# Classical Physics Models

Yingbo Ma, Chris Rackauckas

February 23, 2019

If you're getting some cold feet to jump in to DiffEq land, here are some handcrafted differential equations mini problems to hold your hand along the beginning of your journey.

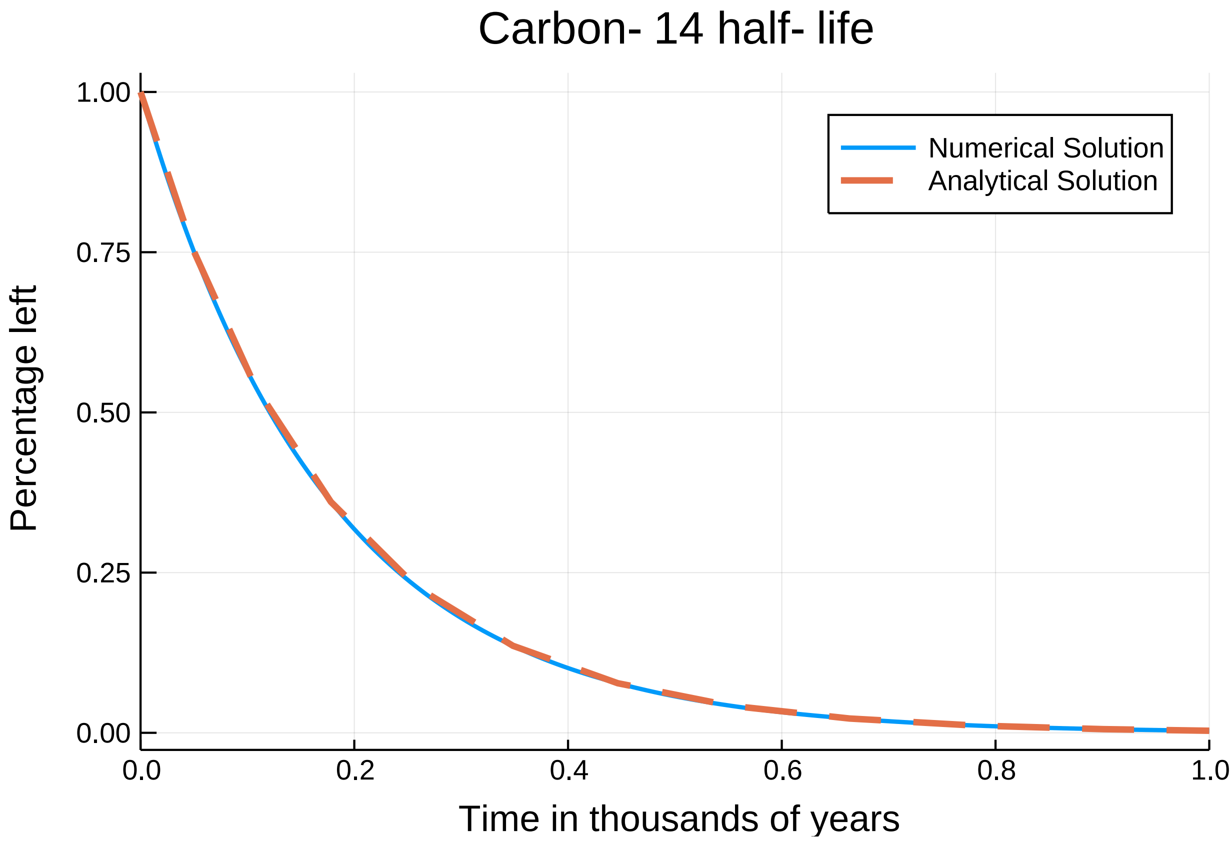
## 0.1 Radioactive Decay of Carbon-14

### First order linear ODE

$$f(t,u) = \frac{du}{dt} \tag{1}$$

The Radioactive decay problem is the first order linear ODE problem of an exponential with a negative coefficient, which represents the half-life of the process in question. Should the coefficient be positive, this would represent a population growth equation.

```
using OrdinaryDiffEq, Plots
gr()

#Half-life of Carbon-14 is 5,730 years.
C_1 = 5.730

#Setup
u_0 = 1.0
tspan = (0.0, 1.0)

#Define the problem
radioactivedecay(u,p,t) = -C_1*u

#Pass to solver
prob = ODEProblem(radioactivedecay,u_0,tspan)
sol = solve(prob,Tsit5())

#Plot
plot(sol,linewidth=2,title ="Carbon-14 half-life", xaxis = "Time in thousands of
    years", yaxis = "Percentage left", label = "Numerical Solution")
plot!(sol.t, t->exp(-C_1*t),lw=3,ls=:dash,label="Analytical Solution")
```




## 0.2 Simple Pendulum

**Second Order Linear ODE** We will start by solving the pendulum problem. In the physics class, we often solve this problem by small angle approximation, i.e. $ sin(\theta) \approx \theta$, because otherwise, we get an elliptic integral which doesn't have an analytic solution. The linearized form is

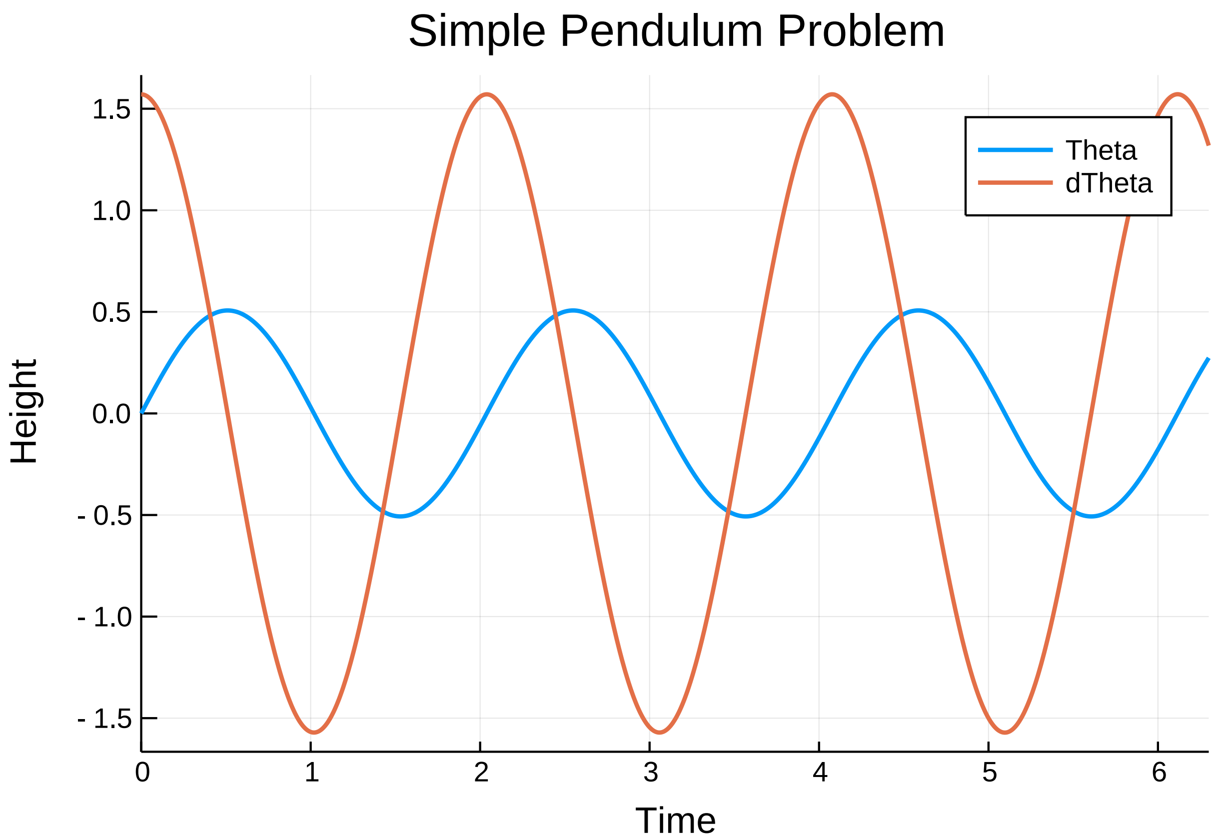
$$\ddot{\theta} + \frac{g}{L}\theta = 0 \tag{2}$$

But we have numerical ODE solvers! Why not solve the *real* pendulum?

$$\ddot{\theta} + \frac{g}{L}\sin(\theta) = 0 \tag{3}$$

```
# Simple Pendulum Problem
using OrdinaryDiffEq, Plots

#Constants
const g = 9.81
L = 1.0

#Initial Conditions
u_0 = [0,π/2]
tspan = (0.0,6.3)
```

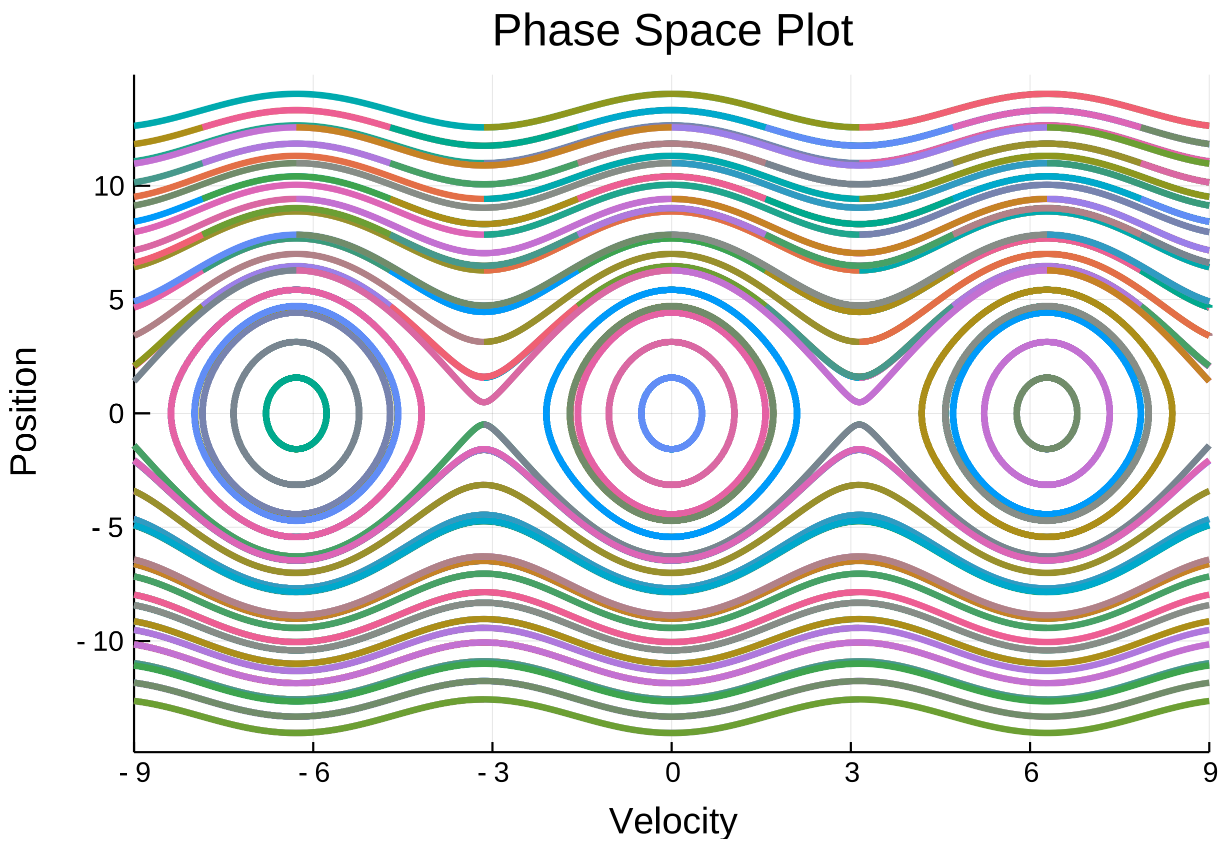
```julia
#Define the problem
function simplependulum(du,u,p,t)
    θ  = u[1]
    dθ = u[2]
    du[1] = dθ
    du[2] = -(g/L)*sin(θ)
end

#Pass to solvers
prob = ODEProblem(simplependulum,u_0, tspan)
sol = solve(prob,Tsit5())

#Plot
plot(sol,linewidth=2,title ="Simple Pendulum
    Problem", xaxis = "Time", yaxis = "Height", label = ["Theta","dTheta"])
```




So now we know that behaviour of the position versus time. However, it will be useful to us to look at the phase space of the pendulum, i.e., and representation of all possible states of the system in question (the pendulum) by looking at its velocity and position. Phase space analysis is ubiquitous in the analysis of dynamical systems, and thus we will provide a few facilities for it.

```julia
p = plot(sol,vars = (1,2), xlims = (-9,9), title = "Phase Space
    Plot", xaxis = "Velocity", yaxis = "Position", leg=false)
function phase_plot(prob, u0, p, tspan=2pi)
    _prob = ODEProblem(prob.f,u0,(0.0,tspan))
    sol = solve(_prob,Vern9()) # Use Vern9 solver for higher accuracy
    plot!(p,sol,vars = (1,2), xlims = nothing, ylims = nothing)
end
for i in -4pi:pi/2:4π
```

```
    for j in -4pi:pi/2:4π
        phase_plot(prob, [j,i], p)
    end
end
plot(p,xlims = (-9,9))
```

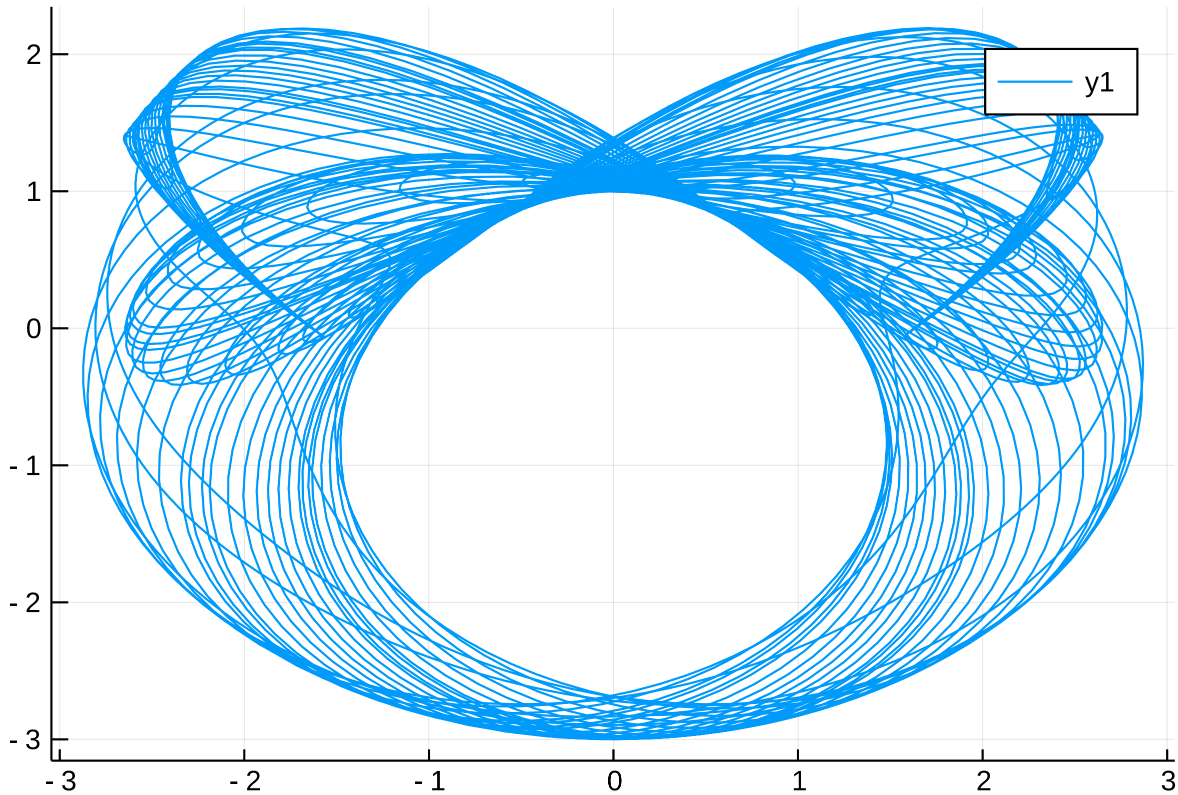



## 0.3 Simple Harmonic Oscillator

### 0.3.1 Double Pendulum

```
#Double Pendulum Problem
using OrdinaryDiffEq, Plots

#Constants and setup
const m_1, m_2, L_1, L_2 = 1, 2, 1, 2
initial = [0, π/3, 0, 3pi/5]
tspan = (0.,50.)

#Convenience function for transforming from polar to Cartesian coordinates
function polar2cart(sol;dt=0.02,l1=L_1,l2=L_2,vars=(2,4))
    u = sol.t[1]:dt:sol.t[end]

    p1 = l1*map(x->x[vars[1]], sol.(u))
    p2 = l2*map(y->y[vars[2]], sol.(u))

    x1 = l1*sin.(p1)
    y1 = l1*-cos.(p1)
    (u, (x1 + l2*sin.(p2),
     y1 - l2*cos.(p2)))
end
```

```
#Define the Problem
function double_pendulum(xdot,x,p,t)
    xdot[1]=x[2]

    xdot[2]=-((g*(2*m_1+m_2)*sin(x[1])+m_2*(g*sin(x[1]-2*x[3])+2*(L_2*x[4]^2+L_1*x[2]^2*cos(x[1]-x[3])
    xdot[3]=x[4]

    xdot[4]=(((m_1+m_2)*(L_1*x[2]^2+g*cos(x[1]))+L_2*m_2*x[4]^2*cos(x[1]-x[3]))*sin(x[1]-x[3]))/(L_2*(
end

#Pass to Solvers
double_pendulum_problem = ODEProblem(double_pendulum, initial, tspan)
sol = solve(double_pendulum_problem, Vern7(), abs_tol=1e-10, dt=0.05);


#Obtain coordinates in Cartesian Geometry
ts, ps = polar2cart(sol, l1=L_1, l2=L_2, dt=0.01)
plot(ps...)
```




### 0.3.2 Poincaré section

The Poincaré section is a contour plot of a higher-dimensional phase space diagram. It helps to understand the dynamic interactions and is wonderfully pretty.

The following equation came from StackOverflow question

$$\frac{d}{dt}\begin{pmatrix}\alpha\\ l_\alpha\\ \beta\\ l_\beta\end{pmatrix} = \begin{pmatrix} 2\frac{l_\alpha-(1+\cos\beta)l_\beta}{3-\cos 2\beta} \\ -2\sin\alpha - \sin(\alpha+\beta) \\ 2\frac{-(1+\cos\beta)l_\alpha+(3+2\cos\beta)l_\beta}{3-\cos 2\beta} \\ -\sin(\alpha+\beta) - 2\sin(\beta)\frac{(l_\alpha-l_\beta)l_\beta}{3-\cos 2\beta} + 2\sin(2\beta)\frac{l_\alpha^2-2(1+\cos\beta)l_\alpha l_\beta+(3+2\cos\beta)l_\beta^2}{(3-\cos 2\beta)^2}\end{pmatrix} \tag{4}$$

The Poincaré section here is the collection of $(\beta, l_\beta)$ when $\alpha = 0$ and $\frac{d\alpha}{dt} > 0$.

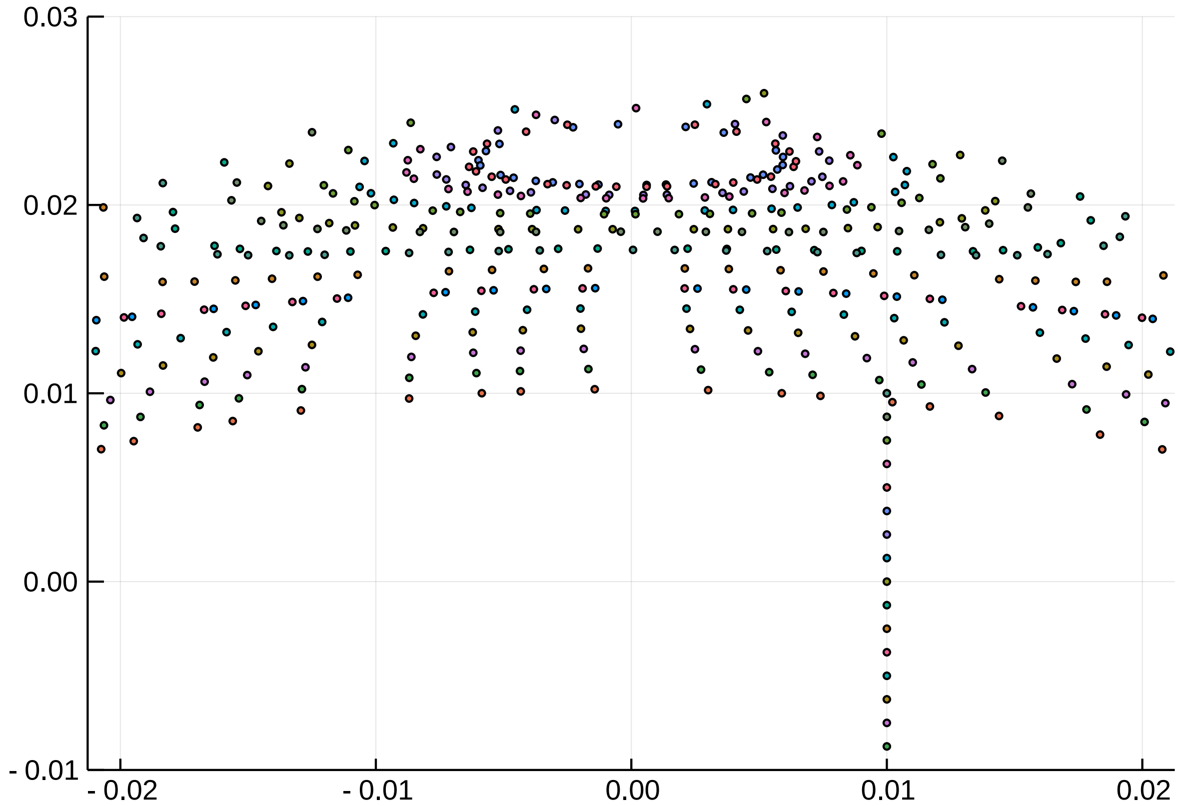
**Hamiltonian of a double pendulum** Now we will plot the Hamiltonian of a double pendulum

```
#Constants and setup
using OrdinaryDiffEq
initial2 = [0.01, 0.005, 0.01, 0.01]
tspan2 = (0.,200.)

#Define the problem
function double_pendulum_hamiltonian(udot,u,p,t)
    α  = u[1]
    lα = u[2]
    β  = u[3]
    lβ = u[4]
    udot .=
    [2(lα-(1+cos(β))lβ)/(3-cos(2β)),
    -2sin(α) - sin(α+β),
    2(-(1+cos(β))lα + (3+2cos(β))lβ)/(3-cos(2β)),
    -sin(α+β) - 2sin(β)*(((lα-lβ)lβ)/(3-cos(2β))) + 2sin(2β)*((lα^2 - 2(1+cos(β))lα*lβ
    + (3+2cos(β))lβ^2)/(3-cos(2β))^2)]
end

# Construct a ContiunousCallback
condition(u,t,integrator) = u[1]
affect!(integrator) = nothing
cb = ContinuousCallback(condition,affect!,nothing,
                        save_positions = (true,false))

# Construct Problem
poincare = ODEProblem(double_pendulum_hamiltonian, initial2, tspan2)
sol2 = solve(poincare, Vern9(), save_everystep = false, callback=cb, abstol=1e-9)

function poincare_map(prob, u_0, p; callback=cb)
    _prob = ODEProblem(prob.f,[0.01, 0.01, 0.01, u_0],prob.tspan)
    sol = solve(_prob, Vern9(), save_everystep = false, callback=cb, abstol=1e-9)
    scatter!(p, sol, vars=(3,4), markersize = 2)
end
```

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

```
p = scatter(sol2, vars=(3,4), leg=false, markersize = 2, ylims=(-0.01,0.03))
for i in -0.01:0.00125:0.01
    poincare_map(poincare, i, p)
```

```
end
plot(p,ylims=(-0.01,0.03))
```




## 0.4 Hénon-Heiles System

The Hénon-Heiles potential occurs when non-linear motion of a star around a galactic center with the motion restricted to a plane.

$$\frac{d^2x}{dt^2} = -\frac{\partial V}{\partial x} \tag{5}$$

$$\frac{d^2y}{dt^2} = -\frac{\partial V}{\partial y} \tag{6}$$

where

$$V(x,y) = \frac{1}{2}(x^2 + y^2) + \lambda\left(x^2 y - \frac{y^3}{3}\right). \tag{7}$$

We pick $\lambda = 1$ in this case, so

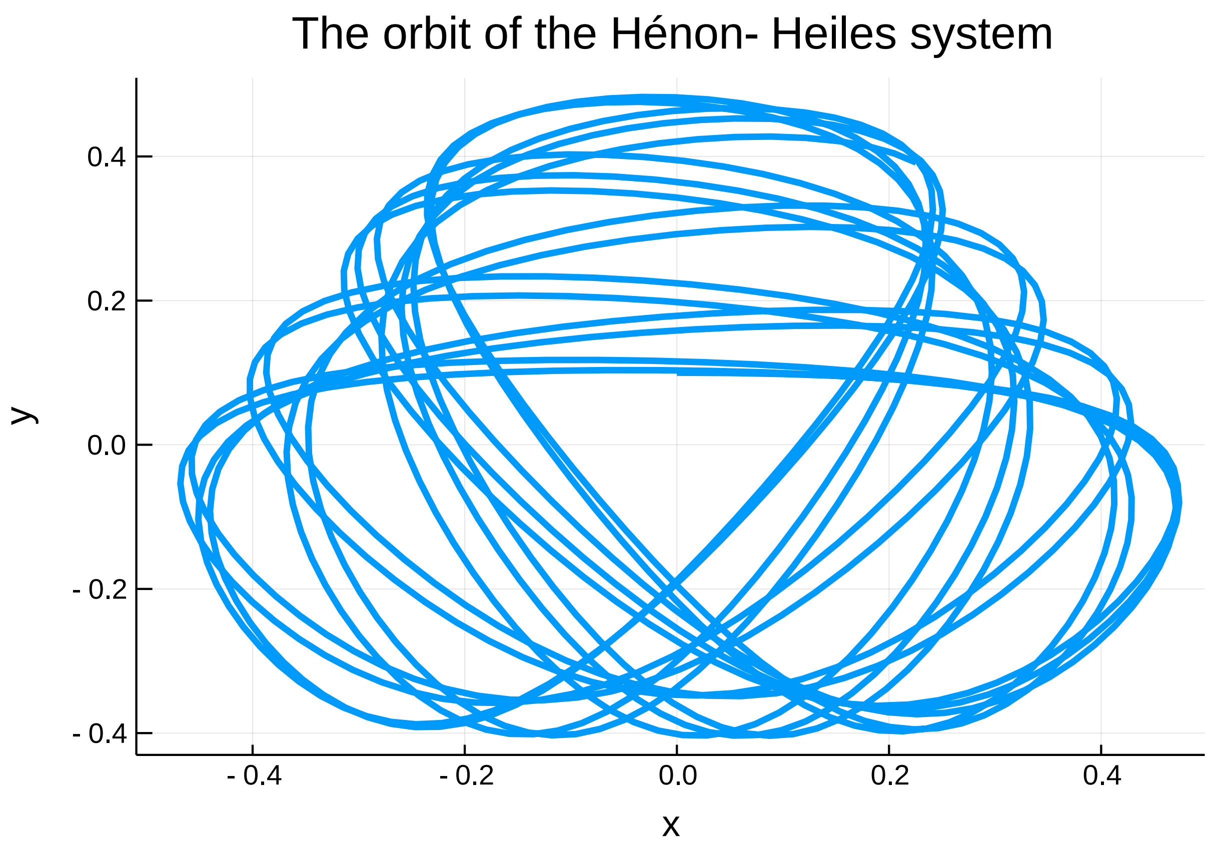
$$V(x,y) = \frac{1}{2}(x^2 + y^2 + 2x^2 y - \frac{2}{3}y^3). \tag{8}$$

Then the total energy of the system can be expressed by

$$E = T + V = V(x, y) + \frac{1}{2}(\dot{x}^2 + \dot{y}^2). \tag{9}$$

The total energy should conserve as this system evolves.

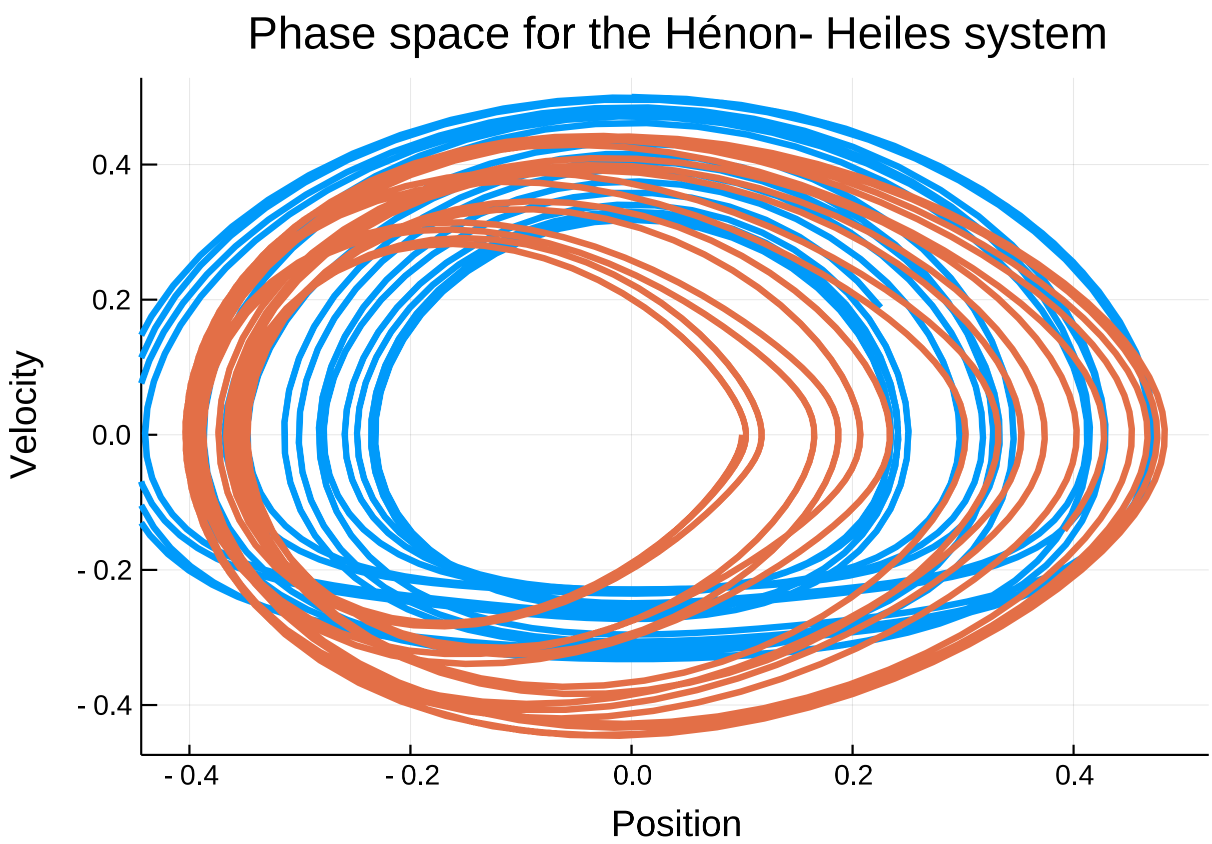
```
using OrdinaryDiffEq, Plots

#Setup
initial = [0.,0.1,0.5,0]
tspan = (0,100.)

#Remember, V is the potential of the system and T is the Total Kinetic Energy, thus E
    will
#the total energy of the system.
V(x,y) = 1//2 * (x^2 + y^2 + 2x^2*y - 2//3 * y^3)
E(x,y,dx,dy) = V(x,y) + 1//2 * (dx^2 + dy^2);

#Define the function
function Hénon_Heiles(du,u,p,t)
    x  = u[1]
    y  = u[2]
    dx = u[3]
    dy = u[4]
    du[1] = dx
    du[2] = dy
    du[3] = -x - 2x*y
    du[4] = y^2 - y -x^2
end

#Pass to solvers
prob = ODEProblem(Hénon_Heiles, initial, tspan)
sol = solve(prob, Vern9(), abs_tol=1e-16, rel_tol=1e-16);



# Plot the orbit
plot(sol, vars=(1,2), title = "The orbit of the Hénon-Heiles
    system", xaxis = "x", yaxis = "y", leg=false)
```




```
#Optional Sanity check - what do you think this returns and why?
@show sol.retcode
```

```
sol.retcode = :Success
```

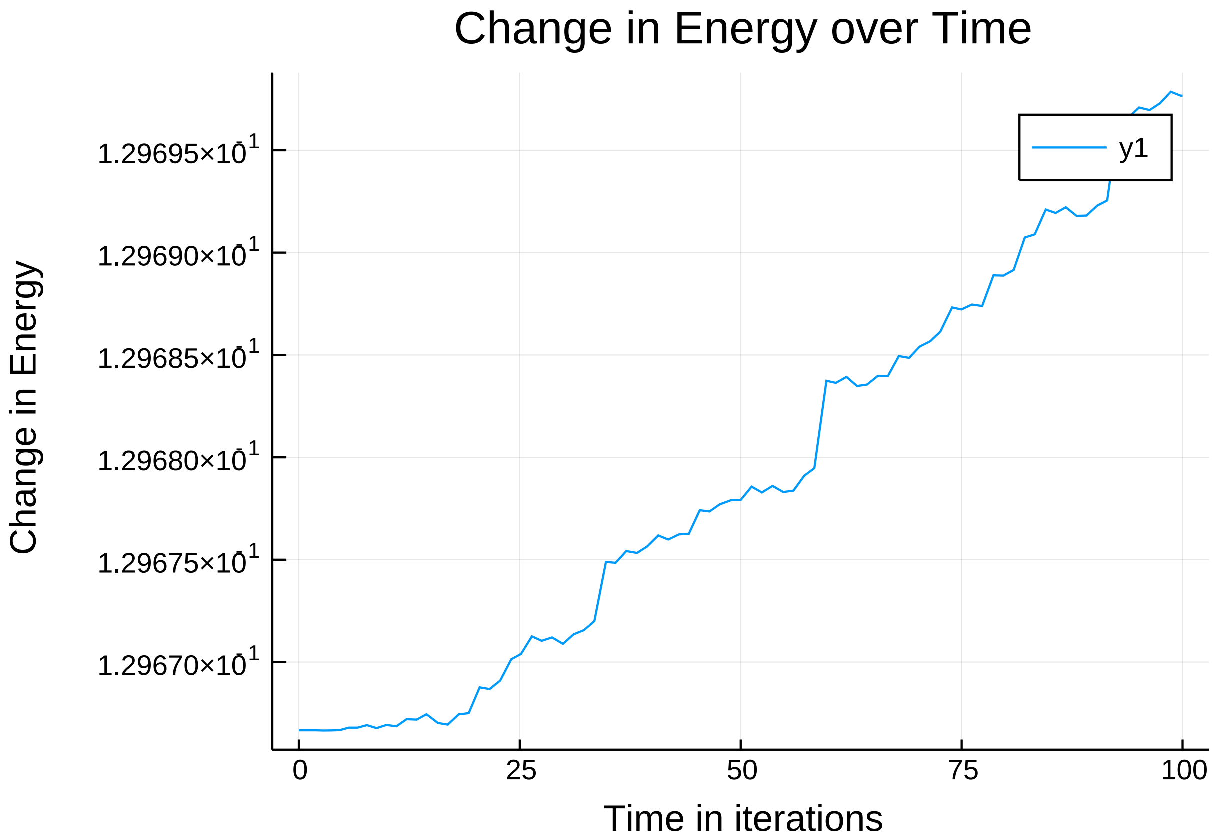
```
#Plot -
plot(sol, vars=(1,3), title = "Phase space for the Hénon-Heiles
    system", xaxis = "Position", yaxis = "Velocity")
plot!(sol, vars=(2,4), leg = false)
```




```
#We map the Total energies during the time intervals of the solution (sol.u here) to a
    new vector
#pass it to the plotter a bit more conveniently
energy = map(x->E(x...), sol.u)

#We use @show here to easily spot erratic behaviour in our system by seeing if the loss
    in energy was too great.
@show ΔE = energy[1]-energy[end]
```

```
ΔE = energy[1] - energy[end] = -3.099845153070602e-5
```

```
#Plot
plot(sol.t, energy, title = "Change in Energy over Time", xaxis = "Time in
    iterations", yaxis = "Change in Energy")
```



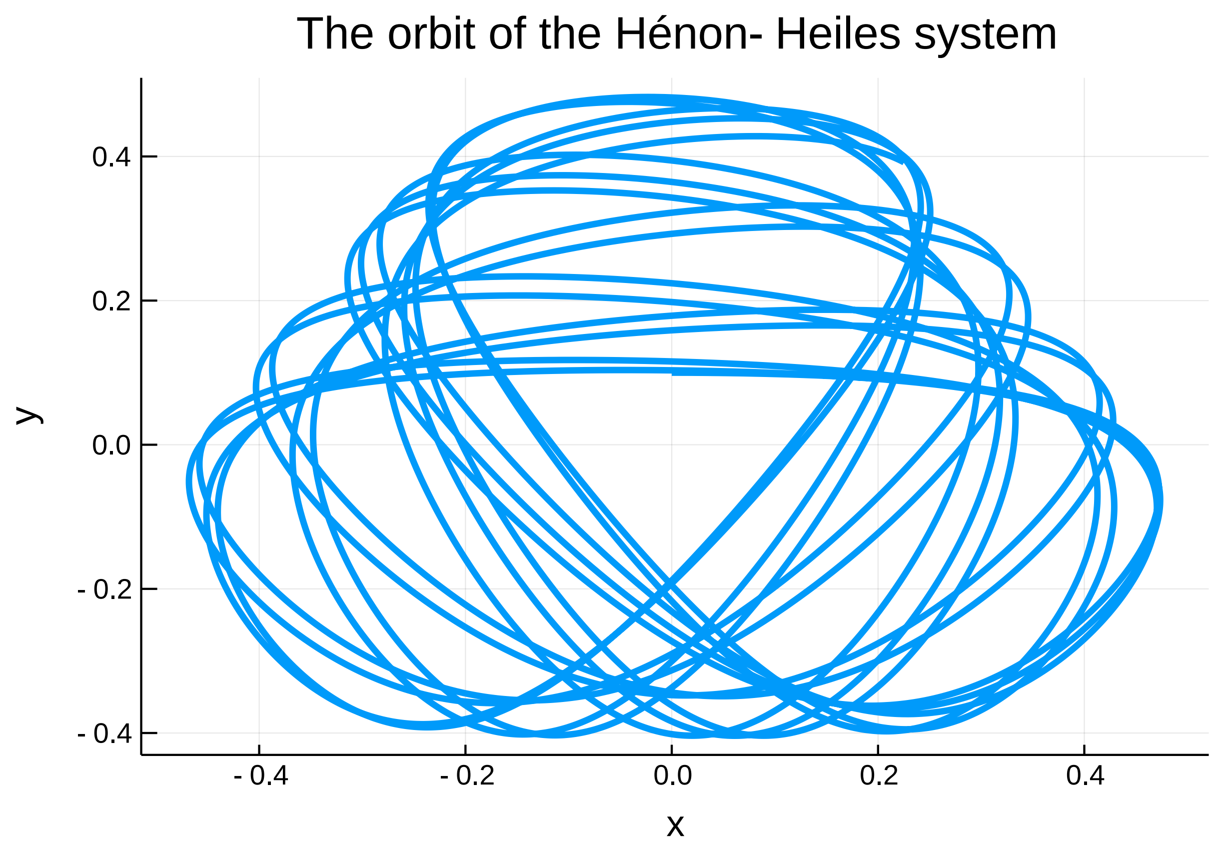

### 0.4.1 Symplectic Integration

To prevent energy drift, we can instead use a symplectic integrator. We can directly define and solve the `SecondOrderODEProblem`:

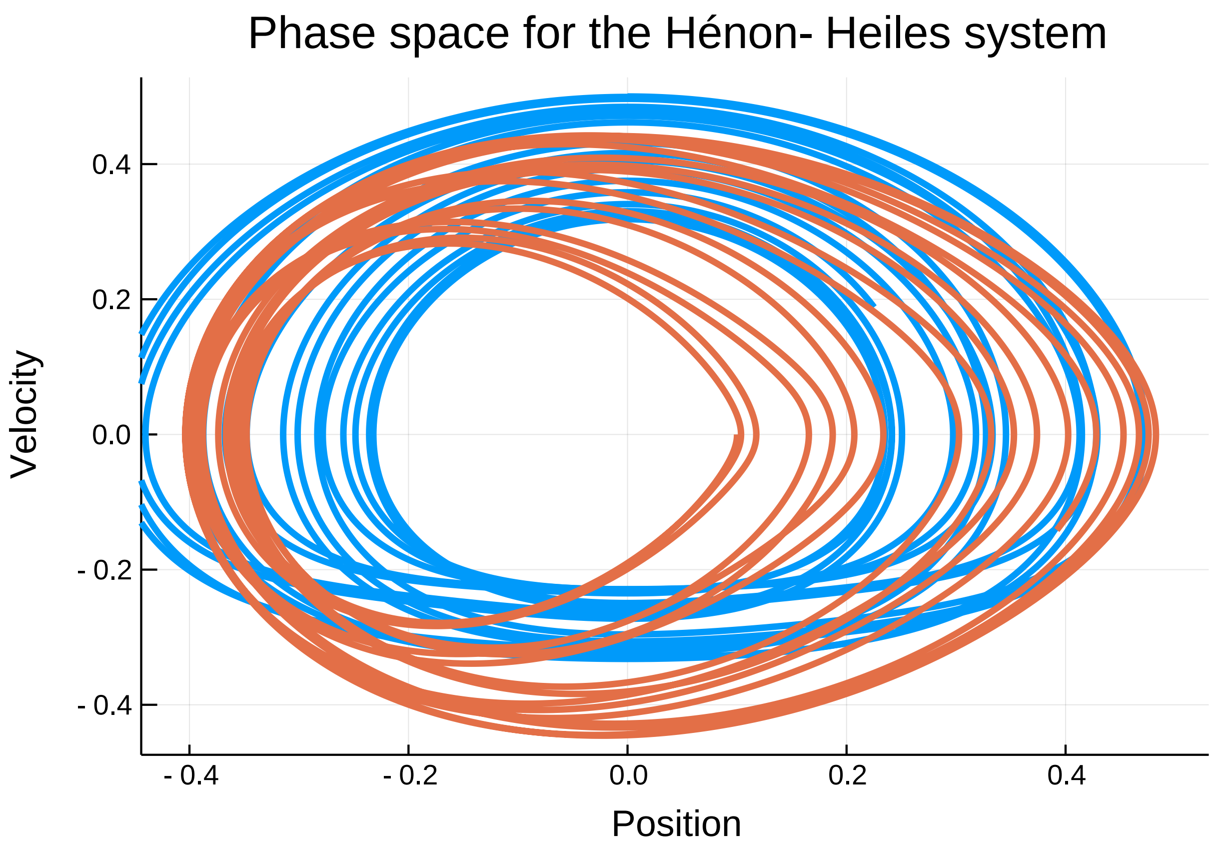
```
function HH_acceleration!(dv,v,u,p,t)
    x,y  = u
    dx,dy = dv
    dv[1] = -x - 2x*y
    dv[2] = y^2 - y -x^2
end
initial_positions = [0.0,0.1]
initial_velocities = [0.5,0.0]
prob = SecondOrderODEProblem(HH_acceleration!,initial_velocities,initial_positions,tspan)
sol2 = solve(prob, KahanLi8(), dt=1/10);
```

Notice that we get the same results:

```
# Plot the orbit
plot(sol2, vars=(3,4), title = "The orbit of the Hénon-Heiles
    system", xaxis = "x", yaxis = "y", leg=false)
```




```
plot(sol2, vars=(3,1), title = "Phase space for the Hénon-Heiles
    system", xaxis = "Position", yaxis = "Velocity")
plot!(sol2, vars=(4,2), leg = false)
```

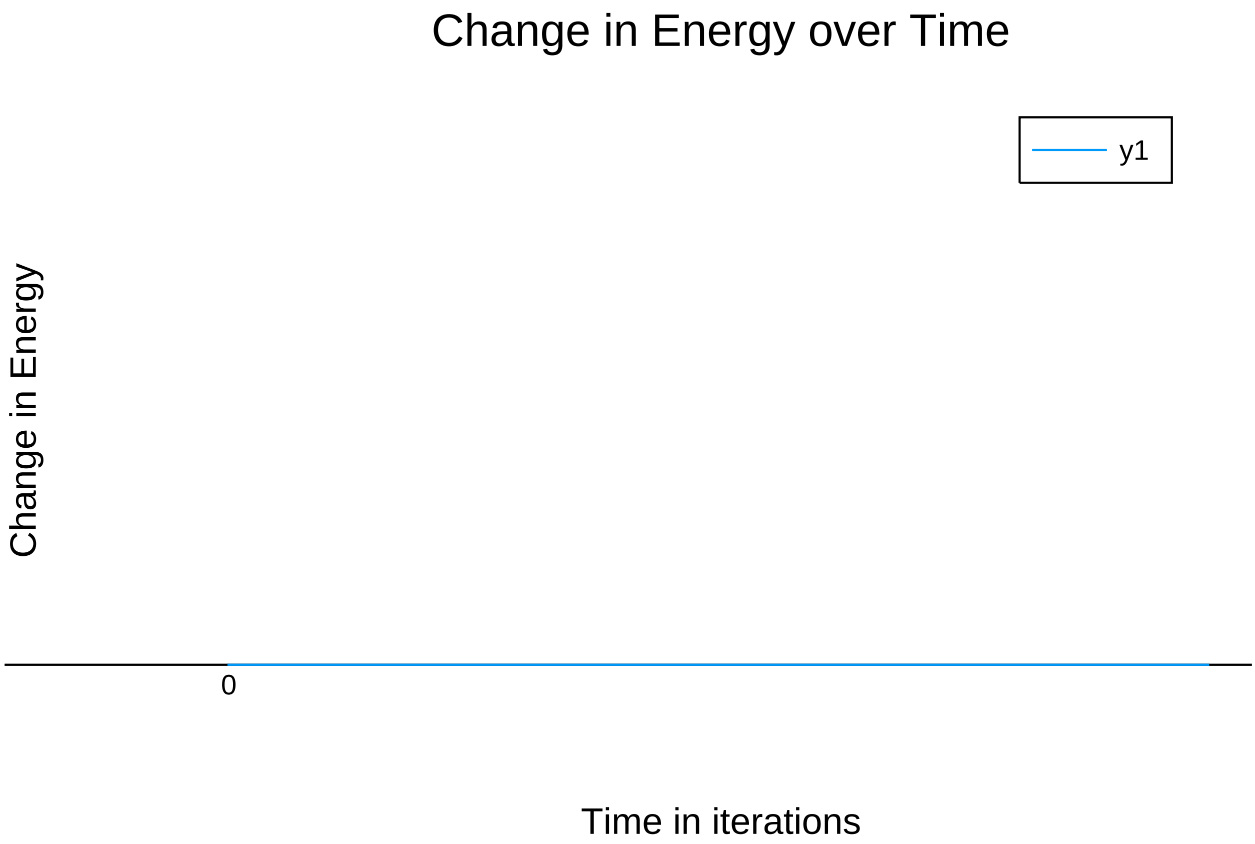


but now the energy change is essentially zero:

```
energy = map(x->E(x[3], x[4], x[1], x[2]), sol2.u)
#We use @show here to easily spot erratic behaviour in our system by seeing if the loss
    in energy was too great.
@show ΔE = energy[1]-energy[end]
```

```
ΔE = energy[1] - energy[end] = 9.020562075079397e-15
```

```
#Plot
plot(sol2.t, energy, title = "Change in Energy over Time", xaxis = "Time in
    iterations", yaxis = "Change in Energy")
```



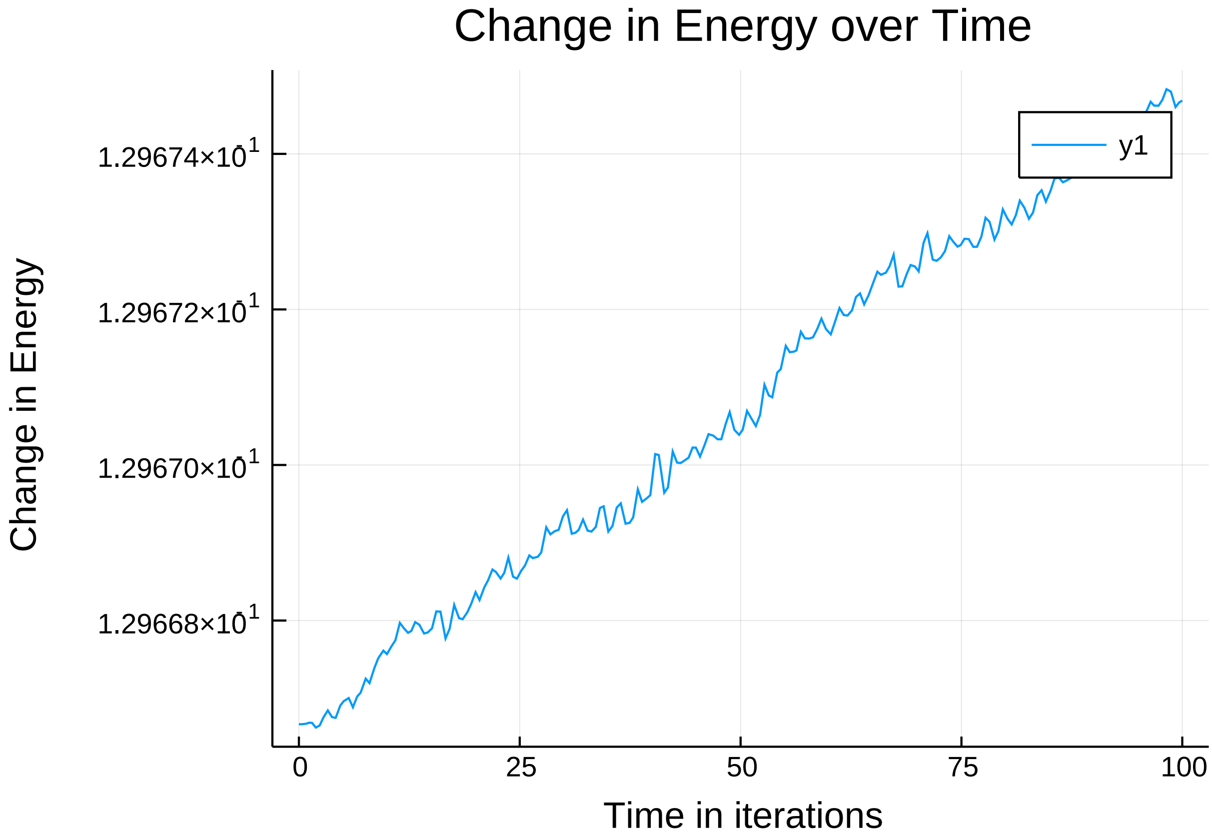

It's so close to zero it breaks GR! And let's try to use a Runge-Kutta-Nyström solver to solve this. Note that Runge-Kutta-Nyström isn't symplectic.

```
sol3 = solve(prob, DPRKN6());
energy = map(x->E(x[3], x[4], x[1], x[2]), sol3.u)
@show ΔE = energy[1]-energy[end]
```

```
ΔE = energy[1] - energy[end] = -8.017994408110463e-6
```

```
gr()
```

```
plot(sol3.t, energy, title = "Change in Energy over Time", xaxis = "Time in
    iterations", yaxis = "Change in Energy")
```




Note that we are using the `DPRKN6` sovler at `reltol=1e-3` (the default), yet it has a smaller energy variation than `Vern9` at `abs_tol=1e-16, rel_tol=1e-16`. Therefore, using specialized solvers to solve its particular problem is very efficient.

## 0.5 Appendix

```
using DiffEqTutorials
DiffEqTutorials.tutorial_footer(WEAVE_ARGS[:folder],WEAVE_ARGS[:file])
```

```
These benchmarks are part of the DiffEqTutorials.jl repository, found at:

https://github.com/JuliaDiffEq/DiffEqTutorials.jl

To locally run this tutorial, do the following commands:

using DiffEqTutorials
DiffEqTutorials.weave_file(".","models/classical_physics.jmd")


Computer Information:

Julia Version 1.1.0
Commit 80516ca202 (2019-01-21 21:24 UTC)
Platform Info:
  OS: Windows (x86_64-w64-mingw32)
```

```
  CPU: Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz
  WORD_SIZE: 64
  LIBM: libopenlibm
  LLVM: libLLVM-6.0.1 (ORCJIT, skylake)
Environment:
  JULIA_EDITOR = "C:\Users\accou\AppData\Local\atom\app-1.34.0\atom.exe" -a
  JULIA_NUM_THREADS = 6

Package Information:

    Status `C:\Users\accou\.julia\environments\v1.1\Project.toml`
  [c52e3926] Atom v0.7.14
  [6e4b80f9] BenchmarkTools v0.4.2
  [336ed68f] CSV v0.4.3
  [be33ccc6] CUDAnative v1.0.1
  [3a865a2d] CuArrays v0.9.1
  [a93c6f00] DataFrames v0.17.1
  [39dd38d3] Dierckx v0.4.1
  [aae7a2af] DiffEqFlux v0.2.0
  [c894b116] DiffEqJump v6.1.0+ [`C:\Users\accou\.julia\dev\DiffEqJump`]
  [1130ab10] DiffEqParamEstim v1.5.1
  [225cb15b] DiffEqTutorials v0.0.0 [`C:\Users\accou\.julia\external\DiffEq
Tutorials.jl`]
  [0c46a032] DifferentialEquations v6.3.0
  [587475ba] Flux v0.7.3
  [f6369f11] ForwardDiff v0.10.3+ [`C:\Users\accou\.julia\dev\ForwardDiff`]
  [7073ff75] IJulia v1.17.0
  [c601a237] Interact v0.9.1
  [b6b21f68] Ipopt v0.5.4
  [4076af6c] JuMP v0.18.5
  [e5e0dc1b] Juno v0.5.4
  [76087f3c] NLopt v0.5.1
  [429524aa] Optim v0.17.2
  [1dea7af3] OrdinaryDiffEq v5.1.4+ [`C:\Users\accou\.julia\dev\OrdinaryDif
fEq`]
  [65888b18] ParameterizedFunctions v4.1.0
  [91a5bcdd] Plots v0.23.0
  [71ad9d73] PuMaS v0.0.0 [`C:\Users\accou\.julia\dev\PuMaS`]
  [d330b81b] PyPlot v2.7.0
  [731186ca] RecursiveArrayTools v0.20.0
  [90137ffa] StaticArrays v0.10.2
  [789caeaf] StochasticDiffEq v6.1.1+ [`C:\Users\accou\.julia\dev\Stochasti
cDiffEq`]
  [c3572dad] Sundials v3.0.0
  [44d3d7a6] Weave v0.7.1
```