N-body simulations tutorial

by Eugene Vasiliev

Version of May 7, 2013

This tutorial is intended for students that don't have any prior experience with computer simulations and want to get familiar with state-of-the-art techniques of computer modelling in the field of gravitational dynamics.

Before discussing practical aspects of computer simulations, I'd like to name a few essential references to read:

- A short introduction to the gravitational N-body simulations is available on Scholarpedia: http://www.scholarpedia.org/article/N-body
- Binney & Tremaine, *Galactic dynamics* (2008) – a "must read" for everyone whose research is related to stellar dynamics. It's a huge book covering many aspects of contemporary theoretical studies of structure and evolution of stellar systems, and also includes a good introduction to numerical methods used in the simulations.
  http://books.google.ru/books?id=6mF4CKxlbLsC
- Heggie & Hut, *The gravitational million-body problem* (2003) is mostly devoted to the evolution of star clusters from various perspectives, both from theoretical point of view and with regard to modelling techniques. A few chapters are available at
  http://www.ids.ias.edu/~piet/act/astro/million/index.html
- A very interesting and creative project is *Art of computational science* by Makino & Hut: http://www.artcompsci.org/kali/development.html – it is a series of lectures, written in the form of informal dialogs, devoted to practical aspects of writing and running simulation tools, focused on "trial and error" way of achieving the goal, rather than giving the best approach right away. This project is not finished and hasn't been updated for quite a while, but even the existing chapters are exciting to read.
- A pretty good recent review article, covering all essential aspects of gravitational N-body simulations, by Dehnen & Read (2011) – perhaps the most balanced and not too long introduction at the beginner's level.
  http://arxiv.org/abs/1105.1082

Some students would jump into the world of gravitational N-body simulations by starting to write their own integrator. This is perhaps not the best way of doing serious research, at least not until you have spent plenty of time getting familiar with existing software tools and approaches. However, it may by difficult to navigate in the wealth of available codes and programs; here I'll give a short introduction to this topic.

Central to the N-body simulation is the program and hardware that integrates the coupled equations of motion for N particles. There are several methods for doing it with various degree of complexity and accuracy. Broadly speaking, there are a few principal approaches:

- Direct integration: force on each particle comes from summing up contributions from all other $N - 1$ particles; therefore, the cost scales as N^2. This is presently the only method that can efficiently deal with binary and multiple stars (by using regularization techniques). It is often coupled with special hardware (presently, GPU cards) to speed up computation. These codes are often used in star cluster dynamics or in other contexts where multiple stars and collisional (relaxation) effects are important, sometimes coming with modules for stellar evolution. Examples of this category include:
  - the famous `NBODYn` set of programs by Sverre Aarseth, where `n` ranges from 1 to 7 (these are not successive versions of the same program, but feature rather different algorithms). Link: http://www.ast.cam.ac.uk/~sverre/web/pages/nbody.htm
  - ϕGRAPE – an efficient hardware-accelerated direct N-body parallel code, or its version with regularization suitable for studying systems around supermassive black holes. http://www-astro.physik.tu-berlin.de/~harfst/index.php?pid=8 (also available as part of AMUSE project, see below)
  - `kira`, an N-body integrator which can handle binary and multiple stars, part of the `Starlab` toolbox (see below).
  - `HiGPUs`, a recent GPU-accelerated parallel code. http://astrowww.phys.uniroma1.it/dolcetta/HPC.html (also available as part of AMUSE).
- Tree-codes make use of the fact that to compute the force at a given position coming from a distant set of many particles, one may replaced them by a single extended massive point. All particles in the simulation are arranged in a hierarchical tree structure and the force evaluation walks this tree down to a given level of detail; the cost scales as $N \log N$ but this only becomes competitive with direct N-body methods at $N \gtrsim 10^3 - 10^4$. These codes are mainly used in collisionless simulations (i.e. galactic encounters, spiral structure formation, cosmology), and often are coupled with a SPH gas-dynamics solver. Examples:
  - `GADGET2`, the most well-known general-purpose code which is used in various contexts, from cosmological simulations down to galactic scales. Includes hydrodynamics module. Actually it uses a hybrid Tree-Mesh approach. http://www.mpa-garching.mpg.de/gadget/
  - `pkdgrav`, a tree-code for cosmological simulations. Hydro version is called `gasoline`. http://pkdgrav2.org/

  - gyrfalcON, a very fast (but not parallelized, unlike the previous two entries) code for galaxy dynamics. Part of the NEMO toolbox (see below).
  - bonsai, a GPU-accelerated tree code. http://castle.strw.leidenuniv.nl/software.html

- Grid-based codes compute gravitational force on a cartesian grid (most often, with possible adaptive refinement in denser regions). They are often used in cosmological simulations, but sometimes also in galactic-scale problems. They are naturally coupled to AMR (adaptive mesh refinement) hydrodynamic solvers.

  - Enzo is a multi-purpose flexible code for a variety of astrophysical problems, well documented and extendable. http://enzo-project.org/
  - RAMSES is a cosmological simulation code with a MHD module. http://www.itp.uzh.ch/~teyssier/Site/RAMSES.html
  - FLASH is a hydrodynamical code primarily targeted at modelling stellar collisions and includes thermodynamics and nuclear physics modules. http://www.flash.uchicago.edu/site/flashcode/

- Planetary dynamics requires very high accuracy, symplectic integration methods to avoid buildup of secular errors over many thousands of orbits.

  - mercury is a monolithic code for simulating planetary systems. http://www.arm.ac.uk/~jec/
  - SyMBA/Swifter is another planetary code using the mixed-variable symplectic (MVS) integrator. http://www.boulder.swri.edu/swifter/

However, the N-body integrator is just one part of the workflow; typically one also needs tools to create initial conditions, analyze and visualize the results of simulations, keep and exchange data in some common format, etc. There are several larger toolboxes that allow to build an entire simulation environment suitable for these purposes.

- NEMO is an almost 30-years-old suite of software tools to manage N-body and hydrodynamical simulations. It is designed in the UNIX toolchain style, that is, it has numerous utilities to perform simple tasks, which can be connected via input/output pipes to achieve more complex data processing needs. It includes an efficient tree-code integrator (gyrfalcON) and a nice visualization tool (glnemo2). http://bima.astro.umd.edu/nemo/

- Starlab – a toolbox with a similar architecture, targeted primary to star cluster dynamics (including stellar evolution and binary system treatment). http://www.sns.ias.edu/~starlab/

- `AMUSE` is a toolbox based on Python, with a uniform interface to a variety of "legacy" codes written in various languages, providing efficient data handling, conversion and visualization using high-level Python scripting. The distribution includes adapted versions of many stellar dynamical and stellar evolution codes. Primarily targeted for star cluster dynamics. http://www.amusecode.org/

1 Getting started with NEMO

1. Download and install NEMO toolbox (http://bima.astro.umd.edu/nemo/). For some reason, it doesn't work correctly in `bash`, use `csh/tcsh` as your shell instead. Sadly, the installation does not always proceed as planned: on some systems several crucial components (notably `gyrfalcON`) do not compile without errors; for the present, I don't have an universal solution except for trying a different version of Linux and/or compiler (perhaps it's better to experiment in a virtual machine).

2. There is extensive online documentation covering many individual programs and functions. Generally, programs may be invoked in a toolchain, piping input/output channels to perform successive operations on the data.

3. Programs usually accept a multitude of arguments using the syntax
   > `snapblahblah in=infile out=outfile param=xxx`
   A brief summary of parameters may be obtained by
   > `snapblahblah help=h`

4. The primary data exchange format in NEMO are snapshot files, which contain data from one or more moments of time; data usually include positions, velocities and masses of particles, but may also contain density, gravitational potential and other auxiliary information. A useful tool for looking at the content of snapshot file is
   > `tsf filename`
   It dumps the first few entries in each data array, which is useful to get the idea what kind of data are there. To export the selected data arrays to a text file, use
   > `snapprint filename tab=outputtext options=(list-of-parameters) [times=t1,t2,t3:t4]`
   Here `options` specifies the quantities to be printed (e.g. `t,x,y,z,vx,vy,vz,m,phi,dens`, etc.), `times` may filter out only certain time moments (for snapshot files which contain data at several moments of time).
   Each snapshot file also has a header which keeps track of all operations that were performed on the data. It can be accessed by
   > `hisf filename`

5. To create an N-body snapshot, one may simply write down positions, velocities and masses in a text file, then issue
   $>$ `tabtos in=infile.txt out=outfile.nemo block1=x,y,z,vx,vy,vz,m`
   which converts the text file into a snapshot file; block1 gives the order of data fields, one particle per line. Important: text file must have UNIX line endings (use `dos2unix` to convert CR/LF to LF).
   Exercise: create a text file with a few points, convert it to NEMO format, display the contents of the snapshot file by `tsf`.

6. A more clever way of creating snapshots is using `mkxxxx` series of programs, which generate equilibrium models for various density profiles. To create a 1000-body Plummer sphere, use
   $>$ `mkplummer out=file.out nbody=1000`

7. To visualize the snapshot file, use `glnemo2` program, which is an interactive OpenGL-based rendering tool. It can display snapshots at a given moment of time or in motion (we will feed it with movies shortly).

2 Running and analyzing a simple simulation

1. There are several N-body integrators included in NEMO. A good choice for collisionless simulations (those in which we are ignoring and suppressing close encounters) is `gyrfalcON`; for small-N systems to be evolved in exact gravity one may use `newton0`. Let's start with the latter.

2. N-body units: in many cases it is convenient to work in dimensionless units, setting the gravitational constant $G = 1$ and using the total mass of the system to be unity (the latter requirement is not necessary but just convenient).
   Exercise 2: Set up a Keplerian system composed of two stars, of masses 0.9 and 0.1, on a circular orbit around each other. Let the first body be located at origin and the second – at (1,0,0). Compute what velocities should each body have to be on a circular orbit. Create a text file and then a NEMO snapshot from it.
   To ensure that the system indeed has been set up properly, check the virial ratio *[what is it and what does it tell about the system?]* by the following command:
   $>$ `addgravity snapshot.nemo - eps=0 | snapvratio -`
   A few things are worth mentioning here. First, parameters `in=xx` and `out=..` are typically the first and the second parameters for a program, and as such, one may omit their names and just type in the values. Second, replacing a filename with a dash `-` means that we pass the data to standard output and in the second command take it from standard input: this is the UNIX pipeline in action. Third, vertical line `|` is just

the way to make the toolchain with piping.
The output of `snapvratio` should contain $2T/W$ in the second field in the line, and this value should be [close to] unity.

3. Now we may run the binary system for a few orbits to see it moving.
   > `newton0 in=binary.nemo out=binary_out.nemo t_end=100 dt_major=0.25 >/dev/null`
   Here we have run the simulation using the simple direct N-body integrator up to time=100 *[how many orbital periods is it?]*, making output each 0.25 time units. It dumps a lot of debug info to stderr, that's why I've redirected it to /dev/null (at first run you probably should not ignore the log messages, but later may safely discard them).

4. Take a look at the results in `glnemo2`. (There is an option to record and display orbits, try it!). A more physical way of checking the result is again by displaying the total energy and virial ratio as a function of time. To this end, type
   > `snapvratio binary_out.nemo`

5. Now let's create an eccentric binary system. One may re-write the text file input, or use one of many snapshot transformation tools available in NEMO. For instance:
   > `snapscale binary.nemo binary_ecc.nemo vscale=0.2`
   multiplies velocities by a factor 0.2, to make orbits highly eccentric. Make another run with these initial conditions and check the results. In my case, the integration accuracy wasn't sufficient and the system has instantly blown up, manifestly violating total energy conservation.

6. To remedy this, one should change the accuracy parameter `eta_acc`. What is the value at which the system is evolved more-or-less correctly? what is the behaviour of energy error with time and how does it depend on the accuracy parameter? what can you conclude about the order of numerical integrator?

7. A better way of dealing with such a system is to use regularized two-body integrator, `newton0reg`. Try it with various accuracy parameters. What is the scaling of error with η?

3 Exploring the future of the Solar System

1. The problem of the long-term evolution of our Solar System is a rather complicated one; in addition to a very high accuracy integrator using specialized symplectic methods, one needs to take into account other effects such as tidal interactions in the Earth-Moon system and the general relativity. Moreover, since almost any N-body system with $N \geq 3$ is chaotic, predictions may only be probabilistic and based on

an ensemble of possible evolution tracks. Here we instead imagine an unlikely but disastrous scenario that a nearby star had a close approach with our Sun, and what is the outcome for the planetary system.

2. To start with, create a model of eight planets orbiting the central star. You may find the ephemerides of all solar system bodies at http://ssd.jpl.nasa.gov/horizons.cgi – there is an option to give the output in cartesian coordinates, relative to the barycenter of Solar System. The output is in physical units (e.g. km/s or a.u./day), and we want the simulation to be run in dimensionless units, say, with the mass of the central object as unity, and the time unit of one year. *[what would be the scaling of distance unit in this system?]*. Create a system of nine points and check that planets do move on quiet, almost circular orbits.

3. Now let's play god and imagine that the planets have been formed more massive. (It's unlikely from astrophysical grounds but we'll ignore it). Multiply planetary masses (but not the mass of Sun) by a factor of 10,100,... and repeat the integration for 100–1000 years. At which point does the system become unstable? what happens to it?
   A useful physical criterion for the instability is that the Hill spheres of adjacent planets overlap in radius. The meaning of the Hill radius is that a third body within this radius feels more gravity from the planet than from the sun; for instance, in the case of Earth $r_{\rm Hill} \simeq 1.5\times10^6$km, while the radius of lunar orbit is ~ 4 times smaller, so the Moon is indeed orbiting Earth. Obviously, when the planets are massive enough to feel each other on a close approach, the system becomes unstable. Check whether the onset of instability corresponds to the overlap of Hill spheres, and for which planets.

4. Now let's put a second star, of $1\,M_\odot$, on an hyperbolic orbit which brings it into the planetary system. Let the star approach in z direction (perpendicularly to the ecliptic plane) with some velocity v at the impact parameter b (it is the shortest distance the star would pass from the Sun if its trajectory was a straight line). Obviously, in an actual system the distance of the closest approach will be smaller than b due to gravitational focusing. Write down the equation for $b_{\rm min}(v,b)$.
   Try running the simulation with various parameters v, b and see what remains from the planetary system. Which planets remain bound to Sun? how does it depend on the parameters of the encounter? Give a physical explanation to your findings.

4 Studying the galaxy mergers

1. We now go on to study a system with a somewhat larger number of bodies, but still not as large as in a real galaxy. In fact the problem

to study is that of a collisionless dynamics, and as such, the number of particles in the simulation should not be important for the global dynamics of the system, provided that it is sufficiently large (i.e. the convergence of results with increasing N is reached).
Namely, we are studying the accretion of a satellite galaxy or a globular cluster onto the primary galaxy, which results from the effect of dynamical friction (see wikipedia). We put this satellite on an orbit around the primary galaxy and watch how does this orbit decay, i.e. the satellite eventually merges with the primary. The main properties of dynamical friction is that the drag force is proportional to the mass of the moving object (i.e. the satellite) and the density of medium through which it is moving.

2. Let's start with a simple case of a "rigid" satellite, represented by just one indivisible particle, placed on a circular orbit around a spherical primary galaxy. First we need to create the snapshot with initial conditions for this simulation. The primary galaxy will be represented by a Plummer sphere with scale radius of 1 and total mass of 1, sampled by $N = 1000$ equal-mass particles. (10^3 is a good starting point, later we are going to study the dependence of the evolution on N). We create it by
```
> mkplum primary.nemo nbody=1000
```
or another program for the same purpose
```
> mkplummer primary.nemo nbody=1000 scale=1
```
The satellite will be just a point mass $m_{\rm sat}$ initially located at a distance $r = 4$ from the center (on x axis), with the velocity in the y direction that puts it on a circular orbit at this radius. Compute the necessary velocity $v_{\rm circ}$ from the known mass profile of the primary galaxy and the associated centrifugal acceleration at this radius. We create this "micro-snapshot" by
```
> echo 4,0,0,0,vcirc,0,msat | tabtos - satellite.nemo nbody=1
block1=x,y,z,vx,vy,vz,m
```
where you substitute the computed velocity for **vcirc** and the satellite mass for **msat** (let's start with $m_{\rm sat} = 0.1$).

3. The next step is to put together the two components to create the initial conditions. This is done by
```
> snapstack satellite.nemo primary.nemo - zerocm=t |
snapshift - initcond.nemo vshift=0,-vcm,0
```
Here the first command takes two snapshots and merges them into the third one, which is given by dash indicating that it's not written to a file, but passed as the input to the second command, which shifts the snapshot in velocity and writes it to the file `initcond.nemo`. This is necessary since we don't want our system to drift away from the origin because of a non-zero net momentum: the primary system has a zero total momentum, but the satellite does not. So you need to compute

the center-of-mass velocity $v_{\rm cm}$ (which will be in the y direction) and compensate for it by adding a negative velocity to all particles. (The coordinates of the center-of-mass are shifted to origin by the parameter `zerocm=t` of `snapstack`).

Actually even this would not achieve the goal, as our primary galaxy did not have exactly zero total momentum because of shot noise (the number of particles is rather small, and $N^{-1/2}$ fluctuations are non-negligible). Find the center-of-mass velocity of the stacked snapshot by running

```
> snapkinem initcond.nemo weight=m
```

and then compensate for it.

4. Now we are ready to start a simulation. We will use the fast tree-code N-body integrator `gyrfalcON` for that purpose. It has a multitude of parameters, of which four are necessary: the input/output file names, the gravitational softening length ϵ and the timestep τ. Let's discuss the choice of the latter two.

   Softening is necessary in collisionless simulations to prevent close encounters that would deflect individual particles too much from their trajectories in the smooth potential. For this purpose, we replace the Newton's gravitation law with the softened one in which the force between two bodies tends to zero at small separation (smaller than ϵ), instead of diverging as r^{-2}. There are various functional forms of softening, most commonly used (but not the optimal, and not the one used in gyrfalcON by default) is the Plummer softening in which the force is given by $r/(r^2+\epsilon^2)^{3/2}$. In effect, we may think of our system being composed not of point-mass particles, but of finite-size objects with radius $\sim \epsilon$.

   The amount of softening which is appropriate for a given simulation is quite often taken from some hand-waving arguments, such as the necessity of resolving some given spatial scale which should be larger than ϵ, or from the considerations of computer time limitations. However, there is always a formally optimal softening length for a given system, which minimizes the sum of random fluctuations of gravitational force from the discreteness of mass distrubution (i.e. the variance error), and the systematical error from altering the gravitational law. Roughly speaking, this optimum is achieved when the softening length is comparable to the mean inter-particle separation. We will use a single softening length for all particles, even though the mean separation is smaller in high-density regions. Thus our choice will be a compromise based on some average value for the entire system, which is just $n^{-1/3}$, with $n \sim N/a^3$ being the average number density of particles in the volume of characteristic size a (which is unity for our Plummer sphere; actually, a better-informed estimate would take into account that the central density of the Plummer profile is $\frac{3}{4\pi}$). Thus, for our 1000-body

system we take $\epsilon = 0.1$.
After determining the softening length, we may compute the time step which should be a small fraction of the time in which forces acting on a single particle may change substantially. To put it differently, if a particle is moving at the speed v, then after passing a distance $\gtrsim \epsilon$ the force from the surrounding configuration of neighbour particles may change significantly. Thus the timestep shoud satisfy $\tau \le \eta\epsilon/v_{\rm typ}$, where the typical velocity is limited by the escape velocity from the center of potential well ($v_{\rm esc} = \sqrt{-2\Phi(0)}$, with the potential at origin $\Phi(0) = -1$ for our Plummer sphere). The accuracy parameter η should be no larger than 0.5, or better perhaps $\lesssim 0.25$; a value too large may manifest itself in the substantial non-conservation of total energy, but unfortunately the conservation of energy conservation is only the necessary, not sufficient condition for the simulation to be valid. In collisionless simulation, a relative error in energy conservation of $10^{-4}..10^{-3}$ is usually tolerable. Getting back to our run, in `gyrfalcON` the timestep is specified by a parameter `kmax` such that $\tau = 2^{-\rm kmax}$. Compute the appropriate value of `kmax` for our simulation.
Now we are ready to start the simulation:
$>$ `gyrfalcON initcond.nemo run1.nemo eps=0.1 kmax=`*your-value-here* `step=0.25 tstop=100 logstep=16`
Other parameters that we specified are `step` – the time interval for output (*not* the integration timestep), `tstop` – the end time of simulation, and `logstep` – the number of timesteps between printing the diagnostic information to the console (otherwise it will print it each timestep, which is too frequently). This diagnostic information includes the total energy (in the second column), which is useful to test its conservation error, and the virial ratio in the 6th column. This run should take only a few dozen seconds, after which you may enjoy watching the movie of the simulation in `glnemo`:
$>$ `glnemo2 run1.nemo select=all com=f texture=f point=t`
You will probably not discern your satellite particle among the other ones, unless you switch on the display of its orbit on the “orbits” tab of the instrument palette.

5. Now we need a more quantitative way of analyzing the results of our run than just watching a movie. For this simple run, we may just print the coordinates of the first particle and analyze its time dependence:
$>$ `snapmask run1.nemo - select=0 | snapprint - options=t,r,x,y,z tab=run1sat.txt`
Now plotting the first two columns of this text file gives you the time evolution of the distance of the satellite to the center. You will notice that it decays first slowly, then accelerates until it drops to a small radius where the random oscillations about the origin keep going forever.

6. Repeat the simulation with a different mass of the satellite $m_{\rm sat}$ (say,

10 times smaller) and observe whether it takes correspondingly longer for the orbit to decay. Repeat the run with a different number of particles in the primary galaxy; does the result depend sensitively on this number?
Run another simulation in which the satellite was not on a circular orbit, but on an eccentric one (start not from the same radius, but from an orbit with the same semimajor axis as the circular orbit). How does the sinking time depend on the eccentricity? Can you give a quantitative explanation for your findings?

7. Now let's take the next challenge and consider a satellite that itself is a composite system of N_2 particles. Create this satellite snapshot by
   `> mkplum satellite2.nemo nbody=`N_2` mass=`**`msat`**` r_s=`**`rsat`**
   where $r_{\rm sat}$ is its scale radius. Let's now use $m_{\rm sat} = 0.1$, $N = 10^4$ particles for the primary and $N_2 = 10^3$ for the satellite (don't forget to adjust ϵ and τ accordingly), and vary the scale radius of the satellite r_s, which changes its concentration. Start with $r_s = 0.25$; since the mass of the satellite is 10x smaller and its volume is 0.25^3 of the primary, its density is substantially larger than that of the primary galaxy. Run the simulation for 100 time units and then watch the output by
   ```
   > glnemo2 run2.nemo select=0:999,1000:10999 com=f point=t texture=f
   ```
   This will display the particles of the primary and the satellite in different colors. Watch how the orbit of the satellite decays and whether it looses mass substantially by tidal stripping. (In the instrument panel of `glnemo`, you may switch off the display of the primary galaxy and see how the satellite orbits an empty space:).
   Now repeat the same run, but now setting the satellite's scale radius to 0.5, which makes it somewhat less dense than the primary. How do the results change? Can you define the moment when the satellite has reached the center of the primary in this run, or it did not survive until that time and was disrupted entirely?
   For a quantitative analysis of the satellite trajectory in this section we need some more sophisticated methods, because some of its particles are stripped off on the way, and we do not know in advance which ones will reside close to the center of the satellite (and even how to define this center). A practical method of finding the center may rely on locating the particle that traces the minimum of the gravitational potential, if we compute it using only the particles belonging to the satellite. To that end, we employ a rather sophisticated sequence of commands:
   ```
   > snapmask run2.nemo - select=0:999 | addgravity - - eps=0.05
   | snapsort - - rank=phi | snapmask - - select=0 | snapprint
   - options=t,r,x,y,z,phi,key tab=run2sat.txt
   ```
   The first command filters out only the satellite particles (the first 1000), the second one computes the self-gravity of this subset of particles (we

need to provide the value of softening length), the third sorts them in order of increase of the gravitational potential; then we select the first particle of this sorted list and print its coordinates to the text file. The last two columns of this text file give the potential and the index of the particle (you will see that this is almost never the same particle). Plot the trajectory of this central particle in $x - y$ plane (columns 3 and 4) to see that indeed it seems to trace the center, at least for the first of our runs. What about the second one when the satellite got disrupted? Can you now define the moment of disruption? Plot the evolution of central potential versus time; can you roughly estimate how much mass does the satellite retain by the given time?

Compare the runs with a point-mass and a composite satellite, in particular, the decay times. Give a qualitative explanation to your findings.