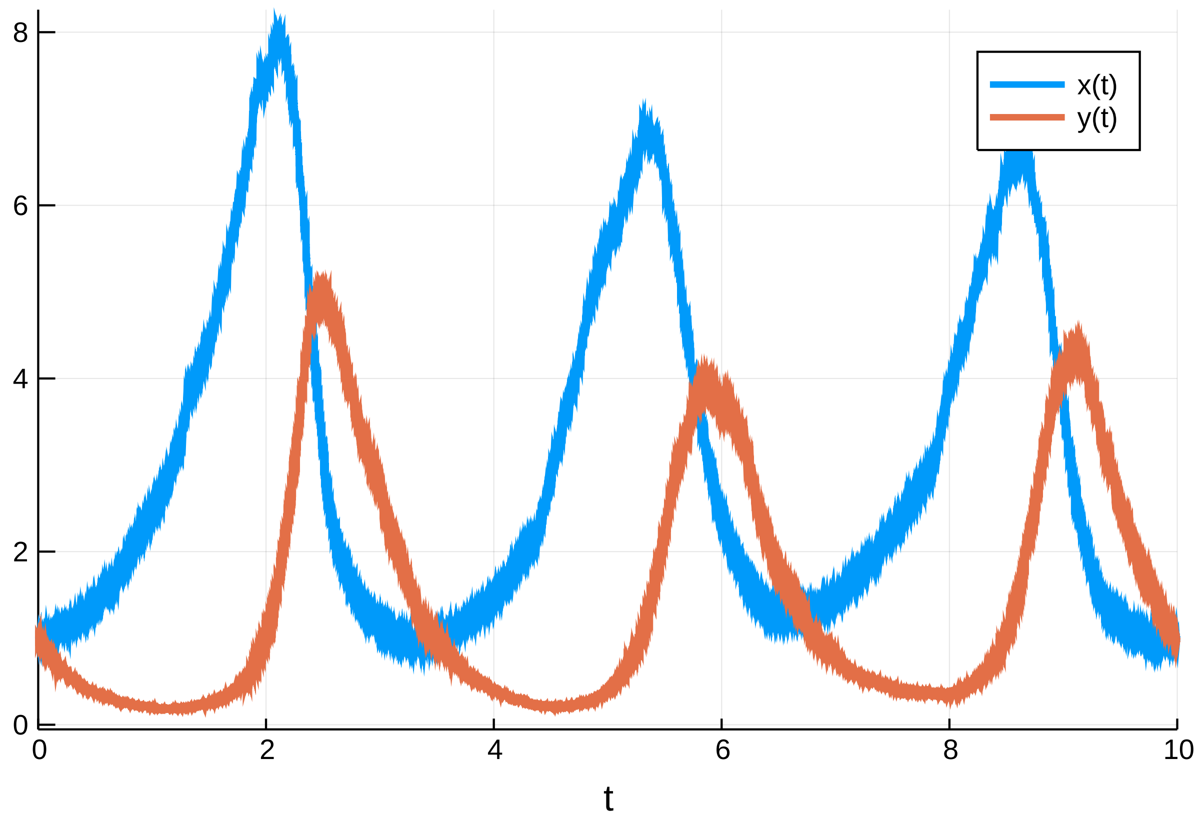
SDE Lokta-Volterra Work-Precision Diagrams

Chris Rackauckas

April 28, 2019

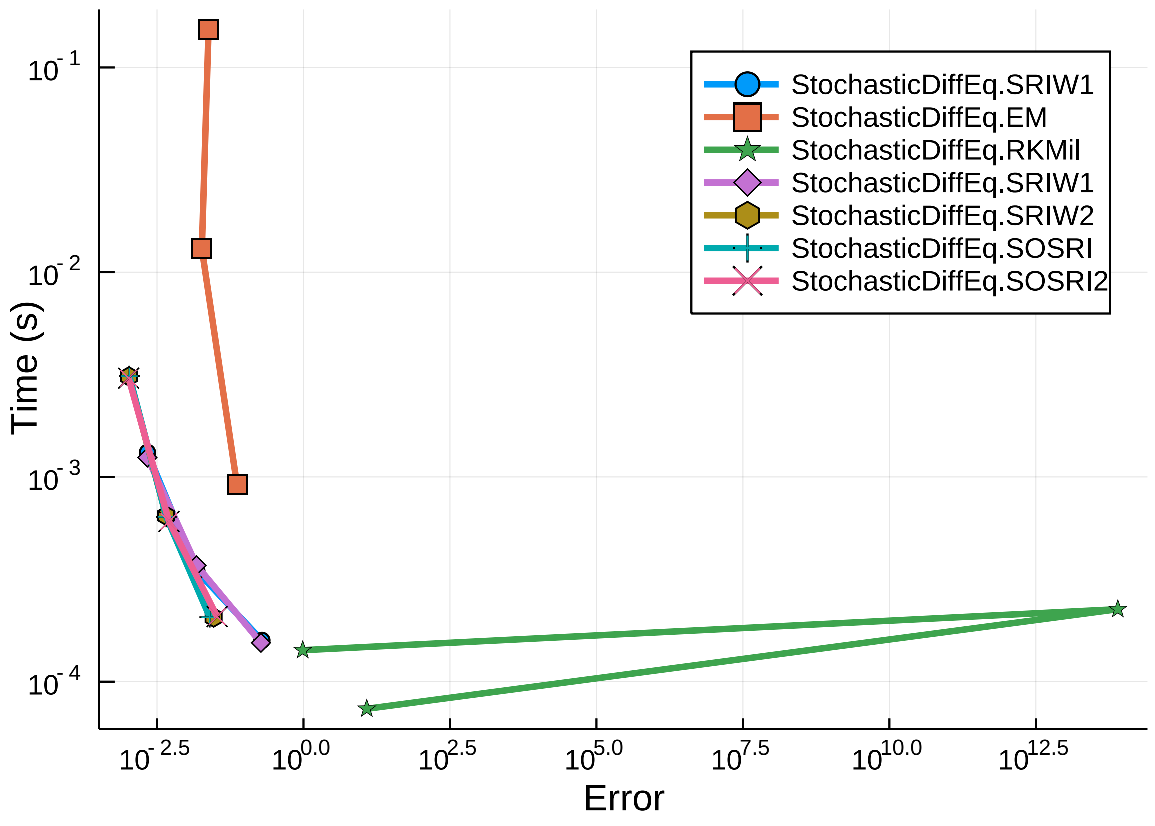
```julia
using StochasticDiffEq, DiffEqDevTools, ParameterizedFunctions
using Plots; gr()
const N = 100

f = @ode_def LotkaVolterraTest begin
  dx = a*x - b*x*y
  dy = -c*y + d*x*y
end a b c d

p = [1.5,1.0,3.0,1.0]

function g(du,u,p,t)
  du .= 0.1u
end
u0 = [1.0;1.0]
tspan = (0.0,10.0)
prob = SDEProblem(f,g,u0,tspan,p);

sol = solve(prob,SRIW1(),abstol=1e-4,reltol=1e-4)
plot(sol)
```



0.1 Strong Error

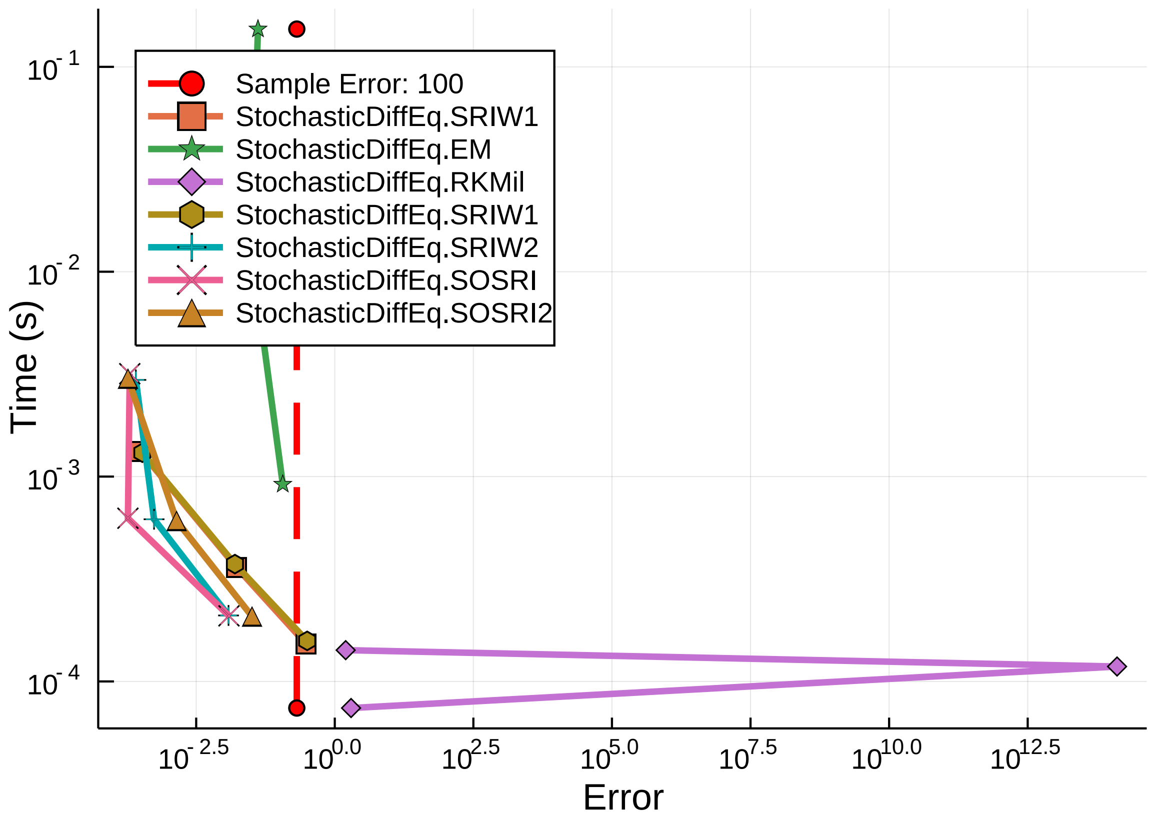
The starting dts was chosen as the largest in the 1/4^i which were stable. All larger dts contained trajectories which would veer off to infinity.

```
reltols = 1.0 ./ 4.0 .^ (2:4)
abstols = reltols#[0.0 for i in eachindex(reltols)]
setups = [Dict(:alg=>SRIW1())
          Dict(:alg=>EM(),:dts=>1.0./12.0.^((1:length(reltols)) .+ 1.5))
          Dict(:alg=>RKMil(),:dts=>1.0./12.0.^((1:length(reltols)) .+ 1.5))
          Dict(:alg=>SRIW1(),:dts=>1.0./4.0.^((1:length(reltols)) .+ 5),:adaptive=>false)
          Dict(:alg=>SRIW2())
          Dict(:alg=>SOSRI())
          Dict(:alg=>SOSRI2())
          ]
test_dt = 1/10^2
appxsol_setup = Dict(:alg=>SRIW1(),:abstol=>1e-4,:reltol=>1e-4)
wp = WorkPrecisionSet(prob,abstols,reltols,setups,test_dt;
                                     verbose=false,save_everystep=false,
                                     parallel_type = :threads,
                                     appxsol_setup = appxsol_setup,
                                     numruns_error=N,error_estimate=:final)
plot(wp)
```



0.2 Weak Error

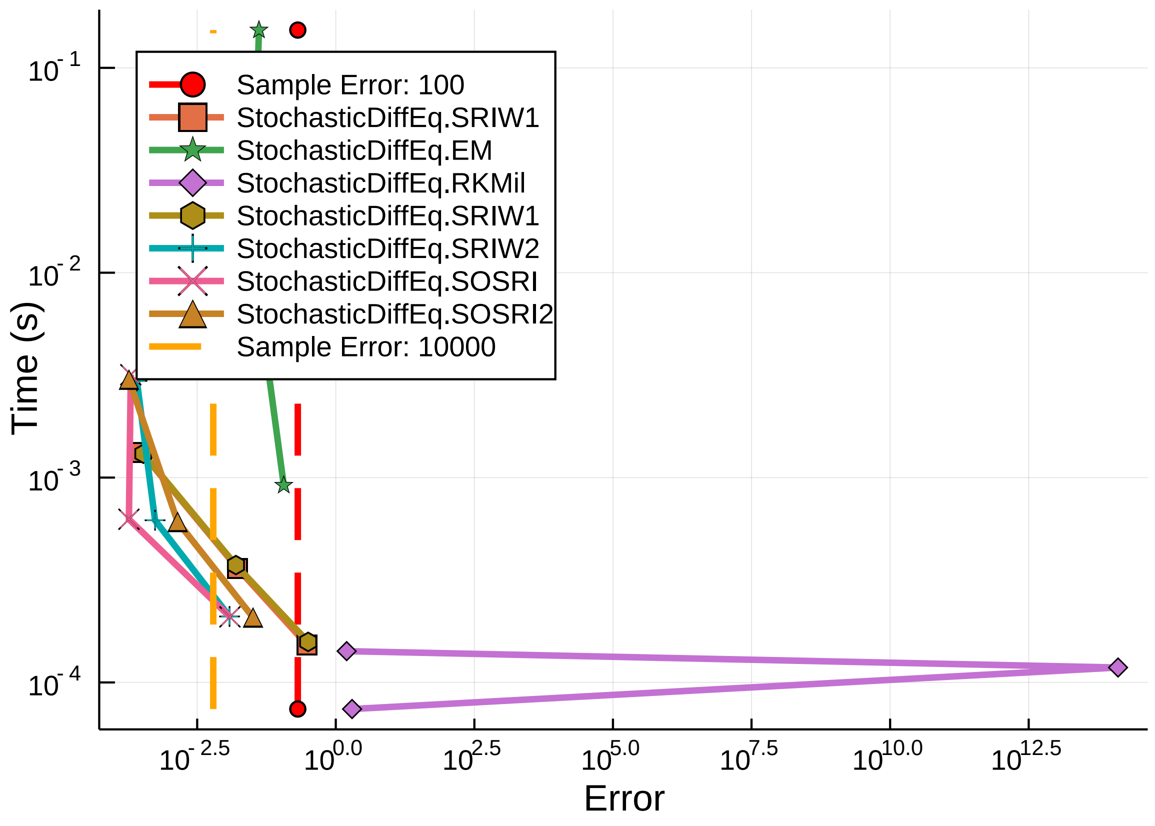
```
reltols = 1.0 ./ 4.0 .^ (2:4)
abstols = reltols#[0.0 for i in eachindex(reltols)]
setups = [Dict(:alg=>SRIW1())
          Dict(:alg=>EM(),:dts=>1.0./12.0.^((1:length(reltols)) .+ 1.5))
          Dict(:alg=>RKMil(),:dts=>1.0./12.0.^((1:length(reltols)) .+ 1.5))
          Dict(:alg=>SRIW1(),:dts=>1.0./4.0.^((1:length(reltols)) .+ 5),:adaptive=>false)
          Dict(:alg=>SRIW2())
          Dict(:alg=>SOSRI())
          Dict(:alg=>SOSRI2())
          ]
test_dt = 1e-2
appxsol_setup = Dict(:alg=>SRIW1(),:abstol=>1e-4,:reltol=>1e-4)
wp = WorkPrecisionSet(prob,abstols,reltols,setups,test_dt;
                                     verbose=false,save_everystep=false,
                                     parallel_type = :threads,
                                     appxsol_setup = appxsol_setup,
                                     numruns_error=N,error_estimate=:weak_final)
plot(wp;legend=:topleft)
```



```
sample_size = Int[10;1e2;1e3;1e4;1e5]
test_dt = 1e-2
appxsol_setup = Dict(:alg=>SRIW1(),:abstol=>1e-4,:reltol=>1e-4)
se = get_sample_errors(prob,test_dt,appxsol_setup=appxsol_setup,
parallel_type = :threads, numruns=sample_size, std_estimation_runs = Int(1e3))
```

```
5-element Array{Float64,1}:
 0.6168500722871536
 0.1950651203266916
 0.061685007228715366
 0.01950651203266916
 0.006168500722871536
```

```
plot(wp;legend=:topleft)
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=:orange,linestyle=:dash,label="Sample Error:
    10000",lw=3)
```



0.3 Conclusion

These results show that in both strong and weak error, the high order method is more efficient. The strong and the weak are track each other well for the methods tested on this problem, with the strong error slightly higher than the weak error. To reach the sample error for a 100 trajectories, the higher order method is around 5x faster. To reach the sampling error for 10000 trajectories, the higher order method is nearly 100x faster.

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

0.4 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("NonStiffSDE","LotkaVolterraSDE.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 2.7.2+
[055956cb-9e8b-5191-98cc-73ae4a59e68a] DiffEqPhysics 3.1.0
[a077e3f3-b75c-5d7f-a0c6-6bc4c8ec64a9] DiffEqProblemLibrary 4.1.0
[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 5.5.0
[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 3.3.0+
[92b13dbe-c966-51a2-8445-caca9f8a7d42] TaylorIntegration 0.4.1
[44d3d7a6-8a23-5bf8-98c5-b353f8df5ec9] Weave 0.9.0
```