SignalProcessing Improvements in Maple 2024

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

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

ResponseSpectrum

- The new SignalProcessing:-ResponseSpectrum command is used to plot the response of a structure or system to varying frequencies of ground motion or input excitation. A response spectrum is commonly used in structural and earthquake engineering to assess the potential response of a structure to seismic events.
- Consider the following vibration data from the El Centro earthquake of 1940:

```
> file := FileTools:-JoinPath( [ kernelopts( 'datadir' ), "datasets", "el-centro_NS.txt" ] ):
```

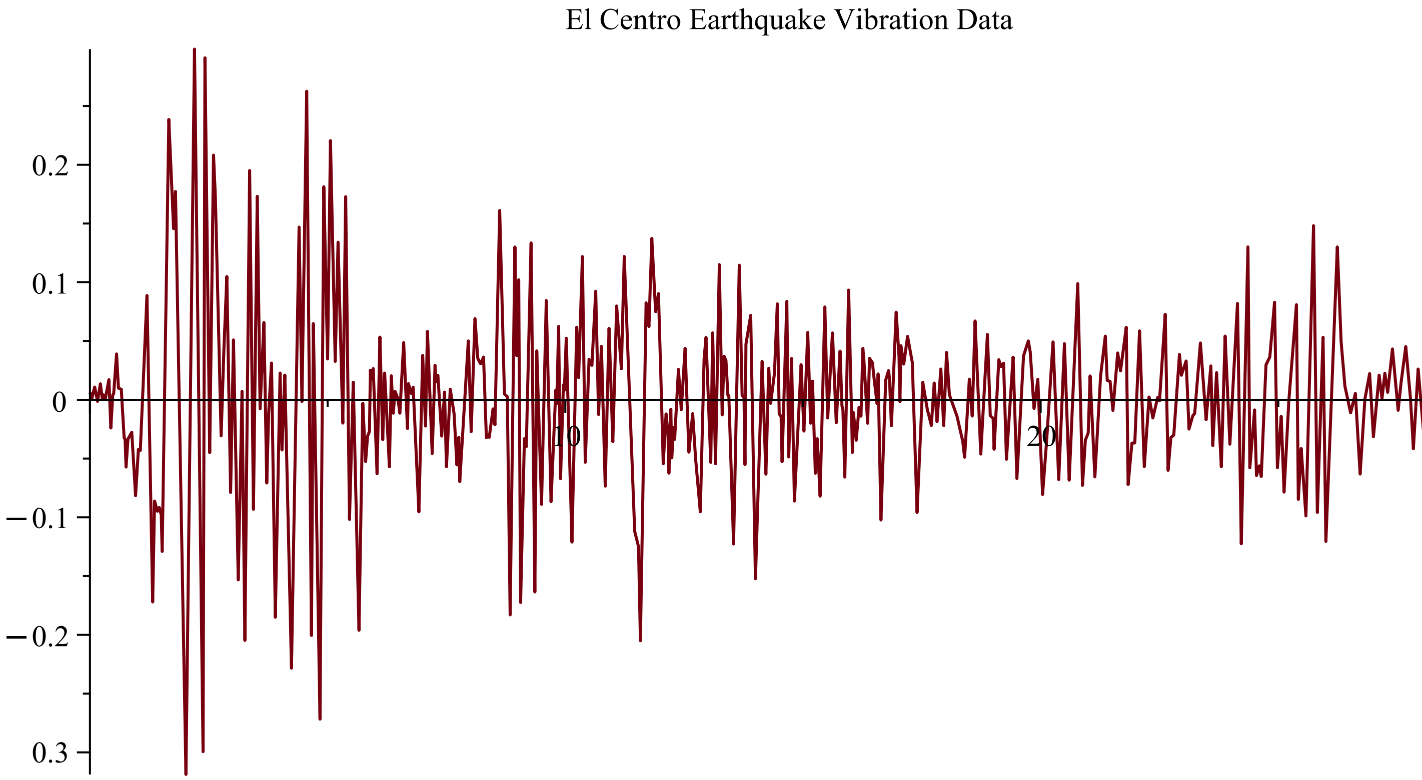
```
> data := ImportMatrix( file, 'delimiter' = " " );
```

data :=

	1	2
1	0.	0.00630000000000000
2	0.0200000000000000	0.00364000000000000
3	0.0400000000000000	0.000990000000000000
4	0.0600000000000000	0.00428000000000000
5	0.0800000000000000	0.00758000000000000
6	0.100000000000000	0.0108700000000000
7	0.120000000000000	0.00682000000000000
8	0.140000000000000	0.00277000000000000
9	0.160000000000000	−0.00128000000000000
	⋮	⋮

1560 × 2 Matrix

```
> dataplot( data[..,1], data[..,2], 'style' = 'line', 'color' = 'burgundy', 'title' = "El Centro Earthquake Vibration Data", 'size' = [800,400] );
```

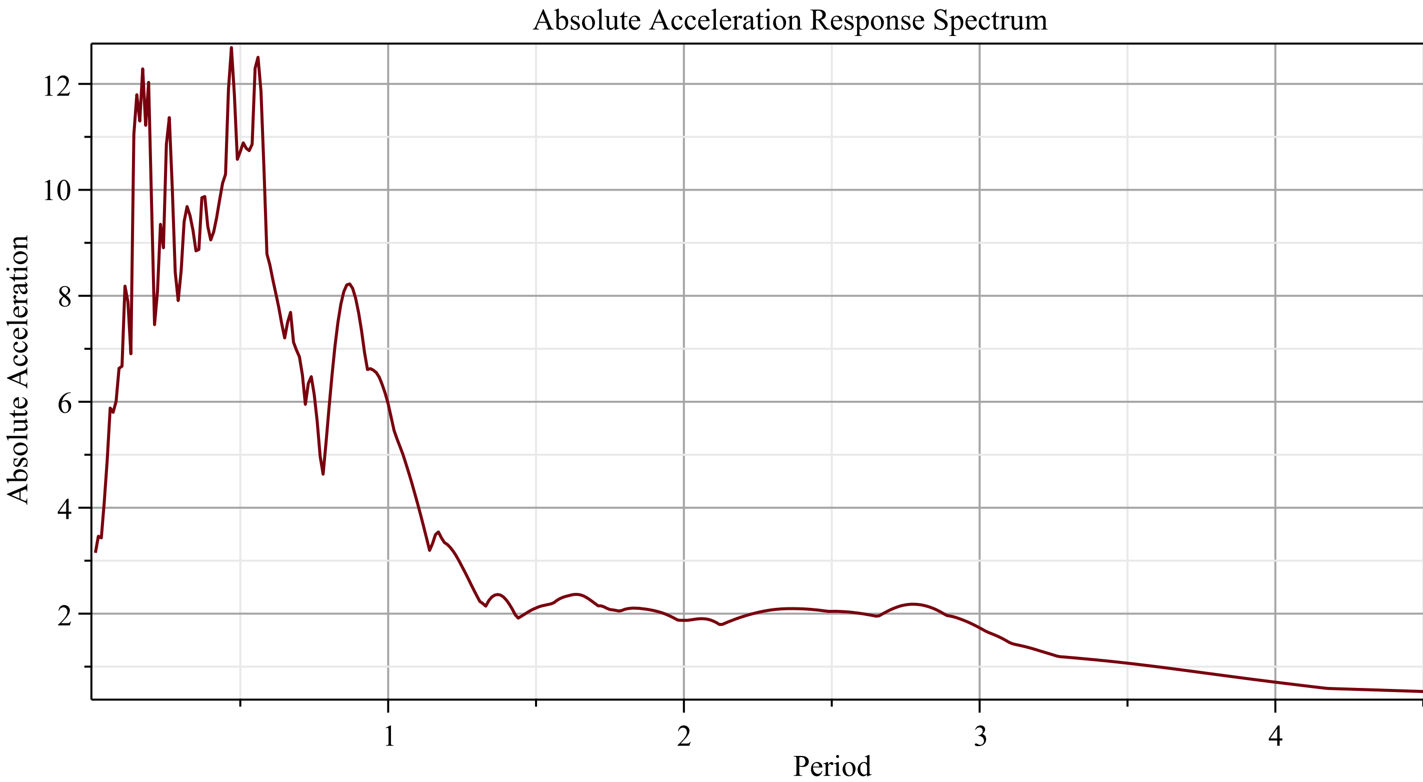


- For appropriate choices of parameters, we can obtain a collection of charts and data containers:

```
> R := ResponseSpectrum( data, 0.02, 0.01, 5, 'zeta' = 0.02, 'beta' =
  0.25, 'gamma' = 0.5, 'output' = 'record' ):
```

• For example, we can view the plot of acceleration:

```
> R['absoluteaccelerationplot'];
```



IntegrateData and IntegrateData2D

• The SignalProcessing:-IntegrateData command has been updated to include an option initial to specify the initial area and output option running to return running totals. For example, suppose we want to determine position from velocity:

```
> t1 := 0.0:
```

```
> t2 := 2.0:
```

```
> n := 100:
```

```
> ( dt, T, V ) := GenerateSignal( t * exp(-t/2) * sin(2*t*Pi), t = t1
  .. t2, n, 'output' = ['timestep','times','signal'] );
```

$dt, T, V := 0.0202020202020202,$

1	0.
2	0.0202020202020202
3	0.0404040404040404
4	0.0606060606060606
5	0.0808080808080808
6	0.101010101010101
7	0.121212121212121
8	0.141414141414141
9	0.161616161616162
	⋮

100 element Vector[column]

,

1	0.
2	0.00253172077517917
3	0.00994445238601361
4	0.0218526600137242
5	0.0377328474375233
6	0.0569400702908049
7	0.0787270525835511
8	0.102265426482862
9	0.126668579468995
	⋮

100 element Vector[column]

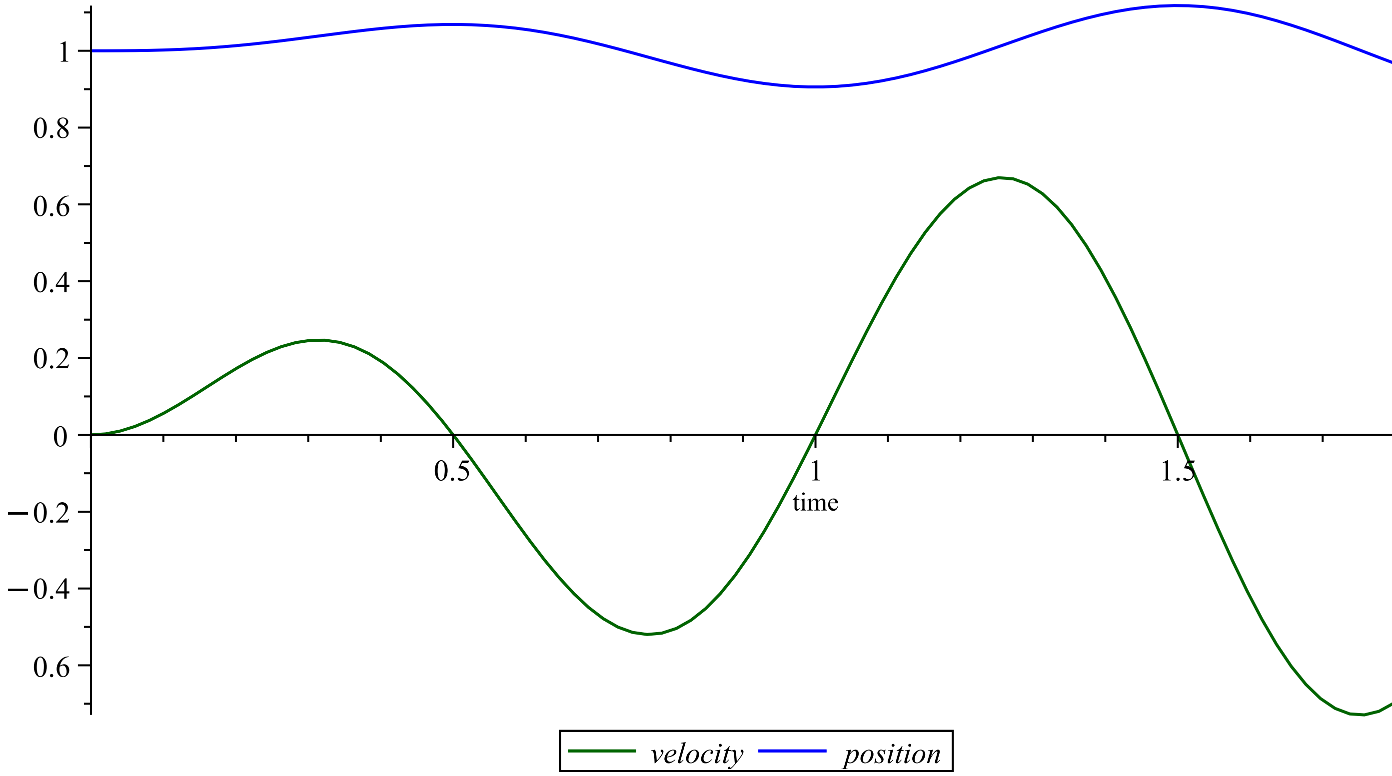
```
> x0 := 1.0:
> X := IntegrateData( V, 'step' = dt, 'initial' = x0, 'method' =
  'trapezoid', 'output' = 'running' );
```

$X :=$

1	1.
2	1.00002557293712
3	1.00015159488825
4	1.00047277784178
5	1.00107465165442
6	1.00203094375268
7	1.00340131873121
8	1.00522952559047
9	1.00754199029705
	⋮

100 element Vector[column]

```
> dataplot( T, [ V, X ], 'color' = ['darkgreen','blue'], 'labels' =
  ["time",""], 'legend' = ['velocity','position'], 'style' = 'line',
  'size' = [800,400] );
```



- The SignalProcessing:-IntegrateData and SignalProcessing:-IntegrateData2D commands have also been updated to be units aware. For example:

```
> T := Vector( [1,2,3], 'datatype' = 'float[8]' ) * Unit('s');
```

$$T := \begin{bmatrix} 1.\ \mathrm{s} \\ 2.\ \mathrm{s} \\ 3.\ \mathrm{s} \end{bmatrix}$$

```
> V := Vector( [1,4,9], 'datatype' = 'float[8]' ) * Unit('m/s');
```

$$V := \begin{bmatrix} 1.\ \frac{\mathrm{m}}{\mathrm{s}} \\ 4.\ \frac{\mathrm{m}}{\mathrm{s}} \\ 9.\ \frac{\mathrm{m}}{\mathrm{s}} \end{bmatrix}$$

```
> IntegrateData( V, 'step' = 1.0 * Unit('s') );
```

$$9.\ \mathrm{m}$$

```
> IntegrateData( T, V );
```

$$9.\ \mathrm{m}$$

```
> X := IntegrateData( T, V, 'output' = 'running' );
```

$$X := \begin{bmatrix} 0. \\ 2.50000000000000\ \mathrm{m} \\ 9.\ \mathrm{m} \end{bmatrix}$$

FindPeakPoints

- The heavy computations involved in the SignalProcessing:-FindPeakPoints command have been moved to external code, resulting in a substantial increase in speed. For example:

```
> n := 25000:
> A := Vector( n, i -> 1 - i * (-1)^(i+1), 'datatype' = 'float[8]' ):
> R := CodeTools:-Usage( FindPeakPoints( A, 'output' = 'record' ),
  'iterations' = 10 ):
memory used=5.59MiB, alloc change=23.43MiB, cpu time=44.70ms, real time=
37.80ms, gc time=10.84ms
```

- The CPU time required in Maple 2024 is about 100 times smaller than in Maple 2023.
- Two new output options, extremes and extremeindices, have also been added to the command:

```
> f := sin(t) + 1/2 * cos(3*t):
> a := 0:
> b := 2 * Pi:
> n := 100:
```

```
> ( T, X ) := GenerateSignal( f, t = a .. b, n, 'output' = ['times',
  'signal'] );
```

$$T, X := \left[\begin{array}{cc} 1 & 0. \\ 2 & 0.0634665182543393 \\ 3 & 0.126933036508679 \\ 4 & 0.190399554763018 \\ 5 & 0.253866073017357 \\ 6 & 0.317332591271696 \\ 7 & 0.380799109526036 \\ 8 & 0.444265627780375 \\ 9 & 0.507732146034714 \\ & \vdots \end{array}\right]_{\text{100 element Vector[column]}}, \left[\begin{array}{cc} 1 & 0.500000000000000 \\ 2 & 0.554388268287918 \\ 3 & 0.590776420081786 \\ 4 & 0.609878010776001 \\ 5 & 0.613015006233614 \\ 6 & 0.602061900484086 \\ 7 & 0.579369962161271 \\ 8 & 0.547674379843885 \\ 9 & 0.509987694012340 \\ & \vdots \end{array}\right]_{\text{100 element Vector[column]}}$$

```
> E := FindPeakPoints( T, X, 'output ' = 'extremes' );
```

$$E := \left[\begin{array}{cc} 0. & 0.500000000000000 \\ 0.253866073017357 & 0.613015006233614 \\ 0.888531255560750 & 0.331728739964295 \\ 2.03092858413886 & 1.38695812292269 \\ 3.36372546747998 & -0.613337080157934 \\ 4.06185716827771 & -0.331577874022796 \\ \vdots & \vdots \end{array}\right]$$

```
> p := dataplot( T, X, 'style' = 'line', 'legend' = "Signal", 'color'
  = 'firebrick' ):
> q := dataplot( E[..,1], E[..,2], 'style' = 'line', 'legend' =
  "Extremes", 'color' = 'blue' ):
```

```
> plots:-display( p, q, 'size' = [800,400] );
```