Programming Updates in Maple 2024

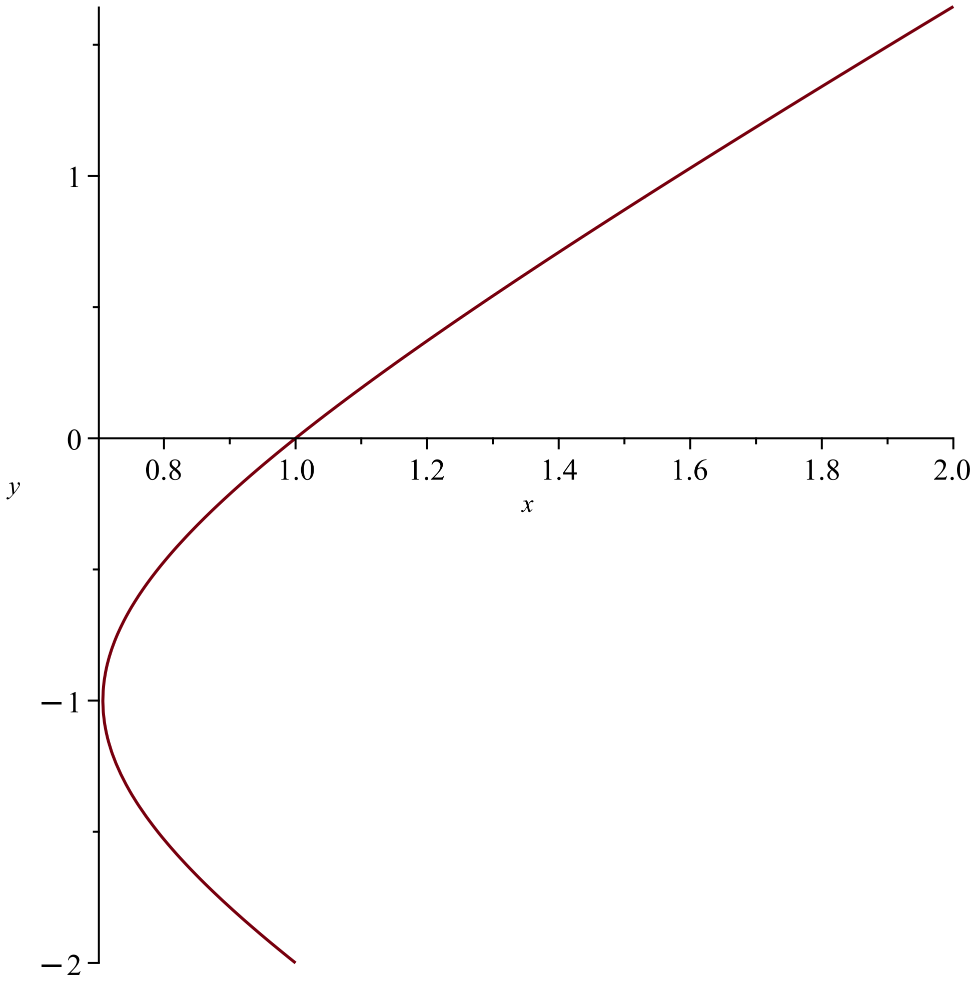
Series Conversion to Polynomial

- By customer request, the series command now supports an option, oterm=false. This option causes the series command to apply convert(s,polynom) before returning. The returned expression is then a standard polynomial data structure with no order term, rather than a specialized SERIES data structure. This is useful, for example, for plotting the result.

```
> S1 := series(exp(y),y=0,3,oterm=false);
```

$$S1 := 1 + y + \frac{1}{2} y^2$$

```
> plots:-implicitplot(x^2=S1, x=0..2, y=-2..2);
```



Mapping over Elements of an Array

- The normal and expand commands now map over the elements of an array.

```
> A := Array( [ (x^2-1)/(x+1), (2*x^2+10*x+12)/(x+3) ]);
```

$$A := \left[\begin{array}{cc} \frac{x^2-1}{x+1} & \frac{2x^2+10x+12}{x+3} \end{array}\right]$$

```
> normal(A);
```

$$\left[\begin{array}{cc} x-1 & 2x+4 \end{array}\right]$$

```
> expand(A);
```

$$\left[\begin{array}{cc} \frac{x^2}{x+1}-\frac{1}{x+1} & \frac{2x^2}{x+3}+\frac{10x}{x+3}+\frac{12}{x+3} \end{array}\right]$$

```
> Normal(Vector([444*x])) mod 11;
```

$$\left[\begin{array}{c} 4x \end{array}\right]$$

evalhf

- A call to evalhf evaluates an expression to a numerical value using the floating-point hardware of the underlying system. Maple includes evalhf for the purpose of gaining speed in numerical computations when you know that infinite precision floating point computations are not needed. When appropriate, you can use evalhf on expressions that contain calls to your own custom procedures, and in Maple 2024, the scope has been expanded to include procedures that employ try/catch statements.

```
> p := proc( a, b )
     local r;
     try
         if b = 0 then
            error "test";
         else
            r := a/b;
         end if;
     catch:
         r := undefined;
     end try;
     return r;
  end proc:
```

```
> evalhf( p(1,0) );
```

$$\mathrm{Float}(\mathrm{undefined})$$

Array Indexing Functions

- An indexing function is a Maple procedure that can be applied to an array in order to control how values are inserted and extracted from an array. Prior versions of Maple required indexing functions to be assigned to a global name of the form index/my_indexing_fn. Now they can be applied directly as a procedure using the shape option.

```
> weighted := proc( idx :: list, A::rtable, val::list );
      if nargs = 3 then
          # assign
          A[op(idx)] := op(val);
      else
          # retrieve
          local i := op(idx);
          local ls := `if`(i=1,A[i],A[i-1]);
          local rs := `if`(i=numelems(A),A[i],A[i+1]);
          return (ls + 2*A[i] + rs)/4.;
      end if;
  end proc:
```

```
> A := Array(1..10,i->i^2,shape=weighted);
```

$$A := \left[\begin{array}{ccccccc} 1.750000000 & 4.500000000 & 9.500000000 & 16.50000000 & 25.50000000 & 36.50000000 & 49.500000 \cdots \end{array}\right]$$

- For this sample indexing function, the value retrieved is weighted by the surrounding elements.

```
> A[4];
```

$$16.50000000$$

```
> A[4] := 77;
```

$$A_4 := 77$$

```
> A[4];
```

$$47.00000000$$

- See rtable_indexfcn for more details about the structure of an indexing function.

Growing an Aliased Array

- The ArrayTools:-Alias command provides a way to create an array that points directly to the storage of another array, providing a different view of the same data. This provides an efficient way to reference a column or block of data without requiring a copy of the original. Maple 2024 tightens up the rules for when the source or aliased array grows, requiring allocation memory to hold the data.
- In this example, the source and target arrays are both one-dimensional, and the alias can be unambiguously updated to maintain the link to the parent.

```
> source := LinearAlgebra:-RandomVector( 4, datatype=float[8] ):
```

```
> V := ArrayTools:-Alias( source ):
```

```
> ArrayTools:-Extend( source, inplace, LinearAlgebra:-RandomVector( 4,
  datatype=float[8] ) ):
```

```
> source;
```

$$\begin{bmatrix} -25. \\ 40. \\ 97. \\ 43. \\ -7. \\ 12. \\ -53. \\ \vdots \end{bmatrix}$$

```
> V[1] := 1;
```

$$V_1 := 1$$

```
> source[1];
```

$$1.$$

- In this example, the source matrix is two-dimensional, and growing will change the order of the underlying data. Therefore, the alias must be invalidated.

```
> source := LinearAlgebra:-RandomMatrix( 2, datatype=float[8] ):
```

```
> M := ArrayTools:-Alias( source ):
```

```
> source(3,3) := 3;
```

$$source := \begin{bmatrix} -70. & -58. & 0. \\ 13. & -94. & 0. \\ \vdots & \vdots & \vdots \end{bmatrix}$$

- This command will raise an error:

```
> M[1,1] := 1;
Error, (in index/rtableAliasError) the parent rtable of this alias has changed its root pointer, thus invalidating this rtable
```

- In this example, the aliased matrix grows, and thus is disconnected from the original parent. Both have independent data after the resize operation.

```
> source := LinearAlgebra:-RandomMatrix( 2, datatype=float[8] ):
> M := ArrayTools:-Alias( source ):
> M(3,3) := 3;
```

$$M := \begin{bmatrix} 89. & -67. & 0. \\ -55. & 77. & 0. \\ \vdots & \vdots & \vdots \end{bmatrix}$$

```
> M[1,1] := 1;
```

$$M_{1,1} := 1$$

```
> source[1,1];
```

$$89.$$

Element-wise Operations

- Tilde (~) can be used after an operator or function name in order to have it applied element-wise, that is, to the elements of a container rather than the container itself. When using natural-math notation, it is not intuitive to use this mechanism for operators like over-bar division, superscript exponentiation, and square roots. New in Maple 2024 is the elementwise function, which, when used, applies all of the operations in the given expression in an element-wise manner. Non-elementwise operators and functions can be used, and they are all interpreted as being element-wise. Specifically, +, -, *, ., /, ^, abs, sqrt, log, ceil, floor, round, trunc, frac, and all the trig and log functions will be applied element-wise.

For example:

```
> A := [5,9]:
> B := [3,4]:
> C := [5,6]:
> G := [16,36]:
> elementwise(A*B/C^2*sqrt(G));
```

$$\left[\frac{12}{5}, 6\right]$$

The tilde function is a synonym for the elementwise function:

```
> `~`(A*B/C^2*sqrt(G));
```

$$\left[\frac{12}{5}, 6\right]$$

- For more information, see operators/elementwise.

Converting Logarithms to a Different Base

- Maple prefers to deal with computations using natural logarithms, and therefore automatically converts log[b](x) to ln(x)/ln(b). When dealing with step-by-step solutions intended to be viewed by students this conversion is not always intuitive. A conversion routine has been added to convert between log bases. Note the use of the % prefix denotes InertForm, which stops evaluation avoiding automatic conversion back to ln.

```
> ex1 := log[10](x);
```

$$ex1 := \frac{\ln(x)}{\ln(10)}$$

```
> convert( ex1, %log[10] );
```

$$\log_{10}(x)$$

```
> convert( ln(x), %log[10] );
```

$$\frac{\log_{10}(x)}{\log_{10}(\mathrm{e})}$$

```
> convert( ln(x)/ln(10) - log[100](x), %log[10]);
```

$$\frac{\log_{10}(x)}{2}$$

Fenwick Tree

- A Fenwick tree, or binary indexed tree, is a data structure for quickly computing sums of values in an array that undergoes changes. It was proposed by Ryabko [1] and later described by Fenwick [2].
- Maple 2024 now supports this data structure. It is described on the FenwickTree help page.

```
> F := FenwickTree([seq(1 .. 100)]);
```

$$F := < \textit{a Fenwick tree with 100 entries} >$$

```
> RangeSum(F, 17, 83);
```

$$3350$$

```
> add(17 .. 83);
```

$$3350$$

Objects

- piecewise can now handle conditions involving objects:

```
> module my_object()
  option object;
  export Piecewise :: static;
  export piecewise :: static := proc()
      return Piecewise(_passed);
  end proc;
  local ModulePrint :: static := proc(self, $)
   return 'my_object';
  end proc;
  end module:
```

```
> piecewise(my_object < 0, 1, 0);
```

$$Piecewise(my\_object < 0, 1, 0)$$

References

[1]: Boris Ryabko (1989). *A fast on-line code*. Soviet Math. Dokl. 39 (3): 533–537.

[2]: Peter M. Fenwick (1994). *A new data structure for cumulative frequency tables*. Software: Practice and Experience. 24 (3): 327–336.