Mackey and Glass Work-Precision Diagrams

David Widmann, Chris Rackauckas

July 16, 2019

1 Model

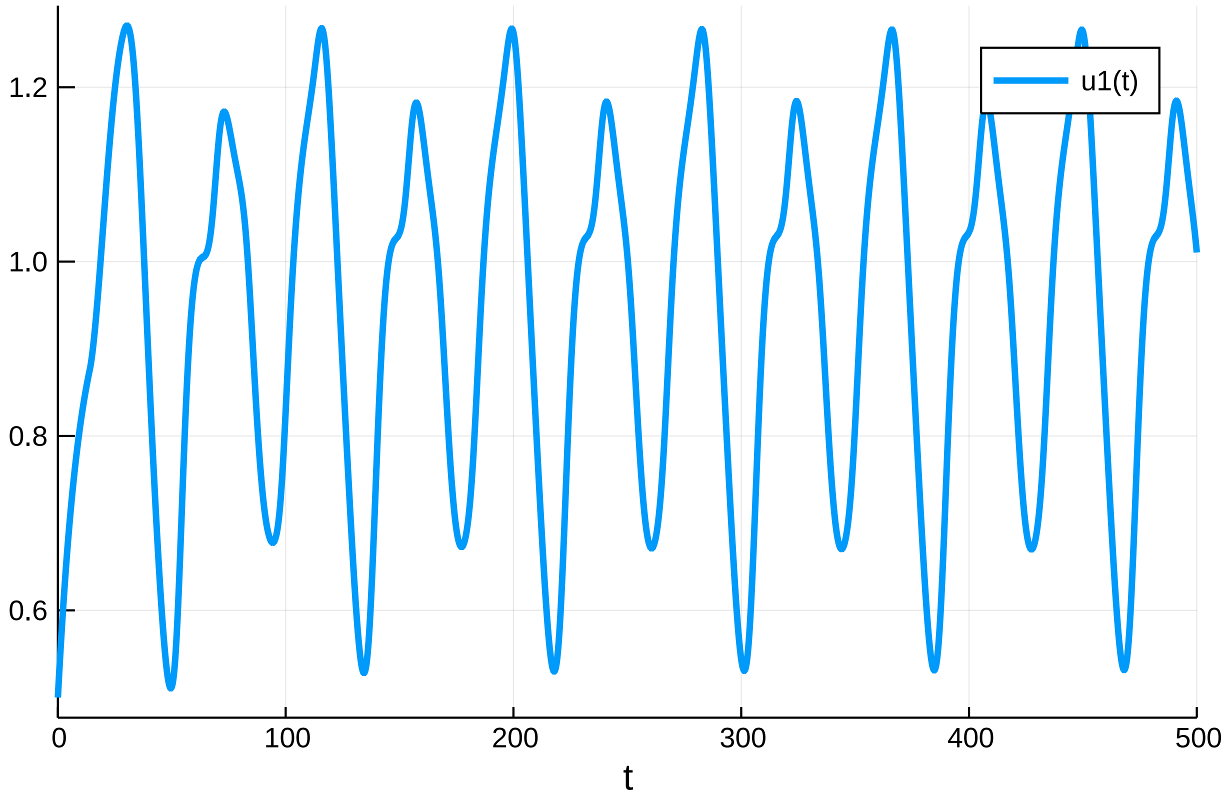
We study a model of blood production that was published by M. C. Mackey and L. Glass in "Oscillation and chaos in physiological control systems", 1977, and is given by

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

for $t \in [0, 500]$ with history function $\phi(t) = 0.5$ for $t \leq 0$. It is test problem A1 of W. H. Enright and H. Hayashi, "The evaluation of numerical software for delay differential equations", 1997.

2 Setup

```
using DelayDiffEq, DiffEqDevTools, DiffEqProblemLibrary.DDEProblemLibrary
DDEProblemLibrary.importddeproblems()

const prob = DDEProblemLibrary.prob_dde_DDETST_A1
const sol = solve(prob, MethodOfSteps(Vern9()); dtmax = 0.1, reltol = 1e-14, abstol =
    1e-14)
const test_sol = TestSolution(sol)

using Plots
gr()
plot(sol)
```



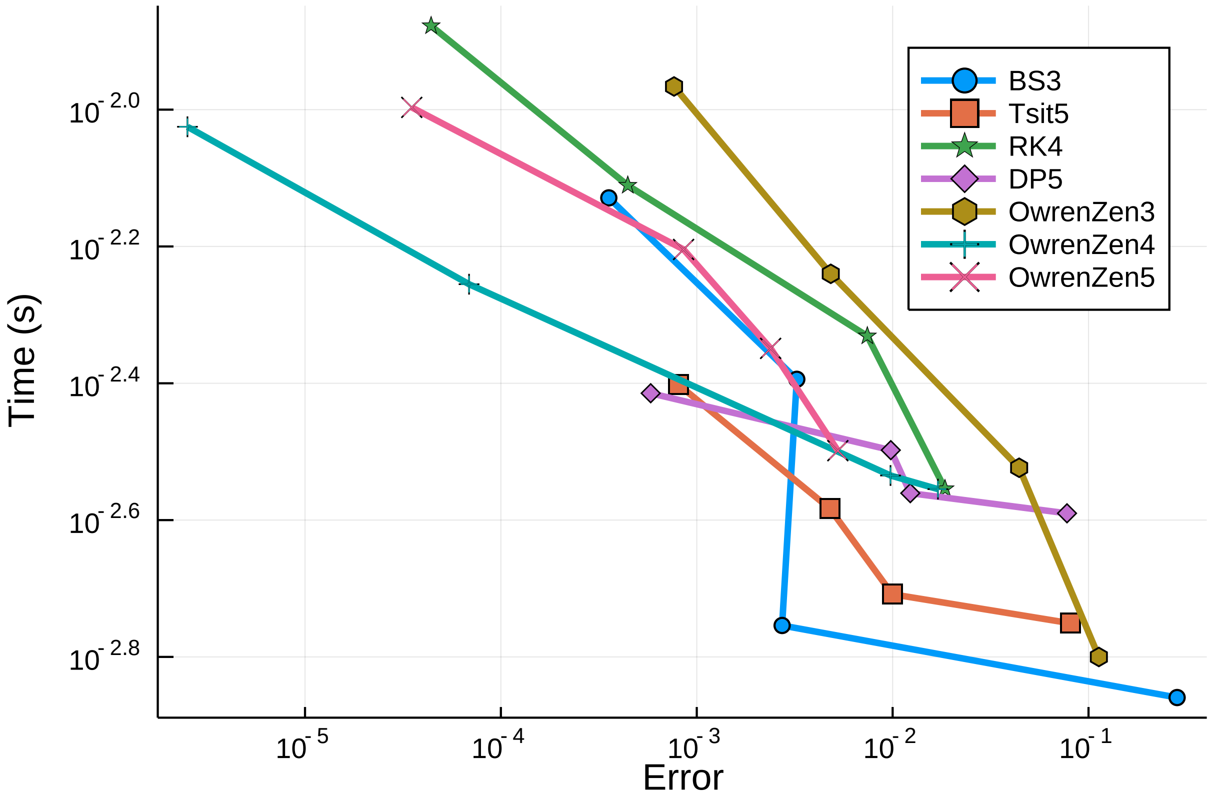

```
function buildWorkPrecisionSet(algs, abstols, reltols; kwargs...)
    setups = [Dict(:alg => MethodOfSteps(alg)) for alg in algs]
    names = nameof.(typeof.(algs))

    WorkPrecisionSet(prob, abstols, reltols, setups;
                     names = names, appxsol = test_sol, maxiters = Int(1e5), kwargs...)
end
```

2.1 Low order RK methods

2.1.1 High tolerances

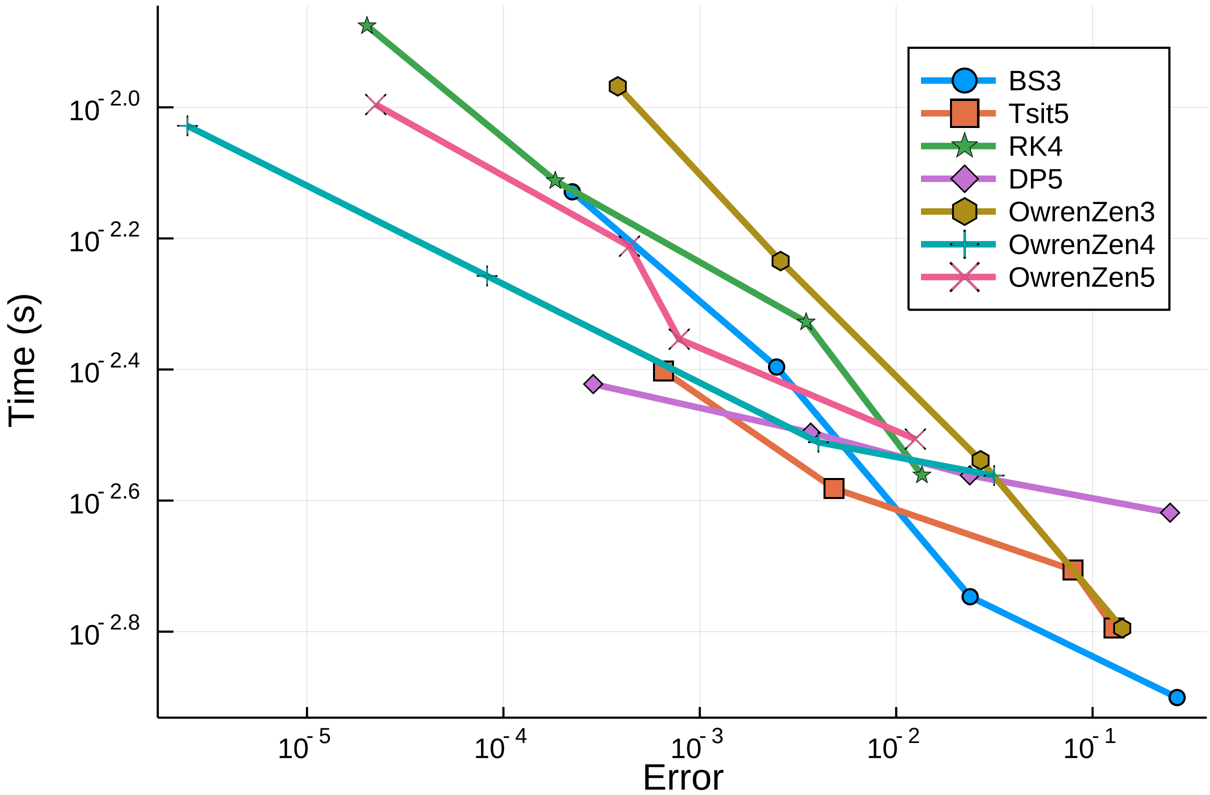
First we test final error estimates of continuous RK methods of low order at high tolerances. `OwrenZen4`, `OwrenZen5`, and `RK4` yield the best error estimates.

```
abstols = @. 1.0 / 10.0^(4:7)
reltols = @. 1.0 / 10.0^(1:4)
algs = [BS3(), Tsit5(), RK4(), DP5(), OwrenZen3(), OwrenZen4(), OwrenZen5()]

wp = buildWorkPrecisionSet(algs, abstols, reltols; 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)
algs = [BS3(), Tsit5(), RK4(), DP5(), OwrenZen3(), OwrenZen4(), OwrenZen5()]

wp = buildWorkPrecisionSet(algs, abstols, reltols; error_estimate = :L2)
plot(wp)
```

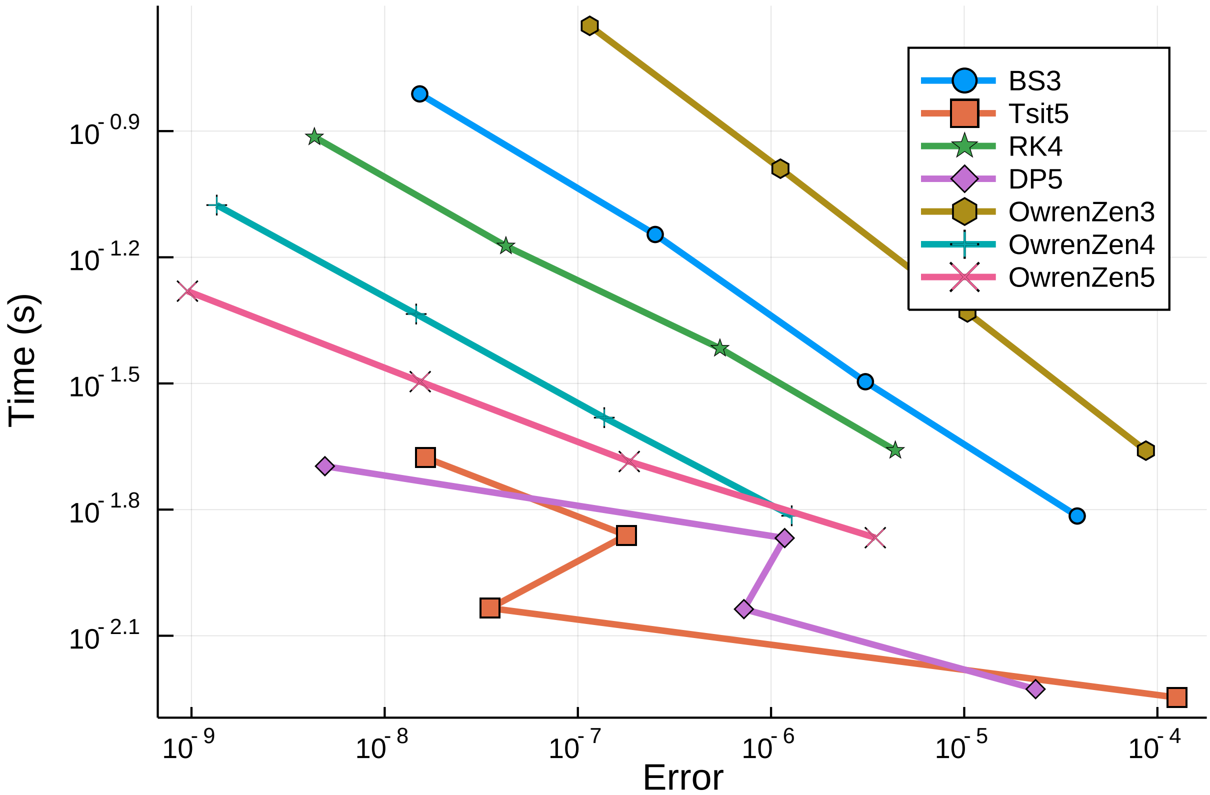


As before, `OwrenZen4` and `OwrenZen5` perform well over the whole range of investigated tolerances.

2.1.2 Low tolerances

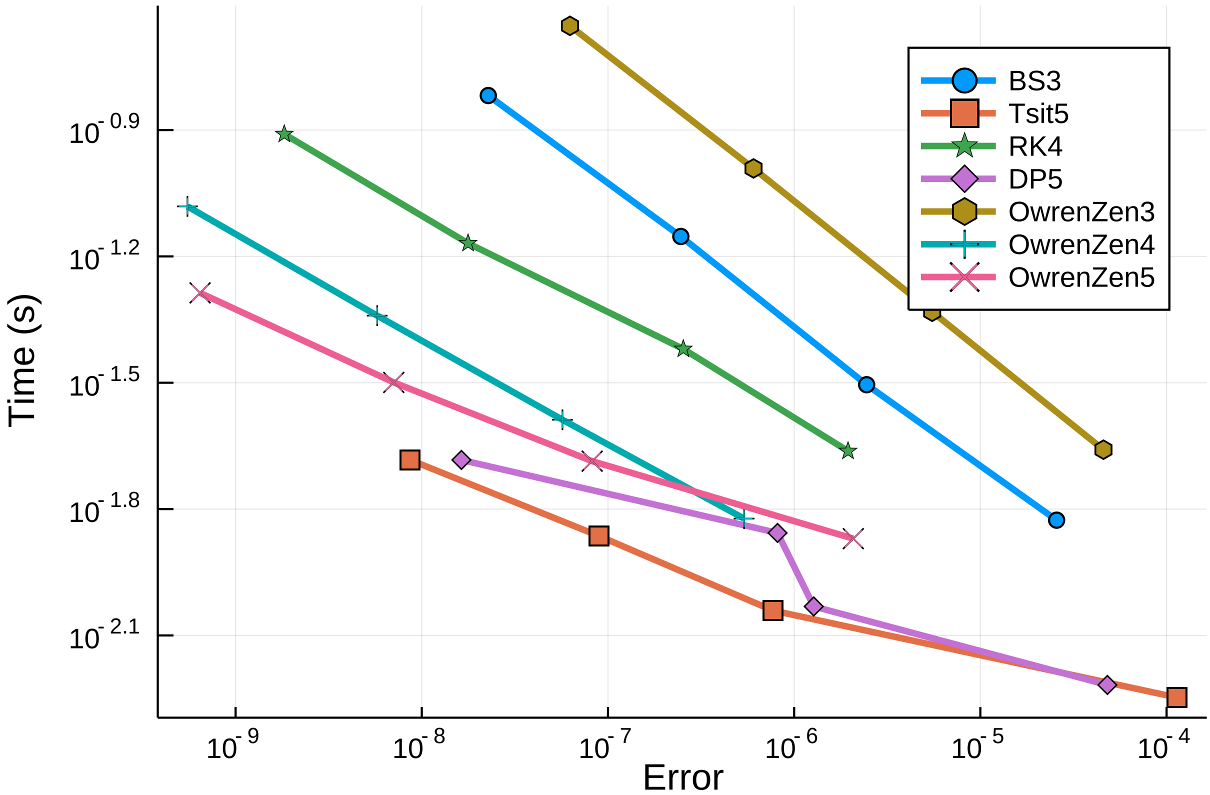
We repeat our tests with low tolerances.

```
abstols = @. 1.0 / 10.0^(8:11)
reltols = @. 1.0 / 10.0^(5:8)
algs = [BS3(), Tsit5(), RK4(), DP5(), OwrenZen3(), OwrenZen4(), OwrenZen5()]

wp = buildWorkPrecisionSet(algs, abstols, reltols; 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)
algs = [BS3(), Tsit5(), RK4(), DP5(), OwrenZen3(), OwrenZen4(), OwrenZen5()]

wp = buildWorkPrecisionSet(algs, abstols, reltols; error_estimate = :L2)
plot(wp)
```



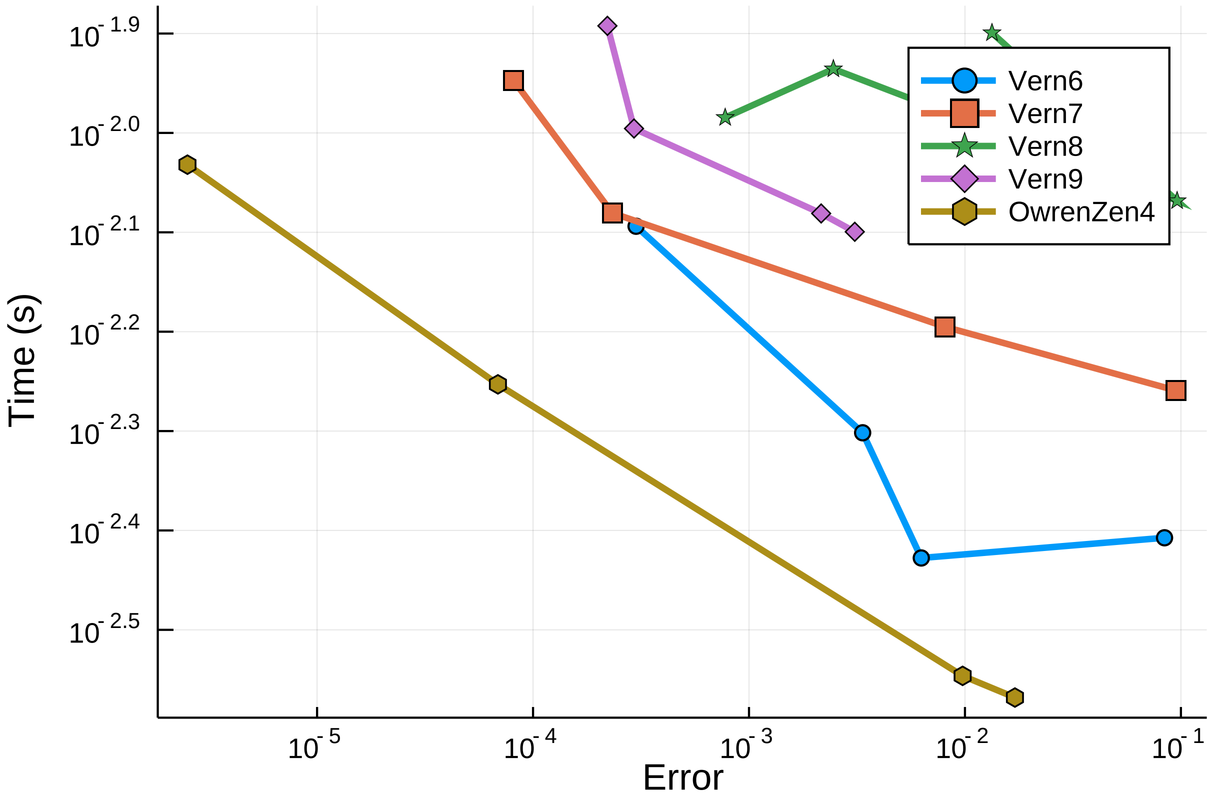
Apparently `Tsit5` and `DP5` perform quite well at low tolerances, but only `OwrenZen5`, `OwrenZen4` and `RK4` achieve interpolation errors of around 1e-9.

2.2 Lazy interpolants

2.2.1 High tolerances

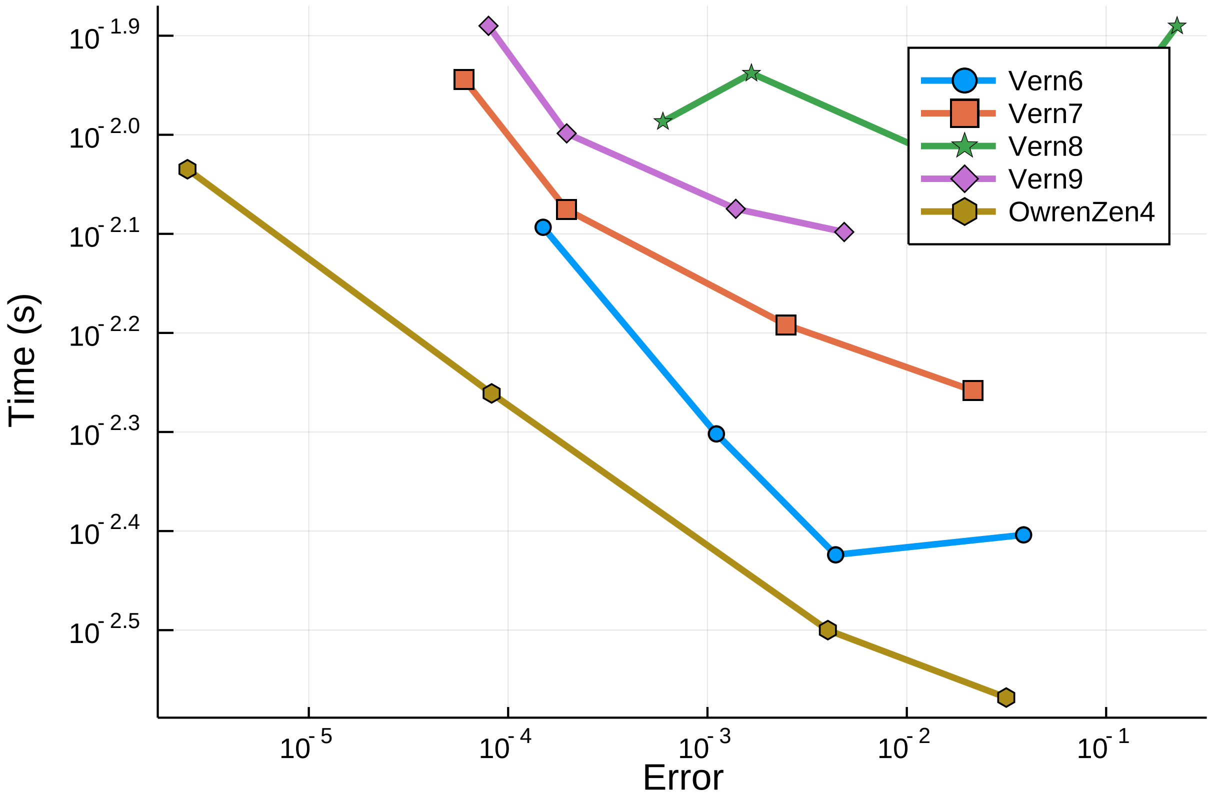
We repeat our tests with the Verner methods which, in contrast to the methods above, use lazy interpolants. As reference we include `OwrenZen4`.

```
abstols = @. 1.0 / 10.0^(4:7)
reltols = @. 1.0 / 10.0^(1:4)
algs = [Vern6(), Vern7(), Vern8(), Vern9(), OwrenZen4()]

wp = buildWorkPrecisionSet(algs, abstols, reltols; 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)
algs = [Vern6(), Vern7(), Vern8(), Vern9(), OwrenZen4()]

wp = buildWorkPrecisionSet(algs, abstols, reltols; error_estimate = :L2)
plot(wp)
```

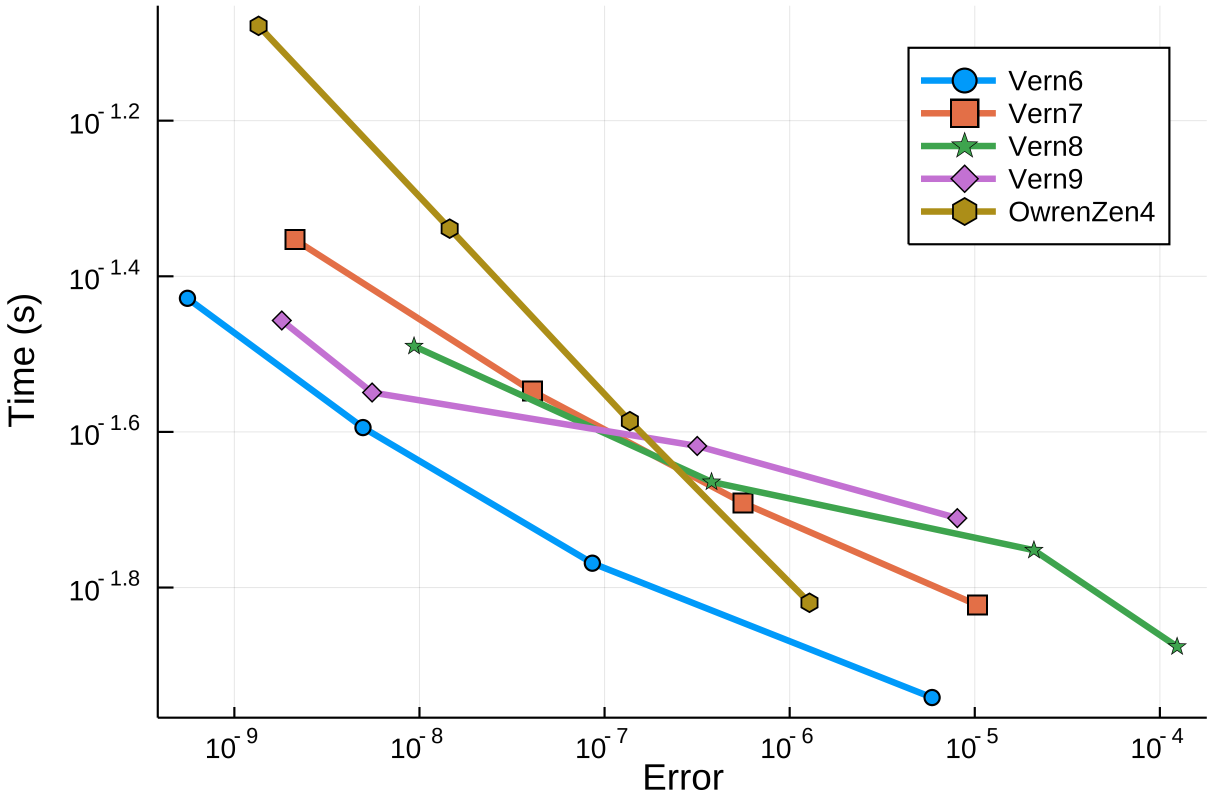


`Vern6`, `Vern7`, and `Vern9` are outperformed by `OwrenZen4`.

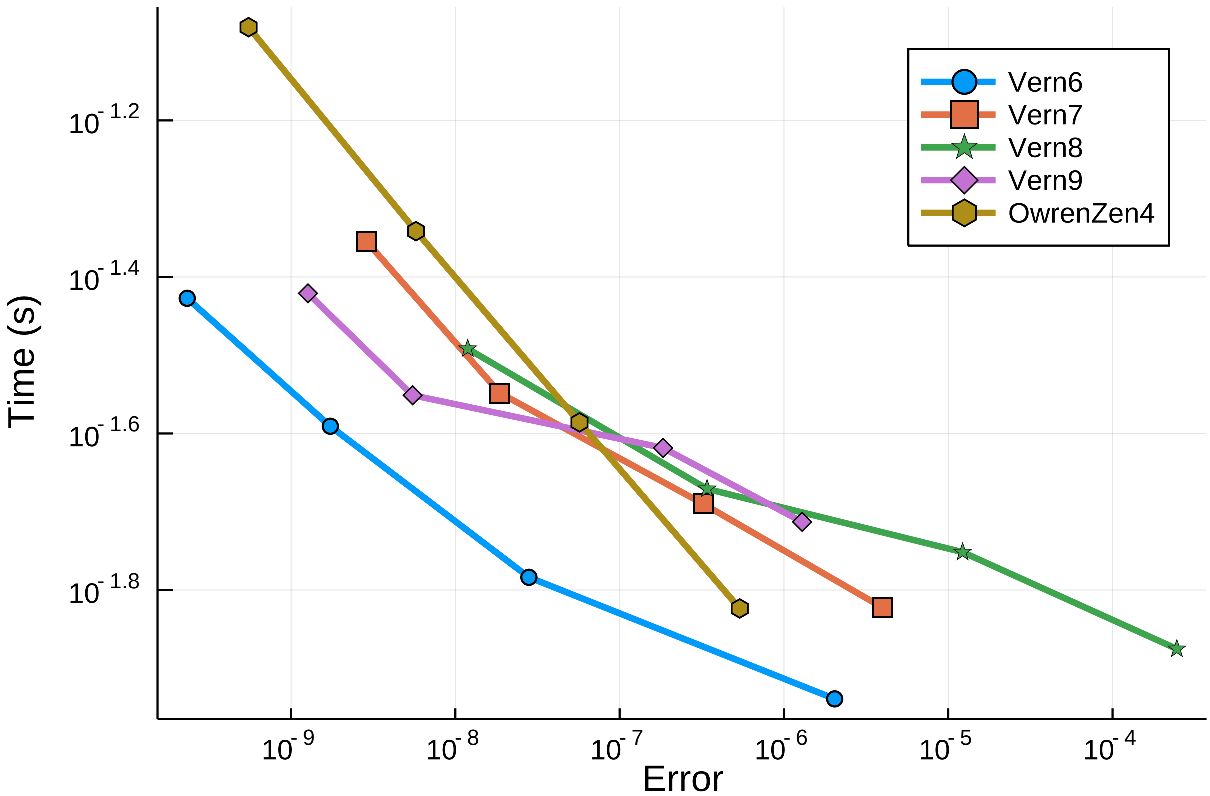
2.2.2 Low tolerances

Again, we repeat our tests at low tolerances.

```
abstols = @. 1.0 / 10.0^(8:11)
reltols = @. 1.0 / 10.0^(5:8)
algs = [Vern6(), Vern7(), Vern8(), Vern9(), OwrenZen4()]

wp = buildWorkPrecisionSet(algs, abstols, reltols; error_estimate = :final)
plot(wp)
```




```
abstols = @. 1.0 / 10.0^(8:11)
reltols = @. 1.0 / 10.0^(5:8)
algs = [Vern6(), Vern7(), Vern8(), Vern9(), OwrenZen4()]

wp = buildWorkPrecisionSet(algs, abstols, reltols; error_estimate = :L2)
plot(wp)
```



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

2.3 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 1.1.1
Commit 55e36cc308 (2019-05-16 04:10 UTC)
Platform Info:
  OS: Linux (x86_64-pc-linux-gnu)
  CPU: Intel(R) Core(TM) i7-6850K CPU @ 3.60GHz
  WORD_SIZE: 64
  LIBM: libopenlibm
  LLVM: libLLVM-6.0.1 (ORCJIT, broadwell)
```

Package Information:

```
Status: `/home/davwi492/Projects/DiffEqBenchmarks/Project.toml`
[a134a8b2-14d6-55f6-9291-3336d3ab0209] BlackBoxOptim 0.4.0
[bcd4f6db-9728-5f36-b5f7-82caef46ccdb] DelayDiffEq 5.7.0
[f3b72e0c-5b89-59e1-b016-84e28bfd966d] DiffEqDevTools 2.13.0
[78ddff82-25fc-5f2b-89aa-309469cbf16f] DiffEqMonteCarlo 0.15.1
[1130ab10-4a5a-5621-a13d-e4788d82bd4c] DiffEqParamEstim 1.6.0
[a077e3f3-b75c-5d7f-a0c6-6bc4c8ec64a9] DiffEqProblemLibrary 4.5.0
[ef61062a-5684-51dc-bb67-a0fcdec5c97d] DiffEqUncertainty 1.1.0
[7073ff75-c697-5162-941a-fcdaad2a7d2a] IJulia 1.18.1
[7f56f5a3-f504-529b-bc02-0b1fe5e64312] LSODA 0.4.0
[76087f3c-5699-56af-9a33-bf431cd00edd] NLopt 0.5.1
[c030b06c-0b6d-57c2-b091-7029874bd033] ODE 2.4.0
[54ca160b-1b9f-5127-a996-1867f4bc2a2c] ODEInterface 0.4.6
[09606e27-ecf5-54fc-bb29-004bd9f985bf] ODEInterfaceDiffEq 3.3.1
[1dea7af3-3e70-54e6-95c3-0bf5283fa5ed] OrdinaryDiffEq 5.12.0
[65888b18-ceab-5e60-b2b9-181511a3b968] ParameterizedFunctions 4.2.0
[91a5bcdd-55d7-5caf-9e0b-520d859cae80] Plots 0.26.0
[c3572dad-4567-51f8-b174-8c6c989267f4] Sundials 3.6.1
[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
```