ROBER Work-Precision Diagrams

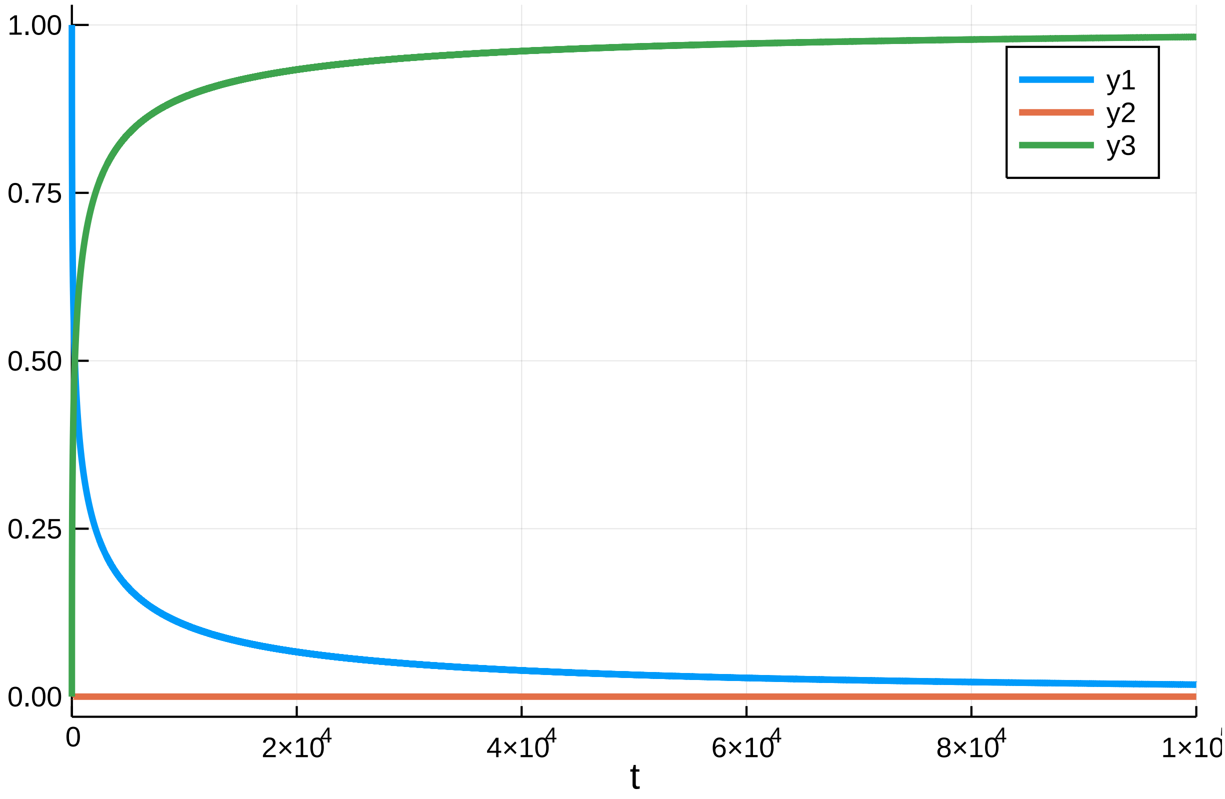
Chris Rackauckas

September 26, 2019

```
using OrdinaryDiffEq, DiffEqDevTools, Sundials, ParameterizedFunctions, Plots, ODE,
    ODEInterfaceDiffEq, LSODA
gr()
using LinearAlgebra
LinearAlgebra.BLAS.set_num_threads(1)

rober = @ode_def begin
  dy_1 = -k_1*y_1+k_3*y_2*y_3
  dy_2 =  k_1*y_1-k_2*y_2^2-k_3*y_2*y_3
  dy_3 =  k_2*y_2^2
end k_1 k_2 k_3
prob = ODEProblem(rober,[1.0,0.0,0.0],(0.0,1e5),(0.04,3e7,1e4))
sol = solve(prob,CVODE_BDF(),abstol=1/10^14,reltol=1/10^14)
test_sol = TestSolution(sol)
abstols = 1.0 ./ 10.0 .^ (4:11)
reltols = 1.0 ./ 10.0 .^ (1:8);

plot(sol,labels=["y1","y2","y3"])
```



0.1 Omissions And Tweaking

The following were omitted from the tests due to convergence failures. ODE.jl's adaptivity is not able to stabilize its algorithms, while GeometricIntegratorsDiffEq has not upgraded to Julia 1.0. GeometricIntegrators.jl's methods used to be either fail to converge at comparable dts (or on some computers errors due to type conversions).

```
#sol = solve(prob,ode23s()); println("Total ODE.jl steps:  $(length(sol))")
#using GeometricIntegratorsDiffEq
#try
# sol = solve(prob,GIRadIIA3(),dt=1/10)
#catch e
# println(e)
#end
```

`ARKODE` needs a lower `nonlinear_convergence_coefficient` in order to not diverge.

```
#sol = solve(prob,ARKODE(nonlinear_convergence_coefficient =
    1e-6),abstol=1e-5,reltol=1e-1); # Noisy, output omitted

sol = solve(prob,ARKODE(nonlinear_convergence_coefficient =
    1e-7),abstol=1e-5,reltol=1e-1);
```

Note that `1e-7` matches the value from the Sundials manual which was required for their example to converge on this problem. The default is `1e-1`.

```
#sol = solve(prob,ARKODE(order=3),abstol=1e-4,reltol=1e-1); # Fails to diverge but
    doesn't finish

#sol = solve(prob,ARKODE(order=5),abstol=1e-4,reltol=1e-1); # Noisy, output omitted

#sol = solve(prob,ARKODE(order=5,nonlinear_convergence_coefficient =
    1e-9),abstol=1e-5,reltol=1e-1); # Noisy, output omitted
```

Additionally, the ROCK methods do not perform well on this benchmark.

```
setups = [
          #Dict(:alg=>ROCK2()) #Unstable
          #Dict(:alg=>ROCK4()) #needs more iterations
          ]

0-element Array{Any,1}
```

Some of the bad Rosenbrocks fail:

```
setups = [
  #Dict(:alg=>Hairer4()),
  #Dict(:alg=>Hairer42()),
  #Dict(:alg=>Cash4()),
]

0-element Array{Any,1}
```

The EPIRK and exponential methods also fail:

```
sol = solve(prob,EXPRB53s3(),dt=2.0^(-8));

Error: InexactError: trunc(Int64, Inf)

sol = solve(prob,EPIRK4s3B(),dt=2.0^(-8));
```

```
Error: InexactError: trunc(Int64, Inf)
```

```
sol = solve(prob,EPIRK5P2(),dt=2.0^(-8));
```

```
Error: InexactError: trunc(Int64, Inf)
```

PDIRK44 also fails

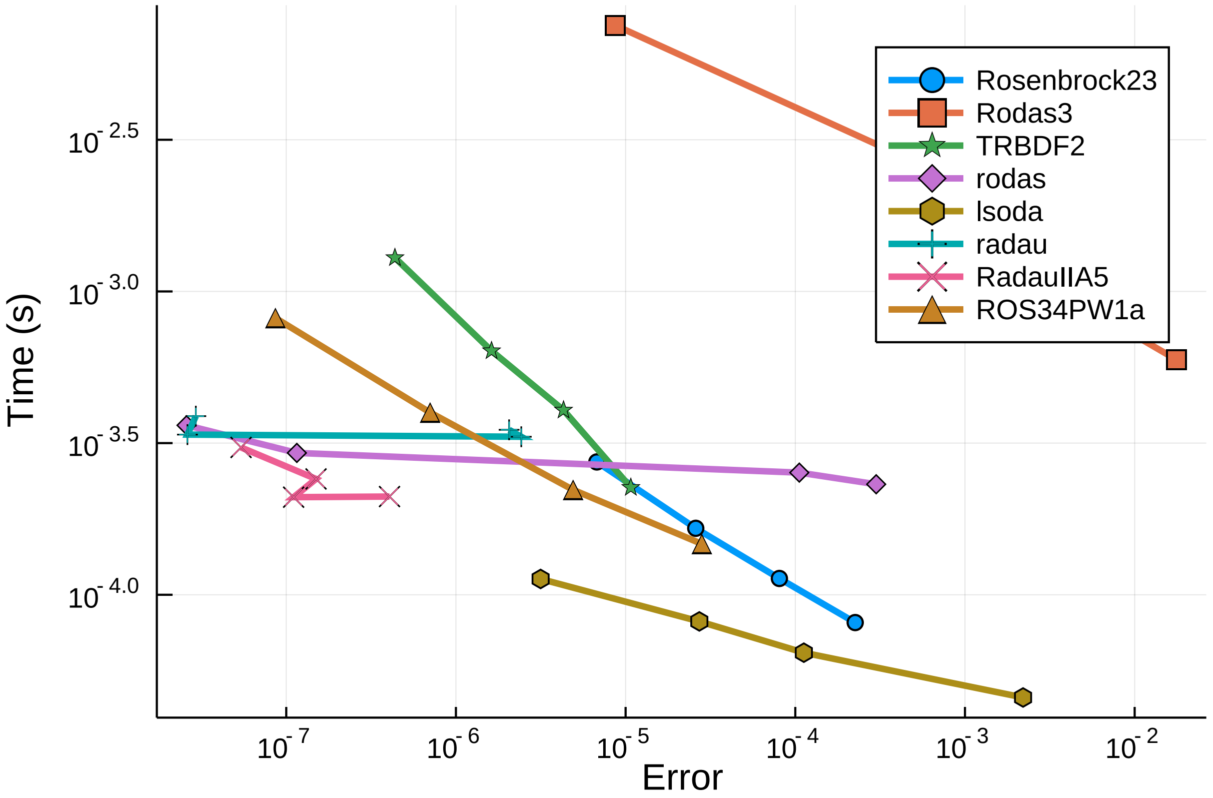
```
sol = solve(prob,PDIRK44(),dt=2.0^(-8));
```

In fact, all non-adaptive methods fail on this problem.

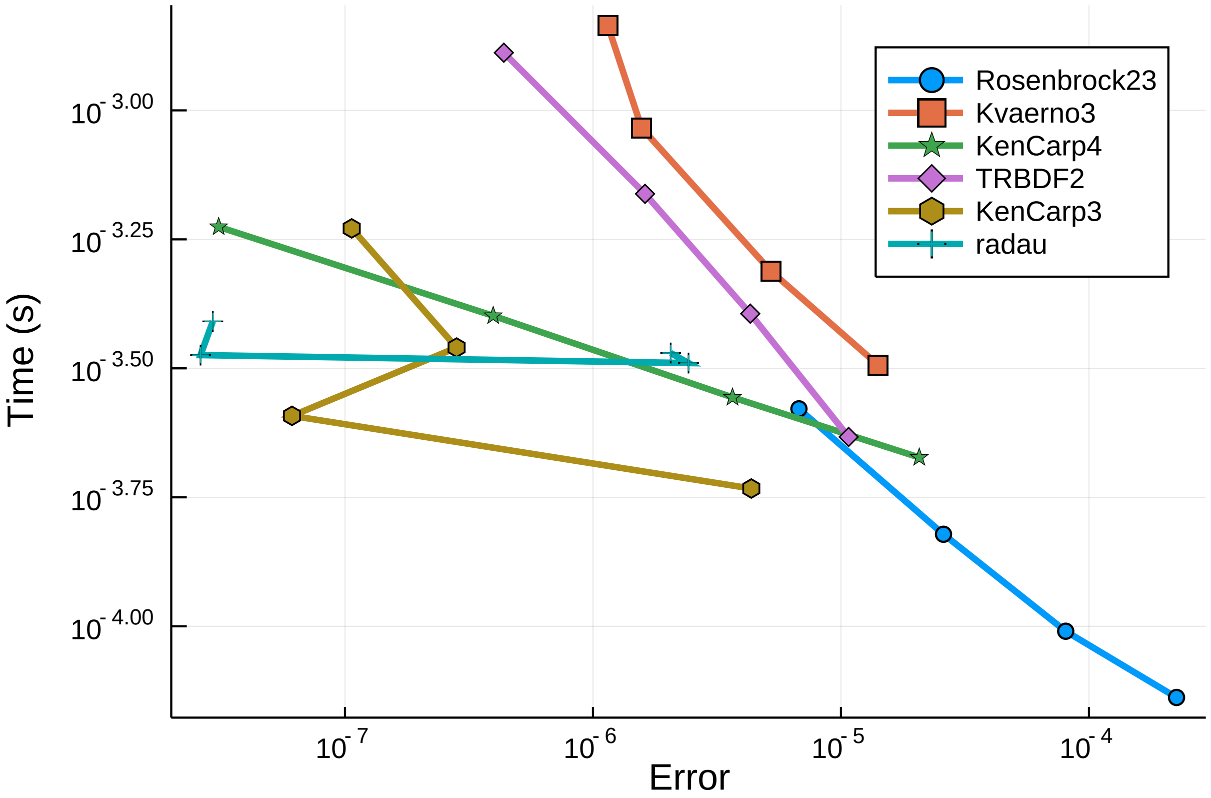
0.2 High Tolerances

This is the speed when you just want the answer. `ode23s` from ODE.jl was removed since it fails. Note that at high tolerances Sundials' `CVODE_BDF` fails as well so it's excluded from this test.

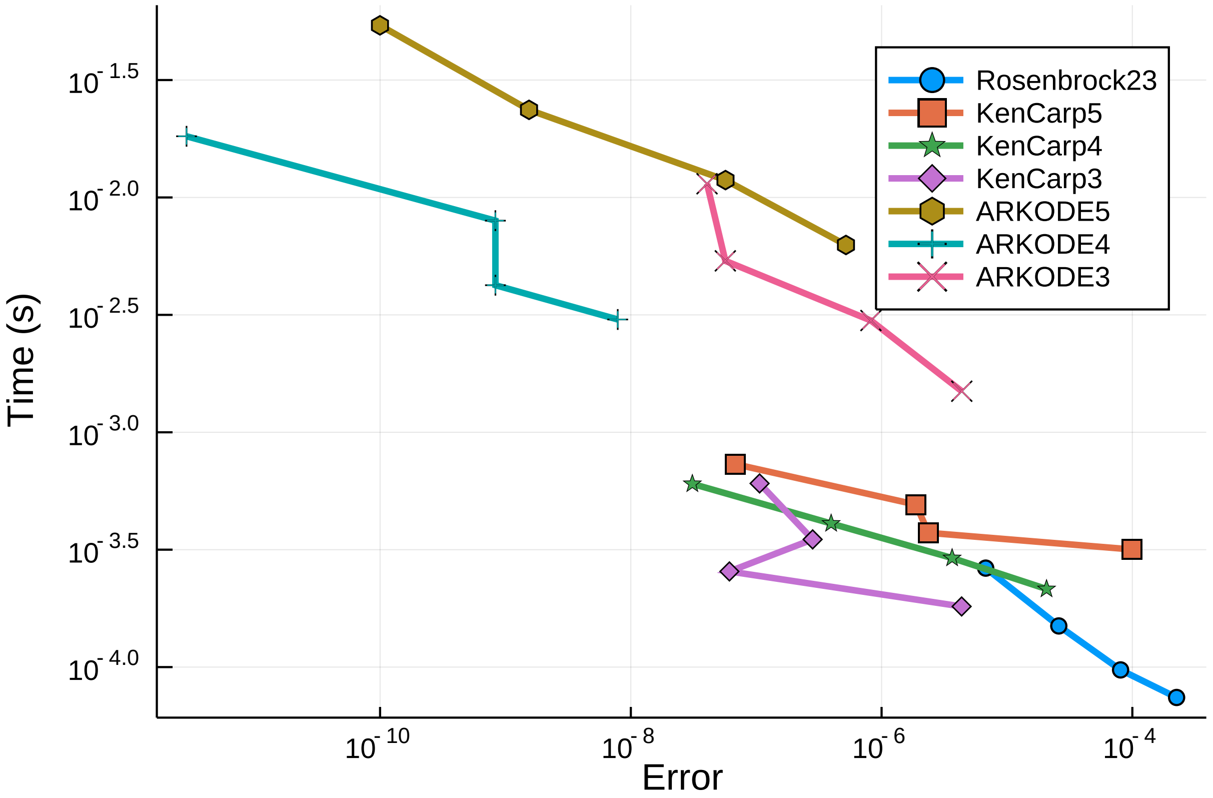
```
abstols = 1.0 ./ 10.0 .^ (5:8)
reltols = 1.0 ./ 10.0 .^ (1:4);
setups = [Dict(:alg=>Rosenbrock23()),
          Dict(:alg=>Rodas3()),
          Dict(:alg=>TRBDF2()),
          Dict(:alg=>rodas()),
          Dict(:alg=>lsoda()),
          Dict(:alg=>radau()),
          Dict(:alg=>RadauIIA5()),
          Dict(:alg=>ROS34PW1a()),
          ]
gr()
wp = WorkPrecisionSet(prob,abstols,reltols,setups;
                      save_everystep=false,appxsol=test_sol,maxiters=Int(1e5),numruns=10)
plot(wp)
```



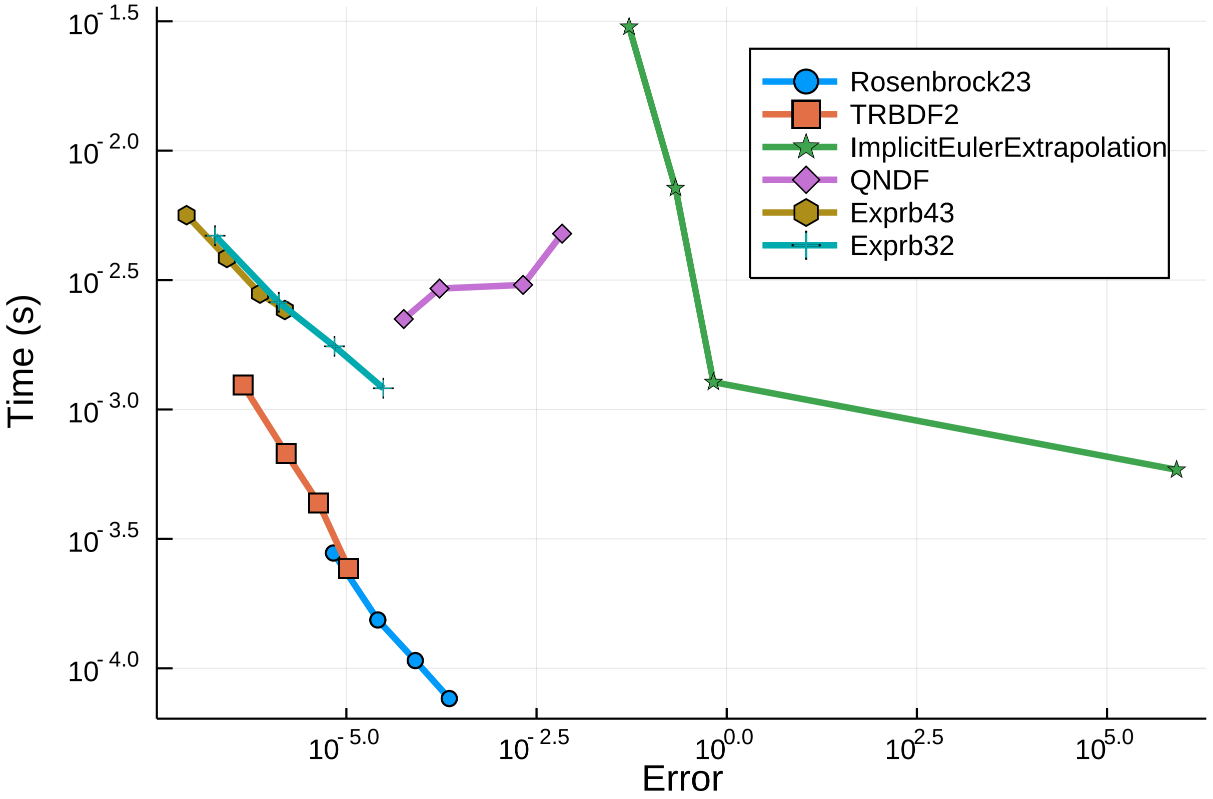
```
setups = [Dict(:alg=>Rosenbrock23()),
          Dict(:alg=>Kvaerno3()),
          Dict(:alg=>KenCarp4()),
          Dict(:alg=>TRBDF2()),
          Dict(:alg=>KenCarp3()),
          # Dict(:alg=>SDIRK2()), # Removed because it's bad
          Dict(:alg=>radau())]
names = ["Rosenbrock23" "Kvaerno3" "KenCarp4" "TRBDF2" "KenCarp3" "radau"]
wp = WorkPrecisionSet(prob,abstols,reltols,setups;names=names,
                      save_everystep=false,appxsol=test_sol,maxiters=Int(1e5),numruns=10)
plot(wp)
```



```
setups = [Dict(:alg=>Rosenbrock23()),
          Dict(:alg=>KenCarp5()),
          Dict(:alg=>KenCarp4()),
          Dict(:alg=>KenCarp3()),
          Dict(:alg=>ARKODE(nonlinear_convergence_coefficient = 1e-9,order=5)),
          Dict(:alg=>ARKODE(nonlinear_convergence_coefficient = 1e-8)),
          Dict(:alg=>ARKODE(nonlinear_convergence_coefficient = 1e-7,order=3))
]
names = ["Rosenbrock23" "KenCarp5" "KenCarp4" "KenCarp3" "ARKODE5" "ARKODE4" "ARKODE3"]
wp = WorkPrecisionSet(prob,abstols,reltols,setups;
                      names=names,
                      save_everystep=false,appxsol=test_sol,maxiters=Int(1e5),numruns=10)
plot(wp)
```

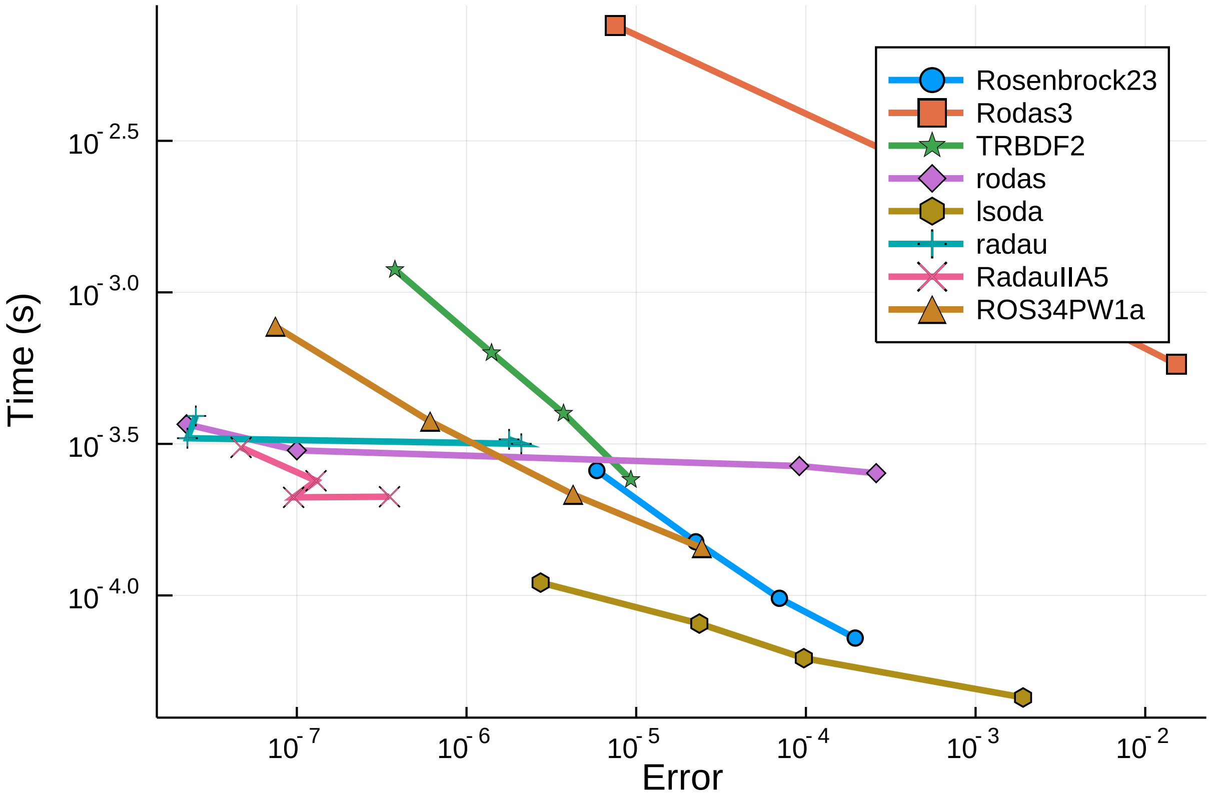


```
setups = [Dict(:alg=>Rosenbrock23()),
          Dict(:alg=>TRBDF2()),
          Dict(:alg=>ImplicitEulerExtrapolation()),
          #Dict(:alg=>ImplicitDeuflhardExtrapolation()), # Diverges
          #Dict(:alg=>ImplicitHairerWannerExtrapolation()), # Diverges
          #Dict(:alg=>ABDF2()), # Maxiters
          Dict(:alg=>QNDF()),
          Dict(:alg=>Exprb43()),
          Dict(:alg=>Exprb32()),
]
wp = WorkPrecisionSet(prob,abstols,reltols,setups;
                      save_everystep=false,appxsol=test_sol,maxiters=Int(1e5),numruns=10)
plot(wp)
```

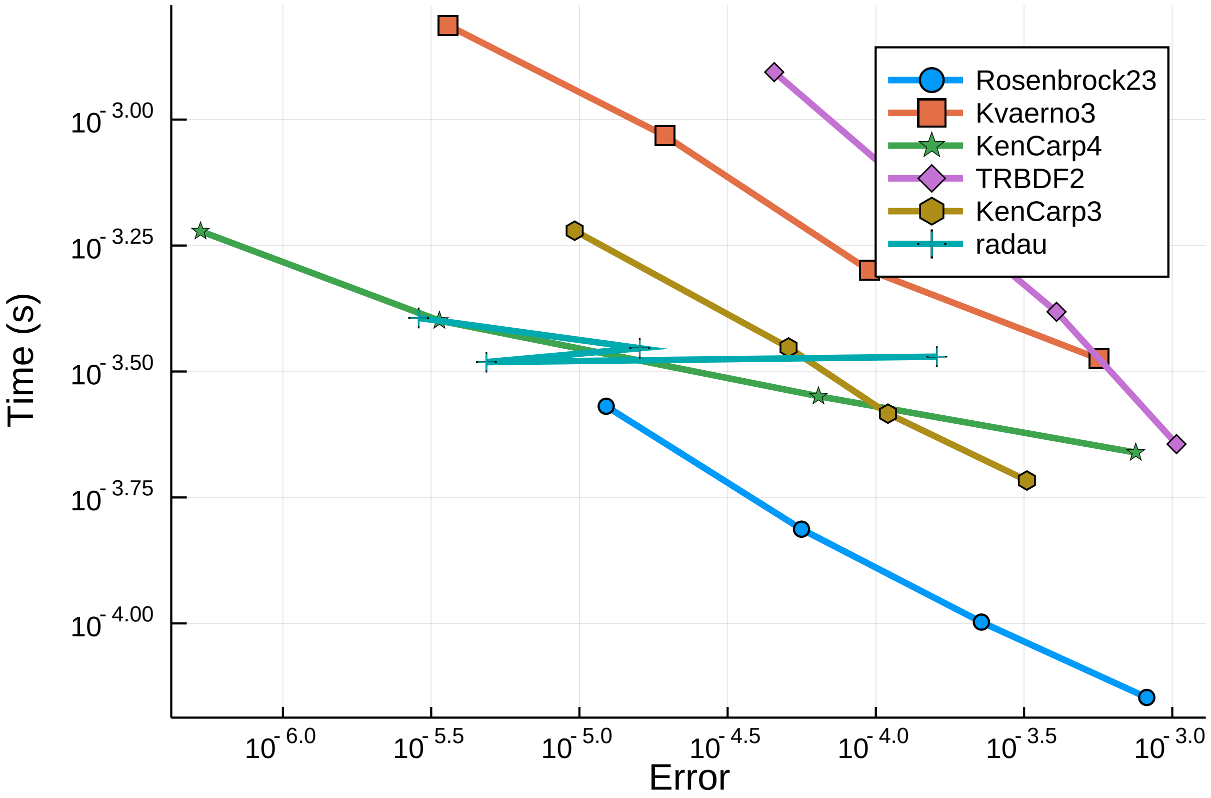


0.2.1 Timeseries Errors

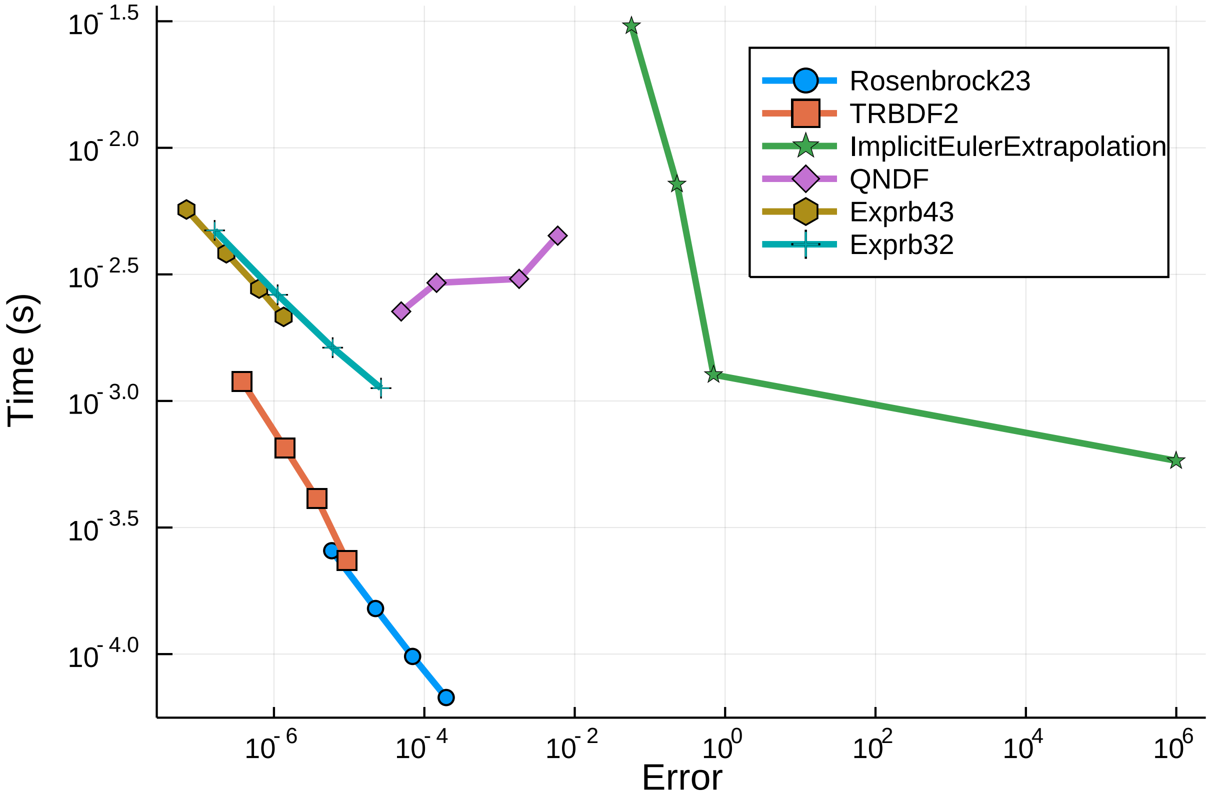
```
abstols = 1.0 ./ 10.0 .^ (5:8)
reltols = 1.0 ./ 10.0 .^ (1:4);
setups = [Dict(:alg=>Rosenbrock23()),
          Dict(:alg=>Rodas3()),
          Dict(:alg=>TRBDF2()),
          Dict(:alg=>rodas()),
          Dict(:alg=>lsoda()),
          Dict(:alg=>radau()),
          Dict(:alg=>RadauIIA5()),
          Dict(:alg=>ROS34PW1a()),
          ]
gr()
wp = WorkPrecisionSet(prob,abstols,reltols,setups;error_estimate=:l2,
                      save_everystep=false,appxsol=test_sol,maxiters=Int(1e5),numruns=10)
plot(wp)
```



```
setups = [Dict(:alg=>Rosenbrock23()),
          Dict(:alg=>Kvaerno3()),
          Dict(:alg=>KenCarp4()),
          Dict(:alg=>TRBDF2()),
          Dict(:alg=>KenCarp3()),
    # Dict(:alg=>SDIRK2()), # Removed because it's bad
          Dict(:alg=>radau())]
names = ["Rosenbrock23" "Kvaerno3" "KenCarp4" "TRBDF2" "KenCarp3" "radau"]
wp = WorkPrecisionSet(prob,abstols,reltols,setups;names=names,
                      appxsol=test_sol,maxiters=Int(1e5),error_estimate=:l2,numruns=10)
plot(wp)
```



```
setups = [Dict(:alg=>Rosenbrock23()),
          Dict(:alg=>TRBDF2()),
          Dict(:alg=>ImplicitEulerExtrapolation()),
          #Dict(:alg=>ImplicitDeuflhardExtrapolation()), # Diverges
          #Dict(:alg=>ImplicitHairerWannerExtrapolation()), # Diverges
          #Dict(:alg=>ABDF2()), # Maxiters
          Dict(:alg=>QNDF()),
          Dict(:alg=>Exprb43()),
          Dict(:alg=>Exprb32()),
]
wp = WorkPrecisionSet(prob,abstols,reltols,setups;verbose=false,error_estimate=:l2,
                      save_everystep=false,appxsol=test_sol,maxiters=Int(1e5),numruns=10)
plot(wp)
```

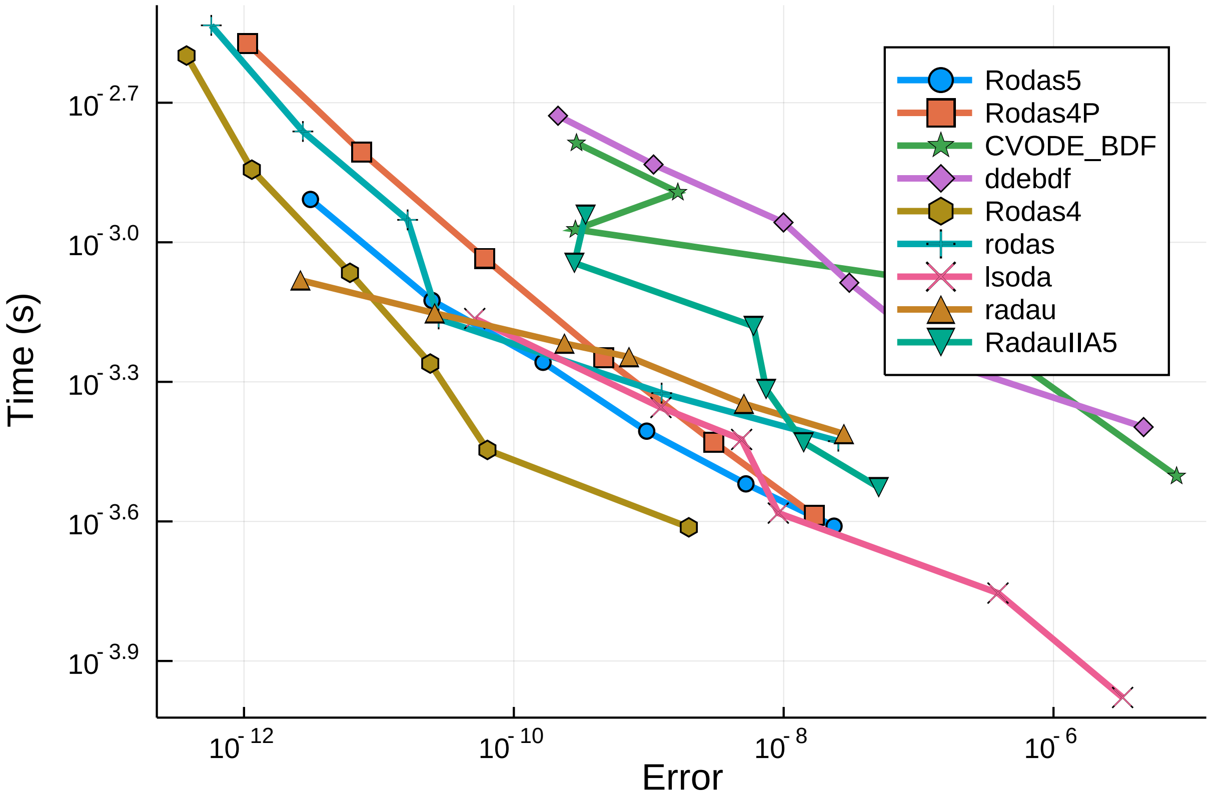


0.2.2 Low Tolerances

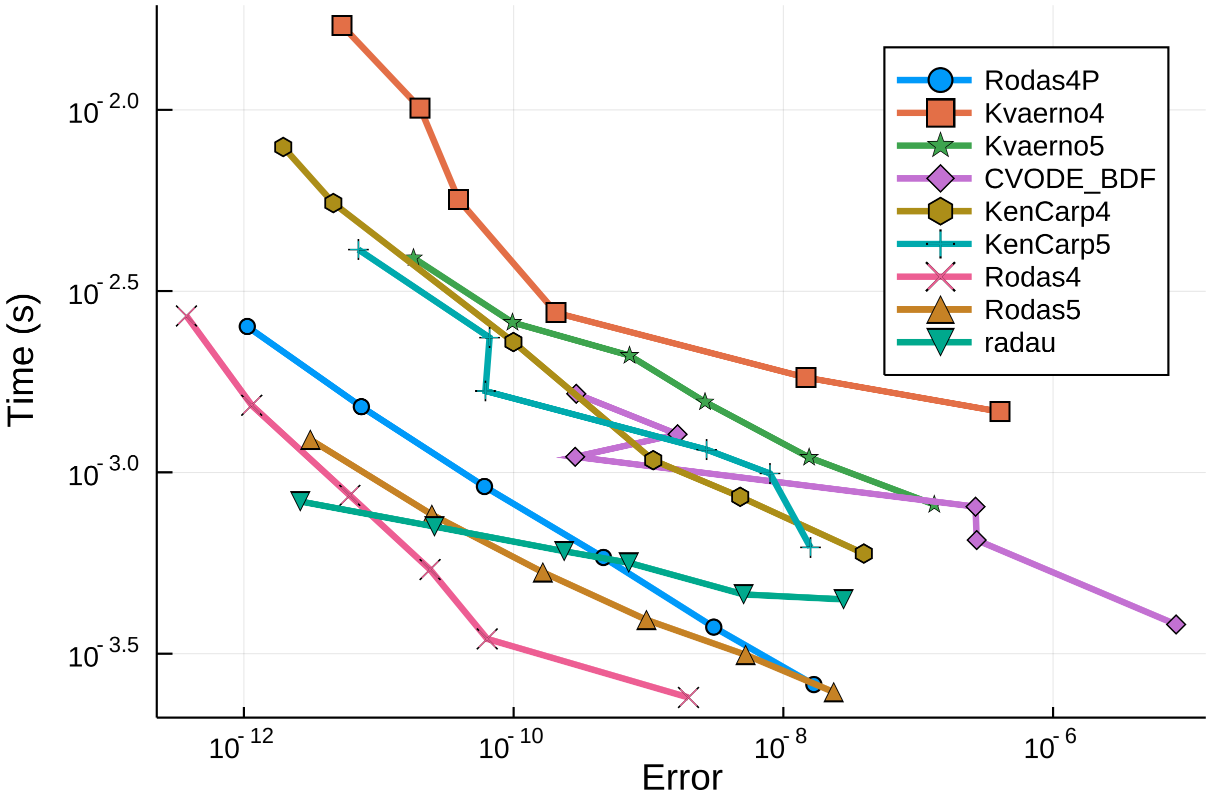
This is the speed at lower tolerances, measuring what's good when accuracy is needed.

```
abstols = 1.0 ./ 10.0 .^ (7:12)
reltols = 1.0 ./ 10.0 .^ (4:9)

setups = [Dict(:alg=>Rodas5()),
          Dict(:alg=>Rodas4P()),
          Dict(:alg=>CVODE_BDF()),
          Dict(:alg=>ddebdf()),
          Dict(:alg=>Rodas4()),
          Dict(:alg=>rodas()),
          Dict(:alg=>lsoda()),
          Dict(:alg=>radau()),
          Dict(:alg=>RadauIIA5()),
]
wp = WorkPrecisionSet(prob,abstols,reltols,setups;
                      save_everystep=false,appxsol=test_sol,maxiters=Int(1e5),numruns=10)
plot(wp)
```

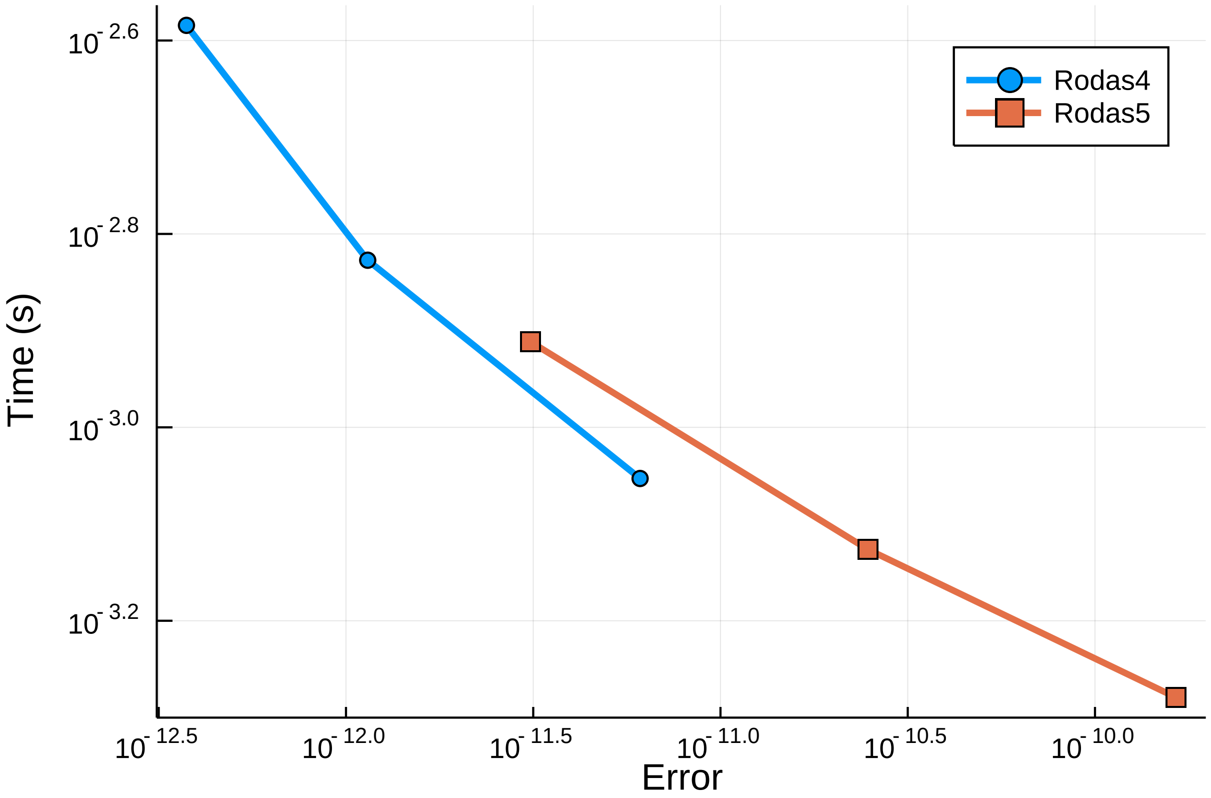


```
setups = [Dict(:alg=>Rodas4P()),
          Dict(:alg=>Kvaerno4()),
          Dict(:alg=>Kvaerno5()),
          Dict(:alg=>CVODE_BDF()),
          Dict(:alg=>KenCarp4()),
          Dict(:alg=>KenCarp5()),
          Dict(:alg=>Rodas4()),
          Dict(:alg=>Rodas5()),
          Dict(:alg=>radau())]
wp = WorkPrecisionSet(prob,abstols,reltols,setups;
                      save_everystep=false,appxsol=test_sol,maxiters=Int(1e5),numruns=10)
plot(wp)
```



```
abstols = 1.0 ./ 10.0 .^ (10:12)
reltols = 1.0 ./ 10.0 .^ (7:9)

setups = [Dict(:alg=>Rodas4())
          Dict(:alg=>Rodas5())]
names = ["Rodas4" "Rodas5"]
wp = WorkPrecisionSet(prob,abstols,reltols,setups;names=names,
                      save_everystep=false,appxsol=test_sol,maxiters=Int(1e5),numruns=10)
plot(wp)
```



0.2.3 Conclusion

At high tolerances, `Rosenbrock23` and `lsoda` hit the the error estimates and are fast. At lower tolerances and normal user tolerances, `Rodas4` and `Rodas5` are extremely fast. `lsoda` does quite well across both ends. When you get down to `reltol=1e-9` `radau` begins to become as efficient as `Rodas4`, and it continues to do well below that.

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

0.3 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("StiffODE","ROBER.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)
Environment:
  JULIA_NUM_THREADS = 16
```

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
```