Symbolic computation in Julia

John Lapeyre

JuliaCon 2018, August 11, 2018 Updated September 9, 2018

What is Computer algebra / Symbolic computation ?

A didactic dichotomy:

- AbstractAlgebra.jl. Algebra is what an "algebraist" does. Or number theorist, or computational number theorist, or...
  - **Not** symbolic.
  - Organization: language types representing algebraic structures.
  - Optimization: (discrete) numeric efficiency above flexibility.
  - Leader: Magma 1993 (proprietary). cited in many publications.
- Symata.jl. Computer algebra is a tool for scientists, engineers.
  - Symbolic: "algebra", calculus, etc. + numbers.
  - Organization: "expressions". Types less important.
  - Optimization: Flexibility, system integration, before efficiency.
  - Leader: Mathematica (proprietary). cited in many publications.

```
symata 1> rotheadargs(f_(args__)) :=
           (Last([args])(f, Splat(Most([args]))));
symata 2> rotheadargs(a + b + c + g(x))
Out(2) = g(x)(Plus,a,b,c) # Nonsense ?
```

- Language or library ? C, C++ Lisp, Python, Julia

Comparison / Benchmarks

Polynomial benchmarks – Fateman-like test

$$f = x + y + z + 1$$
$$p = f^{20}$$
$$q = p * (p + 1)$$

- Polynomial problem one $n = 20$
  - 0.05s TaylorSeries.jl (with 128 bit integers)
  - 0.19 AbstractAlgebra.jl
  - 0.12–0.28s Mathematica
  - 1.25s TaylorSeries.jl (with arbitrary precision integers)
  - 1.50 Pari
  - 3.80 SymEngine.jl
  - 7.70 Mathematica
  - 198 MultivariatePolynomials.jl
  - 490 SymPy (SymPy, SymPy.jl, . . .)
  - 3347 Maxima (sbcl)

```
symata 1> FullForm(1 + 3x + 4x^2)
Out(1) = Plus(1,Times(3,x),Times(4,Power(x,2)))
```

(partial) History of symbolic computation systems

- Schoonschip *1963 assembly*. Veltman. REDUCE *1965–present lisp-algolish*. A. Hearn.
- Macsyma *1968–1992 Lisp*. MIT, Symbolics, DOE. Complicated history. Changing hardware; OS; personal and organizational goals.
- Maxima *1982–1998–present Common Lisps*. OSS fork of Macsyma. Schelter. Now Macsyma people: Fateman, Macrakis, and others.
- Maple *1980-1988–present C core*. Expressions + Pascal-like language. "An interpreted, untyped, procedural language with lexical scoping and first-class procedures."
- SMP: Symbolic Manipulation Program *1979–1981–1988 C core*. Chris. A. Cole and S. Wolfram. Pattern matching and expressions. "version 0 of Mathematica". Very popular in CAS research by at least 1984.

(partial) History of Symbolic computation systems

- Mathematica *1988 C core*. Uniform design. *Everything* is an expression. Above all a product. Organizational focus. Extremely assertive marketing, media strategy, corporate relations.

- MuPAD *1997 C ?*. Somewhat typed, Algol-like. Subsumed by MATLAB.

- SageMath *2005 Everything*. Python interface.

- SymPy *2006-present Python*. Library + Python interfaces. Expressions and OO classes.

- Mathics *????-present Python*. Implementation of Mathematica. Backend is SymPy. Expressions. Correct, but slow. Angus Griffith.

- Symata.jl *2015–present Julia*. SymPy as library. Expression-based. Expression-based language. Preceding work: MockMma (Fateman). Mixima. Extensions to Maxima.

Why is there no competitive OSS computer algebra ?

- Competitive means,... viable alternative, has non-negligible market share,
- None. Neither for abstract algebra, nor for symbolic manipulation.
- W. Stein "In 2005 Magma was the only proprietary software on my machines". Same holds for many scientists with Mathematica.
- Reasons need not be the same. May not be general. GAP, Pari, etc. very narrow. Want CA in Julia tomorrow ? Support Julia for CRUD today.
- Development "model" or modes. Academy only cares about publications ? Mediocre paper better than great software. Much smaller use base ? Inherently more difficult minimum viable product ?

Why is there no competitive OSS computer algebra ?

- SageMath. W Stein. Combine all the best math programs into one competitor to Mma, Maple, Matlab, Magma. Very ambitious. What is the design? No funding. 2016 started company.
- 2016 OSCAR "massive transregional grant" for computational mathematics. Includes a big project to unify GAP, Singular, Polymake, ANTIC (Flint, Pari ?) using ... Julia. W. B. Hart and others promoting, using, Julia. Objective (of one part) is to create competitor to Magma.
- Symata. Want functionality of Mathematica, Maple, Maxima. Don't want "this pudding has no theme". Mma offers great starting point. At least a bit more. A layer in Julia offering control over evaluation sequence: easier to get efficiency.

Julia algebra / symbolic packages

- Number theory and abstract algebra ("algebra" to mathematicians)
  - AbstractAlgebra.jl Fastest!, Nemo.jl
  - Hecke.jl (Algebraic number theory), Singular.jl
  - AlgebraicNumbers.jl Exact arithmetic with algebraic numbers.
  - SemialgebraicSets.jl, SymmetricTensors.jl
- Polynomials, etc.
  - Polynomials.jl, MultivariatePolynomials.jl
- General purpose
  - Symata.jl
  - SymPy.jl SymPy and mpmath, SymEngine.jl C++ core.
  - Reduce.jl Interface to Reduce.
  - Giac.jl

Why Now ? Why Julia ?

Greenspun's oft-appropriated 10th rule

Any sufficiently complicated symbolic language written in C (or even Java, golang,...), contains an ad-hoc, informally-specified, bug-ridden, slow implementation of half of Julia. And forget about the ecosystem.

Symata *repurposes* features of Julia. Cuts development time!

- Symbols, Expressions. Syntactic macros. see MacroTools.jl
- Parser. IO of Symata code. Line numbers.
- Memory management, High-performance data structures. Type system. Generic methods. C-like speed.
- REPL, color, completion, history, multiline editing, modes. Notebook, beautiful math.
- Simple language interfaces. Python, C, Fortran.
- Easy, efficient, access to **discrete** and floating point numerics.
- Dev community, mathematicians. Current technology. Forward-thinking. github(lab), CI, Discourse/Slack.

Other options: C, Lisp, Python,... Go ?

Symata

- Everything is an expression (including atoms). (Mathematica maintains this user-facing semantics, but adds heroic optimization.) No flow control statements. `For`, `If` are expression heads.
- Uniform design. Tightly integrated components, builtin functions.
- Expressions traverse the *evaluation sequence*, which transforms them.
- Pattern matching plays central role. Mma designed when AI meant rule-based systems. Best integrator: Rubi Rule-based Mathematics Symbolic Integration Rules. Popularity of MacroTools.jl.

Features

Pattern Matching

Matching is syntactic, ignorant of mathematics.
associative => Flat, commutative => Orderless

Elements

`name_type`, `name__type`, `name___type`, `Repeated(expr, n)`, |, `Except`, Default values.

```
countprimes = Count(_`PrimeQ`)
countprimes(Range(100)) --> 25
```

Consistent features

Levels, “iterators”

Related

- Rewrite.jl
- MacroTools.jl

Simplification function ExpandSinCos

Implement the angle-addition formulas.

```
SinRule = Sin(a_ + b__) :>
   Cos(Plus(b)) * Sin(a) + Cos(a) * Sin(Plus(b))
CosRule = Cos(a_ + b__) :>
   Cos(a) * Cos(Plus(b)) - Sin(a) * Sin(Plus(b))
ExpandSinCos(ex_) := ex .\\ [SinRule, CosRule]
```

Apply this function (or rule).

```
symata 1> ExpandSinCos(1 + Sin(x + y + w*z))

Out(1) = 1 + Cos(x)*(Cos(w*z)*Sin(y) + Cos(y)*Sin(w*z)) +
  (Cos(y)*Cos(w*z) - Sin(y)*Sin(w*z))*Sin(x)
```

Mathematica: fourth wall impenetrable.

Symata: lots of portals

Call Julia function from Symata

```
symata 1> J(time)()
Out(1) = 1.533295814802480e9
```

```
x1 = Range(10.0^3)
y1 = Range(10.0^3)

g(x_, y_) := Module([s=0],
   begin
     For(i=1, i<=Length(x), i += 1,
              s += x[i]^2 / y[i]^(-3)),
     s
   end)

applySum := Apply(Plus, x1^2 / y1^3)

juliaSum = J((x,y) -> sum(u -> u[1]^2 / u[2]^(3),
                           zip(x,y)));
```

Time, `g(x1, y1)` : 1, `applySum` : 1/2, `juliaSum(x1, y1)` : 1/20

Symata

Generic methods

```
juliaSum = J((x,y) -> sum(u -> u[1]^2 / u[2]^(3),
                           zip(x,y)));
symata 1> juliaSum([a + b, c + d], [u + v, y + z])
```

$$\frac{(a+b)^2}{(u+v)^3} + \frac{(c+d)^2}{(y+z)^3}$$

Translate/compile Symata to Julia function

```
expr = Collect(Integrate(x^2 * Exp(x) * Cos(x), x),
          Exp(x))
```

$$e^x \left(\frac{-\operatorname{Cos}(x)}{2} + \frac{x^2 \operatorname{Cos}(x)}{2} + \frac{\operatorname{Sin}(x)}{2} + \frac{x^2 \operatorname{Sin}(x)}{2} - x \operatorname{Sin}(x) \right)$$

```
cexpr = Compile([x], Evaluate(expr));
native_julia(x) = exp(x)*((-1/2)*cos(x) + (1/2)*x^2*cos(x)
  + (1/2)*sin(x) + (1/2)*x^2*sin(x) - x*sin(x))

> @btime ccexpr(2.0)  ==>  56 ns
> @btime native_julia(2.0) ==> 56 ns
> ccexpr(:y)
E^y*((-1/2)*Cos(y) + (1/2)*y^2*Cos(y) +
(1/2)*Sin(y) + (1/2)*y^2*Sin(y) - y*Sin(y))
```

Translate/compile Symata to Julia function

Wrap Symata expression in Julia function

```
symata 1> symzeta = SymataCall([x], Zeta(x));
symata 2> ExportJ(symzeta)
julia> symzeta(4)
(1//90)*:Pi^4
```

Compile Symata to Julia

```
symata 1> hypot = Compile([x, y], Sqrt(x^2 + y^2));
symata 2> hypot(3, 4)
Out(2) = 5
```

Why is the hypotenuse exact ?

```
symata 3> ToJuliaString(Sqrt(x^2 + y^2),
                                NoSymata => False)
Out(4) = "mpow(mplus(mpow(x, 2), mpow(y, 2)), 1//2)"
```

SymPy

- SymPy. mpmath moved to stand alone. both pure Python.
- Python slow, not suited for expression-based language.
- Not prematurely optimized. Great sucess. Relatively complete. Often fast enough.
- Essentially a library. For free: tested/optimized language; huge ecosystem; interactive UIs.
- Design. Symbolic expressions. Mainly standard Python (heavily) OO approach. Not suitable for people who want a super calculator.

When would you use (or not) Symata ?

- Mma. Often used because a student was introduced to it early.
- Super-calculator. Simple programming.
- Useful for prototyping an idea. (L Shifrin)
- Can be very slow, even Mma, after lots of years and dollars.
- *Julia* may be a substitute for special purpose "symbolic" or discrete problems. (Jesse [tiling], Jared [Dirac notation], and . . .)
- Mma has completeness in domains. Want lots of special functions, high precision, complex ? (E.g. inverse Laplace transform). Integrate exotic integrands over exotic domains.
- Symata most likely will want to break out of the expression/evaluation model.

Thank you!