Co-funded by
the European Union

GeomDAE.jl

High-performance library of geometric integrators for ODEs and DAEs in Julia

Michael Kraus[1,2,3] (michael.kraus@ipp.mpg.de)

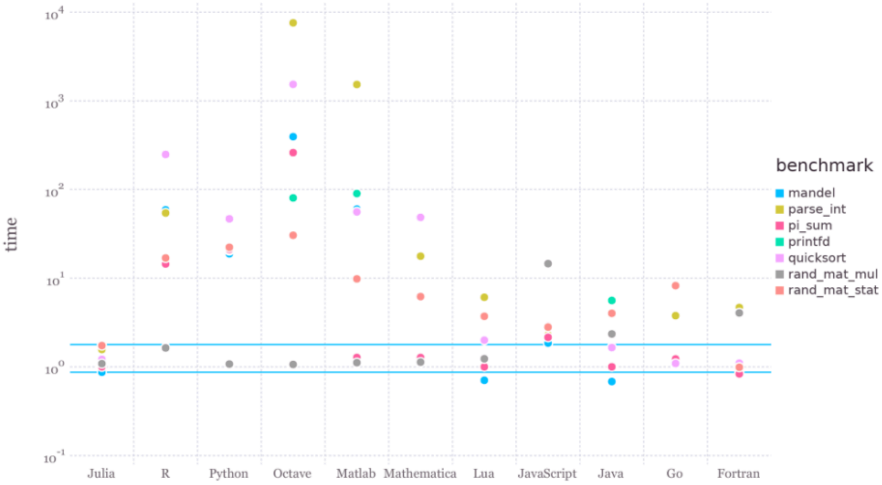
[1] Max-Planck-Institut für Plasmaphysik, Garching
[2] Technische Universität München, Zentrum Mathematik
[3] Waseda University, School of Science and Engineering

Julia

Julia? [http://julialang.org/]

- high-level, high-performance, just-in-time compiled, dynamic programming language for technical and scientific computing
- developed at MIT since 2009, public since 2012, growing exponentially
- multiple dispatch: functions are dynamically dispatched based on the type of their arguments (generalisation of single-dispatch polymorphism)
- programming paradigms: object-oriented and functional programming
- dynamic type system (with type indication): automatic generation of efficient, specialised code for different argument types
- performance approaching that of statically-compiled languages (C, Fortran)
- macros (Lisp-like) and rich meta-programming facilities
- call C/Fortran functions directly and Python functions via the PyCall package
- built-in support for multi-dimensional arrays
- built-in high-level abstractions for parallelism and distributed computing
- unicode: α, π, Δt, f_1, g_a, r^2, $\hat{a}$, $\neq$, $\leq$, $\geq$, ...
- rich ecosystem: BLAS/LAPACK, MPI, DA, HDF5, MUMPS, PETSc, FEM, DE, AD, GPU, OpenCL, SymPy, splines, optimisation, plotting, jupyter, ...

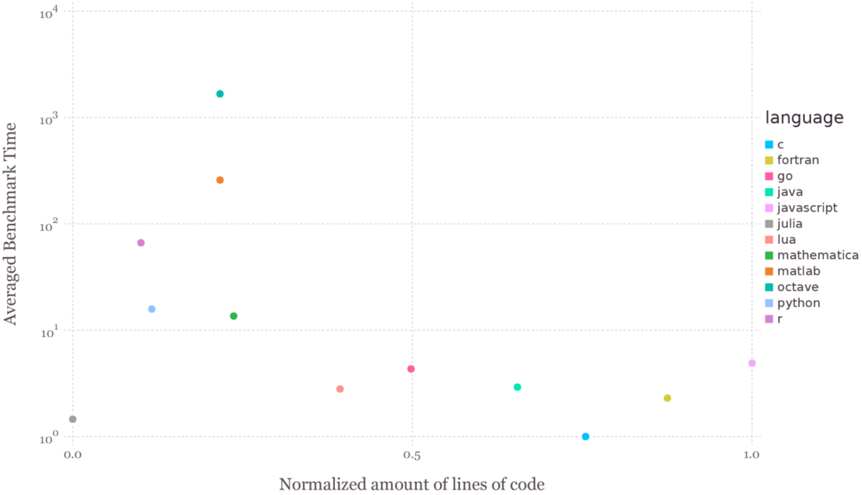
Benchmarks



Benchmark times relative to C (smaller is better, C performance = 1)

[http://www.juliabloggers.com/speed-expectations-for-julia/]

Benchmark: Runtime vs. Lines of Code



[http://www.juliabloggers.com/speed-expectations-for-julia/]

GeomDAE.jl

GeomDAE.jl

- high-performance library of geometric integrators for ODEs and DAEs in Julia
- main aim: implementation and verification of novel geometric integrators
- minimal overhead, maximum performance, allow for long time simulations
- flexibility and ease of use: easily implement new models and new integrators
- features
  - ODEs and DAEs with holonomic, nonholonomic and Dirac constraints
  - methods: ERK, PRK, DIRK, FIRK, SIRK, SPARK, ...
  - solvers: Newton, quasi-Newton, Broyden, fixed-point, ...
- work in progress
  - parallel computation of multiple initial conditions
  - continuous/discontinuous Galerkin variational integrators
  - symbolic and automatic differentiation of Lagrangian, Hamiltonian and symplectic systems

GitHub repository: https://github.com/ddmgni/

Modules

Equation	Solution	Tableau	Integrator	Solver
ODE	Serial	ERK	ERK	LU (Julia)
DAE	Parallel	PRK	PRK	LU (LAPACK)
partitioned ODE		DIRK	DIRK	Newton
partitioned DAE		FIRK	FIRK	Quasi-Newton
		SIRK	SIRK	Broyden
		SARK	SARK	Fixed-Point
		SPARK	SPARK	JFNK
		GLM	GLM	
		Splitting	Splitting	

Example: Pendulum

```
using GeomDAE
using PyPlot

function f(x, fx)
    fx[1] = x[2]
    fx[2] = sin(x[1])
end

nt = 1000Δ
t = 0.1

x0  = [acos(0.4), 0.0]
ode = ODE(f, x0)
int = Integrator(ode, getTableauERK4(), Δt)
sol = Solution(ode, nt)

integrate!(int, sol)

fig = figure(figsize=(6,6))
plot(sol.x[1,:], sol.x[2,:])
savefig("pendulum_erk4.pdf")
```

Example: Pendulum (Short Version)

```julia
using GeomDAE
using PyPlot

function f(x, fx)
    fx[1] = x[2]
    fx[2] = sin(x[1])
end

sol = integrate!(f, [acos(0.4), 0.0], getTableauERK4(),
                 0.1, 1000)

fig = figure(figsize=(6,6))
plot(sol.x[1,:], sol.x[2,:])
savefig("pendulum_erk4.pdf")
```

Example: Pendulum

Timings for pendulum example with 100.000 time steps:

```
Running explicit_euler...       0.011861 seconds
Running explicit_midpoint...    0.021892 seconds
Running heun...                 0.017710 seconds
Running kutta...                0.023724 seconds
Running erk4_16...              0.030747 seconds
Running erk4_38...              0.032947 seconds
Running implicit_euler...       0.169213 seconds
Running glrk1...                0.176967 seconds
Running glrk2...                0.370926 seconds
Running glrk3...                0.716073 seconds
Running symplectic_euler_a...  0.028389 seconds
Running symplectic_euler_b...  0.028001 seconds
```

Comparison with ODE.jl, DifferentialEquations.jl, Sundials.jl

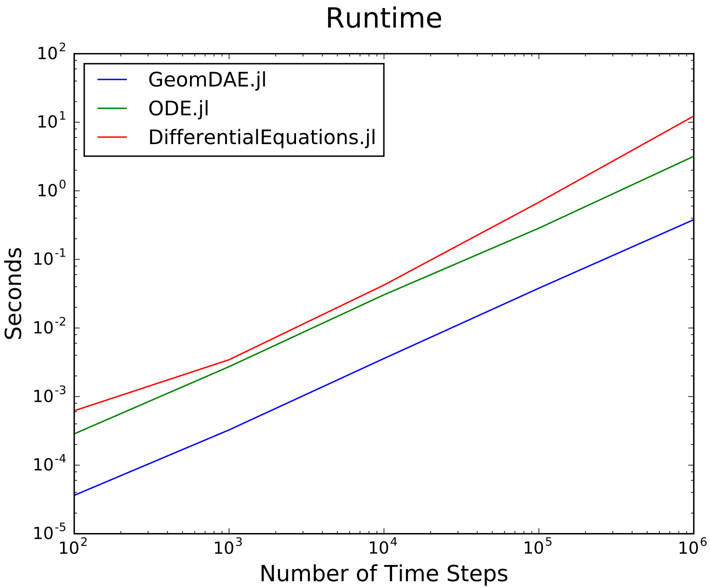
100.000 time steps

```
Running GeomDAE.erk4...  0.030747 seconds
        (0 allocations)
Running GeomDAE.erk4...  0.032098 seconds
        (12 allocations: 1.527 MB)
Running ODE.ode4...  0.346907 seconds
        (8.59 M allocations: 164.704 MB, 3.99% gc time)
Running DifferentialEquations.solve...  0.158038 seconds
        (3.00 M allocations: 124.554 MB, 20.35% gc time)
Running Sundials.cvode (BDF)...  0.065306 seconds
        (796.77 k allocations: 19.776 MB, 16.90% gc time)
Running Sundials.cvode (Adams)...  0.059689 seconds
        (912.25 k allocations: 23.270 MB, 3.89% gc time)
```

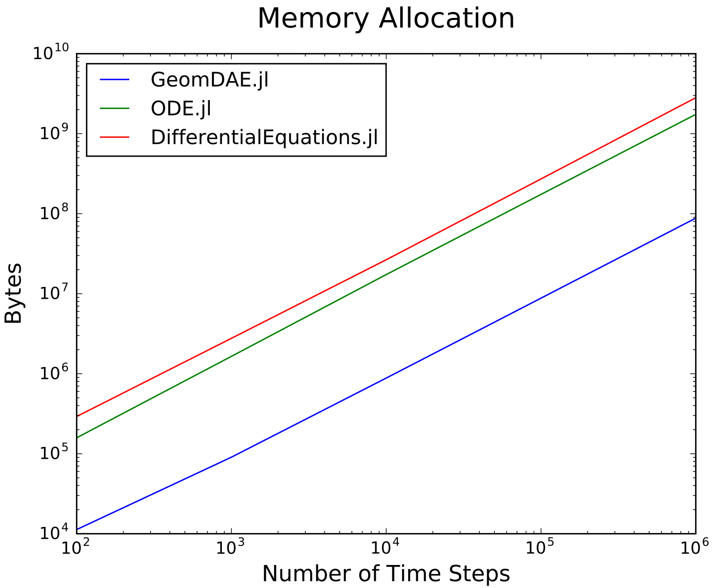
1.000.000 time steps

```
Running GeomDAE.erk4...  0.333444 seconds
        (0 allocations)
Running GeomDAE.erk4...  0.351395 seconds
        (12 allocations: 15.260 MB)
Running ODE.ode4...  3.812113 seconds
        (85.99 M allocations: 1.609 GB, 8.31% gc time)
Running DifferentialEquations.solve...  2.260353 seconds
        (30.00 M allocations: 1.188 GB, 43.77% gc time)
Running Sundials.cvode (BDF)...  0.538429 seconds
        (7.95 M allocations: 196.935 MB, 10.24% gc time)
Running Sundials.cvode (Adams)...  1.006567 seconds
        (13.10 M allocations: 353.876 MB, 1.63% gc time)
```

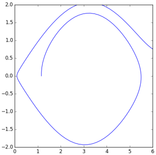
Comparison with ODE.jl and DifferentialEquations.jl



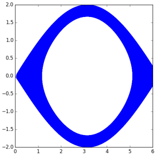
Comparison with ODE.jl and DifferentialEquations.jl



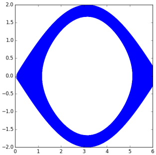
Example: Pendulum (100.000 Time Steps)



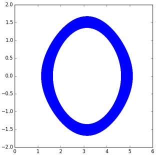
explicit Euler



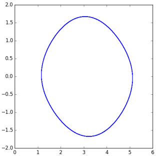
explicit midpoint



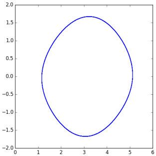
Heun



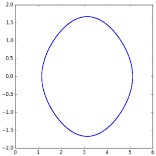
Kutta



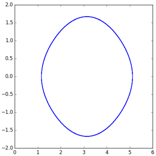
Symplectic Euler A



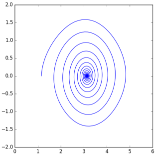
Symplectic Euler B



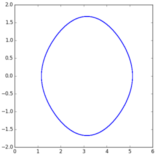
Explicit RK4 (1/6)



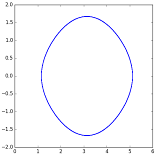
Explicit RK4 (3/8)



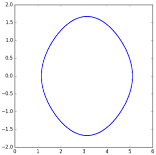
implicit Euler



GLRK1



GLRK2



GLRK3