Quadratic Stiffness Benchmarks

Chris Rackauckas

July 5, 2020

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

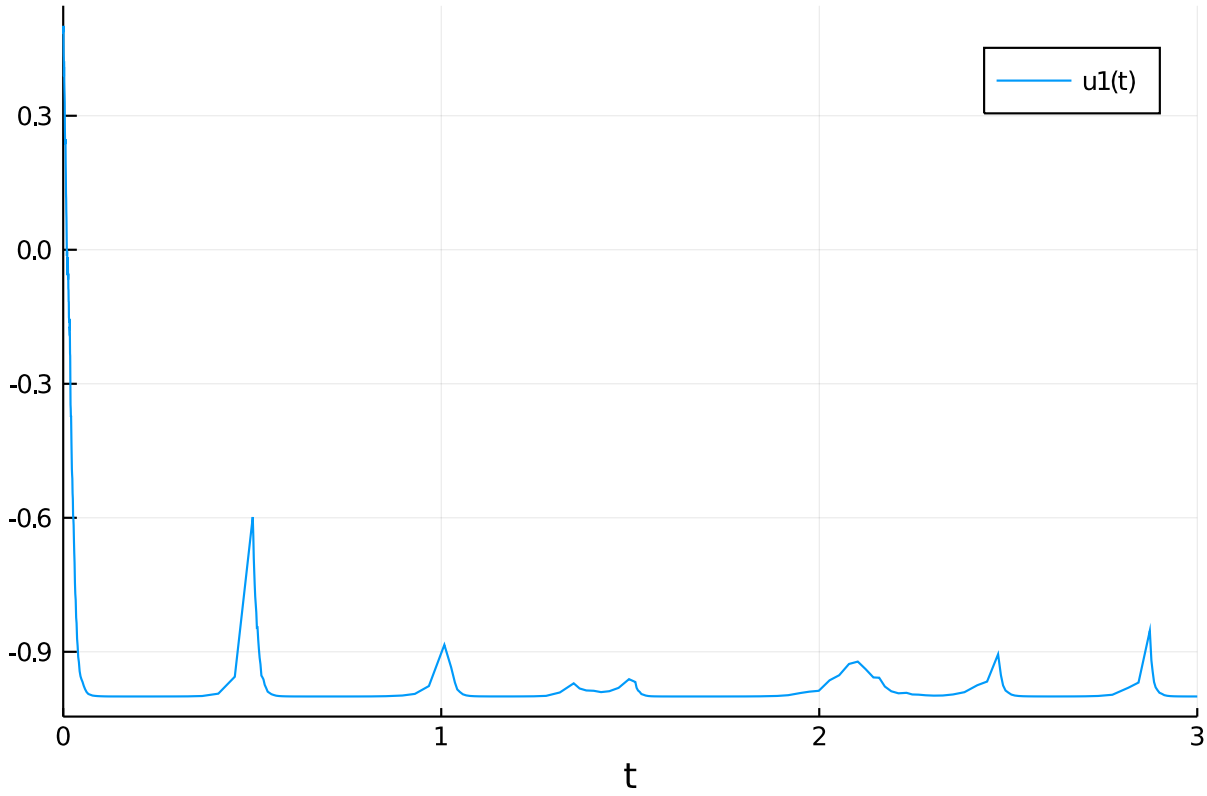
ASSYR ABDULLE AND STEPHANE CIRILLI

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

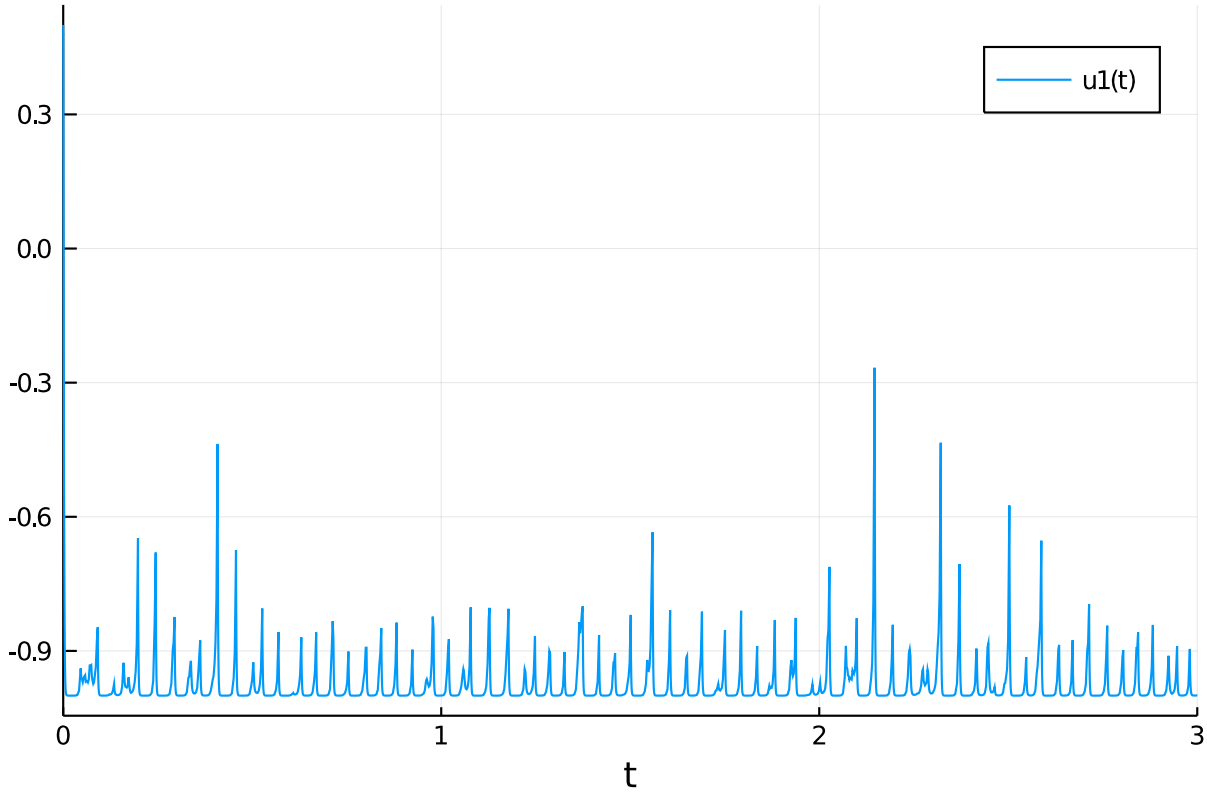
```
using DiffEqProblemLibrary, StochasticDiffEq, DiffEqDevTools
using DiffEqProblemLibrary.SDEProblemLibrary: importsdeproblems; importsdeproblems()
import DiffEqProblemLibrary.SDEProblemLibrary: prob_sde_stiffquadito
using Plots; gr()
const N = 10
```

```
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())
```

```
0.000148 seconds (1.83 k allocations: 71.594 KiB)
```

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

```
0.000160 seconds (2.20 k allocations: 98.656 KiB)
```

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

```
0.000061 seconds (419 allocations: 17.766 KiB)
```

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

```
0.000133 seconds (1.59 k allocations: 80.781 KiB)
retcode: Success
Interpolation: 1st order linear
t: 302-element Array{Float64,1}:
 0.0
 0.01
 0.02
 0.03
 0.04
 0.05
 0.060000000000000005
 0.07
 0.08
 0.09
 ⋮
 2.9299999999999815
 2.9399999999999813
 2.949999999999981
 2.959999999999981
 2.9699999999999807
 2.9799999999999804
 2.9899999999999802
 2.99999999999998
 3.0
u: 302-element Array{Float64,1}:
  0.5
  0.04249022658156923
 -0.4214338677282473
 -0.8969386255984535
 -0.9829449407063323
 -0.9999448123192103
 -0.999996688807814
 -0.9999993850083438
 -0.99999992831608
 -0.9999999877008195
```

```
  ⋮
 -1.0
 -1.0
 -1.0
 -1.0
 -1.0
 -1.0
 -1.0
 -1.0
 -1.0
```

1.1.2 Deterministic Stiffness High

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

0.000950 seconds (14.83 k allocations: 558.656 KiB)

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

0.000587 seconds (10.60 k allocations: 438.906 KiB)

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

0.000063 seconds (451 allocations: 18.719 KiB)

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

0.000522 seconds (7.59 k allocations: 359.406 KiB)
retcode: Success
Interpolation: 1st order linear
t: 1502-element Array{Float64,1}:
 0.0
 0.002
 0.004
 0.006
 0.008
 0.01
 0.012
 0.014
 0.016
 0.018000000000000002
 ⋮
 2.9859999999998927
 2.9879999999998925
 2.9899999999998923
 2.991999999999892
 2.993999999999892
 2.9959999999998916
 2.9979999999998914
 2.999999999999891
 3.0
u: 1502-element Array{Float64,1}:
  0.5
 -0.3202373479135767
 -1.208258622082896
 -0.7645486534810654
 -1.1537013767402515
```

```
 -0.8155284461567386
 -1.1617998351904886
 -0.8012036121776286
 -1.175763605387999
 -0.7692517166790679
  ⋮
 -0.999999998729584
 -1.0000000014056998
 -0.9999999986438097
 -1.000000001521946
 -0.9999999986318017
 -1.0000000014065296
 -0.9999999985719947
 -1.0000000012054135
 -1.000000001205414
```

1.1.3 Mixed Stiffness

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

0.002111 seconds (23.31 k allocations: 1.504 MiB)

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

0.118300 seconds (2.10 M allocations: 70.361 MiB)

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

0.277309 seconds (1.00 M allocations: 61.394 MiB)

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

0.105319 seconds (1.50 M allocations: 56.205 MiB)
retcode: Success
Interpolation: 1st order linear
t: 300001-element Array{Float64,1}:
 0.0
 1.0e-5
 2.0e-5
 3.0000000000000004e-5
 4.0e-5
 5.0e-5
 6.0e-5
 7.000000000000001e-5
 8.0e-5
 9.0e-5
  ⋮
 2.9999200000111856
 2.9999300000111857
 2.9999400000111858
 2.999950000011186
 2.999960000011186
 2.999970000011186
 2.999980000011186
 2.999990000011186
 3.0
u: 300001-element Array{Float64,1}:
```

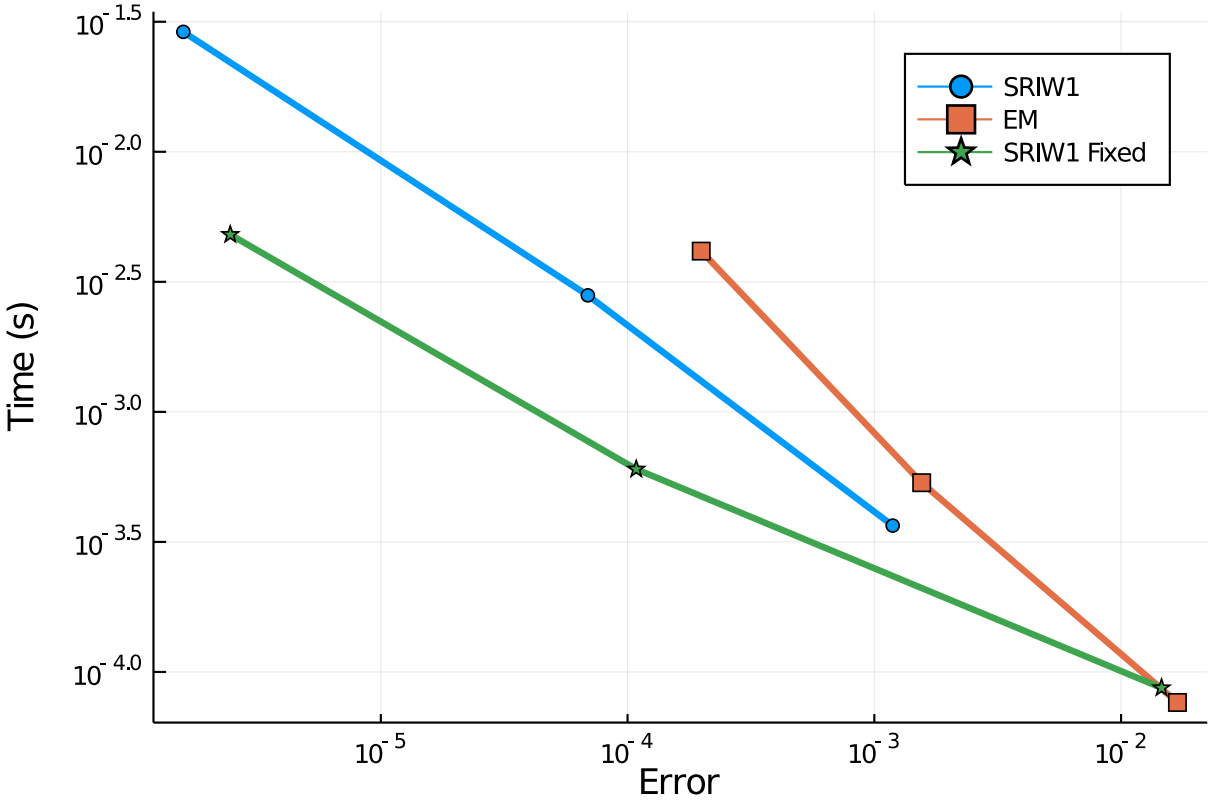
```
  0.5
  0.5020936165219779
  0.16417683769867658
  0.1232643715925845
  0.11328068468840269
  0.11519086196959863
  0.014483863260863608
 -0.08725406859150953
  0.41584545923654015
  0.4800546988013958
  ⋮
 -1.0
 -1.0
 -1.0
 -1.0
 -1.0
 -1.0
 -1.0
 -1.0
 -1.0
```

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

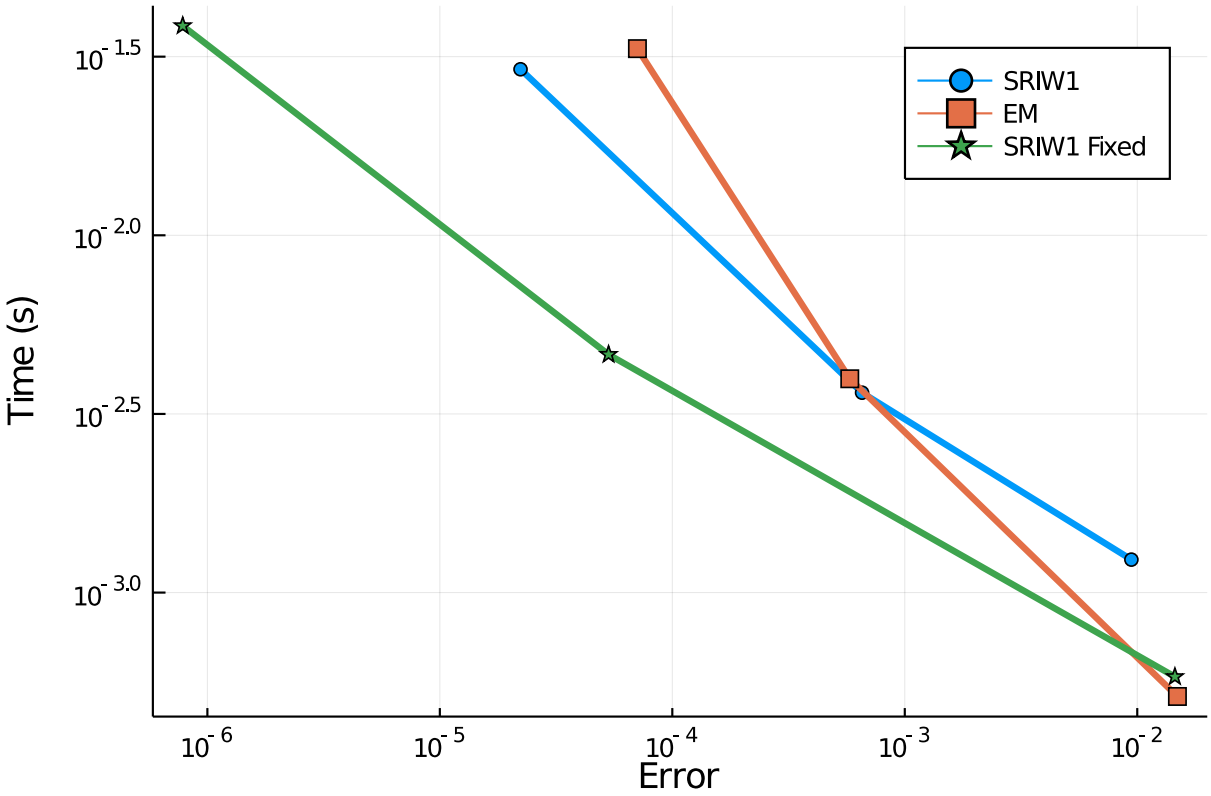
```
prob = remake(prob_sde_stiffquadito,p=(50.0,1.0))

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=true
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 1.4.2
Commit 44fa15b150* (2020-05-23 18:35 UTC)
Platform Info:
  OS: Linux (x86_64-pc-linux-gnu)
  CPU: Intel(R) Core(TM) i7-9700K CPU @ 3.60GHz
  WORD_SIZE: 64
  LIBM: libopenlibm
  LLVM: libLLVM-8.0.1 (ORCJIT, skylake)
```

```
Environment:
  JULIA_DEPOT_PATH = /builds/JuliaGPU/DiffEqBenchmarks.jl/.julia
  JULIA_CUDA_MEMORY_LIMIT = 2147483648
  JULIA_PROJECT = @.
  JULIA_NUM_THREADS = 8
```

Package Information:

```
Status: `/builds/JuliaGPU/DiffEqBenchmarks.jl/benchmarks/StiffSDE/Project.toml`
[f3b72e0c-5b89-59e1-b016-84e28bfd966d] DiffEqDevTools 2.22.0
[77a26b50-5914-5dd7-bc55-306e6241c503] DiffEqNoiseProcess 5.0.2
[a077e3f3-b75c-5d7f-a0c6-6bc4c8ec64a9] DiffEqProblemLibrary 4.8.0
[91a5bcdd-55d7-5caf-9e0b-520d859cae80] Plots 1.5.3
[789caeaf-c7a9-5a7d-9973-96adeb23e2a0] StochasticDiffEq 6.24.0
[37e2e46d-f89d-539d-b4ee-838fcccc9c8e] LinearAlgebra
[9a3f8284-a2c9-5f02-9a11-845980a1fd5c] Random
[10745b16-79ce-11e8-11f9-7d13ad32a3b2] Statistics
```