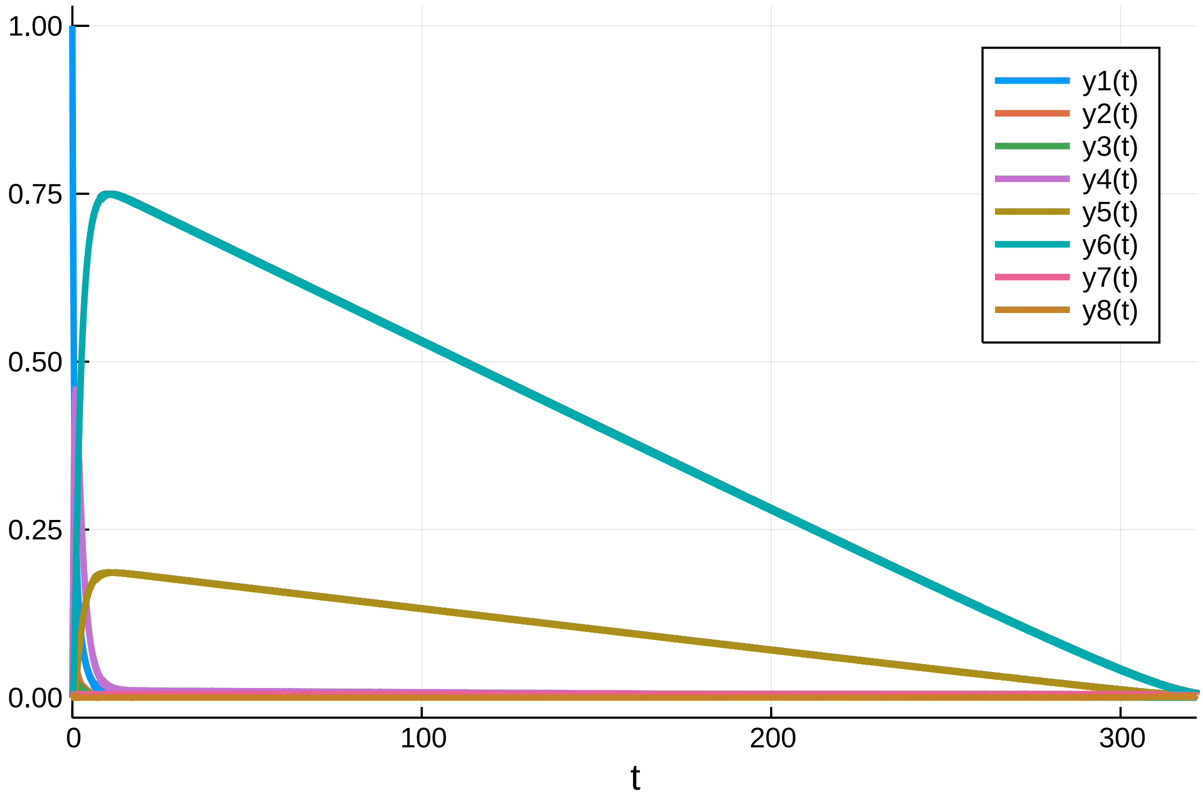
HIRES Work-Precision Diagrams

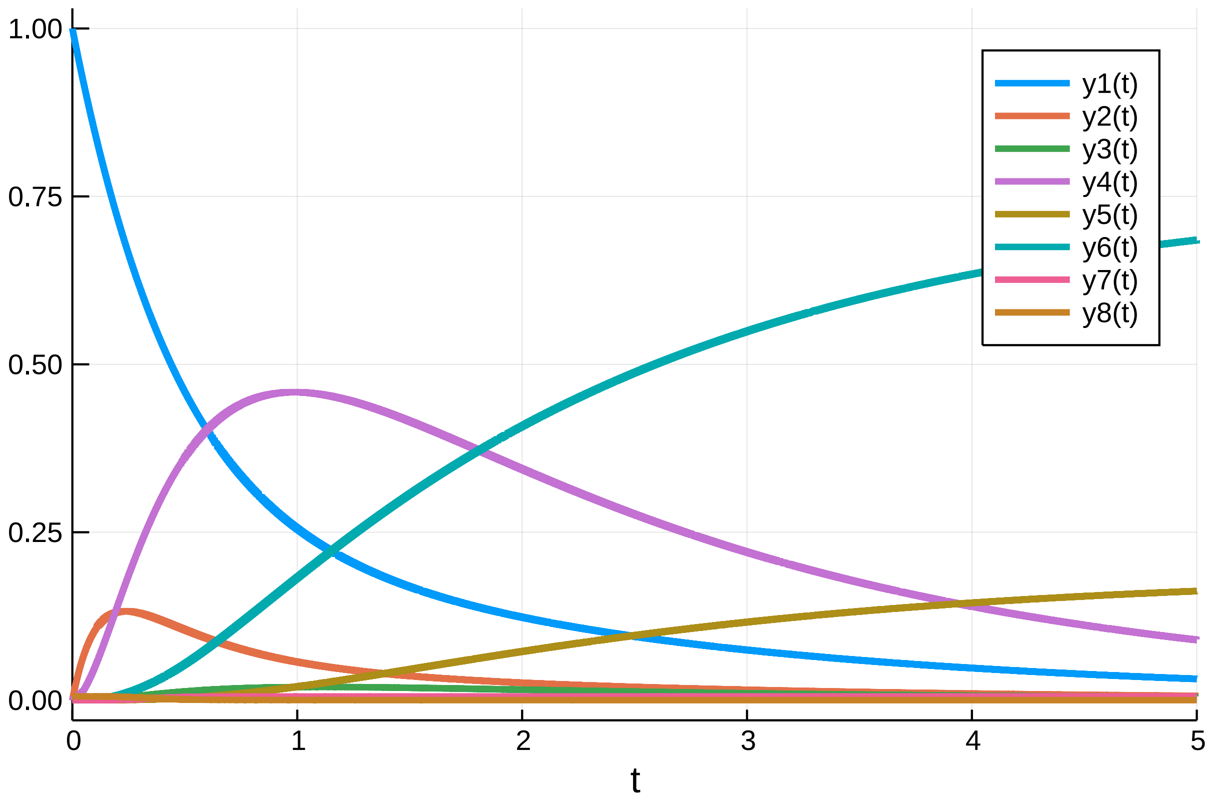
Chris Rackauckas

July 9, 2019

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

gr() #gr(fmt=:png)

f = @ode_def Hires begin
  dy1 = -1.71*y1 + 0.43*y2 + 8.32*y3 + 0.0007
  dy2 = 1.71*y1 - 8.75*y2
  dy3 = -10.03*y3 + 0.43*y4 + 0.035*y5
  dy4 = 8.32*y2 + 1.71*y3 - 1.12*y4
  dy5 = -1.745*y5 + 0.43*y6 + 0.43*y7
  dy6 = -280.0*y6*y8 + 0.69*y4 + 1.71*y5 -
           0.43*y6 + 0.69*y7
  dy7 = 280.0*y6*y8 - 1.81*y7
  dy8 = -280.0*y6*y8 + 1.81*y7
end

u0 = zeros(8)
u0[1] = 1
u0[8] = 0.0057
prob = ODEProblem(f,u0,(0.0,321.8122))

sol = solve(prob,Rodas5(),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)
```




```
plot(sol,tspan=(0.0,5.0))
```



0.1 Omissions

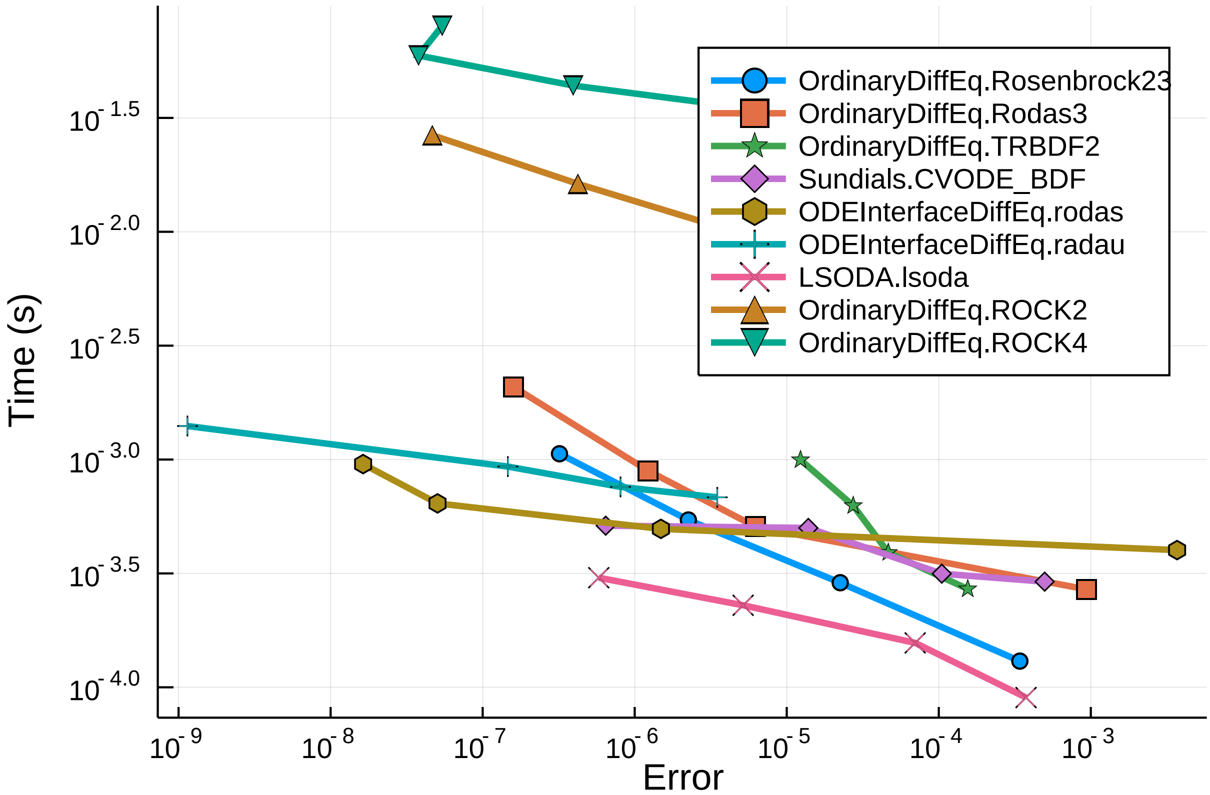
The following were omitted from the tests due to convergence failures. ODE.jl's adaptivity is not able to stabilize its algorithms, while GeometricIntegratorsDiffEq has not upgraded to Julia 1.0. GeometricIntegrators.jl's methods used to be either fail to converge at comparable dts (or on some computers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

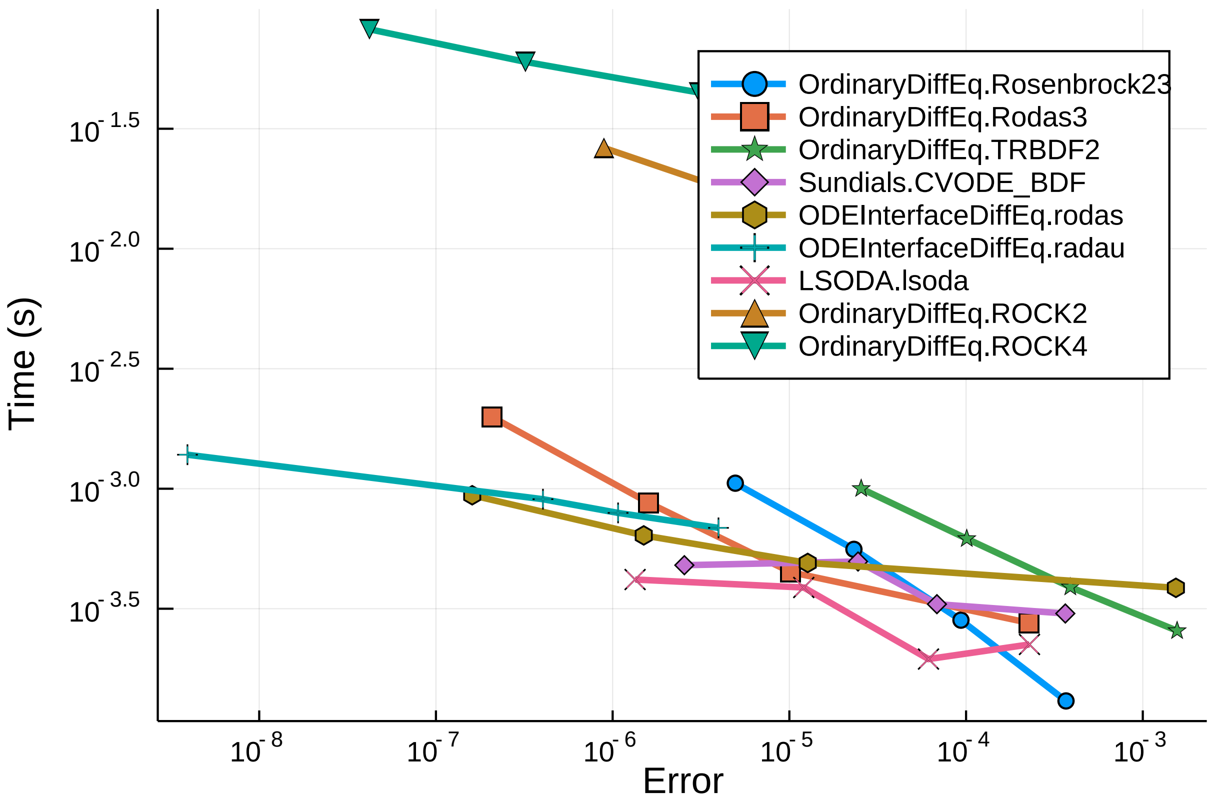
0.2 High Tolerances

This is the speed when you just want the answer.

```
solve(prob, ddebdf())
solve(prob, rodas())
solve(prob, radau())
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=>CVODE_BDF()),
          Dict(:alg=>rodas()),
          Dict(:alg=>radau()),
          Dict(:alg=>lsoda()),
          Dict(:alg=>ROCK2()),
          Dict(:alg=>ROCK4())
          ]
wp = WorkPrecisionSet(prob,abstols,reltols,setups;
                      save_everystep=false,appxsol=test_sol,maxiters=Int(1e5),numruns=10)
plot(wp)
```

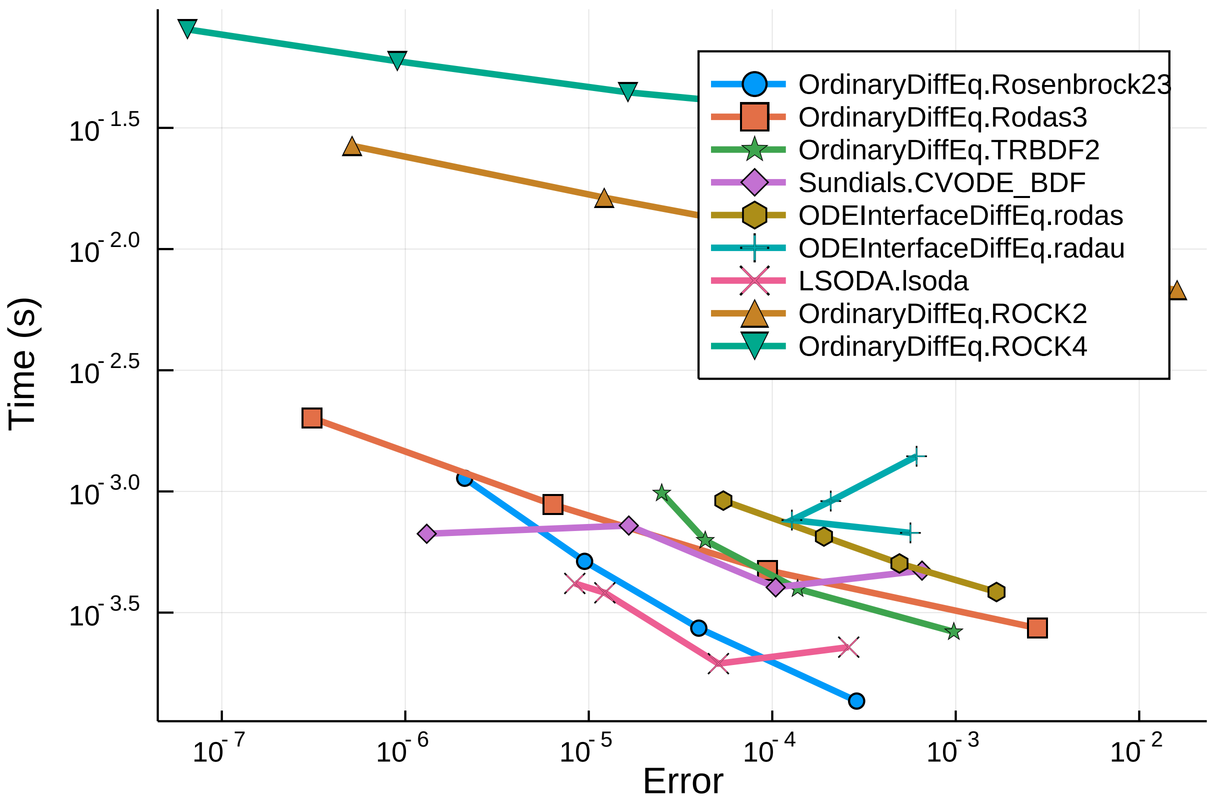



```
wp = WorkPrecisionSet(prob,abstols,reltols,setups;dense = false,verbose=false,
                      appxsol=test_sol,maxiters=Int(1e5),error_estimate=:l2)
plot(wp)
```

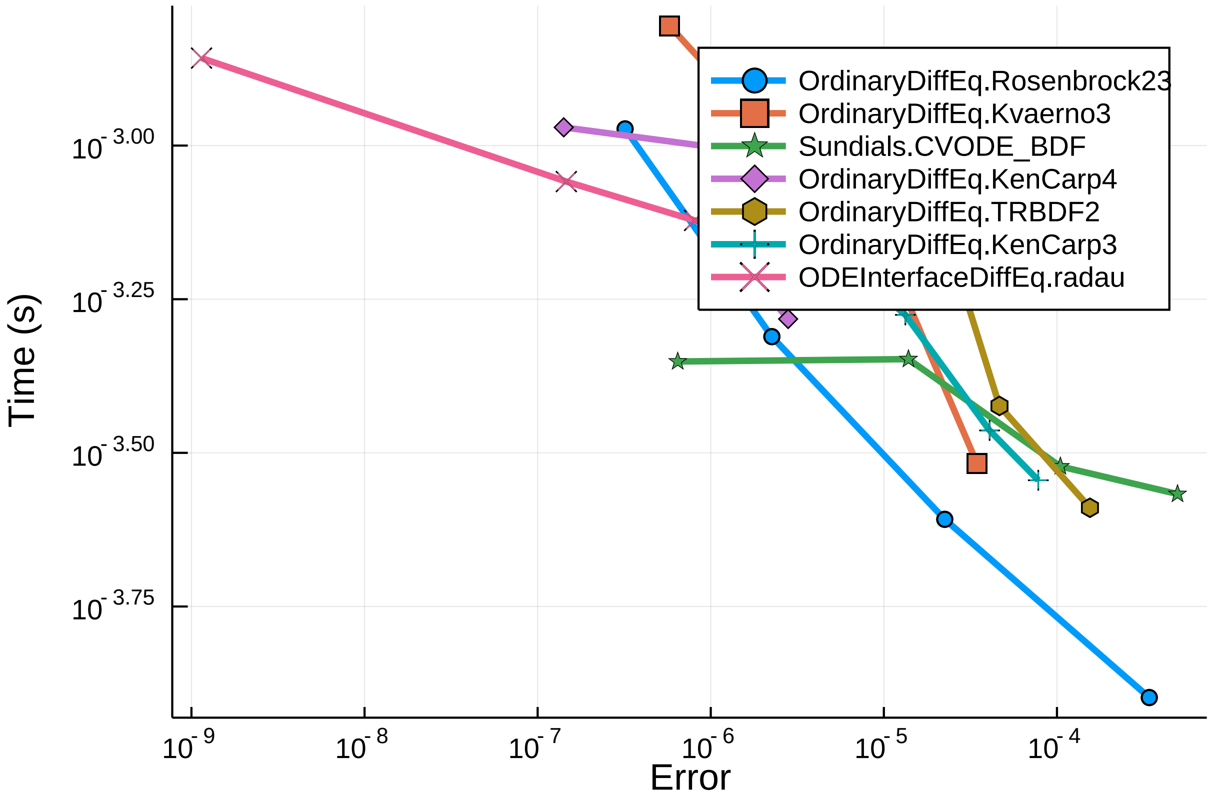



```
wp = WorkPrecisionSet(prob,abstols,reltols,setups;
                      appxsol=test_sol,maxiters=Int(1e5),error_estimate=:L2)
```

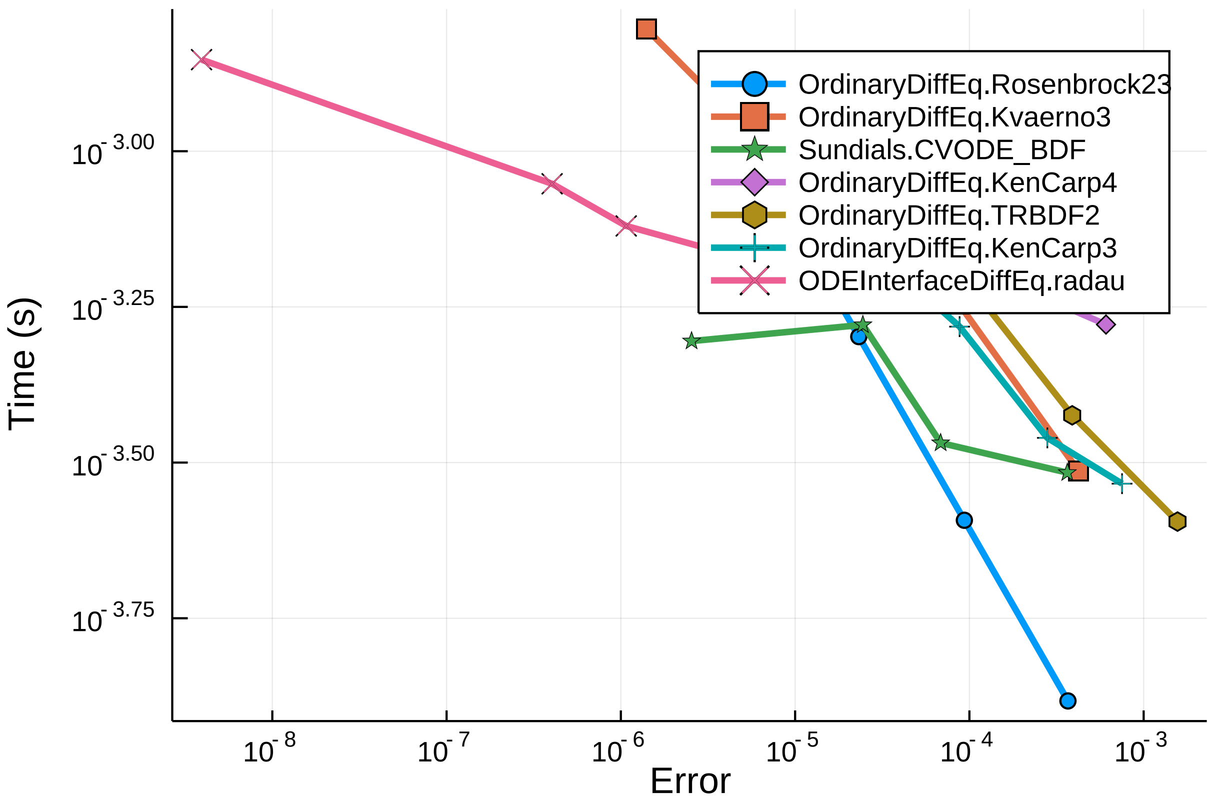
```
plot(wp)
```




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

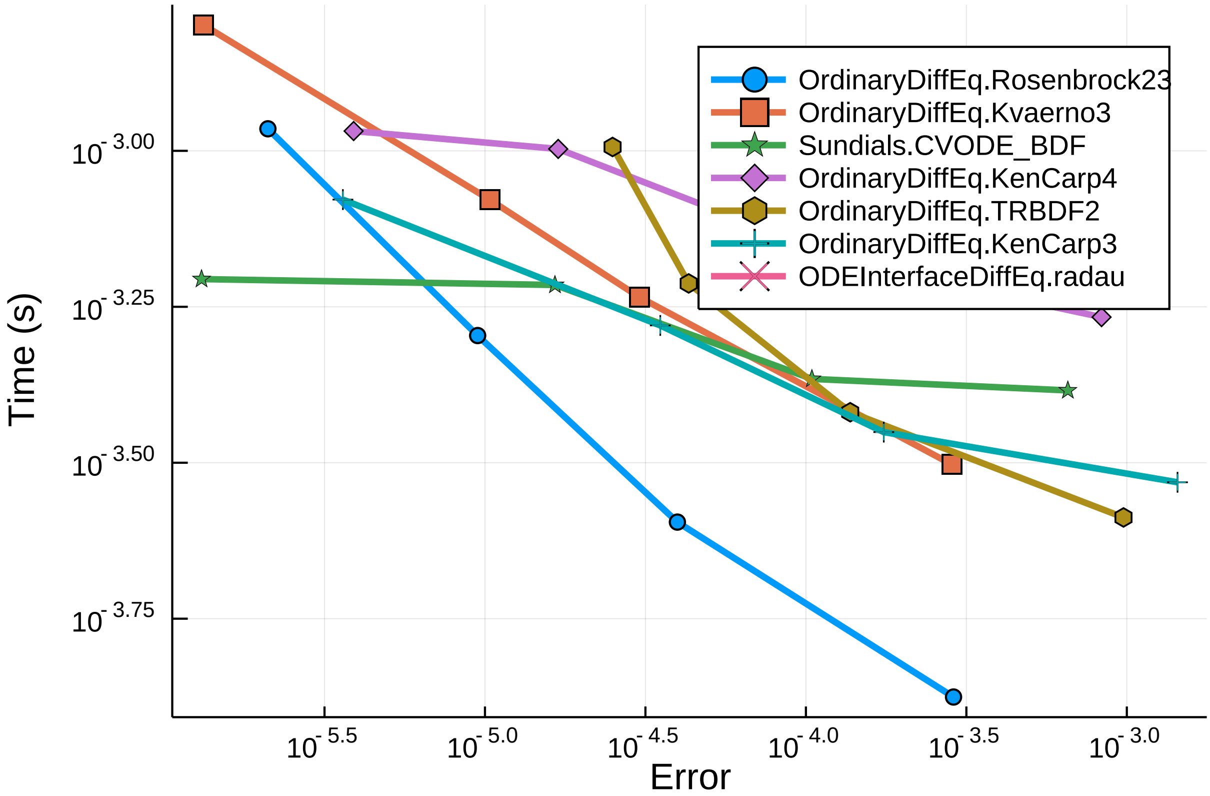



```
wp = WorkPrecisionSet(prob,abstols,reltols,setups;dense = false,verbose=false,
                      appxsol=test_sol,maxiters=Int(1e5),error_estimate=:l2)
plot(wp)
```

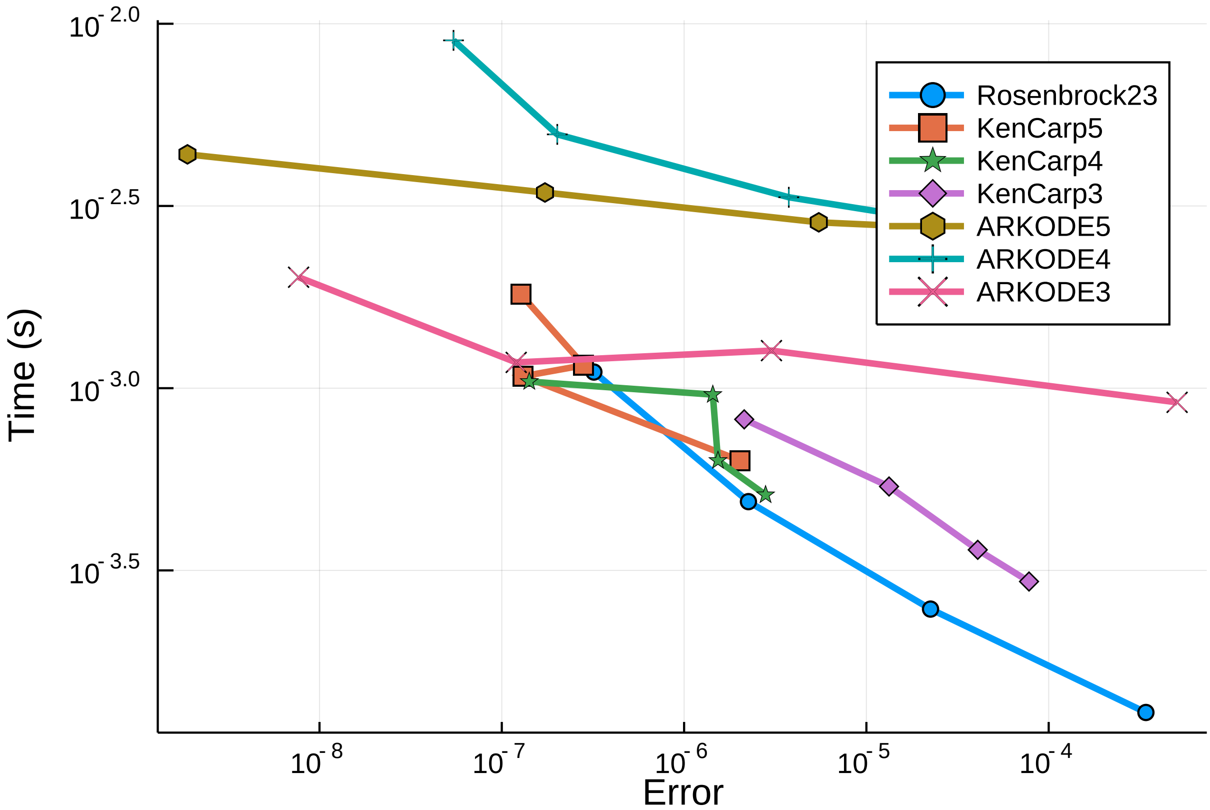



```
wp = WorkPrecisionSet(prob,abstols,reltols,setups;
                      appxsol=test_sol,maxiters=Int(1e5),error_estimate=:L2)
```

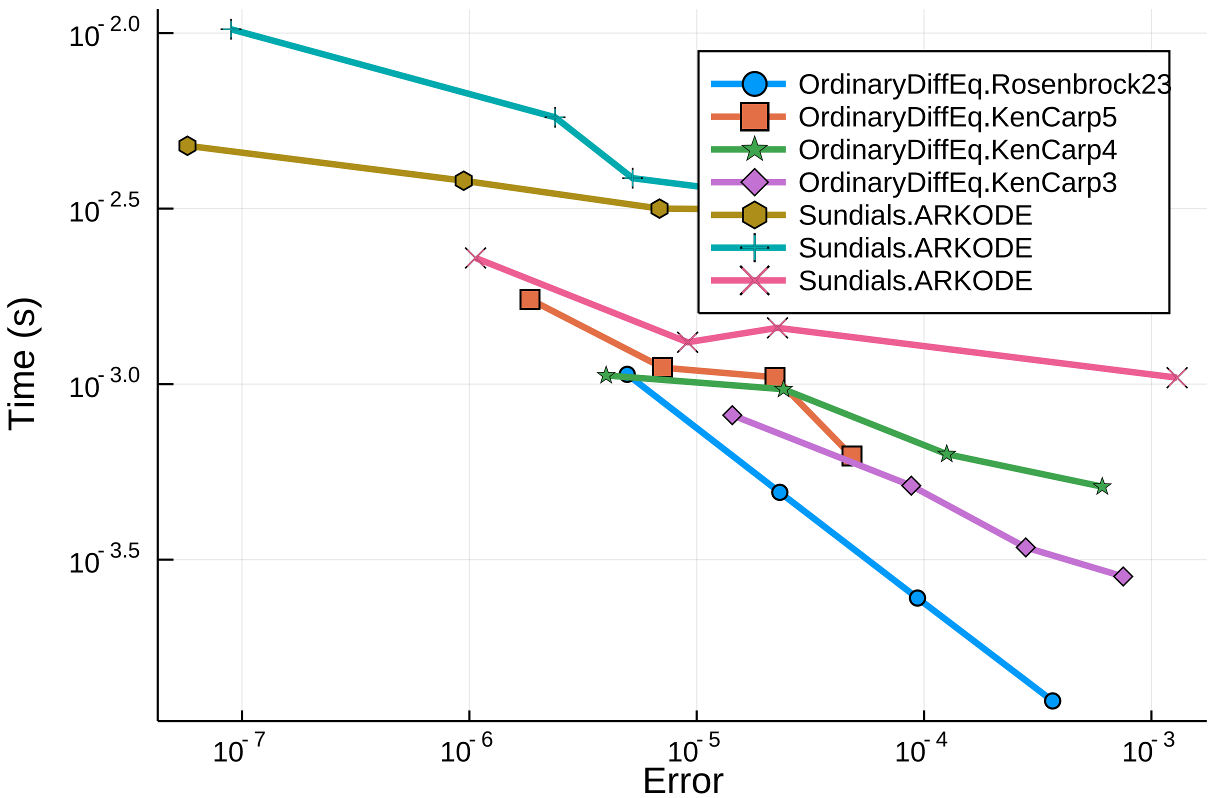
```
plot(wp)
```




```
setups = [Dict(:alg=>Rosenbrock23()),
          Dict(:alg=>KenCarp5()),
          Dict(:alg=>KenCarp4()),
          Dict(:alg=>KenCarp3()),
          Dict(:alg=>ARKODE(order=5)),
          Dict(:alg=>ARKODE()),
          Dict(:alg=>ARKODE(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))
plot(wp)
```



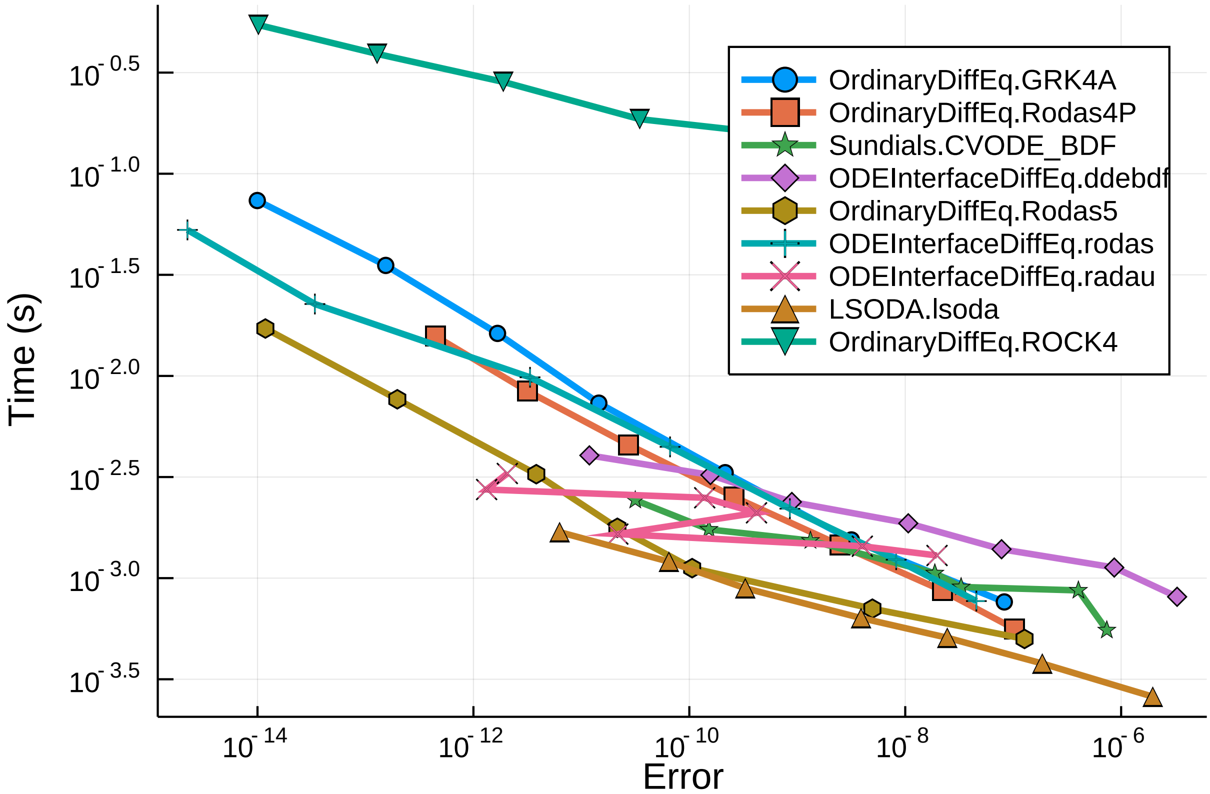

```
wp = WorkPrecisionSet(prob,abstols,reltols,setups;dense = false,verbose=false,
                      appxsol=test_sol,maxiters=Int(1e5),error_estimate=:l2)
plot(wp)
```



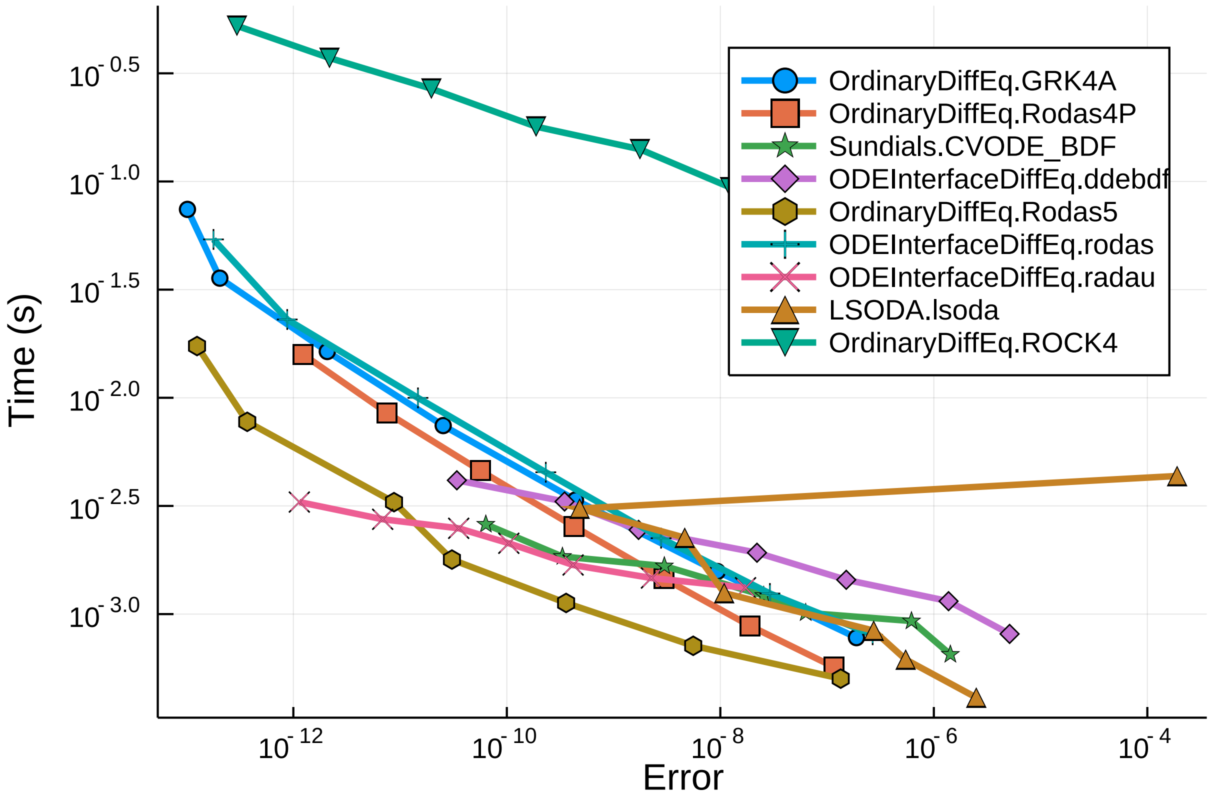
0.2.1 Low Tolerances

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

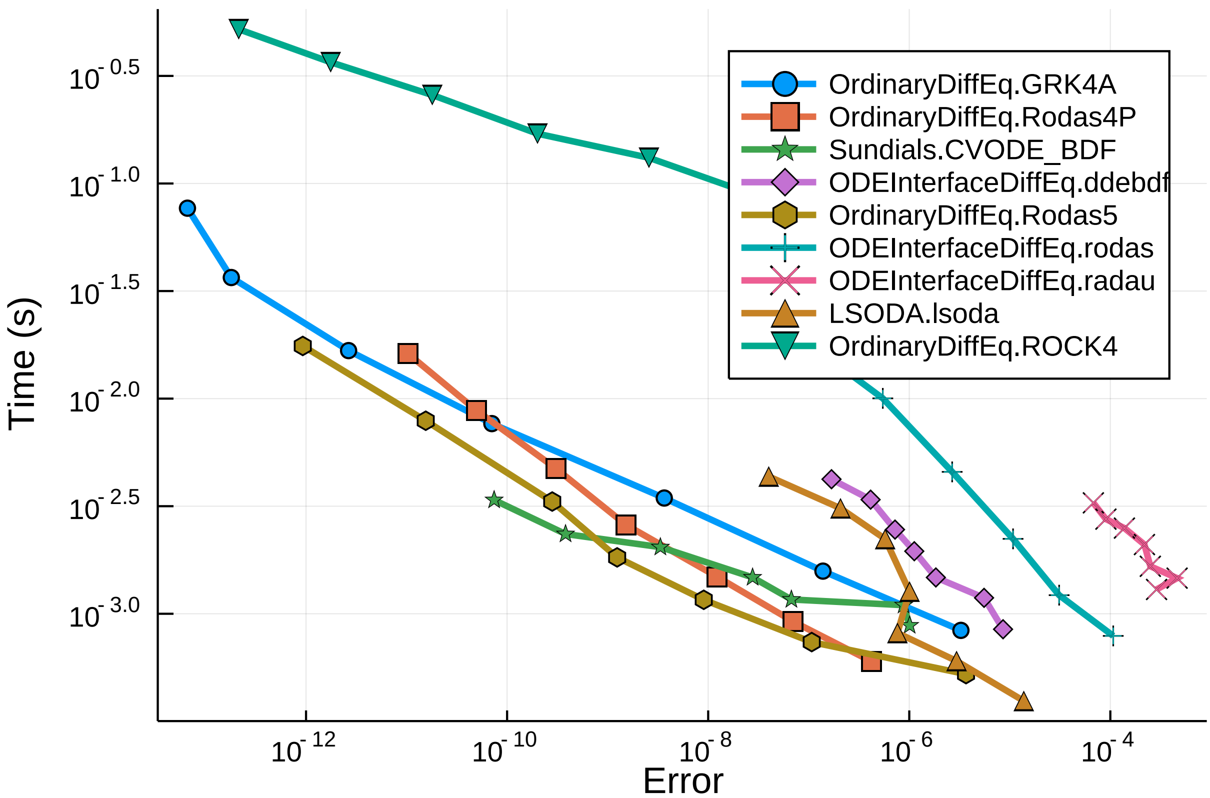
```
abstols = 1.0 ./ 10.0 .^ (7:13)
reltols = 1.0 ./ 10.0 .^ (4:10)

setups = [Dict(:alg=>GRK4A()),
          Dict(:alg=>Rodas4P()),
          Dict(:alg=>CVODE_BDF()),
          Dict(:alg=>ddebdf()),
          Dict(:alg=>Rodas5()),
          Dict(:alg=>rodas()),
          Dict(:alg=>radau()),
          Dict(:alg=>lsoda()),
          #Dict(:alg=>ROCK2()), #Needs more iterations
          Dict(:alg=>ROCK4())
]
wp = WorkPrecisionSet(prob,abstols,reltols,setups;
                      save_everystep=false,appxsol=test_sol,maxiters=Int(1e5))
plot(wp)
```




```
wp = WorkPrecisionSet(prob,abstols,reltols,setups;verbose=false,
                      dense=false,appxsol=test_sol,maxiters=Int(1e5),error_estimate=:l2)
plot(wp)
```

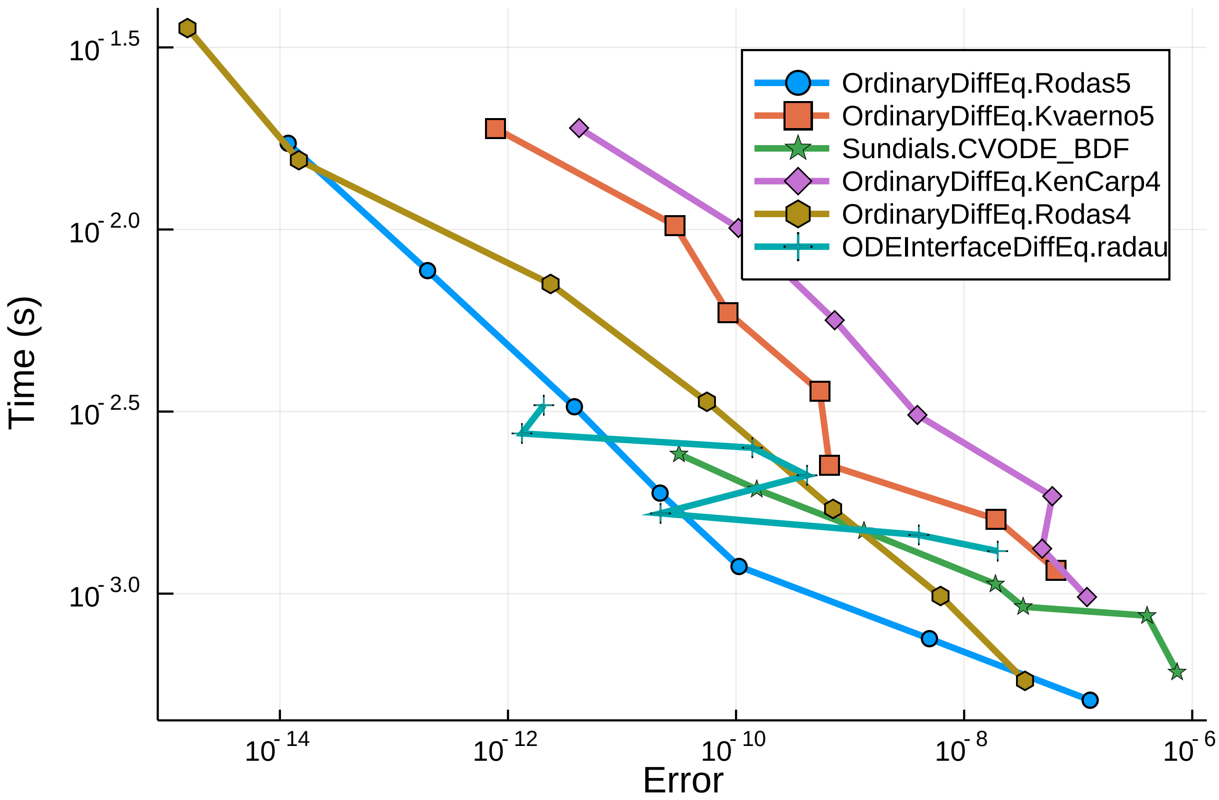



```
wp = WorkPrecisionSet(prob,abstols,reltols,setups;
                      appxsol=test_sol,maxiters=Int(1e5),error_estimate=:L2)
plot(wp)
```

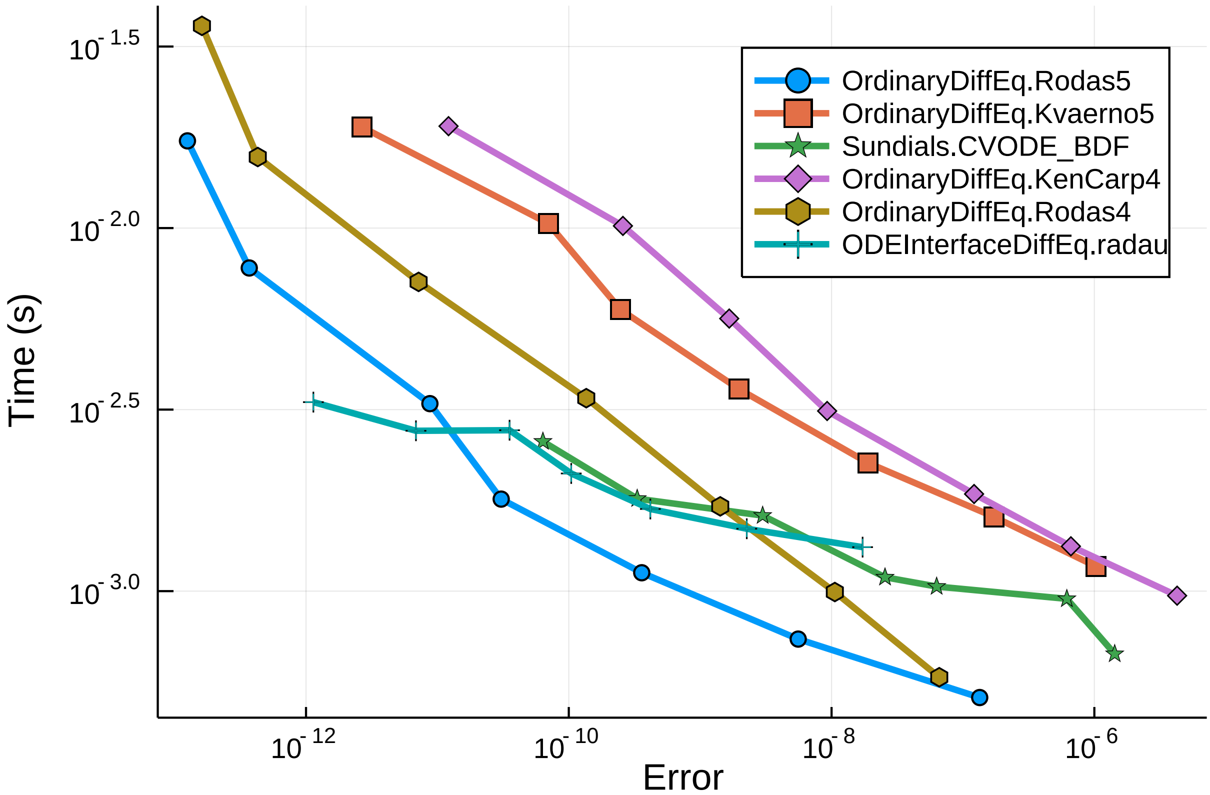



```
setups = [
          Dict(:alg=>Rodas5()),
```

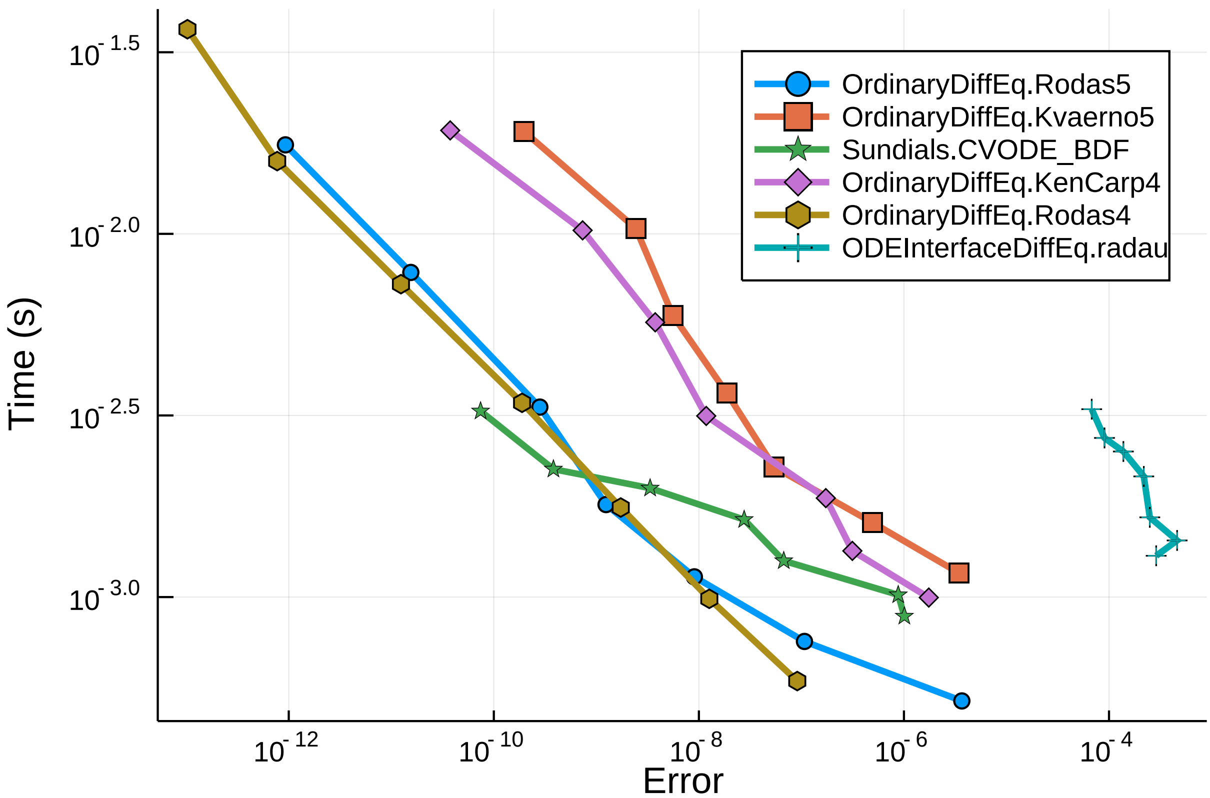
```
        Dict(:alg=>Kvaerno5()),
        Dict(:alg=>CVODE_BDF()),
        Dict(:alg=>KenCarp4()),
        Dict(:alg=>Rodas4()),
        Dict(:alg=>radau())]
wp = WorkPrecisionSet(prob,abstols,reltols,setups;
                      save_everystep=false,appxsol=test_sol,maxiters=Int(1e5))
plot(wp)
```




```
wp = WorkPrecisionSet(prob,abstols,reltols,setups;verbose=false,
                      dense=false,appxsol=test_sol,maxiters=Int(1e5),error_estimate=:l2)
plot(wp)
```

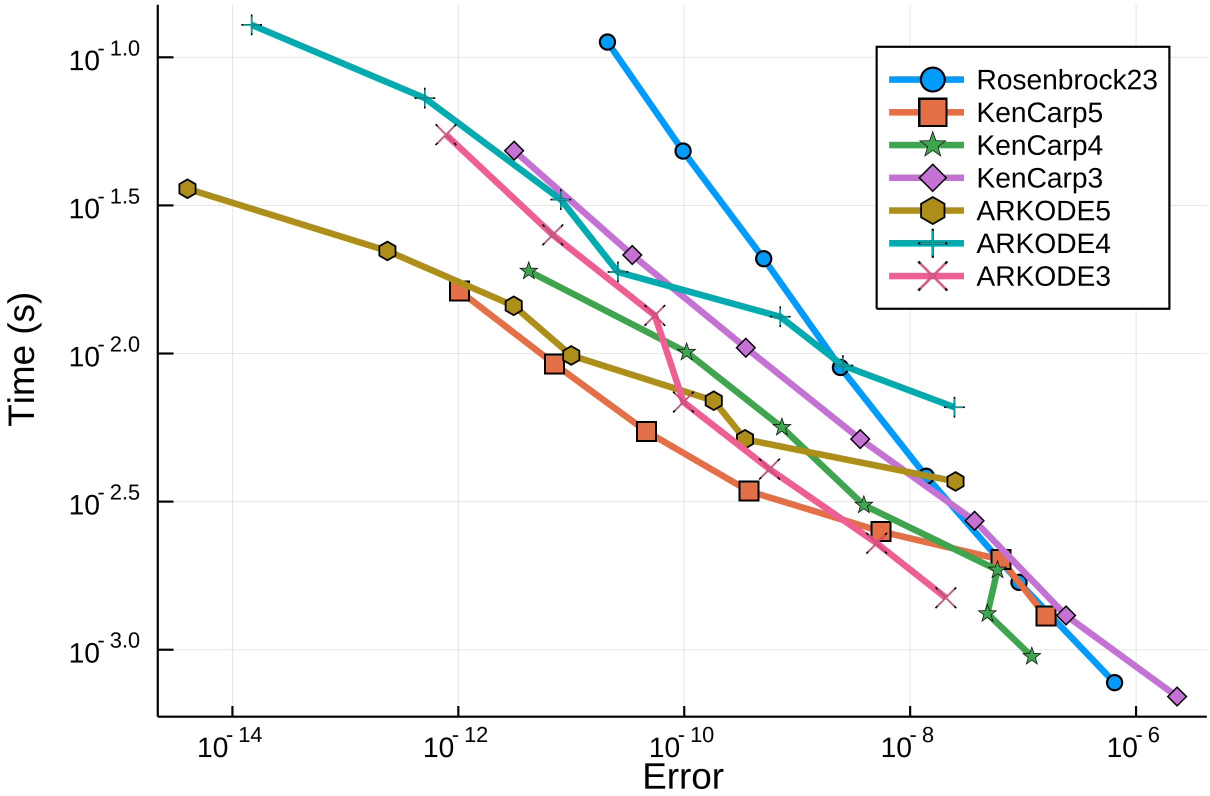



```
wp = WorkPrecisionSet(prob,abstols,reltols,setups;
                      appxsol=test_sol,maxiters=Int(1e5),error_estimate=:L2)
plot(wp)
```

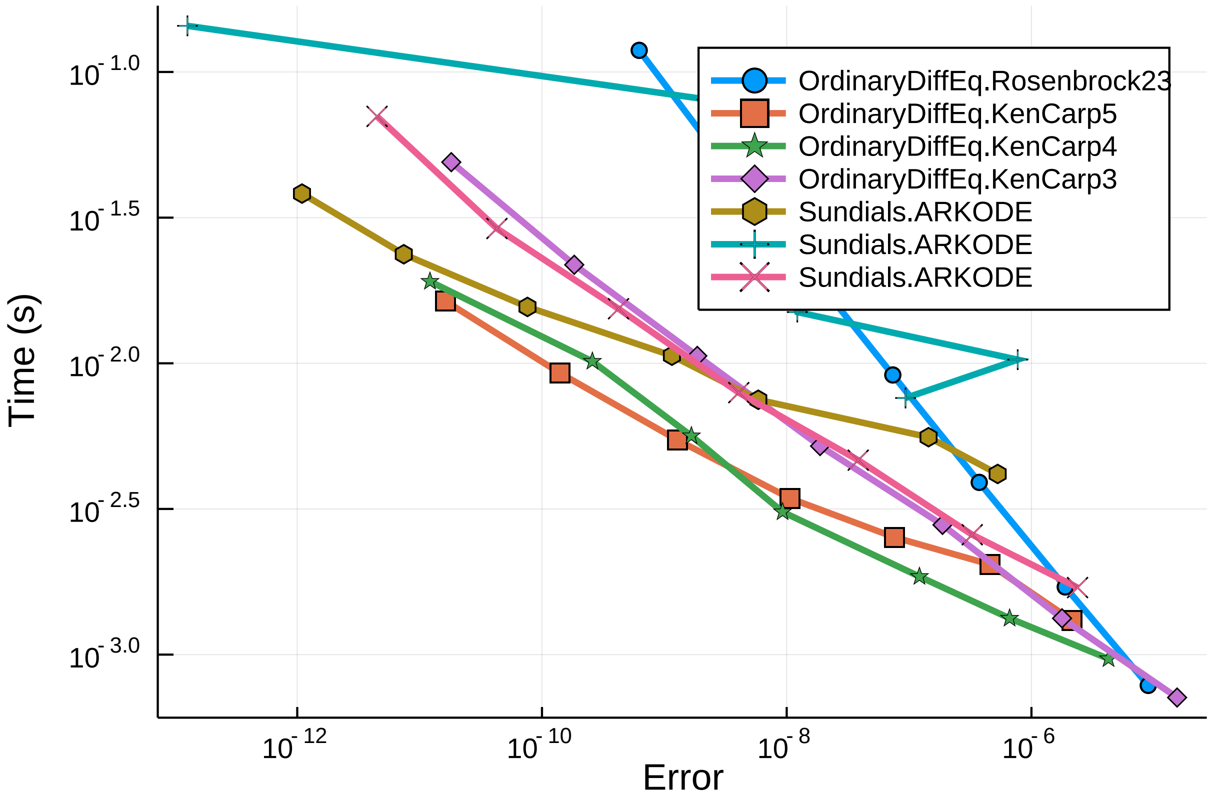



```
setups = [Dict(:alg=>Rosenbrock23()),
          Dict(:alg=>KenCarp5()),
```

```
          Dict(:alg=>KenCarp4()),
          Dict(:alg=>KenCarp3()),
          Dict(:alg=>ARKODE(order=5)),
          Dict(:alg=>ARKODE()),
          Dict(:alg=>ARKODE(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))
plot(wp)
```




```
wp = WorkPrecisionSet(prob,abstols,reltols,setups;verbose=false,
                      dense=false,appxsol=test_sol,maxiters=Int(1e5),error_estimate=:l2)
plot(wp)
```



The following algorithms were removed since they failed.

```
#setups = [#Dict(:alg=>Hairer4()),
          #Dict(:alg=>Hairer42()),
          #Dict(:alg=>Rodas3()),
          #Dict(:alg=>Kvaerno4()),
#Dict(:alg=>KenCarp5()),
          #Dict(:alg=>Cash4())
#]
#wp = WorkPrecisionSet(prob,abstols,reltols,setups;
# save_everystep=false,appxsol=test_sol,maxiters=Int(1e5))
#plot(wp)
```

0.2.2 Conclusion

At high tolerances, `Rosenbrock23` and `lsoda` hitsthe the error estimates and are fast. At lower tolerances and normal user tolerances, `Rodas5` is extremely fast. When you get down to `reltol=1e-10` `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","Hires.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/crackauckas/.julia/environments/v1.1/Project.toml`
[c52e3926-4ff0-5f6e-af25-54175e0327b1] Atom 0.8.7
[bcd4f6db-9728-5f36-b5f7-82caef46ccdb] DelayDiffEq 5.4.1
[bb2cbb15-79fc-5d1e-9bf1-8ae49c7c1650] DiffEqBenchmarks 0.1.0
[459566f4-90b8-5000-8ac3-15dfb0a30def] DiffEqCallbacks 2.5.2
[f3b72e0c-5b89-59e1-b016-84e28bfd966d] DiffEqDevTools 2.8.0
[aae7a2af-3d4f-5e19-a356-7da93b79d9d0] DiffEqFlux 0.5.0
[78ddff82-25fc-5f2b-89aa-309469cbf16f] DiffEqMonteCarlo 0.15.1
[77a26b50-5914-5dd7-bc55-306e6241c503] DiffEqNoiseProcess 3.3.1
[9fdde737-9c7f-55bf-ade8-46b3f136cc48] DiffEqOperators 3.5.0
[055956cb-9e8b-5191-98cc-73ae4a59e68a] DiffEqPhysics 3.1.0
[a077e3f3-b75c-5d7f-a0c6-6bc4c8ec64a9] DiffEqProblemLibrary 4.1.0
[0c46a032-eb83-5123-abaf-570d42b7fbaa] DifferentialEquations 6.4.0
[b305315f-e792-5b7a-8f41-49f472929428] Elliptic 0.5.0
[587475ba-b771-5e3f-ad9e-33799f191a9c] Flux 0.8.3
[e5e0dc1b-0480-54bc-9374-aad01c23163d] Juno 0.7.0
[7f56f5a3-f504-529b-bc02-0b1fe5e64312] LSODA 0.4.0
[c030b06c-0b6d-57c2-b091-7029874bd033] ODE 2.4.0
[54ca160b-1b9f-5127-a996-1867f4bc2a2c] ODEInterface 0.4.5
[09606e27-ecf5-54fc-bb29-004bd9f985bf] ODEInterfaceDiffEq 3.3.1
[1dea7af3-3e70-54e6-95c3-0bf5283fa5ed] OrdinaryDiffEq 5.8.1
[2dcacdae-9679-587a-88bb-8b444fb7085b] ParallelDataTransfer 0.5.0
[65888b18-ceab-5e60-b2b9-181511a3b968] ParameterizedFunctions 4.1.1
[91a5bcdd-55d7-5caf-9e0b-520d859cae80] Plots 0.25.2
[d330b81b-6aea-500a-939a-2ce795aea3ee] PyPlot 2.8.1
[731186ca-8d62-57ce-b412-fbd966d074cd] RecursiveArrayTools 0.20.0
[295af30f-e4ad-537b-8983-00126c2a3abe] Revise 2.1.6
[90137ffa-7385-5640-81b9-e52037218182] StaticArrays 0.11.0
[789caeaf-c7a9-5a7d-9973-96adeb23e2a0] StochasticDiffEq 6.2.0
[c3572dad-4567-51f8-b174-8c6c989267f4] Sundials 3.6.1
[92b13dbe-c966-51a2-8445-caca9f8a7d42] TaylorIntegration 0.5.0
[44d3d7a6-8a23-5bf8-98c5-b353f8df5ec9] Weave 0.9.0
[e88e6eb3-aa80-5325-afca-941959d7151f] Zygote 0.3.2
```