

Three Body Work-Precision Diagrams

Chris Rackauckas

March 2, 2019

```
using OrdinaryDiffEq, ODE, ODEInterfaceDiffEq, LSODA, Sundials, DiffEqDevTools, Plots;
gr()

## Define the ThreeBody Problem
const threebody_μ = parse(Float64, "0.012277471")
const threebody_μ' = 1 - threebody_μ

f = (du,u,p,t) -> begin
    @inbounds begin
        # 1 = y_1
        # 2 = y_2
        # 3 = y_1'
        # 4 = y_2'
        D_1 = ((u[1]+threebody_μ)^2 + u[2]^2)^(3/2)
        D_2 = ((u[1]-threebody_μ')^2 + u[2]^2)^(3/2)
        du[1] = u[3]
        du[2] = u[4]
        du[3] = u[1] + 2u[4] - threebody_μ'*(u[1]+threebody_μ)/D_1 -
            threebody_μ*(u[1]-threebody_μ')/D_2
        du[4] = u[2] - 2u[3] - threebody_μ'*u[2]/D_1 - threebody_μ*u[2]/D_2
    end
end

t_0 = 0.0; T = parse(Float64, "17.0652165601579625588917206249")
tspan = (t_0, 2T)

prob = ODEProblem(f, [0.994, 0.0, 0.0,
    parse(Float64, "-2.00158510637908252240537862224")], tspan)

test_sol = TestSolution(T, [prob.u0])
abstols = 1.0 ./ 10.0 .^ (3:13)
```

See that it's periodic in the chosen timespan:

```
#sol = solve(prob, Vern9(), abstol=1e-14, reltol=1e-14)
#@show sol[1] - sol[end]
#@show sol[end] - prob.u0;

#apr = appxtrue(sol, test_sol)
#@show sol[end]
#@show apr.u[end]
#@show apr.errors
```

This three-body problem is known to be a tough problem. Let's see how the algorithms fair at standard tolerances.

0.0.1 5th Order Runge-Kutta Methods

```
setups = [Dict(:alg=>DP5())
          #Dict(:alg=>ode45()) #fails
          Dict(:alg=>BS5())
          Dict(:alg=>Tsit5())
          Dict(:alg=>dopri5())];
wp =
  WorkPrecisionSet(prob, abstols, reltols, setups; appxsol=test_sol, save_everystep=false, numruns=100)
```

Error: UndefVarError: reltols not defined

```
plot(wp)
```

Error: UndefVarError: wp not defined

```
setups = [Dict(:alg=>DP5(), :dense=>false)
          #Dict(:alg=>ode45()) # Fails
          Dict(:alg=>BS5(), :dense=>false)
          Dict(:alg=>Tsit5(), :dense=>false)
          Dict(:alg=>dopri5())];
wp = WorkPrecisionSet(prob, abstols, reltols, setups; appxsol=test_sol, numruns=100)
```

Error: UndefVarError: reltols not defined

```
plot(wp)
```

Error: UndefVarError: wp not defined

```
setups = [Dict(:alg=>DP5())
          #Dict(:alg=>ode45()) #fails
          Dict(:alg=>BS5())
          Dict(:alg=>Tsit5())
          Dict(:alg=>dopri5())];
wp = WorkPrecisionSet(prob, abstols, reltols, setups; appxsol=test_sol, numruns=100)
```

Error: UndefVarError: reltols not defined

```
plot(wp)
```

Error: UndefVarError: wp not defined

In these tests we see that most of the algorithms are close, with BS5 and DP5 showing much better than Tsit5. ode45 errors.

0.0.2 Higher Order Algorithms

```
setups = [Dict(:alg=>DP5())
          Dict(:alg=>Vern6())
          Dict(:alg=>Vern7())
          Dict(:alg=>TanYam7())
          Dict(:alg=>Vern8())
          Dict(:alg=>DP8())
```

```

Dict(:alg=>dop853())
Dict(:alg=>Vern9())];
wp =
  WorkPrecisionSet(prob, abstols, reltols, setups; appxsol=test_sol, save_everystep=false, numruns=100)

```

Error: UndefVarError: reltols not defined

```
plot(wp)
```

Error: UndefVarError: wp not defined

```

setups = [Dict(:alg=>DP5())
          Dict(:alg=>Vern6())
          Dict(:alg=>Vern7())
          Dict(:alg=>TanYam7())
          Dict(:alg=>Vern8())
          Dict(:alg=>DP8())
          Dict(:alg=>dop853())
          Dict(:alg=>Vern9())];
wp =
  WorkPrecisionSet(prob, abstols, reltols, setups; appxsol=test_sol, dense=false, numruns=100, verbose=false)

```

Error: UndefVarError: reltols not defined

```
plot(wp)
```

Error: UndefVarError: wp not defined

```

setups = [Dict(:alg=>DP5())
          Dict(:alg=>Vern6())
          Dict(:alg=>Vern7())
          Dict(:alg=>TanYam7())
          Dict(:alg=>Vern8())
          Dict(:alg=>DP8())
          Dict(:alg=>dop853())
          Dict(:alg=>Vern9())];
wp = WorkPrecisionSet(prob, abstols, reltols, setups; appxsol=test_sol, numruns=100)

```

Error: UndefVarError: reltols not defined

```
plot(wp)
```

Error: UndefVarError: wp not defined

In this test we see Vern7 and Vern8 shine.

0.0.3 Other Algorithms

Once again we separate ODE.jl because it fails. We also separate Sundials' CVODE_Adams since it fails at high tolerances.

```

setups = [Dict(:alg=>ode78())
          Dict(:alg=>CVODE_Adams())];
wp =
  WorkPrecisionSet(prob, abstols, reltols, setups; appxsol=test_sol, dense=false, numruns=100)

```

Error: UndefVarError: reltols not defined

```

setups = [Dict(:alg=>DP5())
          Dict(:alg=>lsoda())
          Dict(:alg=>Vern8())
          Dict(:alg=>ddeabm())
          Dict(:alg=>odex())
          Dict(:alg=>ARKODE(Sundials.Explicit(),order=6))
        ];
wp =
    WorkPrecisionSet(prob, abstols, reltols, setups; appxsol=test_sol, save_everystep=false, numruns=100)

```

Error: UndefVarError: reltols not defined

```
plot(wp)
```

Error: UndefVarError: wp not defined

Again, on cheap function calculations the Adams methods are shown to not be efficient once the error is sufficiently small. Also, as seen in other places, the extrapolation methods do not fare as well as the Runge-Kutta methods.

0.0.4 Conclusion

As in the other tests, the OrdinaryDiffEq.jl algorithms with the Verner Efficient methods are the most efficient solvers at stringent tolerances for most of the tests, while the order 5 methods do well at cruder tolerances. ODE.jl fails to run the test problems without erroring.

```

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/JuliaDiffEq/DiffEqBenchmarks.jl>

To locally run this tutorial, do the following commands:

```

using DiffEqBenchmarks
DiffEqBenchmarks.weave_file("NonStiffODE", "ThreeBody_wpd.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/yingboma/.julia/dev/DiffEqBenchmarks/Project.toml`  
[f3b72e0c-5b89-59e1-b016-84e28bfd966d] DiffEqDevTools 2.7.0  
[7073ff75-c697-5162-941a-fcdaad2a7d2a] IJulia 1.17.0  
[7f56f5a3-f504-529b-bc02-0b1fe5e64312] LSODA 0.4.0  
[c030b06c-0b6d-57c2-b091-7029874bd033] ODE 2.4.0  
[09606e27-ecf5-54fc-bb29-004bd9f985bf] ODEInterfaceDiffEq 3.0.0  
[1dea7af3-3e70-54e6-95c3-0bf5283fa5ed] OrdinaryDiffEq 5.3.0  
[65888b18-ceab-5e60-b2b9-181511a3b968] ParameterizedFunctions 4.1.1  
[91a5bcdd-55d7-5caf-9e0b-520d859cae80] Plots 0.23.1  
[c3572dad-4567-51f8-b174-8c6c989267f4] Sundials 3.1.0  
[44d3d7a6-8a23-5bf8-98c5-b353f8df5ec9] Weave 0.7.2  
[b77e0a4c-d291-57a0-90e8-8db25a27a240] InteractiveUtils  
[d6f4376e-aef5-505a-96c1-9c027394607a] Markdown  
[44cfe95a-1eb2-52ea-b672-e2afdf69b78f] Pkg  
[9a3f8284-a2c9-5f02-9a11-845980a1fd5c] Random
```