Kepler Problem

Yingbo Ma, Chris Rackauckas

June 29, 2020

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

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

Also, we know that

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

```
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);
```

```
retcode: Success
Interpolation: 3rd order Hermite
t: 201-element Array{Float64,1}:
  0.0
  0.1
  0.2
  0.30000000000000004
  0.4
  0.5
  0.6
  0.7
  0.7999999999999999
  0.8999999999999999
  ⋮
 19.200000000000003
 19.300000000000004
 19.400000000000006
 19.500000000000007
```

```
 19.60000000000001
 19.70000000000001
 19.80000000000001
 19.900000000000013
 20.0
u: 201-element Array{RecursiveArrayTools.ArrayPartition{Float64,Tuple{Array
{Float64,1},Array{Float64,1}}},1}:
 [0.0, 2.0][0.4, 0.0]
 [-0.5830949354540153, 1.8556656829703986][0.36982713146498514, 0.195035965
14776078]
 [-0.9788105843777312, 1.5274462532150213][0.28987830863610903, 0.364959747
35762693]
 [-1.17547762665905, 1.1751394486895783][0.18078065407309682, 0.49984577206
18293]
 [-1.2440239387295458, 0.8720450804540057][0.05902925334751511, 0.601695680
2132387]
 [-1.2441259417439434, 0.6289994697149073][-0.06577256855272472, 0.67627471
02291482]
 [-1.210142434136089, 0.4368770315976506][-0.188677607179601, 0.72919425685
91364]
 [-1.159918613868923, 0.28408169071815415][-0.30726896099260204, 0.76495839
90935568]
 [-1.1025329550493486, 0.16100716005909121][-0.42042727561865095, 0.7869985
179897889]
 [-1.0426125487031446, 0.060470449725238l7][-0.5276934467175253, 0.79790892
70264804]
 ⋮
 [-1.2216434770974676, 1.0146166139270498][0.12021680827053512, 0.555011377
5144692]
 [-1.2499499900381417, 0.7423750265723883][-0.003918416420213356, 0.6423528
468283568]
 [-1.2298310873691611, 0.5265058660314975][-0.12818281922639643, 0.70537248
17632256]
 [-1.1861148292768293, 0.3555788492114466][-0.24911096207713992, 0.74915056
05432051]
 [-1.1314960903670108, 0.2188164842264573][-0.36504892367796954, 0.77762418
23022721]
 [-1.0724336821492226, 0.10787192092148691][-0.47526538003987784, 0.7937719
633967374]
 [-1.0122234000273465, 0.016617787590286977][-0.579499110310746, 0.79985306
90972269]
 [-0.9525349461454056, -0.05939856743324051][-0.6777283550436937, 0.7976021
348885407]
 [-0.894185739649566, -0.12343221924182135][-0.7700512224747644, 0.78837185
320841]
```

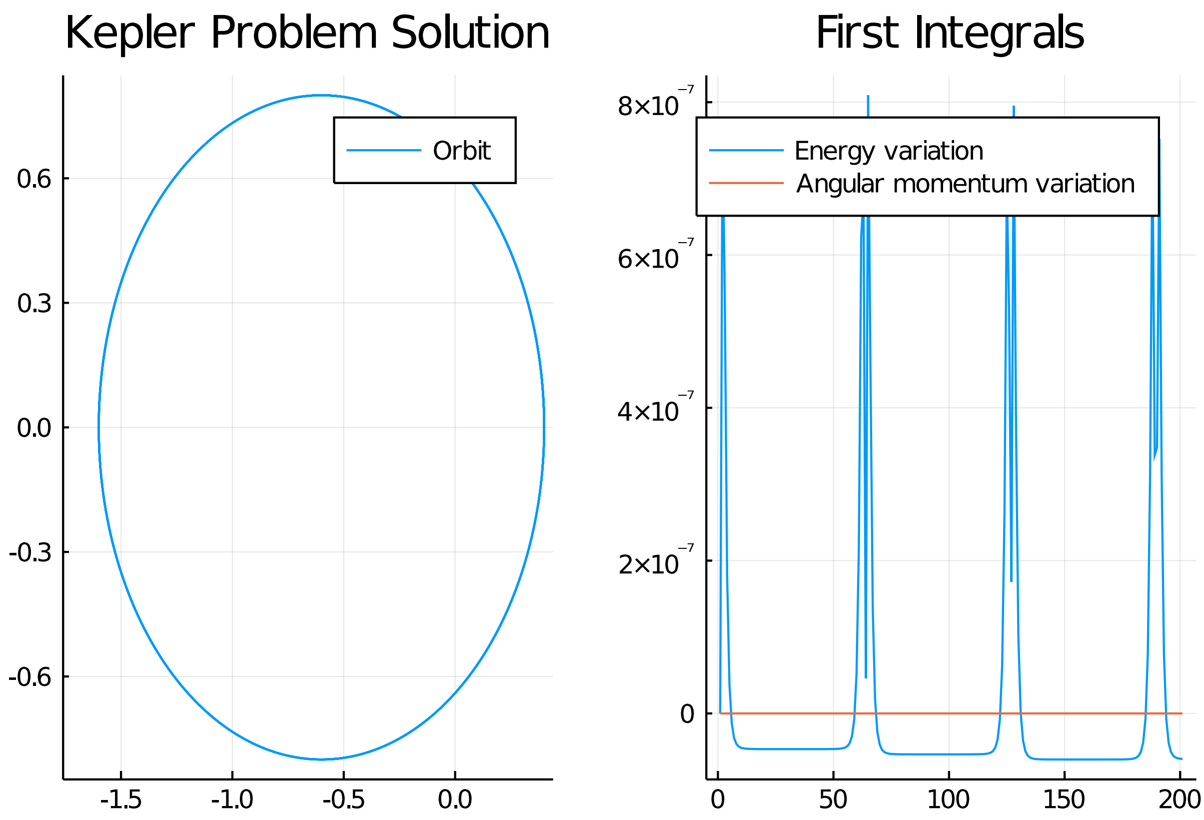
Let's plot the orbit and check the energy and angular momentum variation. We know that energy and angular momentum should be constant, and they are also called first integrals.

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

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



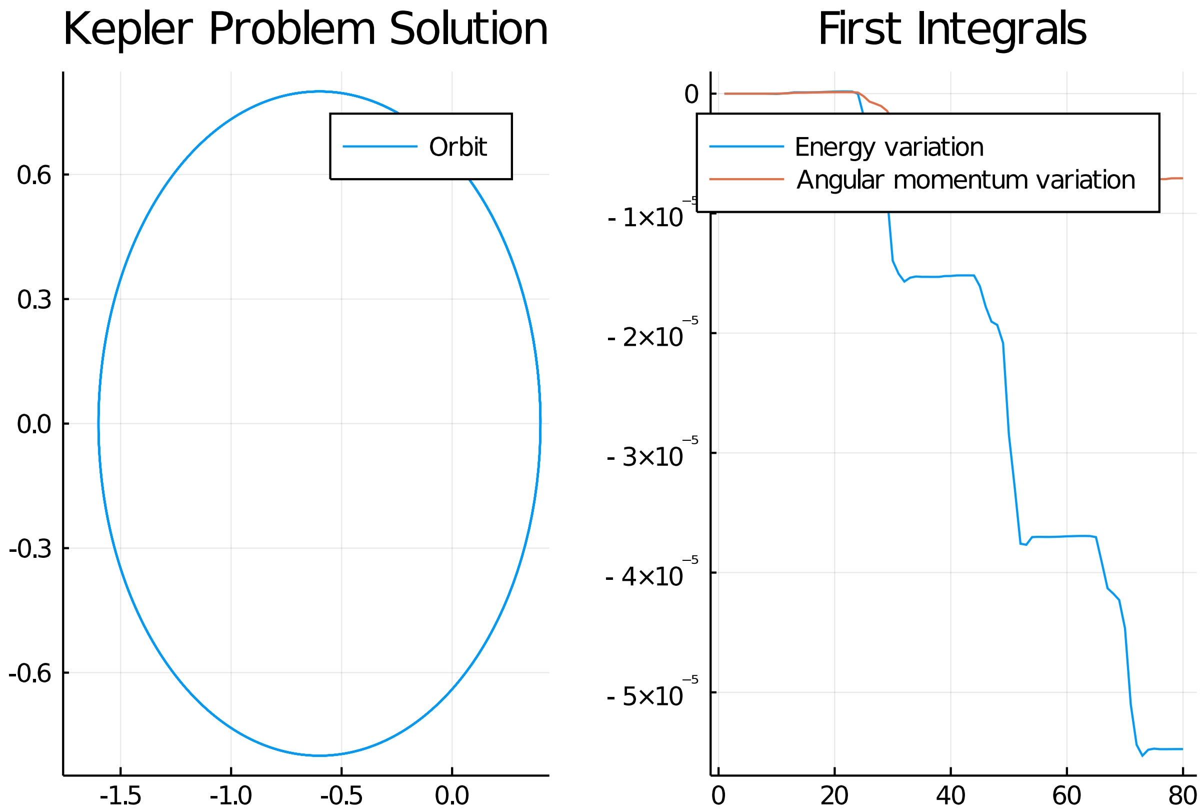
Let's try to use a Runge-Kutta-Nyström solver to solve this problem and check the first integrals' variation.

```
sol2 = solve(prob, DPRKN6())  # dt is not necessary, because unlike symplectic
                              # integrators DPRKN6 is adaptive
```

```
@show sol2.u |> length
```

```
sol2.u |> length = 80
```

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

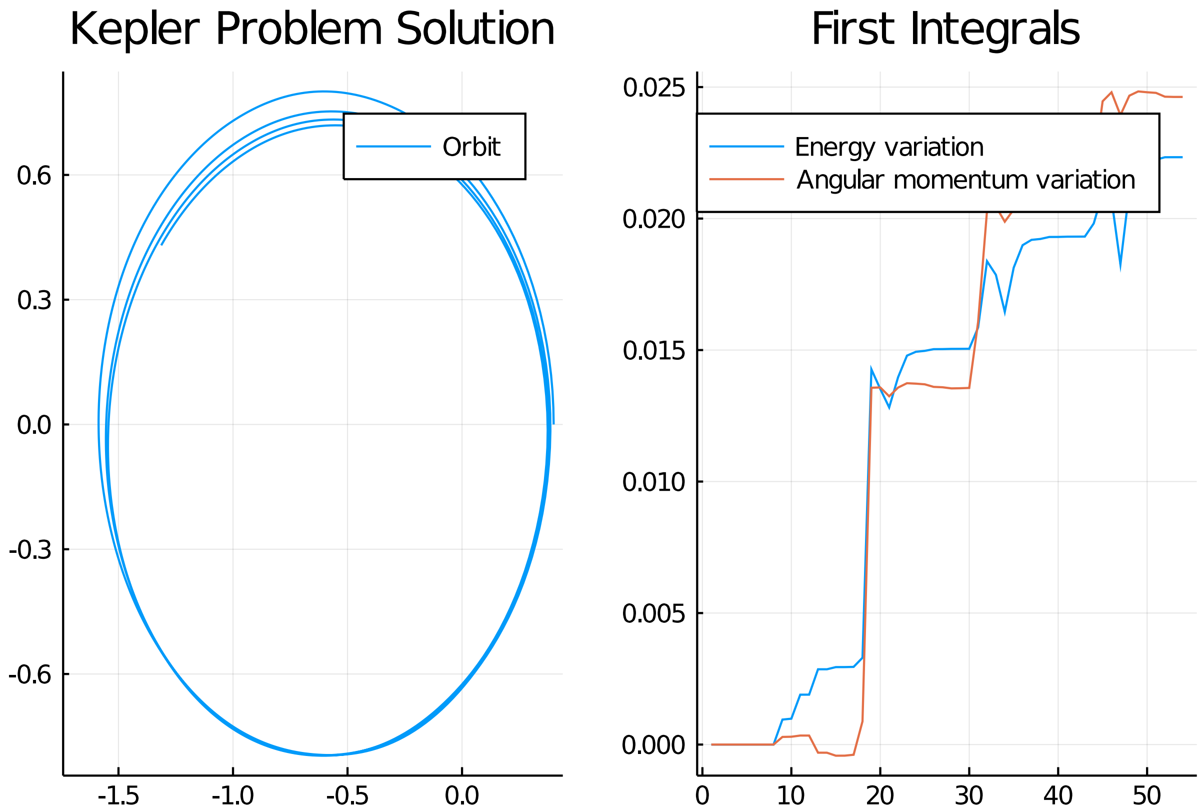


Let's then try to solve the same problem by the `ERKN4` solver, which is specialized for sinusoid-like periodic function

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

```
sol3.u |> length = 54
```

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



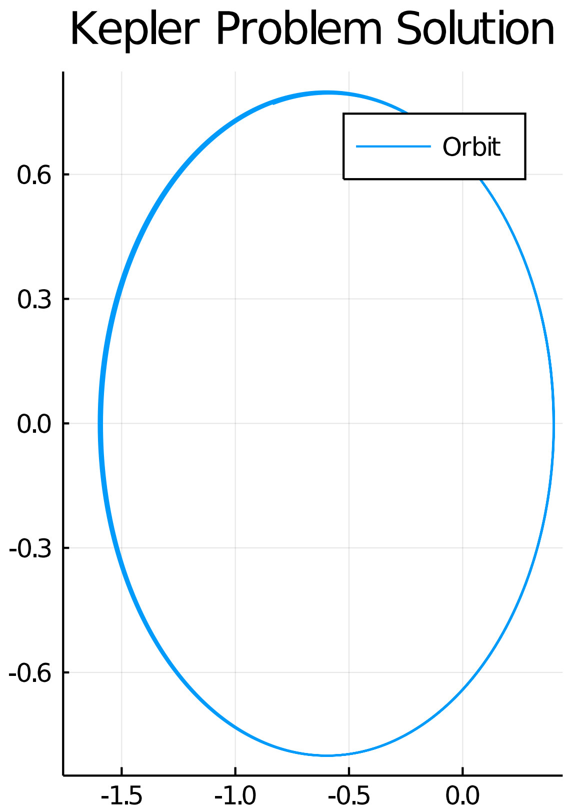
We can see that `ERKN4` does a bad job for this problem, because this problem is not sinusoid-like.

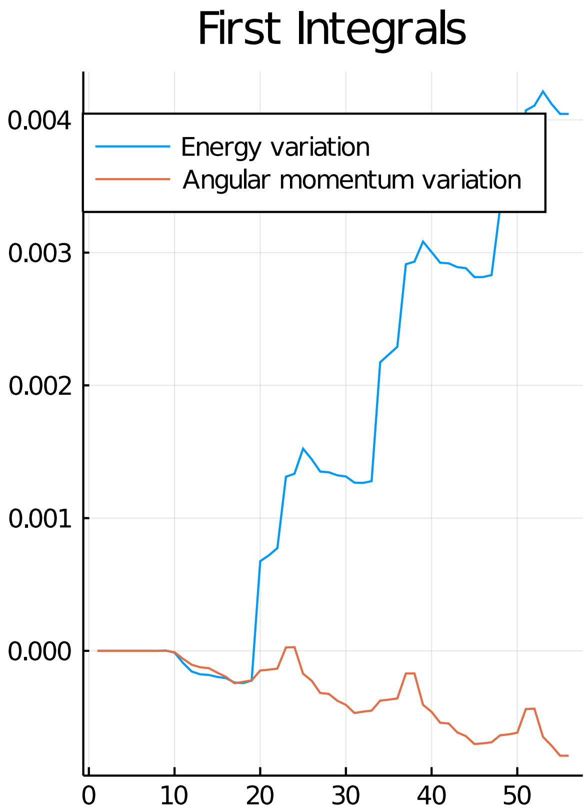
One advantage of using `DynamicalODEProblem` is that it can implicitly convert the second order ODE problem to a *normal* system of first order ODEs, which is solvable for other ODE solvers. Let's use the `Tsit5` solver for the next example.

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

```
sol4.u |> length = 56
```

```
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
```

```
Error: ArgumentError: Package DiffEqCallbacks not found in current path:
- Run `import Pkg; Pkg.add("DiffEqCallbacks")` to install the DiffEqCallbac
ks package.
```

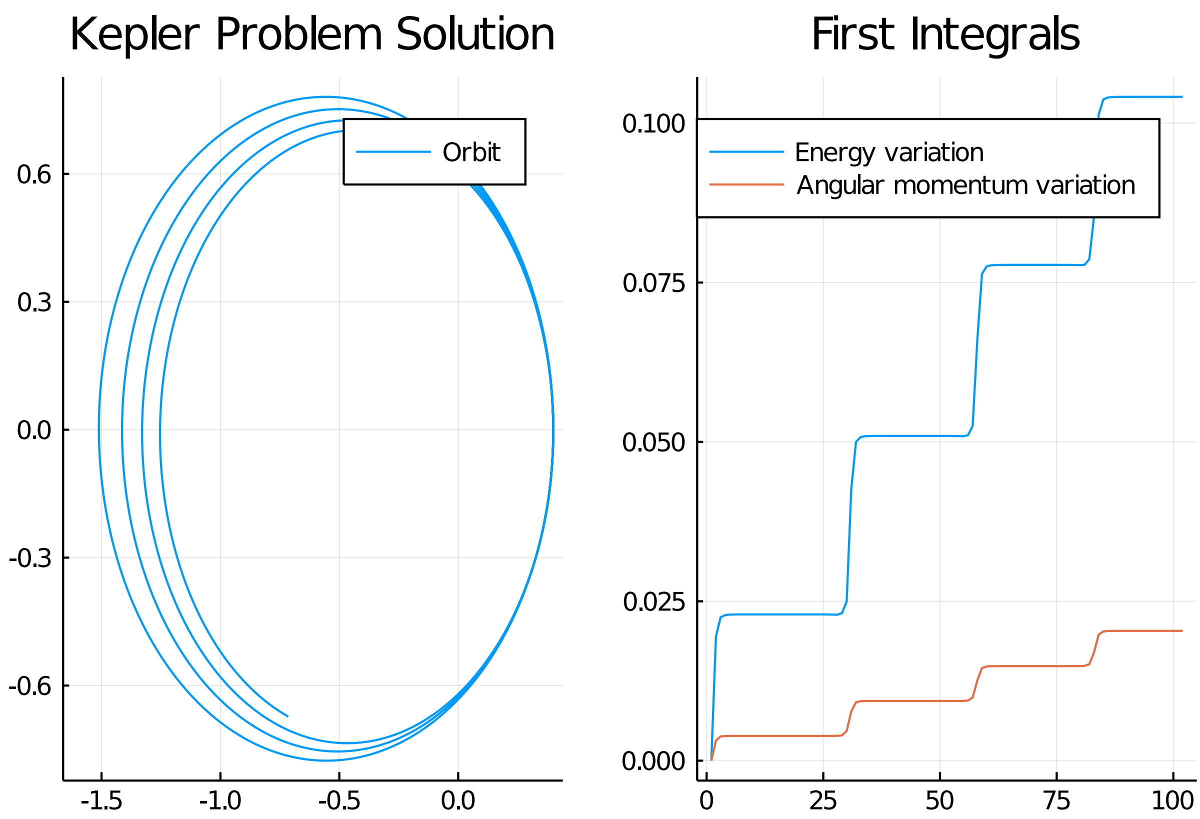
```
plot_orbit2(sol) = plot(sol,vars=(1,2), lab="Orbit", title="Kepler Problem Solution")

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)
```

```
Error: UndefVarError: ManifoldProjection not defined
```

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

```
Error: UndefVarError: cb not defined
```

```
analysis_plot2(sol5, H, L)
```

```
Error: UndefVarError: sol5 not defined
```

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)
```

```
Error: UndefVarError: ManifoldProjection not defined
```

```
sol6 = solve(prob2, RK4(), dt=1//5, adaptive=false, callback=energy_cb)
```

```
Error: UndefVarError: energy_cb not defined
```

```
analysis_plot2(sol6, H, L)
```

```
Error: UndefVarError: sol6 not defined
```

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)
```

```
Error: UndefVarError: ManifoldProjection not defined
```

```
sol7 = solve(prob2, RK4(), dt=1//5, adaptive=false, callback=angular_cb)
```

```
Error: UndefVarError: angular_cb not defined
```

```
analysis_plot2(sol7, H, L)
```

```
Error: UndefVarError: sol7 not defined
```

Again, we see what we expect.

0.2 Appendix

This tutorial is part of the DiffEqTutorials.jl repository, found at: https://github.com/JuliaDiffEq/DiffEq

To locally run this tutorial, do the following commands:

```
using DiffEqTutorials
DiffEqTutorials.weave_file("models","05-kepler_problem.jmd")
```

Computer Information:

```
Julia Version 1.4.2
Commit 44fa15b150* (2020-05-23 18:35 UTC)
Platform Info:
  OS: Linux (x86_64-pc-linux-gnu)
  CPU: Intel(R) Core(TM) i7-9700K CPU @ 3.60GHz
  WORD_SIZE: 64
  LIBM: libopenlibm
  LLVM: libLLVM-8.0.1 (ORCJIT, skylake)
Environment:
  JULIA_DEPOT_PATH = /builds/JuliaGPU/DiffEqTutorials.jl/.julia
  JULIA_CUDA_MEMORY_LIMIT = 536870912
  JULIA_PROJECT = @.
  JULIA_NUM_THREADS = 4
```

Package Information:

```
Status `/builds/JuliaGPU/DiffEqTutorials.jl/tutorials/models/Project.toml`
[eb300fae-53e8-50a0-950c-e21f52c2b7e0] DiffEqBiological 4.3.0
[f3b72e0c-5b89-59e1-b016-84e28bfd966d] DiffEqDevTools 2.22.0
[055956cb-9e8b-5191-98cc-73ae4a59e68a] DiffEqPhysics 3.2.0
[0c46a032-eb83-5123-abaf-570d42b7fbaa] DifferentialEquations 6.14.0
[31c24e10-a181-5473-b8eb-7969acd0382f] Distributions 0.23.4
[587475ba-b771-5e3f-ad9e-33799f191a9c] Flux 0.10.4
[f6369f11-7733-5829-9624-2563aa707210] ForwardDiff 0.10.11
[23fbe1c1-3f47-55db-b15f-69d7ec21a316] Latexify 0.13.5
[961ee093-0014-501f-94e3-6117800e7a78] ModelingToolkit 3.11.0
[2774e3e8-f4cf-5e23-947b-6d7e65073b56] NLsolve 4.4.0
[8faf48c0-8b73-11e9-0e63-2155955bfa4d] NeuralNetDiffEq 1.6.0
[429524aa-4258-5aef-a3af-852621145aeb] Optim 0.21.0
[1dea7af3-3e70-54e6-95c3-0bf5283fa5ed] OrdinaryDiffEq 5.41.0
[91a5bcdd-55d7-5caf-9e0b-520d859cae80] Plots 1.4.4
[731186ca-8d62-57ce-b412-fbd966d074cd] RecursiveArrayTools 2.5.0
[789caeaf-c7a9-5a7d-9973-96adeb23e2a0] StochasticDiffEq 6.23.1
[37e2e46d-f89d-539d-b4ee-838fcccc9c8e] LinearAlgebra
[2f01184e-e22b-5df5-ae63-d93ebab69eaf] SparseArrays
```