POLLU Work-Precision Diagrams

Chris Rackauckas

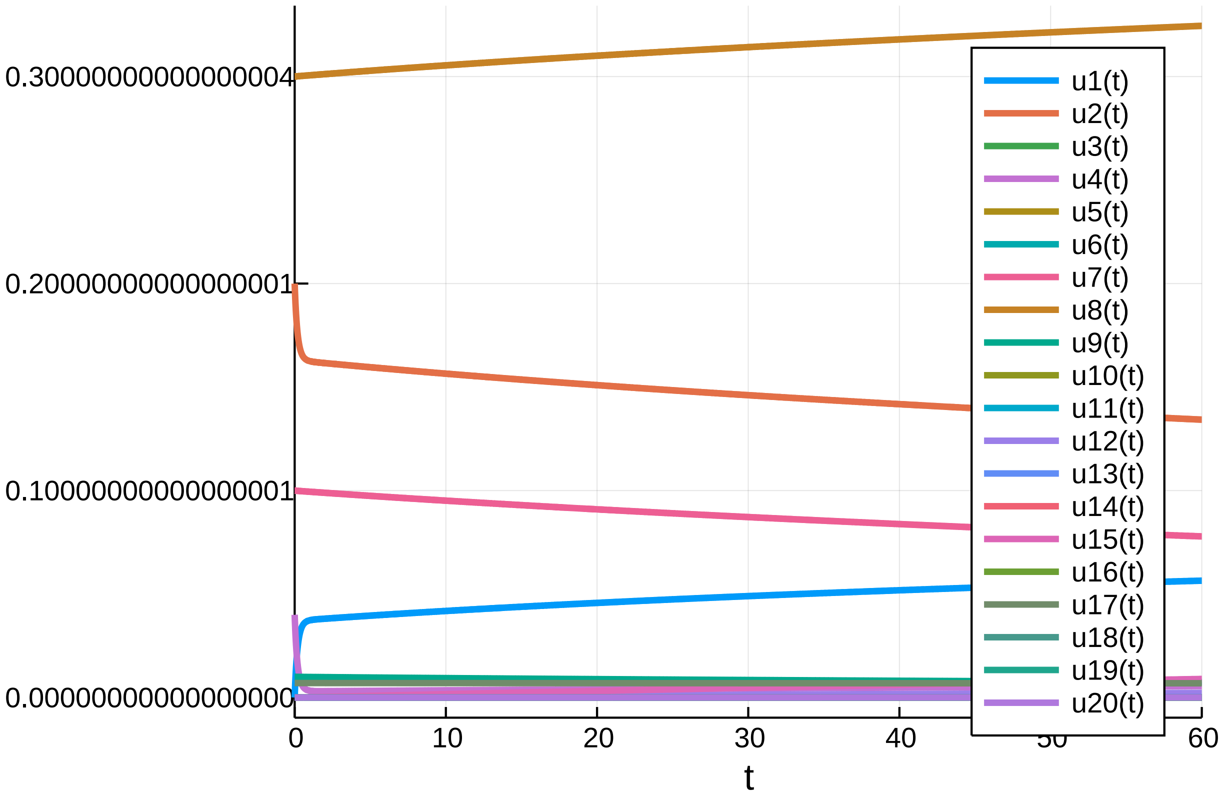
July 9, 2019

```
using OrdinaryDiffEq, DiffEqDevTools, Sundials, ParameterizedFunctions, Plots, ODE,
    ODEInterfaceDiffEq, LSODA
gr() # gr(fmt=:png)
using LinearAlgebra
LinearAlgebra.BLAS.set_num_threads(1)

const k1=.35e0
const k2=.266e2
const k3=.123e5
const k4=.86e-3
const k5=.82e-3
const k6=.15e5
const k7=.13e-3
const k8=.24e5
const k9=.165e5
const k10=.9e4
const k11=.22e-1
const k12=.12e5
const k13=.188e1
const k14=.163e5
const k15=.48e7
const k16=.35e-3
const k17=.175e-1
const k18=.1e9
const k19=.444e12
const k20=.124e4
const k21=.21e1
const k22=.578e1
const k23=.474e-1
const k24=.178e4
const k25=.312e1

function f(dy,y,p,t)
 r1  = k1 *y[1]
 r2  = k2 *y[2]*y[4]
 r3  = k3 *y[5]*y[2]
 r4  = k4 *y[7]
 r5  = k5 *y[7]
 r6  = k6 *y[7]*y[6]
 r7  = k7 *y[9]
 r8  = k8 *y[9]*y[6]
 r9  = k9 *y[11]*y[2]
 r10 = k10*y[11]*y[1]
 r11 = k11*y[13]
 r12 = k12*y[10]*y[2]
```

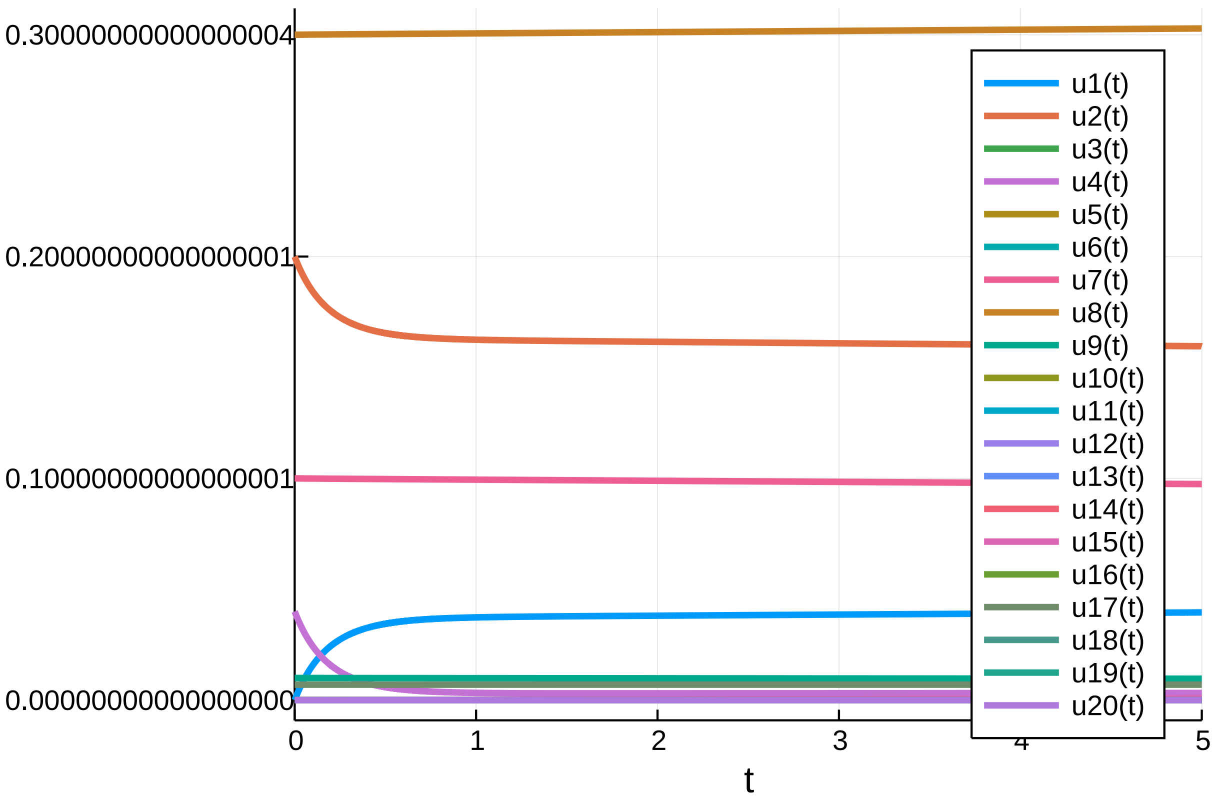
```
 r13 = k13*y[14]
 r14 = k14*y[1]*y[6]
 r15 = k15*y[3]
 r16 = k16*y[4]
 r17 = k17*y[4]
 r18 = k18*y[16]
 r19 = k19*y[16]
 r20 = k20*y[17]*y[6]
 r21 = k21*y[19]
 r22 = k22*y[19]
 r23 = k23*y[1]*y[4]
 r24 = k24*y[19]*y[1]
 r25 = k25*y[20]

 dy[1]  = -r1-r10-r14-r23-r24+
          r2+r3+r9+r11+r12+r22+r25
 dy[2]  = -r2-r3-r9-r12+r1+r21
 dy[3]  = -r15+r1+r17+r19+r22
 dy[4]  = -r2-r16-r17-r23+r15
 dy[5]  = -r3+r4+r4+r6+r7+r13+r20
 dy[6]  = -r6-r8-r14-r20+r3+r18+r18
 dy[7]  = -r4-r5-r6+r13
 dy[8]  = r4+r5+r6+r7
 dy[9]  = -r7-r8
 dy[10] = -r12+r7+r9
 dy[11] = -r9-r10+r8+r11
 dy[12] = r9
 dy[13] = -r11+r10
 dy[14] = -r13+r12
 dy[15] = r14
 dy[16] = -r18-r19+r16
 dy[17] = -r20
 dy[18] = r20
 dy[19] = -r21-r22-r24+r23+r25
 dy[20] = -r25+r24
end

function fjac(J,y,p,t)
      J .= 0.0
      J[1,1]   = -k1-k10*y[11]-k14*y[6]-k23*y[4]-k24*y[19]
      J[1,11]  = -k10*y[1]+k9*y[2]
      J[1,6]   = -k14*y[1]
      J[1,4]   = -k23*y[1]+k2*y[2]
      J[1,19]  = -k24*y[1]+k22
      J[1,2]   = k2*y[4]+k9*y[11]+k3*y[5]+k12*y[10]
      J[1,13]  = k11
      J[1,20]  = k25
      J[1,5]   = k3*y[2]
      J[1,10]  = k12*y[2]

      J[2,4]   = -k2*y[2]
      J[2,5]   = -k3*y[2]
      J[2,11]  = -k9*y[2]
      J[2,10]  = -k12*y[2]
      J[2,19]  = k21
      J[2,1]   = k1
      J[2,2]   = -k2*y[4]-k3*y[5]-k9*y[11]-k12*y[10]

      J[3,1]   = k1
```

```
J[3,4]   = k17
J[3,16]  = k19
J[3,19]  = k22
J[3,3]   = -k15

J[4,4]   = -k2*y[2]-k16-k17-k23*y[1]
J[4,2]   = -k2*y[4]
J[4,1]   = -k23*y[4]
J[4,3]   = k15

J[5,5]   = -k3*y[2]
J[5,2]   = -k3*y[5]
J[5,7]   = 2k4+k6*y[6]
J[5,6]   = k6*y[7]+k20*y[17]
J[5,9]   = k7
J[5,14]  = k13
J[5,17]  = k20*y[6]

J[6,6]   = -k6*y[7]-k8*y[9]-k14*y[1]-k20*y[17]
J[6,7]   = -k6*y[6]
J[6,9]   = -k8*y[6]
J[6,1]   = -k14*y[6]
J[6,17]  = -k20*y[6]
J[6,2]   = k3*y[5]
J[6,5]   = k3*y[2]
J[6,16]  = 2k18

J[7,7]   = -k4-k5-k6*y[6]
J[7,6]   = -k6*y[7]
J[7,14]  = k13

J[8,7]   = k4+k5+k6*y[6]
J[8,6]   = k6*y[7]
J[8,9]   = k7

J[9,9]   = -k7-k8*y[6]
J[9,6]   = -k8*y[9]

J[10,10] = -k12*y[2]
J[10,2]  = -k12*y[10]+k9*y[11]
J[10,9]  = k7
J[10,11] = k9*y[2]

J[11,11] = -k9*y[2]-k10*y[1]
J[11,2]  = -k9*y[11]
J[11,1]  = -k10*y[11]
J[11,9]  = k8*y[6]
J[11,6]  = k8*y[9]
J[11,13] = k11

J[12,11] = k9*y[2]
J[12,2]  = k9*y[11]

J[13,13] = -k11
J[13,11] = k10*y[1]
J[13,1]  = k10*y[11]

J[14,14] = -k13
J[14,10] = k12*y[2]
```

```
    J[14,2]  = k12*y[10]

    J[15,1]  = k14*y[6]
    J[15,6]  = k14*y[1]

    J[16,16] = -k18-k19
    J[16,4]  = k16

    J[17,17] = -k20*y[6]
    J[17,6]  = -k20*y[17]

    J[18,17] = k20*y[6]
    J[18,6]  = k20*y[17]

    J[19,19] = -k21-k22-k24*y[1]
    J[19,1]  = -k24*y[19]+k23*y[4]
    J[19,4]  = k23*y[1]
    J[19,20] = k25

    J[20,20] = -k25
    J[20,1]  = k24*y[19]
    J[20,19] = k24*y[1]

    return
end

u0 = zeros(20)
u0[2]  = 0.2
u0[4]  = 0.04
u0[7]  = 0.1
u0[8]  = 0.3
u0[9]  = 0.01
u0[17] = 0.007
prob = ODEProblem(ODEFunction(f, jac=fjac),u0,(0.0,60.0))

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

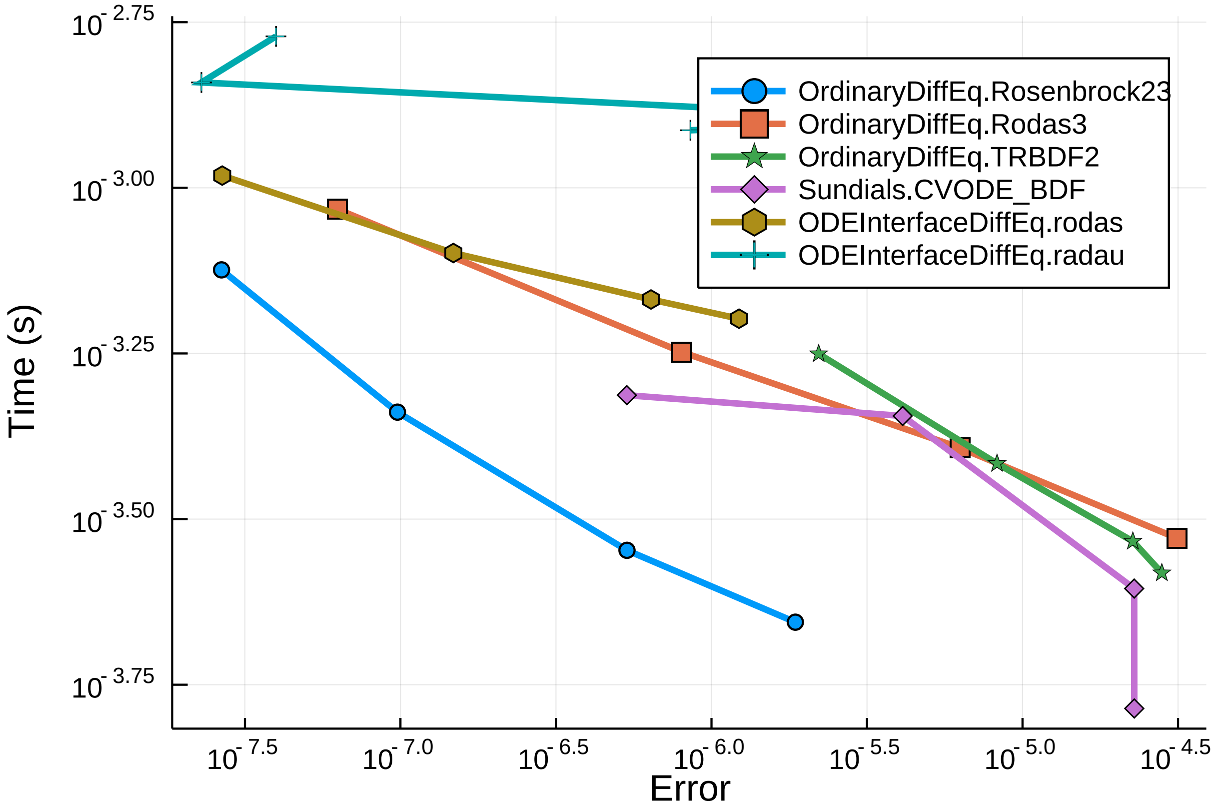
The following were omitted from the tests due to convergence failures. ODE.jl's adaptivity is not able to stabilize its algorithms, while GeometricIntegratorsDiffEq has not upgraded to Julia 1.0. GeometricIntegrators.jl's methods used to be either fail to converge at comparable dts (or on some computers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

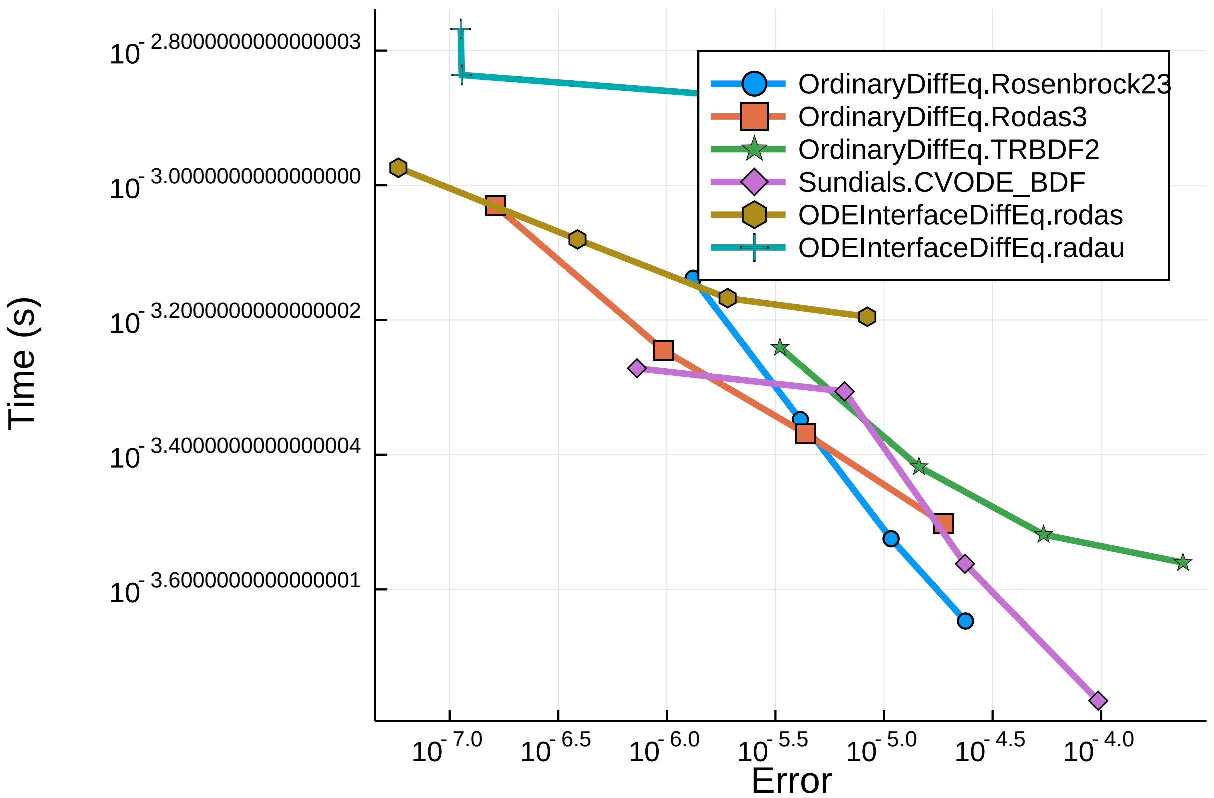
0.2 High Tolerances

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())
          ]
wp = WorkPrecisionSet(prob,abstols,reltols,setups;verbose=false,
                      save_everystep=false,appxsol=test_sol,maxiters=Int(1e5),numruns=10)
plot(wp)
```

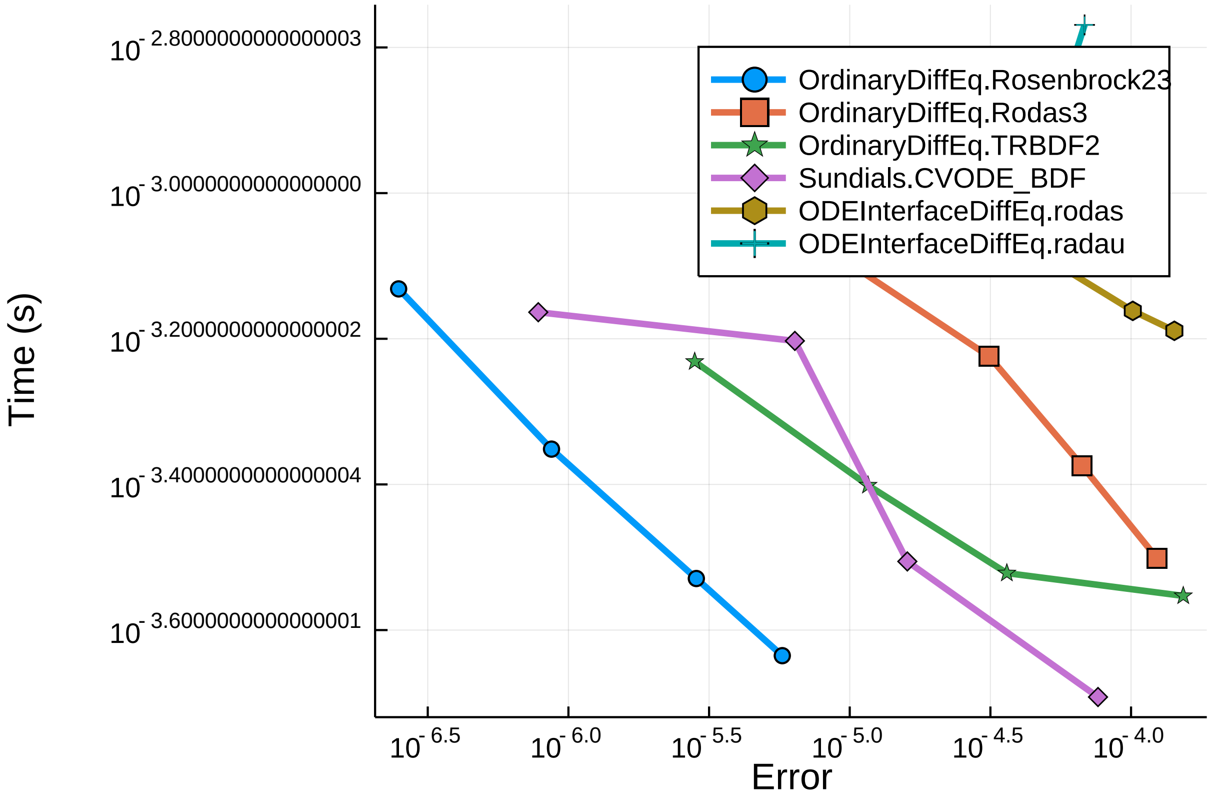



```
wp = WorkPrecisionSet(prob,abstols,reltols,setups;dense = false,verbose = false,
                      appxsol=test_sol,maxiters=Int(1e5),error_estimate=:l2,numruns=10)
plot(wp)
```

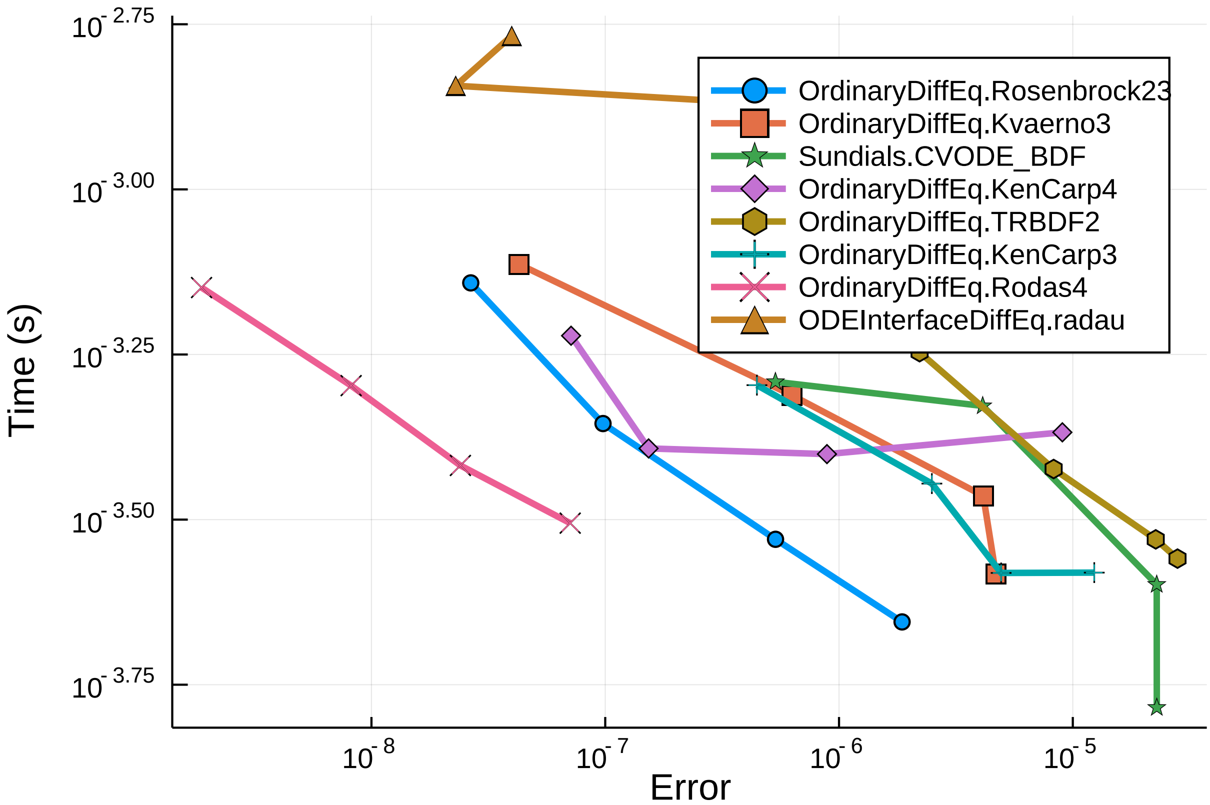



```
wp = WorkPrecisionSet(prob,abstols,reltols,setups;verbose=false,
                      appxsol=test_sol,maxiters=Int(1e5),error_estimate=:L2,numruns=10)
```

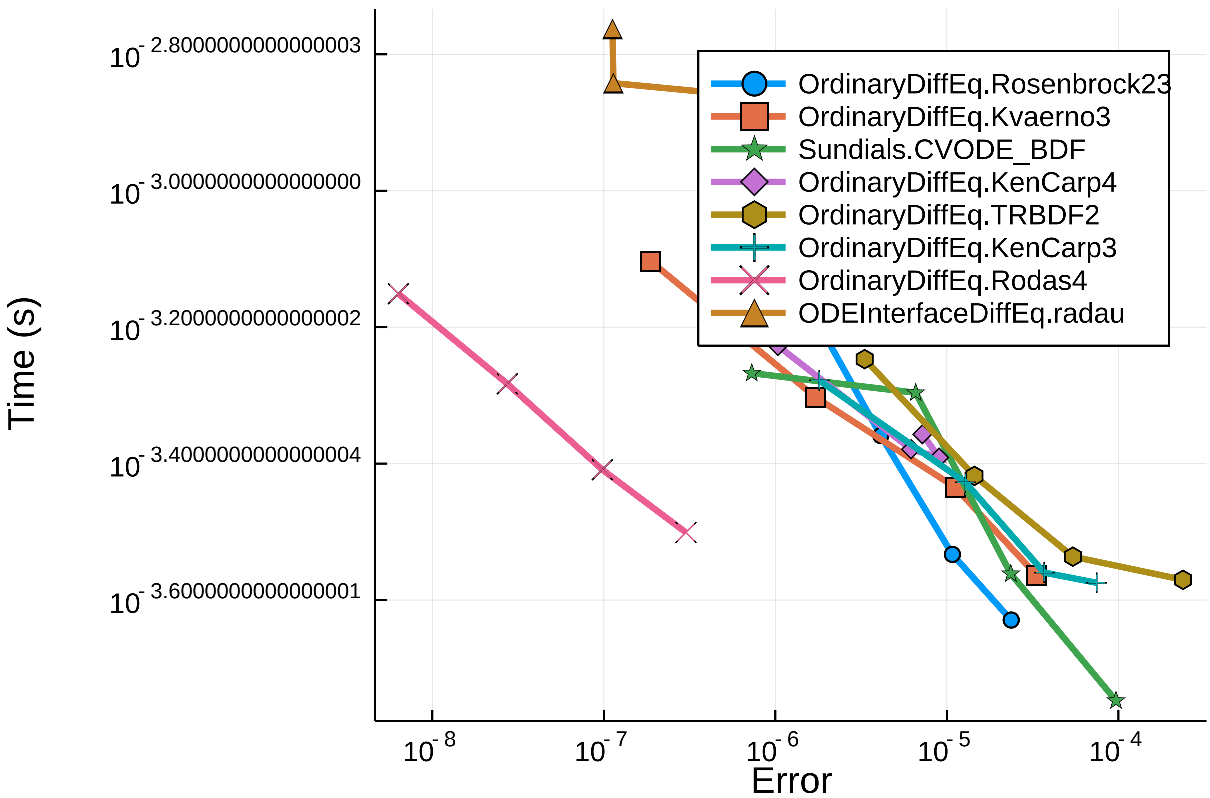
```
plot(wp)
```




```
setups = [Dict(:alg=>Rosenbrock23()),
          Dict(:alg=>Kvaerno3()),
          Dict(:alg=>CVODE_BDF()),
          Dict(:alg=>KenCarp4()),
          Dict(:alg=>TRBDF2()),
          Dict(:alg=>KenCarp3()),
          Dict(:alg=>Rodas4()),
          Dict(:alg=>radau())]
wp = WorkPrecisionSet(prob,abstols,reltols,setups;
                      save_everystep=false,appxsol=test_sol,maxiters=Int(1e5),numruns=10)
plot(wp)
```

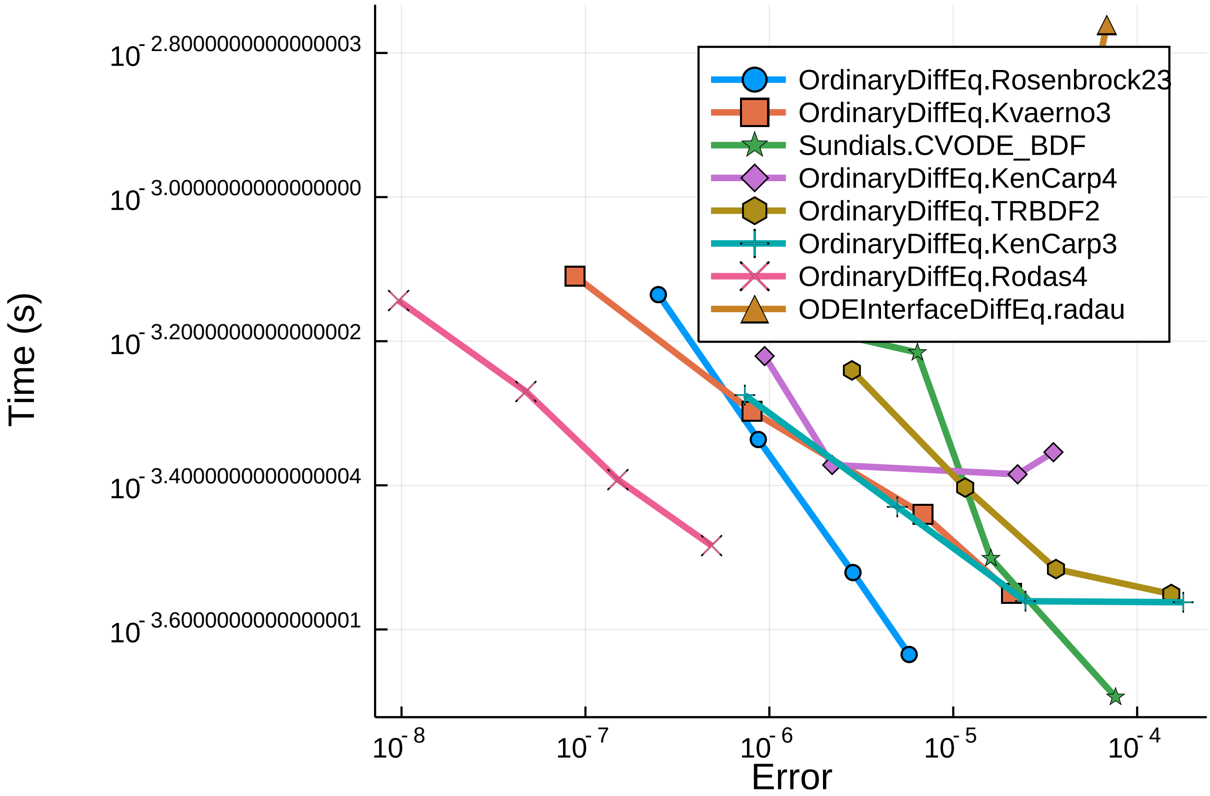



```
wp = WorkPrecisionSet(prob,abstols,reltols,setups;dense = false,verbose = false,
                      appxsol=test_sol,maxiters=Int(1e5),error_estimate=:l2,numruns=10)
plot(wp)
```

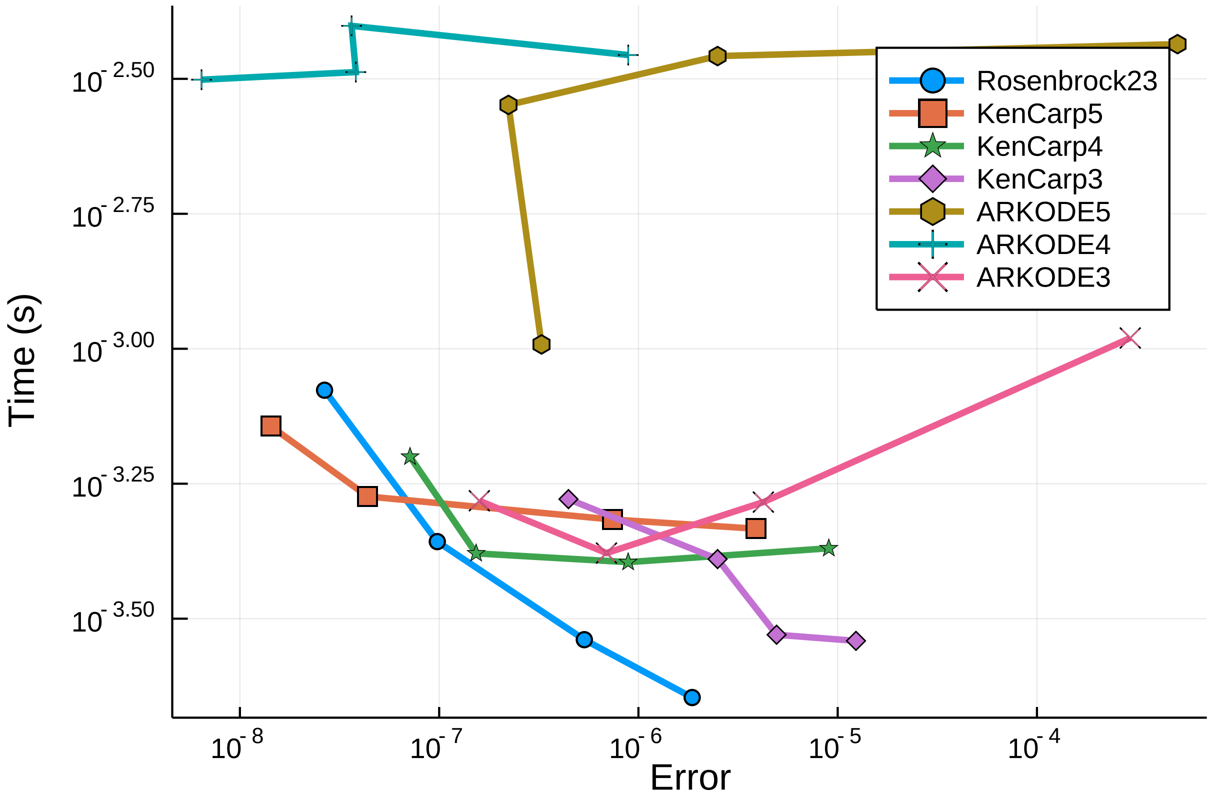



```
wp = WorkPrecisionSet(prob,abstols,reltols,setups;
                      appxsol=test_sol,maxiters=Int(1e5),error_estimate=:L2,numruns=10)
```

```
plot(wp)
```



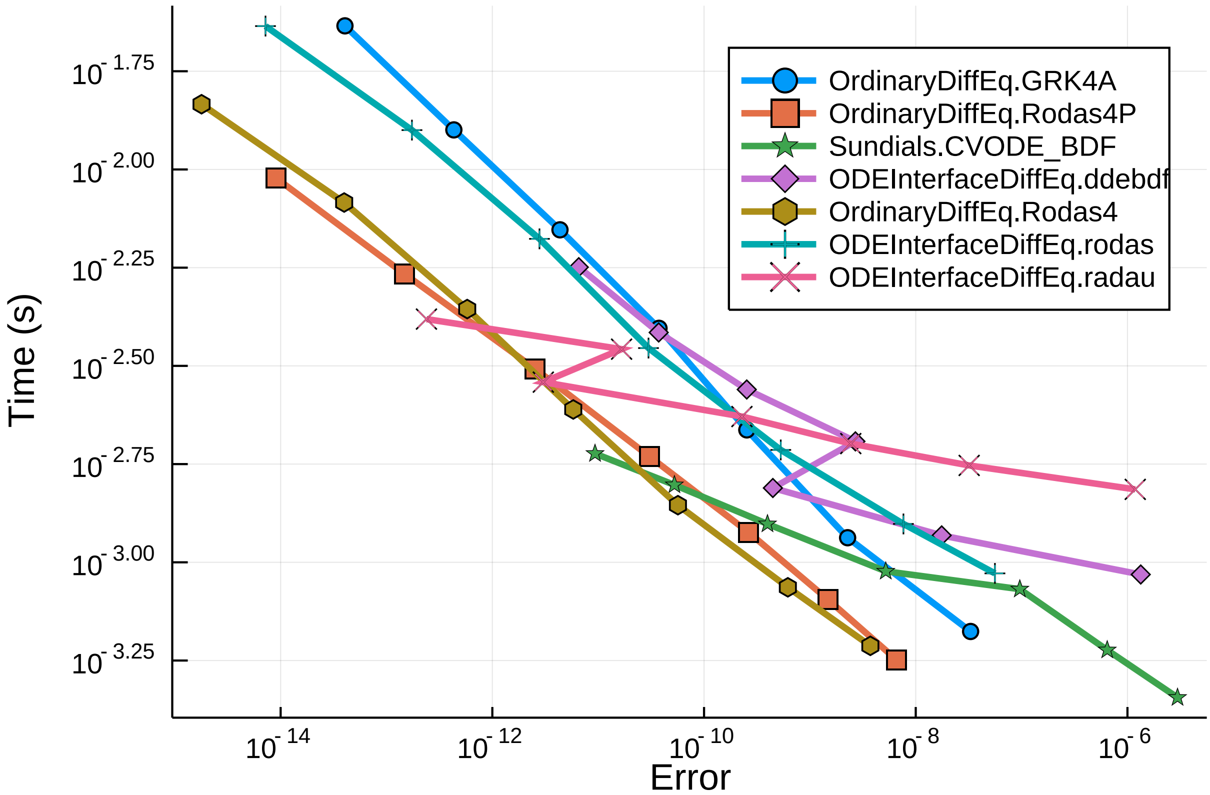

```
setups = [Dict(:alg=>Rosenbrock23()),
          Dict(:alg=>KenCarp5()),
          Dict(:alg=>KenCarp4()),
          Dict(:alg=>KenCarp3()),
          Dict(:alg=>ARKODE(order=5)),
          Dict(:alg=>ARKODE()),
          Dict(:alg=>ARKODE(order=3))]
names = ["Rosenbrock23" "KenCarp5" "KenCarp4" "KenCarp3" "ARKODE5" "ARKODE4" "ARKODE3"]
wp = WorkPrecisionSet(prob,abstols,reltols,setups;

    names=names,save_everystep=false,appxsol=test_sol,maxiters=Int(1e5),numruns=10)
plot(wp)
```



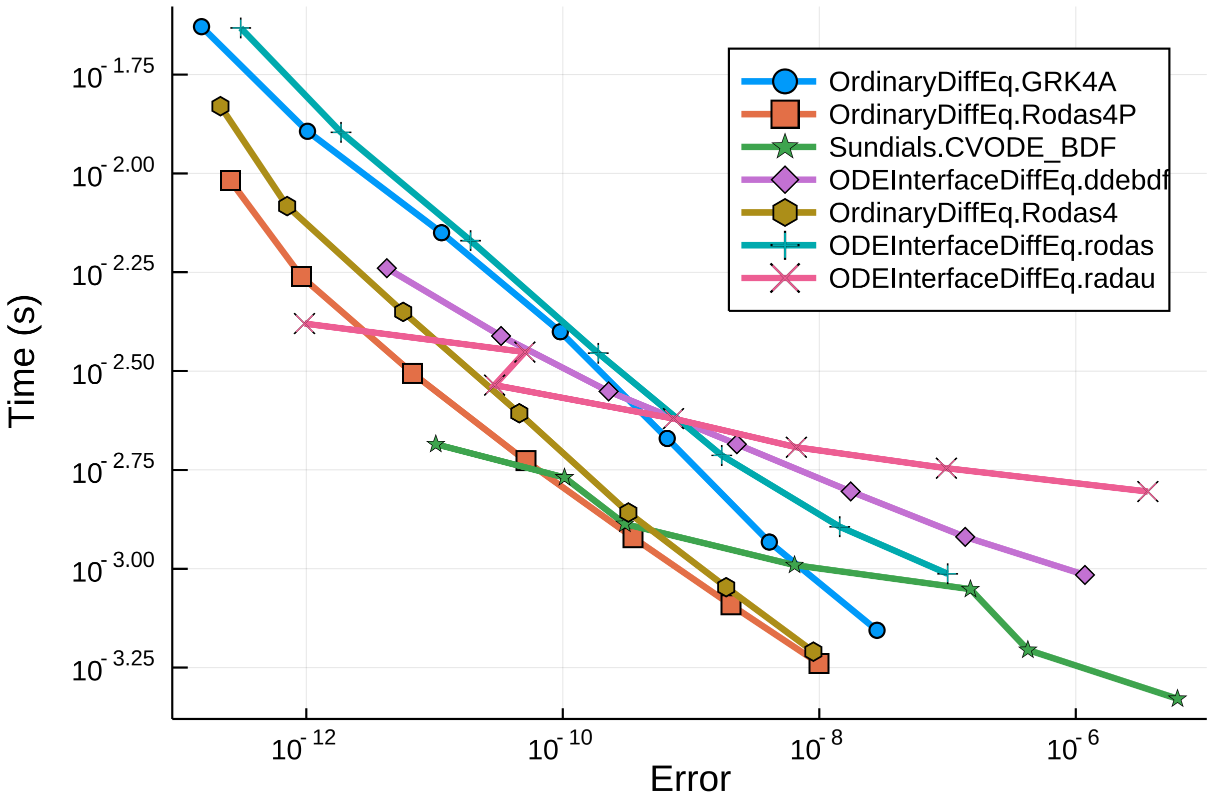
0.2.1 Low Tolerances

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

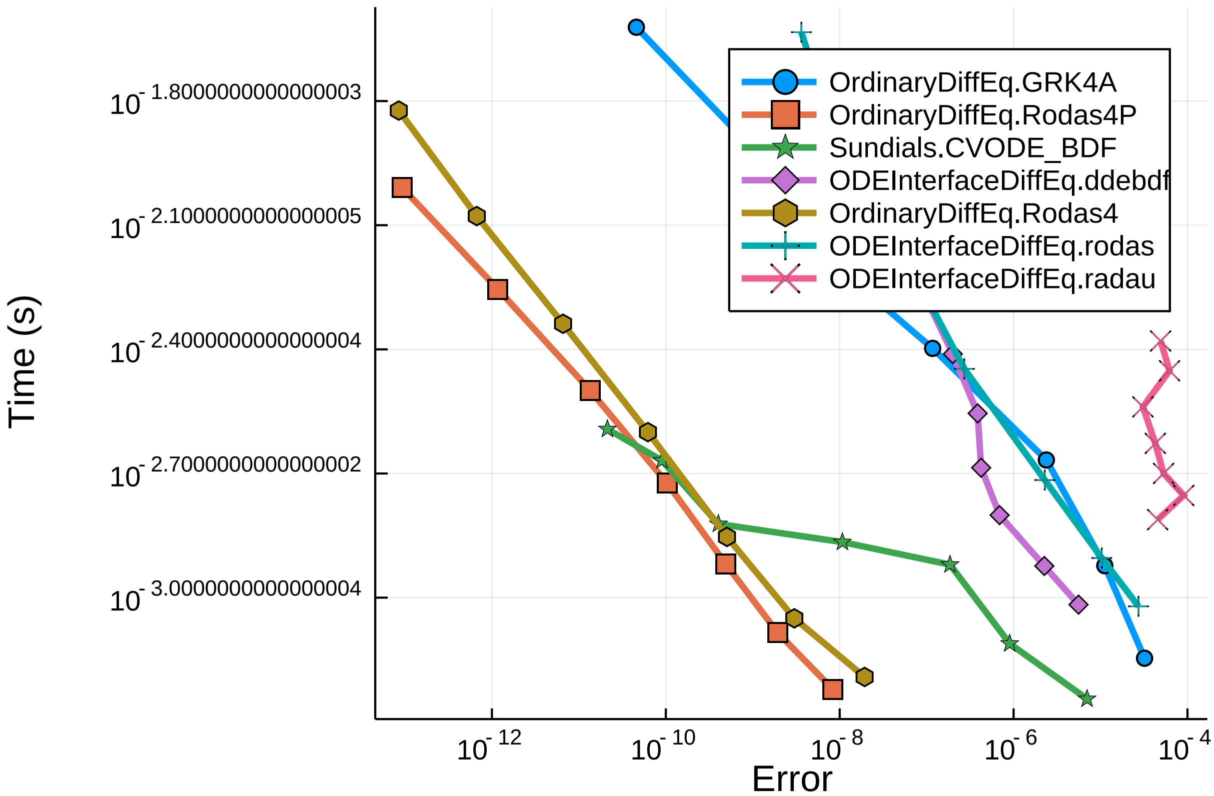



```
wp = WorkPrecisionSet(prob,abstols,reltols,setups;verbose=false,

    dense=false,appxsol=test_sol,maxiters=Int(1e5),error_estimate=:l2,numruns=10)
plot(wp)
```

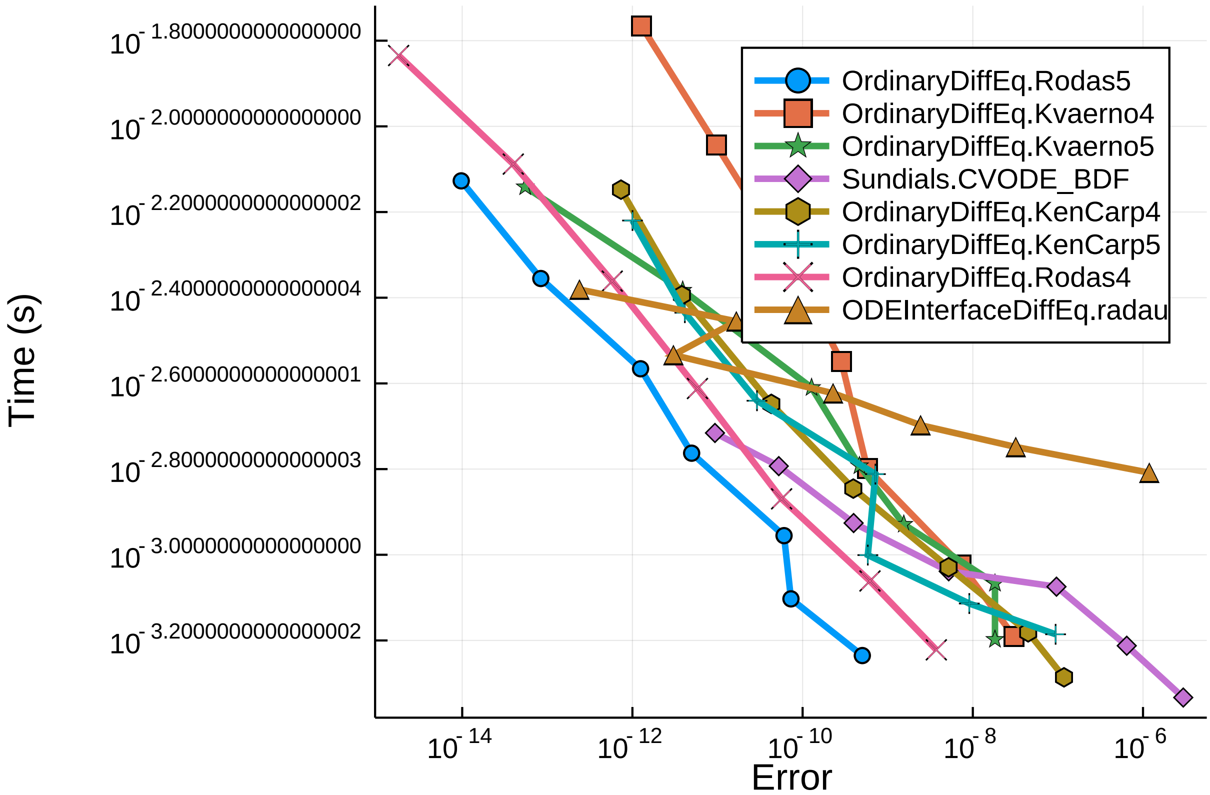



```
wp = WorkPrecisionSet(prob,abstols,reltols,setups;verbose=false,
```

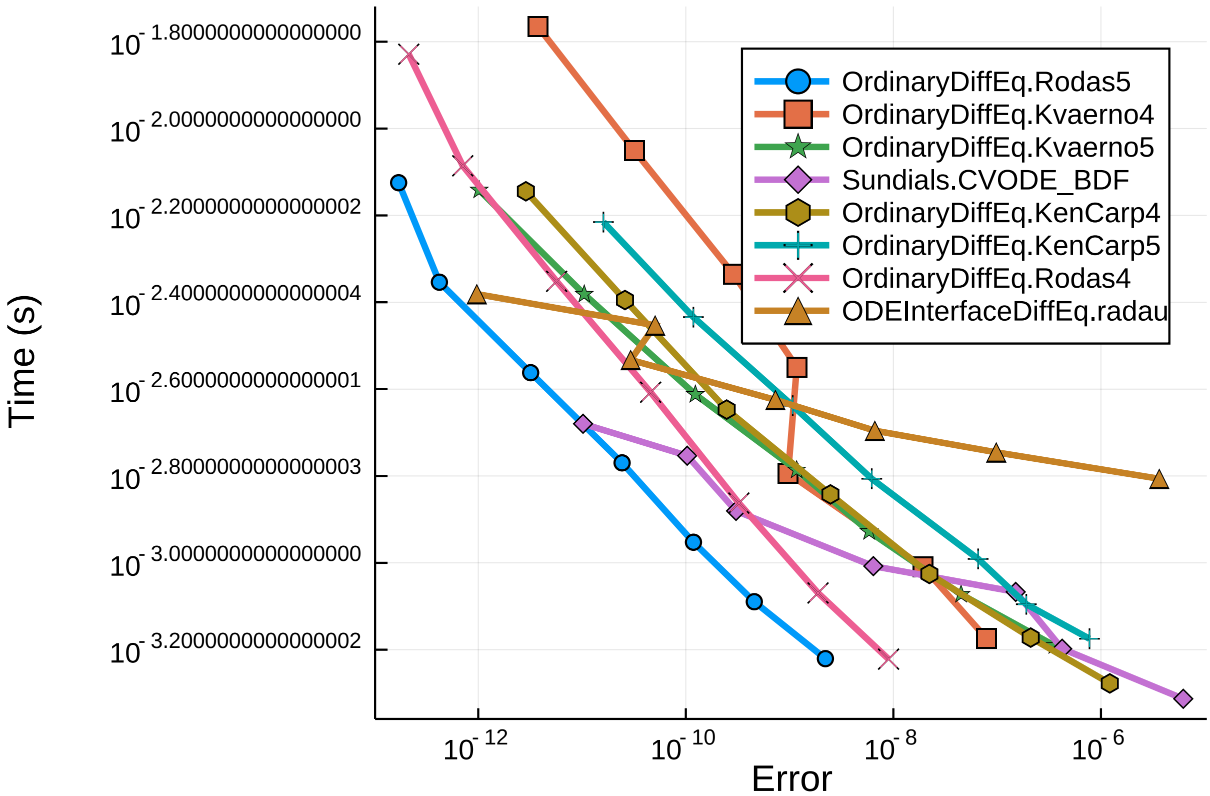
```
                              appxsol=test_sol,maxiters=Int(1e5),error_estimate=:L2,numruns=10)
plot(wp)
```




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

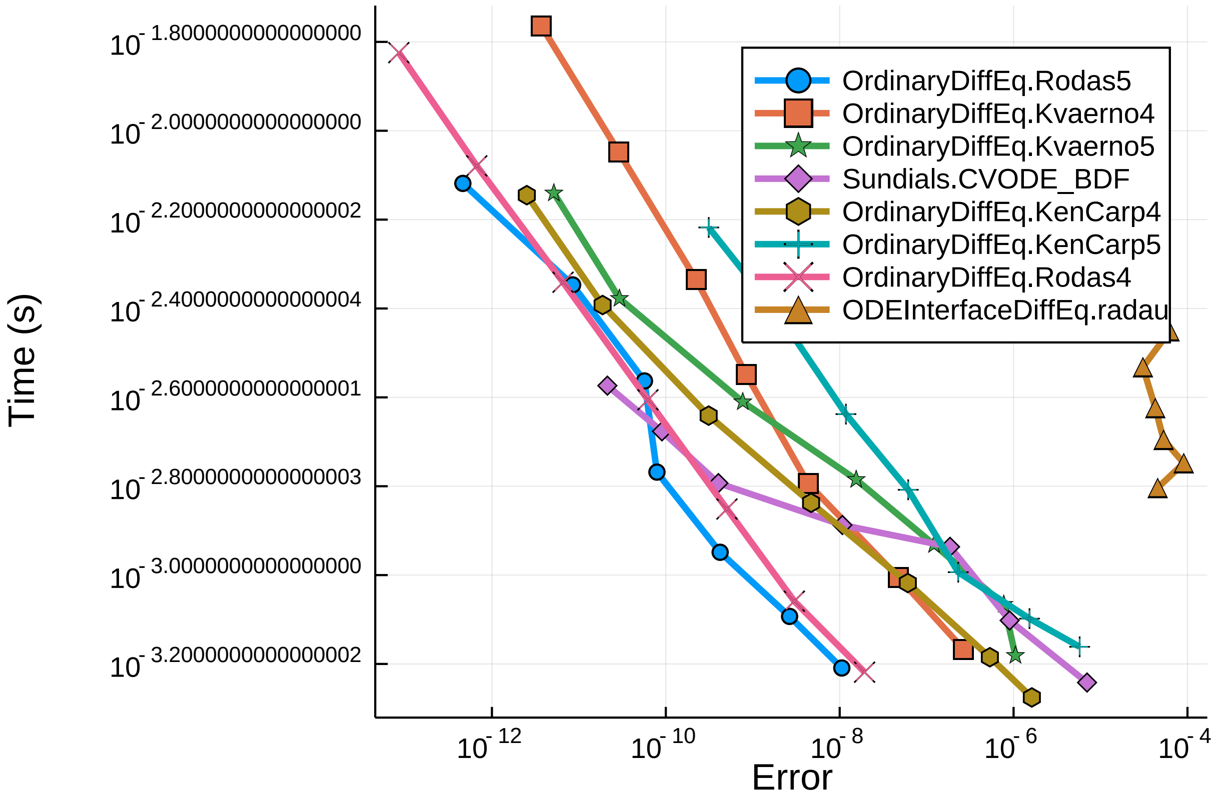



```
wp = WorkPrecisionSet(prob,abstols,reltols,setups;verbose=false,

    dense=false,appxsol=test_sol,maxiters=Int(1e5),error_estimate=:l2,numruns=10)
plot(wp)
```




```
wp = WorkPrecisionSet(prob,abstols,reltols,setups;
```

```
                              appxsol=test_sol,maxiters=Int(1e5),error_estimate=:L2,numruns=10)
plot(wp)
```



The following algorithms were removed since they failed.

```
#setups = [#Dict(:alg=>Hairer4()),
          #Dict(:alg=>Hairer42()),
          #Dict(:alg=>Rodas3()),
          #Dict(:alg=>Cash4())
#]
#wp = WorkPrecisionSet(prob,abstols,reltols,setups;
# save_everystep=false,appxsol=test_sol,maxiters=Int(1e5),numruns=10)
#plot(wp)
```

0.2.2 Conclusion

Sundials `CVODE_BDF` the best here. `lsoda` does well at high tolerances but then grows fast when tolerances get too low. `KenCarp4` or `Rodas5` is a decent substitute when necessary.

```
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","Pollution.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
```