The Outer Solar System

Yingbo Ma, Chris Rackauckas

July 2, 2019

0.1 Data

The chosen units are: masses relative to the sun, so that the sun has mass 1. We have taken $m_0 = 1.00000597682$ to take account of the inner planets. Distances are in astronomical units , times in earth days, and the gravitational constant is thus $G = 2.95912208286 \cdot 10^{-4}$.

planet	mass	
Jupiter	$m_1 = 0.000954786104043$	-3.5023653 -3.8169847 -1.5
Saturn	$m_2 = 0.000285583733151$	9.0755314 -3.0458353 -1.6
Uranus	$m_3 = 0.0000437273164546$	8.3101420 -16.2901086 -7.2
Neptune	$m_4 = 0.0000517759138449$	11.4707666 -25.7294829 -10.8
Pluto	$ m_5 = 1/(1.3 \cdot 10^8)$	-15.5387357 -25.2225594 -3.1

The data is taken from the book "Geometric Numerical Integration" by E. Hairer, C. Lubich and G. Wanner.

```
using Plots, OrdinaryDiffEq, DiffEqPhysics, RecursiveArrayTools
gr()

G = 2.95912208286e-4
M = [1.00000597682, 0.000954786104043, 0.000285583733151, 0.0000437273164546,
    0.0000517759138449, 1/1.3e8]
planets = ["Sun", "Jupiter", "Saturn", "Uranus", "Neptune", "Pluto"]

pos_x = [0.0,-3.5023653,9.0755314,8.3101420,11.4707666,-15.5387357]
pos_y = [0.0,-3.8169847,-3.0458353,-16.2901086,-25.7294829,-25.2225594]
pos_z = [0.0,-1.5507963,-1.6483708,-7.2521278,-10.8169456,-3.1902382]
pos = ArrayPartition(pos_x,pos_y,pos_z)

vel_x = [0.0,0.00565429,0.00168318,0.00354178,0.00288930,0.00276725]
vel_y = [0.0,-0.00412490,0.00483525,0.00137102,0.00114527,-0.00170702]
vel_z = [0.0,-0.00190589,0.00192462,0.00055029,0.00039677,-0.00136504]
vel = ArrayPartition(vel_x,vel_y,vel_z)

tspan = (0.,200_000)
```

```
(0.0, 200000)
```

The N-body problem's Hamiltonian is

$$H(p,q) = \frac{1}{2}\sum_{i=0}^{N}\frac{p_i^T p_i}{m_i} - G\sum_{i=1}^{N}\sum_{j=0}^{i-1}\frac{m_i m_j}{\|q_i - q_j\|}$$

Here, we want to solve for the motion of the five outer planets relative to the sun, namely, Jupiter, Saturn, Uranus, Neptune and Pluto.

```
const ∑ = sum
const N = 6
potential(p, t, x, y, z, M) = -G*∑(i->∑(j->(M[i]*M[j])/sqrt((x[i]-x[j])^2 +
    (y[i]-y[j])^2 + (z[i]-z[j])^2), 1:i-1), 2:N)
```

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

0.2 Hamiltonian System

`NBodyProblem` constructs a second order ODE problem under the hood. We know that a Hamiltonian system has the form of

$$\dot{p} = -H_q(p,q) \quad \dot{q} = H_p(p,q)$$

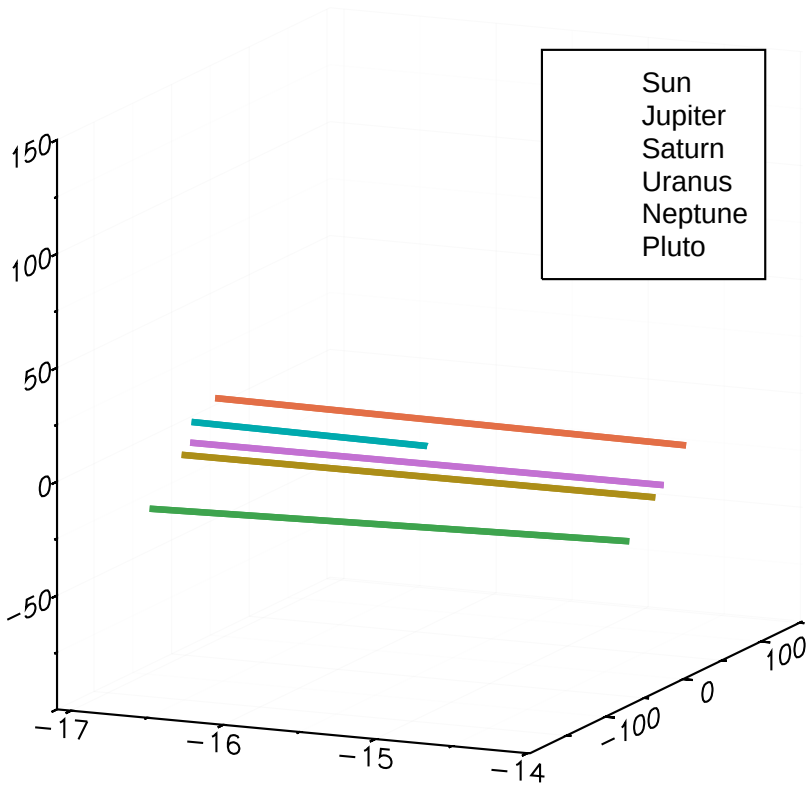
For an N-body system, we can symplify this as:

$$\dot{p} = -\nabla V(q) \quad \dot{q} = M^{-1}p.$$

Thus $\dot{q}$ is defined by the masses. We only need to define $\dot{p}$, and this is done internally by taking the gradient of V. Therefore, we only need to pass the potential function and the rest is taken care of.

```
nprob = NBodyProblem(potential, M, pos, vel, tspan)
sol = solve(nprob,Yoshida6(), dt=100);
```

```
orbitplot(sol,body_names=planets)
```



0.3 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","07-outer_solar_system.jmd")
```

Computer Information:

```
Julia Version 1.1.1
Commit 55e36cc308 (2019-05-16 04:10 UTC)
Platform Info:
  OS: Linux (x86_64-pc-linux-gnu)
  CPU: Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz
  WORD_SIZE: 64
  LIBM: libopenlibm
  LLVM: libLLVM-6.0.1 (ORCJIT, ivybridge)
```

Package Information:

```
Status `~/.julia/environments/v1.1/Project.toml`
[7e558dbc-694d-5a72-987c-6f4ebed21442] ArbNumerics 0.5.4
```

```
[6e4b80f9-dd63-53aa-95a3-0cdb28fa8baf] BenchmarkTools 0.4.2
[be33ccc6-a3ff-5ff2-a52e-74243cff1e17] CUDAnative 2.2.0
[159f3aea-2a34-519c-b102-8c37f9878175] Cairo 0.5.6
[3a865a2d-5b23-5a0f-bc46-62713ec82fae] CuArrays 1.0.2
[55939f99-70c6-5e9b-8bb0-5071ed7d61fd] DecFP 0.4.8
[abce61dc-4473-55a0-ba07-351d65e31d42] Decimals 0.4.0
[ebbdde9d-f333-5424-9be2-dbf1e9acfb5e] DiffEqBayes 1.1.0
[eb300fae-53e8-50a0-950c-e21f52c2b7e0] DiffEqBiological 3.8.2
[459566f4-90b8-5000-8ac3-15dfb0a30def] DiffEqCallbacks 2.5.2
[f3b72e0c-5b89-59e1-b016-84e28bfd966d] DiffEqDevTools 2.9.0
[1130ab10-4a5a-5621-a13d-e4788d82bd4c] DiffEqParamEstim 1.6.0
[055956cb-9e8b-5191-98cc-73ae4a59e68a] DiffEqPhysics 3.1.0
[6d1b261a-3be8-11e9-3f2f-0b112a9a8436] DiffEqTutorials 0.1.0
[0c46a032-eb83-5123-abaf-570d42b7fbaa] DifferentialEquations 6.4.0
[31c24e10-a181-5473-b8eb-7969acd0382f] Distributions 0.20.0
[497a8b3b-efae-58df-a0af-a86822472b78] DoubleFloats 0.9.1
[f6369f11-7733-5829-9624-2563aa707210] ForwardDiff 0.10.3
[c91e804a-d5a3-530f-b6f0-dfbca275c004] Gadfly 1.0.1
[7073ff75-c697-5162-941a-fcdaad2a7d2a] IJulia 1.18.1
[4138dd39-2aa7-5051-a626-17a0bb65d9c8] JLD 0.9.1
[23fbe1c1-3f47-55db-b15f-69d7ec21a316] Latexify 0.8.2
[eff96d63-e80a-5855-80a2-b1b0885c5ab7] Measurements 2.0.0
[961ee093-0014-501f-94e3-6117800e7a78] ModelingToolkit 0.2.0
[76087f3c-5699-56af-9a33-bf431cd00edd] NLopt 0.5.1
[2774e3e8-f4cf-5e23-947b-6d7e65073b56] NLsolve 4.0.0
[429524aa-4258-5aef-a3af-852621145aeb] Optim 0.18.1
[1dea7af3-3e70-54e6-95c3-0bf5283fa5ed] OrdinaryDiffEq 5.8.1
[65888b18-ceab-5e60-b2b9-181511a3b968] ParameterizedFunctions 4.1.1
[91a5bcdd-55d7-5caf-9e0b-520d859cae80] Plots 0.25.1
[d330b81b-6aea-500a-939a-2ce795aea3ee] PyPlot 2.8.1
[731186ca-8d62-57ce-b412-fbd966d074cd] RecursiveArrayTools 0.20.0
[90137ffa-7385-5640-81b9-e52037218182] StaticArrays 0.11.0
[f3b207a7-027a-5e70-b257-86293d7955fd] StatsPlots 0.11.0
[c3572dad-4567-51f8-b174-8c6c989267f4] Sundials 3.6.1
[1986cc42-f94f-5a68-af5c-568840ba703d] Unitful 0.15.0
[44d3d7a6-8a23-5bf8-98c5-b353f8df5ec9] Weave 0.9.0
[b77e0a4c-d291-57a0-90e8-8db25a27a240] InteractiveUtils
[37e2e46d-f89d-539d-b4ee-838fcccc9c8e] LinearAlgebra
[44cfe95a-1eb2-52ea-b672-e2afdf69b78f] Pkg
```