Random Notes about SYMSCI Toolbox

Symbolic Math Toolbox for Scilab

21st March 2017

These notes refer to SYMSCI Toolbox 0.4 under SCILAB 6.0.0

Contents

1 Naming and Overloading of Functions

In general, SYMSCI adopts the lower *and* upper camel case naming convention to indicate, whether a function returns a *symbolic* object, a *native* SCILAB data type or nothing.

upper camel case function returns a *symbolic* object (e.g. `Sym`, `Add`, `GetArgs`, ...)

lower camel case function returns a *native* Scilab data type or nothing (e.g. `isSym`, `typeClass`, `toString`, ...)

Only common SCILAB operators (e.g `+`, `-`, `*`, `/`, ...) are overloaded by default. For common functions (e.g. `inv`, `det`, `sin`, `cos`, ...) it is up to the user of the toolbox to do that, if he really needs it.

2 Wormhole aka Symbolic Object

The connection of native SCILAB with the inner guts[1] of SYMSCI is based on the C++ API of SCILAB 6.0, which allowed the implementation of a new *symbolic* data type.

The predicate function `isSym(obj)` may be used[2], to check whether an `obj` is symbolic (`%t`) or not (`%f`). Every symbolic object is either a *basic* object (cf. `isScalar`) or a *container* (cf. `isSet`, `isVector`, `isMatrix`) of basic objects. Furthermore, every basic object is classified further (cf. `typeID`, `typeClass`) as `Symbol`, `Integer`, `Rational`, `Add`, `Mul`, `Div`, ...

3 (Not) Everything is a Matrix, ...

SCILAB's paradigma for calculations, that „*everything is a matrix*“, is honoured, such that the following statement holds: „*Every scalar is a matrix, but not every matrix is a scalar!*“.

Besides matrices (cf. `Mat`), there are – in the context of this toolbox – two other symbolic data types, which are *not* matrices, but foremost suited to act as *containers* for *symbolic scalars* (cf. `Sym`). Thus, calculation operations like `+`, `-`, `*`, `/`, ... are *not* defined for *symbolic vectors* and *sets* (cf. `Vec`, `Set`).

3.1 ... but Vectors ...

Symbolic vectors are one-dimensional arrays, whose elements are symbolic scalars.

`[a,b]` Horizontal *catenation* of vectors and scalars yields a vector, if at least one operand is a vector and the other is a scalar.

[1] SYMENGINE https://github.com/symengine/symengine
[2] or alternatively `typeof(obj) == „Symbolic"`

`a(i)` If the index `i` is an integer scalar, then the *extraction* `a(i)` yields a scalar. If the index `i` is an integer vector, then it yields a corresponding vector.

`a(i)=b` *Insertion* works, as one might expect, for an integer scalar or vector index `i`. It is possible to delete elements by using `b=[]`.

`a==b` Comparison for *equality* returns a SCILAB boolean matrix as expected.

`a~=b` Comparison for *non-equality* returns a SCILAB boolean matrix as expected.

3.2 and Sets are different!

Symbolic sets are collections of unique symbolic scalar elements.

`A(i)` If the index `i` is an integer scalar, then the *extraction* `a(i)` yields a scalar. If the index `i` is an integer vector, then it yields a corresponding set.

`A(B)` If the index is a set `B`, then the *extraction* `A(B)` yields a set whose elements are members of `A` *and* `B`. A scalar `B` will be interpreted as a set with just one element.

`A(B)=C` The *insertion* operation works for sets in a *different* way. It is a two-step operation: first all elements of set `B` are removed from `A`, and then all elements of set `C` are inserted into `A`. A scalar `B` or `C` will be interpreted as a set with just one element.

`A==B` If `A` and `B` are sets, then the comparision for *equality* yields a SCILAB boolean scalar (`%t` or `%f`). If either `A` or `B` is scalar, then it yields a SCILAB boolean matrix.

`A~=B` If `A` and `B` are sets, then the comparision for *non-equality* yields a SCILAB boolean scalar (`%t` or `%f`). If either `A` or `B` is scalar, then it yields a SCILAB boolean matrix.

4 Working with Symbolic Objects

Due to the brand new C++ API of SCILAB 6.0, there are *no* considerations about memory management to be taken into account, when working with symbolic objects. Usually, symbolic objects are created by using the functions `Syms`, `Sym`, `Mat`, `Vec`, or `Set`, which provide built-in parsing of string representations and conversion from SCILAB `double` and `integer` matrices.

5 Arbitrary Precision Floating Point

Apart from the truly symbolic capabilities, one may carry out *arbitrary precision* floating point calculations as well.

Every exact symbolic number may be converted to floating point representation with abitrary precision of your choice.

Example How to calculate the value of $\frac{\pi}{\sqrt{2}}$ with 1.000.000 decimal digits of precision? Well, this is really easy and rather fast ($<$ 5s on a really ordinary PC) using `Evalf` as follows:

```
x = Sym("pi/sqrt(2)")
tic; X = Evalf(x, 3321929); toc
```

6 Code Generation

It is possible to generate SCILAB code, or even ready-to-run SCILAB functions from symbolic expressions. Please refer to the help pages of `toSciCode` and `toSciFunction` for further explanations and examples.

7 Epilog

The SYMSCI toolbox is in a *work-in-progress* state, and thus incomplete and may contain numerous and severe bugs. Any feedback is highly appreciated and should be sent to info@kybdr.de.