

# Single Pedulum Comparison

Gen Kuroki (), Chris Rackauckas

May 5, 2019

## 1 Table of Contents

[Solving single pendulums by DifferentialEquations.jl](#)  
Solving single pendulums by DifferentialEquations.jl  
[Tests](#)  
Tests  
[Comparison of symplectic Integrators](#)  
Comparison of symplectic Integrators

## 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:

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

Maximum of  $q(t)$ :

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

Define  $y(t)$  by

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

```
# 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 solution 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")
end
```

```

        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:")
    sol = solve(prob, integrator, dt=Δt)
    @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/types/dynamical_types.html#Hamiltonian-Problems-1)

[http://docs.juliadiffeq.org/latest/solvers/dynamical\\_solve.html](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():          0.000329 seconds (6.91 k allocations: 359.141 Ki
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

```

```

sol = solve(prob, integrator, dt= $\Delta t$ )
@time sol = solve(prob, integrator, dt= $\Delta t$ )

0.000760 seconds (10.13 k allocations: 685.313 KiB)

sleep(0.1)
plotsolenergy(H, integrator,  $\Delta t$ , sol)

PyObject Text(0.5, 0.98, '==== VelocityVerlet(),  $\Delta t$  = 0.1 =====')

```

## 2.2 Comparison of symplectic Integrators

[http://docs.juliadiffeq.org/latest/solvers/dynamical\\_solve.html#Symplectic-Integrators-1](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
 $\Delta t$  = 0.1
for integrator in SymplecticIntegrators
    singlependulum(k, integrator,  $\Delta t$ )
end

SymplecticEuler():      0.002193 seconds (17.13 k allocations: 936.922 KiB)
VelocityVerlet():      0.000579 seconds (17.13 k allocations: 936.938 KiB)
VerletLeapfrog():      0.000772 seconds (18.13 k allocations: 952.688 KiB)
PseudoVerletLeapfrog(): 0.000961 seconds (18.13 k allocations: 952.688 KiB)
McAte2():              0.001144 seconds (18.14 k allocations: 952.828 KiB)
Ruth3():               0.001171 seconds (20.14 k allocations: 984.188 KiB)
McAte3():              0.001237 seconds (20.14 k allocations: 984.109 KiB)
CandyRoz4():           0.001285 seconds (22.17 k allocations: 1015.906 KiB)
McAte4():              0.001261 seconds (22.13 k allocations: 1015.172 KiB)

```

```

CalvoSanz4():          0.001312 seconds (24.15 k allocations: 1.022 MiB
)
McAte42():             0.001061 seconds (24.14 k allocations: 1.022 MiB
)
McAte5():              0.001098 seconds (26.14 k allocations: 1.053 MiB
)
Yoshida6():            0.001249 seconds (30.16 k allocations: 1.114 MiB
)
KahanLi6():            0.001270 seconds (34.14 k allocations: 1.175 MiB
)
McAte8():              0.001742 seconds (46.16 k allocations: 1.358 MiB
)
KahanLi8():            0.001840 seconds (50.16 k allocations: 1.420 MiB
)
SofSpa10():            0.003786 seconds (86.20 k allocations: 1.970 MiB
)

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

SymplecticEuler():      0.006434 seconds (170.11 k allocations: 8.627 MiB)
VelocityVerlet():       0.009569 seconds (170.11 k allocations: 8.627 MiB)
VerletLeapfrog():       0.007053 seconds (180.11 k allocations: 8.780 MiB)
PseudoVerletLeapfrog(): 0.007425 seconds (180.11 k allocations: 8.780 MiB)

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

SymplecticEuler():      0.420196 seconds (1.70 M allocations: 86.218 MiB, 84.57% gc time)
PyObject Text(0.5, 0.98, '==== SymplecticEuler(), Δt = 0.001 ====')

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

SymplecticEuler():      1.522603 seconds (17.00 M allocations: 862.127 MiB, 43.66% 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/JuliaDiffeq/DiffEqBenchmarks.jl>

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 0.8.5
[bcd4f6db-9728-5f36-b5f7-82caef46ccdb] DelayDiffEq 5.2.0
[bb2cbb15-79fc-5d1e-9bf1-8ae49c7c1650] DiffEqBenchmarks 0.1.0
[459566f4-90b8-5000-8ac3-15dfb0a30def] DiffEqCallbacks 2.5.2
[f3b72e0c-5b89-59e1-b016-84e28bfd966d] DiffEqDevTools 2.8.0
[78ddff82-25fc-5f2b-89aa-309469cbf16f] DiffEqMonteCarlo 0.14.0
[77a26b50-5914-5dd7-bc55-306e6241c503] DiffEqNoiseProcess 3.2.0
[055956cb-9e8b-5191-98cc-73ae4a59e68a] DiffEqPhysics 3.1.0
[a077e3f3-b75c-5d7f-a0c6-6bc4c8ec64a9] DiffEqProblemLibrary 4.1.0
[41bf760c-e81c-5289-8e54-58b1f1f8abe2] DiffEqSensitivity 3.2.2
[0c46a032-eb83-5123-abaf-570d42b7fbba] DifferentialEquations 6.3.0
[b305315f-e792-5b7a-8f41-49f472929428] Elliptic 0.5.0
[e5e0dc1b-0480-54bc-9374-aad01c23163d] Juno 0.7.0
[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 3.2.0
[1dea7af3-3e70-54e6-95c3-0bf5283fa5ed] OrdinaryDiffEq 5.6.0
[2dcacdae-9679-587a-88bb-8b444fb7085b] ParallelDataTransfer 0.5.0
[65888b18-ceab-5e60-b2b9-181511a3b968] ParameterizedFunctions 4.1.1
[91a5bcd-d55d7-5caf-9e0b-520d859cae80] Plots 0.24.0
[d330b81b-6aea-500a-939a-2ce795aea3ee] PyPlot 2.8.1
[295af30f-e4ad-537b-8983-00126c2a3abe] Revise 2.1.4
[90137ffa-7385-5640-81b9-e52037218182] StaticArrays 0.10.3
[789caeaf-c7a9-5a7d-9973-96adeb23e2a0] StochasticDiffEq 6.2.0
[c3572dad-4567-51f8-b174-8c6c989267f4] Sundials 3.4.1
[92b13dbe-c966-51a2-8445-caca9f8a7d42] TaylorIntegration 0.4.1
[44d3d7a6-8a23-5bf8-98c5-b353f8df5ec9] Weave 0.9.0
[e88e6eb3-aa80-5325-afca-941959d7151f] Zygote 0.3.0
```