Kepler Problem

Yingbo Ma, Chris Rackauckas

February 25, 2019

The Hamiltonian $\mathcal{H}$ and the angular momentum L for the Kepler problem are

$$\mathcal{H} = \frac{1}{2}(\dot{q}_1^2 + \dot{q}_2^2) - \frac{1}{\sqrt{q_1^2 + q_2^2}}, \quad L = q_1\dot{q}_2 - \dot{q}_1 q_2 \tag{1}$$

Also, we know that

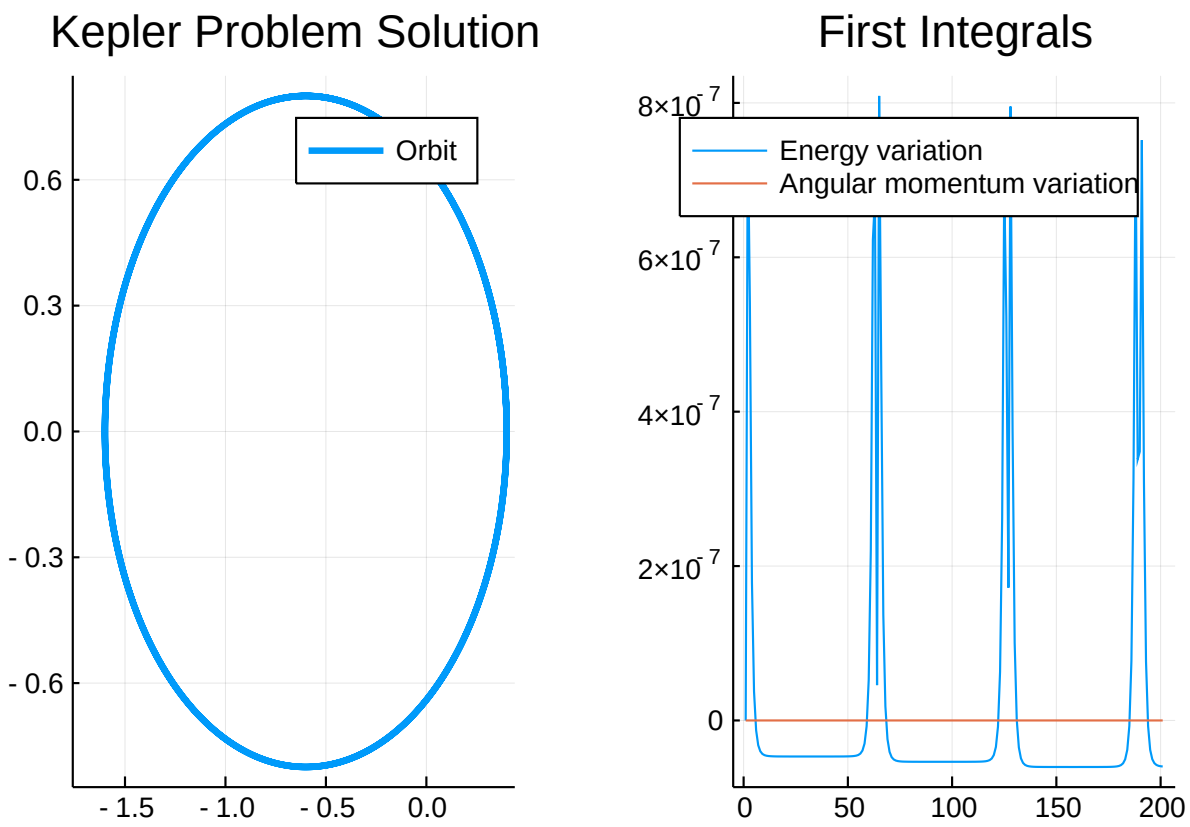
$$\frac{\mathrm{d}\boldsymbol{p}}{\mathrm{d}t} = -\frac{\partial \mathcal{H}}{\partial \boldsymbol{q}} \quad , \quad \frac{\mathrm{d}\boldsymbol{q}}{\mathrm{d}t} = +\frac{\partial \mathcal{H}}{\partial \boldsymbol{p}} \tag{2}$$

```
using OrdinaryDiffEq, LinearAlgebra, ForwardDiff, Plots; gr()
H(q,p) = norm(p)^2/2 - inv(norm(q))
L(q,p) = q[1]*p[2] - p[1]*q[2]

pdot(dp,p,q,params,t) = ForwardDiff.gradient!(dp, q->-H(q, p), q)
qdot(dq,p,q,params,t) = ForwardDiff.gradient!(dq, p-> H(q, p), p)

initial_position = [.4, 0]
initial_velocity = [0., 2.]
initial_cond = (initial_position, initial_velocity)
initial_first_integrals = (H(initial_cond...), L(initial_cond...))
tspan = (0,20.)
prob = DynamicalODEProblem(pdot, qdot, initial_velocity, initial_position, tspan)
sol = solve(prob, KahanLi6(), dt=1//10);
```

Let's plot the orbit and check the energy and angular momentum variation. We know that energy and angular momentum should be constant, and they are also called first integrals.

```
plot_orbit(sol) = plot(sol,vars=(3,4), lab="Orbit", title="Kepler Problem Solution")

function plot_first_integrals(sol, H, L)
    plot(initial_first_integrals[1].-map(u->H(u[2,:], u[1,:]), sol.u), lab="Energy
    variation", title="First Integrals")
    plot!(initial_first_integrals[2].-map(u->L(u[2,:], u[1,:]), sol.u), lab="Angular
    momentum variation")
end
analysis_plot(sol, H, L) = plot(plot_orbit(sol), plot_first_integrals(sol, H, L))
```

```
analysis_plot(sol, H, L)
```

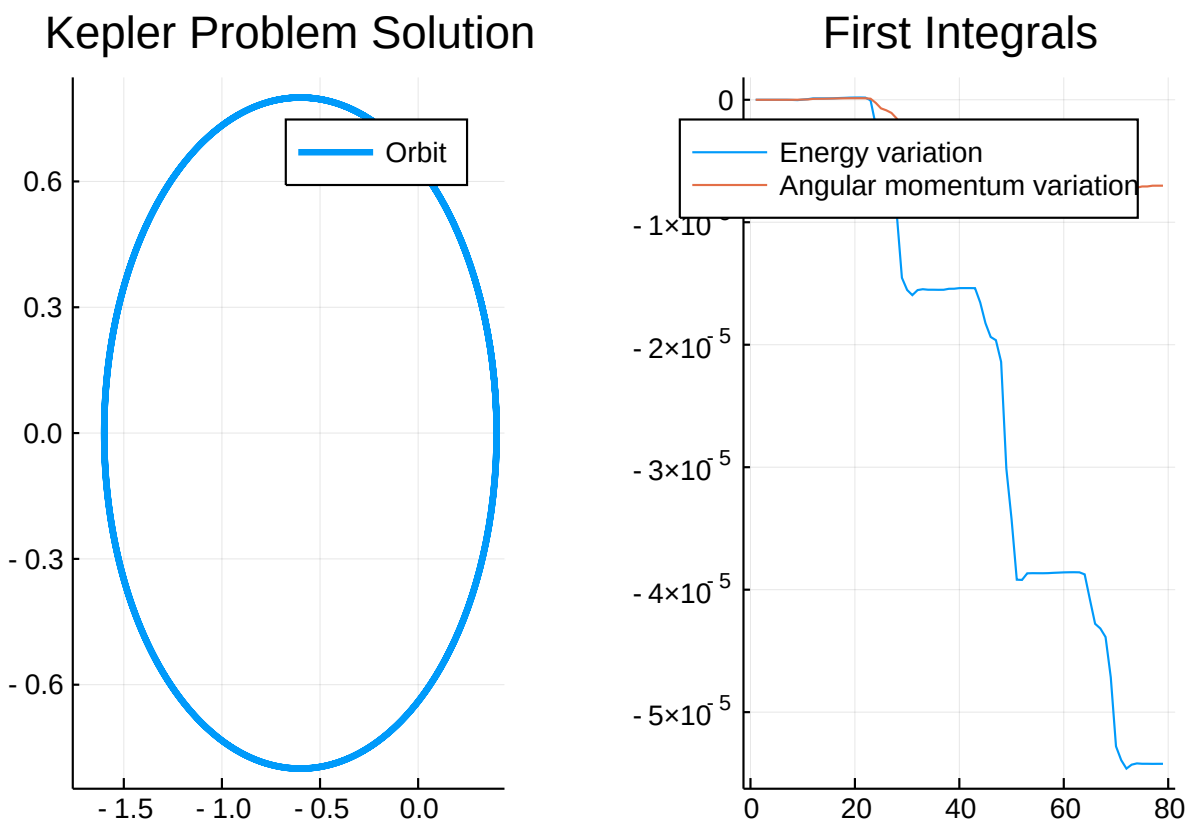


Let's try to use a Runge-Kutta-Nyström solver to solve this problem and check the first integrals' variation.

```
sol2 = solve(prob, DPRKN6())  # dt is not necessary, because unlike symplectic
                              # integrators DPRKN6 is adaptive
@show sol2.u |> length
```

```
sol2.u |> length = 79
```

```
analysis_plot(sol2, H, L)
```

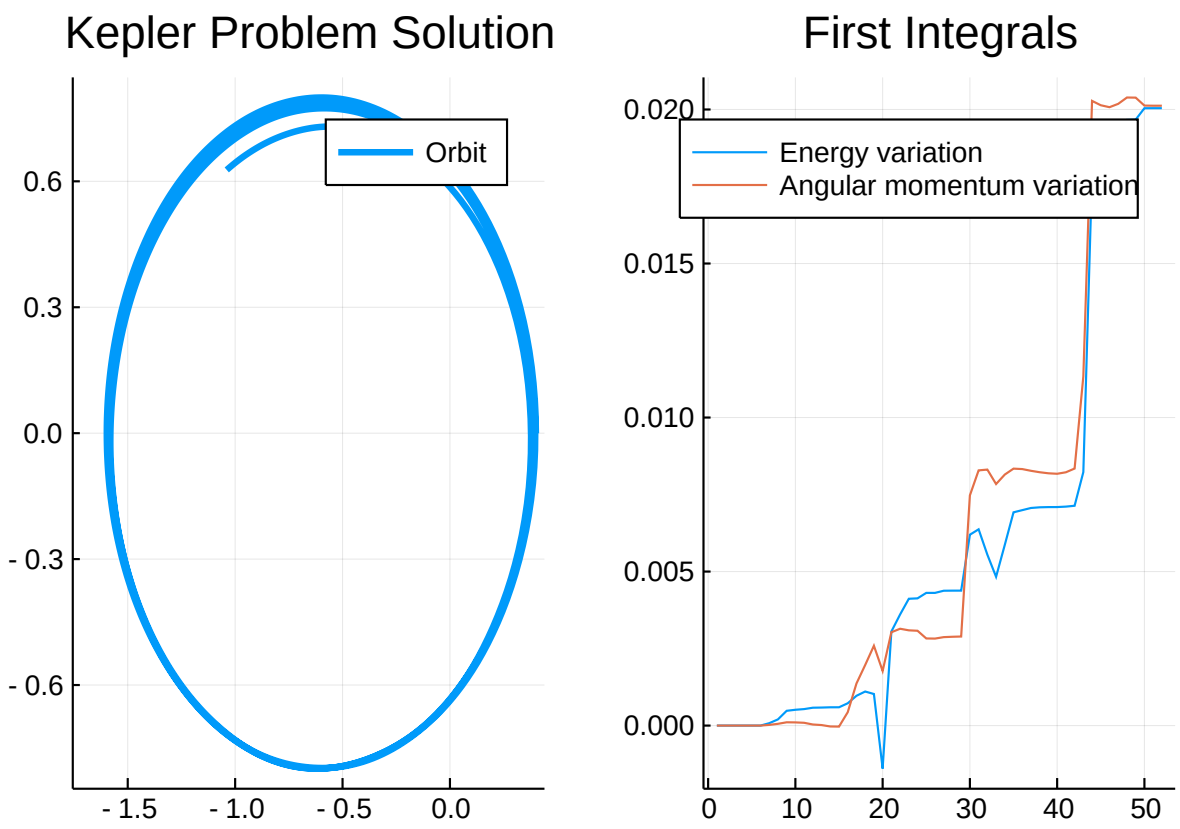


Let's then try to solve the same problem by the `ERKN4` solver, which is specialized for sinusoid-like periodic function

```
sol3 = solve(prob, ERKN4()) # dt is not necessary, because unlike symplectic
                            # integrators ERKN4 is adaptive
@show sol3.u |> length
```

```
sol3.u |> length = 52
```

```
analysis_plot(sol3, H, L)
```



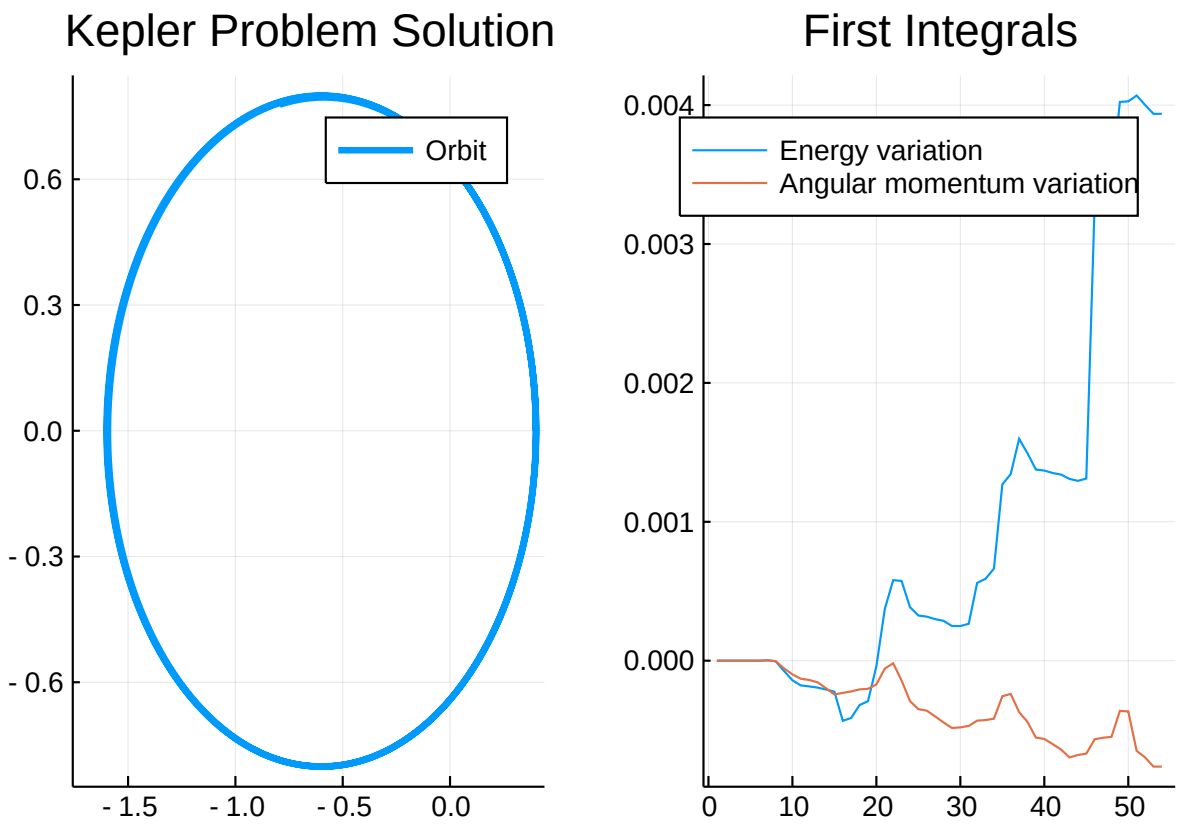
We can see that `ERKN4` does a bad job for this problem, because this problem is not sinusoid-like.

One advantage of using `DynamicalODEProblem` is that it can implicitly convert the second order ODE problem to a *normal* system of first order ODEs, which is solvable for other ODE solvers. Let's use the `Tsit5` solver for the next example.

```
sol4 = solve(prob, Tsit5())
@show sol4.u |> length
```

```
sol4.u |> length = 54
```

```
analysis_plot(sol4, H, L)
```



Note There is drifting for all the solutions, and high order methods are drifting less because they are more accurate.

0.0.1 Conclusion

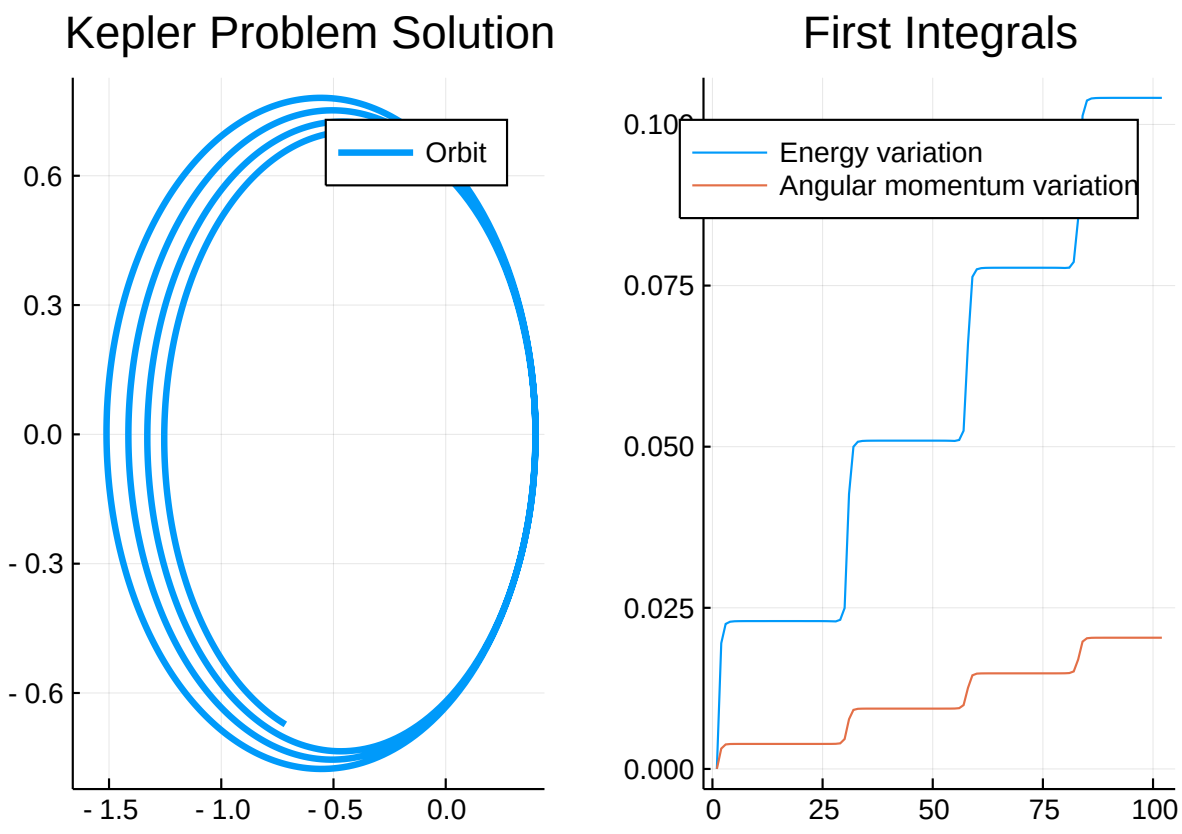
Symplectic integrator does not conserve the energy completely at all time, but the energy can come back. In order to make sure that the energy fluctuation comes back eventually, symplectic integrator has to have a fixed time step. Despite the energy variation, symplectic integrator conserves the angular momentum perfectly.

Both Runge-Kutta-Nyström and Runge-Kutta integrator do not conserve energy nor the angular momentum, and the first integrals do not tend to come back. An advantage Runge-Kutta-Nyström integrator over symplectic integrator is that RKN integrator can have adaptivity. An advantage Runge-Kutta-Nyström integrator over Runge-Kutta integrator is that RKN integrator has less function evaluation per step. The `ERKN4` solver works best for sinusoid-like solutions.

0.1 Manifold Projection

In this example, we know that energy and angular momentum should be conserved. We can achieve this through mainfold projection. As the name implies, it is a procedure to project the ODE solution to a manifold. Let's start with a base case, where mainfold projection isn't being used.

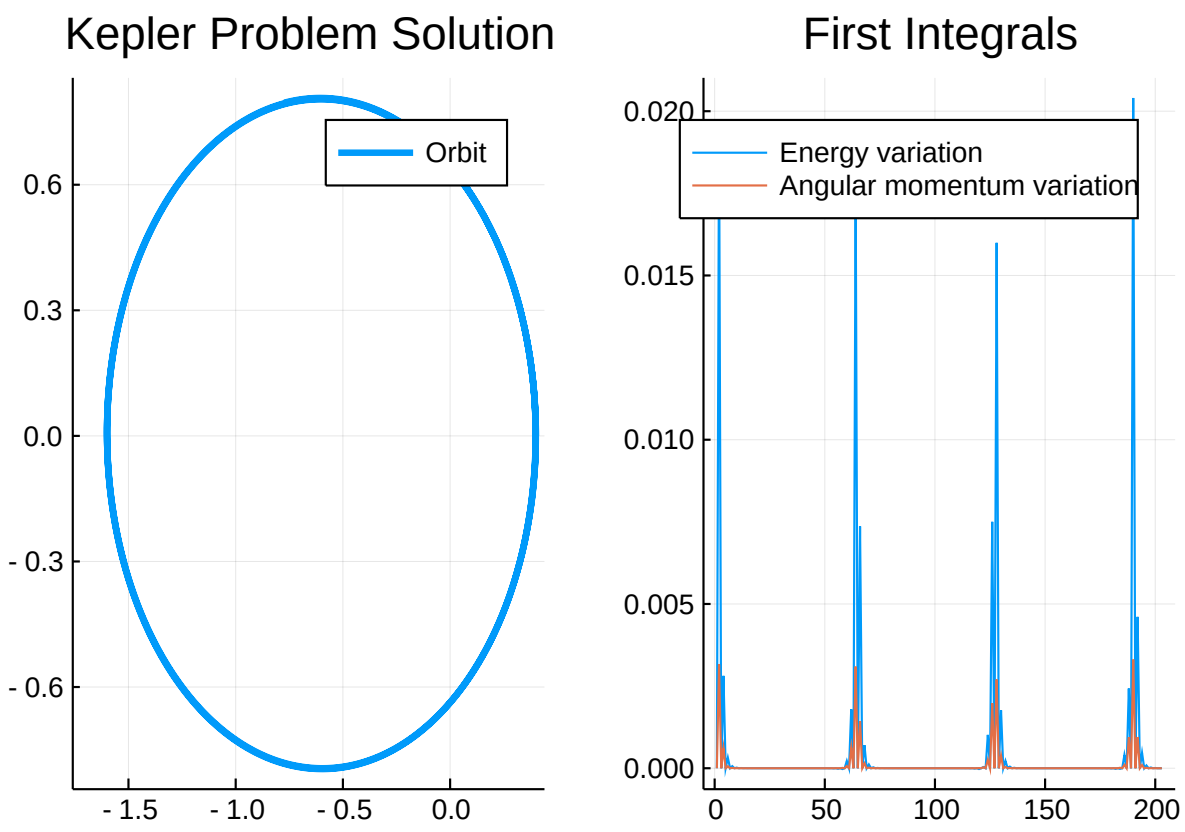
```
using DiffEqCallbacks

plot_orbit2(sol) = plot(sol,vars=(1,2), lab="Orbit", title="Kepler Problem Solution")

function plot_first_integrals2(sol, H, L)
    plot(initial_first_integrals[1].-map(u->H(u[1:2],u[3:4]), sol.u), lab="Energy
    variation", title="First Integrals")
    plot!(initial_first_integrals[2].-map(u->L(u[1:2],u[3:4]), sol.u), lab="Angular
    momentum variation")
end

analysis_plot2(sol, H, L) = plot(plot_orbit2(sol), plot_first_integrals2(sol, H, L))

function hamiltonian(du,u,params,t)
    q, p = u[1:2], u[3:4]
    qdot(@view(du[1:2]), p, q, params, t)
    pdot(@view(du[3:4]), p, q, params, t)
end

prob2 = ODEProblem(hamiltonian, [initial_position; initial_velocity], tspan)
sol_ = solve(prob2, RK4(), dt=1//5, adaptive=false)
analysis_plot2(sol_, H, L)
```



There is a significant fluctuation in the first integrals, when there is no mainfold projection.

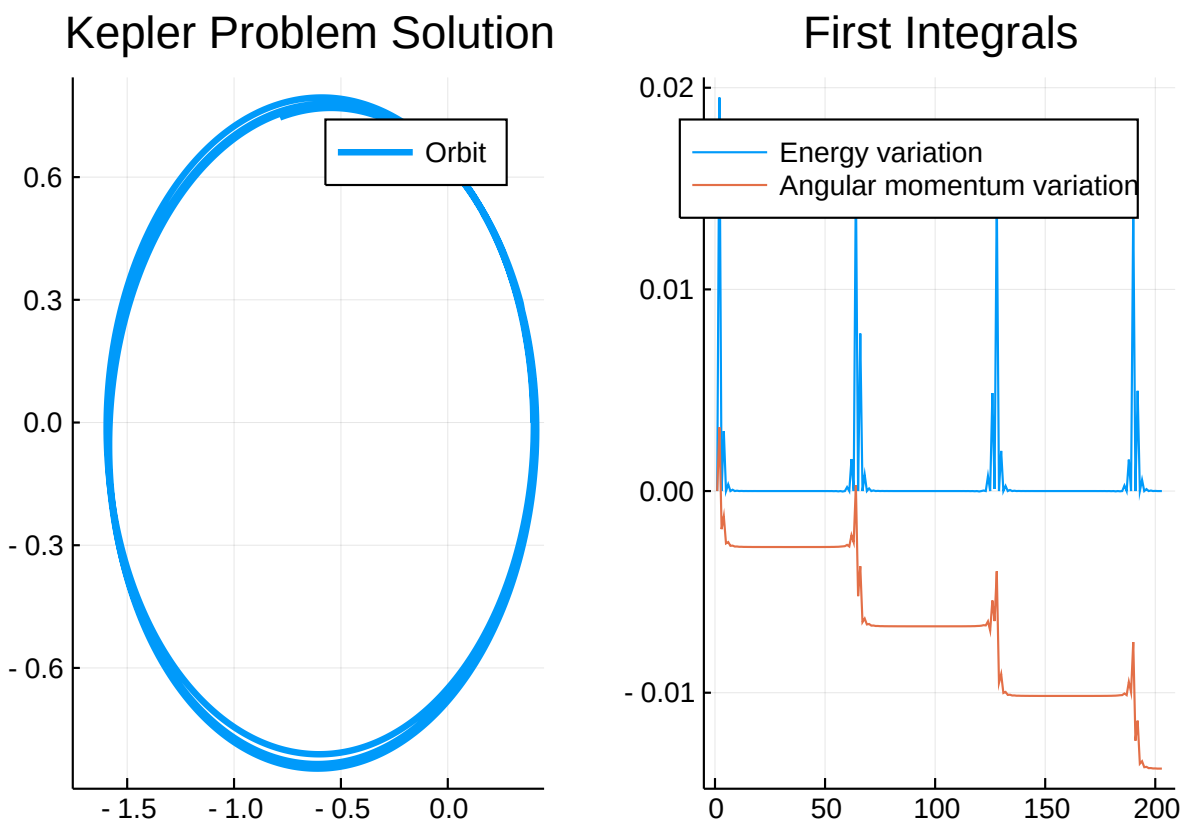
```
function first_integrals_manifold(residual,u)
    residual[1:2] .= initial_first_integrals[1] - H(u[1:2], u[3:4])
    residual[3:4] .= initial_first_integrals[2] - L(u[1:2], u[3:4])
end

cb = ManifoldProjection(first_integrals_manifold)
```

```
sol5 = solve(prob2, RK4(), dt=1//5, adaptive=false, callback=cb)
analysis_plot2(sol5, H, L)
```



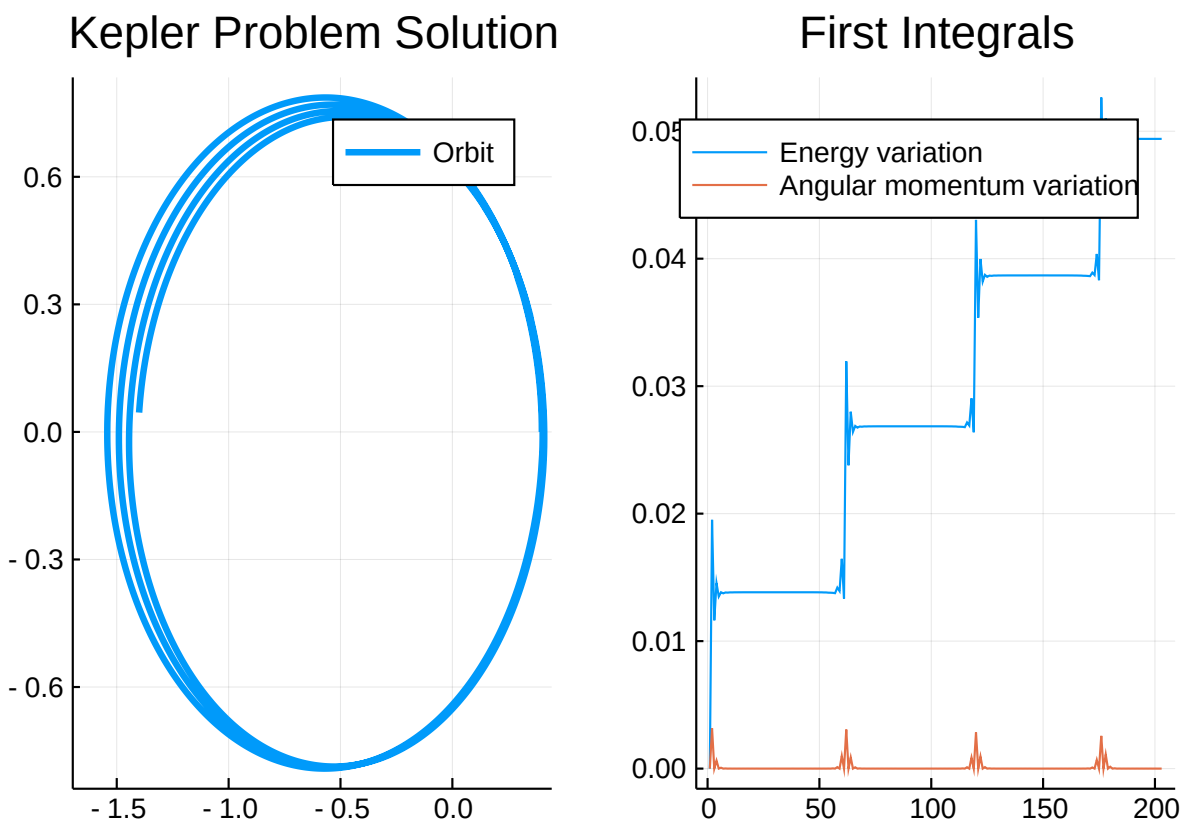
We can see that thanks to the manifold projection, the first integrals' variation is very small, although we are using `RK4` which is not symplectic. But wait, what if we only project to the energy conservation manifold?

```
function energy_manifold(residual,u)
    residual[1:2] .= initial_first_integrals[1] - H(u[1:2], u[3:4])
    residual[3:4] .= 0
end
energy_cb = ManifoldProjection(energy_manifold)
sol6 = solve(prob2, RK4(), dt=1//5, adaptive=false, callback=energy_cb)
analysis_plot2(sol6, H, L)
```



There is almost no energy variation but angular momentum varies quite bit. How about only project to the angular momentum conservation manifold?

```
function angular_manifold(residual,u)
    residual[1:2] .= initial_first_integrals[2] - L(u[1:2], u[3:4])
    residual[3:4] .= 0
end
angular_cb = ManifoldProjection(angular_manifold)
sol7 = solve(prob2, RK4(), dt=1//5, adaptive=false, callback=angular_cb)
analysis_plot2(sol7, H, L)
```



Again, we see what we expect.

0.2 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","kepler_problem.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`
  [7e558dbc] ArbNumerics v0.3.6
  [c52e3926] Atom v0.7.14
  [6e4b80f9] BenchmarkTools v0.4.2
  [336ed68f] CSV v0.4.3
  [3895d2a7] CUDAapi v0.5.4
  [be33ccc6] CUDAnative v1.0.1
  [3a865a2d] CuArrays v0.9.1
  [a93c6f00] DataFrames v0.17.1
  [55939f99] DecFP v0.4.8
  [abce61dc] Decimals v0.4.0
  [39dd38d3] Dierckx v0.4.1
  [459566f4] DiffEqCallbacks v2.5.2
  [f3b72e0c] DiffEqDevTools v2.6.1
  [aae7a2af] DiffEqFlux v0.2.0
  [c894b116] DiffEqJump v6.1.0+ [`C:\Users\accou\.julia\dev\DiffEqJump`]
  [1130ab10] DiffEqParamEstim v1.6.0+ [`C:\Users\accou\.julia\dev\DiffEqPar
amEstim`]
  [055956cb] DiffEqPhysics v3.1.0
  [225cb15b] DiffEqTutorials v0.0.0 [`C:\Users\accou\.julia\external\DiffEq
Tutorials.jl`]
  [0c46a032] DifferentialEquations v6.3.0
  [497a8b3b] DoubleFloats v0.7.5
  [587475ba] Flux v0.7.3
  [f6369f11] ForwardDiff v0.10.3+ [`C:\Users\accou\.julia\dev\ForwardDiff`]
  [28b8d3ca] GR v0.38.1
  [7073ff75] IJulia v1.17.0
  [c601a237] Interact v0.9.1
  [b6b21f68] Ipopt v0.5.4
  [4076af6c] JuMP v0.19.0
  [e5e0dc1b] Juno v0.5.4
  [eff96d63] Measurements v2.0.0
  [76087f3c] NLopt v0.5.1
  [429524aa] Optim v0.17.2
  [1dea7af3] OrdinaryDiffEq v5.2.1+ [`C:\Users\accou\.julia\dev\OrdinaryDif
fEq`]
  [65888b18] ParameterizedFunctions v4.1.1
  [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
  [1986cc42] Unitful v0.14.0
  [2a06ce6d] UnitfulPlots v0.0.0 #master (https://github.com/ajkeller34/Uni
tfulPlots.jl)
  [44d3d7a6] Weave v0.7.1 [`C:\Users\accou\.julia\dev\Weave`]
```