3 Analyzing Signals

One of the major tasks of signal processing is the determination of characteristics of a signal, in both the time and frequency domains. Signals and Systems provides a number of functions to assist in this process. Two general functions to report on a variety of signal characteristics are included (one for continuous and one for discrete signals). (Note that all capabilities of these omnibus functions can be accessed individually in separate functions throughout the packages.) Also, a variety of plotting functions can be used to generate standard graphic representations of signals.

■ 3.1 Reporting Signal Characteristics

The signal analysis functions `ASPAnalyze` and `DSPAnalyze` create reports of various signal characteristics. For `ASPAnalyze`, the components include a plot with respect to the independent variable, the Laplace transform with its region of convergence, a pole-zero plot, and the frequency response with a magnitude-phase plot. `DSPAnalyze` generates the Z transform in place of the Laplace transform, and uses the discrete-time Fourier transform in place of a Fourier transform. You can choose which of these elements are to be displayed, and the order in which they are returned.

`ASPAnalyze[`*signal*`,` `{`*var*`,` *min*`,` *max*`}]`	analyze *signal*, assuming independent variable *var* is continuous and ranges from *min* to *max*
`ASPAnalyze[`*signal*`,` `{`var_1`,` min_1`,` max_1`},` `{`var_2`,` min_2`,` max_2`}]`	analyze the two-dimensional *signal* with independent continuous variables var_1 and var_2
`DSPAnalyze[`*signal*`,` `{`*var*`,` *min*`,` *max*`}]`	analyze *signal*, assuming independent variable *var* is discrete and ranges from *min* to *max*
`DSPAnalyze[`*signal*`,` `{`var_1`,` min_1`,` max_1`},` `{`var_2`,` min_2`,` max_2`}]`	analyze the two-dimensional *signal* with independent discrete variables var_1 and var_2

Analyzing signals.

The routines handle only one- and two-dimensional signals, primarily because visualization techniques for higher-dimensional signals are not well defined. The analysis proceeds with the named independent variable(s), while any other parameters in the signal expression are given the value 1 while graphing. Explicit alternate values to be used for graphing can be specified by the `ParameterValues` option. This option takes a list of transformation rules to be applied to the signal before it is plotted.

- First, verify that the signal processing functions are available.

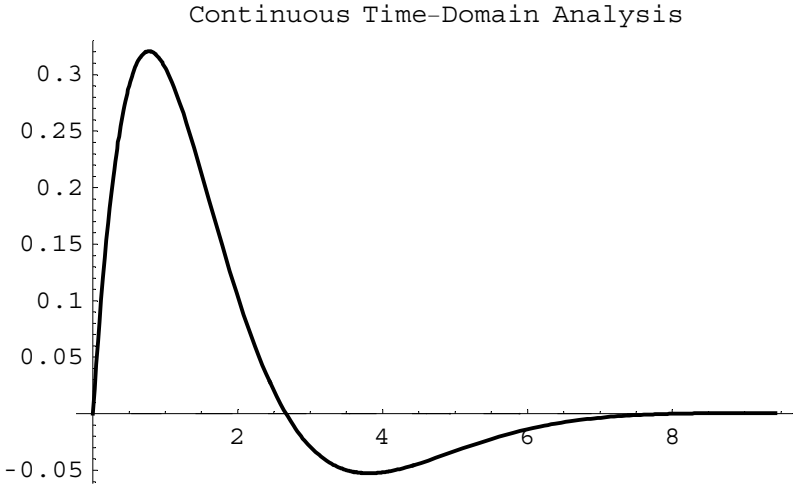
```
In[1]:= Needs["SignalProcessing`"]
```

- Signals are analyzed with a single command. Note that a value is specified for the parameter `a` during graphing.

```
In[2]:= ASPAnalyze[ t Exp[-a t] Cos[3 Pi t/16] UnitStep[t],
          {t, 0, 3 Pi},
          ParameterValues -> {a -> 1}
        ]
```

```
While creating all plots in this analysis,
    the following substitutions will be applied
    to give numeric values to various symbols:

{a → 1}
```




```
Given the input function:
```

$$e^{-a\,t}\, t\, \text{Cos}\left[\frac{3\,\pi\,t}{16}\right]\, \text{UnitStep}[t]$$

The Laplace transform is:

$$\frac{-\frac{9\pi^2}{256} + (a+s)^2}{\left(\frac{9\pi^2}{256} + (a+s)^2\right)^2}$$

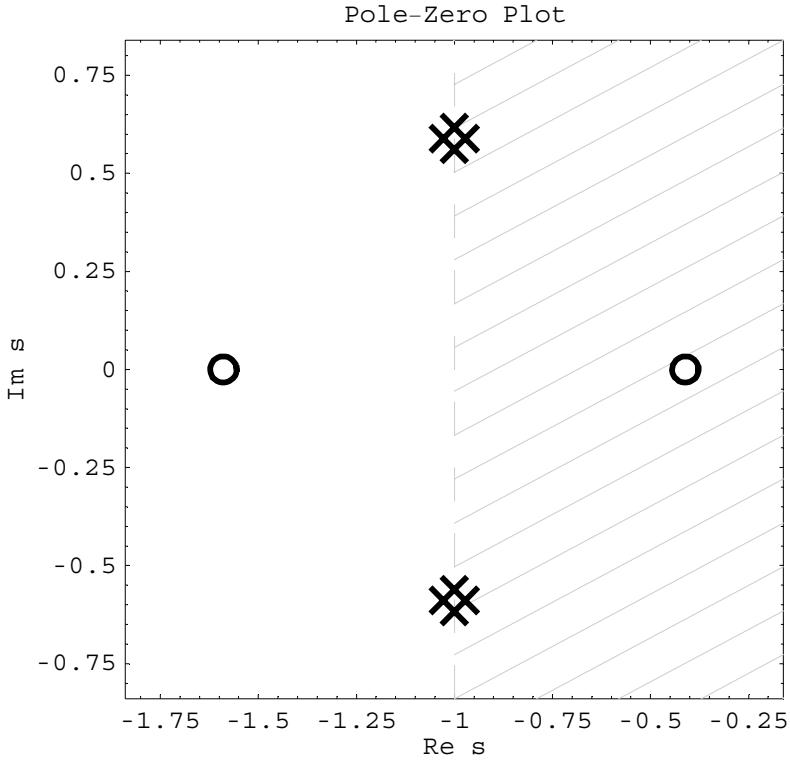
The region of convergence is:

$-\text{Re}[a] < \text{Re}(s) < \infty$

The system is stable if $\text{Re}[a] > 0$

The zeroes are: $\{-1.58905, -0.410951\}$

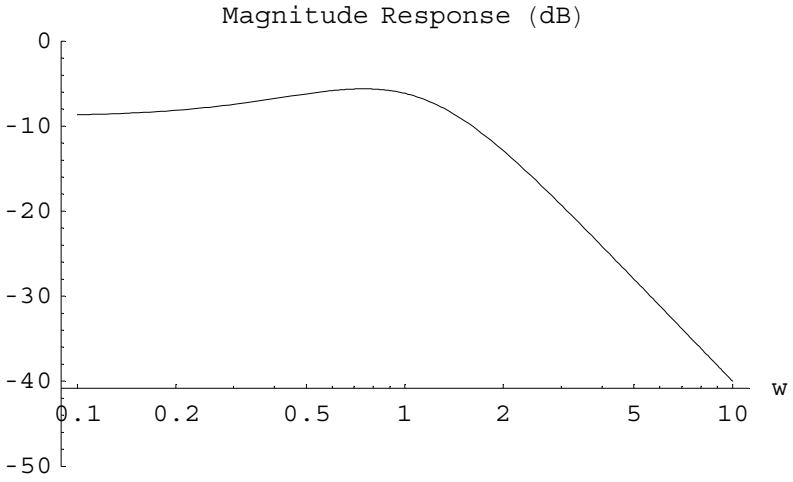
The poles are:
$\{-1. - 0.589049\,i, -1. - 0.589049\,i, -1. + 0.589049\,i, -1. + 0.589049\,i\}$

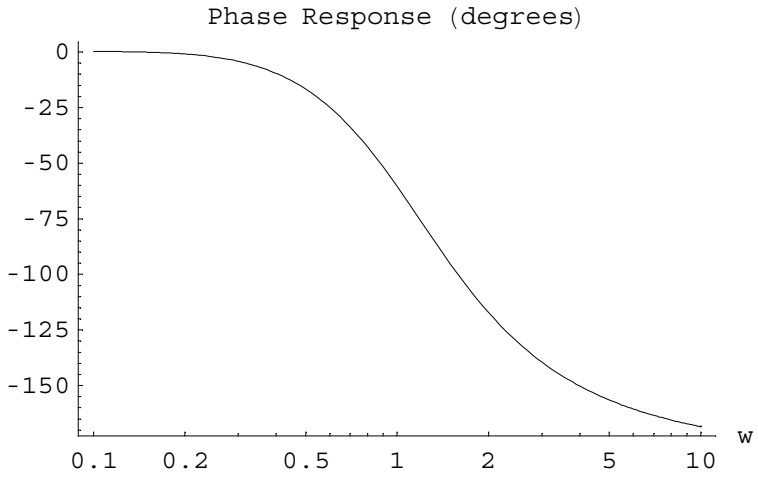


Since the sequence is stable with the default
values assumed for any of the parameters, the
frequency response can be computed
directly from the Laplace transform.

The frequency response as determined by
the Laplace transform is:

$$\frac{-\frac{9\pi^2}{256} + (a + i\,w)^2}{\left(\frac{9\pi^2}{256} + (a + i\,w)^2\right)^2}$$





Out[2]=

$$\left\{\text{LaplaceTransformData}\left[\frac{-\frac{9\pi^2}{256} + (a + s)^2}{\left(\frac{9\pi^2}{256} + (a + s)^2\right)^2}, \text{RegionOfConvergence}[-\text{Re}[a], \infty], \text{TransformVariables}[s]\right], \text{FourierTransformData}\left[\frac{-\frac{9\pi^2}{256} + (a + i\,w)^2}{\left(\frac{9\pi^2}{256} + (a + i\,w)^2\right)^2}, \text{TransformVariables}[w]\right]\right\}$$

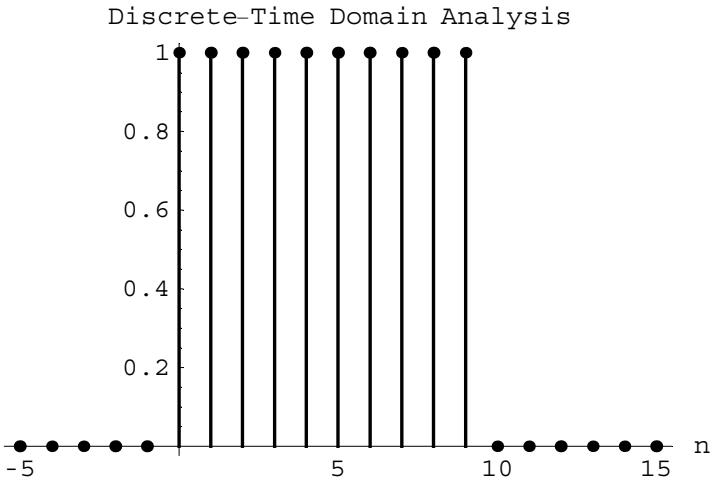
options name	*default value*	
AnalysisReport	Automatic	specify reported characteristics
AnalysisOutput	Automatic	specify which of the outputs are to be returned, and their order
ParameterValues	Automatic	list of transformation rules to set parameter values for graphing
SignalPlotOptions	{}	list of options for the signal plotting function
TransformOptions	{}	list of options for the Laplace or Z transform
PoleZeroPlotOptions	{}	list of options for the pole-zero plot
FrequencyResponseOptions	{}	list of options for the Fourier transform
MagnitudePhasePlotOptions	{}	list of options for the magnitude-phase plot

Options specific to the analysis function.

Important options include AnalysisReport and AnalysisOutput. The AnalysisReport option accepts a list of tags that determine which components of the report will be generated, though the order in which they are generated cannot be specified, due to dependencies among the computations. These include SignalPlot, Transform, PoleZeroPlot, FrequencyResponse, and MagnitudePhasePlot. Some of these may not be generated when specified if the computation cannot be performed (*e.g.*, when the transform cannot be found). The AnalysisOutput option determines which objects will be returned. It accepts the same tags as AnalysisReport. Null objects will be returned for items that could not be computed.

- First, a straightforward analysis of a discrete-time pulse is generated.

```
In[3]:= DSPAnalyze[DiscretePulse[10, n], {n, -5, 15}]
```



Given the input function:

$\text{DiscretePulse}_{10}[n]$

The z-transform is:

$$\frac{\left(1-\frac{1}{z^{10}}\right) z}{-1+z}$$

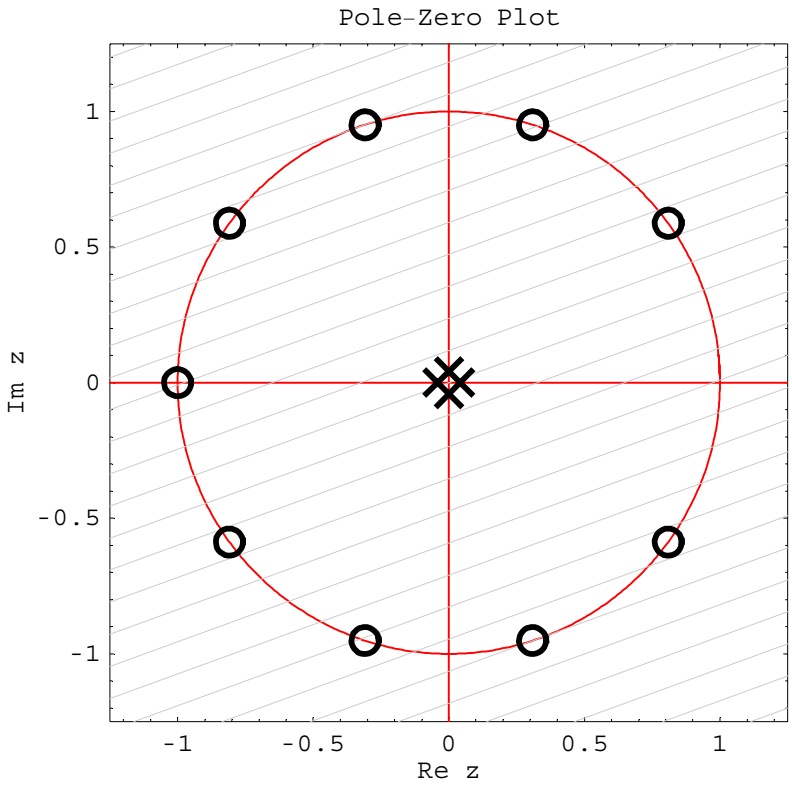
The region of convergence is:

$0 < |z| < \infty$

The system is stable.

The zeroes are: $\{-1.,\ -0.809017-0.587785\,i,\ 0.809017+0.587785\,i,\ -0.309017-0.951057\,i,\ 0.309017+0.951057\,i,\ 0.309017-0.951057\,i,\ -0.309017+0.951057\,i,\ 0.809017-0.587785\,i,\ -0.809017+0.587785\,i\}$

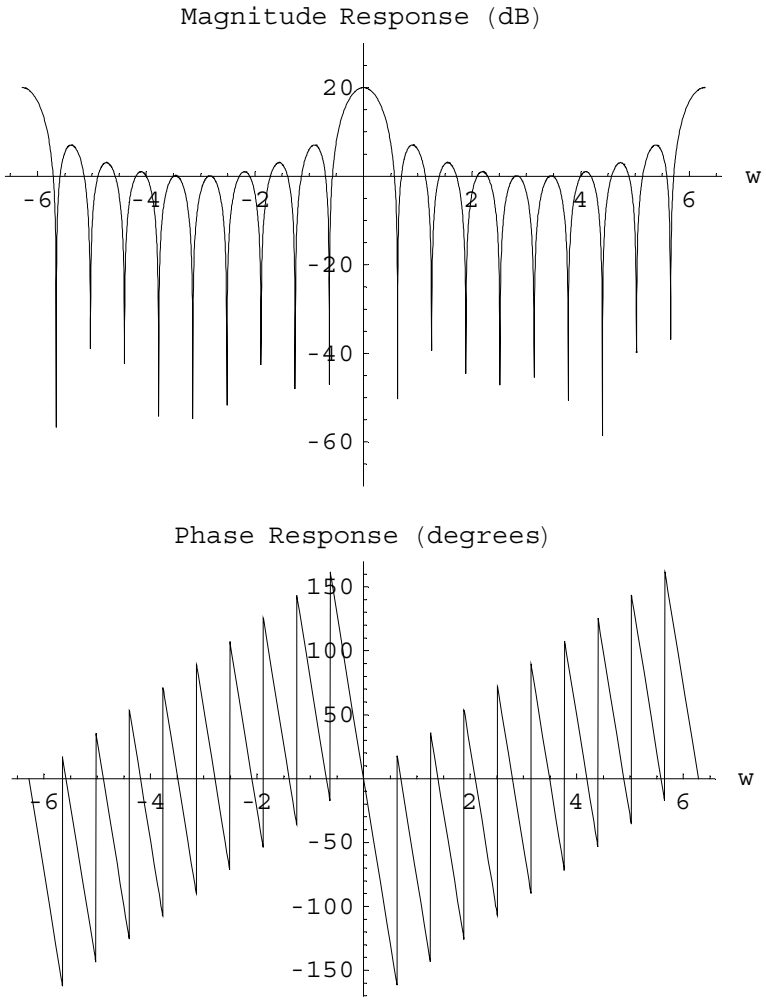
The poles are: $\{0.,\ 0.,\ 0.,\ 0.,\ 0.,\ 0.,\ 0.,\ 0.,\ 0.\}$



Since the sequence is stable with the default values assumed for any of the parameters, the frequency response can be computed directly from the z-transform.

The frequency response is:

$$10\, e^{-\frac{9 i w}{2}}\, \text{DirichletSinc}_{10}[w]$$




```
Out[3]= {ZTransformData[ (1 - 1/z^10) z / (-1 + z),
    RegionOfConvergence[0, ∞], TransformVariables[z]],
  DTFTData[10 E^(-9 i w/2) DirichletSinc_10[w], TransformVariables[w]]}
```

The transform routines will attempt to use the `Global`` variables `z` and `w` in the returned Z and Laplace transforms (`z1`, `z2`, `w1`, and `w2` in two-dimensional transforms). If the user has already assigned values to these symbols, unique symbols of the form `z$`*n*, `w$`*n*, etc., will be used instead.

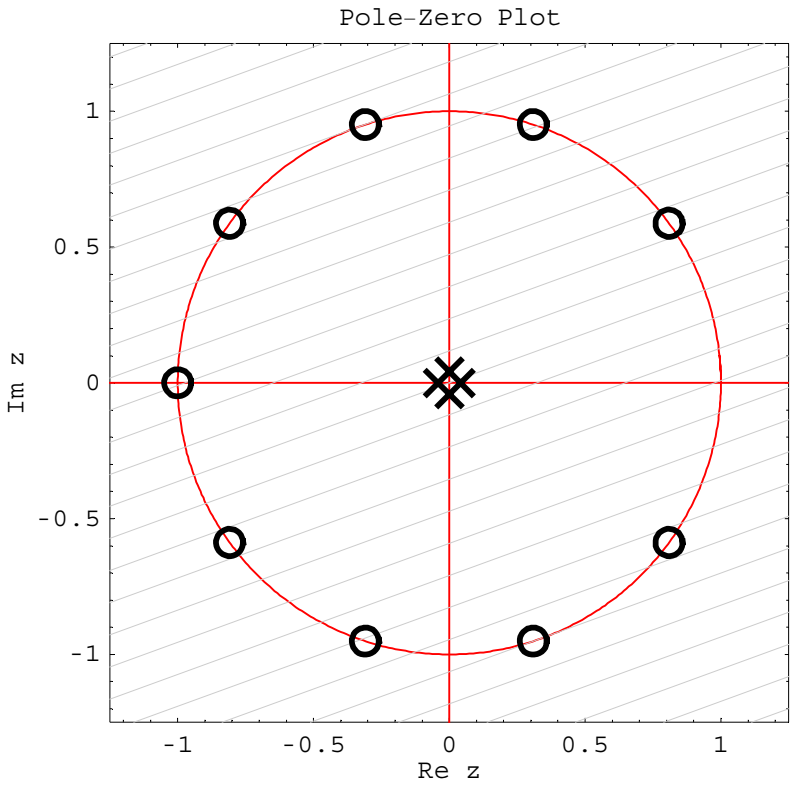
- Set a value for w (a variable used by the analysis routines).

```
In[4]:= w = 3

Out[4]= 3
```

- These options cause only the pole-zero plot to be displayed, and the discrete-time Fourier transform to be returned with the plot. An option is passed to the plotting function to suppress the printing of the list of poles and zeros.

```
In[5]:= DSPAnalyze[DiscretePulse[10, n], {n, -5, 15},
          AnalysisReport -> {PoleZeroPlot},
          AnalysisOutput -> {FrequencyResponse, PoleZeroPlot},
          PoleZeroPlotOptions -> {Justification -> False}
        ]
```




```
DSPAnalyze::hasval :
 The variable w has value so it could not be used
   as a transform variable. w$1 was used instead.
```

```
Out[5]= {DTFTData[10 e^(-9 i w$1/2) DirichletSinc10[w$1], TransformVariables[w$1]],
          - Graphics -}
```

- Clear the value that was set before the last computation.

```
In[6]:= Clear[w]
```

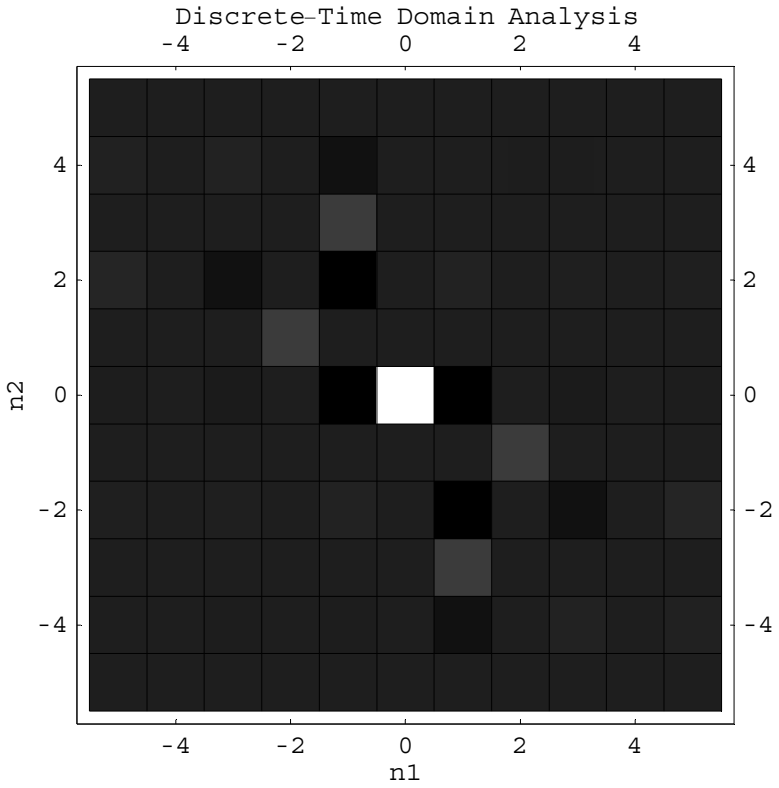
- This sets up a signal by first determining a passband in the frequency domain, and then transforming to the time domain.

```
In[7]:= (lowpass2Dtile =
            ContinuousPulse[Pi, w1 + Pi/2]*
                ContinuousPulse[Pi, w2 + Pi/2];
         lowpass2D =
            InverseDiscreteTimeFourierTransform[
                lowpass2Dtile,
                {w1, w2},
                {n1, n2}
            ]
        )
```

$$\text{Out[7]= } \frac{1}{4}\,\text{Sinc}\left[\frac{n1\,\pi}{2}\right]\,\text{Sinc}\left[\frac{n2\,\pi}{2}\right]$$

- Here is an analysis of a downsampled version of the two-dimensional signal.

```
In[8]:= DSPAnalyze[
           Downsample[{{2,1},{1,3}}, {n1, n2}][lowpass2D],
        {n1, -5, 5}, {n2, -5, 5}]
```



Given the input function:

$$\left(\text{Downsample}_{\substack{2 \\ 1}}{}_{\substack{1 \\ 3}}{}_{\substack{\text{n1} \\ \text{n2}}}\right)\left[\frac{1}{4}\,\text{Sinc}\left[\frac{\text{n1}\,\pi}{2}\right]\text{Sinc}\left[\frac{\text{n2}\,\pi}{2}\right]\right]$$

The z-transform cannot be determined.

Unable to generate a pole-zero plot
without a valid z-transform.

Even though the stability of the sequence
is not known given the assumed default values for
any of the parameters, the frequency response will
still be plotted.

The frequency response is:

$$\frac{1}{5}\left(\text{Periodic}_{\substack{2\pi & 0 & | & w1\\ 0 & 2\pi & | & w2}}\right)\Big[$$

$$\text{ContinuousPulse}_{\frac{5\pi}{2}}\Big[\frac{5\pi}{4}-\frac{w1}{2}+w2\Big]\,\text{ContinuousPulse}_{5\pi}\Big[\frac{5\pi}{2}-3\,w1+w2\Big]+$$

$$\text{ContinuousPulse}_{\frac{5\pi}{2}}\Big[-\frac{7\pi}{4}-\frac{w1}{2}+w2\Big]$$

$$\text{ContinuousPulse}_{5\pi}\Big[\frac{9\pi}{2}-3\,w1+w2\Big]+$$

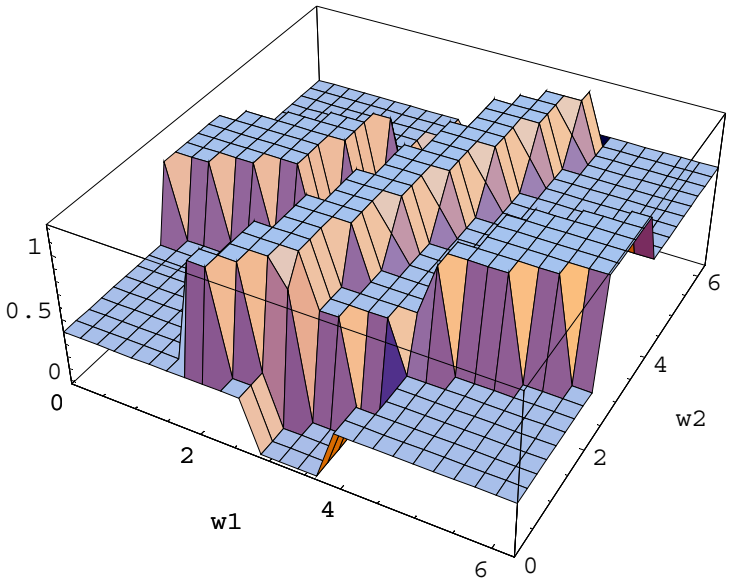
$$\text{ContinuousPulse}_{\frac{5\pi}{2}}\Big[\frac{1}{4}\,(\pi-2\,w1)+w2\Big]\,\text{ContinuousPulse}_{5\pi}\Big[$$

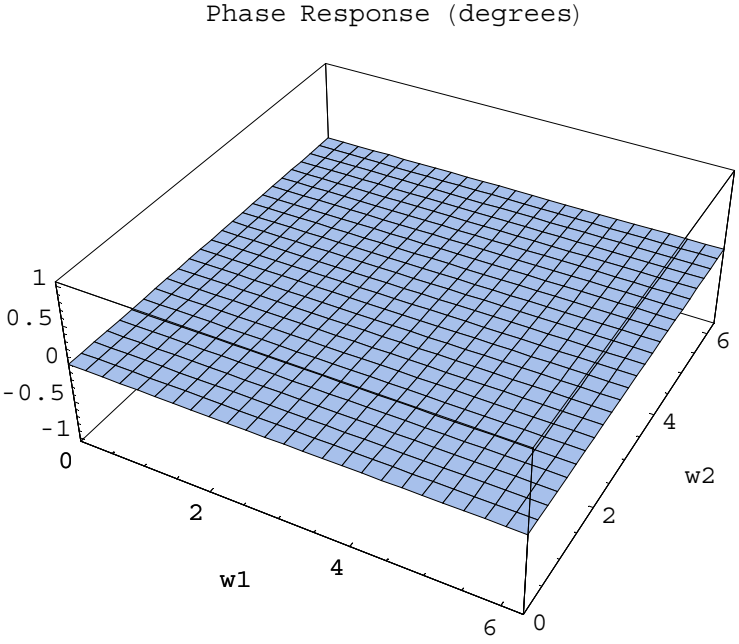
$$\frac{13\pi}{2}-3\,w1+w2\Big]+\text{ContinuousPulse}_{\frac{5\pi}{2}}\Big[-\frac{11\pi}{4}-\frac{w1}{2}+w2\Big]$$

$$\text{ContinuousPulse}_{5\pi}\Big[\frac{17\pi}{2}-3\,w1+w2\Big]+\text{ContinuousPulse}_{\frac{5\pi}{2}}\Big[$$

$$-\frac{3\pi}{4}-\frac{w1}{2}+w2\Big]\,\text{ContinuousPulse}_{5\pi}\Big[\frac{21\pi}{2}-3\,w1+w2\Big]\Big]$$

Magnitude Response





Out[8]= $\{$-Incomplete z-Transform-,

$$\text{DTFTData}\left[\frac{1}{5}\left(\text{Periodic}_{\substack{2\pi & 0 & | & w1\\ 0 & 2\pi & | & w2}}\right)\left[\text{ContinuousPulse}_{\frac{5\pi}{2}}\left[\frac{5\pi}{4}-\frac{w1}{2}+w2\right]\right.\right.$$
$$\text{ContinuousPulse}_{5\pi}\left[\frac{5\pi}{2}-3\,w1+w2\right]+\text{ContinuousPulse}_{\frac{5\pi}{2}}\left[-\frac{7\pi}{4}-\frac{w1}{2}+w2\right]\text{ContinuousPulse}_{5\pi}\left[\frac{9\pi}{2}-3\,w1+w2\right]+$$
$$\text{ContinuousPulse}_{\frac{5\pi}{2}}\left[\frac{1}{4}(\pi-2\,w1)+w2\right]\text{ContinuousPulse}_{5\pi}\left[\frac{13\pi}{2}-3\,w1+w2\right]+\text{ContinuousPulse}_{\frac{5\pi}{2}}\left[-\frac{11\pi}{4}-\frac{w1}{2}+w2\right]$$
$$\text{ContinuousPulse}_{5\pi}\left[\frac{17\pi}{2}-3\,w1+w2\right]+\text{ContinuousPulse}_{\frac{5\pi}{2}}\left[-\frac{3\pi}{4}-\frac{w1}{2}+w2\right]\text{ContinuousPulse}_{5\pi}\left[\frac{21\pi}{2}-3\,w1+w2\right]\Big],$$
$$\left.\text{TransformVariables}[\{w1, w2\}]\right]\}$$

■ 3.2 Plotting Signals

In signal processing, it is useful to get a feel for signal behavior by plotting the signal. A variety of visualization techniques can be found in Signals and Systems. Extended versions of a number of standard *Mathematica* graphics functions have been created by adding the prefix `Signal` to the function names and are used for basic plots of time (or spatial) response. Magnitude-phase plots (and the variant called the Bode plot) are used for frequency response analysis, while pole-zero, root-locus, and Nyquist plots allow various signal characteristics, such as stability, to be examined.

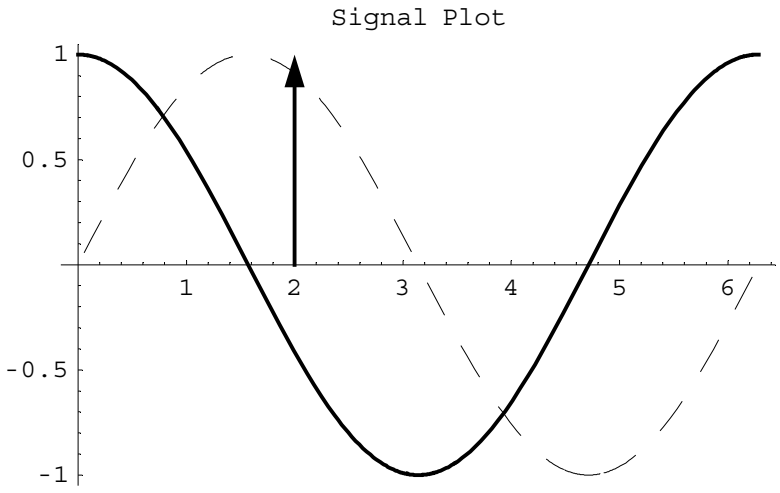
`SignalPlot[` *fun*`, {`*t*`,` *start*`,` *end*`}]`	plot a continuous signal with one independent variable ranging from *start* to *end*
`SignalPlot3D[` *fun*`, {`t_1`,` s_1`,` e_1`}, {`t_2`,` s_2`,` e_2`}]`	plot a continuous signal with two independent variables
`DiscreteSignalPlot[` *fun*`, {`*n*`,` *start*`,` *end*`}]`	plot a discrete signal with one independent variable ranging from *start* to *end*
`DiscreteSignalPlot3D[` *fun*`, {`n_1`,` s_1`,` e_1`}, {`n_2`,` s_2`,` e_2`}]`	plot a discrete signal with two independent variables as a surface
`DiscreteSignalDensityPlot[` *fun*`, {`n_1`,` s_1`,` e_1`}, {`n_2`,` s_2`,` e_2`}]`	plot a discrete signal with two independent variables as a density plot

Signal plotting functions modeled after standard *Mathematica* functions.

The common functions for displaying the time (or spatial) response of a signal are enhanced versions of traditional *Mathematica* plotting functions. The functions for continuous signals add the ability to handle complex values and impulse functions, while those for discrete signals generate a "fencepost" plot (also known as "lollipop" or "stem" plots) in one dimension, and a density plot in two dimensions.

- A complex-valued signal with a singularity function can be displayed by `SignalPlot`.

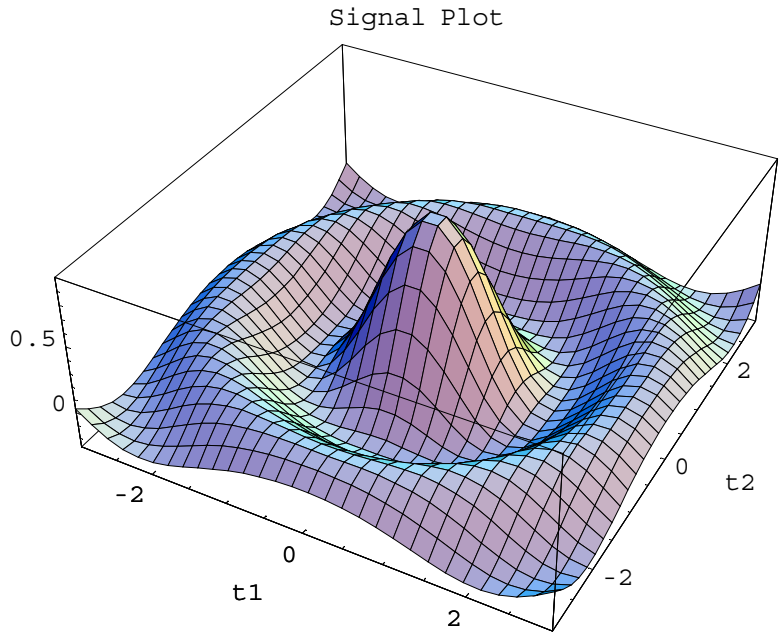
```
In[9]:= SignalPlot[
          Exp[I t] + DiracDelta[t - 2],
          {t, 0, 2 Pi}
        ]
```




```
Out[9]= - Graphics -
```

- Here is a plot of a signal defined over a two-dimensional support.

```
In[10]:= SignalPlot3D[
           DirichletSinc[6, Sqrt[t1^2 + t2^2]],
           {t1, -3, 3}, {t2, -3, 3},
           PlotPoints -> 30
         ]
```

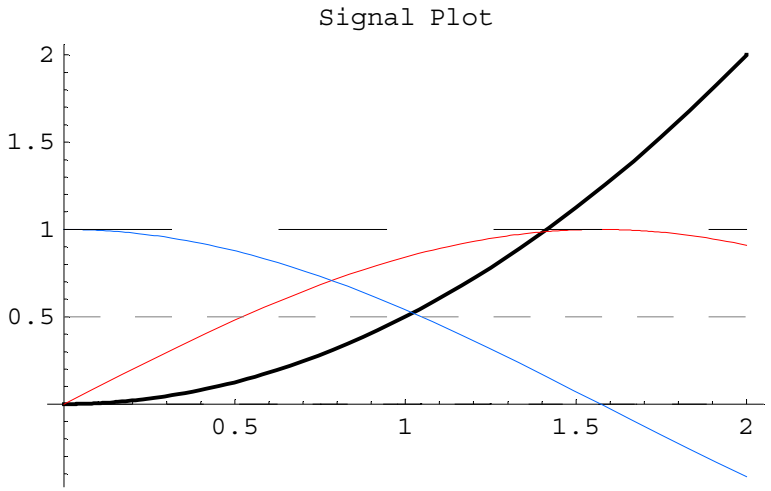



```
Out[10]= - SurfaceGraphics -
```

For `SignalPlot` (and only `SignalPlot`) the `PlotStyle` option has been redefined somewhat. Because there are two curves (the real part and the imaginary part) associated with each function, it is expected that a style will be specified as a pair, the first part of which applies to the real curve and the second to the imaginary curve. If only a single style is given, it will apply to both curves.

- Here are three functions displayed with different styles.

```
In[11]:= SignalPlot[
             {x^2/2,
              Sin[x] + I Cos[x],
              UnitStep[x] + I/2 UnitStep[x]},
             {x, 0, 2},
             PlotStyle ->
                 {GrayLevel[0],
                  {Hue[0], Hue[.6]},
                  {{Dashing[{0.15, 0.15}], GrayLevel[0]},
                   {Dashing[{0.05, 0.05}], GrayLevel[0.5]}}
                 }
         ]
```




```
Out[11]= - Graphics -
```

`DiracDeltaScaling -> False`	draw the delta functions in `SignalPlot` as uniform infinite impulses

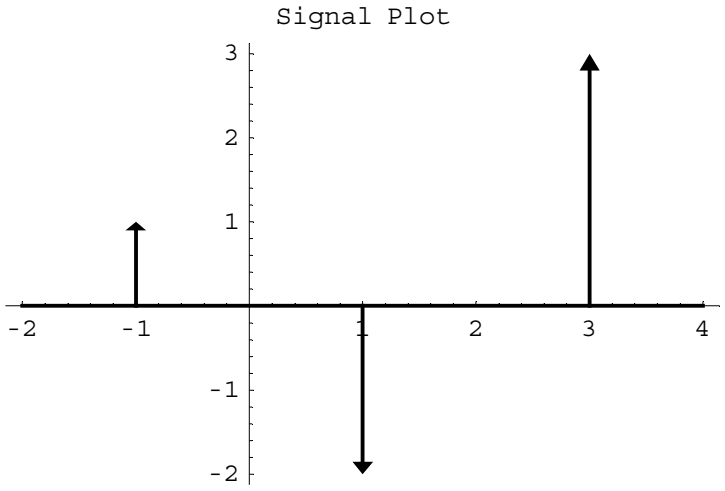
Option specific to continuous two-dimensional plotting functions.

Dirac delta functions are graphed in more than one way in the signals and systems literature. One technique is to let the height of the impulse represent the area under the delta function, while another is to make them of uniform height, since they have infinite value at a point. Both of these can be generated with the `DiracDeltaScaling` option to `SignalPlot` and `MagnitudePhasePlot`. When set to `True`, the heights are scaled to the areas under

the deltas. When `False`, the impulses are drawn as "infinite" pulses, running from the horizontal axis to the top of the graph.

- Here, the delta functions are drawn scaled to the areas under the deltas.

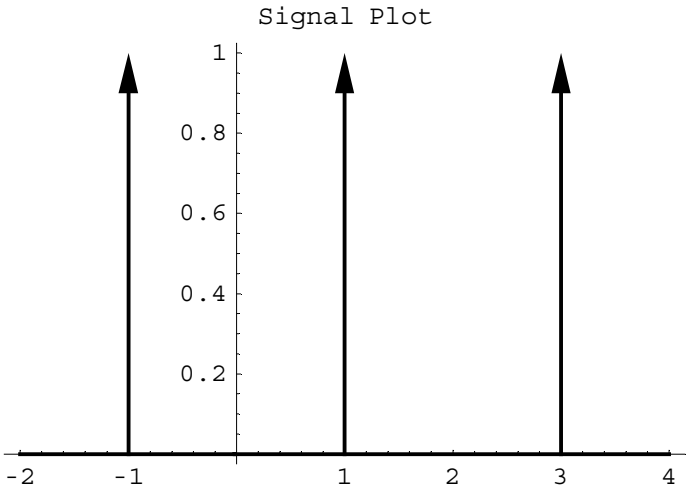
```
In[12]:= SignalPlot[
             DiracDelta[t + 1] -
                 2 DiracDelta[t - 1] +
                 3 DiracDelta[t - 3],
             {t, -2, 4},
             DiracDeltaScaling -> True
         ]
```




```
Out[12]= - Graphics -
```

- In this graph, the deltas are drawn as uniform impulses.

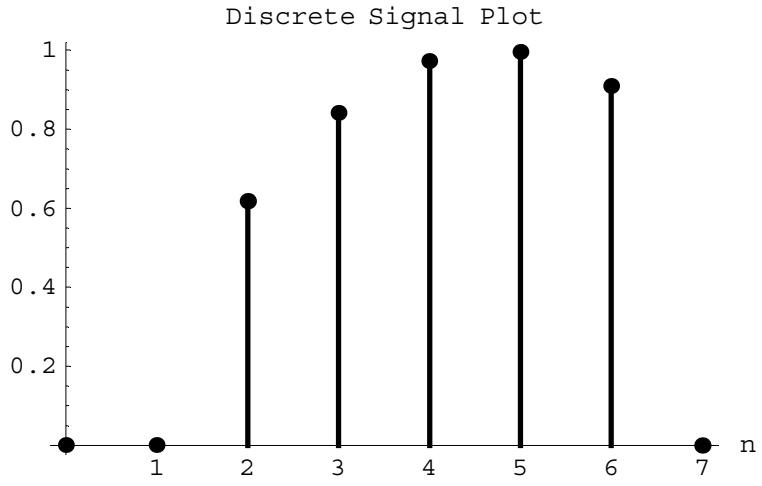
```
In[13]:= SignalPlot[
             DiracDelta[t + 1] -
                 2 DiracDelta[t - 1] +
                 3 DiracDelta[t - 3],
             {t, -2, 4},
             DiracDeltaScaling -> False
         ]
```




```
Out[13]= - Graphics -
```

- A plot of a discrete signal can also be generated. Note that the independent variable can only take integer values for a discrete signal.

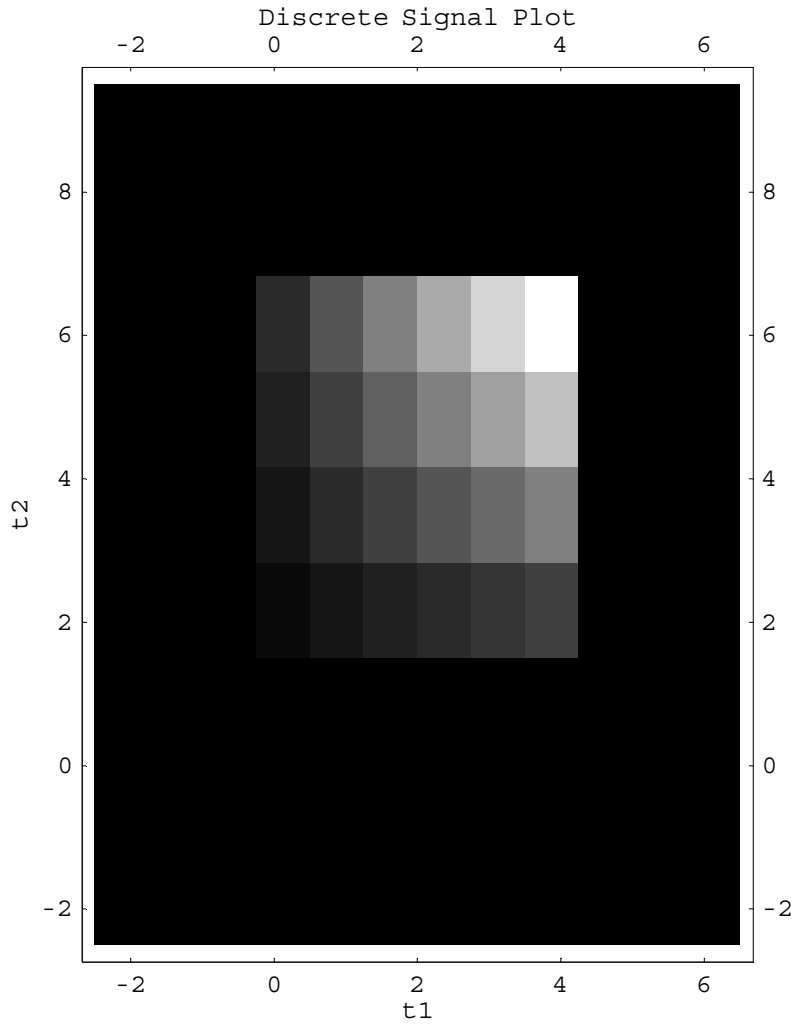
```
In[14]:= DiscreteSignalPlot[
           Sin[n/3] DiscretePulse[5, n - 2],
           {n, 0, 7}
         ]
```




```
Out[14]= - Graphics -
```

- A convenient representation for a discrete signal over a two-dimensional support is a density plot. The usual `DensityPlot` options can be passed through `DiscreteSignalPlot3D`.

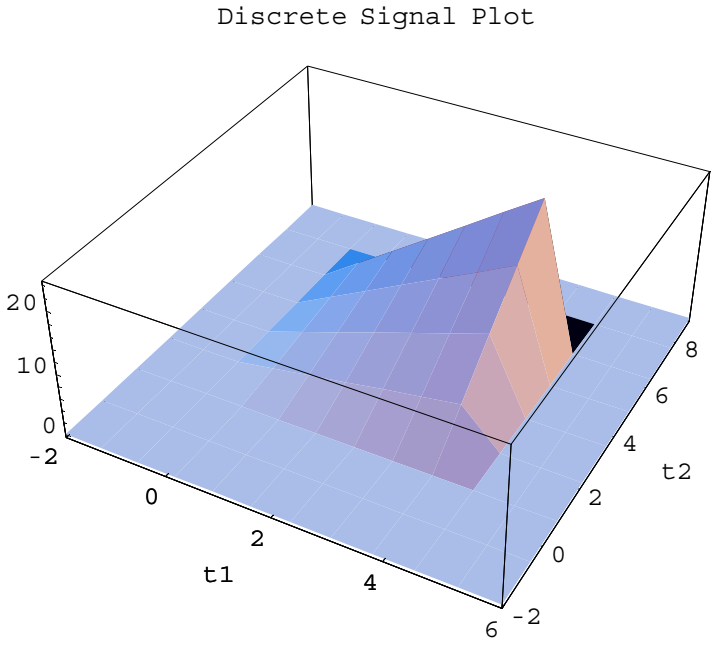
```
In[15]:= DiscreteSignalDensityPlot[
           t1 t2 DiscretePulse[{5, 7}, {t1, t2}],
           {t1, -2, 6}, {t2, -2, 9},
           Mesh -> False
         ]
```




```
Out[15]= - DensityGraphics -
```

- Here is the same function plotted as a surface. Note that the samples are located at the polygon intersections, not in the center of each polygon as they are for the density plot.

```
In[16]:= DiscreteSignalPlot3D[
           t1 t2 DiscretePulse[{5, 7}, {t1, t2}],
           {t1, -2, 6}, {t2, -2, 9},
           Mesh -> False
         ]
```




```
Out[16]= - SurfaceGraphics -
```

These functions accept essentially the same options as their equivalent standard functions. The default behavior of some options is modified; for instance, axes labels are generated by default, as is a plot label. For all continuous signal plots, the default value of the `MaxBend` option has been set to 2, to allow the creation of smoother plots.

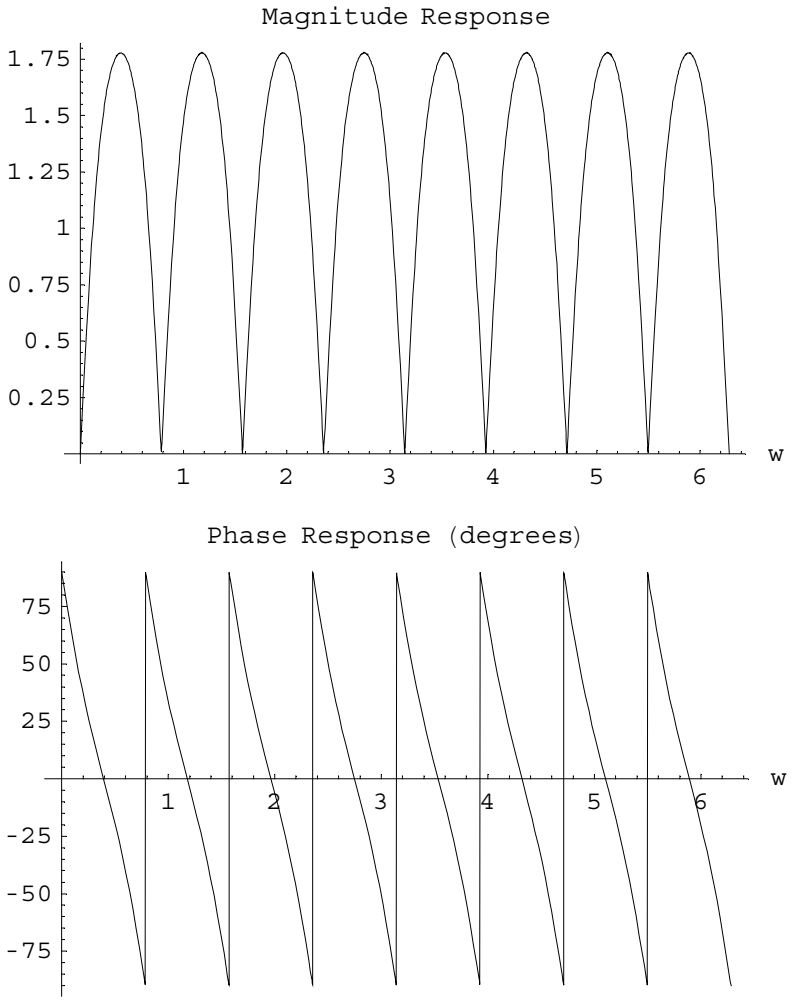
MagnitudePhasePlot[*freqresp*, {*f*, *start*, *end*}]	plot the magnitude and the phase responses of the function
MagnitudePhasePlot3D[*freqresp*, {f_1, s_1, e_1}, {f_2, s_2, e_2}]	plot the magnitude and phase of a function with two independent variables
BodePlot[*freqresp*, {*f*, *start*, *end*}]	create a Bode plot of the given frequency response function
BodePlot3D[*freqresp*, {f_1, s_1, e_1}, {f_2, s_2, e_2}]	create the Bode plot of a function with two independent variables
DiscreteMagnitudePhasePlot[*freqresp*, {*f*, *start*, *end*}]	plot the magnitude and the phase responses of the discrete function
DiscreteMagnitudePhasePlot3D[*freqresp*, {f_1, s_1, e_1}, {f_2, s_2, e_2}]	plot the magnitude and phase of a function with two independent discrete variables

Functions for plotting the frequency response.

A common technique for visualizing a frequency response is to separate the magnitude and phase, and plot each. The implementation of MagnitudePhasePlot generates two graphics, and returns a list of graphics objects. BodePlot is essentially a call to the magnitude-phase plot with axes scales set for logarithmic points. These functions assume that a frequency response has been computed, and is being passed as the argument to the plotting function.

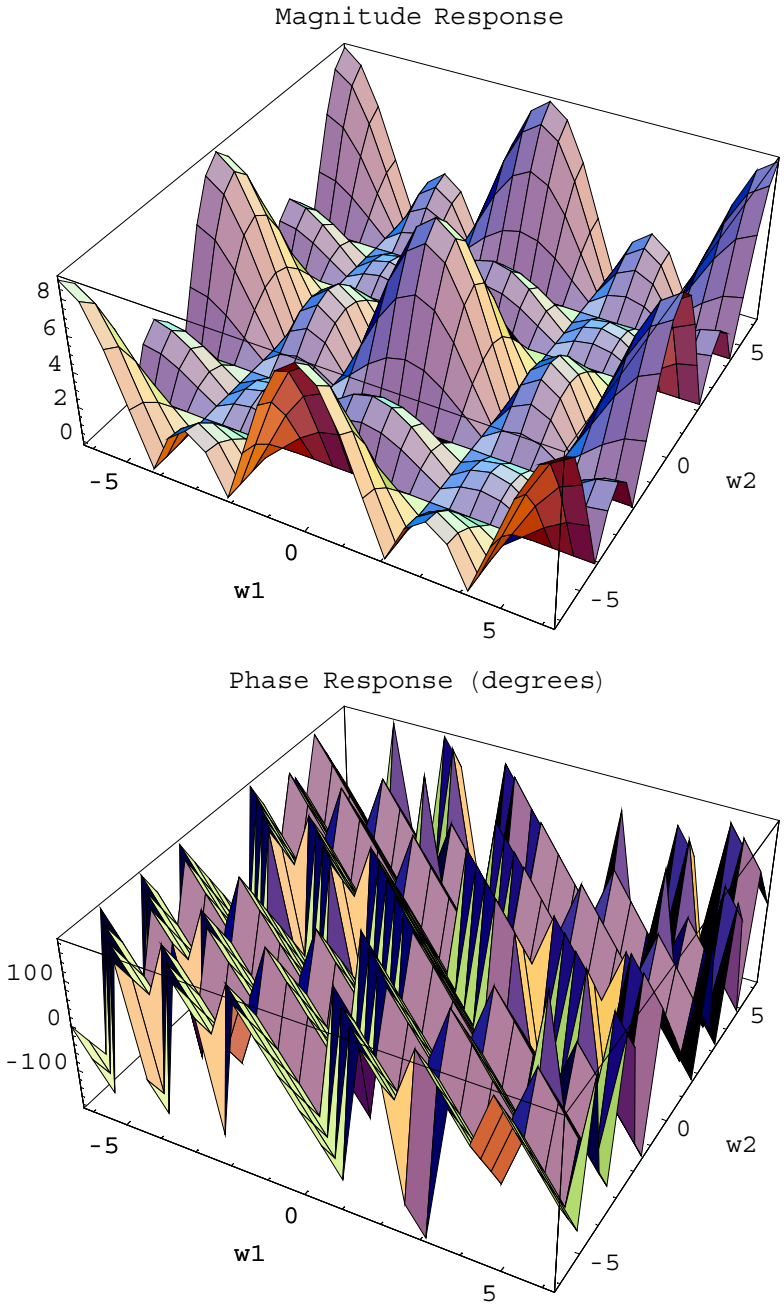
- Here is a plot of the frequency response of an eighth-order comb filter.

```
In[17]:= MagnitudePhasePlot[
           (1 - z^(-8))/(1 - z^(-8)/8)/.
             z -> Exp[I w],
           {w, 0, 2 Pi}
         ];
```



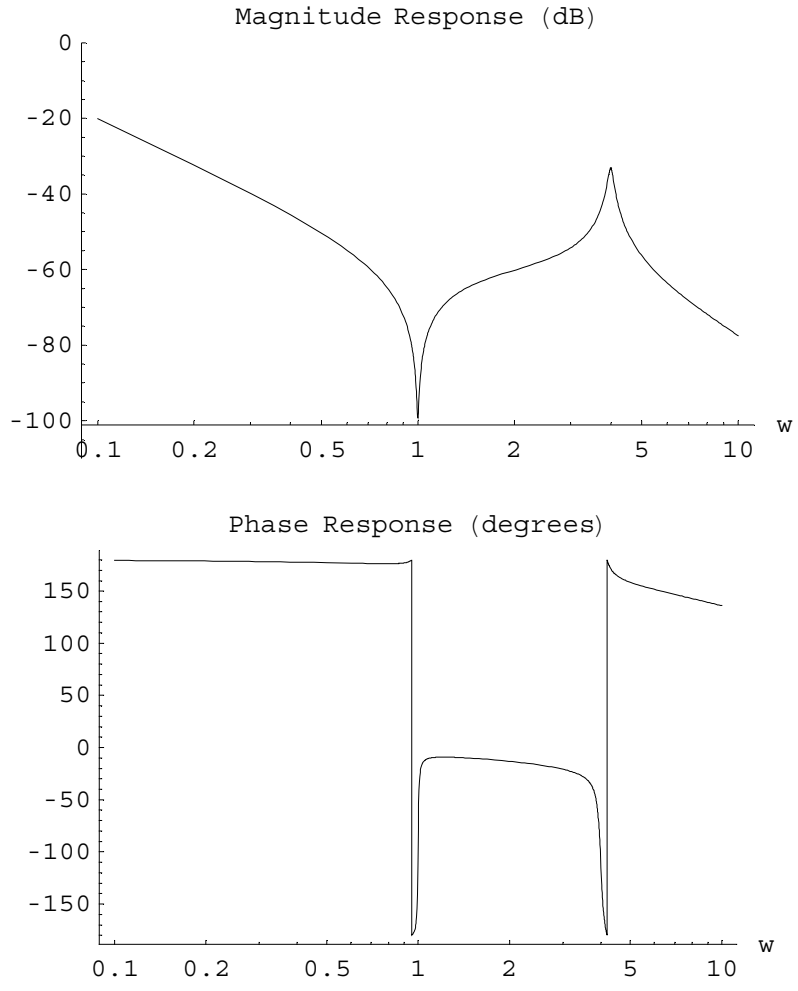
- The magnitude-phase plot is the customary tool for frequency analysis of signals. Here is the discrete-time Fourier transform of a two-dimensional signal.

```
In[18]:= MagnitudePhasePlot3D[
           DiscreteTimeFourierTransform[
              DiscretePulse[3, n1 - 1] *
                  DiscretePulse[3, n2 - 1],
              {n1, n2}, {w1, w2}
           ],
           {w1, -2 Pi, 2 Pi},
           {w2, -2 Pi, 2 Pi}
         ];
```


Magnitude Response
8
6
4
2
0
-5
0
w1
5
5
0
w2
-5
Phase Response (degrees)
100
0
-100
-5
0
w1
5
5
0
w2
-5

- The Bode plot is equivalent to `MagnitudePhasePlot` with certain options set to different default values. Here is a Bode plot of the frequency response of a particular analog transfer function.

```
In[19]:= BodePlot[
             0.01 (0.01 s + s^2 + 1)/
             (10 s^2 + 1.1 s^3 + 0.635 s^4 + s^5/16)/.
                 s -> I w,
             {w, 0.1, 10}
         ];
```



- Here is a discrete transform of a pulse of width 10 starting at the origin.

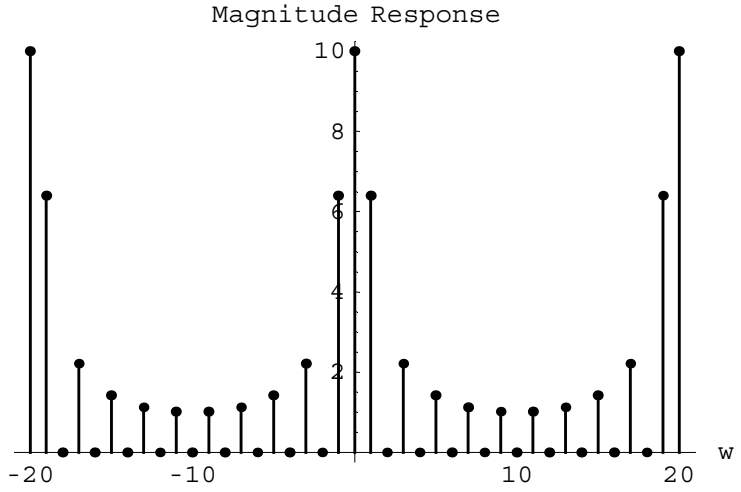
```
In[20]:= trans = DiscreteFourierTransform[
             DiscretePulse[10, n],
             20, n, w
         ]
```

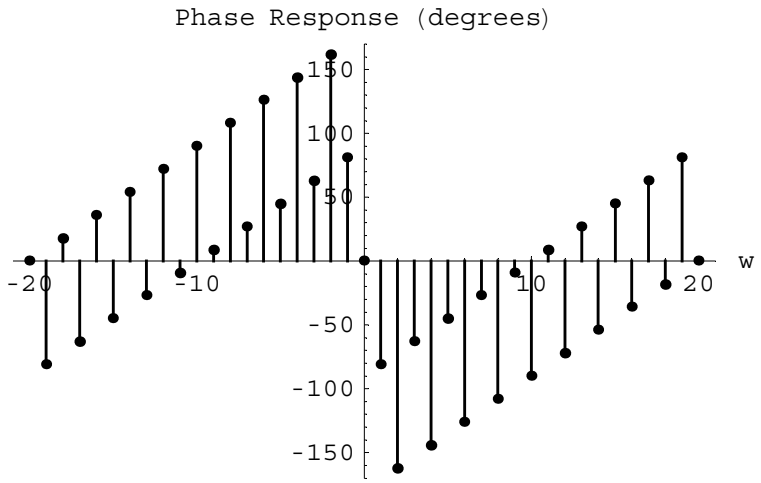
$$\text{Out[20]= DFTData}\left[10\ \text{Periodic}_{20,w}\left[e^{-\frac{9}{20}\,i\,\pi\,w}\ \text{DirichletSinc}_{10}\left[\frac{\pi\,w}{10}\right]\right],\ \text{Start[0], Finish[19], TransformVariables[w]}\right]$$

- Here is a magnitude-phase plot of the result. Note that the transform consisted of an infinite summation of shifted functions (the effect of the `Periodic` operator). Since the exponential and sinc functions have infinite domain, the plotting function wasn't able to compute a closed form for the behavior of `Periodic`, and instead only computed a single cycle. Hence, the resulting image ignores aliasing effects of the shifted copies of the exponential and sinc terms in the transform.

```
In[21]:= DiscreteMagnitudePhasePlot[
             trans,
             {w, -20, 20}
         ]
```

```
Summation::extent :
 Could not compute which part of the summation should be plotted.
```





Out[21]= {- Graphics -, - Graphics -}

options name	*default value*	
DomainScale	Linear	whether the domain (horizontal axis) is plotted with Linear or Log scaling
MagnitudeScale	Linear	whether the magnitude (vertical axis of the first graph) is plotted with Linear or Log scaling
PhaseScale	Degree	whether the phase (vertical axis of the second graph) is marked in degrees or radians

Options specific to MagnitudePhasePlot.

MagnitudePhasePlot and MagnitudePhasePlot3D accept options standard to Plot and Plot3D, respectively. They also accept certain other options dictating the scaling to be used in the plots, and specifying whether the plots are over continuous or discrete frequency domains.

The DomainScale and MagnitudeScale options can take the values Log or Linear, specifying logarithmic or linear scaling on the horizontal axis of both graphs or the vertical axis of the first graph, respectively. If logarithmic scaling is used, the domain in question must be positive. The MagnitudeScale option can also be set to None, which will prevent the magnitude plot from being displayed.

PhaseScale determines whether the phase angle is plotted in degrees or radians when

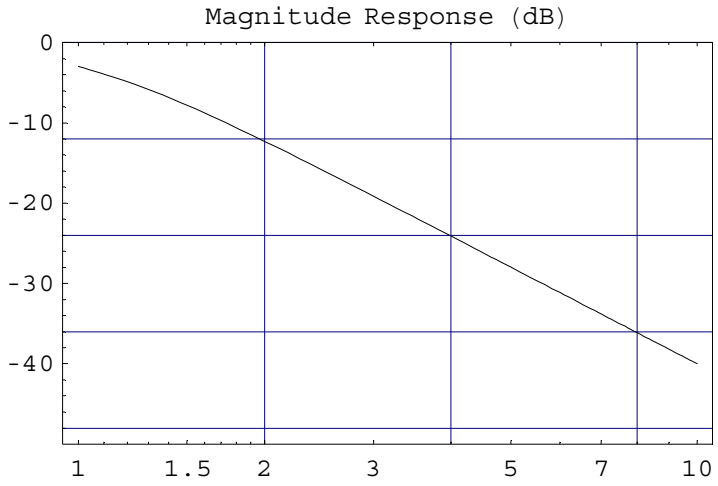
given the value `Degree` or `Radian`, respectively. This option affects only the vertical axis of the second (phase) graph. The `PhaseScale` option can also be set to `None`, which will prevent the phase plot from being displayed.

The `Domain` option will cause the plot to be generated in the continuous frequency domain (if set to `Continuous`) or the discrete domain (if given as `Discrete`). The range will take on only integer values in the discrete domain.

Of these specific options, `BodePlot` accepts only `PhaseRangeScale`. `MagnitudePhasePlot` also accepts the `DiracDeltaScaling` option described earlier.

- Here is the magnitude frequency response of a second-order Butterworth filter plotted on a logarithmic scale. Note that as the frequency grows much greater than the cutoff frequency, the rolloff approaches $-6\,N$ decibels per octave, where N is the order of the Butterworth filter.

```
In[22]:= MagnitudePhasePlot[
             FourierTransform[
                 AnalogFilter[
                     GetRootList[s^2 + Sqrt[2] s + 1, s],
                     {}, t
                 ],
                 t, w
             ],
             {w, 1, 10},
             DomainScale -> Log,
             MagnitudeScale -> Log,
             PhaseScale -> None,
           GridLines -> {{2,4,8},{0, -12, -24, -36, -48}},
           Frame -> True,
           Axes -> False
          ]
```



Out[22]= {- Graphics -, - Graphics -}

PoleZeroPlot[*transferfun*]	generate a pole-zero plot from a Z or Laplace transform data object
PoleZeroPlot[*transferfun*, *var*]	generate a pole-zero plot, given a transfer function in the named variable
PoleZeroPlot[*transferfun*, {*var*, *min*, *max*}]	generate a pole-zero plot with the region of convergence running from *min* to *max*, given a transfer function in the named variable
PoleZeroPlot[*transferfun*, var_1, var_2]	generate a pole-zero plot, given a two-dimensional transfer function in the named variables
PoleZeroPlot[*transferfun*, {v_1, min_1, max_1}, {v_2, min_2, max_2}]	generate a pole-zero plot with the region of convergence running from min_n to max_n, given a two-dimensional transfer function in the named variables

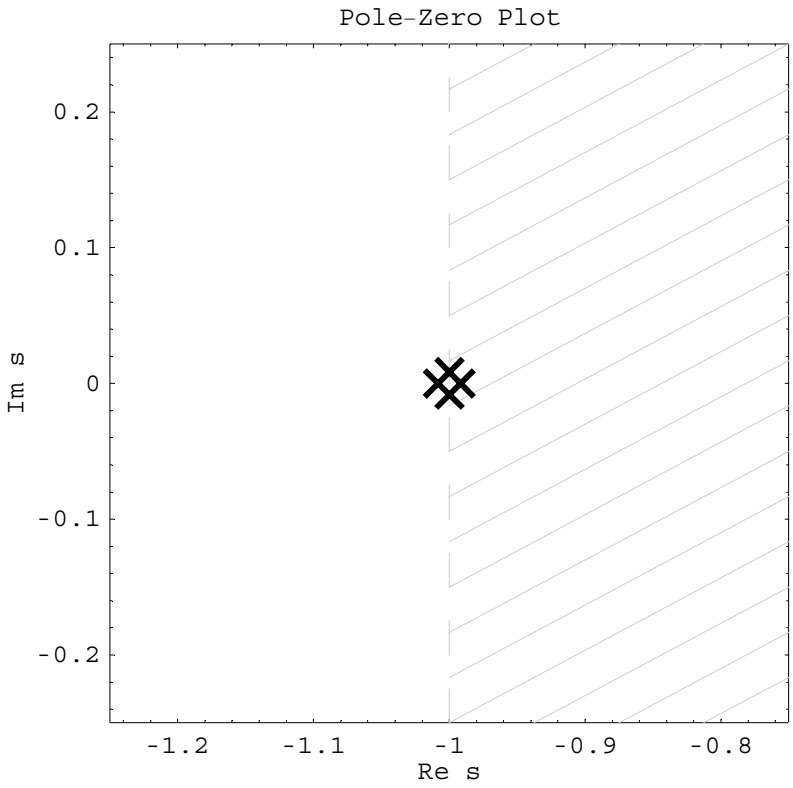
Pole-zero plots in Signals and Systems.

The basic pole-zero plot is designed for plotting poles and zeros of rational transfer functions, along with appropriate regions of convergence. Often, these are taken from the results of ZTransform and LaplaceTransform. If so, most of the information necessary to generate the plot is taken from the transform object, and does not need to be specified again. Alternate forms of the syntax for PoleZeroPlot allow arbitrary functions to be used,

although interpretation of the region of convergence in the Z or Laplace domain then needs to be specified by the `Domain` option. The plot range will usually be determined automatically (by the locations of the poles and zeros), but can be overridden by use of the `PlotRange` option.

- Here is a pole-zero plot of a Laplace transform.

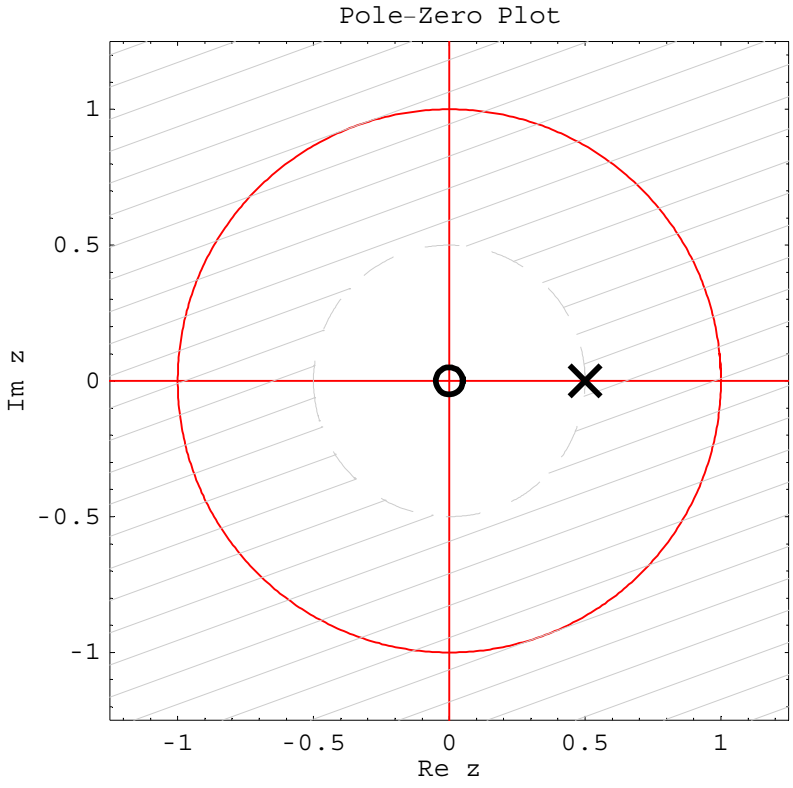
```
In[23]:= PoleZeroPlot[
             LaplaceTransform[
                 t Exp[-t] UnitStep[t],
                 t, s
             ]
         ]
```




```
Out[23]= - Graphics -
```

- This is a pole-zero plot of a Z transform.

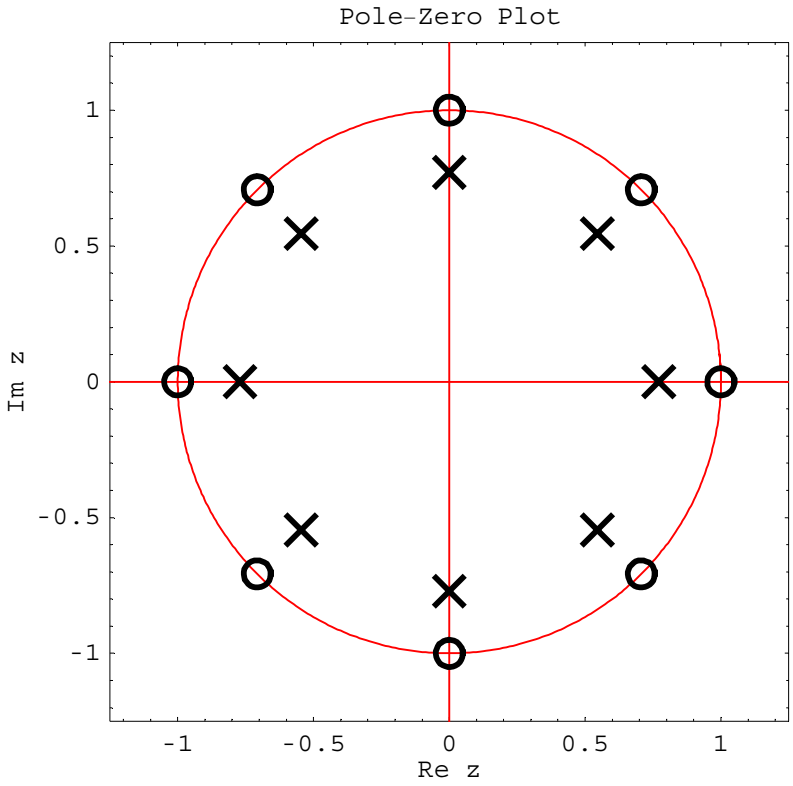
```
In[24]:= PoleZeroPlot[
             ZTransform[
                 0.5^n DiscreteStep[n],
                 n, z
             ]
         ]
```




```
Out[24]= - Graphics -
```

- Here is a plot of the comb-filter transfer function used earlier. Note that with the `ZTransform` default domain, the unit circle is drawn to clarify the plot.

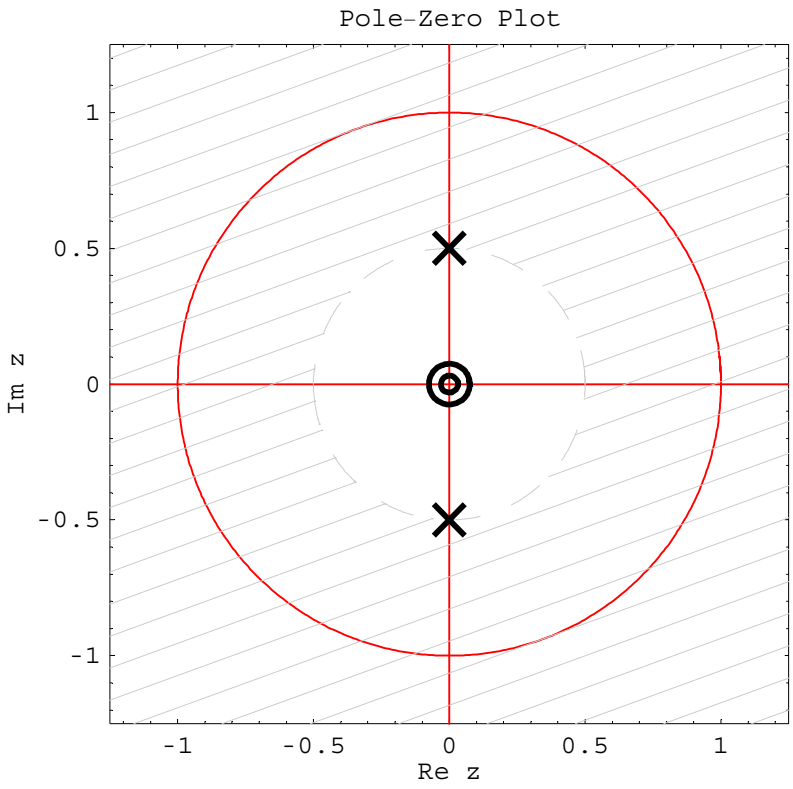
```
In[25]:= PoleZeroPlot[
             (1 - z^(-8))/(1 - z^(-8)/8),
             z
         ]
```




```
Out[25]= - Graphics -
```

- If a region of convergence is specified, the appropriate area is shaded.

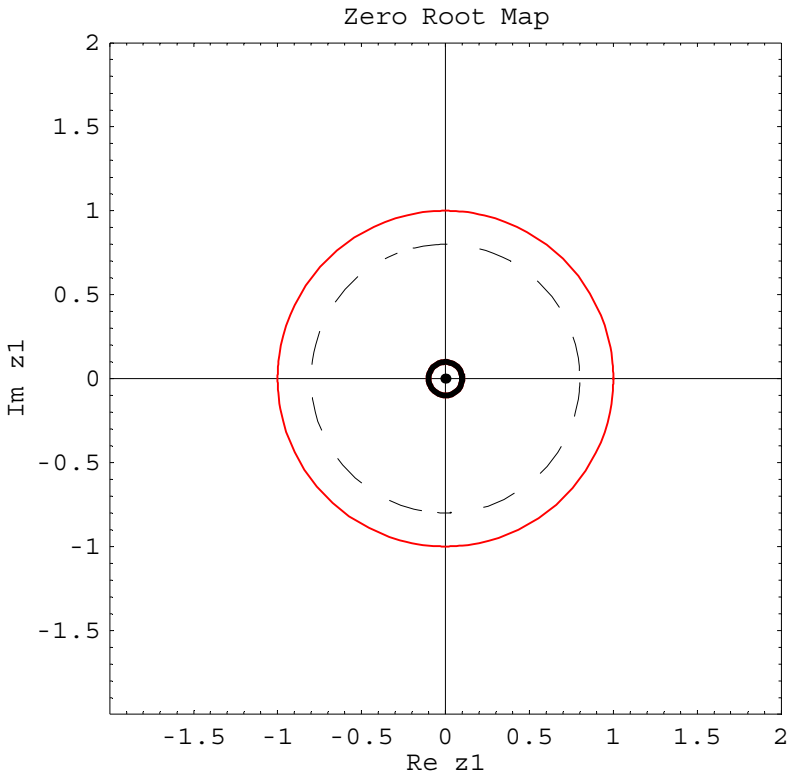
```
In[26]:= PoleZeroPlot[
             z^2/(z^2 + 0.25),
             {z, .5, Infinity}
         ]
```

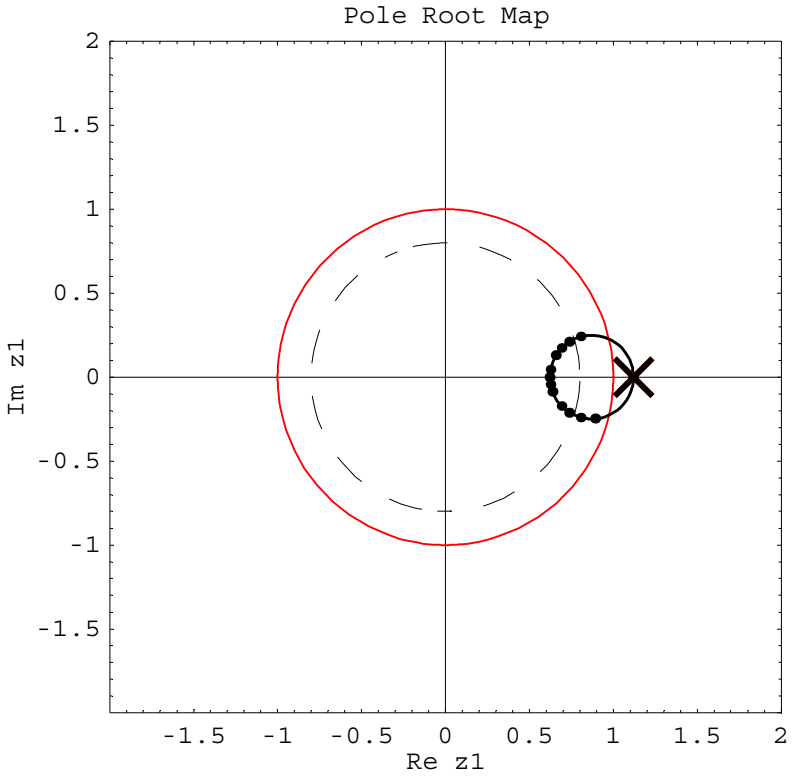


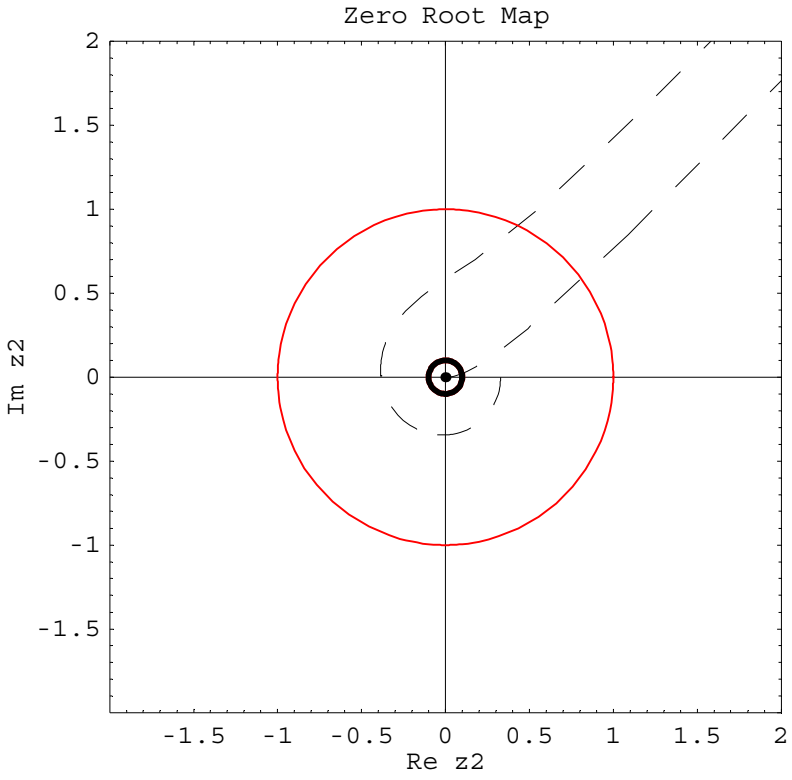

```
Out[26]= - Graphics -
```

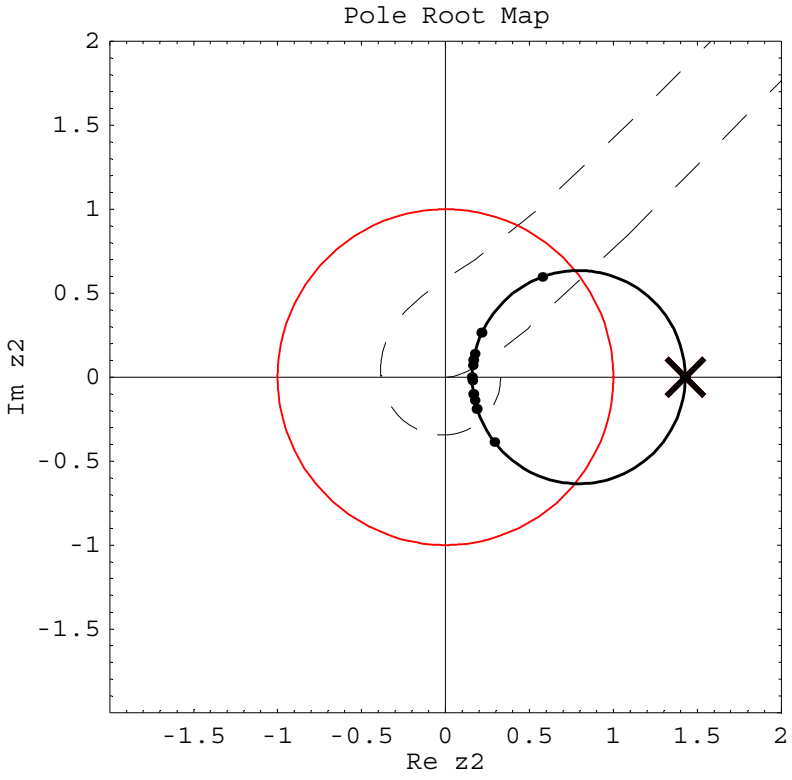
- Here is a pole-zero plot of a signal over a two-dimensional support. This particular system is unstable; note that poles and zeros can be found outside of the region of convergence. The lower bound of the region of convergence is marked by the dashed line. The zeros and poles are drawn in separate graphs, in which the first two plots correspond to the first variable, with the second variable mapped by `Exp[I w1]`, while the second two graphs reverse this order, mapping the first variable to `Exp[I w2]`.

```
In[27]:= PoleZeroPlot[
           ZTransform[
             ((n1 + n2)! (4/5)^n1 *
                 (2/7)^n2 DiscreteStep[n1, n2])/
                 (n1! n2!),
             {n1, n2}, {z1, z2}
           ]
         ]
```




Pole Root Map
Im z1
Re z1
2
1.5
1
0.5
0
-0.5
-1
-1.5
-1.5
-1
-0.5
0
0.5
1
1.5
2


Zero Root Map
Im z2
Re z2
2
1.5
1
0.5
0
-0.5
-1
-1.5
-1.5
-1
-0.5
0
0.5
1
1.5
2



Out[27]= {{- Graphics -, - Graphics -}, {- Graphics -, - Graphics -}}

`Domain -> LaplaceTransform`	assume that the region of convergence is in the Laplace domain (specified as the left and right bounds of a strip of the complex plane)
`Justification -> All`	print information about the locations of poles and zeros in the plot
`PoleZeroPhasor -> {x, y}`	draw phasors from all poles and zeros to the specified location (one variable only)

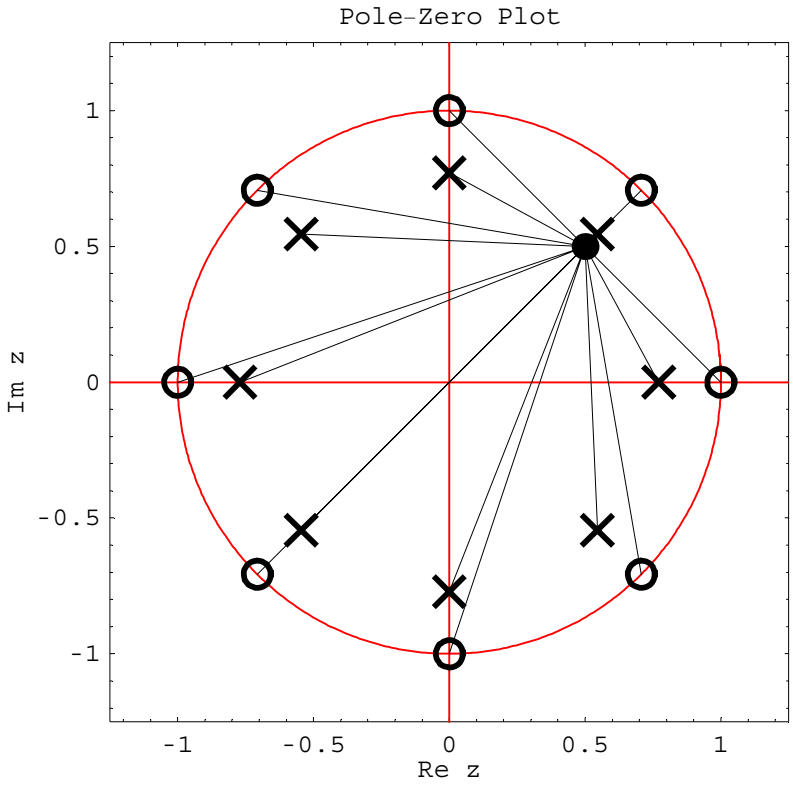
Options for `PoleZeroPlot`.

- The `Justification` option causes the locations of the poles and zeros to be printed, while the `PoleZeroPhasors` option causes lines to be drawn from the poles and zeros to the specified point.

```
In[28]:= PoleZeroPlot[
           (1 - z^(-8))/(1 - z^(-8)/8),
           z,
           Domain -> ZTransform,
           Justification -> All,
           PoleZeroPhasor -> {.5, .5}
         ]
```

```
The zeroes are:  {-1., 0. - 1. i, 0. + 1. i, 1., -0.707107 - 0.707107 i,
  0.707107 + 0.707107 i, 0.707107 - 0.707107 i, -0.707107 + 0.707107 i}

The poles are:   {-0.771105, 0. - 0.771105 i, 0. + 0.771105 i,
  0.771105, -0.545254 - 0.545254 i, 0.545254 + 0.545254 i,
  0.545254 - 0.545254 i, -0.545254 + 0.545254 i}
```



Out[28]= - Graphics -

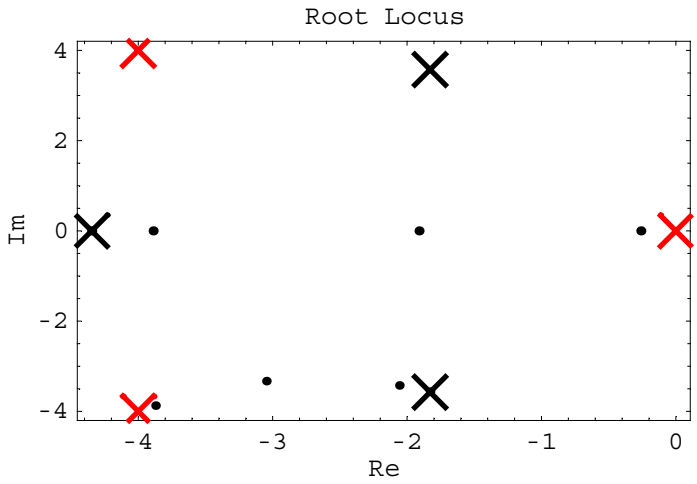
RootLocusPlot[*function*, *var*, {*param*, *min*, *max*}]	generate a root-locus plot for the given function in the named variable, as the free parameter varies from *min* to *max*

The root-locus plot.

The root-locus plot is used to investigate the evolution of the positions of roots of some function as a parameter changes over a series of values. The RootLocusPlot function will attempt to symbolically factor the function so that separate root curves can be plotted parametrically. The initial locations of the poles and zeros are displayed in red, while the final locations are in black.

- Here is a root-locus plot.

```
In[29]:= RootLocusPlot[
             1/(s ((s + 4)^2 + 16) + k),
             s, {k, 0, 70}
         ]
```



Out[29]= - Graphics -

Standard graphics options can be used with RootLocusPlot, and will be applied to all of the images.

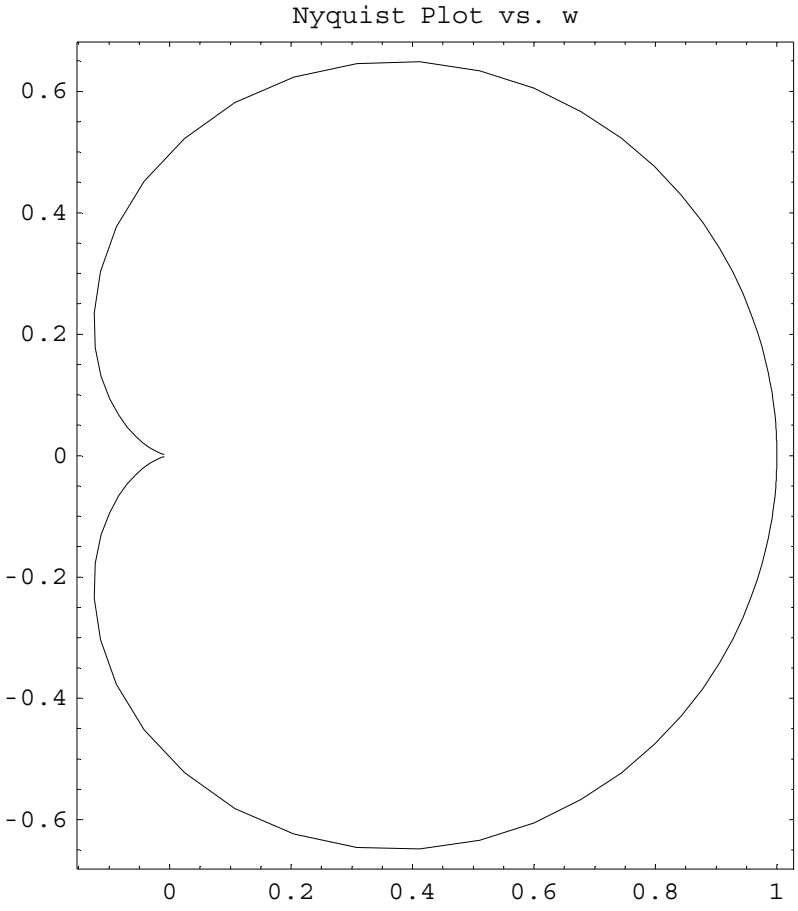
`NyquistPlot[` *function*`,` `{` *freq*`,` *min*`,` *max*`}]`	create a Nyquist plot of the given function in the complex plane, dependent upon the frequency variable *freq*

The Nyquist plot.

The Nyquist plot is useful for investigating the frequency response of a transfer function in the complex plane.

■ Here is the Nyquist plot of a transfer function.

```
In[30]:= NyquistPlot[1/(I w + 1)^2, {w, -10, 10},
           Frame -> True, Axes -> False
         ]
```




```
Out[30]= - Graphics -
```

NyquistPlot will accept standard Graphics options, as well as the option PlotPoints. The PlotPoints option is set to a default value of 100, since curves plotted by this method tend to have considerable variability.