PSTricks

News - 2010

new macros and bugfixes for the basic package pstricks

December 13, 2017

2010

Package author(s):
Herbert Voß

Contents

Part I.
pstricks – package

1. General

There exists a new document class pst-doc for writing PSTricks documentations, like this news document. It depends on the KOMA-Script document class scrartcl. pst-doc defines a lot of special macros to create a good index. Take one of the already existing package documentation and look into the source file. Then it will be easy to understand, how all these macros have to be used.

When running pdflatex the title page is created with boxes and inserted with the macro \AddToShipoutPic from the package eso-pic. It inserts the background title page image pst-doc-pdf to use directly pdflatex. When running latex the title page is created with PSTricks macros.This allows to use the Perl script pst2pdf or the package pst-pdf or auto-pst-pdf or any other program/package which supports PostScript code in the document.

2. pstricks.sty

2.1. New optional argument

With the setting of the optional argument pdf the package auto-pst-pdf will be loaded by PSTricks. This requires that you run pdflatex as

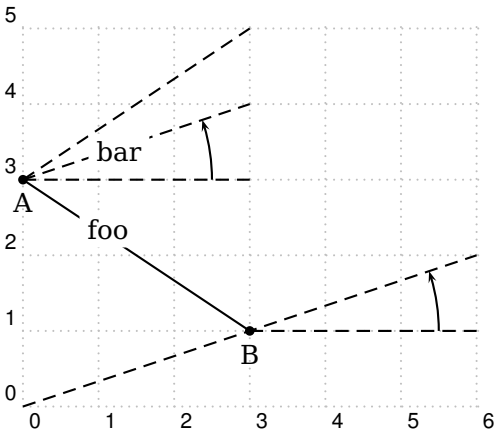
```
pdflatex --shell-escape <file>    % TeX Live users
pdflatex --enable-write18 <file>% MiKTeX users
```

The package exports the pspicture environments into single images which are collected in a created file <file>-pics.pdf and inserted automatically in the last pdflatex run.

3. pstricks.tex (0.2– 1999/07/01)

3.1. Coordinates

With the setting \SpecialCoor the package allows different kinds of coordinates. The macro \uput can now be used in a different way. The default behaviour for nodes with a relative point puts its argument *without* rotation on the line $\overline{AB}$. When using the prefix > before the node or the x-value for cartesian coordinates, the behaviour is different. Now the angle between the line $\overline{OB}$ and the horizontal line is taken into account and the placement of the argument of \uput is different to the default behaviour.

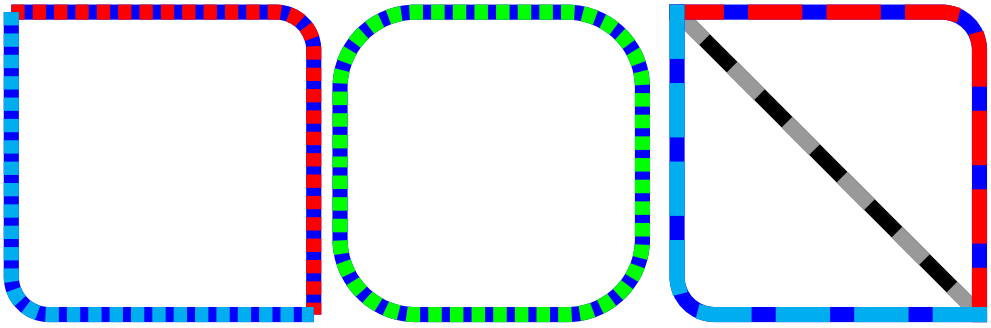



```
\begin{pspicture}[showgrid](-0.25,-0.25)(6,5)
\pnodes(0,3){A}(3,1){B}
\psline[showpoints](A)(B)
\uput[-90](A){A}\uput[-90](B){B}
\psline[linestyle=dashed](A)(3,4)
\psline[linestyle=dashed](A)(3,5)
\psline[linestyle=dashed](A)(3,3)
\psline[linestyle=dashed](6,2)
\psline[linestyle=dashed](B)(6,1)
\psarc{->}(0,3){2.5}{0}{(3,1)}
\psarc{->}(3,1){2.5}{0}{(3,1)}
\uput*{1cm}[(B)](A){foo} \uput*{1cm}[(B)](>A){bar}
\end{pspicture}
```

3.2. New optional arguments

The new arguments are only valid for the macros \psellipse, \pscircle, \psarc. \psellipticarc, \pscurve, \psplot, and \psparametricplot.

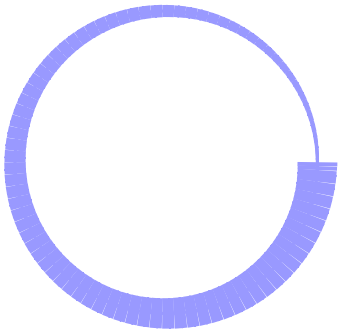
name	*type*	*default*	*description*
dashcolor	*color*	\relax	for colored dashed lines
startLW	*length*	\pslinewidth	starting linewidth
endLW	*length*	\pslinewidth	ending linewidth
startWL	*integer*	380	starting wave length
endWL	*integer*	780	ending wave length
variableLW	*boolean*	false	use variable linewidth
variableColor	*boolean*	false	use variable color



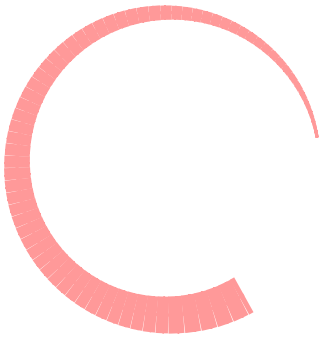
```
\psset{linewidth=2mm,linestyle=dashed}
\begin{pspicture}(4,-4)
\psline[linecolor=blue,dashcolor=red,linearc=0.5](0,0)(4,0)(4,-4)
\psline[linecolor=blue,dashcolor=cyan,linearc=0.5](0,0)(0,-4)(4,-4)
\end{pspicture}\quad
\begin{pspicture}(4,4)
\psframe[linecolor=blue,dashcolor=green,framearc=0.5](0,0)(4,4)
\end{pspicture}
\quad \psset{linecap=2,dash=5mm 5mm }
\begin{pspicture}(4,-4)
\psline[linecolor=black,dashcolor=black!40,linecap=0](0,0)(4,-4)
\psline[linecolor=blue,dashcolor=red,linearc=0.5](0,0)(4,0)(4,-4)
\psline[linecolor=blue,dashcolor=cyan,linearc=0.5](0,0)(0,-4)(4,-4)
\end{pspicture}
```



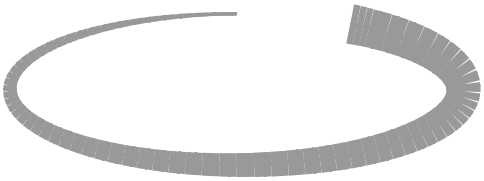
```
\psset{endLW=15pt}
\begin{pspicture}(-3.5,-2.5)(3.5,2.5)
\psellipse[linejoin=2,variableLW,startLW=1pt,
  linecolor=green!40](0,0)(3,1)
\end{pspicture}
```



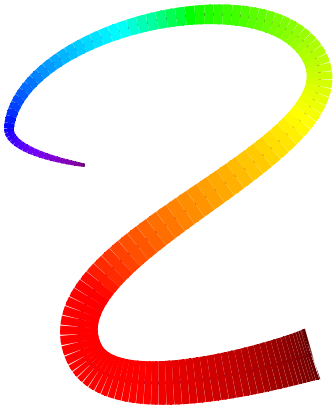
```
\psset{endLW=15pt}
\begin{pspicture}(-2.5,-2.5)(2.5,2.5)
\pscircle[variableLW,startLW=1pt,
  linecolor=blue!40]{2}
\end{pspicture}
```



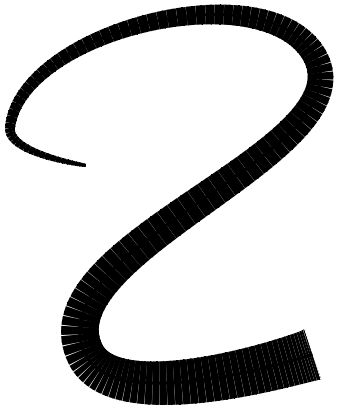
```
\psset{endLW=15pt}
\begin{pspicture}(-2.5,-2.5)(2.5,2.5)
\psarc[variableLW,startLW=1pt,
  linecolor=red!40](0,0){2}{10}{300}
\end{pspicture}
```



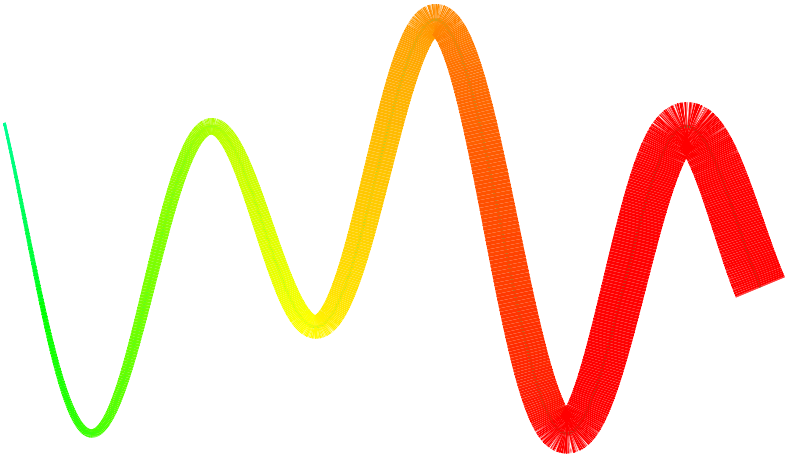
```
\psset{endLW=15pt}
\begin{pspicture}(-3.5,-2.5)(3.5,2.5)
\psellipticarc[variableLW,startLW=1pt,
  linecolor=black!40](0,0)(3,1){90}{30}
\end{pspicture}
```



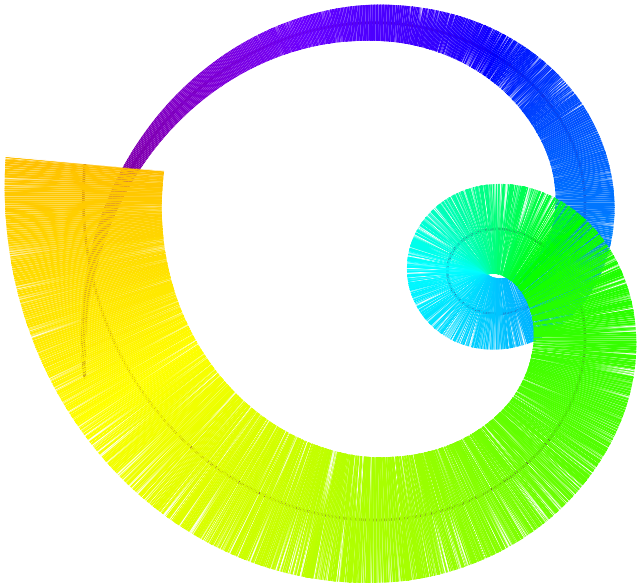
```
\begin{pspicture}(-2.5,-2.5)(2.5,2.5)
\pscurve[variableLW,startLW=1pt,endLW=20pt,
  variableColor](-1,0.5)(-2,1)(2,2)(-1,-2)(2,-2)
\end{pspicture}
```



```
\begin{pspicture}(-2.5,-2.5)(2.5,2.5)
\pscurve[variableLW,startLW=1pt,endLW=20pt]%
  (-1,0.5)(-2,1)(2,2)(-1,-2)(2,-2)
\end{pspicture}
```



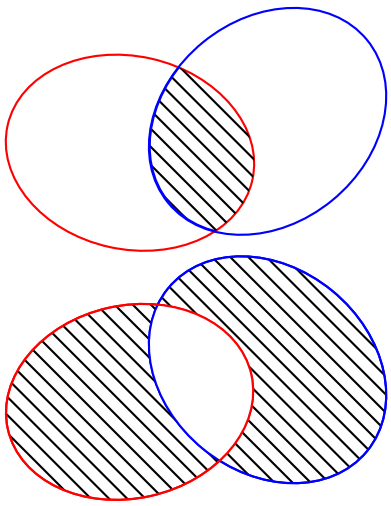
```
\begin{pspicture}(-5,-3)(5,3)
\psplot[variableLW,startLW=1pt,endLW=20pt,
  linecolor=magenta!60,variableColor,
  algebraic,plotpoints=3000,startWL=500,
  endWL=700]{-5}{5}{2*sin(2*x)+cos(x)}
\end{pspicture}
```



```
\psset{endLW=24pt}
\begin{pspicture}(-5,-5)(5,5)
\psparametricplot[variableLW,startLW=1pt,
  endLW=60pt,linecolor=red,variableColor,
  algebraic,plotpoints=3000,plotstyle=curve,
  opacity=0.4,strokeopacity=0.4,
  endWL=600]{-5}{5}{t*sin(t) | t*cos(t)}
\end{pspicture}
```

3.3. Macro \psellipse

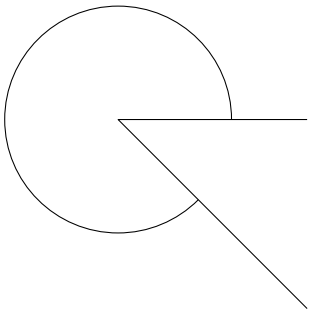
To rotate an ellipse the already existing keyword rot can be used. This is easier than using the \rput command and its optional argument for rotating.



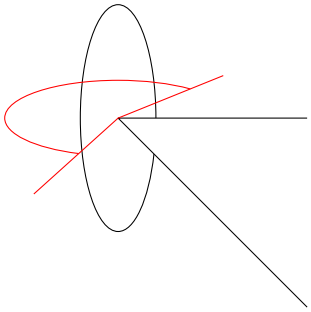
```
\psset{unit=0.25}
\begin{pspicture}(-1,5)(20,18)
\psclip{\psellipse[linecolor=red,
  rot=-12.606](5.821,10.04)(6.633,5.103)}
  \psellipse[linecolor=blue,fillstyle=vlines,
    rot=39.29](13.141,11.721)(6.8,5.4)
\endpsclip
\psellipse[linecolor=blue,rot=39.29](13.1,11.7)(6.8,5.4)
\end{pspicture}

\begin{pspicture}(-1,5)(20,18)
\psellipse[linecolor=blue,rot=-39.29,
  fillstyle=vlines](13.1,11.7)(6.8,5.4)
\psclip{\psellipse[linecolor=red,rot=12.6,
  fillstyle=vlines](5.8,10)(6.6,5.1)}
  \psellipse*[linecolor=white,rot=-39.29](13.1,11.7)(6.8,5.4)
\endpsclip
\psellipse[linecolor=blue,rot=-39.29](13.1,11.7)(6.8,5.4)
\psellipse[linecolor=red,rot=12.6](5.8,10)(6.6,5.1)
\end{pspicture}
```

3.4. Macro \psellipticarc

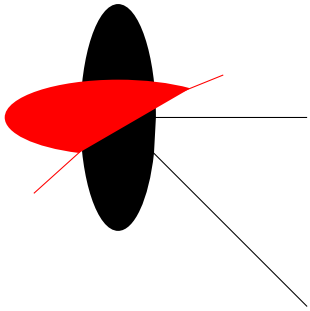
In a circle the angle is proportional to the bow: $b = r\alpha$. In an elliptic arc this is no more the case, which is the reason why angles are internally corrected by PSTricks, to get the same arc lengthts for different radii:



```
\psset{unit=0.5cm}
\begin{pspicture}(-5.5,-5.5)(5.5,5.5)%
\psset{linewidth=0.4pt,linejoin=1}
\psline(5,0)(0,0)(5,-5)
\psellipticarc(0,0)(3,3){0}{315}
\end{pspicture}%
```

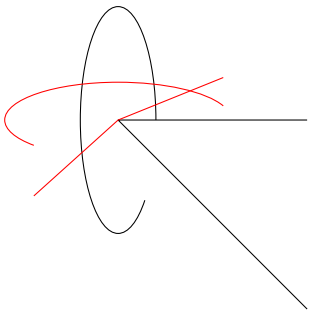


```
\psset{unit=0.5cm}
\begin{pspicture}(-5.5,-5.5)(5.5,5.5)%
\psset{linewidth=0.4pt,linejoin=1}
\psline(5,0)(0,0)(5,-5)
\psellipticarc(0,0)(1,3){0}{315}%
\psset{linecolor=red}
\psellipticarc(0,0)(3,1){22}{222}%
\psline(3;22)\psline(3;222)
\end{pspicture}%
```

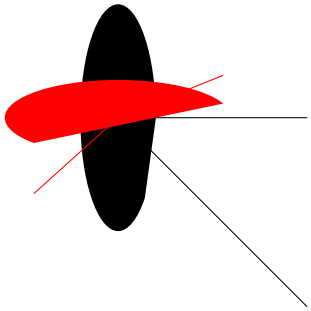


```
\psset{unit=0.5cm}
\begin{pspicture}(-5.5,-5.5)(5.5,5.5)%
\psset{linewidth=0.4pt,linejoin=1}
\psline(5,0)(0,0)(5,-5)
\psellipticarc*(0,0)(1,3){0}{315}%
\psset{linecolor=red}
\psellipticarc*(0,0)(3,1){22}{222}%
\psline(3;22)\psline(3;222)
\end{pspicture}%
```

If you do not want the angle correction, then use the keyword setting `correctAngle=false`:



```
\psset{unit=0.5cm}
\begin{pspicture}(-5.5,-5.5)(5.5,5.5)%
\psset{linewidth=0.4pt,linejoin=1,
  correctAngle=false}
\psline(5,0)(0,0)(5,-5)
\psellipticarc(0,0)(1,3){0}{315}%
\psset{linecolor=red}
\psellipticarc(0,0)(3,1){22}{222}%
\psline(3;22)\psline(3;222)
\end{pspicture}%
```



```
\psset{unit=0.5cm}
\begin{pspicture}(-5.5,-5.5)(5.5,5.5)%
\psset{linewidth=0.4pt,linejoin=1,
  correctAngle=false}
\psline(5,0)(0,0)(5,-5)
\psellipticarc*(0,0)(1,3){0}{315}%
\psset{linecolor=red}
\psellipticarc*(0,0)(3,1){22}{222}%
\psline(3;22)\psline(3;222)
\end{pspicture}%
```

3.5. Option algebraic

The option algebraic moved from the other packages into the main package pstricks to get rid of the dependencies.

By default the function in \psplot has to be described in Reversed Polish Notation. The option algebraic allows you to do this in the common algebraic notation. E.g.:

RPN	algebraic
x ln	ln(x)
x cos 2.71 x neg 10 div exp mul	cos(x)*2.71^(-x/10)
1 x div cos 4 mul	4*cos(1/x)
t cos t sin	cos(t)|sin(t)

Setting the option algebraic, allow the user to describe all expression to be written in the classical algebraic notation (infix notation). The four arithmetic operations are obviously defined +-*/, and also the exponential operator ^. The natural priorities are used : $3+4\times5^5 = 3+(4\times(5^5))$, and by default the computation is done from left to right. The following functions are defined :

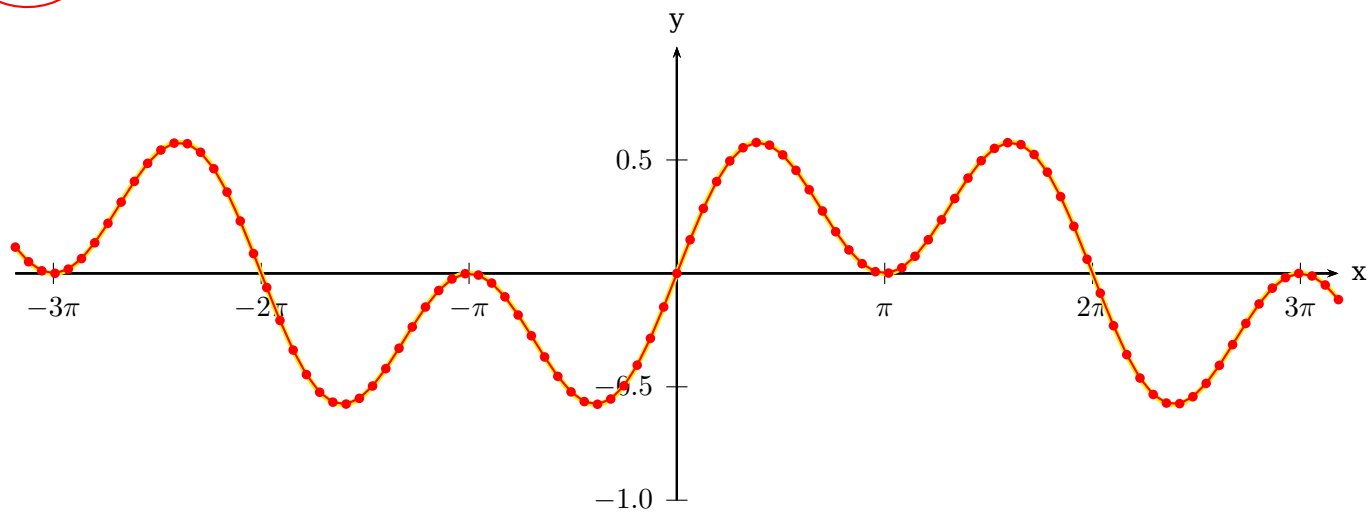
sin, cos, tan, acos, asin	in radians
log, ln	
ceiling, floor, truncate, round	
sqrt	square root
abs	absolute value
fact	for the factorial
Sum	for building sums
IfTE	for an easy case structure

These options can be used with **all** plot macros.

Using the option algebraic implies that all angles have to be in radians!

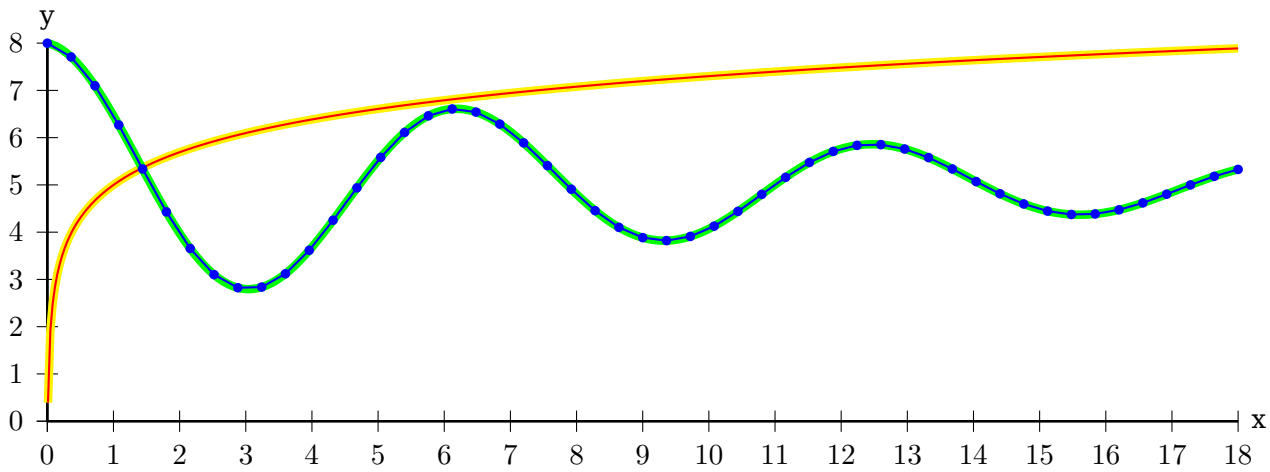
For the \parametricplot the two parts must be divided by the | character:

```
\begin{pspicture}(-0.5,-0.5)(0.5,0.5)
\parametricplot[algebraic,linecolor=red]{-3.14}{3.14}{cos(t)|sin(t)}
\end{pspicture}
```




```
\psset{lly=-0.5cm}
\psgraph[trigLabels,dx=\psPi,dy=0.5,Dy=0.5]{->}(0,0)(-10,-1)(10,1){\linewidth}{6cm}
  \psset{algebraic,plotpoints=1000}
```

```
  \psplot[linecolor=yellow,linewidth=2pt]{-10}{10}{0.75*sin(x)*cos(x/2)}
  \psplot[linecolor=red,showpoints=true,plotpoints=101]{-10}{10}{0.75*sin(x)*cos(x/2)}
\endpsgraph
```

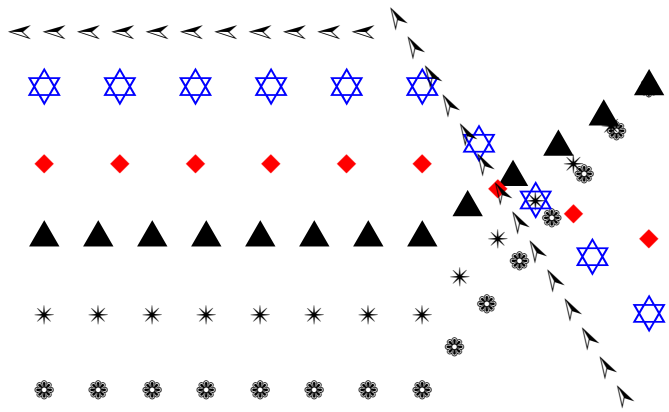



```
\psset{lly=-0.5cm}
\psgraph(0,-5)(18,3){0.9\linewidth}{5cm}
  \psset{algebraic,plotpoints=501}
  \psplot[linecolor=yellow, linewidth=4\pslinewidth]{0.01}{18}{ln(x)}
  \psplot[linecolor=red]{0.01}{18}{ln(x)}
  \psplot[linecolor=yellow,linewidth=4\pslinewidth]{0}{18}{3*cos(x)*2.71^(-x/10)}
  \psplot[linecolor=blue,showpoints=true,plotpoints=51]{0}{18}{3*cos(x)*2.71^(-x/10)}
\endpsgraph
```

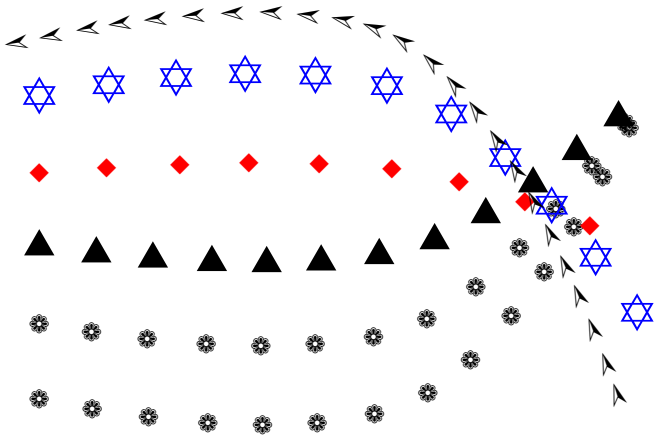
4. New linestyle symbol

Instead of drawing a continous line or curve for a series of coordinates, one can now out a symbol in a given size, direction, and step. This works only for the line style `symbol`. It takes the symbol defined by the optional argument `symbol`, which can have a single character or a octal number of three digits. The font is specified by the key `symbolFont`, which can take as argument one of the valid `PostScript`fonts or the internal `PSTricksDotFont`. If the symbol is given by a single character then the equivilant character in the given font is used. The difference between two symbols is set by `symbolStep` and the symbol rotation by `rotateSymbol`. For the first symbol there is an additional keyword `startAngle`. The default values for these new optional keywords are:

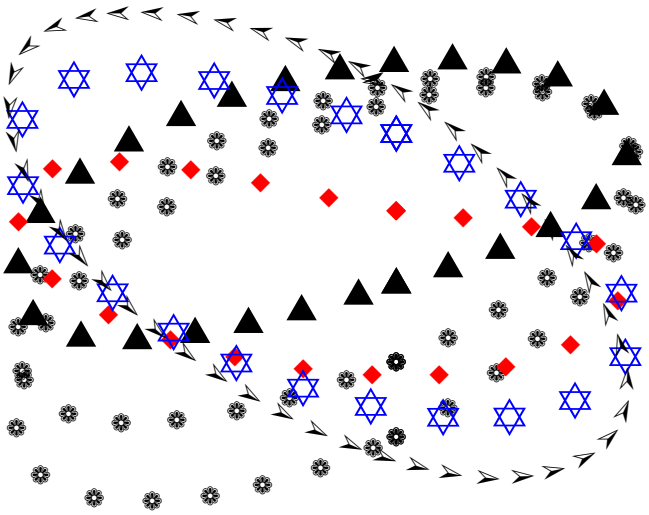
```
\psset[pst-base]{symbolStep=20pt}
\psset[pst-base]{symbolWidth=10pt}
\psset[pst-base]{symbolFont=Dingbats}
\psset[pst-base]{rotateSymbol=false}
\psset[pst-base]{startAngle=0}
```



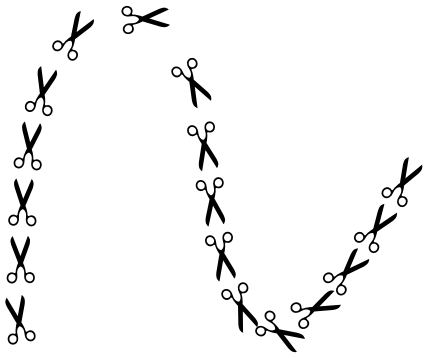
```
\pspicture(-1,-1)(8,6)
\psline[linestyle=symbol](0,0)(5,0)(8,4)
\psline[linestyle=symbol,symbol=T](0,1)(5,1)(8,4)
\psline[linestyle=symbol,symbol=u,symbolFont=PSTricksDotFont](0,2)(5,2)(8,4)
\psline[linestyle=symbol,symbol=u,symbolStep=25pt,linecolor=red](0,3)(5,3)(8,2)
\psline[linestyle=symbol,symbol=A,symbolStep=25pt,
  symbolWidth=20pt,linecolor=blue](0,4)(5,4)(8,1)
\psline[linestyle=symbol,symbol=342,rotateSymbol=true,symbolStep=12pt](0,5)(5,5)(8,0)
\endpspicture
```



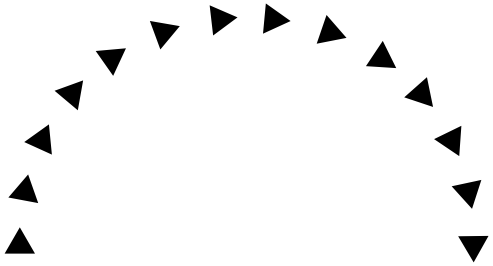
```
\pspicture(-1,-1)(8,6)
\pscurve[linestyle=symbol](0,0)(5,0)(8,4)
\pscurve[linestyle=symbol](0,1)(5,1)(8,4)
\pscurve[linestyle=symbol,symbol=u,symbolFont=PSTricksDotFont](0,2)(5,2)(8,4)
\pscurve[linestyle=symbol,symbol=u,symbolStep=25pt,linecolor=red](0,3)(5,3)(8,2)
\pscurve[linestyle=symbol,symbol=A,symbolStep=25pt,
  symbolWidth=20pt,linecolor=blue](0,4)(5,4)(8,1)
\pscurve[linestyle=symbol,symbol=342,rotateSymbol=true,
  startAngle=190,symbolStep=12pt](0,5)(5,5)(8,0)
\endpspicture
```



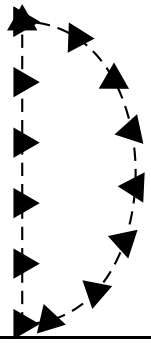
```
\pspicture(-1,-1)(8,6)
\psccurve[linestyle=symbol](0,0)(5,0)(8,4)
\psccurve[linestyle=symbol](0,1)(5,1)(8,4)
\psccurve[linestyle=symbol,symbol=u,symbolFont=PSTricksDotFont](0,2)(5,2)(8,4)
\psccurve[linestyle=symbol,symbol=u,symbolStep=25pt,linecolor=red](0,3)(5,3)(8,2)
\psccurve[linestyle=symbol,symbol=A,symbolStep=25pt,
  symbolWidth=20pt,linecolor=blue](0,4)(5,4)(8,1)
\psccurve[linestyle=symbol,symbol=342,rotateSymbol=true,
  startAngle=190,symbolStep=12pt](0,5)(5,5)(8,0)
\endpspicture
```



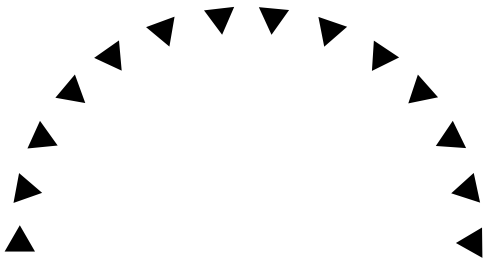
```
\pspicture(-1,-1)(5,4)
\pscurve[rotateSymbol=true,linestyle=symbol,
  rot=180,startAngle=100,symbol=",
    symbolWidth=20pt](0,0)(1,4)(3,0)(5,2)
\endpspicture
```



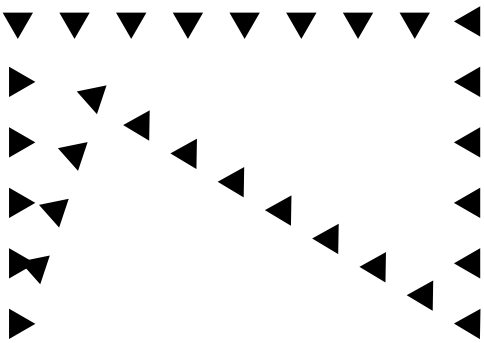
```
\pspicture(-1,-1)(6,4)
\psbezier[rotateSymbol=true,linestyle=symbol,symbol=u,
 symbolFont=PSTricksDotFont,rot=-90,startAngle=0](0,0)(0,4)(6,4)(6,0)
\endpspicture
```



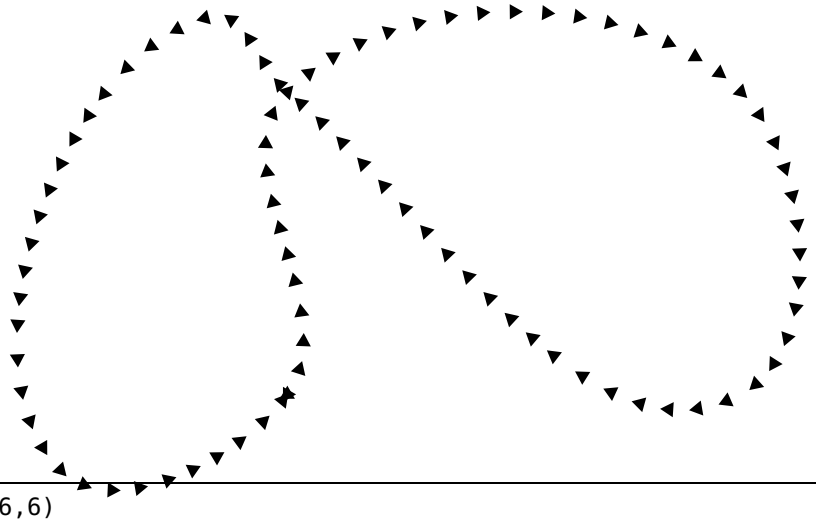
```
\psset{unit=0.5cm}
\pspicture(-1,-4)(6,4)
\pscbezier[rotateSymbol=true,linestyle=symbol,symbol=u,
 symbolFont=PSTricksDotFont](0,4)(4,4)(4,-4)(0,-4)
\pscbezier[linestyle=dashed](0,4)(4,4)(4,-4)(0,-4)
\endpspicture
```



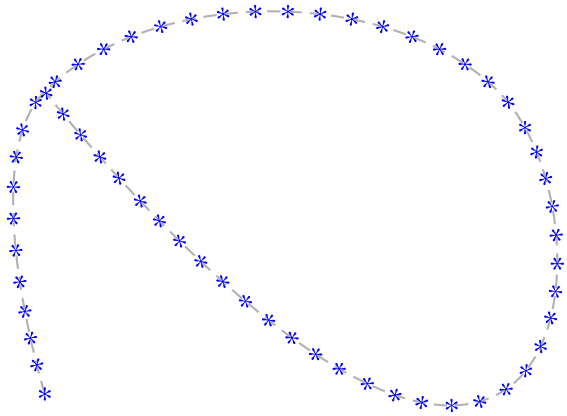
```
\pspicture(-1,-1)(6,4)
\psbezier[rotateSymbol=true,linestyle=symbol,symbol=u,
 symbolFont=PSTricksDotFont](0,0)(0,4)(6,4)(6,0)
\endpspicture
```



```
\pspicture(-1,-1)(6,4)
\pspolygon[rotateSymbol=true,linestyle=symbol,symbol=u,
 symbolFont=PSTricksDotFont](0,0)(0,4)(6,4)(6,0)(1,3)
\endpspicture
```



```
\pspicture(-3,-1)(6,6)
\psccurve[linestyle=symbol,symbol=u, rot=-90,rotateSymbol,
  symbolFont=PSTricksDotFont, symbolWidth=5pt, symbolStep=10pt
](-3,-1)(0,0)(0,4)(6,4)(6,0)(0,4)(-1,5)
\endpspicture
```



```
\pspicture(-1,-1)(6,6)
\pscurve[linestyle=dashed,linecolor=black!30](0,0)(0,4)(6,4)(6,0)(0,4)
\pscurve[rotateSymbol=true,linestyle=symbol,symbol=k,
  symbolFont=PSTricksDotFont, symbolWidth=5pt, symbolStep=10pt,linecolor=blue
](0,0)(0,4)(6,4)(6,0)(0,4)
\endpspicture
```

5. Numeric functions

All macros have a @ in their name, because they are only for internal use, but it is no problem to use them like other macros. One can define another name without a @:

```
\makeatletter
\let\pstdivide\pst@divide
\makeatother
```

or put the macro inside the `\makeatletter` – `\makeatother` sequence.

6. Numeric functions

By default `PSTricks` loads the file `pst-fp` which is derived from the `fp` package. It supports the following macros:

6.1. \pstFPadd, \pstFPsub, \pstFPmul, and \pstFPdiv

Multiplication and division:

```
\pstFPadd{result}{number}{number}
\pstFPsub{result}{number}{number}
\pstFPmul{result}{number}{number}
\pstFPdiv{result}{number}{number}
```

-0.079847250000000000
-145.202558635394456289
-0.079847250000000000
-0.006886930983847283
7726.059000000000000000
7.094636363636363636
267.123000000000000000
201.123000000000000000
201.123000000000000000
-267.123000000000000000

```
\pstFPmul\Result{-3.405}{0.02345} \Result\quad
\pstFPdiv\Result{-3.405}{0.02345} \Result\\
\pstFPmul\Result{0.02345}{-3.405} \Result\quad
\pstFPdiv\Result{0.02345}{-3.405} \Result\\
\pstFPmul\Result{234.123}{33} \Result\quad
\pstFPdiv\Result{234.123}{33} \Result\\
\pstFPadd\Result{234.123}{33} \Result\quad
\pstFPadd\Result{234.123}{-33} \Result\\
\pstFPsub\Result{234.123}{33} \Result\quad
\pstFPsub\Result{-234.123}{33} \Result
```

The zeros can be stripped with the macro `\pstFPstripZeros`. Expect always rounding errors, TEX was not made for calculations ... The value is converted into a length and then reconverted to a number by stripping the unit. Which also strips the zeros.

-0.07985 -145.20256
-0.07985 -0.00688

```
\pstFPmul\Result{-3.405}{0.02345}
\pstFPstripZeros{\Result}{\Result}\Result\quad
\pstFPdiv\Result{-3.405}{0.02345}
\pstFPstripZeros{\Result}{\Result}\Result\\
\pstFPmul\Result{0.02345}{-3.405}
\pstFPstripZeros{\Result}{\Result}\Result\quad
\pstFPdiv\Result{0.02345}{-3.405}
\pstFPstripZeros{\Result}{\Result}\Result
```

6.2. `\pstFPMul` and `\pstFPDiv`

Integer multiplication and division:

`\pstFPMul{`*result as a truncated integer*`}{`*number*`}{`*number*`}`
`\pstFPDiv{`*result as a truncated integer*`}{`*number*`}{`*number*`}`

-0 -145
-79 -6
7726 7

```
\makeatletter
\pstFPMul\Result{-34.05}{0.02345} \Result\quad
\pstFPDiv\Result{-3.405}{0.02345} \Result\\
\pstFPMul\Result{23.45}{-3.405} \Result\quad
\pstFPDiv\Result{0.2345}{-0.03405} \Result\\
\pstFPMul\Result{234.123}{33} \Result\quad
\pstFPDiv\Result{234.123}{33} \Result
\makeatother
```

7. The PostScript header files

7.1. `pstricks.pro`

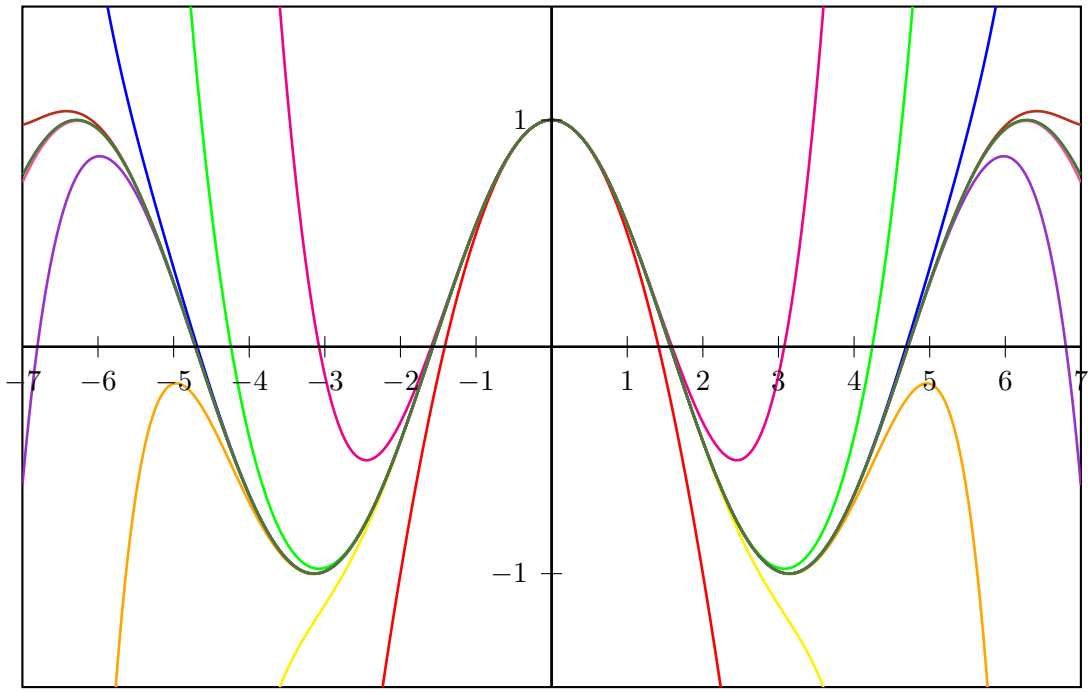
It contains now most of the stuff from `pstricks-add` and the new routines for plotting lines/curves with symbols.

7.2. pst-algparser.pro

Using the Sum function

\Sum(*<index name>,<start>,<step>,<end>,<function>*)

Let's plot the first development of cosine with polynomials: $\sum_{n=0}^{+\infty} \frac{(-1)^n x^{2n}}{n!}$.

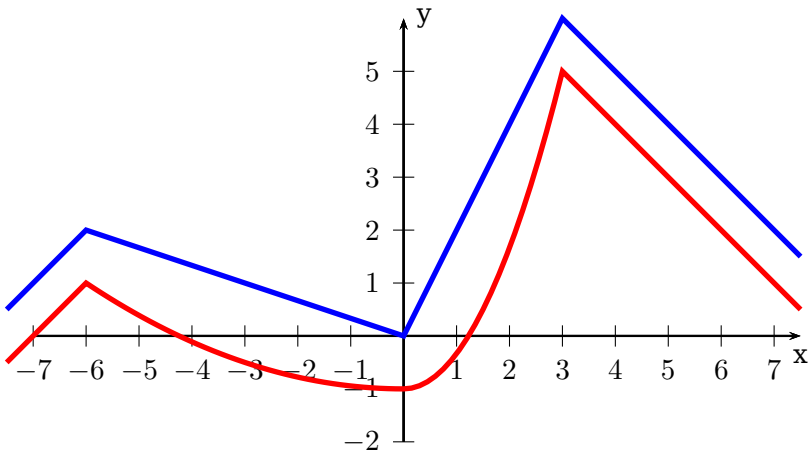



```
\psset{algebraic=true, plotpoints=501, yunit=3}
\def\getColor#1{\ifcase#1 black\or red\or magenta\or yellow\or green\or Orange\or blue\or
 DarkOrchid\or BrickRed\or Rhodamine\or OliveGreen\fi}
\begin{pspicture}(-7,-1.5)(7,1.5)
 \psclip{\psframe(-7,-1.5)(7,1.5)}
  \psplot{-7}{7}{cos(x)}
  \multido{\n=1+1}{10}{%
   \psplot[linewidth=1pt,linecolor=\getColor{\n}]{-7}{7}{%
     Sum(ijk,0,1,\n,(-1)^ijk*x^(2*ijk)/fact(2*ijk))}}
 \endpsclip
 \psaxes(0,0)(-7,-1.5)(7,1.5)
\end{pspicture}
```

7.3. The variable step algorithm together with the PostScript function IfTE

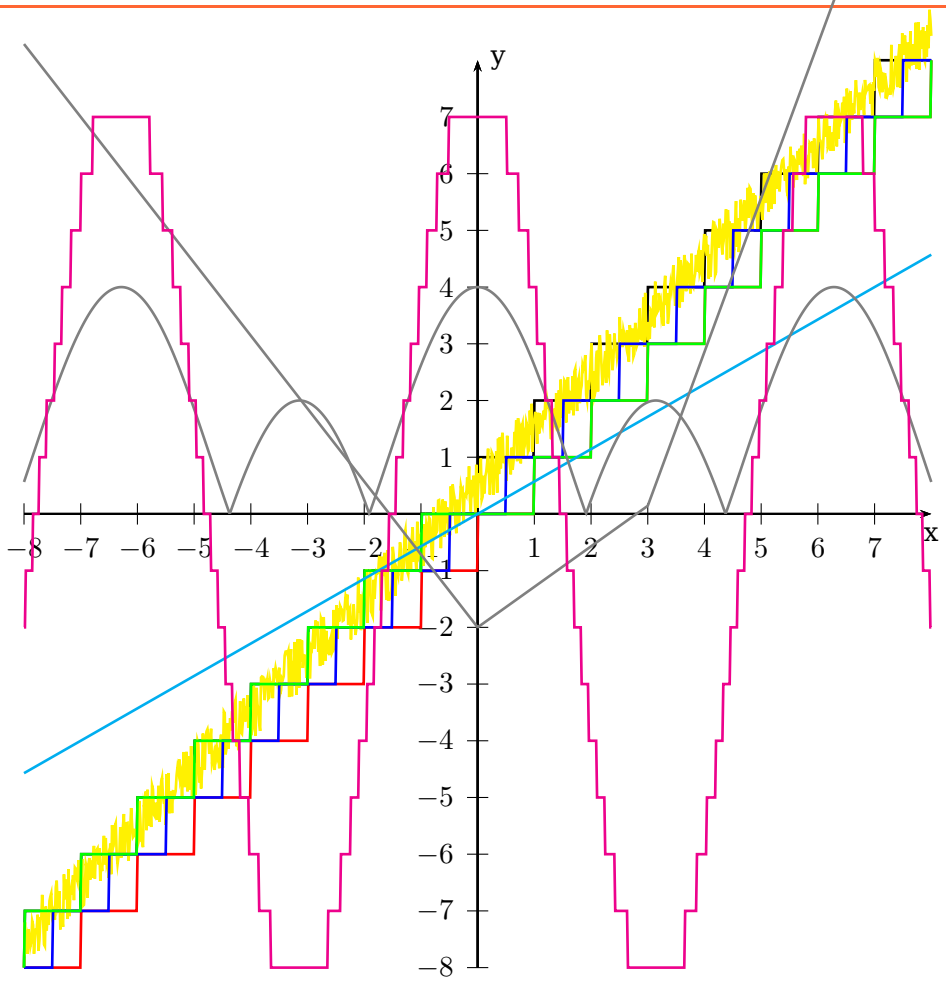
IfTE(<*condition*>,<*true part*>,<*false part*>)

Nesting of several IfTE is possible and seen in the following examples. A classic example is a piece-wise linear function.



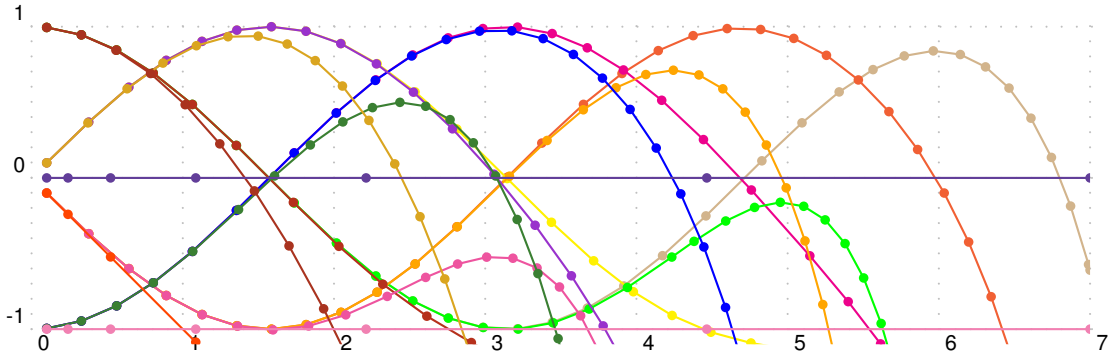

```
\psset{unit=1.5, algebraic, VarStep, showpoints, VarStepEpsilon=.001}
\begin{pspicture}[showgrid=true](-7,-2)(2,4)
  \psplot{-7}{2}{IfTE(x<-5,-(x+5)^3/2,IfTE(x<0,0,x^2))}
  \psplot{-7}{2}{5*x/9+26/9}
  \psplot[linecolor=blue]{-7}{2}{(x+7)^30/9^30*4.5-1/2}
  \psplot[linecolor=red]{-6.9}{2}
     {IfTE(x<-6,ln(x+7),IfTE(x<-3,x+6,IfTE(x<0.1415926,sin(x+3)+3,3.1415926-x)))}
\end{pspicture}
```

When you program a piece-wise defined function you must take care that a plotting point must be put at each point where the description changes. Use showpoints=true to see what's going on when there is a problem. You are on the safe side when you choose a big number for plotpoints.




```
\psset{unit=0.75}
\begin{pspicture}(-8,-8)(8,8)
  \psaxes{->}(0,0)(-8,-8)(8,8)[x,-90][y,0]
  \psset{plotpoints=1000,linewidth=1pt}
  \psplot[algebraic=true, linecolor=yellow]{-8}{8}{rand/(2^31-1)+x}
  \psplot[algebraic=true]{-8}{8}{ceiling(x)}
  \psplot[algebraic=true, linecolor=red]{-8}{8}{floor(x)}
  \psplot[algebraic=true, linecolor=blue]{-8}{8}{round(x)}
  \psplot[algebraic=true, linecolor=green]{-8}{8}{truncate(x)}
  \psplot[algebraic=true, linecolor=cyan]{-8}{8}{div(mul(4,x),7)}
  \psplot[algebraic=true, linecolor=gray]{-8}{8}{abs(x)+abs(x-3)-abs(5-5*x/7)}
  \psplot[algebraic=true, linecolor=gray]{-8}{8}{abs(3*cos(x)+1)}
  \psplot[algebraic=true, linecolor=magenta]{-8}{8}{floor(8*cos(x))}
\end{pspicture}
```

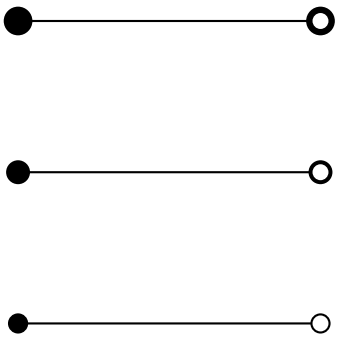
7.4. Successive derivatives of a polynomial with the PostScript function Derive



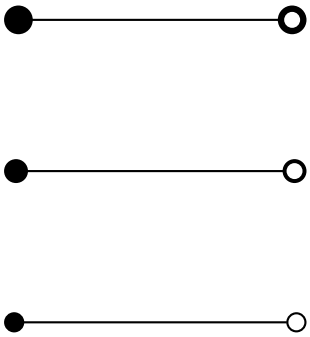

```
\psset{unit=2, algebraic=true, VarStep=true, showpoints=true, VarStepEpsilon=.001}
\def\getColor#1{\ifcase#1 Tan\or RedOrange\or magenta\or yellow\or green\or Orange\or blue
    \or
  DarkOrchid\or BrickRed\or Rhodamine\or OliveGreen\or Goldenrod\or Mahogany\or
  OrangeRed\or CarnationPink\or RoyalPurple\or 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[algebraic=true, linecolor=\getColor{\in}]{0.1}{7}
    {Derive(\in,Sum(i,0,1,7,(-1)^i*x^(2*i)/Fact(2*i)))}}
  \endpsclip
\end{pspicture}
```

7.5. Special arrow option arrowLW

Only for the arrowtype o, oo, *, and ** it is possible to set the arrowlinewidth with the optional keyword arrowLW. When scaling an arrow by the keyword arrowscale the width of the borderline is also scaled. With the optional argument arrowLW the line width can be set separately and is not taken into account by the scaling value.



```
\begin{pspicture}(4,6)
\psline[arrowscale=3,arrows=*-o](0,5)(4,5)
\psline[arrowscale=3,arrows=*-o,
  arrowLW=0.5pt](0,3)(4,3)
\psline[arrowscale=3,arrows=*-o,
  arrowLW=0.3333\pslinewidth](0,1)(4,1)
\end{pspicture}
```



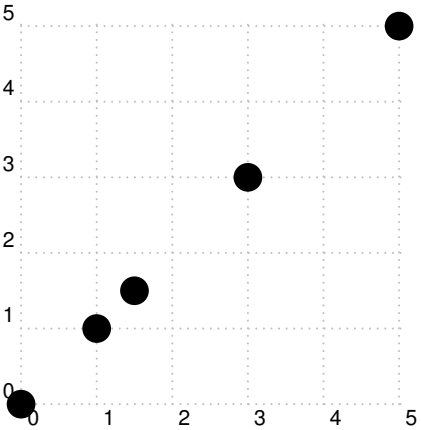
```
\begin{pspicture}(4,6)
\psline[arrowscale=3,arrows=**-oo](0,5)(4,5)
\psline[arrowscale=3,arrows=**-oo,
  arrowLW=0.5pt](0,3)(4,3)
\psline[arrowscale=3,arrows=**-oo,
  arrowLW=0.3333\pslinewidth](0,1)(4,1)
\end{pspicture}
```

8. \psforeach and \psForeach

The macro \psforeach allows a loop with an individual increment.

```
\psforeach{variable}{value list}{action}
\psForeach{variable}{value list}{action}
```

With \psforeach the *action* is done inside a group and for \psForeach not. This maybe useful when using the macro to create tabular cells, which are alread grouped itself.



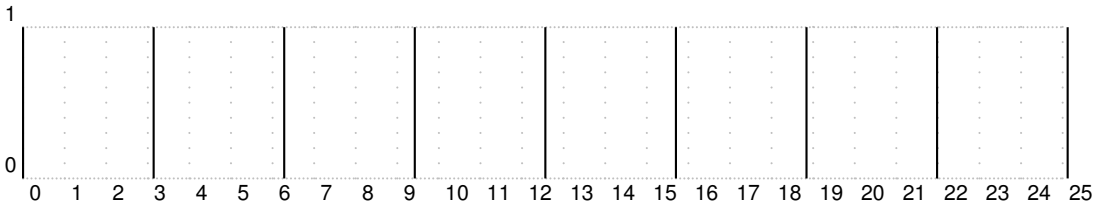

```
\begin{pspicture}[showgrid=true](5,5)
 \psforeach{\nA}{0, 1, 1.5, 3, 5}{%
   \psdot[dotscale=3](\nA,\nA)}
\end{pspicture}
```

$y = 2^x$

2	4	6	8	10	12
4.0	16.0	64.0	256.0	1024.0	4096.0

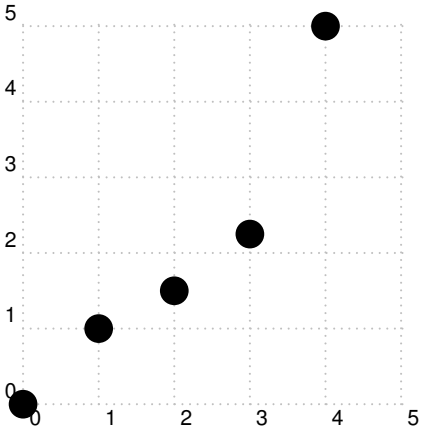
```
%\usepackage{pst-func}
\makeatletter
\newcommand*\InitToks{\toks@={}}
\newcommand\AddToks[1]{\toks@=\expandafter{\the\toks@ #1}}
\newcommand*\PrintToks{\the\toks@}
\newcommand*{\makeTable}[4][5mm]{%
  \begingroup
    \InitToks%
    \AddToks{\begin{tabular}{|*{#2}{>{\RaggedLeft}p{#1}|}@{}l@{}}\cline{1-#2}}
    \psForeach{\iA}{#3}{\expandafter\AddToks\expandafter{\iA & }}
    \AddToks{\\\cline{1-#2}}%
    \psForeach{\iA}{#3}{\expandafter\AddToks\expandafter{\expandafter%
      \psPrintValue\expandafter{\iA\space /x ED #4} & }}
    \AddToks{\\\cline{1-#2}\end{tabular}}%
    \PrintToks
  \endgroup
}
\makeatother

\sffamily
\psset{decimals=2,valuewidth=7,xShift=-20}
$y=2^x$\\
\makeTable[1cm]{6}{2,4,6,8,10,12}{2 x exp}
```

The value List can also be given by the first two and the last value, e.g. `1,4,..,31`, then PSTrickscalculates all values with the distance given by the first two values.




```
\psset{xunit=0.55cm,yunit=2cm}
\begin{pspicture}[showgrid](0,-5mm)(25,1)
  \psforeach{\nA}{0, 3.14,..,25}{\psline(\nA,0)(\nA,1)}
\end{pspicture}
```

The internal counter for the steps is named `psLoopIndex` and can be used for own purposes.




```
\begin{pspicture}[showgrid=true](5,5)
\psforeach{\nA}{0, 1, 1.5, 2.25, 5}{%
  \psdot[dotscale=3](\the\psLoopIndex,\nA)}
\end{pspicture}
```

Part II.
pst-node – package

9. pst-node.tex

The package `pst-node` now uses the advanced key handling from `xkeyval`. The reason why it moved from the base into the contrib sections, where all packages uses `xkeyval`.

Part III.
pst-plot – package

10. pst-plot.tex

The package `pst-plot` now uses the advanced key handling from `xkeyval`. The reason why it moved from the base into the contrib sections, where all packages uses `xkeyval`.

11. List of all optional arguments for pstricks

Key	Type	Default

References

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

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

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

[4] Herbert Voß. “Die mathematischen Funktionen von Postscript”. In: *Die TeXnische Komödie* 1/02 (Mar. 2002), pp. 40–47.

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

[6] Herbert Voß. *PSTricks – Graphics for LaTeX*. 1st ed. Cambridge/UK: UIT, 2011.

[7] Herbert Voß. *PSTricks Support for pdf*. 2002. URL: http://PSTricks.tug.org/main.cgi?file=pdf/pdfoutput.

[8] Herbert Voß. *LaTeX Quick Reference*. 1st ed. Cambridge/UK: UIT, 2011.

[9] Herbert Voß. *LaTeX Referenz*. 3rd ed. Heidelberg and Berlin: DANTE – lehmanns media, 2014.

[10] Michael Wiedmann and Peter Karp. *References for TeX and Friends*. 2003. URL: http://www.miwie.org/tex-refs/.

Index