Mackey and Glass Work-Precision Diagrams

David Widmann, Chris Rackauckas

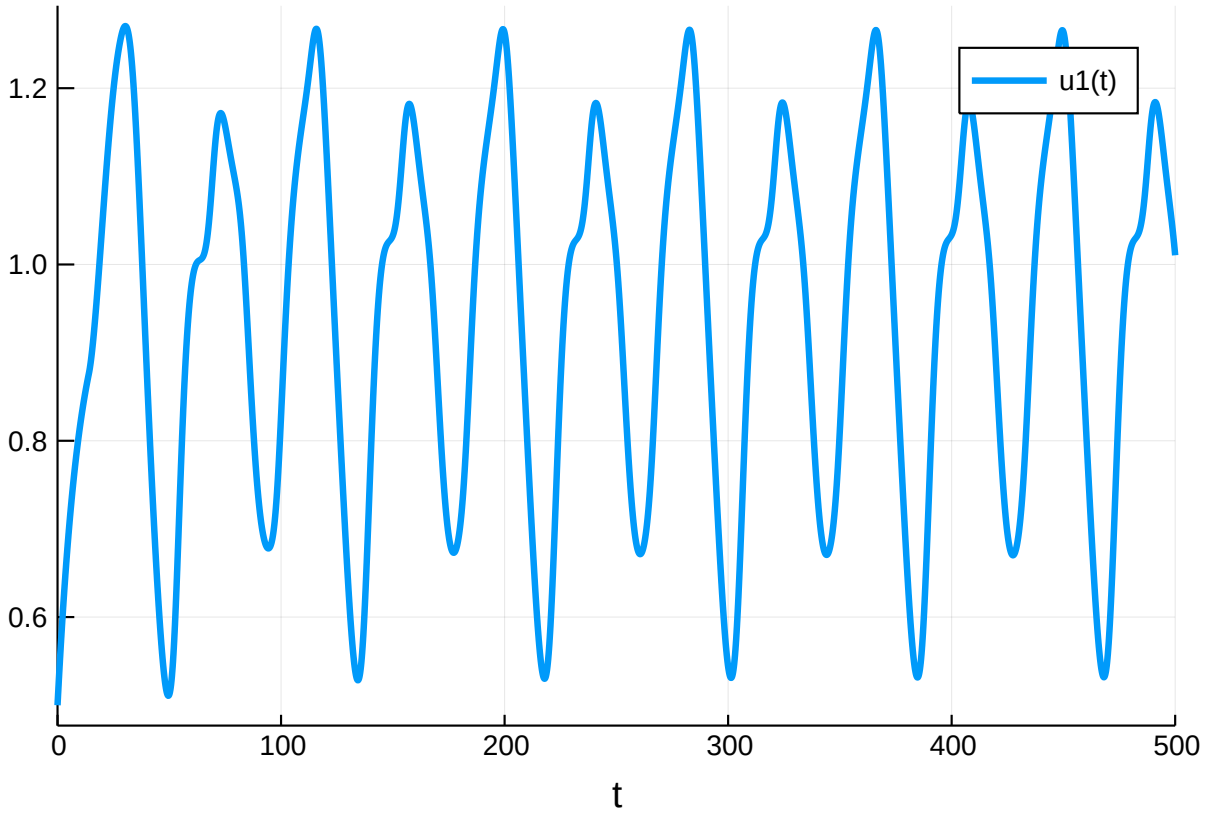
May 3, 2019

1 Mackey and Glass

We study algorithms for solving constant delay differential equations with a test problem from W.H. Enright and H. Hayashi, "The evaluation of numerical software for delay differential equations", 1997. It is a model of blood production that was published by M. C. Mackey and L. Glass in "Oscillation and chaos in physiological control systems", 1977, and is given by

$$y'(t) = \frac{0.2y(t-14)}{1+y(t-14)^{10}} - 0.1y(t) \tag{1}$$

```
using DelayDiffEq, DiffEqDevTools, DiffEqProblemLibrary, Plots
using DiffEqProblemLibrary.DDEProblemLibrary: importddeproblems; importddeproblems()
import DiffEqProblemLibrary.DDEProblemLibrary: prob_dde_mackey
gr()

sol = solve(prob_dde_mackey, MethodOfSteps(Vern9(), max_fixedpoint_iters=1000);
    reltol=1e-14, abstol=1e-14)
test_sol = TestSolution(sol)
plot(sol)
```

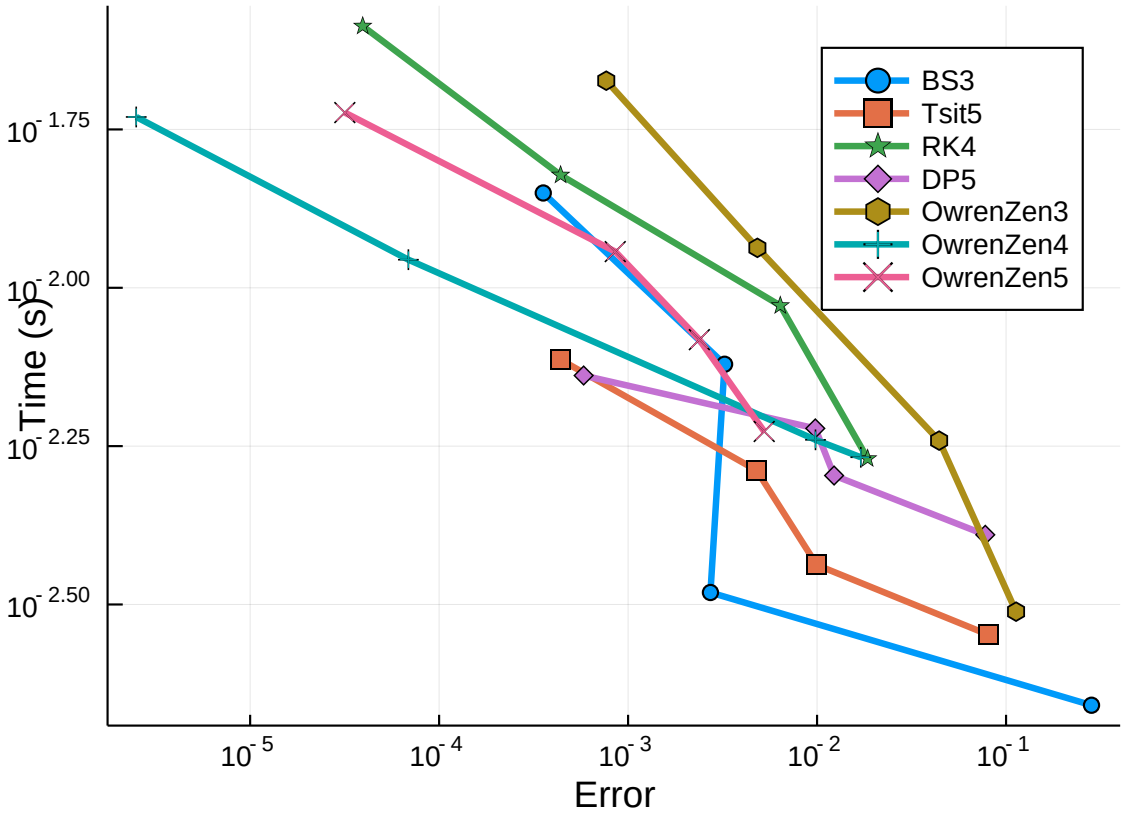


1.1 Low order RK methods

1.1.1 High tolerances

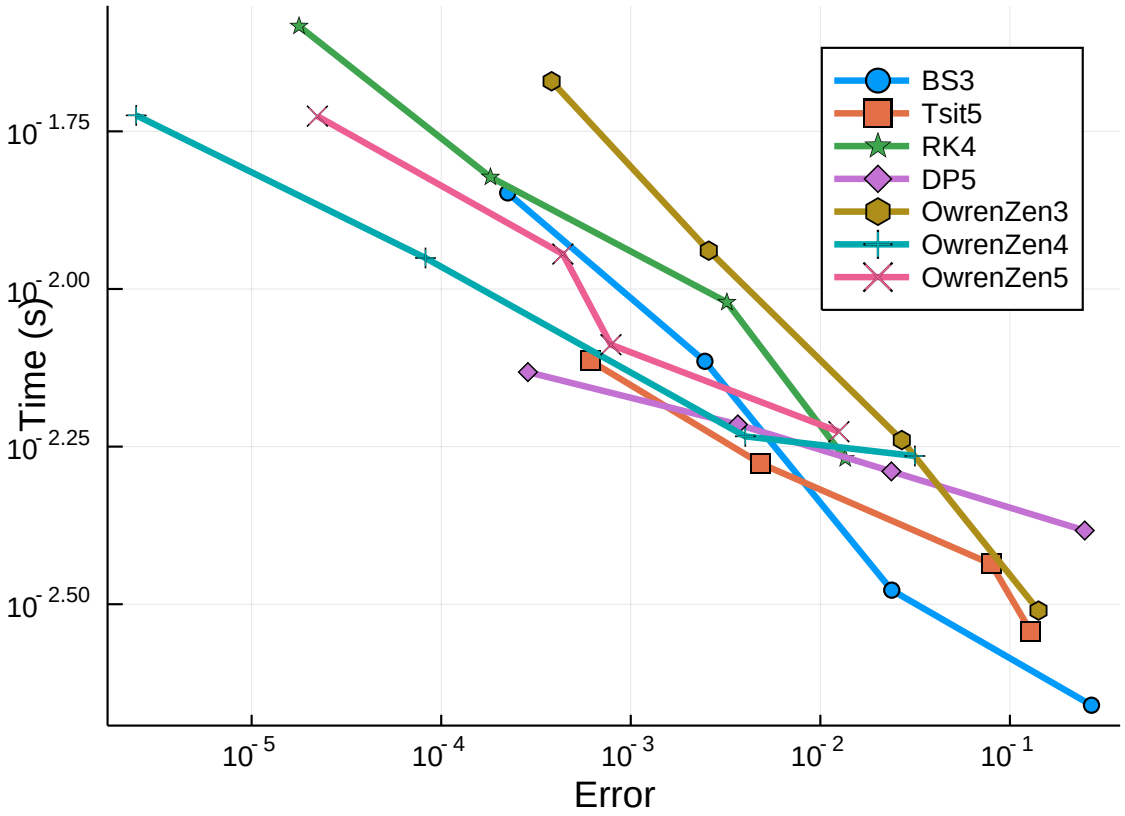
First we test final error estimates of continuous RK methods of low order at high tolerances. `OwrenZen4`, `OwrenZen5`, and `RK4` yield the best error estimates.

```
abstols = 1.0 ./ 10.0 .^ (4:7)
reltols = 1.0 ./ 10.0 .^ (1:4)

setups = [Dict(:alg=>MethodOfSteps(BS3())),
          Dict(:alg=>MethodOfSteps(Tsit5())),
          Dict(:alg=>MethodOfSteps(RK4())),
          Dict(:alg=>MethodOfSteps(DP5())),
          Dict(:alg=>MethodOfSteps(OwrenZen3())),
          Dict(:alg=>MethodOfSteps(OwrenZen4())),
          Dict(:alg=>MethodOfSteps(OwrenZen5()))]
names = ["BS3", "Tsit5", "RK4", "DP5", "OwrenZen3", "OwrenZen4", "OwrenZen5"]
wp = WorkPrecisionSet(prob_dde_mackey,abstols,reltols,setups;names=names,
                      appxsol=test_sol,maxiters=Int(1e5),error_estimate=:final)
plot(wp)
```



Next we test average interpolation errors:

```
abstols = 1.0 ./ 10.0 .^ (4:7)
reltols = 1.0 ./ 10.0 .^ (1:4)

setups = [Dict(:alg=>MethodOfSteps(BS3())),
          Dict(:alg=>MethodOfSteps(Tsit5())),
          Dict(:alg=>MethodOfSteps(RK4())),
          Dict(:alg=>MethodOfSteps(DP5())),
          Dict(:alg=>MethodOfSteps(OwrenZen3())),
          Dict(:alg=>MethodOfSteps(OwrenZen4())),
          Dict(:alg=>MethodOfSteps(OwrenZen5()))]
names = ["BS3", "Tsit5", "RK4", "DP5", "OwrenZen3", "OwrenZen4", "OwrenZen5"]
wp = WorkPrecisionSet(prob_dde_mackey,abstols,reltols,setups;names=names,
                      appxsol=test_sol,maxiters=Int(1e5),error_estimate=:L2)
plot(wp)
```

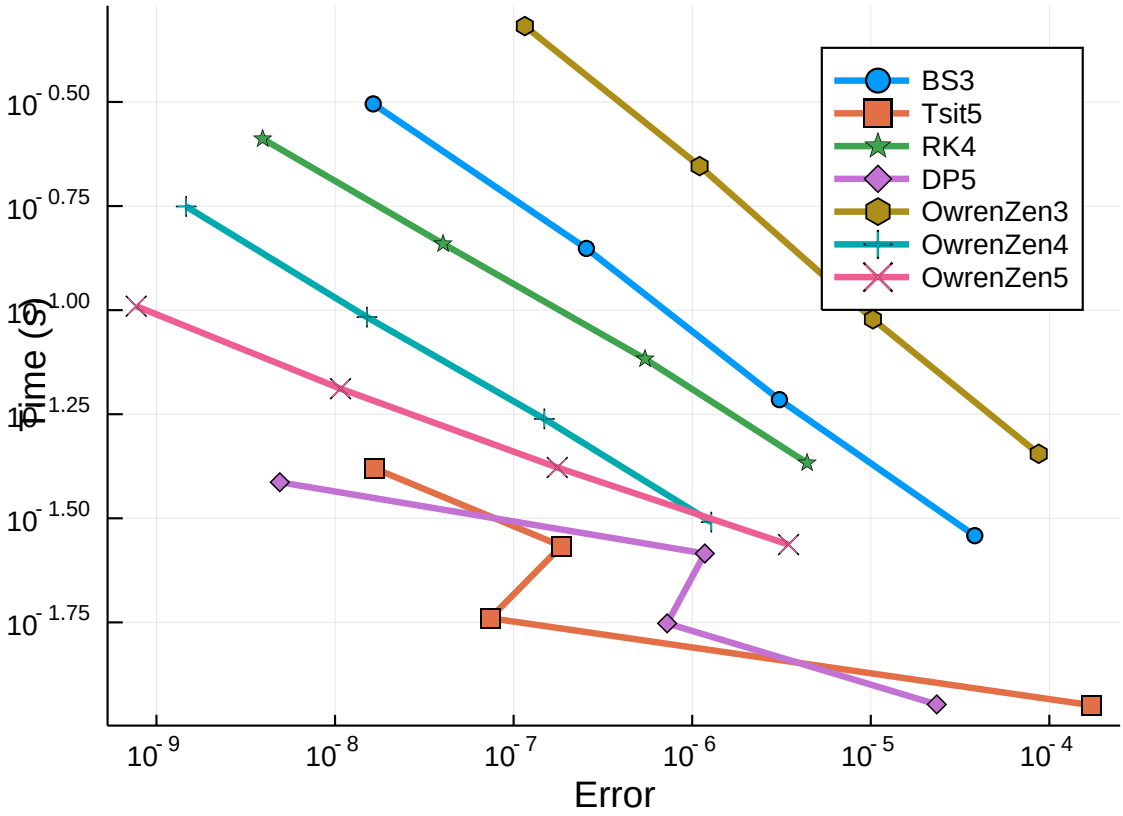


As before, `OwrenZen4` and `OwrenZen5` perform well over the whole range of investigated tolerances.

1.1.2 Low tolerances

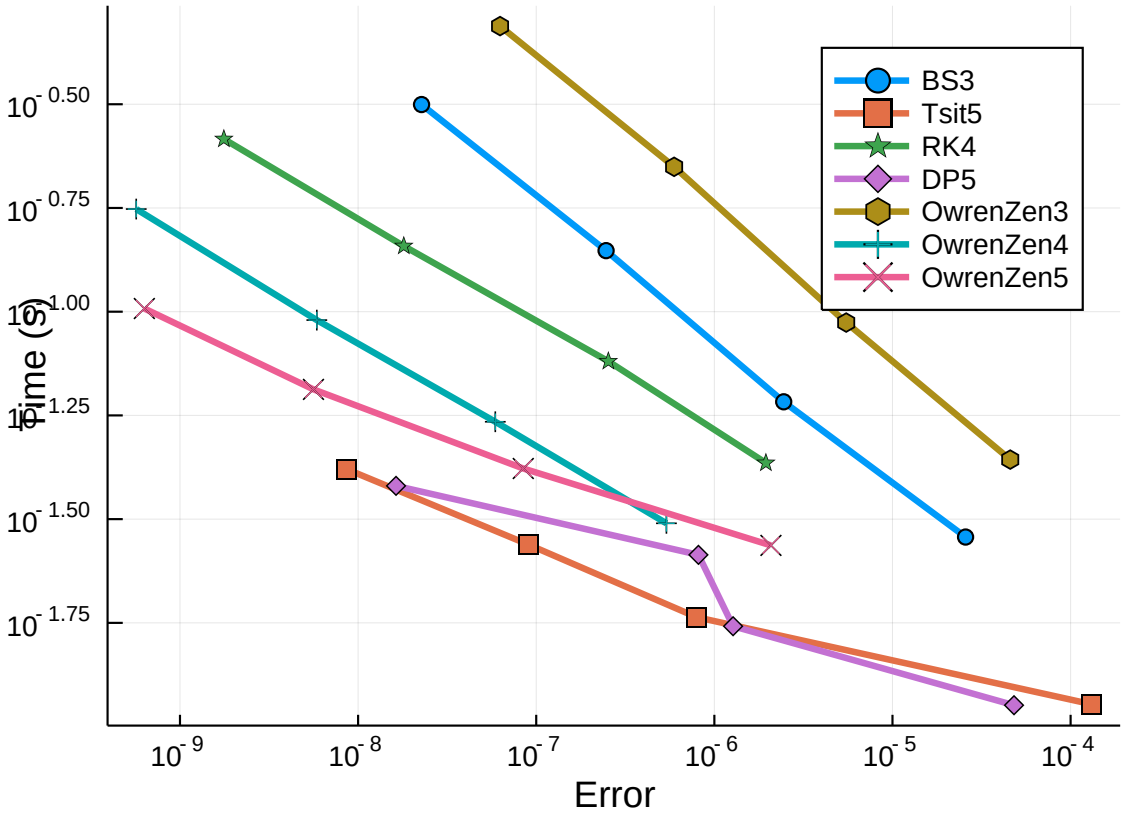
We repeat our tests with low tolerances.

```
abstols = 1.0 ./ 10.0 .^ (8:11)
reltols = 1.0 ./ 10.0 .^ (5:8)

setups = [Dict(:alg=>MethodOfSteps(BS3())),
          Dict(:alg=>MethodOfSteps(Tsit5())),
          Dict(:alg=>MethodOfSteps(RK4())),
          Dict(:alg=>MethodOfSteps(DP5())),
          Dict(:alg=>MethodOfSteps(OwrenZen3())),
          Dict(:alg=>MethodOfSteps(OwrenZen4())),
          Dict(:alg=>MethodOfSteps(OwrenZen5()))]
names = ["BS3", "Tsit5", "RK4", "DP5", "OwrenZen3", "OwrenZen4", "OwrenZen5"]
wp = WorkPrecisionSet(prob_dde_mackey,abstols,reltols,setups;names=names,
                      appxsol=test_sol,maxiters=Int(1e5),error_estimate=:final)
plot(wp)
```



And once again we also test the interpolation errors:

```
abstols = 1.0 ./ 10.0 .^ (8:11)
reltols = 1.0 ./ 10.0 .^ (5:8)

setups = [Dict(:alg=>MethodOfSteps(BS3())),
          Dict(:alg=>MethodOfSteps(Tsit5())),
          Dict(:alg=>MethodOfSteps(RK4())),
          Dict(:alg=>MethodOfSteps(DP5())),
          Dict(:alg=>MethodOfSteps(OwrenZen3())),
          Dict(:alg=>MethodOfSteps(OwrenZen4())),
          Dict(:alg=>MethodOfSteps(OwrenZen5()))]
names = ["BS3", "Tsit5", "RK4", "DP5", "OwrenZen3", "OwrenZen4", "OwrenZen5"]
wp = WorkPrecisionSet(prob_dde_mackey,abstols,reltols,setups;names=names,
                      appxsol=test_sol,maxiters=Int(1e5),error_estimate=:L2)
plot(wp)
```



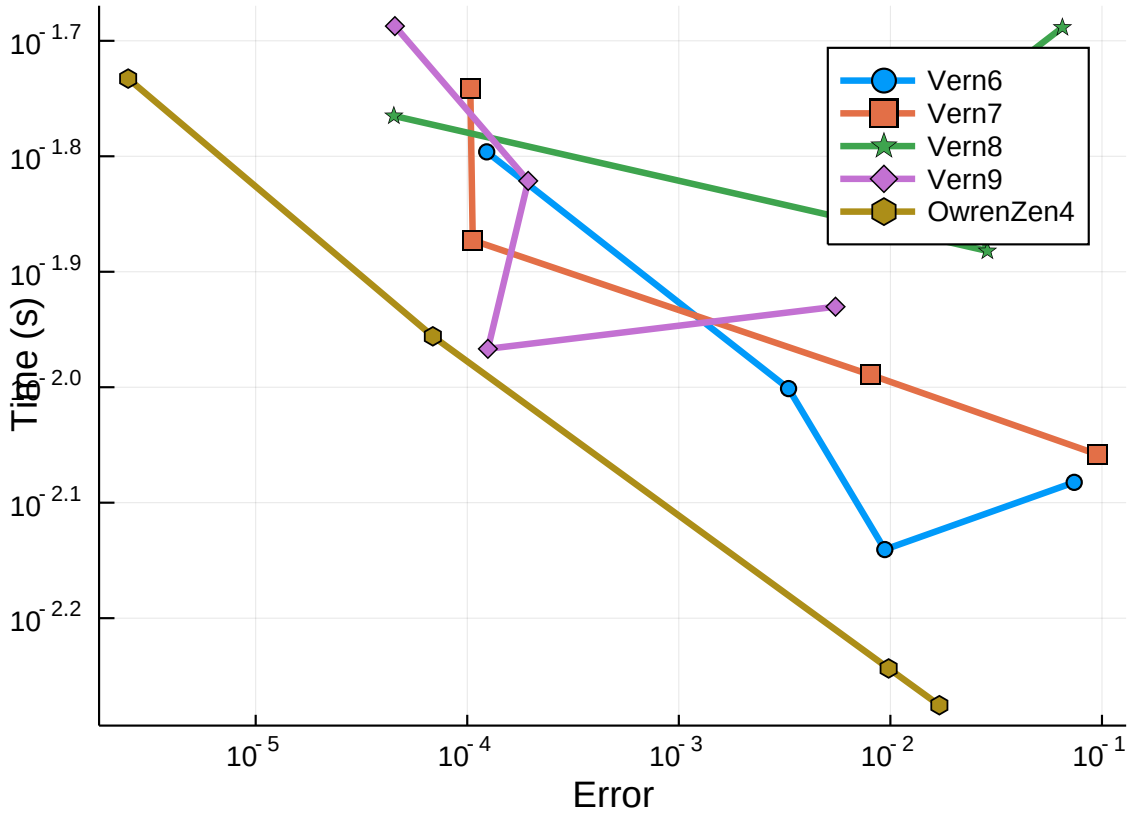
Apparently `Tsit5` and `DP5` perform quite well at low tolerances, but only `OwrenZen5`, `OwrenZen4` and `RK4` achieve interpolation errors of around 1e-9.

1.2 Lazy interpolants

1.2.1 High tolerances

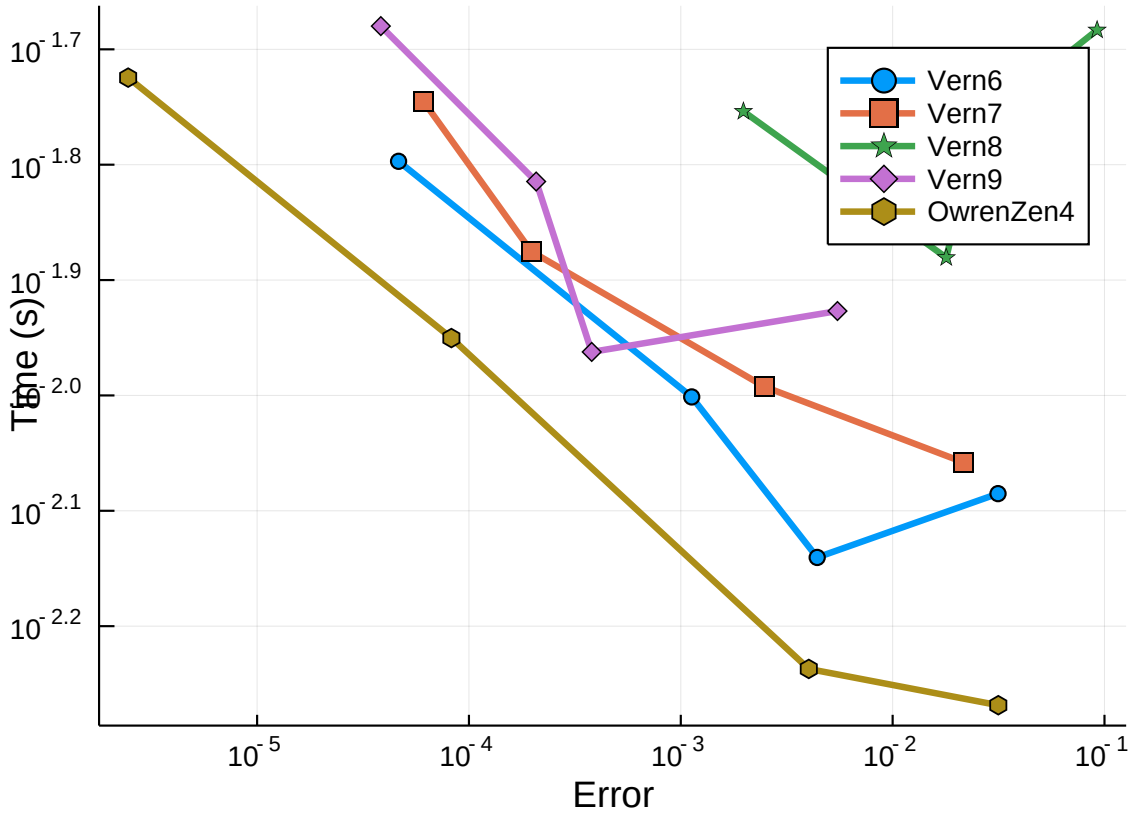
We repeat our tests with the Verner methods which, in contrast to the methods above, use lazy interpolants. As reference we include `OwrenZen4`.

```
abstols = 1.0 ./ 10.0 .^ (4:7)
reltols = 1.0 ./ 10.0 .^ (1:4)

setups = [Dict(:alg=>MethodOfSteps(Vern6())),
          Dict(:alg=>MethodOfSteps(Vern7())),
          Dict(:alg=>MethodOfSteps(Vern8())),
          Dict(:alg=>MethodOfSteps(Vern9())),
          Dict(:alg=>MethodOfSteps(OwrenZen4()))]
names = ["Vern6", "Vern7", "Vern8", "Vern9", "OwrenZen4"]
wp = WorkPrecisionSet(prob_dde_mackey,abstols,reltols,setups;names=names,
                      appxsol=test_sol,maxiters=Int(1e5),error_estimate=:final)
plot(wp)
```



And we obtain the following interpolation errors:

```
abstols = 1.0 ./ 10.0 .^ (4:7)
reltols = 1.0 ./ 10.0 .^ (1:4)

setups = [Dict(:alg=>MethodOfSteps(Vern6())),
          Dict(:alg=>MethodOfSteps(Vern7())),
          Dict(:alg=>MethodOfSteps(Vern8())),
          Dict(:alg=>MethodOfSteps(Vern9())),
          Dict(:alg=>MethodOfSteps(OwrenZen4()))]
names = ["Vern6", "Vern7", "Vern8", "Vern9", "OwrenZen4"]
wp = WorkPrecisionSet(prob_dde_mackey,abstols,reltols,setups;names=names,
                      appxsol=test_sol,maxiters=Int(1e5),error_estimate=:L2)
plot(wp)
```

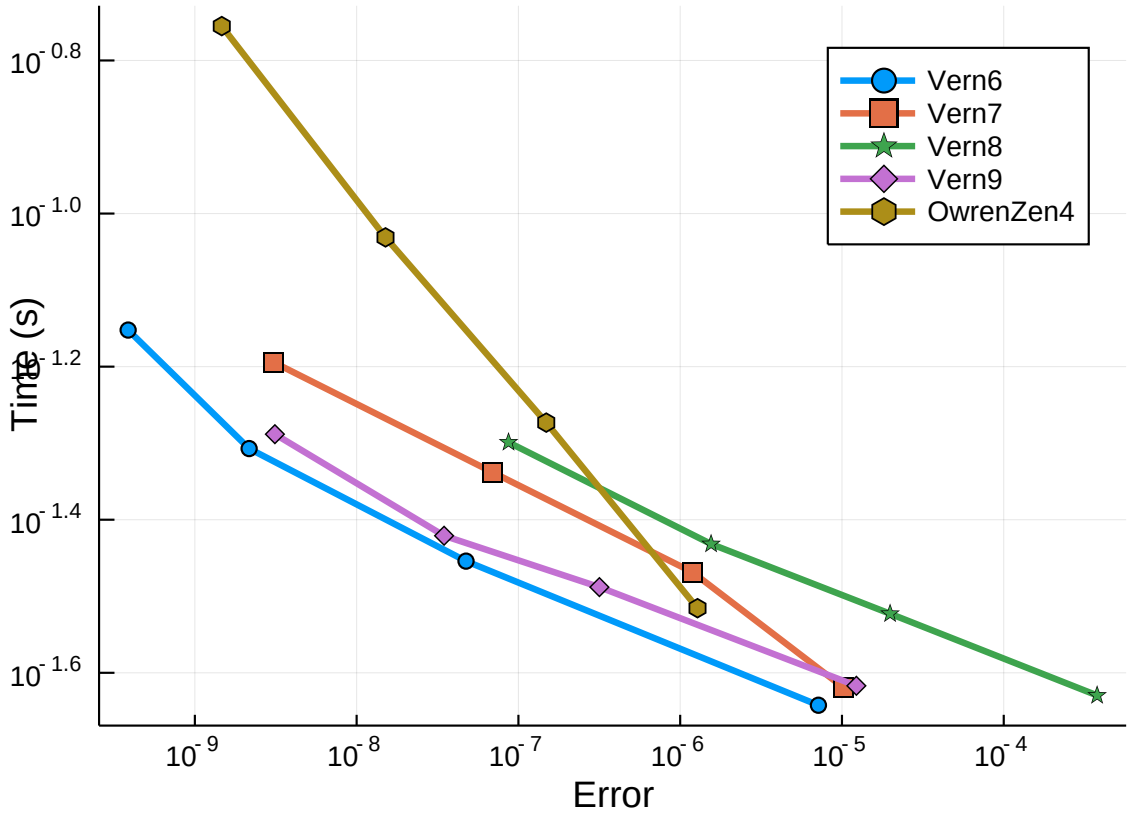


`Vern6`, `Vern7`, and `Vern9` are outperformed by `OwrenZen4`.

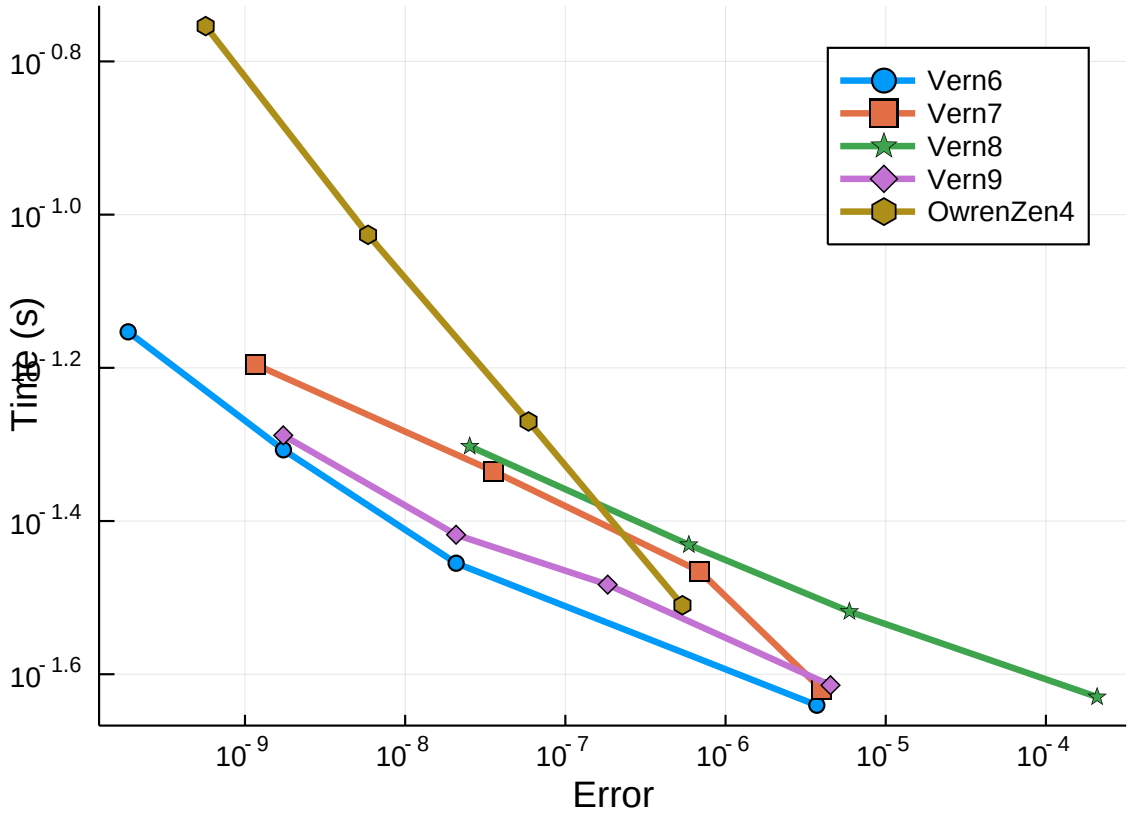
1.2.2 Low tolerances

Again, we repeat our tests at low tolerances.

```
abstols = 1.0 ./ 10.0 .^ (8:11)
reltols = 1.0 ./ 10.0 .^ (5:8)

setups = [Dict(:alg=>MethodOfSteps(Vern6())),
          Dict(:alg=>MethodOfSteps(Vern7())),
          Dict(:alg=>MethodOfSteps(Vern8())),
          Dict(:alg=>MethodOfSteps(Vern9())),
          Dict(:alg=>MethodOfSteps(OwrenZen4()))]
names = ["Vern6", "Vern7", "Vern8", "Vern9", "OwrenZen4"]
wp = WorkPrecisionSet(prob_dde_mackey,abstols,reltols,setups;names=names,
                      appxsol=test_sol,maxiters=Int(1e5),error_estimate=:final)
plot(wp)
```




```
abstols = 1.0 ./ 10.0 .^ (8:11)
reltols = 1.0 ./ 10.0 .^ (5:8)

setups = [Dict(:alg=>MethodOfSteps(Vern6())),
          Dict(:alg=>MethodOfSteps(Vern7())),
          Dict(:alg=>MethodOfSteps(Vern8())),
          Dict(:alg=>MethodOfSteps(Vern9())),
          Dict(:alg=>MethodOfSteps(OwrenZen4()))]
names = ["Vern6", "Vern7", "Vern8", "Vern9", "OwrenZen4"]
wp = WorkPrecisionSet(prob_dde_mackey,abstols,reltols,setups;names=names,
                      appxsol=test_sol,maxiters=Int(1e5),error_estimate=:L2)
plot(wp)
```



Vern6, Vern7, and Vern9 show similar results at low tolerances, and perform even better than OwrenZen4.

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

1.3 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("NonStiffDDE","Mackey_Glass_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/crackauckas/.julia/environments/v1.1/Project.toml`
[c52e3926-4ff0-5f6e-af25-54175e0327b1] Atom 0.8.5
[bcd4f6db-9728-5f36-b5f7-82caef46ccdb] DelayDiffEq 5.2.0
[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
[78ddff82-25fc-5f2b-89aa-309469cbf16f] DiffEqMonteCarlo 0.14.0
[77a26b50-5914-5dd7-bc55-306e6241c503] DiffEqNoiseProcess 3.1.0
[055956cb-9e8b-5191-98cc-73ae4a59e68a] DiffEqPhysics 3.1.0
[a077e3f3-b75c-5d7f-a0c6-6bc4c8ec64a9] DiffEqProblemLibrary 4.1.0
[41bf760c-e81c-5289-8e54-58b1f1f8abe2] DiffEqSensitivity 3.2.2
[0c46a032-eb83-5123-abaf-570d42b7fbaa] DifferentialEquations 6.3.0
[b305315f-e792-5b7a-8f41-49f472929428] Elliptic 0.5.0
[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 3.1.0
[1dea7af3-3e70-54e6-95c3-0bf5283fa5ed] OrdinaryDiffEq 5.6.0
[2dcacdae-9679-587a-88bb-8b444fb7085b] ParallelDataTransfer 0.5.0
[65888b18-ceab-5e60-b2b9-181511a3b968] ParameterizedFunctions 4.1.1
[91a5bcdd-55d7-5caf-9e0b-520d859cae80] Plots 0.24.0
[d330b81b-6aea-500a-939a-2ce795aea3ee] PyPlot 2.8.1
[90137ffa-7385-5640-81b9-e52037218182] StaticArrays 0.10.3
[789caeaf-c7a9-5a7d-9973-96adeb23e2a0] StochasticDiffEq 6.1.1
[c3572dad-4567-51f8-b174-8c6c989267f4] Sundials 3.4.0
[92b13dbe-c966-51a2-8445-caca9f8a7d42] TaylorIntegration 0.4.1
[44d3d7a6-8a23-5bf8-98c5-b353f8df5ec9] Weave 0.9.0
[e88e6eb3-aa80-5325-afca-941959d7151f] Zygote 0.3.0
```