Lotka-Volterra Work-Precision Diagrams

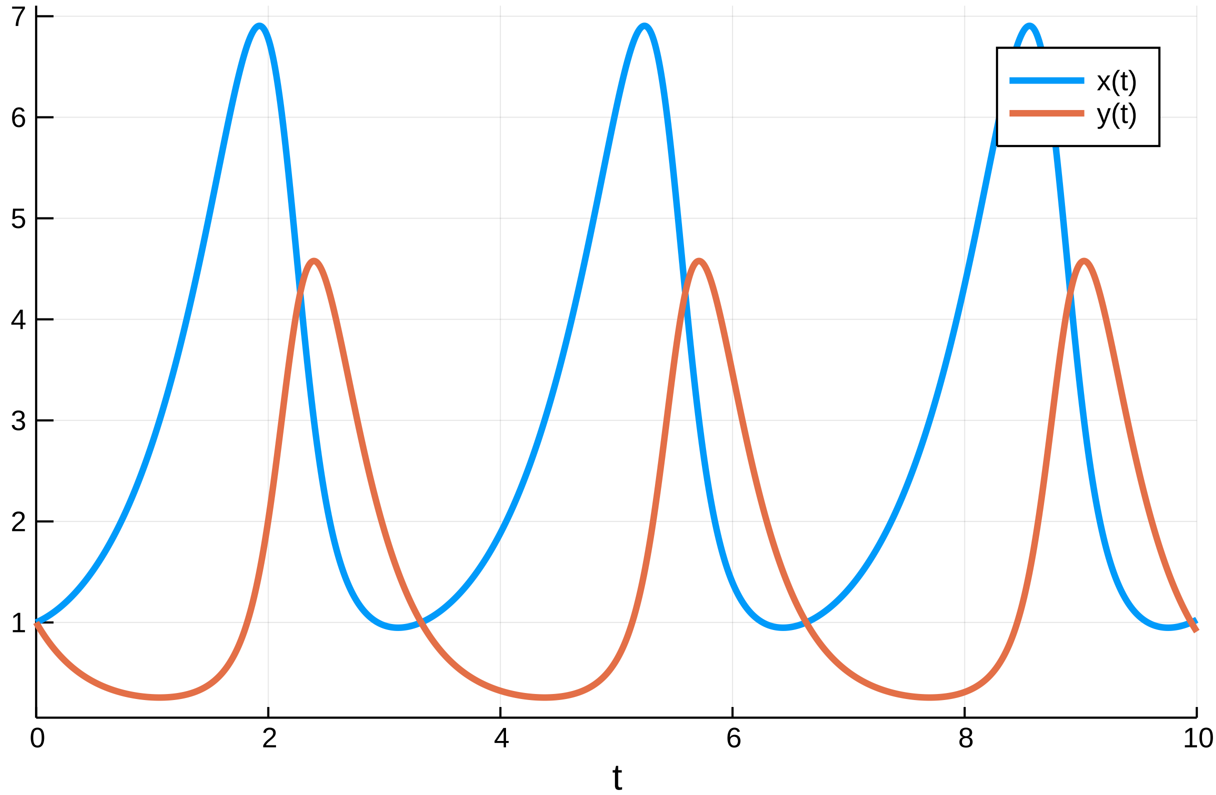
Chris Rackauckas

September 26, 2019

0.1 Lotka-Volterra

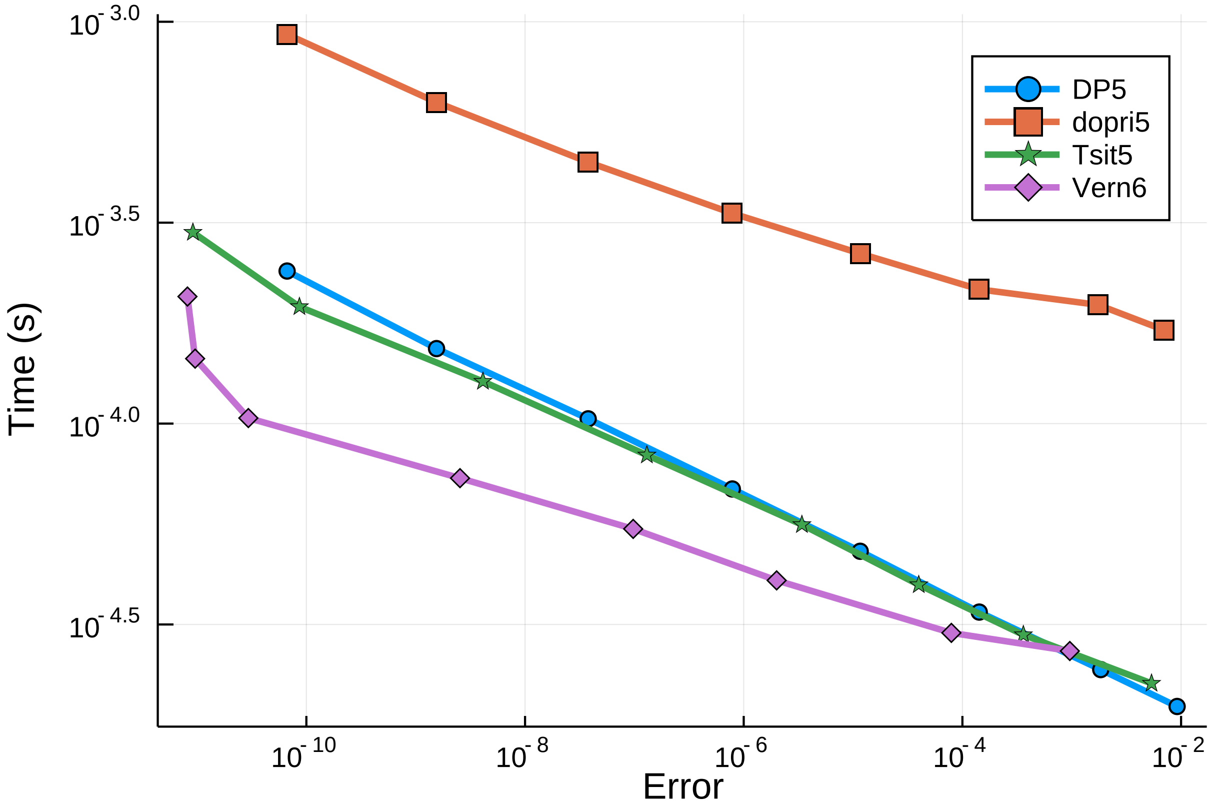
The purpose of this problem is to test the performance on easy problems. Since it's periodic, the error is naturally low, and so most of the difference will come down to startup times and, when measuring the interpolations, the algorithm choices.

```
using OrdinaryDiffEq, ParameterizedFunctions, ODE, ODEInterfaceDiffEq, LSODA,
      Sundials, DiffEqDevTools

f = @ode_def LotkaVolterra begin
  dx = a*x - b*x*y
  dy = -c*y + d*x*y
end a b c d

p = [1.5,1.0,3.0,1.0]
prob = ODEProblem(f,[1.0;1.0],(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)
```



0.1.1 Low Order

```
setups = [Dict(:alg=>DP5())
          #Dict(:alg=>ode45()) # fail
          Dict(:alg=>dopri5())
          Dict(:alg=>Tsit5())
          Dict(:alg=>Vern6())
]
wp =
    WorkPrecisionSet(prob,abstols,reltols,setups;appxsol=test_sol,save_everystep=false,maxiters=10000,
plot(wp)
```

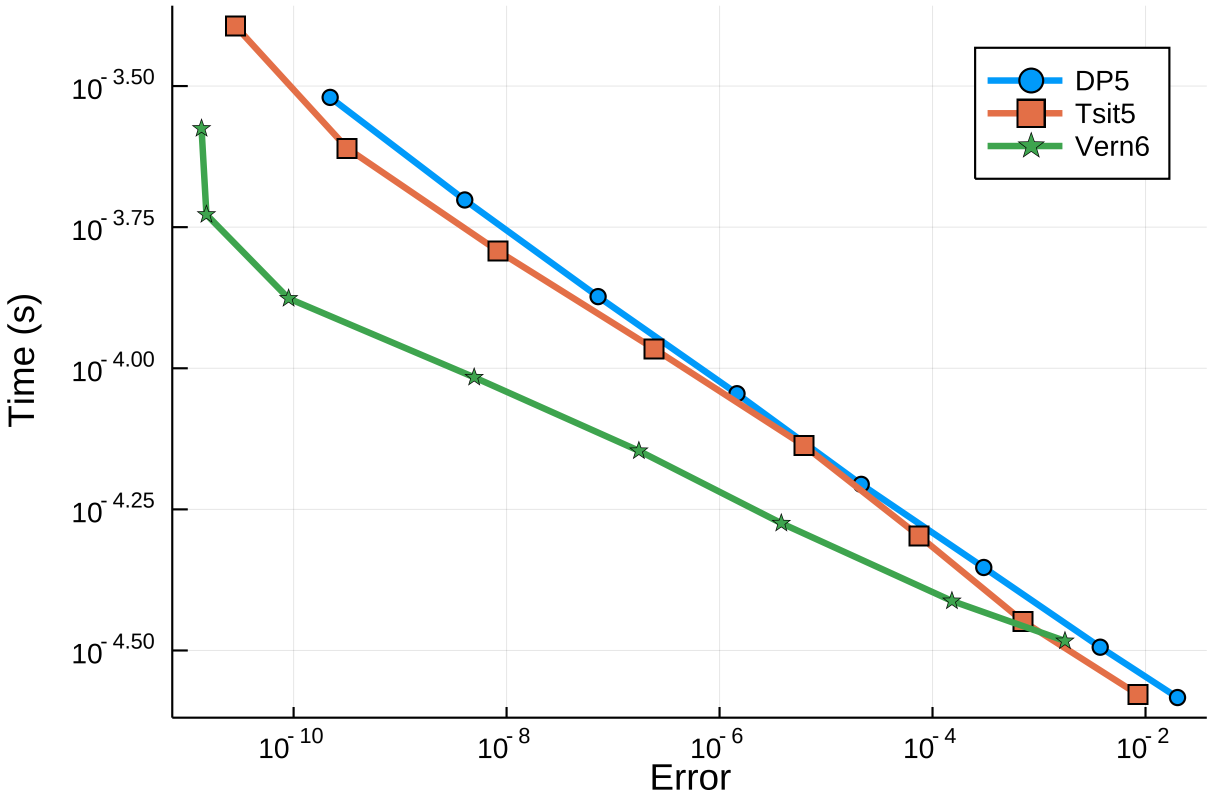


Here we see the OrdinaryDiffEq.jl algorithms once again far in the lead.

0.1.2 Interpolation Error

Since the problem is periodic, the real measure of error is the error throughout the solution.

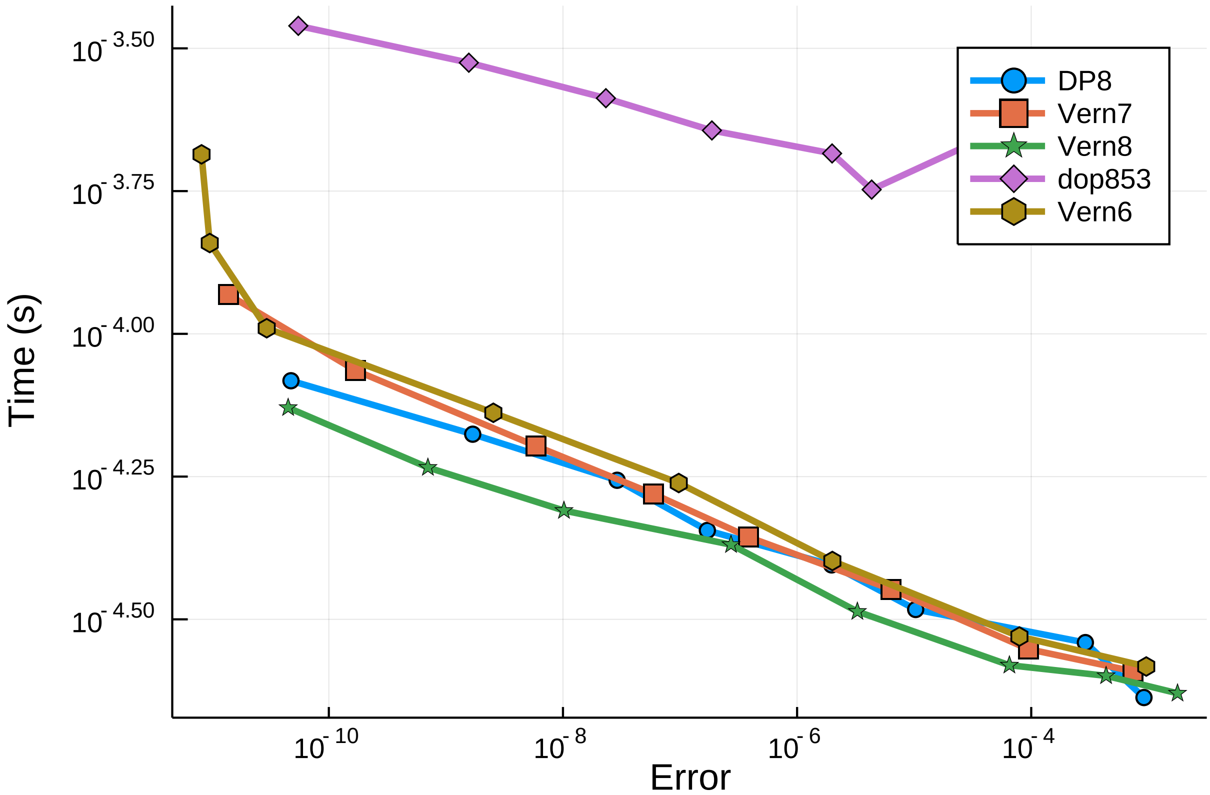
```
setups = [Dict(:alg=>DP5())
          #Dict(:alg=>ode45())
          Dict(:alg=>Tsit5())
          Dict(:alg=>Vern6())
]
wp =
    WorkPrecisionSet(prob,abstols,reltols,setups;appxsol=test_sol,maxiters=10000,error_estimate=:L2,de
plot(wp)
```



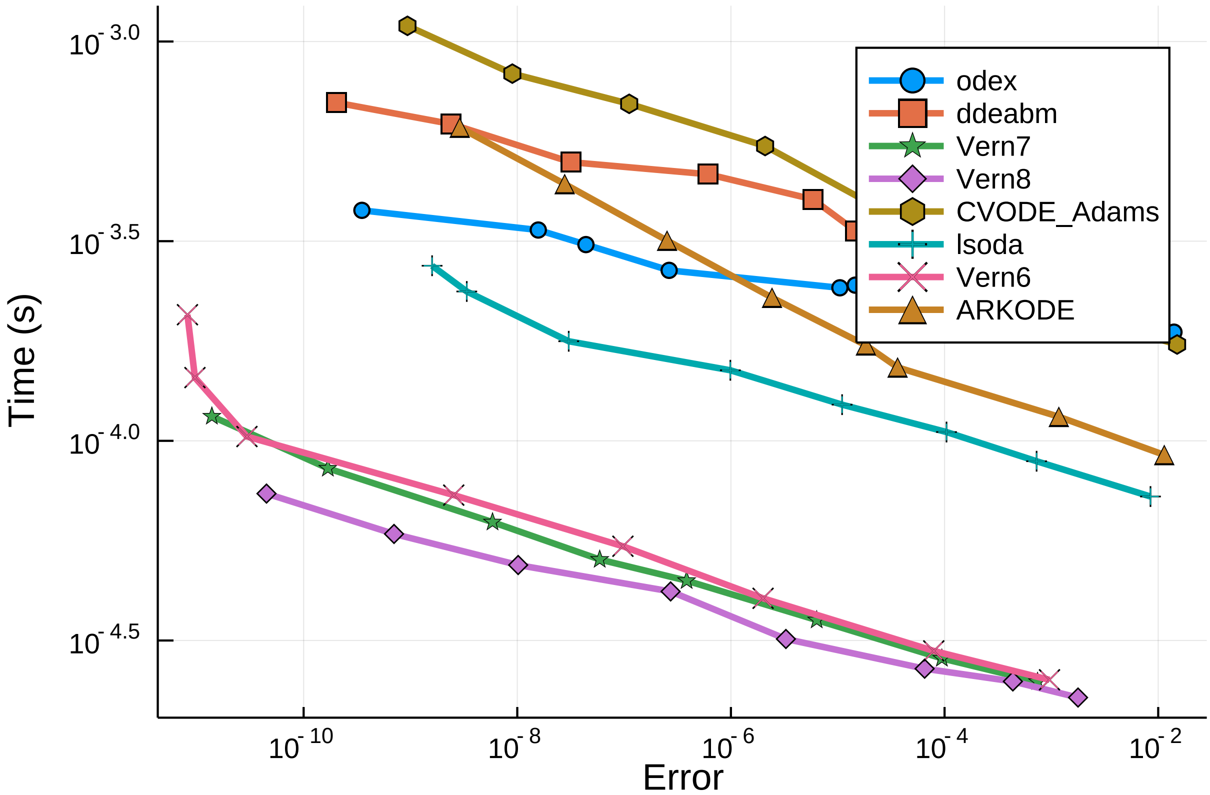
Here we see the power of algorithm specific interpolations. The ODE.jl algorithm is only able to reach 10^{-7} error even at a tolerance of 10^{-13}, while the DifferentialEquations.jl algorithms are below 10^{-10}

0.2 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,maxiters=1000,r
plot(wp)
```

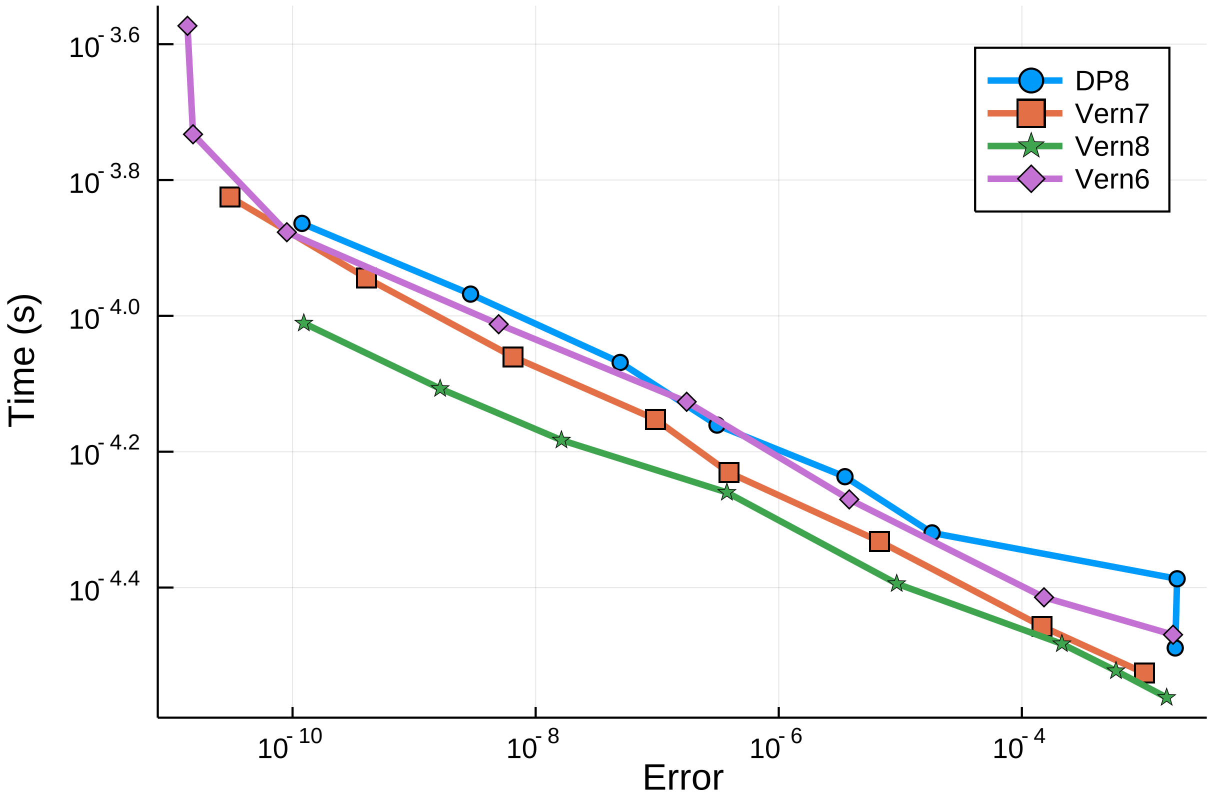



```
setups = [Dict(:alg=>odex())
          Dict(:alg=>ddeabm())
          Dict(:alg=>Vern7())
          Dict(:alg=>Vern8())
          Dict(:alg=>CVODE_Adams())
          Dict(:alg=>lsoda())
          Dict(:alg=>Vern6())
          Dict(:alg=>ARKODE(Sundials.Explicit(),order=6))
          ]
wp =
    WorkPrecisionSet(prob,abstols,reltols,setups;appxsol=test_sol,save_everystep=false,maxiters=1000,n
plot(wp)
```



Again we look at interpolations:

```
setups = [Dict(:alg=>DP8())
          #Dict(:alg=>ode78())
          Dict(:alg=>Vern7())
          Dict(:alg=>Vern8())
          Dict(:alg=>Vern6())
]
wp =
    WorkPrecisionSet(prob,abstols,reltols,setups;appxsol=test_sol,dense=true,maxiters=1000,error_estim
plot(wp)
```

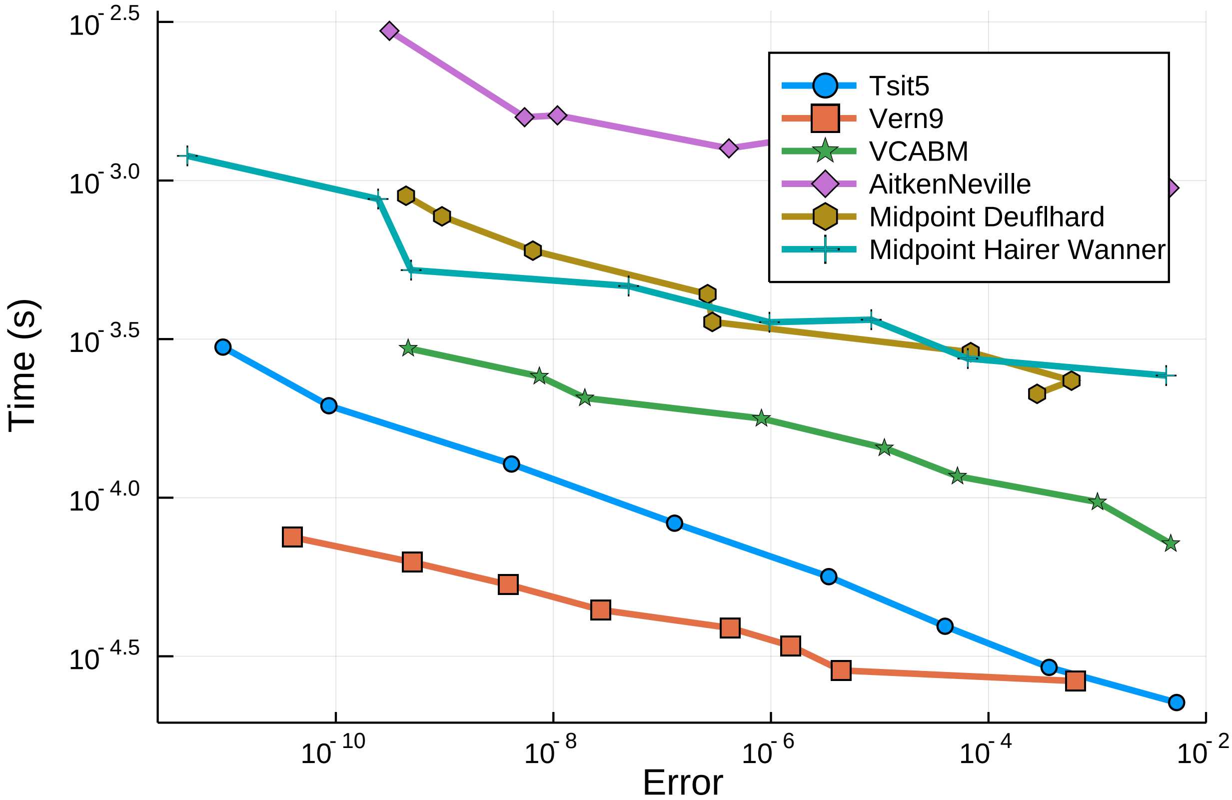


Again, the ODE.jl algorithms suffer when measuring the interpolations due to relying on an order 3 Hermite polynomial instead of an algorithm-specific order matching interpolation which uses the timesteps.

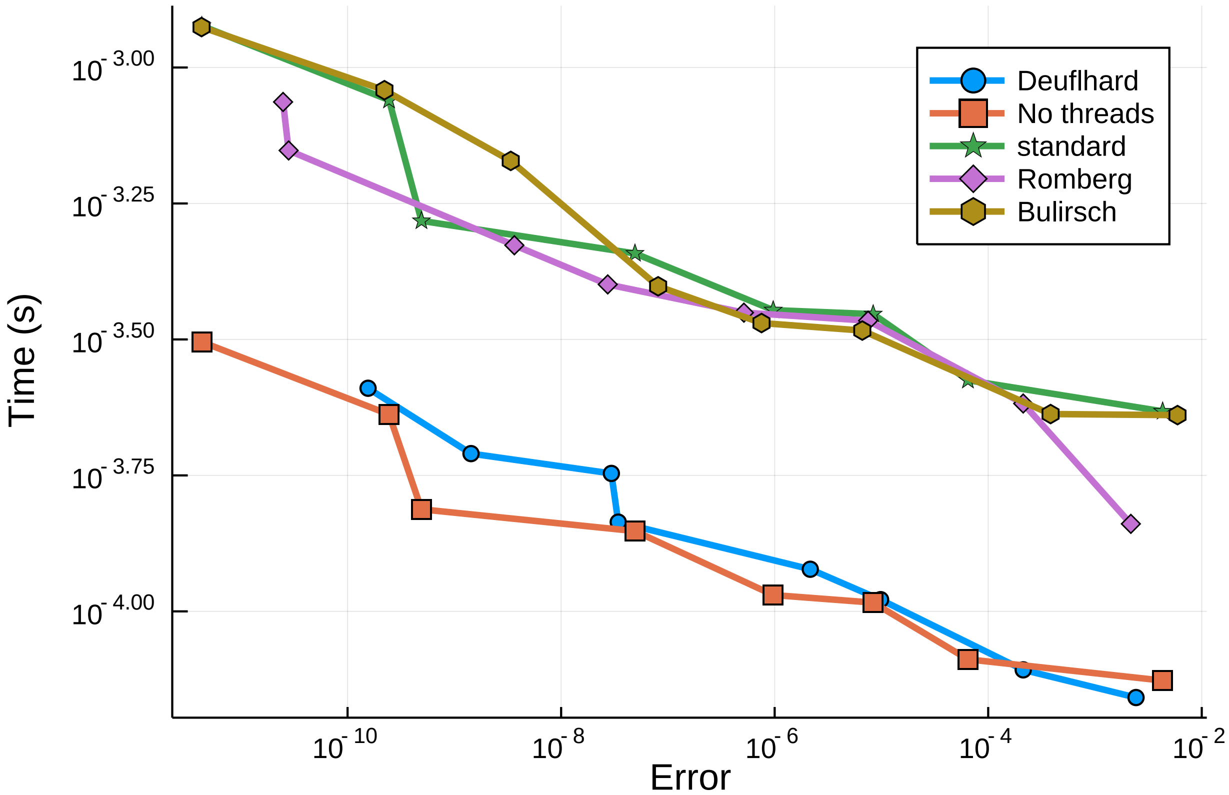
0.3 Comparison with Non-RK methods

Now let's test Tsit5 and Vern9 against parallel extrapolation methods and an Adams-Bashforth-Moulton:

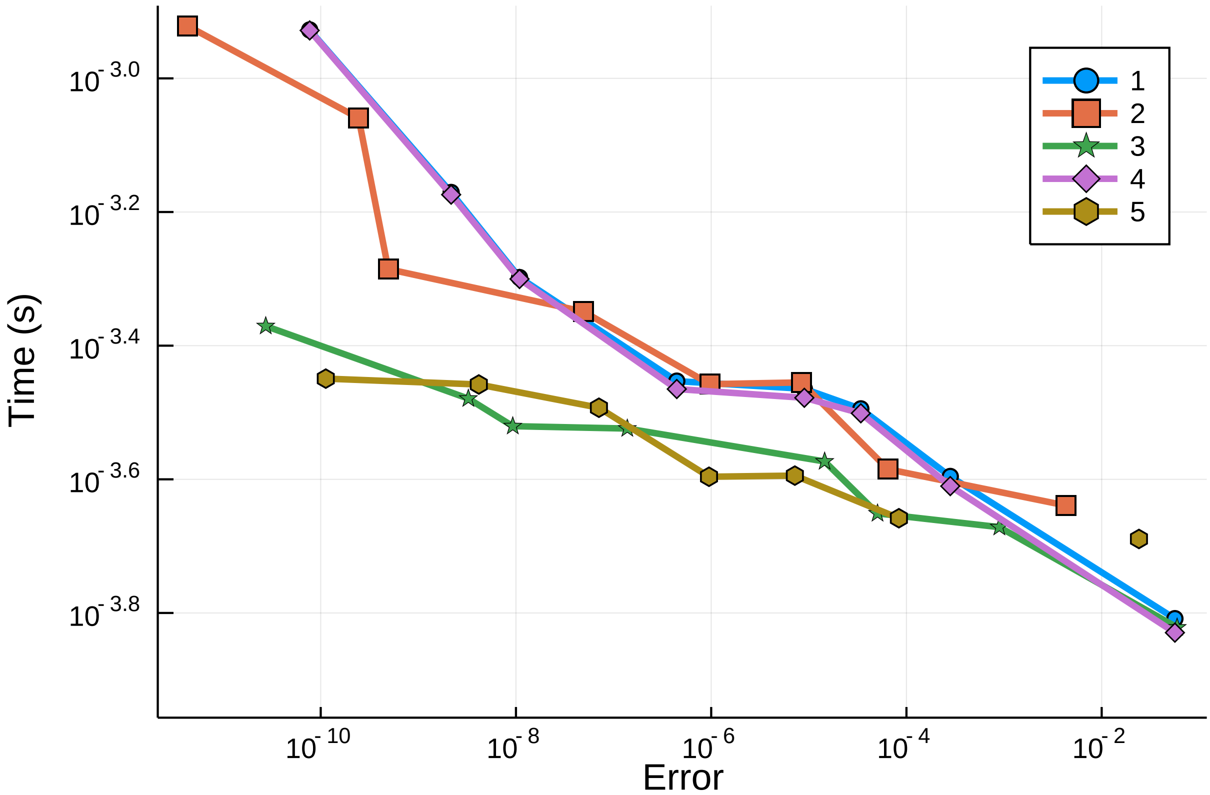
```
setups = [Dict(:alg=>Tsit5())
          Dict(:alg=>Vern9())
          Dict(:alg=>VCABM())
          Dict(:alg=>AitkenNeville(min_order=1, max_order=9, init_order=4,
    threading=true))
          Dict(:alg=>ExtrapolationMidpointDeuflhard(min_order=1, max_order=9,
    init_order=4, threading=true))
          Dict(:alg=>ExtrapolationMidpointHairerWanner(min_order=2, max_order=11,
    init_order=4, threading=true))]
solnames = ["Tsit5","Vern9","VCABM","AitkenNeville","Midpoint Deuflhard","Midpoint Hairer
    Wanner"]
wp = WorkPrecisionSet(prob,abstols,reltols,setups;appxsol=test_sol,names=solnames,
                      save_everystep=false,verbose=false,numruns=100)
plot(wp)
```




```
setups = [Dict(:alg=>ExtrapolationMidpointDeuflhard(min_order=1, max_order=9,
    init_order=9, threading=false))
          Dict(:alg=>ExtrapolationMidpointHairerWanner(min_order=2, max_order=11,
    init_order=4, threading=false))
          Dict(:alg=>ExtrapolationMidpointHairerWanner(min_order=2, max_order=11,
    init_order=4, threading=true))
          Dict(:alg=>ExtrapolationMidpointHairerWanner(min_order=2, max_order=11,
    init_order=4, sequence = :romberg, threading=true))
          Dict(:alg=>ExtrapolationMidpointHairerWanner(min_order=2, max_order=11,
    init_order=4, sequence = :bulirsch, threading=true))]
solnames = ["Deuflhard","No threads","standard","Romberg","Bulirsch"]
wp = WorkPrecisionSet(prob,abstols,reltols,setups;appxsol=test_sol,names=solnames,
                      save_everystep=false,verbose=false,numruns=100)
plot(wp)
```




```
setups = [Dict(:alg=>ExtrapolationMidpointHairerWanner(min_order=2, max_order=11,
    init_order=10, threading=true))
          Dict(:alg=>ExtrapolationMidpointHairerWanner(min_order=2, max_order=11,
    init_order=4, threading=true))
          Dict(:alg=>ExtrapolationMidpointHairerWanner(min_order=5, max_order=11,
    init_order=10, threading=true))
          Dict(:alg=>ExtrapolationMidpointHairerWanner(min_order=2, max_order=15,
    init_order=10, threading=true))
          Dict(:alg=>ExtrapolationMidpointHairerWanner(min_order=5, max_order=7,
    init_order=6, threading=true))]
solnames = ["1","2","3","4","5"]
wp = WorkPrecisionSet(prob,abstols,reltols,setups;appxsol=test_sol,names=solnames,
                      save_everystep=false,verbose=false,numruns=100)
plot(wp)
```



0.4 Conclusion

The OrdinaryDiffEq.jl are quicker and still solve to a much higher accuracy, especially when the interpolations are involved. ODE.jl errors a lot.

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

0.5 Appendix

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

To locally run this tutorial, do the following commands:

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

Computer Information:

```
Julia Version 1.2.0
Commit c6da87ff4b (2019-08-20 00:03 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/crackauckas/.julia/dev/DiffEqBenchmarks/Project.toml`
[a134a8b2-14d6-55f6-9291-3336d3ab0209] BlackBoxOptim 0.5.0
[f3b72e0c-5b89-59e1-b016-84e28bfd966d] DiffEqDevTools 2.15.0
[1130ab10-4a5a-5621-a13d-e4788d82bd4c] DiffEqParamEstim 1.8.0
[a077e3f3-b75c-5d7f-a0c6-6bc4c8ec64a9] DiffEqProblemLibrary 4.5.1
[ef61062a-5684-51dc-bb67-a0fcdec5c97d] DiffEqUncertainty 1.2.0
[7073ff75-c697-5162-941a-fcdaad2a7d2a] IJulia 1.20.0
[7f56f5a3-f504-529b-bc02-0b1fe5e64312] LSODA 0.6.1
[76087f3c-5699-56af-9a33-bf431cd00edd] NLopt 0.5.1
[c030b06c-0b6d-57c2-b091-7029874bd033] ODE 2.5.0
[54ca160b-1b9f-5127-a996-1867f4bc2a2c] ODEInterface 0.4.6
[09606e27-ecf5-54fc-bb29-004bd9f985bf] ODEInterfaceDiffEq 3.4.0
[1dea7af3-3e70-54e6-95c3-0bf5283fa5ed] OrdinaryDiffEq 5.17.1
[65888b18-ceab-5e60-b2b9-181511a3b968] ParameterizedFunctions 4.2.1
[91a5bcdd-55d7-5caf-9e0b-520d859cae80] Plots 0.26.3
[c3572dad-4567-51f8-b174-8c6c989267f4] Sundials 3.7.0
[44d3d7a6-8a23-5bf8-98c5-b353f8df5ec9] Weave 0.9.1
[b77e0a4c-d291-57a0-90e8-8db25a27a240] InteractiveUtils
[d6f4376e-aef5-505a-96c1-9c027394607a] Markdown
[44cfe95a-1eb2-52ea-b672-e2afdf69b78f] Pkg
[9a3f8284-a2c9-5f02-9a11-845980a1fd5c] Random
```