SignalProcessing Improvements in Maple 2023

The SignalProcessing package has been expanded with new and updated commands.

```
> with( SignalProcessing ):
```

Quantize

SavitzkyGolayFilter

Convolution

FFT and InverseFFT

Quantize

- The new SignalProcessing:-Quantize command is used to replace real-valued data in a container with values from a codebook.
- For example, suppose we want to quantize the values generated by $\sin(t)$ over the interval $[0, 2\pi]$ so that the values are to the nearest quarter integer. With this new command, we can do either of the following:

```
> X := GenerateSignal( sin(t), t = 0 .. 2 * Pi, 50 )^+;
```

$$X := \left[0., 0.127877161684506, 0.253654583909507, 0.375267004879374, 0.490717552003938, 0.598110530491216, 0.695682550603486, 0.781831482468030, 0.855142763005346, 0.914412623015812, 0.958667853036661, 0.987181783414450, 0.999486216200688, 0.995379112949198, 0.974927912181824, 0.938468422049760, 0.886599306373000, 0.820172254596956, 0.740277997075316, 0.648228395307788, 0.545534901210549, 0.433883739117558, 0.315108218023621, 0.191158628701373, 0.0640702199807132, -0.0640702199807126, -0.191158628701372, -0.315108218023621, -0.433883739117558, -0.545534901210548, -0.648228395307788, -0.740277997075315, -0.820172254596956, -0.886599306373000, -0.938468422049760, -0.974927912181824, -0.995379112949198, -0.999486216200688, -0.987181783414450, -0.958667853036661, -0.914412623015813, -0.855142763005347, -0.781831482468030, -0.695682550603487, -0.598110530491217, -0.490717552003938, -0.375267004879375, -0.253654583909508, -0.127877161684507, -1.13310777952960 \times 10^{-15}\right]$$

```
> Y := Quantize( X, -1 .. 1, 9 );
```

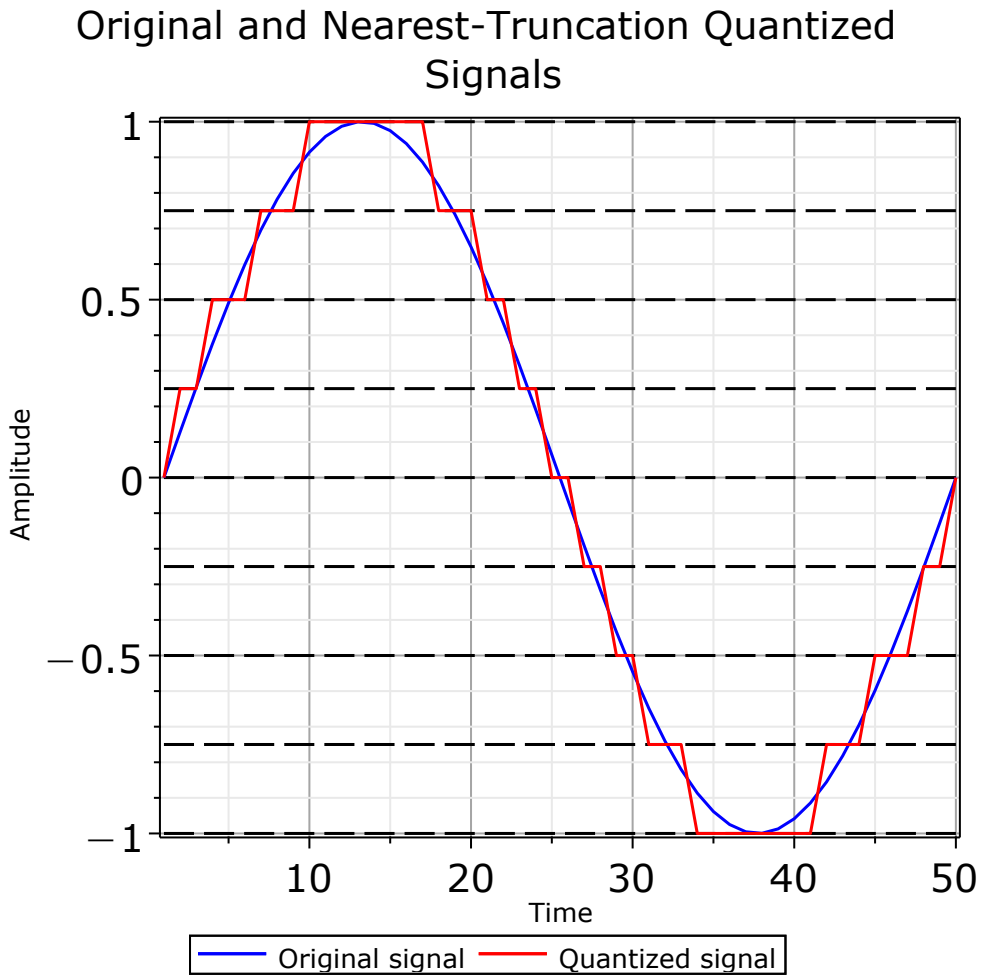
$$Y := [0., 0.250000000000000, 0.250000000000000, 0.500000000000000, 0.500000000000000, 0.500000000000000, 0.750000000000000, 0.750000000000000, 0.750000000000000, 1., 1., 1., 1., 1., 1., 1., 0.750000000000000, 0.750000000000000, 0.750000000000000, 0.500000000000000, 0.500000000000000, 0.250000000000000, 0.250000000000000, 0., 0., -0.250000000000000, -0.250000000000000, -0.500000000000000, -0.500000000000000, -0.750000000000000, -0.750000000000000, -0.750000000000000, -1., -1., -1., -1., -1., -1., -1., -1., -0.750000000000000, -0.750000000000000, -0.750000000000000, -0.500000000000000, -0.500000000000000, -0.500000000000000, -0.250000000000000, -0.250000000000000, 0.]$$

```
> Z := Quantize( X, < seq( 0.25 * k, k = -4 .. 4 ) > );
```

$$Z := [0., 0.250000000000000, 0.250000000000000, 0.500000000000000, 0.500000000000000, 0.500000000000000, 0.750000000000000, 0.750000000000000, 0.750000000000000, 1., 1., 1., 1., 1., 1., 1., 0.750000000000000, 0.750000000000000, 0.750000000000000, 0.500000000000000, 0.500000000000000, 0.250000000000000, 0.250000000000000, 0., 0., -0.250000000000000, -0.250000000000000, -0.500000000000000, -0.500000000000000, -0.750000000000000, -0.750000000000000, -0.750000000000000, -1., -1., -1., -1., -1., -1., -1., -1., -0.750000000000000, -0.750000000000000, -0.750000000000000, -0.500000000000000, -0.500000000000000, -0.500000000000000, -0.250000000000000, -0.250000000000000, 0.]$$

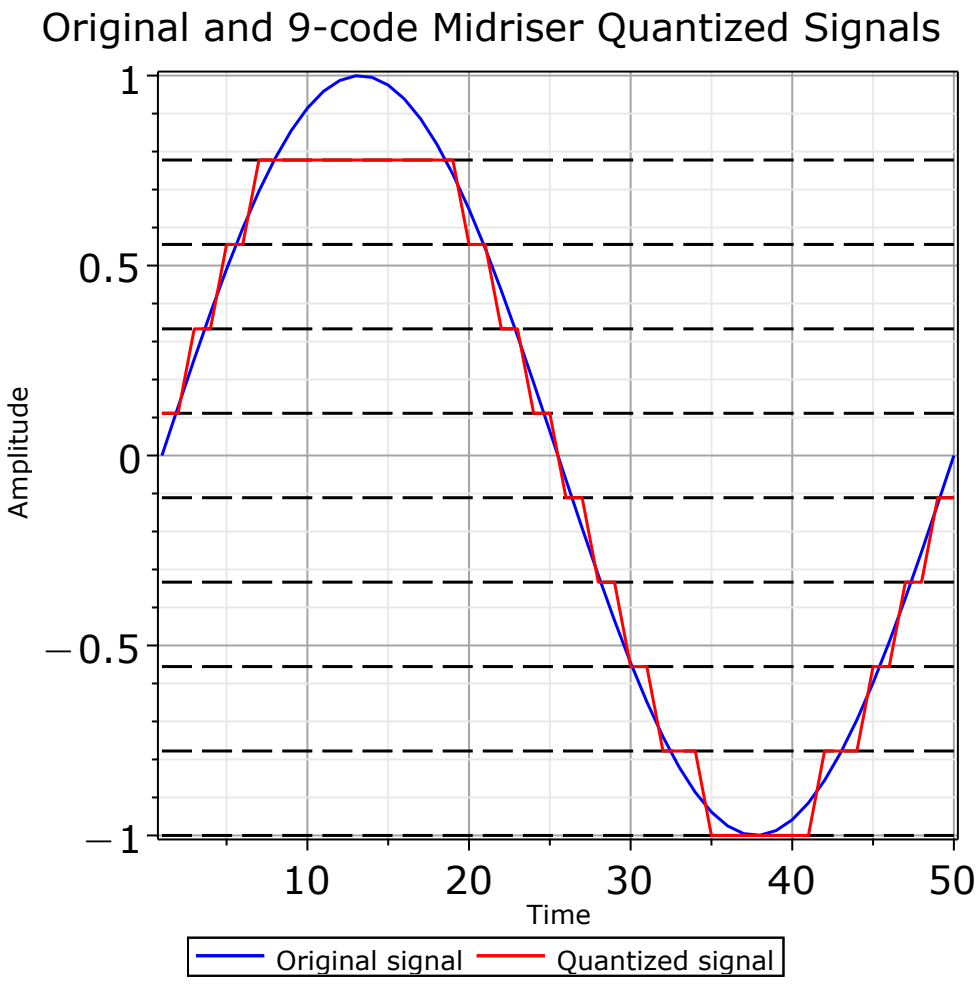
• The command can also display the original and quantized signals together:

```
> Quantize( X, -1 .. 1, 9, 'output' = 'plot', 'emphasizecodes' );
```



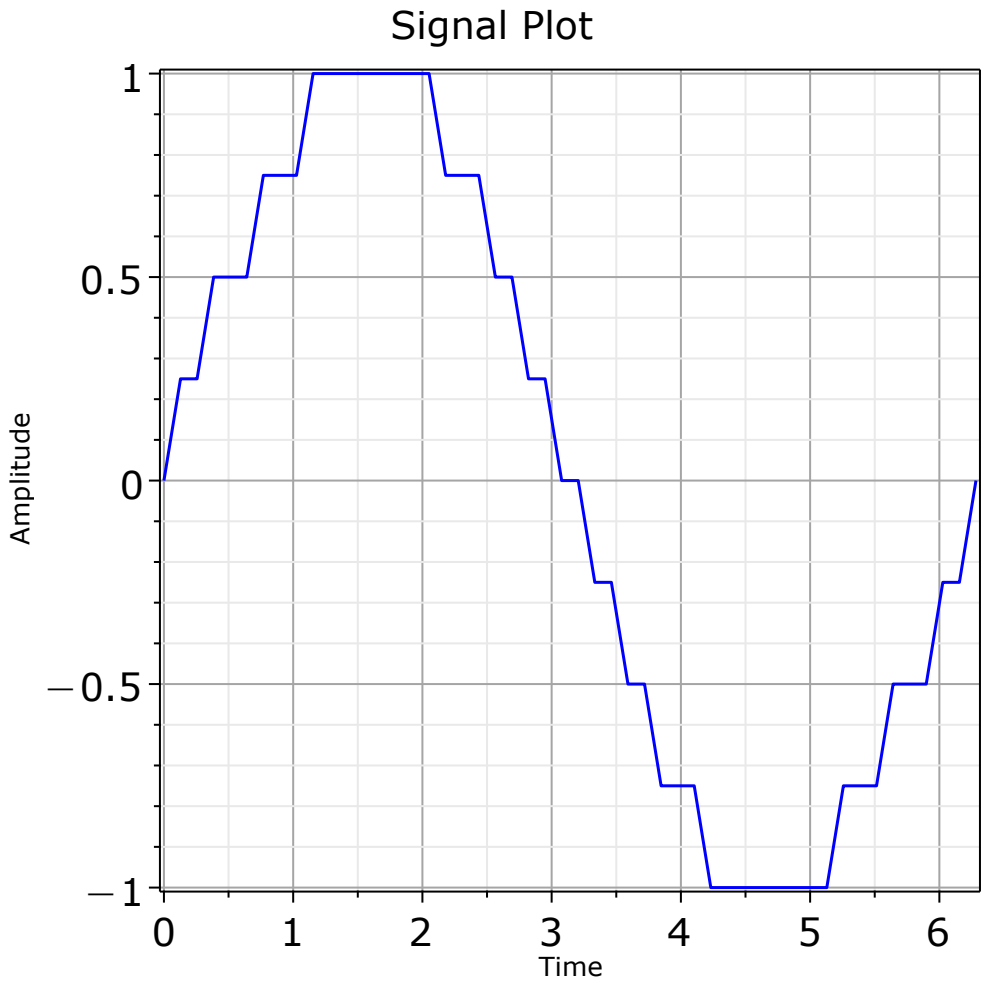
• The example above uses the default **nearest** for the option **truncation**, but **below** and **above** are also available. Moreover, classical quantization techniques **midriser** and **midtread** are provided:

```
> Quantize( X, 'midriser', 1, 9, 'output' = 'plot', 'emphasizecodes' )
  ;
```



- The SignalProcessing:-GenerateSignal command has also been updated to include the **quantization** option:

```
> GenerateSignal( sin(t), t = 0 .. 2 * Pi, 50, 'quantization' = [-1.
  .1,9], 'output' = 'signalplot' );
```



SavitzkyGolayFilter

- The new SignalProcessing:-SavitzkyGolayFilter command applies the Savitzky-Golay Filter to a signal, with applications including smoothing noisy data and estimating derivatives of data. The filter is also known as Polynomial Smoothing, Least-Squares Smoothing, and Locally Weighted Scatterplot Smoothing (LOWESS).
- For example, consider the following:

```
> f := sin(t) + 1/5 * cos(3*t); # original expression
```

$$f := \sin(t) + \frac{\cos(3\,t)}{5}$$

```
> t1 := 0; # start time
```

$$t1 := 0$$

```
> t2 := 2 * Pi; # finish time
```

$$t2 := 2\,\pi$$

```
> m := 200; # number of points
```

$$m := 200$$

```
> (T,X) := GenerateSignal( f, t = t1 .. t2, m, 'noisedeviation' = 0.2,
  'includefinishtime' = 'false', 'output' = ['times','signal'] );
```

$$T, X := \begin{bmatrix} 0. \\ 0.0314159265358979 \\ 0.0628318530717959 \\ 0.0942477796076938 \\ 0.125663706143592 \\ 0.157079632679490 \\ 0.188495559215388 \\ 0.219911485751286 \\ 0.251327412287183 \\ 0.282743338823081 \\ \vdots \end{bmatrix}, \begin{bmatrix} -0.0144848255996538 \\ 0.164707577889331 \\ 0.135829582293093 \\ 0.329060399502961 \\ 0.306202905133478 \\ 0.680400026713471 \\ 0.0292755488173467 \\ 0.690408678222541 \\ 0.428555296931219 \\ 0.612549160854321 \\ \vdots \end{bmatrix}$$

200 element Vector[column] 200 element Vector[column]

• The smoothing, both unweighted and weighted, noticeably reduces the noise:

```
> d := 2; # degree of polynomial interpolants
```

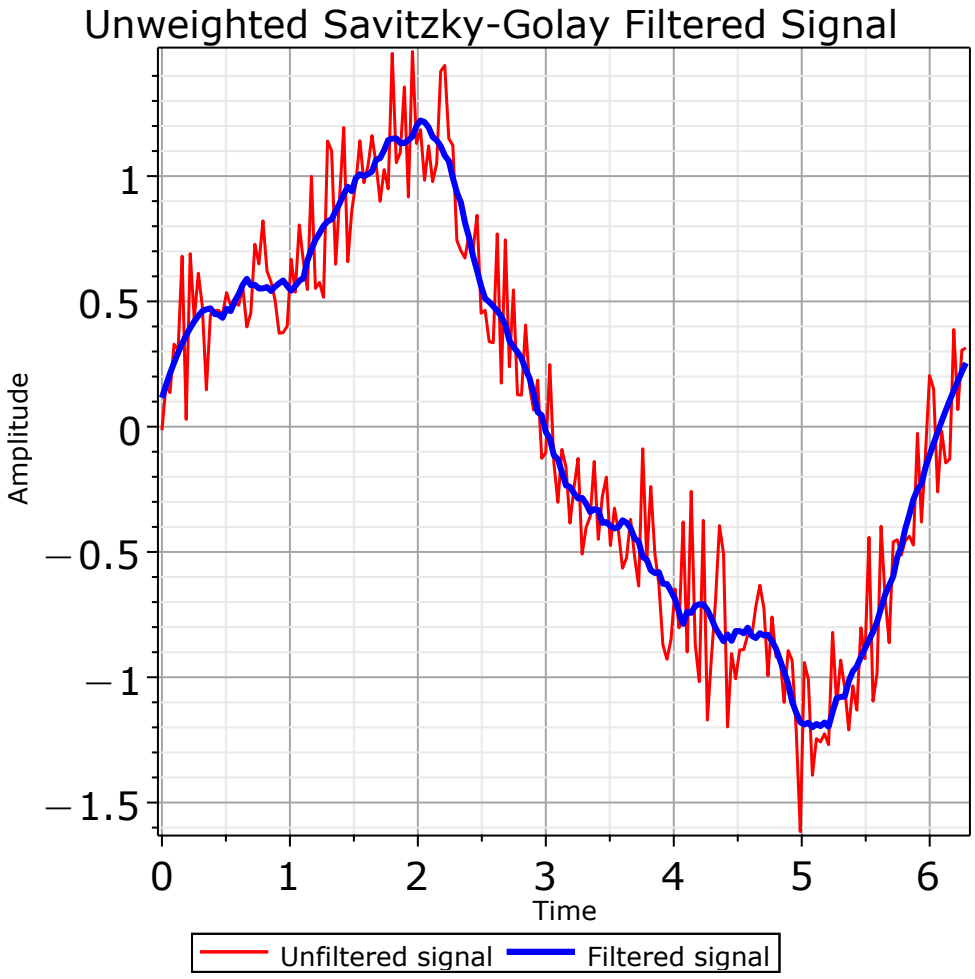
$$d := 2$$

```
> r := 10; # radius of frame
```

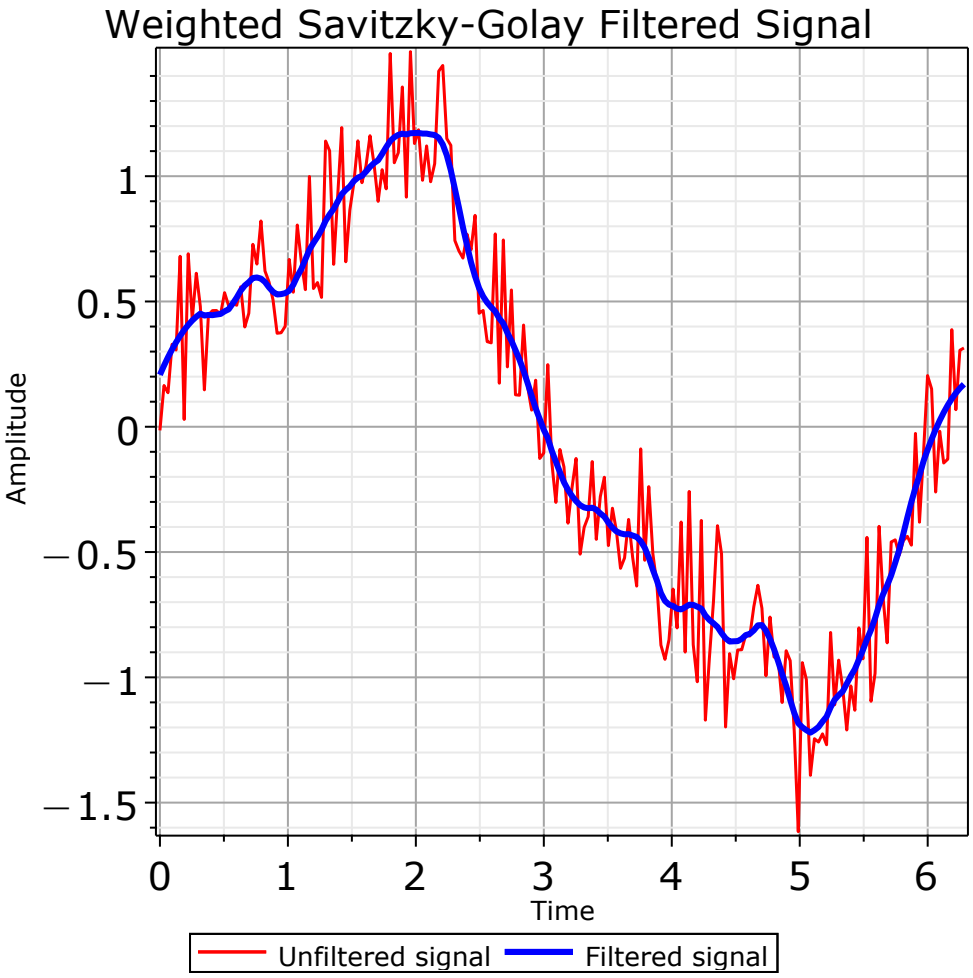
$$r := 10$$

```
> W := < seq( 1 .. r + 1 ), seq( r .. 1, -1 ) >: # weights
```

```
> SavitzkyGolayFilter( X, d, r, 'timerange' = t1 .. t2, 'plotoptions'
  = [ 'title' = "Unweighted Savitzky-Golay Filtered Signal" ],
  'output' = 'plot' );
```



```
> SavitzkyGolayFilter( X, d, W, 'timerange' = t1 .. t2, 'plotoptions'
  = [ 'title' = "Weighted Savitzky-Golay Filtered Signal" ], 'output'
  = 'plot' );
```



- The SignalProcessing:-DifferentiateData command now includes an option to estimate the derivative of a signal using the Savitzky-Golay Filter;

```
> (dt,T,U) := GenerateSignal( t * (t^2-1)^3, t = -1 .. 1, 75,
  'includefinishtime' = 'false', 'output' = ['timestep','times',
  'signal'] );
```

$$dt, T, U := 0.0266666666666666684, \begin{bmatrix} -1. \\ -0.973333333333333 \\ -0.946666666666667 \\ -0.920000000000000 \\ -0.893333333333333 \\ -0.866666666666667 \\ -0.840000000000000 \\ -0.813333333333333 \\ -0.786666666666667 \\ -0.760000000000000 \\ \vdots \end{bmatrix}, \begin{bmatrix} -0. \\ 0.000141830341814137 \\ 0.00105941970462669 \\ 0.00333396836352000 \\ 0.00735835867283022 \\ 0.0133619123914037 \\ 0.0214334806425600 \\ 0.0315429148381890 \\ 0.0435609668964129 \\ 0.0572776670822400 \\ \vdots \end{bmatrix}$$

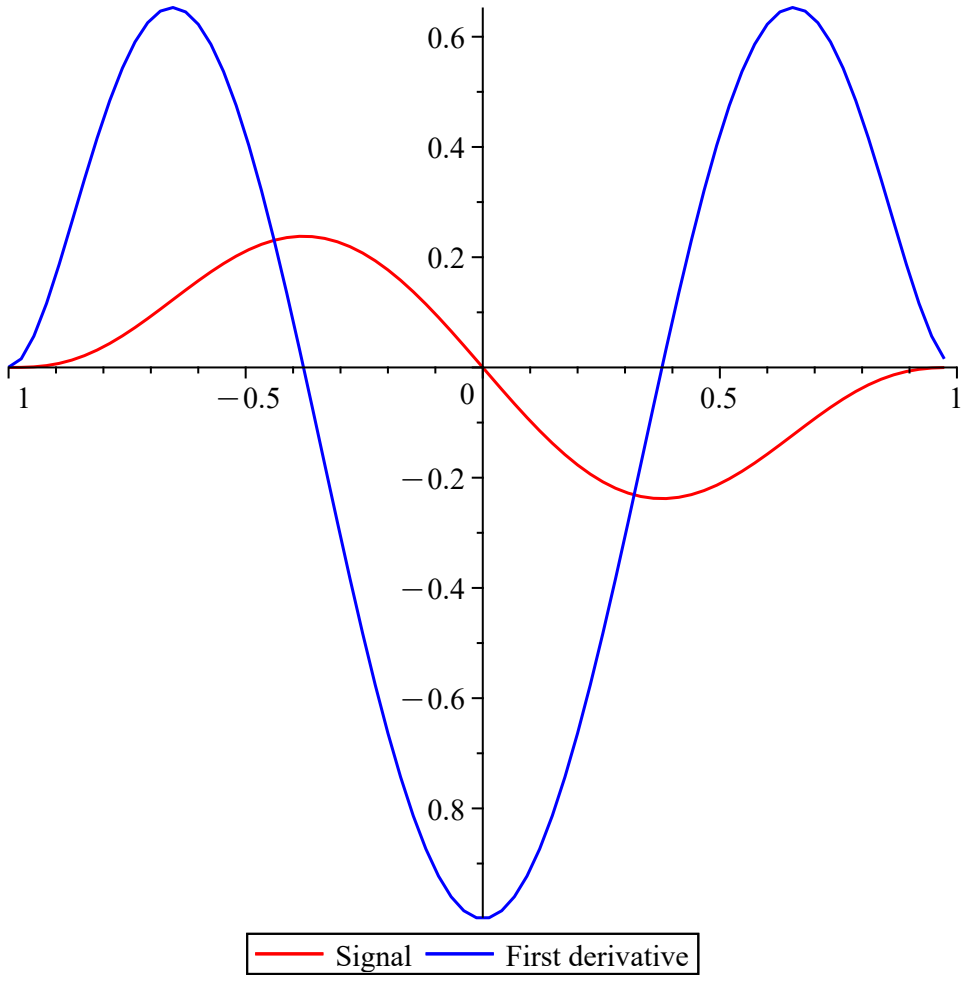
75 element Vector[column] 75 element Vector[column]

```
> V := DifferentiateData( U, 1, 'step' = dt, 'method' =
  'savitzkygolay', 'extrapolation' = 'periodic', 'frameradius' = 2 );
```

$$V := \begin{bmatrix} 0.000470143936790068 \\ 0.0156236216614980 \\ 0.0568085796900521 \\ 0.116160717800121 \\ 0.187029660266052 \\ 0.263725091509905 \\ 0.341391910470936 \\ 0.415950422937459 \\ 0.484038346232275 \\ 0.542954626251677 \\ \vdots \end{bmatrix}$$

75 element Vector[column]

```
> dataplot( T, [U,V], 'style' = 'line', 'legend' = ["Signal","First
  derivative"], 'color' = ['red','blue'], 'view' = [-1..1,'DEFAULT'] )
  ;
```



Convolution

- The SignalProcessing:-Convolution command has been updated to include shape options **full** (the default), **same**, and **valid**. For example:

```
> A := LinearAlgebra:-RandomVector( 5, 'generator' = -1.0 .. 1.0,
  'datatype' = 'float[8]' );
```

$$A := \begin{bmatrix} -0.844885936681858 \\ 0.604222951371852 \\ -0.434653429149560 \\ -0.809289661861375 \\ 0.964721117374344 \end{bmatrix}$$

```
> B := LinearAlgebra:-RandomVector( 3, 'generator' = -1.0 .. 1.0,
  'datatype' = 'float[8]' );
```

$$B := \begin{bmatrix} 0.360574124053584 \\ -0.983812180981714 \\ 0.254768677798811 \end{bmatrix}$$

```
> C1 := Convolution( A, B, 'shape' = 'full' );
```

$$C1 := [-0.304644006544253,\ 1.04907623747173,\ -0.966417152050085,\ 0.289745509588224,\ 1.03330641968989,\ -1.15528604363184,\ 0.245780723518053]$$

```
> C2 := Convolution( A, B, 'shape' = 'same' );
```

$$C2 := [1.04907623747173,\ -0.966417152050085,\ 0.289745509588224,\ 1.03330641968989,\ -1.15528604363184]$$

```
> C3 := Convolution( A, B, 'shape' = 'valid' );
```

$$C3 := \begin{bmatrix} -0.966417152050085 & 0.289745509588224 & 1.03330641968989 \end{bmatrix}$$

- Moreover, lists can now be passed for signals in addition to rtables.

FFT and InverseFFT

- The SignalProcessing:-FFT and SignalProcessing:-InverseFFT commands now support padding and truncation. Padding is helpful due to the increase in both the speed of computation and the frequency resolution. The **size** option accepts a positive integer for the new size and keywords **same** (the default, for not changing the length) and **recommended** (for changing the length to the next power of **2**). For example:

```
> X := LinearAlgebra:-RandomVector( 3, 'generator' = -1.0 .. 1.0,
  'datatype' = 'complex[8]' );
```

$$X := \begin{bmatrix} -0.235334127009411 + 0.524842563866319\,I \\ -0.600897572737194 - 0.723997348520040\,I \\ 0.0678662154384722 - 0.122666338816995\,I \end{bmatrix}$$

```
> Y1 := FFT( X, 'size' = 'same' );
```

$$Y1 := \begin{bmatrix} -0.443616019201318 - 0.185803512266725\,I \\ -0.282662827618448 + 0.881810643516785\,I \\ 0.318668182084598 + 0.213046855341119\,I \end{bmatrix}$$

```
> Y2 := FFT( X, 'size' = 'recommended' );
```

$$Y2 := \begin{bmatrix} -0.384182742154067 - 0.160910561735358\,I \\ -0.513598845483962 + 0.624203237710254\,I \\ 0.216714830583127 + 0.563086786784682\,I \\ 0.210398503036078 + 0.0233056649730601\,I \end{bmatrix}$$

```
> Y3 := FFT( X, 'size' = 5 );
```

$$Y3 := \begin{bmatrix} -0.343623490895507 - 0.143922781735175\,\mathrm{I} \\ -0.553020070919267 + 0.416781168114568\,\mathrm{I} \\ -0.0165998120551015 + 0.666529860535818\,\mathrm{I} \\ 0.259681898881381 + 0.292888860328475\,\mathrm{I} \\ 0.127338369569880 - 0.0586934569533205\,\mathrm{I} \end{bmatrix}$$

- The **size** option also works with multi-dimensional input. Moreover, lists can now be passed for signals in addition to rtables.