Adaptive Efficiency Tests

Chris Rackauckas

May 8, 2021

```
using Distributed
addprocs(2)

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

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

using 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(adaptivea
probs[1,2] =
SDEProblem(prob.f,prob.g,prob.u0,prob.tspan,prob.p,noise=WienerProcess(0.0,0.0,0.0,rswm=RSWM(adaptivea
probs[1,3] =
SDEProblem(prob.f,prob.g,prob.u0,prob.tspan,prob.p,noise=WienerProcess(0.0,0.0,0.0,rswm=RSWM(adaptivea
## 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(adaptivea
probs[2,2] =
SDEProblem(prob.f,prob.g,prob.u0,prob.tspan,prob.p,noise=WienerProcess(0.0,0.0,0.0,rswm=RSWM(adaptivea
probs[2,3] =
SDEProblem(prob.f,prob.g,prob.u0,prob.tspan,prob.p,noise=WienerProcess(0.0,0.0,0.0,rswm=RSWM(adaptivea
## 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(adaptivea
probs[3,2] =
SDEProblem(prob.f,prob.g,prob.u0,prob.tspan,prob.p,noise=WienerProcess(0.0,0.0,0.0,rswm=RSWM(adaptivea
probs[3,3] =
SDEProblem(prob.f,prob.g,prob.u0,prob.tspan,prob.p,noise=WienerProcess(0.0,0.0,0.0,rswm=RSWM(adaptivea
```

```
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]
```

```
Error: On worker 2:
ArgumentError: Package DiffEqProblemLibrary [a077e3f3-b75c-5d7f-a0c6-6bc4c8
ec64a9] is required but does not seem to be installed:
 - Run `Pkg.instantiate()` to install all recorded dependencies.

Stacktrace:
 [1] _require
   @ ./loading.jl:990
 [2] require
   @ ./loading.jl:914
 [3] #1
   @ /buildworker/worker/package_linux64/build/usr/share/julia/stdlib/v1.6/
Distributed/src/Distributed.jl:79
 [4] #103
   @ /buildworker/worker/package_linux64/build/usr/share/julia/stdlib/v1.6/
Distributed/src/process_messages.jl:274
 [5] run_work_thunk
   @ /buildworker/worker/package_linux64/build/usr/share/julia/stdlib/v1.6/
Distributed/src/process_messages.jl:63
 [6] run_work_thunk
   @ /buildworker/worker/package_linux64/build/usr/share/julia/stdlib/v1.6/
Distributed/src/process_messages.jl:72
 [7] #96
   @ ./task.jl:411

...and 3 more exceptions.
```

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

```
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 = EnsembleProblem(prob)

solve(monte_prob,SRIW1(),dt=1/2^(4),adaptive=true,trajectories=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(),
                                          trajectories=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 = EnsembleProblem(prob)

solve(monte_prob,SRIW1(),dt=1/2^(4),adaptive=true,trajectories=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(),
                                          trajectories=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 = EnsembleProblem(prob)

solve(monte_prob,SRIW1(),dt=1/2^(4),adaptive=true,trajectories=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,trajectories=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

Error: UndefVarError: probs not defined

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),tickfor
  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),tick
  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),tickfor
end

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

```
Error: UndefVarError: gr not defined
```

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

plot(p3[1],p3[2],p3[3],layout=(3,1),size=(1000,800))
#savefig("timevstol.png")
#savefig("timevstol.pdf")
```

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

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

```
Error: ArgumentError: Package SciMLBenchmarks not found in current path:
- Run `import Pkg; Pkg.add("SciMLBenchmarks")` to install the SciMLBenchmar
ks package.
```