PSTricks

pst-tools

Helper functions; v.0.12

April 13, 2023

Package author(s):
Herbert Voß

This package defines some tools which are useful for all packages not only the PSTricks like packages. Since the version 0.10 it includes the macros from `random.tex`.

Thanks to: Marcel Krüger; Pablo Gonzáles Luengo; Rolf Niepraschk;

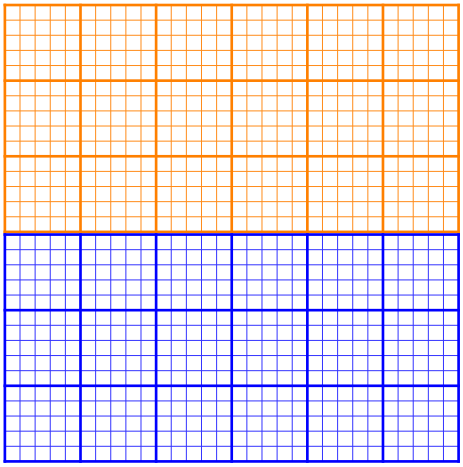
Contents

1 Predefined styles

The style `mmpaper` is defined for `\psgrid`:

```
\begin{pspicture}(6,3)
\psgrid[style=mmpaper](6,3)
\end{pspicture}

\begin{pspicture}(6,3)
\psgrid[style=mmpaper,
  gridcolor=blue,subgridcolor=blue!80](6,3)
\end{pspicture}
```



2 \psPrintValue

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

`\psPrintValue` `[Options]` {*PostScript code*}
`\psPrintValue` `[algebraic,…]` {*x value, algebraic 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 valid options for `\psPrintValue`:

name	value	default	
`printfont`	`font name`	`Times`	only the current font (`printfont=`) or valid `PostScript` font names are possible, e.g. `Times-Roman`, `Helvetica`, `Courier`, `Helvetica`, `Bookman`. If you want to embed the fonts use always the URW names NimbusRomNo9L-Regu, NimbusSanL-Regu and NimbusMonL-Regu. However, the names may vary on different operating systems. If you leave the argument empty, it will choose the currently active font.

name	value	default	
postString	<string>	{}	will be appended to the number string
trimSpaces	<boolean>	false	will strip spaces on the right
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
decimals	<number>	-1	the number of printed decimals, a negative value prints all possible digits.
xShift	<number>	0	the x shift in pt for the output, relative to the current point.
algebraic	<boolean>	false	function in algebraic notation.
VarName	<string>	{}	saves the value in /<VarName> for further use
comma	<boolean>	false	comma instead of the dor for decimals

x(deg)	$\sin x$	$\cos x$	$\sqrt{x}$	$\sin x + \cos x$	$\sin^2 x + \cos^2 x$
0	0	1	0	*1*	1
10	0.173648	0.984	3.16228	*1,15846*	1
20	0.34202	0.939	4.47214	*1,28171*	1
30	0.5	0.866	5.47723	*1,36603*	1
40	0.642788	0.766	6.32456	*1,40883*	1
50	0.766044	0.642	7.07107	*1,40883*	1
60	0.866025	0.5	7.74597	*1,36603*	1
70	0.939693	0.342	8.3666	*1,28171*	1
80	0.984808	0.173	8.94427	*1,15846*	1
90	1	0	9.48683	*1*	1
100	0.984808	-0.174	10	*0,81116*	1
110	0.939693	-0.343	10.4881	*0,597672*	1
120	0.866025	-0.5	10.9545	*0,366025*	1
130	0.766044	-0.643	11.4018	*0,123257*	1
140	0.642788	-0.767	11.8322	*-0,123257*	1
150	0.5	-0.867	12.2474	*-0,366025*	1
160	0.34202	-0.94	12.6491	*-0,597672*	1
170	0.173648	-0.985	13.0384	*-0,81116*	1

```
\psset{fontscale=12}
\makebox[2em]{x(deg)} \makebox[5em]{$\sin x$} \makebox[4em]{$\cos x$}\hspace{1em}
\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[printfont=NimbusRomNo9L-Regu,xShift=-10]{\iA\space sin}}
  \makebox[4em][r]{\psPrintValue[printfont={},fontscale=10,decimals=3,xShift=-20]{\iA\space cos}}\hspace{1em}
  \makebox[5em]{\psPrintValue[valuewidth=15,linecolor=blue,printfont=NimbusSanL-Regu]{\iA\space sqrt}}
  \makebox[7em]{\psPrintValue[comma,printfont=NimbusRomNo9L-ReguItal]{\iA\space dup sin exch cos add}}
  \makebox[6em]{\psPrintValue[printfont=Palatino-Roman]{\iA\space dup sin dup mul exch cos dup mul add}}\\}
```

With enabled `algebraic` option there must be two arguments, separated by a comma. The first one is the x value as a number, which can also be PostScript code, which leaves a number on the stack. The second part is the function described in algebraic notation. Pay attention, in algebraic notation angles must be in radian and not degrees.

x(deg)	$\sin x$	$\cos x$	$\sqrt{x}$	$\sin x + \cos x$	$\sin^2 x + \cos^2 x$
0.0	0	1	0	*1*	1
0.1	0.0998334	0.995	0.316228	*1,09484*	1
0.20001	0.198679	0.98	0.447225	*1,17874*	1
0.30002	0.295539	0.955	0.547741	*1,25087*	1
0.40002	0.389437	0.921	0.632471	*1,31049*	1
0.50003	0.479452	0.877	0.707128	*1,35702*	1
0.60004	0.564675	0.825	0.774622	*1,38999*	1
0.70004	0.644248	0.764	0.836684	*1,40906*	1
0.80005	0.717391	0.696	0.894455	*1,41406*	1
0.90005	0.783358	0.621	0.94871	*1,40493*	1
1.00006	0.841503	0.54	1.00003	*1,38176*	1
1.10007	0.891239	0.453	1.04884	*1,34477*	1
1.20007	0.932064	0.362	1.09548	*1,29436*	1
1.30008	0.96358	0.267	1.14021	*1,231*	1
1.40009	0.985465	0.169	1.18325	*1,15534*	1
1.50009	0.997501	0.07	1.22478	*1,06815*	1
1.6001	0.999571	-0.03	1.26495	*0,970271*	1
1.7001	0.991652	-0.129	1.30388	*0,862708*	1

```
\psset{algebraic, fontscale=12}% All functions now in algebraic notation
\makebox[2em]{x(deg)} \makebox[5em]{$\sin x$} \makebox[4em]{$\cos x$}\hspace{1em}
\makebox[5em]{$\sqrt x$}\makebox[7em]{$\sin x+\cos x$}\makebox[6em]{$\sin^2 x+\cos^2 x$}\\[3pt]
\multido{\rA=0+0.1}{18}{\makebox[1em]{\rA}
  \makebox[5em]{\psPrintValue[printfont=NimbusSanL-Regu,xShift=-10]{\rA, sin(x)}}
  \makebox[4em][r]{\psPrintValue[printfont={},fontscale=10,decimals=3,xShift=-20]{\rA,cos(x)}}\hspace{1em}
  \makebox[5em]{\psPrintValue[valuewidth=15,linecolor=blue,printfont=NimbusSanL-Regu]{\rA,sqrt(x)}}
  \makebox[7em]{\psPrintValue[comma,printfont=NimbusRomNo9L-ReguItal]{\rA,sin(x)+cos(x)}}
  \makebox[6em]{\psPrintValue[printfont=Palatino-Roman]{\rA,sin(x)^2+cos(x)^2}}\\}
```

foo 3,1 bar

foo 3,1° bar

foo 9.8596° bar

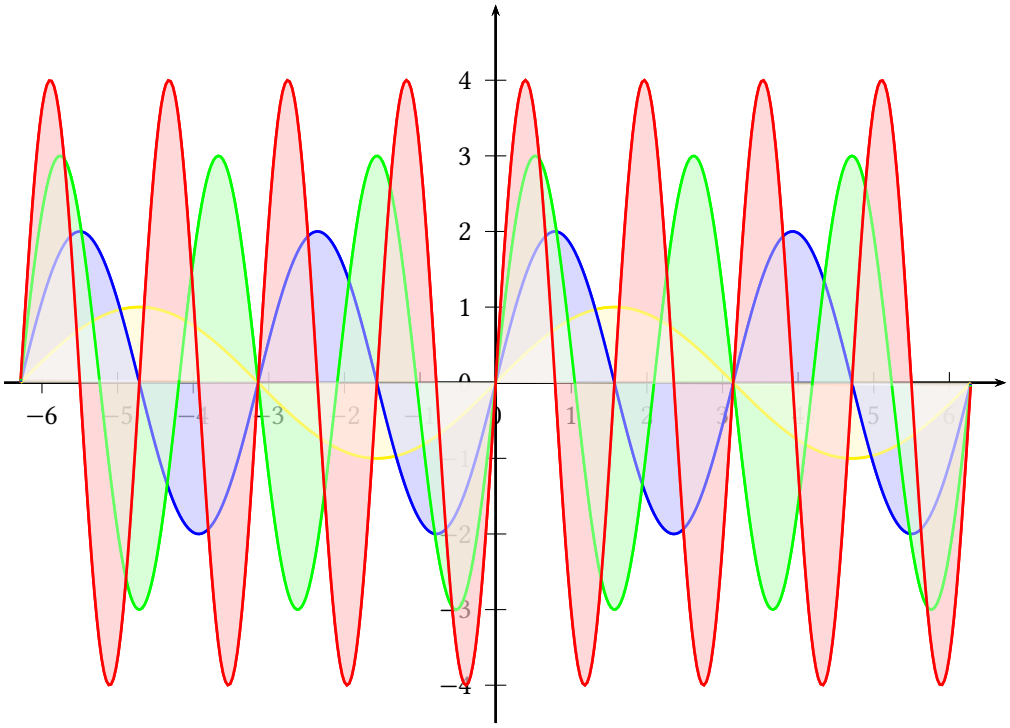
```
foo \makebox[2em][l]{\psPrintValue[comma]{3.14 10 mul round 10 div}}bar\\[3pt]
foo \makebox[2em][l]{\psPrintValue[comma,printfont=StandardSymL,
  postString=\string\260]{3.14 10 mul round 10 div}}bar\\[3pt]
foo \makebox[3.5em][l]{\psPrintValue[printfont=StandardSymL,decimals=6,
  postString=\string\260]{3.14 dup mul}}bar
```

3 \psRegisterList

The macro defines for every list item an own macro for an easy access to the items. It must be a comma separated list.

\psRegisterList{*Name*}{*value list*}
\⟨*Name*⟩{*Index*}

```
\psRegisterList{Color}{yellow,blue,green,red}% defines macro \Color
\begin{pspicture}(-7,-4.5)(7,5.5)
\psaxes{->}(0,0)(-6.5,-4.5)(6.75,5)
\psset{plotpoints=400,algebraic,linewidth=1pt,fillstyle=solid,opacity=0.4}
\multido{\iA=1+1}{4}{%
  \psplot[linecolor=\Color{\iA},
    fillcolor=\Color{\iA}!60]{-6.283}{6.283}{\iA*sin(\iA*x)}}%
\psset{plotpoints=400,algebraic}
\psforeach{\iA}{1,2,3,4}{%
  \psplot[linecolor=\Color{\iA}]{-6.28}{6.28}{\iA*sin(\iA*x)}}
\end{pspicture}
```



4 Random numbers

The file `random.tex` from Donald Arseneau is no more part of CTAN due to a missing licence statement. `pst-tools` at version 0.10 includes the code. The documentation was inside the package otself:

Random integers are generated in the range 1 to 2147483646 by the macro `\nextrandom`. The result is returned in the counter `\randomi`. Do not change `\randomi` except, perhaps, to initialize it at some random value. If you do not initialize it, it will be initialized using the time and date. (This is a sparse initialization, giving fewer than a million different starting values, but you should use other sources of numbers if they are available–just remember that most of the numbers available to TeX are not at all random.)

The `\nextrandom` command is not very useful by itself, unless you have exactly 2147483646 things to choose from. Much more useful is the `\setrannum` command which sets a given counter to a random value within a specified range. There are three parameters:

`\setrannum{<counter>}{<minimum>}{<maximum>}`

For example, to simulate a die-roll:

`\setrannum{\die}{1}{6} \ifcase\die... .`

If you need random numbers that are not integers, you will have to use dimen registers and `\setrandimen`. For example, to set a random page width:

`\setrandimen \hsize{3in}{6.5in}`

The »`\pointless`« macro will remove the »pt« that TeX gives so you can use the dimensions as pure 'real' numbers. In that case, specify the range in pt units. For example,

`\setrandimen\answer{2.71828pt}{3.14159pt}`

The answer is `\pointless\answer`.

The random number generator is the one by Lewis, Goodman, and Miller (1969) and used as `ran0` in »Numerical Recipies« using Schrage's method for avoiding overflows. The multiplier is 16807(7^5), the added constant is 0, and the modulus is 2147483647($2^{31}-1$). The range of integers generated is $1-2147483646$. A smaller range would reduce the complexity of the macros a bit, but not much–most of the code deals with initialization and type-conversion. On the other hand, the large range may be wasted due to the sparse seed initialization.

5 List of the defined PostScript functions

```
/Pi2 1.5707963267948966192564 0 def
/factorial { % n on stack, returns n!
/MoverN { % m n on stack, returns the binomial coefficient m over n
/ps@ReverseOrderOfPoints { % on stack [P1 P2 P3 ...Pn]=>[Pn,Pn-1,...,P2,P1]
/cxadd { % [a1 b1] [a2 b2] = [a1+a2 b1+b2]
/cxneg { % [a b]
/cxsub { cxneg cxadd } def  % same as negative addition
/cxmul { % [a1 b1] [a2 b2]
/cxsqr { % % [a b]^2 = [a^2-b^2 2ab] = [a2 b2]
/cxsqrt { %
/cxarg {  % [a b]->arg(z)=atan(b/a)
/cxlog { % [a b]->log[a b] = [a^2-b^2 2ab] = [a2 b2]
/cxnorm2 { % [a b]->a^2+b^2
/cxnorm { %
/cxconj { % [a b]->[a -b]
/cxre { 0 get } def % real value
/cxim { 1 get } def % imag value
/cxrecip { % [a b]->1/[a b] = ([a -b]/(a^2+b^2)
/cxmake1 { 0 2 array astore } def % make a complex number, real given
/cxmake2 { 2 array astore } def   % dito, both given
/cxdiv { cxrecip cxmul } def
/cxrmul { % [a b] r->[r*a r*b]
/cxrdiv { % [a b] r->[1/r*a 1/r*b]
/cxconv { % theta->exp(i theta) = cos(theta)+i sin(theta) polar<->cartesian
/bubblesort { % on stack must be an array [ ... ]
/concatstringarray{  %  [(a) (b) ... (z)] --> (ab...z)  20100422
/concatstrings{ % (a) (b) -> (ab)
/reversestring { % (aBC) -> (CBa)
/concatarray{ % [a c] [b d] -> [a c b d]
/dot2comma {% on stack a string (...)
/rightTrim { % on stack the string and the character number to be stripped
/psStringwidth /stringwidth load def
/psShow /show load def
```

6 List of all optional arguments for pst-tools

Key	Type	Default
decimalSeparator	ordinary	.
comma	boolean	true
trimSpaces	boolean	true
xShift	ordinary	0
yShift	ordinary	0
postString	ordinary	
VarName	ordinary	
printfont	ordinary	NimbusRomNo9L-Regu
valuewidth	ordinary	10
fontscale	ordinary	10
decimals	ordinary	-1
round	boolean	true
science	boolean	true

References

[1] Denis Girou. “Présentation de PSTricks”. **in***Cahier GUTenberg*: 16 (**april** 1994), **pages** 21–70.

[2] Michel Goosens **andothers**. *The LaTeX Graphics Companion*. 2 **edition**. 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] Nikolai G. Kollock. *PostScript richtig eingesetzt: vom Konzept zum praktischen Einsatz*. Vaterstetten: IWT, 1989.

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

[6] Herbert Voß. *PSTricks Grafik für TeX und LaTeX*. 7 **edition**. Heidelberg/Berlin: DANTE **and** Lehmanns, 2016.

[7] Herbert Voß. *PSTricks Graphics for LaTeX*. 1 **edition**. Cambridge: UIT, 2011.

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

[9] Timothy Van Zandt. *PSTricks - PostScript macros for generic TeX*. http://www.tug.org/application/PSTricks, 1993.

[10] Timothy Van Zandt **and** Denis Girou. “Inside PSTricks”. **in***TUGboat*: 15 (**september** 1994), **pages** 239–246.

Index