GalaxyModel Technical Note

Haihong
Hangzhou, Zhejiang. China

Valentine's Day, 2015

Abstract

This document describes a MATLAB program which conducts gravitational dynamic computer simulation of interacting astronomical bodies. *N-body problem* is an enduring and intriguing classical physics problem, which can be described as below (Wikipedia):

> *Given the quasi-steady orbital properties (instantaneous position, velocity and time) of a group of celestial bodies, predict their interactive forces; and consequently, predict their true orbital motions for all future times.*

This program is constructed using Object-Oriented Programming paradigm. It integrates both direct integration (when n is small) and *Octree Method* approximation (when n is relatively large). *Verlet Algorithm* is applied to reduce global error and provide good numerical stability instead of common *Explicit Euler Algorithm*, while there is no significant additional computational cost.

1. Overview

Newtonian Mechanics is the foundation of this program. By neglecting gravity-induced space-time bending, the space where these celestial bodies exist is flat three-dimensional Euclid space. *Verlet Algorithm* is stated as follows:

$$\begin{cases} \boldsymbol{a}_i^{(k)} = f\left(m_1, \boldsymbol{r}_1^{(k)}; m_2, \boldsymbol{r}_2^{(k)}; ...; m_n, \boldsymbol{r}_n^{(k)}\right) \\ \boldsymbol{r}_i^{(k+1)} = \boldsymbol{r}_i^{(k)} + \boldsymbol{v}_i^{(k+1)}\Delta t + \dfrac{1}{2}\boldsymbol{a}_i^{(k)}\Delta t^2 \\ \boldsymbol{a}_i^{(k+1)} = f\left(m_1, \boldsymbol{r}_1^{(k+1)}; m_2, \boldsymbol{r}_2^{(k+1)}; ...; m_n, \boldsymbol{r}_n^{(k+1)}\right) \\ \boldsymbol{v}_i^{(k+1)} = \boldsymbol{v}_i^{(k)} + \dfrac{1}{2}\left(\boldsymbol{a}_i^{(k)} + \boldsymbol{a}_i^{(k+1)}\right)\Delta t \end{cases} \quad i = 1,2,...,n$$

Newton's Second Law and Newton's Universal Gravitational Law are applied, described as follows:

$$\boldsymbol{F}_i = m_i\boldsymbol{a}_i, \text{ and } \quad \boldsymbol{F}_i = \sum_{j\neq i} \frac{Gm_im_j}{\left|\boldsymbol{r}_{ij}\right|^2} \cdot \frac{\boldsymbol{r}_{ij}}{\left|\boldsymbol{r}_{ij}\right|} = Gm_i \sum_{j\neq i} \frac{m_j\left(\boldsymbol{r}_j - \boldsymbol{r}_i\right)}{\left|\boldsymbol{r}_j - \boldsymbol{r}_i\right|^3}.$$

If the number of celestial bodies is relatively small, the simulation can be performed straightforwardly by calculating each pairs' gravitational forces. However, the computation process, based on the order of n^2 computations, could be very slow, if the number of celestial bodies is large. Several approximation methods have been developed to reduce the time complexity, including *Octree Method* (in astronomical field it is sometimes referred as

Barnes-Hut algorithm), which is of the order of *n*lg*n*.

An *Octree* is a tree data structure in which each node has eight children nodes, and its two-dimensional analog is a *quadtree*. The essence of *Octree Method* is recursively partitioning a three-dimensional cubical space into eight identical octants (bins), and treat celestial bodies in each octants as a whole group.

For a particular celestial body, if a bin is sufficiently far away from it, then gravitational forces exerted on this body by bodies within the bin can be regarded as a gravitational force exerted by the center of mass of this bin. And if the bin is not sufficiently far from this given body, gravitational forces must be calculated separately. Whether a bin is or is not sufficiently far from the given body is determined by the quotient of the size of this bin and the distance between the given body and the center of mass of this bin, and the bin is sufficiently far away if the quotient is smaller than a given threshold.

For each time step, all celestial bodies in the space are traversed and each body's acceleration is calculated by Newton's gravitational law (with or without *Octree Method*, depending on the total number of the bodies). Then the velocities and coordinates of these bodies are determined by the *Verlet Algorithm*.

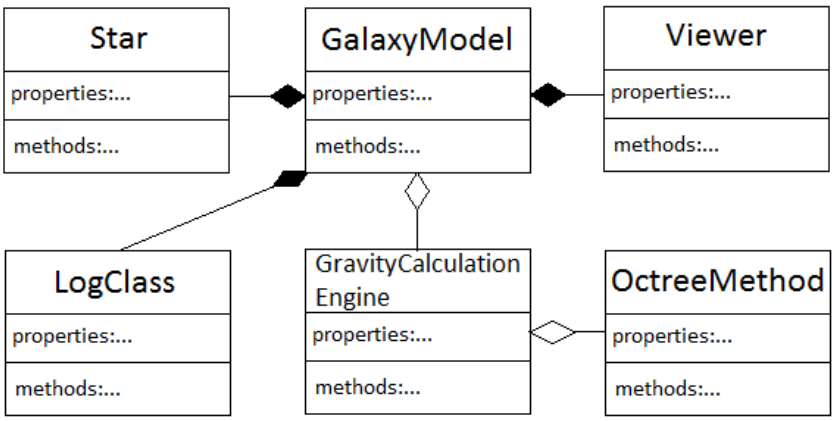

Fig.1 UML diagram

2. Usage

2.1 Simple Usage

The initial input variable `initialState` should be a struct array whose elements represent stars' initial states. Each element (star) contains the following fields: `mass` (a positive scalar), `position` (a 1-by-3 vector, relative to the origin point), `velocity` (a 1-by-3 vector), and `acceleration` (a 1-by-3 vector).

From the file provided, `makeTestRand.m` can be used to generate a randomly assigned test case, where `n` is the number of stars:

```
initialState = makeTestRand(n);
```

Of course, users are allowed to use their own test case, if the initial input is formatted correctly.

Before using, `clear classes` to remove old data (recommended). Then call the constructor in the command window to generate an object `g`:

```
g = GalaxyModel(initialState);
```

And call the member function of GalaxyModel to execute the simulation, where the total length of time `time`, and the length of each time step `dtime` are given respectively:

```
g.simulate(time,dtime);
```

Then the figure of stars' orbits would appear on the screen synchronously, the corresponding time, star index, coordinates and velocities being recorded into a text file whose file name indicates the simulation date and time, in `Log` file folder.

2.2 Advanced Usage

Some parameters and settings are user-adjustable.

(1) Visualization settings. Relevant settings are stored in class `Viewer`'s properties. There are (a) marker size, (b) axis limit, (c) choice of whether fixed axis limit is applied, (d) choice of whether minor grid is displayed, (e) choice of whether stars' trails are displayed, and (f) choice of whether star `1` is highlighted.
(2) Threshold determining whether *Octree Method* is applied. This critical number is in class `GravityCalculationEngine`'s property `octreeThreshold`. If the total number of stars is greater than this critical number, *Octree Method* is put into effect. Hint: it drastically slows down program's execution speed to use *Octree Method* if the number of stars is small .
(3)Threshold determining whether a bin is sufficiently far from a given star. This critical value is in class `GravityCalculationEngine`'s property `farThreshold`. If this value is set to 0, then *Octree Method* would never be invoked because no bin is sufficiently far.
(4)Gravitational constant `CONST_G`. This constant is in `GalaxyModel`'s member function `gravityAffect`. Defualted to 1.
And lastly, this program is still a prototype. If equipped with visualization module and controller module (using MVC model - Model, Visualization and Controller), the program would be more user-friendly and more open to extension.

3. Problem of singularity

The problem of singularity remains unsolved. This problem arises when two (or more) celestial bodies are extremely close, and therefore gravitational forces between them become unusually strong, resulting in ultra-high accelerations and velocities. In reality, these bodies might collide with each other (and merge into one entity or smash into pieces). In my program, however, collision cannot be simulated - they just "pass through" each other with very high speeds, which is contradicted to physical reality.

One possible technique is collision prediction mechanism. There are mature algorithms out there (mostly used in video games) but it is difficult to construct the codes (college's new semester is around the corner!). And from literature I have reviewed, *symplectic transformation* and *regularization* might be helpful. *Regularization* is also used in other field such statistics and machine learning.

4. Possible extensions

N-body problem is not restricted to astronomy or classic physics. For instance, in subatomic physics *n-body problem* arises as well, in which electrons and nucleons' behaviors are dominantly bound by electromagnetic interactions and strong interactions instead of gravity, and quantum mechanics plays a significant role in it.

Understanding the formation process of stars involves *n-body problem* as well. What makes it complicated is that this process covers many orders of magnitude. One possible approach is integrate *smoothed-particle hydrodynamics* (SPH).

This program has the potential of being modified to simulate other fields' n-body problems as well, such as dynamic *market* simulations or *complex network* simulations, where multiple individuals/nodes are involved.

5. Simple Examples

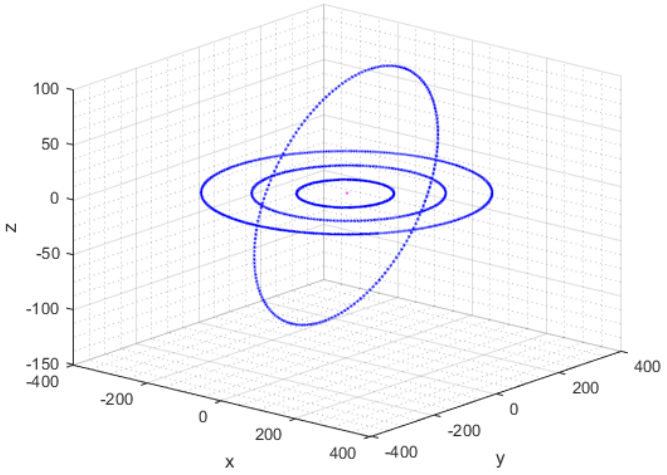

Fig.2 Example 1(a).

Test Case: Test1Big3Small, in which Octree Method is not applied.

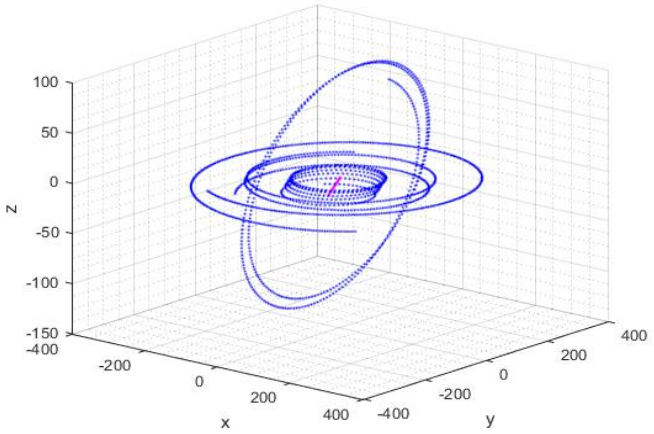

Fig.3 Example 1(b).

Test Case: Test1Big3Small, in which Octree Method (an approximation) is applied.

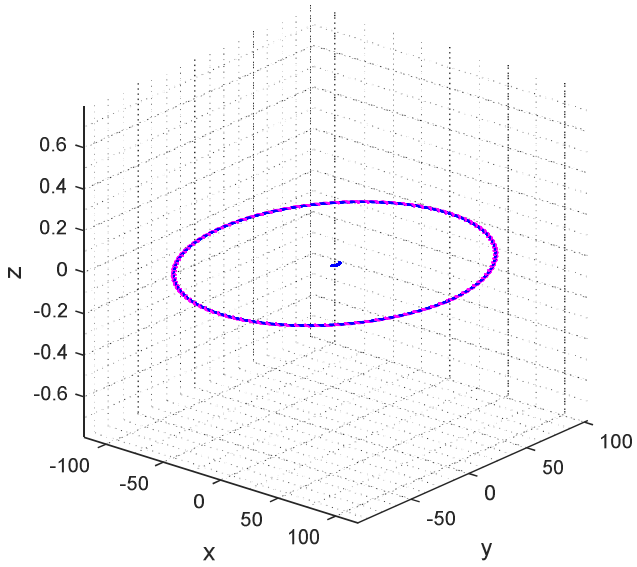

Fig.4 Example 2.

Test Case: TestBigMiddleSmall, in which the moon's initial relative velocity to the planet ensures its orbit around the planet is circle, and the planet's orbit (blue) around the central star is also a circle. From this figure we can see that the central star is also slightly moving around the center of mass of this entire system.

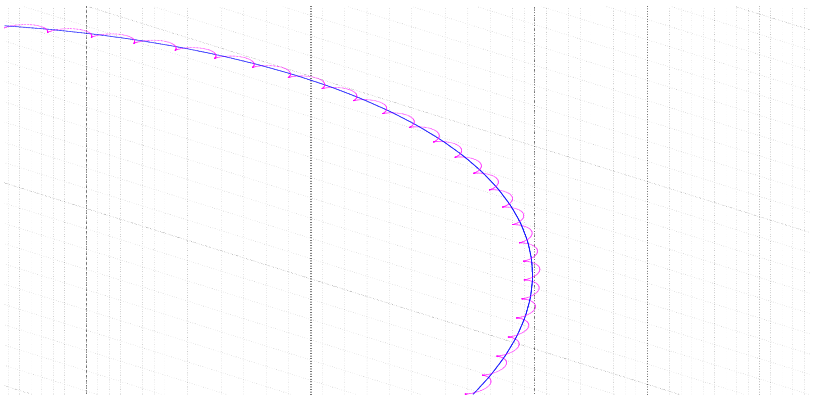

Fig.5 Example 2. Zoom in.

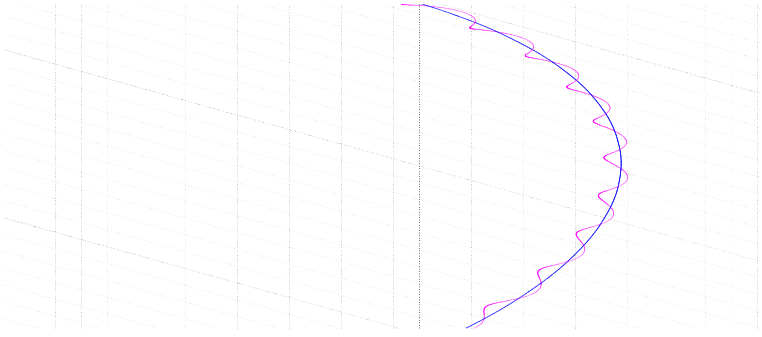

Fig.6

Another version of Example 2, in which the moon's initial relative velocity to the planet makes its orbit around the planet to be elliptical.

6. Acknowledgements

Salman Khan's video on Astronomy (on https://www.khanacademy.org/) serves as the initial inspiration of the program. My original intent was to develop a program which can simulate the evolution of interstellar matters and their process of condensing into galaxies.

University of Michigan's Sven's work Octree (MATLAB File Exchange, 2013) was of great help. And United Kingdom's Dr. Andrew French's work Gravity Simulator (MATLAB File Exchange, 2013) inspired me to use *Verlet Algorithm*. Though his programmainly focused on two-body simulation and treated other celestial bodies as effectively massless objects, I found the visualization function in this program quite impressive, and his technical note is both readable and illuminating (and his website http://www.eclecticon.info/ is interesting). Both programs mentioned above can be downloaded via MATLAB File Exchange, a free and friendly online community.

GADGET is a free software (Volker Springel, 2000 and 2005) for cosmological collisionless self-gravitating n-body system simulation and is more professionally designed. Its website is http://wwwmpa.mpa-garching.mpg.de/gadget/. V. Springel also runs Millennium Run project.