pst-func
plotting special mathematical functions

v.0.45

Herbert Voß*

April 22, 2006

Contents

1 \psPolynomial **2**

2 \psFourier **8**

3 \psBessel **11**

4 \psGauss **and** \psGaussI **14**

5 \psSi**,** \pssi **and** \psCi **16**

6 \psIntegral**,** \psCumIntegral **and** \psConv **18**

7 \psBinomial **and** \psBinomialN **21**

8 \psplotImp **– plotting implicit defined functions** **27**

9 \psPrintValue **31**

10 Credits **32**

*Thanks to: Attila Gati, John Frampton and Lars Kotthoff, Jose-Emilio Vila-Forcen.

1 \psPolynomial

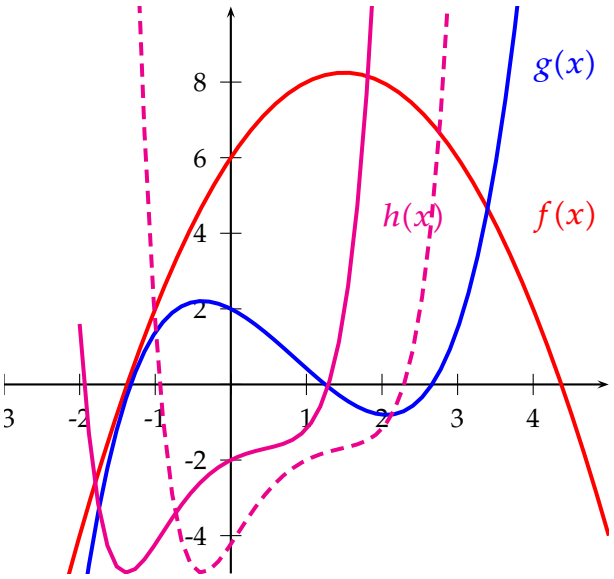
The polynomial function is defined as

$$f(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + \ldots + a_{n-1}x^{n-1} + a_nx^n \tag{1}$$
$$f'(x) = a_1 + 2a_2x + 3a_3x^2 + \ldots + (n-1)a_{n-1}x^{n-2} + na_nx^{n-1} \tag{2}$$
$$f''(x) = 2a_2 + 6a_3x + \ldots + (n-1)(n-2)a_{n-1}x^{n-3} + n(n-1)a_nx^{n-2} \tag{3}$$

so `pst-func` needs only the coefficients of the polynomial to calculate the function. The syntax is

`\psPolynomial[<options>]{xStart}{xEnd}`

With the option `xShift` one can do a horizontal shift to the graph of the function. With another than the predefined value the macro replaces x by $x - x\text{Shift}$; `xShift=1` moves the graph of the polynomial function one unit to the right.




```
\psset{yunit=0.5cm,xunit=1cm}
\begin{pspicture*}(-3,-5)(5,10)
  \psaxes[Dy=2]{->}(0,0)(-3,-5)(5,10)
  \psset{linewidth=1.5pt}
  \psPolynomial[coeff=6 3 -1,linecolor=red]{-3}{5}
  \psPolynomial[coeff=2 -1 -1 .5 -.1 .025,linecolor=
    blue]{-2}{4}
```

```
  \psPolynomial[coeff=-2 1 -1 .5 .1 .025 .2 ,linecolor=
    magenta]{-2}{4}
  \psPolynomial[coeff=-2 1 -1 .5 .1 .025 .2 ,linecolor=
    magenta,xShift=1,linestyle=dashed]{-2}{4}
  \rput[lb](4,4){\textcolor{red}{$f(x)$}}
  \rput[lb](4,8){\textcolor{blue}{$g(x)$}}
  \rput[lb](2,4){\textcolor{magenta}{$h(x)$}}
\end{pspicture*}
```

The plot is easily clipped using the star version of the `pspicture` environment, so that points whose coordinates are outside of the desired range are not plotted. The plotted polynomials are:

$$f(x) = 6 + 3x - x^2 \tag{4}$$

$$g(x) = 2 - x - x^2 + 0.5x^3 - 0.1x^4 + 0.025x^5 \tag{5}$$

$$h(x) = -2 + x - x^2 + 0.5x^3 + 0.1x^4 + 0.025x^5 + 0.2x^6 \tag{6}$$

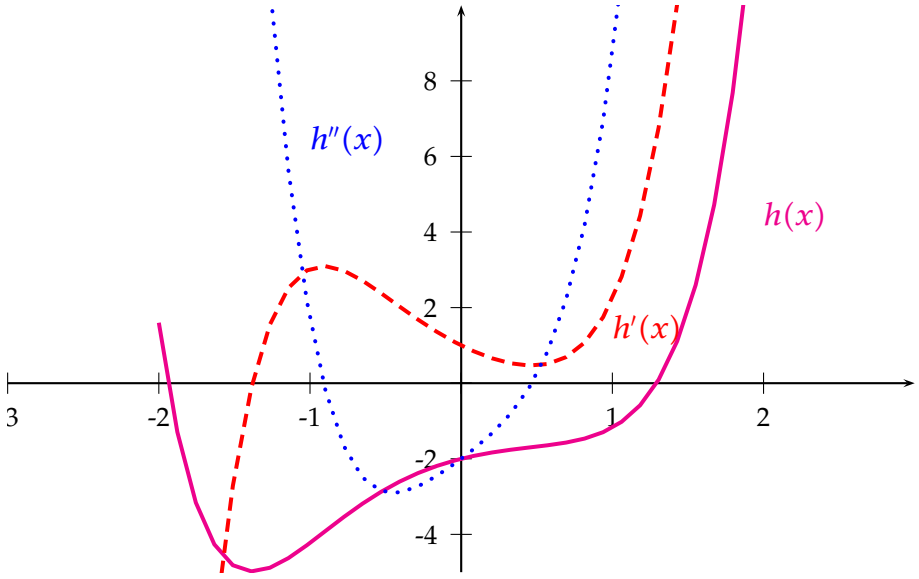
$$\begin{aligned} h^*(x) = {} & -2 + (x-1) - (x-1)^2 + 0.5(x-1)^3 + \\ & + 0.1(x-1)^4 + 0.025(x-1)^5 + 0.2(x-1)^6 \end{aligned} \tag{7}$$

There are the following new options:

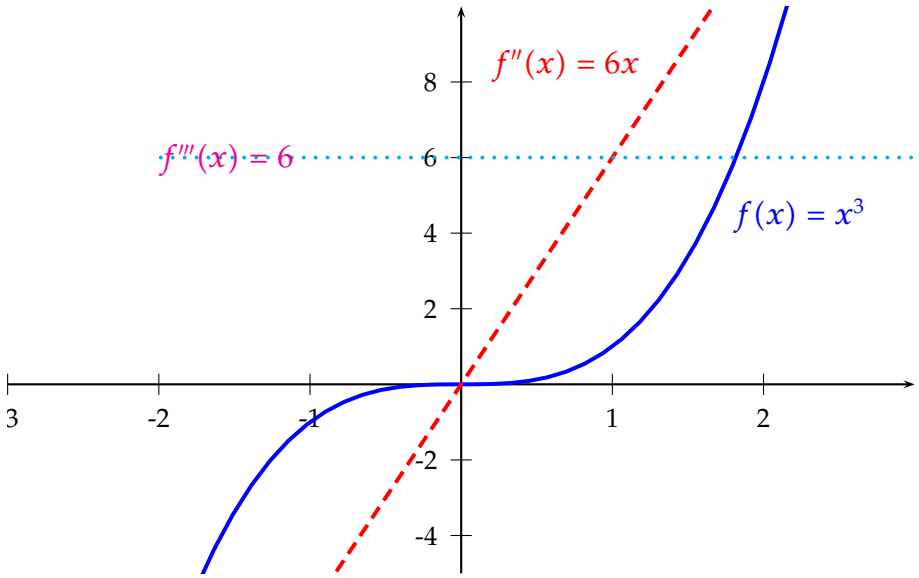
Name	Value	Default	
`coeff`	`a0 a1 a2 ...`	`0 0 1`	The coefficients must have the order a_0 a_1 $a_2 \ldots$ and be separated by **spaces**. The number of coefficients is limited only by the memory of the computer ... The default value of the parameter `coeff` is `0 0 1`, which gives the parabola $y = a_0 + a_1 x + a_2 x^2 = x^2$.
`xShift`	`<number>`	`0`	$(x - xShift)$ for the horizontal shift of the polynomial
`Derivation`	`<number>`	`0`	the default is the function itself
`markZeros`	`false|true`	`false`	dotstyle can be changed
`epsZero`	`<value>`	`0.1`	The distance between two zeros, important for the iteration function to test, if the zero value still exists
`dZero`	`<value>`	`0.1`	When searching for all zero values, the function is scanned with this step
`zeroLineTo`	`<number>`	`false`	plots a line from the zero point to the value of the zeroLineTo's Derivation of the polynomial function

Name	Value	Default	
zeroLineStyle	<line style>	dashed	the style is one of the for PSTricks valid styles.
zeroLineColor	<color>	black	any valid xolor is possible
zeroLineWidth	<width>	0.5\pslinewidth	

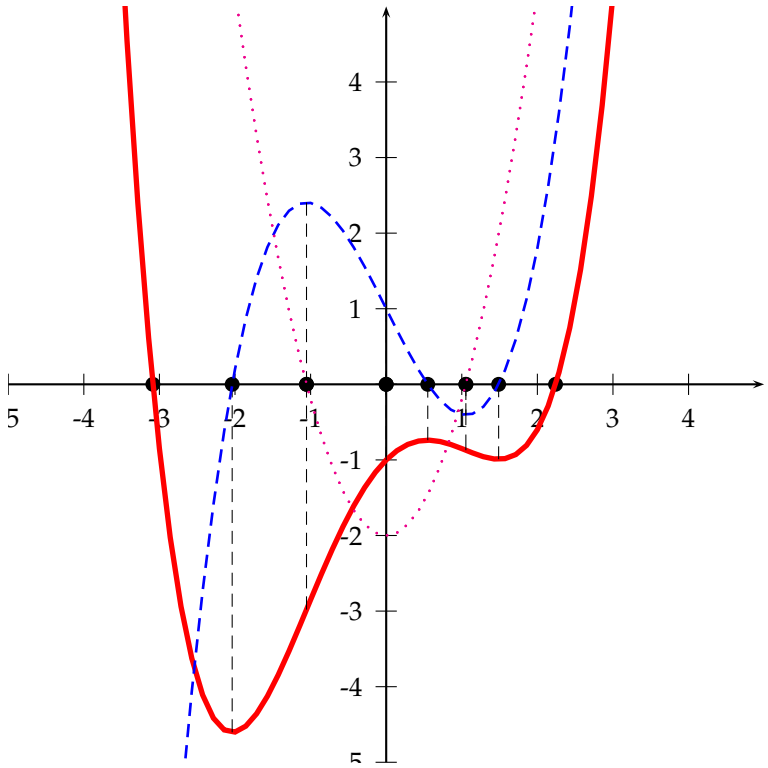
The above parameter are only valid for the \psPolynomial macro, except x0, which can also be used for the Gauss function. All options can be set in the usual way with \psset.



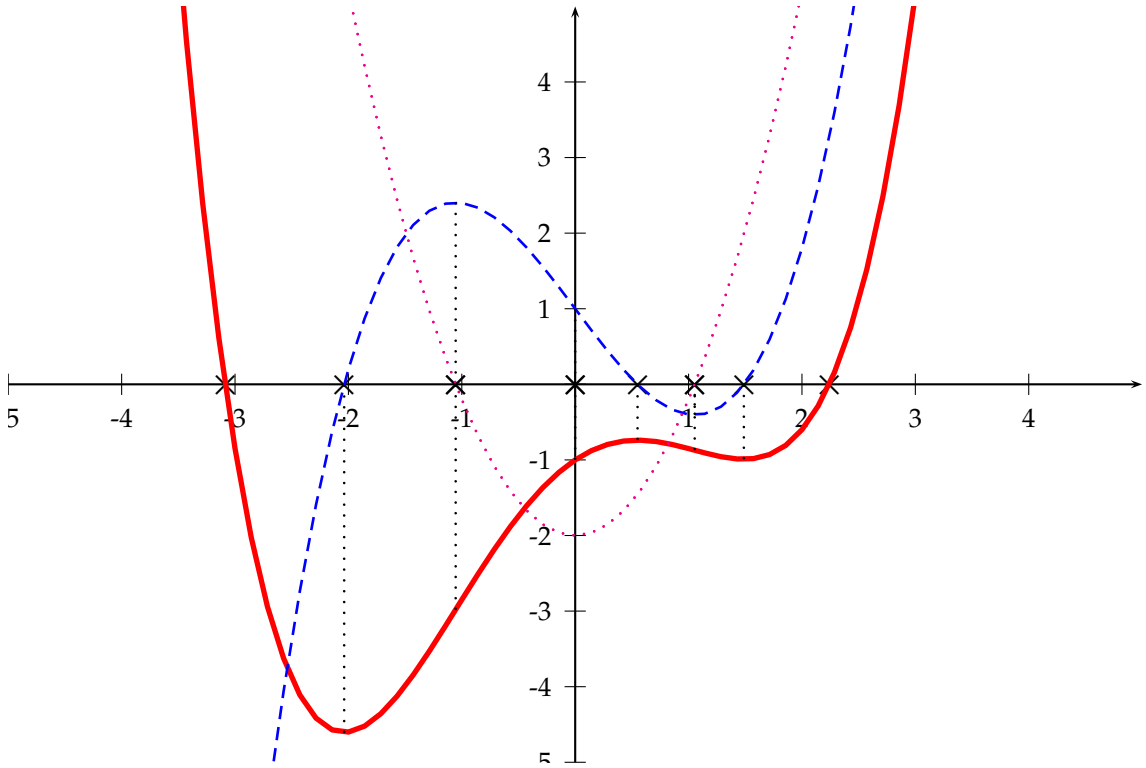

```
\psset{yunit=0.5cm,xunit=2cm}
\begin{pspicture*}(-3,-5)(3,10)
  \psaxes[Dy=2]{->}(0,0)(-3,-5)(3,10)
  \psset{linewidth=1.5pt}
  \psPolynomial[coeff=-2 1 -1 .5 .1 .025 .2 ,linecolor=magenta]{-2}{4}
  \psPolynomial[coeff=-2 1 -1 .5 .1 .025 .2 ,linecolor=red,%
    linestyle=dashed,Derivation=1]{-2}{4}
  \psPolynomial[coeff=-2 1 -1 .5 .1 .025 .2 ,linecolor=blue,%
    linestyle=dotted,Derivation=2]{-2}{4}
  \rput[lb](2,4){\textcolor{magenta}{$h(x)$}}
  \rput[lb](1,1){\textcolor{red}{$h^{\prime}(x)$}}
  \rput[lb](-1,6){\textcolor{blue}{$h^{\prime\prime}(x)$}}
\end{pspicture*}
```




```
\psset{yunit=0.5cm,xunit=2cm}
\begin{pspicture*}(-3,-5)(3,10)
  \psaxes[Dy=2]{->}(0,0)(-3,-5)(3,10)
  \psset{linewidth=1.5pt}
  \psPolynomial[coeff=0 0 0 1,linecolor=blue]{-2}{4}
  \psPolynomial[coeff=0 0 0 1,linecolor=red,%
    linestyle=dashed,Derivation=2]{-2}{4}
  \psPolynomial[coeff=0 0 0 1,linecolor=cyan,%
    linestyle=dotted,Derivation=3]{-2}{4}
  \rput[lb](1.8,4){\textcolor{blue}{$f(x)=x^3$}}
  \rput[lb](0.2,8){\textcolor{red}{$f^{\prime\prime}(x)=6x$}}
  \rput[lb](-2,5.5){\textcolor{magenta}{$f^{\prime\prime\prime}(x)=6$}}
\end{pspicture*}
```




```
\begin{pspicture*}(-5,-5)(5,5)
  \psaxes{->}(0,0)(-5,-5)(5,5)%
  \psset{dotscale=2}
  \psPolynomial[markZeros,linecolor=red,linewidth=2pt,coeff=-1 1 -1 0
    0.15]{-4}{3}%
  \psPolynomial[markZeros,linecolor=blue,linewidth=1pt,linestyle=dashed,%
    coeff=-1 1 -1 0 0.15,Derivation=1,zeroLineTo=0]{-4}{3}%
  \psPolynomial[markZeros,linecolor=magenta,linewidth=1pt,linestyle=dotted
    ,%
    coeff=-1 1 -1 0 0.15,Derivation=2,zeroLineTo=0]{-4}{3}%
  \psPolynomial[markZeros,linecolor=magenta,linewidth=1pt,linestyle=dotted
    ,%
    coeff=-1 1 -1 0 0.15,Derivation=2,zeroLineTo=1]{-4}{3}%
\end{pspicture*}
```




```
\psset{xunit=1.5}
\begin{pspicture*}(-5,-5)(5,5)
  \psaxes{->}(0,0)(-5,-5)(5,5)%
  \psset{dotscale=2,dotstyle=x,zeroLineStyle=dotted,zeroLineWidth=1pt}
  \psPolynomial[markZeros,linecolor=red,linewidth=2pt,coeff=-1 1 -1 0
    0.15]{-4}{3}%
  \psPolynomial[markZeros,linecolor=blue,linewidth=1pt,linestyle=dashed,%
    coeff=-1 1 -1 0 0.15,Derivation=1,zeroLineTo=0]{-4}{3}%
  \psPolynomial[markZeros,linecolor=magenta,linewidth=1pt,linestyle=dotted
    ,%
    coeff=-1 1 -1 0 0.15,Derivation=2,zeroLineTo=0]{-4}{3}%
  \psPolynomial[markZeros,linecolor=magenta,linewidth=1pt,linestyle=dotted
    ,%
    coeff=-1 1 -1 0 0.15,Derivation=2,zeroLineTo=1]{-4}{3}%
\end{pspicture*}
```

2 \psFourier

A Fourier sum has the form:

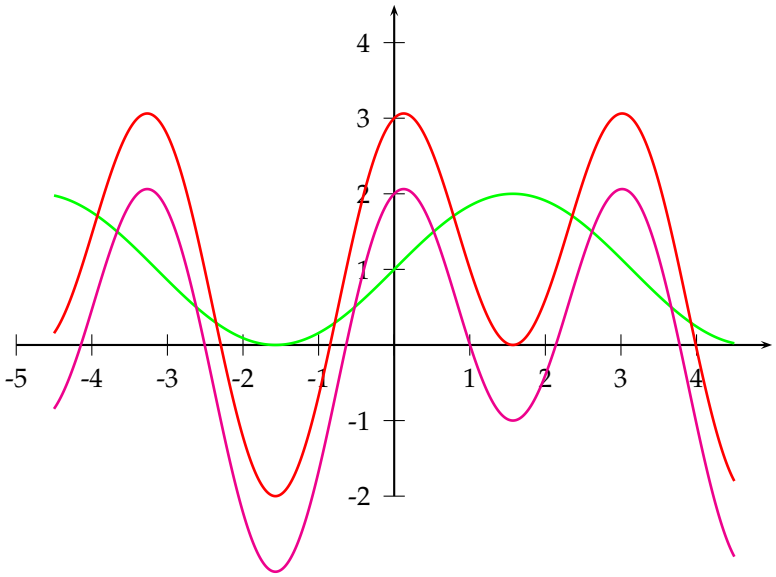
$$s(x) = \frac{a_0}{2} + a_1 \cos \omega x + a_2 \cos 2\omega x + a_3 \cos 3\omega x + \ldots + a_n \cos n\omega x \quad (8)$$

$$+ b_1 \sin \omega x + b_2 \sin 2\omega x + b_3 \sin 3\omega x + \ldots + b_m \sin m\omega x \quad (9)$$

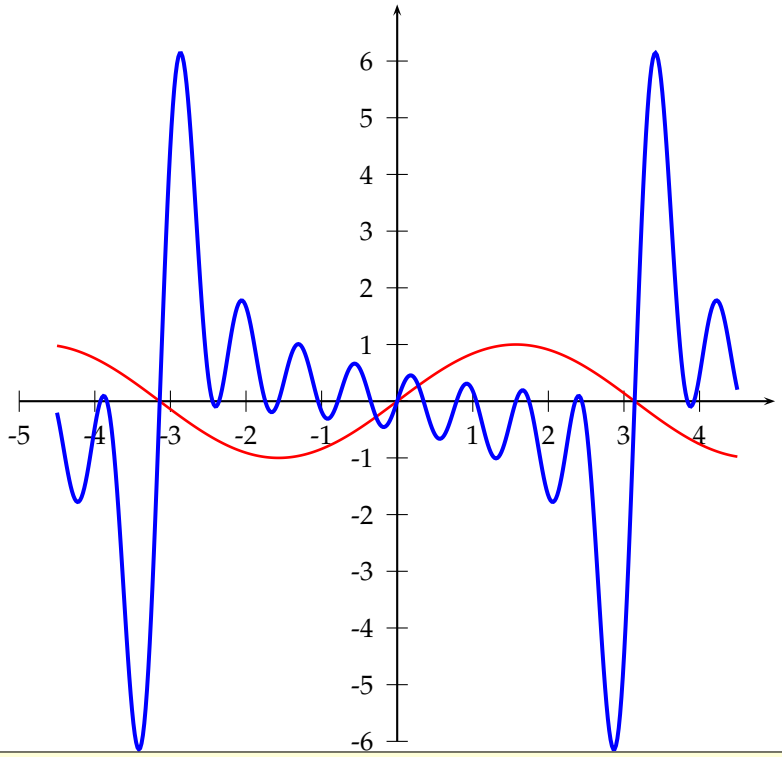
The macro `psFourier` plots Fourier sums. The syntax is similiar to `psPolynomial`, except that there are two kinds of coefficients:

```
\psPolynomial[cosCoeff=a0 a1 a2 ..., sinCoeff=b1 b2 ...]{xStart}{xEnd}
```

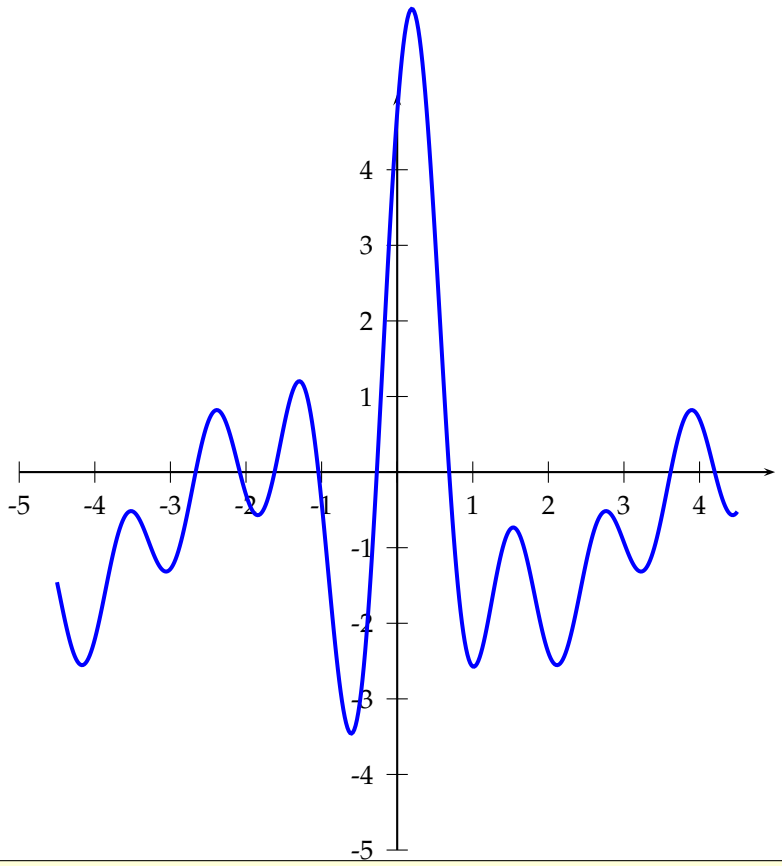
The coefficients must have the orders a_0 a_1 a_2 ... and b_1 b_2 b_3 ... and be separated by **spaces**. The default is `cosCoeff=0,sinCoeff=1`, which gives the standard `sin` function. Note that the constant value can only be set with `cosCoeff=a0`.




```
\begin{pspicture}(-5,-3)(5,5.5)
\psaxes{->}(0,0)(-5,-2)(5,4.5)
\psset{plotpoints=500,linewidth=1pt}
\psFourier[cosCoeff=2, linecolor=green]{-4.5}{4.5}
\psFourier[cosCoeff=0 0 2, linecolor=magenta]{-4.5}{4.5}
\psFourier[cosCoeff=2 0 2, linecolor=red]{-4.5}{4.5}
\end{pspicture}
```




```
\psset{yunit=0.75}
\begin{pspicture}(-5,-6)(5,7)
\psaxes{->}(0,0)(-5,-6)(5,7)
\psset{plotpoints=500}
\psFourier[linecolor=red,linewidth=1pt]{-4.5}{4.5}
\psFourier[sinCoeff= -1 1 -1 1 -1 1 -1 1,%
   linecolor=blue,linewidth=1.5pt]{-4.5}{4.5}
\end{pspicture}
```




```
\begin{pspicture}(-5,-5)(5,5.5)
\psaxes{->}(0,0)(-5,-5)(5,5)
\psset{plotpoints=500,linewidth=1.5pt}
\psFourier[sinCoeff=-.5 1 1 1 1 ,cosCoeff=-.5 1 1 1 1 1,%
    linecolor=blue]{-4.5}{4.5}
\end{pspicture}
```

3 \psBessel

The Bessel function of order n is defined as

$$J_n(x) = \frac{1}{\pi}\int_0^{\pi} \cos(x\sin t - nt)\,\mathrm{d}t \tag{10}$$

$$= \sum_{k=0}^{\infty} \frac{(-1)^k \left(\frac{x}{2}\right)^{n+2k}}{k!\Gamma(n+k+1)} \tag{11}$$

The syntax of the macro is

```
\psBessel[options]{order}{xStart}{xEnd}
```

There are two special parameters for the Bessel function, and also the settings of many `pst-plot` or `pstricks` parameters affect the plot.

```
\def\psset@constI#1{\edef\psk@constI{#1}}
\def\psset@constII#1{\edef\psk@constII{#1}}
\psset{constI=1,constII=0}
```

These two "'constants"' have the following meaning:

$$f(t) = constI \cdot J_n + constII$$

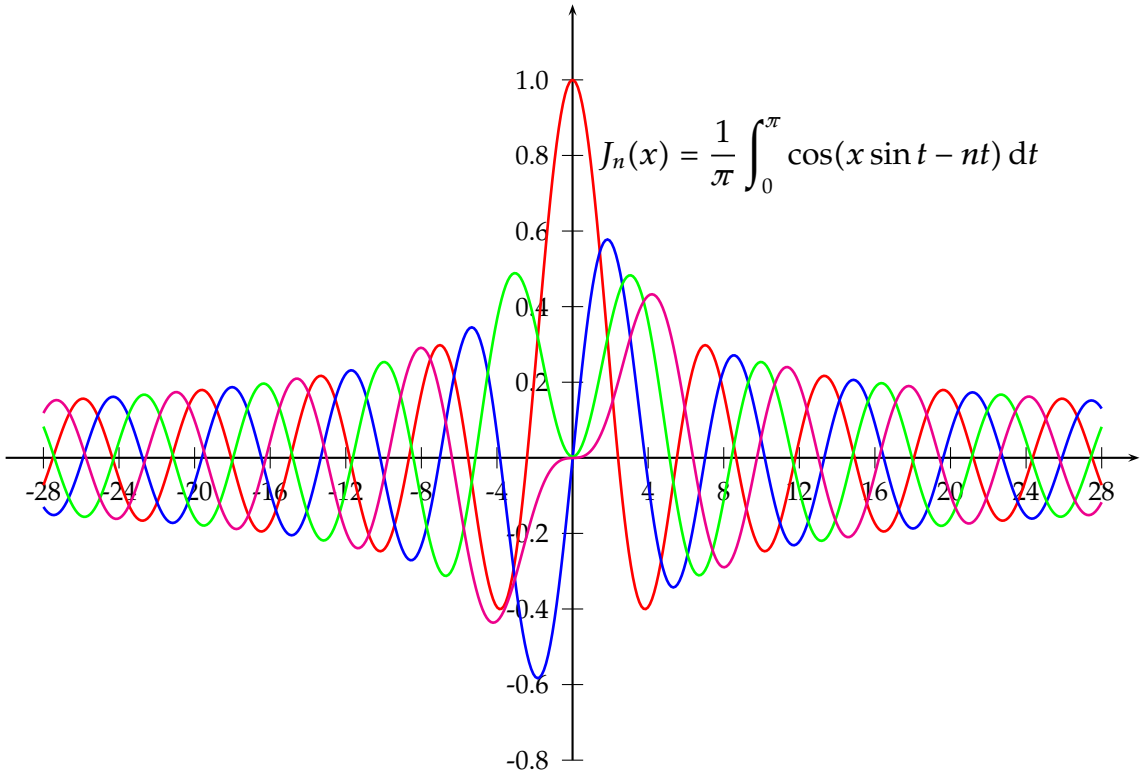
where *constI* and *constII* must be real PostScript expressions, e.g.:

```
\psset{constI=2.3,constII=t k sin 1.2 mul 0.37 add}
```

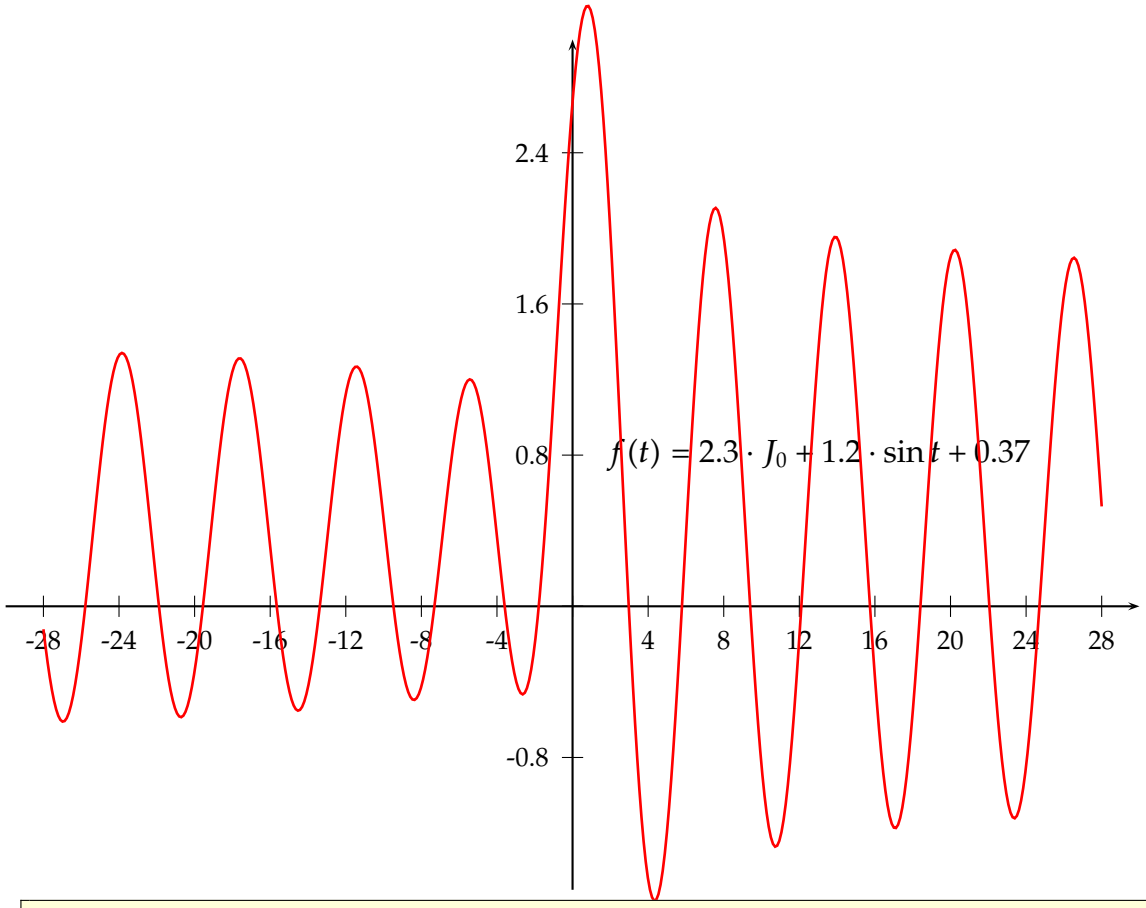
The Bessel function is plotted with the parametricplot macro, this is the reason why the variable is named `t`. The internal procedure `k` converts the value t from radian into degrees. The above setting is the same as

$$f(t) = 2.3 \cdot J_n + 1.2 \cdot \sin t + 0.37$$

In particular, note that the default for `plotpoints` is 500. If the plotting computations are too time consuming at this setting, it can be decreased in the usual way, at the cost of some reduction in graphics resolution.




```
{
\psset{xunit=0.25,yunit=5}
\begin{pspicture}(-13,-.85)(13,1.25)
\rput(13,0.8){%
    $\displaystyle J_n(x)=\frac{1}{\pi}\int_0^\pi\cos(x\sin t-nt)\dt$%
}
\psaxes[Dy=0.2,Dx=4]{->}(0,0)(-30,-.8)(30,1.2)
\psset{linewidth=1pt}
\psBessel[linecolor=red]{0}{-28}{28}%
\psBessel[linecolor=blue]{1}{-28}{28}%
\psBessel[linecolor=green]{2}{-28}{28}%
\psBessel[linecolor=magenta]{3}{-28}{28}%
\end{pspicture}
}
```




```
{
\psset{xunit=0.25,yunit=2.5}
\begin{pspicture}(-13,-1.5)(13,3)
\rput(13,0.8){%
    $\displaystyle f(t) = 2.3 \cdot J_0 + 1.2\cdot \sin t + 0.37$%
}
\psaxes[Dy=0.8,dy=2cm,Dx=4]{->}(0,0)(-30,-1.5)(30,3)
\psset{linewidth=1pt}
\psBessel[linecolor=red,constI=2.3,constII={t k sin 1.2 mul 0.37 add
  }]{0}{-28}{28}%
\end{pspicture}
}
```

4 \psGauss and \psGaussI

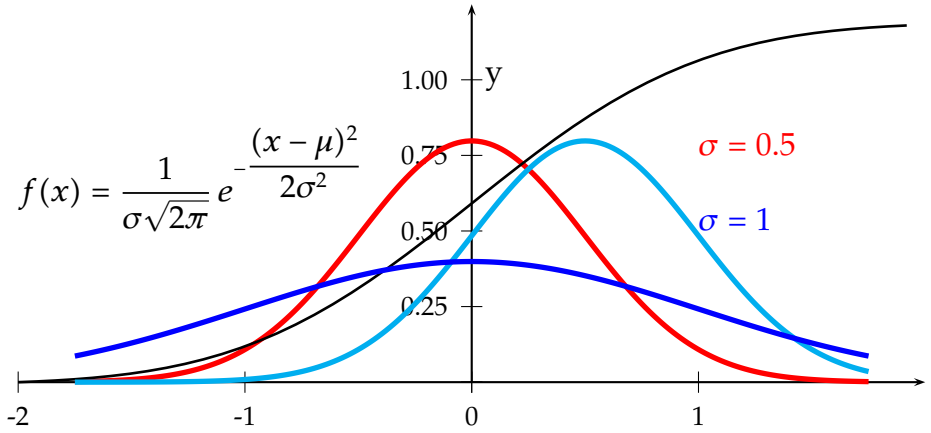
The Gauss function is defined as

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}}\, e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{12}$$

The syntax of the macros is

```
\psGauss[options]{xStart}{xEnd}
\psGaussI[options]{xStart}{xEnd}
```

where the only new parameter are `sigma=<value>` and `mue=<value>` for the horizontal shift, which can also be set in the usual way with `\psset`. It is significant only for the `psGauss`- and `\psGaussI`-macro. The default is `sigma=0.5` and `mue=0`. The integral is caclulated wuth the Simson algorithm and has one special option, called `Simpson`, which defines the number of intervalls per step and is predefined with 5.




```
\psset{yunit=4cm,xunit=3}
\begin{pspicture}(-2,-0.5)(2,1.25)
%  \psgrid[griddots=10,gridlabels=0pt, subgriddiv=0]
  \psaxes[Dy=0.25]{->}(0,0)(-2,0)(2,1.25)
  \uput[-90](6,0){x}\uput[0](0,1){y}
  \rput[lb](1,0.75){\textcolor{red}{$\sigma =0.5$}}
  \rput[lb](1,0.5){\textcolor{blue}{$\sigma =1$}}
  \rput[lb](-2,0.5){$f(x)=\dfrac{1}{\sigma\sqrt{2\pi}}\,e
    ^{-\dfrac{(x-x_0)^2}{2\sigma{}^2}}$}
  \psGauss[linecolor=red, linewidth=2pt]{-1.75}{1.75}%
  \psGaussI[linewidth=1pt,yunit=0.75cm]{-2}{2}%
```

```
11    \psGauss[linecolor=cyan, mue=0.5, linewidth=2pt
      ]{-1.75}{1.75}%
12    \psGauss[sigma=1, linecolor=blue, linewidth=2pt
      ]{-1.75}{1.75}
13  \end{pspicture}
```

5 \psSi, \pssi and \psCi

The integral sin and cosin are defined as

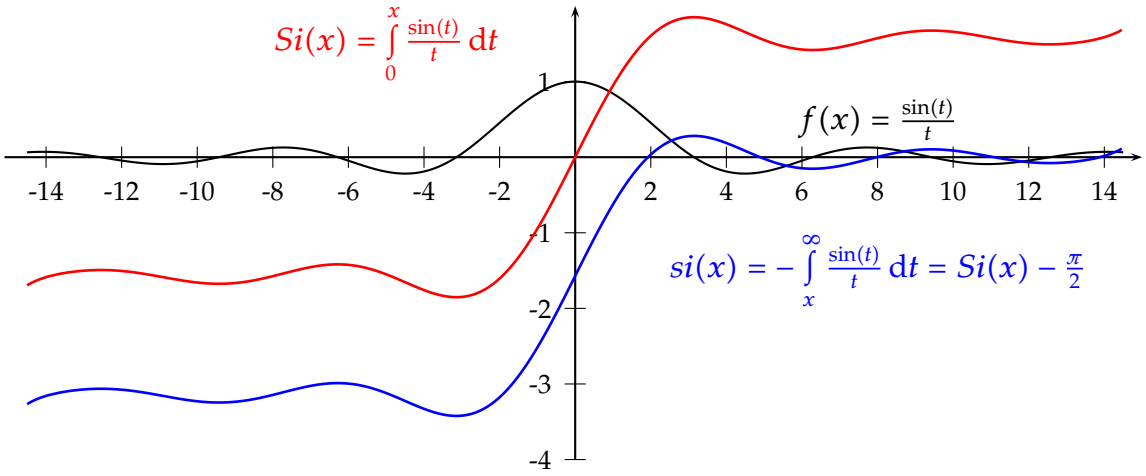
$$\mathrm{Si}(x) = \int_0^x \frac{\sin t}{t}\,\mathrm{d}t \tag{13}$$

$$\mathrm{si}(x) = -\int_x^\infty \frac{\sin t}{t}\,\mathrm{d}t = \mathrm{Si}(x) - \frac{\pi}{2} \tag{14}$$

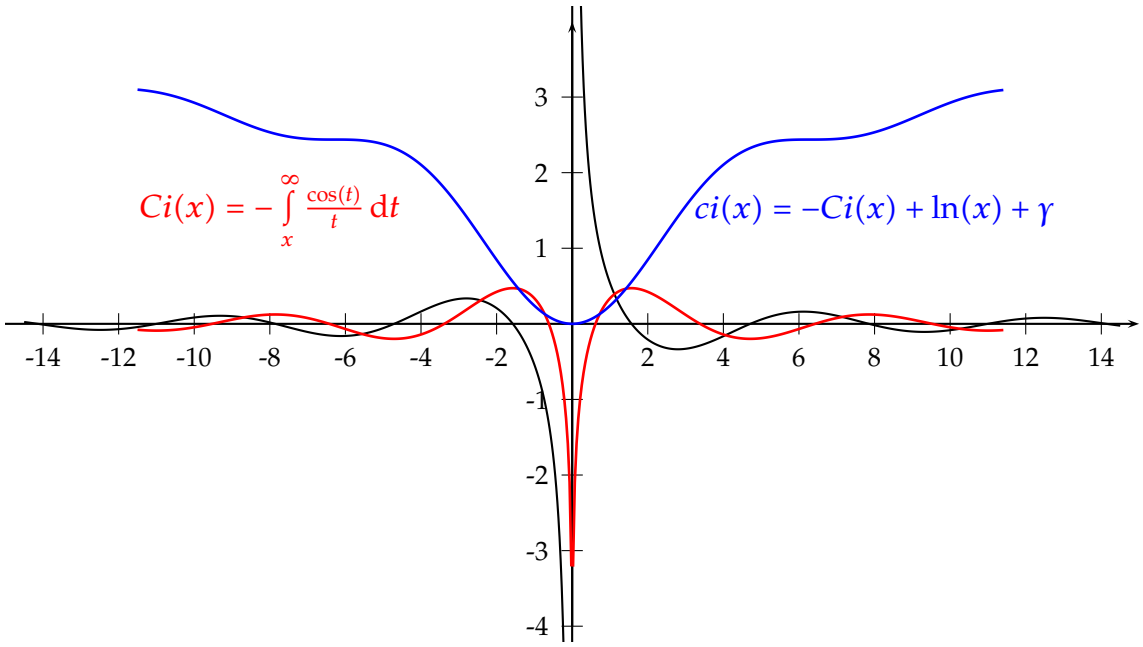
$$\mathrm{Ci}(x) = -\int_x^\infty \frac{\cos t}{t}\,\mathrm{d}t = \gamma + \ln x + \int_0^x \frac{\cos t - 1}{t}\,\mathrm{d}t \tag{15}$$

The syntax of the macros is

```
\psSi[options]{xStart}{xEnd}
\pssi[options]{xStart}{xEnd}
\psCi[options]{xStart}{xEnd}
```




```
\def\pshlabel#1{\footnotesize#1} \def\psvlabel#1{\footnotesize#1}
\psset{xunit=0.5}
\begin{pspicture}(-15,-4.5)(15,2)
  \psaxes[dx=1cm,Dx=2]{->}(0,0)(-15.1,-4)(15,2)
  \psplot[plotpoints=1000]{-14.5}{14.5}{ x RadtoDeg sin x div }
  \psSi[plotpoints=1500,linecolor=red,linewidth=1pt]{-14.5}{14.5}
  \pssi[plotpoints=1500,linecolor=blue,linewidth=1pt]{-14.5}{14.5}
  \rput(-5,1.5){\color{red}$Si(x)=\int\limits_{0}^x \frac{\sin(t)}{t}\dt$}
  \rput(8,-1.5){\color{blue}$si(x)=-\int\limits_{x}^{\infty} \frac{\sin(t)
    }{t}\dt=Si(x)-\frac{\pi}{2}$}
  \rput(8,.5){$f(x)= \frac{\sin(t)}{t}$}
\end{pspicture}
```




```
\def\pshlabel#1{\footnotesize#1} \def\psvlabel#1{\footnotesize#1}
\psset{xunit=0.5}
\begin{pspicture*}(-15,-4.2)(15,4.2)
  \psaxes[dx=1cm,Dx=2]{->}(0,0)(-15.1,-4)(15,4)
  \psplot[plotpoints=1000]{-14.5}{14.5}{ x RadtoDeg cos x Div }
  \psCi[plotpoints=500,linecolor=red,linewidth=1pt]{-11.5}{11.5}
  \psci[plotpoints=500,linecolor=blue,linewidth=1pt]{-11.5}{11.5}
  \rput(-8,1.5){\color{red}$Ci(x)=-\int\limits_{x}^{\infty} \frac{\cos(t)}{
    t}\dt$}
  \rput(8,1.5){\color{blue}$ci(x)=-Ci(x)+\ln(x)+\gamma$}
\end{pspicture*}
```

6 \psIntegral, \psCumIntegral and \psConv

These new macros[1] allows to plot the result of an integral using the Simpson numerical integration rule. The first one is the result of the integral of a function with two variables, and the integral is performed over one of them. The second one is the cumulative integral of a function (similar to \psGaussI but valid for all functions). The third one is the result of a convolution. They are defined as:

$$\text{psIntegral}(x) = \int_a^b f(x,t)dt \tag{16}$$

$$\text{psCumIntegral}(x) = \int_{\text{xStart}}^x f(t)dt \tag{17}$$

$$\text{psConv}(x) = \int_a^b f(t)g(x-t)dt \tag{18}$$

In the first one, the integral is performed from a to b and the function f depends on two parameters. In the second one, the function f depends on only one parameter, and the integral is performed from the minimum value specified for x (xStart) and the current value of x in the plot. The third one uses the \psIntegral macro to perform an approximation to the convolution, where the integration is performed from a to b.

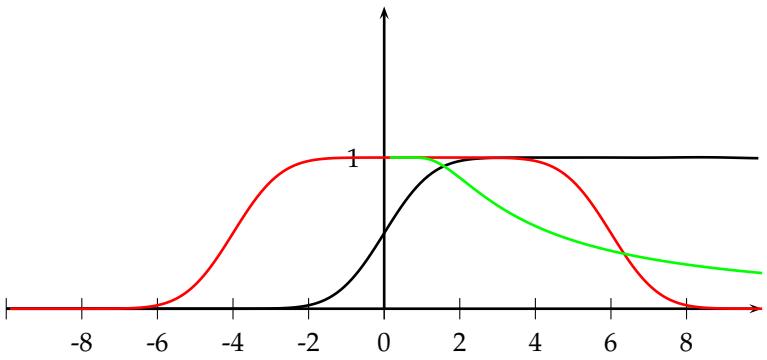
The syntax of these macros is:

```
\psIntegral[<options>]{xStart}{xEnd}(a,b){ function }
\psCumIngegral[<options>]{xStart}{xEnd}{ function }
\psConv[<options>]{xStart}{xEnd}(a,b){ function f }{ function g }
```

In the first macro, the function should be created such that it accepts two values: <x t function> should be a value. For the second and the third functions, they only need to accept one parameter: <x function> should be a value.

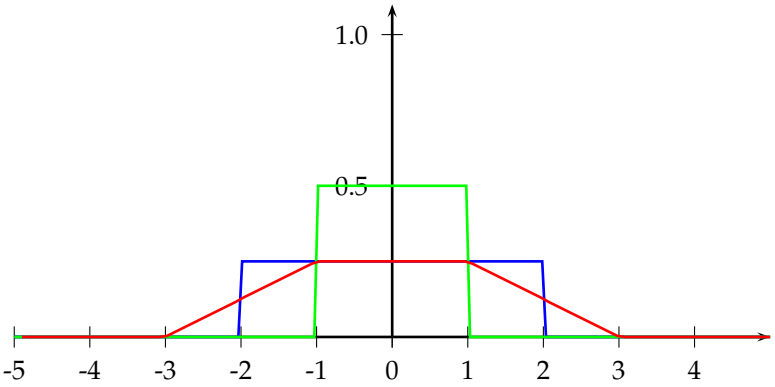
There are no new parameters for these functions. The two most important ones are plotpoints, which controls the number of points of the plot (number of divisions on x for the plot) and Simpson, which controls the precision of the integration (a larger number means a smallest step). The precision and the smoothness of the plot depend strongly on these two parameters.

[1] Created by Jose-Emilio Vila-Forcen




```
%\usepackage{pst-math}
\psset{xunit=0.5cm,yunit=2cm}
\begin{pspicture}[linewidth=1pt](-10,-.5)(10,2)
  \psaxes[dx=1cm,Dx=2]{->}(0,0)(-10,0)(10,2)
  \psCumIntegral[plotpoints=200,Simpson=10]{-10}{10}{0 1 GAUSS}
  \psIntegral[plotpoints=200,Simpson=10,linecolor=red]{-10}{10}(-4,6){1
    GAUSS}
  \psIntegral[plotpoints=200,Simpson=100,linecolor=green]{.1}{10}(-3,3){0
    exch GAUSS}
\end{pspicture}
```

In the example, the cumulative integral of a Gaussian is presented in black. In red, a Gaussian is varying its mean from -10 to 10, and the result is the integral from -4 to 6. Finally, in green it is presented the integral of a Gaussian from -3 to 3, where the variance is varying from .1 to 10.




```
\psset{xunit=1cm,yunit=4cm}
\begin{pspicture}[linewidth=1pt](-5,-.2)(5,1.1)
  \psaxes[dx=1cm,Dx=1,Dy=0.5]{->}(0,0)(-5,0)(5,1.1)
  \psplot[linecolor=blue,plotpoints=200]{-5}{5}{x abs 2 le {0.25}{0} ifelse
    }
  \psplot[linecolor=green,plotpoints=200]{-5}{5}{x abs 1 le {.5}{0} ifelse}
  \psConv[plotpoints=100,Simpson=1000,linecolor=red]{-5}{5}(-10,10)%
    {abs 2 le {0.25}{0} ifelse}{abs 1 le {.5} {0} ifelse}
\end{pspicture}
```

In the second example, a convolution is performed using two rectangle functions. The result (in red) is a trapezoid function.

7 \psBinomial and \psBinomialN

These two macros plot binomial distribution, \psBinomialN the normalized one. It is always done in the x-Intervall $[0;1]$. Rescaling to another one can be done by setting the xunit option to any other value.

The binomial distribution gives the discrete probability distribution $P_p(n|N)$ of obtaining exactly n successes out of N Bernoulli trials (where the result of each Bernoulli trial is true with probability p and false with probability $q = 1 - p$. The binomial distribution is therefore given by

$$P_p(n|N) = \binom{N}{n} p^n q^{N-n} \tag{19}$$

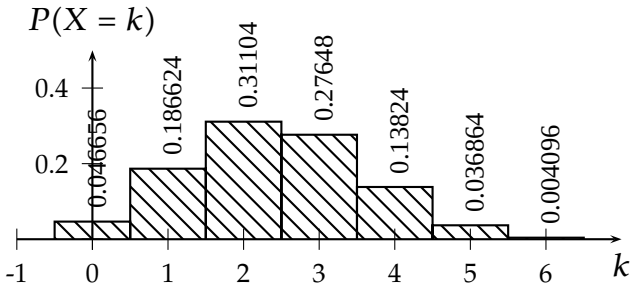
$$= \frac{N!}{n!(N-n)!} p^n (1-p)^{N-n}, \tag{20}$$

where $(N;n)$ is a binomial coefficient and P the probability.

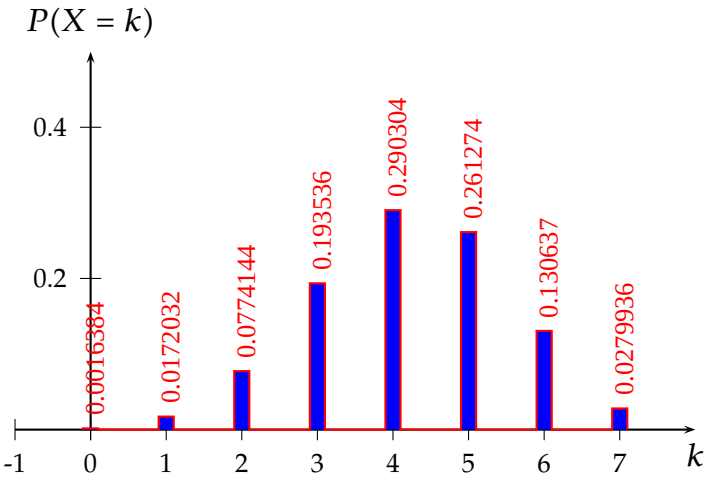
The syntax is quite easy:

```
\psBinomial[<options>]{N}{probability p}
\psBinomialN[<options>]{N}{probability p}
```

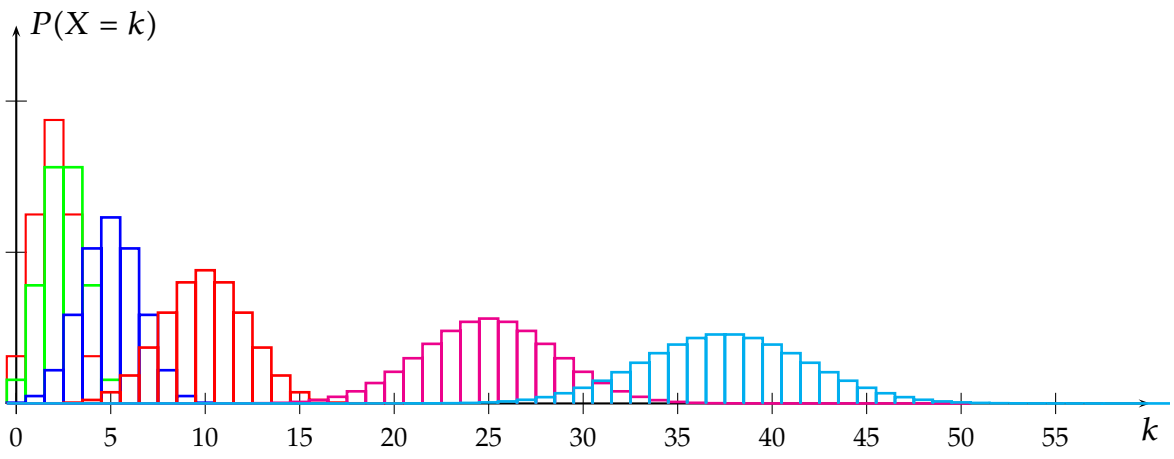
There is a restriction in using the value for N. It depends to the probability, but in general one should expect problems with $N > 100$. PostScript cannot handle such small values and there will be no graph printed. This happens on PostScript side, so TEX doesn't report any problem in the log file. The valid options for the macros are markZeros to draw rectangles instead of a continous line and printValue for printing the y-values on top of the lines, rotated by 90°. For this option all other options from section 9 for the macro \psPrintValue are valid, too. The only special option is barwidth, which is a factor (no dimension) and set by default to 1. This option is only valid for the macro \psBinomial and not for the normalized one!




```
\psset{xunit=1cm,yunit=5cm}%
\begin{pspicture}(-1,-0.15)(7,0.55)%
\psaxes[Dy=0.2,dy=0.2\psyunit]{->}(0,0)(-1,0)(7,0.5)
\uput[-90](7,0){$k$} \uput[90](0,0.5){$P(X=k)$}
\psBinomial[markZeros,printValue,fillstyle=vlines]{6}{0.4}
\end{pspicture}
```



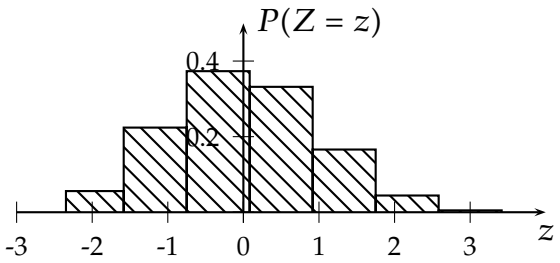

```
\psset{xunit=1cm,yunit=10cm}%
\begin{pspicture}(-1,-0.1)(8,0.6)%
\psaxes[Dy=0.2,dy=0.2\psyunit]{->}(0,0)(-1,0)(8,0.5)
\uput[-90](8,0){$k$} \uput[90](0,0.5){$P(X=k)$}
\psBinomial[linecolor=red,markZeros,printValue,fillstyle=solid,
    fillcolor=blue,barwidth=0.2]{7}{0.6}
\end{pspicture}
```



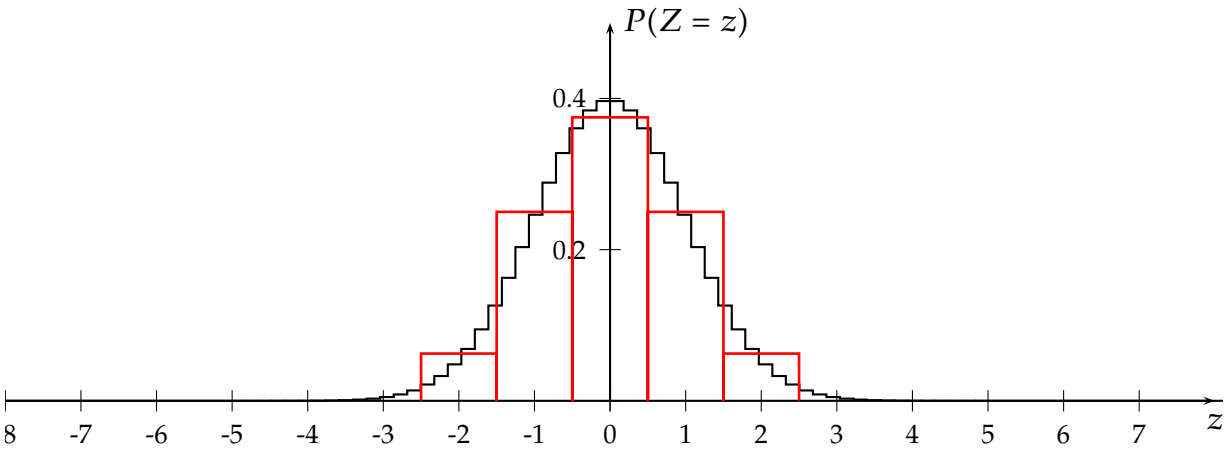

```
\psset{xunit=0.25cm,yunit=10cm}
\begin{pspicture*}(-1,-0.1)(61,0.52)
\psaxes[Dx=5,dx=5\psxunit,Dy=0.2,dy=0.2\psyunit]{->}(60,0.5)
\uput[-90](60,0){$k$} \uput[0](0,0.5){$P(X=k)$}
\psBinomial[markZeros,linecolor=red]{4}{.5}
\psset{linewidth=1pt}
\psBinomial[linecolor=green]{5}{.5}
\psBinomial[linecolor=blue]{10}{.5}
\psBinomial[linecolor=red]{20}{.5}
\psBinomial[linecolor=magenta]{50}{.5}
\psBinomial[linecolor=cyan]{75}{.5}
\end{pspicture*}
```

The default binomial distribution has the mean of $\mu = E(X) = N \cdot p$ and a variant of $\sigma^2 = \mu \cdot (1-p)$. The normalized distribution has a mean of 0. Instead of $P(X = k)$ we use $P(Z = z)$ with $Z = \dfrac{X - E(X)}{\sigma(X)}$ and $P \leftarrow P \cdot \sigma$. The macros use the rekursive definition of the binomial distribution:

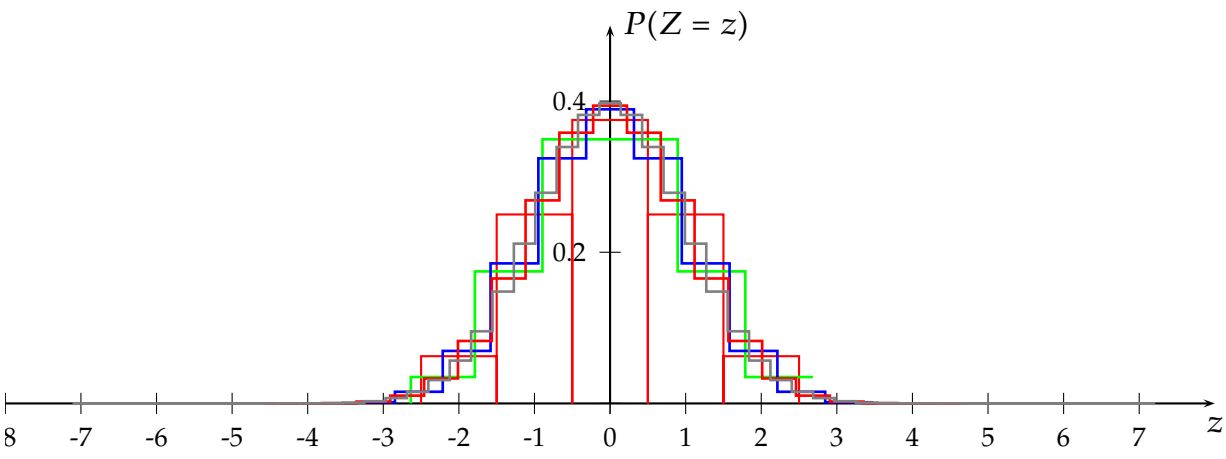
$$P(k) = P(k-1) \cdot \frac{N-k+1}{k} \cdot \frac{p}{1-p} \tag{21}$$




```
\psset{xunit=1cm,yunit=5cm}%
\begin{pspicture}(-3,-0.15)(4,0.55)%
\psaxes[Dy=0.2,dy=0.2\psyunit]{->}(0,0)(-3,0)(4,0.5)
\uput[-90](4,0){$z$} \uput[0](0,0.5){$P(Z=z)$}
\psBinomialN[markZeros,fillstyle=vlines]{6}{0.4}
\end{pspicture}
```

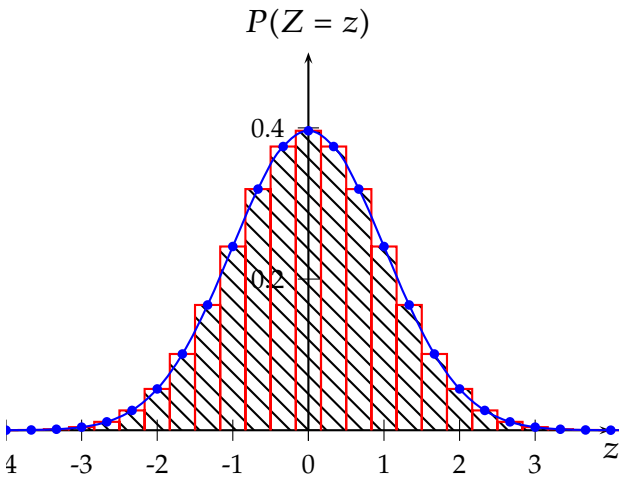



```
\psset{yunit=10}
\begin{pspicture*}(-8,-0.07)(8.1,0.55)
\psaxes[Dy=0.2,dy=0.2\psyunit]{->}(0,0)(-8,0)(8,0.5)
\uput[-90](8,0){$z$} \uput[0](0,0.5){$P(Z=z)$}
\psBinomialN{125}{.5}
\psBinomialN[markZeros,linewidth=1pt,linecolor=red]{4}{.5}
\end{pspicture*}
```

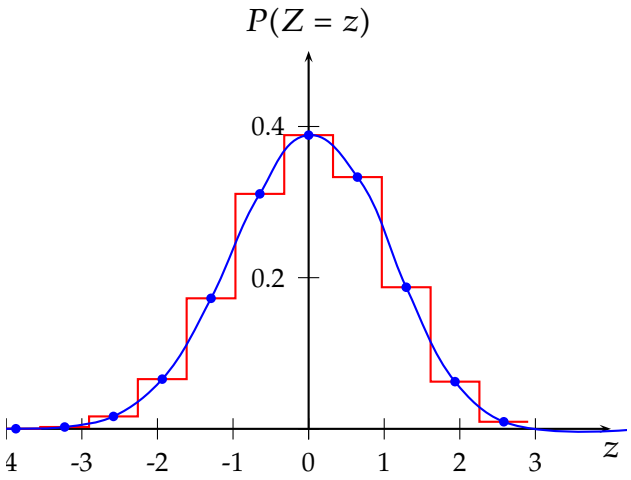



```
\psset{yunit=10}
\begin{pspicture*}(-8,-0.07)(8.1,0.52)
\psaxes[Dy=0.2,dy=0.2\psyunit]{->}(0,0)(-8,0)(8,0.5)
\uput[-90](8,0){$z$} \uput[0](0,0.5){$P(Z=z)$}
\psBinomialN[markZeros,linecolor=red]{4}{.5}
\psset{linewidth=1pt}
\psBinomialN[linecolor=green]{5}{.5}\psBinomialN[linecolor=blue]{10}{.5}
\psBinomialN[linecolor=red]{20}{.5} \psBinomialN[linecolor=gray]{50}{.5}
\end{pspicture*}
```

For the normalized distribution the plotstyle can be set to `curve` (`plotstyle=curve`), then the binomial distribution looks like a normal distribution. This option is only valid vor `\psBinomialN`. The option `showpoints` is valid if `curve` was chosen.




```
\psset{xunit=1cm,yunit=10cm}%
\begin{pspicture*}(-4,-0.06)(4.1,0.57)%
\psaxes[Dy=0.2,dy=0.2\psyunit]{->}(0,0)(-4,0)(4,0.5)%
\uput[-90](4,0){$z$} \uput[90](0,0.5){$P(Z=z)$}%
\psBinomialN[linecolor=red,fillstyle=vlines,showpoints=true,markZeros
  ]{36}{0.5}%
\psBinomialN[linecolor=blue,showpoints=true,plotstyle=curve]{36}{0.5}%
\end{pspicture*}
```



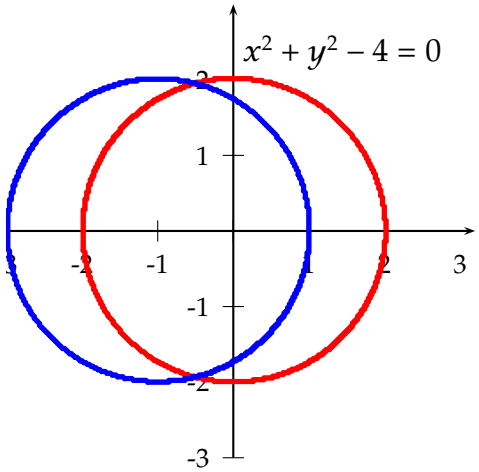

```
\psset{xunit=1cm,yunit=10cm}%
\begin{pspicture*}(-4,-0.06)(4.2,0.57)%
\psaxes[Dy=0.2,dy=0.2\psyunit]{->}(0,0)(-4,0)(4,0.5)%
\uput[-90](4,0){$z$} \uput[90](0,0.5){$P(Z=z)$}%
\psBinomialN[linecolor=red]{10}{0.6}%
\psBinomialN[linecolor=blue,showpoints=true,plotstyle=curve]{10}{0.6}%
\end{pspicture*}
```

8 \psplotImp – plotting implicit defined functions

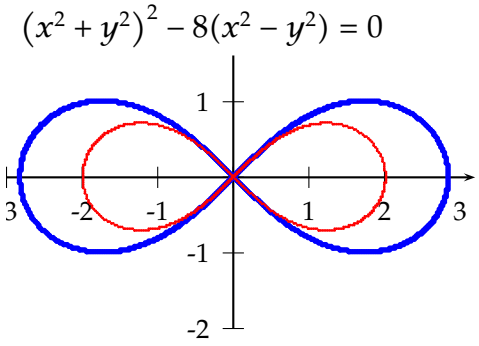
This macro is still experimental! For a given area, the macro calculates in a first step row by row for every pixel (1pt) the function $f(x,y)$ and checks for an changing of the value from $f(x,y) < 0$ to $f(x,y) > 0$ or vice versa. If this happens, then the pixel must be a part of the curve of the function $f(x,y) = 0$. In a second step the same is done column by column. This will take some time because an area of 400×300 pixel needs 120 thousand calculations of the function value. The user still defines this area in his own coordinates, the translation into pixel (pt) is done internally by the macro.

`\psplotImp[<options>](xMin,yMin)(xMax,yMax){<function f(x,y)>}`

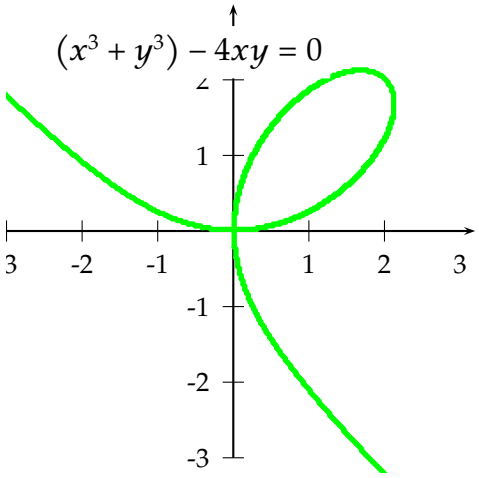
The function must be of $f(x,y) = 0$ and described in PostScript code, or alternatively with the option `algebraic` (`pstricks-add`) in an algebraic form. No other value names than x and y are possible. In general a starred `pspicture` environment maybe a good choice here. The given area for `\psplotImp` should be **greater** than the given `pspicture` area.



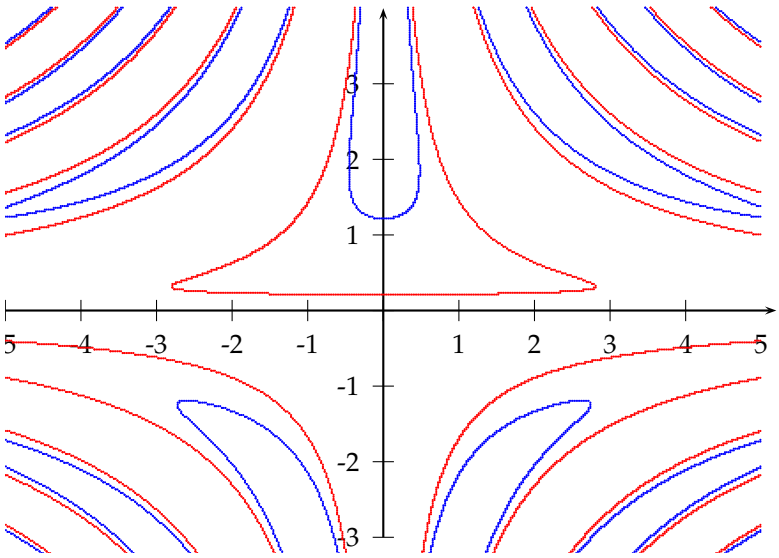

```
\begin{pspicture*}(-3,-3.2)(3.5,3.5)
\psaxes{->}(0,0)(-3,-3)(3.2,3)%
\psplotImp[linewidth=2pt,linecolor=red](-5,-2.1)(5,2.1){%
    x dup mul y dup mul add 4 sub }% circle r=2
\uput[45](0,2){$x^2+y^2-4=0$}
\psplotImp[linewidth=2pt,linecolor=blue,algebraic]%
   (-5,-3)(4,2.4){ (x+1)^2+y^2-4 }% circle r=2
\end{pspicture*}
```




```
\begin{pspicture*}(-3,-2.2)(3.5,2.5)
\psaxes{->}(0,0)(-3,-2)(3.2,2)%
\psplotImp[linewidth=2pt,linecolor=blue](-5,-2.2)(5,2.4){%
     /xqu x dup mul def
     /yqu y dup mul def
     xqu yqu add dup mul 2 dup add 2 mul xqu yqu sub mul sub }
\uput*[0](-3,2){$\left(x^2+y^2\right)^2-8(x^2-y^2)=0$}
\psplotImp[linewidth=1pt,linecolor=red,algebraic](-5,-2.2)(5,2.4){%
  Lemniskate a =2
    (x^2+y^2)^2-4*(x^2-y^2) }
\end{pspicture*}
```

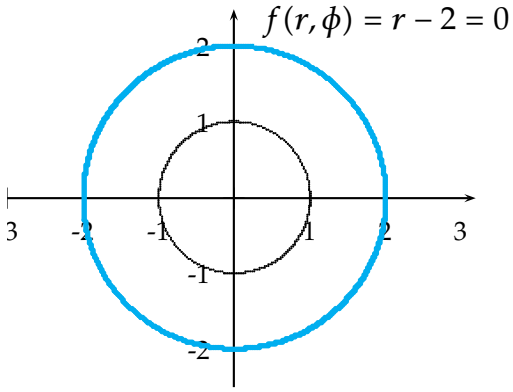



```
\begin{pspicture*}(-3,-3.2)(3.5,3.5)
\psaxes{->}(0,0)(-3,-3)(3.2,3)%
\psplotImp[linewidth=2pt,linecolor=green](-6,-6)(4,2.4){%
    x 3 exp y 3 exp add 4 x y mul mul sub }
\uput*[45](-2.5,2){$\left(x^3+y^3\right)-4xy=0$}
\end{pspicture*}
```

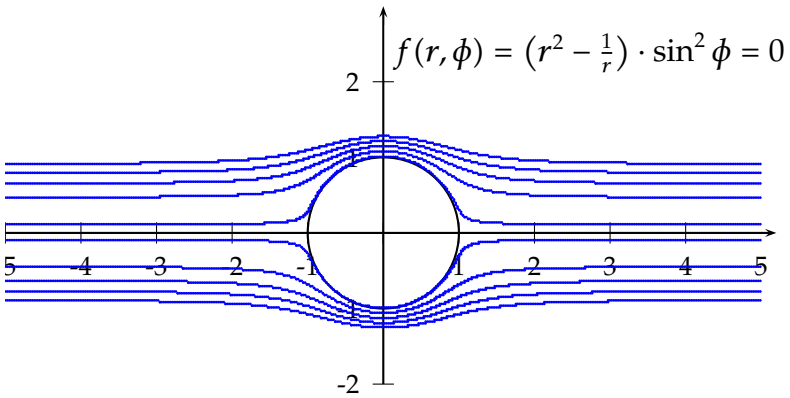



```
\begin{pspicture*}(-5,-3.2)(5.5,4.5)
\psaxes{->}(0,0)(-5,-3)(5.2,4)%
\psplotImp[algebraic,linecolor=red](-6,-4)(5,4){ y*cos(x*y)-0.2 }
\psplotImp[algebraic,linecolor=blue](-6,-4)(5,4){ y*cos(x*y)-1.2 }
\end{pspicture*}
```

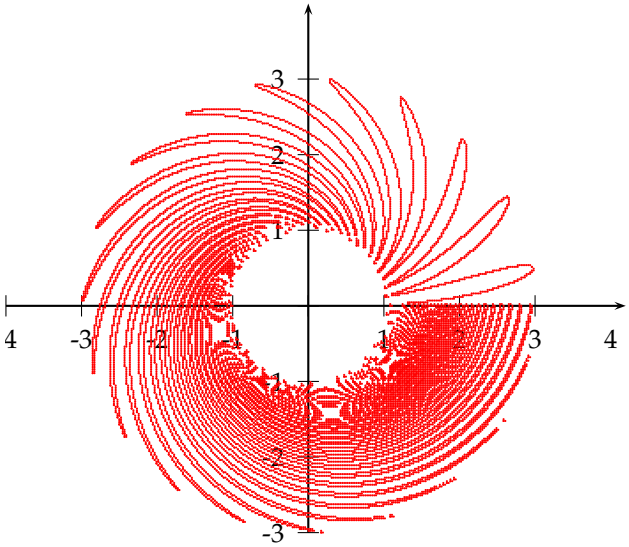
Using the `polarplot` option implies using the variables *r* and *phi* for describing the function, *y* and *x* are not respected in this case. Using the `algebraic` option for polar plots are also possible (see next example).




```
\begin{pspicture*}(-3,-2.5)(3.75,2.75)\psaxes{->}(0,0)(-3,-2.5)(3.2,2.5)%
\psplotImp[linewidth=2pt,linecolor=cyan,polarplot](-6,-3)(4,2.4){ r 2 sub
  }% circle r=2
\uput*[45](0.25,2){$f(r,\phi)=r-2=0$}
\psplotImp[polarplot,algebraic](-6,-3)(4,2.4){ r-1 }% circle r=1
\end{pspicture*}
```




```
\begin{pspicture*}(-5,-2.2)(5.5,3.5)
\pscircle(0,0){1}%
\psaxes{->}(0,0)(-5,-2)(5.2,3)%
\multido{\rA=0.01+0.2}{5}{%
\psplotImp[linewidth=1pt,linecolor=blue,polarplot](-6,-6)(5,2.4){%
    r dup mul 1.0 r div sub phi sin dup mul mul \rA\space sub }}%
\uput*[45](0,2){$f(r,\phi)=\left(r^2-\frac{1}{r}\right)\cdot\sin^2\phi=0$}
\end{pspicture*}
```




```
\begin{pspicture*}(-4,-3.2)(4.5,4.5)
\psaxes{->}(0,0)(-4,-3)(4.2,4)%
\psplotImp[algebraic,polarplot,linecolor=red](-5,-4)(5,4){ r+cos(phi/r)-2
  }
\end{pspicture*}
```

9 \psPrintValue

This new macro allows to print single values of a math function. It has the syntax

```
\psPrintValue[<options>]{<PostScript code>}
```

Important is the fact, that \psPrintValue works on PostScript side. For TEX it is only a box of zero dimension. This is the reason why you have to put it into a box, which reserves horizontal space.

There are the following new options:

Name	Value	Default	
PSfont	PS font name	Times	only valid PostScript font names are possible, e.g. Times-Roman, Helvetica, Courier, AvantGard, Bookman
fontscale	<number>	10	the font scale in pt
valuewidth	<number>	10	the width of the string for the converted real number; if it is too small, no value is printed

x(deg)	$\sin x$	$\cos x$	$\sqrt{x}$	$\sin x + \cos x$	$\sin^2 x + \cos^2 x$
0	0.0	1.0	0.0	*1.0*	1.0
10	0.173648	0.984808	3.16228	*1.15846*	1.0
20	0.34202	0.939693	4.47214	*1.28171*	1.0
30	0.5	0.866025	5.47723	*1.36603*	1.0
40	0.642788	0.766044	6.32456	*1.40883*	1.0
50	0.766044	0.642788	7.07107	*1.40883*	1.0
60	0.866025	0.5	7.74597	*1.36603*	1.0
70	0.939693	0.34202	8.3666	*1.28171*	1.0
80	0.984808	0.173648	8.94427	*1.15846*	1.0
90	1.0	0.0	9.48683	*1.0*	1.0
100	0.984808	-0.173648	10.0	*0.81116*	1.0
110	0.939693	-0.34202	10.4881	*0.597672*	1.0
120	0.866025	-0.5	10.9545	*0.366025*	1.0
130	0.766044	-0.642788	11.4018	*0.123257*	1.0
140	0.642788	-0.766044	11.8322	*-0.123257*	1.0
150	0.5	-0.866025	12.2474	*-0.366025*	1.0
160	0.34202	-0.939693	12.6491	*-0.597672*	1.0
170	0.173648	-0.984808	13.0384	*-0.81116*	1.0

```
\psset{fontscale=12}
\makebox[2em]{x(deg)} \makebox[5em]{$\sin x$} \makebox
  [5em]{$\cos x$}
\makebox[5em]{$\sqrt x$}\makebox[7em]{$\sin x+\cos x$}\
  makebox[6em]{$\sin^2 x+\cos^2 x$}\\[3pt]
\multido{\iA=0+10}{18}{
  \makebox[1em]{\iA}
  \makebox[5em]{\psPrintValue[PSfont=Helvetica]{\iA\
    space sin}}
  \makebox[5em]{\psPrintValue[PSfont=Courier,fontscale
    =10]{\iA\space cos}}
  \makebox[5em]{\psPrintValue[valuewidth=15,linecolor=
    blue,PSfont=AvantGarde]{\iA\space sqrt}}
  \makebox[7em]{\psPrintValue[PSfont=Times-Italic]{\iA\
    space dup sin exch cos add}}
  \makebox[6em]{\psPrintValue[PSfont=Palatino-Roman]{\
    iA\space dup sin dup mul exch cos dup mul add}}\\}
```

10 Credits

Denis Girou | Manuel Luque | Timothy Van Zandt

References

[1] Denis Girou. Présentation de PSTricks. *Cahier GUTenberg*, 16:21–70, April 1994.

[2] Michel Goosens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Graphics Companion*. Addison-Wesley Publishing Company, Reading, Mass., 1997.

[3] Laura E. Jackson and Herbert Voß. Die Plot-Funktionen von `pst-plot`. *Die TEXnische Komödie*, 2/02:27–34, June 2002.

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

[5] Herbert Voß. *Chaos und Fraktale selbst programmieren: von Mandelbrotmengen über Farbmanipulationen zur perfekten Darstellung*. Franzis Verlag, Poing, 1994.

[6] Herbert Voß. Die mathematischen Funktionen von PostScript. *Die TEXnische Komödie*, 1/02, March 2002.

[7] Herbert Voß. *PSTricks – Grafik für TEX und LATEX*. DANTE – Lehmanns, Heidelberg/Hamburg, third edition, 2006.

[8] Timothy van Zandt. *PSTricks - PostScript macros for generic TEX*. http://www.tug.org/application/PSTricks, 1993.

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

[10] Timothy van Zandt. *pst-plot: Plotting two dimensional functions and data*. CTAN:graphics/pstricks/generic/pst-plot.tex, 1999.

[11] Timothy van Zandt and Denis Girou. Inside PSTricks. *TUGboat*, 15:239–246, September 1994.