

Quadratic Stiffness Benchmarks

Chris Rackauckas

July 4, 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

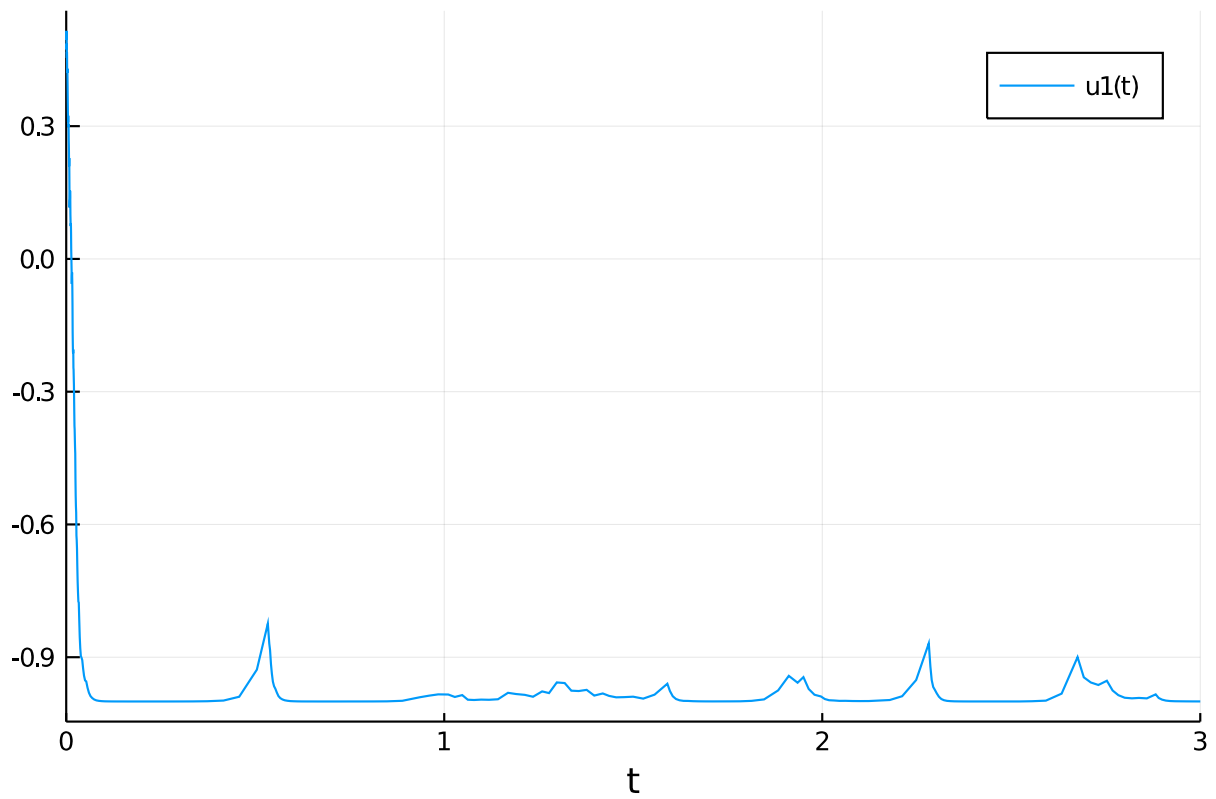
ASSYR ABDULLE AND STEPHANE CIRILLI

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

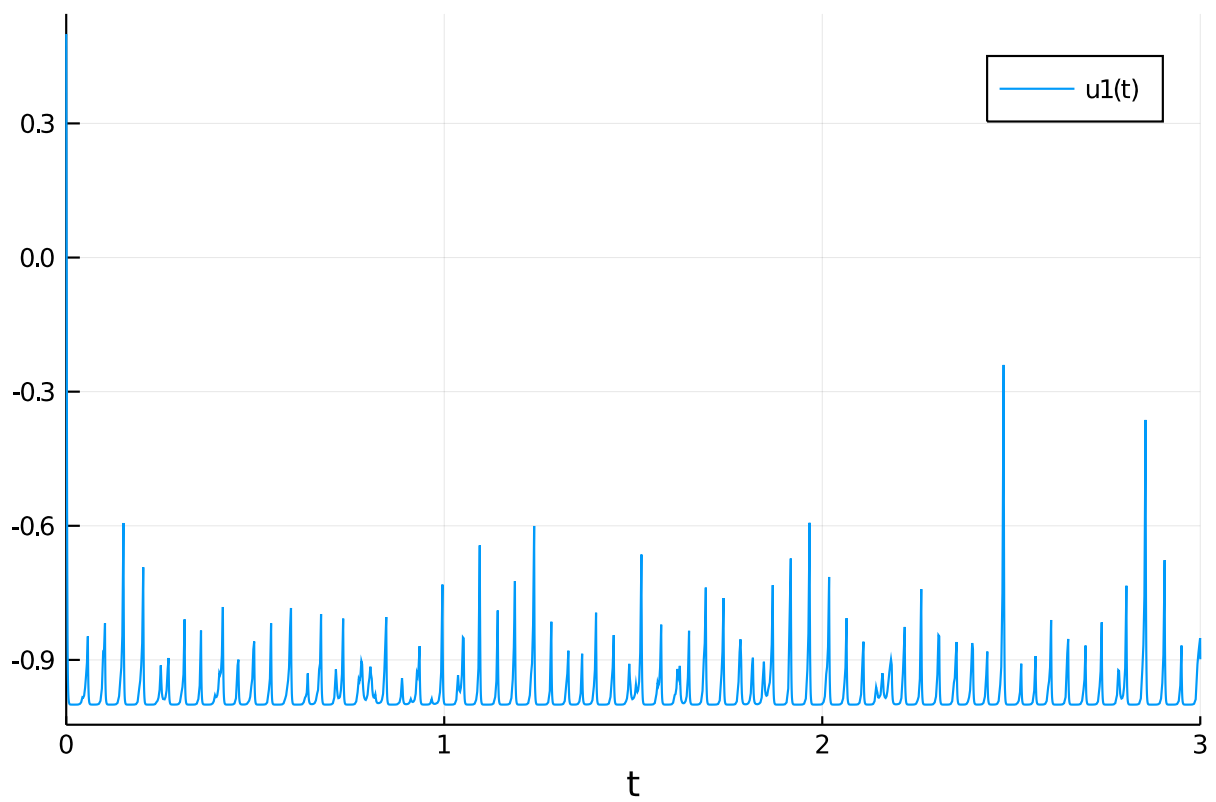
```
using DiffEqProblemLibrary, StochasticDiffEq, DiffEqDevTools
using DiffEqProblemLibrary.SDEProblemLibrary: importsdeproblems; importsdeproblems()
import DiffEqProblemLibrary.SDEProblemLibrary: prob_sde_stiffquadito
using Plots; gr()
const N = 10
```

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())

0.000138 seconds (1.78 k allocations: 70.375 KiB)

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

0.000145 seconds (2.20 k allocations: 98.656 KiB)

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

0.000058 seconds (351 allocations: 14.781 KiB)

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

0.000140 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
 ⋮
 2.92999999999999815
 2.93999999999999813
 2.9499999999999981
 2.9599999999999981
 2.96999999999999807
 2.97999999999999804
 2.98999999999999802
 2.999999999999998
 3.0
u: 302-element Array{Float64,1}:
 0.5
 0.05230950489470505
 -0.34342813440035214
 -0.8708484665405394
 -0.9928596317989153
 -0.9975384684914178
 -0.9999665557848525
 -0.9999939903566428
 -0.99999749922506
 -0.999999032763497
```

```

:
-1.0
-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())

0.000993 seconds (15.07 k allocations: 563.562 KiB)

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

0.000601 seconds (10.60 k allocations: 438.906 KiB)

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

0.000066 seconds (478 allocations: 22.344 KiB)

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

0.000525 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
:
 2.98599999999998927
 2.98799999999998925
 2.98999999999998923
 2.9919999999999892
 2.9939999999999892
 2.99599999999998916
 2.99799999999998914
 2.9999999999999891
 3.0
u: 1502-element Array{Float64,1}:
 0.5
-0.21001434728503593
-1.125284465626116
-0.8683262129415142
-1.116704795964397

```

```

-0.8720717993962069
-1.1227530057219544
-0.8743920798495215
-1.0994564426874924
-0.9047647418906125
⋮
-0.9989433194740754
-1.0010929273137
-0.9988929115884099
-1.0011261840008168
-0.9990064707051705
-1.0009436651751338
-0.9989824401586443
-1.0011290306602463
-1.0011290313685017

```

1.1.3 Mixed Stiffness

```

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

```

```

0.000855 seconds (8.11 k allocations: 526.156 KiB)

```

```

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

```

```

0.116019 seconds (2.10 M allocations: 70.361 MiB)

```

```

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

```

```

0.265202 seconds (1.00 M allocations: 61.394 MiB)

```

```

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

```

```

0.105074 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
0.620895138229494
0.30794272822485774
0.22708997989687202
0.12203856757448313
0.2820398500403145
0.14494009468123106
-0.06581899923129933
0.2953833598397875
0.702846965592292
:
-1.0
-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.

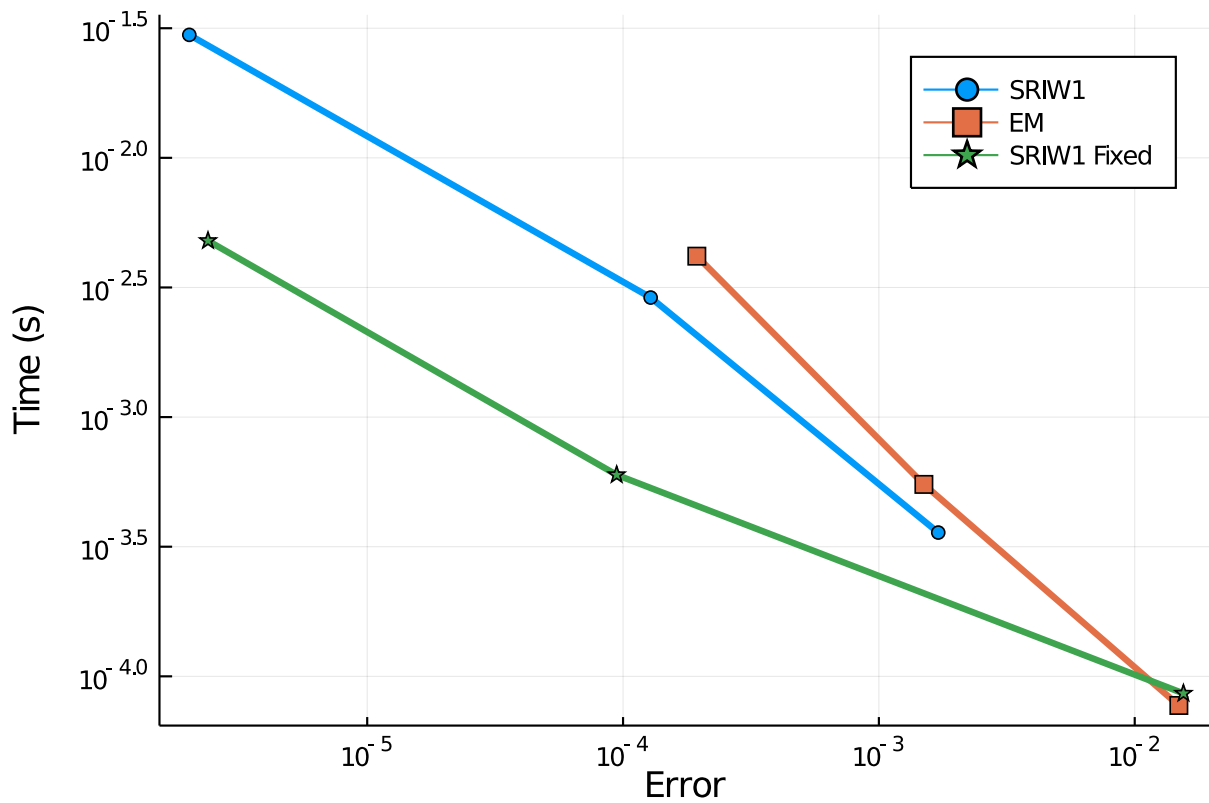
1.2 Work-Precision Diagrams

```

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)

```

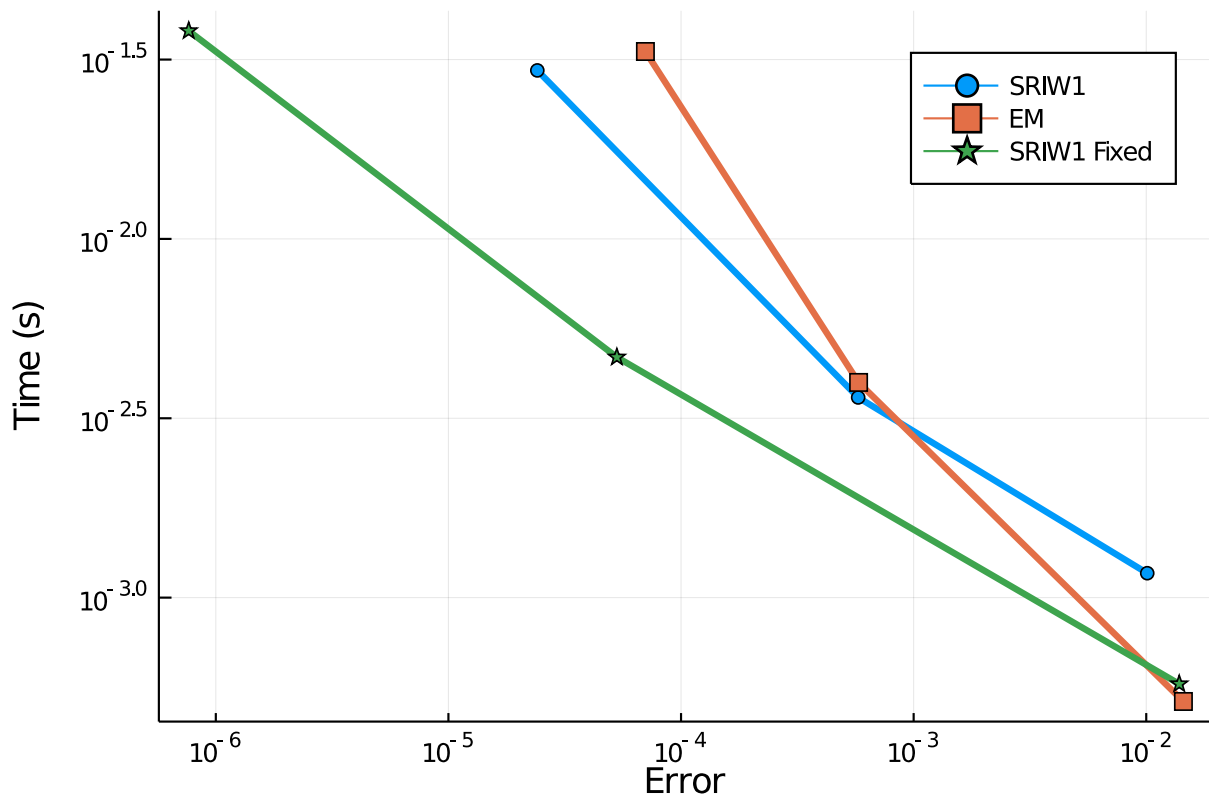


```

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/JuliaDiffEq/DiffEqBenchmarks.jl>

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 = 4
```

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 5.0.1
[a077e3f3-b75c-5d7f-a0c6-6bc4c8ec64a9] DiffEqProblemLibrary 4.8.0
[91a5bcdd-55d7-5caf-9e0b-520d859cae80] Plots 1.5.2
[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
```