Single Pedulum Comparison

Gen Kuroki (), Chris Rackauckas

March 13, 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=====")

    tight_layout()
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=====")

    tight_layout()
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():          1.476837 seconds (3.13 M allocations: 158.979 Mi
B)
```

```
# 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)
```

```
1.637005 seconds (3.21 M allocations: 162.555 MiB)
```

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

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():         1.304278 seconds (2.74 M allocations: 138.891 Mi
B)
VelocityVerlet():          0.001096 seconds (17.13 k allocations: 936.938 K
iB)
VerletLeapfrog():          1.351251 seconds (2.94 M allocations: 146.079 Mi
B)
PseudoVerletLeapfrog():    1.295570 seconds (2.91 M allocations: 144.273 Mi
B)
McAte2():                  1.293466 seconds (2.90 M allocations: 143.951 Mi
B)
Ruth3():                   1.945690 seconds (3.05 M allocations: 149.717 Mi
B, 28.17% gc time)
McAte3():                  1.358815 seconds (3.02 M allocations: 148.192 Mi
B)
CandyRoz4():               1.456774 seconds (3.16 M allocations: 153.556 Mi
B)
McAte4():                  1.533293 seconds (3.13 M allocations: 152.122 Mi
B)
CalvoSanz4():              1.435660 seconds (3.24 M allocations: 156.347 Mi
```

```
B)
McAte42():                 1.422057 seconds (3.24 M allocations: 156.004 Mi
B)
McAte5():                  1.501270 seconds (3.40 M allocations: 162.027 Mi
B)
Yoshida6():                1.555638 seconds (3.58 M allocations: 168.328 Mi
B)
KahanLi6():                1.651580 seconds (3.80 M allocations: 176.275 Mi
B)
McAte8():                  2.052037 seconds (4.49 M allocations: 200.127 Mi
B, 6.61% gc time)
KahanLi8():                2.154814 seconds (4.72 M allocations: 208.620 Mi
B, 6.61% gc time)
SofSpa10():                3.383498 seconds (6.79 M allocations: 280.508 Mi
B, 13.20% gc time)
```

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

```
SymplecticEuler():         0.005542 seconds (170.11 k allocations: 8.627 Mi
B)
VelocityVerlet():          0.006183 seconds (170.11 k allocations: 8.627 Mi
B)
VerletLeapfrog():          0.022814 seconds (180.11 k allocations: 8.780 Mi
B, 70.91% gc time)
PseudoVerletLeapfrog():    0.006402 seconds (180.11 k allocations: 8.780 Mi
B)
```

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

```
SymplecticEuler():         0.187144 seconds (1.70 M allocations: 86.218 MiB
, 67.83% gc time)
```

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

```
SymplecticEuler():         2.414798 seconds (17.00 M allocations: 862.127 M
iB, 64.81% gc time)
```