Raga reference

Eugene Vasiliev
Lebedev Physical Institute, Moscow, Russia
email: eugvas@lpi.ru

Version $0.99\beta_1$
November 10, 2014

Contents

1 Introduction

Raga[1] is a Monte Carlo code for dynamical evolution of self-gravitating non-spherical stellar systems. The method is described in [1]; here comes a more technical and practical guide.

The program is based on the *SMILE* software for orbit analysis and Schwarzschild modelling [2]. The main features are:

- Simulation of stellar systems with a much smaller number of particles N than the number of stars in the actual system;
- Representation of an arbitrary non-spherical potential with a basis-set or spline spherical-harmonic expansion (same as in SMILE), with the coefficients of expansion computed from particle trajectories;

[1] Relaxation in Any Geometry

- Two-body relaxation modelled by local (position-dependent) velocity diffusion coefficients (as in Spitzer's Monte Carlo formulation); the magnitude of relaxation can be adjusted to the actual number of stars in the target system and is not related to the number of particles in the simulation;
- Particle trajectories are computed independently and in parallel, using a high-accuracy adaptive-timestep integrator; the potential expansion and diffusion coefficients are updated at rather long intervals (possibly comprising many dynamical times, but much shorter than the relaxation time);
- Can model the effect of a central massive black hole (capture of low angular momentum stars);
- Includes a Fokker-Planck and orbit-averaged Monte Carlo codes for isotropic spherical systems (the latter is temporarily unavailable).

In the present version, *Raga* comes as a standalone program which takes an input N-body snapshot and evolves it forward in time, optionally storing N-body snapshots and other parameters at regular intervals during the evolution. In the future we plan to integrate it into the *AMUSE* framework [3].

Caution: due to ongoing development of both *SMILE* and *Raga*, the present version is different from the one used to obtain the results in the paper; moreover, some features may be messed up in the process of restructuring the code – in short, this is not a stable release. If you want to use it, better contact me first, or wait until it is integrated into *AMUSE*. This release was prepared by taking the relevant files from *SMILE* and throwing away pieces of code unrelated to *Raga*, this might also add to confusion.

2 Obtaining and compiling the software

Raga is available for download at `http://td.lpi.ru/~eugvas/raga/`. To compile, one needs the following additional libraries:

- GSL (C math library).
- interp2d (GSL-compatible 2d interpolation library).
- optional: Odeint library (now part of *boost*) – to allow more variants of ODE integrators (various Runge-Kutta methods and Bulirsch-Stoer). Without it the built-in 8th order Runge-Kutta is happily used. To include the support for Odeint, uncomment HAVE_ODEINT in the makefile.
- optional: UNSIO library – to enable support for *GADGET* and *NEMO* N-body snapshot formats; without it only the text format is available for input and *NEMO* for output. To use UNSIO, turn on the flag HAVE_UNSIO in the makefile.

Check and correct paths to various libraries and compilation flags in the `Makefile` file, then run `make`.

3 Structure of the algorithm

The simulation progresses in so-called episodes; the duration of each episode can be much longer than the dynamical time, but shorter than the relaxation time. The episode length is constant for all particles, but may change in time as the evolution leads to core collapse. Particle trajectories are computed independently from each other during each episode, using the smooth potential that is either initialized at the beginning of simulation and is kept constant, or is updated after each episode using new positions of particles. This orbit integration is done in parallel (using `openmp` approach) with a high-accuracy, adaptive-timestep ODE solver (standard is a 8th order Runge-Kutta method, but other choices are available). During the orbit integration, one or more "tasks" can be attached to each particle, that collect data and/or change properties of the orbit. After all particles have been processed, each task is performing its own "finalization" step, and the entire episode is repeated until the end of simulation time.

The available tasks are described below, in the same order that they are called in the simulation (the ordering matters).

- *Trajectory output*: store points at regular intervals of time. Finalization step just collects the sample points from each orbit at each output time and writes them to an N-body snapshot file.

- *Relaxation*: apply local (position-dependent) perturbations to particle velocity after each internal timestep of ODE solver. The perturbations are computed using the drift and diffusion coefficients calculated from a spherically symmetric isotropic distribution of background scatterers, which most closely approximates the true distribution function of test particles that are actually moving in the system. The amplitude of perturbation terms is scaled to a desired number of stars in the target system (not necessarily the number of particles in the simulation), see next section. Since the random perturbations are assigned in an uncorrelated way for each orbit, the total energy of the system is not conserved by this process. At the finalization step, the total change in energy accumulated for all particles is summed up and distributed among all particles to cancel out this imbalance.

- *Loss cone treatment*: record distance of closest approach to the origin; if it is smaller than the pre-defined capture radius, the particle is eliminated from the subsequent simulation and its mass is added to the black hole at the end of episode. This task stores the list of captured particles in two text files: `[outfile].captdata` contains the information about the particle – capture time, angular moment at capture and its change during the last orbit before capture (if it has not completed a single orbit, this value is -1), energy of the particle, and its index in the input file; `[outfile].captorb` contains the position and velocity of the particle at the moment of capture (it can be used for the purpose of orbit classification of the captured particles, by loading this file as an orbit library to *SMILE* and performing orbit library integration and analysis).

- *Binary black hole*: record changes in energy and angular momentum for each particle if it passes near a central binary black hole; at the finalization step, adjust the orbital parameters of the binary using conservation laws (work in progress).

- *Potential and distribution function update*: collect sampling points from each particle's orbit during the episode (in the simplest case, only one point at the end of episode, but more than one sampling point per particle is possible by setting the `numSamplesPerEpisode` parameter, see next section). At the finalization stage, recompute the potential (if `updatePotential` option is set) using all sampling points, then apply a correction to each particle's energy arising from the changed potential and an overall correction to cancel out any random change in the total system energy. If two-body relaxation is simulated, then also update the distribution function of stars in energy, which enters the calculation of diffusion coefficients, using the same sampling points as for the potential. If a filename in `fileOutputPotential` was provided, then it stores the potential and the spherical model used to compute diffusion coefficients at regular intervals of time into text files.

4 INI parameters

All parameters are set in the INI file. By default, the program attempts to read `raga.ini`; alternatively, the name of the INI file may be passed as a single command-line parameter. The file has the following structure (many parameters have the same meaning as in *SMILE* INI file):

`[Potential]`

These parameters are the same as in *SMILE*, but not all of them make sense for *Raga*. Here we describe only the relevant ones.

- `Type` – can be any type that *SMILE* supports, however, if one intends to update the potential during the evolution of the system, then it must be one of the following variants: BSE, Spline, Spherical. The first two are spherical-harmonic expansions suitable for any non-spherical potential, the third is restricted to spherical symmetry, but otherwise analogous to Spline potential.

- `DensityModel` – provides the actual density model from which the potential expansions are initialized. It does not need to be the same density profile as the input snapshot, but if potential is being updated in the course of evolution, then it makes sense to use the initial density model from the input snapshot itself. This is switched on by setting `DensityModel=Nbody`; if no `NbodyFile=` is provided then the input snapshot will be used. Alternatively, one may use any analytic mass model such as Dehnen or Plummer, or an arbitrary profile stored in a text file (when `DensityModel=Coefs` and `NbodyFile=<filename>`).

- `Ncoefs_radial` – determines the number of radial basis functions (in BSE) or points in radial direction (in Spline or Spherical expansions). In the latter two cases, one may also specify the extent of radial grid in [`splineRadiusMin`, `splineRadiusMax`]; by default they are determined from the input snapshot or from the analytic mass model.

- `Ncoefs_angular` – for BSE and Spline potential this sets the order of angular expansion (should be an even number, 0 means spherical symmetry).
- `Alpha` – for BSE potential this determines the shape of basis functions (1 corresponds to the Hernquist–Ostriker basis set and is a good default choice).
- `Mbh` – mass of a central black hole (0 by default). The black hole does not move but contributes to the total potential, and optionally may capture particles that pass within a given radius.
- other parameters are described in *SMILE* documentation, and make sense only if `DensityModel` refers to some analytical mass model.

The bottom line is that there are two possible modes of operation – with a pre-determined potential set at the beginning of simulation and never updated (in which case any analytical density model or a profile stored in a coefs text file is possible, and it does not need to correspond to the density profile of the input snapshot), or with dynamically updated potential which is expected to be self-consistent with the evolving N-body system (therefore, `DensityModel=Nbody` should be used at the beginning of simulation).

`[Orbit]`

The parameters of this section determine the type (`integratorType`) and accuracy (`accuracyAbsolute`, `accuracyRelative`) of orbit integrator, and most interestingly, the amplitude of velocity perturbation term that mimics the effect of two-body relaxation (`relaxationRate`). Its numerical value corresponds to $N^{-1} \ln \Lambda$ of the target system. In other words, if one wants to simulate a nuclear star cluster with $N = 10^8$ stars and a massive black hole of $M_\bullet = 10^6\, M_\odot$, then the value of Coulomb logarithm is usually determined by the number of stars within the influence radius of the black hole ($\ln \Lambda \simeq \ln M_\bullet/M_\odot \sim 15$), and one should set `relaxationRate=1.5e-7`. The crucial feature of the Monte Carlo algorithm is that the actual number of particles in the simulation may be far less than N, say, it could be only 10^5 and still the relaxation rate will be modelled correctly. (One should keep in mind that when using the option of dynamical update of the potential in the course of simulation, this introduces artificial numerical relaxation at an amplitude of 1–2 orders of magnitude lower than the inverse number of particles, so in the above example it would still overwhelm the "intrinsic" relaxation rate of a $N = 10^8$ star system).

`[Raga]`

- `mode` may be `FokkerPlanck` for the approach based on one-dimensional spherical isotropic Fokker-Planck equation, or `Raga` for the full-featured non-spherical Monte Carlo code. (The other two variants use Monte Carlo approach in spherical geometry with orbit-averaged diffusion coefficients, and are currently unavailable). The Fokker-Planck option is not yet covered in this documentation.
- `fileInput` provides the name of the input N-body snapshot file. It may be any of the formats supported by UNSIO library (e.g. *NEMO* or *GADGET*), or – even without this

library – a simple text file with 7 columns: 3 positions, 3 velocities, and mass of each particle (not including the central massive black hole). The same file is used to initialize the potential at the beginning of simulation, if `[Potential]/DensityModel=Nbody` (see above).

- `fileOutput` gives the file name of optional output N-body snapshot file (presently, in the *NEMO* format only).
- `fileOutputPotential` is the base file name for storing the potential coefficients (for BSE, Spline or Spherical potentials that are updated in the course of evolution). At each output time, a new file is created with the value of simulation time appended to the base file name.
- `fileOutputLosscone` is the file for storing the information about particles captured by the central black hole.
- `fileLog` is the file name for writing the diagnostic information (energy, etc.) after each episode.
- `timeTotal` is the total duration of the simulation.
- `timestepOutput` is the frequency of output of N-body snapshots and potential coefficients.
- `episodeLength` is the duration of one evolution episode (integration of the entire collection of orbits), see Section 3. Zero means that the entire simulation is done in one step.
- `numSamplesPerEpisode` defines how many sampling points from the orbit of each particle will be stored during each episode, for the purpose of re-computing the potential and distribution function at the end of episode.
- `updatePotential` (true/false) determines whether the simulation continues in a fixed potential (but with the distribution function still being updated after each episode) or using the self-consistent potential generated from particles themselves as they evolve.
- `captureRadius` – if nonzero and a central black hole is present, particles that pass within the given distance from the origin are captured by the black hole.
- `gridSize` (default 50) determines the number of bins used to compute distribution function.

References

[1] Vasiliev E., 2014, MNRAS, in press; arXiv:1411.1757

[2] Vasiliev E., 2013, MNRAS, 434, 3174

[3] Portegies Zwart, S., McMillan, S., van Elteren, E., Pelupessy, I., de Vries, N. 2013, Comp.Phys.Comm., 184, 456