Contents

Part I.

New Functionality

1. Statistics

1.1. Introduction

MuPAD 2.5 includes 70 new functions in the statistics library `stats`. These include random number generators, quantiles, cumulative and probability densities for 16 types of parameterized distributions, three goodness-of-fit-tests, and other statistical functions such as the calculation of covariances, mean values, or regression parameters. Further, the new function `plot::boxplot` for generating box plot representations of statistical samples was provided.

1.2. Example use

To give an impression of the new `stats` library at work, we present a classical statistical problem: Given a set of data, test whether it fits the distribution predicted by some mathematical model of reality. For this example, we generate a sample—in a more realistic situation, this data would be the output of some experiment:

```
>> rng := stats::logisticRandom(0, 1):
   data := [rng() $ i = 1..1000]:
```

We are going to use the χ^2-test on these data to check, whether they do in fact satisfy the logistic distribution and how sensitive the test is with respect to variations of the parameters of the distribution. The documentation of this test tells us that for a sample of size 1000, we should use a division into 32 cells. The function `stats::equiprobableCells` is used in conjunction with the quantile function of the logistic distribution to generate a cell division to compare the data with:

```
>> cells := stats::equiprobableCells(32,
                         stats::logisticQuantile(0, 1)):
   stats::csGOFT(data, cells,
                         CDF = stats::logisticCDF(0, 1))

                    [23.04, 0.1523191895, 31.25]
```

In this result, the second number, $0.15\ldots$, is the significance level of the test, i.e., the conditional probability that the null hypothesis is false, given the observed data. It is well below 1, so the test does not justify rejecting the hypothesis that the data is distributed as proclaimed—as it should not, since we generated the data exactly that way using `stats::logisticRandom` with the corresponding parameters. Changing the parameters of the assumed distribution, however, we find much higher values of the significance level indicating a worse fit:

```
>> cells := stats::equiprobableCells(32,
                     stats::logisticQuantile(0.2, 1)):
   stats::csGOFT(data, cells,
                     CDF = stats::logisticCDF(0.2, 1))

                 [59.2, 0.9983250516, 31.25]

>> cells := stats::equiprobableCells(32,
                     stats::logisticQuantile(0, 1.2)):
   stats::csGOFT(data, cells,
                     CDF = stats::logisticCDF(0, 1.2))

                [67.648, 0.9998457666, 31.25]
```

2. Numerics

2.1. Extensions to the `numeric` Library

The `numeric` library has been extended in two ways:

Firstly, the `Scilab` floating point tool was connected to MuPAD and some routines in the `numeric` package were changed to make use of the link if scilab is installed and the current floating point precision permits the use of hardware floats. This work is the topic of the next chapter.

Secondly, the following new functions have been implemented:

- `numeric::cubicSpline2d` computes a cubic spline interpolation of scalar data over a rectangular grid.

- `numeric::leastSquares` calculates a least square solution of a linear system, i.e., it solves an optimization problem.

- `numeric::odesolveGeometric` solves an ordinary differential equation numerically, under the side condition that the solution is constrained to a homogeneous manifold.

- `numeric::ode2vectorfield` transforms an ordinary differential equation in the form accepted by `ode` into the vector-field representation required by the numerical integrators `numeric::odesolve` und `numeric::odesolve2`.

2.2. Scilab

2.2.1. Introduction

Scilab is a software package for numerical calculations, see `www.scilab.org`. This program is optionally included with MuPAD 2.5, together with a MuPAD module providing access to the Scilab functionality. If installed, this link is used by a number of functions in the `numeric` library when appropriate, to provide the user with faster calculations than programs written in the MuPAD language can achieve. Sparse matrices (see `?sparsematrix`) are passed to Scilab in sparse form. Not all Scilab functions make use of sparsity, but those that do may show significant speed-ups compared to dense matrices.

Functions in the `numeric` library using Scilab do so in the following way:

- The user can ask for the use of the Scilab routine(s) with the option `HardwareFloats`, or she can request a MuPAD library calculation with the option `SoftwareFloats`.

- If none of these option is given, `HardwareFloats` is the default if `DIGITS < 16`. Otherwise, `SoftwareFloats` are used, since Scilab computes with hardware floating points which have an accuracy of about 16 significant decimal digits.
- If Scilab is not available or fails for some reason, the code automatically falls back to the `SoftwareFloats` code. There is no warning unless `HardwareFloats` was requested explicitly. This way, versions without Scilab do not come up with awkward errors or warnings.

The following routines in the `numeric` library support Scilab as written above:

- `linsolve`
- `matlinsolve`
- `leastSquares`
- `det`
- `expMatrix`
- `fMatrix`
- `factorCholesky`
- `factorLU`
- `factorQR`
- `inverse`
- `eigenvalues`
- `eigenvectors`
- `singularvalues`
- `singularvectors`
- `fft`
- `invfft`

2.2.2. Effects For The User

Scilab, in contrast to MuPAD, uses hardware floating point numbers. This makes calculations faster if we assume the algorithms to use the same number of elementary operations. There is, however, an overhead for conversion between the different representations and some of the routines have to do pre- and post-processing to achieve input and output semantics in line with the MuPAD library. As a rule of thumb, using hardware floats is faster for matrices of size 5×5 and larger.

Apart from the higher speed, calculations with hardware floats exhibit slightly different round-off errors, so results can be expected to be slightly different between hardware and software floats. More importantly, the range of hardware floats (smallest positive number to largest representable number) is considerably smaller than that of MuPAD software floats, which may cause errors for special examples. In a few special cases, the solution of a numerical problem is not unique, even if calculated with exact numbers. In these cases, the return value of a `numeric` routine may be different with hardware and software floats, yet both results may be assumed to be correct:

```
>> A := matrix([[1,2],[2,4]]):
   DIGITS := 3:
   numeric::singularvectors(A, HardwareFloats)[1]

                          +-              -+
                          |  0.447,  0.894  |
                          |                 |
```

```
                          |  0.894, -0.447  |
                          +-               -+
```

```
>> numeric::singularvectors(A, SoftwareFloats)[1]

                          +-               -+
                          |  0.447, -0.894  |
                          |                 |
                          |  0.894,  0.447  |
                          +-               -+
```

2.2.3. Direct Use of Scilab Routines

For users who are familiar with Scilab, the MuPAD-Scilab installation includes `scilab::func`, which allows sending of arbitrary Scilab commands. In the following example, we solve the ordinary differential equation $\frac{\mathrm{d}y}{\mathrm{d}t} = f(y,t) = y^2 - y\ \sin(t) + \cos(t)$ with the initial condition $y(0) = 0$ numerically in the interval $[0, \pi]$:

```
>> odesolveHF := scilab::func(
        [], ["y"],
        ["deff('[ydot] = f(t, y)', 'ydot = y^2 - y*sin(t) + cos(t)')",
        "y0 = 0; t0 = 0; t = 0:0.1:%pi",
        "y = ode(y0, t0, t, f)"]):
   odesolveHF()

                     array(1..1, 1..32,
                       (1, 1) = 0.0,
                       (1, 2) = 0.09983342632,
                       (1, 3) = 0.1986693619,
                      ...
                       (1, 31) = 0.1411205029,
                       (1, 32) = 0.04158117166
                     )
```

2.3. Floating Point Intervals: DOM_INTERVAL

Introduction

MuPAD 2.5 introduces a new kernel data type: `DOM_INTERVAL` represents (rectangular) intervals in the complex plane, using numerical (floating point) borders. These intervals enable MuPAD to have fast numerical computations with absolutely reliable results: If, for example, the result of a computation with floating point intervals does not contain 0, you can rest assured that the exact calculation, if it can be performed, can never yield 0.

As is usual in MuPAD, these intervals are first class objects which you can manipulate in an intuitive way—as long as you regard mathematical notation as intuitive. For common operations, you do not need to learn any new functions, apart from those for generating floating point intervals in the first place. In the examples below, you find two of those functions: The operator `...` is mainly used to denote the interval between two numerical endpoints, while `hull` computes the "hull" of an expression, that is, a rectangle enclosing all values in the complex plane which the expression could take on.

MuPAD 2.5 can perform interval arithmetics with the following functions:

- The basic arithmetical operations: `+`, `-`, `*`, `/`, `^`, `sqrt`.
- The trigonometric functions and their inverses: `sin`, `cos`, `tan`, `sec`, `csc`, `cot`, `arcsin`, `arccos`, `arctan`, `arccsc`, `arccot`.
- The exponential function and the logarithm.
- The hyperbolic functions and their inverses: `sinh`, `cosh`, `tanh`, `sech`, `csch`, `coth`, `arcsinh`, `arccosh`, `arctanh`, `arccsch`, `arccoth`.
- The functions `Re`, `Im` (real and imaginary part), `abs`, `sign` and `arg` (the 'argument' = polar angle of a complex number).
- For real intervals, `gamma` and `beta`.
- `ceil`, `floor`, `trunc`, `round`.

Some Examples

On the interval from 0 to 1, the sine takes on values from 0 to about 0.84; on the interval from 0 to 3, the values range from 0 to 1:

```
>> sin(0...1), sin(0...3)

                 0.0 ... 0.8414709849, 0.0 ... 1.0
```

The square root of a positive number is always positive:[1]

```
>> sqrt(0...infinity)

                          0.0 ... RD_INF
```

The square root of any complex number has a nonnegative real part. We use the function `hull` to obtain the hull of the expression `sqrt(z)` for arbitrary `z`:

```
>> hull(sqrt(z))

             (0.0 ... RD_INF) + (RD_NINF ... RD_INF) I
```

As indicated above, one of the major differences between interval arithmetics and the usual floating point evaluation is that the latter does not indicate the presence of rounding problems:

[1] `RD_INF` is almost identical to `infinity`; the differences are explained in the documentation of `hull`.

```
>> DIGITS:= 10:
   sin(1e20)
```

```
                    0.1455000338
```

In this result, not even the sign is correct. The cause of the problem is that 10^{20} cannot be represented accurately in binary representation with the default precision of 10 decimal places. Increasing `DIGITS` yields a much more precise result:

```
>> DIGITS:= 20:
   sin(1e20)
```

```
                -0.64525128448909673708
```

Using interval arithmetics does not eliminate the rounding problems, but, because the intervals are always rounded "outwards", the result is guaranteed to contain the correct value. In our example, calculating with 10 decimal digits, we can see immediately that the result is useless:

```
>> DIGITS:=10:
   sin(hull(1e20))
```

```
                    -1.0 ... 1.0
```

Increasing `DIGITS` to 20 and recalculating, we get an interval of a width smaller than 10^{-8} and can be sure that the first eight digits after the decimal point are, in fact, correct:

```
>> DIGITS:=20:
   sin(hull(1e20))
```

```
     -0.64525128894383185026 ... -0.64525128182878537308
```

3. Combinatorics

3.1. Introduction

MuPAD 2.5 contains a new implementation of the `combinat` library. It consists of a subset of the algebraic combinatorics contribution package MuPAD-Combinat version 0.2. The full featured package, when installed, can be loaded with:

```
>> package("Combinat")
```

See `http://mupad-combinat.sourceforge.net` for details.

3.2. Structure

Most of the new `combinat` library is organized in separate sub-libraries with similar interfaces. For example, `combinat::cartesianProduct::count` is used to determine the number of elements in a Cartesian product, while `combinat::subsets::count` returns the number of subsets of a set. The following sub-libraries are provided:

1. `combinat::cartesianProduct` provides functions dealing with the Cartesian product of a finite number of sets or lists.

2. `combinat::compositions` deals with compositions of natural numbers, i.e., with lists of natural numbers with a fixed sum.

3. `combinat::dyckWords` deals with Dyck words. These can be interpreted as words of well balanced parentheses.

4. `combinat::generators` provides utilities for creating and manipulating generators, i.e., functions which, on each call, return the next element of a (possibly infinite) sequence of values.

5. `combinat::integerVectors` deals with vectors of nonnegative integers of fixed length and fixed sum.

6. `combinat::integerVectorsWeighted` is similar, but the sum is replaced by a weighted sum.

7. `combinat::partitions` provides functions dealing with partitions, i.e., decreasing lists of natural numbers with a fixed sum.

8. `combinat::permutations` works on permutations of lists.

9. `combinat::subwords` deals with subwords: Starting with a given list, delete any number of entries.

10. `combinat::tableaux` provides utilities for tableaux. These are useful for computations on symmetric functions and polynomials, among other things.

11. `combinat::words` deals with words, i.e., lists of fixed length consisting of elements of a specified alphabet.

The following methods are implemented in more than one library:

	count	generator	list	first	last	Next	previous
cartesianProduct	+	+	+	−	−	−	−
compositions	+	+	+	+	+	+	+
dyckWords	+	+	+	−	−	−	−
generators	+	−	−	−	−	−	−
integerVectors	+	+	+	+	+	+	+
integerVectors-Weighted	+	+	+	+	−	−	−
partitions	+	+	+	+	+	+	+
permutations	+	+	+	+	+	+	+
subwords	+	+	+	+	+	−	−
tableaux	+	+	+	+	+	−	−
words	+	+	+	−	−	−	−

Their meanings, in general, are as follows:

`count(size <, options>)`	number of structures of size `size`
`generator(<size>)`	generator for all the structures (of size `size`)
`list(<size>)`	list of all the structures (of size `size`)
`first(<size>)`	first structure (of size `size`)
`last(<size>)`	last structure (of size `size`)
`Next(current <, size>)`	next structure (of size `size`)
`previous(current <, size>)`	previous structure (of size `size`)

Many of the libraries provide further methods, such as `combinat::permutations::random`. See the corresponding documentation for details.

3.3. Examples

In this section, we pick a few sample applications at random. First, we export the `combinat` library to save us some typing:

```
>> export(combinat):
```

In our first examples, we use the abbreviated calls for list generation. We generate the Cartesian product of three lists, we compute all permutations of the numbers $1, 2, 3$, and we ask for all sub-words of the word `[a, b, c, d]`:

```
>> cartesianProduct([1, 2, 3], [a, b],[i, ii, iii])

 [[1, a, i], [1, a, ii], [1, a, iii], [1, b, i], [1, b, ii],

    [1, b, iii], [2, a, i], [2, a, ii], [2, a, iii], [2, b, i],

    [2, b, ii], [2, b, iii], [3, a, i], [3, a, ii],

    [3, a, iii], [3, b, i], [3, b, ii], [3, b, iii]]

>> permutations(3)

 [[1, 2, 3], [2, 1, 3], [1, 3, 2], [3, 1, 2], [2, 3, 1],

    [3, 2, 1]]

>> subwords([a, b, c, d])

 [[], [a], [b], [c], [d], [a, b], [a, c], [a, d], [b, c],

    [b, d], [c, d], [a, b, c], [a, b, d], [a, c, d], [b, c, d],

    [a, b, c, d]]
```

Dyck words, as stated above, can be interpreted as balanced sequences of parentheses:

```
>> map(dyckWords(4), dyckWords::toString)

 ["(((())))", "((()()))", "((())())", "((()))()", "(()(()))",

    "(()()())", "(()())()", "(())(())", "(())()()", "()((()))",

    "()(()())", "()(())()", "()()(())", "()()()()"]
```

4. Miscellaneous

4.1. OpenGL 3D-Viewer

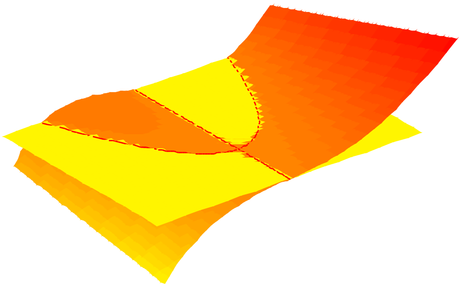
The Windows-version MuPAD Pro 2.5 contains a 3D-graphics tool based on OpenGL. This industry standard allows to display three-dimensional objects in high quality, taking advantage of special hardware if installed. You can invoke this tool from the general MuPAD graphics program VCam:

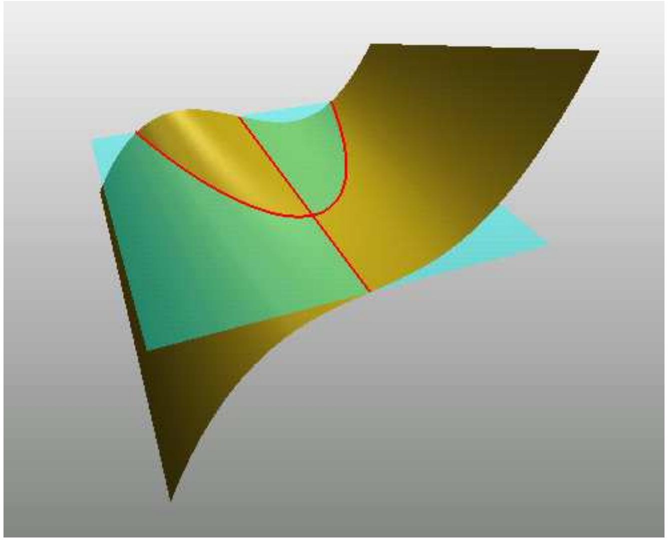
- When a three-dimensional graphic is selected (double-click) inside a MuPAD notebook, in the menu "Edit" select the entry "Open in 3D Viewer".
- For graphics opened in a separate window, the entry is located in the menu "File".

For documentation of the 3D-graphics tool we refer to the online documentation.

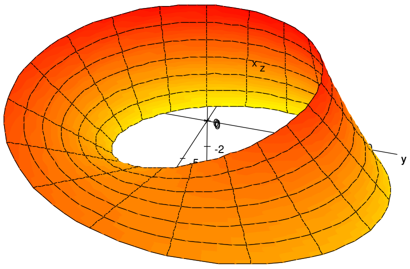
While the advantages of the new viewer are best seen in actual use, we provide a couple of screen shots for comparison:

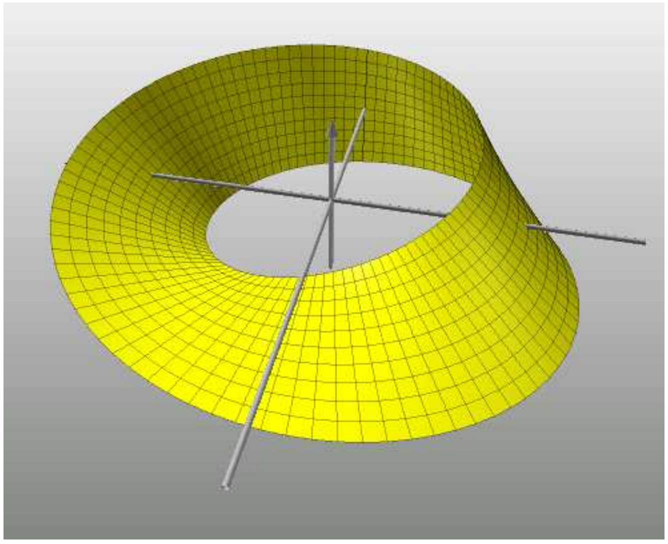
The classic example of a pitchfork bifurcation, $x^3 - x\,y = 0$:



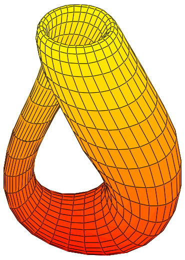


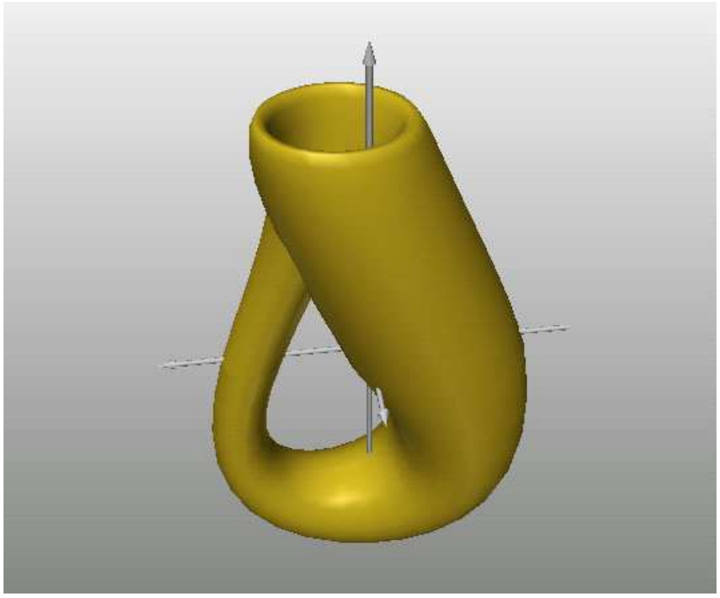
The Möbius strip is a famous example of a non-orientable manifold:



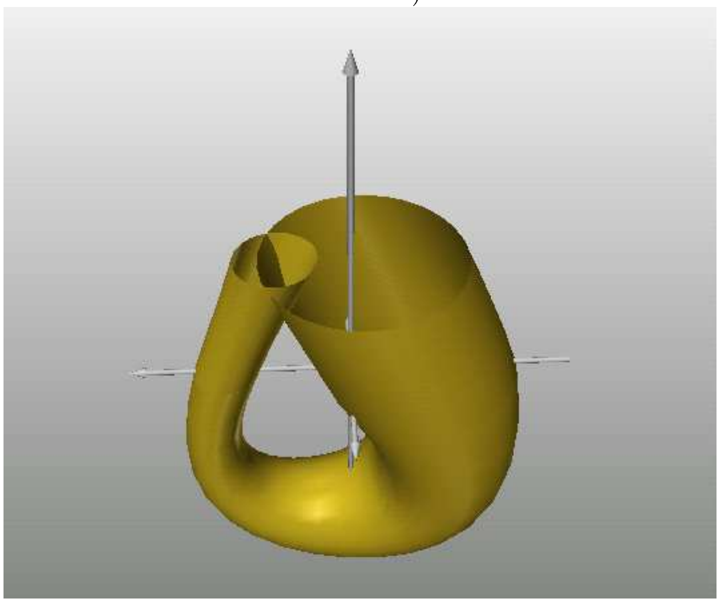


Its cousin, the Klein bottle, is not orientable and has no border:

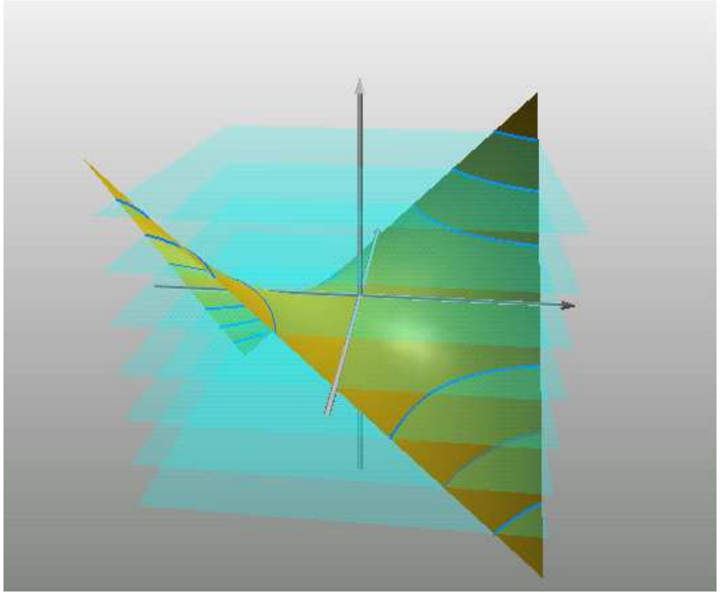




One of the new functions which make this tool ideal for interactive exploration is the "clipping plane", which you can use to peek inside objects. From the image above, we simply inserted a clipping plane and positioned it to see the place where the Klein bottle intersects itself (the only way to immerse a Klein bottle in three dimensions):



The new transparency option allows you to make visualizations which were simply not possible before:



4.2. Frames

4.2.1. Introduction

In MuPAD 2.5, the language has been extended with the concept of frames. Similar to namespaces in languages like C++ or Java, frames allow to temporarily alter the values of identifiers without 'polluting' the global state of MuPAD. This is a highly technical concept and not intended for the beginning user. For most of the users actually using frames, the primary application will be scratch pad calculations.

The realization of frames is similar to directories in a file system: They are ordered hierarchically, you can move up and down with the keyword `frame`, creating frames as you go along. Here, we will simply try to give you an idea of what frames can do. For a more thorough description, please see `?frame`.

4.2.2. An Introductory Example

We start by assigning the value 42 to the identifier `x`:

```
>> x := 42:
```

We change to frame `A`, which is automatically created along the way:

```
>> frame A

                         frame ::A
```

As you can see, each `frame` command returns the current frame, which is printed as above. The value of `x` is still visible:

```
>> x
```

```
                         42
```

Now, inside frame `::A`, we set `x` and `y` to new values:

```
>> x := 23:
   y := 1:
```

The following syntax is one way to go back to a parent frame (which, in this case, is the root frame where we started):

```
>> frame ..
```

```
                      frame ::
```

Our assignments did not change the value of `x` here, neither is `y` defined:

```
>> x, y
```

```
                        42, y
```

The newly set values are, however, still there, if we change back to the right frame:

```
>> frame A:
   x, y;
   frame ..:
```

```
                        23, 1
```

Incidentally, the frame `::A` is now the value of the identifier `A`, and the local values we set can be retrieved using the slot operator `::`:

```
>> A;
   A::x, A::y
```

```
                      frame ::A

                        23, 1
```

Frames can be deleted just like any other value:

```
>> delete A, x:
```

4.3. Programming Utilities

The following utility functions have been introduced in MuPAD 2.5:

- `assert` is used to check conditions during the run of a program which are guaranteed to hold under the assumptions the programmer made. Note that `assert` calls are simply skipped under normal conditions and take almost no time; their conditions are checked only if `testargs(TRUE)` has been called.

- `copyClosure` is a very technical method: It copies the "lexical closure" of a method. This closure mainly consists of the local variables of outer procedures having `option hold` set.

- `fname` returns the name of an opened file. It is especially useful with the new option `TempFile` of `fopen`.

- `Ax::indetElements` is a new axiom which permits the use of domain elements as indeterminates, for example in polynomials. This is important to allow polynomials over coefficient domains which contain all identifiers, such as `Dom::ExpressionField()`.

- `Type::Indeterminate` can be used with `testtype` to validate indeterminates.

- `Type::Set` is a type (for use with `testtype`) accepting a general form of sets.

- `prog::tcov` analyzes a test run and reports on the code coverage of the test.

- `FILEPATH` contains the path of the file currently read (if not interactive).

4.4. Mathematical Functions

Apart from the many functions introduced in the previous chapters, MuPAD 2.5 offers the following new ones:

- `card` computes the cardinality of a set.

- `contfrac` returns (truncated) continued fraction representations of real numbers and univariate expressions.

- `frandom` generates uniformly distributed floating point random numbers or generators thereof.

- `hypergeom` represents the hypergeometric function.

- `Dom::SparseMatrix` is a new matrix type which stores the data internally in a sparse representation. Compared to `Dom::Matrix`, arithmetic with this new matrix type is much more efficient for sparse matrices, i.e., matrices with a large number of zero entries. In analogy to `matrix`, `sparsematrix` was provided as a shorthand notation for `Dom::SparseMatrix()`. `Dom::SparseMatrix` has been designed as a drop-in replacement for `Dom::Matrix`, i.e., you should be able to simply replace matrices with sparse matrices if you wish.

- `groebner::stronglyIndependentSets` computes a strongly independent set of variables in a Gröbner Basis.

- `linalg::laplacian` computes the Laplacian of an expression.

Part II.

Changed Functionality

5. Analysis

5.1. Series

The datatypes `Series::Puiseux` and `Series::gseries`, which are returned by the functions `series` and `asympt`, have been revised for MuPAD 2.5. Their mathematical properties have been extended, so you can now perform arithmetics with series much more directly than in previous versions, and handling of branch cuts and the expansion point `complexInfinity` has been improved. The precision management has been stabilized.

We calculate a few series expansions of functions at branch cuts:

```
>> series(arcsin(x + 1), x = 0, 4)

                        1/2     1/2      2  1/2     1/2
 PI    1/2     1/2   x 2    (-x)      3 x  2    (-x)
 -- - 2    (-x)    + -------------- - ---------------- +
  2                        12               160

       3  1/2     1/2
    5 x  2    (-x)          9/2
    ----------------- + O(x   )
           896

>> series(ln(x), x = -1, 4)

                                        2          3
                                 (x + 1)    (x + 1)              4
  I PI signIm(x) - (x + 1) - -------- - -------- + O((x + 1) )
                                   2          3
```

Arithmetics on series expansions has been improved:

```
>> s := series(exp(x), x, 3)

                          2
                         x       3
                 1 + x + -- + O(x )
                          2

>> normal(s^PI) = series(exp(PI*x), x, 3)
```

```
            2   2                           2   2
           x  PI       3               x  PI       3
1 + x PI + ------ + O(x ) = 1 + x PI + ------ + O(x )
              2                           2
```

```
>> subs(s, x=y-1)
```

```
                     2
              (y - 1)            3
1 + (y - 1) + -------- + O((y - 1) )
                  2
```

A number of special functions accept series expansions as input:

```
>> s := series(x/(1 - x)^2, x)
```

```
       2      3      4      5      6      7
x + 2 x  + 3 x  + 4 x  + 5 x  + 6 x  + O(x )
```

```
>> exp(s), sin(s)
```

```
           2       3        4         5
        5 x    31 x    241 x    2261 x       6
1 + x + ---- + ----- + ------ + ------- + O(x ),
          2      6       24       120

               3                5       6
       2   17 x       4    181 x    13 x       7
x + 2 x  + ----- + 3 x  + ------ - ----- + O(x )
             6               120      4
```

```
>> s := series(arcsin(x + 1), x = 0, 4):
   sin(s)
```

```
           5
1 + x + O(x )
```

Expansions around `complexInfinity` are now possible:

```
>> series(arccot(x), x = complexInfinity)
```

```
1     1      1     /  1 \
- - ---- + ---- + O| -- |
x      3      5    |  7 |
    3 x    5 x     \ x  /
```

6. Numerics

6.1. Changes in the `numeric` library

As indicated in section 2.2, many functions of the `numeric` library were extended to make use of the external program scilab if installed. This integration not only includes conversions of data types and automated decisions on when to use these external routines, but also numerical pre-conditioning and other stabilizations as well as functionality adapting the interface design to that of MuPAD. In the course of writing these adaptations, a number of functions were given a more flexible interface, allowing the user to choose from a variety of input and output formats.

Especially, the functions dealing with linear algebra problems were given the new option `ReturnType` which can be used to choose from a variety of datatypes for the matrices returned: `matrix`, `sparsematrix`, or `DOM_ARRAY`. The following functions were extended this way:

- `numeric::eigenvectors`,
- `numeric::expMatrix`,
- `numeric::factorCholesky`,
- `numeric::factorLU`,
- `numeric::factorQR`,
- `numeric::fMatrix`,
- `numeric::inverse`,
- `numeric::matlinsolve`, and
- `numeric::singularvectors`.

Another minor change resulting from the inclusion of the scilab link is that the return values of `eigenvalues`, `eigenvectors`, `singularvalues`, and `singularvectors` are now returned in a different ordering.

The function `numeric::realroots` now uses kernel intervals instead of the library domain `Dom::Interval`. The result for the user, apart from a higher speed, is that more special functions are supported (see section 2.3 for a list) and that the functions for which zeroes are searched may contain complex expressions.

Cubic and bi-cubic spline functions, computed with `numeric::cubicSpline` or the new function `numeric::cubicSpline2d`, respectively, are now able to compute derivatives.

Other, minor, changes can be found in the documentation of the corresponding functions. See also appendix B.

Part III.

All New, Changed, and Renamed Functions

A. New Functions, Keywords, and Datatypes

Name	Description
`==>`	Logical implication
`<=>`	Logical equivalence
`assert`	Assertions for debugging
`card`	Cardinality of a set
`copyClosure`	Copies the lexical closure of a procedure.
`DOM_INTERVAL`	Rectangular complex intervals with floating point borders.
`FILEPATH`	Location of a file
`fname`	Get a file's name
`frame`	Switch the current frame
`frandom`	Generate random floating point numbers
`hull`	Convert to a floating point interval
`hypergeom`	The hypergeometric function
`interval`	Convert subexpressions to intervals
`PACKAGEPATH`	Search path for packages
`save`	Save the state of an identifier.
`xor`	Logical exclusive-or

Name	Description
`Ax::indetElements`	The axiom that indeterminates may be elements
`Dom::SparseMatrix`	Sparse matrices
`Dom::FloatIV`	Façade domain for `DOM_INTERVAL`
`Type::Indeterminate`	A type representing indeterminates
`Type::Set`	`Type::Set` comprises all expressions in which the operators are set-theoretic operations.
`groebner::stronglyIndependentSets`	Strongly independent set of variables
`linalg::laplacian`	The Laplacian
`numeric::cubicSpline2d`	Interpolation by 2-dimensional bi-cubic splines
`numeric::leastSquares`	Least squares solution of linear equations
`numeric::odesolveGeometric`	Numerical solution of an ordinary differential equation on a homogeneous manifold
`numeric::ode2vectorfield`	Convert a system of ordinary differential equations of arbitrary order to a vectorfield representation suitable for the numerical ODE solver `numeric::odesolve2`.
`plot::bars`	Create a two-dimensional plot of data as a bar chart (histogram plot)
`plot::boxplot`	Create a box plot of statistical data
`plot::piechart2d`	Create a two-dimensional plot of data as a piechart
`plot::piechart3d`	Create a three-dimensional plot of data as a piechart

Name	Description
`plot::matrixplot`	Create a three-dimensional plot of matrix data as a surface
`plot::surface`	Create a three-dimensional plot of matrix data as a surface. This is just an alias for `plot::matrixplot`.
`prog::tcov`	Report on test coverage (passed program lines)
`stats::betaCDF`	The cumulative distribution function of the β-distribution
`stats::betaPDF`	The probability density function of the β-distribution
`stats::betaQuantile`	The quantile function of the β-distribution
`stats::betaRandom`	Generate a random number generator for β-deviates
`stats::binomialCDF`	The (discrete) cumulative distribution function of the binomial distribution
`stats::binomialPF`	The probability function of the binomial distribution
`stats::binomialQuantile`	The quantile function of the binomial distribution
`stats::binomialRandom`	Generate a random number generator for binomial deviates
`stats::cauchyCDF`	The cumulative distribution function of the Cauchy distribution
`stats::cauchyPDF`	The probability density function of the Cauchy distribution
`stats::cauchyQuantile`	The quantile function of the Cauchy distribution

Name	Description
`stats::cauchyRandom`	Generate a random number generator for Cauchy deviates
`stats::chisquareCDF`	The cumulative distribution function of the χ^2 distribution
`stats::chisquarePDF`	The probability density function of the χ^2 distribution
`stats::chisquareQuantile`	The quantile function of the χ^2 distribution
`stats::chisquareRandom`	Generate a random number generator for χ^2 deviates
`stats::covariance`	The covariance of data samples
`stats::csGOFT`	The classical χ^2 goodness-of-fit test
`stats::empiricalCDF`	The empirical (discrete) cumulative distribution function of a finite data sample
`stats::equiprobableCells`	Divide the real line into equiprobable intervals
`stats::erlangCDF`	The cumulative distribution function of the Erlang distribution
`stats::erlangPDF`	The probability density function of the Erlang distribution
`stats::erlangQuantile`	The quantile function of the Erlang distribution
`stats::erlangRandom`	Generate a random number generator for Erlang deviates
`stats::exponentialCDF`	The cumulative distribution function of the exponential distribution
`stats::exponentialPDF`	The probability density function of the exponential distribution

Name	Description
`stats::exponentialQuantile`	The quantile function of the exponential distribution
`stats::exponentialRandom`	Generate a random number generator for exponential deviates
`stats::fCDF`	The cumulative distribution function of Fisher's f-distribution (fratio distribution)
`stats::fPDF`	The probability density function of Fisher's f-distribution (fratio distribution)
`stats::fQuantile`	The quantile function of Fisher's f-distribution (fratio distribution)
`stats::fRandom`	Generate a random number generator for Fisher's f-deviates (fratio deviates)
`stats::gammaCDF`	The cumulative distribution function of the γ distribution
`stats::gammaPDF`	The probability density function of the γ distribution
`stats::gammaQuantile`	The quantile function of the γ distribution
`stats::gammaRandom`	Generate a random number generator for γ deviates
`stats::geometricCDF`	The (discrete) cumulative distribution function of the geometric distribution
`stats::geometricPF`	The probability function of the geometric distribution
`stats::geometricQuantile`	The quantile function of the geometric distribution
`stats::geometricRandom`	Generate a random number generator for geometric deviates
`stats::hypergeometricCDF`	The (discrete) cumulative probability function of the hypergeometric distribution

Name	Description
`stats::hypergeometricPF`	The probability function of the hypergeometric distribution
`stats::hypergeometricQuantile`	The quantile function of the hypergeometric distribution
`stats::hypergeometricRandom`	Generate a random number generator for hypergeometric deviates
`stats::ksGOFT`	The Kolmogorov-Smirnov goodness-of-fit test
`stats::logisticCDF`	The cumulative distribution function of the logistic distribution
`stats::logisticPDF`	The probability density function of the logistic distribution
`stats::logisticQuantile`	The quantile function of the logistic distribution
`stats::logisticRandom`	Generate a random number generator for logistic deviates
`stats::meandev`	The mean deviation of a data sample
`stats::moment`	The k-th moment of a data sample
`stats::normalPDF`	The probability density function of the normal distribution
`stats::normalQuantile`	The quantile function of the normal distribution
`stats::normalRandom`	Generate a random number generator for normal deviates
`stats::poissonCDF`	The (discrete) cumulative distribution function of the Poisson distribution
`stats::poissonPF`	The probability function of the Poisson distribution

Name	Description
`stats::poissonQuantile`	The quantile function of the Poisson distribution
`stats::poissonRandom`	Generate a random number generator for Poisson deviates
`stats::swGOFT`	The Shapiro-Wilk goodness-of-fit test for normality
`stats::tPDF`	The probability density function of Student's t-distribution
`stats::tQuantile`	The quantile function of Student's t-distribution
`stats::tRandom`	Generate a random number generator for Student deviates (t-deviates)
`stats::uniformCDF`	The cumulative distribution function of the uniform distribution
`stats::uniformPDF`	The probability density function of the uniform distribution
`stats::uniformQuantile`	The quantile function of the uniform distribution
`stats::uniformRandom`	Generate a random number generator for uniformly continous deviates
`stats::weibullCDF`	The cumulative distribution function of the Weibull distribution
`stats::weibullPDF`	The probability density function of the Weibull distribution
`stats::weibullQuantile`	The quantile function of the Weibull distribution
`stats::weibullRandom`	Generate a random number generator for Weibull deviates

B. Changed Functions

The following table lists all MuPAD functions with changes since version 2.0, with the exception that functions in the `numeric` library are not listed if their only change was the introduction of the scilab link. See chapter 2.2 for these changes. Likewise, the table does not list the functions now accepting series expansions or kernel intervals as arguments.

The MuPAD command-line options on Unix-like systems have changed. See `?UNIX` for details.

Name	Description
`*`	Multiplications involving sets, equalities, inequalities, and/or comparisons have changed.
`+`	Additions involving sets, equalities, inequalities, and/or comparisons have changed.
`anames`	The new option `Protected` was introduced.
`arg`	`arg` may now also be called with only one argument.
`asympt`	Now may return series expansions of type `Series::Puiseux`.
`beta`	The function `rewrite` can now express `beta` in terms of `gamma` or `fact`.
`binomial`	The functions `diff` and `series` were overloaded for binomial.
`bool`	`bool` is now also overloadable by function environments.

Name	Description
`case`	The slightly obscure (and undocumented) behavior of `next` in `case`-switches was removed.
`Ci`	Series expansions via `series` were implemented.
`complexInfinity`	Addition of a finite number to `complexInfinity` yields again `complexInfinity`.
`cot`	The `"diff"` attribute was changed for consistency with `tan`.
`D`	Partial derivatives are assumed to commute. `D` now uses `option remember`.
`debug`	Calling `debug()` returns the debug state of the kernel.
`diff`	Partial derivatives are assumed to commute.
`evalp`	The evaluation points may also be given as list.
`exp`	`exp(1)^x` is no longer simplified to `exp(x)` automatically, use `combine` instead.
`extsubsop`	Now works for kernel datatypes as expected.
`fopen`	New options `TempFile`, `Read`, and `Raw`. Arguments can be given in any order.
`gamma`	The system function `expand` now implements the reflection formula and the Gauß multiplication formula.
`igamma`	The normalization to first arguments was restricted.
`in`	`_in` now just returns the unevaluated function call; a new slot `"expand"` was introduced to provide the old functionality.

Name	Description
_index	Indexed access to matrices now evaluates the entry before returning it.
interpolate (replaces numeric::lagrange)	A simplified input syntax for 1-dimensional grids was introduced.
info	The output is more verbose.
lambertV	Series expansions via series were implemented.
lambertW	Series expansions via series were implemented.
limit	The option Real was introduced.
normal	Floating point numbers in the input are not longer replaced by auxiliary identifiers, but by rational approximants now. The new option List was introduced.
partfrac	partfrac is now overloadable.
piecewise	The methods "map", "op", and "discont" were changed, "branch", "splitBranch", and "numberOfBranches" were added.
plotOptions2d	The new option AxesInFront was introduced.
protect	The option names were changed to avoid confusion. Identifiers of the form `#...` are implicitly protected and cannot be unprotected.
psi	Special formulas for rational numbers with denominators 1, 2, 3, 4, or 6, respectively, were implemented. The system function expand now implements the reflection formula and the Gauß multiplication formula.

Name	Description
`rewrite`	The new targets `D` and `andor` were introduced. Calls to `max` and `min` can be rewritten to `piecewise`. The code for the target `ln` was rewritten and enhanced.
`series`	The implementation was largely rewritten, see the special chapter 5.1.
`signIm`	The reflection `signIm(-x) = -signIm(x)` was implemented.
`solve`	The new option `DontRewriteBySystem` was introduced.
`tanh`	The `"diff"` attribute was changed for consistency with `tan`.
`taylor`	Taylor expansions around `complexInfinity` are now possible.
`rtime`	The measurement precision was increased.
`unprotect`	The option names were changed to avoid confusion.
`userinfo`	The option `NoNL` was added.
`Cat::BaseCategory::testtype`	The semantics was corrected.
`Dom::Float`	The method `"pivotSize"` was implemented.
`Dom::Matrix`	The method `"_index"` was improved and accepts symbolic indices.
`Dom::MatrixGroup`	The method `"_index"` was improved and accepts symbolic indices.
`Dom::Multiset`	Overloading of the new function `card` was added.
`Dom::Product`	The slot `"card"` was renamed to `"dimen"`.

Name	Description
`Dom::SquareMatrix`	The method `"_index"` was improved and accepts symbolic indices.
`groebner::gbasis`	The first argument may be a set. The options `Order` and `Monic` were introduced.
`linalg::charmat`	For an input of type `Dom::SparseMatrix`, the return value is of this type, too.
`linalg::eigenvectors`	The ordering of the eigenvalues in numerical approximations has changed. For numerical approximations, fast hardware floating point arithmetic is available.
`linalg::factorQR`	For an input of type `Dom::SparseMatrix`, the return value is of this type, too.
`linalg::hermiteForm`	With the option `All`, the user may now request the transformation matrix as well.
`linalg::matlinsolve`	For an input of type `Dom::SparseMatrix`, the return value is of this type, too.
`linalg::minpoly`	For matrices over `Dom::Float`, `linalg::minpoly` can now return `FAIL` in certain situations.
`linalg::ogCoordTab`	New information is contained in the table and can be accessed with the new options `Transformation`, `InverseTransformation`, `Ranges`, and `Dimension`.
`misc::maprec`	A new option `NoOperators` was added.
`numeric::cubicSpline`	The input syntax was extended. The spline function can now return its derivatives.
`numeric::eigenvalues`	The ordering of the return values has changed.
`numeric::eigenvectors`	The ordering of the return values has changed. Furthermore, the new option `ReturnType` was introduced.

Name	Description
`numeric::expMatrix`	The return type depends on the type of the input and on the option `ReturnType`, if specified.
`numeric::factorCholesky`	The new option `ReturnType` was introduced.
`numeric::factorLU`	The new option `ReturnType` was introduced.
`numeric::factorQR`	The new option `ReturnType` was introduced.
`numeric::fMatrix`	The new option `ReturnType` was introduced.
`numeric::inverse`	The new option `ReturnType` was introduced.
`numeric::linsolve`	The new option `NoWarning` was introduced.
`numeric::matlinsolve`	The new options `ReturnType` and `NoWarning` were introduced. A trivial kernel of A is now returned as the integer 0.
`numeric::odesolve`	The vectorfield procedure may now be passed as the first or as the second argument.
`numeric::realroots`	Internally, `numeric::realroots` now uses kernel intervals. This implies that the function to be solved may now contain complex expressions.
`numeric::singularvalues`	The ordering of the return values has changed.
`numeric::singularvectors`	The ordering of the return values has changed. Furthermore, the new option `ReturnType` was introduced.
`numeric::solve`	For linear systems of equations, `numeric::linsolve` is called.

Name	Description
`numlib::contfrac`	The implementation was rewritten. The screen output has changed. Rational numbers are approximated too, if a precision argument is given. New methods `"rationalInterval"`, `"convert"`, `"nthcoeff"`, and `"rational"` were implemented. The methods `"approx"` and `"unapprox"` are no longer available.
`numlib::mroots`	Instead of `FAIL`, an empty list is returned in case of no solution.
`plot::data`	This function was redesigned. Some formats such as *`Column`*, *`Piechart2d`* etc. disappeared; the corresponding functionality is now implemented by new routines `plot::bars`, `plot::piechart2d` etc. The input syntax of `plot::data` was changed an extended.
`plot::Function2d`	This function now accepts `piecewise` expressions and polynomials.
`plot::implicit`	The error handling was improved and the default of the maximum step size was increased.
`plot::inequality`	The input syntax was changed.
`plot::ode`	The vectorfield procedure may now be specified as the first or as the second argument.
`plot::Pointlist`	The option `Vertical` in 3 dimensions now draws lines parallel to the z-axis.
`plot::Scene`	The new scene option `AxesInFront` was introduced for 2-dimensional scenes.
`Pref::callOnExit`	The functions registered are now called at a `reset` call, too.
`prog::check`	The option `Save` was introduced.

Name	Description
`Series::Puiseux`	The internal representation was changed, functionality was improved and extended.
`Series::gseries`	The internal representation was changed, functionality was improved and extended.
`stats::chisquareCDF` (replaces `stats::ChiSquare`)	This function returns a procedure.
`stats::correlation` (replaces `stats::BPCorr`, `stats::FCorr`)	Unifies the functionality of `stats::BPCorr` and `stats::FCorr` with extended input syntax.
`stats::empiricalQuantile` (replaces `stats::a_quantil`)	This function returns a procedure.
`stats::normalCDF` (replaces `stats::normal`)	This function returns a procedure.
`stats::sample`	The new method `"size"` was implemented.
`stats::stdev`	The new option `Population` was introduced. The default used to be the functionality referred to by `Population`. Now, `Sample` is the default.
`stats::tCDF` (replaces `stats::Tdist`)	This function returns a procedure.
`stats::tTest` (replaces `stats::meanTest`)	For consistency and flexibility, the calling syntax was changed as well as the return value.
`stats::variance`	The new option `Population` was introduced. The default used to be the functionality referred to by `Population`. Now, `Sample` is the default.
`Type::Arithmetical`	By default, elements of domains defined on the library level are not considered arithmetical any more.

Name	Description
`Type::Series`	Specifying a direction is now possible. Optional arguments are also allowed for `Puiseux` and `Laurent`.

C. Renamed Functions

The following functions have been renamed since MuPAD 2.0, mostly for consistency reasons:

Old Name	New Name	Comments
`numeric::lagrange`	`interpolate`	A simplified input syntax was added.
`plot::data`	`plot::bars`	The formats *Beam* and *Column* of `plot::data` are now implemented in `plot::bars`.
`plot::data`	`plot::piechart2d`	The format *Piechart2d* of `plot::data` is now implemented in `plot::piechart2d`.
`plot::data`	`plot::piechart3d`	The format *Piechart3d* of `plot::data` is now implemented in `plot::piechart3d`.

Old Name	New Name	Comments
`plot::data`	`plot::matrixplot`	The formats *`Plain`* and *`Surface`* of `plot::data` are now implemented in `plot::matrixplot` and `plot::surface`, respectively.
`stats::BPCorr`	`stats::correlation`	Unified with `stats::FCorr`
`stats::FCorr`	`stats::correlation`	Unified with `stats::BPCorr`
`stats::a_quantil`	`stats::empiricalQuantile`	The calling syntax was changed.
`stats::geometric`	`stats::geometricMean`	
`stats::harmonic`	`stats::harmonicMean`	
`stats::normal`	`stats::normalCDF`	The calling syntax was changed.
`stats::quadratic`	`stats::quadraticMean`	
`stats::Tdist`	`stats::tCDF`	The calling syntax was changed.
`stats::meanTest`	`stats::tTest`	The calling syntax and the return type were changed.