N-body choreographies

Subject: Plots and animations, function interpolations, built-in methods.

Introduction

The motion of three or more bodies subject to Newtonian gravity is known to be chaotic, but this does not exclude the possibility of having periodic orbits for very particular initial conditions. Indeed, for three bodies the Euler-Lagrange configurations are the best-known example of periodic solutions, but others have been found numerically. For example, the "figure 8" solution of [1] is shown in Fig. 1

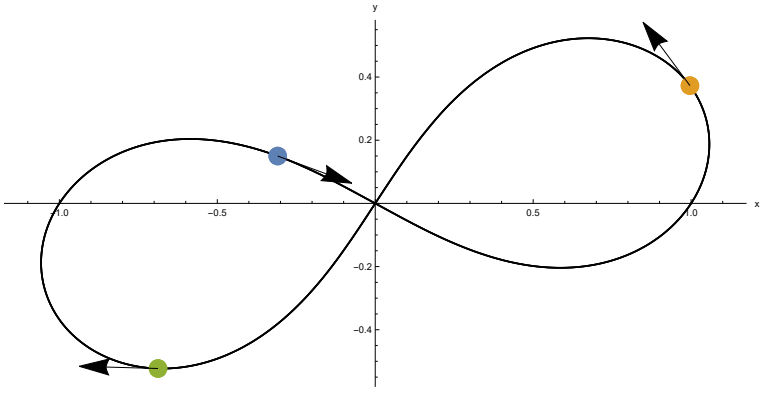

Figure 1: Figure 8 solution of the 3-body problem with equal masses, [1]. This solution has zero angular momentum.

In this problem, we will explore the motion of N-bodies numerically by constructing simulations for arbitrary initial conditions. While we will concentrate mostly on the case with $N = 3$ equal masses moving in $d = 2$ spatial dimensions subject to the pairwise potential $V_{ij} = r_{ij}^{-1}$, you should

try to write your code to be as flexible as possible. Periodic orbits have been observed with more bodies, other potentials (power laws, Van der Waals, Lennard-Jones, ...), different masses, higher dimensions, etc.

Denoting the bodies' positions and momenta by $\vec{x}_i(t)$ and $\vec{p}_i(t)$ for $i = 1, 2, \ldots, N$, the system's time evolution is completely determined by the initial conditions $\vec{x}_i(0)$ and $\vec{p}_i(0)$. We say there is periodic motion with period T if $(\vec{x}_i(T), \vec{p}_i(T)) = (\vec{x}_i(0), \vec{p}_i(0))$ and this property does not hold for any other $t \in (0, T)$. In order to look for initial conditions leading to periodic orbits, we can then define the "return error" at time t as

$$E(t) = \sqrt{\sum_{i=1}^{N} \left(\vec{x}_i(t) - \vec{x}_i(0)\right)^2 + \left(\vec{p}_i(t) - \vec{p}_i(0)\right)^2} \geq 0\,, \tag{1}$$

which is simply the Euclidean distance in phase space to the initial configuration. Since periodic motion with period T implies $E(T) = 0$, we look for orbits with period $T \leq T_{\max}$ by numerically computing

$$F \equiv \min_{0<t\leq T_{\max}} E(t)\,. \tag{2}$$

We say a periodic motion of N bodies is "choreographic" if all the particles have the same trajectory, and after time T/N each of them occupies another one's position in phase space in a cyclic fashion. That is, $(\vec{x}_i(t + \frac{T}{N}), \vec{p}_i(t + \frac{T}{N})) = (\vec{x}_{i+1}(t), \vec{p}_{i+1}(t))$, where we identify bodies $N + 1 \equiv 1$ and we permute the labels following the trajectory. The figure 8 solution shown in Fig. 1 is a choreography in this sense, and we can look for other choreograhies by defining an error function in a similar manner to (1).

Finally, we should notice that since the return error depends on the $2N$ parameters given by the initial conditions $(\vec{x}_i(0), \vec{p}_i(0))$, and is very sensitive to even small changes in these, we should try to reduce their number as much as possible in order to constrain the search space. Two obvious conditions we should impose are

$$\sum_{i=1}^{N} \vec{x}_i(0) = 0 \qquad \text{and} \qquad \sum_{i=1}^{N} \vec{p}_i(0) = 0\,, \tag{3}$$

i.e. the system's center of mass is fixed at the origin. We may also impose other conditions that, while not preserving full generality, might be motivated

by physical or symmetry considerations. For example, we can demand the vanishing of total angular momentum or the initial positions to fall within a single straight line.

Problem statement

Fixed-timestep integration

We begin by implementing a fixed-timestep simulation, in which the trajectories of the bodies are obtained at successive times $t_n = n\,\delta t$ for $n = 1, 2, \ldots$, starting from the arbitrary initial conditions at $t = 0$. Given the state at time t_n, we compute the acceleration of the bodies from Newton's law,

$$\left.\frac{d\vec{p}_i}{dt}\right|_{t_n} = \sum_{j\neq i} -\vec{\nabla} V_{ij} \tag{4}$$

and then obtain the state at time t_{n+1} from

$$\vec{x}_i(t_{n+1}) = \vec{x}_i(t_n) + \frac{\vec{p}_i(t_n)}{m_i}\delta t + \left.\frac{d\vec{p}_i}{dt}\right|_{t_n} \frac{\delta t^2}{2m_i}\,, \tag{5}$$

$$\vec{p}_i(t_{n+1}) = \vec{p}_i(t_n) + \left.\frac{d\vec{p}_i}{dt}\right|_{t_n} \delta t\,. \tag{6}$$

Write a function that repeats this process for a given number of steps, taking as additional parameters the masses, initial conditions as well as the timestep δt. Make sure your function works for an arbitrary number N of particles in any number of dimensions, with possibly other potentials too. Your function should return a list of N `InterpolatingFunction` objects, the i-th of which gives the position $\vec{x}_i(t)$ of the i-th body at time t.

Once you've implemented your solver, write functions to visualize arbitrary simulations, both as a snapshot of the orbits and as animations of the bodies moving along their trajectories, see[1] Fig. 2 (take a look at the `Animate` function).

[1]For reference purposes, the initial conditions used in Fig. 2 are

$\vec{x}_1(0) = (-0.750288, -0.346587)$ $\quad \vec{x}_2(0) = (-0.276912, -0.120384)$ $\quad \vec{x}_3(0) = (0.726513, -0.0532461)$
$\vec{x}_4(0) = (0.284695, 0.37585)$ $\quad \vec{x}_5(0) = (0.0159924, 0.144367)$
$\vec{p}_1(0) = (0.447765, -0.410461)$ $\quad \vec{p}_2(0) = (0.157868, 0.807091)$ $\quad \vec{p}_3(0) = (-0.0093872, 0.0908442)$
$\vec{p}_4(0) = (0.342885, -0.3195)$ $\quad \vec{p}_5(0) = (-0.939132, -0.167974)$

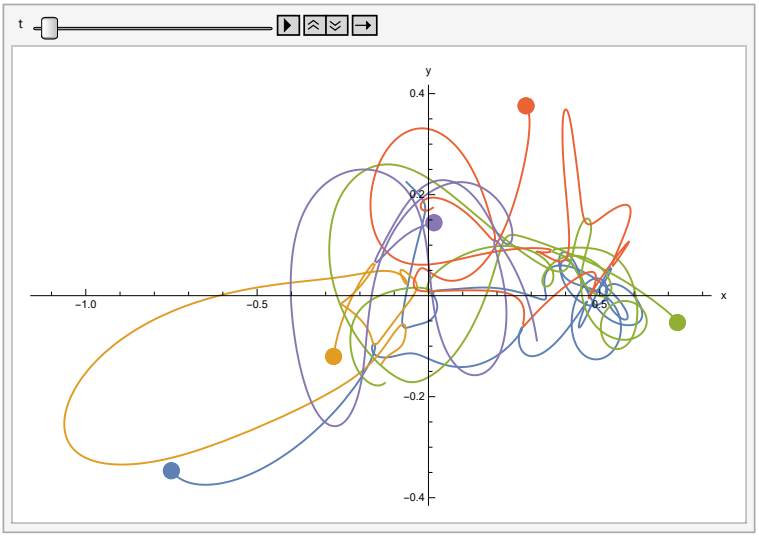

Figure 2: Plot of the `Manipulate` object created by `Animate`, showing the chaotic motion of five bodies with equal masses.

The figure 8 solution shown in Fig. 1 has a period of $T = 6.325$ and initial conditions

$$\vec{x}_1(0) = (-1, 0) \qquad \vec{x}_2(0) = (0, 0) \qquad \vec{x}_3(0) = (1, 0)\,, \tag{7}$$

$$\vec{p}_1(0) = (p_x, p_y) \qquad \vec{p}_2(0) = -2(p_x, p_y) \qquad \vec{p}_3(0) = (p_x, p_y)\,, \tag{8}$$

with $p_x = 0.3471128135672417$ and $p_y = 0.532726851767674$. Experimenting with different values of δt, make sure you can find this solution using your code. What happens if δt is too large?

Adaptive timestep integration with `NDSolve`

The main problem with the fixed-timestep integration of the previous subsection is that very small steps are required whenever velocities are large, but not when they are small. Yet having fixed δt, the maximum velocity attained during the whole simulation introduces a constraint acting as a bottleneck for our algorithm.

While we could try to write a variant of the fixed-timestep integrator using a different value of δt_n for steps $n = 1, 2, \ldots$, we should realize this

is exactly what `Mathematica` does when solving one-dimensional differential equations. Therefore, we can use the powerful `NDSolve` as a shortcut to avoid implementing a complicated adaptive timestep integrator: we simply plug Newton's equation into `NDSolve`, specifying the appropriate initial conditions and variables, to get out the simulated trajectories efficiently computed with very high precision.

Write a wrapper for `NDSolve` implementing the process described in the previous paragraph. Include a "bounding" parameter R such that the simulation stops whenever $\vec{x}_i^2(t) \geq R$. In this way, we can make sure that we don't produce long simulations where bodies are actually escaping from each other and periodic motion is clearly not present.

There are many examples in the literature of periodic orbits for three bodies with initial conditions of the form (7). Try to reproduce some of the ones found in [2], see *e.g.* Fig. 3.

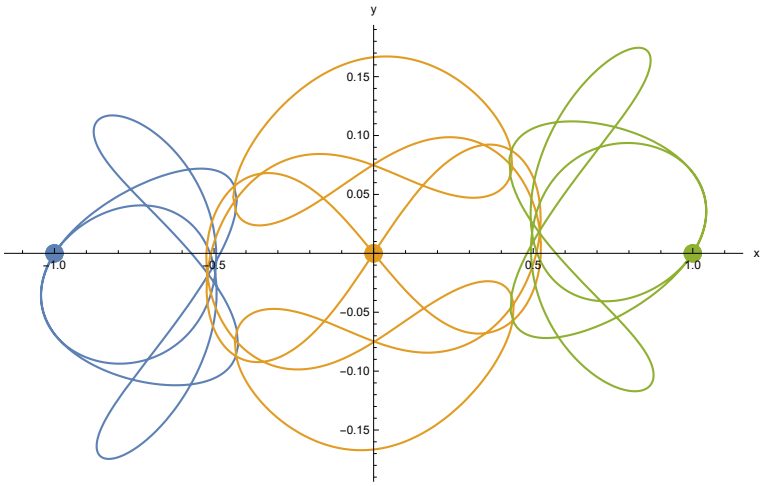

Figure 3: Plot of the "butterfly I" periodic solution of [2], with initial conditions given by (7) with $(p_x, p_y) = (0.306892758965492, 0.125506782829762)$.

Using the built-in `NBodySimulation` (experimental)

In fact, `Mathematica` version 12 incorporates an `NBodySimulation` function that streamlines the process of producing N-body simulations in very general conditions. Write another wrapper for this function, and compare the results

and efficiency to both of the functions you wrote for the previous sections. What are the advantages / disadvantages of each approach?

Finding periodic orbits

In order to look for periodic motion, define a function in `Mathematica` computing the return error (1) for an arbitrary simulation at any given time. Then use the built-in minimization functions to find the minimum return error during all of the simulation, *i.e.* implement (2) numerically. Remember that you want to find the *global* minimum, so you may have to try a few different starting points to avoid falling into local minima.

The minimal return error is a function of the parameters entering the specification of the system's initial conditions, which we collect in a vector $\vec{\theta}$. For example, in the case of (7) we have $\vec{\theta} = (p_x, p_y)$, but in general $\vec{\theta}$ is $2d(N-1)$ dimensional We would like to find the special points $\vec{\theta}^*$ in this parameter space such that

$$F_{\vec{\theta}^*} \equiv \min_{0<t\leq T_{\max}} E_{\vec{\theta}^*}(t) = 0\,, \tag{9}$$

where we made explicit the $\vec{\theta}$ dependence everywhere. However, this can be challenging if $\vec{\theta}$ is high-dimensional.

One way to proceed is to again use `Mathematica` minimization functions to minimize F in (2) with respect to $\vec{\theta}$, but here we will take a different approach for pedagogical purposes. Instead, we will use gradient descent to find local minima of F starting from an initial set of parameters $\vec{\theta}_0$. This works as follows: at each step of gradient descent, we have a set of parameters $\vec{\theta}_n$, and numerically compute the gradient of $F_{\vec{\theta}}$ at this point; the next step is to then move to a new point $\vec{\theta}_{n+1}$ in the direction of maximal descent, *i.e.* the negative gradient. In formulas,

$$\vec{\theta}_{n+1} = \vec{\theta}_n - \alpha \frac{\partial F}{\partial \vec{\theta}}\,, \tag{10}$$

where α is the *learning rate*, a parameter of the gradient descent algorithm that needs to be fine-tuned by hand.

Implement the gradient descent algorithm to try and find the initial conditions leading to the figure 8 solution. You may have to play with the learning rate to achieve convergence (*i.e.* making it depend on the iteration

number, changing it according to past history, etc). While you will probably not be able to achieve better performance than `Mathematica` minimization functions for the two-parameter family of initial conditions in (7), this approach may have an advantage in higher-dimensional settings and, in any case, lets you control and follow the minimization procedure in detail.

Optional

You can repeat the above procedure defining a "choreographic" return error, and then look now for choreographies instead of just periodic motion.

Useful functions

You may find it useful to read the `Mathematica` help pages of the following functions:

- **Others:** `NDSolve`, `WhenEvent`, `FindMinimum`, `NMinimize`, `MinimalBy`, `Interpolation`, ...

References

[1] C. Moore, "Braids in classical dynamics," Phys. Rev. Lett. **70** (1993), 3675 doi:10.1103/PhysRevLett.70.3675.

[2] M. Suvakov and V. Dmitrasinović, "A guide to hunting periodic three-body orbits," Am. J. Phys. **82**, 609 (2014), doi:10.1119/1.4867608.