

# Fitzhugh-Nagumo Work-Precision Diagrams

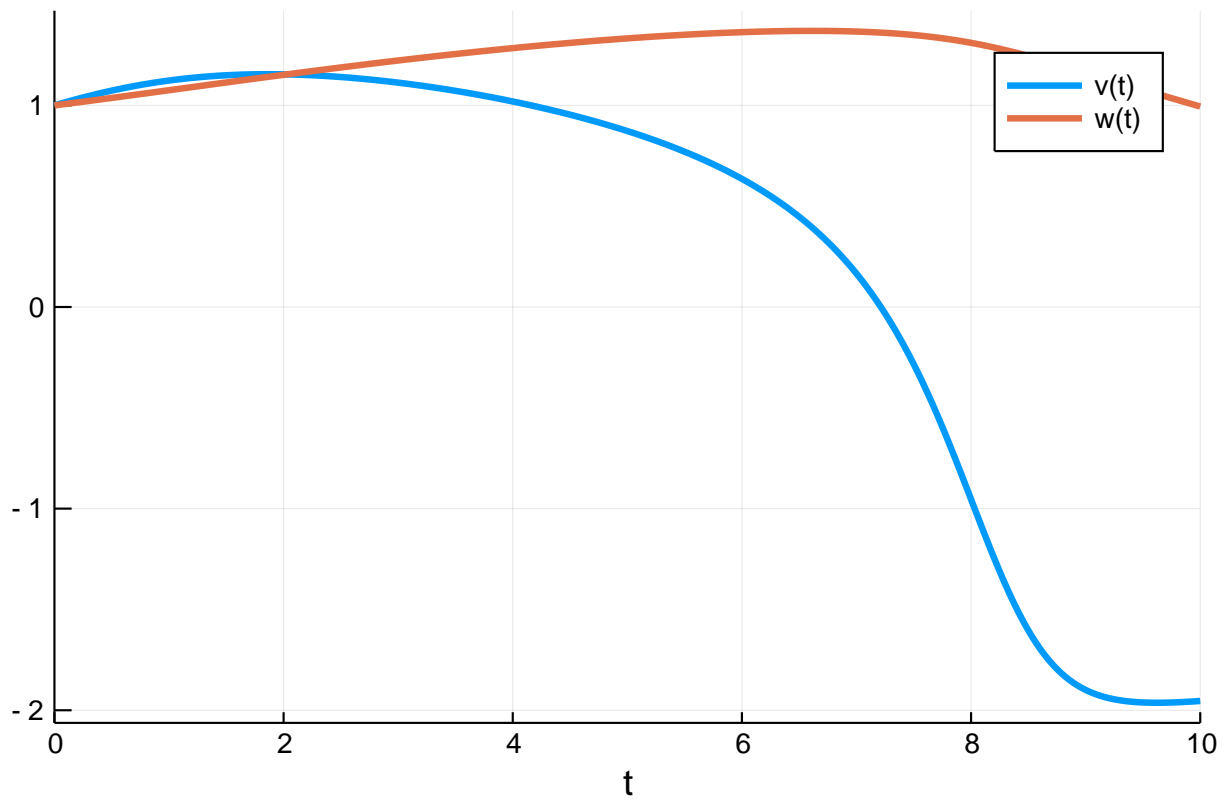
Chris Rackauckas

September 26, 2019

## 1 Fitzhugh-Nagumo

The purpose of this is to see how the errors scale on a standard nonlinear problem.

```
using OrdinaryDiffEq, ParameterizedFunctions, ODE, ODEInterface,  
    ODEInterfaceDiffEq, LSODA, Sundials, DiffEqDevTools  
  
f = @ode_def FitzhughNagumo begin  
    dv = v - v^3/3 - w + 1  
    dw = τinv*(v + a - b*w)  
end a b τinv 1  
  
p = [0.7,0.8,1/12.5,0.5]  
prob = ODEProblem(f,[1.0;1.0],(0.0,10.0),p)  
  
abstols = 1.0 ./ 10.0 .^ (6:13)  
reltols = 1.0 ./ 10.0 .^ (3:10);  
  
sol = solve(prob,Vern7(),abstol=1/10^14,reltol=1/10^14)  
test_sol = TestSolution(sol)  
using Plots; gr()  
  
plot(sol)
```

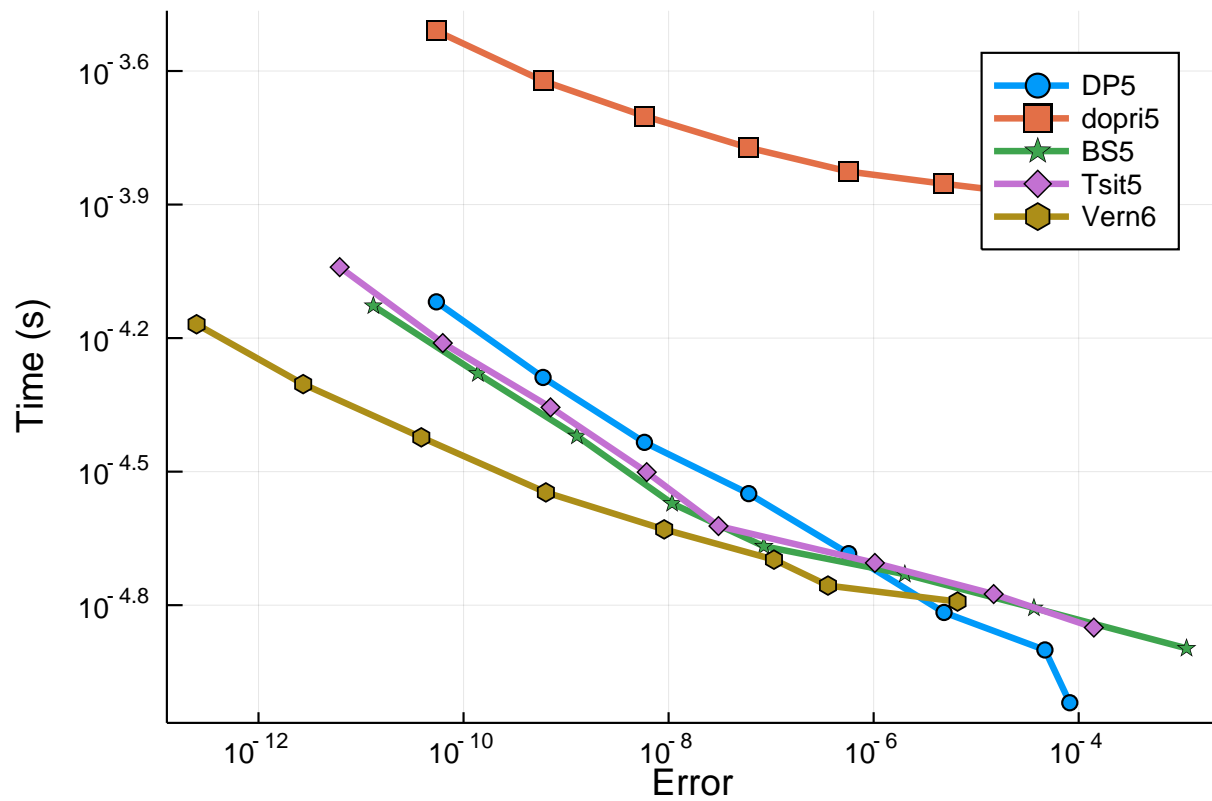


## 1.1 Low Order

```

setups = [Dict(:alg=>DP5())
           #Dict(:alg=>ode45()) #fails
           Dict(:alg=>dopri5())
           Dict(:alg=>BS5())
           Dict(:alg=>Tsit5())
           Dict(:alg=>Vern6())
]
wp =
  WorkPrecisionSet(prob, abstols, reltols, setups; appxsol=test_sol, save_everystep=false, numruns=100, max
plot(wp)

```

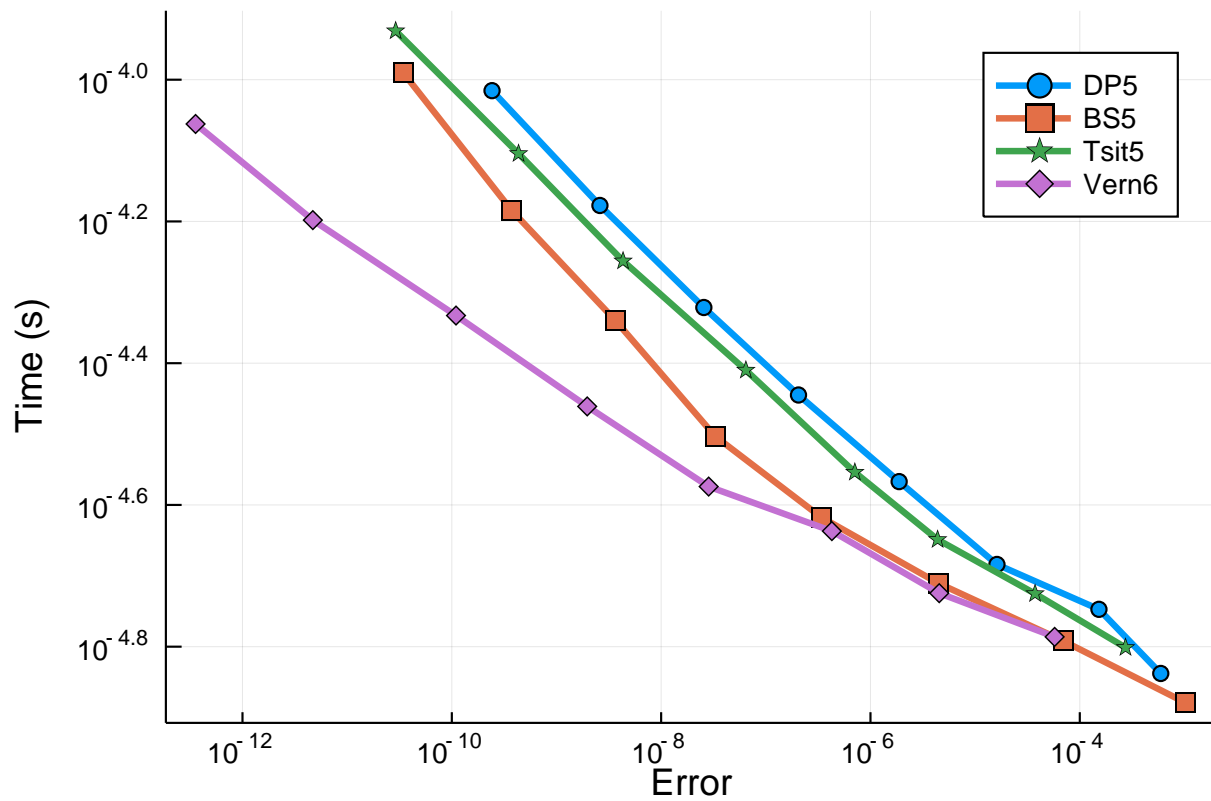


### 1.1.1 Interpolation

```

setups = [Dict(:alg=>DP5())
           #Dict(:alg=>ode45()) # fails
           Dict(:alg=>BS5())
           Dict(:alg=>Tsit5())
           Dict(:alg=>Vern6())
]
wp =
  WorkPrecisionSet(prob, abstols, reltols, setups; appxsol=test_sol, numruns=100, maxiters=10000, error_est=
plot(wp)

```

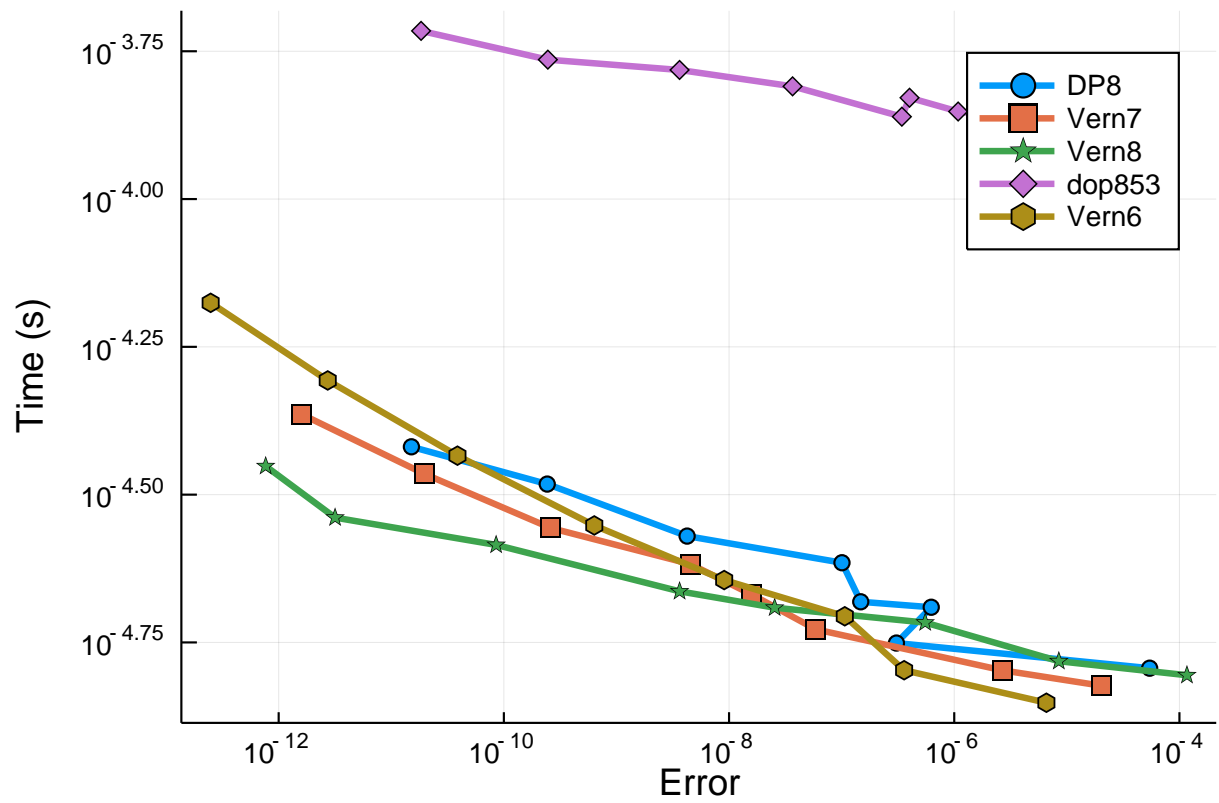


## 1.2 Higher Order

```

setups = [Dict(:alg=>DP8())
           #Dict(:alg=>ode78()) # fails
           Dict(:alg=>Vern7())
           Dict(:alg=>Vern8())
           Dict(:alg=>dop853())
           Dict(:alg=>Vern6())
]
wp =
  WorkPrecisionSet(prob, abstols, reltols, setups; appxsol=test_sol, save_everystep=false, numruns=100, max
plot(wp)

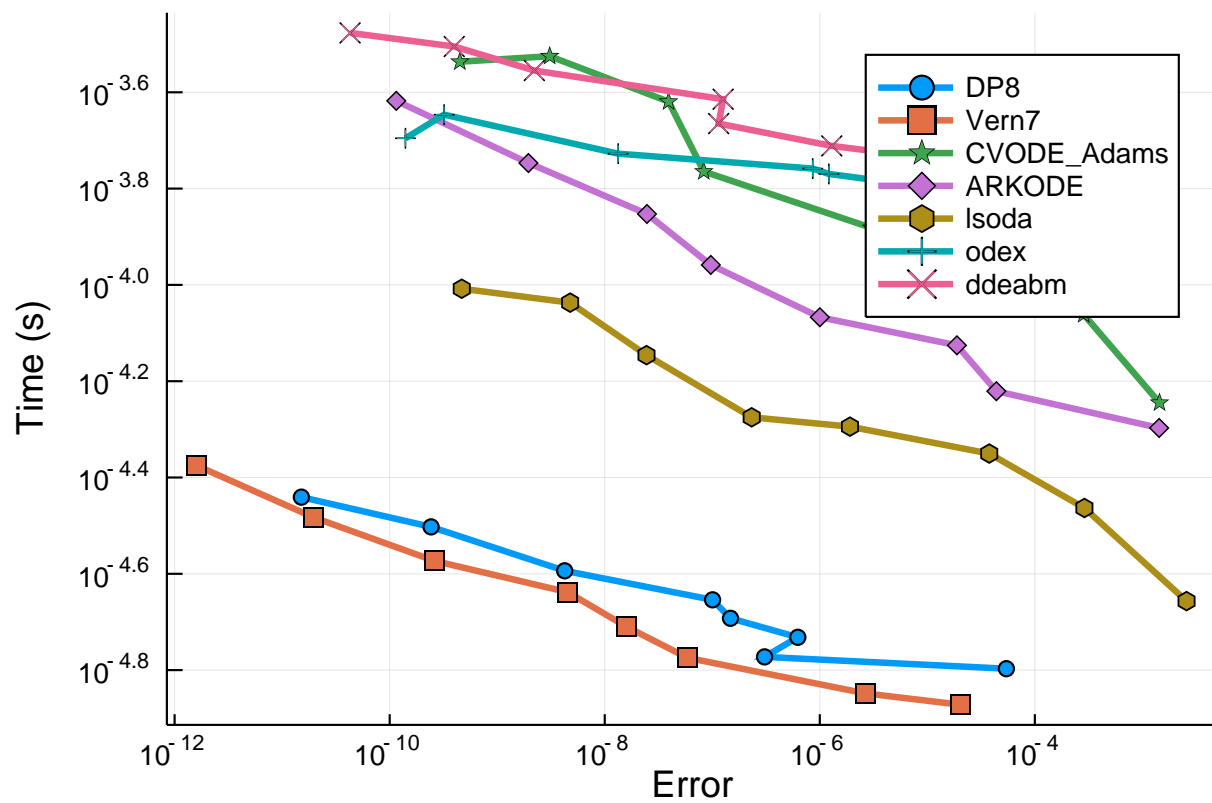
```



```

setups = [Dict(:alg=>DP8())
          Dict(:alg=>Vern7())
          Dict(:alg=>CVMODE_Adams())
          Dict(:alg=>ARKODE(Sundials.Explicit(),order=6))
          Dict(:alg=>lsoda())
          Dict(:alg=>odex())
          Dict(:alg=>ddeabm())
]
wp =
  WorkPrecisionSet(prob,abstols,reltols,setups;appxsol=test_sol,save_everystep=false,numruns=100,max
plot(wp)

```



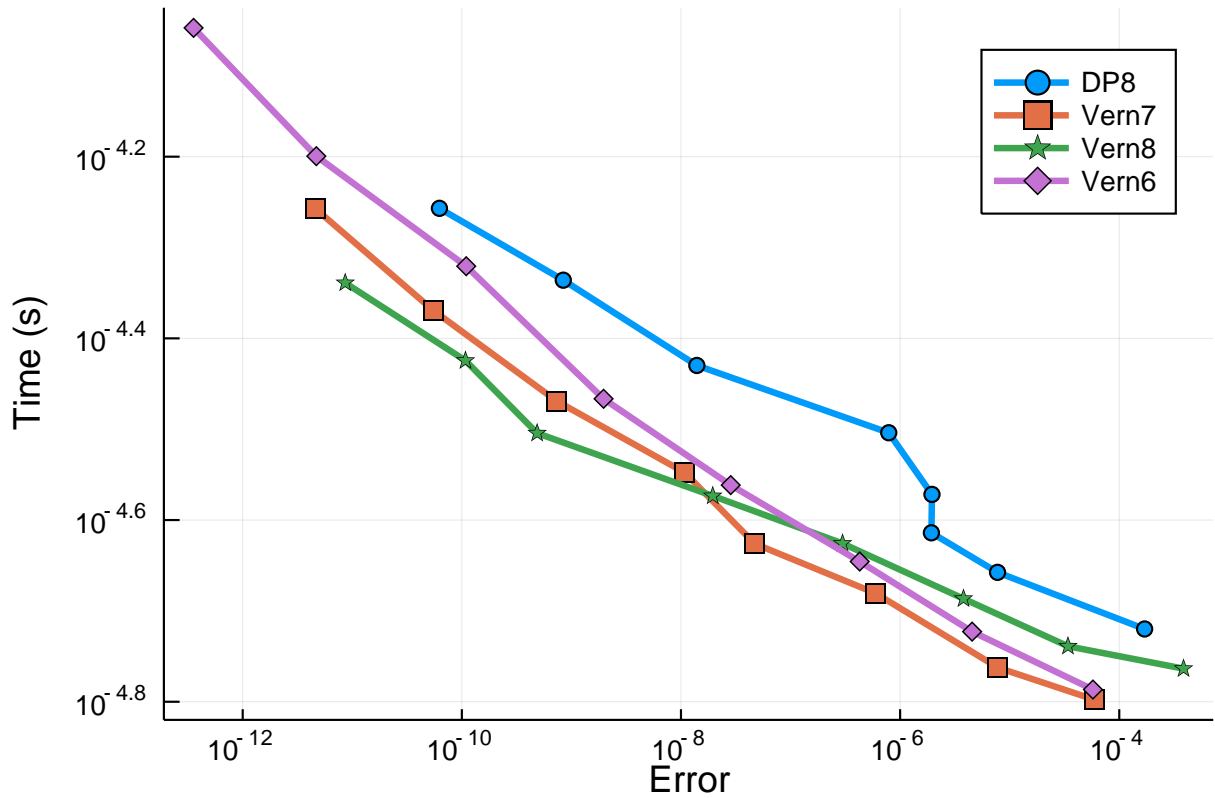
### 1.2.1 Interpolation

```

setups = [Dict(:alg=>DP8())
           #Dict(:alg=>ode78()) # fails
           Dict(:alg=>Vern7())
           Dict(:alg=>Vern8())
           Dict(:alg=>Vern6())
]

wp =
    WorkPrecisionSet(prob, abstols, reltols, setups; appxsol=test_sol, numruns=100, maxiters=1000, error_esti
plot(wp)

```



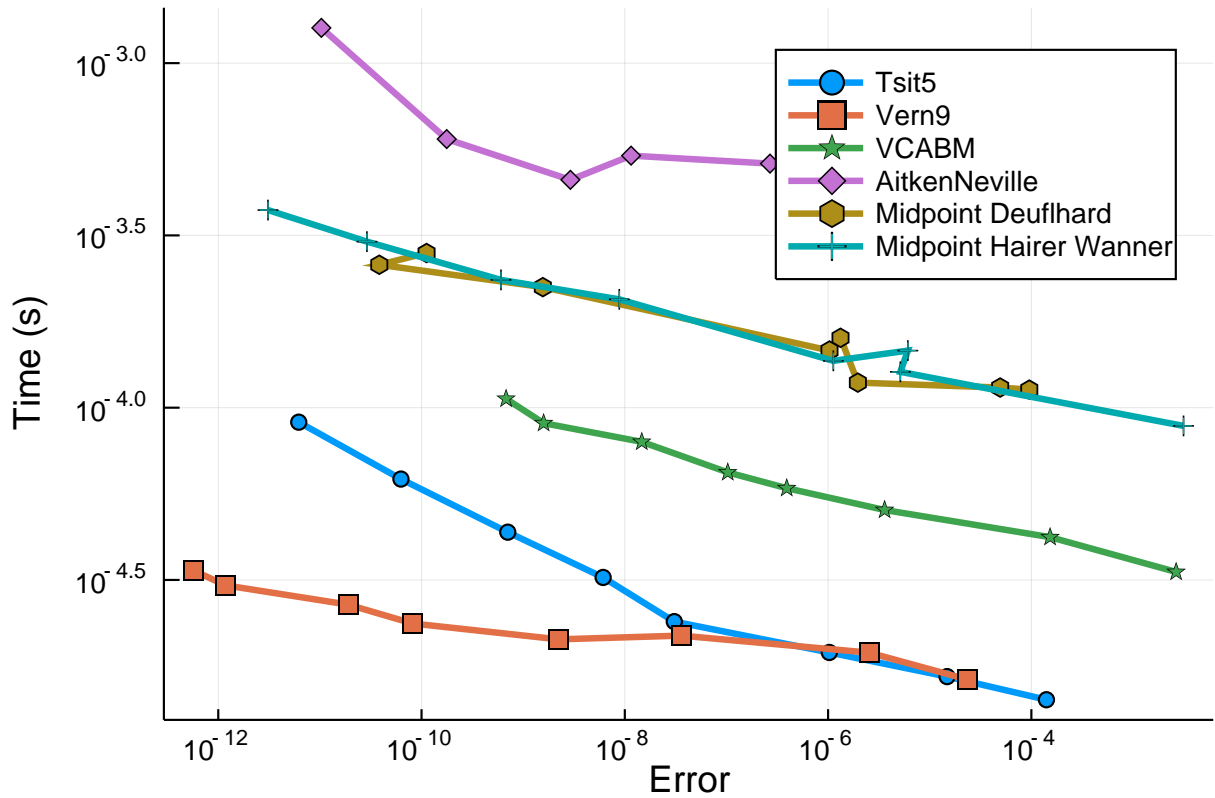
### 1.3 Comparison with Non-RK methods

Now let's test Tsit5 and Vern9 against parallel extrapolation methods and an Adams-Bashforth-Moulton:

```

setups = [Dict(:alg=>Tsit5())
          Dict(:alg=>Vern9())
          Dict(:alg=>VCABM())
          Dict(:alg=>AitkenNeville(min_order=1, max_order=9, init_order=4,
threading=true))
          Dict(:alg=>ExtrapolationMidpointDeuflhard(min_order=1, max_order=9,
init_order=4, threading=true))
          Dict(:alg=>ExtrapolationMidpointHairerWanner(min_order=2, max_order=11,
init_order=4, threading=true))]
solnames = ["Tsit5", "Vern9", "VCABM", "AitkenNeville", "Midpoint Deuflhard", "Midpoint Hairer
Wanner"]
wp = WorkPrecisionSet(prob, abstols, reltols, setups; appxsol=test_sol, names=solnames,
save_everystep=false, verbose=false, numruns=100)
plot(wp)

```

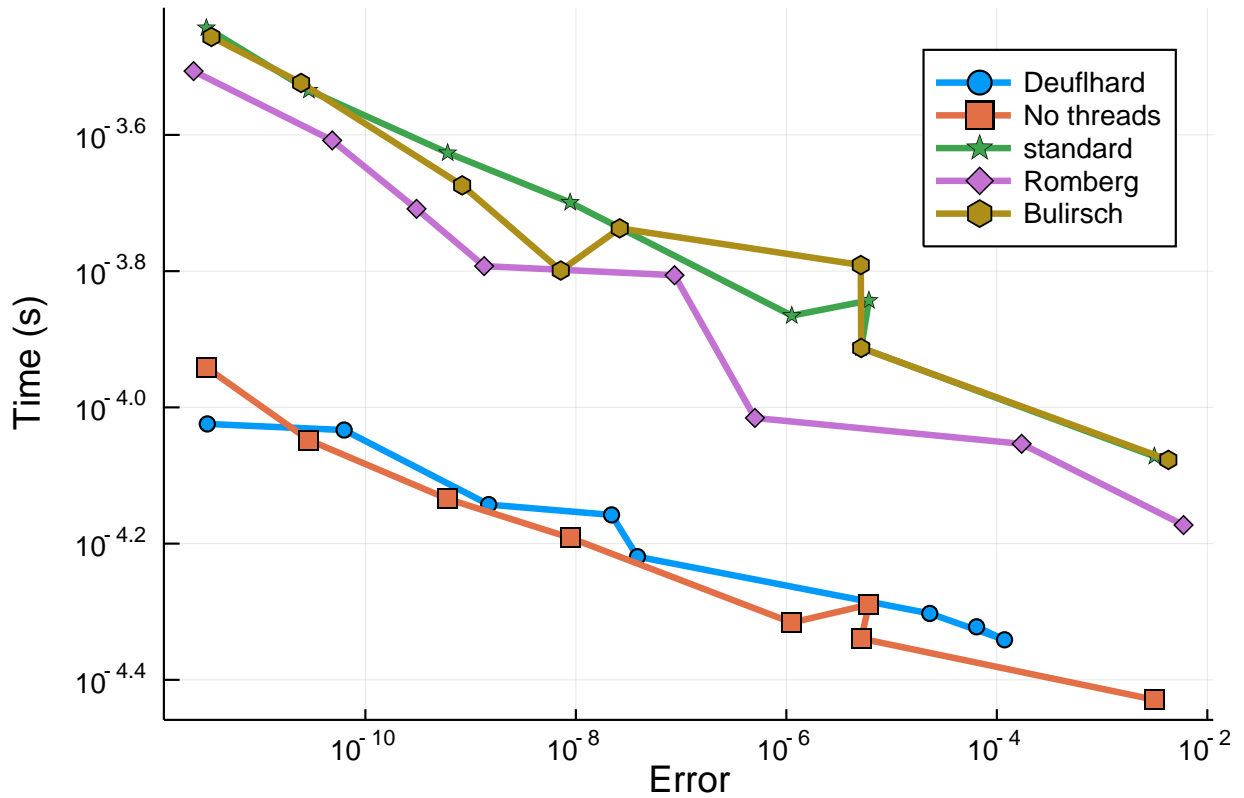


```

setups = [Dict(:alg=>ExtrapolationMidpointDeuflhard(min_order=1, max_order=9,
  init_order=9, threading=false))
  Dict(:alg=>ExtrapolationMidpointHairerWanner(min_order=2, max_order=11,
  init_order=4, threading=false))
  Dict(:alg=>ExtrapolationMidpointHairerWanner(min_order=2, max_order=11,
  init_order=4, threading=true))
  Dict(:alg=>ExtrapolationMidpointHairerWanner(min_order=2, max_order=11,
  init_order=4, sequence = :romberg, threading=true))
  Dict(:alg=>ExtrapolationMidpointHairerWanner(min_order=2, max_order=11,
  init_order=4, sequence = :bulirsch, threading=true))]
solnames = ["Deuflhard", "No threads", "standard", "Romberg", "Bulirsch"]
wp = WorkPrecisionSet(prob, abstols, reltols, setups; appxsol=test_sol, names=solnames,
  save_everystep=false, verbose=false, numruns=100)
plot(wp)

```

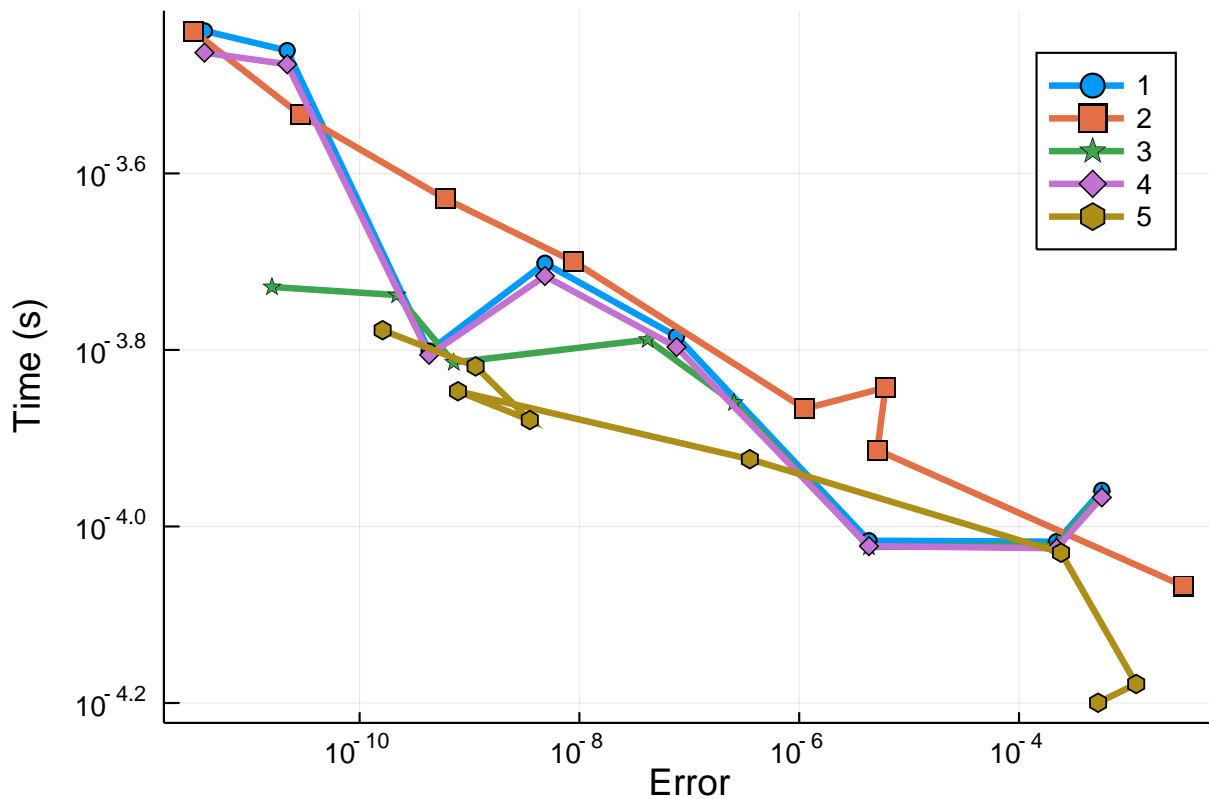




```

setups = [Dict(:alg=>ExtrapolationMidpointHairerWanner(min_order=2, max_order=11,
    init_order=10, threading=true))
    Dict(:alg=>ExtrapolationMidpointHairerWanner(min_order=2, max_order=11,
    init_order=4, threading=true))
    Dict(:alg=>ExtrapolationMidpointHairerWanner(min_order=5, max_order=11,
    init_order=10, threading=true))
    Dict(:alg=>ExtrapolationMidpointHairerWanner(min_order=2, max_order=15,
    init_order=10, threading=true))
    Dict(:alg=>ExtrapolationMidpointHairerWanner(min_order=5, max_order=7,
    init_order=6, threading=true))]
solnames = ["1", "2", "3", "4", "5"]
wp = WorkPrecisionSet(prob, abstols, reltols, setups; appxsol=test_sol, names=solnames,
    save_everystep=false, verbose=false, numruns=100)
plot(wp)

```



## 1.4 Conclusion

As expected, the algorithms are all pretty matched on time for this problem. However, you can clearly see the OrdinaryDiffEq.jl algorithms solving to a much higher accuracy and still faster, especially when the interpolations are involved.

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

## 1.5 Appendix

These benchmarks are a part of the DiffEqBenchmarks.jl repository, found at: <https://github.com/JuliaDiffeq/DiffEqBenchmarks.jl>

To locally run this tutorial, do the following commands:

```
using DiffEqBenchmarks
DiffEqBenchmarks.weave_file("NonStiffODE","FitzhughNagumo_wpd.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)

#### 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
```