N-body simulator writing in Cython, Python and C++

Ignas Anikevičius

July 15, 2012

1 Introduction

This document describes the writing and designing of a N-body simulator. The main purpose of the study is to find out, which approach takes less time, yields more precise results and is easier to code.

The selection of language can be justified as follows:

Python It is very easy to prototype and it has a wide array of libraries, which can be useful in scientific computing.

Cython This is a way to speed up code written in Python with additional C definitions, but without limiting yourself to only C or Python.

C/C++ Software written with in these languages will be fast and the coding itself will be very instructive. However, the main drawback is the amount of time it takes to code some trivial expressions.

2 The problem

The problem is to calculate the motion of particles in infinite space, which interact with each other via gravitational forces only. The particles are considered to be material points and no dark matter or dark energy is taken into account. Therefore, we need to solve the following set of equations:

$$\mathbf{x_i}''(t) = \frac{\mathbf{F_i}(t)}{m_i} = \sum_{i \neq j}^{N} \frac{m_j (\mathbf{x_j} - \mathbf{x_i})}{|\mathbf{x_j} - \mathbf{x_i}|^3} \tag{1}$$

From this relation we can predict all the motion of the system if the initial conditions are known. Hence, we can clearly state that this is a Neumann problem, however, there are precise analytic solutions only for the case of $N = 2$. For $N > 2$ the solutions need to be approximated, which will be done by writting a computer program.

However, in order to use RK4 or other accurate integration methods, we need to have an equation of a form:

$$\frac{\mathrm{d}y}{\mathrm{d}x} = f(x, y)$$

We can re-express eq. 1 as a set of 2 first order equations as follows:

$$\begin{cases} \mathbf{x_i}'(t) = \mathbf{v}(t) \\ \mathbf{v_i}'(t) = \sum_{i \neq j}^{N} \frac{m_j (\mathbf{x_j} - \mathbf{x_i})}{|\mathbf{x_j} - \mathbf{x_i}|^3} \end{cases} \tag{2}$$

Also, since these equations are vector equations, we can split each vector equation into several equations for each component. Having done that we should have $\mathcal{N}$ equations where:

$$\mathcal{N} = 2 \times \# \text{ of dimensions} \times \# \text{ of particles} \tag{3}$$

The above findings will be very useful when using external libraries to solve the system of equations.

Note: in all of the equations it was assumed that the numerical value of G is 1*, so I am calculating everything in natural units. This would also mean, that the numerical value of c (speed of light) is equal to* 1 *as well*

3 The algorithm of the computer program

The exact algorithm which will be used here will be the Barnes-Hut algorithm together with 4th Runge-Kutta integration. Although using the RK4 method of integration might not be the fastest to get an answer, however, it is the fastest to get a reliable answer.

The whole algorithm can be outlined as follows:

1. Divide the whole space spanned by the particles into octants (or quadrants in 2-D case), such that there is only one particle in each segment.

2. Create a tree from all the octants and when done, calculate the interaction forces with all the particles.

3. If some cluster is far away, approximate it as one particle and do not traverse the tree anymore.

4. After having out the net force for the particle, do RK4 integration.

4 Requirements for the program

The following requirements need to be met in order for the software to be equivalent:

- Output various physical parameters of the system to different files, which would be all located in one directory and for one particle there would be a separate file.

- The physical constants are:
  - Eccentricity (e);
  - Kinetic energy (T);
  - Potential energy (V);
  - Total energy (E);
  - Possition in all coordinates;

- The initial conditions can be read from a single file which can be found in the same directory.

- The source code needs to be documented in a clear and concise way.

5 Implementation in Python

In this section I am going to talk more about the implementation in Python and how it should be done.

6 Implementation in Cython

7 Implementation in C/C++

8 Wishlist

This is the things I would like to do with my program:

- Incorporate general relativity.
- Find out how I can incorporate dark matter and dar energy. For gravitation I need only the dark matter. Is it true?
- Change the algorithm, so that the step change would be adaptive. Check whether it is feasable to do it when there are many particles.
- Can I find the viscosity of the fluid?