multido{\rA=-4.1+1.3}{5}{\rput(\rA,-2.4){\psframe[style=weiss](1.1,6)
  \psline(0,0)(1.1,0.5)(0,1)(1.1,1.6)(0,2.2)(1.1,2.7)(0,3.2)(1.1,3.2)}}
\pspolygon*(-4.1,3.7)(-4.1,3)(-3,3)(-3.01,3.7)(-3.54,4.19)
\pspolygon*(1.09,3.7)(1.1,3)(2.2,3)(2.18,3.7)(1.65,4.24)
\pspolygon*(-2.78,3.7)(-2.8,3)(-1.7,3)(-1.71,3.7)(-2.27,4.04)
\pspolygon*(-1.51,3.7)(-1.5,3)(-0.4,3)(-0.41,3.7)(-1.02,4.17)
\pspolygon*(-0.21,3.7)(-0.2,3)(0.9,3)(0.89,3.7)(0.3,4.04)
\psline(-5,3.83)(-4.15,3.86)(-3.5,4.3)(-2.85,3.81)(-2.22,4.21)(-1.6,3.86)(-0.99,4.33)
     (-0.28,3.83)(0.35,4.19)(0.97,3.83)(1.65,4.39)(2.2,4.01)(3.57,4.89)(2.41,5.8)
  \psline(-5,5.8)(-5.78,5.8) \psline(-5.78,5.47)(2.85,5.47)
  \psline(-5.8,3.52)(-5,3.5) \psline(3.57,4.89)(-5.8,4.89)
  \psComment*[ref=r]{->}(-8.14,1.19)(-4.31,3.27){Mantelstift}
  \psComment*[ref=r]{->}(-8.17,-0.56)(-4.37,1.59){Kernstift}[\ncarc]
  \psComment*[ref=r]{->}(-7.91,-2.24)(-4.44,-0.23){Feder}[\ncarc]
  \psComment[npos=-0.1]{->}(-3.48,8.72)(-1.33,5.46){Nur f\"ur Profil}
\end{pspicture}
```

9. \psChart: a pie chart

\psChart [Options] {*comma separated value list*}{*comma separated value list*}{*radius*}

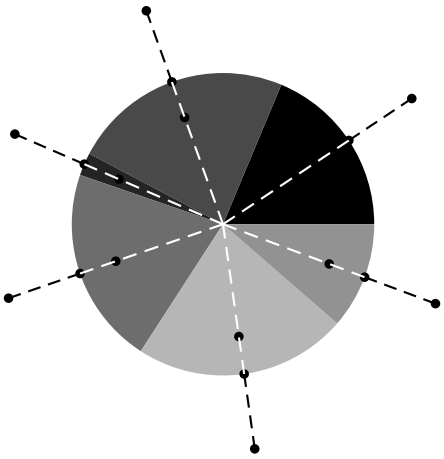
The special optional arguments for the \psChart macro are as follows:

name	*description*	*default*
chartSep	distance from the pie chart center to an outraged pie piece	10pt
chartColor	gray or colored pie (values are: gray or color)	gray
userColor	a comma separated list of user defined colors for the pie	{}
chartNodeI	the position of the inner node, relative to the radius	0.75
chartNodeO	the position of the outer node, relative to the radius	1.5

The first mandatory argument is the list of the values and may not be empty. The second one is a list of outraged pieces, numbered consecutively from 1 to up the total number of values. The list of user defined colors must be enclosed in braces!

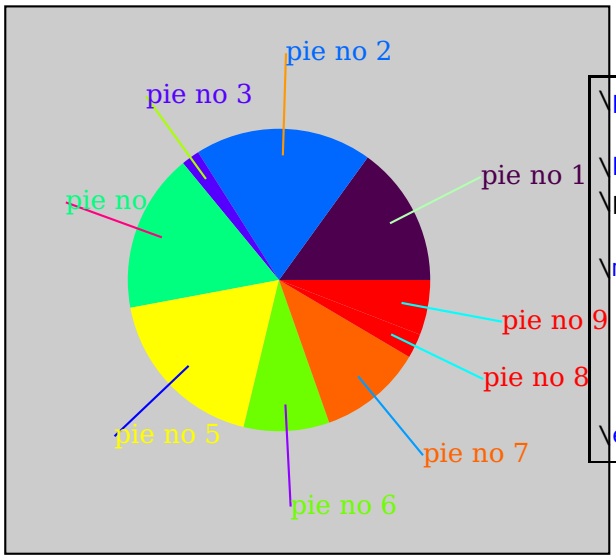
The macro \psChart defines for every value three nodes at the half angle and in distances from 0.75, 1, and 1.25 times of the radius from the origin. The nodes are named as psChartI?, psChart?, and psChartO?, where ? is the number of the pie. The letter I leads to the inner node and the letter O to the outer node. The distance can be changed with the optional arguments chartNodeI and chartNodeO in the usual way with \psset{chartNodeI=...,chartNodeO=...}.

The other one is the node on the circle line. The origin is by default (0,0). Moving the pie to another position can be done as usual with the \rput-macro. The used colors are named internally as chartFillColor? and can be used by the user for coloring lines or text.

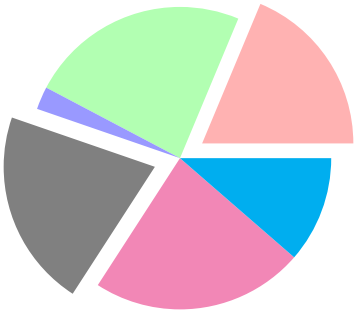


```
\begin{pspicture}(-3,-3)(3,3)
\psChart{ 23, 29, 3, 26, 28, 14 }{}{2}
\multido{\iA=1+1}{6}{%
  \psdot(psChart\iA)\psdot(psChartI\iA)\psdot(psChartO\iA)%
  \psline[linestyle=dashed,linecolor=white](psChart\iA)
  \psline[linestyle=dashed](psChart\iA)(psChartO\iA)}
\end{pspicture}
```

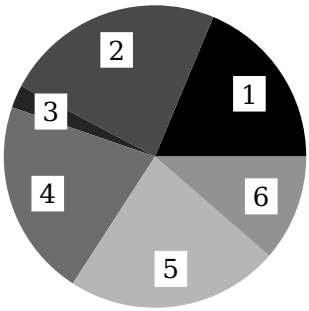

```
\begin{pspicture}(-3,-3)(3,3)
\psChart[chartColor=color]{45,90}{1}{2}
\ncline[linecolor=-chartFillColor1,
  nodesepB=-20pt]{psChartO1}{psChart1}
\rput[l](psChartO1){%
  \textcolor{chartFillColor1}{pie no 1}}
\ncline[linecolor=-chartFillColor2,
  nodesepB=-20pt]{psChartO2}{psChart2}
\rput[lt](psChartO2){%
  \textcolor{chartFillColor2}{pie no 2}}
\end{pspicture}
```



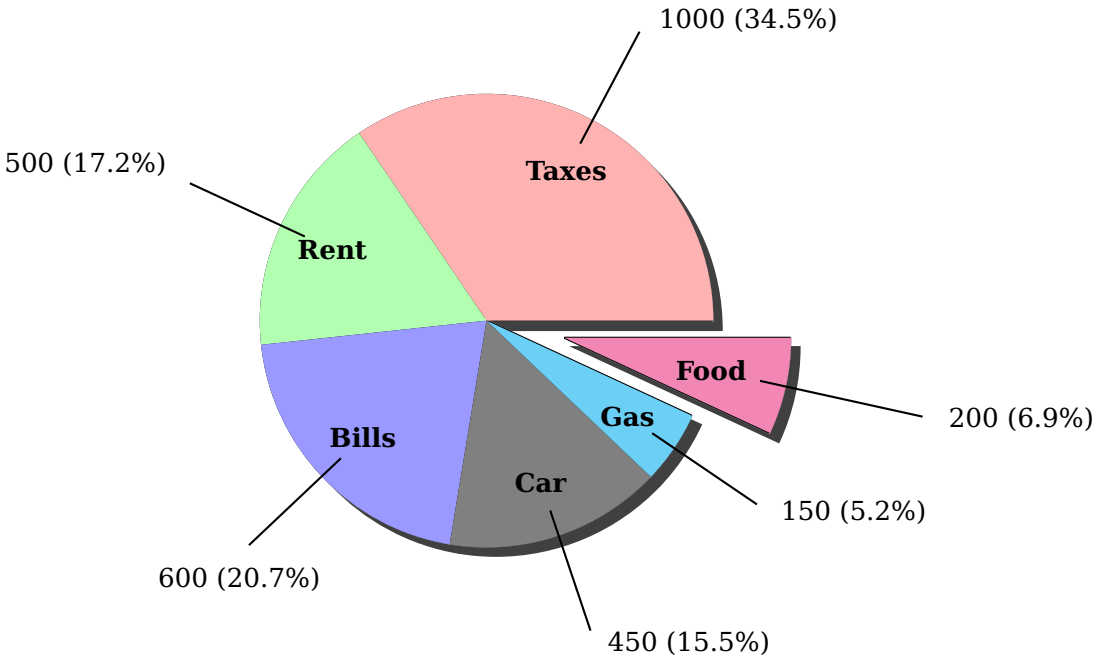

```
\psframebox[fillcolor=black!20,
 fillstyle=solid]{%
\begin{pspicture}(-3.5,-3.5)(4.25,3.5)
\psChart[chartColor=color]%
 {23, 29, 3, 26, 28, 14, 17, 4, 9}{}{2}
\multido{\iA=1+1}{9}{%
 \ncline[linecolor=-chartFillColor\iA,
   nodesepB=-10pt]{psChartO\iA}{psChart\iA}
 \rput[l](psChartO\iA){%
   \textcolor{chartFillColor\iA}{pie no \iA}}}
\end{pspicture}}
```



```
\begin{pspicture}(-3,-3)(3,3)
\psChart[userColor={red!30,green!30,
   blue!40,gray,magenta!60,cyan}]%
     { 23, 29, 3, 26, 28, 14 }{1,4}{2}
\end{pspicture}
```

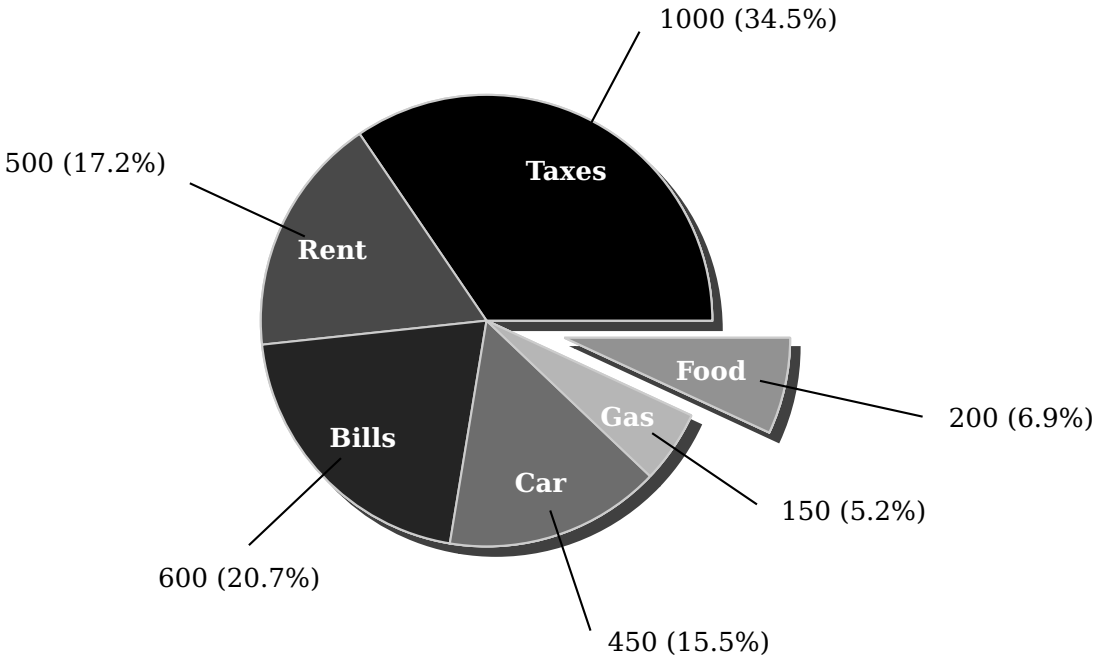



```
\begin{pspicture}(-3,-2.5)(3,2.5)
\psChart{ 23, 29, 3, 26, 28, 14 }{}{2}
\multido{\iA=1+1}{6}{\rput*(psChartI\iA){\iA}}
\end{pspicture}
```




```
\psset{unit=1.5}
\begin{pspicture}(-3,-3)(3,3)
\psChart[userColor={red!30,green!30,blue!40,gray,cyan!50,
   magenta!60,cyan},chartSep=30pt,shadow=true,shadowsize=5pt
       ]{34.5,17.2,20.7,15.5,5.2,6.9}{6}{2}
\psset{nodesepA=5pt,nodesepB=-10pt}
\ncline{psChartO1}{psChart1}\nput{0}{psChartO1}{1000 (34.5\%)}
\ncline{psChartO2}{psChart2}\nput{150}{psChartO2}{500 (17.2\%)}
\ncline{psChartO3}{psChart3}\nput{-90}{psChartO3}{600 (20.7\%)}
\ncline{psChartO4}{psChart4}\nput{0}{psChartO4}{450 (15.5\%)}
\ncline{psChartO5}{psChart5}\nput{0}{psChartO5}{150 (5.2\%)}
\ncline{psChartO6}{psChart6}\nput{0}{psChartO6}{200 (6.9\%)}
\bfseries%
\rput(psChartI1){Taxes}\rput(psChartI2){Rent}\rput(psChartI3){Bills}
\rput(psChartI4){Car}\rput(psChartI5){Gas}\rput(psChartI6){Food}
\end{pspicture}
```

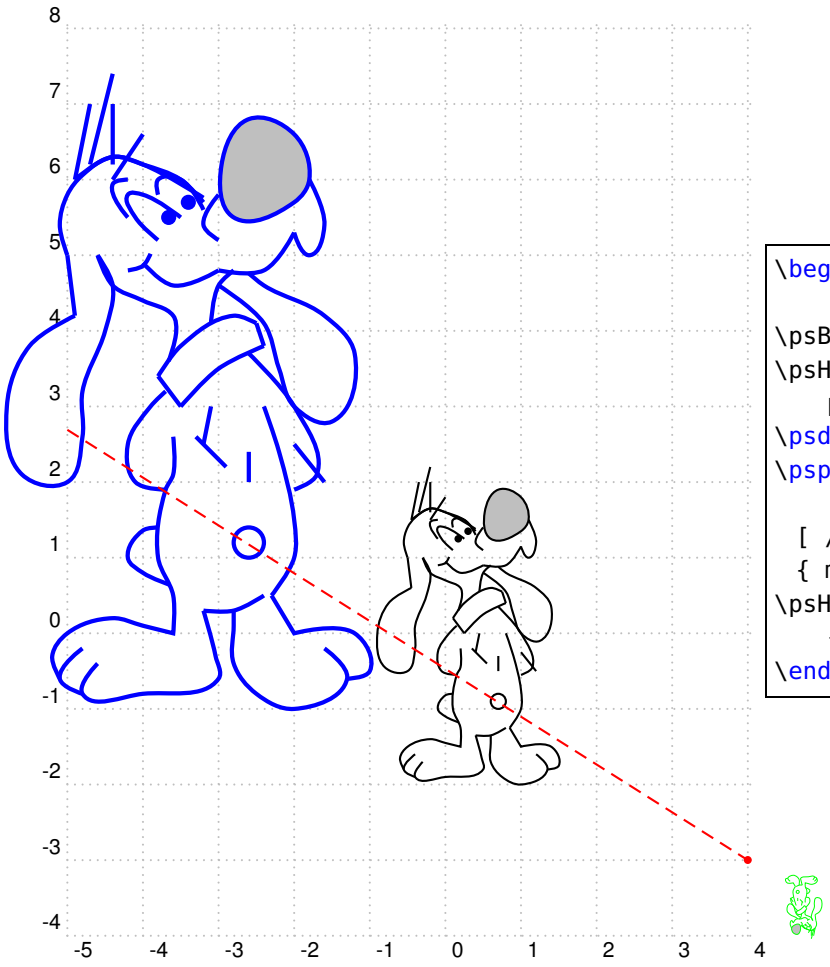
The linecolor of the pies is by default identical to the fillcolor. If you want another line color for all pies then use the optional argument `uselinecolor`. In this case the current setting of `linecolor` is taken into account:




```
\psset{unit=1.5}
\begin{pspicture}(-3,-3)(3,3)
\psChart[chartSep=30pt,shadow=true,shadowsize=5pt,
       uselinecolor,linecolor=black!20]{34.5,17.2,20.7,15.5,5.2,6.9}{6}{2}
\psset{nodesepA=5pt,nodesepB=-10pt}
\ncline{psChartO1}{psChart1}\nput{0}{psChartO1}{1000 (34.5\%)}
\ncline{psChartO2}{psChart2}\nput{150}{psChartO2}{500 (17.2\%)}
\ncline{psChartO3}{psChart3}\nput{-90}{psChartO3}{600 (20.7\%)}
\ncline{psChartO4}{psChart4}\nput{0}{psChartO4}{450 (15.5\%)}
\ncline{psChartO5}{psChart5}\nput{0}{psChartO5}{150 (5.2\%)}
\ncline{psChartO6}{psChart6}\nput{0}{psChartO6}{200 (6.9\%)}
\bfseries\color{white}%
\rput(psChartI1){Taxes}\rput(psChartI2){Rent}\rput(psChartI3){Bills}
\rput(psChartI4){Car}\rput(psChartI5){Gas}\rput(psChartI6){Food}
\end{pspicture}
```

10. \psHomothetie: central dilatation

\psHomothetie [Options] (*center*){*factor*}{*object*}

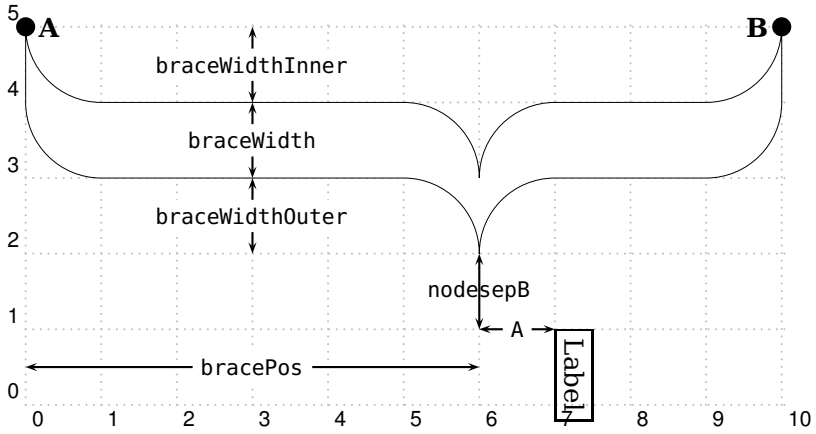



```
\begin{pspicture}[showgrid=true](-5,-4)
    (4,8)
\psBill% needs package pst-fun
\psHomothetie[linecolor=blue](4,-3){2}{\
    psBill}
\psdots[dotsize=3pt,linecolor=red](4,-3)
\psplot[linestyle=dashed,linecolor=red
    ]{-5}{4}%
  [ /m -3 -0.85 sub 4 0.6 sub div def ]
  { m x mul m 4 mul sub 3 sub }%
\psHomothetie[linecolor=green](4,-3)
    {-0.2}{\psBill}
\end{pspicture}
```

11. \psbrace

\psbrace * [Options] (A)(B){text}

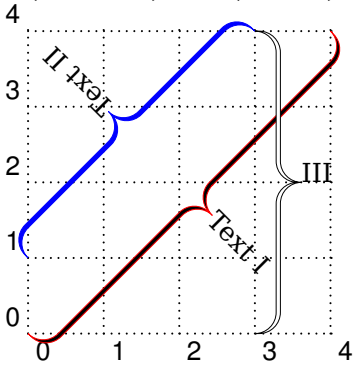
Additional to all other available options from pstricks or the other related packages, there are two new option, named braceWidth and bracePos. All important ones are shown in the following graphics and table.



A positive value for nodesepA and nodesepB shifts the label to the upper right and a negative value to the lower left. This does not depends on the value for the rotating of the label!

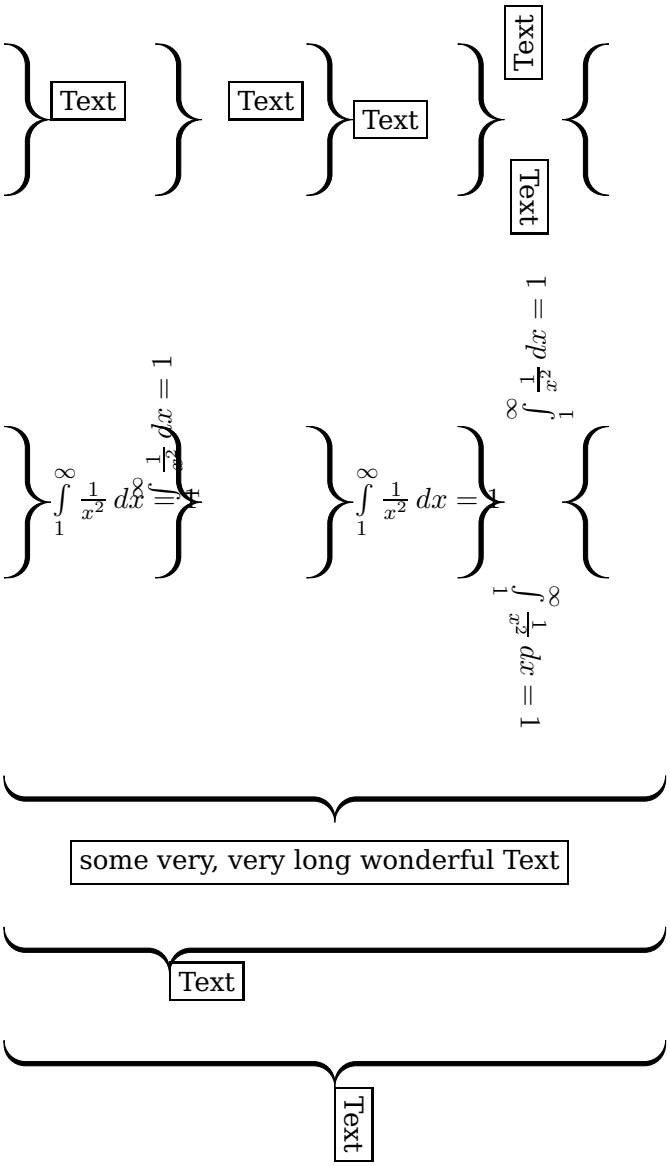
name	meaning
braceWidth	default is \pslinewidth
braceWidthInner	default is 10\pslinewidth
braceWidthOuter	default is 10\pslinewidth
bracePos	relative position (default is 0.5)
nodesepA	x-separation (default is $0pt$)
nodesepB	y-separation (default is $0pt$)
rot	additional rotating for the text (default is 0)
ref	reference point for the text (default is c)
fillcolor	default is black

By default the text is written perpendicular to the brace line and can be changed with the pstricks option rot=... The text parameter can take any object and may also be empty. The reference point can be any value of the combination of l (left) or r (right) and b (bottom) or B (Baseline) or C (center) or t (top), where the default is c, the center of the object.




```
\begin{pspicture}(4,4)
\psgrid[subgriddiv=0,griddots=10]
\pnode(0,0){A}
\pnode(4,4){B}
\psbrace[linecolor=red,ref=lC](A)(B){Text I}
\psbrace*[linecolor=blue,ref=lC](3,4)(0,1){Text II}
\psbrace[fillcolor=white](3,0)(3,4){III}
\end{pspicture}
```

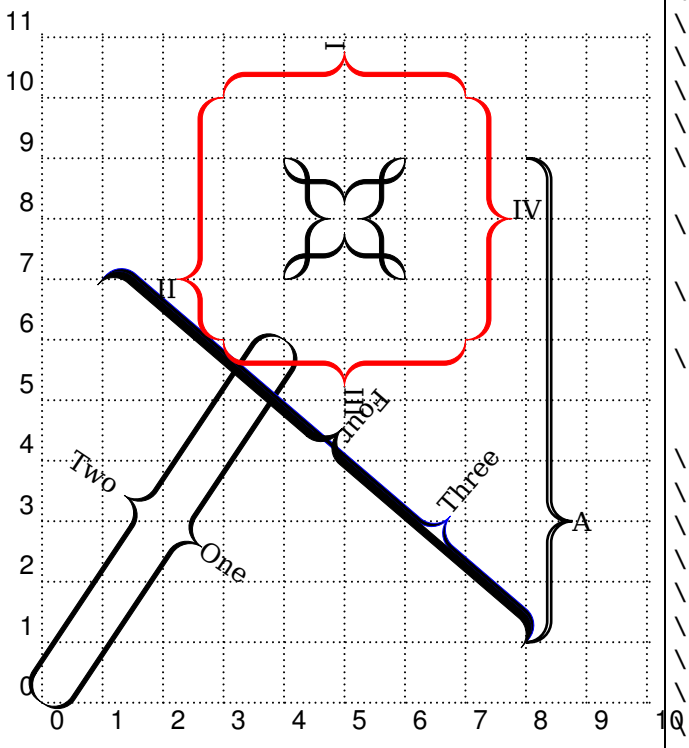
The option \specialCoor is enabled, so that all types of coordinates are possible, (nodename), (x, y), $(nodeA|nodeB)$, ... The star version fills the inner of the brace with the current linecolor. With the fillcolor white or any other background color the brace can be "'unfilled"'.



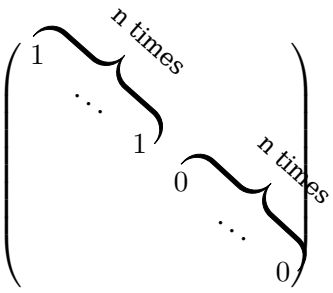

```
\begin{pspicture}(8,2.5)
\psbrace(0,0)(0,2){\fbox{Text}}%
\psbrace[nodesepA=10pt](2,0)(2,2){\fbox{Text
    }}
\psbrace[ref=lC](4,0)(4,2){\fbox{Text}}
\psbrace[ref=lt,rot=90,nodesepB=-15pt](6,0)
    (6,2){\fbox{Text}}
\psbrace[ref=lt,rot=90,nodesepA=-5pt,
    nodesepB=15pt](8,2)(8,0){\fbox{Text}}
\end{pspicture}
```

```
\def\someMath{$\int\limits_1^{\infty}\frac
    {1}{x^2}\,dx=1$}
\begin{pspicture}(8,2.5)
\psbrace[ref=lC](0,0)(0,2){\someMath}%
\psbrace[rot=90](2,0)(2,2){\someMath}
\psbrace[ref=lC](4,0)(4,2){\someMath}
\psbrace[ref=lt,rot=90,nodesepB=-30pt](6,0)
    (6,2){\someMath}
\psbrace[ref=lt,rot=90,nodesepB=30pt](8,2)
    (8,0){\someMath}
\end{pspicture}
```

```
\begin{pspicture}(\linewidth,5)
\psbrace(0,0.5)(\linewidth,0.5){\fbox{Text}}
    %
\psbrace[bracePos=0.25,nodesepB=10pt,rot
    =90](0,2)(\linewidth,2){\fbox{Text}}
\psbrace[ref=lC,nodesepA=-3.5cm,nodesepB=15
    pt,rot=90](0,4)(\linewidth,4){%
  \fbox{some very, very long wonderful Text
     }}
\end{pspicture}
```




```
\psset{unit=0.8}
\begin{pspicture}(10,11)
\psgrid[subgriddiv=0,griddots=10]
\pnode(0,0){A}
\pnode(4,6){B}
\psbrace[ref=lC](A)(B){One}
\psbrace[rot=180,nodesepA=-5pt,ref=rb](B)(A){Two
    }
\psbrace[linecolor=blue,bracePos=0.25,ref=lB
    ](8,1)(1,7){Three}
\psbrace[braceWidth=-1mm,rot=180,ref=rB](8,1)
    (1,7){Four}
\psbrace*[linearc=0.5,fillstyle=none,linewidth=1
    pt,braceWidth=1.5pt,
  bracePos=0.25,ref=lC](8,1)(8,9){A}
\psbrace(4,9)(6,9){}
\psbrace(6,9)(6,7){}
\psbrace(6,7)(4,7){}
\psbrace(4,7)(4,9){}
\psset{linecolor=red}
\psbrace*[ref=lb](7,10)(3,10){I}
\psbrace*[ref=lb,bracePos=0.75](3,10)(3,6){II}
\psbrace*[ref=lb](3,6)(7,6){III}
\psbrace*[ref=lb](7,6)(7,10){IV}
\end{pspicture}
```




```
\[
\begin{pmatrix}
   \Rnode[vref=2ex]{A}{~1} \\
   & \ddots \\
   && \Rnode[href=2]{B}{1} \\
   &&& \Rnode[vref=2ex]{C}{0} \\
   &&&& \ddots \\
   &&&&& \Rnode[href=2]{D}{0}~ \\
\end{pmatrix}
\]
\psbrace[rot=-90,nodesepB=-0.5,nodesepA=-0.2](B)(A){\small n
    times}
\psbrace[rot=-90,nodesepB=-0.5,nodesepA=-0.2](D)(C){\small n
    times}
```

It is also possible to put a vertical brace around a default paragraph. This works by setting two invisible nodes at the beginning and the end of the paragraph. Indentation is possible with a minipage.

Some nonsense text, which is nothing more than nonsense. Some nonsense text, which is nothing more than nonsense.

Some nonsense text, which is nothing more than nonsense. Some nonsense text, which is nothing more than nonsense. Some nonsense text, which is nothing more than nonsense. Some nonsense text, which is nothing more than nonsense. Some nonsense text, which is nothing more than nonsense. Some nonsense text, which is nothing more than nonsense. Some nonsense text, which is nothing more than nonsense. Some nonsense text, which is nothing more than nonsense.

Some nonsense text, which is nothing more than nonsense. Some nonsense text, which is nothing more than nonsense.

Some nonsense text, which is nothing more than nonsense. Some nonsense text, which is nothing more than nonsense. Some nonsense text, which is nothing more than nonsense. Some nonsense text, which is nothing more than nonsense. Some nonsense text, which is nothing more than nonsense. Some nonsense text, which is nothing more than nonsense. Some nonsense text, which is nothing more than nonsense. Some nonsense text, which is nothing more than nonsense.

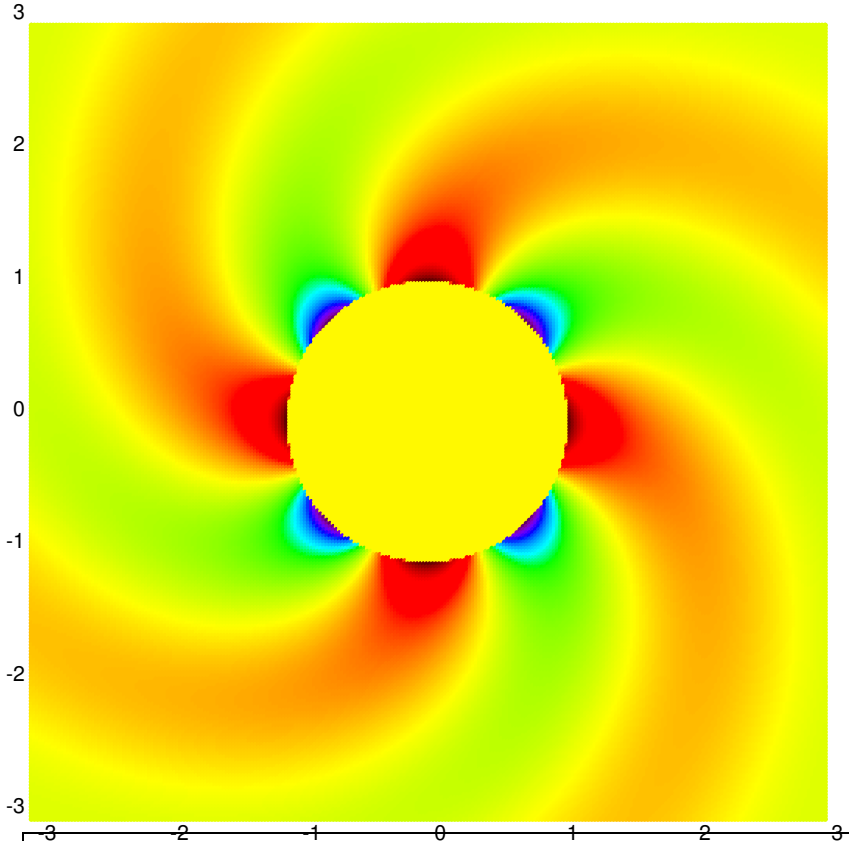
```
Some nonsense text, which is nothing more than nonsense.
Some nonsense text, which is nothing more than nonsense.

\noindent\rnode{A}{}

\vspace*{-1ex}
Some nonsense text, which is nothing more than nonsense.
Some nonsense text, which is nothing more than nonsense.
Some nonsense text, which is nothing more than nonsense.
Some nonsense text, which is nothing more than nonsense.
Some nonsense text, which is nothing more than nonsense.
Some nonsense text, which is nothing more than nonsense.
Some nonsense text, which is nothing more than nonsense.
Some nonsense text, which is nothing more than nonsense.

\vspace*{-2ex}\noindent\rnode{B}{}\psbrace[linecolor=red](A)(B){}

Some nonsense text, which is nothing more than nonsense.
Some nonsense text, which is nothing more than nonsense.

\medskip\hfill\begin{minipage}{0.95\linewidth}
\noindent\rnode{A}{}

\vspace*{-1ex}
Some nonsense text, which is nothing more than nonsense.
Some nonsense text, which is nothing more than nonsense.
Some nonsense text, which is nothing more than nonsense.
Some nonsense text, which is nothing more than nonsense.
Some nonsense text, which is nothing more than nonsense.
Some nonsense text, which is nothing more than nonsense.
Some nonsense text, which is nothing more than nonsense.
Some nonsense text, which is nothing more than nonsense.

\vspace*{-2ex}\noindent\rnode{B}{}\psbrace[linecolor=red](A)(B){}
\end{minipage}
```

12. Contour plots

Contour plots are 2D-images but representing 3D data. The color is the representation of the z coordinate. It is only possible to plot data files which must have the following structure:

```
/contourdata [[
x y z
x y z
][
x y z
...
][
...
] def
```

it is an PostScript array of array. The Perl script http://tug.org/pstricks/pst-plot/3D/MakeData.pl allows to plot a file of the 3D-data of a mathematical function $z = f(x, y)$ and the Perl script http://tug.org/pstricks/pst-plot/3D/PrepareData.pl prepares the data file into the above structure for using it with the example file.




```
\psset{unit=1.75cm}
\begin{pspicture}[showgrid](-3,-3)(3,3)
\pstContour[colored]{data/contourN.data}
\end{pspicture}
```

Important is the option `-dNOSAFER` for the `ps2pdf` run, otherwise Ghostscript didn't allow the run of external data files. Important optional arguments are `colored`, `colorOffset`, and `colSteps`.

13. Random dots

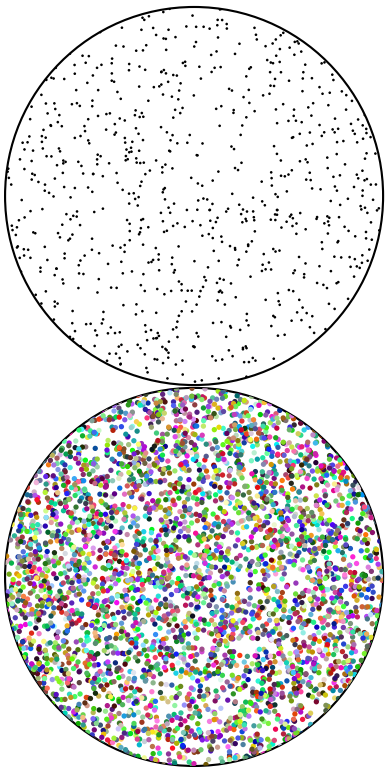
The syntax of the new macro \psRandom is:

```
\psRandom [Options] {}
\psRandom [Options] (x_Min, y_Min) (x_Max, y_Max) {clip path} \psRandomPointArea [Options] {No dots}
```

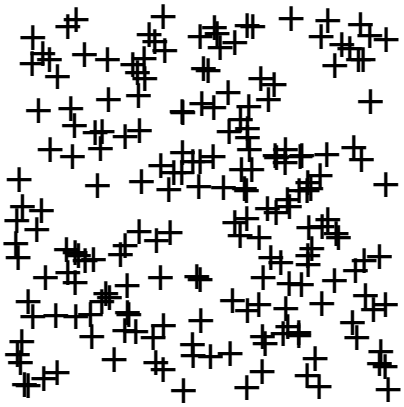
13.1. Simple random dots

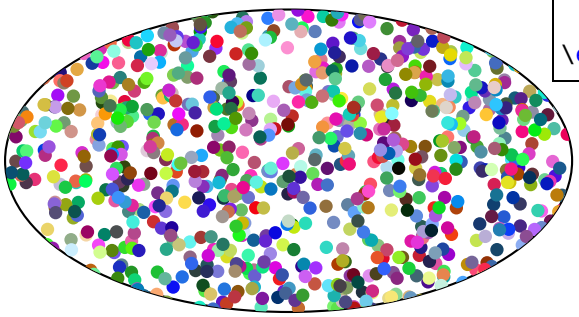
If there is no area for the dots defined, then (0,0)(1,1) in the current scale setting is used for placing the dots. If there is only one (x_{Max}, y_{Max}) defined, then (0,0) is used for the other point. This area should be greater than the clipping path to be sure that the dots are placed over the full area. The clipping path can be everything. If no clipping path is given, then the frame (0,0)(1,1) in user coordinates is used. The new options are:

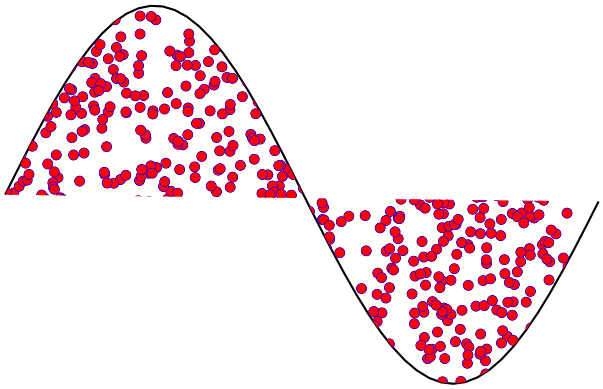
name	default	
randomPoints	1000	number of random dots
randInit	rrand	initial value for the generator
color	false	random color



```
\psset{unit=5cm}
\begin{pspicture}(1,1)
  \psRandom[dotsize=1pt,fillstyle=solid](1,1){\pscircle
      (0.5,0.5){0.5}}
\end{pspicture}
\begin{pspicture}(1,1)
  \psRandom[randInit=42,dotsize=2pt,randomPoints=5000,color,%
     fillstyle=solid](1,1){\pscircle(0.5,0.5){0.5}}
\end{pspicture}
```





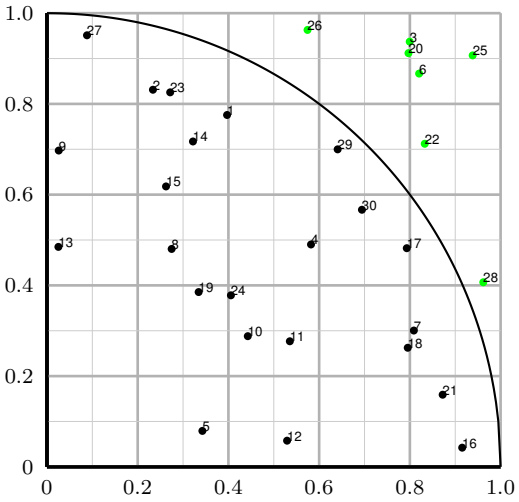


```
\psset{unit=5cm}
\begin{pspicture}(1,1)
  \psRandom[randomPoints=200,dotsize=8pt,dotstyle
      =+]{}
\end{pspicture}
\begin{pspicture}(1.5,1)
  \psRandom[dotsize=5pt,color](0,0)(1.5,0.8){\
      psellipse(0.75,0.4)(0.75,0.4)}
\end{pspicture}
```

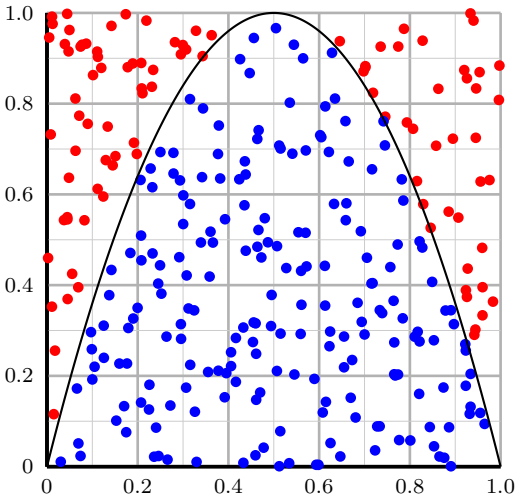
```
\psset{unit=2.5cm}
\begin{pspicture}(0,-1)(3,1)
  \psRandom[dotsize=4pt,dotstyle=o,linecolor
      =blue,fillcolor=red,%
    fillstyle=solid,randomPoints=1000]%
     (0,-1)(3,1){\psplot{0}{3.14}{ x 114 mul
          sin }}
\end{pspicture}
```

13.2. Simple random dots devided by a function

The predefined colors are blue (lower part) and red (upper part).



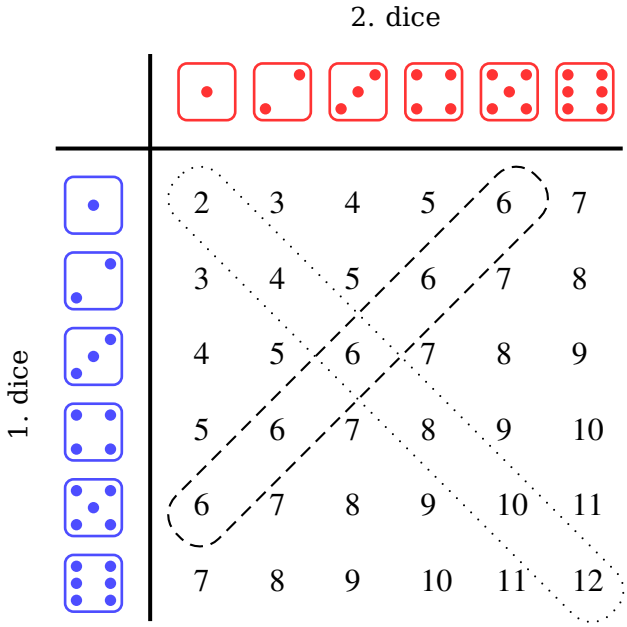

```
\psset{unit=6cm}
\begin{pspicture}(-0.2,-0.1)(1.1,1.2)
\psaxes[linewidth=1.25pt,Dx=0.2,Dy=0.2,
  labelFontSize=\scriptstyle,ticksize=0 1,subticks=2,
  subticksize=1,tickwidth=1pt,tickcolor=black!30,subtickcolor=black!20](0,0)(1,1)%
\psRandomPointArea[radius=1.5pt,countDots,algebraic,
  fillcolorA=black,fillcolorB=green]{30}{sqrt(1-x^2)}%
\psplot[plotpoints=200]{0}{1}{1 x dup mul sub sqrt}
\end{pspicture}
```




```
\psset{unit=6cm}
\begin{pspicture}(-0.2,-0.1)(1.1,1.2)
\psaxes[linewidth=1.25pt,Dx=0.2,Dy=0.2,
  labelFontSize=\scriptstyle,ticksize=0 1,subticks=2,
  subticksize=1,tickwidth=1pt,tickcolor=black!30,subtickcolor=black!20](0,0)(1,1)%
\psRandomPointArea[algebraic]{300}{-4*(x^2-x)}%
\psplot[plotpoints=200,algebraic]{0}{1}{-4*(x^2-x)}
\end{pspicture}
```

14. \psDice

\psdice creates the view of a dice. The number on the dice is the only parameter. The optional parameters, like the color can be used as usual. The macro is a box of dimension zero and is placed at the current point. Use the \rput macro to place it anywhere. The optional argument unit can be used to scale the dice. the default size of the dice $1\text{cm} \times 1\text{cm}$.

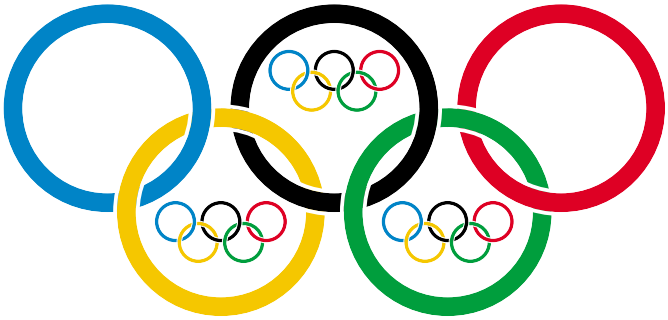



```
\begin{pspicture}(-1,-1)(8,8)
\multido{\iA=1+1}{6}{%
  \rput(\iA,7.5){\Huge\psdice[unit=0.75,linecolor=red!80]{\iA}}
  \rput(! -0.5 7 \iA\space sub){\Huge\psdice[unit=0.75,linecolor=blue!70]{\iA}}%
  \multido{\iB=1+1}{6}{%
    \rput(! \iA\space 7 \iB\space sub){%
      \rnode[c]{p\iA\iB}{\makebox[1em][l]{\strut\psPrintValue[fontscale=12]{\iA\space \iB\
          space add}}}%
}}}
\ncbox[linearc=0.35,nodesep=0.2,linestyle=dotted]{p11}{p66}
\ncbox[linearc=0.35,nodesep=0.2,linestyle=dashed]{p15}{p51}
\rput{90}(-1.5,3.5){1. dice}
\rput{0}(3.5,8.5){2. dice}
\psline[linewidth=1.5pt](0.25,0.5)(0.25,8)
\psline[linewidth=1.5pt](-1,6.75)(6.5,6.75)
\end{pspicture}
```

15. Olympic Rings

The colors for the Rings are defined as `OlympicBlue`, `OlympicRed`, `OlympicGreen`, and `OlympicYellow` and can be overwritten by the user. The only valid optional argument is `psscale` for scaling.

`\psOlympicRings [Options]` (x_1, y_1)



```
\begin{pspicture}(-4.5,-3)(4.5,1.5)
\psOlympicRings(0,0)
\psOlympicRings[psscale=0.2](1.5,-1.5)
\psOlympicRings[psscale=0.2](-1.5,-1.5)
\psOlympicRings[psscale=0.2](0,0.5)
\end{pspicture}
```

16. \psFormatInt

There exist some packages and a lot of code to format an integer like $1\,000\,000$ or $1,234,567$ (in Europe $1.234.567$). But all packages expect a real number as argument and cannot handle macros as an argument. For this case `pstricks-add` has a macro `\psFormatInt` which can handle both:

1,234,567
1,234,567
1.234.567
1·234·567
965,432

```
\psFormatInt{1234567}\\
\psFormatInt[intSeparator={,}]{1234567}\\
\psFormatInt[intSeparator=.]{1234567}\\
\psFormatInt[intSeparator=$\cdot$]{1234567}\\
\def\temp{965432}
\psFormatInt{\temp}
```

With the option `intSeparator` the symbol can be changed to any any non-number character.

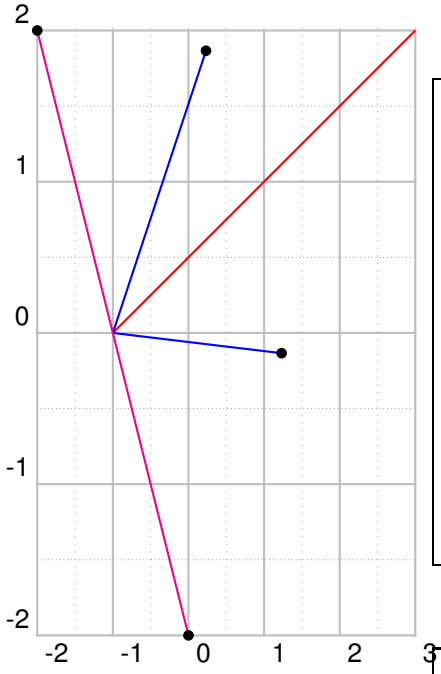
17. \psRelLine

With this macro it is possible to plot lines relative to a given one. Parameter are the angle and the length factor:

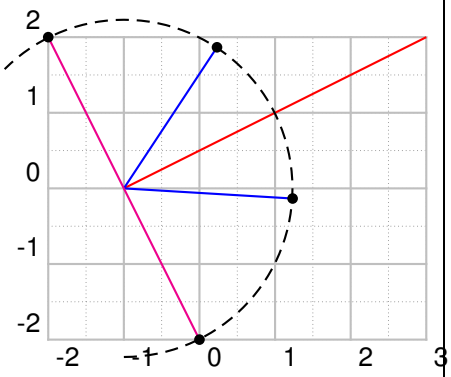
```
\psRelLine(P0)(P1){length factor}{<end node name>}
\psRelLine [{arrows}] (P0)(P1){length factor}{end node name}
\psRelLine [Options] (P0)(P1){length factor}{end node name}
\psRelLine [Options] [{arrows}] (P0)(P1){length factor}{end node name}
```

The length factor relates to the distance $\overline{P_0P_1}$ and the end node name must be a valid nodename and shouldn't contain any of the special PostScript characters. There are two valid options which are described in the foregoing section for `\psRelNode`.

The following two figures show the same, the first one with a scaling different to $1:1$, this is the reason why the end points are on an ellipse and not on a circle like in the second figure.

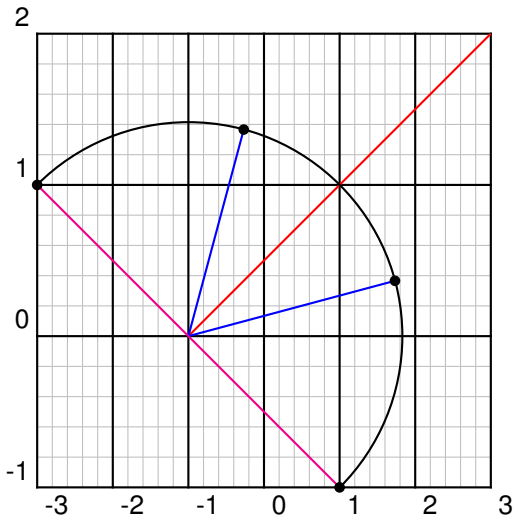



```
\psset{yunit=2,xunit=1}
\begin{pspicture}(-2,-2)(3,2)
\psgrid[subgriddiv=2,subgriddots=10,gridcolor=lightgray]
\pnode(-1,0){A}\pnode(3,2){B}
\psline[linecolor=red](A)(B)
\psRelLine[linecolor=blue,angle=30](-1,0)(B){0.5}{EndNode}
\qdisk(EndNode){2pt}
\psRelLine[linecolor=blue,angle=-30](A)(B){0.5}{EndNode}
\qdisk(EndNode){2pt}
\psRelLine[linecolor=magenta,angle=90](-1,0)(3,2){0.5}{EndNode}
\qdisk(EndNode){2pt}
\psRelLine[linecolor=magenta,angle=-90](A)(B){0.5}{EndNode}
\qdisk(EndNode){2pt}
\end{pspicture}
```



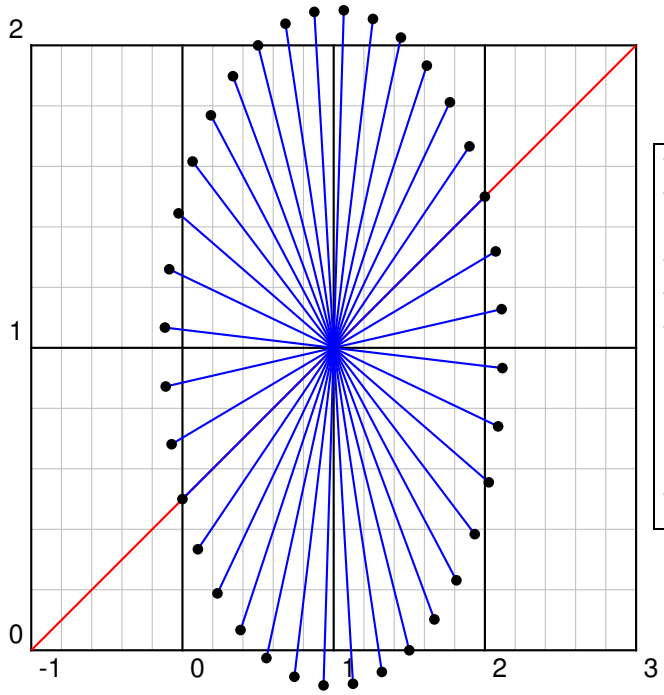

```
\begin{pspicture}(-2,-2)(3,2)
\psgrid[subgriddiv=2,subgriddots=10,gridcolor=lightgray]
\pnode(-1,0){A}\pnode(3,2){B}
\psline[linecolor=red](A)(B)
\psarc[linestyle=dashed](A){2.23}{-90}{135}
\psRelLine[linecolor=blue,angle=30](-1,0)(B){0.5}{EndNode}
\qdisk(EndNode){2pt}
\psRelLine[linecolor=blue,angle=-30](A)(B){0.5}{EndNode}
\qdisk(EndNode){2pt}
\psRelLine[linecolor=magenta,angle=90](-1,0)(3,2){0.5}{EndNode}
\qdisk(EndNode){2pt}
\psRelLine[linecolor=magenta,angle=-90](A)(B){0.5}{EndNode}
\qdisk(EndNode){2pt}
\end{pspicture}
```

The following figure has also a different scaling, but has set the option `trueAngle`, all angles refer to "what you see".

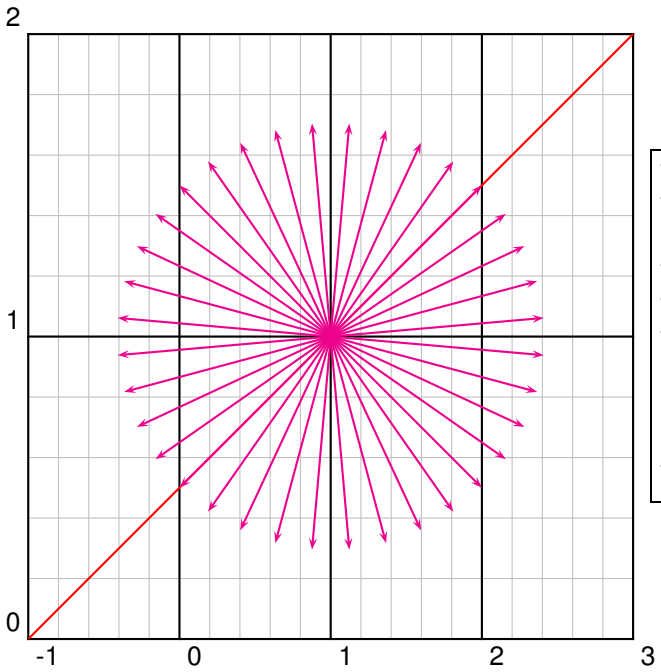



```
\psset{yunit=2,xunit=1}
\begin{pspicture}(-3,-1)(3,2)\psgrid[subgridcolor=
    lightgray]
\pnode(-1,0){A}\pnode(3,2){B}
\psline[linecolor=red](A)(B)
\psarc(A){2.83}{-45}{135}
\psRelLine[linecolor=blue,angle=30,trueAngle](A)(B)
    {0.5}{EndNode}
\qdisk(EndNode){2pt}
\psRelLine[linecolor=blue,angle=-30,trueAngle](A)(B)
    {0.5}{EndNode}
\qdisk(EndNode){2pt}
\psRelLine[linecolor=magenta,angle=90,trueAngle](A)(B)
    {0.5}{EndNode}
\qdisk(EndNode){2pt}
\psRelLine[linecolor=magenta,angle=-90,trueAngle](A)(B)
    {0.5}{EndNode}
\qdisk(EndNode){2pt}
\end{pspicture}
```

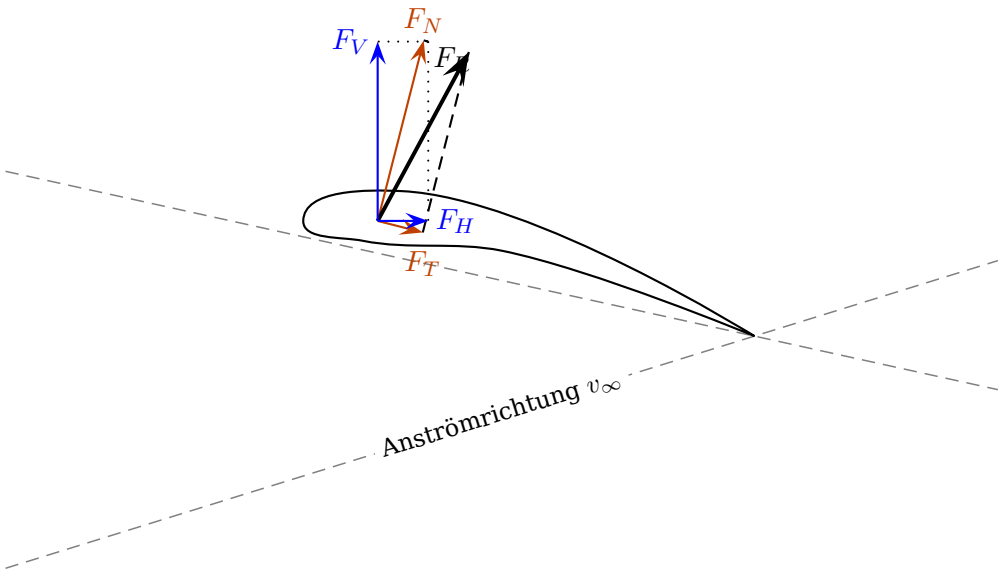
Two examples using `\multido` to show the behaviour of the options `trueAngle` and `angle`.




```
\psset{yunit=4,xunit=2}
\begin{pspicture}(-1,0)(3,2)\psgrid[subgridcolor
    =lightgray]
\pnode(-1,0){A}\pnode(1,1){B}
\psline[linecolor=red](A)(3,2)
\multido{\iA=0+10}{36}{%
  \psRelLine[linecolor=blue,angle=\iA](B)(A)
      {-0.5}{EndNode}
  \qdisk(EndNode){2pt}
}
\end{pspicture}
```




```
\psset{yunit=4,xunit=2}
\begin{pspicture}(-1,0)(3,2)\psgrid[subgridcolor
    =lightgray]
\pnode(-1,0){A}\pnode(1,1){B}
\psline[linecolor=red](A)(3,2)
\multido{\iA=0+10}{36}{%
  \psRelLine[linecolor=magenta,angle=\iA,
     trueAngle]{->}(B)(A){-0.5}{EndNode}
}
\end{pspicture}
```




```
\psset{xunit=0.75\linewidth,yunit=0.75\linewidth,trueAngle}%
\end{center}
\begin{pspicture}(1,0.6)%\psgrid
  \pnode(.3,.35){Vk} \pnode(.375,.35){D} \pnode(0,.4){DST1} \pnode(1,.18){DST2}
  \pnode(0,.1){A1} \pnode(1,.31){A1}
  { \psset{linewidth=.02,linestyle=dashed,linecolor=gray}%
    \pcline(DST1)(DST2) % <- Druckseitentangente
    \pcline(A2)(A1) % <- Anstr"omrichtung
    \lput*{:U}{\small Anstr"omrichtung $v_{\infty}$} }%
  \psIntersectionPoint(A1)(A2)(DST1)(DST2){Hk}
  \pscurve(Hk)(.4,.38)(Vk)(.36,.33)(.5,.32)(Hk)
  \psParallelLine[linecolor=red!75!green,arrows=->,arrowscale=2](Vk)(Hk)(D){.1}{FtE}
  \psRelLine[linecolor=red!75!green,arrows=->,arrowscale=2,angle=90](D)(FtE){4}{Fn}% why
      "4"?
  \psParallelLine[linestyle=dashed](D)(FtE)(Fn){.1}{Fnr1}
  \psRelLine[linestyle=dashed,angle=90](FtE)(D){-4}{Fnr2} % why "-4"?
```

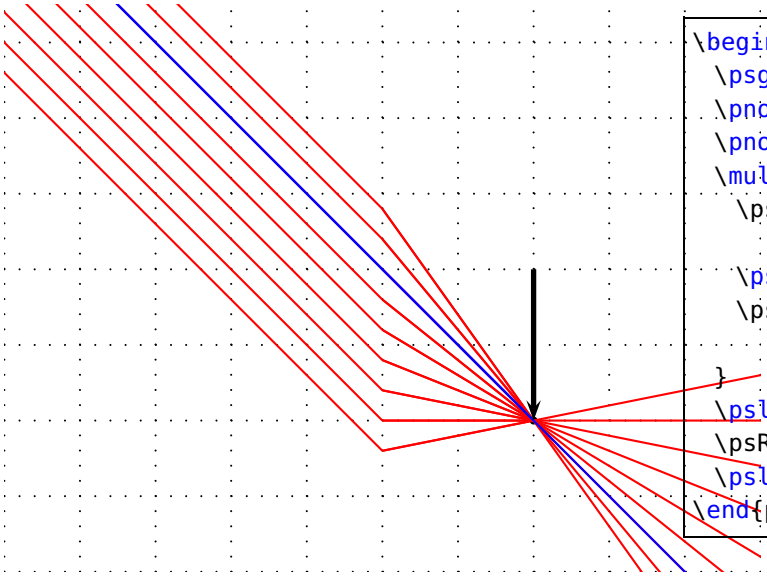
```
  \psline[linewidth=1.5pt,arrows=->,arrowscale=2](D)(Fnr2)
  \psIntersectionPoint(D)([nodesep=2]D)(Fnr1)([offset=-4]Fnr1){Fh}
  \psIntersectionPoint(D)([offset=2]D)(Fnr1)([nodesep=4]Fnr1){Fv}
  \psline[linecolor=blue,arrows=->,arrowscale=2](D)(Fh)
  \psline[linecolor=blue,arrows=->,arrowscale=2](D)(Fv)
  \psline[linestyle=dotted](Fh)(Fnr1) \psline[linestyle=dotted](Fv)(Fnr1)
  \uput{.1}[0](Fh){\blue $F_{H}$} \uput{.1}[180](Fv){\blue $F_{V}$}
  \uput{.1}[-45](Fnr1){$F_{R}$} \uput{.1}[90](Fn){\color{red!75!green}$F_{N}$}
  \uput{.25}[-90](FtE){\color{red!75!green}$F_{T}$}
\end{pspicture}
```

18. \psParallelLine

With this macro it is possible to plot lines relative to a given one, which is parallel. There is no special parameter here.

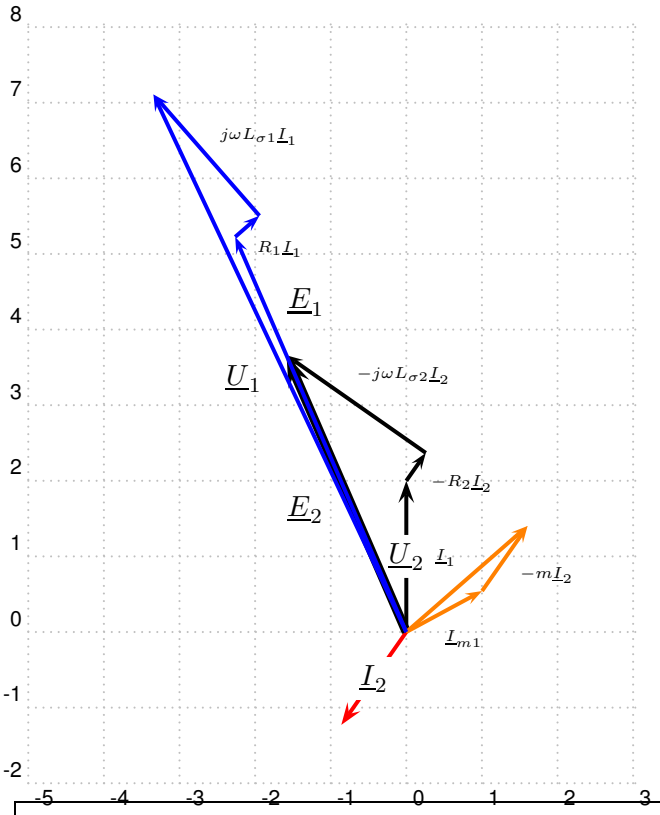
```
\psParallelLine(<P0>)(<P1>)(<P2>){<length>}{<end node name>}
\psParallelLine{<arrows>}(<P0>)(<P1>)(<P2>){<length>}{<end node name>}
\psParallelLine[<options>](<P0>)(<P1>)(<P2>){<length>}{<end node name>}
\psParallelLine[<options>]{<arrows>}(<P0>)(<P1>)(<P2>){<length>}{<end node name>}
```

The line starts at P_2, is parallel to $\overline{P_0P_1}$ and the length of this parallel line depends on the length factor. The end node name must be a valid nodename and shouldn't contain any of the special PostScript characters.



```
\begin{pspicture*}(-5,-4)(5,3.5)
  \psgrid[subgriddiv=0,griddots=5]
  \pnode(2,-2){FF}\qdisk(FF){1.5pt}
  \pnode(-5,5){A}\pnode(0,0){O}
  \multido{\nCountA=-2.4+0.4}{9}{%
    \psParallelLine[linecolor=red](O)(A)(0,\
      nCountA){9}{P1}
    \psline[linecolor=red](0,\nCountA)(FF)
    \psRelLine[linecolor=red](0,\nCountA)(FF)
      {9}{P2}
  }
  \psline[linecolor=blue](A)(FF)
  \psRelLine[linecolor=blue](A)(FF){5}{END1}
  \psline[linewidth=2pt,arrows=->](2,0)(FF)
\end{pspicture*}
```

the following example was created by Patrice Mégret.

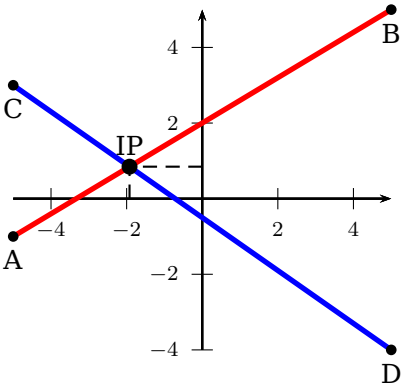



```
\newcommand\pmsc[1]{\ensuremath{\underline{#1}}}
\newcommand\pmIc{\pmsc{I}}
\newcommand\pmUc{\pmsc{U}}
\newcommand\pmEc{\pmsc{E}}
\begin{pspicture}[showgrid=true](-5,-2)(3,8)
\psStartPoint[I](0,0)
\psVector[linewidth=1.5pt,linecolor=red](1.5;235)\ncput*{$\pmIc_2$}%I_2
\psStartPoint[U](0,0)
\psVector[linewidth=1.5pt](2;90)\ncput*{$\pmUc_2$}%U_2
\psParallelLine[linewidth=1.5pt,arrows=->](0,0)(I1)(U1){-0.3}{NUa}\nbput{\tiny $-R_2\pmIc_2$}
\psRelLine[linewidth=1.5pt,arrows=->,angle=90](NUa)(U1){-5}{NUb}%
   \nbput{\tiny $-j\omega L_{\sigma2}\pmIc_2$}%-j\omega I_2
\pcline[linewidth=3.5pt,arrows=->](0,0)(NUb)\naput{$\pmEc_2$}%E_2
\psParallelLine[linewidth=1.5pt,arrows=->,linecolor=blue](0,0)(NUb)(0,0){1.428}{NUc}%
      \nbput[npos=0.8]{$\pmEc_1$}%E_1 (m=0.7)
\psRelLine[linewidth=1.5pt,arrows=->,angle=-85,linecolor=orange](0,0)(NUc){0.2}{NIm1}
   \nbput{\tiny $\pmIc_{m1}$}%I_m1
\psParallelLine[linewidth=1.5pt,arrows=->,linecolor=orange](0,0)(I1)(NIm1){-0.7}{NI1}
   \nbput{\tiny $-m\pmIc_2$}%m.I_2
\pcline[linewidth=1.5pt,arrows=->,linecolor=orange](0,0)(NI1)\naput{\tiny $\pmIc_{1}$}%I_1
\psParallelLine[linewidth=1.5pt,arrows=->,linecolor=blue](0,0)(NI1)(NUc){0.2}{NUd}
   \nbput{\tiny $R_1\pmIc_1$}%R_1 I_1
\psRelLine[linewidth=1.5pt,arrows=->,angle=90,linecolor=blue](NUd)(NUc){-5}{NUe}
   \nbput{\tiny $j\omega L_{\sigma1}\pmIc_1$}%-j\omega I_2
\pcline[linewidth=1.5pt,arrows=->,linecolor=blue](0,0)(NUe)\naput{$\pmUc_{1}$}%I_1
\end{pspicture}
```

19. \psIntersectionPoint

This macro calculates the intersection point of two lines, given by the four coordinates. There is no special parameter here.

```
\psIntersectionPoint(<P0>)(<P1>)(<P2>)(<P3>){<node name>}
```



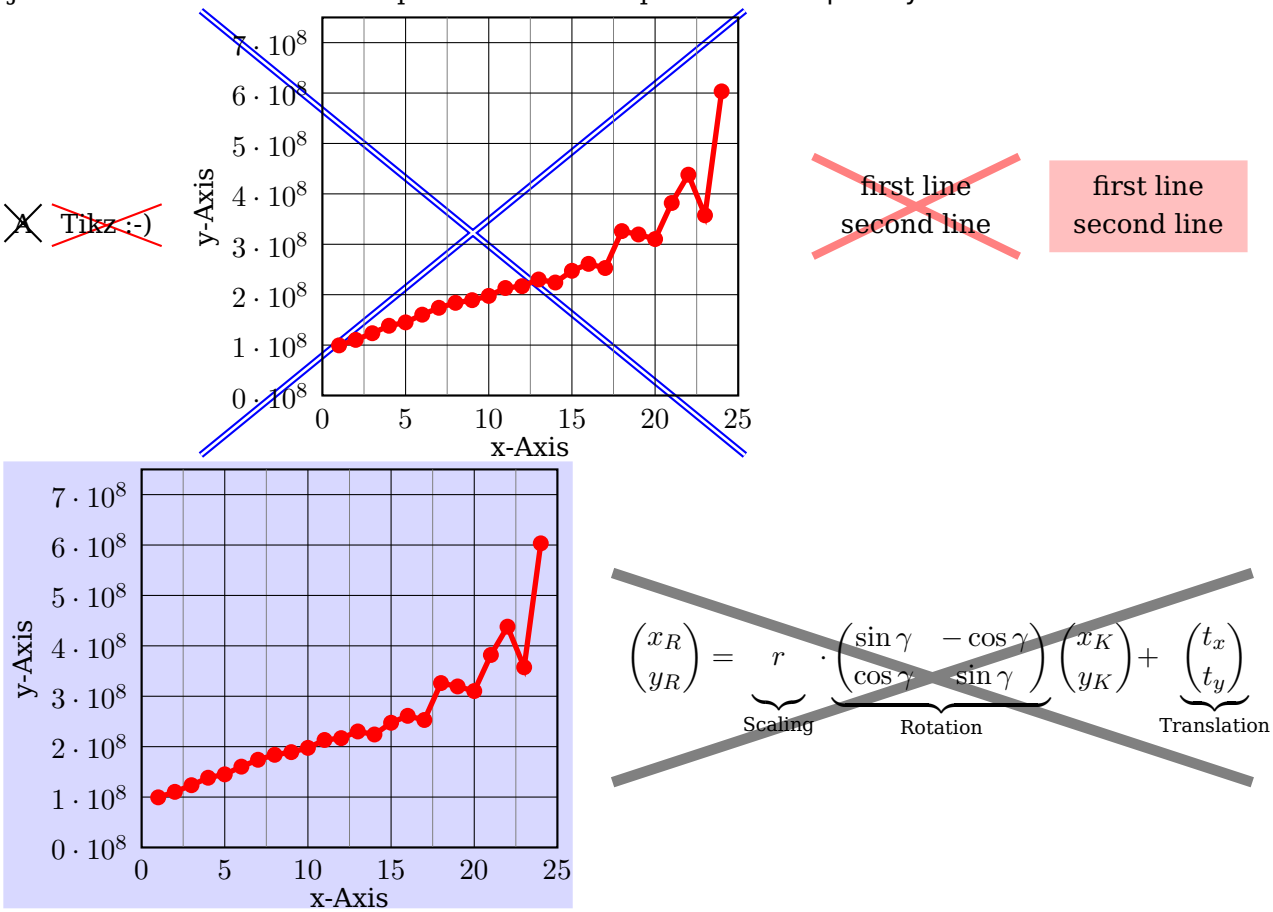

```
\psset{unit=0.5cm}
\begin{pspicture}(-5,-4)(5,5)
  \psaxes[labelFontSize=\scriptstyle,
    dx=2,Dx=2,dy=2,Dy=2]{->}(0,0)(-5,-4)(5,5)
  \psline[linecolor=red,linewidth=2pt](-5,-1)(5,5)
  \psline[linecolor=blue,linewidth=2pt](-5,3)(5,-4)
  \qdisk(-5,-1){2pt}\uput[-90](-5,-1){A}
  \qdisk(5,5){2pt}\uput[-90](5,5){B}
  \qdisk(-5,3){2pt}\uput[-90](-5,3){C}
  \qdisk(5,-4){2pt}\uput[-90](5,-4){D}
  \psIntersectionPoint(-5,-1)(5,5)(-5,3)(5,-4){IP}
  \qdisk(IP){3pt}\uput{0.3}[90](IP){IP}
  \psline[linestyle=dashed](IP|0,0)(IP)(0,0|IP)
\end{pspicture}
```

20. \psCancel[1]

This macro works like the \cancel macro from the package of the same name but it allows as argument any contents, not only letters but also a complex graphic.

\psCancel * [Options] {*contents*}

All optional arguments for lines and boxes are valid and can be used in the usual way. The star option fills the underlying box rectangle with the linecolor. This can be transparent if opacity is set to a value less than 1. This can be used in presentation to strike out words, equations, and graphic objects. Lines can also be transparent when the option strokeopacity is used.

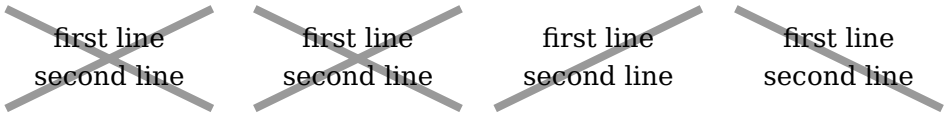



```
\psCancel{A} \psCancel[linecolor=red]{Tikz :-)} \quad
\psCancel[linecolor=blue,doubleline=true]{%
  \readdata{\data}{data/demo1.data}
  \psset{shift=*,xAxisLabel=x-Axis,yAxisLabel=y-Axis,llx=-13mm,lly=-7mm,
     xAxisLabelPos={c,-1},yAxisLabelPos={-7,c}}
  \pstScalePoints(1,0.00000001){}{}
  \begin{psgraph}[axesstyle=frame,xticksize=0 7.5,yticksize=0 25,subticksize=1,
    ylabelFactor=\cdot 10^8,Dx=5,Dy=1,xsubticks=2](0,0)(25,7.5){5.5cm}{5cm}
  \listplot[linecolor=red, linewidth=2pt, showpoints=true]{\data}
  \end{psgraph}} \qquad% end of Cancel
\psCancel[linewidth=3pt,linecolor=red,
   strokeopacity=0.5]{\tabular[b]{c}first line\\second line\endtabular}\quad
\psCancel*[linecolor=red!50,opacity=0.5]{\tabular[b]{c}first line\\second line\endtabular}
```

1 Thanks to by Stefano Baroni

```
\quad
\psCancel*[linecolor=blue!30,opacity=0.5]{%
  \readdata{\data}{data/demo1.data}
  \psset{shift=*,xAxisLabel=x-Axis,yAxisLabel=y-Axis,llx=-15mm,lly=-7mm,urx=1mm,
     xAxisLabelPos={c,-1},yAxisLabelPos={-7,c}}
  \pstScalePoints(1,0.00000001){}{}
  \begin{psgraph}[axesstyle=frame,xticksize=0 7.5,yticksize=0 25,subticksize=1,
    ylabelFactor=\cdot 10^8,Dx=5,Dy=1,xsubticks=2](0,0)(25,7.5){5.5cm}{5cm}
  \listplot[linecolor=red, linewidth=2pt, showpoints=true]{\data}
  \end{psgraph}} \quad% end of Cancel
\psCancel[linewidth=4pt,strokeopacity=0.5]{\parbox{8cm}{\[
  \binom{x_R}{y_R} = \underbrace{r\vphantom{\binom{A}{B}}}_{\text{Scaling}}\cdot
    \underbrace{\begin{pmatrix}
       \sin\gamma & -\cos\gamma \\
     \cos \gamma & \sin \gamma \\
     \end{pmatrix}}_{\text{Rotation}} \binom{x_K}{y_K} +
  \underbrace{\binom{t_x}{t_y}}_{\text{Translation}} \]} }% end of psCancel
```

The optional argument `cancelType` allows to define the lines for the non star version. Possible values are `x` for a cross, `s` for a slash, and `b` for a backslash. It is also possible to use the long words for the `slash` and the `backslash`. An empty value is always assumed as a `x`.




```
\psset{linewidth=3pt,strokeopacity=0.4}
\psCancel{\tabular[b]{c}first line\\second line\endtabular} \quad
\psCancel[cancelType=x]{\tabular[b]{c}first line\\second line\endtabular}\quad
\psCancel[cancelType=s]{\tabular[b]{c}first line\\second line\endtabular}\quad
\psCancel[cancelType=b]{\tabular[b]{c}first line\\second line\endtabular}
```

21. \psStep

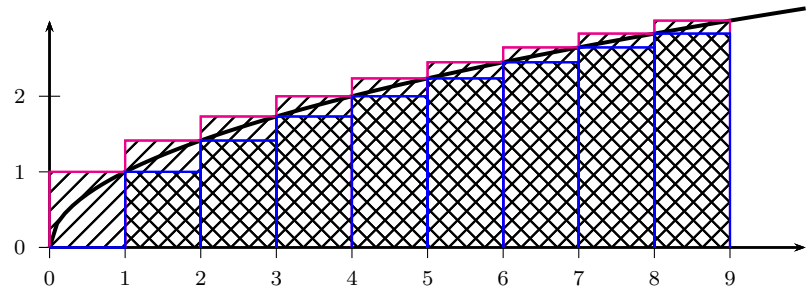
\psStep calculates a step function for the upper or lower sum or the max/min of the Riemann integral definition of a given function. The available option is

StepType=lower |upper|Riemann|infimum|supremum or alternative StepType=l |u|R|i|s

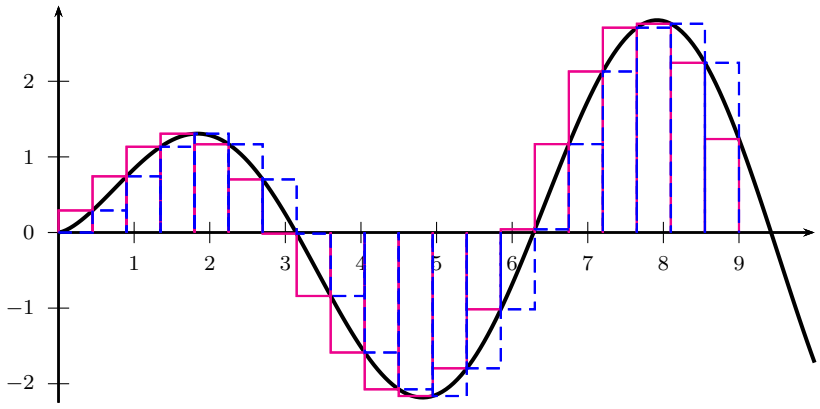
with lower as the default setting. The syntax of the function is

\psStep [Options] (x1, x2){n}{function}

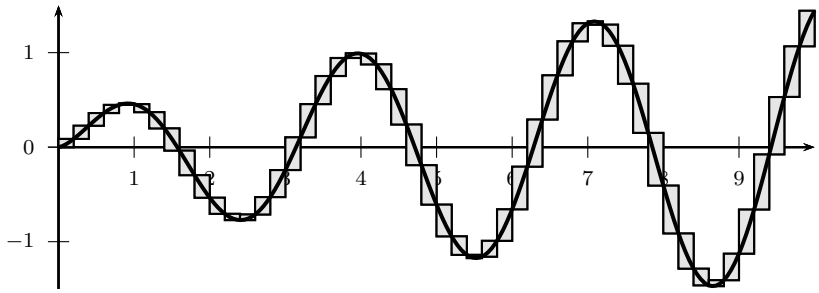
(x1,x2) is the given interval for the step wise calculated function, n is the number of the rectangles and *function* is the mathematical function in postfix or algebraic notation (with algebraic).



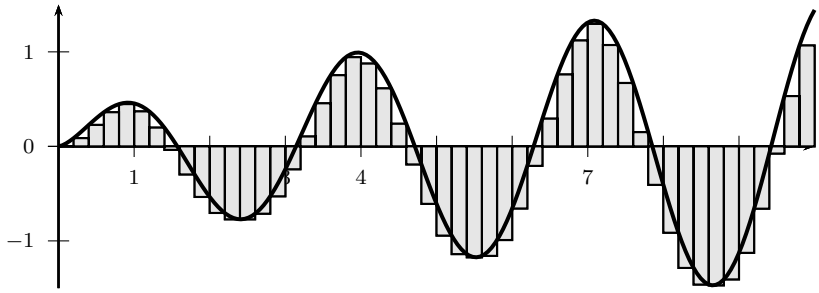

```
\begin{pspicture}(-0.5,-0.5)(10,3)
 \psaxes[labelFontSize=\scriptstyle]{->}(10,3)
 \psplot[plotpoints=100,linewidth=1.5pt,algebraic]{0}{10}{sqrt(x)}
 \psStep[linecolor=magenta,StepType=upper,fillstyle=hlines](0,9){9}{x sqrt}
 \psStep[linecolor=blue,fillstyle=vlines](0,9){9}{x sqrt }
\end{pspicture}
```



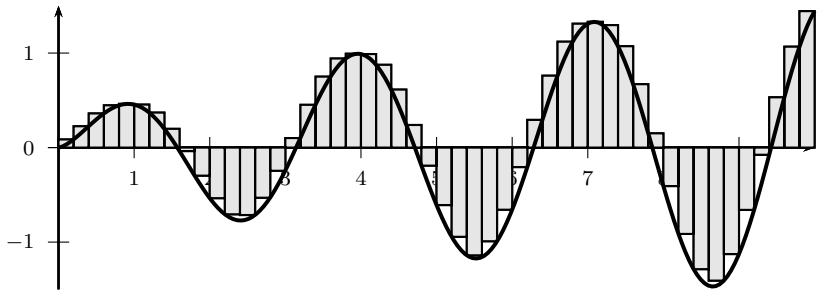

```
\psset{plotpoints=200}
\begin{pspicture}(-0.5,-2.25)(10,3)
  \psaxes[labelFontSize=\scriptstyle]{->}(0,0)(0,-2.25)(10,3)
 \psplot[linewidth=1.5pt,algebraic]{0}{10}{sqrt(x)*sin(x)}
 \psStep[algebraic,linecolor=magenta,StepType=upper](0,9){20}{sqrt(x)*sin(x)}
 \psStep[linecolor=blue,linestyle=dashed](0,9){20}{x sqrt x RadtoDeg sin mul}
\end{pspicture}
```



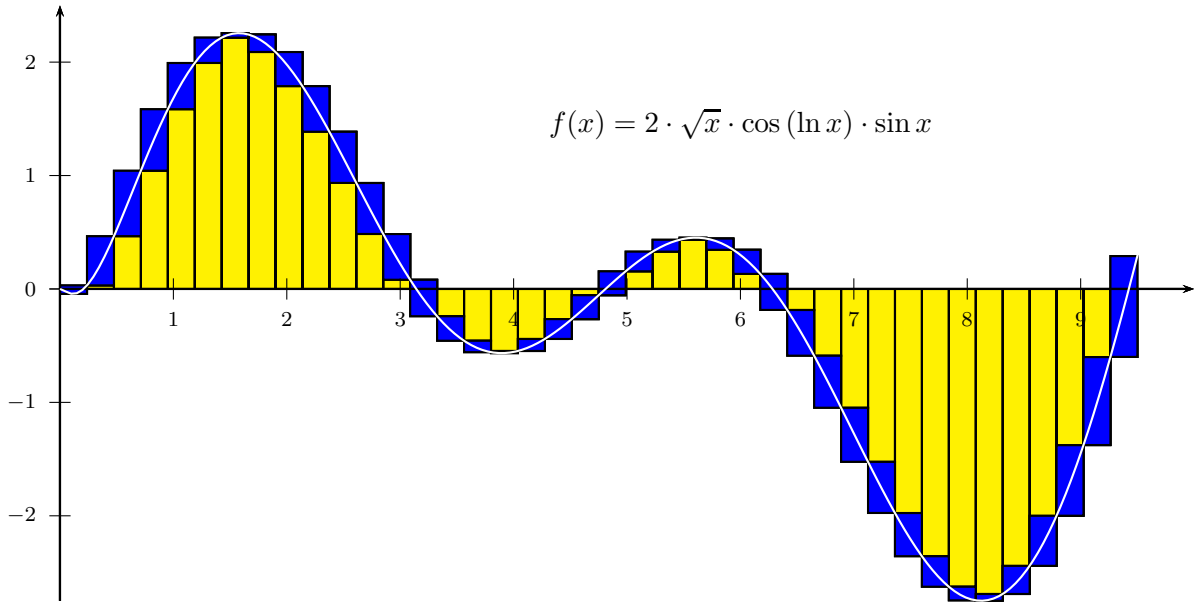

```
\psset{yunit=1.25cm,plotpoints=200}
\begin{pspicture}(-0.5,-1.5)(10,1.5)
 \psaxes[labelFontSize=\scriptstyle]{->}(0,0)(0,-1.5)(10,1.5)
 \psStep[algebraic,StepType=Riemann,fillstyle=solid,fillcolor=black!10](0,10){50}%
   {sqrt(x)*cos(x)*sin(x)}
 \psplot[linewidth=1.5pt,algebraic]{0}{10}{sqrt(x)*cos(x)*sin(x)}
\end{pspicture}
```




```
\psset{yunit=1.25cm,plotpoints=200}
\begin{pspicture}(-0.5,-1.5)(10,1.5)
 \psaxes[labelFontSize=\scriptstyle]{->}(0,0)(0,-1.5)(10,1.5)
 \psStep[algebraic,StepType=infimum,fillstyle=solid,fillcolor=black!10](0,10){50}%
   {sqrt(x)*cos(x)*sin(x)}
 \psplot[linewidth=1.5pt,algebraic]{0}{10}{sqrt(x)*cos(x)*sin(x)}
\end{pspicture}
```



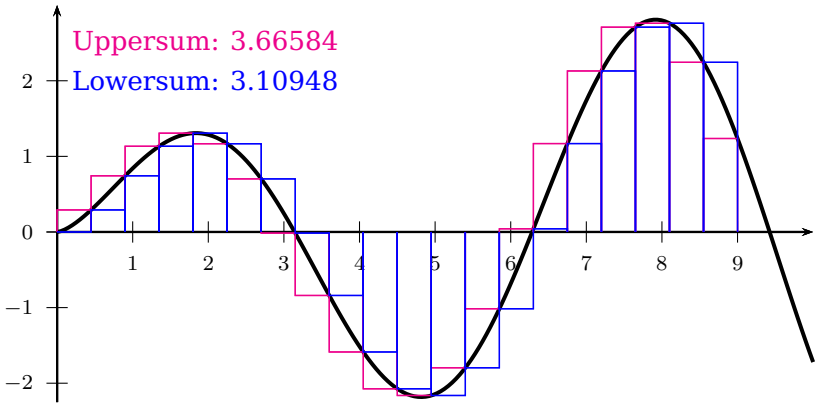

```
\psset{yunit=1.25cm,plotpoints=200}
\begin{pspicture}(-0.5,-1.5)(10,1.5)
 \psaxes[labelFontSize=\scriptstyle]{->}(0,0)(0,-1.5)(10,1.5)
 \psStep[algebraic,StepType=supremum,fillstyle=solid,fillcolor=black!10](0,10){50}%
   {sqrt(x)*cos(x)*sin(x)}
 \psplot[linewidth=1.5pt,algebraic]{0}{10}{sqrt(x)*cos(x)*sin(x)}
\end{pspicture}
```



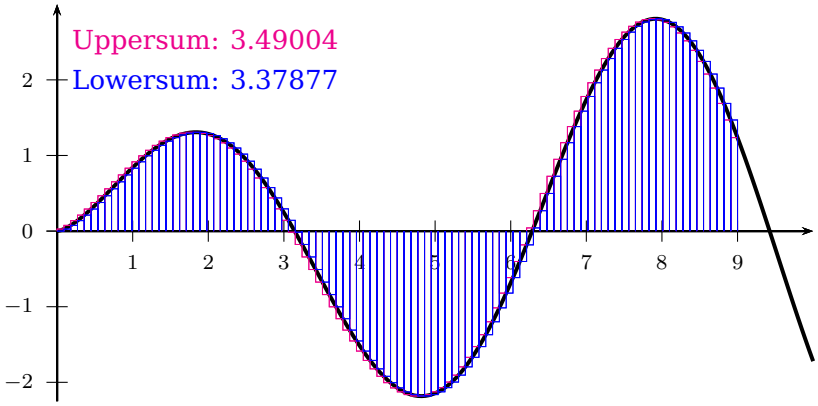

```
\psset{unit=1.5cm,plotpoints=200}
\begin{pspicture}[plotpoints=200](-0.5,-3)(10,2.5)
  \psStep[algebraic,fillstyle=solid,fillcolor=yellow](0.001,9.5){40}{2*sqrt(x)*cos(ln(x))*
      sin(x)}
  \psStep[algebraic,StepType=Riemann,fillstyle=solid,fillcolor=blue](0.001,9.5){40}{2*sqrt(x
      )*cos(ln(x))*sin(x)}
  \psaxes[labelFontSize=\scriptstyle]{->}(0,0)(0,-2.75)(10,2.5)
  \psplot[algebraic,linecolor=white]{0.001}{9.75}{2*sqrt(x)*cos(ln(x))*sin(x)}
  \uput[90](6,1.2){$f(x)=2\cdot\sqrt{x}\cdot\cos{(\ln{x})}\cdot\sin{x}$}
\end{pspicture}
```

There is also an optional argument `noVerticalLines` which suppresses all vertical lines of the step function in the output.

With setting the optional argument `saveSumValue` it is possible to calculate the area under the rectangles. The value is saved in a macro `\pstAreaA`, for the next call of `\psStep` in the macro `\pstAreaB` and so on. The values are calculated on PostScript level, the reason why two LaTeX runs are needed. for every `\psStep` there will be an external file `\jobname-area#.tex` which has the definition of the macros `\pstArea?`. In this documentation the following `\psStep` is the 10th call of this function, the reason why we have to choose `\pstAreaJ` and so on:




```
\psset{plotpoints=200}%
\begin{pspicture}(-0.5,-2.25)(10,3)
\psaxes[labelFontSize=\scriptstyle]{->}(0,0)(0,-2.25)(10,3)%
\psplot[linewidth=1.5pt,algebraic]{0}{10}{sqrt(x)*sin(x)}%
\psset{linewidth=0.5pt}%
\psStep[algebraic,linecolor=magenta,StepType=upper,saveSumValue](0,9){20}{sqrt(x)*sin(x)}%
\rput[l](0.2,2.5){\textcolor{magenta}{Uppersum: \pstAreaJ}}%
\psStep[algebraic,linecolor=blue,saveSumValue](0,9){20}{sqrt(x)*sin(x)}%
\rput[l](0.2,2){\textcolor{blue}{Lowersum: \pstAreaK}}%
\end{pspicture}
```



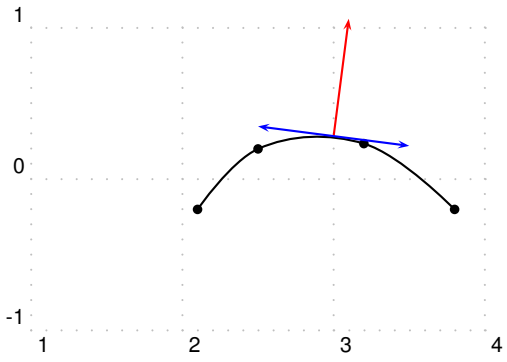

```
\psset{plotpoints=200}%
\begin{pspicture}(-0.5,-2.25)(10,3)
\psaxes[labelFontSize=\scriptstyle]{->}(0,0)(0,-2.25)(10,3)%
\psplot[linewidth=1.5pt,algebraic]{0}{10}{sqrt(x)*sin(x)}%
\psset{linewidth=0.1pt}%
\psStep[algebraic,linecolor=magenta,StepType=upper,saveSumValue](0,9){100}{sqrt(x)*sin(x)}
    %
\rput[l](0.2,2.5){\textcolor{magenta}{Uppersum: \pstAreaL}}%
\psStep[algebraic,linecolor=blue,saveSumValue](0,9){100}{sqrt(x)*sin(x)}%
\rput[l](0.2,2){\textcolor{blue}{Lowersum: \pstAreaM}}%
\end{pspicture}
```

22. Tangent lines

There are two macros for plotting a tangent line or the tangent normal line. The first one is `\psTangentLine` which expects three pairs of coordinates, a x and a dx value. The second one is `\psplotTangent` which expects a function for the curve.

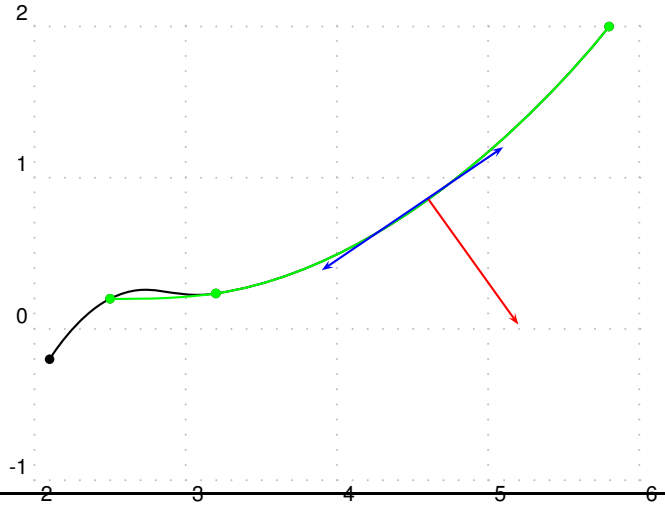
22.1. `\psTangentLine` and option `Tnormal`

`\psTangentLine [Options]` $(x_1,y_1)(x_2,y_2)(x_3,y_3)\{x\}\{dx\}$



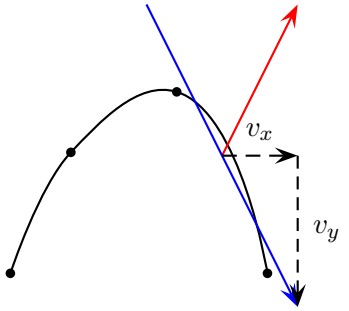

```
\psset{unit=2cm}
\begin{pspicture}[showgrid=true](1,-1)(4,1)
  \pscurve[showpoints=true]
    (2.1,-0.2)(2.5,0.2)(3.2,0.235)(3.8,-0.2)
  \psTangentLine[Tnormal,arrows=->,
    linecolor=red](2.5,0.2)(3.2,0.235)%
      (3.8,-0.2){3}{0.1}
  \psTangentLine[arrows=<->,
    linecolor=blue](2.5,0.2)(3.2,0.235)%
      (3.8,-0.2){3}{0.5}
\end{pspicture}
```

In special cases one has to use `curvature=1 1 1` for the macro `\pscurve` to get the same equation for the curve as `\psplotTangentLine` does.




```
\psset{unit=2cm}
\begin{pspicture}[showgrid=true](2,-1)(6,2)
\pscurve[showpoints=true,
  curvature=1 1 1](2.1,-0.2)(2.5,0.2)(3.2,0.235)(5.8,2)
\pscurve[showpoints=true,linecolor=green,
  curvature=1 1 1](2.5,0.2)(3.2,0.235)(5.8,2)
\psTangentLine[Tnormal,arrows=->,linecolor=red](2.5,0.2)(3.2,0.235)(5.8,2){4.6}{0.6}
\psTangentLine[arrows=<->,linecolor=blue](2.5,0.2)(3.2,0.235)(5.8,2){4.5}{0.6}
\end{pspicture}
```

The end points are saved as nodes `OCurve`, `ETangent`, and `ENormal`. They can be used in the default ways for nodes:




```
\psset{yunit=4cm,xunit=2cm,arrowscale=2}
\begin{pspicture}(0.1,-0.3)(4,1)
\pscurve[showpoints=true](2.1,-0.2)(2.5,0.2)(3.2,0.4)(3.8,-0.2)
\psTangentLine[Tnormal,arrows=->,linecolor=red](2.5,0.2)(3.2,0.4)(3.8,-0.2){3.5}{0.5}
\psTangentLine[arrows=->,linecolor=blue](2.5,0.2)(3.2,0.4)(3.8,-0.2){3.5}{0.5}
\pcline[linestyle=dashed]{->}(OCurve)(ETangent|OCurve)\naput{$v_x$}
\pcline[linestyle=dashed]{->}(ETangent|OCurve)(ETangent)\naput{$v_y$}% double coordinate (x,y|x,y)
\end{pspicture}
```

22.2. \psplotTangent and option Tnormal

There is an additional option, named `Derive` for an alternative function (see following example) to calculate the slope of the tangent. This will be in general the first derivative, but can also be any other function. If this option is different to to the default value `Derive=default` , then this function is taken to calculate the slope. For the other cases, `pstricks-add` builds a secant with -0.00005<x<0.00005, calculates the slope and takes this for the tangent. This may be problematic in some cases of special functions or x values, then it may be appropriate to use the Derive option.

`\psplotTangent*[Options]{x}{dx}{function}`

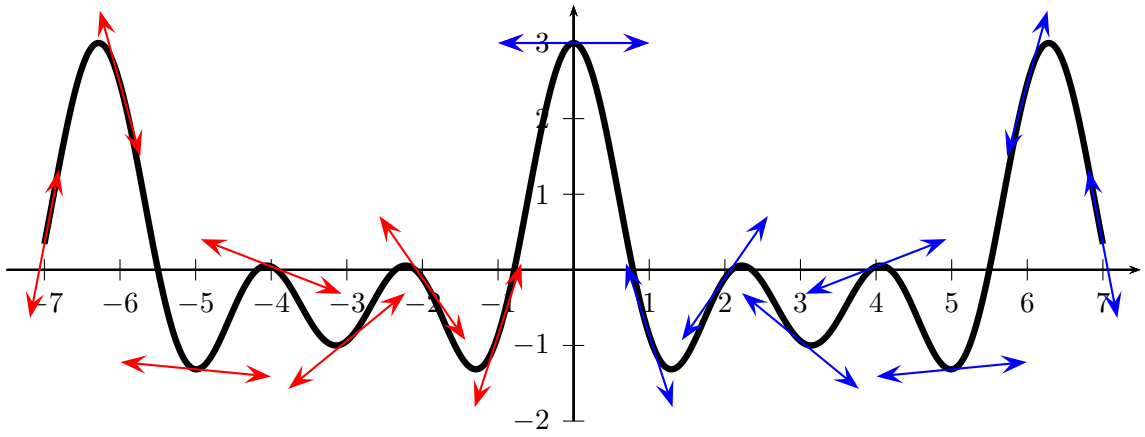
The macro expects three parameters:

x : the x value of the function for which the tangent should be calculated

dx : the dx to both sides of the x value

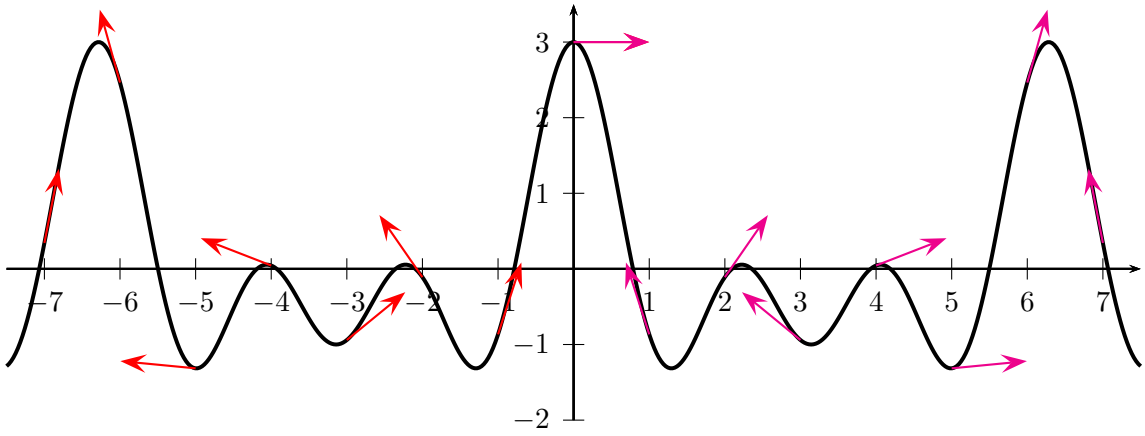
$f(x)$: the function in infix (with option `algebraic`) or the default postfix (PostScript) notation

The following examples show the use of the algebraic option together with the Derive option. Remember that using the `algebraic` option implies that the angles have to be in the radian unit!



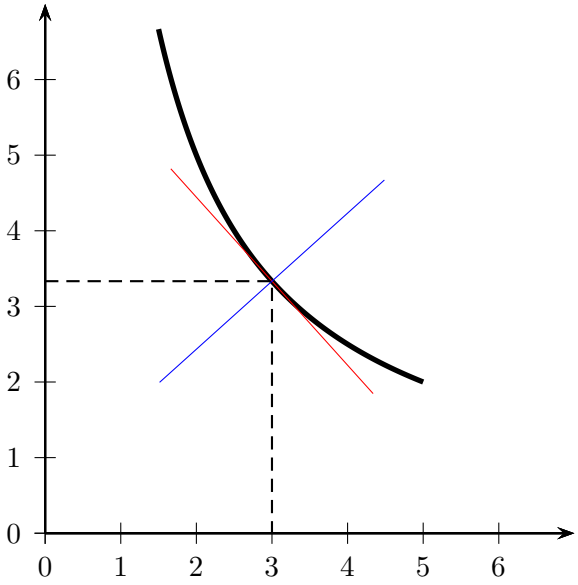

```
\def\F{x RadtoDeg dup dup cos exch 2 mul cos add exch 3 mul cos add}
\def\Fp{x RadtoDeg dup dup sin exch 2 mul sin 2 mul add exch 3 mul sin 3 mul add neg}
\psset{plotpoints=1001}
\begin{pspicture}(-7.5,-2.5)(7.5,4)%X\psgrid
  \psaxes{->}(0,0)(-7.5,-2)(7.5,3.5)
  \psplot[linewidth=3\pslinewidth]{-7}{7}{\F}
  \psset{linecolor=red, arrows=<->, arrowscale=2}
  \multido{\n=-7+1}{8}{\psplotTangent{\n}{1}{\F}}
  \psset{linecolor=magenta, arrows=<->, arrowscale=2}%
  \multido{\n=0+1}{8}{\psplotTangent[linecolor=blue, Derive=\Fp]{\n}{1}{\F}}
\end{pspicture}
```

The star version plots only the tangent line in the positive x-direction:



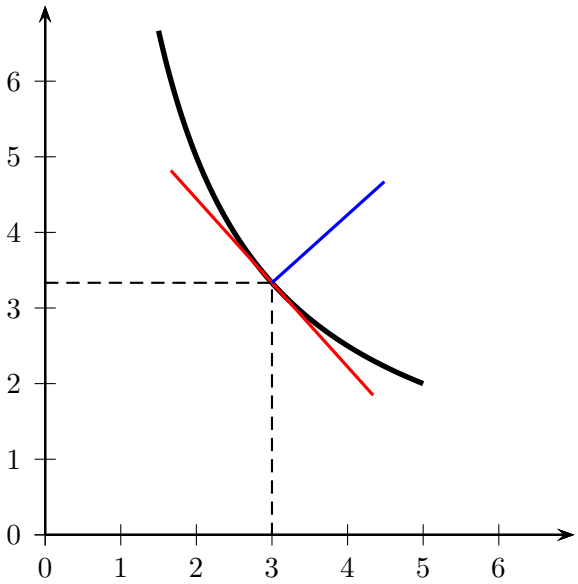

```
\def\Falg{cos(x)+cos(2*x)+cos(3*x)} \def\Fpalg{-sin(x)-2*sin(2*x)-3*sin(3*x)}
\begin{pspicture}(-7.5,-2.5)(7.5,4)%\psgrid
  \psaxes{->}(0,0)(-7.5,-2)(7.5,3.5)
  \psplot[linewidth=1.5pt,algebraic,plotpoints=500]{-7.5}{7.5}{\Falg}
  \multido{\n=-7+1}{8}{\psplotTangent*[linecolor=red,arrows=->,arrowscale=2,algebraic]{\n
      }{1}{\Falg}}
  \multido{\n=0+1}{8}{\psplotTangent*[linecolor=magenta,%
    arrows=->,arrowscale=2,algebraic,Derive={\Fpalg}]{\n}{1}{\Falg}}
\end{pspicture}
```

The next example shows the use of the `Derive` option to draw the perpendicular line to the tangent.




```
\begin{pspicture}(-0.5,-0.5)(7.25,7.25)
  \def\Func{10 x div}
  \psaxes[arrowscale=1.5]{->}(7,7)
  \psplot[linewidth=2pt,algebraic]{1.5}{5}{10/x}
  \psplotTangent[linewidth=.5\pslinewidth,linecolor=red,
      algebraic]{3}{2}{10/x}
  \psplotTangent[linewidth=.5\pslinewidth,linecolor=blue,
      algebraic,Derive=(x*x)/10]{3}{2}{10/x}
  \psline[linestyle=dashed](!0 /x 3 def \Func)(!3 /x 3 de
      \Func)(3,0)
\end{pspicture}
```

By setting the optional argument `Tnormal` one can plot the normal of the tangent line. It always starts at the given point.

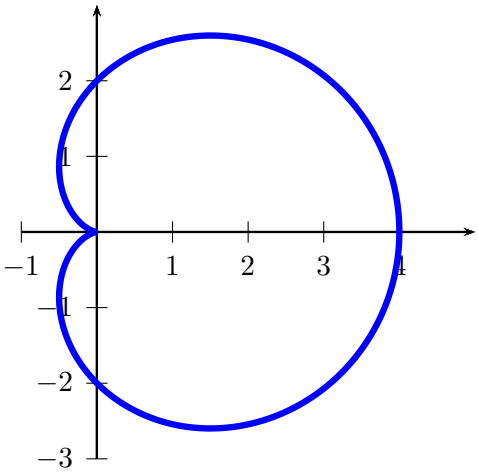



```
\begin{pspicture}(-0.5,-0.5)(7.25,7.25)
  \def\Func{10 x div}
  \psaxes[arrowscale=1.5]{->}(7,7)
  \psplot[linewidth=2pt]{1.5}{5}{\Func}
  \psplotTangent[linewidth=1.5\pslinewidth,linecolor=red
      ]{3}{2}{\Func}
  \psplotTangent[linewidth=1.5\pslinewidth,linecolor=blue
      Tnormal]{3}{2}{\Func}
  \psline[linestyle=dashed](!0 /x 3 def \Func)(!3 /x 3 de
      \Func)(3,0)
\end{pspicture}
```

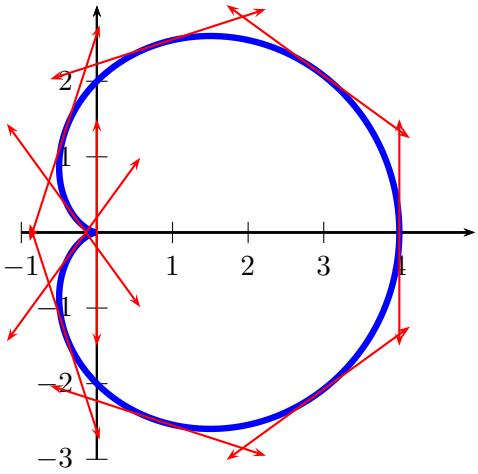
Let's work with the classical cardioid: $r = 2(1 + \cos(\theta))$ and $\frac{dr}{d\theta} = -2\sin(\theta)$. The `Derive` option always expects the $\frac{dr}{d\theta}$ value and uses internally the equation for the derivative of implicitly defined functions:

$$\frac{dy}{dx} = \frac{r' \cdot \sin\theta + x}{r' \cdot \cos\theta - y}$$

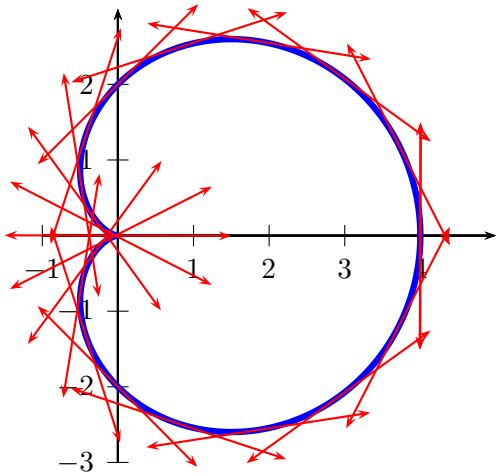
where $x = r \cdot \cos\theta$ and $y = r \cdot \sin\theta$




```
\begin{pspicture}(-1,-3)(5,3)%\psgrid[subgridcolor=lightgray]
  \psaxes{->}(0,0)(-1,-3)(5,3)
  \psplot[polarplot,linewidth=3\pslinewidth,linecolor=blue,%
    plotpoints=500]{0}{360}{1 x cos add 2 mul}
\end{pspicture}
```



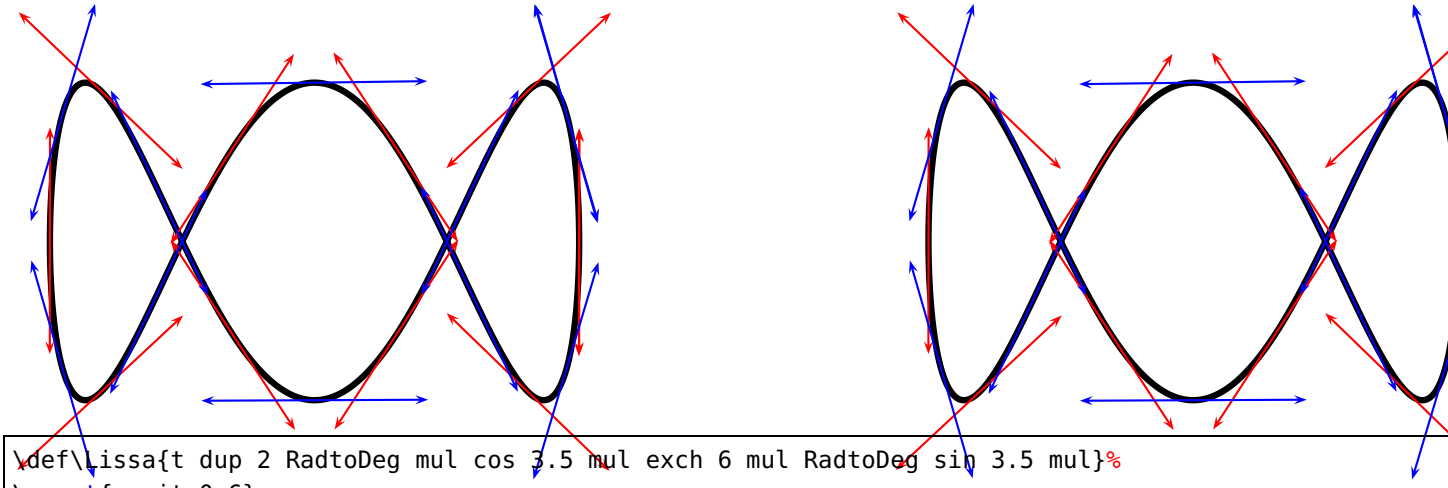

```
\begin{pspicture}(-1,-3)(5,3)%\psgrid[subgridcolor=lightgray]
  \psaxes{->}(0,0)(-1,-3)(5,3)
  \psplot[polarplot,linewidth=3\pslinewidth,linecolor=blue,plotpoin
     =500]{0}{360}{1 x cos add 2 mul}
  \multido{\n=0+36}{10}{%
    \psplotTangent[polarplot,linecolor=red,arrows=<->]{\n}{1.5}{1 x
       cos add 2 mul} }
\end{pspicture}
```




```
\begin{pspicture}(-1,-3)(5,3)%\psgrid[subgridcolor=lightgray]
  \psaxes{->}(0,0)(-1,-3)(5,3)
  \psplot[polarplot,linewidth=3\pslinewidth,linecolor=blue,algebrai
     plotpoints=500]{0}{6.289}{2*(1+cos(x))}
  \multido{\r=0.000+0.314}{21}{%
    \psplotTangent[polarplot,Derive=-2*sin(x),algebraic,linecolor=r
       ,arrows=<->]{\r}{1.5}{2*(1+cos(x))} }
\end{pspicture}
```

Let's work with a Lissajou curve: $\begin{cases} x = 3.5\cos(2t) \\ y = 3.5\sin(6t) \end{cases}$ whose derivative is : $\begin{cases} x = -7\sin(2t) \\ y = 21\cos(6t) \end{cases}$

The parameter must be the letter t instead of x and when using the `algebraic` option you must separate the two equations by a `|` (see example).



```
\def\Lissa{t dup 2 RadtoDeg mul cos 3.5 mul exch 6 mul RadtoDeg sin 3.5 mul}%
\psset{yunit=0.6}
\begin{pspicture}(-4,-4)(4,6)
  \parametricplot[plotpoints=500,linewidth=3\pslinewidth]{0}{3.141592}{\Lissa}
  \multido{\r=0.000+0.314}{11}{%
    \psplotTangent[linecolor=red,arrows=<->]{\r}{1.5}{\Lissa} }
  \multido{\r=0.157+0.314}{11}{%
    \psplotTangent[linecolor=blue,arrows=<->]{\r}{1.5}{\Lissa} }
\end{pspicture}\hfill%
\def\LissaAlg{3.5*cos(2*t)|3.5*sin(6*t)} \def\LissaAlgDer{-7*sin(2*t)|21*cos(6*t)}%
\begin{pspicture}(-4,-4)(4,6)
  \parametricplot[algebraic,plotpoints=500,linewidth=3\pslinewidth]{0}{3.141592}{\LissaAlg}
  \multido{\r=0.000+0.314}{11}{%
    \psplotTangent[algebraic,linecolor=red,arrows=<->]{\r}{1.5}{\LissaAlg}}
  \multido{\r=0.157+0.314}{11}{%
    \psplotTangent[algebraic,linecolor=blue,arrows=<->,
      Derive=\LissaAlgDer]{\r}{1.5}{\LissaAlg} }
\end{pspicture}
```

23. Successive derivatives of a function

The new PostScript function `Derive` has been added for plotting successive derivatives of a function. It must be used with the `algebraic` option. This function has two arguments:

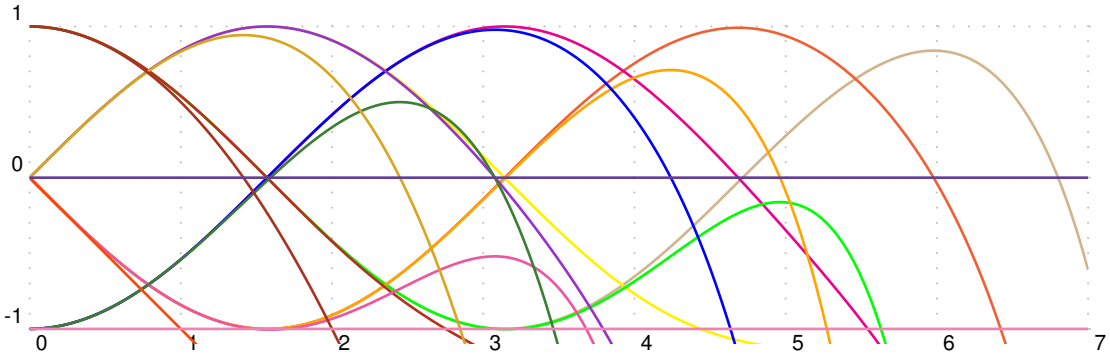
1. a positive integer which defines the order of the derivative; obviously 0 means the function itself!
2. a function of variable x which can be any function using common operators,

Do not think that the derivative is approximated, the internal PostScript engine will compute the real derivative using a formal derivative engine.

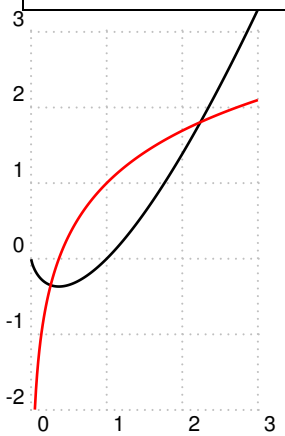
The following diagram contains the plot of the polynomial:

$$f(x)=\sum_{i=0}^{14}\frac{(-1)^i x^{2i}}{i!}=1-\frac{x^2}{2}+\frac{x^4}{4!}-\frac{x^6}{6!}+\frac{x^8}{8!}-\frac{x^{10}}{10!}+\frac{x^{12}}{12!}-\frac{x^{14}}{14!}$$

and of its first 15 derivatives. It is the sequence definition of the cosine.




```
\psset{unit=2}
\def\getColor#1{\ifcase#1 Tan\or RedOrange\or magenta\or yellow\or green\or Orange\or blue\or
  DarkOrchid\or BrickRed\or Rhodamine\or OliveGreen\or Goldenrod\or Mahogany\or
  OrangeRed\or CarnationPink\or RoyalPurple\or Lavender\fi}
\begin{pspicture}[showgrid=true](0,-1.2)(7,1.5)
  \psclip{\psframe[linestyle=none](0,-1.1)(7,1.1)}
  \multido{\in=0+1}{16}{%
    \psplot[linewidth=1pt,algebraic,linecolor=\getColor{\in}]{0}{7}
     {Derive(\in,1-x^2/2+x^4/24-x^6/720+x^8/40320-x^10/3628800+x^12/479001600-x^14/87178291200)}}
  \endpsclip
\end{pspicture}
```




```
\begin{pspicture}[shift=-2.5,showgrid=true,linewidth=1pt](0,-2)(3,3)
  \psplot[algebraic]{.001}{3}{x*ln(x)} % f(x)
  \psplot[algebraic,linecolor=red]{.05}{3}{Derive(1,x*ln(x))} % f'(x)=1+
     ln(x)
\end{pspicture}
```

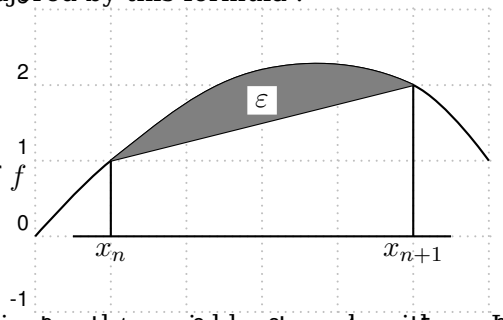
24. Variable step for plotting a curve

24.1. Theory

As you know with the `\psplot` macro, the curve is plotted using a piece-wise linear curve. The step is given by the parameter `plotpoints`. For each step between x_i and x_{i+1}, the area defined between the curve and its approximation (a segment) is majored by this formula :

$$|\varepsilon| \leq \frac{M_2(f)(x_{i+1} - x_i)^3}{12}$$

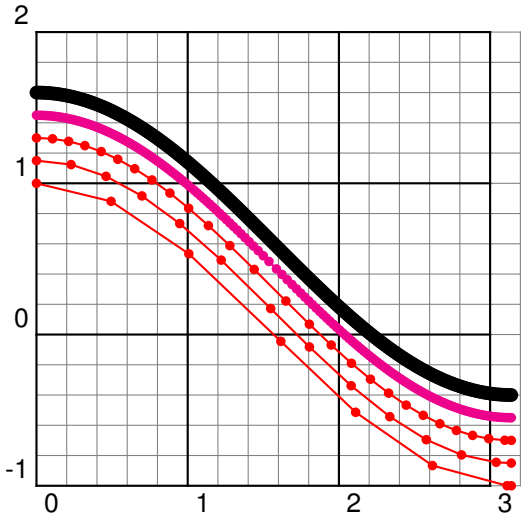
$M_2(f)$ is a majorant of the second derivative of f in the interval $[x_i; x_{i+1}]$.



The parameter `VarStep` (`false` by default) activates the variable step algorithm. It is set to a tolerance defined by the parameter `VarStepEpsilon` (`default` by default, accept real value). If this parameter is not set by the user, then it is automatically computed using the default first step given by the parameter `plotpoints`. Then, for each step, $f''(x_n)$ and $f''(x_{n+1})$ are computed and the smaller is used as $M_2(f)$, and then the step is approximated. This means that the step is constant for second order polynomials.

24.2. The cosine

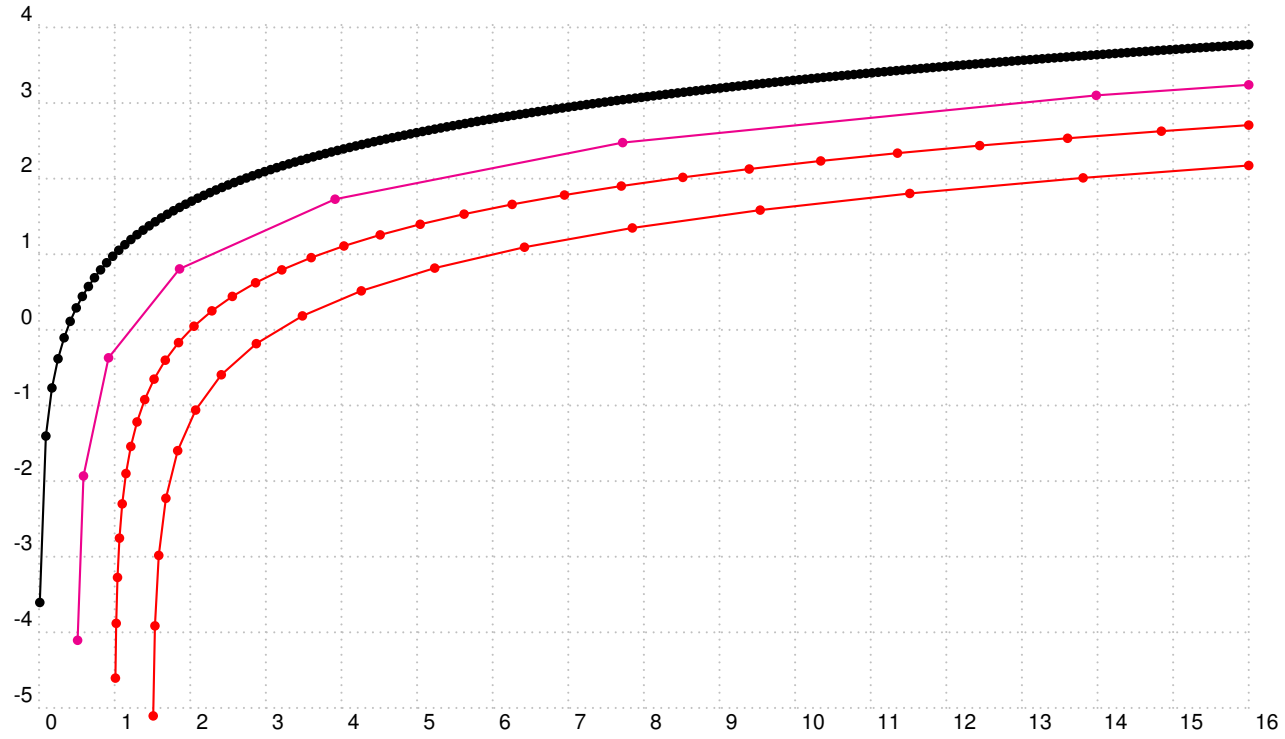
Different value for the tolerance from 0.01 to $0.000\,1$, a factor 10 between each of them. In black, there is the classic `\psplot` behavior, and in magenta the default variable step behavior.




```
\psset{algebraic, VarStep=true, unit=2, showpoints=true, linecolor=red}
\begin{pspicture}[showgrid=true](-0,-1)(3.14,2)
  \psplot[VarStepEpsilon=.01]{0}{3.14}{cos(x)}
  \psplot[VarStepEpsilon=.001]{0}{3.14}{cos(x)+.15}
  \psplot[VarStepEpsilon=.0001]{0}{3.14}{cos(x)+.3}
  \psplot[linecolor=magenta]{0}{3.14}{cos(x)+.45}
  \psplot[VarStep=false,linewidth=1pt,linecolor=black]{-0}{3.14}{cos(x)+.6}
\end{pspicture}
```

24.3. The Napierian Logarithm

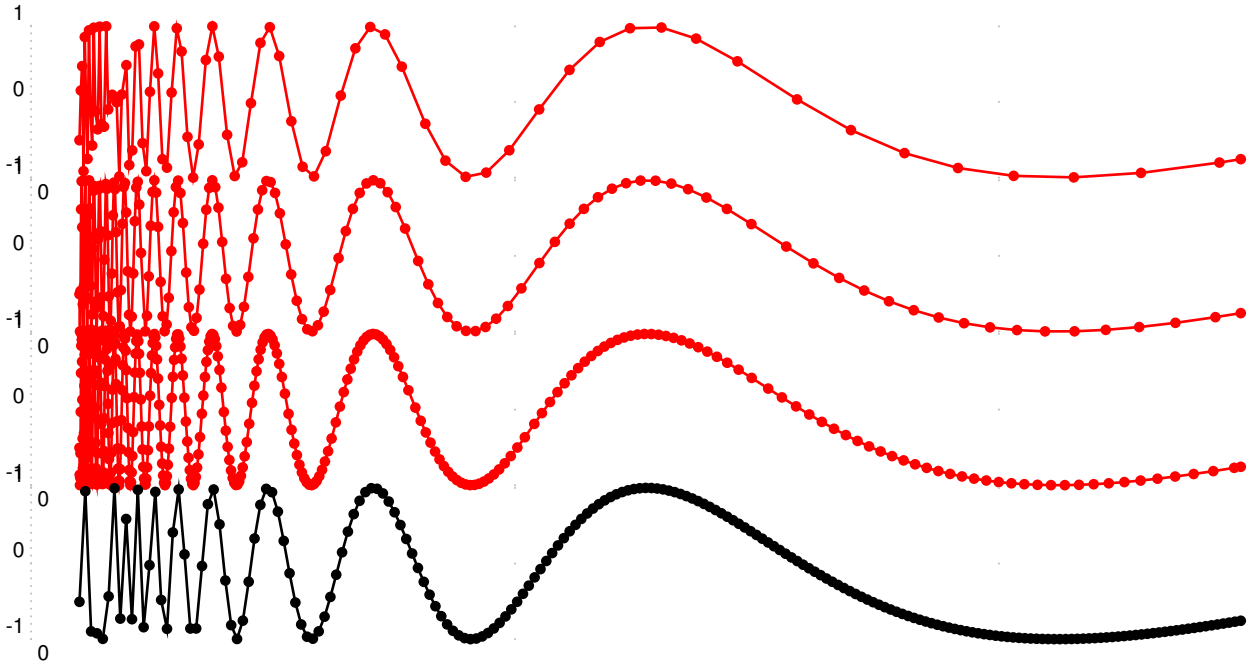
A really classic example which gives a bad beginning, the tolerance is set to 0.001.




```
\psset{algebraic, VarStep=true, linecolor=red, showpoints=true}
\begin{pspicture}[showgrid=true](0,-5)(16,4)
  \psplot[VarStep=false, linecolor=black]{.01}{16}{ln(x)+1}
  \psplot[linecolor=magenta]{.51}{16}{ln(x-1/2)+1/2}
  \psplot[VarStepEpsilon=.001]{1.01}{16}{ln(x-1)}
  \psplot[VarStepEpsilon=.01]{1.51}{16}{ln(x-1.5)-100/200}
\end{pspicture}
```

24.4. Sine of the inverse of x

Impossible to draw, but let's try!

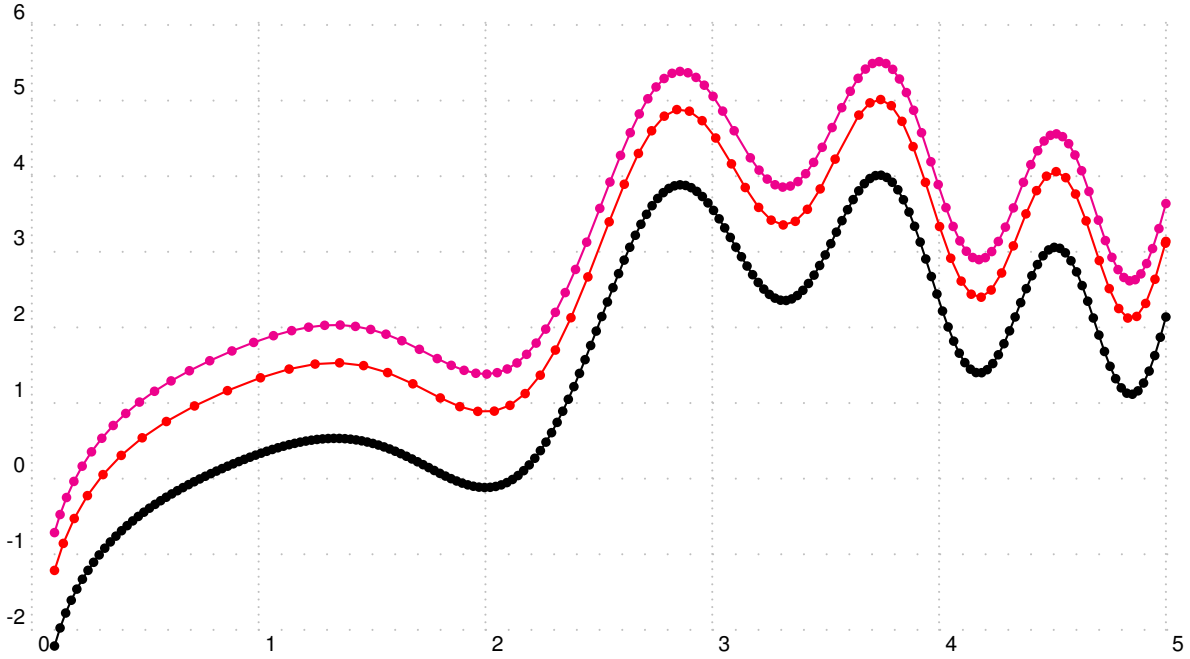



```
\psset{xunit=64,algebraic,VarStep,linecolor=red,showpoints=true,linewidth=1pt}
\begin{pspicture}[showgrid=true](0,-1)(.5,1)
  \psplot[VarStepEpsilon=.0001]{.01}{.25}{sin(1/x)}
\end{pspicture}\\
\begin{pspicture}[showgrid=true](0,-1)(.5,1)
  \psplot[VarStepEpsilon=.00001]{.01}{.25}{sin(1/x)}
\end{pspicture}\\
\begin{pspicture}[showgrid=true](0,-1)(.5,1)
  \psplot[VarStepEpsilon=.000001]{.01}{.25}{sin(1/x)}
\end{pspicture}\\
\begin{pspicture}[showgrid=true](0,-1)(.5,1)
  \psplot[VarStep=false, linecolor=black]{.01}{.25}{sin(1/x)}
\end{pspicture}
```

24.5. A really complicated function

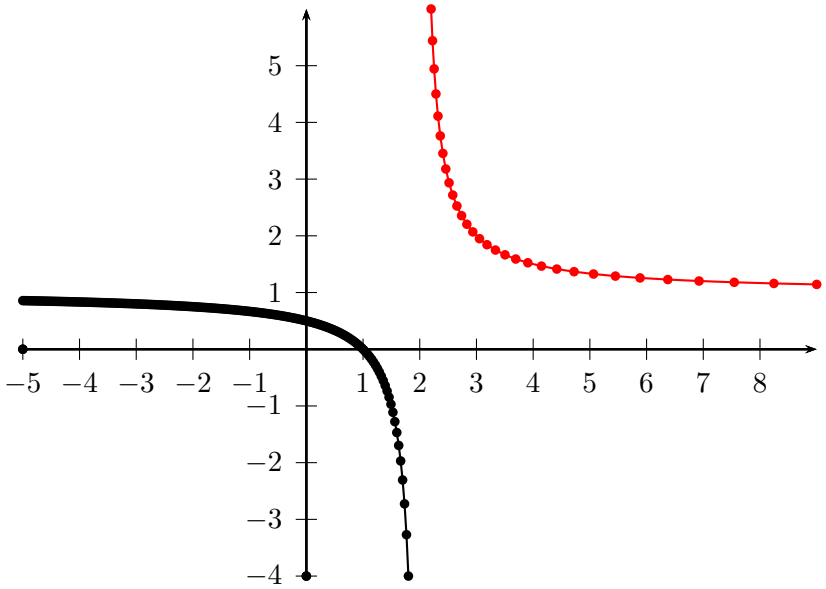
Just appreciate the difference between the normal behavior and the plotting with the `varStep` option. The function is:

$$f(x) = x - \frac{x^2}{10} + \ln(x) + \cos(2x) + \sin(x^2) - 1$$




```
\psset{xunit=3, algebraic, VarStep, showpoints=true}
\begin{pspicture}[showgrid=true](0,-2)(5,6)
  \psplot[VarStepEpsilon=.0005, linecolor=red]{.1}{5}{x-x^2/10+ln(x)+cos(2*x)+sin(x^2)}
  \psplot[linecolor=magenta]{.1}{5}{x-x^2/10+ln(x)+cos(2*x)+sin(x^2)+.5}
  \psplot[VarStep=false]{.1}{5}{x-x^2/10+ln(x)+cos(2*x)+sin(x^2)-1}
\end{pspicture}
```

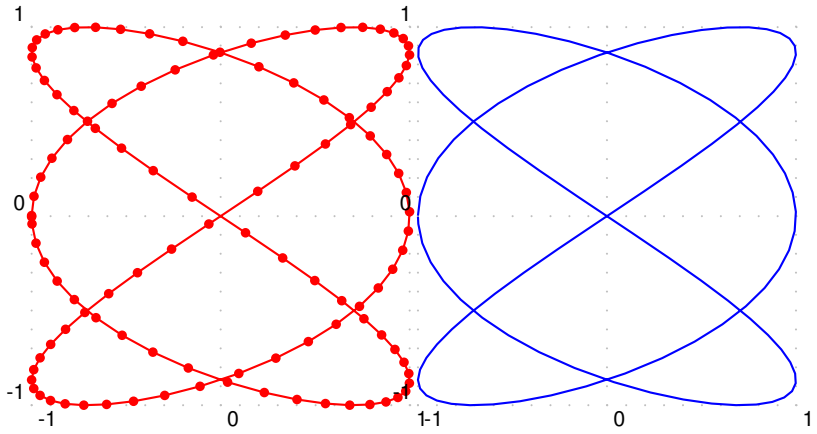
24.6. A hyperbola



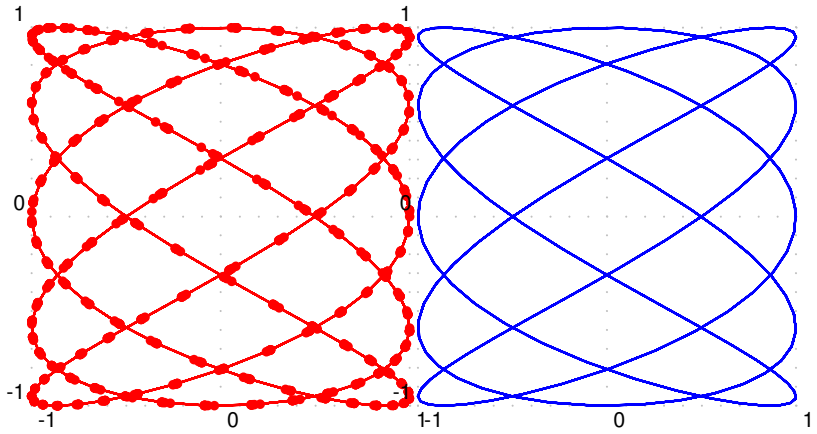

```
\psset{algebraic, showpoints=true, unit=0.75}
\begin{pspicture}(-5,-4)(9,6)
  \psplot[linecolor=black]{-5}{1.8}{(x-1)/(x-2)}
  \psplot[VarStep=true, VarStepEpsilon=.001, linecolor=red]{2.2}{9}{(x-1)/(x-2)}
  \psaxes{->}(0,0)(-5,-4)(9,6)
\end{pspicture}
```

24.7. Using \psparametricplot

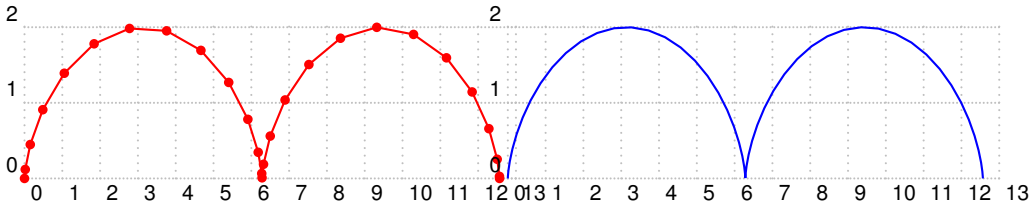
\parametricplot [Options] {t0}{t1} [PS commands] {x(t) y(t)}




```
\psset{unit=3}
\begin{pspicture}[showgrid=true](-1,-1)(1,1)
\parametricplot[algebraic,linecolor=red,VarStep=true, showpoints=true,
            VarStepEpsilon=.0001]
            {-3.14}{3.14}{cos(3*t)|sin(2*t)}
\end{pspicture}
\begin{pspicture}[showgrid=true](-1,-1)(1,1)
\parametricplot[algebraic,linecolor=blue,VarStep=true, showpoints=false,
            VarStepEpsilon=.0001]
            {-3.14}{3.14}{cos(3*t)|sin(2*t)}
\end{pspicture}
```



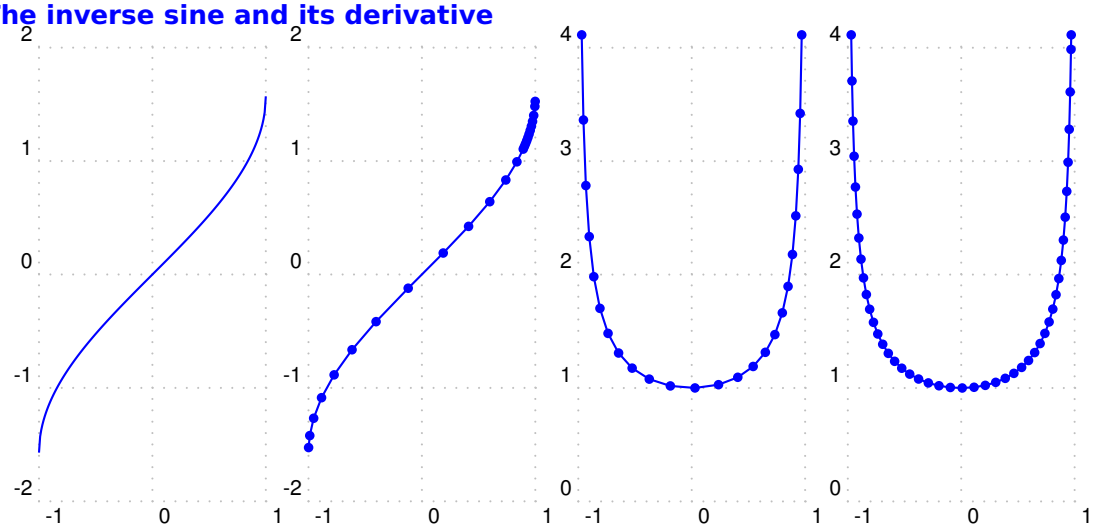

```
\psset{unit=2.5}
\begin{pspicture}[showgrid=true](-1,-1)(1,1)
\parametricplot[algebraic,linecolor=red,VarStep=true, showpoints=true,
            VarStepEpsilon=.0001]
            {0}{47.115}{cos(5*t)|sin(3*t)}
\end{pspicture}
\begin{pspicture}[showgrid=true](-1,-1)(1,1)
\parametricplot[algebraic,linecolor=blue,VarStep=true, showpoints=false,
            VarStepEpsilon=.0001]
            {0}{47.115}{cos(5*t)|sin(3*t)}
\end{pspicture}
```



```
\psset{xunit=.5}
\begin{pspicture}[showgrid=true](0,0)(12.566,2)
\parametricplot[algebraic,linecolor=red,VarStep, showpoints=true,
      VarStepEpsilon=.01]{0}{12.566}{t+cos(-t-Pi/2)|1+sin(-t-Pi/2)}
\end{pspicture}
%
\begin{pspicture}[showgrid=true](0,0)(12.566,2)
\parametricplot[algebraic,linecolor=blue,VarStep, showpoints=false,
      VarStepEpsilon=.001]{0}{12.566}{t+cos(-t-Pi/2)|1+sin(-t-Pi/2)}
\end{pspicture}
```

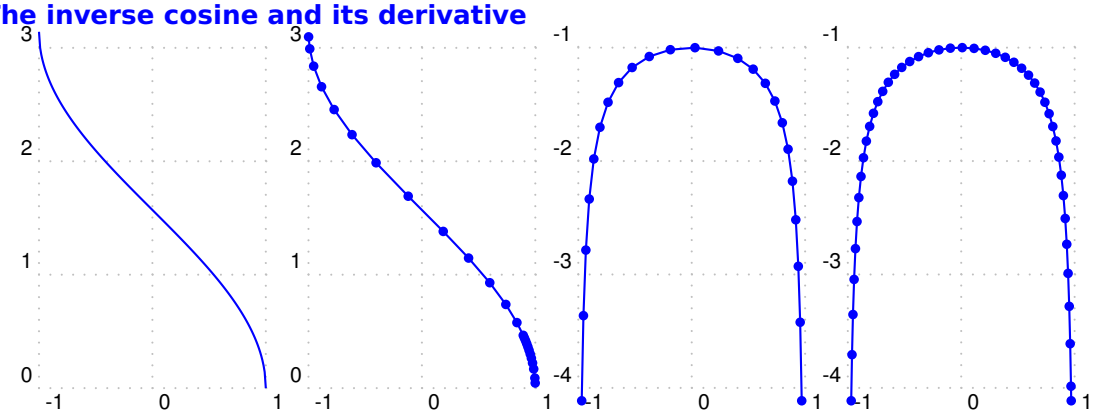
25. New math functions and their derivatives

25.1. The inverse sine and its derivative



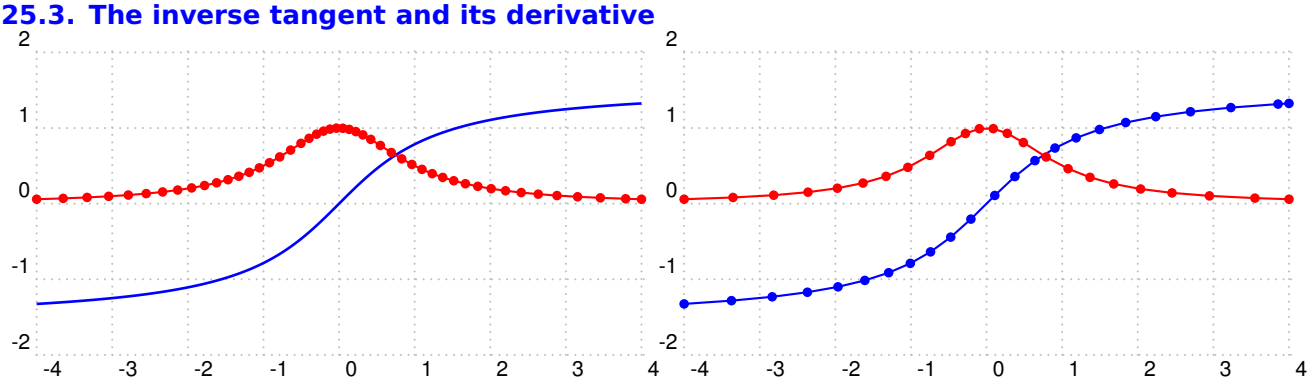
```
\psset{unit=1.5}
\begin{pspicture}[showgrid=true](-1,-2)(1,2)
  \psplot[linecolor=blue,algebraic]{-1}{1}{asin(x)}
\end{pspicture}
\hspace{1em}
\psset{algebraic, VarStep, VarStepEpsilon=.001, showpoints=true}
\begin{pspicture}[showgrid=true](-1,-2)(1,2)
  \psplot[linecolor=blue]{-.999}{.999}{asin(x)}
\end{pspicture}
\hspace{1em}
\begin{pspicture}[showgrid=true](-1,0)(1,4)
  \psplot[linecolor=red]{-.97}{.97}{Derive(1,asin(x))}
\end{pspicture}
\hspace{1em}
\psset{algebraic, VarStep, VarStepEpsilon=.0001, showpoints=true}
\begin{pspicture}[showgrid=true](-1,0)(1,4)
  \psplot[linecolor=red]{-.97}{.97}{Derive(1,asin(x))}
\end{pspicture}
```

25.2. The inverse cosine and its derivative



```
\psset{unit=1.5}
\begin{pspicture}[showgrid=true](-1,0)(1,3)
  \psplot[linecolor=blue,algebraic]{-1}{1}{acos(x)}
\end{pspicture}
\hspace{1em}
\psset{algebraic, VarStep, VarStepEpsilon=.001, showpoints=true}
\begin{pspicture}[showgrid=true](-1,0)(1,3)
  \psplot[linecolor=blue]{-.999}{.999}{acos(x)}
\end{pspicture}
\hspace{1em}
\begin{pspicture}[showgrid=true](-1,-4)(1,-1)
  \psplot[linecolor=red]{-.97}{.97}{Derive(1,acos(x))}
\end{pspicture}
\hspace{1em}
\psset{algebraic, VarStep, VarStepEpsilon=.0001, showpoints=true}
\begin{pspicture}[showgrid=true](-1,-4)(1,-1)
  \psplot[linecolor=red]{-.97}{.97}{Derive(1,acos(x))}
\end{pspicture}
```

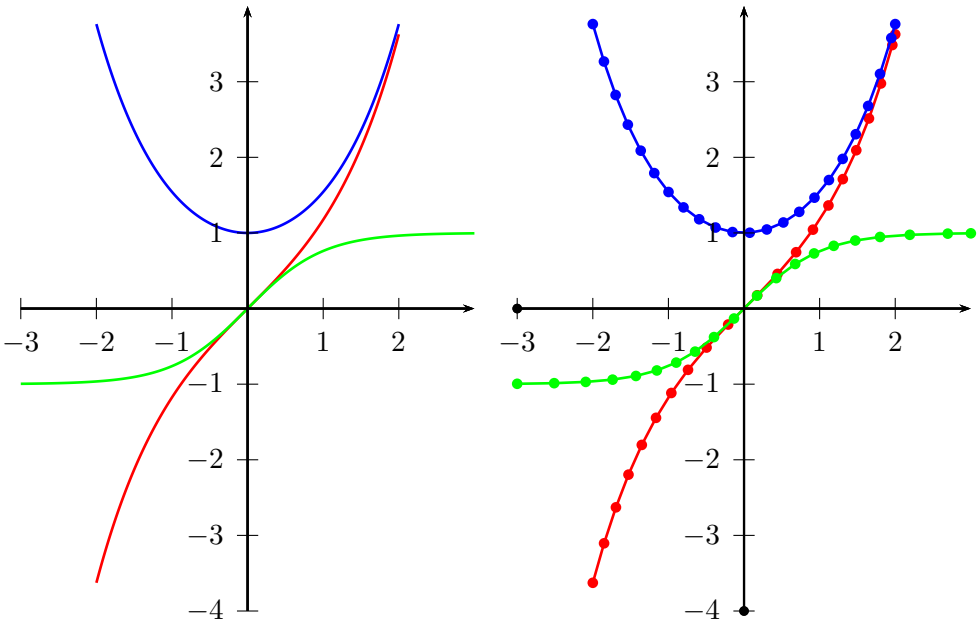
25.3. The inverse tangent and its derivative



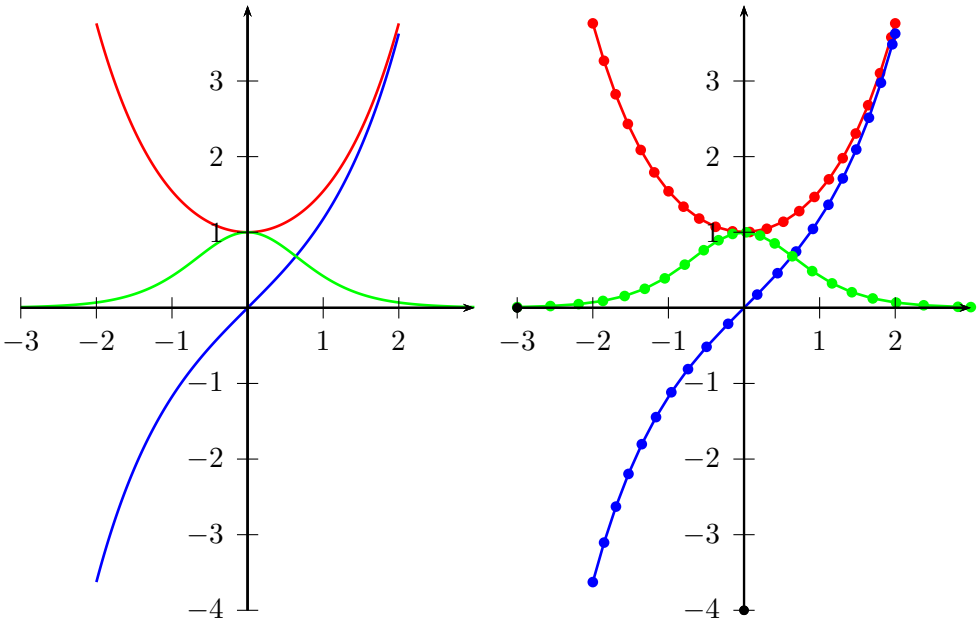
```
\begin{pspicture}[showgrid=true](-4,-2)(4,2)
\psset{algebraic}
  \psplot[linecolor=blue,linewidth=1pt]{-4}{4}{atg(x)}
  \psplot[linecolor=red,VarStep, VarStepEpsilon=.0001, showpoints=true]{-4}{4}{Derive(1,
      atg(x))}
\end{pspicture}
\hspace{1em}
\begin{pspicture}[showgrid=true](-4,-2)(4,2)
\psset{algebraic, VarStep, VarStepEpsilon=.001, showpoints=true}
```

```
  \psplot[linecolor=blue]{-4}{4}{atg(x)}
  \psplot[linecolor=red]{-4}{4}{Derive(1,atg(x))}
\end{pspicture}
```

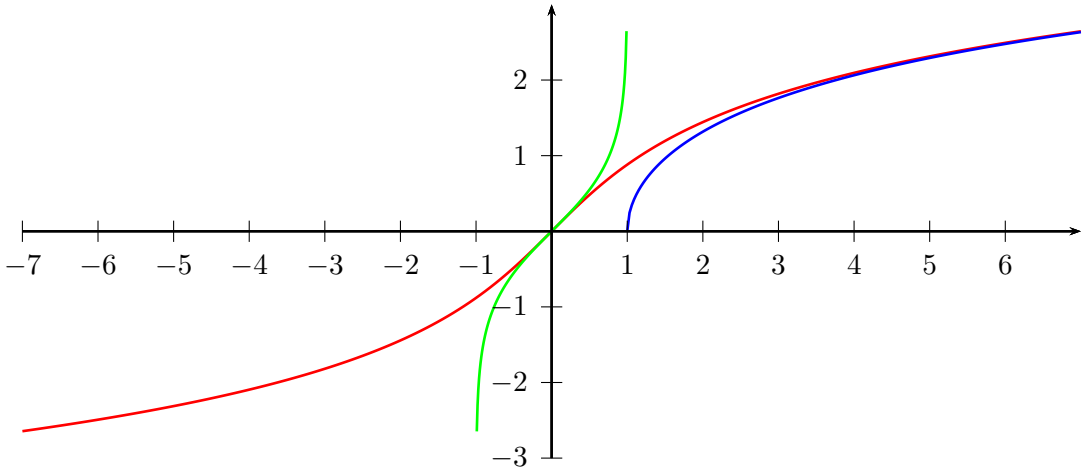
25.4. Hyperbolic functions

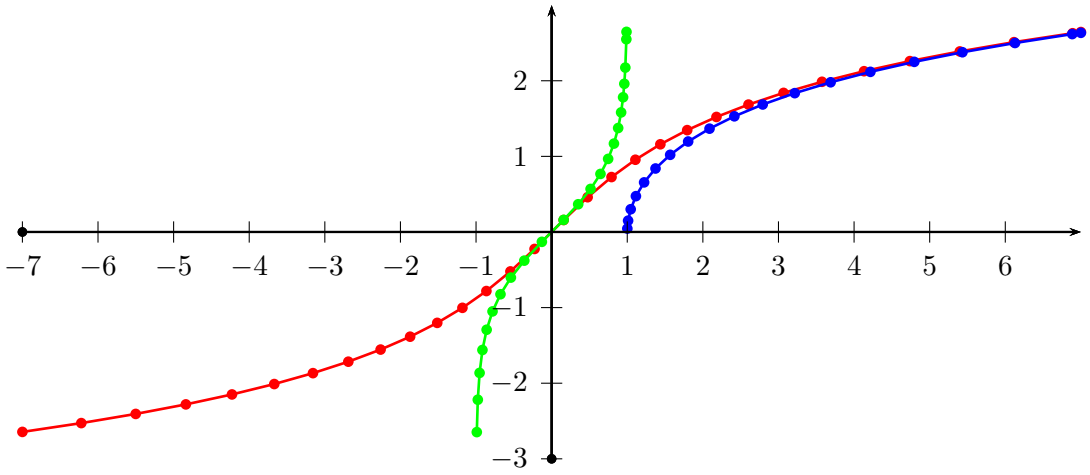



```
\begin{pspicture}(-3,-4)(3,4)
\psset{algebraic}
  \psplot[linecolor=red,linewidth=1pt]{-2}{2}{sh(x)}
  \psplot[linecolor=blue,linewidth=1pt]{-2}{2}{ch(x)}
  \psplot[linecolor=green,linewidth=1pt]{-3}{3}{th(x)}
  \psaxes{->}(0,0)(-3,-4)(3,4)
\end{pspicture}
\hspace{1em}
\begin{pspicture}(-3,-4)(3,4)
\psset{algebraic, VarStep=true, VarStepEpsilon=.001, showpoints=true}
  \psplot[linecolor=red,linewidth=1pt]{-2}{2}{sh(x)}
  \psplot[linecolor=blue,linewidth=1pt]{-2}{2}{ch(x)}
  \psplot[linecolor=green,linewidth=1pt]{-3}{3}{th(x)}
  \psaxes{->}(0,0)(-3,-4)(3,4)
\end{pspicture}
```

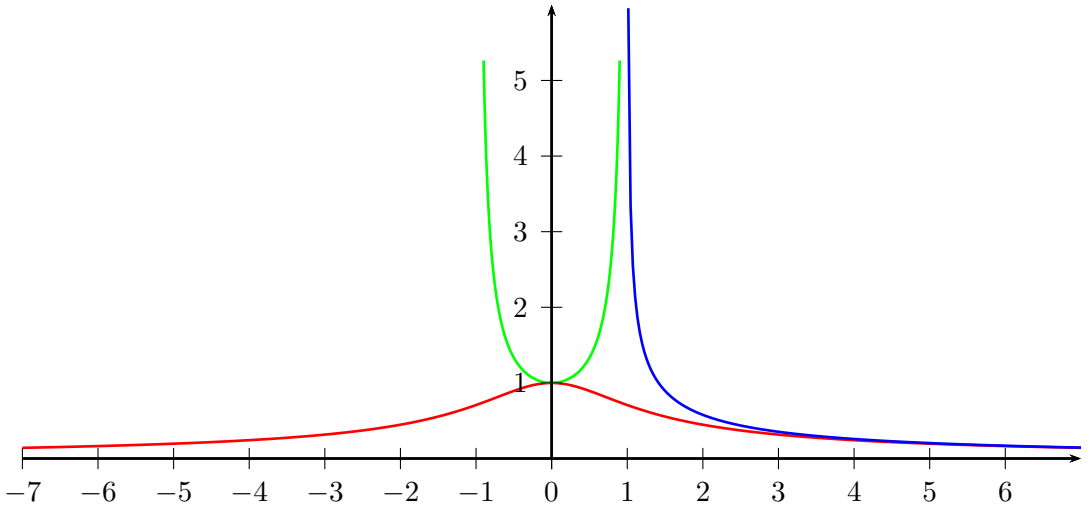


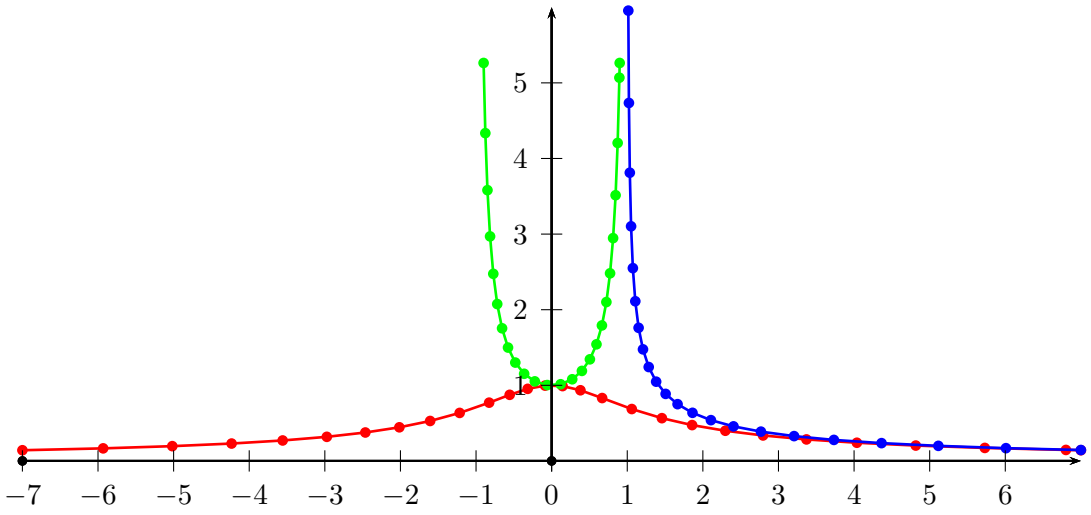

```
\begin{pspicture}(-3,-4)(3,4)
\psset{algebraic,linewidth=1pt}
  \psplot[linecolor=red,linewidth=1pt]{-2}{2}{Derive(1,sh(x))}
  \psplot[linecolor=blue,linewidth=1pt]{-2}{2}{Derive(1,ch(x))}
  \psplot[linecolor=green,linewidth=1pt]{-3}{3}{Derive(1,th(x))}
  \psaxes{->}(0,0)(-3,-4)(3,4)
\end{pspicture}
\hspace{1em}
\begin{pspicture}(-3,-4)(3,4)
\psset{algebraic, VarStep=true, VarStepEpsilon=.001, showpoints=true}
  \psplot[linecolor=red,linewidth=1pt]{-2}{2}{Derive(1,sh(x))}
  \psplot[linecolor=blue,linewidth=1pt]{-2}{2}{Derive(1,ch(x))}
  \psplot[linecolor=green,linewidth=1pt]{-3}{3}{Derive(1,th(x))}
  \psaxes{->}(0,0)(-3,-4)(3,4)
\end{pspicture}
```






```
\begin{pspicture}(-7,-3)(7,3)
\psset{algebraic}
  \psplot[linecolor=red,linewidth=1pt]{-7}{7}{Argsh(x)}
  \psplot[linecolor=blue,linewidth=1pt]{1}{7}{Argch(x)}
  \psplot[linecolor=green,linewidth=1pt]{-.99}{.99}{Argth(x)}
  \psaxes{->}(0,0)(-7,-3)(7,3)
\end{pspicture}\\[\baselineskip]
\begin{pspicture}(-7,-3)(7,3)
  \psset{algebraic, VarStep, VarStepEpsilon=.001, showpoints=true}
  \psplot[linecolor=red,linewidth=1pt]{-7}{7}{Argsh(x)}
  \psplot[linecolor=blue,linewidth=1pt]{1.001}{7}{Argch(x)}
  \psplot[linecolor=green,linewidth=1pt]{-.99}{.99}{Argth(x)}
  \psaxes{->}(0,0)(-7,-3)(7,3)
\end{pspicture}
```






```
\begin{pspicture}(-7,-0.5)(7,6)
\psset{algebraic}
  \psplot[linecolor=red,linewidth=1pt]{-7}{7}{Derive(1,Argsh(x))}
  \psplot[linecolor=blue,linewidth=1pt]{1.014}{7}{Derive(1,Argch(x))}
  \psplot[linecolor=green,linewidth=1pt]{-.9}{.9}{Derive(1,Argth(x))}
  \psaxes{->}(0,0)(-7,0)(7,6)
\end{pspicture}\\[\baselineskip]
\begin{pspicture}(-7,-0.5)(7,6)
\psset{algebraic}
  \psset{algebraic, VarStep=true, VarStepEpsilon=.001, showpoints=true}
  \psplot[linecolor=red,linewidth=1pt]{-7}{7}{Derive(1,Argsh(x))}
  \psplot[linecolor=blue,linewidth=1pt]{1.014}{7}{Derive(1,Argch(x))}
  \psplot[linecolor=green,linewidth=1pt]{-.9}{.9}{Derive(1,Argth(x))}
  \psaxes{->}(0,0)(-7,0)(7,6)
\end{pspicture}
```

26. \psplotDiffEqn – solving diffential equations

A differential equation of first order is like

$$y' = f(x, y, y') \tag{1}$$

where y is a function of x. We define some vectors $Y = [y, y', \cdots, y^{(n-1)}]$ and $Y' = [y', y'', \cdots, y^n]$, depending on the order n. The syntax of the macro is

```
\psplotDiffEqn [Options] {x0}{x1}{y0}{f(x,y,y',...)}
```

- `options`: the `\psplotDiffEqn` specific options and all other of PSTricks, which make sense;
- x_0: the start value;
- x_1: the end value of the definition interval;
- y_0: the initial values for $y(x_0)$ $y'(x_0)$ ...;
- $f(x, y, y', ...)$: the differential equation, depending to the number of initial values, e.g.: `{0 1}` for y_0 are two initial values, so that we have a differential equation of second order $f(x, y, y')$ and the macro leaves y y' on the stack.

The new options are:

- `method`: integration method (`euler` for order 1 euler method, `rk4` for 4^{th} order Runge-Kutta method);
- `whichabs`: select the abscissa for plotting the graph, by default it is x, but you can specify a number which represent a position in the vector y;
- `whichord`: same as precedent for the ordinate, by default $y(0)$;
- `plotfuncx`: describe a ps function for the abscissa, parameter `whichabs` becomes useless;
- `plotfuncy`: idem for the ordinate;
- `buildvector`: boolean parameter for specifying the input-output of the f description:

  `true` (default): y is put on the stack element by element, y' must be given in the same way;

  `false` : y is put on the stack as a vector, y' must be returned in the same way;
- `algebraic`: algebraic description for f, `buildvector` parameter is useless when activating this option.

26.1. Variable step for differential equations

A new algorithm has been added for adjusting the step according to the variations of the curve. The parameter `method` has a new possible value : `varrkiv` to activate the Runge-Kutta method with variable step, then the parameter `varsteptol` (real value; `.01` by default) can control the tolerance of the algortihm.

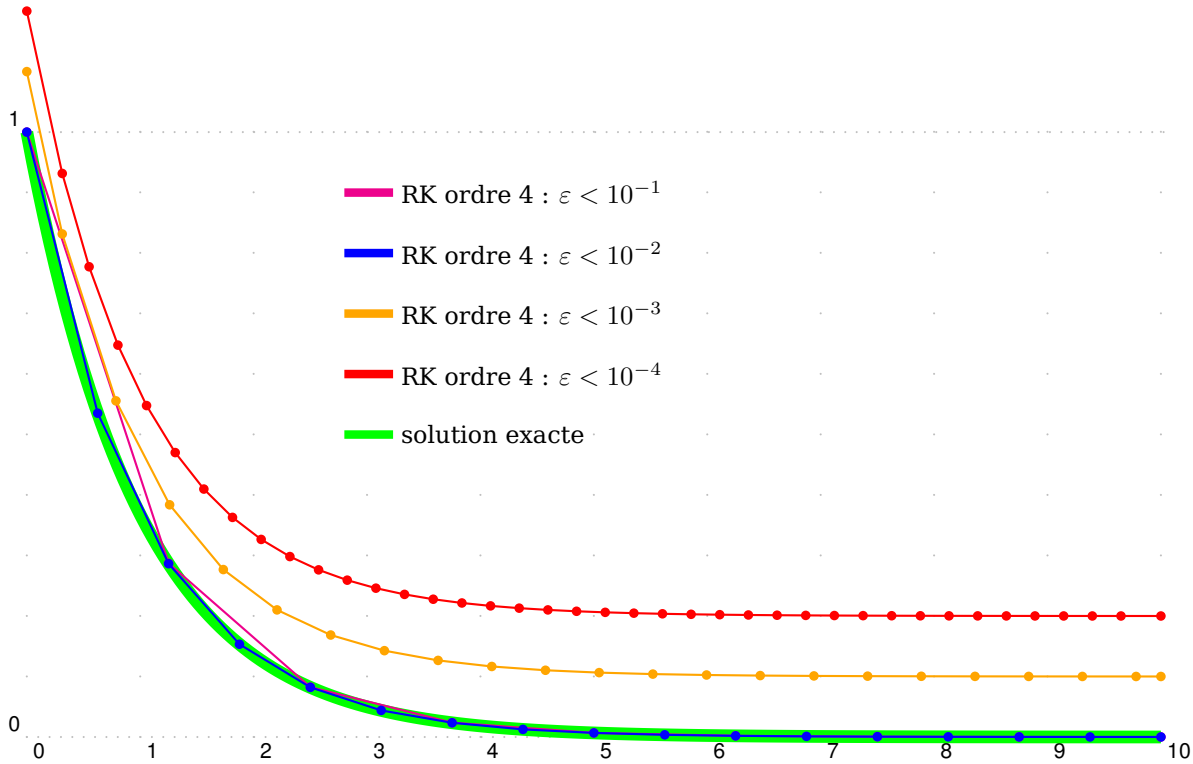


Figure 1: Equation $y' = -y$ with $y_0 = 1$.

```
\def\Funct{neg}\def\FunctAlg{-y[0]}
\psset{xunit=1.5, yunit=8, showpoints=true}
\begin{pspicture}[showgrid=true](0,0)(10,1.2)
  \psplot[linewidth=6\pslinewidth, linecolor=green, showpoints=false]{0}{10}{Euler x neg
      exp}
  \psplotDiffEqn[linecolor=magenta, method=varrkiv, varsteptol=.1, plotpoints
      =2]{0}{10}{1}{\Funct}
  \rput(0,.0){\psplotDiffEqn[linecolor=blue, method=varrkiv, varsteptol=.01, plotpoints
      =2]{0}{10}{1}{\Funct}}
  \rput(0,.1){\psplotDiffEqn[linecolor=Orange, method=varrkiv, varsteptol=.001, plotpoints
      =2]{0}{10}{1}{\Funct}}
  \rput(0,.2){\psplotDiffEqn[linecolor=red, method=varrkiv, varsteptol=.0001, plotpoints
      =2]{0}{10}{1}{\Funct}}
  \psset{linewidth=4\pslinewidth,showpoints=false}
  \rput*(3.3,.9){\psline[linecolor=magenta](-.75cm,0)}
  \rput*[l](3.3,.9){\small RK ordre 4 : $\varepsilon<10^{-1}$}
  \rput*(3.3,.8){\psline[linecolor=blue](-.75cm,0)}
  \rput*[l](3.3,.8){\small RK ordre 4 : $\varepsilon<10^{-2}$}
  \rput*(3.3,.7){\psline[linecolor=Orange](-.75cm,0)}
  \rput*[l](3.3,.7){\small RK ordre 4 : $\varepsilon<10^{-3}$}
  \rput*(3.3,.6){\psline[linecolor=red](-.75cm,0)}
  \rput*[l](3.3,.6){\small RK ordre 4 : $\varepsilon<10^{-4}$}
```

```
  \rput*(3.3,.5){\psline[linecolor=green](-.75cm,0)}
  \rput*[l](3.3,.5){\small solution exacte}
\end{pspicture}
```

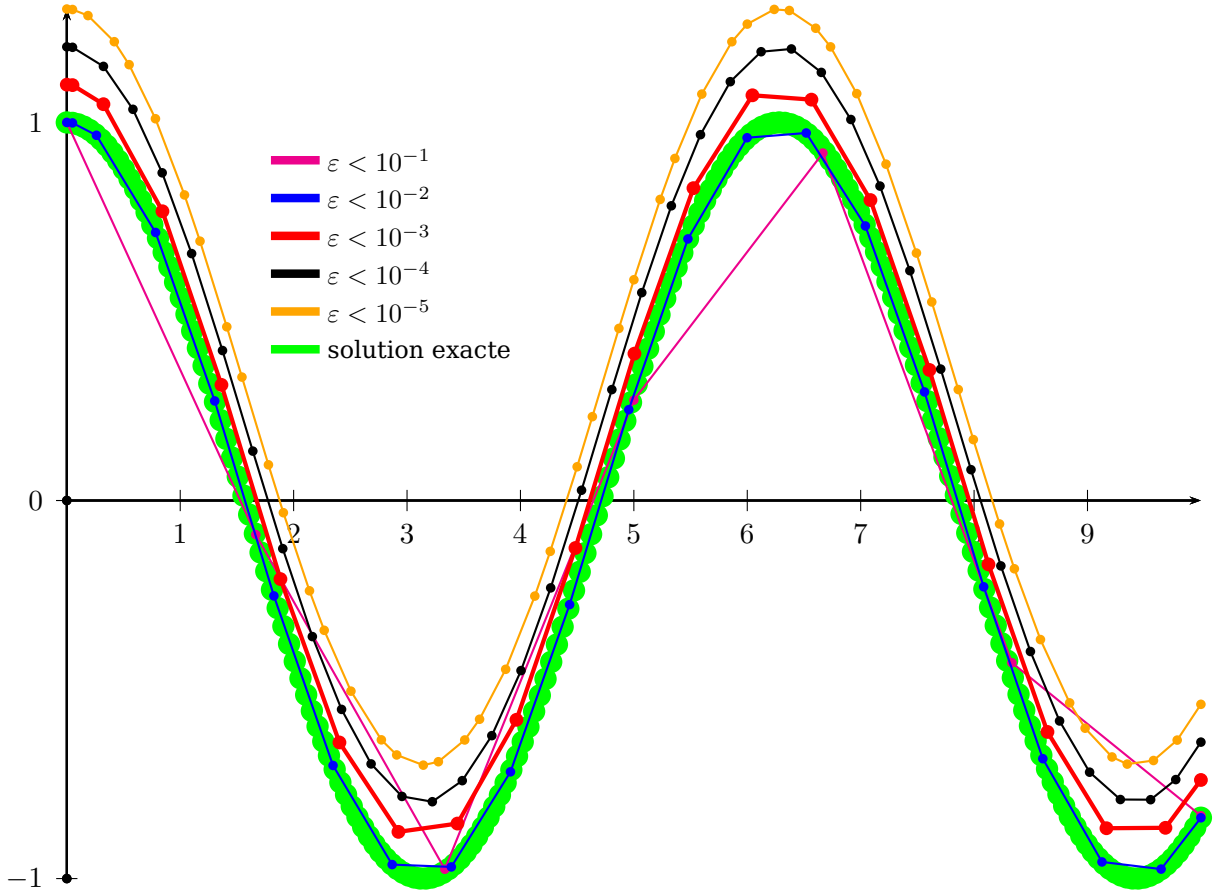


Figure 2: Equation $y'' = -y$

```
\def\Funct{exch neg}
\psset{xunit=1.5, yunit=5, method=varrkiv, showpoints=true}%%
\def\quatrepi{12.5663706144}
\begin{pspicture}(0,-1)(10,1.3)
  \psaxes{->}(0,0)(0,-1)(10,1.3)
  \psplot[linewidth=4\pslinewidth, linecolor=green, algebraic]{0}{10}{cos(x)}
  \rput(0,.0){\psplotDiffEqn[linecolor=magenta, plotpoints=7, varsteptol=.1]{0}{10}{1 0}{\
      Funct}}
  \rput(0,.0){\psplotDiffEqn[linecolor=blue, plotpoints=201, varsteptol=.01]{0}{10}{1 0}{\
      Funct}}
  \rput(0,.1){\psplotDiffEqn[linewidth=2\pslinewidth, linecolor=red, varsteptol
      =.001]{0}{10}{1 0}{\Funct}}
  \rput(0,.2){\psplotDiffEqn[linecolor=black, varsteptol=.0001]{0}{10}{1 0}{\Funct}}
  \rput(0,.3){\psplotDiffEqn[linecolor=Orange, varsteptol=.00001]{0}{10}{1 0}{\Funct}}
  \psset{linewidth=4\pslinewidth,showpoints=false}
  \rput*(2.3,.9){\psline[linecolor=magenta](-.75cm,0)}
  \rput*[l](2.3,.9){\small $\varepsilon<10^{-1}$}
  \rput*(2.3,.8){\psline[linecolor=blue](-.75cm,0)}
  \rput*[l](2.3,.8){\small $\varepsilon<10^{-2}$}
  \rput*(2.3,.7){\psline[linecolor=red](-.75cm,0)}
```

```
  \rput*[l](2.3,.7){\small $\varepsilon<10^{-3}$}
  \rput*(2.3,.6){\psline[linecolor=black](-.75cm,0)}
  \rput*[l](2.3,.6){\small $\varepsilon<10^{-4}$}
  \rput*(2.3,.5){\psline[linecolor=Orange](-.75cm,0)}
  \rput*[l](2.3,.5){\small $\varepsilon<10^{-5}$}
  \rput*(2.3,.4){\psline[linecolor=green](-.75cm,0)}
  \rput*[l](2.3,.4){\small solution exacte}
\end{pspicture}
```

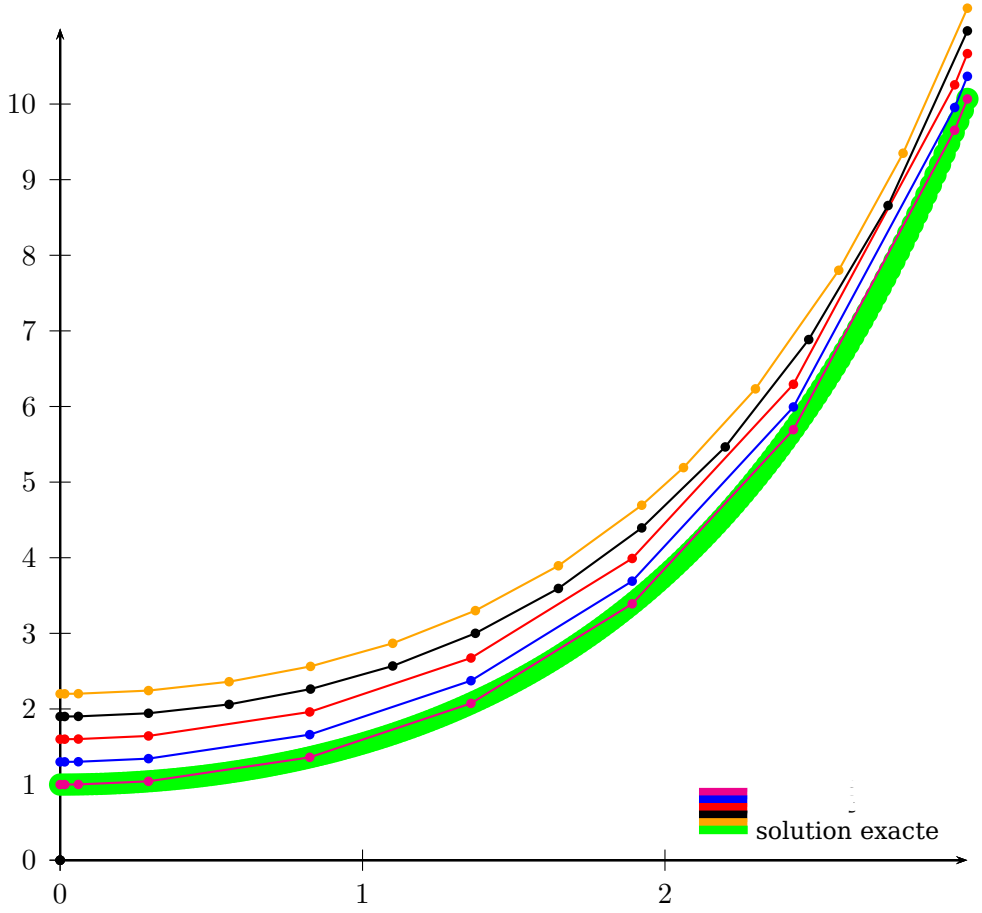


Figure 3: Equation $y'' = y$

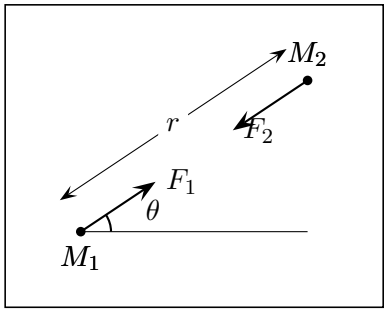
```
\def\Funct{exch}
\psset{xunit=4, yunit=1, method=varrkiv, showpoints=true}%%
\def\quatrepi{12.5663706144}
\begin{pspicture}(0,-0.5)(3,11)
  \psaxes{->}(0,0)(3,11)
  \psplot[linewidth=4\pslinewidth, linecolor=green, algebraic]{0}{3}{ch(x)}
  \rput(0,.0){\psplotDiffEqn[linecolor=magenta, varsteptol=.1]{0}{3}{1 0}{\Funct}}
  \rput(0,.3){\psplotDiffEqn[linecolor=blue, varsteptol=.01]{0}{3}{1 0}{\Funct}}
  \rput(0,.6){\psplotDiffEqn[linecolor=red, varsteptol=.001]{0}{3}{1 0}{\Funct}}
  \rput(0,.9){\psplotDiffEqn[linecolor=black, varsteptol=.0001]{0}{3}{1 0}{\Funct}}
  \rput(0,1.2){\psplotDiffEqn[linecolor=Orange, varsteptol=.00001]{0}{3}{1 0}{\Funct}}
  \psset{linewidth=4\pslinewidth,showpoints=false}
  \rput*(2.3,.9){\psline[linecolor=magenta](-.75cm,0)}
  \rput*[l](2.3,.9){\small $\varepsilon<10^{-1}$}
  \rput*(2.3,.8){\psline[linecolor=blue](-.75cm,0)}
```

```
  \rput*[l](2.3,.8){\small $\varepsilon<10^{-2}$}
  \rput*(2.3,.7){\psline[linecolor=red](-.75cm,0)}
  \rput*[l](2.3,.7){\small $\varepsilon<10^{-3}$}
  \rput*(2.3,.6){\psline[linecolor=black](-.75cm,0)}
  \rput*[l](2.3,.6){\small $\varepsilon<10^{-4}$}
  \rput*(2.3,.5){\psline[linecolor=Orange](-.75cm,0)}
  \rput*[l](2.3,.5){\small $\varepsilon<10^{-5}$}
  \rput*(2.3,.4){\psline[linecolor=green](-.75cm,0)}
  \rput*[l](2.3,.4){\small solution exacte}
\end{pspicture}
```

26.2. Equation of second order

Here is the traditional simulation of two stars attracting each other according to the classical gravitation law in $\frac{1}{r^2}$. In 2-Dimensions, the system to be solved is composed of four second order differential equations. In order to be described, each of them gives two first order equations, then we obtain a 8 sized vectorial equation. In the following example the masses of the stars are 1 and 20.

$$\begin{cases} x_1'' = \dfrac{M_2}{r^2}\cos(\theta) \\ y_1'' = \dfrac{M_2}{r^2}\sin(\theta) \\ x_2'' = \dfrac{M_1}{r^2}\cos(\theta) \\ y_2'' = \dfrac{M_1}{r^2}\sin(\theta) \end{cases} \text{ avec } \begin{cases} r^2 = (x_1 - x_2)^2 + (y_1 - y_2)^2 \\ \cos(\theta) = \dfrac{(x_1 - x_2)}{r} \\ \sin(\theta) = \dfrac{(y_1 - y_2)}{r} \end{cases}$$

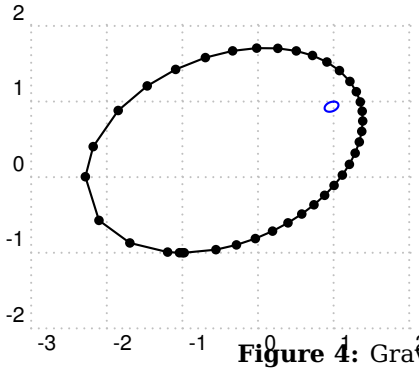



```
                                                    %% x1 y1 x'1 y'1 x2 y2 x'2 y'2
/yp2 exch def /xp2 exch def /ay2 exch def /ax2 exch def %% mise en variables
/yp1 exch def /xp1 exch def /ay1 exch def /ax1 exch def %% mise en variables
/ro2 ax2 ax1 sub dup mul ay2 ay1 sub dup mul add def    %% calcul de r*r
xp1 yp1                                                 %%
ax2 ax1 sub ro2 sqrt div ro2 div                        %% calcul de x"1
ay2 ay1 sub ro2 sqrt div ro2 div                        %% calcul de y"1
xp2 yp2                                                 %%
3 index -20 mul                                         %% calcul de x"2=-20x"1
3 index -20 mul                                         %% calcul de y"2=-20y"1
```

Table 1: PostScriptsource code for the gravitational interaction

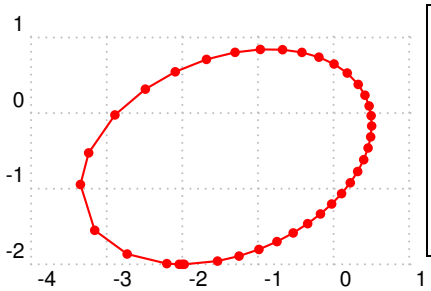
```
y[2]|                                                   %% y'[0]
y[3]|                                                   %% y'[1]
(y[4]-y[0])/((y[4]-y[0])^2+(y[5]-y[1])^2)^1.5|          %% y'[2]=y"[0]
(y[5]-y[1])/((y[4]-y[0])^2+(y[5]-y[1])^2)^1.5|          %% y'[3]=y"[1]
y[6]|                                                   %% y'[4]
y[7]|                                                   %% y'[5]
20*(y[0]-y[4])/((y[4]-y[0])^2+(y[5]-y[1])^2)^1.5|       %% y'[6]=y"[4]
20*(y[1]-y[5])/((y[4]-y[0])^2+(y[5]-y[1])^2)^1.5        %% y'[7]=y"[5]
```

Table 2: Algebraic description for the gravitational interaction




```
\def\InitCond{ 1 1 .1 0 -1 -1 -2 0}
\begin{pspicture}[shift=-2,showgrid=true](-3,-1.75)(2,1.5)
  \psplotDiffEqn[whichabs=0, whichord=1, linecolor=blue, method=rk4,
      plotpoints=100]{0}{3.95}{\InitCond}{\Grav}
  \psset{showpoints=true,whichabs=4, whichord=5}
  \psplotDiffEqn[linecolor=black, method=varrkiv, varsteptol=.0001,
      plotpoints=200]{0}{3.9}{\InitCond}{\Grav}
\end{pspicture}
```

Figure 4: Gravitational interaction: fixed landmark, trajectory of the stars




```
\def\InitCond{ 1 1 .1 0 -1 -1 -2 0}
\begin{pspicture}[shift=-1.5,showgrid=true](-4,-1.75)(1,1)
 \psplotDiffEqn[linecolor=red, plotpoints=200,method=varrkiv, varsteptol
    =.0001, showpoints=true,
    plotfuncx=y dup 4 get exch 0 get sub,
    plotfuncy=dup 5 get exch 1 get sub ]{0}{3.9}{\InitCond}{\Grav}
\end{pspicture}
```

Figure 5: Gravitational interaction : landmark defined by one star

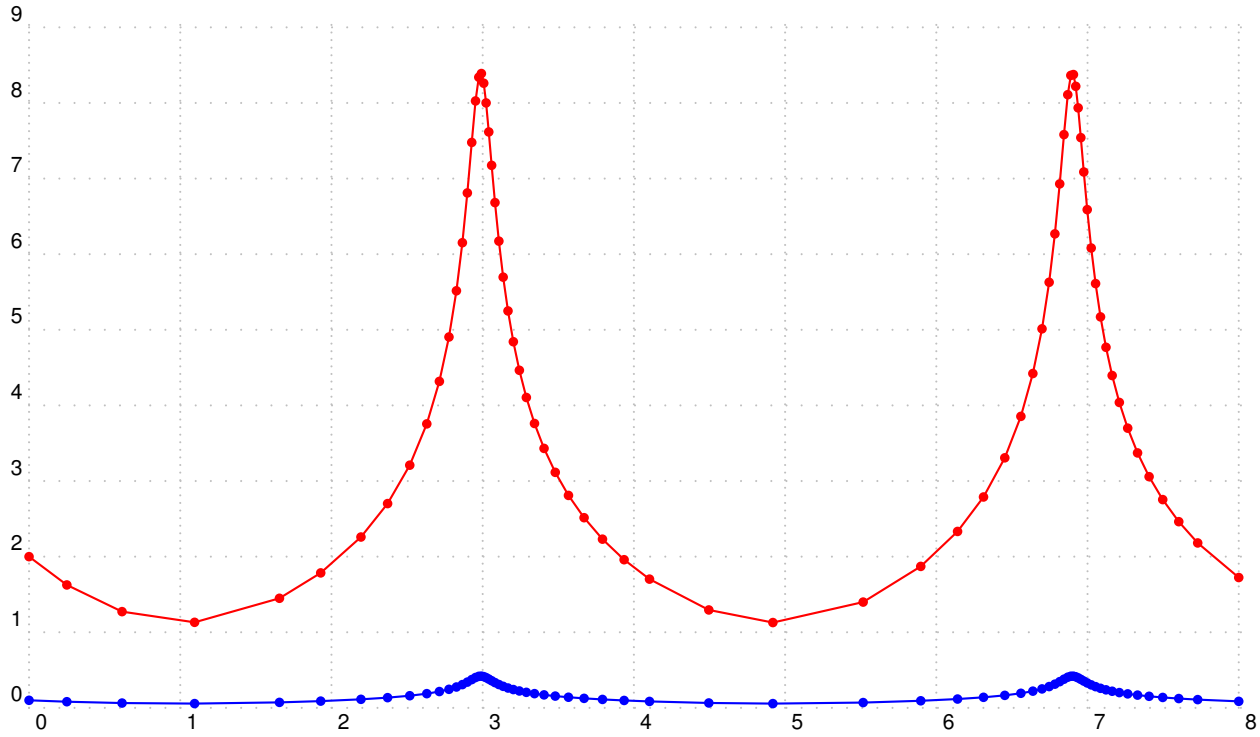
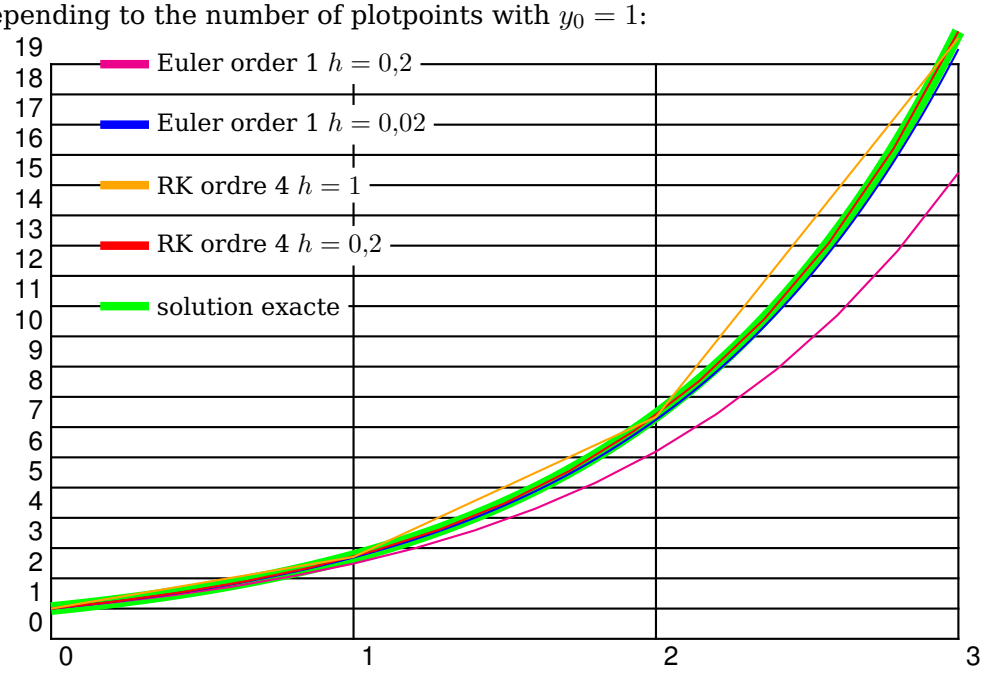


Figure 6: Gravitational interaction : speeds of the stars

```
\psset{xunit=2}
\begin{pspicture}[showgrid=true](0,0)(8,9)
 \psset{showpoints=true}
 \psplotDiffEqn[linecolor=red, method=varrkiv, plotpoints=2, varsteptol=.0001,
    plotfuncy=dup 6 get dup mul exch 7 get dup mul add sqrt]{0}{8}{\InitCond}{\Grav}
 \psplotDiffEqn[linecolor=blue, method=varrkiv, plotpoints=2, varsteptol=.0001,
    plotfuncy=dup 2 get dup mul exch 3 get dup mul add sqrt]{0}{8}{\InitCond}{\Grav}
\end{pspicture}
```

Simple equation of first order $y' = y$

For the initial value $y(0) = 1$ we have the solution $y(x) = e^x$. y is always on the stack, so we have to do nothing. Using the `algebraic` option, we write it as `y[0]`. The following example shows different solutions depending to the number of plotpoints with $y_0 = 1$:




```
\psset{xunit=4, yunit=.4}
\begin{pspicture}(3,19)\psgrid[subgriddiv=1]
  \psplot[linewidth=6\pslinewidth, linecolor=green]{0}{3}{Euler x exp}
  \psplotDiffEqn[linecolor=magenta,plotpoints=16,algebraic]{0}{3}{1}{y[0]}
  \psplotDiffEqn[linecolor=blue,plotpoints=151]{0}{3}{1}{}
  \psplotDiffEqn[linecolor=red,method=rk4,plotpoints=15]{0}{3}{1}{}
  \psplotDiffEqn[linecolor=Orange,method=rk4,plotpoints=4]{0}{3}{1}{}
  \psset{linewidth=4\pslinewidth}
  \rput*(0.35,19){\psline[linecolor=magenta](-.75cm,0)}
  \rput*[l](0.35,19){\small Euler order 1 $h=0{,}2$}
  \rput*(0.35,17){\psline[linecolor=blue](-.75cm,0)}
  \rput*[l](0.35,17){\small Euler order 1 $h=0{,}02$}
  \rput*(0.35,15){\psline[linecolor=Orange](-.75cm,0)}
  \rput*[l](0.35,15){\small RK ordre 4 $h=1$}
  \rput*(0.35,13){\psline[linecolor=red](-.75cm,0)}
  \rput*[l](0.35,13){\small RK ordre 4 $h=0{,}2$}
  \rput*(0.35,11){\psline[linecolor=green](-.75cm,0)}
  \rput*[l](0.35,11){\small solution exacte}
\end{pspicture}
```

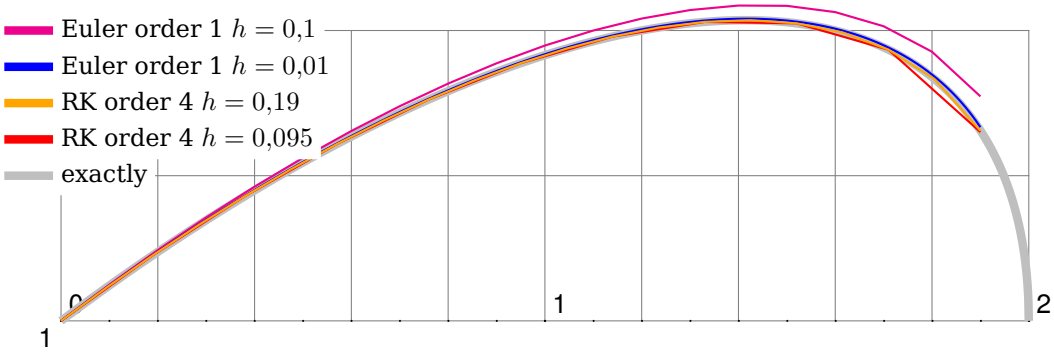
$$y' = \frac{2 - ty}{4 - t^2}$$

For the initial value $y(0) = 1$ the exact solution is $y(x) = \frac{t + \sqrt{4 - t^2}}{2}$. The function f described in PostScript code is like (y is still on the stack):

```
x           %% y x
mul         %% x*y
2 exch sub  %% 2-x*y
4 x dup mul %% 2-x*y 4 x^2
sub         %% 2-x*y 4-x^2
div         %% (2-x*y)/(4-x^2)
```

The following example uses $y_0 = 1$.

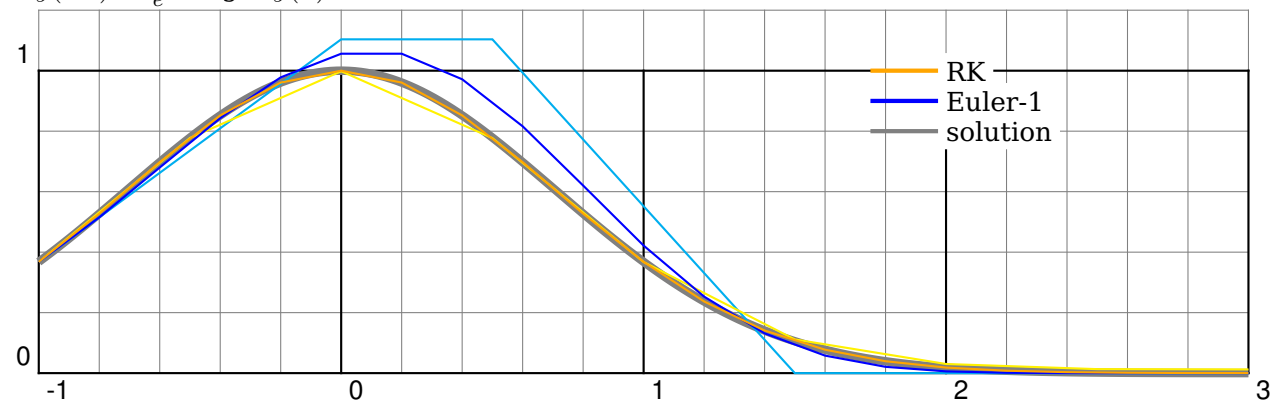
```
\newcommand{\InitCond}{1}
\newcommand{\Func}{x mul 2 exch sub 4 x dup mul sub div}
\newcommand{\FuncAlg}{(2-x*y[0])/(4-x^2)}
```




```
\psset{xunit=6.4, yunit=9.6, showpoints=false}
\begin{pspicture}(0,1)(2,1.7) \psgrid[subgriddiv=5]
 { \psset{linewidth=4\pslinewidth,linecolor=lightgray}
 \psplot{0}{1.8}{x dup dup mul 4 exch sub sqrt add 2 div}
 \psplot{1.8}{2}{x dup dup mul 4 exch sub sqrt add 2 div} }
 \def\InitCond{1}
 \def\Func{x mul 2 exch sub 4 x dup mul sub div}
 \psplotDiffEqn[linecolor=magenta, plotpoints=20]{0}{1.9}{\InitCond}{\Func}
 \psplotDiffEqn[linecolor=blue, plotpoints=191]{0}{1.9}{\InitCond}{\Func}
 \psplotDiffEqn[linecolor=red, method=rk4, plotpoints=11,%
   algebraic]{0}{1.9}{\InitCond}{(2-x*y[0])/(4-x^2)}
 \psplotDiffEqn[linecolor=Orange, method=rk4, plotpoints=21,%
   algebraic]{0}{1.9}{\InitCond}{(2-x*y[0])/(4-x^2)}
 \psset{linewidth=4\pslinewidth}
 \rput*(0.3,1.6){\psline[linecolor=magenta](-.75cm,0)}\rput*[l](0.3,1.6){\small Euler order 1 $h
    =0{,}1$}
 \rput*(0.3,1.55){\psline[linecolor=blue](-.75cm,0)}\rput*[l](0.3,1.55){\small Euler order 1 $h
    =0{,}01$}
 \rput*(0.3,1.5){\psline[linecolor=Orange](-.75cm,0)}\rput*[l](0.3,1.5){\small RK order 4 $h=0{,}19$}
 \rput*(0.3,1.45){\psline[linecolor=red](-.75cm,0)}\rput*[l](0.3,1.45){\small RK order 4 $h=0{,}095$}
 \rput*(0.3,1.4){\psline[linecolor=lightgray](-.75cm,0)}\rput*[l](0.3,1.4){\small exactly}
\end{pspicture}
```

$y' = -2xy$

For $y(-1) = \frac{1}{e}$ we get $y(x) = e^{-x^2}$.




```
\psset{unit=4}
\begin{pspicture}(-1,0)(3,1.1)\psgrid
  \psplot[linewidth=4\pslinewidth,linecolor=gray]{-1}{3}{Euler x dup mul neg exp}
  \psset{plotpoints=9}
  \psplotDiffEqn[linecolor=cyan]{-1}{3}{1 Euler div}{x -2 mul mul}
  \psplotDiffEqn[linecolor=yellow, method=rk4]{-1}{3}{1 Euler div}{x -2 mul mul}
  \psset{plotpoints=21}
  \psplotDiffEqn[linecolor=blue]{-1}{3}{1 Euler div}{x -2 mul mul}
  \psplotDiffEqn[linecolor=Orange, method=rk4]{-1}{3}{1 Euler div}{x -2 mul mul}
  \psset{linewidth=2\pslinewidth}
  \rput*(2,1){\psline[linecolor=Orange](-0.25,0)}
  \rput*[l](2,1){RK}
  \rput*(2,.9){\psline[linecolor=blue](-0.25,0)}
  \rput*[l](2,.9){\textsc{Euler}-1}
  \rput*(2,.8){\psline[linecolor=gray](-0.25,0)}
  \rput*[l](2,.8){solution}
\end{pspicture}
```

Spiral of Cornu

The integrals of Fresnel:

$$x = \int_0^t \cos\frac{\pi t^2}{2}\mathrm{d}t \tag{2}$$

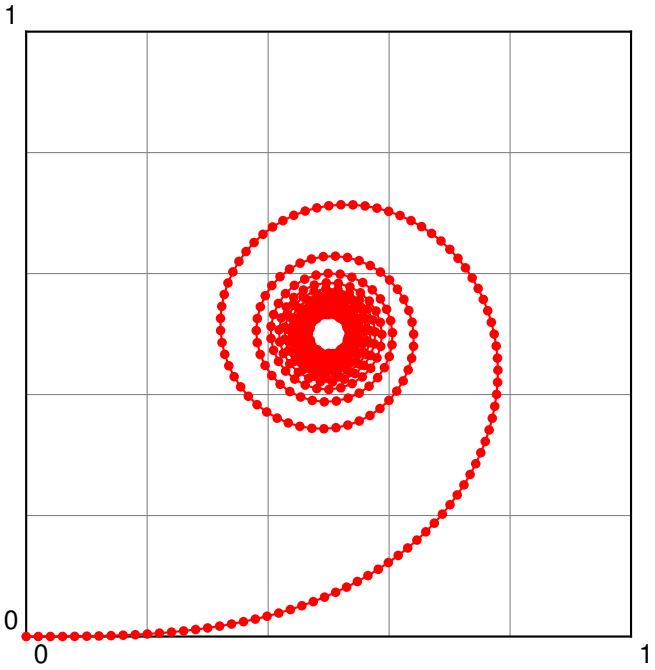
$$y = \int_0^t \sin\frac{\pi t^2}{2}\mathrm{d}t \tag{3}$$

with

$$\dot{x} = \cos\frac{\pi t^2}{2} \tag{4}$$

$$\dot{y} = \sin\frac{\pi t^2}{2} \tag{5}$$

```
\psset{unit=8}
\begin{pspicture}(1,1)\psgrid[subgriddiv=5]
  \psplotDiffEqn[whichabs=0,whichord=1,linecolor=red,method=rk4,algebraic,%
    plotpoints=500,showpoints=true]{0}{10}{0 0}{cos(Pi*x^2/2)|sin(Pi*x^2/2)}
\end{pspicture}
```



Lotka-Volterra

The Lotka-Volterra model describes interactions between two species in an ecosystem, a predator and a prey. This represents our first multi-species model. Since we are considering two species, the model will involve two equations, one which describes how the prey population changes and the second which describes how the predator population changes.

For concreteness let us assume that the prey in our model are rabbits, and that the predators are foxes. If we let $R(t)$ and $F(t)$ represent the number of rabbits and foxes, respectively, that are alive at time t, then the Lotka-Volterra model is:

$$\dot{R} = a \cdot R - b \cdot R \cdot F \tag{6}$$

$$\dot{F} = e \cdot b \cdot R \cdot F - c \cdot F \tag{7}$$

where the parameters are defined by:

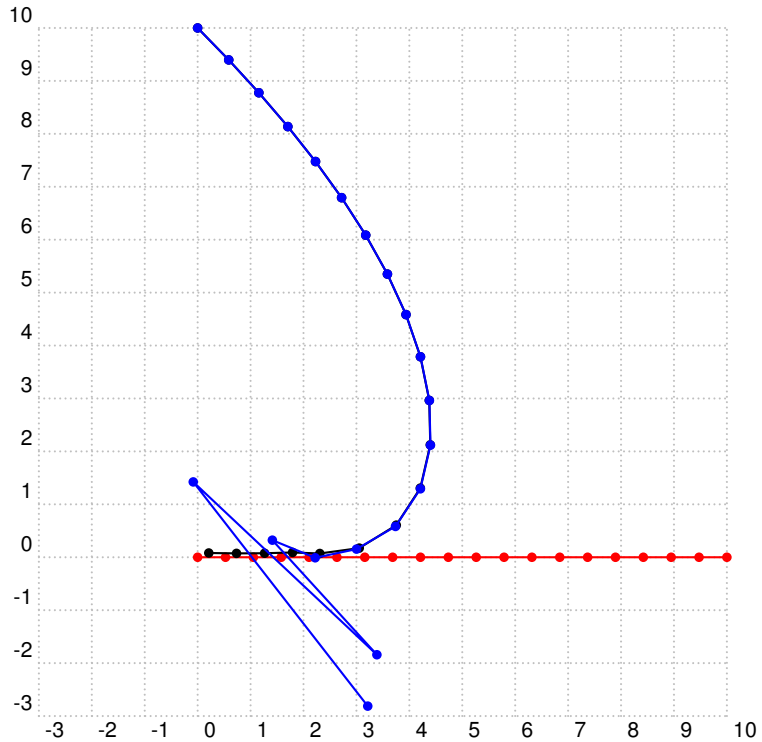
a is the natural growth rate of rabbits in the absence of predation,

c is the natural death rate of foxes in the absence of food (rabbits),

b is the death rate per encounter of rabbits due to predation,

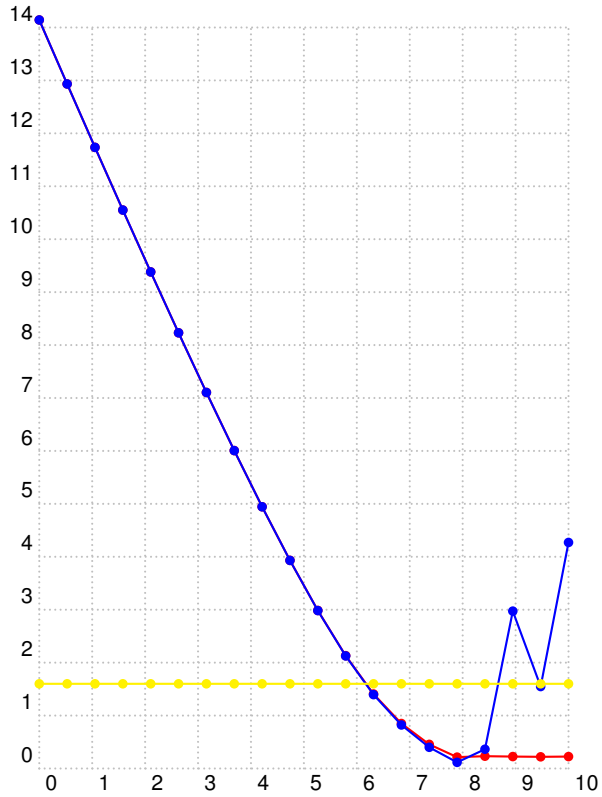
e is the efficiency of turning predated rabbits into foxes.

The Stella model representing the Lotka-Volterra model will be slightly more complex than the single species models we've dealt with before. The main difference is that our model will have two stocks (reservoirs), one for each species. Each species will have its own birth and death rates. In addition, the Lotka-Volterra model involves four parameters rather than two. All told, the Stella representation of the Lotka-Volterra model will use two stocks, four flows, four converters and many connectors.




```
\def\InitCond{ 0 10 10}%% xa ya xl
\def\Faiglelapin{\Vaigle*(y[2]-y[0])/sqrt(y[1]^2+(y[2]-y[0])^2)|%
```

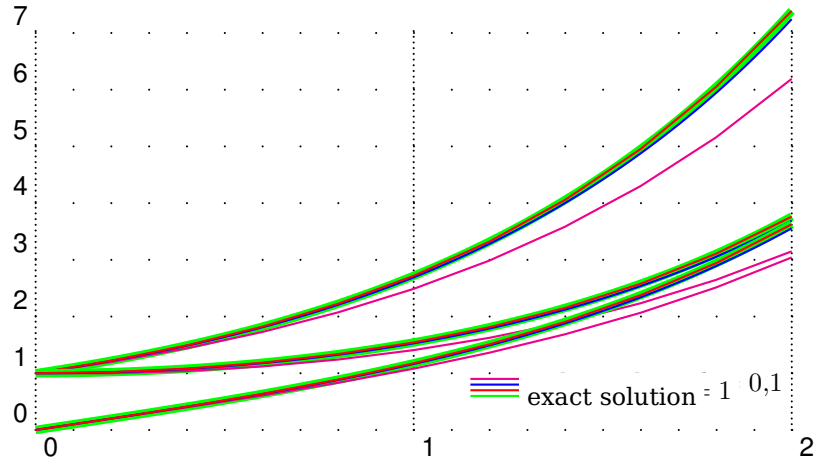
```
            -\Vaigle*y[1]/sqrt(y[1]^2+(y[2]-y[0])^2)|%
            -\Vlapin}
\def\Vlapin{1} \def\Vaigle{1.6}
\psset{unit=.7,subgriddiv=0,gridcolor=lightgray,method=adams,algebraic,%
  plotpoints=20,showpoints=true}
\begin{pspicture}[showgrid=true](-3,-3)(10,10)
 \psplotDiffEqn[plotfuncy=pop 0,whichabs=2,linecolor=red]{0}{10}{\InitCond}{\Faiglelapin}
 \psplotDiffEqn[whichabs=0,whichord=1,linecolor=black,method=rk4]{0}{10}{\InitCond}{\Faiglelapin}
  \psplotDiffEqn[whichabs=0,whichord=1,linecolor=blue]{0}{10}{\InitCond}{\Faiglelapin}
\end{pspicture}
```




```
\def\InitCond{ 0 10 10}%% xa ya xl
\def\Faiglelapin{\Vaigle*(y[2]-y[0])/sqrt(y[1]^2+(y[2]-y[0])^2)|%
            -\Vaigle*y[1]/sqrt(y[1]^2+(y[2]-y[0])^2)|%
            -\Vlapin}
\def\Vlapin{1} \def\Vaigle{1.6}
\psset{unit=.7,subgriddiv=0,gridcolor=lightgray,method=adams,algebraic,%
  plotpoints=20,showpoints=true}
\begin{pspicture}[showgrid=true](10,12)
 \psplotDiffEqn[plotfuncy=dup 1 get dup mul exch dup 0 get exch 2 get sub dup
   mul add sqrt,linecolor=red,method=rk4]{0}{10}{\InitCond}{\Faiglelapin}
 \psplotDiffEqn[plotfuncy=dup 1 get dup mul exch dup 0 get exch 2 get sub dup
   mul add sqrt,linecolor=blue]{0}{10}{\InitCond}{\Faiglelapin}
 \psplotDiffEqn[plotfuncy=pop Func aload pop pop dup mul exch dup mul add sqrt,
   linecolor=yellow]{0}{10}{\InitCond}{\Faiglelapin}
\end{pspicture}
```

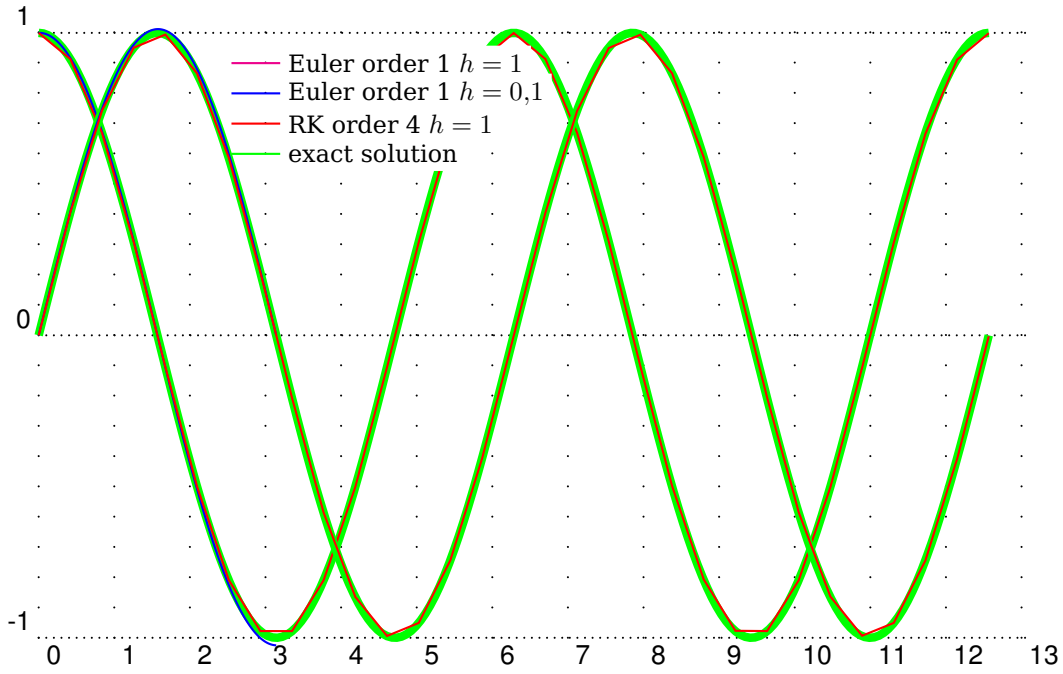
$y'' = y$

Beginning with the initial equation $y(x) = Ae^x + Be^{-x}$ we get the hyperbolic trigonometrical functions.




```
\def\Funct{exch} \psset{xunit=5cm, yunit=0.75cm}
\begin{pspicture}(0,-0.25)(2,7)\psgrid[subgriddiv=1,griddots=10]
 \psplot[linewidth=4\pslinewidth, linecolor=green]{0}{2}{Euler x exp} %%e^x
 \psplotDiffEqn[linecolor=magenta, plotpoints=11]{0}{2}{1 1}{\Funct}
 \psplotDiffEqn[linecolor=blue, plotpoints=101]{0}{2}{1 1}{\Funct}
 \psplotDiffEqn[linecolor=red, method=rk4, plotpoints=11]{0}{2}{1 1}{\Funct}
 \psplot[linewidth=4\pslinewidth, linecolor=green]{0}{2}{Euler dup x exp %%ch(x)
   exch x neg exp add 2 div}
 \psplotDiffEqn[linecolor=magenta, plotpoints=11]{0}{2}{1 0}{\Funct}
 \psplotDiffEqn[linecolor=blue, plotpoints=101]{0}{2}{1 0}{\Funct}
 \psplotDiffEqn[linecolor=red, method=rk4, plotpoints=11]{0}{2}{1 0}{\Funct}
 \psplot[linewidth=4\pslinewidth, linecolor=green]{0}{2}{Euler dup x exp
    exch x neg exp sub 2 div} %%sh(x)
 \psplotDiffEqn[linecolor=magenta, plotpoints=11]{0}{2}{0 1}{\Funct}
 \psplotDiffEqn[linecolor=blue, plotpoints=101]{0}{2}{0 1}{\Funct}
 \psplotDiffEqn[linecolor=red, method=rk4, plotpoints=11]{0}{2}{0 1}{\Funct}
 \rput*(1.3,.9){\psline[linecolor=magenta](-.75cm,0)}\rput*[l](1.3,.9){\small\textsc{Euler} order
    1 $h=1$}
 \rput*(1.3,.8){\psline[linecolor=blue](-.75cm,0)}\rput*[l](1.3,.8){\small\textsc{Euler} order 1 $
    h=0{,}1$}
 \rput*(1.3,.7){\psline[linecolor=red](-.75cm,0)}\rput*[l](1.3,.7){\small RK order 4 $h=1$}
 \rput*(1.3,.6){\psline[linecolor=green](-.75cm,0)}\rput*[l](1.3,.6){\small exact solution}
\end{pspicture}
```

$y'' = -y$




```
\def\Funct{exch neg}
\psset{xunit=1, yunit=4}
\def\quatrepi{12.5663706144}%%4pi=12.5663706144
\begin{pspicture}(0,-1.25)(\quatrepi,1.25)\psgrid[subgriddiv=1,griddots=10]
 \psplot[linewidth=4\pslinewidth,linecolor=green]{0}{\quatrepi}{x RadtoDeg cos}%%cos(x)
 \psplotDiffEqn[linecolor=blue, plotpoints=201]{0}{3.1415926}{1 0}{\Funct}
 \psplotDiffEqn[linecolor=red, method=rk4, plotpoints=31]{0}{\quatrepi}{1 0}{\Funct}
 \psplot[linewidth=4\pslinewidth,linecolor=green]{0}{\quatrepi}{x RadtoDeg sin} %%sin(x)
 \psplotDiffEqn[linecolor=blue,plotpoints=201]{0}{3.1415926}{0 1}{\Funct}
 \psplotDiffEqn[linecolor=red,method=rk4, plotpoints=31]{0}{\quatrepi}{0 1}{\Funct}
 \rput*(3.3,.9){\psline[linecolor=magenta](-.75cm,0)}\rput*[l](3.3,.9){\small Euler order
     1 $h=1$}
 \rput*(3.3,.8){\psline[linecolor=blue](-.75cm,0)}\rput*[l](3.3,.8){\small Euler order 1 $
     h=0{,}1$}
 \rput*(3.3,.7){\psline[linecolor=red](-.75cm,0)}\rput*[l](3.3,.7){\small RK order 4 $h
     =1$}
 \rput*(3.3,.6){\psline[linecolor=green](-.75cm,0)}\rput*[l](3.3,.6){\small exact solution
     }
\end{pspicture}
```

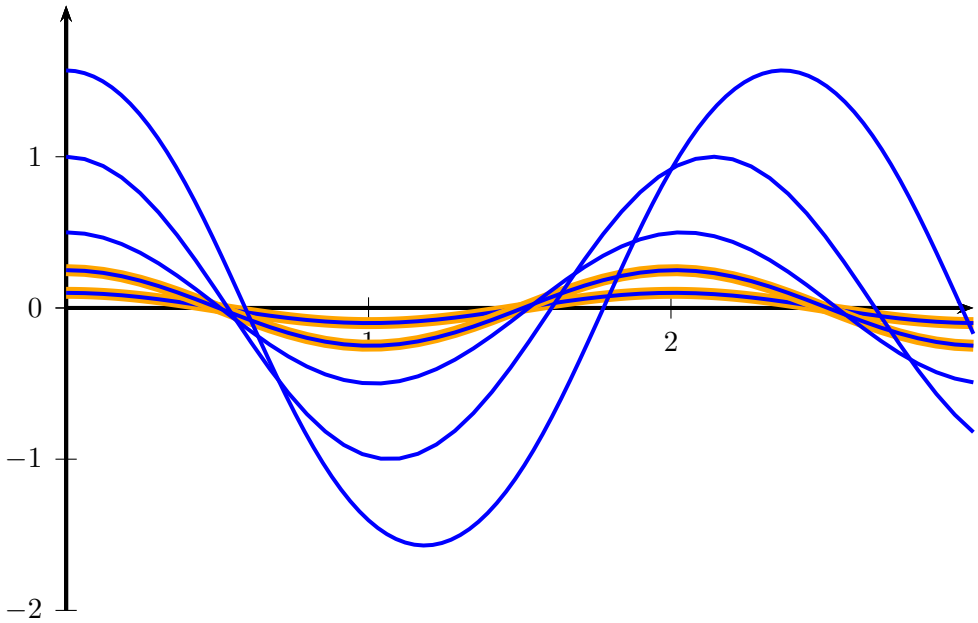
The mechanical pendulum: $y'' = -\frac{g}{l}\sin(y)$

For small oscillations $\sin(y) \simeq y$:

$$y(x) = y_0 \cos\left(\sqrt{\frac{g}{l}}x\right)$$

The function f is written in PostScript code:

```
exch RadtoDeg sin -9.8 mul %% y' -gsin(y)
```

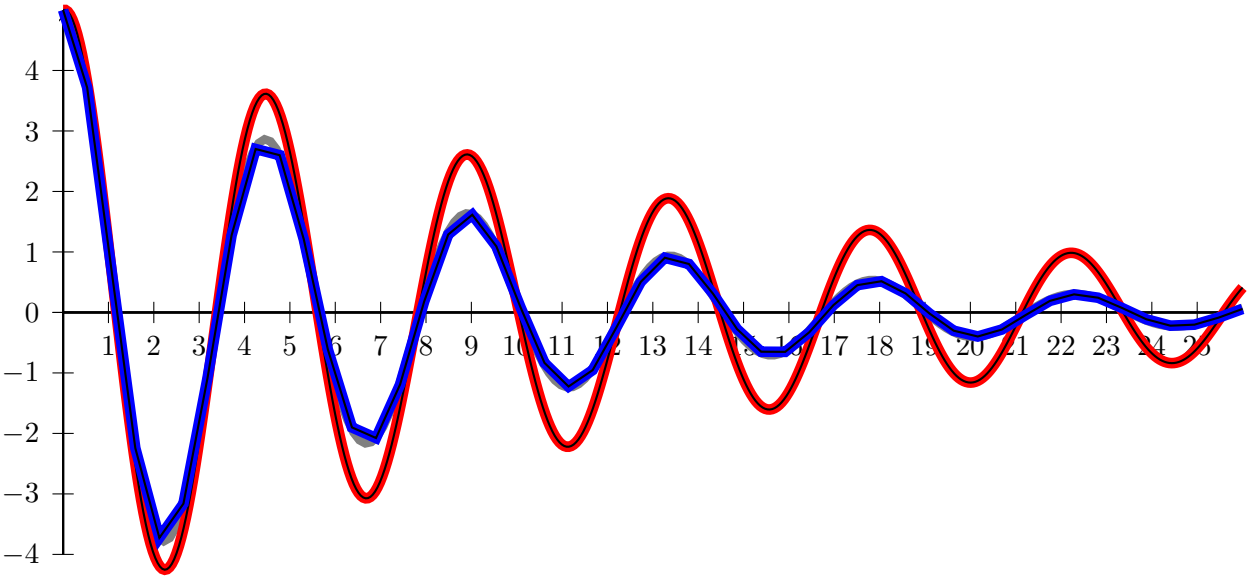



```
\def\Func{y[1]|-9.8*sin(y[0])}
\psset{yunit=2,xunit=4,algebraic,linewidth=1.5pt}
\begin{pspicture}(0,-2.25)(3,2.25)
  \psaxes{->}(0,0)(0,-2)(3,2)
  \psplot[linewidth=3\pslinewidth, linecolor=Orange]{0}{3}{.1*cos(sqrt(9.8)*x)}
  \psset{method=rk4,plotpoints=50,linecolor=blue}
  \psplotDiffEqn{0}{3}{.1 0}{\Func}
  \psplot[linewidth=3\pslinewidth,linecolor=Orange]{0}{3}{.25*cos(sqrt(9.8)*x)}
  \psplotDiffEqn{0}{3}{.25 0}{\Func}
  \psplotDiffEqn{0}{3}{.5 0}{\Func}
  \psplotDiffEqn{0}{3}{1 0}{\Func}
  \psplotDiffEqn[plotpoints=100]{0}{3}{Pi 2 div 0}{\Func}
\end{pspicture}
```

$y'' = -\frac{y'}{4} - 2y$

For $y_0 = 5$ and $y'_0 = 0$ the solution is:

$$5e^{-\frac{x}{8}}\left(\cos(\omega x) + \frac{\sin(\omega x)}{8\omega}\right) \text{ avec } \omega = \frac{\sqrt{127}}{8}$$

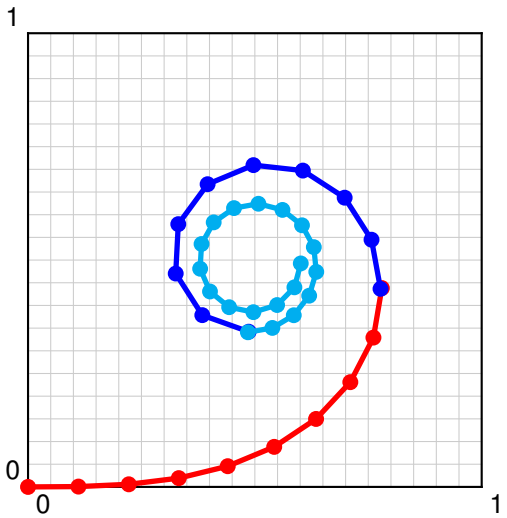



```
\psset{xunit=.6,yunit=0.8,plotpoints=500}
\begin{pspicture}(0,-4.25)(26,5.25)
  \psaxes{->}(0,0)(0,-4)(26,5)
  \psplot[plotpoints=200,linewidth=4\pslinewidth,linecolor=gray]{0}{26}{%
    Euler x -8 div exp x 127 sqrt 8 div mul RadtoDeg dup cos 5 mul exch sin 127 sqrt div 5
         mul add mul}
  \psplotDiffEqn[linecolor=red,linewidth=5\pslinewidth]{0}{26}{5 0}
    {dup 3 1 roll -4 div exch 2 mul sub}
  \psplotDiffEqn[linecolor=black,algebraic]{0}{26}{5 0} {y[1]|-y[1]/4-2*y[0]}
  \psset{method=rk4, plotpoints=50}
  \psplotDiffEqn[linecolor=blue,linewidth=5\pslinewidth]{0}{26}{5 0}{%
     dup 3 1 roll -4 div exch 2 mul sub}
  \psplotDiffEqn[linecolor=black,algebraic]{0}{26}{5 0}{y[1]|-y[1]/4-2*y[0]}
\end{pspicture}
```

26.3. Save final state of a equation

With the macros \BeginSaveFinalState and \EndSaveFinalState the end values of a differential equation can be saved and then used with the optional argument GetFinalState as starting values for another equation.

```
\psset{unit=10cm,linewidth=2pt}
\begin{pspicture}(1,1)\psgrid[subgridcolor=black!20,subgriddiv=20]
\BeginSaveFinalState
 \psplotDiffEqn[
  whichabs=0,whichord=1,linecolor=red,method=rk4,
  plotpoints=10,showpoints=true]{0}{1}{0 0}{
  pop pop
  x dup mul 2 div 180 mul cos %% dx/dt
  x dup mul 2 div 180 mul sin %% dy/dt
 }
 \psplotDiffEqn[GetFinalState,
  whichabs=0,whichord=1,linecolor=blue,method=rk4,%FinalState,
  plotpoints=10,showpoints=true]{1}{2}{0 0}{
  pop pop
  x dup mul 2 div 180 mul cos %% dx/dt
  x dup mul 2 div 180 mul sin %% dy/dt
 }
 \psplotDiffEqn[GetFinalState,
  whichabs=0,whichord=1,linecolor=cyan,method=rk4,%FinalState,
  plotpoints=19,showpoints=true]{2}{3}{0 0 }{
  pop pop
  x dup mul 2 div 180 mul cos %% dx/dt
  x dup mul 2 div 180 mul sin %% dy/dt
 }
\EndSaveFinalState
\end{pspicture}
```



27. \psMatrixPlot

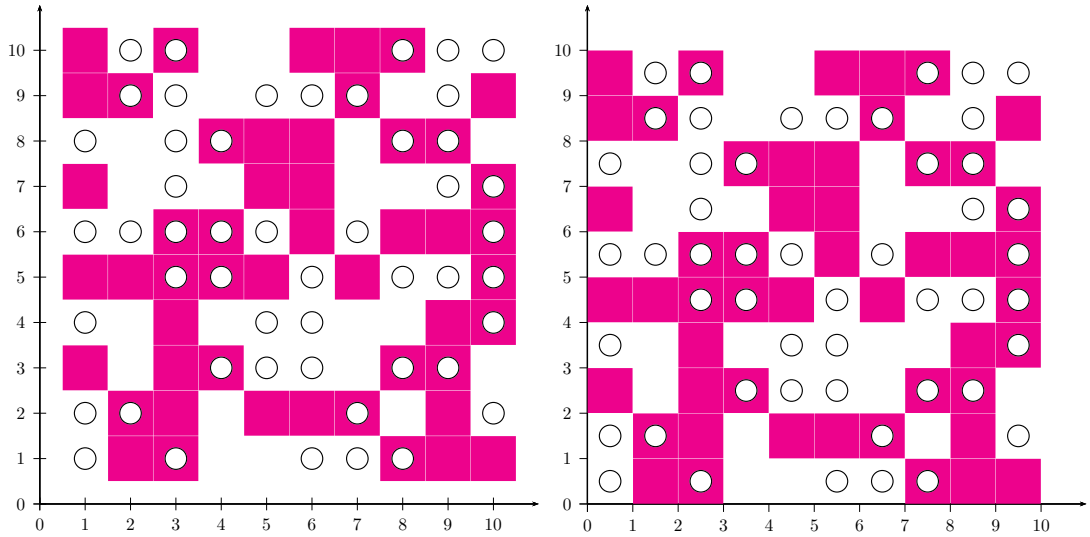
This macro allows you to visualize a matrix. The datafile must be defined as a PostScript matrix named `dotmatrix`:

```
/dotmatrix [ % <------------ important line
0 1 1 0 0 0 0 1 1 1
0 1 1 0 1 1 1 0 1 0
1 0 1 1 0 0 0 1 1 0
0 0 1 0 0 0 0 0 1 1
1 1 1 1 1 0 1 0 0 1
0 0 1 1 0 1 0 1 1 1
1 0 0 0 1 1 0 0 0 1
0 0 0 1 1 1 0 1 1 0
1 1 0 0 0 0 1 0 0 1
1 0 1 0 0 1 1 1 0 0
] def     % <------------ important line
```

Only the value 0 is important, in which case nothing happens, and for all other cases a dot is printed. The syntax of the macro is:

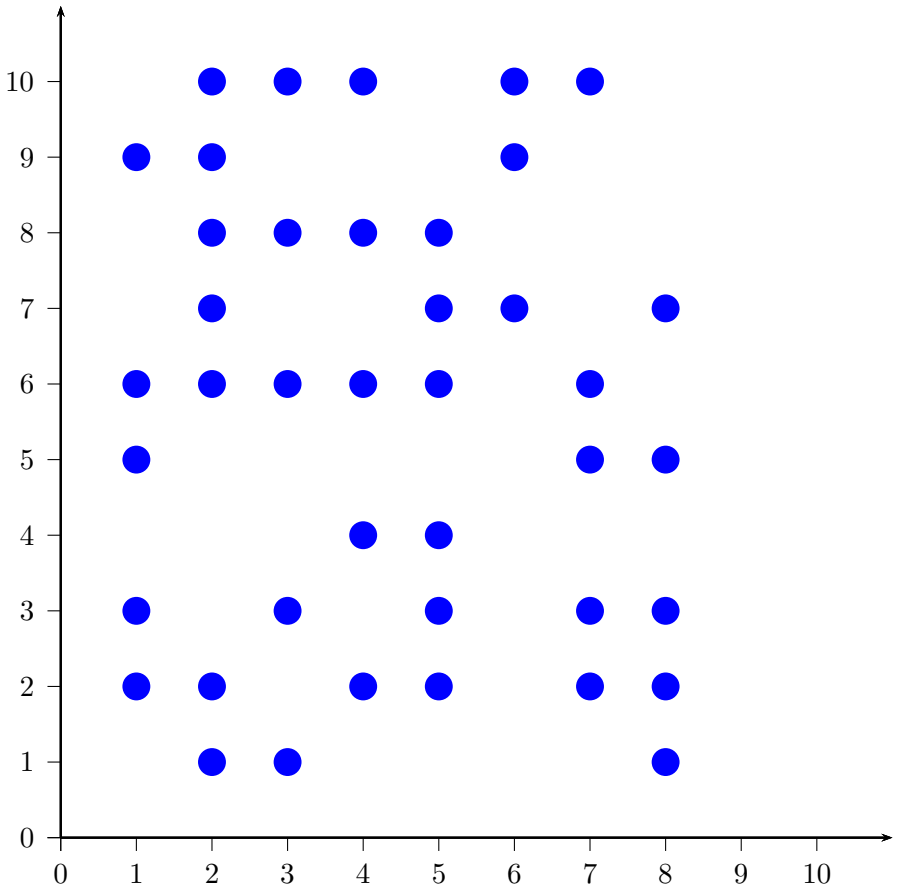
`\psMatrixPlot [Options] {rows}{columns}{data file}`

The matrix is scanned line by line from the the first one to the last. In general it appears as a bottom-to-top version of the above listed matrix, the first row $0\,1\,1\,0\,0\,0\,0\,1\,1\,1$ is the first plotted line ($y = 1$). With the option `ChangeOrder=true` it looks exactly like the above view.




```
\psscalebox{0.6}{%
\begin{pspicture}(-0.5,-0.75)(11,11)
  \psaxes[ticksize=-5pt 0]{->}(11,11)
  \psMatrixPlot[dotsize=1.1cm,dotstyle=square*,linecolor=magenta]%
    {10}{10}{data/matrix.data}
  \psMatrixPlot[dotsize=.5cm,dotstyle=o,ChangeOrder]{10}{10}{data/matrix.data}
\end{pspicture}}\quad
\psscalebox{0.6}{%
\begin{pspicture}(-0.5,-0.75)(11,11)
  \psaxes{->}(11,11)
```

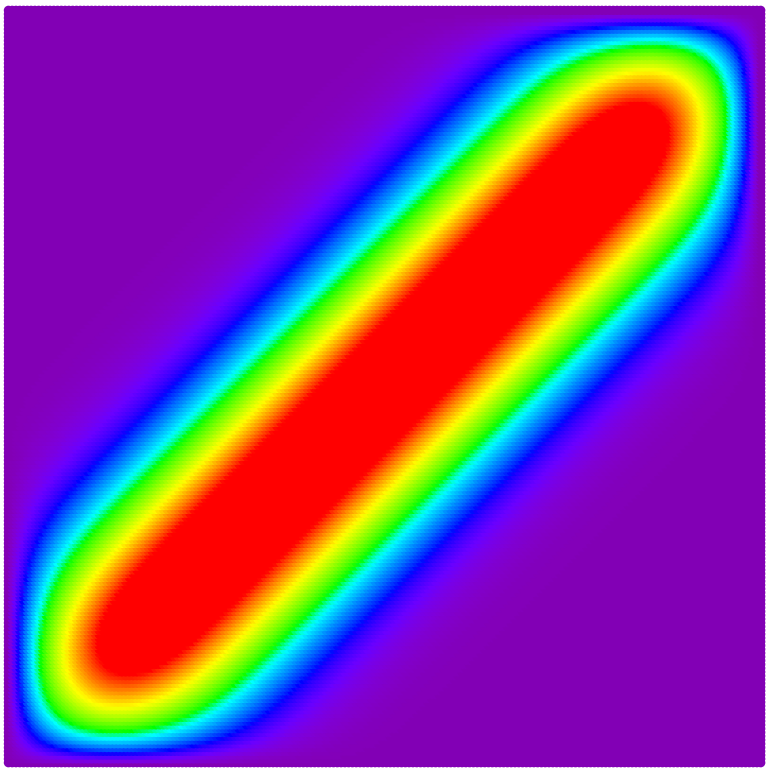
```
  \psMatrixPlot[dotsize=1.1cm,dotstyle=square*,linecolor=magenta,XYoffset=-0.5]%
    {10}{10}{data/matrix.data}
  \psMatrixPlot[dotsize=.5cm,dotstyle=o,ChangeOrder,XYoffset=-0.5]{10}{10}{data/matrix.
      data}
\end{pspicture}}
```



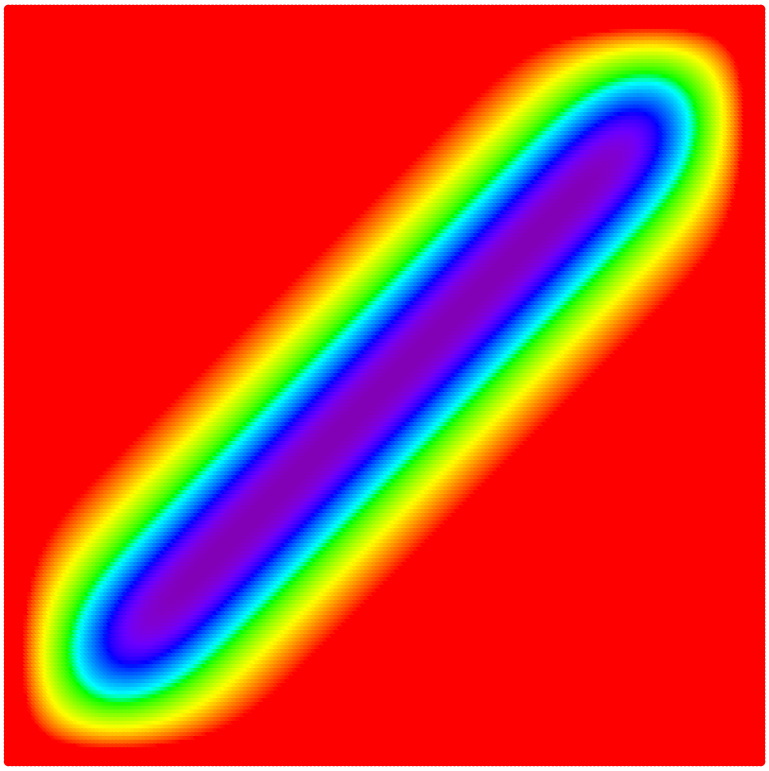

```
\begin{pspicture}(-0.5,-0.75)(11,11)
  \psaxes[ticksize=-5pt 0]{->}(11,11)
  \psMatrixPlot[dotscale=3,dotstyle=*,linecolor=blue]{10}{8}{data/matrix.data}
\end{pspicture}
```

With the colorType=1 the data is printed as continous color in the range of the wavelength. The smallest value of the data array is set to red and the biggest value is set to violett. All other values are substituted by the corresponding color of the wavlength. colorType=2 ist the same, but vice versa with the color, from violet to red. colorType=3 is the grayscale image and colorType=4 the same invers.

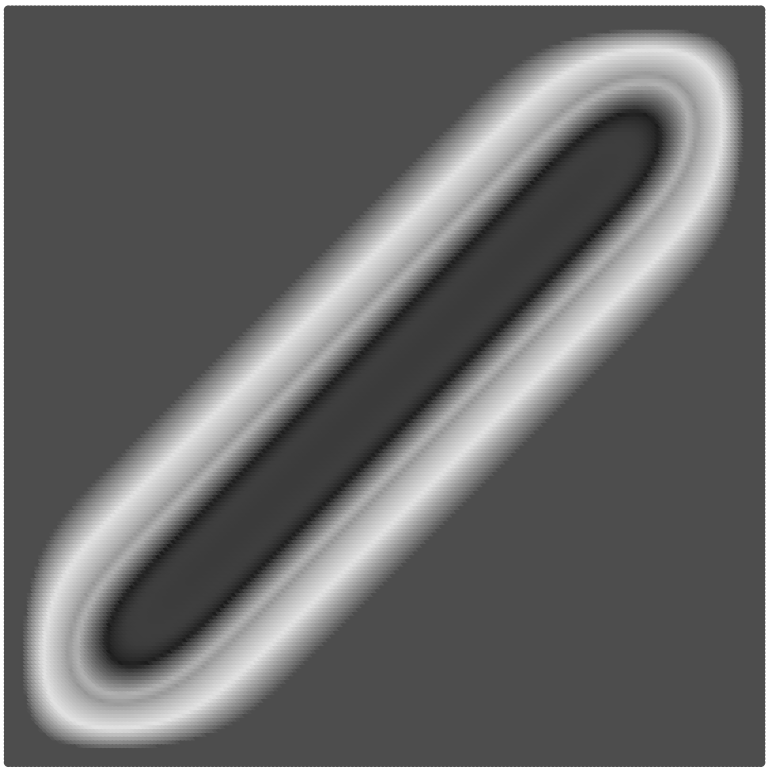
The following examples use a 200×200 matrix data, which is saved as /dotmatrix [...] in the file pstricks-add-doc.dat.



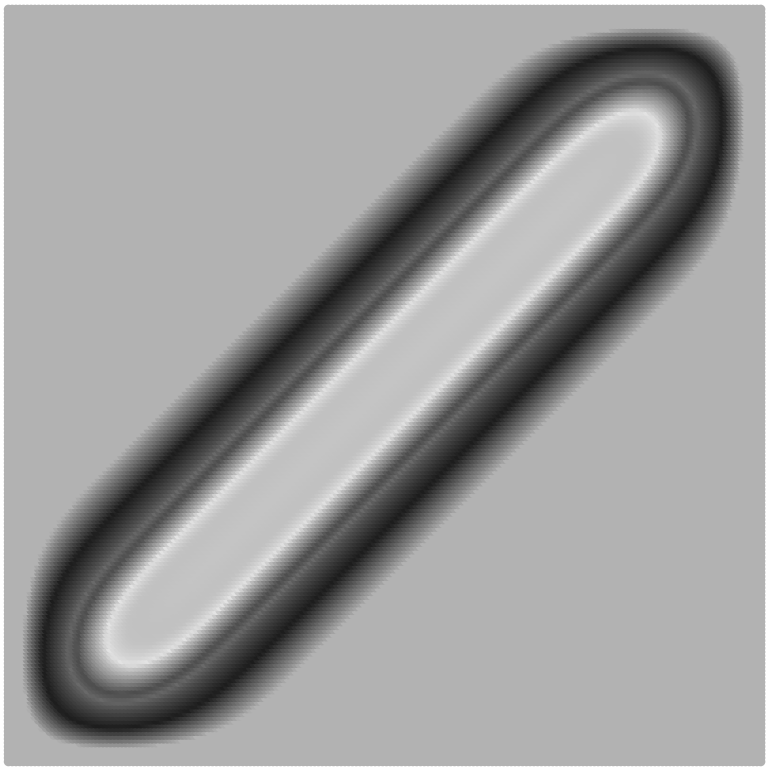
```
\begin{pspicture}(10,10)
  \psMatrixPlot[colorType=1,xStep=0.05,yStep=0.05]{200}{200}{data/dotmatrix.data}
\end{pspicture}
```



```
\begin{pspicture}(10,10)
  \psMatrixPlot[colorType=2,xStep=0.05,yStep=0.05]{200}{200}{data/dotmatrix.data}
\end{pspicture}
```

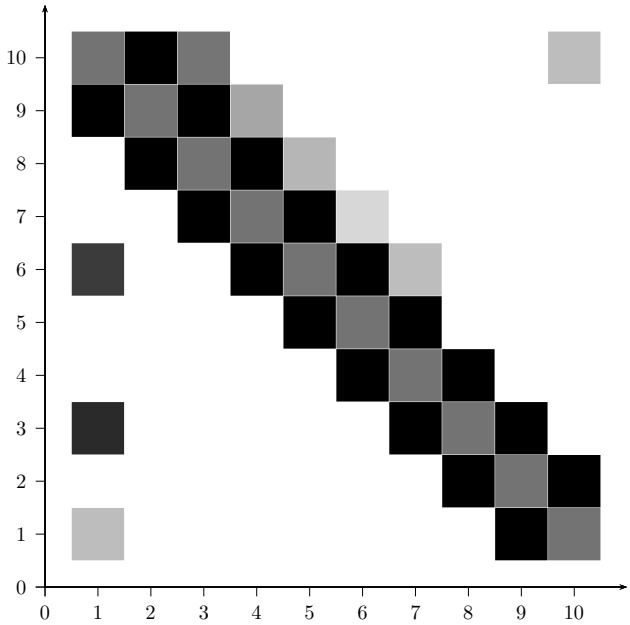


```
\begin{pspicture}(10,10)
  \psMatrixPlot[colorType=3,xStep=0.05,yStep=0.05]{200}{200}{data/dotmatrix.data}
\end{pspicture}
```



```
\begin{pspicture}(10,10)
  \psMatrixPlot[colorType=4,xStep=0.05,yStep=0.05]{200}{200}{data/dotmatrix.data}
\end{pspicture}
```

With the colorType=5 the color setting can be user defined by the optional argument colorTypeDef. On the stack is the current value which can be used for the setting but must be left on the stack when everything is finished. The following example prints the 0 as color white, the value 1 as black and all other values depending to the corresponding gray value.




```
\begin{filecontents}{data/matrix1.data}
/dotmatrix [ % <------------ important line
3 0 0 0 0 0 0 0 1 2
0 0 0 0 0 0 0 1 2 1
8 0 0 0 0 0 1 2 1 0
0 0 0 0 0 1 2 1 0 0
0 0 0 0 1 2 1 0 0 0
9 0 0 1 2 1 3 0 0 0
0 0 1 2 1 4 0 0 0 0
0 1 2 1 5 0 0 0 0 0
1 2 1 6 0 0 0 0 0 0
2 1 7 0 0 0 0 0 0 3
] def % <------------ important line
\end{filecontents}
\psscalebox{0.7}{%
\begin{pspicture}(-0.5,-0.75)(11,11)
\psaxes[ticksize=-5pt 0]{->}(11,11)
\psMatrixPlot[
  colorType=5,
  colorTypeDef={
    dup /value exch def % save value and leave one on the stack
    value Min sub dMaxMin div neg 1 add 300 mul 400 add \pswavelengthToGRAY
    value 0 eq \pslbrace 1 \psrbrace if %
    value 1 eq \pslbrace 0 \psrbrace if
    setgray
  },
  dotsize=1.1cm,xStep=1,yStep=1,dotstyle=square*]{10}{10}{data/matrix1.data}
\end{pspicture}}
```

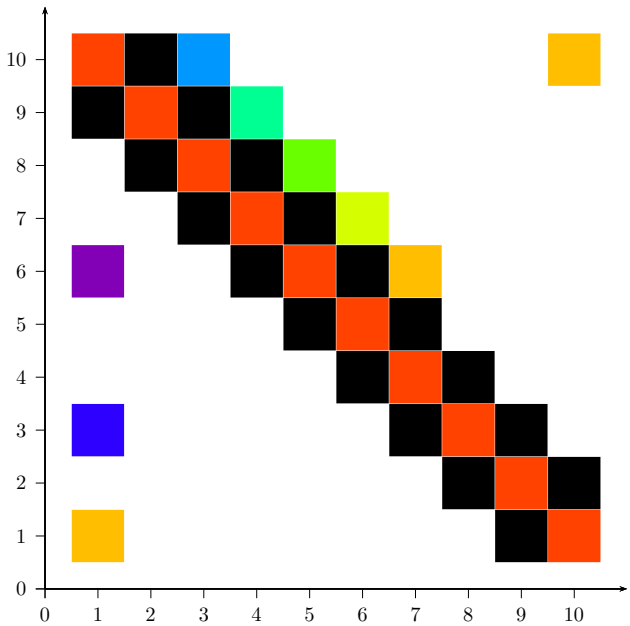
if statements in the color definition must be enclosed with \pslbrace and \psrbrace when they are parentheses used in PostScript. In the above example the color definition should be modified when the matrix is a real big one, in such a case a nested ifelse makes more sense:

```
colorTypeDef={
  dup /value exch def
  value 0 eq
    \pslbrace 1 setgray \psrbrace
    \pslbrace value 1 eq
      \pslbrace 0 setgray \psrbrace
      \pslbrace Min sub dMaxMin div neg 1 add 300 mul 400 add
        \pswavelengthToGRAY setgray \psrbrace ifelse
    \psrbrace ifelse
},
```

Replace the \pslbrace and \psrbrace with { and } if it maybe confusing to read:

```
  dup /value exch def
  value 0 eq
    { 1 setgray }
    { value 1 eq
      { 0 setgray }
      { Min sub dMaxMin div neg 1 add 300 mul 400 add
        \pswavelengthToGRAY setgray } ifelse
    } ifelse
```

Another possibility is to define the color procedure onside the data file, where it *must* be named colorTypeDef. If such a definition exists, the one from the optional argument colorTypeDef will be ignored. There can be no TEX-specific code inside this definition because it is read on PostScript level, the reason why \pswavelengthToGRAY cannot be used.



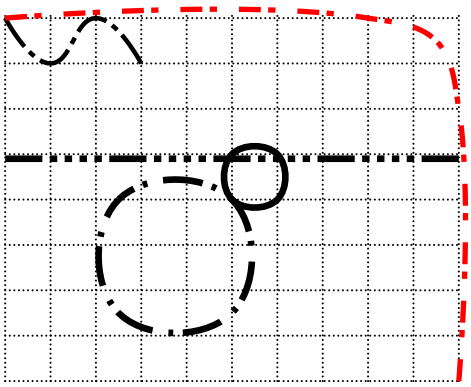

```
\begin{filecontents}{data/matrix2.data}
/colorTypeDef {
  dup /value exch def
  value 0 eq
```

```
   { 1 setgray }
   { value 1 eq
     { 0 setgray }
     { Min sub dMaxMin div neg 1 add 300 mul 400 add
%      \pswavelengthToRGB not possible
       tx@addDict begin wavelengthToRGB Red Green Blue end
      setrgbcolor
     } ifelse
   } ifelse
} def
/dotmatrix [ % <------------ important line
3 0 0 0 0 0 0 0 1 2
0 0 0 0 0 0 0 1 2 1
8 0 0 0 0 0 1 2 1 0
0 0 0 0 0 1 2 1 0 0
0 0 0 0 1 2 1 0 0 0
9 0 0 1 2 1 3 0 0 0
0 0 1 2 1 4 0 0 0 0
0 1 2 1 5 0 0 0 0 0
1 2 1 6 0 0 0 0 0 0
2 1 7 0 0 0 0 0 0 3
] def % <------------ important line
\end{filecontents}
\psscalebox{0.7}{%
\begin{pspicture}(-0.5,-0.75)(11,11)
\psaxes[ticksize=-5pt 0]{->}(11,11)
\psMatrixPlot[colorType=5,dotsize=1.1cm,xStep=1,yStep=1,
  dotstyle=square*]{10}{10}{data/matrix1.data}
\end{pspicture}}
```

28. Dashed Lines

Tobias Nähring has implemented an enhanced feature for dashed lines. The number of arguments is no longer limited.

```
dash=value1 unit value2 unit ...
```



```
\psset{linewidth=2.5pt,unit=0.6}
\begin{pspicture}(-5,-4)(5,4)
 \psgrid[subgriddiv=0,griddots=10,gridlabels=0pt]
  \psset{linestyle=dashed}
  \pscurve[dash=5mm 1mm 1mm 1mm,linewidth=0.1](-5,4)
      (-4,3)(-3,4)(-2,3)
  \psline[dash=5mm 1mm 1mm 1mm 1mm 1mm 1mm 1mm 1mm 1
      mm](-5,0.9)(5,0.9)
  \psccurve[linestyle=solid](0,0)(1,0)(1,1)(0,1)
  \psccurve[linestyle=dashed,dash=5mm 2mm 0.1 0.2,
      linetype=0](0,0)(-2.5,0)(-2.5,-2.5)(0,-2.5)
  \pscurve[dash=3mm 3mm 1mm 1mm,linecolor=red,
      linewidth=2pt](5,-4)(5,2)(4.5,3.5)(3,4)(-5,4)
\end{pspicture}
```

29. Ticks and other marks along a curve

29.1. Quick overview

The macros described below allow you to place tick and other marks along an arbitrary parametric curve with placement rules similar to those used by `\psaxes` in the `pst-plot` package. You have to define a metric function along the curve to govern tick placement. That function can be a specified function of `x,y` which should increase along the curve, or it can be an function whose increment is a specified positive function of `x, y, dx, dy, ds` where the last term is the arc-length element that you could specify alternately as `dx dup mul dy dup mul add sqrt`.

In addition, a new command `\Put` is proposed, expanding as appropriate to `\rput` or `\uput`. Its syntax is

`\Put`*`*` `[Options]` `{`*<ref>*`}` `(`*<position>*`)`{*<stuff>*}

where the optional * blanks the background, the optional `[Options]` may be used to specify a rotation using any form acceptable to `\SpecialCoor` (eg, `rot=45` or `rot={(1,1)}` or `rot=(P)`, and *ref* takes one of two forms: (a) a refpt such as `Bl`, in which case `\rput` is called; (b) a polar form of offset (eg, `7pt;30`, or `;(P)` — in the latter case, `\pslabelsep` is substituted for the missing radius), in which case a modified form of `\uput` is called. The idea of `\Put` is to allow `position`, `ref` and `rot` to be specified in any of the forms acceptable to `\SpecialCoor` and to do so with the same output no matter what form is used. The cost of this consistency is that `\Put` can lead to results that differ from `\uput` in some special cases.

29.2. Details

Suppose you have drawn a parametric curve using `\psparametricplot`, and you wish to indicate some points on the curve using tick-marks like those on the axes. This is a two-step process, the first of which serves to define at the PostScript level a number of data arrays containing information about the curve. Those arrays are used in the second step to compute tick positions and draw the ticks. The first step is to run the macro `\pscurvepoints`. For example,

```
\pscurvepoints[plotpoints=20]{0}{6}{t t t mul 12 div}{Pt}%
```

makes a virtual (ie, data only—nothing is rendered) polyline with 20 vertices approximating the curve $x(t) = t, y(t) = t^2/12$, $0 \le t \le 6$. The last argument `Pt` is the root name given to the data arrays. PostScript arrays will be created with the following names: `Pt.X, Pt.Y` for the coordinates of the vertices, `PtDelta.X, PtDelta.Y` for the increments between the vertices (using, eg, `PtDelta.X[2]=Pt.X[2]-Pt.X[1]`) and `PtNormal.X, PtNormal.Y` for a vector normal to `PtDelta.X, PtDelta.Y` in the visual, not mathematical, sense. (Both senses are the same if the scales on the axes are identical.) The `Normal` is always constructed so as to point “upward” (ie, to your left) as you traverse the curve in the positive direction. The PostScript variable `unitratio` provides the ratio of the unit on the y axis to that on x axis, and `unitratiosq` is its square. All of these PostScript objects are stored in the main `pstricks` dictionary `tx@Dict` which should be automatically made available when using many `pstricks` macros. If `gs` returns you an error message like

```
Error: /undefined in Pt.X
```

then you may need to enclose the offending PostScript code within a block of the form

```
tx@Dict begin ... end
```

so that the dictionary is made available.

With this preparation, the main tick-making macro may be run. For example,

```
\pspolylineticks{Pt}{ dx dy add 3 div }{1}{2}%
```

looks for data arrays made using `\pscurvepoints` with the root name `Pt`. The next argument, `dx dy add 3 div`, specifies the (PostScript) function of increments that should be used to construct the metric. If the keyword `metricInitValue` is defined, eg, with `\psset{`*`metricInitValue=2.5`*`}`, it is used as the initial value of the metric, otherwise it is defined to be 0. In the previous example, the increment function is always positive, and care should be taken to guarantee this is so or the results will not be meaningful. (If we wanted to use arc-length, the function would have been `ds`, assuming equal scales on the axes.) The last two arguments determine the index of the first tick and the number of ticks. Tick numbering begins with index 0, so the example says to drop the first tick and draw the next 2 ticks. In this example, where all keywords take their default values, ticks are potentially located at values on the curve where the metric takes a positive integer value. In the arc-length example, the tick with index 0 is at the beginning of the curve, and subsequent ticks are at unit distance, measured along the curve. At each index where a tick is drawn, a `\pnode` is created: In this example, you create nodes `PtTick1, PtTick2` on the curve where the ticks are located. This is handy for placing labels using, eg, `\Put`. In addition, PostScript data arrays (in this example, `PtTickN.X, PtTickN.Y` of the normals at these nodes are stored in the dictionary `TDict`. More importantly, the tangent and normal vectors at `PtTick0` etc are constructed as nodes with names `PtTangent0, PtNormal0` etc. See the last example below for typical usage.

The shape of the ticks is governed by the keywords `ticksize` (default value `-4pt 4pt`) and `tickwidth` (default value `.5\linewidth`.) With the default settings, ticks are drawn perpendicular the the curve extending `4pt` to each side. The line

```
\pspolylineticks[ticksize=-6pt 6pt]{Pt}{ dx dy add 3 div }{1}{2}%
```

would draw longer ticks than the default.

Placement of the ticks is governed by the keywords `Ds` and `Os`, whose meaning for the curve is similar to (but not the same as) the meanings of `Dx` and `Ox` with respect to the x axis. That is, if `Ds=2` and `Os=0`, ticks will be drawn where the metric takes values 0, 2, 4 and so on. More generally, ticks are placed where the metric takes value `Os, Os+Ds, Os+2*Ds,...`, as long as those positions are on the curve. If `Os` has an empty value as a result, say, of `\psset{Os=}`, then `Os` is set internally to the initial metric value. If `Ds` has an empty value, it is set internally to the final metric value less the initial metric value, divided by 10.

To draw major and minor ticks requires two passes—one to draw the minor ticks and then one to draw the major ticks.

Note that a ticks may be placed at arbitrary metric values on the curve by running the macro once for each point, like:

```
\pspolylineticks[ticksize=-6pt 6pt,Os=1.3]{Pt}{ dx dy add 3 div }{0}{1}%
\pspolylineticks[ticksize=-6pt 6pt,Os=2.4]{Pt}{ dx dy add 3 div }{0}{1}%
```

You may also dispense entirely with the tick and use the macro to generate a node sequence that can be used to place other graphic objects. For example:

```
\pspolylineticks[ticksize=0pt 0pt]{Pt}{ dx dy add 3 div }{0}{3}%
%This defines nodes PtTick0..PtTick2
\multido{\iA=0+1}{3}{\psdot(PtTick\iA)}
```

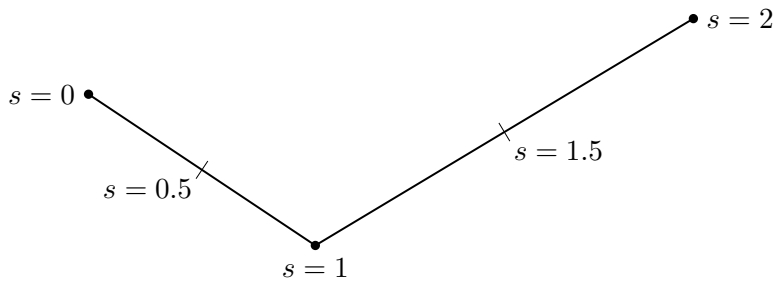
There is another way to define a metric function without using increments. If the keywork `metricFunction` is set to `true`, then the function you present as an argument to `\pspolylineticks` must be a function of x and y only, and must be designed to increase along the curve. It is useful only in those cases where, in essence, the increment function can be explicitly integrated. For example, in the elliptical motion of planets and comets around the sun, it is not hard to integrate the area function explicitly, and this provides a convenient metric, being proportional to time elapsed.

There is some useful information left in the log by these macros. They report the starting and ending values of the metric function, the the range of indices for the Tick related arrays.

29.3. Examples

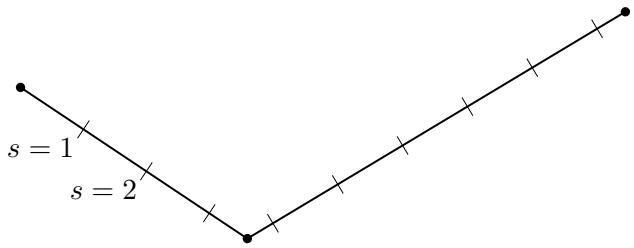
The examples in this section make use of very recent (as of May, 2010) versions of `pstricks` and related packages.

The first couple of examples are constructed entirely by hand, and have no interest other than to illustrate what is going on under the surface in the simplest case.



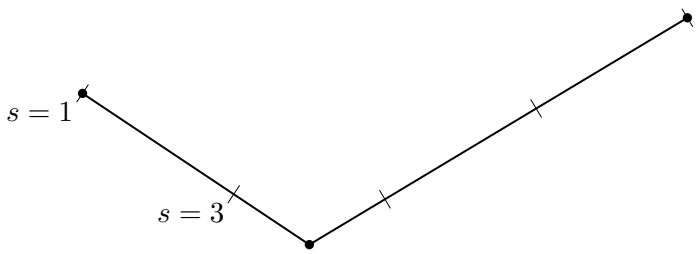

```
\begin{pspicture}(-1,-1)(10,4)
\psline[showpoints=true](1,2)(4,0)(9,3)%
\uput[180](1,2){$s=0$}%
\uput[-90](4,0){$s=1$}%
\uput[0](9,3){$s=2$}%
\makeatletter% need to use macro names containing @
\pstVerb{tx@Dict begin %the pstricks dictionary
% declare arrays of length 3 (indices 0,1,2) to hold points, differences and normals
/unitratiosq 1 def % yunit=xunit
/P.X [ 1 4 9 ] def %array of x coords
/P.Y [ 2 0 3 ] def %array of y coords
/PDelta.X [ 0 3 5 ] def % 3=4-1, 5=9-4, 0 never used
/PDelta.Y [ 0 -2 3 ] def % -2=0-2, 3=3-0, 0 never used
% normal to (3,-2) is (2,3), normal to (5,3) is (-3,5)
/PNormal.X [ 2 2 -3 ] def % index 0 =index 1
/PNormal.Y [ 3 3 5 ] def % index 0 = index 1
end }
\def\Ppointcount{2}
\makeatother % make ticks using metric function with values 0,1,2
\pspolylineticks[Os=.5,Ds=1]{P}{1}{0}{2}
\uput[-135](PTick0){$s=0.5$}% % ticks at s=0.5,1.5 (increment function =1)
\uput[-45](PTick1){$s=1.5$}%
\end{pspicture}
```

Now the same data, but with arc-length as metric. We change the last few lines:



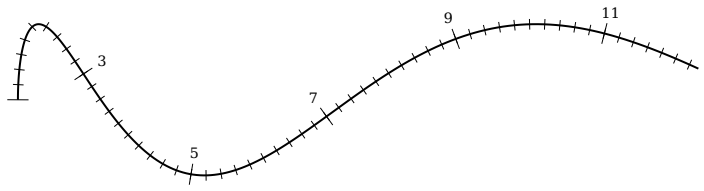

```
\begin{pspicture}(-1,-1)(10,4)
\psline[showpoints=true](1,2)(4,0)(9,3)%
%\uput[180](1,2){$s=0$}%
%\uput[-90](4,0){$s=1$}%
%\uput[0](9,3){$s=2$}%
\makeatletter% need to use macro names containing @
\pstVerb{tx@Dict begin %the pstricks dictionary
% declare arrays of length 3 (indices 0,1,2) to hold points,
% differences and normals
/unitratiosq 1 def % yunit=xunit
/P.X [ 1 4 9 ] def %array of x coords
/P.Y [ 2 0 3 ] def %array of y coords
/PDelta.X [ 0 3 5 ] def % 3=4-1, 5=9-4, 0 never used
/PDelta.Y [ 0 -2 3 ] def % -2=0-2, 3=3-0, 0 never used
% normal to (3,-2) is (2,3), normal to (5,3) is (-3,5)
/PNormal.X [ 2 2 -3 ] def % index 0 =index 1
/PNormal.Y [ 3 3 5 ] def % index 0 = index 1
end }
\def\Ppointcount{2}
\makeatother
% make ticks using metric function arc-length
\pspolylineticks[Os=1,Ds=1]{P}{ ds }{0}{9}
% ticks at s=1,2... (increment function = distance)
\uput[-135](PTick0){$s=1$}%
\uput[-135](PTick1){$s=2$}%
\end{pspicture}
```

Once again the same data, but with metric equal to the x coordinate. Change the last few lines to:




```
\begin{pspicture}(-1,-1)(10,4)
\psline[showpoints=true](1,2)(4,0)(9,3)%
%\uput[180](1,2){$s=0$}%
%\uput[-90](4,0){$s=1$}%
%\uput[0](9,3){$s=2$}%
\makeatletter% need to use macro names containing @
\pstVerb{tx@Dict begin %the pstricks dictionary
% declare arrays of length 3 (indices 0,1,2) to hold points,
% differences and normals
/unitratiosq 1 def % yunit=xunit
/P.X [ 1 4 9 ] def %array of x coords
/P.Y [ 2 0 3 ] def %array of y coords
/PDelta.X [ 0 3 5 ] def % 3=4-1, 5=9-4, 0 never used
/PDelta.Y [ 0 -2 3 ] def % -2=0-2, 3=3-0, 0 never used
% normal to (3,-2) is (2,3), normal to (5,3) is (-3,5)
/PNormal.X [ 2 2 -3 ] def % index 0 =index 1
/PNormal.Y [ 3 3 5 ] def % index 0 = index 1
end }
\def\Ppointcount{2}
\makeatother
% make ticks using metric function arc-length
\pspolylineticks[metricFunction,Os=1,Ds=2]{P}{ x }{0}{5}
% ticks at x=1,3,... , start at tick index 0, draw 5 ticks
% the tick at s=1 has index 0
% ticks at s=1,2... (increment function = distance)
\uput[-135](PTick0){$s=1$}%
\uput[-135](PTick1){$s=3$}%
\end{pspicture}
```

The next example is a smooth path where subticks are drawn first, followed by major ticks. The metric is arc-length with initial value $s = 1$.

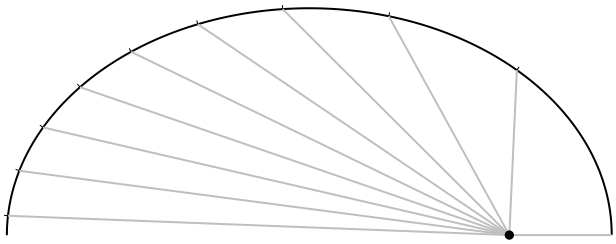



```
\begin{pspicture}(-1,-1)(10,4)
%\parametricplot[algebraic]{0}{9}{(t^2)/9 | 4*Ex(-t)*(1+t+(t^{2})/2+(t^{3})/6)}
\psparametricplot[algebraic]{0}{9}{t^2/9 | sin(t)+1}%
\pscurvepoints{0}{9}{(t^2)/9 | sin(t)+1}{P}%
% make ticks using arc-length metric
\pspolylineticks[metricInitValue=1,ticksize=-2pt 2pt,Os=1,Ds=.2]{P}{ ds }{1}{56}%
\pspolylineticks[metricInitValue=1,Os=1,Ds=2]{P}{ ds }{0}{6}%
\multido{\iA=1+1,\iB=3+2}{5}{\Put{6pt;(PNormal\iA)}(PTick\iA){\tiny \iB}}
%\nodexn{(PTick\iA)+(10pt;{(PNormal\iA)})}{Q}\rput(Q){\tiny \iB}}%
%\multido{\iA=1+1,\iB=3+2}{5}{\uput{6pt}[{(PNormal\iA)}](PTick\iA){\iB}}%
% ticks at x=1,3,... , start at tick index 0, draw 5 ticks
% the tick at s=1 has index 0
% ticks at s=1,2... (increment function = distance)
\end{pspicture}
```

Suppose for the next example that we have an ellipse $x^2/a^2 + y^2/b^2 = 1$ ($a > b$) with eccentricity $\epsilon = (1 - b^2/a^2)^{1/2}$. With planetary motion in mind, a natural metric for the ellipse is the area swept out by the radial line from the focus $(\epsilon a, 0)$ starting from $(a, 0)$ around to an arbitrary location (x, y), where $y > 0$, as this quantity is proportional to the time elapsed since perihelion. A routine calculation gives the following formula:

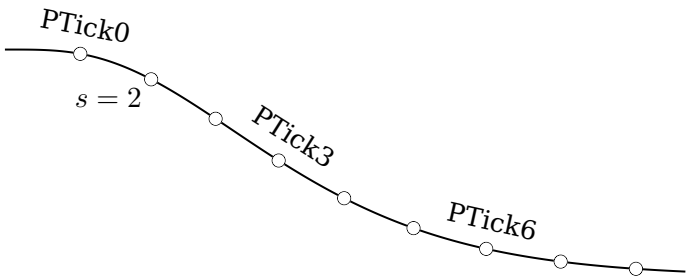
$$A = \frac{ab}{2} \arccos\left(\frac{x}{a}\right) - \frac{\epsilon a y}{2}.$$

Remembering that PostScript's `acos` gives its result in degrees, not radians, we have the following, drawn for the case $a = 4$, $b = 3$.



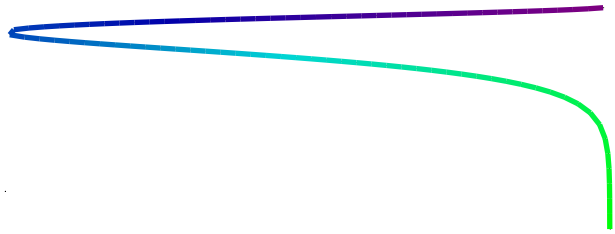
```
\begin{pspicture}(-4.5,-.5)(4.5,3.5)
\pstVerb{ /smajor 4 def /sminor 3 def % define semimajor, semiminor
/ecc 1 sminor smajor div dup mul sub sqrt def % compute eccentricity
/ab smajor sminor mul 2 div def %first coeff
/ea smajor ecc mul 2 div def }% second coeff
\psparametricplot[algebraic]{0}{3.142}{smajor*cos(t) | sminor*sin(t)}%
\pscurvepoints{0}{3.142}{smajor*cos(t) | sminor*sin(t)}{P}%
\pspolylineticks[metricFunction,Ds=2,ticksize=-1.5pt 0]{P}{ ab x smajor div acos %
180 div PI mul mul ea y mul sub }{1}{9}%
\pnode(! ecc smajor mul 0){S}% focus
\psline[linecolor=lightgray](S)(!smajor 0)%
\multido{\i=1+1}{9}{\psline[linecolor=lightgray](S)(PTick\i)}
\psdot(S)
\end{pspicture}
```

The next examples works without visible ticks, using the macros to construct nodes at which other objects will be placed.



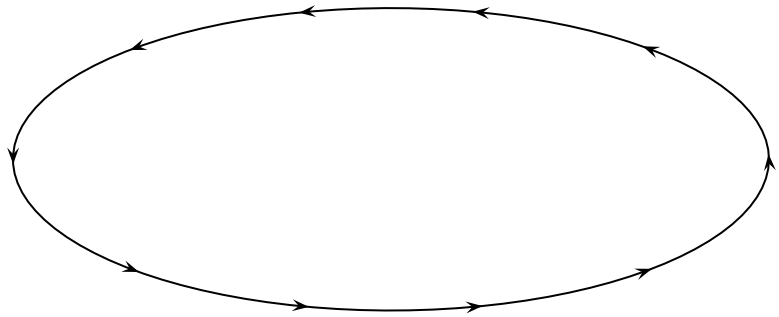

```
\begin{pspicture}(-1,-1)(10,4)
\psparametricplot[algebraic]{0}{9}{t| 3*Ex(-t)*(1+t+t^2/2+t^3/6)}
\pscurvepoints{0}{9}{t| 3*Ex(-t)*(1+t+t^2/2+t^3/6)}{P}%
\pspolylineticks[Os=1,Ds=1,ticksize=0 0]{P}{ ds }{0}{9}%
\multido{\i=0+1}{9}{\psdot[dotscale=1.5,dotstyle=o](PTick\i)}%
% ticks at s=1,2,... , start at tick index 0, set 9 ticks
% the tick at s=1 has index 0
% ticks at s=1,2... (increment function = distance)
\multido{\i=0+3}{3}{\Put[rot=(PTangent\i)]{7pt;(PNormal\i)}(PTick\i){PTick\i}}%
\uput[-135](PTick1){$s=2$}%
\end{pspicture}
```

This variant also has no visible ticks, but makes a color gradient along the curve based on arc-length from the start.



```
\begin{pspicture}(-1,-1)(10,4)
\psparametricplot[plotpoints=200,linecolor=white]{0}{360}{ t cos 1 add 4 mul t 1 add 20 div
    ln 2 div 1 add }
\pscurvepoints[plotpoints=200]{0}{360}{ t cos 1 add 4 mul t 1 add 20 div ln 2 div 1 add }{P
    }%
\pspolylineticks[Os=0,Ds=.2,ticksize=0 0]{P}{ ds }{0}{90}%
\definecolorseries{ctest}{hsb}{last}{green}{violet}
\resetcolorseries[88]{ctest}%
\multido{\iA=0+1,\iB=1+1}{87}{\psline[linewidth=2pt,linecolor=ctest!![\iB](PTick\iA)(PTick\
    iB)}%
\end{pspicture}
```

Here is a another variant of this technique which allows arrows to be placed at locations on the curve where the metric takes particular values.



```
\begin{pspicture}(-1,-1)(10,4.5)
\psparametricplot[plotpoints=100]{0}{360}{t cos 1 add 5 mul t sin 1 add 2 mul}
\pscurvepoints[plotpoints=100]{0}{360}{t cos 1 add 5 mul t sin 1 add 2 mul}{P}%
\pspolylineticks[Os=0,Ds=2.3,ticksize=0 0]{P}{ ds }{0}{10}% distance
\multido{\i=0+1}{10}{\psrline[arrows=->,arrowscale=1.5](PTick\i)(2pt;{(PTangent\i)})}%
\end{pspicture}
```

30. Troubleshooting

If you get PostScript errors when you process your file, the most likely culprit is the function you specified to define the metric. There are some things to look out for:

- If `metricFunction`, the function you specify in PostScript code must involve only `x` and `y`, and must leave exactly one real value on the stack as a result of substituting specific values for `x` and `y`. The function must be strictly increasing on the curve.
- If `metricFunction=false`(the default), the function you specify in PostScript code must involve only the variables `x`, `y`, `dx`, `dy`, `ds` (where `ds` is defined to be the arc-length element `dx dup mul dy dup mul add sqrt`, and must leave exactly one strictly positive real value on the stack when specific values are substituted for those variables. The constant function `1` gives equal weight to each segment in the curve, so in effect it gives you the original parametrization, up to a constant factor.
- If the function you specify in `\parametricplot` and `\pscurvepoints` is `algebraic`, make sure you follow precisely the syntax it understands. In complex cases, PostScript may be the safer solution.
- It is unwise to use a different resolution for `\psparametricplot` and `\pscurvepoints`. The default value of `plotpoints=50` is marginal except for modest curve segments, and 200 should suffice for most smooth curves.

31. Transparent colors

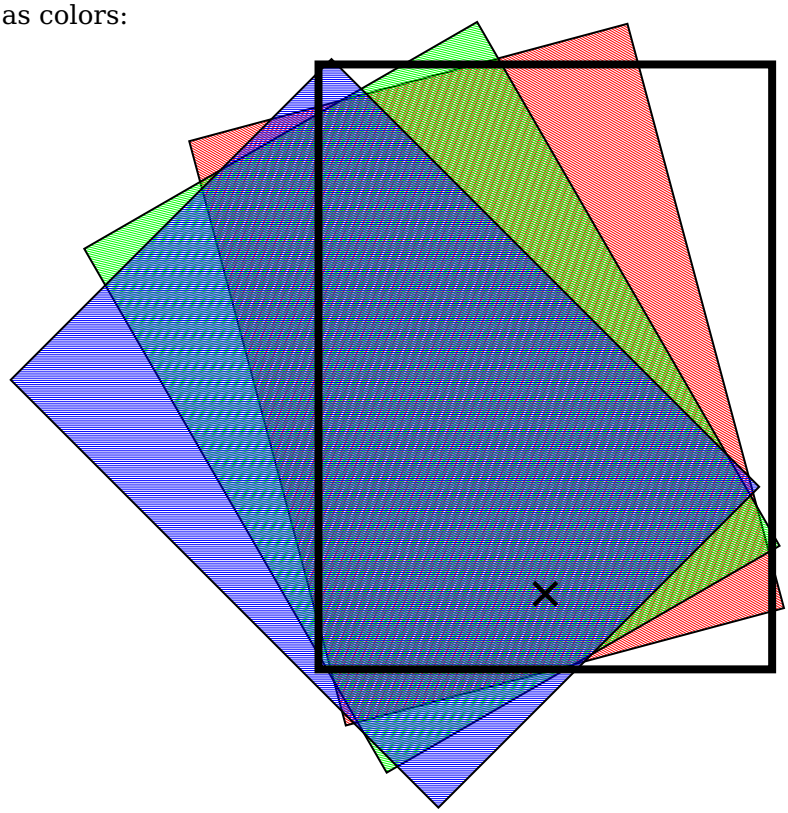
Transparency is now part of the main `pstricks` package. But pay attention, the names and syntax have changed and you need to run `ps2pdf` with the option `-dCompatibilityLevel=1.4`.

32. „Manipulating transparent colors”

pstricks-add supports real transparency and a simulated one with hatch lines:

```
\def\defineTColor{\@ifnextchar[{\defineTColor@i}{\defineTColor@i[]}}
\def\defineTColor@i[#1]#2#3{% transparency "Colors"
  \newpsstyle{#2}{%
    fillstyle=vlines,hatchwidth=0.1\pslinewidth,
    hatchsep=1\pslinewidth,hatchcolor=#3,#1%
  }%
}
\defineTColor{TRed}{red}
\defineTColor{TGreen}{green}
\defineTColor{TBlue}{blue}
```

There are three predefined "'transparent"' colors TRed, TGreen, TBlue. They are used as PSTricks styles and not as colors:



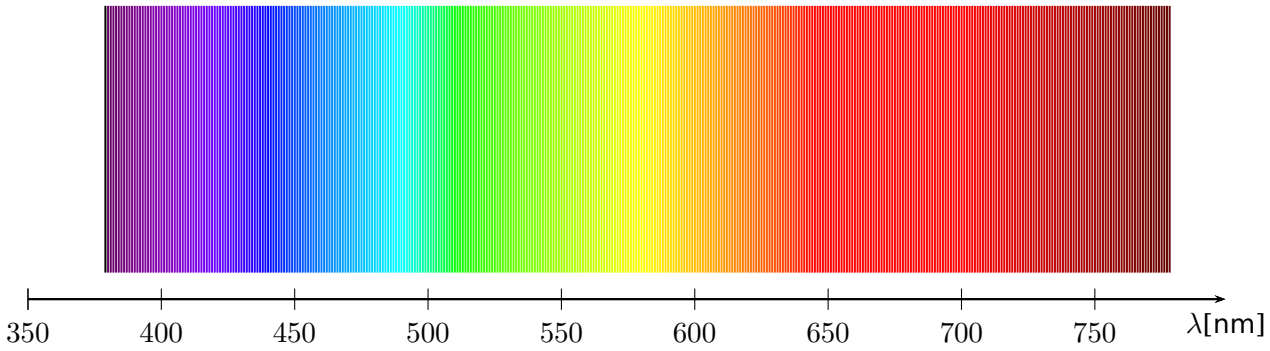
```
\begin{pspicture}(-3,-5)(5,5)
\psframe(-1,-3)(5,5) % objet de base
\psrotate(2,-2){15}{%
  \psframe[style=TRed](-1,-3)(5,5)}
\psrotate(2,-2){30}{%
  \psframe[style=TGreen](-1,-3)(5,5)}
\psrotate(2,-2){45}{%
  \psframe[style=TBlue](-1,-3)(5,5)}
\psframe[linewidth=3pt](-1,-3)(5,5)
\psdots[dotstyle=+,dotangle=45,dotscale=3](2,-2) % centre de la rotation
\end{pspicture}
```

33. Calculated colors

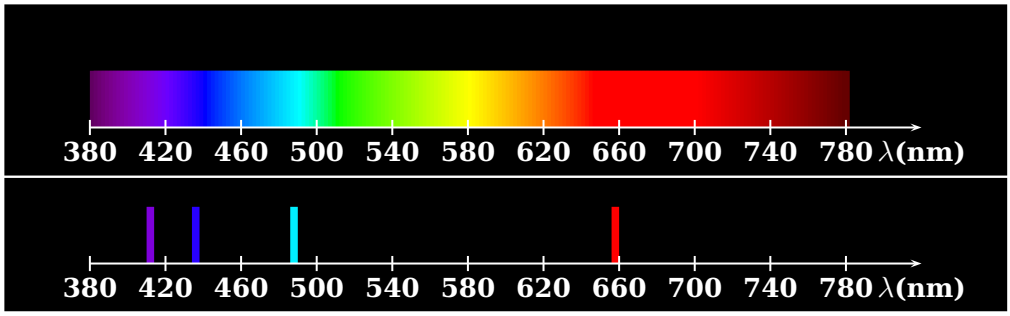
The xcolor package (version 2.6) has a new feature for defining colors:

```
\definecolor[ps]{<name>}{<model>}{< PS code >}
```

model can be one of the color models, which PostScriptwill understand, e.g. rgb. With this definition the color is calculated on the PostScriptside.




```
\definecolor[ps]{bl}{rgb}{tx@addDict begin Red Green Blue end}%
\psset{unit=1bp}
\begin{pspicture}(0,-30)(400,100)
\multido{\iLAMBDA=0+1}{400}{%
  \pstVerb{
    \iLAMBDA\space 379 add dup /lambda exch def
    tx@addDict begin wavelengthToRGB end
  }%
  \psline[linecolor=bl](\iLAMBDA,0)(\iLAMBDA,100)%
}
\psaxes[yAxis=false,Ox=350,dx=50bp,Dx=50]{->}(-29,-10)(420,100)
\uput[-90](420,-10){$\lambda$[\textsf{nm}]}
\end{pspicture}
```



Spectrum of hydrogen emission (Manuel Luque)

```
\newcommand\Touch{%
\psframe[linestyle=none,fillstyle=solid,fillcolor=bl,dimen=middle](0.1,0.75)}
\definecolor[ps]{bl}{rgb}{tx@addDict begin Red Green Blue end}%
% Echelle 1cm <-> 40 nm
%       1 nm <-> 0.025 cm
\psframebox[fillstyle=solid,fillcolor=black]{%
\begin{pspicture}(-1,-0.5)(12,1.5)
\multido{\iLAMBDA=380+2}{200}{%
  \pstVerb{
```

```
   /lambda \iLAMBDA\space def
   lambda
   tx@addDict begin wavelengthToRGB end
  }%
 \rput(! lambda 0.025 mul 9.5 sub 0){\Touch}
}
\multido{\n=0+1,\iDiv=380+40}{11}{%
   \psline[linecolor=white](\n,0.1)(\n,-0.1)
   \uput[270](\n,0){\textbf{\white\iDiv}}}
   \psline[linecolor=white]{->}(11,0)
   \uput[270](11,0){\textbf{\white$\lambda$(nm)}}
\end{pspicture}}

\psframebox[fillstyle=solid,fillcolor=black]{%
\begin{pspicture}(-1,-0.5)(12,1)
  \pstVerb{
   /lambda 656 def
   lambda
   tx@addDict begin wavelengthToRGB end
  }%
 \rput(! 656 0.025 mul 9.5 sub 0){\Touch}
  \pstVerb{
   /lambda 486 def
   lambda
   tx@addDict begin wavelengthToRGB end
  }%
 \rput(! 486 0.025 mul 9.5 sub 0){\Touch}
   \pstVerb{
   /lambda 434 def
   lambda
   tx@addDict begin wavelengthToRGB end
  }%
 \rput(! 434 0.025 mul 9.5 sub 0){\Touch}
  \pstVerb{
   /lambda 410 def
   lambda
   tx@addDict begin wavelengthToRGB end
  }%
 \rput(! 410 0.025 mul 9.5 sub 0){\Touch}
\multido{\n=0+1,\iDiv=380+40}{11}{%
   \psline[linecolor=white](\n,0.1)(\n,-0.1)
   \uput[270](\n,0){\textbf{\white\iDiv}}}
   \psline[linecolor=white]{->}(11,0)
   \uput[270](11,0){\textbf{\white$\lambda$(nm)}}
\end{pspicture}}

Spectrum of hydrogen emission (Manuel Luque)
```

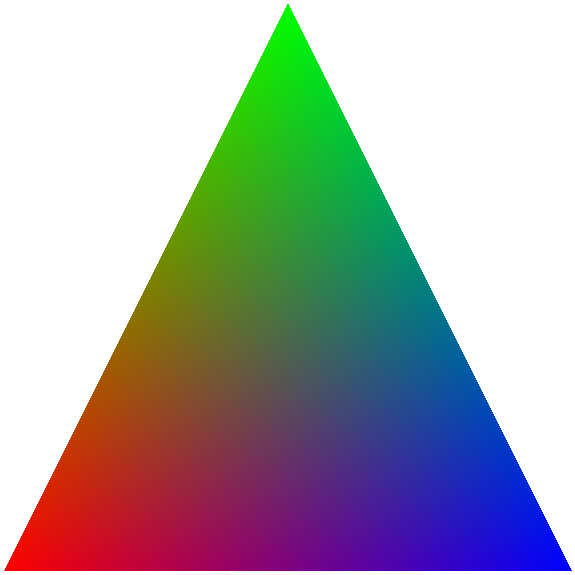
34. Gouraud shading

Gouraud shading is a method used in computer graphics to simulate the differing effects of light and colour across the surface of an object. In practice, Gouraud shading is used to achieve smooth lighting on low-polygon surfaces without the heavy computational

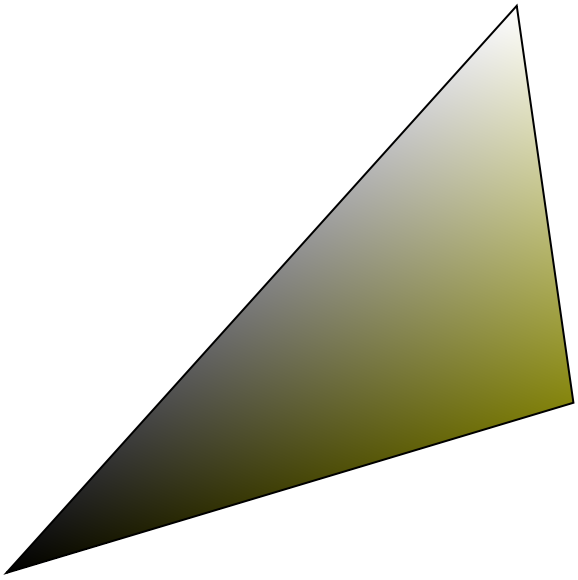
requirements of calculating lighting for each pixel. The technique was first presented by Henri Gouraud in 1971.
http://www.wikipedia.org

PostScript level 3 supports this kind of shading and it can only be seen with Acroread 7 or later. The syntax is easy:

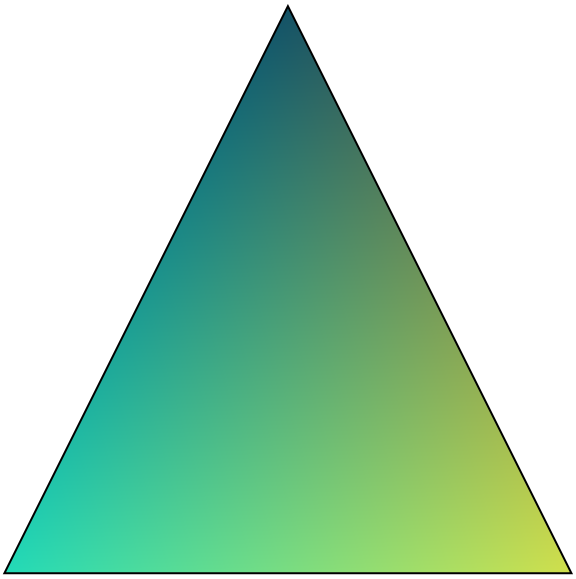
```
\psGTriangle(x1,y1)(x2,y2)(x3,y3){color1}{color2}{color3}
```



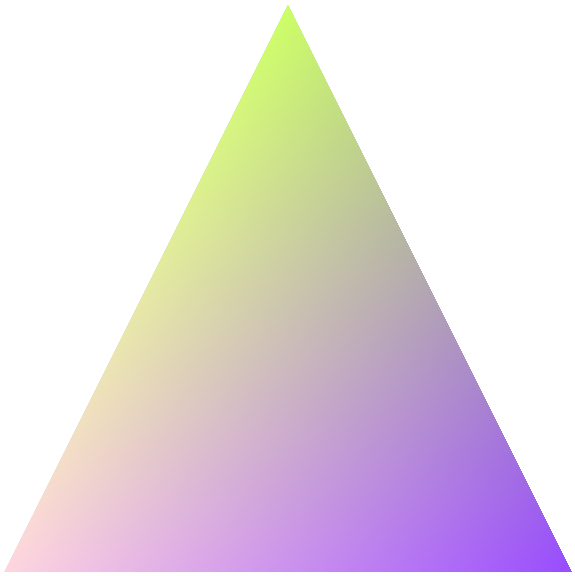
```
\begin{pspicture}(0,-.25)(10,10)
  \psGTriangle(0,0)(5,10)(10,0){red}{green}{blue}
\end{pspicture}
```



```
\begin{pspicture}(0,-.25)(10,10)
  \psGTriangle*(0,0)(9,10)(10,3){black}{white!50}{red!50!green!95}
\end{pspicture}
```



```
\begin{pspicture}(0,-.25)(10,10)
  \psGTriangle*(0,0)(5,10)(10,0){-red!100!green!84!blue!86}
                        {-red!80!green!100!blue!40}
                        {-red!60!green!30!blue!100}
\end{pspicture}
```



```
\definecolor{rose}{rgb}{1.00, 0.84, 0.88}
\definecolor{vertpommepasmure}{rgb}{0.80, 1.0, 0.40}
\definecolor{fushia}{rgb}{0.60, 0.30, 1.0}
\begin{pspicture}(0,-.25)(10,10)
  \psGTriangle(0,0)(5,10)(10,0){rose}{vertpommepasmure}{fushia}
\end{pspicture}
```

35. Internal color macros

The internal macros \pswavelengthToRGB and \pswavelengthToRGB can be used for own purposed. They are defines as follows:

```
\def\pswavelengthToGRAY{ tx@addDict begin wavelengthToGRAY end }
\def\pswavelengthToRGB{ tx@addDict begin wavelengthToRGB Red Green Blue end }
```

both macros leave the value(s) on the stack which then can be used for further manipulating or setting the color with `setgray` or `setrgbcolor`. For an example see Section 27.

A. \resetOptions

Sometimes it is difficult to know what options, which are changed inside a long document, are different to the default ones. With this macro all options belonging to `pst-plot` can be reset. This refers to all options of the packages `pstricks`, `pst-plot` and `pst-node`.

B. PostScript

PostScript uses the stack system and the LIFO system, "'Last In, First Out"'.

Table 3: Some primitive PostScript macros

Function	*Meaning* *on stack before* $\rightarrow$ *after*
`add`	$x \quad y \rightarrow x+y$
`sub`	$x \quad y \rightarrow x-y$
`mul`	$x \quad y \rightarrow x \times y$
`div`	$x \quad y \rightarrow x \div y$
`sqrt`	$x \rightarrow \sqrt{x}$
`abs`	$x \rightarrow \lvert x \rvert$
`neg`	$x \rightarrow -x$
`cos`	$x \rightarrow \cos(x)$ (`x in degrees`)
`sin`	$x \rightarrow \sin(x)$ (`x in degrees`)
`tan`	$x \rightarrow \tan(x)$ (`x in degrees`)
`atan`	$y \quad x \rightarrow \angle(\vec{O}x;\vec{OM})$ (`in degrees of` $M(x,y)$)
`ln`	$x \rightarrow \ln(x)$
`log`	$x \rightarrow \log(x)$
`array`	$n \rightarrow v$ (of dimension n)
`aload`	$v \rightarrow x_1 \quad x_2 \quad \cdots \quad x_n \quad v$
`astore`	$x_1 \quad x_2 \quad \cdots \quad x_n \quad v \rightarrow [v]$
`pop`	$x \rightarrow$ -
`dup`	$x \rightarrow x \quad x$

C. List of all optional arguments for pstricks-add

Key	Type	Default
blName	command	
bcName	command	
brName	command	
clName	command	
ccName	command	
crName	command	
tlName	command	
tcName	command	
trName	command	
CMYK	boolean	true
intSeparator	ordinary	[none]
braceWidth	ordinary	[none]
bracePos	ordinary	[none]
braceWidthInner	ordinary	[none]
braceWidthOuter	ordinary	[none]
parallelogramsep	ordinary	3mm
Os	ordinary	0
Ds	ordinary	1
metricInitValue	ordinary	0
metricFunction	boolean	true
randomPoints	ordinary	1000
randInit	ordinary	rrand
color	boolean	true
fillcolorA	ordinary	blue
fillcolorB	ordinary	red
countDots	boolean	true
whichabs	ordinary	[none]
whichord	ordinary	[none]
plotfuncx	ordinary	[none]
plotfuncy	ordinary	[none]
expression	ordinary	[none]
buildvector	boolean	true
varsteptol	ordinary	[none]
adamsorder	ordinary	[none]
saveSumValue	boolean	true
StepType	ordinary	[none]
noVerticalLines	boolean	true
Derive	ordinary	[none]
Tnormal	boolean	true
GetFinalState	boolean	true
filename	ordinary	[none]
saveData	boolean	true
Xoffset	ordinary	0pt
Yoffset	ordinary	0pt

Continued on next page

Continued from previous page

Key	Type	Default
XYoffset	ordinary	0pt
colorType	ordinary	0
colorTypeDef	ordinary	
chartStyle	ordinary	[none]
chartColor	ordinary	[none]
chartSep	ordinary	[none]
chartStack	ordinary	[none]
chartStackDepth	ordinary	[none]
chartStackWidth	ordinary	[none]
chartHeight	ordinary	[none]
uselinecolor	boolean	true
userColor	ordinary	[none]
chartNodeI	ordinary	[none]
chartNodeO	ordinary	[none]
cancelType	ordinary	[none]
markAngle	boolean	true
basename	ordinary	[none]
rotate	ordinary	[none]
colSteps	ordinary	[none]
colored	boolean	true
colorOffset	ordinary	

References

[1] Denis Girou. “Présentation de PSTricks”. In: *Cahier GUTenberg* 16 (Apr. 1994), pp. 21–70.

[2] Michel Goosens et al. *The LaTeX Graphics Companion*. 2nd ed. Reading, Mass.: Addison-Wesley Publishing Company, 2007.

[3] Alan Hoenig. *TeX Unbound: LaTeX & TeX Strategies, Fonts, Graphics, and More*. London: Oxford University Press, 1998.

[4] Laura E. Jackson and Herbert Voß. “Die Plot-Funktionen von `pst-plot`”. In: *Die TeXnische Komödie* 2/02 (June 2002), pp. 27–34.

[5] Nikolai G. Kollock. *PostScript richtig eingesetzt: vom Konzept zum praktischen Einsatz*. Vaterstetten: IWT, 1989.

[6] Frank Mittelbach and Michel Goosens et al. *Der LaTeX Begleiter*. 2nd ed. München: Pearson Education, 2005.

[7] Frank Mittelbach and Michel Goosens et al. *The LaTeX Companion*. 2nd ed. Boston: Addison-Wesley Publishing Company, 2004.

[8] Herbert Voß. “Die mathematischen Funktionen von PostScript”. In: *Die TeXnische Komödie* 1/02 (Mar. 2002).

[9] Herbert Voß. *PSTricks Grafik für TeX und LaTeX*. 7th ed. Heidelberg/Berlin: DANTE – Lehmanns, 2016.

[10] Herbert Voß. *PSTricks Graphics for LaTeX*. 1st ed. Cambridge: UIT, 2011.

[11] Timothy Van Zandt. *multido.tex - a loop macro, that supports fixed-point addition*. 1997. url: CTAN:/graphics/pstricks/generic/multido.tex.

[12] Timothy Van Zandt. *pst-plot: Plotting two dimensional functions and data*. 1999. url: CTAN:graphics/pstricks/generic/pst-plot.tex.

[13] Timothy Van Zandt and Denis Girou. “Inside PSTricks”. In: *TUGboat* 15 (Sept. 1994), pp. 239–246.