Quadratic Stiffness Benchmarks

Chris Rackauckas

May 5, 2019

1 Quadratic Stiffness

In this notebook we will explore the quadratic stiffness problem. References:

The composite Euler method for stiff stochastic differential equations

Kevin Burrage, Tianhai Tian

And

S-ROCK: CHEBYSHEV METHODS FOR STIFF STOCHASTIC DIFFERENTIAL EQUATIONS

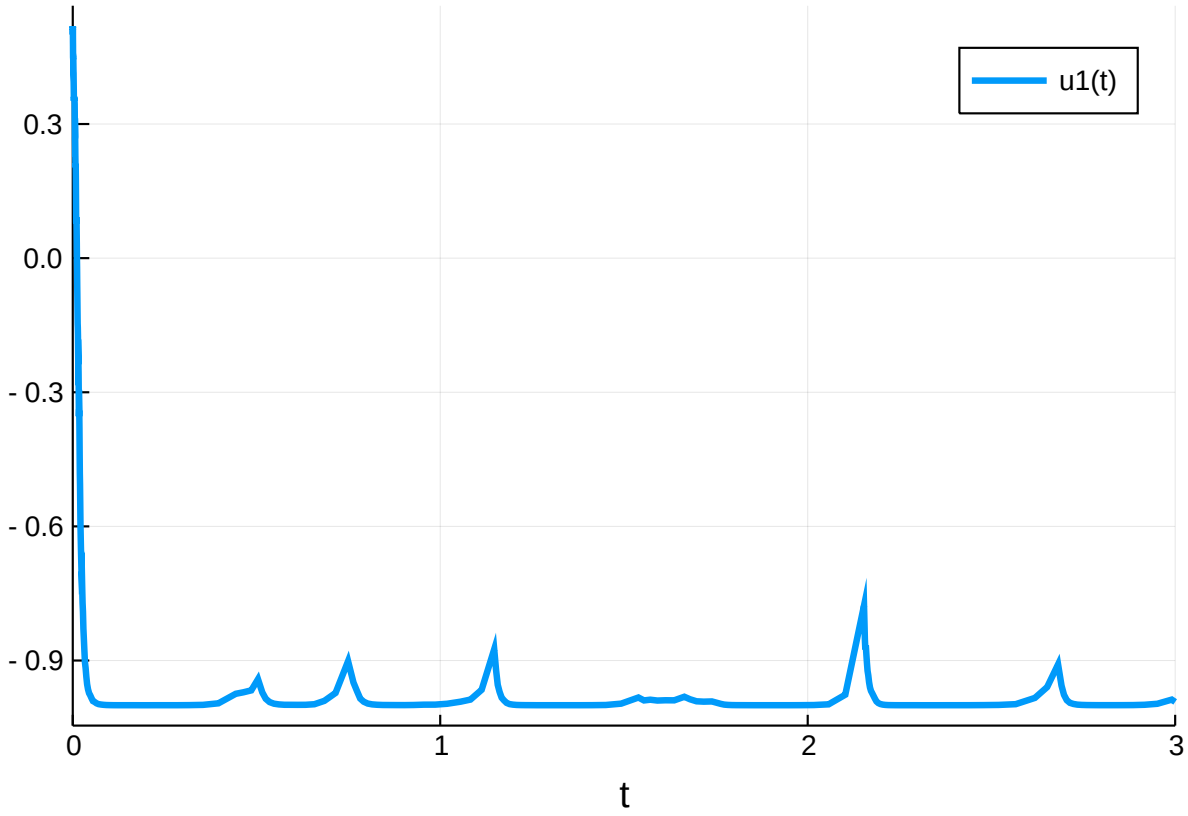
ASSYR ABDULLE AND STEPHANE CIRILLI

This is a scalar SDE with two arguments. The first controls the deterministic stiffness and the later controls the diffusion stiffness.

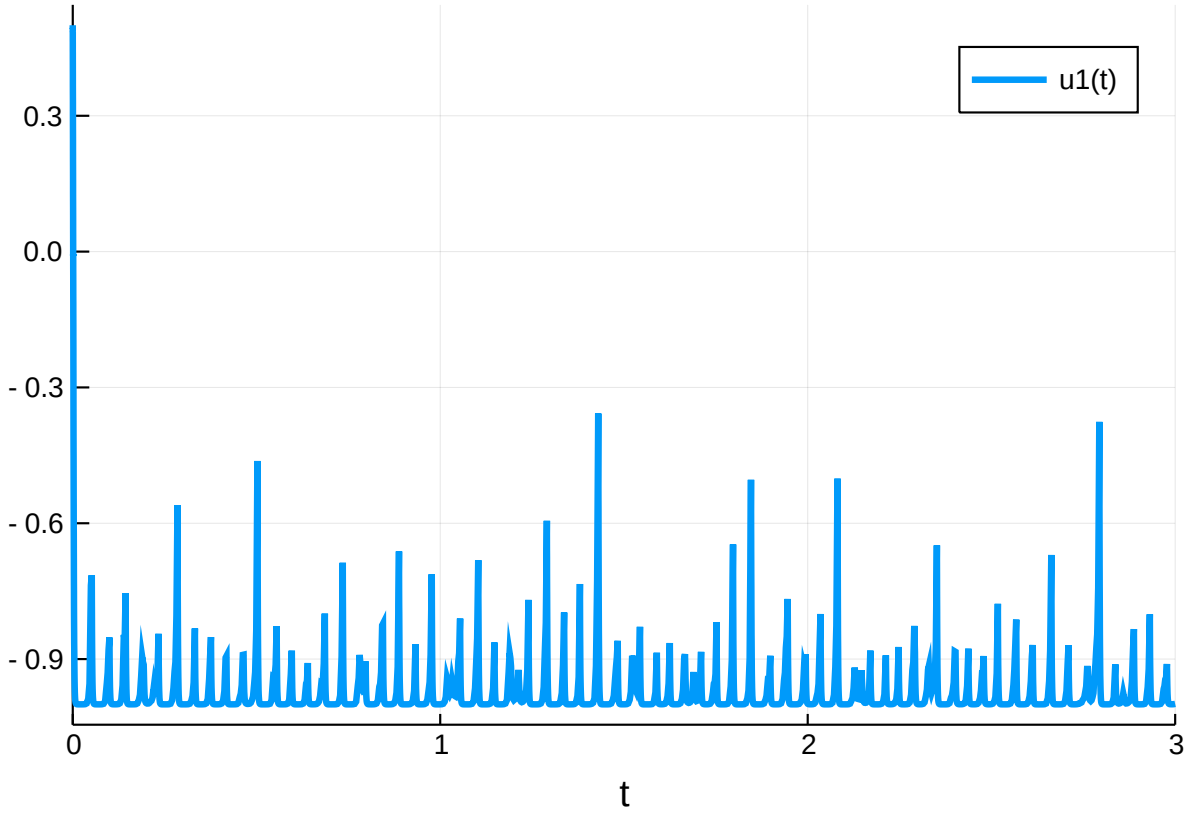
```
using DiffEqProblemLibrary, StochasticDiffEq, DiffEqDevTools
using DiffEqProblemLibrary.SDEProblemLibrary: importsdeproblems; importsdeproblems()
import DiffEqProblemLibrary.SDEProblemLibrary: prob_sde_stiffquadito
using Plots; gr()
const N = 10
```

```
10
```

```
prob = remake(prob_sde_stiffquadito,p=(50.0,1.0))
sol = solve(prob,SRIW1())
plot(sol)
```




```
prob = remake(prob_sde_stiffquadito,p=(500.0,1.0))
sol = solve(prob,SRIW1())
plot(sol)
```



1.1 Top dts

Let's first determine the maximum dts which are allowed. Anything higher is mostly unstable.

1.1.1 Deterministic Stiffness Mild

```
prob = remake(prob_sde_stiffquadito,p=(50.0,1.0))
@time sol = solve(prob,SRIW1())
```

```
0.000222 seconds (1.89 k allocations: 97.641 KiB)
```

```
@time sol = solve(prob,SRIW1(),adaptive=false,dt=0.01)
```

```
0.000193 seconds (2.27 k allocations: 125.109 KiB)
```

```
@time sol = solve(prob,ImplicitRKMil(),dt=0.005)
```

```
0.000489 seconds (3.60 k allocations: 177.203 KiB)
```

```
@time sol = solve(prob,EM(),dt=0.01);
```

```
0.000170 seconds (1.65 k allocations: 99.250 KiB)
```

1.1.2 Deterministic Stiffness High

```
prob = remake(prob_sde_stiffquadito,p=(500.0,1.0))
@time sol = solve(prob,SRIW1())
```

```
0.001245 seconds (14.23 k allocations: 569.453 KiB)
```

```
@time sol = solve(prob,SRIW1(),adaptive=false,dt=0.002)
```

```
0.000577 seconds (10.67 k allocations: 465.359 KiB)
```

```
@time sol = solve(prob,ImplicitRKMil(),dt=0.001)
```

```
0.002731 seconds (27.65 k allocations: 1.223 MiB)
```

```
@time sol = solve(prob,EM(),dt=0.002);
```

```
0.000472 seconds (7.66 k allocations: 377.828 KiB)
```

1.1.3 Mixed Stiffness

```
prob = remake(prob_sde_stiffquadito,p=(5000.0,70.0))
@time sol = solve(prob,SRIW1(),dt=0.0001)
```

```
0.002130 seconds (10.86 k allocations: 792.453 KiB)
```

```
@time sol = solve(prob,SRIW1(),adaptive=false,dt=0.00001)
```

```
0.103948 seconds (2.10 M allocations: 70.387 MiB, 23.81% gc time)
```

```
@time sol = solve(prob,ImplicitRKMil(),dt=0.00001)
```

```
0.408291 seconds (1.00 M allocations: 61.422 MiB, 2.42% gc time)
```

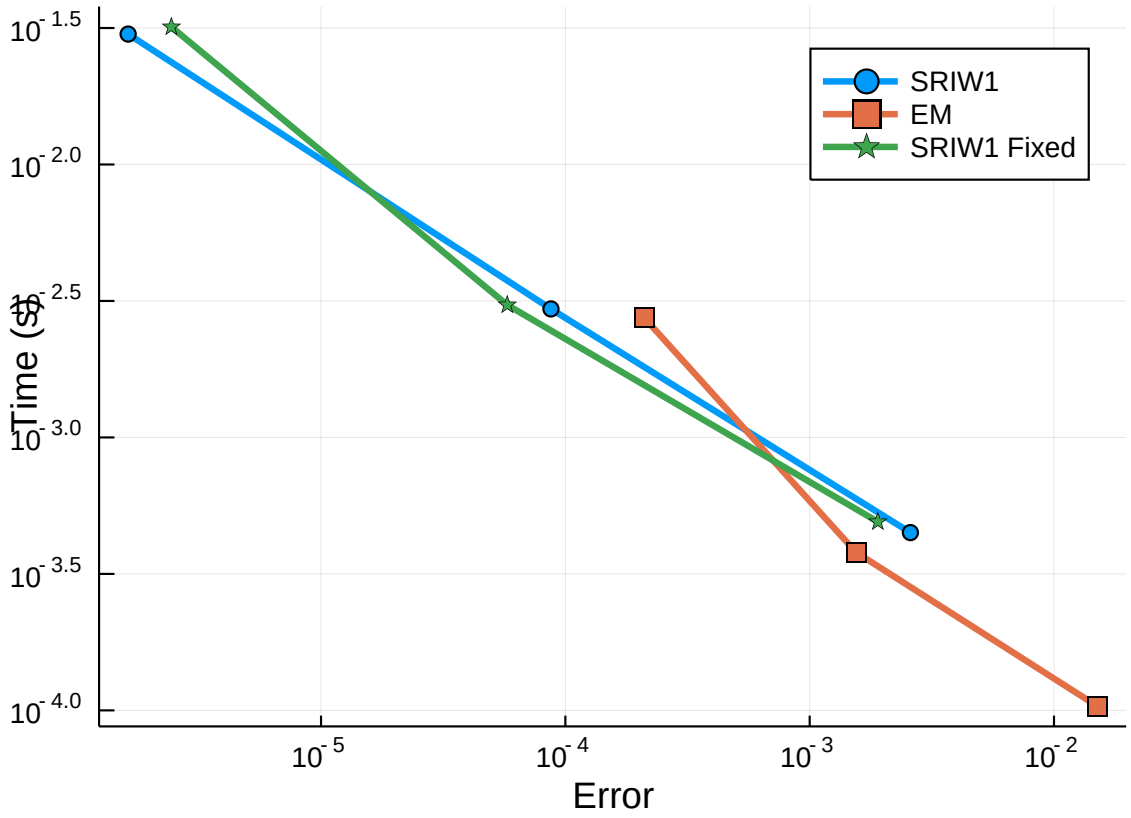
```
@time sol = solve(prob,EM(),dt=0.00001);
```

```
0.265204 seconds (1.50 M allocations: 56.223 MiB, 76.42% gc time)
```

Notice that in this problem, the stiffness in the noise term still prevents the semi-implicit integrator to do well. In that case, the advantage of implicitness does not take effect, and thus explicit methods do well. When we don't care about the error, Euler-Maruyama is fastest. When there's mixed stiffness, the adaptive algorithm is unstable.

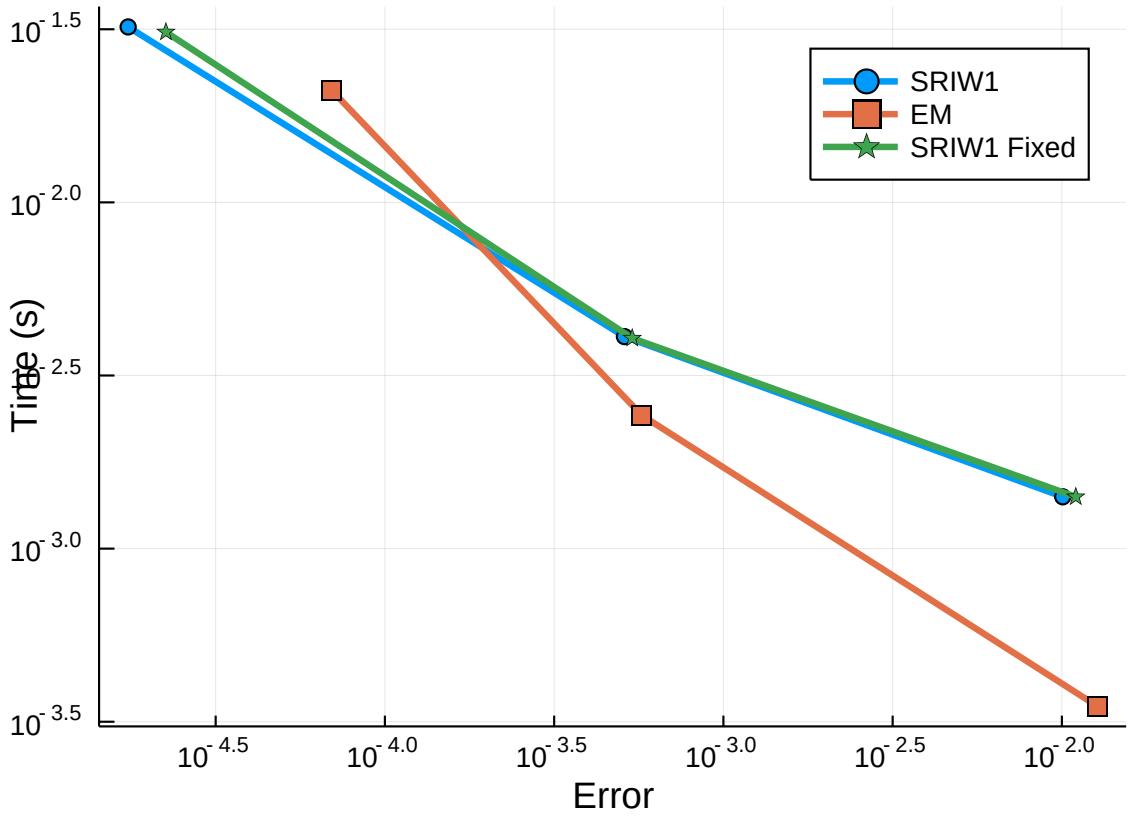
1.2 Work-Precision Diagrams

```
prob = remake(prob_sde_stiffquadito,p=(50.0,1.0))

reltols = 1.0 ./ 10.0 .^ (3:5)
abstols = reltols#[0.0 for i in eachindex(reltols)]
setups = [Dict(:alg=>SRIW1()),
          Dict(:alg=>EM(),:dts=>1.0./8.0.^((1:length(reltols)) .+ 1)),
          Dict(:alg=>SRIW1(),:dts=>1.0./8.0.^((1:length(reltols)) .+ 1),:adaptive=>false)
          #Dict(:alg=>RKMil(),:dts=>1.0./8.0.^((1:length(reltols)) .+
    1),:adaptive=>false),
          ]
names = ["SRIW1","EM","SRIW1 Fixed"] #"RKMil",
wp =
    WorkPrecisionSet(prob,abstols,reltols,setups;numruns=N,names=names,error_estimate=:l2)
plot(wp)
```




```
prob = remake(prob_sde_stiffquadito,p=(500.0,1.0))

reltols = 1.0 ./ 10.0 .^ (3:5)
abstols = reltols#[0.0 for i in eachindex(reltols)]
setups = [Dict(:alg=>SRIW1()),
          Dict(:alg=>EM(),:dts=>1.0./8.0.^((1:length(reltols)) .+ 2)),
```

```
        Dict(:alg=>SRIW1(),:dts=>1.0./8.0.^((1:length(reltols)) .+ 2),:adaptive=>false)
        #Dict(:alg=>RKMil(),:dts=>1.0./8.0.^((1:length(reltols)) .+
    2),:adaptive=>false),
        ]
names = ["SRIW1","EM","SRIW1 Fixed"] #"RKMil",
wp =
    WorkPrecisionSet(prob,abstols,reltols,setups;numruns=N,names=names,error_estimate=:l2,print_names=
plot(wp)
```



1.3 Conclusion

Noise stiffness is tough. Right now the best solution is to run an explicit integrator with a low enough dt. Adaptivity does have a cost in this case, likely due to memory management.

```
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("StiffSDE","QuadraticStiffness.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 2.8.0
[78ddff82-25fc-5f2b-89aa-309469cbf16f] DiffEqMonteCarlo 0.14.0
[77a26b50-5914-5dd7-bc55-306e6241c503] DiffEqNoiseProcess 3.2.0
[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 6.3.0
[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 3.2.0
[1dea7af3-3e70-54e6-95c3-0bf5283fa5ed] OrdinaryDiffEq 5.6.0
[2dcacdae-9679-587a-88bb-8b444fb7085b] ParallelDataTransfer 0.5.0
[65888b18-ceab-5e60-b2b9-181511a3b968] ParameterizedFunctions 4.1.1
[91a5bcdd-55d7-5caf-9e0b-520d859cae80] Plots 0.24.0
[d330b81b-6aea-500a-939a-2ce795aea3ee] PyPlot 2.8.1
[295af30f-e4ad-537b-8983-00126c2a3abe] Revise 2.1.4
[90137ffa-7385-5640-81b9-e52037218182] StaticArrays 0.10.3
[789caeaf-c7a9-5a7d-9973-96adeb23e2a0] StochasticDiffEq 6.2.0
[c3572dad-4567-51f8-b174-8c6c989267f4] Sundials 3.4.1
[92b13dbe-c966-51a2-8445-caca9f8a7d42] TaylorIntegration 0.4.1
[44d3d7a6-8a23-5bf8-98c5-b353f8df5ec9] Weave 0.9.0
[e88e6eb3-aa80-5325-afca-941959d7151f] Zygote 0.3.0
```