Adaptive Efficiency Tests

Chris Rackauckas

May 4, 2019

```
using Distributed
addprocs(2)

p1 = Vector{Any}(undef,3)
p2 = Vector{Any}(undef,3)
p3 = Vector{Any}(undef,3)

@everywhere begin
  using DiffEqMonteCarlo, StochasticDiffEq, DiffEqProblemLibrary, DiffEqNoiseProcess,
    Plots, ParallelDataTransfer
  using DiffEqProblemLibrary.SDEProblemLibrary: importsdeproblems; importsdeproblems()
  import DiffEqProblemLibrary.SDEProblemLibrary: prob_sde_additive,
              prob_sde_linear, prob_sde_wave
end

using DiffEqMonteCarlo, StochasticDiffEq, DiffEqProblemLibrary, DiffEqNoiseProcess,
    Plots, ParallelDataTransfer
using DiffEqProblemLibrary.SDEProblemLibrary: importsdeproblems; importsdeproblems()
import DiffEqProblemLibrary.SDEProblemLibrary: prob_sde_additive,
            prob_sde_linear, prob_sde_wave

probs = Matrix{SDEProblem}(undef,3,3)
## Problem 1
prob = prob_sde_linear
probs[1,1] =
    SDEProblem(prob.f,prob.g,prob.u0,prob.tspan,prob.p,noise=WienerProcess(0.0,0.0,0.0,rswm=RSWM(adapt
probs[1,2] =
    SDEProblem(prob.f,prob.g,prob.u0,prob.tspan,prob.p,noise=WienerProcess(0.0,0.0,0.0,rswm=RSWM(adapt
probs[1,3] =
    SDEProblem(prob.f,prob.g,prob.u0,prob.tspan,prob.p,noise=WienerProcess(0.0,0.0,0.0,rswm=RSWM(adapt
## Problem 2
prob = prob_sde_wave
probs[2,1] =
    SDEProblem(prob.f,prob.g,prob.u0,prob.tspan,prob.p,noise=WienerProcess(0.0,0.0,0.0,rswm=RSWM(adapt
probs[2,2] =
    SDEProblem(prob.f,prob.g,prob.u0,prob.tspan,prob.p,noise=WienerProcess(0.0,0.0,0.0,rswm=RSWM(adapt
probs[2,3] =
    SDEProblem(prob.f,prob.g,prob.u0,prob.tspan,prob.p,noise=WienerProcess(0.0,0.0,0.0,rswm=RSWM(adapt
## Problem 3
prob = prob_sde_additive
probs[3,1] =
    SDEProblem(prob.f,prob.g,prob.u0,prob.tspan,prob.p,noise=WienerProcess(0.0,0.0,0.0,rswm=RSWM(adapt
probs[3,2] =
    SDEProblem(prob.f,prob.g,prob.u0,prob.tspan,prob.p,noise=WienerProcess(0.0,0.0,0.0,rswm=RSWM(adapt
probs[3,3] =
    SDEProblem(prob.f,prob.g,prob.u0,prob.tspan,prob.p,noise=WienerProcess(0.0,0.0,0.0,rswm=RSWM(adapt
```

```julia
fullMeans = Vector{Array}(undef,3)
fullMedians = Vector{Array}(undef,3)
fullElapsed = Vector{Array}(undef,3)
fullTols = Vector{Array}(undef,3)
offset = 0

Ns = [17,23,
17]
```

```
3-element Array{Int64,1}:
 17
 23
 17
```

Timings are only valid if no workers die. Workers die if you run out of memory.

```julia
for k in 1:size(probs,1)
  global probs, Ns, fullMeans, fullMedians, fullElapsed, fullTols
  println("Problem $k")
  ## Setup
  N = Ns[k]

  msims = Vector{Any}(undef,N)
  elapsed = Array{Float64}(undef,N,3)
  medians = Array{Float64}(undef,N,3)
  means   = Array{Float64}(undef,N,3)
  tols    = Array{Float64}(undef,N,3)

  #Compile
  prob = probs[k,1]
  ParallelDataTransfer.sendto(workers(), prob=prob)
  monte_prob = MonteCarloProblem(prob)

    solve(monte_prob,SRIW1(),dt=1/2^(4),adaptive=true,num_monte=1000,abstol=2.0^(-1),reltol=0)

  println("RSwM1")
  for i=1+offset:N+offset
    tols[i-offset,1] = 2.0^(-i-1)
    msims[i-offset] = DiffEqBase.calculate_monte_errors(solve(monte_prob,SRIW1(),
                                        num_monte=1000,abstol=2.0^(-i-1),
                                        reltol=0,force_dtmin=true))
    elapsed[i-offset,1] = msims[i-offset].elapsedTime
    medians[i-offset,1] = msims[i-offset].error_medians[:final]
    means[i-offset,1]   = msims[i-offset].error_means[:final]
  end

  println("RSwM2")
  prob = probs[k,2]

  ParallelDataTransfer.sendto(workers(), prob=prob)
  monte_prob = MonteCarloProblem(prob)

    solve(monte_prob,SRIW1(),dt=1/2^(4),adaptive=true,num_monte=1000,abstol=2.0^(-1),reltol=0)

  for i=1+offset:N+offset
    tols[i-offset,2] = 2.0^(-i-1)
    msims[i-offset] = DiffEqBase.calculate_monte_errors(solve(monte_prob,SRIW1(),
                                        num_monte=1000,abstol=2.0^(-i-1),
                                        reltol=0,force_dtmin=true))
```

```
    elapsed[i-offset,2] = msims[i-offset].elapsedTime
    medians[i-offset,2] = msims[i-offset].error_medians[:final]
    means[i-offset,2]   = msims[i-offset].error_means[:final]
  end

  println("RSwM3")
  prob = probs[k,3]
  ParallelDataTransfer.sendto(workers(), prob=prob)
  monte_prob = MonteCarloProblem(prob)

    solve(monte_prob,SRIW1(),dt=1/2^(4),adaptive=true,num_monte=1000,abstol=2.0^(-1),reltol=0)

  for i=1+offset:N+offset
    tols[i-offset,3] = 2.0^(-i-1)
        msims[i-offset] = DiffEqBase.calculate_monte_errors(solve(monte_prob,SRIW1(),
                                    adaptive=true,num_monte=1000,abstol=2.0^(-i-1),
                                    reltol=0,force_dtmin=true))
    elapsed[i-offset,3] = msims[i-offset].elapsedTime
    medians[i-offset,3] = msims[i-offset].error_medians[:final]
    means[i-offset,3]   = msims[i-offset].error_means[:final]
  end

  fullMeans[k] = means
  fullMedians[k] =medians
  fullElapsed[k] = elapsed
  fullTols[k] = tols
end
```

```
Problem 1
RSwM1
RSwM2
Error: KeyError: key :l∞ not found
```

```
gr(fmt=:svg)
lw=3
leg=String["RSwM1","RSwM2","RSwM3"]

titleFontSize = 16
guideFontSize = 14
legendFontSize= 14
tickFontSize  = 12

for k in 1:size(probs,1)
  global probs, Ns, fullMeans, fullMedians, fullElapsed, fullTols
  p1[k] = Plots.plot(fullTols[k],fullMeans[k],xscale=:log10,yscale=:log10,
    xguide="Absolute Tolerance",yguide="Mean Final Error",title="Example
    $k"
    ,linewidth=lw,grid=false,lab=leg,titlefont=font(titleFontSize),legendfont=font(legendFontSize),tic
  p2[k] =
    Plots.plot(fullTols[k],fullMedians[k],xscale=:log10,yscale=:log10,xguide="Absolute
    Tolerance",yguide="Median Final Error",title="Example
    $k",linewidth=lw,grid=false,lab=leg,titlefont=font(titleFontSize),legendfont=font(legendFontSize),
  p3[k] =
    Plots.plot(fullTols[k],fullElapsed[k],xscale=:log10,yscale=:log10,xguide="Absolute
    Tolerance",yguide="Elapsed Time",title="Example $k"
    ,linewidth=lw,grid=false,lab=leg,titlefont=font(titleFontSize),legendfont=font(legendFontSize),tic
end
```

```
Error: UndefRefError: access to undefined reference
```

```
Plots.plot!(p1[1])
```

```
Error: UndefRefError: access to undefined reference
```

```
Plots.plot(p1[1],p1[2],p1[3],layout=(3,1),size=(1000,800))
```

```
Error: UndefRefError: access to undefined reference
```

```
#savefig("meanvstol.png")
#savefig("meanvstol.pdf")
```

```
plot(p3[1],p3[2],p3[3],layout=(3,1),size=(1000,800))
```

```
Error: UndefRefError: access to undefined reference
```

```
#savefig("timevstol.png")
#savefig("timevstol.pdf")
```

```
plot(p1[1],p3[1],p1[2],p3[2],p1[3],p3[3],layout=(3,2),size=(1000,800))
```

```
Error: UndefRefError: access to undefined reference
```

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

0.1 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("AdaptiveSDE","AdaptiveEfficiencyTests.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 3.1.0+
[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
[90137ffa-7385-5640-81b9-e52037218182] StaticArrays 0.10.3
[789caeaf-c7a9-5a7d-9973-96adeb23e2a0] StochasticDiffEq 6.1.1+
[c3572dad-4567-51f8-b174-8c6c989267f4] Sundials 3.4.1
[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
```