Stochastic Heat Equation Benchmarks

Chris Rackauckas

May 5, 2019

1 Stochastic Heat Equation Benchmarks

In this notebook we will benchmark against the stochastic heat equation with Dirichlet BCs and scalar noise. The function for generating the problem is as follows:

Stochastic Heat Equation with scalar multiplicative noise

S-ROCK: CHEBYSHEV METHODS FOR STIFF STOCHASTIC DIFFERENTIAL EQUATIONS

ASSYR ABDULLE AND STEPHANE CIRILLI

Raising D or k increases stiffness

```
using StochasticDiffEq, DiffEqNoiseProcess, LinearAlgebra, Statistics

function generate_stiff_stoch_heat(D=1,k=1;N = 100, t_end = 3.0, adaptivealg = :RSwM3)
    A = Array(Tridiagonal([1.0 for i in 1:N-1],[-2.0 for i in 1:N],[1.0 for i in 1:N-1]))
    dx = 1/N
    A = D/(dx^2) * A
    function f(du,u,p,t)
        mul!(du,A,u)
    end
    #=function f(::Type{Val{:analytic}},u0,p,t,W)exp((A-k/2)*t+W*I)*u0 # no -k/2 for
    Stratend=#
    function g(du,u,p,t)
        @. du = k*u
    end

    SDEProblem(f,g,ones(N),(0.0,t_end),noise=WienerProcess(0.0,0.0,0.0,rswm=RSWM(adaptivealg=adaptivea
end

N = 100
D = 1; k = 1
    A = Array(Tridiagonal([1.0 for i in 1:N-1],[-2.0 for i in 1:N],[1.0 for i in 1:N-1]))
    dx = 1/N
    A = D/(dx^2) * A;
```

Now lets solve it with high accuracy.

```
prob = generate_stiff_stoch_heat(1.0,1.0)
@time sol = solve(prob,SRIW1(),progress=true,abstol=1e-6,reltol=1e-6);
```

```
29.276539 seconds (38.12 M allocations: 12.577 GiB, 13.89% gc time)
```

1.1 Highest dt

Let's try to find the highest possible dt:

```
@time sol = solve(generate_stiff_stoch_heat(1.0,1.0),SRIW1());
```

```
16.152796 seconds (31.76 M allocations: 11.935 GiB, 13.97% gc time)
```

```
@time sol =
    solve(generate_stiff_stoch_heat(1.0,1.0),SRIW1(),progress=true,adaptive=false,dt=0.00005);
```

```
1.680081 seconds (566.88 k allocations: 84.725 MiB, 22.80% gc time)
```

```
@time sol =
    solve(generate_stiff_stoch_heat(1.0,1.0),EM(),progress=true,adaptive=false,dt=0.00005);
```

```
16.678357 seconds (34.76 M allocations: 12.037 GiB, 14.05% gc time)
```

```
@time sol =
    solve(generate_stiff_stoch_heat(1.0,1.0),ImplicitRKMil(),progress=true,dt=0.1);
```

```
23.218279 seconds (36.32 M allocations: 12.277 GiB, 16.75% gc time)
```

```
@time sol =
    solve(generate_stiff_stoch_heat(1.0,1.0),ImplicitRKMil(),progress=true,dt=0.01);
```

```
1.693866 seconds (165.56 k allocations: 231.209 MiB, 2.39% gc time)
```

```
@time sol =
    solve(generate_stiff_stoch_heat(1.0,1.0),ImplicitRKMil(),progress=true,dt=0.001);
```

```
1.724042 seconds (165.11 k allocations: 230.566 MiB, 2.29% gc time)
```

```
@time sol =
    solve(generate_stiff_stoch_heat(1.0,1.0),ImplicitEM(),progress=true,dt=0.001);
```

```
21.085077 seconds (35.72 M allocations: 12.251 GiB, 14.88% gc time)
```

1.2 Simple Error Analysis

Now let's check the error at an arbitrary timepoint in there. Our analytical solution only exists in the Stratanovich sense, so we are limited in the methods we can calculate errors for.

```
function simple_error(alg;kwargs...)
    sol = solve(generate_stiff_stoch_heat(1.0,1.0,t_end=0.25),alg;kwargs...);
    sum(abs2,sol[end] - exp(A*sol.t[end]+sol.W[end]*I)*prob.u0)
end
```

```
simple_error (generic function with 1 method)
```

```
mean(simple_error(EulerHeun(),dt=0.00005) for i in 1:400)
```

```
3.264413901901836e-9
```

```
mean(simple_error(ImplicitRKMil(interpretation=:Stratanovich),dt=0.1) for i in 1:400)
```

```
0.014261798983766836
```

```
mean(simple_error(ImplicitRKMil(interpretation=:Stratanovich),dt=0.01) for i in 1:400)
```

```
0.014260847327379597
```

```
mean(simple_error(ImplicitRKMil(interpretation=:Stratanovich),dt=0.001) for i in 1:400)
```

```
0.014785158795471378
```

```
mean(simple_error(ImplicitEulerHeun(),dt=0.001) for i in 1:400)
```

```
0.00014594564690801603
```

```
mean(simple_error(ImplicitEulerHeun(),dt=0.01) for i in 1:400)
```

```
0.00014487073671060952
```

```
mean(simple_error(ImplicitEulerHeun(),dt=0.1) for i in 1:400)
```

```
0.00014838153613446603
```

1.3 Interesting Property

Note that RSwM1 and RSwM2 are not stable on this problem.

```
sol = solve(generate_stiff_stoch_heat(1.0,1.0,adaptivealg=:RSwM1),SRIW1());
```

1.4 Conclusion

In this problem, the implicit methods do not have a stepsize limit. This is because the stiffness almost entirely deteriministic due to diffusion. In that case, if we do not care about the error too much, the implicit methods dominate. Of course, as the tolerance gets lower there is a tradeoff point where the higher order methods will become more efficient. The explicit methods are clearly stability-bound and thus unless we want an error of like 10^-10 we are better off using an implicit method here.

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

1.5 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","StochasticHeat.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
```