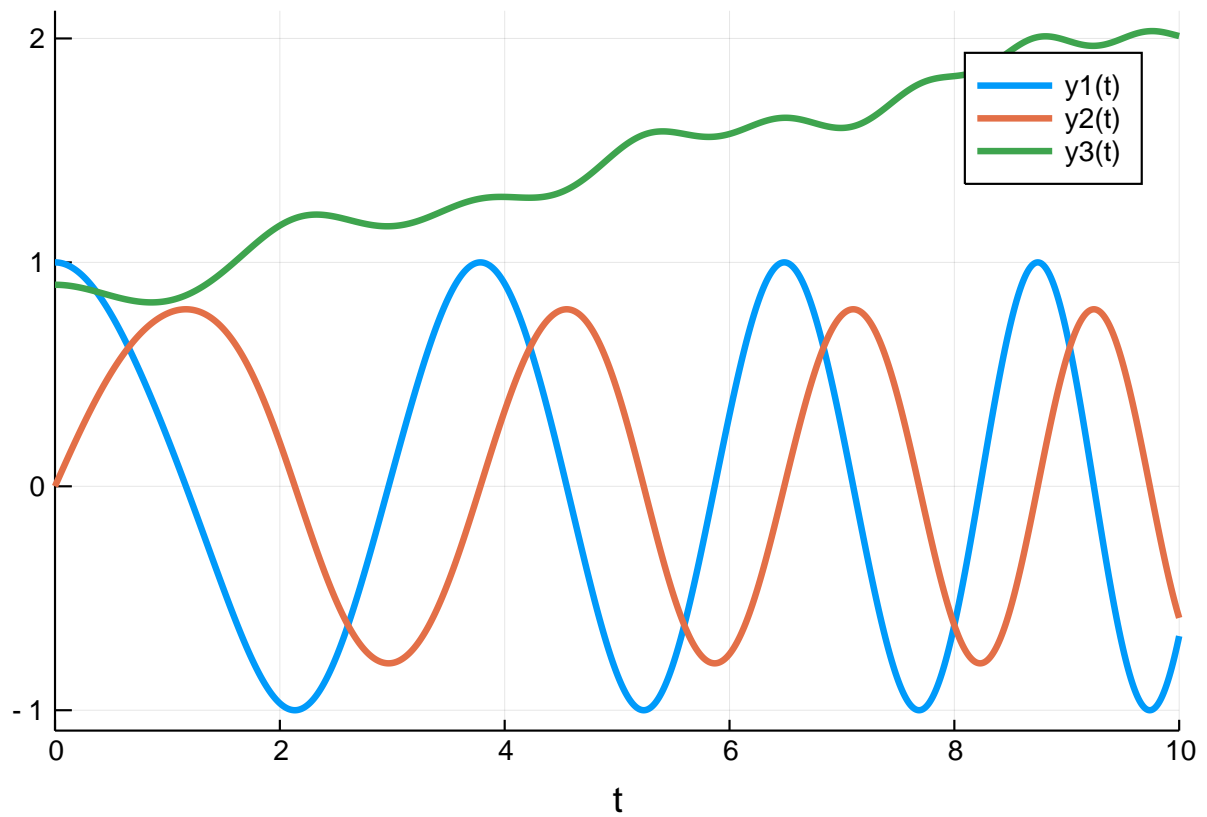


Rigid Body Work-Precision Diagrams

Chris Rackauckas

March 2, 2019

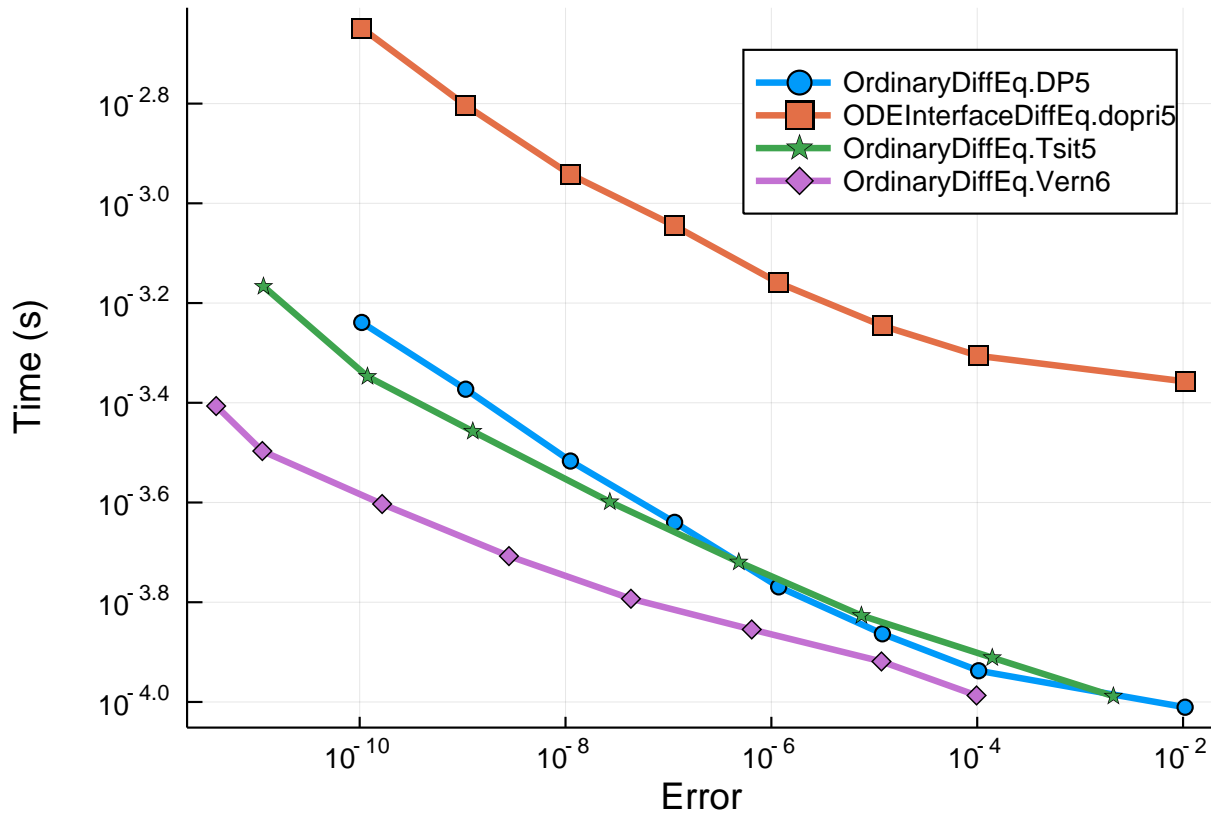
```
using OrdinaryDiffEq, ParameterizedFunctions, ODE, ODEInterfaceDiffEq, LSODA,  
    Sundials, DiffEqDevTools  
  
k(t) = 0.25*sin(t)^2  
  
g = @ode_def RigidBody begin  
    dy1 = I_1*y2*y3  
    dy2 = I_2*y1*y3  
    dy3 = I_3*y1*y2 + k(t)  
end I_1 I_2 I_3  
  
p = [-2.0, 1.25, -0.5]  
prob = ODEProblem(g, [1.0; 0.0; 0.9], (0.0, 10.0), p)  
  
abstols = 1.0 ./ 10.0 .^ (6:13)  
reltols = 1.0 ./ 10.0 .^ (3:10);  
sol = solve(prob, Vern7(), abstol=1/10^14, reltol=1/10^14)  
test_sol = TestSolution(sol)  
using Plots; gr()  
  
plot(sol)
```



```

setups = [Dict(:alg=>DP5())
          #Dict(:alg=>ode45()) # fails
          Dict(:alg=>dopri5())
          Dict(:alg=>Tsit5())
          Dict(:alg=>Vern6())
]
wp =
  WorkPrecisionSet(prob, abstols, reltols, setups; appxsol=test_sol, save_everystep=true, numruns=100, maxi
plot(wp)

```



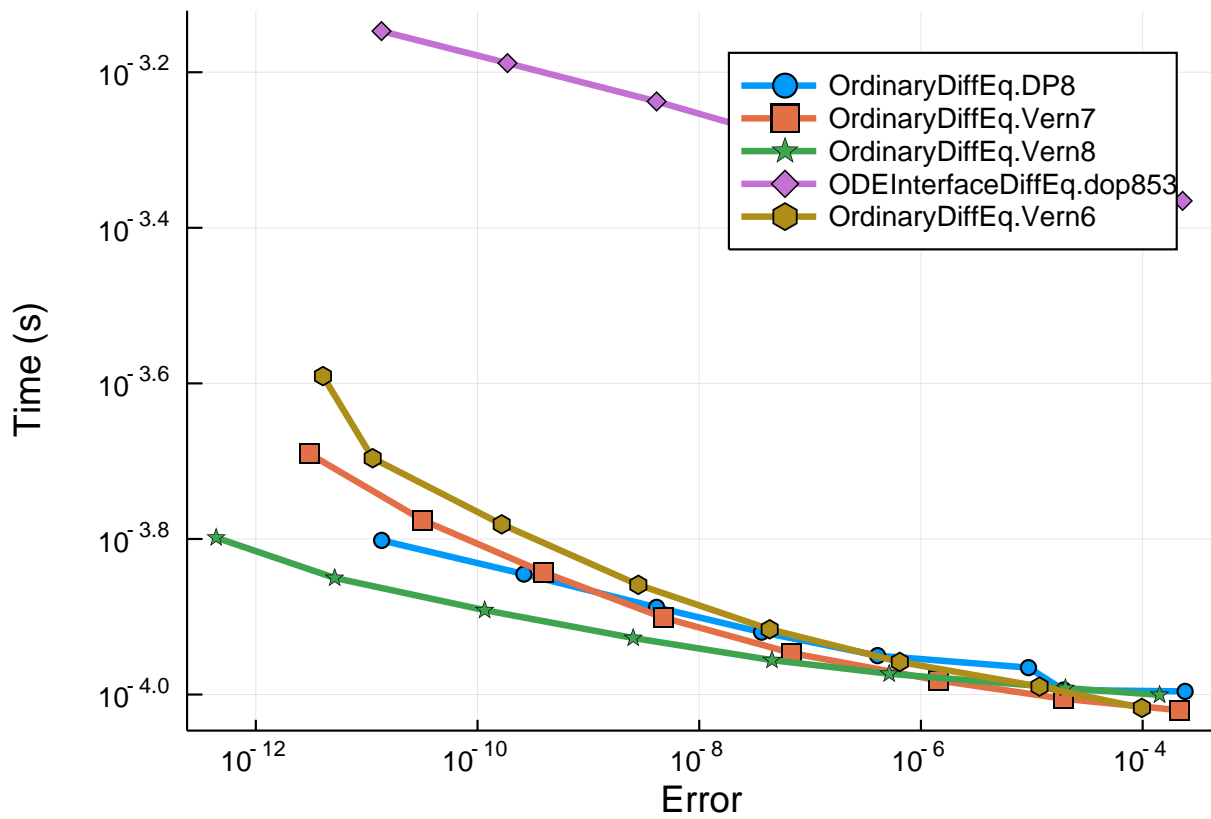
The DifferentialEquations.jl algorithms once again pull ahead. This is the first benchmark we've ran where `ode45` doesn't fail. However, it still doesn't do as well as `Tsit5`. One reason why it does so well is that the maximum norm that ODE.jl uses (as opposed to the L2 norm of Sundials, DifferentialEquations, and ODEInterface) seems to do really well on this problem. `dopri5` does surprisingly bad in this test.

0.1 Higher Order

```

setups = [Dict(:alg=>DP8())
           #Dict(:alg=>ode78()) # fails
           Dict(:alg=>Vern7())
           Dict(:alg=>Vern8())
           Dict(:alg=>dop853())
           Dict(:alg=>Vern6())
]
wp =
  WorkPrecisionSet(prob, abstols, reltols, setups; appxsol=test_sol, save_everystep=false, numruns=100, max
plot(wp)

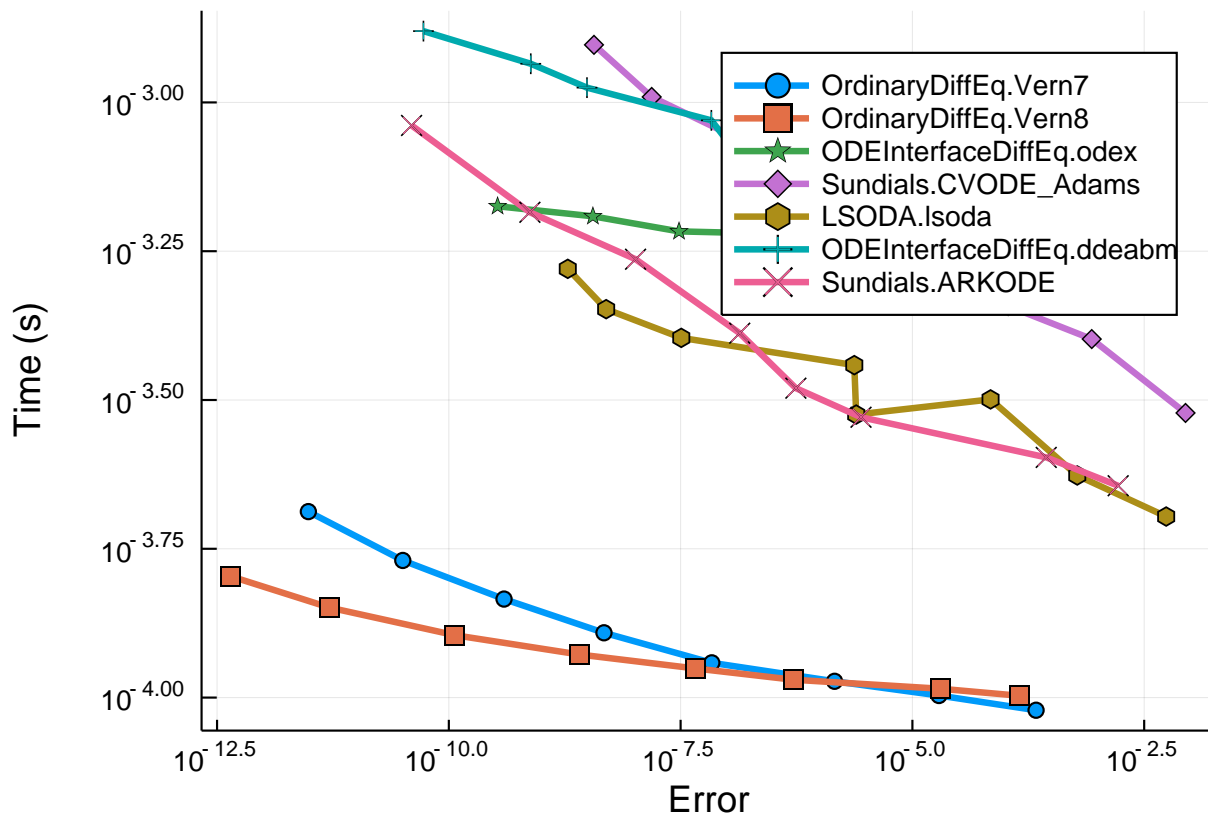
```



```

setups = [Dict(:alg=>Vern7())
          Dict(:alg=>Vern8())
          Dict(:alg=>odex())
          Dict(:alg=>CVODE_Adams())
          Dict(:alg=>lsoda())
          Dict(:alg=>ddeabm())
          Dict(:alg=>ARKODE(Sundials.Explicit(),order=6))
]
wp =
  WorkPrecisionSet(prob, abstols, reltols, setups; appxsol=test_sol, save_everystep=false, numruns=100, max
plot(wp)

```



0.1.1 Conclusion

Once again, the OrdinaryDiffEq.jl pull far ahead in terms of speed and accuracy.

```
using DiffEqBenchmarks
DiffEqBenchmarks.bench_footer(WEAVE_ARGS[:folder],WEAVE_ARGS[:file])
```

0.2 Appendix

These benchmarks are a part of the DiffEqBenchmarks.jl repository, found at: <https://github.com/JuliaDiffEq/DiffEqBenchmarks.jl>

To locally run this tutorial, do the following commands:

```
using DiffEqBenchmarks
DiffEqBenchmarks.weave_file("NonStiffODE","RigidBody_wpd.jmd")
```

Computer Information:

```
Julia Version 1.1.0
Commit 80516ca202 (2019-01-21 21:24 UTC)
Platform Info:
  OS: Linux (x86_64-pc-linux-gnu)
  CPU: Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz
  WORD_SIZE: 64
  LIBM: libopenlibm
  LLVM: libLLVM-6.0.1 (ORCJIT, haswell)
```

Package Information:

```
Status: `~/home/yingboma/.julia/dev/DiffEqBenchmarks/Project.toml`
[f3b72e0c-5b89-59e1-b016-84e28bfd966d] DiffEqDevTools 2.7.0
[7073ff75-c697-5162-941a-fcdaad2a7d2a] IJulia 1.17.0
[7f56f5a3-f504-529b-bc02-0b1fe5e64312] LSODA 0.4.0
[c030b06c-0b6d-57c2-b091-7029874bd033] ODE 2.4.0
[09606e27-ecf5-54fc-bb29-004bd9f985bf] ODEInterfaceDiffEq 3.0.0
[1dea7af3-3e70-54e6-95c3-0bf5283fa5ed] OrdinaryDiffEq 5.3.0
[65888b18-ceab-5e60-b2b9-181511a3b968] ParameterizedFunctions 4.1.1
[91a5bcd-d55d7-5caf-9e0b-520d859cae80] Plots 0.23.1
[c3572dad-4567-51f8-b174-8c6c989267f4] Sundials 3.1.0
[44d3d7a6-8a23-5bf8-98c5-b353f8df5ec9] Weave 0.7.2
[b77e0a4c-d291-57a0-90e8-8db25a27a240] InteractiveUtils
[d6f4376e-aef5-505a-96c1-9c027394607a] Markdown
[44cfe95a-1eb2-52ea-b672-e2afdf69b78f] Pkg
[9a3f8284-a2c9-5f02-9a11-845980a1fd5c] Random
```