Runge-Kutta Benchmarks on Linear ODEs

Chris Rackauckas

March 2, 2019

1 Runge-Kutta Benchmarks on Linear ODEs

This notebook is to document benchmarks between the Runge-Kutta solvers implemented in Julia. These are the implementations native to OrdinaryDiffEq.jl, ODE.jl, and those the classic Hairer methods provided by ODEInterface.jl. We will use OrdinaryDiffEq.jl's benchmarking framework for performing these tests. The purpose of this notebook is to complement the work-precision diagrams by showing how the same algorithm (Runge-Kutta Order 4) is many times faster in OrdinaryDiffEq.jl than in ODE.jl, and that OrdinaryDiffEq.jl specializes on scalar problems for more efficiency gains.

First, let's define our problems:

```
using OrdinaryDiffEq, ODE, ODEInterfaceDiffEq, DiffEqDevTools
tspan = (0.0,1.0)
# Linear ODE
f = (u,p,t) -> (1.01*u)
(::typeof(f))(::Type{Val{:analytic}},u_0,p,t) = u_0*exp(1.01*t)
probnum = ODEProblem(f,1/2,tspan)

# 2D Linear ODE
f = (du,u,p,t) -> begin
  @inbounds for i in eachindex(u)
    du[i] = 1.01*u[i]
  end
end
(::typeof(f))(::Type{Val{:analytic}},u_0,p,t) = u_0*exp(1.01*t)
prob = ODEProblem(f,rand(100,100),tspan)
using Plots; gr()
```

Here, `probnum` is a simply the 1-dimensional linear ODE, and `prob` is a 100x100 matrix of linear ODEs.

Although the linear ODE may be the most "basic" case, the linear ODE on [0,10] has some interesting qualities. Since it rises so fast in the end, most of the error is accumulated there. However, error from earlier is compounded in later stages. Therefore, in order for an algorithm to achieve low error, it has to be able to evenly distribute its error (in percentage) about the whole problem. Thus it is a good test of the ability for the adaptivity algorithm to control error globally. Also, the small time it takes for the actual function calculations better highlights the differences between implementations. Lastly, the steep rise in the end can be a problem for methods trying to scrape by with higher error but faster speeds. This means that lower order methods tend to get trapped, but too high of an order simply costs too much. Who will do best? Let's see what we get!