Fitzhugh-Nagumo Work-Precision Diagrams

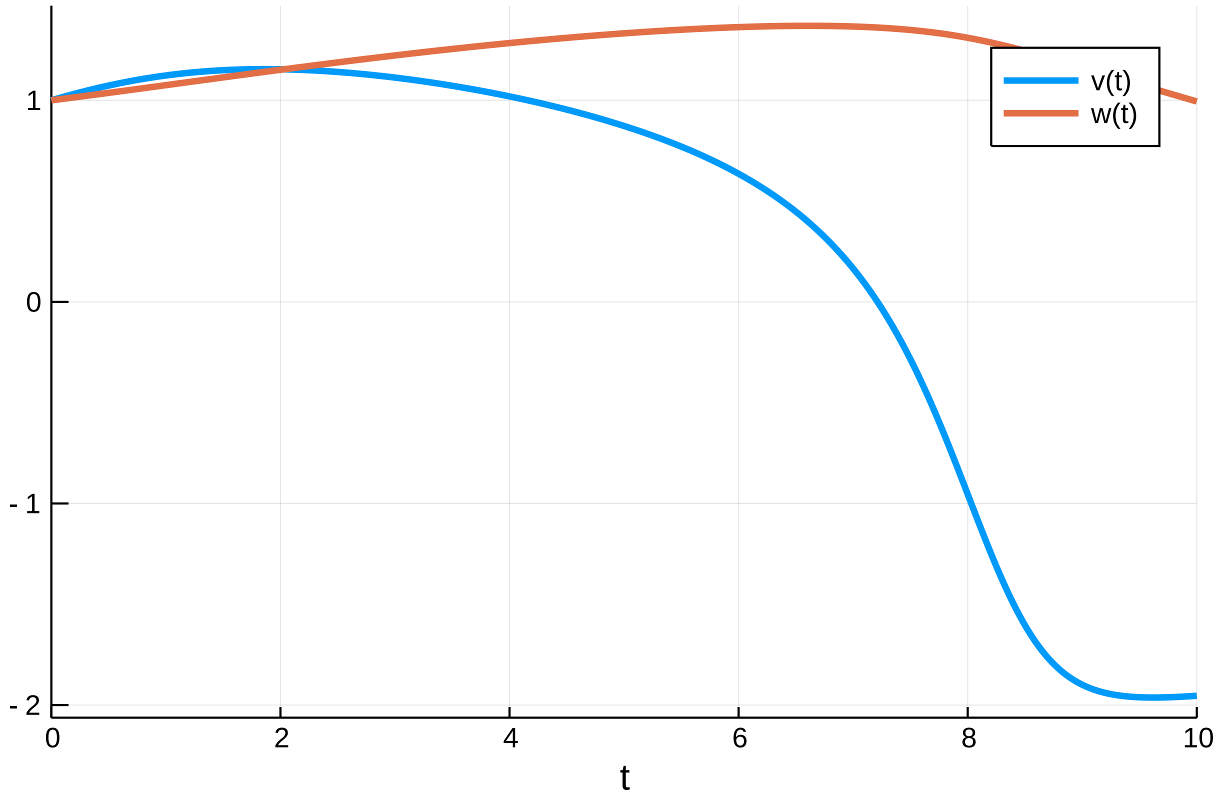
Chris Rackauckas

June 12, 2019

1 Fitzhugh-Nagumo

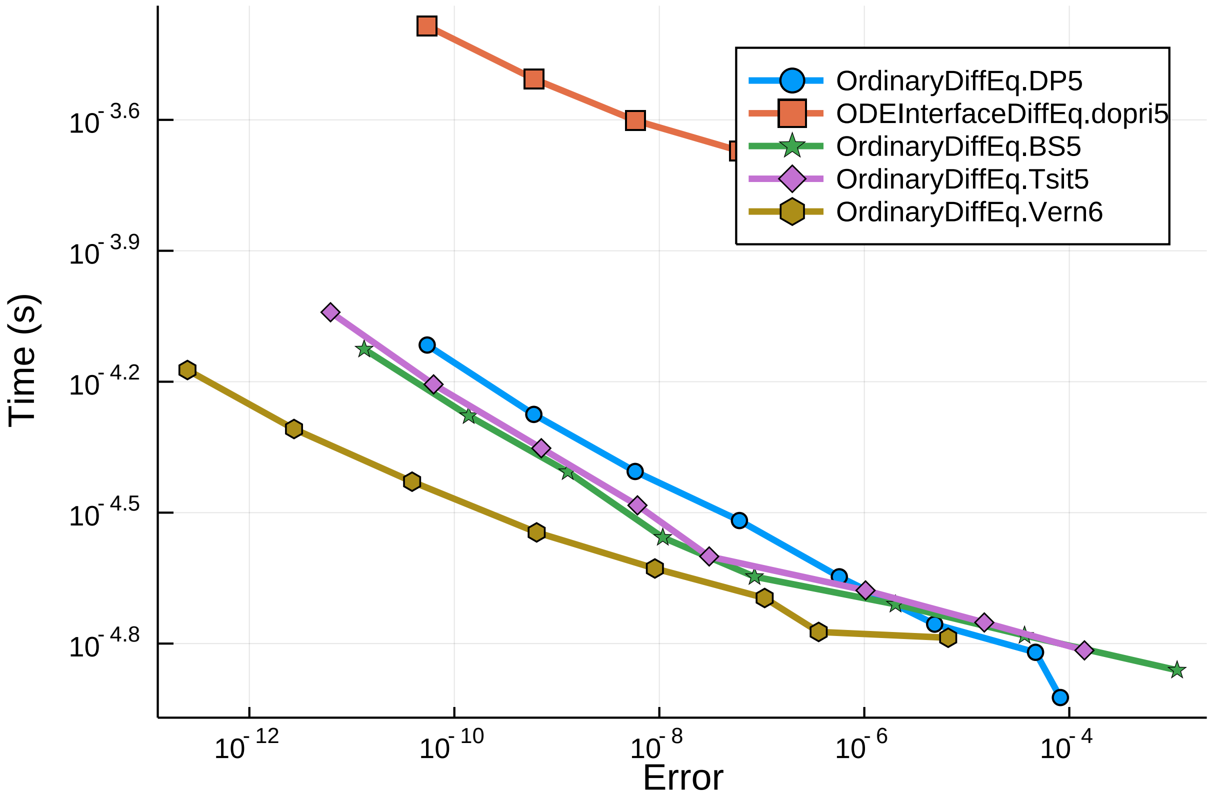
The purpose of this is to see how the errors scale on a standard nonlinear problem.

```
using OrdinaryDiffEq, ParameterizedFunctions, ODE, ODEInterfaceDiffEq,
      LSODA, Sundials, DiffEqDevTools

f = @ode_def FitzhughNagumo begin
  dv = v - v^3/3 -w + l
  dw = τinv*(v +  a - b*w)
end a b τinv l

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)
```



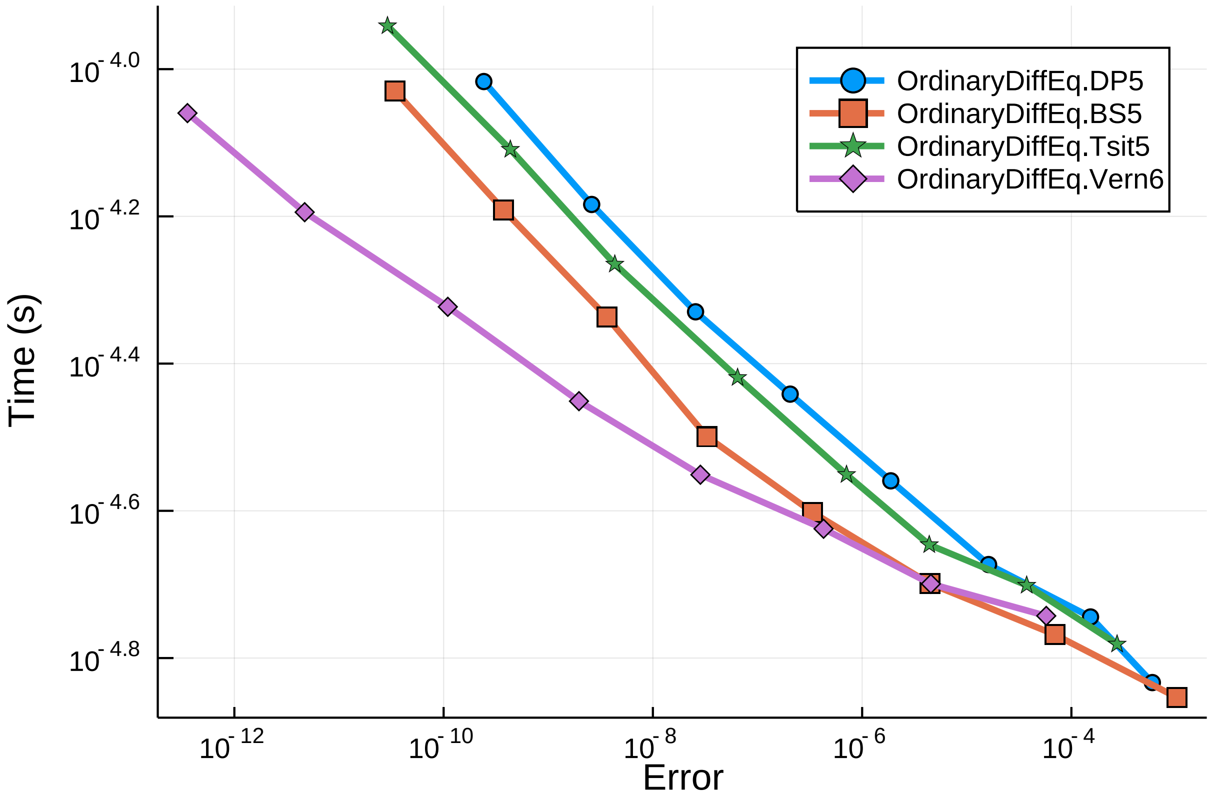
1.1 Low Order

```
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)
```



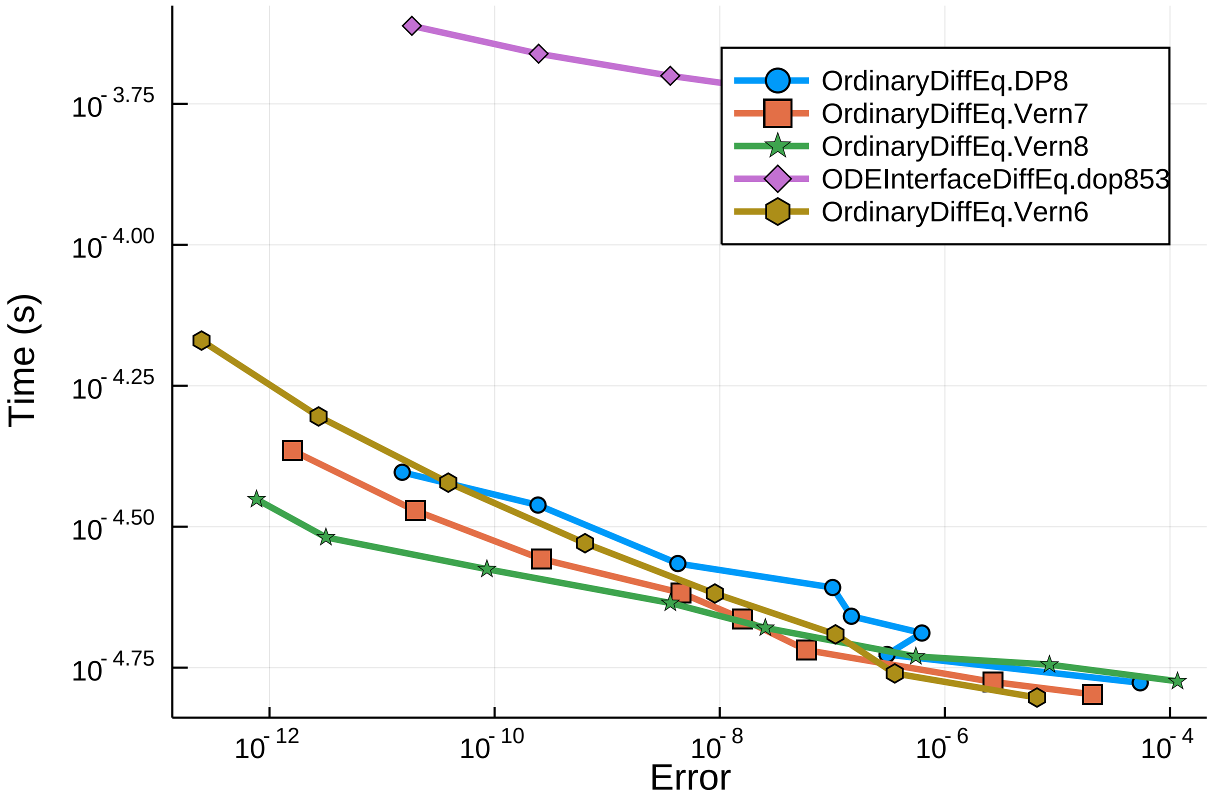
1.1.1 Interpolation

```
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)
```

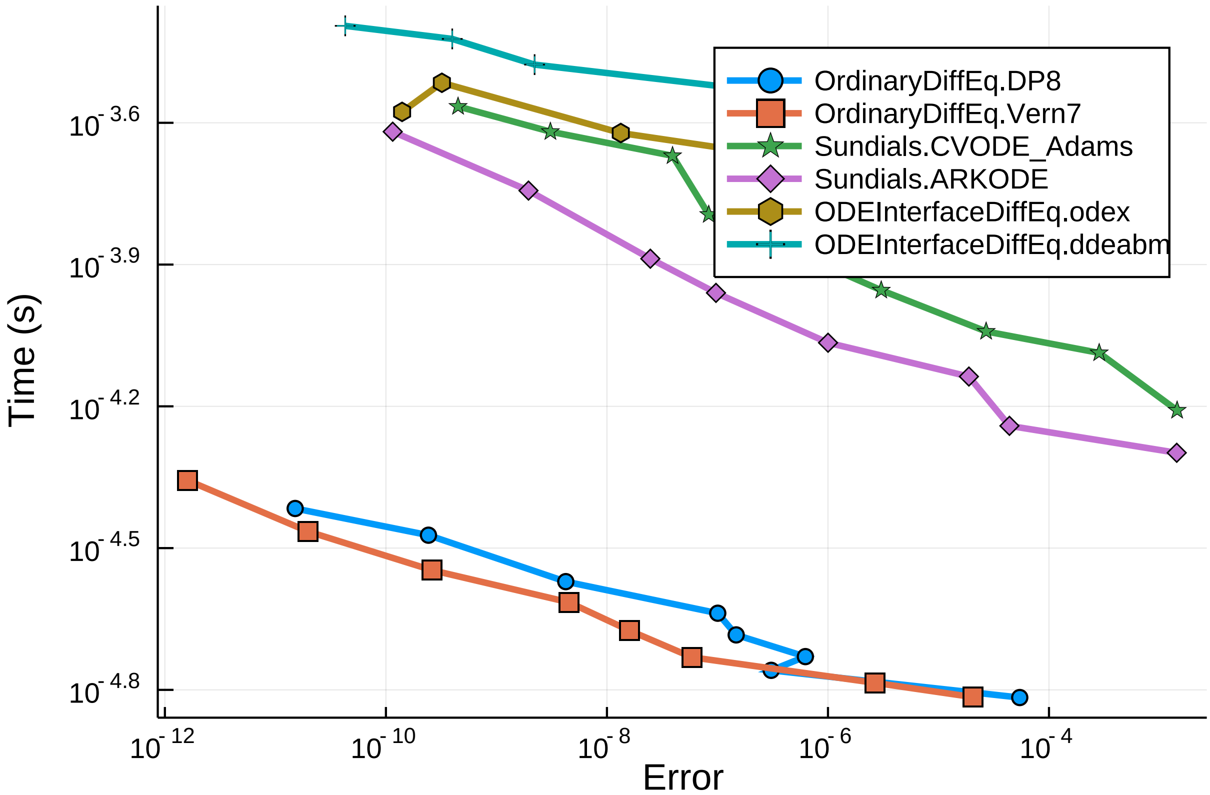


1.2 Higher Order

```
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)
```

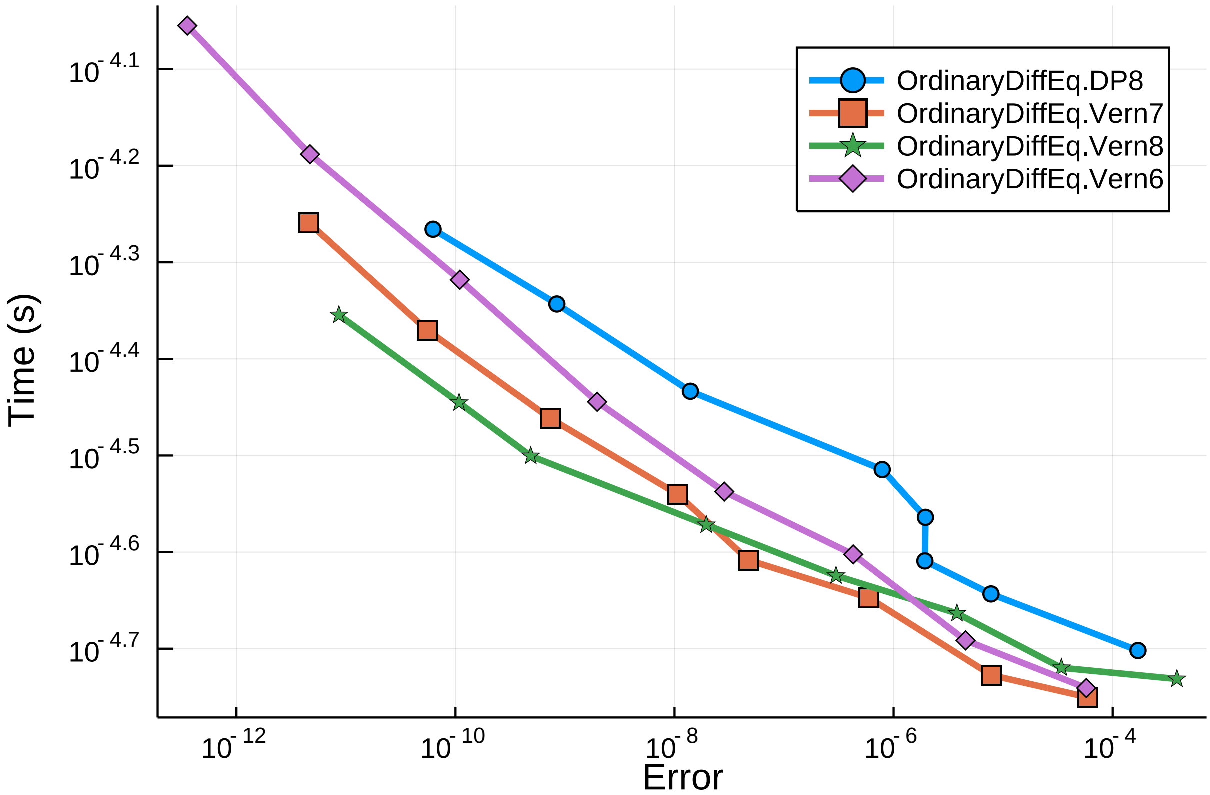



```
setups = [Dict(:alg=>DP8())
          Dict(:alg=>Vern7())
          Dict(:alg=>CVODE_Adams())
          Dict(:alg=>ARKODE(Sundials.Explicit(),order=6))
          #Dict(:alg=>lsoda())
          Dict(:alg=>odex())
          Dict(:alg=>ddeabm())
]
wp =
    WorkPrecisionSet(prob,abstols,reltols,setups;appxsol=test_sol,save_everystep=false,numruns=100,max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

1.4 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("NonStiffODE","FitzhughNagumo_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 0.8.7
[bcd4f6db-9728-5f36-b5f7-82caef46ccdb] DelayDiffEq 5.3.0
[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
[aae7a2af-3d4f-5e19-a356-7da93b79d9d0] DiffEqFlux 0.5.0
[78ddff82-25fc-5f2b-89aa-309469cbf16f] DiffEqMonteCarlo 0.14.0
[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
[41bf760c-e81c-5289-8e54-58b1f1f8abe2] DiffEqSensitivity 3.2.2
[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 3.3.0
[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 0.25.1
[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 3.6.0
[92b13dbe-c966-51a2-8445-caca9f8a7d42] TaylorIntegration 0.5.0
[44d3d7a6-8a23-5bf8-98c5-b353f8df5ec9] Weave 0.9.0
[e88e6eb3-aa80-5325-afca-941959d7151f] Zygote 0.3.1
```