dspTricks
PsTricks Macros for Digital Signal Processing

Paolo Prandoni

The package dspTricks is a set of psTricks macros for plotting graphs and figures useful in the illustration of one-dimensional signal processing concepts and applications. These macros have been developed by the author while working on the textbook "Signal Processing for Communication" (EPFL Press, 2007). The basic dspTricks plot is a boxed chart displaying a discrete-time or a continuous-time signal, or a superposition of both; discrete-time signals are plotted using the "lollipop" formalism while continuous-time functions are rendered as smooth curves. Other types of plots that commonly occur in the signal processing literature and for which dspTricks offers macros are frequency plots and pole-zero plots. The companion package dspFunctions defines some signals commonly used in basic signal processing in terms of PostScript primitives, while the package dspBlocks provides a set of macros to design simple signal processing block diagrams.

1 Data Plots

`dspPlot` Data plots are defined by the `dspPlot` environment as

```
\begin{dspPlot}[⟨options⟩]{⟨xmin, xmax⟩}{⟨ymin, ymax⟩}
...
\end{dspPlot}
```

This sets up a data plot with the horizontal axis spanning the ⟨*xmin*⟩-⟨*xmax*⟩ interval and with the vertical axis spanning the ⟨*ymin*⟩-⟨*ymax*⟩ interval. The following options are available for all data plots:

`width`=⟨*dim*⟩ : width of the plot (using any units)

`height`=⟨*dim*⟩ : height of the plot. Width and height specify the size of the active plot area, i.e., of the *boxed region* of the cartesian plane specified by the x and y ranges for the plot. This is possibly augmented by the space required by the optional labels and axis marks. You can set the default size for a plot by setting the `\dspW` and `\dspH` lengths at the beginning of your document `xtype` = `time` | `freq` : type of plot: time domain (default) or digital frequency plot

`xticks` = `auto` | `custom` | `none` | ⟨*step*⟩ : labeling of the horizontal axis

`yticks` = `auto` | `custom` | `none` | ⟨*step*⟩ : labeling of the vertical axis. When the option specifies a numeric value ⟨*step*⟩, that will be the spacing between two consecutive ticks on the axis[1]. When `auto` is selected, the spacing will be computed automatically as a function of the axis range. When `none` is selected, no ticks will be drawn. When `custom` is selected, no ticks will be drawn but the plot will include the appropriate spacing for ticks to be drawn later via the `\dspCustomTicks` macro.

`sidegap` = ⟨*gap*⟩ : extra space to the left and the right of the x-axis range. Useful in discrete-time plots not to have stems overlapping the plot's frame. By default, it's automatically determined as a function of the range.

`xout` = `true` | `false` : normally, ticks and tick labels for the horizontal axis are placed on the axis, which may be inside the box; set this option to `true` if you want to place the ticks on the lower edge of the box in all cases.

`inticks` = `true` | `false` : x-axis ticks are normally extending downwards; by setting this option to `true` ticks will be pointing upwards, i.e. they will be inside the plot box even when the x-axis coincides with the bottom of the box.

`xlabel` = ⟨*label*⟩ : label for the horizontal axis; placed outside the plot box

`ylabel` = ⟨*label*⟩ : label for the vertical axis; placed outside the plot box, on the left

`rlabel` = ⟨*label*⟩ : additional label for the horizontal axis; placed outside the plot box on the right

Within a `dspPlot` environment you can use the plotting commands described in the next sections, as well as any psTricks command; in the latter case, the psTricks values for `xunit` and `yunit` are scaled to the axes (i.e., they correspond to the cartesian values of the plot). Other useful commands for all data plots are the following:

`dspClip`
- in order to make sure that all drawing commands are clipped to the bounding box defined by the box chart, you can enclose them individually in a predefined `dspClip` environment. See section 1.1.2 for an example.

`dspPlotFrame`
- to redraw the framing box (useful to "smooth out" plots touching the

[1]For digital frequency plots, `xticks` has a different meaning; see Section 1.2 for details.

frame) you can use the command `\dspPlotFrame`

`dspCustomTicks`

- to draw arbitrarily placed ticks (and tick labels) on either axis, use

  `\dspCustomTicks[`⟨*options*⟩`]{`⟨*pos label pos label ...*⟩`}`

  where the axis is specified in the options field as either `axis=x` (default) or `axis=y` and where the argument is a list of space-separated coordinate-label pairs. If you use math mode for the labels, *do not use spaces in your formulas* as that will confuse the list-parsing macros.

1.1 Time-Domain Plots

1.1.1 Discrete-Time Signals

The following commands generate stem (or "lollipop") plots; available options in the commands are all standards psTricksoptions plus other specialized options when applicable.

`dspTaps`

- To plot a set of discrete time points use

  `\dspTaps[`⟨*options*⟩`]{`⟨*data*⟩`}`

  where ⟨*data*⟩ is a list of space-separated index-value pairs (e.g., values pre-computed by an external numerical package). Allowed options are only generic psTricks options.

`dspTapsFile`

- for large data sets, you can use

  `\dspTapsFile[`⟨*options*⟩`]{`⟨*fileName*⟩`}`

  where now ⟨*fileName*⟩ points to an external text file of space-separated index-value pairs.

`dspSignal`

- To plot a discrete-time signal defined in terms of PostScript primitives use

  `\dspSignal[`⟨*options*⟩`]{`⟨*PostScript code*⟩`}`

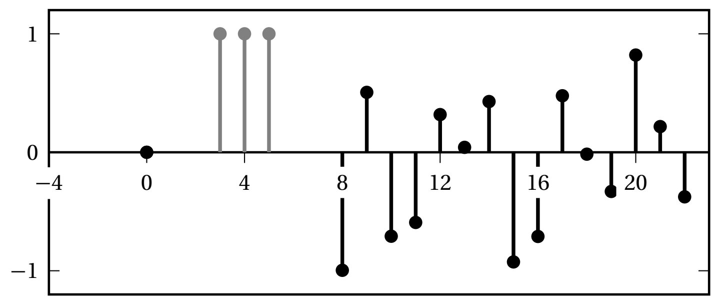
  `xmin,xmax`

  The PostScript code must use the variable `x` as the independent variable; the `\dspPlot` environment sweeps `x` over all integers in the ⟨*xmin*⟩-⟨*xmax*⟩ interval defined for the plot; this can be changed for each individual signal by using the options `xmin=`⟨*m*⟩ and/or `xmax=`⟨*n*⟩. If you use TEXmacros in your PS code, make sure you include a space at the end of the macro definition. For instance, use `\def\gain\{0.75␣}`.

For example:

```
\begin{dspPlot}{-3, 22}{-1.2, 1.2}
  \dspSignal[xmin=8]{rand 2147483647 div 0.5 sub 2 mul}
  \dspTaps[linecolor=gray]{3 1 4 1 5 1}
  \dspTaps{0 0}
\end{dspPlot}
```

produces the following plot:



If you are viewing this document in a PostScript viewer, you can see that the random signal is different every time you reload the page, since the taps values are computed on the fly by the PostScript interpreter.

1.1.2 Continuous-Time Signals

Continuous-time functions can be plotted with the following commands:

`dspFunc`

- You can draw a continuous-time signal by using the command

  `\dspFunc[`⟨*options*⟩`]{`⟨*PostScript code*⟩`}`

  again, the PostScript code must use `x` as the independent variable; the range for `x` is the ⟨*xmin*⟩-⟨*xmax*⟩ interval and can be controlled for each signal independently via the`xmin` and `xmax` options.

`dspFuncData`

- To plot a smooth function obtained by interpolating a list of space separated time-value pairs use

  `\dspFuncData[`⟨*options*⟩`]{`⟨*data*⟩`}`

  the interpolation is performed by the PostScript interpreter and can be controlled if necessary by using the appropriate psTricks options.

`dspFuncFile`

- For a continuous-time smooth interpolation of a pre-computed set of data points, use

\dspFuncFile[⟨*options*⟩]{⟨*fileName*⟩}

where ⟨*fileName*⟩ points to a text file containing the data points as a space-separated list of abscissae and ordinates.

dspDiracs

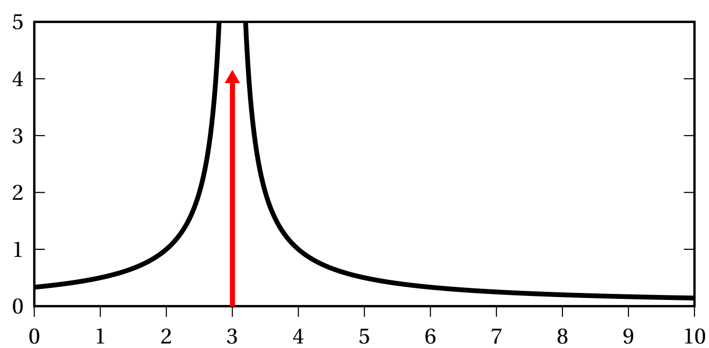
- To plot one or more Dirac deltas (symbolized by a vertical arrow) use

  \dspDiracs[⟨*options*⟩]{⟨*pos value pos value ...*⟩}

  where the argument is a list of space-separated time-value pairs.

In the following example, note the use of the `dspClip` environment when plotting the hyperbola[2]:

```
\begin{dspPlot}[yticks=1,sidegap=0]{0,10}{0,5}
  \begin{dspClip}\dspFunc{1 3 x sub div abs}\end{dspClip}
  \dspDiracs[linecolor=red]{3 4}
\end{dspPlot}
```

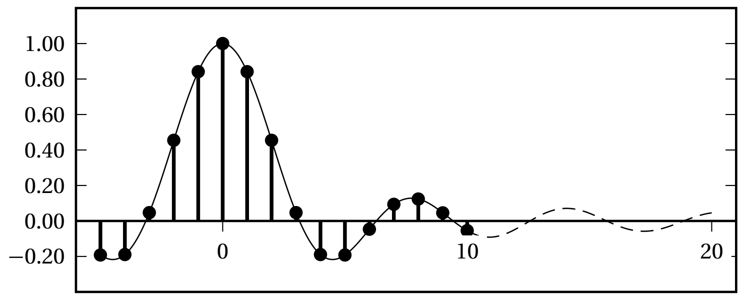


[2]Make sure not to leave any blank space in between the beginning and end of the `dspClip` environment, otherwise the axis labels may fall out of alignment.

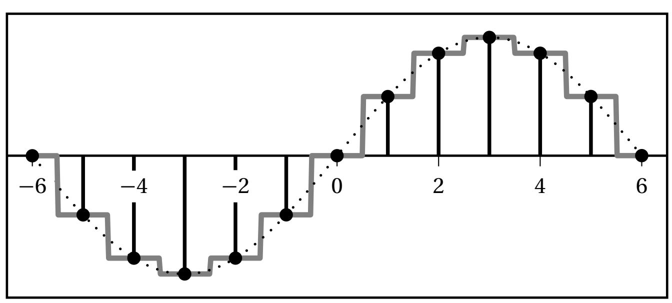
1.1.3 Mixed-Domain Examples

In the following examples we mix discrete- and continuous-time signals in the same plot:

```
\begin{dspPlot}[xticks=10,yticks=0.2]{-5, 20}{-0.4, 1.2}
  \def\sincx{x 0 eq {1} {x RadtoDeg sin x div} ifelse}
  \dspSignal[xmax=10]{\sincx}
  \dspFunc[linewidth=0.5pt,xmax=10]{\sincx}
  \dspFunc[linestyle=dashed,linewidth=0.5pt,xmin=10]{\sincx}
\end{dspPlot}
```




```
\begin{dspPlot}[sidegap=0.5,yticks=none]{-6, 6}{-1.2, 1.2}
  \def\signal{ 0.5235 mul RadtoDeg sin }
  \def\quantize{ dup 0 gt {-0.5} {0.5} ifelse sub truncate }
  \dspFunc[linecolor=gray,linewidth=2pt]{x \quantize \signal }
  \dspFunc[linestyle=dotted,linewidth=1pt]{x \signal}
  \dspSignal{x \signal}
\end{dspPlot}
```



1.2 Digital Frequency Plots

Digital frequency[3] plots are set up by setting the option `xtype=freq` in the `dspPlot` environment; they are very similar to continuous-time plots, except for the following:

- the horizontal axis represents angular frequency; its range is specified in normalized units so that, for instance, a range of `{-1,1}` as the first argument to `dspPlot` indicates the frequency interval $[-\pi, \pi]$.
- tick labels on the horizontal axis are expressed as integer fractions of π; in this sense, the `xticks` parameter, when set to a numeric value, indicates the denominator of said fractions.
- `sidegap` is always zero in digital frequency plots.

All digital spectra are 2π-periodic, hence the $[-\pi, \pi]$ interval is sufficient to completely represent the function. However, if you want to explicitly plot the function over a wider interval, it is your responsibility to make the plotted data

`dspPeriodize` 2π-periodic; the `\dspPeriodize` macro can help you do that, as shown in the examples below. Also, when writing PostScript code, don't forget to scale the x variable appropriately; in particular, PostScript functions of an angle use units in degrees, so you need to multiply `x` by 180 before computing trigonometric functions.

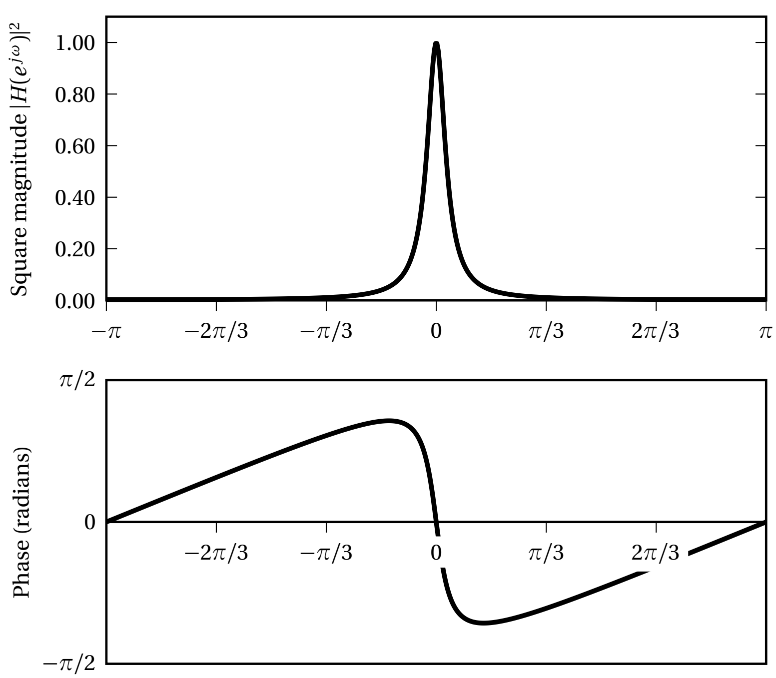
[3]By "digital spectrum" of a discrete-time sequence $x[n]$ I mean the Discrete-Time Fourier transform

$$X(e^{j\omega}) = \sum_{n=-\infty}^{\infty} x[n] e^{-j\omega n}$$

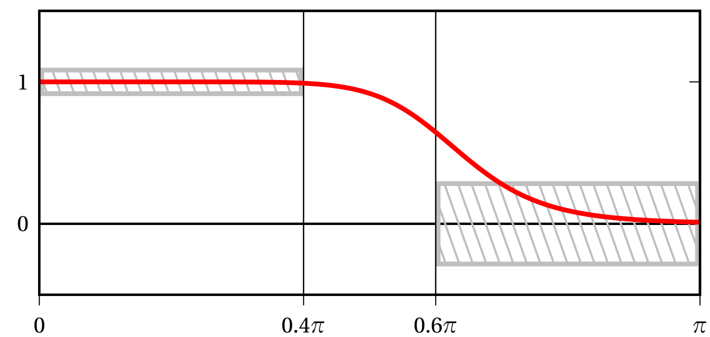
```
\def\lambda{0.9 }
\def\magn{\lambda 1 sub dup mul 1 \lambda \lambda mul %
  add  x 180 mul cos 2 mul \lambda mul sub div }
\def\phase{\lambda x 180 mul sin mul -1 mul 1 \lambda %
  x 180 mul cos mul sub atan 180 div 3.1415 mul }

\begin{dspPlot}[xtype=freq,xticks=3,yticks=0.2, %
  ylabel={Square magnitude $|H(e^{j\omega})|^2$}]{-1,1}{0,1.1}
  \dspFunc{\magn }
\end{dspPlot}

\begin{dspPlot}[xtype=freq,xticks=3,yticks=custom, %
  ylabel={Phase (radians)}]{-1,1}{-1.57,1.57}
  \dspFunc[xmax=0]{\phase }
  \dspFunc[xmin=0]{-\phase -1 mul}
  \dspCustomTicks[axis=y]{-1.57 $-\pi/2$ 0 0 1.57 $\pi/2$}
\end{dspPlot}
```



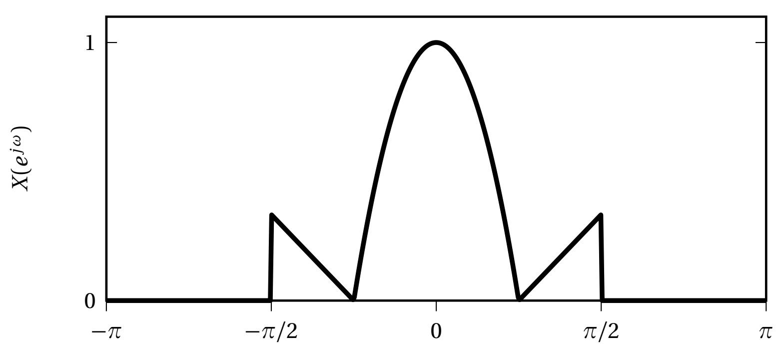
```
\begin{dspPlot}[xtype=freq,xticks=1,yticks=1,,xout=true]{0, 1}{-0.5,1.5}
  \psframe[fillstyle=vlines,%
            hatchcolor=lightgray,hatchangle=20,%
            linecolor=lightgray]%
            (0,1.1)(0.4,0.9)
  \psframe[fillstyle=vlines,%
            hatchcolor=lightgray,hatchangle=20,%
            linecolor=lightgray]%
            (0.6,0.3)(1,-0.3)
  \psline[linewidth=0.5pt](0.4,-.5)(0.4,1.5)
  \psline[linewidth=0.5pt](0.6,-.5)(0.6,1.5)
  \dspFunc[linecolor=red]{x 3.14 mul 0.5 mul 10 exp 1 add 1 exch div}
  \dspPlotFrame
  \dspCustomTicks{0.4 $0.4\pi$ 0.6 $0.6\pi$}
\end{dspPlot}
```



The following example shows how to repeat an arbitrary spectral shape over more than one period. First let's define (and plot) a non-trivial spectral shape making sure that the free variable x appears only at the beginning of the PostScript code:

```
% triangular shape:
\def\triFun{abs 0.25 sub 1 0.25 sub div }
% parabolic shape:
\def\parFun{abs 0.25 div dup mul 1 exch sub }
% composite shape (cutoff at 0.5pi)
\def\comFun{
  dup dup dup dup %
  -0.5 lt {pop pop pop pop 0} {  % zero for x < -0.5
     0.5 gt {pop pop pop 0 } {   % zero for x > 0.5
      -0.25 lt {pop \triFun } {  % triangle between
        0.25 gt {\triFun }       %    -.25 and -.5
          {\parFun}              % else parabola
        ifelse }%
      ifelse }%
    ifelse }%
  ifelse }

\begin{dspPlot}[xtype=freq,ylabel={$X(e^{j\omega})$}]{-1,1}{0,1.1}
  \dspFunc{x \comFun }
\end{dspPlot}
```

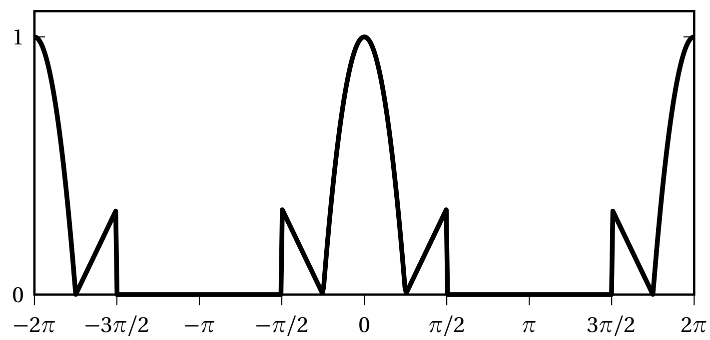


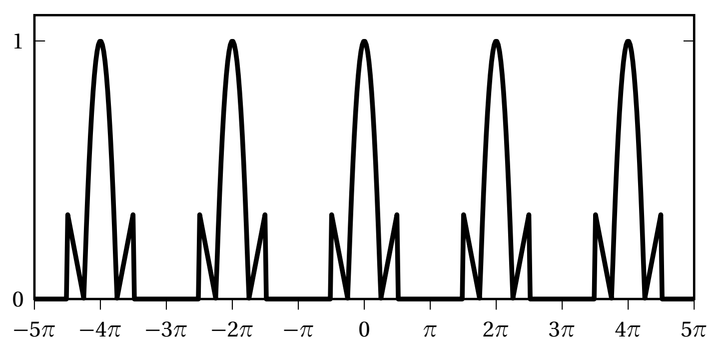
dspPeriodize

Now we can periodize the function using the `\dspPeriodize` macro; plotting multiple periods becomes as simple as changing the axis range:

```
\begin{dspPlot}[xtype=freq]{-2,2}{0,1.1}
  \dspFunc{x \dspPeriodize  \comFun }
\end{dspPlot}

\begin{dspPlot}[xtype=freq,xticks=1]{-5,5}{0,1.1}
  \dspFunc{x \dspPeriodize  \comFun }
\end{dspPlot}
```

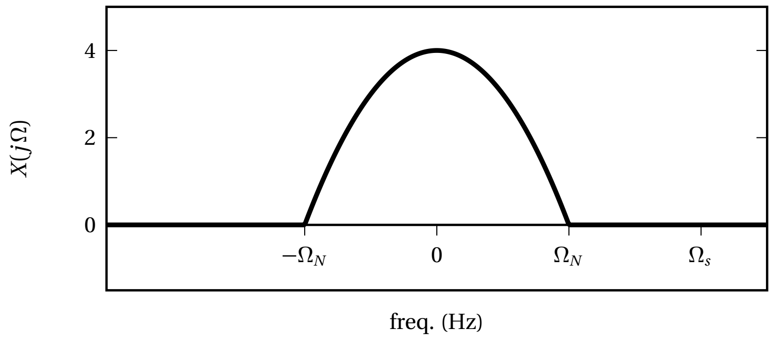




1.2.1 Analog Frequency Plots

To plot analog spectra, just set up the plot environment with the option `xticks=custom` and place your own frequency labels using `\dspCustomTicks` as in the example below:

```
\begin{dspPlot}[xtype=freq,xticks=custom,xlabel={freq. (Hz)},%
    yticks=2,ylabel={$X(j\Omega)$}]{-10,10}{-1,5}
  \dspFunc{x abs 4 gt {0} {x abs 2 div dup mul 4 exch sub} ifelse}
  \dspCustomTicks{-4 $-\Omega_N$ 0 $0$  4 $\Omega_N$ 8 $\Omega_s$}
\end{dspPlot}
```



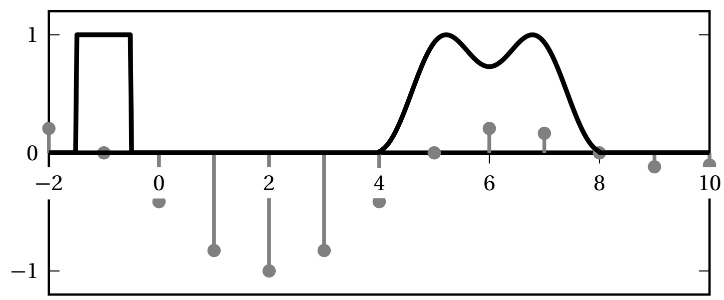
1.3 Common Functions

To facilitate the creation of plots that commonly occur in signal processing theory, the package `dspFunctions` provides the PostScript code for the following functions:

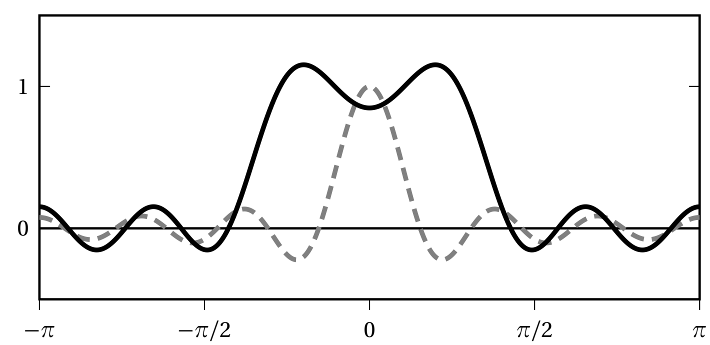
- `\dspRect{a}{b}` plots the function $\text{rect}((x-a)/b)$
- `\dspSinc{a}{b}` plots the function $\text{sinc}((x-a)/b)$
- `\dspSincN{a}{b}` plots the function $(1/b)\text{sinc}((x-a)/b)$
- `\dspTri{a}{b}` plots a triangle function centered in a and with support $2b$
- `\dspQuad{a}{b}` plots a quadratic function (inverted parabola) centered in a and with support $2b$
- `\dspPorkpie{a}{b}` plots a "porkpie hat" shape centered in a and with support $2b$
- `\dspRaisedCos{c}{r}` plots a raised cosine centered in zero with cutoff c and rolloff r
- `\dspSincS{a}{N}` plots the Fourier transform (DTFT) of a symmetric $2N+1$-tap rectangular signal $\sin((x-a)(2N+1)/2)/\sin((x-a)/2)$
- `\dspSincC{a}{N}` plots the DTFT magnitude of a causal N-tap rectangular signal $\sin((x-a)(N/2))/\sin((x-a)/2)$
- `\dspFIRI{a_0 a_1 ... a_{N-1}}` plots the magnitude response of a $(2N+1)$-tap Type-I FIR filter, zero centered; a_0 is the center tap

For instance:

```
\begin{dspPlot}[sidegap=0]{-2,10}{-1.2, 1.2}
  \dspFunc{x \dspRect{-1}{1}}
  \dspSignal[linecolor=gray]{x \dspSinc{2}{3} -1 mul}
  \dspFunc{x \dspPorkpie{6}{2}}
\end{dspPlot}
```




```
\begin{dspPlot}[xtype=freq,xout=true]{-1,1}{-0.5,1.5}
  \dspFunc[linecolor=gray,linestyle=dashed]{x \dspSincS{0}{6} 13 div}
  \dspFunc{x \dspFIRI{ 0.3501    0.2823    0.1252   -0.0215   -0.0876
     -0.0868    0.0374} }
\end{dspPlot}
```



2 Pole-Zero Plots

`dspPZPlot` Pole-zero plots are defined by the environment

\begin{dspPZPlot}[⟨*options*⟩]{⟨M⟩}
...
\end{dspPZPlot}

This plots a square section of the complex plane in which both axes span the $[-M, M]$ interval. Options for the plot are:

`width` = ⟨*dim*⟩ : width of the plot

`height` = ⟨*dim*⟩ : height of the plot. Normally, since the range is the same for both the real and the imaginary axis, width and height should be equal. You can therefore specify just one of them and the other will be automatically set. If you explicitly specify both, you will be able to obtain an asymmetric figure. By default, width and height are equal to `\dspH`.

`xticks` = `auto` | `none` | ⟨*d*⟩ : labeling of the real axis

`yticks` = `auto` | `none` | ⟨*d*⟩ : labeling of the imaginary axis. When the option specifies a numeric value ⟨*d*⟩, that will be the spacing between two consecutive ticks on the axis.

`cunits` = `true` | `false` : if true, labels the real and imaginary axis with "Re" and "Im" respectively.

`circle` = ⟨*r*⟩ : draws a circle centered in $z = 0$ with radius r; by default $r = 1$, so that the unit circle will be drawn; set to zero for no circle.

`clabel` = ⟨*label*⟩ : for a circle of radius r, places the selected label text at $z = r + j0$. By default the label is equal to the value of r.

`roc` = ⟨*r*⟩ : draws a *causal* region of convergence with radius r.

`antiroc` = ⟨*r*⟩ : draws an *anticausal* region of convergence with radius r.

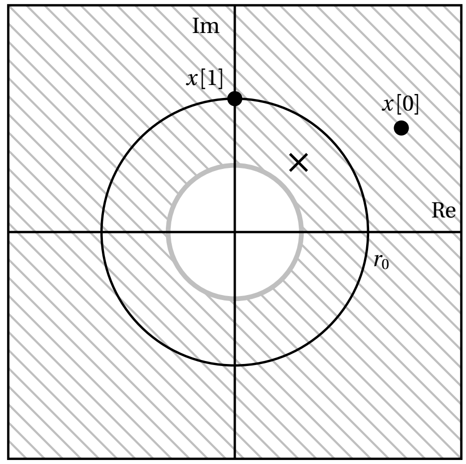
2.1 Poles and Zeros

dspPZ To plot a pole or a zero at $z = a + jb$ use

`\dspPZ[`⟨*options*⟩`]{`⟨*a, b*⟩`}`

which plots a pole by default; to plot a zero use the option `type=zero`. To associate a label to the point, use the option `label=`⟨*text*⟩; if ⟨*text*⟩ is `none` no label is printed; if ⟨*text*⟩ is `auto` (which is the default) the point's coordinates are printed; otherwise the specified text is printed. Finally, you can specify the position of the label using the option `lpos=`⟨*angle*⟩; by default, the angle's value is 45 degrees.

```
\begin{dspPZPlot}[clabel={$r_0$},roc=0.5]{1.5}
  \dspPZPoint[label=none]{0.5,0.5}
  \dspPZPoint[type=zero,label={$x[1]$},lpos=135]{0,1}
  \dspPZPoint[type=zero,label={$x[0]$},lpos=90]{1.25, 0.78}
\end{dspPZPlot}
```



3 Block Diagrams

`dspBlocks` Block diagrams rely heavily on psTricks' `psmatrix` environment, for which ample documentation is available. To set up a block diagram use the environment

```
\begin{dspBlocks}{⟨x⟩}{⟨y⟩}
...
\end{dspBlocks}
```

where ⟨x⟩ and ⟨y⟩ define the horizontal and vertical spacing of the blocks in the diagram. Predefined functional blocks are listed in the table below and they can be used anywhere a node is required. Nodes are labeled in top-left matrix notation, i.e. the topmost leftmost node is at coordinates $(0,0)$ and indices increase rightward and downward. Connections between nodes can be drawn using psTricks' standard primitive `\ncline`; the package defines the following shorthands:

`BDConnHNext`
- to connect with an arrow a node at (n,m) to its neighboring node at $(n,m+1)$ use `\BDConnHNext{`⟨*n*⟩`}{`⟨*m*⟩`}`

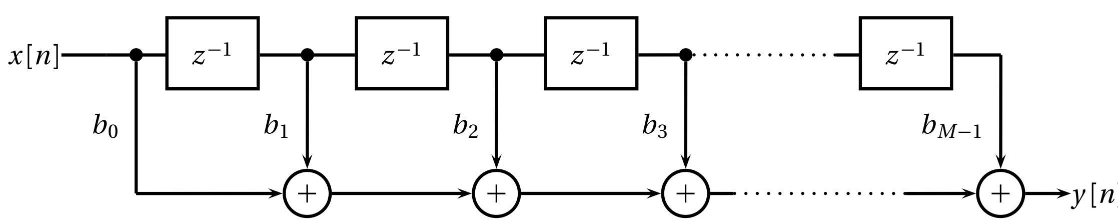
`BDConnH`
- to connect with an arrow a node at (n,m) to a node on the same row at (n,p) use `\BDConnH[`⟨*options*⟩`]{`⟨*n*⟩`}{`⟨*m*⟩`}{`⟨*p*⟩`}{`⟨*label*⟩`}`, which uses ⟨*options*⟩ as line options and ⟨*label*⟩ as the label for the connection

`BDConnV`
- to connect with an arrow a node at (n,m) to a node on the same column at (q,m) use `\BDConnV[`⟨*options*⟩`]{`⟨*n*⟩`}{`⟨*m*⟩`}{`⟨*q*⟩`}{`⟨*label*⟩`}`

function	**macro**	**example**
adder	`\BDadd`	$\oplus$
multiplier	`\BDmul`	$\otimes$
delay	`\BDdelay`	$\boxed{z^{-1}}$
delay by N	`\BDdelayN{`$\langle N \rangle$`}`	$\boxed{z^{-N}}$
filter	`\BDfilter{`$\langle$*label*$\rangle$`}`	$\boxed{H(z)}$
splitter	`\BDsplit`	$\bullet$
upsampler	`\BDupsmp{`$\langle N \rangle$`}`	$3\uparrow$
downsampler	`\BDdwsmp{`$\langle N \rangle$`}`	$3\downarrow$

Table 1: Block diagram macros

```
\begin{dspBlocks}{.3}{1}
% first row:
$x[n]$   &          &          & \BDsplit & \BDdelay &%
\BDsplit & \BDdelay & \BDsplit & \BDdelay & \BDsplit & \hspace{3em} & %
         & \BDdelay &  \\
%
% second row:
         &          &          &          &          &%
\BDadd   &          & \BDadd   &          & \BDadd   & \hspace{3em} & %
         &          & \BDadd & &  $y[n]$
%
% connections:
  \ncline{1,1}{1,3}
  \ncline{1,3}{1,5}
  \ncline{1,5}{1,7}
  \ncline{1,7}{1,9}
  \ncline{1,9}{1,10}
  \ncline[linestyle=dotted]{1,10}{1,12}
  \ncline{1,12}{1,13}
  \ncline{1,13}{1,14}
  \ncline{2,10}{2,11}
  \ncline[linestyle=dotted]{2,10}{2,13}
  \ncline{1,4}{2,4}\tlput{$b_0$}
  \BDConnH{2}{4}{6}{}
  \BDConnV{1}{6}{2}{$b_1$}
  \BDConnH{2}{6}{8}{}
  \BDConnV{1}{8}{2}{$b_2$}
  \BDConnH{2}{8}{10}{}
  \BDConnV{1}{10}{2}{$b_3$}
  \BDConnHNext{2}{13}
  \BDConnV{1}{14}{2}{$b_{M-1}$}
  \BDConnH{2}{14}{16}{}
\end{dspBlocks}
```



More complicated building blocks can be drawn using standard psTricksprimitives:

```
\begin{dspBlocks}{1.3}{0.4}
  $x(t)$
  &
  \raisebox{-1.6em}{\psframebox[linewidth=1.5pt]{%
    \psset{xunit=1em,yunit=1em}%
    \pspicture(-3,-2)(3,2)%
      \psline{->}(-2.8,-1)(2.8,-1)%
      \psline{->}(0,-1.8)(0,1.8)%
      \psline[linewidth=1.8pt](-1,-1)(-1,0.8)(1,0.8)(1,-1)%
    \endpspicture}}
  &
  &
  \raisebox{-1.4em}{\psframebox[linewidth=1.5pt]{%
    \psset{xunit=1em,yunit=1em,linewidth=1.8pt}%
    \pspicture(-3,-1.8)(2,1.8)%
      \psline(-2.8,0)(-1.6,0)(1.2,1.4)
      \psline(1.1,0)(1.8,0)
      \psarc[linewidth=1pt]{<-}(-1.6,0){2em}{-10}{55}
    \endpspicture}}
  &
  $x[n]$
%
  \psset{linewidth=1.5pt}
  \ncline{->}{1,1}{1,2}
  \ncline{1,2}{1,4}^{$x_{LP}(t)$}
  \ncline{->}{1,4}{1,5}
\end{dspBlocks}
```