Language and Programming

Maple 2017 includes the following enhancements to Maple language and programming facilities.

Improvements to Debugging Facilities

Debugging with Source Code

The function **`showsource`** (and the corresponding debugger command of the same name) takes a procedure name and statement number or range thereof, and displays the source line(s) corresponding to the specified statement(s). The lines are displayed preceded by one or two numbers, the first being the source line number, and the second being the statement number if the source line corresponds to a statement.

The existing function and debugger command, **`stopat`**, can be used to set breakpoints based on source line numbers by calling it in the form, **`stopat(filePathString,lineNums)`**, where **`filePathString`** is a string (instead of a procedure name).

Note that to do source level debugging, you must *have* the source. You cannot do source debugging on libraries provided only in **.mla** format (such as the libraries that ship with Maple).

Inspecting Variables on the Maple Stack

The **`inspect`** debugger command is a general purpose command for inspecting the current state of the procedure activation stack. It can be used to inspect parameters, local variables, and the code of currently active procedures.

In support of the inspect command, the output of the **`where`** debugger command (which shows the activation stack) now has numbered separating lines between the displayed stack levels.

Counted Breakpoints

The **stopat** function and debugger command can now be passed an integer (N) or range of integers (N..M) for its optional **condition** argument. This specifies that the debugger should be invoked only the Nth (through Mth) time that the specified statement is executed.

Depth-limited Tracing

The **trace** function now takes an optional keyword argument, **depth=N**, where N is an integer, and limits tracing to N levels of invocation of the specified procedure. This makes it easier to trace highly recursive procedures when only the top few levels are of interest.

Miscellaneous Changes and Improvements

The output of **showstat** now respects the settings of **interface(indentamount)** and **interface(longdelim)**.

New Commands

index

The index command constructs an indexed expression. The calling sequence $index(f, x)$ is equivalent to constructing the expression $f[x]$.

```
> index( f, x );
```

$$f_x$$

If the first argument, f, is an indexable expression, then the **index** command returns the result of indexing the first argument by the second argument:

```
> index( [a, b, c ], 2 );
```

$$b$$

MTM:-unwrap

The MTM:-unwrap command modifies phase angles in radians in the array **M** by adding integer multiples of $2\ \mathrm{Pi}$ to ensure the difference between consecutive elements in the result is less than Pi.

```
> A := <3.8699, 0.62832, -1.2566, 0>;
```

$$A := \begin{bmatrix} 3.8699 \\ 0.62832 \\ -1.2566 \\ 0 \end{bmatrix}$$

```
> MTM:-unwrap( A );
```

$$\begin{bmatrix} 3.8699 \\ 6.911505308 \\ 5.026585308 \\ 6.283185308 \end{bmatrix}$$

verify/wildcard

The verify/wildcard command verifies a relation between two expressions, independent of variable names.

For example, the following returns true because both expressions contain a single name and substituting y for x in the first expression yields the second:

```
> verify(x^2-x, y^2-y, wildcard)
```

$$true$$

Updated Commands

andmap and ormap

Maple uses three-valued logic (**true**, **false**, **FAIL**) for its Boolean operators. The commands andmap and ormap now use the same three-valued logic. For example, the following returns **FAIL** because **true and FAIL** evaluates to **FAIL**.

```
> andmap(x->evalb(x<4),[3,I])
```

$$FAIL$$

In prior versions of Maple, this example raised an exception.

Improvements to max and min

When programming with max and min, you now have a new way to handle cases when the function is called with no arguments or only empty container arguments. You can now opt to have max (or min) return **NULL** in those cases instead of negative (or positive) infinity by using the index **nodefault**.

```
> min();
```

$$\infty$$

```
> min[nodefault]();
```

Sorting of Data Frames, Data Series, and Objects

You can now sort data frames and data series.

Additionally, object overloading of the sort function is now supported.