Single Pedulum Comparison

Gen Kuroki (), Chris Rackauckas

April 23, 2019

1 Table of Contents

2 Solving single pendulums by DifferentialEquations.jl

In this notebook, we shall solve the single pendulum equation:

$$\ddot{q} = -\sin q,$$

where q means the angle.

Hamiltonian:

$$H(q,p) = \frac{1}{2}p^2 - \cos q + 1.$$

Canonical equation:

$$\dot{q} = p, \quad \dot{p} = -\sin q.$$

Initial condition:

$$q(0) = 0, \quad p(0) = 2k.$$

Exact solution:

$$q(t) = 2\arcsin(k\,\mathrm{sn}(t,k)).$$

Maximum of $q(t)$:

$$\sin(q_{\max}/2) = k, \quad q_{\max} = \max\{q(t)\}.$$

Define $y(t)$ by

$$y(t) = \sin(q(t)/2) = k\,\mathrm{sn}(t,k), \quad y_{\max} = k.$$

```
# Single pendulums shall be solved numerically.
#
using OrdinaryDiffEq, Elliptic, Printf, DiffEqPhysics, Statistics

sol2q(sol) = [sol.u[i][j] for i in 1:length(sol.u), j in 1:length(sol.u[1])÷2]
sol2p(sol) = [sol.u[i][j] for i in 1:length(sol.u), j in
    length(sol.u[1])÷2+1:length(sol.u[1])]
sol2tqp(sol) = (sol.t, sol2q(sol), sol2p(sol))

# The exact solutions of single pendulums can be expressed by the Jacobian elliptic
    functions.
#
sn(u, k) = Jacobi.sn(u, k^2) # the Jacobian sn function

# Use PyPlot.
#
using PyPlot

colorlist = [
    "#1f77b4", "#ff7f0e", "#2ca02c", "#d62728", "#9467bd",
    "#8c564b", "#e377c2", "#7f7f7f", "#bcbd22", "#17becf",
]
cc(k) = colorlist[mod1(k, length(colorlist))]

# plot the sulution of a Hamiltonian problem
#
function plotsol(sol::ODESolution)
    local t, q, p
    t, q, p = sol2tqp(sol)
    local d = size(q)[2]
    for j in 1:d
        j_str = d > 1 ? "[$j]" : ""
        plot(t, q[:,j], color=cc(2j-1), label="q$(j_str)", lw=1)
        plot(t, p[:,j], color=cc(2j),   label="p$(j_str)", lw=1, ls="-")
    end
    grid(ls=":")
    xlabel("t")
    legend()
end

# plot the solution of a Hamiltonian problem on the 2D phase space
#
function plotsol2(sol::ODESolution)
    local t, q, p
    t, q, p = sol2tqp(sol)
    local d = size(q)[2]
    for j in 1:d
        j_str = d > 1 ? "[$j]" : ""
        plot(q[:,j], p[:,j], color=cc(j), label="(q$(j_str),p$(j_str))", lw=1)
    end
    grid(ls=":")
    xlabel("q")
```

```
    ylabel("p")
    legend()
end

# plot the energy of a Hamiltonian problem
#
function plotenergy(H, sol::ODESolution)
    local t, q, p
    t, q, p = sol2tqp(sol)
    local energy = [H(q[i,:], p[i,:], nothing) for i in 1:size(q)[1]]
    plot(t, energy, label="energy", color="red", lw=1)
    grid(ls=":")
    xlabel("t")
    legend()
    local stdenergy_str = @sprintf("%.3e", std(energy))
    title(" std(energy) = $stdenergy_str", fontsize=10)
end

# plot the numerical and exact solutions of a single pendulum
#
# Warning:  Assume q(0) = 0, p(0) = 2k.  (for the sake of laziness)
#
function plotcomparison(k, sol::ODESolution)
    local t, q, p
    t, q, p = sol2tqp(sol)
    local y = sin.(q/2)
    local y_exact = k*sn.(t, k) # the exact solution

    plot(t, y,       label="numerical", lw=1)
    plot(t, y_exact, label="exact",     lw=1, ls="-")
    grid(ls=":")
    xlabel("t")
    ylabel("y = sin(q(t)/2)")
    legend()
    local error_str = @sprintf("%.3e", maximum(abs.(y - y_exact)))
    title("maximum(abs(numerical - exact)) = $error_str", fontsize=10)
end

# plot solution and energy
#
function plotsolenergy(H, integrator, Δt, sol::ODESolution)
    local integrator_str = replace("$integrator", r"^[^.]*\." => "")

    figure(figsize=(10,8))

    subplot2grid((21,20), ( 1, 0), rowspan=10, colspan=10)
    plotsol(sol)

    subplot2grid((21,20), ( 1,10), rowspan=10, colspan=10)
    plotsol2(sol)

    subplot2grid((21,20), (11, 0), rowspan=10, colspan=10)
    plotenergy(H, sol)

    suptitle("===== $integrator_str, Δt = $Δt=====")
end

# Solve a single pendulum
#
```

```
function singlependulum(k, integrator, Δt; t0 = 0.0, t1 = 100.0)
    local H(p,q,params) = p[1]^2/2 - cos(q[1]) + 1
    local q0 = [0.0]
    local p0 = [2k]
    local prob = HamiltonianProblem(H, p0, q0, (t0, t1))

    local integrator_str = replace("$integrator", r"^[^.]*\." => "")
    @printf("%-25s", "$integrator_str:")
    @time local sol = solve(prob, integrator, dt=Δt)

    sleep(0.1)
    figure(figsize=(10,8))

    subplot2grid((21,20), ( 1, 0), rowspan=10, colspan=10)
    plotsol(sol)

    subplot2grid((21,20), ( 1,10), rowspan=10, colspan=10)
    plotsol2(sol)

    subplot2grid((21,20), (11, 0), rowspan=10, colspan=10)
    plotenergy(H, sol)

    subplot2grid((21,20), (11,10), rowspan=10, colspan=10)
    plotcomparison(k, sol)

    suptitle("===== $integrator_str, Δt = $Δt=====")
end
```

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

2.1 Tests

http://docs.juliadiffeq.org/latest/types/dynamical_types.html#Hamiltonian-Problems-1

http://docs.juliadiffeq.org/latest/solvers/dynamical_solve.html

```
# Single pendulum

k = rand()
integrator = VelocityVerlet()
Δt = 0.1
singlependulum(k, integrator, Δt, t0=-20.0, t1=20.0)
```

```
VelocityVerlet():           1.541034 seconds (3.14 M allocations: 159.335 Mi
B)
PyObject Text(0.5, 0.98, '=====    VelocityVerlet(),    Δt = 0.1    =====')
```

```
# Two single pendulums

H(q,p,param) = sum(p.^2/2 .- cos.(q) .+ 1)
q0 = pi*rand(2)
p0 = zeros(2)
t0, t1 = -20.0, 20.0
prob = HamiltonianProblem(H, q0, p0, (t0, t1))

integrator = VelocityVerlet()
Δt = 0.1
@time sol = solve(prob, integrator, dt=Δt)
```

```
1.694019 seconds (3.22 M allocations: 162.926 MiB)
```

```
sleep(0.1)
plotsolenergy(H, integrator, Δt, sol)
```

```
PyObject Text(0.5, 0.98, '=====    VelocityVerlet(),   Δt = 0.1    =====')
```

2.2 Comparison of symplectic Integrators

http://docs.juliadiffeq.org/latest/solvers/dynamical_solve.html#Symplectic-Integrators-1

```
SymplecticIntegrators = [
    SymplecticEuler(),
    VelocityVerlet(),
    VerletLeapfrog(),
    PseudoVerletLeapfrog(),
    McAte2(),
    Ruth3(),
    McAte3(),
    CandyRoz4(),
    McAte4(),
    CalvoSanz4(),
    McAte42(),
    McAte5(),
    Yoshida6(),
    KahanLi6(),
    McAte8(),
    KahanLi8(),
    SofSpa10(),
]
```

```
k = 0.999
Δt = 0.1
for integrator in SymplecticIntegrators
    singlependulum(k, integrator, Δt)
end
```

```
SymplecticEuler():          1.353551 seconds (2.75 M allocations: 139.078 Mi
B)
VelocityVerlet():           0.001053 seconds (17.13 k allocations: 936.938 K
iB)
VerletLeapfrog():           1.382036 seconds (2.95 M allocations: 146.229 Mi
B)
PseudoVerletLeapfrog():     1.352417 seconds (2.91 M allocations: 144.421 Mi
B)
McAte2():                   1.369248 seconds (2.91 M allocations: 144.120 Mi
B)
Ruth3():                    1.650907 seconds (3.05 M allocations: 149.776 Mi
B)
McAte3():                   1.420511 seconds (3.03 M allocations: 148.310 Mi
B)
CandyRoz4():                1.479580 seconds (3.17 M allocations: 153.728 Mi
B)
McAte4():                   1.453108 seconds (3.14 M allocations: 152.235 Mi
B)
CalvoSanz4():               1.502208 seconds (3.24 M allocations: 156.492 Mi
B)
McAte42():                  1.490374 seconds (3.25 M allocations: 156.116 Mi
```

```
B)
McAte5():                   1.564355 seconds (3.40 M allocations: 162.134 Mi
B)
Yoshida6():                 2.294495 seconds (3.58 M allocations: 168.476 Mi
B, 26.82% gc time)
KahanLi6():                 1.885492 seconds (3.81 M allocations: 176.420 Mi
B, 7.60% gc time)
McAte8():                   2.184856 seconds (4.49 M allocations: 200.254 Mi
B, 7.37% gc time)
KahanLi8():                 2.576592 seconds (4.72 M allocations: 208.294 Mi
B, 17.52% gc time)
SofSpa10():                 3.318507 seconds (6.79 M allocations: 280.692 Mi
B, 7.25% gc time)
```

```
k = 0.999
Δt = 0.01
for integrator in SymplecticIntegrators[1:4]
    singlependulum(k, integrator, Δt)
end
```

```
SymplecticEuler():          0.026864 seconds (170.11 k allocations: 8.627 Mi
B, 78.80% gc time)
VelocityVerlet():           0.005964 seconds (170.11 k allocations: 8.627 Mi
B)
VerletLeapfrog():           0.006256 seconds (180.11 k allocations: 8.780 Mi
B)
PseudoVerletLeapfrog():     0.022612 seconds (180.11 k allocations: 8.780 Mi
B, 71.11% gc time)
```

```
k = 0.999
Δt = 0.001
singlependulum(k, SymplecticEuler(), Δt)
```

```
SymplecticEuler():          0.263267 seconds (1.70 M allocations: 86.218 MiB
, 79.29% gc time)
PyObject Text(0.5, 0.98, '=====    SymplecticEuler(),    Δt = 0.001    =====
')
```

```
k = 0.999
Δt = 0.0001
singlependulum(k, SymplecticEuler(), Δt)
```

```
SymplecticEuler():          2.818312 seconds (17.00 M allocations: 862.127 M
iB, 71.08% gc time)
PyObject Text(0.5, 0.98, '=====    SymplecticEuler(),    Δt = 0.0001    ====
=')
```

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

2.3 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("DynamicalODE","single_pendulums.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 0.7.15
[bb2cbb15-79fc-5d1e-9bf1-8ae49c7c1650] DiffEqBenchmarks 0.1.0
[459566f4-90b8-5000-8ac3-15dfb0a30def] DiffEqCallbacks 2.5.2
[f3b72e0c-5b89-59e1-b016-84e28bfd966d] DiffEqDevTools 2.7.2
[055956cb-9e8b-5191-98cc-73ae4a59e68a] DiffEqPhysics 3.1.0
[0c46a032-eb83-5123-abaf-570d42b7fbaa] DifferentialEquations 6.3.0
[b305315f-e792-5b7a-8f41-49f472929428] Elliptic 0.5.0
[e5e0dc1b-0480-54bc-9374-aad01c23163d] Juno 0.5.5
[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 3.1.0
[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
[d330b81b-6aea-500a-939a-2ce795aea3ee] PyPlot 2.8.0
[90137ffa-7385-5640-81b9-e52037218182] StaticArrays 0.10.3
[c3572dad-4567-51f8-b174-8c6c989267f4] Sundials 3.2.0+
[92b13dbe-c966-51a2-8445-caca9f8a7d42] TaylorIntegration 0.4.1
[44d3d7a6-8a23-5bf8-98c5-b353f8df5ec9] Weave 0.9.0
```