SDE Basic Weak Work-Precision Diagrams

Chris Rackauckas

July 23, 2019

1 SDE Basic Weak Work-Precision Diagrams

In this notebook we will run some benchmarks for the weak error on some simple sample SDEs. The weak error is defined as:

$$E_W = \mathbb{E}[Y_\delta(t)] - \mathbb{E}[Y(t)]$$

and is thus a measure of how close the mean of the numerical solution is to the mean of the true solution. Other moments can be measured as well, but the mean is a good stand-in for other properties. Note that convergence of the mean is calculated on a sample. Thus there's acutally two sources of error. We have not only the error between the numerical and actual results, but we also have the error of the mean to the true mean due to only taking a finite sample. Using the normal confidence interval of the mean due to the Central Limit Theorem, the error due to finite sampling is

$$E_S = V[Y(t)]/\sqrt(N)$$

for N being the number of samples. In practice,

$$E = minimum(E_W, E_S)$$

Thus in each case, we will determine the variance of the true solution and use that to estimate the sample error, and the goal is to thus find the numerical method that achieves the sample error most efficiently.

```
using StochasticDiffEq, DiffEqDevTools, ParameterizedFunctions, DiffEqProblemLibrary
using Plots; gr()
using DiffEqProblemLibrary.SDEProblemLibrary: importsdeproblems; importsdeproblems()
import DiffEqProblemLibrary.SDEProblemLibrary: prob_sde_additive,
            prob_sde_linear, prob_sde_wave
const N = 1000
```

```
1000
```

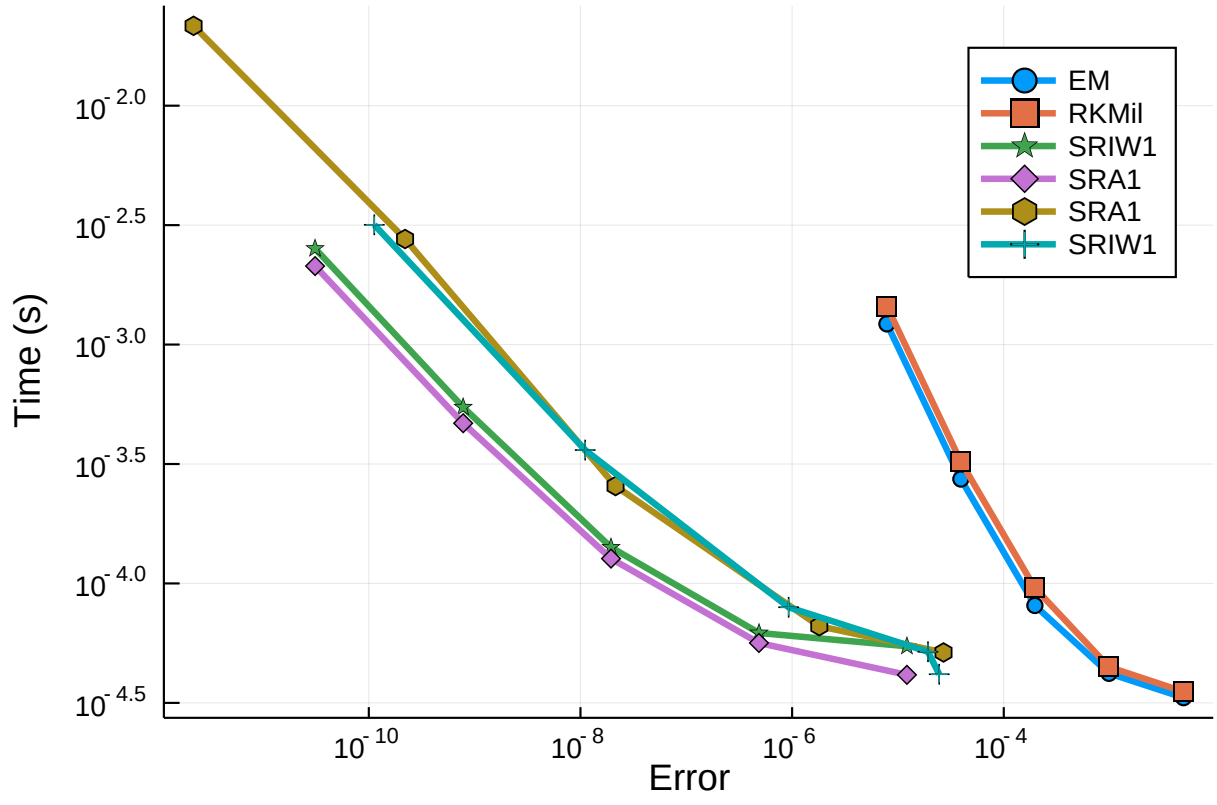
1.0.1 Additive Noise Problem

$$dX_t = \left(\frac{\beta}{\sqrt{1+t}} - \frac{1}{2\left(1+t\right)}X_t\right)dt + \frac{\alpha\beta}{\sqrt{1+t}}dW_t, \quad X_0 = \frac{1}{2}$$

where $\alpha = \frac{1}{10}$ and $\beta = \frac{1}{20}$. Actual Solution:

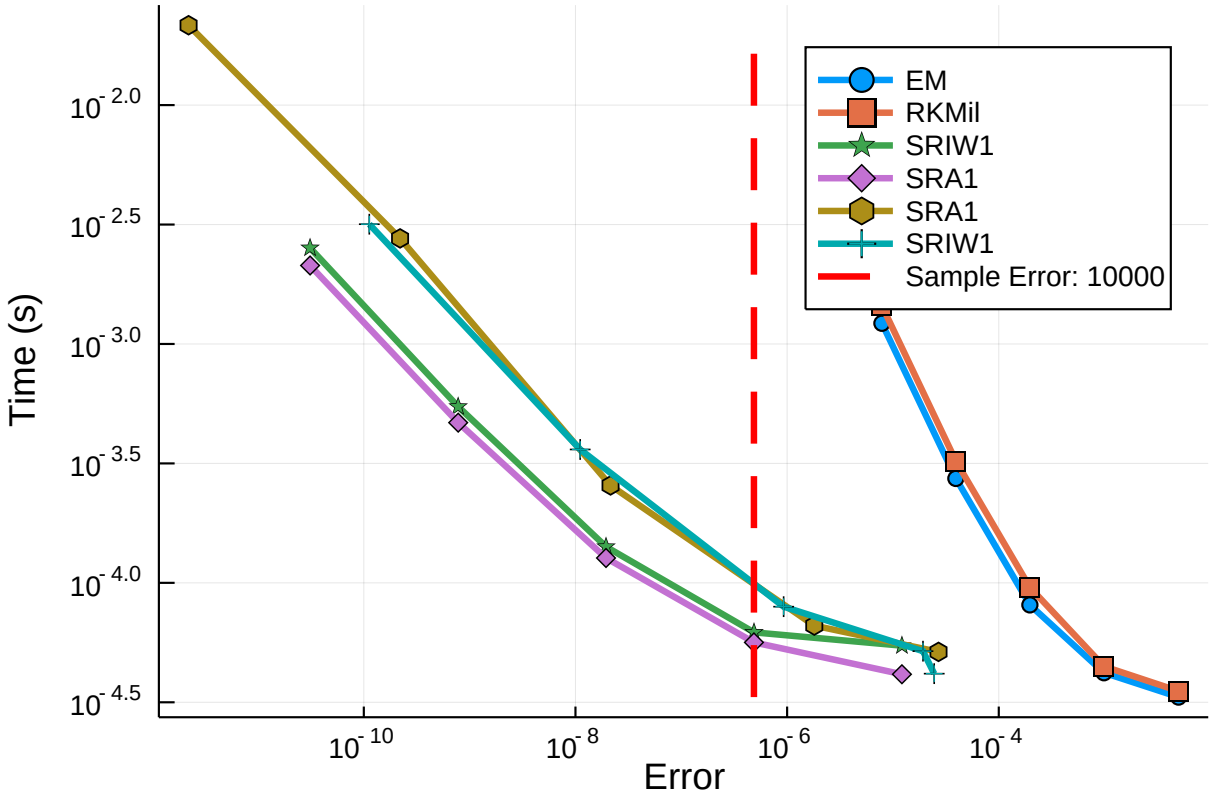
$$X_t = \frac{1}{\sqrt{1+t}} X_0 + \frac{\beta}{\sqrt{1+t}} \left(t + \alpha W_t\right).$$

```
prob = prob_sde_additive

reltols = 1.0 ./ 10.0 .^ (1:5)
abstols = reltols#[0.0 for i in eachindex(reltols)]
setups = [
          Dict(:alg=>EM(),:dts=>1.0./5.0.^((1:length(reltols)) .+ 1))
          Dict(:alg=>RKMil(),:dts=>1.0./5.0.^((1:length(reltols)) .+ 1),:adaptive=>false)
          Dict(:alg=>SRIW1(),:dts=>1.0./5.0.^((1:length(reltols)) .+ 1),:adaptive=>false)
          Dict(:alg=>SRA1(),:dts=>1.0./5.0.^((1:length(reltols)) .+ 1),:adaptive=>false)
          Dict(:alg=>SRA1())
          Dict(:alg=>SRIW1())
          ]
wp = WorkPrecisionSet(prob,abstols,reltols,setups;numruns_error=N,
                      save_everystep = false,
                      parallel_type = :none,
                      error_estimate=:weak_final)#
plot(wp)
```

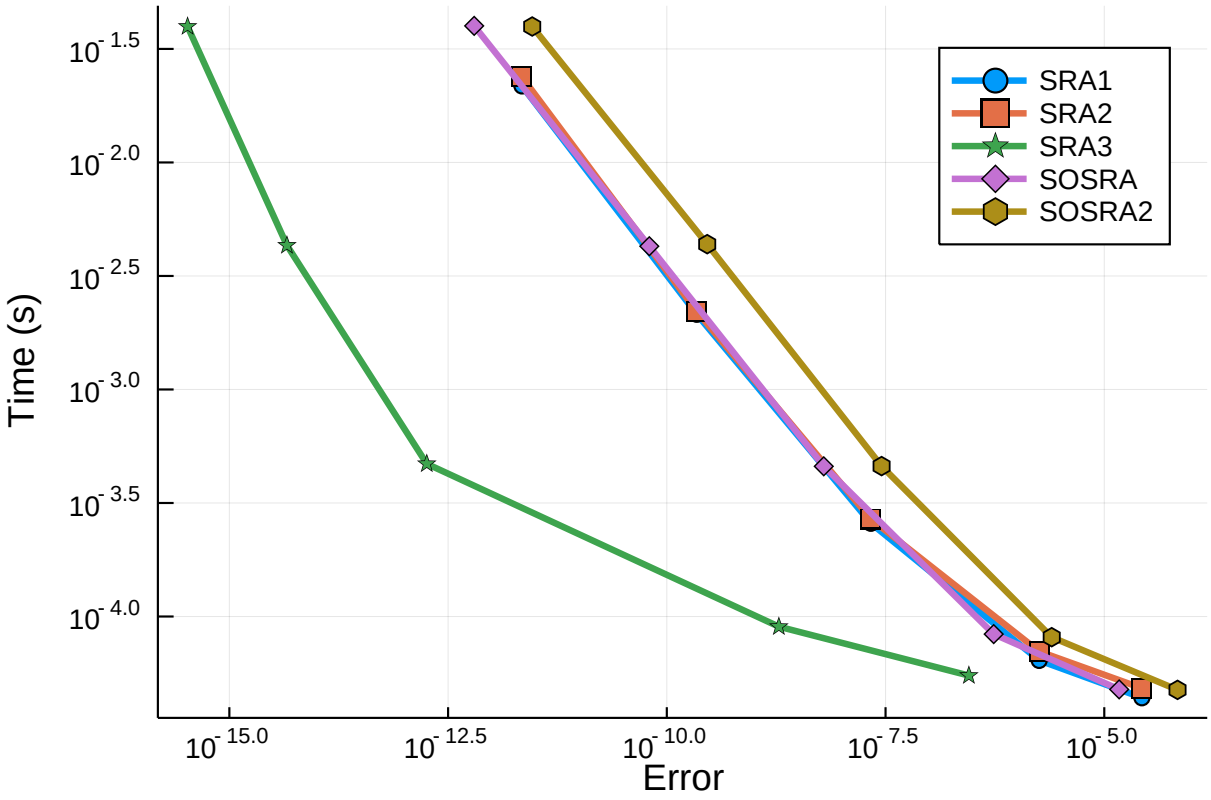



```
sample_size = Int[10;1e2;1e3;1e4]
se = get_sample_errors(prob,setups[6],numruns=sample_size,
                                      sample_error_runs = 100_000,solution_runs=100)

4-element Array{Float64,1}:
 0.00045977122950632046
 3.991124997773582e-5
 4.4762697028100585e-6
 4.857459785646766e-7
```

```
times = [wp[i].times for i in 1:length(wp)]
times = [minimum(minimum(t) for t in times),maximum(maximum(t) for t in times)]
plot!([se[end];se[end]],times,color=:red,linestyle=:dash,label="Sample Error:
    10000",lw=3)
```



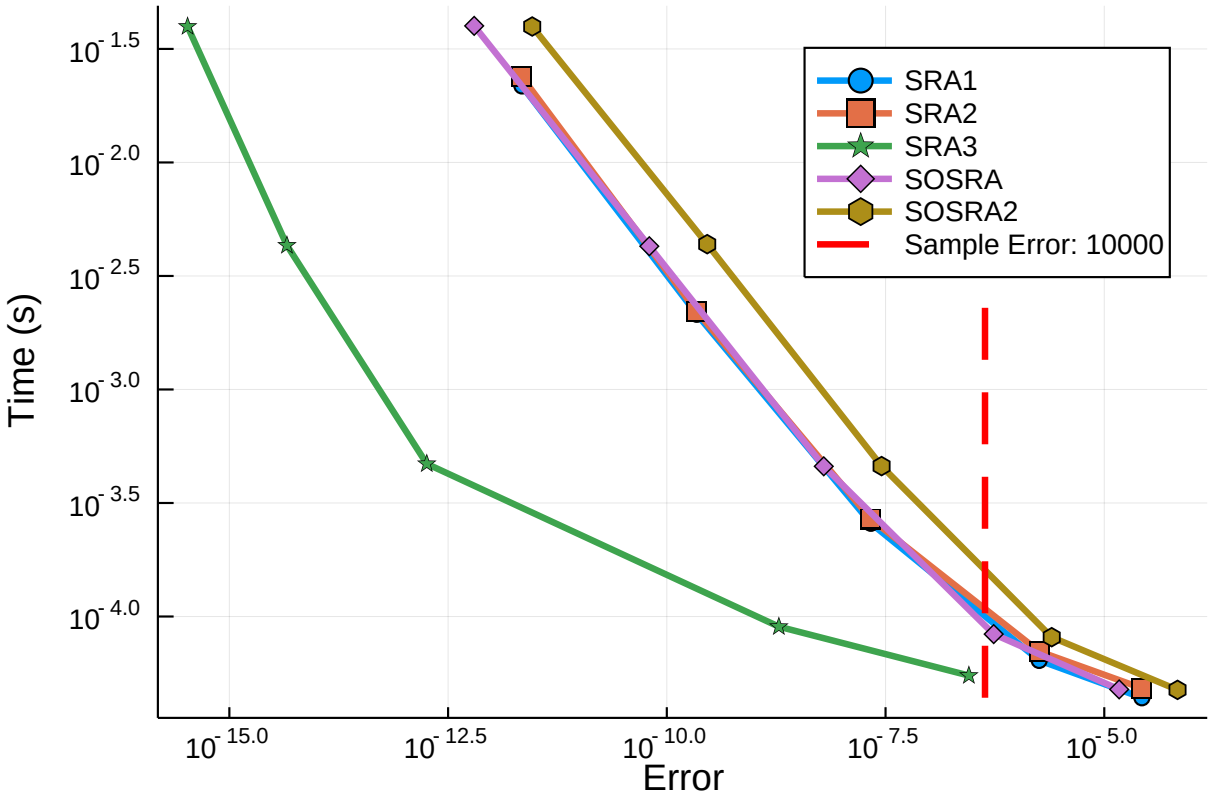

```
prob = prob_sde_additive

reltols = 1.0 ./ 10.0 .^ (1:5)
abstols = reltols#[0.0 for i in eachindex(reltols)]
setups = [
          Dict(:alg=>SRA1())
          Dict(:alg=>SRA2())
          Dict(:alg=>SRA3())
          Dict(:alg=>SOSRA())
          Dict(:alg=>SOSRA2())
          ]
wp = WorkPrecisionSet(prob,abstols,reltols,setups;numruns_error=N,
                      save_everystep = false,
                      maxiters = 1e7,
                      parallel_type = :none,
                      error_estimate=:weak_final)
plot(wp)
```




```
sample_size = Int[10;1e2;1e3;1e4]
se = get_sample_errors(prob,setups[4],numruns=sample_size,
                                    sample_error_runs = 100_000,solution_runs=100)
```

```
4-element Array{Float64,1}:
 0.00040890964801965654
 4.002036756733215e-5
 4.154225820854675e-6
 4.33239165984015e-7
```

```
times = [wp[i].times for i in 1:length(wp)]
times = [minimum(minimum(t) for t in times),maximum(maximum(t) for t in times)]
plot!([se[end];se[end]],times,color=:red,linestyle=:dash,label="Sample Error:
    10000",lw=3)
```



1.0.2 Scalar Noise

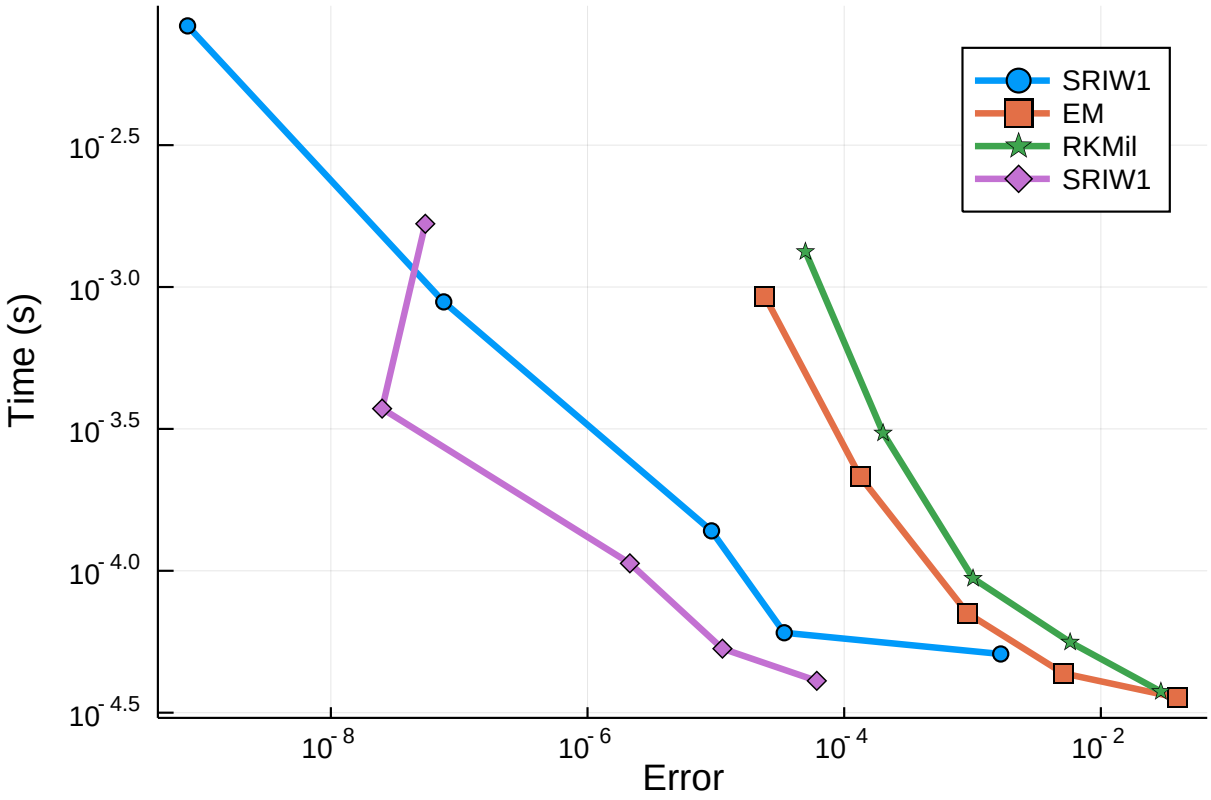
We will use a the linear SDE (also known as the Black-Scholes equation)

$$dX_t = \alpha X_t dt + \beta X_t dW_t, \quad X_0 = \frac{1}{2}$$

where $\alpha = \frac{1}{10}$ and $\beta = \frac{1}{20}$. Actual Solution:

$$X_t = X_0 e^{\left(\beta - \frac{\alpha^2}{2}\right)t + \alpha W_t}.$$

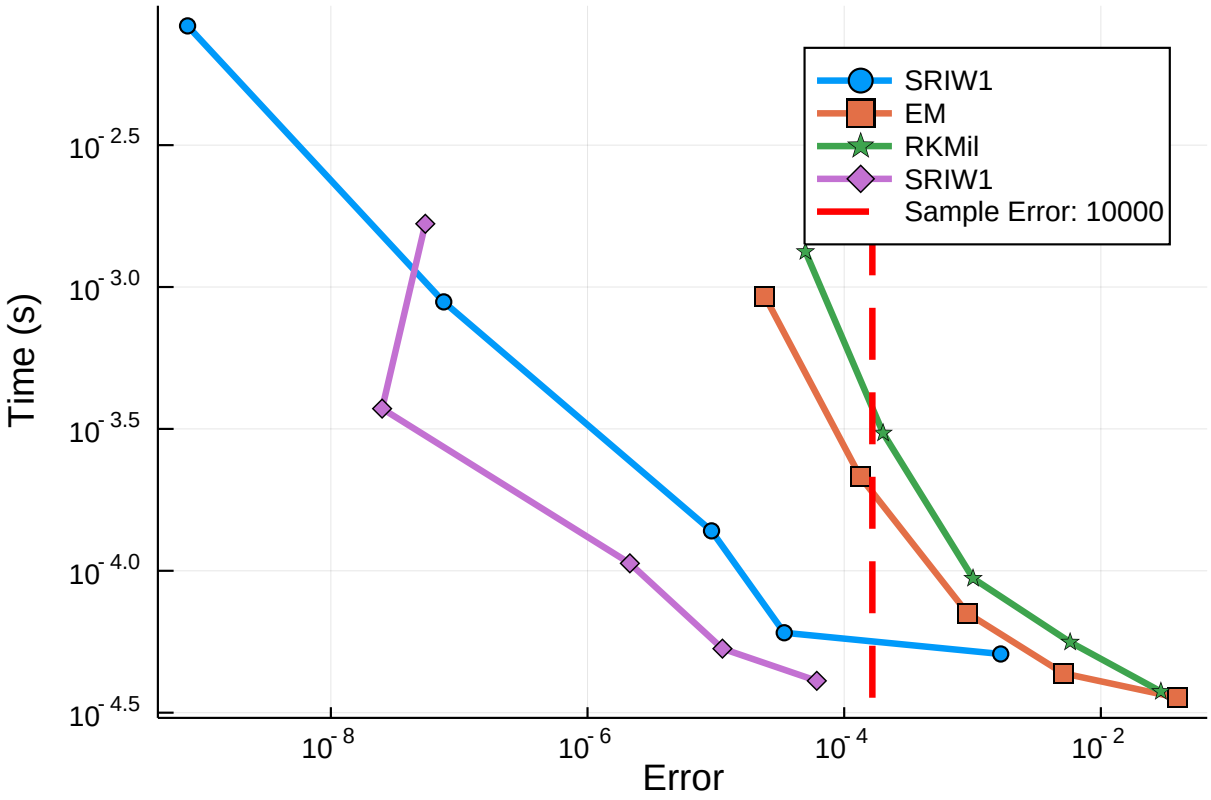
```
prob = prob_sde_linear

reltols = 1.0 ./ 10.0 .^ (1:5)
abstols = reltols#[0.0 for i in eachindex(reltols)]

setups = [Dict(:alg=>SRIW1())
          Dict(:alg=>EM(),:dts=>1.0./5.0.^((1:length(reltols)) .+ 1))
          Dict(:alg=>RKMil(),:dts=>1.0./5.0.^((1:length(reltols)) .+ 1),:adaptive=>false)
          Dict(:alg=>SRIW1(),:dts=>1.0./5.0.^((1:length(reltols)) .+ 1),:adaptive=>false)
          ]
wp = WorkPrecisionSet(prob,abstols,reltols,setups;numruns_error=N,
                      save_everystep = false,
                      maxiters = 1e7,
                      parallel_type = :none,
                      error_estimate=:weak_final)
plot(wp)
```



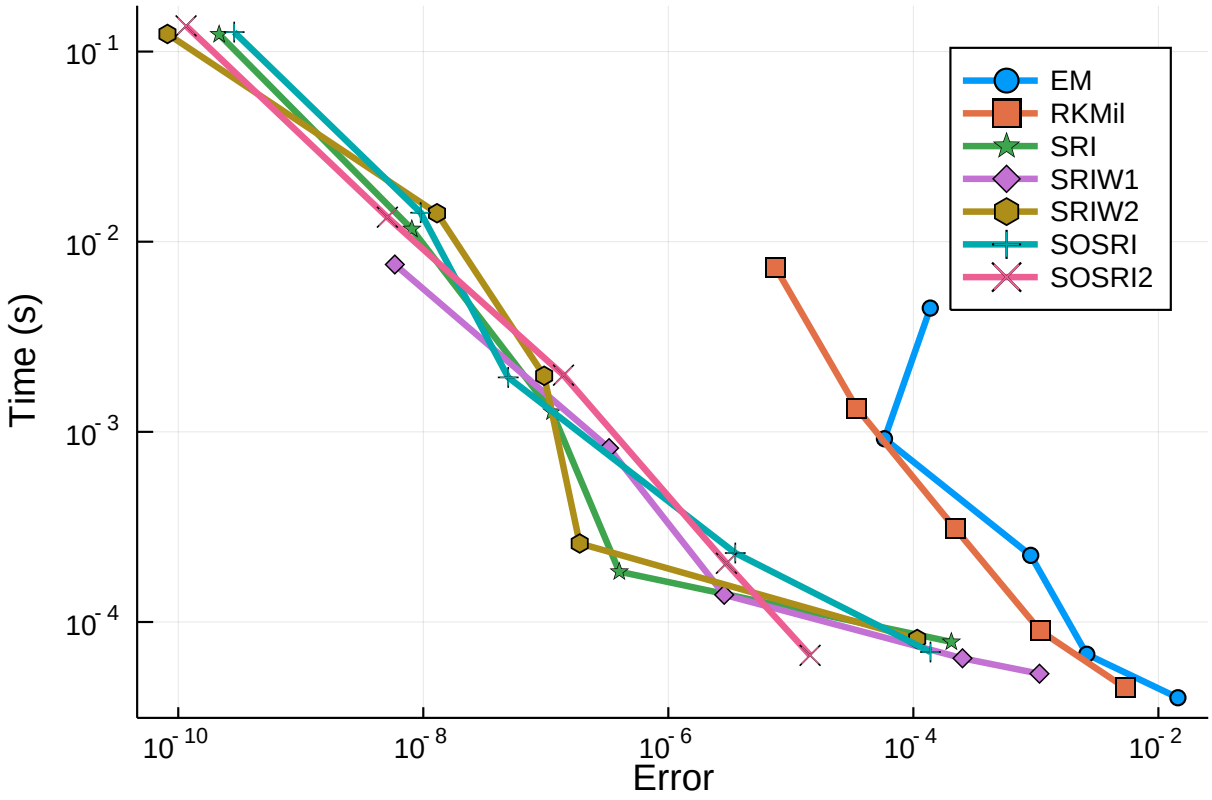

```
sample_size = Int[10;1e2;1e3;1e4]
se = get_sample_errors(prob,setups[1],numruns=sample_size,
                                    sample_error_runs = 100_000,solution_runs=100)
```

```
4-element Array{Float64,1}:
 0.24740489982367606
 0.01804203670912063
 0.0015297713810891263
 0.00016571433046187793
```

```
times = [wp[i].times for i in 1:length(wp)]
times = [minimum(minimum(t) for t in times),maximum(maximum(t) for t in times)]
plot!([se[end];se[end]],times,color=:red,linestyle=:dash,label="Sample Error:
    10000",lw=3)
```



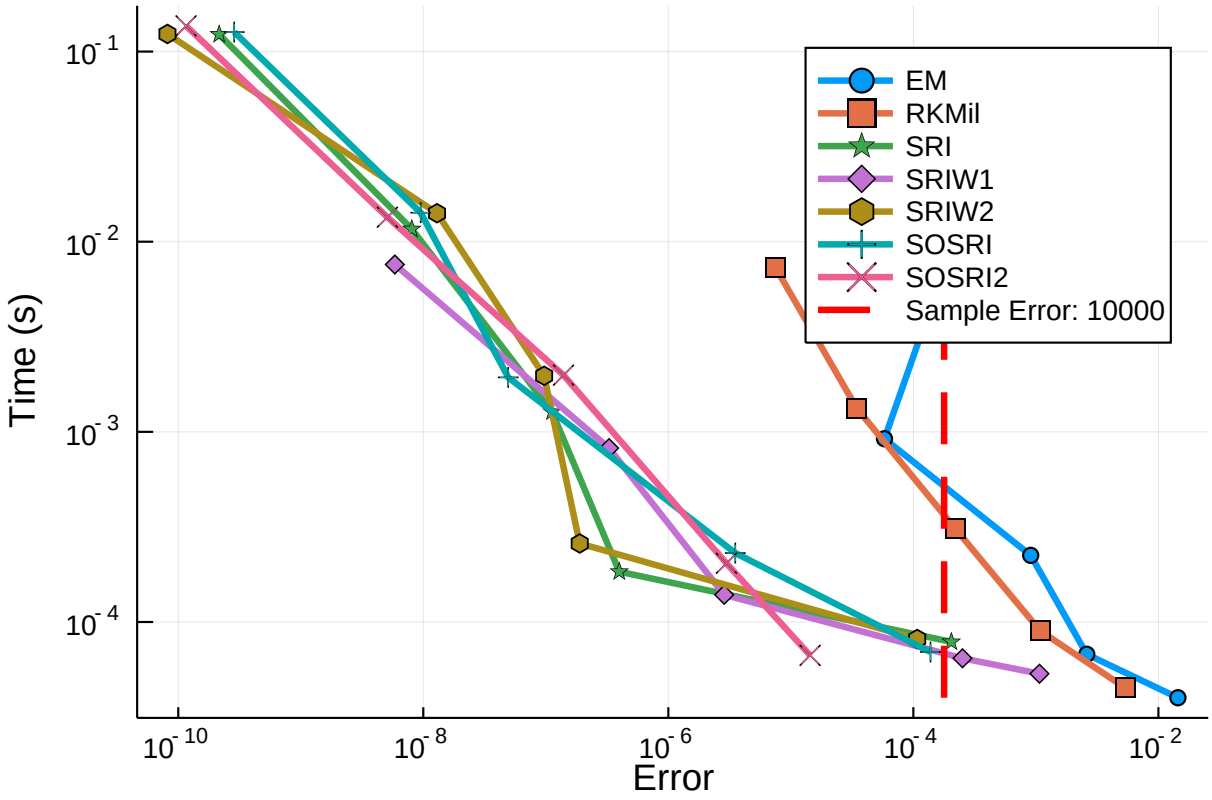

```
prob = prob_sde_linear

reltols = 1.0 ./ 10.0 .^ (1:5)
abstols = reltols#[0.0 for i in eachindex(reltols)]

setups = [Dict(:alg=>EM(),:dts=>1.0./5.0.^((1:length(reltols)) .+ 2))
          Dict(:alg=>RKMil(),:dts=>1.0./5.0.^((1:length(reltols)) .+ 2),:adaptive=>false)
          Dict(:alg=>SRI())
          Dict(:alg=>SRIW1())
          Dict(:alg=>SRIW2())
          Dict(:alg=>SOSRI())
          Dict(:alg=>SOSRI2())
          ]
wp = WorkPrecisionSet(prob,abstols,reltols,setups;numruns_error=N,
                      save_everystep = false,
                      maxiters = 1e7,
                      parallel_type = :none,
                      error_estimate=:weak_final)
plot(wp)
```




```
sample_size = Int[10;1e2;1e3;1e4]
se = get_sample_errors(prob,setups[6],numruns=sample_size,
                                    sample_error_runs = 100_000,solution_runs=100)
```

```
4-element Array{Float64,1}:
 0.1596148183531954
 0.017068851370481402
 0.0019548825760399924
 0.00017830137005002192
```

```
times = [wp[i].times for i in 1:length(wp)]
times = [minimum(minimum(t) for t in times),maximum(maximum(t) for t in times)]
plot!([se[end];se[end]],times,color=:red,linestyle=:dash,label="Sample Error:
    10000",lw=3)
```



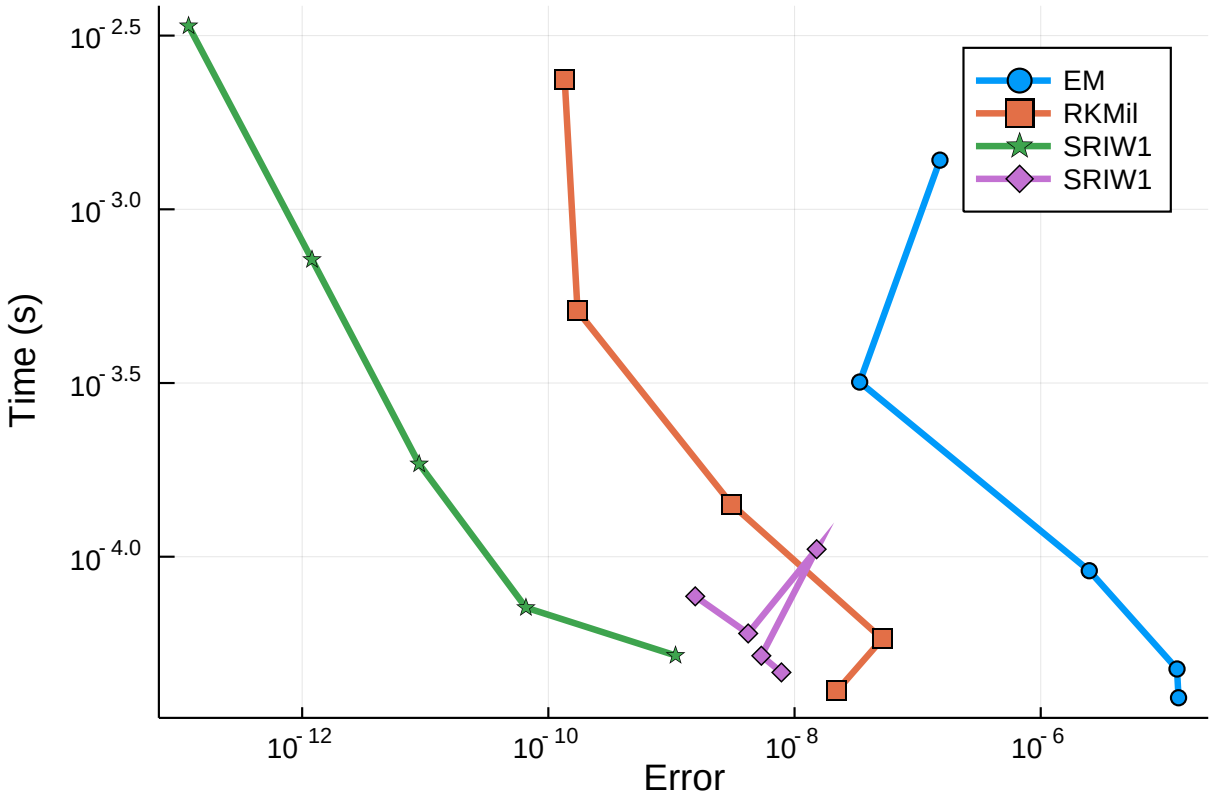
1.1 Scalar Wave SDE

$$dX_t = -\left(\frac{1}{10}\right)^2 \sin\left(X_t\right)\cos^3\left(X_t\right)dt + \frac{1}{10}\cos^2\left(X_t\right)dW_t, \quad X_0 = \frac{1}{2}$$

Actual Solution:

$$X_t = \arctan\left(\frac{1}{10}W_t + \tan\left(X_0\right)\right).$$

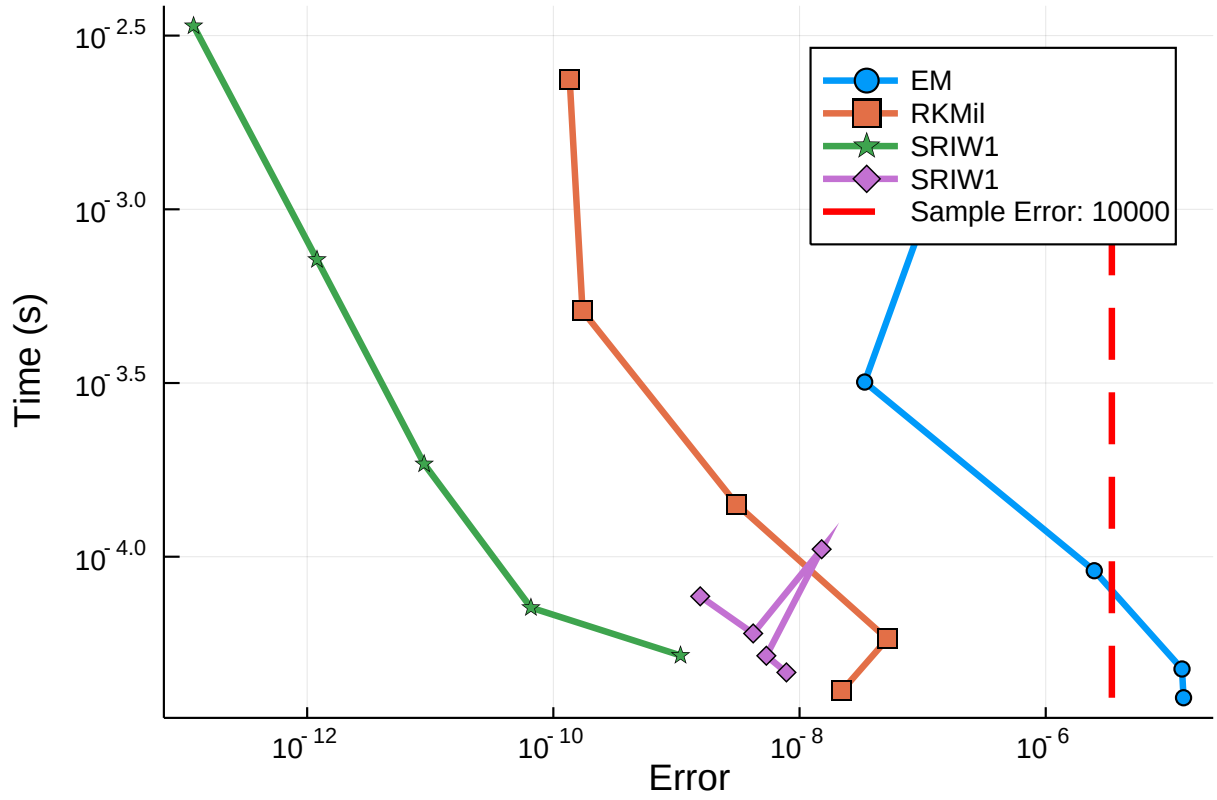
```
prob = prob_sde_wave

reltols = 1.0 ./ 10.0 .^ (1:5)
abstols = reltols#[0.0 for i in eachindex(reltols)]

setups = [
          Dict(:alg=>EM(),:dts=>1.0./5.0.^((1:length(reltols)) .+ 1))
          Dict(:alg=>RKMil(),:dts=>1.0./5.0.^((1:length(reltols)) .+ 1),:adaptive=>false)
          Dict(:alg=>SRIW1(),:dts=>1.0./5.0.^((1:length(reltols)) .+ 1),:adaptive=>false)
          Dict(:alg=>SRIW1())
          ]

wp = WorkPrecisionSet(prob,abstols,reltols,setups;numruns_error=N,
                      save_everystep = false,
                      maxiters = 1e7,
                      parallel_type = :none,
                      error_estimate=:weak_final)
plot(wp)
```



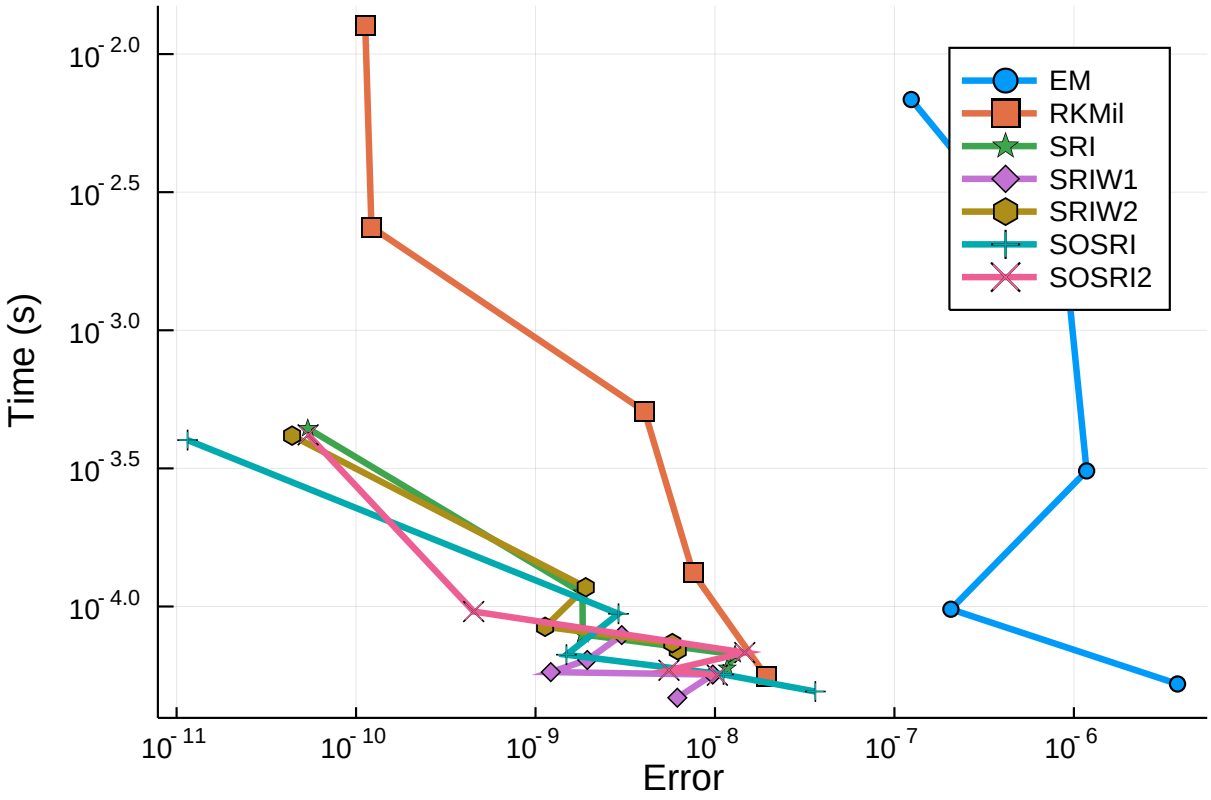

```
sample_size = Int[10;1e2;1e3;1e4]
se = get_sample_errors(prob,setups[4],numruns=sample_size,
                                    sample_error_runs = 100_000,solution_runs=100)
```

```
4-element Array{Float64,1}:
 0.003568809814084504
 0.00032841936883073314
 3.556455544487414e-5
 3.4445774050646317e-6
```

```
times = [wp[i].times for i in 1:length(wp)]
times = [minimum(minimum(t) for t in times),maximum(maximum(t) for t in times)]
plot!([se[end];se[end]],times,color=:red,linestyle=:dash,label="Sample Error:
    10000",lw=3)
```



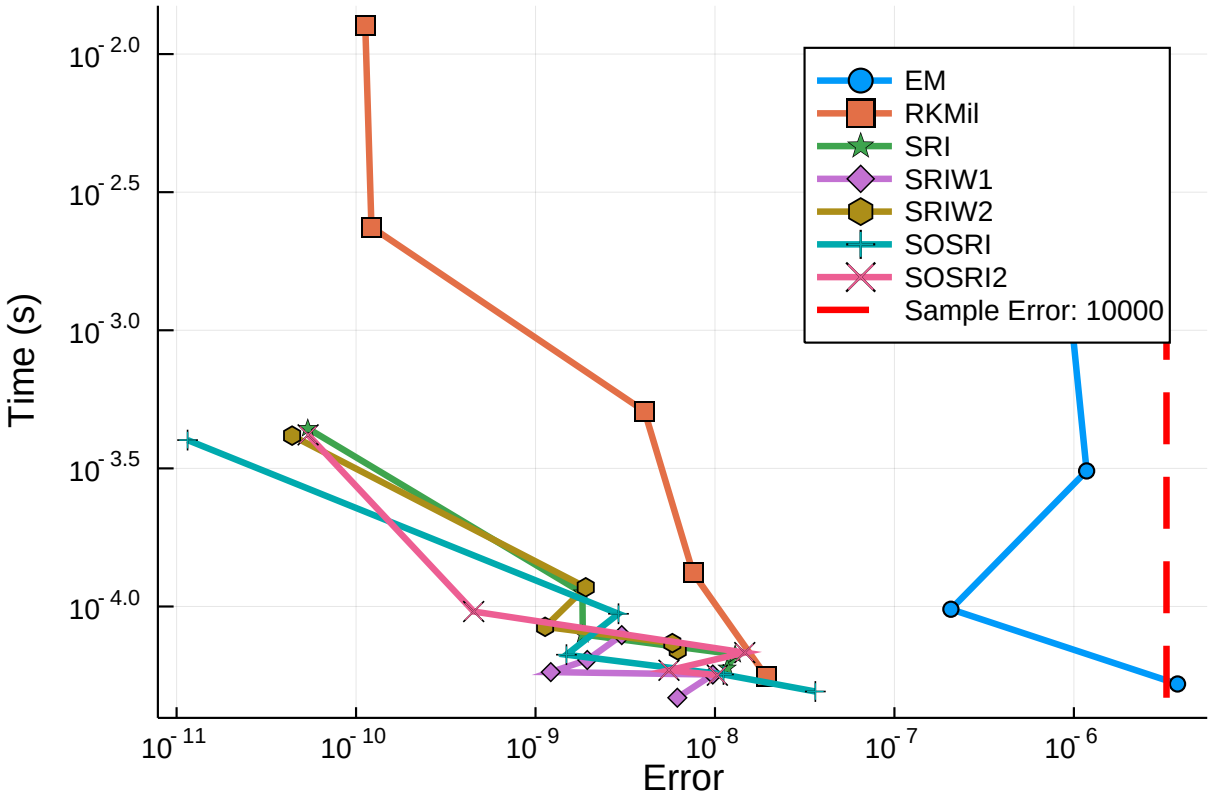

```
prob = prob_sde_wave

reltols = 1.0 ./ 10.0 .^ (1:5)
abstols = reltols#[0.0 for i in eachindex(reltols)]

setups = [Dict(:alg=>EM(),:dts=>1.0./5.0.^((1:length(reltols)) .+ 2))
          Dict(:alg=>RKMil(),:dts=>1.0./5.0.^((1:length(reltols)) .+ 2),:adaptive=>false)
          Dict(:alg=>SRI())
          Dict(:alg=>SRIW1())
          Dict(:alg=>SRIW2())
          Dict(:alg=>SOSRI())
          Dict(:alg=>SOSRI2())
          ]
wp = WorkPrecisionSet(prob,abstols,reltols,setups;numruns_error=N,
                      save_everystep = false,
                      maxiters = 1e7,
                      parallel_type = :none,
                      error_estimate=:weak_final)
plot(wp)
```




```
sample_size = Int[10;1e2;1e3;1e4]
se = get_sample_errors(prob,setups[6],numruns=sample_size,
                                    sample_error_runs = 100_000,solution_runs=100)
```

```
4-element Array{Float64,1}:
 0.0038570624678876246
 0.0003594068923068809
 3.183099157220022e-5
 3.2811425985485232e-6
```

```
times = [wp[i].times for i in 1:length(wp)]
times = [minimum(minimum(t) for t in times),maximum(maximum(t) for t in times)]
plot!([se[end];se[end]],times,color=:red,linestyle=:dash,label="Sample Error:
    10000",lw=3)
```



1.2 Summary

In the additive noise problem, the `EM` and `RKMil` algorithms are not effective at reaching the sample error. In the other two problems, the `EM` and `RKMil` algorithms are as efficient as the higher order methods at achieving the maximal weak error.

```
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("NonStiffSDE","BasicSDEWeakWorkPrecision.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 0.8.8
[bcd4f6db-9728-5f36-b5f7-82caef46ccdb] DelayDiffEq 5.9.1
[bb2cbb15-79fc-5d1e-9bf1-8ae49c7c1650] DiffEqBenchmarks 0.1.0
[459566f4-90b8-5000-8ac3-15dfb0a30def] DiffEqCallbacks 2.5.2
[f3b72e0c-5b89-59e1-b016-84e28bfd966d] DiffEqDevTools 2.13.0
[aae7a2af-3d4f-5e19-a356-7da93b79d9d0] DiffEqFlux 0.6.1
[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 3.2.0
[a077e3f3-b75c-5d7f-a0c6-6bc4c8ec64a9] DiffEqProblemLibrary 4.5.0
[0c46a032-eb83-5123-abaf-570d42b7fbaa] DifferentialEquations 6.6.0
[35a29f4d-8980-5a13-9543-d66fff28ecb8] DocumenterTools 0.1.1
[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 0.4.6
[09606e27-ecf5-54fc-bb29-004bd9f985bf] ODEInterfaceDiffEq 3.3.1
[1dea7af3-3e70-54e6-95c3-0bf5283fa5ed] OrdinaryDiffEq 5.12.0
[2dcacdae-9679-587a-88bb-8b444fb7085b] ParallelDataTransfer 0.5.0
[65888b18-ceab-5e60-b2b9-181511a3b968] ParameterizedFunctions 4.2.0
[91a5bcdd-55d7-5caf-9e0b-520d859cae80] Plots 0.26.0
[9cded84f-2146-5f9f-b71f-7cd68310d1ff] PumasDocs 0.0.0
[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 6.6.0
[c3572dad-4567-51f8-b174-8c6c989267f4] Sundials 3.6.1
[92b13dbe-c966-51a2-8445-caca9f8a7d42] TaylorIntegration 0.5.0
[44d3d7a6-8a23-5bf8-98c5-b353f8df5ec9] Weave 0.9.1
[e88e6eb3-aa80-5325-afca-941959d7151f] Zygote 0.3.2
```