DSPTricks
A Set of Macros for Digital Signal Processing Plots

Paolo Prandoni

The package DSPTricks is a set of \LaTeX{} macros for plotting the kind of graphs and figures that are usually employed in digital signal processing publications\(^1\); the package relies on PSTricks \([1, 2]\) to generate its graphic output.

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 Drawing Signals

Signal plots in DSPTricks are defined by a Cartesian grid enclosed by a box; there are three fundamental types of plots:

- discrete-time plots,
- continuous-time plots,
- frequency-domain plots;

discrete- and continuous-time plots can be mixed, whereas frequency-domain plots involve a re-labeling of the horizontal axis in trigonometric units.

1.1 The \texttt{dspPlot} environment

Data plots are defined by the \texttt{dspPlot} environment as

\begin{verbatim}
\begin{dspPlot} [...]
\end{dspPlot}
\end{verbatim}

\begin{verbatim}
\begin{dspPlot} [...]
\end{dspPlot}
\end{verbatim}

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:

- \texttt{width=\langle dim\rangle} : width of the plot (using any units)

\(^{1}\text{The original macros have been written by the author while working on the manuscript for [6].}\)
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 \texttt{\textbackslash dspW} and \texttt{\textbackslash dspH} lengths at the beginning of your document.

dspW, dspH

\texttt{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\(^2\). 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 \texttt{\textbackslash dspCustomTicks} macro.

sidegap = (gap) : extra space (in horizontal units) 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; use a value of zero to eliminate the side gap.

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

r1label = (label) : additional label for the vertical axis; placed outside the plot box on the right

Within a \texttt{\textbackslash 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 \texttt{xunit} and \texttt{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:

\texttt{\textbackslash 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 \texttt{\textbackslash dspClip} environment. See section 1.3 for an example.

\texttt{\textbackslash dspPlotFrame} • to redraw the framing box (useful to “smooth out” plots touching the frame) you can issue the command \texttt{\textbackslash dspPlotFrame}

\texttt{\textbackslash dspCustomTicks} • to draw arbitrarily placed ticks (and tick labels) on either axis, use\(^2\)

\(^2\)For digital frequency plots, \texttt{xticks} has a different meaning; see Section 1.5 for details.
\ DSPCustomTicks \{\langle options\rangle\} \{\langle pos label pos label ...\rangle\}

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

dspText

- place a text label anywhere in the plot using the axes coordinates:

\ DSPText(x, y) \{\langle label\rangle\}

1.2 Plotting Discrete-Time Signals

The following commands generate stem (or “lollipop”) plots; available options in the commands are all standards PSTricks options plus other specialized options when applicable:

dspTaps

- to plot a set of discrete time points use

\ DSPTaps \{\langle options\rangle\} \{\langle data\rangle\}

where \langle data\rangle is a list of space-separated index-value pairs (e.g., values pre-computed by an external numerical package). Allowed options are the generic PSTricks plot options.

dspTapsAt

- to plot a set of discrete time points use

\ DSPTapsAt \{\langle options\rangle\} \{\langle start\rangle\} \{\langle data\rangle\}

where \langle start\rangle is the initial index value and \langle data\rangle is a list of space-separated signal values. Allowed options are the generic PSTricks plot options.

dspTapsFile

- for large data sets, you can use

\ DSPTapsFile \{\langle options\rangle\} \{\langle fileName\rangle\}

where now \langle fileName\rangle 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 \{\langle options\rangle\} \{\langle PostScript code\rangle\}

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

\ xmin,\ xmax

- to perform a PostScript initialization sequence before evaluating the signal, use

\ DSPSignal10pt \{\langle options\rangle\} \{\langle init\rangle\} \{\langle PostScript code\rangle\}

where \langle init\rangle is a valid PostScript sequence (e.g. \texttt{\{/A [1 2 3] def} to initialize an array of data).
For example:

\begin{verbatim}
\begin{dspPlot}{-3, 22}{-1.2, 1.2}
  \% for my postscript interpreter rand_max is 0x7FFFDFFFFF
  \dspSignal[xmin-8]{rand 2147483647 div 0.5 sub 2 mul}
  \dspTaps[linecolor-red]{3 1 4 1 5 1}
  \dspTapsAt[linecolor-blue@60]{-2}{-0.5}
\end{dspPlot}
\end{verbatim}

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.

1.3 Plotting Continuous-Time Signals

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

- **\texttt{dspFunc}**
  - You can draw a continuous-time signal by using the command
    \begin{verbatim}
    \dspFunc[(options)]{(PostScript code)}
    \end{verbatim}
  - again, the PostScript code must use \texttt{x} as the independent variable; the range for \texttt{x} is the \texttt{(xmin)-(xmax)} interval and can be controlled for each signal independently via the \texttt{xmin} and \texttt{xmax} options.

- **\texttt{dspFuncData}**
  - To plot a smooth function obtained by interpolating a list of space separated time-value pairs use
    \begin{verbatim}
    \dspFuncData[(options)]{(data)}
    \end{verbatim}
  - the interpolation is performed by the PostScript interpreter and can be controlled if necessary by using the appropriate PSTricks options.

- **\texttt{dspFuncFile}**
  - For a continuous-time smooth interpolation of a pre-computed set of data points, use
    \begin{verbatim}
    \dspFuncFile[(options)]{(fileName)}
    \end{verbatim}
  - where \texttt{(fileName)} points to a text file containing the data points as a space-separated list of abscissae and ordinates.
To plot one or more Dirac deltas (symbolized by a vertical arrow) use
\texttt{\textbackslash 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 \texttt{\textbackslash dspClip} environment when plotting the hyperbola:

\begin{verbatim}
\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}
\end{verbatim}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{hyperbola_plot.png}
\caption{Plotting a hyperbola.}
\end{figure}

\subsection*{1.4 Plotting Discrete- and Continuous-Time Signals Together}

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

\begin{verbatim}
\begin{dspPlot}[xticks=10,yticks=0.2](-5,20){-0.4,1.2}
\def\sinx{x 0 eq 1 \{x RadToDeg \sin x div\} ifelse}
\dspSignal[xmax=10]{\sinx}
\dspFunc[linestyle=dashed,linewidth=0.5pt,xmax=10]{\sinx}
\end{dspPlot}
\end{verbatim}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{mixed_signals_plot.png}
\caption{Plotting discrete- and continuous-time signals together.}
\end{figure}

\footnote{Make sure not to leave any blank space in between the beginning and end of the \texttt{\textbackslash dspClip} environment, otherwise the axis labels may fall out of alignment.}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{mixed_signals_plot.png}
\caption{Plotting discrete- and continuous-time signals together.}
\end{figure}
1.5 Plotting Digital Spectra

Digital frequency 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 \(\pi\); 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 2\(\pi\)-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 2\(\pi\)-periodic; the `\[\text{dspPeriodize}\]` 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.

---

\[X(e^{j\omega}) = \sum_{n=-\infty}^{\infty} x[n]e^{-j\omega n}\]
\add x 180 mul cos 2 mul \lambda_mul sub div \)
\def\phase{\lambda_mul x 180 mul sin mul -1 mul 1 \lambda_mul %
\x 180 mul cos mul atan 180 div 3.1415 mul }
\begin{dspPlot}[xtype=.freq,xticks=3,yticks=0.2, %
ylabel=\{Square magnitude $|H(e^{i \omega})|^2$\}
{-1,1}{0,1.1}
\dspFunc{\text{mag} }
\end{dspPlot}
\begin{dspPlot}[xtype=.freq,xticks=3,yticks=custom, %
ylabel=\{Phase (radians)\}
{-1,1}{-1.57,1.57}
\dspFunc{xmax=0}{\text{phase} }
\dspFunc[xmin=0]{-\text{phase} -1 mul}
\dspCustomTicks[\text{axis-y}]{-1.57 \pi/2 0 0 1.57 \pi/2} \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:

```postscript
\begin{pspicture}(0,0)(8,8)
\psframe[fillstyle=vlines,]
  hatchcolor=lightgray,hatchangle=20,%
  linecolor=lightgray%
  (0,1)(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)
\dpsFunc[linecolor=red]{x 3.14 mul 0.5 mul 10 exp 1 add 1 exch div}
\dpsPlotFrame
\dpsCustomTicks(0.4 $0.4\pi$ 0.6 $0.6\pi$)
\end{pspicture}
```

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 pop 0} { % zero for x > 0.5
  -0.25 lt {pop \triFun} { % triangle between
   0.25 gt {\parFun} % -0.25 and -0.5
   {\parFun} % else parabola
  }%else }
}%ifelse }
%ifelse }
%ifelse }
\begin{pspicture}(0,0)(8,8)
\psframe[fillstyle=vlines,]
  hatchcolor=lightgray,hatchangle=20,%
  linecolor=lightgray%
  (0,1)(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)
\dpsFunc[linecolor=red]{x 3.14 mul 0.5 mul 10 exp 1 add 1 exch div}
\dpsPlotFrame
\dpsCustomTicks(0.4 $0.4\pi$ 0.6 $0.6\pi$)
\end{pspicture}
```

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

```latex
\begin{verbatim}
\begin{dspPlot}[xtype=\texttt{freq}]{-2,2}{0,1.1}
\texttt{\textbackslash dspFunc}(x \texttt{\textbackslash \textbackslash dspPeriodize} \texttt{\textbackslash comFun })
\end{dspPlot}
\end{verbatim}
```

Now we can periodize the function using the `\texttt{\textbackslash dspPeriodize}` macro; plotting multiple periods becomes as simple as changing the axis range:
1.6 Plotting Analog Spectra

To plot analog spectra, just set up a plot environment as you would to plot a continuous-time signal, then set the option xticks=custom and place your own frequency labels using \texttt{\textbackslash dspCustomTicks} as in the example below:

```latex
\begin{verbatim}
\begin{dspPlot}[xtype=freq,xticks=custom,zlabel=(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 \texttt{\$Omega\_N\$} 0 \texttt{\$O\_N\$} 4 \texttt{\$\Omega\_N\$} 8 \texttt{\$\Omega\_s\$}}
\end{dspPlot}
\end{verbatim}
```

![Plot of analog spectrum](image)

freq. (Hz)

1.7 Common Signal Shapes and Helper Functions

To facilitate the creation of plots that commonly occur in signal processing theory, the package DSPFunctions provides a PostScript implementation for the following set of functions; each macro acts on the free variable \(x\) in a plot command (see examples below).

**Basic Shapes:**

- \texttt{\textbackslash dspRect\{a\}\{b\}} computes a rectangular (box) function centered in \(a\) and with support \(2b\), i.e. \(\text{rect}(x-a)/b\) where

  \[
  \text{rect}(x) = \begin{cases} 
  1 & \text{if } x < 1/2 \\
  0 & \text{otherwise}
  \end{cases}
  \]

- \texttt{\textbackslash dspTri\{a\}\{b\}} computes a triangle function centered in \(a\) and with support \(2b\)

- \texttt{\textbackslash dspSinc\{a\}\{b\}} computes the scaled sinc function \(\text{sinc}(x-a)/b\), where

  \[
  \text{sinc}(x) = \frac{\sin(\pi x)}{\pi x}
  \]

- \texttt{\textbackslash dspQuad\{a\}\{b\}} computes a quadratic function (inverted parabola) centered in \(a\) and with support \(2b\)
• \texttt{\textbackslash{}dspExpDec\{a\}\{b\}} computes the decaying exponential response $b^{x-a}u[x-a]$

• \texttt{\textbackslash{}dspPorkpie\{a\}\{b\}} computes a “porkpie hat” shape centered in $a$ and with support $2b$

• \texttt{\textbackslash{}dspRaisedCos\{a\}\{b\}\{r\}} computes a raised cosine centered in $a$ with cutoff $b$ and rolloff $r$

**Discrete Fourier Transform:**

• \texttt{\textbackslash{}dspDFT\{x\}_{0}x_{1}\ldots x_{\{N-1\}}\}} computes the magnitude of the Discrete Fourier transform (DFT) of the provided data points. The input value should be an integer.

\[
X[k] = |\sum_{n=0}^{N-1} x[n] e^{j\frac{2\pi}{N}nk}|
\]

• \texttt{\textbackslash{}dspDFTRe\{x\}_{0}x_{1}\ldots x_{\{N-1\}}\}} computes the real part of the DFT

• \texttt{\textbackslash{}dspDFTIM\{x\}_{0}x_{1}\ldots x_{\{N-1\}}\}} computes the imaginary part of the DFT

**Notable DTFTs:**

• \texttt{\textbackslash{}dspSinCS\{a\}\{N\}} computes the Discrete-Time Fourier transform (DTFT) of a zero-centered, symmetric $2N+1$-tap rectangular signal:

\[
X(e^{j\omega}) = \frac{\sin(\omega(2N+1)/2)}{\sin(\omega/2)}
\]

The parameter $a$ can be used to shift the DTFT to the chosen center frequency.

• \texttt{\textbackslash{}dspSinCG\{a\}\{N\}} computes the DTFT magnitude of a causal $N$-tap rectangular signal:

\[
|X(e^{j\omega})| = \frac{\sin(\omega(N/2))}{\sin(\omega/2)}
\]

The parameter $a$ can be used to shift the DTFT to the chosen center frequency.

**Frequency Responses:**

• \texttt{\textbackslash{}dspFIR\{a\}_{0}a_{1}\ldots a_{\{N-1\}}\}} computes the (real-valued) frequency response of a zero centered $(2N+1)$-tap Type-I FIR filter with coefficients $a_{N-1}, a_{N-2}, \ldots, a_{1}, a_{0}, a_{1}, a_{2}, \ldots, a_{N-1}$.

The coefficient $a_0$ is the center tap and you need only specify the coefficients from $a_0$ to $a_{N-1}$.

• \texttt{\textbackslash{}dspFIR\{a\}_{0}a_{1}\ldots a_{\{N-1\}}\{b\}_{1}b_{2}\ldots b_{\{M-1\}}\}} computes the magnitude response of a generic digital filter defined by the constant-coefficient difference equation:

\[
y[n] = a_0 x[n] + a_{N-1} x[n-N+1] - b_1 y[n-1] - \ldots - b_{M-1} y[n-M+1]
\]

5Please note that the underlying implementation of the macro is not optimized; the computing time will be quadratic in the number of data points.
For instance:

```latex
\usepackage{dspFunctions}
\begin{dspPlot}[sidegap=1]{-2,10}{-1.2, 1.2}
\hold[red]
\dspFunc{x \ \text{dspRect(-1)(1)}}
\dspFunc{x \ \text{dspPorkpie(6)(2)}}
\dspSignal[linecolor=gray]{x \ \text{dspSinc(2)(3) \ -1 mul}}
\end{dspPlot}
```

![Plot of DFT magnitude]

We can generate a finite-length signal and plot its DFT magnitude as in this example, where the 32-point input signal is \( \cos(\frac{2\pi}{32} \cdot \frac{27}{5} n) \):

```latex
\begin{dspPlot}{{0, 31}\{0, 18}}
\dpsignal\text{opt}{/A [ 0 1 31 \{360 32 \text{ div} 5.4 \text{ mul} \text{ mul} \text{ cos} \} \text{ for }]}\text{ def}
{x \ \text{dspDFTMag}{ \ A \ \text{aload pop } } }
\end{dspPlot}
```

![Plot of magnitude for a finite-length signal]

The magnitude of simple FIR and IIR filter can be graphed easily like so:

```latex
\begin{dspPlot}[xtype(freq,xout=true)\{-1,1\}\{-0.5,1.5}\]
\dpsignal[\text{linecolor=gray},\text{linestyle=dashed}]{x \ \text{dpsinc(0)(6) 13 div}}
\dpsignal{x \ \text{dpsFIRI}\{\ 0.3501 \ 0.2823 \ 0.1252 \ -0.0215 \ -0.0876 \\
-0.0866 \ 0.0374 \}\}
\end{dspPlot}
```

![Plot of FIR and IIR filter magnitude]
2 Drawing Regions of Convergence, Poles and Zeros

dspPZPlot Pole-zero plots are defined by the environment

\begin{dspPZPlot}[\langle options\rangle \{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 = \langle dim \rangle : width of the plot
\begin{align*}
\text{height} &= \langle \text{dim} \rangle : \text{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 1.}
\end{align*}

\begin{align*}
\text{xticks} &= \text{auto} \mid \text{none} \mid \langle d \rangle : \text{labeling of the real axis}\nonumber
\text{yticks} &= \text{auto} \mid \text{none} \mid \langle d \rangle : \text{labeling of the imaginary axis. When the option specifies a numeric value \langle d \rangle, that will be the spacing between two consecutive ticks on the axis.}
\end{align*}

\begin{align*}
\text{cunits} &= \text{true} \mid \text{false} : \text{if true, labels the real and imaginary axis with “Re” and “Im” respectively.}
\end{align*}

\begin{align*}
\text{circle} &= \langle r \rangle : \text{draws a circle centered in } z = 0 \text{ with radius } r; \text{ by default } r = 1, \text{ so that the unit circle will be drawn; set to zero for no circle.}
\end{align*}

\begin{align*}
\text{clabel} &= \langle \text{label} \rangle : \text{for a circle of radius } r, \text{ places the selected label text at } z = r + j 0. \text{ By default the label is equal to the value of } r.
\end{align*}

\begin{align*}
\text{roc} &= \langle r \rangle : \text{draws a causal region of convergence with radius } r.
\text{antiroc} &= \langle r \rangle : \text{draws an anticausal region of convergence with radius } r.
\end{align*}

\subsection{2.1 Poles and Zeros}

dspPZ To plot a pole or a zero at \( z = a + jb \) use

\begin{align*}
\text{\texttt{\textbackslash dspPZ}} \langle \text{options} \rangle \{\langle a, b \rangle\}
\end{align*}

which plots a pole by default; to plot a zero use the option \texttt{type=zero}. To associate a label to the point, use the option \texttt{label=(text)}; if \texttt{(text)} is none no label is printed; if \texttt{(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 \texttt{1pos=(angle)}; by default, the angle’s value is 45 degrees.

\begin{verbatim}
\begin{dspPZlot}\clabel={\$r_0\$},roc=0.5\end{dspPZlot}
\dspPZPoint[\text{label=none}]{0.5,0.5}
\dspPZPoint[type=zero,\text{label=\$x_1\$},1pos=135]{0,1}
\dspPZPoint[type=zero,\text{label=\$x_0\$},1pos=90]{1.25,0.75}
\end{verbatim}
3 Block Diagrams

\texttt{dsplibks} Block diagrams rely heavily on PSTricks’ \texttt{psmatrix} environment, for which ample documentation is available. To set up a block diagram use the environment

\begin{verbatim}
\begin{dsplibks}\{x\}\{y\}
...
\end{dsplibks}
\end{verbatim}

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 top-most leftmost node is at coordinates \((1,1)\) and indices increase rightward and downward. Connections between nodes can be drawn using PSTricks’ standard primitive \texttt{ncline}; the package defines the following shorthands:

- \texttt{BDConnHNext} to connect with an arrow a node at \((n,m)\) to its neighboring node at \((n,m+1)\) use \texttt{BDConnHNext \{\textit{options}\} \{\textit{node}\}\{\textit{node}\}}

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

- \texttt{BDConnV} to connect with an arrow a node at \((n,m)\) to a node on the same column at \((q,m)\) use \texttt{BDConnV[\{\textit{options}\} \{\textit{node}\}\{\textit{node}\}\{\textit{node}\}\{\textit{label}\}]}
<table>
<thead>
<tr>
<th>function</th>
<th>macro</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>nodes</td>
<td>\BDsplit</td>
<td>( \cdot )</td>
</tr>
<tr>
<td></td>
<td>\BDadd</td>
<td>( + )</td>
</tr>
<tr>
<td></td>
<td>\BDmul</td>
<td>( \times )</td>
</tr>
<tr>
<td>delays</td>
<td>\BDdelay</td>
<td>( z^{-1} )</td>
</tr>
<tr>
<td></td>
<td>\BDdelayN{(N)}</td>
<td>( z^{-N} )</td>
</tr>
<tr>
<td>filters</td>
<td>\BDfilter{(label)}</td>
<td>( H(z) )</td>
</tr>
<tr>
<td></td>
<td>\BDfilterMulti{(labels)}</td>
<td>multiple lines</td>
</tr>
<tr>
<td></td>
<td>\BDlowpass</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(use \ to separate lines)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>\BDlowpass</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(you can specify the size of the block, eg \BDlowpass[2em])</td>
<td></td>
</tr>
<tr>
<td>function</td>
<td>macro</td>
<td>output</td>
</tr>
<tr>
<td>------------</td>
<td>---------------------</td>
<td>--------</td>
</tr>
<tr>
<td>sampler</td>
<td>\BDsampler</td>
<td><img src="image" alt="sampler" /></td>
</tr>
<tr>
<td></td>
<td>\BDsamplerFramed</td>
<td><img src="image" alt="samplerFramed" /></td>
</tr>
<tr>
<td></td>
<td>(you can specify the size of the block, eg \BDsamplerFramed[2em])</td>
<td></td>
</tr>
<tr>
<td>interpolator</td>
<td>\BDSinc</td>
<td><img src="image" alt="interpolator" /></td>
</tr>
<tr>
<td></td>
<td>(you can specify the size of the block, eg \BDSinc[2em])</td>
<td></td>
</tr>
<tr>
<td>upsampler</td>
<td>\BDupsmp{(N)}</td>
<td><img src="image" alt="upsampler" /></td>
</tr>
<tr>
<td>downsample</td>
<td>\BDdownspmp{(N)}</td>
<td><img src="image" alt="downsample" /></td>
</tr>
</tbody>
</table>
If you need to label nodes for complex connections, you may need to revert to the actual code for the node element, eg:
\begin{dspBlocks}(0.8)(0)
& \hat{a}[n] \quad \text{Slicer} \quad \text{Descrambler} \quad \ldots 01100 \\
& b[n] \quad 01010... \\
\end{dspBlocks}

\[ \hat{s}(t) \quad \hat{s}[n] \quad \hat{c}[n] \quad e^{-j\omega n} \]

\[ \hat{c}[n] \quad \hat{b}[n] \quad H(z) \quad j \]

\[ j \quad e^{-j\omega n} \quad \hat{b}[n] \quad 01010... \]
References


