User's Guide to the Signal Processing Packages and Notebooks for Implementing Linear System Theory in *Mathematica* 1.2 and 2.0

Brian L. Evans, James H. McClellan, Lina J. Karam, Kevin A. West, Wallace B. McClure, Terry A. Crone, Todd R. Bacon, and James A. Proctor

Digital Signal Processing Laboratory
School of Electrical Engineering
Georgia Institute of Technology
Atlanta, GA 30332-0250

EMAIL: evans@eedsp.gatech.edu

Abstract

We describe a hierarchical set of packages to perform basic analyses of signals (functions) and systems (operators). The packages are based on transform theory, and implement a general mechanism for encoding knowledge about transforms, so their applicability extends beyond signal processing to any field where transform analysis is needed. We support (bilateral) z- and Laplace transforms, as well as continuous-time, discrete-time, and discrete Fourier transforms, all in arbitrary dimension. These rule bases can fully justify their answers. When they cannot find a particular transform, users have the choice of either specifying the missing transform pair(s) or letting the transform apply its definition using built-in *Mathematica* operations.

The packages can perform a variety of symbolic, graphical, and numerical operations on signals and systems. In symbolic terms, the packages can solve linear constant-coefficient difference and differential equations by using the z and Laplace transform

capabilities, respectively. Another supported symbolic operation is convolution which is implemented in both the discrete and continuous domains. Symbolic analyses include simplification of signal processing expressions, finding optimal implementations of algorithms, and reasoning about properties of signals, such as stability. Plotting capabilities for 1-D and 2-D signals include continuous and discrete plots, magnitude and phase plots, and root-locus plots. Pole-zero diagrams for 1-D and 2-D signals are based on their z or Laplace transforms and include the region of convergence.

The packages and the accompanying Notebooks work simultaneously under *Mathematica* 1.2, 2.0, and 2.1. Four of the Notebooks are tutorials on the topics of analog filter design, discrete Fourier analysis, discrete/continuous convolution, and the z-transform (in three parts). Another set of Notebooks provide on-line help. A third set of Notebooks form the handout we give students learning *Mathematica* and our extensions. Two supplemental sets of Notebooks also exist. One set helps students evaluate their own knowledge about signal processing. Another set is written by an undergraduate student explaining difficult concepts in signals and systems.

Introduction

Transform theory is a powerful tool employed by scientists [Oberhettinger and Badii 1973], mathematicians [Churchill 1958], and engineers [Lathi 1983; Muth 1978; Oppenheim and Schafer 1989]. One important use of this theory is to transform equations involving integrals and derivatives into algebraic equations that are easier to solve. The solutions to the original equations are obtained by inverse transforming the algebraic solutions. This is the case when the Laplace transform is applied to linear differential equations with constant coefficients, or when the z-transform is applied to linear difference equations with constant coefficients.

This process is one of the staples of *signal processing*, the area of engineering that deals with the design and analysis of systems that act on signals in prescribed ways. A *signal* is essentially a function of one or more variables; often, in the one-variable case, the variable is time. The variable can be continuous (for an *analog signal*), or discrete (for a *digital*, or *sampled*, signal). Thus a one-dimensional discrete-time signal is a finite or infinite sequence of values, or *samples*.

A *system* is a mathematical operator that transforms one signal into another, according to characteristic rules. *Filters*, or time-invariant linear systems, form a particularly important class of systems, eminently suited to the application of transform methods. A filter is characterized by a function, its *impulse response*, in terms of which the filter's action on any signal can be determined. The output signal is the *convolution* of the input signal with the filter's impulse response; the operation of convolution involves an integral for continuous signals and a summation for discrete ones. Using transforms, one can reduce the task of convolving these two functions to that of multiplying two algebraic expressions— the functions' transforms— and then inverse transforming the product.

For an example of analog system design, consider the transmission of AM radio signals. Each AM radio station broadcasts at a specific frequency f_0 which is between 550 kHz and 1600 kHz and is a multiple of 10 kHz [Lathi 1983]. In order to avoid overlap with other stations, a station must transmit its signal between the frequencies $f_0 - 5\,\text{kHz}$ and $f_0 + 5\,\text{kHz}$. Before transmission, a station filters its signal so that frequencies above 5 kHz are rejected and frequencies below 5 kHz are passed. This filtered signal is then shifted by f_0 in the frequency domain and transmitted. The entire design and analysis of this system can be carried out in the frequency domain: the Laplace transform would be used for the design and implementation of the 5 kHz low-pass filter, and the Fourier transform would be used to analyze the transmission system.

By contrast, digital signal processing (DSP) deals with discrete-time signals (sequences) and systems that act on them [Oppenheim and Schafer 1989]. Primarily, digital signals come from sampling and quantizing analog signals. In many cases, implementing a system in digital hardware is faster, more flexible, and more cost-effective than building it from analog components. One reason for this is that increasing precision in discrete systems is as simple as adding bits, whereas more expensive components may be required in analog systems. Another reason is that the time to prototype a system is often shorter for discrete systems. Apart from that, some DSP operators, like those that alter the sampling rate, have little meaning in the continuous domain, and may be very difficult to implement with analog components.

The most commonly used transforms in DSP are the z-transform, the discrete-time Fourier transform (DTFT) and the discrete Fourier transform (DFT). Just as the Laplace transform is a generalization of the Fourier transform, the z-transform is a generalized form of the DTFT. The z-transform is most useful in the implementation of digital filters as digital hardware or programs for DSP chips, while the DTFT is preferred when the frequency-domain characteristics of a system must be defined.

Neither digital nor analog signal processing is confined to a single dimension. Many spatial array and imaging applications require two- and three-dimensional processing. This is the case for detecting edges in monochrome digital images, or finding angles of arrival with an array of sensors. A digitized image can be thought of as a function defined on an $N \times M$ grid, where each grid square is assigned an intensity value known as a grey level (usually a positive integer). In such a digitized image, human eyes perceive an edge as a sharp transition between grey levels in adjacent regions. Such a sharp transition gives rise to high-frequency components in the image. Therefore, filtering the image to boost its high-frequency components and decrease its low-frequency components would sharpen edges in the image. This operation on the image is a two-dimensional high-pass filter; it corresponds to convolution in the two-dimensional "time" domain.

Digital signal processing has always been tied closely to computer implementations, where the signals are viewed as a stream of numbers. The machine provides an efficient way to compute the DSP operator that transforms one stream of numbers into another. On the other hand, the design of a signal processing system treats the signals as functions (in the mathematical sense).

Over the last decade, special attention has been devoted by several researchers to the field of symbolic signal processing [Kopec 1980, 1985; Myers 1986; Covell 1989]. In symbolic processing, the signal is represented in a computer as a formula, rather than as a sequence of numbers. Thus, the value of a signal might only be known in terms of a formula, instead of a number. In a similar manner, signal processing operators, the building blocks for systems, are maintained in symbolic form. This enables a machine to simplify, rearrange, and rewrite symbolic expressions until they take a desired form. For example, the desired form might be an optimal implementation of an expression in which the number of additions and multiplications is minimized.

The remainder of this paper describes a set of packages that provide a general platform for implementing symbolic transforms in *Mathematica* and that represent a first step toward the building of a comprehensive signal processing environment in *Mathematica*. The packages are divided into three groups, corresponding to the contexts `SignalProcessing`Support``, `SignalProcessing`Digital`` and `SignalProcessing`Analog``. Each of the next three sections focuses on one of these groups of packages [Evans and McClellan, 1990; Evans and McClellan, 1992; Evans, 1993]. The remaining section describes the Notebooks that accompany the packages [Evans and McClellan, 1991; Evans *et al.*, 1993; Evans, 1993]. Note that the signal processing packages are properly initialized in *Mathematica* by evaluating the command

```
Needs[ "SignalProcessing`Master`" ]
```

in which the single quote is the back quote.

Description of the Platform

Mathematica and Signal Processing

Mathematica presents an attractive platform upon which to build an environment for signal processing, providing as it does many computational functions, arbitrary-precision arithmetic, and an array of symbolic operations, such as differentiation and integration, partial fraction decomposition, power series expansion, and polynomial factoring and expansion. These symbolic capabilities play a key role in computing symbolic transforms.

As a programming environment, *Mathematica* supports not only traditional programming paradigms (procedural, functional, data-directed), but also an object-oriented approach, since the programmer can attach data and code to the head of an expression, so that an object can "know" information about itself. Therefore, one can write code that does not have to be modified to accommodate a new data type. That is, properly written *Mathematica* code is incrementally extensible.

Three other appealing aspects of *Mathematica* for the representation and manipulation of signal processing expressions are the cascading of systems, dynamic data typing, and (conditional) pattern matching. As a consequence of *Mathematica*'s support of functional programming, a cascade of systems (operators) can be represented as nested calls to the corresponding objects. Dynamic data typing allows valueless symbols and permits variables to assume any data type. Finally, the availability of a powerful pattern matcher, including conditional rules, permits the development of complex rule bases to perform mathematical transforms.

Signal-Processing Functions

Many of the common mathematical functions used in signal processing are built-in: sinusoids, exponentials, Bessel functions, etc. Other functions that are not built-in, and all the common signal-processing operators, are defined in the packages making up the context `SignalProcessing‘Support‘`. We now examine them in more detail.

Table 1 lists the twelve new functions. There are discrete and continuous versions of the impulse (`Impulse` and `Delta`), step (`Step` and `CStep`), and pulse functions (`Pulse` and `CPulse`). The family of unit functions `Unit[k]` generalizes `CStep` and `Delta`: u_{-1} is the continuous step function, $u_0 = \delta$ is the Dirac delta, and u_k is defined inductively by $u_k(t) = du_{k-1}(t)/dt$. For $k \geq 0$ these are really generalized functions, in the sense that they must be integrated with another function in order to give a number: thus the Dirac delta is defined

Object	Meaning
`CPulse`	continuous pulse: $$\sqcap_L(t) = \begin{cases} 0 & t < 0 \\ \frac{1}{2} & t = 0 \\ 1 & 0 < t < L \\ \frac{1}{2} & t = L \\ 0 & t > L \end{cases}$$
`CStep`	continuous step function: $$u_{-1}(t) = \begin{cases} 1 & t > 0 \\ \frac{1}{2} & t = 0 \\ 0 & t < 0 \end{cases}$$
`Delta`	Dirac delta function
`Dirichlet`	Dirichlet discrete kernel: $$d[N, \omega] = \frac{\sin(N\omega/2)}{N \sin(\omega/2)}$$ Also called the `ASinc` function
`Impulse`	Kronecker delta function
`LineImpulse`	m-D delta function, e.g. $\delta(t_1 - t_2)$
`Pulse`	discrete pulse: $$\sqcap_L[n] = \begin{cases} 0 & n < 0 \\ 1 & 0 \leq n \leq L-1 \\ 0 & n \geq L \end{cases}$$
`Sinc`	$\text{sinc}(t) \equiv \sin(t)/t$ so $\text{sinc}(0) = 1$
`Step`	discrete step function: $$u[n] = \begin{cases} 1 & n \geq 0 \\ 0 & n < 0 \end{cases}$$
`Unit`	family of functions which includes `CStep` and `Delta`

Table 1: Signal objects defined in `` SignalProcessing`Support` ``. For the syntax of `CPulse`, `Dirichlet`, `FIR`, `IIR`, `LineImpulse` and `Pulse`, consult the on-line documentation (e.g., `?CPulse`).

by the condition $\int \delta(t)f(t)\,dt = f(0)$ for every function f. The multidimensional impulse function `LineImpulse` is a generalization of the Dirac delta to several dimensions.

The function $\operatorname{sinc} t = \sin(t)/t$ (usually defined as $sin(\pi t)/(\pi t)$), and the related *aliased sinc* function, also known as the *Dirichlet kernel*, occur so commonly in signal processing that they deserve their own primitives, `Sinc` and `Dirichlet` (alternatively `AliasSinc` or `ASinc`).

Finally, the finite and infinite impulse-response functions, `FIR` and `IIR` for digital filters and `CFIR` and `CIIR` for analog filters, are unique to the field of signal processing; they represent the impulse response of an all-pole filter (infinite in extent) and of an all-zero filter (finite in extent), respectively [Myers 1986]. For these objects, the first argument gives the variable, and the second is a list giving the gain and minus the feedback coefficients; for example, `IIR[n, {1, 1/2, 1/4}]` represents a digital filter with a gain of 1/1 and feedback coefficients of $-\frac{1}{2}$ and $-\frac{1}{4}$. This means that the filter's input response is the solution $h[n]$ of the difference equation $h[n] = \delta[n] + \frac{1}{2}h[n-1] + \frac{1}{4}h[n-2]$, with $h[-1] = h[-2] = 0$. Likewise, the object `CIIR[t, {1, 1/2, 1/4}]` represents an analog filter that corresponds to the differential equation $h(t) = \delta(t) + \frac{1}{2}h'(t) + \frac{1}{4}h''(t)$ with $h(t) < 0$ for $t < 0$. CIIR and CFIR objects can also be specified in terms of their roots. For example, `CIIR[t, Roots -> -1, -2, -3]` yields `CIIR[t, 6, 11, 6, 1]`— the roots are embedded in the filter but not displayed. To see the entire data structure, use `FullForm`. Applying `TheFunction` to these filters will generate a time-domain function corresponding to their impulse response. Thus, the time-response of filters can be plotted using `SignalPlot` and `SequencePlot` for analog and digital filters, respectively.

An expression with head `IIR` or `FIR` can also be seen as the operator of convolution with the respective functions; that is, when `IIR` or `FIR` operate on a signal, the operation can be seen as the action of a filter with the specified zeros or poles. The filter can be analog or digital, depending on what it is applied to; for example, `IIR[n, {1, 1/2, 1/4}][a^n Step[n]]` represents the output of the filter described in the previous paragraph, when the input is $a^n u[n]$. This filter is digital only because of the presence of the discrete step function.

Signal Processing Operators

The twenty-one new operators for signal processing are listed in Table 2. Most of them need one or more arguments in order to be fully specified. One of the arguments— the last, or, for the transforms, the second-to-last— says what symbols are to be treated as variables in the function to be acted on. Thus, `Shift[7, t]` has no effect on the expression

Cos[x], but Shift[7, t][Cos[t]] *represents* Cos[t-7]. An impulse train, for example, in continuous time would be written either as Periodic[T, t][Delta[t]] or as Summation[n, -Infinity, Infinity, 1][Delta[t + n T]] where n is an integer index variable, T is the sampling period, and t is the continuous time variable. For the transforms, the last two arguments indicate the variables before or after transformation. Other arguments are explained in the table.

For one-dimensional operators, a variable is a symbol, like t. Arguments other than variables can be numbers, symbols or expressions. By using a list of symbols in place of a variable, like {t1, t2, t3} instead of t, we make the operator multidimensional, the length of the list indicating the dimensionality.

For multidimensional operators, upsampling and downsampling factors become square matrices, and other parameters become lists of expressions, one expression per dimension. For example, the downsample operator requires a downsampling factor m and a symbolic variable n. In one-dimensional downsampling, m is an integer and n is a symbolic index. In the multidimensional case, m is a matrix and n is a list of symbolic indices. In either case, the form of the operator is Downsample[m, n], and its syntax is Downsample[m, n][f], where f is a discrete function in n.

It must be emphasized that when one of the operators of Table 2 is applied to a function, no evaluation takes place— the idea is to store the resulting function symbolically, until it becomes convenient to compute it explicitly. This is done by applying TheFunction to the expression, much as a numerical evaluation in *Mathematica* is triggered by the action of N. Thus Z[n, z][a^n Step[n]] represents the z-transform of $a^n u[n]$, but the transformation is not carried out; to obtain an explicit transform one must apply TheFunction to the expression. (However, certain obvious simplifications are carried out: for example, InvZ[z,n][Z[n,z][f]] reduces to f.)

Similarly, the functions of in Table 1 are not reduced to *Mathematica* built-in objects until they appear as arguments to TheFunction. One may wish to reduce them, for example, in order to use *Mathematica*'s built-in plotting routines to plot them. Naturally, some functions in Table 1, like Delta and Unit, cannot be expressed in terms of *Mathematica* built-in objects, so TheFunction leaves them alone.

Other Features

Another facility provided by SignalProcessing`Support` is the plotting of signals and transforms. For discrete-time signals, time-domain plots are displayed in the conventional

`CFIR[t, {h0, h1, ...}]`	"all-zero" analog filter
`CIIR[t, {a0, a1, ...}]`	all-pole analog filter
`CircularShift[n0, N, n]`	shifts a sequence by `(n0 + n) mod N` in `n`
`CConvolve[t]`	continuous convolution in `t`
`Convolve[n]`	discrete convolution in `n`
`DFT[L, n, k]`	the `L`-sample discrete Fourier transform of a function in `n` to a function in `k`
`DTFT[n, w]`	the discrete-time Fourier transform of a sequence in `n` to a continuous periodic function in `w`
`Difference[i, n]`	the ith backward difference in `n`
`Downsample[m, n]`	keep first sample in every block of `m` input samples; sampling rate decreases by a factor of `m`
`FIR[n, {h0, h1, ...}]`	all-zero digital filter with finite impulse response `h0, h1, ...`.
`FT[t, w]`	continuous Fourier transform of a function of `t` into a function of `w`
`IIR[n, {a0, a1, ...}]`	all-pole digital filter with infinite impulse response
`Interleave[n]`	interleaves (combines) samples from each input function of `n` into one
`InvDFT[L, k, n]`	inverse discrete Fourier transform (see `DFT`)
`InvDTFT[w, n]`	inverse discrete-time Fourier transform (see `DTFT`)
`InvFT[w, t]`	inverse continuous Fourier transform (see `FT`)
`InvL[s, t]`	inverse Laplace transform (see `L`)
`InvZ[z, n]`	inverse z-transform (see `Z`)
`L[t, s]`	Laplace transform of a function of `t`
`Periodic[p, v]`	argument is made periodic with period `p` w/r to variable `v`
`Rev[v]`	reverse the function of variable `v` (flip the `v` axis)
`Shift[v0, v]`	shift function of `v` by `v0` points
`Summation[i, ib, ie, inc]`	summation operator, `i` = `ib` to `ie` step `inc`
`Upsample[l, n]`	insert `l` - 1 zeroes after each input sample; sampling rate increases by a factor of `l`
`Z[n, z]`	z-transform of a function of `n`

Table 2: (Parameterized) Operators defined in `` SignalProcessing`Support` ``. For the transforms, the last two arguments indicate the variable(s) in the function to be acted on and in the result of the transformation. For all other operators except `Summation`, the last argument indicates the variable(s) in the function to be acted on.

“lollipop” style [Oppenheim and Schafer 1989]. Continuous-time signals are plotted using built-in routines, augmented by new routines that display Dirac delta functions as arrows. Finally, pole-zero diagrams, root maps, and plots of magnitude and phase responses are available via the functions `PoleZeroPlot` and `MagPhasePlot`.

Other support routines provide drivers for multidimensional transforms. The multidimensional drivers simply call a one-dimensional rule base for each dimension of the transform and properly combine the region of convergence information obtained from each call. The end result of these drivers is a transform object. However, if a call to a transform rule base produces an incomplete transform, the multidimensional drivers exits immediately. In the digital and analog signal processing packages, these drivers are incorporated into the forward and inverse z- and Laplace-transform packages.

The last set of support routines, listed in Table 3, is needed in order to implement transform properties, because transform properties alter the transform function and the region of convergence. For all of these objects, the first argument is the name of a routine that returns true when its argument is a valid transform (`LForm` for the Laplace transform

`AddT`	`ConjT`	`ConvolveT`
`DerivativeT`	`IntegrateT`	`LineImpulsemDT`
`MultT`	`ScaleT`	`SubT`
	`SubstituteForT`	

Table 3: Objects implementing transform properties

and `ZForm` for the z-transform). The second argument is a transform or transform code to be evaluated. When the second argument is a valid transform, the routine implements the property on it. Other arguments vary with the object. This convention allows a nesting or cascading of properties.

Digital Signal Processing

We now turn to the packages in the context `SignalProcessing‘Digital‘`. Their important features are the *rule bases* for evaluation of forward and inverse z-transforms, and forward and inverse discrete-time Fourier transforms. Application of these rule bases is triggered by calls to the evaluation functions `ZTransform`, `InvZTransform`, `DTFTransform` and InvDTFTransform. Such calls are automatically performed when TheFunction is applied: thus, TheFunction[Z[n, z] [Exp[- n] Step[n]]] evaluates to ZTransform[Exp[- n] Step[n], n, z], which is then further evaluated to an explicit form.

The Forward z-Transform

Before we get into the details of the rule bases, we define the z- and discrete-time Fourier transforms. The forward z-transform produces an analytic function $X(z)$, from a sequence $x[n]$:

$$X(z) = \mathcal{Z}\{x[n]\} = \sum_{n=-\infty}^{\infty} x[n]z^{-n} \tag{1}$$

[Oppenheim and Schafer 1989]. (The bilateral version of the z- and Laplace transforms was adopted in these packages because it commonly used in signal processing. The unilateral form, where the sum extends only over non-positive n, is sometimes better suited to solving initial-value problems; but the bilateral form can also handle initial conditions by means of delta and step functions.)

The sum in (1) converges for values of z in a region of the form $\{z : r_- < |z| < r_+\}$, that is, an annulus (r_- and r_+ are both finite), an open disc ($r_- = 0$ and r_+ is finite), the complement of a closed disc or the whole plane. This information is found and preserved by ZTransform:

```
In[1]:= ZTransform[a^n Step[n],n]
                         1
Out[1]= ZTransData[---------, Rminus[Abs[a]],
                           a
                      ( 1 - - )
                           z
>              Rplus[Infinity], ZVariables[z]]
```

In this way, information about the stability of a signal can be obtained. (A signal is called *stable* if the sum of its absolute values remains bounded as time increases.) The function Stable operates on the z-transform of a digital signal, returning True if the signal is stable and False otherwise. If the stability is dependent on values of free parameters, Stable returns a set of stability conditions:

```
In[2]:= Stable[%]
Out[2]= Abs[a] < 1
```

The multidimensional z-transform is obtained by applying the one-dimensional z-transform once for each dimension. For example, the two-dimensional z-transform $X(z_1, z_2)$ of the function $x[n_1, n_2]$ is defined as

$$\begin{aligned} X(z_1, z_2) &= \sum_{n_2=-\infty}^{\infty} \sum_{n_1=-\infty}^{\infty} x[n_1, n_2] z_1^{-n_1} z_2^{-n_2} \\ &= \sum_{n_2=-\infty}^{\infty} \left(\sum_{n_1=-\infty}^{\infty} x[n_1, n_2] z_1^{-n_1} \right) z_2^{-n_2} \\ &= \mathcal{Z}_{n_2}\{\mathcal{Z}_{n_1}\{x[n_1, n_2]\}\} \end{aligned} \tag{2}$$

[Dudgeon and Mersereau 1984]. However, care must be taken in computing the region of convergence. If the function $x[n_1, n_2]$ is *separable*, that is, a product of functions of the two variables separately, the region of convergence of the double series (2) is the Cartesian product of the regions of convergence of the series (1) for the factors. For a non-separable function, however, this is not so, and we must take that into account.

The `ZTransform` function requires two arguments: the function to be transformed and the name of the "time" variable(s). The second argument determines the dimensionality and makes all other symbols in the expression to be treated as constants. An optional third argument allows the user to specify the names of the transform variable(s); the default is `z` in one dimension and `{z1, z2, ...}` in several. Any other arguments are treated as options. Currently there are only two options, `TransformLookup` (explained in Appendix A) and `Dialogue`. Setting `Dialogue` to `True` causes the program to tell the user about what strategies are being employed, and what functions, if any, could not be transformed. This is the default option. `Dialogue` can also be set to `False` (no justification) or `All` (complete justification). The `All` setting causes each step of the transformation to be displayed, so it is useful for teaching the mechanics of taking the z-transform.

`ZTransform` works by calling a one-dimensional rule base, `MyZTransform`, once for each variable to be transformed. The seventy-six rules in the `MyZTransform` rule base are divided into six groups, as shown in Table 4. The first group is needed in order to track the region of convergence of non-separable functions.

The next two groups encode rational (see Table 5) and non-rational (see Table 6) transform pairs. These rules apply to terms that contain either an impulse or a step function, so that every term is one-sided. Only right-sided transform pairs are encoded because special rules exist that take the left-sided transform by reversing the sequence in time, finding the transform, and then adjusting the result.

The next group of rules implements the properties of the z-transform listed in Table 7.

Type of rule	Number	Table
multidimensional hooks	9	
rational transform pairs	15	5
non-rational transform pairs	7	6
transform properties	13	7
transforms of operators	22	8
transform strategies	10	9

Table 4: Synopsis of rules for one-dimensional forward z-transforms

For example, the shift property

$$\mathcal{Z}\{x[n+m]\} = z^m \mathcal{Z}\{x[n]\}$$

is implemented by the (somewhat simplified) rules

```
MyZTransform[ f_. Step[n_ + m_], n_, z_] :=
  ScaleZ[ MyZTransform[(f /. n -> n - m) Step[n], n, z], z^m ] /;
    FreeQ[m,n]

MyZTransform[ f_. Step[m_ - n_], n_, z_] :=
  ScaleZ[ MyZTransform[(f /. n -> n + m) Step[-n], n, z], z^m ] /;
    FreeQ[m,n]

MyZTransform[ a_. Impulse[n_ + m_.], n_, z_] :=
  Transform[ (a /. n -> -m ) z^m, 0, Infinity ]
```

The first rule is for right-sided (causal) sequences, and the second is for left-sided (anti-causal) sequences; thus the pattern `f_ Step[m_ - n_]` matches anti-causal functions such as `Cos[Pi k / 5] Step[3 - k]`, where `k` is the variable being transformed. The third rule transforms shifted impulses. The `z^m` term has been factored out in all three rules.

The fifth section of `MyZTransform` takes the z-transform of different operators, according to Table 8. Because this rule base is recursive, z-transforms of cascaded systems can be found— for example, the z-transform of an upsampled, time-reversed, impulse response of an FIR filter.

The last group of rules (Table 9) adds strategies to try if all preceding rules have failed to give the transform completely. The repeated application of any of the starred strategy rules

Expression	z-Transform
0	0
$u[n]$	$\dfrac{1}{1-z^{-1}}$
$\delta[n]$	1
$\sin[\omega n+b]u[n]$	$\dfrac{\sin(b)+\sin(\omega-b)z^{-1}}{1-2\cos(\omega)z^{-1}+z^{-2}}$
$\cos[\omega n+b]u[n]$	$\dfrac{\cos(b)+\cos(\omega-b)z^{-1}}{1-2\cos(\omega)z^{-1}+z^{-2}}$
$\sinh[\omega n+b]u[n]$	$\dfrac{\sinh(b)+\sinh(\omega-b)z^{-1}}{1-2\cosh(\omega)z^{-1}+z^{-2}}$
$\cosh[\omega n+b]u[n]$	$\dfrac{\cosh(b)+\cosh(\omega-b)z^{-1}}{1-2\cosh(\omega)z^{-1}+z^{-2}}$
$a^n u[n]$	$\dfrac{1}{1-az^{-1}}$, if $\|z\|>\|a\|$
$a^{-n}u[-1-n]$	$\dfrac{1}{1-az^{-1}}$, if $\|z\|<\|a\|$
$\begin{pmatrix} n+k-j \\ k \end{pmatrix} u[n]$	$\dfrac{a^j z^{-j}}{(1-az^{-1})^{k+1}}$
$\begin{pmatrix} k \\ n \end{pmatrix} a^n b^{k-n} u[n]$	$(b+az^{-1})^k$
$\mathrm{FIR}[n,\{h_0,h_1,h_2,\ldots\}]$	$h_0+h_1z^{-1}+h_2z^{-2}+\ldots$
$\mathrm{IIR}[n,\{a_0,a_1,a_2,\ldots\}]$	$\dfrac{a_0}{1-a_1z^{-1}-a_2z^{-2}-\ldots}$

Table 5: Rational z-transform pairs; adapted from [Muth 1978] and [Oppenheim and Schafer 1989].

Expression	z-Transform
$\dfrac{u[n]}{n!}$	$e^{z^{-1}}$
$\dfrac{u[n]}{2n+1}$	$\sqrt{z}\,\mathrm{arctan}(z^{-1})$
$\dfrac{u[n-1]}{n}$	$-\ln(1-z^{-1})$
$\dfrac{u[n]}{(2n)!}$	$\cosh\sqrt{z^{-1}}$
$\dfrac{(2n)!}{(2^n n!)^2}u[n]$	$\sqrt{\dfrac{1}{1-z^{-1}}}$
$\dfrac{\Gamma[r]}{\Gamma[n+1]\Gamma[r-n]}u[n]$	$(1+z^{-1})^{r-1}$for real-valued $r > 0$

Table 6: Non-rational z-Transform Pairs. The last transform pair is from [Clements and Pease 1989]; all others are from [Muth 1978].

Structure	z-Transform
$x^*[n]$	$X^*(z^*)$
$x[n] * y[n]$	$X(z)Y(z)$
$\Delta_k x[n]$	$(1 - z^{-1})^k X(z)$
$\Im m(x[n])$	$\frac{1}{2j}[X(z) - X^*(z^*)]$
$x[n]u[n]$	$\frac{1}{1 - z^{-k}} X_k(z)$, $x[n]$ periodic of period k and $X_k(z) = \sum_{n=0}^{k-1} x[n] z^{-n}$
$\Re e(x[n])$	$\frac{1}{2}[X(z) + X^*(z^*)]$
$\uparrow_M x[n]$	$X(z^M)$
$\sum_{j=0}^{n} x[j]$	$\frac{1}{1 - z^{-1}} X(z)$

Table 7: z-Transforms of operators: adapted from [Muth 1978] and [Oppenheim and Schafer 1989].

1. rewrite affine step functions
2. break up any time-dependent absolute value function into a sum of a causal and an anti-causal step function
3. collect exponential terms (change an exponential divided by another into a ratio raised to an exponent)
4. expand $\frac{1}{n}x[n]u[n]$ into $\left\{\lim_{n\to 0}\frac{x[n]}{n}\right\}\delta[n] + \frac{1}{n}x[n]u[n-1]$ if $\lim_{n\to 0} x[n] = 0$

*5. distribute products like $(n+1)(n+2)$ into n^2+3n+2

*6. factor the denominator

*7. expand all numerator terms

*8. expand all denominator terms

9. try a two-sided transform

*10. substitute mathematical definitions for all terms

Table 8: Strategies for forward z-transforms. An asterisk means that once the rule is applied to an expression, it will no longer be applied to any part of that expression.

will cause an infinite loop, so local state information is associated with every expression (and inherited by every sub-expression) to prevent each of these rules from being applied more than once. The local state information is implemented as a list of five boolean values, one for each critical rule.

The Inverse z-Transform

The inverse z-transform requires the evaluation of the contour integral

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

[Oppenheim and Schafer 1989], where C is a counterclockwise contour encircling the origin and j is the imaginary number $\sqrt{-1}$, represented in *Mathematica* by `I`. Usually, the Cauchy Residue Theorem is applied to $X(z)$ in order to reduce the contour integral to an algebraic evaluation of residues. The inverse z-transform is equivalent to the Laurent series expansion of $X(z)$.

The command `InvZTransform` implements the inverse z-transform of one-dimensional and multidimensional expressions. In two or more dimensions, the general inverse is quite difficult because it involves multiple contour integrations, which *Mathematica* cannot perform. The rule base, however, covers many of these cases.

`InvZTransform` calls a one-dimensional rule base for each dimension to be transformed. The rules are organized in a manner similar to the one-dimensional forward z-transform rules (Table 9), the main difference being that only two multidimensional hooks are encoded (which invert transforms corresponding to upsampled sequences). Application of the one-dimensional rule bases automatically invert transforms corresponding to line impulses and multinomials.

Type of rule	Number	Table
multidimensional hooks	2	
rational transform pairs	26	5
non-rational transform pairs	9	6
transform properties	12	7
transforms of operators	4	8
transform strategies	9	10

Table 9: Synopsis of rules for one-dimensional inverse z-transforms

Six of the inverse transform pairs concern `Impulse`, `Step`, and sinusoidal forms. The other pairs return the inverse transform of rational z-transforms (with first-order and second-order denominators) and non-rational z-transforms (containing logarithmic, factorial and square-root forms). The properties of the transform and the transforms of operators are the same as in the forward case. Transforms of exponentials in the time domain are inverse-transformed by the exponential property rule, not by table lookup.

Some strategies for inverting z-transforms (Table 10) are similar to those applied in taking forward z-transforms, but some new ones are also needed. Two such strategies are partial fractions and power series expansion. The goal of partial fractions is to break up rational functions into a sum of terms involving only first-order and second-order denominators. Then, each individual term can be easily transformed. Power series expansion can be used to approximate the first N polynomial terms of the inverse transform. The repeated application of either of these two strategies, however, would cause an infinite loop. In order to avoid this, local state information ensures that any strategy with this behavior is applied at most once per expression.

*1. perform partial fraction expansion on rational polynomials

*2. perform partial fraction expansion

3. factor terms inside logarithmic functions

4. factor terms inside exponential functions

*5. normalize the denominator of a rational polynomial: $1 + a_1 z^{-1} + a_2 z^{-2} + \cdots$

*6. expand the definitions of all terms

*7. take a left-sided transform using right-sided rules by substituting z^{-1} for z, taking the inverse z-transform, and substituting $-n$ for n in the result

8. complex cepstrum: $\mathcal{Z}^{-1}\{\log X(z)\} \rightarrow -\frac{1}{n}\mathcal{Z}^{-1}\left\{\frac{z}{X(z)}\frac{d}{dz}X(z)\right\}$

*9. apply the inverse z-transform to the first N terms of a series expansion about $z = 0$

Table 10: Strategies for inverse z-transforms. An asterisk means that once the rule is applied to an expression, it will no longer be applied to any part of that expression.

As an example, consider the complex cepstrum of a sum of two impulses, $\delta[n] + \delta[n-3]$. (The *complex cepstrum* of a sequence is the inverse z-transform of the logarithm of the z-transform of the sequence.) Thus, we are interested in the inverse z-transform of $\log(1 + az^{-3})$. In the first pass through the rule base, none of the transform pairs and none of the property rules apply. However, the upsampling structure rule does apply, since the greatest common divisor of all exponents of `z` in `Log [ 1 + a z^-3 ]` is 3. This rule fires and rewrites the inverse transform as the upsampling by three of the inverse z-transform of `Log [ 1 + a z^-1 ]`. This can now be inverse-transformed by table lookup (Table 5), giving `Upsample[3,n][- (-1)^n a^n Step[n-1]/n]`.

In addition to the `Dialogue` option, `InvZTransform` accepts the option `Terms`. If set to a positive integer, this causes `InvZTransform` to attempt a series expansion as a last resort, as shown in Figure 1.

```
In[3]:= InvZTransform[ Exp[Tan[z]], z, n,
              Terms -> 5, Dialogue -> True ]

                                  Tan[z]
    Breaking up the expression  E
    into its series representation
             2    3      4       5
            z    z    3 z    37 z         6
    1 + z + -- + -- + ---- + ----- + O[z]
            2    2     8      120
    The inverse z-transform will now be
    applied to the first 5 terms.

Out[3]= Impulse[n] + Impulse[1 + n] +

        Impulse[2 + n]   Impulse[3 + n]
>       -------------- + -------------- +
              2                2

        3 Impulse[4 + n]   37 Impulse[5 + n]
>       ---------------- + -----------------
               8                  120
```

Figure 1: An example of series expansion

Solving Difference Equations

A common use of the z-transform is to solve difference equations. `ZSolve`, which is defined in the context `SignalProcessing‘Digital‘`, solves linear difference equations with constant coefficients:

```
In[4]:= ZSolve[ y[n] - 3/2 y[n - 1] + 1/2 y[n - 2] == (1/4)^n Step[n],
        y[n], y[-1] -> 4, y[-2] -> 10, Dialogue -> False ]


              1 n      1 n
        (2 + (-)  + 3 (-) ) Step[n]
              4        2
Out[4]= ---------------------------
                     3
```

Here, we have asked for no justification (`Dialogue -> False` and implied that the solution is right-sided since the default value of the `RightSided` option is `True`. Because the solution is assumed to be right-sided, the `Step` term in the driving function is not necessary. In this case, a full set of initial conditions is given. In general, initial conditions are optional and the missing ones are considered to be zero. When solving a difference equation, `ZSolve` substitutes the initial conditions into the left-hand side of the difference equation. This gives rise to impulse functions on the right-hand side of the difference equation that account for behavior before the driving function turns on. This approach enables the bilateral z-transform to solve the augmented difference equation completely.

> The bilateral approach to solving difference equations is first to assume that all missing initial conditions equal zero. Then, augment the difference equations to account for the initial conditions. Finally, transform each side, solve for the transfer function $Y(z)$, and inverse transform to obtain the digital function $y[n]$. For example, the difference equation
>
> $$y[n] - \frac{3}{2}y[n-1] + \frac{1}{2}y[n-2] = \frac{1}{4}^n u[n] \text{ such that } y[-1] = 4 \text{ and } y[-2] = 10 \qquad (3)$$
>
> provides all of the initial conditions so none have to be assumed zero. Substituting the initial conditions into Equation 3 gives
>
> for n = -1: $\quad y[-1] - \frac{3}{2}y[-2] + \frac{1}{2}y[-3] = (4) - \frac{3}{2}(10) + \frac{1}{2}(0) = -11$
>
> for n = -2: $\quad y[-2] - \frac{3}{2}y[-3] + \frac{1}{2}y[-4] = (10) - \frac{3}{2}(0) + \frac{1}{2}(0) = 10$
>
> The augmented difference equation, therefore, is
>
> $$y[n] - \frac{3}{2}y[n-1] + \frac{1}{2}y[n-2] = \frac{1}{4}^n u[n] - 11\delta[n-1] + 10\delta[n-2] \qquad (4)$$

which can be solved by taking the z-transform of both sides, solving for $Y(z)$, and inverse transforming the result to obtain $y[n]$. [Evans and McClellan 1992]

The Discrete-Time Fourier Transform

The one-dimensional discrete-time Fourier transform (DTFT) transforms a sequence $x[n]$ into a function $X(e^{j\omega})$ of a continuous variable ω, called frequency:

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

The inverse transformation is given by

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

[Oppenheim and Schafer 1989]. Thus, the Fourier transform is a periodic function of period 2π.

The equation for the forward DTFT is the same as for the forward z-transform, with $z = \exp(j\omega)$, so it is tempting to think that the DTFT can always be obtained from the z-transform. But this is not so for two reasons: first, some functions, such as $\operatorname{sinc} t$, have a DTFT but not a z-transform. Conversely, if the function does have a z-transform, the substitution $z = \exp(j\omega)$ is only valid when the unit circle is in the region of convergence.

The context `SignalProcessing‘Digital‘` provides the command `DTFTransform`, which first checks a special DTFT rule base, then applies the z-transform rule base (the substitution $z = \exp(j\omega)$ rewrites the z-transform as a DTFT). The specific DTFT rule base contains transform pairs of forms that do not have z-transforms, and rules describing downsampling and upsampling. (A downsampled signal does not have a z-transform, and the upsample operator behaves differently in the frequency domain.) The only redundancy between the two rule bases is the recasting of three property rules (homogeneity, additivity, and multiplication by n) and one strategy rule (expand all terms).

The inverse discrete-time Fourier transform command, `InvDTFTransform`, is implemented in a similar manner. Since both DTFT rule bases rely on the z-transform rule bases, the DTFT works in several dimensions, for one- and multi-sided signals. The inverse DTFT properly handles many non-separable transforms.

The Discrete Fourier Transform

When the sequence $x[n]$ is finite— say it is defined for only N points, $0 \leq n \leq N-1$— then N samples of the Fourier transform, uniformly distributed through the period, give enough information to reconstruct the original signal. This new sequence of samples is called the discrete Fourier transform (DFT). Thus the one-dimensional DFT $X[k]$ of the N-point sequence $x[n]$ is given by

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

and the inverse transformation is

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] e^{j2\pi kn/N}.$$

The symbolic DFT can be implemented either by this formula, or as the DTFT of $x[n]$ over the range $n = 0, 1, \ldots, N-1$, followed by sampling at the points $w_k = 2\pi k/N$, for $k = 0, 1, \ldots, N-1$.

Since the DFT is discrete in both the time and frequency domains, it can be computed and stored numerically, as well as symbolically. The `Fourier` function provided with the 1.2 release of *Mathematica* implements the numerical DFT, whereas the `SignalProcessing‘Digital‘` function `DFTransform` implements a symbolic DFT. Given a sequence, this function first takes its DTFT, then factors the result, if possible, into the product of a real-valued amplitude and a phase function. For example:

```
In[5]:= DFTransform[Exp[I 2 n], 10, n, k]


                     I (9 - 9 / 10 Pi k)
                    E                    Sin[ 10 - Pi k ]
Out[5]= DFTData[ -------------------------------------,
                                     Pi k
                          Sin[ 1 - ------ ]
                                      10


            Start[0], Finish[9], FVariables[k] ]
```

The phase function here is the exponential, and the amplitude function a ratio of two sinusoids. In fact, the amplitude is none other than the aliased sinc function introduced in Table 1. In discrete time, this function is the DTFT of an N-point rectangular window.

Like the forward DFT, the inverse DFT rule base relies on a DTFT rule base. `InvDFTransform` operates on finite-extent sequences and requires three arguments: $X[k]$, N, and k. Its only strategy rewrites the sampled frequency response into a continuous one and then calls the inverse DTFT.

Putting it all together

The `SignalProcessing‘Digital‘` packages provide a high-level function, `DSPAnalyze`, which exercises most of the features discussed above. This function, like `ZTransform`, requires two arguments: the expression to analyze and the variable(s) involved. Optionally, a third and fourth arguments specify the domain over which to plot the discrete-time function. Only one- and two-dimensional signals can be plotted.

`DSPAnalyze` finds the signal's z-transform, if it exists, and gives a stability criterion based on the transform's region of convergence. If the z-transform does not exist, the frequency response is obtained using `DTFTransform`. Using extensions to *Mathematica*'s graphics abilities, `DSPAnalyze` plots the expression, its magnitude response, and its phase response. For one-dimensional and separable two-dimensional expressions, it plots the pole-zero diagram, and for non-separable two-dimensional expressions, it plots the pole and zero root locus in each dimension.

As an example, consider the one-dimensional signal `Upsample[3,n][Sinc[n]]`. This expression does not have a z-transform (which causes the generation of many warning messages), but it does have a Fourier transform, so the magnitude and phase response can be computed and plotted. The magnitude response is periodic of period $2\pi/3$ (a period of $2\pi/k$ is characteristic of signals upsampled by a factor of k), and the phase response is zero, so it is omitted here, although `DSPAnalyze` plots it.

```
In[6]:= DSPAnalyze[ Upsample[3,n][Sinc[n]], n, -20, 20 ]

[graphs/ex1.time.ps]
                                  1
Power::infy: Infinite expression - encountered.
                                  0


Transform::incomplete:
                                                                1
    The rule base could not compute the forward z-transform of -
                                                                n
 with respect to n.


             1
The function - does not have a valid z-transform with respect to n
             n
```

ZTransform::notvalid:
The forward z-transform could not be found.

```
Upsample     [Sinc[n]]
        3,0,n
has the following frequency response:
ScaleAxis   [Pi (-CStep[-1 + w] + CStep[1 + w])]
         3,w
[graphs/ex1.mag.ps]

Out[6]= -Incomplete z-Transform-
```

As a second example, consider the two-variable signal

$$\frac{(n_1+n_2)!}{n_1!n_2!}a^{n_1}b^{n_2}u[n_1,n_2],$$

with $a=\frac{1}{2}$ and $b=\frac{4}{5}$. The increasing nature of the discrete domain plot hints at instability, but the z-transform cannot determine this. Both pole root maps, however, do indicate instability because some of the poles in the loci reside outside the unit circle. Dudgeon and Mersereau [1984] have shown that a function of this form is stable if and only if $|a|+$

$|b| < 1$. Because the time function is not stable, and because the z-transform is used to compute the Fourier transform, the Fourier transform and the frequency response are physically meaningless. For this reason we omit the magnitude and phase response graphs from the output.

```
In[7]:= DSPAnalyze[ (n1 + n2)! (1/2)^n1 (4/5)^n2 Step[n1,n2] / (n1! n2!),
                    {n1, n2} ]
[graphs/ex2.time.ps]
 1 n1  4 n2
(-)   (-)   (n1 + n2)! Step[n1] Step[n2]
 2     5
----------------------------------------
               n1! n2!
has the following z-transform:
     -10
-------------
      5    8
-10 + -- + --
      z1   z2
The region of convergence is:
1                               4         1
- < |z1| < Infinity             - Abs[--------] < |z2| < Infinity
2                               5          1
                                      1 - ----
                                          2 z1
The system is not stable.

The zeroes are:  {}
                               I w
                  -5. 2.71828
The poles are:   {-------------------}
                                  I w
                8. - 10. 2.71828
```

```
[graphs/ex2.pole1.ps]
The zeroes are:  {}
                               I w
                    -8. 2.71828
The poles are:   {-------------------}
                                  I w
                  5. - 10. 2.71828

[graphs/ex2.pole2.ps]

 1 n1  4 n2
(-)   (-)   (n1 + n2)! Step[n1] Step[n2]
 2     5
----------------------------------------
                n1! n2!
has the following frequency response:
           -10
-------------------------
         -I w1      -I w2
-10 + 5 E      + 8 E
                                                     1
                                          4 Abs[--------]
                                                       1
                                                1 - ----
                        -10               1           2 z1
Out[7]= ZTransData[-------------, Rminus[{-, ---------------}],
                         5    8           2         5
                   -10 + -- + --
                         z1   z2

>            Rplus[{Infinity, Infinity}], ZVariables[{z1, z2}]]
```

Analog Signal Processing

The packages making up the context `SignalProcessing‘Analog‘` implement the Laplace transform and the (continuous-time) Fourier transform. Given a continuous-time function $f(t)$, its bilateral Laplace transform $F(s)$ is defined as

$$F(s) = \int_{-\infty}^{+\infty} f(t)\, e^{-st}\, dt \quad \text{and} \quad f(t) = \frac{1}{2\pi j} \oint_{\sigma - j\infty}^{\sigma + j\infty} F(s)\, e^{st}\, ds$$

[Oppenheim and Willsky 1983]. The region of convergence for the Laplace transform is a strip, of the form $\{s \in \mathbf{C} : R_- < \Re e\ s < R_+\}$. In the inverse transform, the contour of integration includes the section of the s-plane whose real components are bounded by the constant σ, i.e., either $\Re e\ s < \sigma$ or $\Re e\ s > sigma$.

The continuous-time Fourier transform is a function of a real variable ω:

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

The corresponding commands are `LaPlace`, `InvLaPlace`, `CTFTransform` and `InvCTFTransform`. Like their digital counterparts, these commands are implemented using rule bases. The organization of the forward Laplace transform rule base follows the same outline as the forward z-transform rule base (see Table 4). There are more transform pairs and property rules than in the z-transform rule base, but fewer structure (operator) rules. The strategies are the same except that strategy 4 of Table 8 does not apply in the continuous case. Similarly, the `InvLaPlace` rule base has the same structure as the `InvZTransform` rule base, but contains more transform pairs and fewer operator rules.

`LaPlace` and `InvLaPlace` are more comprehensive than the Laplace-transform commands provided with the 1.2 release of *Mathematica*, which support only one-dimensional, one-sided Laplace transforms, and do not keep track of the region of convergence. Both versions of the Laplace transform, however, can be loaded into *Mathematica* since the bilateral and unilateral transforms have different naming conventions.

The continuous-time analogs of `DTFTransform`, `InvDTFTransform`, and `ZSolve` are `CTFTransform`, `InvCTFTransform`, and `LSolve`. The forward Fourier transform command `CTFTransform` relies on `LaPlace`, and the inverse Fourier transform command `InvCTFTransform` is built on top of `InvLaPlace`. Because their definitions are very similiar, `CTFTransform` and `InvCTFTransform` share the same rule base. `LSolve` solves linear differential equations with constant coefficients; it takes the equation and the unknown function as arguments and the initial conditions and justification level as options. For example, the differential equation $y''(t) + \frac{3}{2}y'(t) + \frac{1}{2}y(t) = \exp(at)u_{-1}(t)$ with initial conditions $y'(0) = \frac{1}{2}$ and $y(0) = 4$ is solved as follows:

```
In[8]:= LSolve[ y''[t] + 3/2 y'[t] + 1/2 y[t] == Exp[a t] CStep[t],
                y[t], y'[0] -> 1/2, y[0] -> 4, Dialogue -> False ]


Out[8]= (-10 - 36 a) Exp[- t/2] CStep[t/2] / (2 (-1 - 2 a)) +
        (3 + 5 a) Exp[-t] CStep[t] / (-1 - a) -
        2 Exp[a t] CStep[t] / (-1 - 3 a - 2 a^2)
```

The function `ASPAnalyze` is the analog counterpart of `DSPAnalyze`. As an example of its use, consider the function

$$te^{-at} \cos\left(\frac{3\pi t}{16}\right) u_{-1}(t).$$

To be able to plot the function, `ASPAnalyze` substitutes a default value of 1 for all unknown parameters (`a` in this case). The Laplace transform is a rational polynomial in s with two poles, each with multiplicity 2. In the pole-zero diagram, a single zero is represented as `O` and a single pole as `X`; multiple poles or zeroes appear as `*X*` or `*O*`.

```
In[9]:= ASPAnalyze [ t Exp[- a t] Cos[3 Pi t / 16] CStep[t], t, 0, 3 Pi ]


For plotting only, these symbols will
be assigned a value of 1: {a}
Real part of the function is shown as solid lines.
Imaginary part of the function is shown as dashed lines.

[graphs/ex4.time.ps]


               3 Pi t
t CStep[t] Cos[------]
                 16
----------------------
          a t
         E
has the following Laplace transform:
```

```
                 2
      -2 (a + s)                1
-(------------------- + ----------------)
       2                     2
   9 Pi           2 2   9 Pi           2
  (----- + (a + s) )    ----- + (a + s)
    256                  256

The strip of convergence is:  -Re[a] < Re(s) < Infinity
The system is stable if 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}

[graphs/ex4.pole.ps]
               3 Pi t
t CStep[t] Cos[------]
                 16
----------------------
          a t
         E
has the following frequency response:
                                       2
          1                2 (a + I w)
-(------------------ - ---------------------)
      2                     2
  9 Pi              2    9 Pi             2 2
  ----- + (a + I w)     (----- + (a + I w) )
   256                   256
[graphs/ex4.mag.ps]
[graphs/ex4.pha.ps]

                                          2
```

```
                        -2 (a + s)               1
Out[9]= LTransData[-(------------------ + ---------------),
                         2                     2
                      9 Pi           2 2   9 Pi           2
                     (----- + (a + s) )    ----- + (a + s)
                       256                   256

>             Rminus[-Re[a]], Rplus[Infinity], LVariables[s]]
```

Finally, here is an example with Dirac delta functions. The time-domain plot represents them with arrowheads, a common engineering convention. The value of the global variable \$DeltaFunctionScaling indicates how Delta functions are to be plotted: `None` means that all Delta functions will have identical heights equal to the height of the plot, and `Scaled` means that the height of each Delta function will be equal to its area (the default). This function has a Laplace transform but no poles or zeroes, so a pole-zero diagram is not displayed. `ASPAnalyze` does plot its magnitude and phase frequency responses, although we have omitted them here.

```
In[10]:= ASPAnalyze [ Delta[t - 1/2] + Delta[t - 3/2], t, 0, 2 ]
Real part of the function is shown as solid lines.
Imaginary part of the function is shown as dashed lines.
[graphs/ex5.time.ps]
        3                 1
Delta[-(-) + t] + Delta[-(-) + t]
        2                 2
has the following Laplace transform:
      s
 1 + E
--------
 (3 s)/2
E

The strip of convergence is:  -Infinity < Re(s) < Infinity
The system is stable.
```

PoleZeroPlot::notrational:

Transform is not a rational polynomial.

PoleZeroPlot::noplot:

A pole-zero plot cannot be generated.

```
        3                 1
Delta[-(-) + t] + Delta[-(-) + t]
        2                 2
has the following frequency response:
 (-3 I)/2 w       I w
E           (1 + E   )
```

ASPAnalyze::notinteresting:

Could not determine the important section of the frequency response.

```
.
.
.
                                s
                         1 + E
Out[10]= LTransData[--------, Rminus[-Infinity],
                         (3 s)/2
                        E

>               Rplus[Infinity], LVariables[s]]
```

Graphical Tools

The signal processing packages introduce new objects to create plots that follow engineering convention. One- and two-dimensional sequences are displayed in "lollipop" style [Oppenheim and Schafer] via `SequencePlot`, whereas `SignalPlot` graphs one- and two-dimensional continuous signals. Both `SignalPlot` and `SequencePlot` support the same options as does the standard plotting function `Plot`. `PoleZeroPlot` displays pole-zero diagrams for one- and two-dimensional transfer functions. This function supports only one option `Dialogue` which if `True` or `All` forces the printing of the values of the poles and zeroes before they are plotted.

`RootLocus` and `MagPhasePlot` generate root locus plots and plot magnitude/phase responses, respectively. `MagPhasePlot` displays continuous-time as well as discrete-time Fourier

`AnalogFilters`	designing/analyzing 1-D analog filters
`DTFT`	discrete Fourier analysis
`EducationalTool`	interactive version of a conference paper describing educational impact of *Mathematica*
`LaPlaceTest`	testing procedure for Laplace transforms
`PiecewiseConvolution`	tutorial on discrete/continuous convolution
`README`	brief introduction
`SignalProcessingExamples`	interactive version of paper in the *The Mathematica Journal*
`SignalProcessingIntroduction`	introduction to *Mathematica*, signal processing, and the signal processing packages
`SignalProcessingUsage`	usage information about every new object defined by the signal processing packages
`zTransformI`	z-transform tutorial, part I
`zTransformII`	z-transform tutorial, part II
`zTransformIII`	z-transform tutorial, part III

Table 11: List of the signal processing Notebooks

transforms as long as the options are set properly. The default options are biased toward DTFT's: `Domain -> Continuous`, `DomainScale -> Linear`, `MagRangeScale -> Linear`, `PhaseRangeScale -> Degree`, and `PlotRange -> All`. For CTFT's, `DomainScale` option should be set to `Log` and the `MagRangeScale` should also be set to `Log` which would give the familiar Bode diagram. These routines also support the same options as `Plot` does, Two of the options, `PlotRange` and `DisplayFunction`, are useful for generating animations of frequency dependence on a parameter.

Signal Processing Notebooks

Accompanying the signal processing packages are several Notebooks. Three are tutorials covering the topics of convolution, analog filter design, and the z-transform (which comes in three parts). One, `SignalProcessingIntroduction`, introduces *Mathematica*, signal processing, and the signal processing packages. The remaining Notebooks serve as on-line help. The signal processing Notebooks are listed in Table 11. This section summarizes the contents of the tutorial Notebooks.

Piecewise Convolution Notebook

The linear convolution of $f(t)$ and $g(t)$ is defined as

$$f(t) \star g(t) = \int_{-\infty}^{+\infty} f(u)\, g(t-u)\, du \quad .$$

Here, $f(t)$ and $g(t)$ are piecewise-continuous functions. Piecewise convolution is a time-domain approach that finds the convolution by computing the integral over each interval where the two functions overlap as $g(t-u)$ "slides" through $f(u)$. This is usually the first approach for solving convolution problems that engineers learn.

The piecewise convolution Notebook shows the student how to solve one-dimensional continuous-time convolution problems using the piecewise convolution package. In the introduction, the Notebook informs the user that a piecewise function can be represented as a list of intervals. Each interval is itself a list containing the start of the interval, the end of the interval, and the function defined on that interval. For example, the function $\triangle(t)$, a unit triangle defined on $t \in [-1, 1]$, can be represented as an expression

```
(1 + t) CPulse[1, 1 + t] + (1 - t) CPulse[1, t]
```

or in list form as

```
{{1 + t, -1, 0}, {1 - t, 0, 1}}.
```

The Notebook gives several examples of piecewise convolution using the newly defined object `PiecewiseConvolution`, which requires three arguments— $f(t)$, $g(t)$, and t. For example, the *Mathematica* dialogue

```
In[11]:= PiecewiseConvolution[Delta[t - a], f[t], t]
Out[11]= {{ f[t - a], -Infinity, Infinity }}
```

illustrates the shifting property of the Dirac delta function. We could have used the piecewise form of `Delta[t - a]`, which is `{Area[1], a, a}`, in this example. The Notebook also gives an example of a convolution that produces an unstable function:

```
In[12]:= PiecewiseConvolution[CStep[t], CStep[t], t]
```

Integrate::divg: Integral does not converge

```
Out[12]= {{ t, 0, Infinity }}
```

Part of computing a piecewise convolution requires identifying the overlapping intervals by drawing pictures of $f(u)$ and $g(t-u)$. So, for two simple functions, the Notebook shows an *animation* of $g(t-u)$ sliding through $f(u)$. Each frame shows an important interval over which to integrate, and the reader can control the frame rate.

The animation was generated by using the `Dialogue -> True` option to `PiecewiseConvolution`. `PiecewiseConvolution` also supports a `Domain` option which is either `Continuous` or `Discrete`. The default is the value of `$ConvolutionDomain` which is initially `Continuous`.

Analog Filter Design Notebook

Linear filters are certainly the workhorse of signal processing. The signal processing packages allow a user to design Butterworth, Chebyshev (type I and type II), elliptic, and Bessel filters using the object `DesignFilter`. For lowpass and highpass filters, `DesignFilter` takes eight arguments: `Analog`, filter class (`Butterworth`, ...), filter type (`Lowpass`, ...), time variable, δ_1, δ_2, w_1, and w_2. Two additional arguments, w_3 and w_4, are necessary for specifying a bandpass or bandstop filter. Frequencies are in rad/sec and given in ascending order (e.g., for a lowpass filter, w_1 is w_p and w_2 is w_s). The magnitude response resides on $[1-\delta_1, 1]$ for all passbands and $[0, \delta_2]$ for all stopbands. Since the Laplace and Fourier rule bases can transform the time-domain expressions returned by `DesignFilter`, the pole-zero diagram and frequency response of a filter can be plotted using `PoleZeroPlot` and `MagnitudePhasePlot`.

The Notebook introduces the topic of filter design and guides the reader through several design examples. These examples include a lowpass design example for each filter type, as well as a bandpass, bandstop, and highpass design example. Using animation, the student gets a chance to visualize the changes in the filter's magnitude response as one design parameter varies (two animations are provided— one for ripple control parameter ϵ and one for the filter order N). The last section of the Notebook examines the effects of precision in implementation— the student can truncate the coefficients of the filter and observe the frequency response.

The z-Transform Notebook

The objects `ZTransform` and `InvZTransform` compute the forward and inverse z-transform, respectively, according to the standard definitions: [12]

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

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

The forward rule base works for 1-D and m-D signals, whereas the inverse rule base only inverts 1-D and separable m-D transforms (*Mathematica* simply cannot perform the multiple contour integrations required in the general m-D case). Since the rule bases implement the bilateral form of the z-transform, `ZTransform` tracks the region of convergence (ROC) so that `InvZTransform` can uniquely determine the inverse.

The z-transform Notebook comes in three parts because of its size. Part I defines the bilateral z-transform and discusses the importance of the ROC in uniquely defining the transform. Discussion of the ROC leads naturally into introducing stability and causality of a signal based on the location of its poles relative to the ROC. Part II links the z-transform with the discrete-time Fourier transform (DTFT) and includes several animations relating pole-zero diagrams to magnitude frequency responses, a basic idea behind digital filter design [Oppenheim & Schafer, 1989, ch. 5]. A figure in Part II, for example, shows one animated frame for a three-pole, two-zero filter whose magnitude response is being evaluated at a frequency of π rad. Scattered throughout the Notebook are several examples and exercises with solutions, many of which use examples of high-level digital signal analysis using `DSPAnalyze`, the discrete-time version of `ASPAnalyze`. Part III concludes the z-transform Notebook with two animations showing the deterioration of the magnitude response of a fourth-order elliptic filter when the location of the poles are slightly perturbed.

Conclusion

These packages described in this article are a first step toward the building of a comprehensive signal processing environment in *Mathematica*. We have implemented the common signal processing transforms and other rudimentary analysis capabilities. But *Mathematica* has the potential to do much more. We have developed packages that aid in filter design and perform discrete and continuous convolution in one dimension. We are developing an exhaustive collection of windowing functions, which are also important in time series analysis. Other packages under development will take further the idea of representing signals and systems as abstract objects, empowering *Mathematica* to reason about properties of signals and transforms, to run simulations of systems, and to find optimal implementations of signal processing expressions. The ultimate goal is to have *Mathematica* facilitate fast prototyping of algorithms by first optimizing an algorithm (subject to design specification and the limitations of the target architecture) and then generate the equivalent code for the architecture.

This paper also describes how students could use *Mathematica* to supplement classroom lectures and laboratory experience. Using Notebooks, students can load the signal processing packages and tackle homework problems in linear systems while documenting the solution as they go. Notebooks can also serve as tutorials. We have written four Notebooks covering the topics of piecewise convolution, analog filter design, Laplace transform, and the z-transform. The Notebook are currently in use in several engineering and mathematics departments, including Rose-Hulman Institute of Technology, Georgia Institute of Technology, Uninversity of Pennsylvannia, and Pennsylvannia State University. Although some schools are integrating **Mathematica** into their curricula, we hope that more schools encourage students to use it.

Acknowledgements

The authors would like to thank Drs. Silvio Levy, Paul Abbott, and Bruce Sawhill at Wolfram Research Inc. for their helpful comments and suggestions.

The authors would also like to recognize some of the pioneers in the field of symbolic signal processing: Michele Covell, Gary Kopec, Cory Myers, and Alan Oppenheim.

References

1. Clements, Mark, and Jerrold Pease. 1989. "On Causal Linear Phase IIR Digital Filters," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, **37** (4), pp. 479-484.

2. Churchill, Ruel. 1958. *Operational Mathematics*. McGraw-Hill, New York.

3. Covell, Michele. 1989. *An Algorithm Design Environment for Signal Processing*. Ph. D. Thesis and RLE Tech. Rep. #549. MIT, Cambridge (MA).

4. Dudgeon, Dan, and Russel Mersereau. 1984. *Multidimensional Digital Signal Processing*. Prentice Hall, Englewood Cliffs (NJ).

5. Evans, Brian L., James H. McClellan, and Wallace B. McClure. 1990. "Symbolic Transforms with Applications to Signal Processing." *The Mathematica Journal*. vol. 1, no. 2 (Fall), pp. 70-80.

6. Evans, Brian L., James H. McClellan, and Kevin A. West. 1991. "*Mathematica* as an Educational Tool for Signal Processing." *IEEE Southeastern Conference.* pp. 1162-1166.

7. Evans, Brian L., and James H. McClellan. 1992. "Symbolic Analysis of Signals and Systems." ch. 3 of *Symbolic and Knowledge-Based Signal Processing.* eds. A. Oppenhiem and H. Nawab. Prentice Hall, Englewood Cliffs (NJ).

8. Evans, Brian L., Lina J. Karam, Kevin A. West, and J. H. McClellan, "Learning Signals and Systems with Mathematica." *IEEE Transactions on Education.* vol. 36, pp. 72–78, Feb. 1993.

9. Evans, Brian L. "A Knowledge-Based Environment for the Design and Analysis of Multidimensional Multirate Signal Processing Algorithms." Ph. D. Thesis. Georgia Institute of Technology. Atlanta, GA. June, 1993.

10. Kopec, Gary. 1980. "The Representation of Discrete-Time Signals and Systems in Programs." Ph. D. Thesis. MIT, Cambridge (MA).

11. Kopec, Gary. 1985. "The Signal Representation Language SRL." *IEEE Transactions on Acoustics, Speech, and Signal Processing*, **33** (4), pp. 921-932.

12. Lathi, B. P. 1983. *Modern Digital and Analog Communication Systems.* Holt, Rinehart, and Winston, New York.

13. Muth, Eginhard. 1978. *Transform Methods.* Prentice Hall, Englewood Cliffs (NJ).

14. Myers, Cory S. 1986. "Signal Representation for Symbolic and Numeric Processing". Ph. D. Thesis and RLE Tech. Rep. #521. MIT, Cambridge (MA).

15. Oberhettinger, Fritz, and Larry Badii. 1973. *Tables of Laplace Transforms.* Springer-Verlag, New York.

16. Oppenheim, Alan, and Alan Willsky. 1983. *Signals and Systems.* Prentice Hall, Englewood Cliffs (NJ).

17. Oppenheim, Alan, and Ronald Schafer. 1989. *Discrete-Time Signal Processing.* Prentice Hall, Englewood Cliffs (NJ).

Appendix A: Options for the Transform Rule Bases

All 10 transform rule bases support the `Dialogue` and `TransformLookup` options. The `Dialogue` option indicates how much information the transform rule bases give to the user: `False` means to print no information, `True` means to tell about strategies when used and about parts of the expression that could not be transformed (if any), and `All` means to display each step of the transformation process. (This `Dialogue` option is also supported by the difference and differential equation solvers `ZSolve` and `LSolve`.) The `TransformLookup` option allows users to specify their own transform pairs. This means that users can transform general (undefined) signals: e.g., letting $y(t) \leftrightarrow Y(s)$:

```
In[15]:= LaPlace[ y''[t] CStep[t], t, s, TransformLookup -> {y[t] :> Y[s]} ]
                          2
Out[15]= LTransData[ s  Y[s] - s y[0] - y'[0], Rminus[0],
                Rplus[Infinity], LVariables[s] ]
```

(The region of convergence is significant because $\mathcal{L}\{y''(t)\} \leftrightarrow s^2\,Y(s)$ for $-\infty < \Re e(s) < \infty$ but $\mathcal{L}\{y''(t)u_{-1}(t)\} \leftrightarrow s^2\,Y(s) - s\,y(0^+) - y'(0^+)$ for $0 < \Re e(s) < \infty$.) The following table lists and explains these and other options.

Option	*Possible Values*	*Meaning*
`Apart`	`Rational, All`	*Partial fraction decomposition only applies to polynomials with real or rational coefficients*
`Definition`	`True, False`	*Use the transform definition if all else fails to find the transform (does not apply to the inverse z or Laplace transforms)*
`Dialogue`	`False, True, All`	*Ascending levels of justification*
`Simplify`	`True, False`	*Apply* `SPSimplify` *to result*
`Terms`	`False` *or* integer	*Number of terms in series expansion (*`False` *means none)*
`TransformLookup`	*list of rules*	*Users can specify their own transform pairs, like* `{x[n] :> X[z]}` *or* `{y[t1,t2] :> Y[s1,s2]}`

Table 12: **Meaning of the Options for the Transform Rule Bases**

Transform	*Option*	*Default Value*
`CTFTransform`	`Dialogue`	`False`
	`Simplify`	`True`
`DFTransform`	`Dialogue`	`False`
`InvDFTransform`	`Dialogue`	`False`
	`Terms`	`False`
`DTFTransform`	`Dialogue`	`False`
`LaPlace`	`Dialogue`	`True`
	`Simplify`	`True`
`InvCTFTransform`	`Apart`	`Rational`
	`Dialogue`	`False`
	`Simplify`	`True`
	`Terms`	`False`
`InvDTFTransform`	`Dialogue`	`False`
	`Terms`	`False`
`InvLaPlace`	`Apart`	`Rational`
	`Dialogue`	`True`
	`Simplify`	`True`
	`Terms`	`10`
`InvZTransform`	`Dialogue`	`True`
	`Terms`	`10`
`ZTransform`	`Dialogue`	`True`

Table 13: Options for the Transform Rule Bases. `Definition` always defaults to `False` and `TransformLookup` always defaults to an empty list.

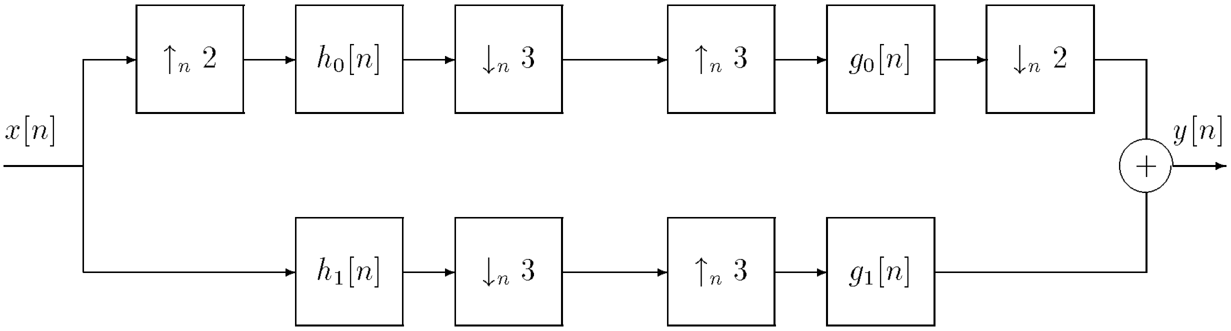
Appendix B: TeX Forms for Signals and Operators

`Aliasby[m, w][X[w]]`	$\frac{Perodic_{\frac{2\,\pi}{m},w}(X(w))}{m}$
`CConvolve[t][f[t], g[t]]`	$f(t) \star_t g(t)$
`CPulse[{L1, L2}, {t1, t2}]`	$\sqcap_{L1,L2}(t1,t2)$
`CStep[a + t]`	$u_{-1}(a+t)$
`CStep[t1] CStep[t2]`	$u_{-1}(t1)\,u_{-1}(t2)$
`Delta[1 + t]`	$\delta(1+t)$
`Difference[k, n]`	$\Delta_{k,n}$
`Downsample[m, n]`	$\downarrow_{m,n}$
`DFT[L, n, k]`	$\mathcal{DFT}_{n\rightarrow k,\mathrm{L}=L}$
`FT[{t1, t2}, {w1, w2}]`	$\mathcal{F}_{\substack{t1\,\rightarrow\,w1\\ t2\,\rightarrow\,w2}}$
`Impulse[K + m]`	$\delta[K+m]$
`Interleave[n][x1[n], x2[n]]`	$Interleave_n(x1(n), x2(n))$
`InvDFT[L, k, n][X[k]]`	$\mathcal{DFT}^{-1}_{k\rightarrow n,\mathrm{L}=L}(X(k))$
`InvDTFT[w, n][X[Exp[I w]]]`	$\mathcal{DTFT}^{-1}_{w\rightarrow n}(X(e^{i\,w}))$
`InvFT[w, t]`	$\mathcal{F}^{-1}_{w\rightarrow t}$
`InvL[s, t][X[s]]`	$\mathcal{L}^{-1}_{s\rightarrow t}(X(s))$
`InvZ[z, n][Y[z]]`	$\mathcal{Z}^{-1}_{z\rightarrow n}(Y(z))$
`L[t, s][X[s]]`	$\mathcal{L}_{t\rightarrow s}(X(s))$
`Pulse[L, n]`	$\sqcap_L[n]$
`Rev[x][y[x]]`	$Rev_x(y(x))$
`ScaleAxis[m, w][W[w]]`	$ScaleAxis_{m,w}(W(w))$
`Shift[n0, n][m[n]]`	$Shift_{n0,n}(m(n))$
`Sinc[Pi x^2]`	$\mathrm{sinc}(\pi\,x^2)$
`Summation[i, 0, -1 + L, 1][i v^2]`	$\sum_{i=0}^{-1+L}(i\,v^2)$
`Summation[j, 0, -1 + K, 2][j^2]`	$\sum_{\substack{j=0\\ j=j+2}}^{-1+K}(j^2)$
`Unit[4][t]`	$u_4(t)$
`Upsample[2, n][x[n]]`	$\uparrow_{2,n}(x(n))$
`Upsample[{{1, 2}, {3, 4}}, {n1, n2}]`	$\uparrow_{\left[\begin{array}{cc}1 & 2\\ 3 & 4\end{array}\right],\substack{n1\\ n2}}$
`Z[n, z][x[n]]`	$\mathcal{Z}_{n\rightarrow z}(x(n))$

Appendix C: A Multirate Signal Processing Example

The following example uses the signal processing packages to derive the input-output relationship of a two-channel non-uniform filter bank with frequency band splits of $\frac{2}{3}$ and $\frac{1}{3}$ in the upper and lower channels, respectively.

The Flow Graph



Representation in *Mathematica*

```
upperchannel =
  Downsample[2,n][
    Convolve[n][g0[n], Upsample[3,n][
                          Downsample[3,n][
                            Convolve[n][h0[n],
                                        Upsample[2,n][x[n]]]]]]]
lowerchannel =
  Convolve[n][g1[n], Upsample[3,n][
                       Downsample[3,n][
                         Convolve[n][h1[n], x[n]]]]]
```

TeX form of the input-output relationship derived using `ZTransform`

$$
\begin{aligned}
&\tfrac{1}{6}(G_0(-\sqrt{z})H_0(-\sqrt{z}) + G_0(\sqrt{z})H_0(\sqrt{z}) + 2G_1(z)H_1(z))\,X(z)+\\
&\tfrac{1}{6}\Big(G_0(-\sqrt{z})H_0(e^{\frac{i}{3}\pi}\sqrt{z}) + G_0(\sqrt{z})H_0(e^{\frac{4i}{3}\pi}\sqrt{z}) + 2G_1(z)H_1(e^{\frac{2i}{3}\pi}z)\Big)\,X(e^{\frac{2i}{3}\pi}z)+\\
&\tfrac{1}{6}\Big(G_0(\sqrt{z})H_0(e^{\frac{2i}{3}\pi}\sqrt{z}) + G_0(-\sqrt{z})H_0(e^{\frac{5i}{3}\pi}\sqrt{z}) + 2G_1(z)H_1(e^{\frac{4i}{3}\pi}z)\Big)\,X(e^{\frac{4i}{3}\pi}z)
\end{aligned}
$$