5 Mathematical Transforms

Signals and Systems implements a variety of standard mathematical linear transforms frequently used in signal processing. While many of these transforms are also implemented in the standard *Mathematica*, there are several differences. For instance, the standard routines do not handle as wide a variety of functions specific to signal processing. Another difference is that Signals and Systems allows the use of bilateral, left-sided, or right-sided transforms (in which the range of integration or summation runs from `-Infinity` to `Infinity`, `-Infinity` to `0`, or `0` to `Infinity`, respectively), while for Laplace transforms, *Mathematica* implements only the right-sided variant. These modifications improve the utility of the transforms for signal processing applications. Other advantages of the transforms as implemented in this application include the ability to track regions of convergence for Laplace and Z transforms, an option allowing you to specify abstract transform pairs, and the implementation of some nonseparable multidimensional transforms.

All of the transforms in Signals and Systems perform symbolic computation, including the discrete Fourier transform. This allows transforms to be performed on signals expressed symbolically, rather than as explicitly specified sequences of data.

■ 5.1 Transforms of Continuous Signals

The Laplace Transform

The Laplace transform is a staple operation from the analysis of continuous systems. The transform and its inverse are used primarily in transient analysis. At its core, Signals and Systems uses the standard definition given in many texts.

$$F(s) = \mathcal{L}\{f(t)\} = \int_{-\infty}^{+\infty} f(t)\, e^{-s\,t}\, d\,t$$

The region of convergence is a strip in the s plane. Signals and Systems provides the convergence information to the user by placing the result in a special data type. The inverse transform is defined as

$$f(t) = \mathcal{L}^{-1}\{F(s)\} = \frac{1}{2\pi j} \oint_{\sigma - j\infty}^{\sigma + j\infty} F(s)\, e^{s t}\, d\, s$$

(The constant σ is appropriately chosen based on convergence of the integrand.)

The actual algorithm used by the application involves a combination of table lookup and rules based on transform properties. You can specify that the standard *Mathematica* integration routines be attempted as well, based on the preceding definition. For more details on the algorithm, see, for instance, Evans (1992).

`LaplaceTransform[`*expr*`, t, s]`	perform a Laplace transform from the variable t to s
`LaplaceTransform[`*expr*`, {`t_1`, `t_2`, ..., `t_n`}, {`s_1`, `s_2`, ..., `s_n`}]`	perform a multidimensional Laplace transform from t_i to s_i
`LaplaceTransform[t, s][`*expr*`]`	nonevaluating operator form of the Laplace transform
`InverseLaplaceTransform[`*expr*`, s, t]`	perform an inverse Laplace transform
`InverseLaplaceTransform[s, t][`*expr*`]`	operator form of the inverse Laplace transform

The Laplace transform and its inverse.

The Laplace transform uses the same syntax as the standard *Mathematica* function `LaplaceTransform`. It has several extensions, however. One modification is a nonevaluating operator form for use in manipulation of systems. All of the transforms provided with Signals and Systems have this alternate syntax.

- First, make the signal processing functions available.

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

- Here is a basic Laplace transform.

```
In[2]:= LaplaceTransform[Exp[-Abs[t]], t, s]

Out[2]= LaplaceTransformData[-2/(-1 + s^2),
         RegionOfConvergence[-1, 1], TransformVariables[s]]
```

Another major difference between the standard *Mathematica* version and the version here is immediately apparent. Now, `LaplaceTransform` returns by default the special data type `LaplaceTransformData`. This data type includes information on the region of convergence and the variables involved in the transformation, which is particularly useful in stability analysis.

- Here is an example of a two-dimensional transform. This function is nonseparable, and represents an impulse of changing magnitude along the line $t_1 = t_2$, where t_1 and t_2 are greater than zero.

```
In[3]:= LaplaceTransform[
            Exp[a t1] *
                DiracDelta[t1 - t2] UnitStep[{t1, t2}],
            {t1, t2}, {s1, s2}
        ]

Out[3]= LaplaceTransformData[1/(-a + s1 + s2),
         RegionOfConvergence[{Re[a], Re[a]}, {∞, ∞}],
         TransformVariables[{s1, s2}]]
```

Like the forward transform, the inverse transforms have rule bases that implement many properties of the transform.

- Here is the inverse transform of a rational function.

```
In[4]:= InverseLaplaceTransform[s/(1 + s^2), s, t]

Out[4]= Cos[t] UnitStep[t]
```

- The rule base knows that multiplication of the function by an exponential term is equivalent to a shift of the inverse transform.

```
In[5]:= InverseLaplaceTransform[
            Exp[-a s] s / ( 1 + s^2 ),
            s, t
        ]

Out[5]= Cos[a - t] UnitStep[-a + t]
```

options name	*default value*	
Justification	None	additional information to be displayed about the generation of the transform
SeriesTerms	10	how many terms should be used if a series expansion is necessary to complete the transform
StandardFormula	False	whether to try the integral definition directly
SimplifyOutput	All	whether to attempt simplification of results
TransformDirection	$TransformDirection	direction of the transform in the "time" domain
TransformPairs	{}	rules to employ in attempting to transform unknown functions

Options for LaplaceTransform.

A variety of options can be employed to enhance the behavior of LaplaceTransform. The option named TransformDirection specifies whether the transform is to be bilateral (TwoSided), right-sided (RightSided), or left-sided (LeftSided). By default, it is set to $TransformDirection, which is set to TwoSided. By changing the current value of $TransformDirection, you can set this characteristic for all of the transform functions simultaneously.

- A left-sided transform is equivalent to the bilateral transform of the function multiplied by a UnitStep running in the negative direction.

```
In[6]:= LaplaceTransform[Exp[-Abs[t]], t, s,
           TransformDirection -> LeftSided]

Out[6]= LaplaceTransformData[ 1/(1 - s),
         RegionOfConvergence[-∞, 1], TransformVariables[s]]
```

The Justification option can be employed to display the steps involved in performing the transform. It takes the values All, Automatic, or None, where All provides all information available, Automatic provides only the most useful information, and None provides no additional information about the procedure.

- Here is a transform displaying the procedures used in the computation.

```
In[7]:= LaplaceTransform[Exp[-Abs[t]], t, s,
          Justification -> All]
```

$$\mathcal{L}_t\{e^{-\text{Abs}[t]}\}$$

which becomes

(after rewriting the two-sided expression

$$e^{-\text{Abs}[t]}$$

as a left-sided plus a right-sided function:

$$e^{t}\,\text{UnitStep}[-t] + e^{-t}\,\text{UnitStep}[t]\ .\ )$$

$$\mathcal{L}_t\{e^{t}\,\text{UnitStep}[-t] + e^{-t}\,\text{UnitStep}[t]\}$$

which becomes

$$\mathcal{L}_t\{e^{t}\,\text{UnitStep}[-t]\} + \mathcal{L}_t\{e^{-t}\,\text{UnitStep}[t]\}$$

which becomes

$$\left\{\mathcal{L}_t\{e^{-t}\,\text{UnitStep}[t]\}\right\}_{s\to -s \text{ and flip ROC}} + \left\{\mathcal{L}_t\{\text{UnitStep}[t]\}\right\}_{s\to 1+s \text{ and shift ROC}}$$

which becomes

$$\left\{\left\{\mathcal{L}_t\{\text{UnitStep}[t]\}\right\}_{s\to 1+s \text{ and shift ROC}}\right\}_{s\to -s \text{ and flip ROC}} + \text{Transform}\left[\frac{1}{1+s}, -1, \infty\right]$$

which becomes

$$\text{Transform}\left[\frac{1}{1-s} + \frac{1}{1+s}, -1, 1\right]$$

which becomes

$$\text{Transform}\left[\frac{1}{1-s}+\frac{1}{1+s},\ -1,\ 1\right]$$

which simplifies to

$$\text{LaplaceTransformData}\left[-\frac{2}{-1+s^2},\ \text{RegionOfConvergence}[-1,\ 1],\ \text{TransformVariables}[s]\right]$$

Out[7]=
$$\text{LaplaceTransformData}\left[-\frac{2}{-1+s^2},\ \text{RegionOfConvergence}[-1,\ 1],\ \text{TransformVariables}[s]\right]$$

By allowing you to specify your own transform pairs, the `TransformPairs` option lets you extend the transform rule bases for unknown functions. These may be functions that have not been entered into the rule base, or may be abstract transform pairs specific to a particular computation. The option accepts a list of `RuleDelayed` transformations giving the target function and its transform. Region of convergence information can be given by writing the rule's target in the form of a list, {*func*, *minconv*, *maxconv*}. Note that the target function must be in the same variable as the entire expression being transformed, while the transform of the target must be in terms of the transform variable.

- `LaplaceTransform` does not know how to handle the `Sign` function by default, but a user-defined rule can be specified to perform the transformation. Note that standard properties (*e.g.*, shift, scaling) are handled automatically.

In[8]:=
```
LaplaceTransform[
   Sign[3 t - 2] + UnitStep[t] Cos[omega t],
   t, s, TransformPairs -> {Sign[t] :> 2/s}
]
```

Out[8]=
$$\text{LaplaceTransformData}\left[\frac{2\,e^{-2s/3}}{s}+\frac{s}{\text{omega}^2+s^2},\ \text{RegionOfConvergence}[0,\ \infty],\ \text{TransformVariables}[s]\right]$$

Many transforms cannot be determined in closed form, but for transforms of continuous functions, we can compute an approximation based on the series expansion of the function. The `SeriesTerms` option specifies the number of terms to use in the expansion. If it is set to `None`, the series expansion will not be attempted.

- Here, a transform is attempted without performing a series expansion. The attempt fails; there are no rules for this function in the rule base.

```
In[9]:= LaplaceTransform[Tan[t], t, s,
            SeriesTerms -> None
        ]

Out[9]= -Incomplete Laplace Transform-
```

- This tries expanding the function to eight terms before giving up. In this case, the series expansion was successful, and an approximation to the transform could be returned.

```
In[10]:= LaplaceTransform[Tan[t], t, s,
             SeriesTerms -> 8
         ]

Out[10]= LaplaceTransformData[-DiracDelta'[s] -
           1/3 DiracDelta^(3)[s] - 2/15 DiracDelta^(5)[s] - 17/315 DiracDelta^(7)[s],
          RegionOfConvergence[0, 0], TransformVariables[s]]
```

The `StandardFormula` option allows the transform function to attempt to directly perform the integral if the transform cannot be found by use of the rule bases. It takes the values `True` or `False`, indicating whether or not to attempt the integral if necessary.

- The transform of this function is not known by the transform rule base, although a series approximation can be generated. The series representation is not useful to us in this example, so we suppress it via the `SeriesTerms` option.

```
In[11]:= LaplaceTransform[
             Sin[x] ContinuousPulse[3, x],
             x, s,
             SeriesTerms -> None
         ]

Out[11]= -Incomplete Laplace Transform-
```

- However, *Mathematica* can directly integrate this transform. Because the `StandardFormula` option is applied before the `SeriesTerms` option, we don't need to suppress the series expansion.

```
In[12]:= LaplaceTransform[
             Sin[x] ContinuousPulse[3, x],
             x, s,
             StandardFormula -> True
         ]
```

$$\text{Out[12]= LaplaceTransformData}\left[\frac{e^{-3s}\,(e^{3s} - \text{Cos}[3] - s\,\text{Sin}[3])}{1+s^2},\ \text{RegionOfConvergence}[0, \infty], \text{TransformVariables}[s]\right]$$

Also available is the `SimplifyOutput` option, which indicates whether automatic simplification will be attempted during the computation. Simplification of algebraic expressions is a computationally intensive task, but it can in some cases significantly improve the form of output. Note that simplification is performed by the `SimplifySignal` function, which may not result in a smaller leaf count, but should result in an expression that is easier to manipulate. The `SimplifyOutput` option can take the values `None`, `Automatic`, or `All`. If set to `All`, the simplification will assume that symbols that must be real-valued for the transform to complete are real-valued for simplification purposes; this is slower but less thorough than the setting `Automatic`, which does not make that assumption. If set to `None`, no simplification will be attempted.

`DecompositionPrecision->` *n*	if the transform needs to employ partial fraction decomposition, it will normally look for exact roots; in cases of large polynomials, it may be advantageous to find numeric roots of the specified precision

Option specific to the inverse Laplace transform.

The inverse Laplace transform has one option not available for the forward transform. The option called `DecompositionPrecision` controls the behavior of partial fraction decomposition in the inverse transform. When partial fraction decomposition must be used to determine the transform, an exact technique is usually employed to find the roots of the denominator. However, for polynomials of higher order, this may be infeasible. Setting this option to a precision less than `Infinity` allows a purely numeric technique to be employed. (For quick computation, it is usually best to set it to `MachinePrecision`; in some cases,

higher precision may be useful.) The resulting transform will not be as precise, but it will still be useful. If the strategy of partial fraction composition should not be attempted, set `DecompositionPrecision` to `None`.

- Here is a transform performed with the (default) exact partial fraction decomposition.

```
In[13]:= InverseLaplaceTransform[
           (s + 3)/(s^2 + 2 s + 3), s, t,
           DecompositionPrecision -> Infinity
         ]
```

$$\text{Out[13]}= \frac{1}{2} e^{\left(-1-i\sqrt{2}\right)t}\left(1+i\sqrt{2}+\left(1-i\sqrt{2}\right)e^{2i\sqrt{2}\,t}\right)\text{UnitStep}[t]$$

- The transform can be performed much more quickly by specifying a finite precision, but at the cost of returning numeric output.

```
In[14]:= InverseLaplaceTransform[
           (s + 3)/(s^2 + 2 s + 3), s, t,
           DecompositionPrecision -> MachinePrecision
         ]
```

$$\text{Out[14]}= 1.73205\, e^{-1.\,t}\,\text{Cos}[0.955317 - 1.41421\,t]\,\text{UnitStep}[t]$$

- You can disable the attempt at partial fraction decomposition. In this case, other rules take over to compute the transform.

```
In[15]:= InverseLaplaceTransform[
           (s + 3)/(s^2 + 2 s + 3), s, t,
           DecompositionPrecision -> None
         ]
```

$$\text{Out[15]}= e^{-t}\left(\text{Cos}\left[\sqrt{2}\,t\right]+\sqrt{2}\,\text{Sin}\left[\sqrt{2}\,t\right]\right)\text{UnitStep}[t]$$

The Fourier Transform

The Fourier transform can sometimes be considered a special case of the Laplace transform, evaluated where the real part of the transform variable *s* is zero. It is often used for frequency analysis of signals. Signals and Systems uses the standard definition

$$F(\omega) = \mathcal{F}\{f(t)\} = \int_{-\infty}^{+\infty} f(t)\, e^{-j\,\omega\, t}\, d\, t$$

The inverse transform is defined as

$$f(t) = \mathcal{F}^{-1}\{F(\omega)\} = \frac{1}{2\,\pi} \int_{-\infty}^{+\infty} F(\omega)\, e^{j\,\omega\, t}\, d\, \omega$$

Note that the nonsymmetric form is used, which parallels the Laplace transform. The actual algorithm depends on table lookup and transformation rules that apply the properties of the transform. The Fourier transform applies to continuous signals. The discrete equivalents are the discrete-time Fourier transform and the discrete Fourier transform, documented in the next section.

`FourierTransform[`*expr*`, `*t*`, `*w*`]`	perform a Fourier transform from the variable *t* to *w*
`FourierTransform[`*expr*`, {`t_1`, `t_2`, ..., `t_n`}, {`w_1`, `w_2`, ..., `w_n`}]`	perform a multidimensional Fourier transform from t_i to w_i
`FourierTransform[`*t*`, `*w*`][`*expr*`]`	nonevaluating operator form of the Fourier transform
`InverseFourierTransform[`*expr*`, `*w*`, `*t*`]`	perform an inverse Fourier transform
`InverseFourierTransform[`*w*`, `*t*`][`*expr*`]`	operator form of the inverse Fourier transform

The Fourier transform and its inverse.

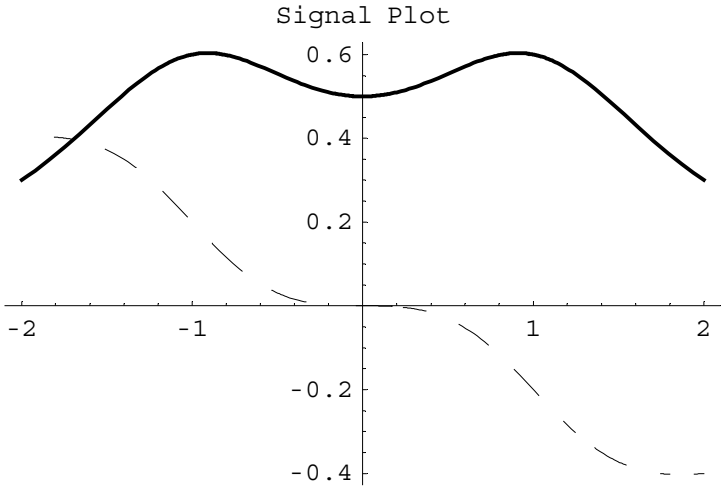
- Here is a Fourier transform.

```
In[16]:= FourierTransform[
             Cos[t] Exp[-t] UnitStep[t],
             t, w
         ]
```

$$\text{Out[16]= FourierTransformData}\left[\frac{1+i\,w}{1+(1+i\,w)^2}, \text{TransformVariables[w]}\right]$$

- Here is a plot of the transformed result.

```
In[17]:= SignalPlot[%, {w, -2, 2}]
```




```
Out[17]= - Graphics -
```

As with `LaplaceTransform`, a special data type is returned, identifying the transform used in the computation. `FourierTransform` accepts all of the options of `LaplaceTransform` and attaches the same meaning to them.

- Here is a transform using the `TransformDirection` option. This performs a left-sided transform (from `-Infinity` to `0`).

```
In[18]:= FourierTransform[
             Exp[t^2],
             t, w,
             TransformDirection -> LeftSided
         ]
```

$$\text{Out[18]= FourierTransformData}\left[\mathbb{i}\, e^{\frac{w^2}{4}} \sqrt{\pi}\, \left(1 + \text{Erf}\left[\frac{w}{2}\right]\right), \text{TransformVariables}[w]\right]$$

- A rectangular pulse centered on the origin is separable, and easily transformed in any number of dimensions.

```
In[19]:= FourierTransform[
             ContinuousPulse[{1, 1, 1},
                  {t1 + 1/2, t2 + 1/2, t3 + 1/2}],
             {t1, t2, t3},{w1, w2, w3}
         ]
```

$$\text{Out[19]= FourierTransformData}\left[\text{Sinc}\left[\frac{w1}{2}\right] \text{Sinc}\left[\frac{w2}{2}\right] \text{Sinc}\left[\frac{w3}{2}\right], \text{TransformVariables}[\{w1, w2, w3\}]\right]$$

The inverse transform employs all of the same options as the forward transform. It is used in much the same way as the forward transform, but does not return a special data type.

- This symbolic inverse Fourier transform demonstrates that the rule base "knows" the modulation theorem, which in this case causes the transform of the modulated signal to be a pulse of width `a` centered at zero.

```
In[20]:= InverseFourierTransform[
            1/2 (ContinuousPulse[a, w + a/2 - w0] +
              ContinuousPulse[a, w + a/2 + w0]),
            w, t
         ]
```

$$\text{Out[20]= } \frac{a \text{ Cos}[t\, w0] \text{ Sign}[a] \text{ Sinc}\left[\frac{a\, t}{2}\right]}{2\pi}$$

- Here is an inverse transform that cannot be performed in closed form. However, a series approximation can be made to give us an inexact result in terms of derivatives of Dirac delta functions.

```
In[21]:= InverseFourierTransform[
           Tan[w],
           w, t,
           SeriesTerms -> 8
         ]
```

$$\text{Out[21]}= \frac{1}{2\pi}\left(-2\,i\,\pi\,\text{DiracDelta}'[t] + \frac{2}{3}\,i\,\pi\,\text{DiracDelta}^{(3)}[t] - \frac{4}{15}\,i\,\pi\,\text{DiracDelta}^{(5)}[t] + \frac{34}{315}\,i\,\pi\,\text{DiracDelta}^{(7)}[t]\right)$$

■ 5.2 Transforms of Discrete Signals

The Z Transform

The Z transform is essentially the discrete equivalent of the Laplace transform. It produces a continuous function from a sequence via the following formula

$$X(z) = \mathcal{Z}\{x[n]\} = \sum_{n=-\infty}^{+\infty} x[n]\, z^{-n}$$

The region of convergence is an annulus (whose interior radius can be zero and exterior radius infinite). The inverse transform is

$$x[n] = \mathcal{Z}^{-1}\{X(z)\} = \frac{1}{2\pi j}\oint_C X(z)\, z^{n-1}\, d\,z$$

where C is a counterclockwise contour encircling the origin. As with the continuous transforms, Signals and Systems primarily uses table lookup and application of the properties of the transform to perform the computation.

ZTransform[*expr*, *n*, *z*]	perform a Z transform from the variable *n* to *z*
ZTransform[*expr*, {n_1, n_2, ..., n_n}, {z_1, z_2, ..., z_n}]	perform a multidimensional Z transform from n_i to z_i
ZTransform[*n*, *z*][*expr*]	nonevaluating operator form of the Z transform
InverseZTransform[*expr*, *z*, *n*]	perform an inverse Z transform
InverseZTransform[*z*, *n*][*expr*]	operator form of the inverse Z transform

The Z transform and its inverse.

The syntax of the discrete transforms is essentially like that of the continuous transforms and the transforms that come with *Mathematica*. Like the continuous transforms provided with Signals and Systems, a nonevaluating operator form of the transform is provided for manipulation of cascaded systems.

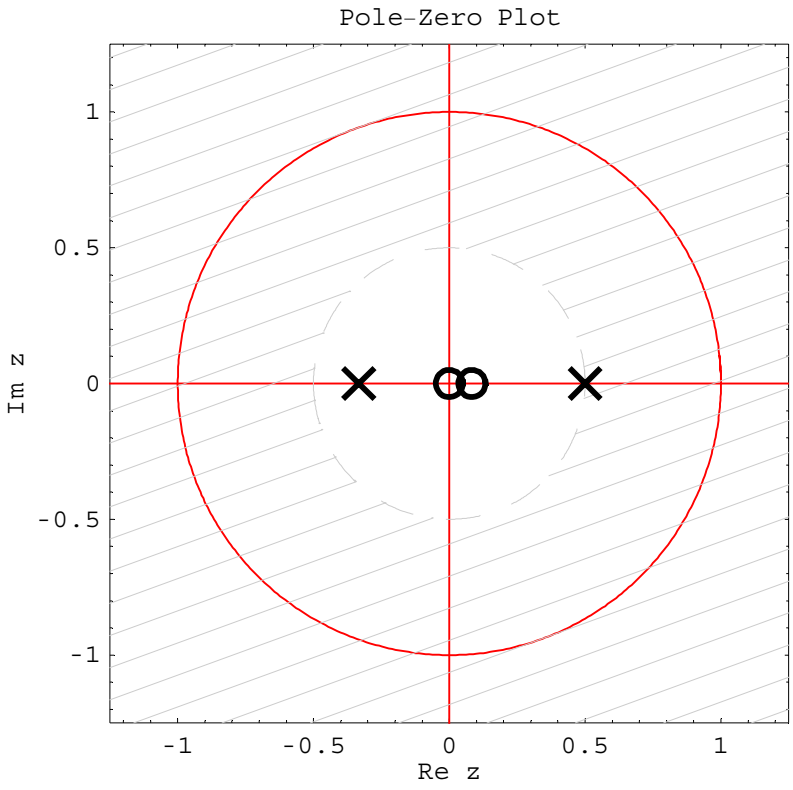
- This is the Z transform of the sum of two exponentials.

```
In[22]:= ZTransform[
            ((1/2)^n + (-1/3)^n) DiscreteStep[n],
            n, z
         ]
```

$$\text{Out[22]= ZTransformData}\left[\frac{z - 12\,z^2}{1 + z - 6\,z^2},\ \text{RegionOfConvergence}\left[\frac{1}{2}, \infty\right],\ \text{TransformVariables[z]}\right]$$

- We can visualize the poles, zeros, and region of convergence of a rational transform.

```
In[23]:= PoleZeroPlot[%]
```




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

options name	*default value*	
Justification	None	additional information to be displayed about the generation of the transform
SimplifyOutput	True	whether to attempt simplification of results
StandardFormula	False	whether to try to perform the summation directly
TransformDirection	$TransformDirection	direction of the transform in the "time" domain
TransformPairs	{}	rules to employ in attempting to transform unknown functions

Options for ZTransform.

The Z transform has most of the options of the Laplace transform. The TransformDirection option controls whether the transform is left-sided, right-sided, or bilateral. The Justification option causes information about the steps taken in the transform to be printed when set to Automatic or All. TransformPairs allows user-specified transformation rules to be given to supplement the built-in rule base. If the rule base fails to compute the transform, the option StandardFormula allows the transform function to attempt to use the defintion given by the summation. SimplifyOutput allows additional simplification of the results to be attempted automatically. It can be set to None to reduce computation time on examples where additional simplification is unlikely to be useful.

- The TransformPairs option can be used to specify abstract transformations, as in the transformation of this system equation, where x[n] is the input signal and y[n] is the output signal. Note that Normal is wrapped around the transform to extract the equation from the transform data type for easy manipulation.

```
In[24]:= Normal[ZTransform[
           y[n] == x[n] - (1/4) y[n - 2],
           n, z,
           TransformPairs -> {y[n] :> Y[z], x[n] :> X[z]}
         ]]
```

$$\text{Out[24]= } Y[z] == X[z] - \frac{Y[z]}{4\,z^2}$$

- The system transfer function can be determined by solving the above system for `Y[z]` and dividing by `X[z]`.

```
In[25]:= (Y[z]/.First[Solve[%, Y[z]]])/X[z]
```

$$\text{Out[25]=}\ \frac{4\,z^2}{1+4\,z^2}$$

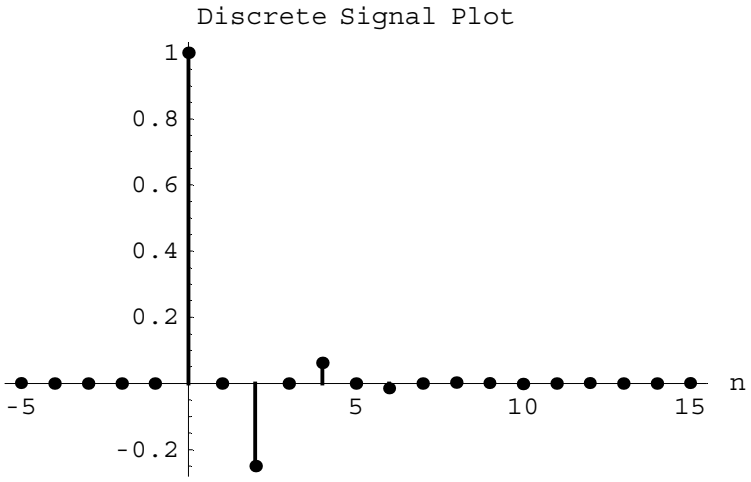
- Taking the inverse transform of the system transfer function gives us the impulse response.

```
In[26]:= InverseZTransform[%, z, n]
```

$$\text{Out[26]=}\ \text{Upsample}_{2,n}\left[\left(-\frac{1}{4}\right)^{n}\text{DiscreteStep}[n]\right]$$

- This is the impulse response of the system we are working with.

```
In[27]:= DiscreteSignalPlot[%, {n, -5, 15}]
```




```
Out[27]= - Graphics -
```

Without region of convergence information, a Z transform is not unique. If you wish to perform an inverse transform that requires specification of the region of convergence, you must enter the transform object directly.

- Here is a multidimensional separable Z transform. Note the region of convergence.

```
In[28]:= ZTransform[
             a^n1 b^n2 DiscreteStep[{n1, n2}],
             {n1, n2}, {z1, z2}
         ]
```

```
Out[28]= ZTransformData[ z1 z2 / ((a - z1) (b - z2)) ,
          RegionOfConvergence[{Abs[a], Abs[b]}, {∞, ∞}],
          TransformVariables[{z1, z2}]]
```

- Here is an inverse transform where we take the function that results from the previous computation, but change the region of convergence. Note how the result varies compared to the input to the preceding example.

```
In[29]:= InverseZTransform[
             ZTransformData[Normal[%],
                 RegionOfConvergence[
                    {0, Abs[b]},
                    {Abs[a], Infinity}
                 ],
                 TransformVariables[{z1, z2}]
             ],
             {z1, z2}, {n1, n2}
         ]
```

```
Out[29]= -a^n1 b^n2 DiscreteStep[-1 - n1] DiscreteStep[n2]
```

SeriesTerms -> *n*	perform a series expansion of the function to be transformed if no standard transform rules are applicable
DecompositionPrecision -> *n*	if partial fractions decomposition is used in the computation, perform it to the specified precision

Options for inverse Z transform.

Since the inverse Z transform is transforming a continuous function, it accepts the SeriesTerms option. This will attempt a series expansion of the function to be transformed to a specified number of terms if no transform can be found. Note that SeriesTerms can also be set to None, indicating that the method is not to be attempted. The inverse Z transform also employs the DecompositionPrecision option. For exact decomposition, the option should be set to Infinity.

- Here is an inverse Z transform performed by the `SeriesTerms` option. The result is only an approximation to what the actual transform would be. `Expand` is used here to arrange the terms in a more attractive form.

```
In[30]:= Expand[InverseZTransform[
             BesselJ[1, z],
             z, n,
             SeriesTerms -> 8
         ]]

Out[30]= 2^n DiscreteDelta[1 + n] - 2^(2+2 n) DiscreteDelta[3 + n] +
          2^(-12-n) 3^(4+n) DiscreteDelta[5 + n] - 2^(-18-n) 3^(5+n) DiscreteDelta[7 + n]
```

The `StandardFormula` option is not available for inverse Z transforms, as standard integration technology is not able to handle most of the contour integrals from transforms that cannot be computed by the standard Z transform rule base.

The Discrete-Time Fourier Transform

The discrete-time Fourier transform is related to the Z transform, and can under certain circumstances be viewed as a special case of the Z transform, with convergence inside the unit circle. Like the Z transform, it accepts a discrete function as input and returns a continuous function. The standard formula is

$$X(e^{j\omega}) = \mathcal{F}_{DT}\{x[n]\} = \sum_{n=-\infty}^{+\infty} x[n]\, e^{-j\omega n}$$

and for the inverse transform

$$x[n] = \mathcal{F}_{DT}^{-1}\{X(e^{j\omega})\} = \frac{1}{2\pi}\int_{-\pi}^{\pi} X(e^{j\omega})\, e^{j\omega n}\, d\omega$$

Because of the similarities between this transform and the Z transform, the implementation in Signals and Systems simply adds rules for those cases where the discrete-time Fourier transform is different, then call the Z transform with the appropriate variable substitution for all other cases.

DiscreteTimeFourierTransform[*expr*, *n*, *w*]	perform a discrete-time Fourier transform from the variable *n* to *w*
DiscreteTimeFourierTransform[*expr*, {n_1, n_2, ..., n_n}, {w_1, w_2, ..., w_n}]	perform a multidimensional discrete-time Fourier transform from n_i to w_i
DiscreteTimeFourierTransform[*n*, *w*][*expr*]	nonevaluating operator form of the discrete-time Fourier transform
InverseDiscreteTimeFourierTransform[*expr*, *w*, *n*]	perform an inverse discrete-time Fourier transform
InverseDiscreteTimeFourierTransform[*w*, *n*][*expr*]	operator form of the inverse discrete-time Fourier transform

The discrete-time Fourier transform and its inverse.

The syntax of the discrete-time Fourier transform is much like that of the Z transform.

- All the transform functions, including the discrete-time Fourier transform, accept symbolic input.

```
In[31]:= DiscreteTimeFourierTransform[
             a^n DiscreteStep[n - 4],
             n, w
         ]
```

$$\text{Out[31]= DTFTData}\left[\frac{a^4}{-a\, e^{3\,i\,w} + e^{4\,i\,w}},\ \text{TransformVariables[w]}\right]$$

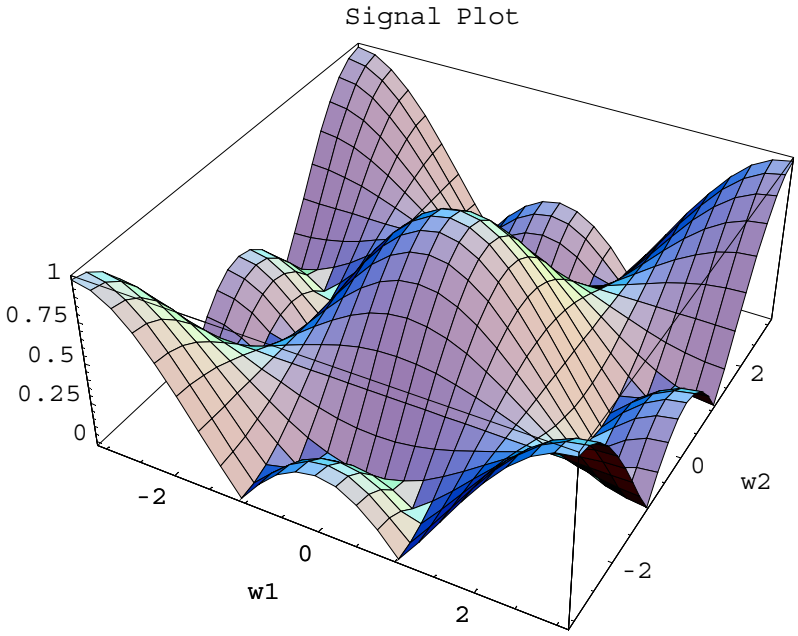
- This is the two-dimensional discrete-time Fourier transform of a particular pattern of samples around the origin.

```
In[32]:= DiscreteTimeFourierTransform[
             (1/6) DiscreteImpulse[n1 - 1, n2 - 1] +
               (1/6) DiscreteImpulse[n1 + 1, n2 - 1] +
               (1/6) DiscreteImpulse[n1 - 1, n2 + 1] +
               (1/6) DiscreteImpulse[n1 + 1, n2 + 1] +
               (1/3) DiscreteImpulse[n1, n2],
             {n1, n2}, {w1, w2}
         ]
```

$$\text{Out[32]= DTFTData}\left[\frac{1}{3}\ (1 + 2\ \text{Cos[w1]}\ \text{Cos[w2]}),\ \text{TransformVariables[\{w1, w2\}]}\right]$$

- This is the magnitude of the transform, plotted. Note that to perform mathematical computations with a transform result, it must be extracted from the data structure that the transform returns.

```
In[33]:= SignalPlot3D[Abs[First[%]],
             {w1, -Pi, Pi}, {w2, -Pi, Pi}
         ]
```




```
Out[33]= - SurfaceGraphics -
```

`DiscreteTimeFourierTransform` accepts the same options as `ZTransform`, with the same meanings. Similarly, the inverse transform uses the same options as the inverse Z transform.

- Here is a simple example of an abstract transform pair. Note that the `TransformPairs` option will only be effective if the standard properties of the transform can be applied to simplify the input to the point that the user-specified transform pair can be applied.

```
In[34]:= DiscreteTimeFourierTransform[
            x[n + 3],
          n, w, TransformPairs -> {x[n] :> X[w]}
          ]

Out[34]= DTFTData[e^(3 i w) X[w], TransformVariables[w]]
```

- The inverse transform can also make use of the `TransformPairs` option.

```
In[35]:= InverseDiscreteTimeFourierTransform[%,
            w, n,
            TransformPairs -> {X[w] :> x[n]}
          ]

Out[35]= x[3 + n]
```

The Discrete Fourier Transform

Given a finite sequence of length N, the Fourier transform can be represented by N uniformly distributed samples of the discrete-time Fourier transform. This is known as a discrete Fourier transform.

$$X[k] = \mathcal{F}_D\{x[n]\} = \sum_{n=0}^{N-1} x[n]\, e^{\frac{-j2\pi kn}{N}}$$

The inverse transform is written as

$$x[n] = \mathcal{F}_D^{-1}\{X[k]\} = \frac{1}{N}\sum_{k=0}^{N-1} X[k]\, e^{\frac{j2\pi kn}{N}}$$

Unlike the built-in *Mathematica* function `Fourier` which computes numeric discrete Fourier transforms, the Signals and Systems function `DiscreteFourierTransform` computes symbolic transforms.

DiscreteFourierTransform[*expr*, *N*, *n*, *k*]	perform a discrete Fourier transform from the variable *n* to *k* for a sequence of length *N*
DiscreteFourierTransform[*expr*, {N_1, N_2, …, N_n}, {n_1, n_2, …, n_n}, {k_1, k_2, …, k_n}]	perform a multidimensional discrete Fourier transform from n_i to k_i with N_i samples in each variable
DiscreteFourierTransform[*N*, *n*, *k*][*expr*]	nonevaluating operator form of the discrete Fourier transform
InverseDiscreteFourierTransform[*expr*, *N*, *k*, *n*]	perform an inverse discrete Fourier transform
InverseDiscreteFourierTransform[*N*, *k*, *n*][*expr*]	operator form of the inverse discrete Fourier transform

The discrete Fourier transform and its inverse.

Because of the symbolic nature of the discrete Fourier transform as implemented in Signals and Systems, you can specify a formula and the number of samples N over which the transform is to be applied. The sequence to be transformed is assumed to run from 0 to $N-1$. The result of the transform is periodic; this is emphasized by use of the Periodic operator in the returned data object.

- Here is a simple transform demonstrating the symbolic aspects of the DiscreteFourierTransform function. Here, the sequence has a length of 10.

```
In[36]:= DiscreteFourierTransform[Sin[n], 10, n, k]

Out[36]= DFTData[
           -(1/2) i Periodic_{10,k}[10 Summation_{l1=-∞}^{∞} [DiracDelta[k + 10 l1 - 5/π]] -
             10 Summation_{l2=-∞}^{∞} [DiracDelta[k + 5 (2 l2 + 1/π)]]],
           Start[0], Finish[9], TransformVariables[k]]
```

■ An inverse transform is computed in a similar fashion.

```
In[37]:= InverseDiscreteFourierTransform[Sin[k], 10, k, n]

Out[37]= -1/20 i (CircularShift_Mod[n, 10] in n) [
           2 π (-DiscreteDelta[n - 5/π] + DiscreteDelta[n + 5/π])]
```

options name	*default value*	
DecompositionPrecision	Infinity	precision to be used if partial fractions decomposition is employed
Justificiation	None	what degree of information about the transform process should be displayed
SimplifyOutput	All	what degree of output simplification should be attempted
StandardFormula	False	whether to try the standard summation formula for computing the transform
TransformPairs	{}	user-specified transform pairs to attempt

Options for DiscreteFourierTransform and InverseDiscreteFourierTransform.

The DiscreteFourierTransform and its inverse use some of the Z and inverse Z transform options. They take essentially the same meanings, but since the selection of applicable options is different, they are listed again here. Note in particular that the TransformDirection option is not applicable to the discrete Fourier transform, nor is the SeriesTerms option.

- When `Justification` is set to `Automatic`, not as much additional information is generated. This tells us that the transform is computed by way of the discrete-time Fourier transform.

```
In[38]:= DiscreteFourierTransform[
             DigitalFIRFilter[
                 {h[0], h[1], h[2], h[3], h[4]},
                 n
             ],
             5, n, w,
             Justification -> Automatic
         ]

            ( after taking the forward DTFT of

              DigitalFIRFilter[{h[0], h[1], h[2], h[3], h[4]}, n]

              with respect to n which yields

              h[0] + ℯ^(-i w) h[1] + ℯ^(-2 i w) h[2] + ℯ^(-3 i w) h[3] + ℯ^(-4 i w) h[4]

              and adjusting the result )
```

$$\text{Out[38]= DFTData}\left[\text{Periodic}_{5,w}\left[h[0]+e^{-\frac{2}{5}i\pi w}h[1]+e^{-\frac{4}{5}i\pi w}h[2]+e^{-\frac{6}{5}i\pi w}h[3]+e^{-\frac{8}{5}i\pi w}h[4]\right],\ \text{Start}[0],\ \text{Finish}[4],\ \text{TransformVariables}[w]\right]$$

`DiscreteFourierTransform[`*vector*`]`	the discrete Fourier transform of a numeric vector of data
`InverseDiscreteFourierTransform[`*vector*`]`	the inverse discrete Fourier transform of a numeric vector of data

Special syntax for transforming a numeric vector.

For convenience, the forward and inverse discrete Fourier transforms have a syntax that allows a numeric vector to be passed directly to the transform function, generating a vector output. The behavior is similar to that of the built-in function `Fourier`, but with the definition of the discrete Fourier transform used in Signals and Systems. The usual options are not available with this syntax, and the symbolic techniques are not used. The same degree of rigor used by the primary algorithm is also not employed, as the output is presented simply as a vector rather than a periodic signal. However, for certain computations (such as filter design), it is sometimes useful to refer to a vector of data that is implicitly assumed to start at

0, with the output implicitly assumed to be periodic (assumptions made explicit when using `SampledData` objects). The input must be numeric when this syntax is employed.

- Here is a numeric transform of a vector of data.

```
In[39]:= DiscreteFourierTransform[
             Table[2^(-n), {n, 0, 10}]//N
         ]

Out[39]= {1.99902 + 0. i, 1.41675 - 0.661017 i, 0.948862 - 0.544695 i,
          0.76896 - 0.355285 i, 0.696525 - 0.198277 i, 0.669395 - 0.0637239 i,
          0.669395 + 0.0637239 i, 0.696525 + 0.198277 i,
          0.76896 + 0.355285 i, 0.948862 + 0.544695 i, 1.41675 + 0.661017 i}
```

- This inverse transform might represent the impulse response of a sampled ideal filter frequency response.

```
In[40]:= InverseDiscreteFourierTransform[
             {1, 1, 1, 1, 0, 0, 0, 0}//N
         ]

Out[40]= {0.5 + 0. i, 0.125 + 0.301777 i, 0. + 0. i, 0.125 + 0.0517767 i,
          0. + 0. i, 0.125 - 0.0517767 i, 0. + 0. i, 0.125 - 0.301777 i}
```

■ 5.3 Information from Transforms

Each returned transform object contains a variety of useful components. In addition, other functions can extract information such as the stability of a transform from the object.

Stability

For the Laplace and Z transforms, the stability of the signal or system under consideration can be derived from the region of convergence of the transform. In the case of the Z transform, the system is stable only if the unit circle is included in the region of convergence, while for the Laplace transform, the imaginary axis must be in the region of convergence.

`SignalStability[` *transform*`]`	return `True`, `False`, or the conditions required for the signal to be stable

Determining stability from a transform object.

The SignalStability function examines the region of convergence information in a transform data object for the stability criteria described above. If the stability can definitely be determined, the function returns True or False. If the convergence criteria include abstract parameters, SignalStability returns the conditions that those parameters must meet for the signal to be stable. The conditions are returned in the form of a logical conjunction.

- Here is a Z transform.

```
In[41]:= ZTransform[
             (1/5)^n Exp[n]/20 DiscreteStep[n],
             n, z
         ]
```

```
Out[41]= ZTransformData[ z/(-4 e + 20 z),
          RegionOfConvergence[e/5, ∞], TransformVariables[z]]
```

- From the region of convergence, we conclude that this is stable.

```
In[42]:= SignalStability[%]
```

```
Out[42]= True
```

- This multidimensional transform from earlier in the chapter is stable when the conditions returned in this example are met.

```
In[43]:= SignalStability[
            ZTransform[
                -a^n1 b^n2 DiscreteStep[{-n1 - 1, n2}],
               {n1, n2}, {z1, z2}
           ]
         ]
```

```
Out[43]= 1 < Abs[a] && Abs[b] < 1
```

Assumptions

Some transforms can only be performed if certain conditions are met on parameters or variables involved in the transform. Signals and Systems currently maintains this information as part of the command history, via the function TransformAssumptions.

`TransformAssumptions[n]`	return a logical conjunction of assumptions made by transforms computed by `In[n]`
`TransformAssumptions[ Out[n]]` or `TransformAssumptions[%n]`	alternate syntax for the `TransformAssumptions` command for ease of use

Assumptions made by transforms during a computation.

During a computation, the transform functions associate any required assumptions with a history mechanism kept in `TransformAssumptions` of the current value of `$Line`. You can retrieve the assumptions by calling `TransformAssumptions` with the input line you want to find out about. (If a negative value is given, `TransformAssumptions` will count backwards from the current line.) If `True` is returned, the assumptions are met by current criteria placed on parameters, or no assumptions were necessary. Otherwise, the assumptions will be returned in the form of a logical conjunction.

- Here is a Laplace transform.

```
In[44]:= LaplaceTransform[
             ExpIntegralEi[n t] UnitStep[t],
             t, s
         ]
```

$$\text{Out[44]= LaplaceTransformData}\left[-\frac{\text{Log}\left[1-\frac{s}{n}\right]}{s},\ \text{RegionOfConvergence}[0,\infty],\ \text{TransformVariables}[s]\right]$$

- This retrieves the assumptions made during the previous input.

```
In[45]:= TransformAssumptions[-1]

Out[45]= Negative[n]
```

- This syntax allows a more familiar notation to be used. Note, however, that the `%%` is not actually evaluated; instead, the `-2` is retrieved from the `Out[-2]` which `%%` represents.

```
In[46]:= TransformAssumptions[%%]

Out[46]= Negative[n]
```

Transform Object Parts

The forward transforms return a data object that contains important information about the transform, such as the region of convergence. The most important information (the function that results from the transformation) is quite visible to the user, but, for programmatic manipulation, it is best to extract the parts of the data object using special functions. This is because the data type is subject to change, as additional information about the transform is retained.

`RegionOfConvergence[`*object*`]`	extract the bounds on the region of convergence
`TransformFunction[`*object*`]`	extract the function resulting from the transform
`TransformRange[`*object*`]`	extract the starting point and endpoint of the periodic part of the discrete Fourier transform
`TransformVariables[`*object*`]`	extract the variables that the transform was performed with

Functions for extracting parts of transform objects.

As per the usual *Mathematica* convention for converting data objects, `Normal` can also be used to extract the function resulting from the transform.

- Here is a Laplace transform.

```
In[47]:= trans = LaplaceTransform[
              Exp[t] UnitStep[t],
              t, s
          ]

Out[47]= LaplaceTransformData[1/(-1 + s),
           RegionOfConvergence[1, ∞], TransformVariables[s]]
```

- This is the function resulting from the transform.

```
In[48]:= TransformFunction[trans]

Out[48]= 1/(-1 + s)
```

Laplace and Z transforms retain information about the region of convergence of the transform in the transform variable. It is not necessary that this information be retained for the other provided transforms. The upper and lower bounds represent different things for each of these tranforms. For the Laplace transform, the region of convergence is a strip in the complex plane bounded by constant real numbers. The upper and lower bounds are these real values. The Z transform, on the other hand, converges in an annulus in the complex plane, and the upper and lower bounds represent the absolute value of the points on the boundary. In the multidimensional case, these take the form `RegionOfConvergence`[{l_1, l_2, …}, {u_1, u_2, …}].

- The region of convergence is extracted in a similar fashion. The lower bounds are in the first argument, while upper bounds are in the second argument.

```
In[49]:= RegionOfConvergence[trans]

Out[49]= RegionOfConvergence[1, ∞]
```

- Now we extract the variables that the function was transformed to. This is necessary to distinguish variables from parameters in the transform object.

```
In[50]:= TransformVariables[trans]

Out[50]= s
```

`DFTData[` *function*`, … ]`	data object resulting from a discrete Fourier transform
`DTFTData[` *function*`, … ]`	data object resulting from a discrete-time Fourier transform
`FourierTransformData[` *function*`, … ]`	data object resulting from a Fourier transform
`LaplaceTransformData[` *function*`, … ]`	data object resulting from a Laplace transform
`ZTransformData[` *function*`, … ]`	data object resulting from a Z transform

Data objects resulting from forward transforms.

Because the contents of a transform data object are in principle not fixed, it is better to access the parts of such an object by the functions listed previously, not by using `Part` and depending on the components to be in particular locations.

■ 5.4 Solving Differential and Recurrence Equations

Mathematical transforms play a wide variety of roles in signal processing. For instance, they can be used to solve differential and recurrence equations. As a demonstration of this utility, Signals and Systems includes the functions `LaplaceDSolve` and `ZRSolve`. They can solve linear constant coefficient differential and recurrence equations, respectively.

These equations can also be handled by the standard *Mathematica* functions `DSolve` and `RSolve`. However, the standard functions employ a variety of techniques. If you specifically want to apply a transform technique, then these supplementary functions provided by Signals and Systems can be useful. `LaplaceDSolve` and `ZRSolve` also make use of the bilateral transforms, which allows a broader range of solutions than the traditional right-sided transforms permit.

`LaplaceDSolve[` $a_1\ y[t] + a_2\ y'[t] + \ldots == f[t],\ y[t],\ t]$	solve a linear constant coefficient differential equation
`LaplaceDSolve[` {*equation*, $y[t_0] == v_1$, $y'[t_0] == v_2, \ldots\}, y[t], t]$	solve equation with boundary conditions
`ZRSolve[` $a_1\ y[n] + a_2\ y[n-1] + \ldots == f[n],\ y[n],\ n]$	solve a linear constant coefficient recurrence equation
`ZRSolve[` {*equation*, $y[0] == c_1$, $y[1] == c_2, \ldots\}, y[n], n]$	solve equation with initial conditions

The transform-based equation solvers.

The syntax of these equation solvers mimics the standard `DSolve` and `RSolve` notations. The main differences are in the class of equations that can be solved and in the allowed boundary conditions. The equations are limited to linear constant coefficient terms, with only

the driving term allowed to be dependent on the variable (that is, the equation can be nonhomogenous). Systems of equations cannot currently be handled.

If initial conditions are not given, they are assumed to be zero for `ZRSolve`. `LaplaceDSolve` assumes that they are symbolic constants `C[1]`, `C[2]`, and so on, with constant `C[n]` corresponding to the derivative of order $n - 1$ of the function at 0.

- Here is the solution to a second-order difference equation, with an initial point specified.

```
In[51]:= ZRSolve[
           {y[n-2] + 1/2 y[n-1] + 1/4 y[n] == 0,
            y[1] == 1},
           y[n], n
         ]
```

$$\text{Out[51]= } \left\{\left\{y[n] \to \frac{2^n \,\text{DiscreteStep}[n] \,\text{Sin}\left[\frac{2 n \pi}{3}\right]}{\sqrt{3}}\right\}\right\}$$

- Here is a second-order differential equation. Initial values for the function are specified.

```
In[52]:= LaplaceDSolve[
             {y''[t] + 3/2 y'[t] + 1/2 y[t] ==
                              Exp[a t] UnitStep[t],
              y[0] == 4, y'[0] == 0},
             y[t], t
         ]
```

$$\text{Out[52]= } \left\{\left\{y[t] \to \frac{1}{1 + 3\,a + 2\,a^2}\,(2\,e^{-t} \left(-1 + 2\,e^{t/2} + e^{t+a\,t} + 2\,a\,\left(-2 + 5\,e^{t/2}\right) + a^2\,\left(-4 + 8\,e^{t/2}\right)\right)\,\text{UnitStep}[t])\right\}\right\}$$

option name	*default value*	
`Justification`	`None`	whether to print information about the steps taken in the computation
`TransformDirection`	`RightSided`	perform the transform over one of the domains `TwoSided`, `LeftSided`, or `RightSided`

Options for the solving functions.

The use of the bilateral transform allows both causal and anticausal sequences to be handled by `ZRSolve`. The `TransformDirection` option can be specified with either solver; it is

passed along to the corresponding transform function.

The `Justification` option is much like that used elsewhere. When set to `None`, no additional information is provided; when `Automatic`, some information about the steps taken is printed; and when `All`, detailed output is given.

- Here is a difference equation solved with the `Justification` option activated. More information can be generated by setting the option to `All`.

```
In[53]:= ZRSolve[
           {y[n-2] + 1/2 y[n-1] + 1/4 y[n] == 0,
            y[0] == 1, y[1] == 0},
           y[n], n, Justification -> Automatic
         ]
```

Solving for y[n] in the difference equation

augmented by the initial conditions:

$$y[-2+n] + \frac{1}{2}\,y[-1+n] + \frac{y[n]}{4} = \frac{1}{2}\,\text{DiscreteDelta}[-1+n] + \frac{\text{DiscreteDelta}[n]}{4}$$

After taking the z-transform of both sides and

solving for the z-transform of y[n]:

$$\frac{1+\frac{2}{z}}{1+\frac{4}{z^2}+\frac{2}{z}}$$

Inverse transforming this gives y[n]:

$$\left\{\left\{y[n] \to \frac{1}{3}\,2^n\left(2\sqrt{3}\,\text{DiscreteStep}[-1+n]\,\text{Sin}\left[\frac{2n\pi}{3}\right] + \text{DiscreteStep}[n]\left(3\,\text{Cos}\left[\frac{2n\pi}{3}\right] - \sqrt{3}\,\text{Sin}\left[\frac{2n\pi}{3}\right]\right)\right)\right\}\right\}$$