SMILE reference

Eugene Vasiliev

Lebedev Physical Institute, Moscow, Russia

Rochester Instutute of Technology, Rochester, NY, USA

email: eugvas@lpi.ru

Version 2.0
Martober 33, 2012

Contents

1 Introduction

SMILE[1] is a software for orbital analysis and Schwarzschild modelling. The scientific reference paper is [1]; here comes a more technical and practical guide.

As the name suggests, the program is intended to study orbits and self-consistent Schwarzschild models in various potentials, and might also have educational purposes. The primary applications are:

- Exploring orbits in various potentials, either given by analytical formulae or several approximate expansions;
- Studying properties of orbits, from builtin orbit integrator or from external data;
- Constructing equilibrium models of triaxial stellar systems with given density profile by Schwarzschild method.

It is not intended for modelling observational data of any kind.

The program comes in two versions – GUI interactive tool (Section 2) and console program with scripting support (Section 3). The GUI version is more suited to "exploratory" and "educational" purposes, since it has many interactive connections between different modules allowing to easily visualize the results. The console one is more appropriate for remote and batch computations, when you know what you're doing.

There are numerous adjustable parameters which are kept in INI file (Section 4.1); most of them may be changed in the GUI, and described in the appropriate section of GUI reference; some are not modifiable from GUI and these will be described in the section about INI file.

The data structures used are split into several components:

- Potential model (potential type, geometric parameters, integration accuracy).
- Single orbit (initial conditions, trajectory calculated by integration of equations of motion in given potential, orbit analysis data – orbit class, chaoticity, etc.).
- Orbit library (parameters for creating initial conditions and the collection of orbits (no trajectories, only results of analysis)).

[1]Schwarzschild Modelling Interactive expLoratory Environment

- Schwarzschild model (grid parameters, orbit library, solution of the optimization problem).

2 GUI – interactive environment

The window is split into three areas – right panel contains the parameters of potential and orbit integration, left side contains one of several tabs depending on the current module (analysis of a single orbit, orbit library or Schwarzschild model); in the top are the parameters for current module, the rest is occupied by plot area (again depending on the selected task).

2.1 Right panel

Potential: Several potential types are implemented, having different subsets of parameters.

- Logarithmic: $\Phi(\tilde{r}) = \ln(R_c^2 + \tilde{r}^2)$;
- Anisotropic harmonic oscillator: $\Phi(\tilde{r}) = \tilde{r}^2$;
- Dehnen: $\rho(\tilde{r}) = \frac{3-\gamma}{4\pi pq}\tilde{r}^{-\gamma}(1+\tilde{r})^{-(4-\gamma)}$;
- Scale-free: $\rho(\tilde{r}) = \tilde{r}^{-\gamma}$;
- Basis-set expansion (BSE, see below);
- Spline expansion (see below)
- Frozen-N-body (see below).

Here $\tilde{r} = (x^2 + y^2/q^2 + z^2/p^2)^{1/2}$ is the elliptical radius. The potential parameters are:

- $q = y/x$ and $p = z/x$ – axis ratios, should be $p \le q \le 1$. Define axis ratio for potential (in case of logarithmic and harmonic) or density (in other cases).
- M_{bh} – mass of central black hole (point mass).
- γ (cusp exponent) – index of power-law density profile in the scale-free model or in the inner region of Dehnen model, should be $0 \le \gamma \le 2$.
- R_c (core radius) – in log.potential denotes region of constant-density core, may be zero.
- N_{radial} and $N_{angular}$ – order of basis-set and spline expansions.
- ϵ (softening length), θ (tree opening angle) – parameters for frozen-N-body tree-code
- Nbody file – holds coordinates of particles in frozen-Nbody or generic BSE/Spline potentials.

BSE and Spline are two general-purpose potential expansions which may be used to approximate almost any potential model. The coefficients of expansion are calculated either from an analytic density profile, or from a set of N point masses. The list of available density profiles includes:

- Dehnen (same as above);
- Plummer: $\rho(\tilde{r}) = \frac{3}{4\pi pq}(1+\tilde{r}^2)^{-5/2}$;
- Perfect ellipsoid: $\rho(\tilde{r}) = \frac{1}{\pi^2 pq}(1+\tilde{r}^2)^{-2}$;
- Isochrone: $\rho(\tilde{r}) = \frac{3}{4\pi pq}\frac{3(1+a)a^2-(1+3a)\tilde{r}^2}{a^3(1+a)^3}$, $a \equiv \sqrt(1+\tilde{r}^2)$;

Note about expanding this list: to add a new density model, one needs to:

1. Add new identifier to `CDensity::POTENTIALTYPE` enum;
2. Implement a new class derived from CDensity in `potential.cpp`, which provides the density profile in virtual function `Rho(x,y,z)`, and, optionally, power-law slope at $r \to 0$ in `getGamma()`; also it should return correct PotentialType and name;
3. Add item to DensityNames list in `initPotentialAndSymmetryNameMap()` in `potential.cpp`;
4. Add corresponding line to `CSmileCore::initPotential()` in `core.cpp` under the section PT_BSE, PT_SPLINE;
5. If the density model has additional parameters besides axis ratio, one needs also to modify `loadSettings()`/`saveSettings()` in `core.cpp` and `potentialChanged()` in `gui.cpp` to show/hide these parameters, and probably additional visual elements in GUI if one needs to change them interactively.

In both expansions, angular dependence of potential and density is represented in spherical harmonics with terms up to $l_{max} \equiv$`N_angular`, while radial dependence is either a sum of small number $N_{radial}+1$ of basis functions (in BSE) or a spline interpolation on a grid of N_{radial} points in radius (in Spline). For BSE, the parameter α is controlling the shape of radial basis functions; 0 means auto-detect, values between 1 and 2 are reasonable in most cases. Depending on assumed type of symmetry, only some terms in angular expansion may be used:

- None: all terms with $l \le l_{max}$, $-l \le m \le l$ are used;
- Reflection: terms with odd l are zero;
- Triaxial: no terms with odd m or $sin(m\phi)$ (implementation note: instead of $e^{im\phi}$ we use $cos(m\phi)$ for $m \ge 0$ and $sin(|m|\phi)$ for $m < 0$, so this type of symmetry implies that terms with $m < 0$ or $m = 1(\text{mod}2)$ are zero);
- Axisymmetric: use $m = 0$ only;
- Spherical: $l_{max} = 0$;

The symmetry type is adjustable when initializing the expansion fron an N-body file, otherwise it is assumed to be triaxial or higher, depending on axis ratios. If initializing from a set of point masses, the potential coefficients are written to a `filename.coef` file, which may be then used instead of original N-body file as an input to BSE/Spline initialization (sec. 4.4.5.

Frozen-N-body is a representation of potential of N particles fixed in space; Barnes&Hut tree-code is used for its computation, with tree opening angle θ (the less its value, the more accurate is tree approximation and the longer computation time) and softening length ϵ (may be even set to zero, since the integration uses adaptive timestep based on acceleration, but it is not recommended as it runs terribly slow and is not optimal in terms of bias/variance tradeoff. A better choice is spatially adaptive softening based on local density, which is selected by assigning a negative value to ϵ, so that the actual softening length is $|\epsilon|$ times local mean inter-particle distance).

In the case of generic BSE and N-body potentials, the file with particle coordinates should be supplied (by pressing the button *Nbody file* and selecting file, or typing the filename and pressing Enter). As the potential initialization may take a long time (especially for Spline), it is only done upon pressing the button "Init potential" or selecting the N-body file, rather than on every change of parameters as for other potentials.

Dimensions: 2d or 3d – switch regimes; 2d is basically for studying motion in principal planes and for Poincaré section, 3d is for real world.

Initial conditions: May specify either 3 coordinates and 3 velocities, or only energy (switch radiobuttons at left). The latter case is primarily used in construction of frequency map, while the former is for studying individual orbits. T_{orb} is the period of x-axis orbit, which is calculated automatically and used as unit of time in integration time and frequency analysis.

Remember that for scale-free and harmonic potential $E > 0$, for Dehnen and BSE/Spline/Nbody $E < 0$, for log it may be arbitrary.

Calc Lyapunov exponent: If turned on, one may look at the behaviour of deviation vector and finite-time Lyapunov exponent on the "Lyapunov" tab, and use its value in distinguishing regular and chaotic orbits. Slows down computation approximately twice. Not applicable for frozen-N-body potential (would give positive values anyway).

Integration time: given in units of T_{orb}; **steps per orbit** affect basically only orbit rendering, but if set too low, it may hinder to find higher-frequency spectral lines, so keep it at least ≥ 10.

Start buttons: **Start** just does what is does, starts an orbit with given initial conditions (also invoked by pressing Enter in most input lines); **Random** sets arbitrary IC with the same energy and starts orbit integration. The integration is performed in separate thread, so one may move around GUI during computation.

Part of orbit: Show i'th part out of N – if $N > 1$, split orbit into N equal intervals, and performs frequency analysis and rendering only for given interval.

Save settings on exit: if checked, saves INI file with most of settings, which are automatically retrieved upon launch.

Print: prints current figure in the left panel to PS or PDF file.

Results text box: this message area contains the results of last operation. For orbit integration – results of orbit classification: leading frequencies, orbit class, minimum distance to center, frequency diffusion rate, Lyapunov exponent (if checked), fractional conservation of energy (should tend to 0), wall-clock time for computation. For frequency analysis and Schwarzschild modelling – orbit population, solution of optimization problem, etc.

2.2 Left panel – tabs

2.2.1 Orbit

Orbit plot type: **2d projection** of an orbit onto one of principal planes (selected by **2d plane**);
3d line rendering of orbit: left mousebutton – rotation, Ctrl+left – move, mousewheel – zoom;
3d mesh rendering of orbit as a solid body (using Delaunay tesselation performed by an external program `qdelaunay`, in a separate thread – so it is available after some delay upon finishing of integration);

3d mesh parameters: Here one may choose to display entire orbit or just a half of it lying above one of principal planes, and also specify the maximal segment length of facets (if it was unlimited, any orbit would look like a convex blob; setting it too large will remove details, too small – create holes; it is automatically selected based on typical segment length of trajectory). To apply changes, press Refresh ([re]starts the tesselation thread).

Load/save orbit to a text file (Section 4.4.2).

2.2.2 Poincaré

Poincaré section is an useful tool for studying orbital structure of 2d systems. (It may be used in 3d, but is mostly meaningless). Needs to be turned on by corresponding checkbox.

Each orbit integration adds a series of points with a new color to the plot (a point of x, v_x coordinates corresponds to the passage of y axis with $v_y > 0$). Regular orbits have these points grouping in one-dimensional cycles; chaotic ones have scattered set of points in two-dimensional regions. Red outer curve marks the equipotential surface (where $v_y = 0$).

Plot may be zoomed in by left mousebutton, Ctrl-right zooms out, middle button moves. Right click within the equipotential boundary sets up the initial condition (x and v_x are taken from the plot, $y = 0$, and v_y is calculated from the given energy). (To integrate the orbit, press Enter thereafter).

Changing the energy clears the plot (as does the eponimous button). One may also export its content to a text file.

2.2.3 Frequencies

Displays spectra of orbit in three coordinates (blue – x, green – y, red – z), frequencies measured in units of inverse X-axis orbit period. Vertical lines show detected spectral lines (white – by precise Hunter method, black – by non-refined Carpintero&Aguilar method which is accurate to within Nyquist frequency; the latter is used when Hunter method produces diverges, typically for very nearby lines). Lines may be turned off by checkbox. Left mousebutton zooms, middle button moves, right click zooms out.

2.2.4 Lyapunov

Displays quantities used to estimate Lyapunov exponent of an orbit, in the case that it is calculated (then the evolution of deviation vector **w** is computed along with the orbit integration)

X axis is for time (in T_{orb} time units); left Y axis is for finite-time estimate of Lyapunov exponent $\Lambda = T_{orb} \ln(|\mathbf{w}|)/t$ (relative, i.e. normalized to unit frequency), in blue; right Y axis is for deviation vector divided by time, $|\mathbf{w}|/t$, in red; both axes are logarithmic.

For regular (part of) orbit Lyapunov exponent decreases as t^{-1}, and deviation vector grows linearly, so the red line is horizontal. When chaos starts to appear, Λ fluctuates around non-zero value, and **w** grows up. (See Fig. 3 in [1] for explanation).

2.2.5 Frequency map

Here one may study an ensemble of orbits by means of frequency map (and other tools).

Frequency map is typically built for given energy (specified in the right panel); however, one may also view orbit library from Schwarzschild model, in this case **View shell** spinbox selects the energy level from Schwarzschild grid (0 means display all orbits).

To create FM, one specifies the number of points in the start-spaces:
stationary (initial conditions on the equipotential surface with zero velocity),
principal-plane (on three principal planes),
$Y - \alpha$ (on y axis, with velocity perpendicular to it, as in Schwarzschild 1982),
random (yeah, anything you like! Ergodic within energy hypersurface).
Or, alternatively, one may use existing start space loaded from a file. In this case, setting 0 as the integration time (in the right panel) forces to use the values for each orbit written in the orbits file (otherwise they are overridden with the settings in GUI).

Start button starts the orbit integration in several parallel threads (their number being based on the number of processor cores). Once started, this button serves to terminate prematurely the threads (after each one finishes its current orbit).

Import/Export loads/stores data in the Orbit Library format (Section 4.2). The current configuration is kept along with the orbits file in the corresponding `.ini` file and automatically loaded during import.

The main area displays plots based on chosen radiobutton in the top-right array:

Frequency map: each orbit is represented by point which coordinates are ω_y/ω_x and ω_z/ω_x (for 3d) or simply ω_x and ω_x (for 2d), where the ωs are the leading frequencies in each coordinate. 3d map also is decorated with a dozen of most important lines representing resonant or thin orbits (most notably, $(0,1,-1)$ line *aka* LAT and $(1,-1,0)$ *aka* SAT).

Histogram: cumulative distribution function of either of two chaos indicators (see below).

Start-space (stationary, principal-plane, and $Y-\alpha$) – points from the corresponding start space are plotted in 2d projection.

The points in the plot are colored in blue (regular) or red (chaotic), based on the criteria in the **chaos criterion** section: an orbit is termed to be chaotic if it has either the frequency diffusion rate $\delta\omega$ larger than the threshold given, or if its Lyapunov exponent Λ is larger than the threshold (usually 0, since all orbits with positive exponents are chaotic; if it was not computed, this has no effect).

The coloring depends on only one of these two criteria (select appropriate radiobutton), but the labelling of an orbit as chaotic happens if either of them is satisfied. This labelling is important in Schwarzschild modelling, and in calculation of fraction of chaotic orbits.

View shell setting enables to filter out only orbits whose initial conditions lie in a particular shell in the Schwarzschild model (0 shows all orbits) [v2???].

Only nonzero weight checkbox filters out only the orbits with nonzero weights in Schwarzschild model (and for the cumulative distribution histograms, the weight of each orbit is also taken from the model). If in addition **view shell** is set nonzero, only the orbits that contribute density to the corresponding radial shell are shown [v2???].

In frequency map and spart-space chart one may right-click on the plot and select the nearest orbit, which set the initial conditions and displays some information about the orbit. Pressing Enter one may then re-integrate the orbit. Additionally, zoom, move and unzoom on frequency map is the same as in Poincaré plot.

2.2.6 Schwarzschild model

Orbit library tab specifies how many orbits are in the model (with randomly assigned initial conditions over the entire range of energies). Number of **sampling points** specifies how many points drawn randomly from each orbit trajectory will be stored in `.sam` file for subsequent creation of N-body initial conditions. If you do not plan to create N-body model, may set it to zero. Import/export *SM* buttons does the same as for Frequency map, but in addition `.ocw` and `.sam` binary files are stored (Sec. 4.3). **Start** button initializes *SM*, creates initial conditions (if **use existing start-space** is unchecked) and begins orbit integration in parallel threads.

Create *N*-body initial conditions button generates a N-body representation of Schwarzschild model, where particles are drawn from sampled points, their number proportional to orbit weight. If there are insufficient points for a particular orbit (that is, if its weight w is greater than $N_{\text{sampling points}}/N_{\text{bodies}}$), this orbit is set to be reintegrated for the same time interval, but collecting more sampling points. The reintegration is supposed to recreate the same orbit, but sometimes it may turn out to be different (e.g. if using Lyapunov exponent calculation, with initial deviation vector generated randomly), so it's best to avoid such situation. On average, one needs to have at least $10\,N_{\rm bodies}/N_{\rm orbits}$ sampling points per orbit.

There is an option to create N-body model with unequal mass particles ("mass refinement"), so that the innermost particles are lighter. If the refinement factor $Rf > 0$, the orbits are sorted in energy and binned into $Rf+1$ bins, yielding approximately the same number (not mass) of particles each. Particle masses in each bin differ by a factor of two.

The N-body model is exported in *NEMO* snapshot format (using the `.nb` extension), and to text file containing set of point masses (sec. 4.4.1). Then some statistical quantities are calculated, including virial ratio.

Model parameters tab contains the choice of *SM* variant, which should be made prior to starting orbit library integration itself:

- **Classic** Schwarzschild model: partitioning configuration space into a number of cells, compute fraction of time each orbit spends in each cell. Parameters: **Number of shells** in radial direction, number of **lines** splitting each of three segments of each shell (so the number of cells in a shell is $3n_{lines}^2$).

- **SHgrid**: evaluating coefficients of spherical-harmonic expansion of the potential at a number of radial grid points (conceptually similar to Spline expansion, except that no splines are constructed, just the values at grid nodes are used). Parameters: number of radial shells, number of **angular coefs** ($= l_{max}$, so that the number of terms is $(l_{max}/2+1)(l_{max}/2+2)/2$ since only terms with even l are used).

- **BSE**: evaluating coefficients of BSE expansion of every orbit. Parameters: number of angular coefs (same as above), radial coefs (equivalent to n_{max} in BSE), α parameter of BSE (may set 0 for auto-detect).

Note that the *SM* variant chosen here doesn't need to be related to the potential used in orbit integration. However, if they are related (SHgrid *SM* and Spline potential, or BSE *SM* and BSE potential with the same value of α), this is used to speed up initialization of model coefficients (often quite substantially).

Regardless of the variant chosen, there is still a radial grid used for recording kinematic data (that is, radial and tangential velocity dispersion of each orbit in each radial shell), which may be used in optimization to constrain velocity anisotropy. For the first two methods, this radial grid coincides with the partitioning of configuration space for the model itself; for BSE it is not related to the coefficients.

Inner and outer shell mass specify which fraction of total model mass (which should be finite!) is contained in the innermost and outermose radial grid node; 0 makes the default choice of $M_{outer} = 1 - 1/(N_{shell}+1)$, $M_{inner} = M_{outer}/N_{shell}$. If the requested inner shell mass is smaller than $1/N_{shell}$ of the total mass of *SM* (which is M_{outer} times the total mass of this density model), then a non-uniform, exponentially spaced grid in shell masses is constructed.

Export button creates a text file with statistics about the solution (after it was obtained). File format is described in Sec. 4.4.3.

Optimization tab contains several parameters controlling the solution of optimization problem. **Chaotic weight factor** enhances (if > 0) or reduces (if < 0) contribution from regular orbits in the solution; its magnitude is in principle unlimited, but if set too high, the cost of adding an unwanted orbit will overweight the cost of cell mass violation. Keeping its absolute value ≤ 1 is typically enough. 0 produces so-called "unconstrained" model.

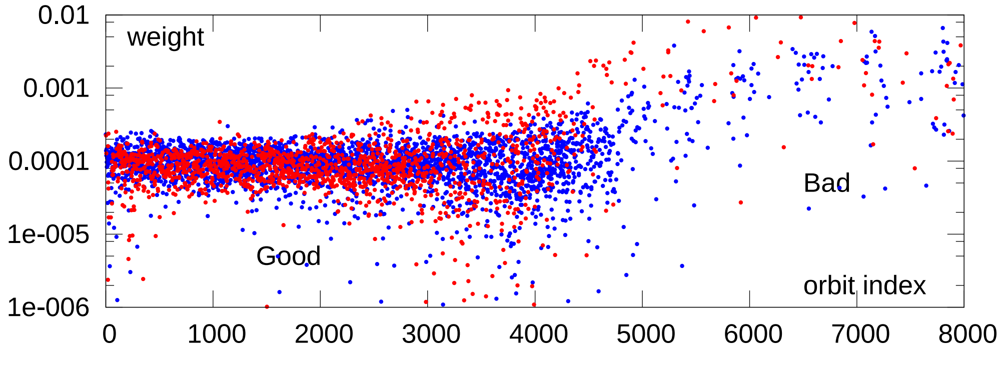
Maximal orbit weight may be used to limit the weight assigned to each orbit by the solver (setting it too low will, of course, result in infeasibility of solution, so it only makes sense to adjust it in the case that a few clear outliers are seen, particularly in linear optimization). 0 turns off this constraint.

Constrain anisotropy option adds constraints to optimization problem, forcing the average velocity anisotropy coefficient β in each shell to equal a predefined value. Its value is linearly interpolated with the enclosed mass from β_{in} in the first shell to β_{out} in the last shell.

There are currently ways of solving the optimization problem – linear and quadratic programming using interior-point method. Either GLPK library is used (for linear programming only), or BPMPD, external solver available from Cs.Mészáros); the latter has additional terms in the cost function which make orbit weight distribution flatter, and runs typically much faster, especially on large problems.

When the solver has processed the optimization problem, its results are displayed in info box in the right panel, including the number of infeasible cells (if the problem cannot be solved).

View options switch between various plots:
Grid cell displays the model coefficients (blue for feasible, red for infeasible, for which the difference from the required value is $> 1\%$). The meaning of coefficients depends on the variant of *SM*. For classic SM, it is mass in corresponding spatial cell, which are displayed in a projection on the 2d plane similarly to stationary start-space points (for each radial shell separately, or together). For SHgrid and BSE variants, these are coefficients in expansion, arranged in 2d array so that each line shows spherical-harmonic coefficients for given radius (in SHgrid) or index of radial basis function (for BSE); they are further separated in groups of $1, 2, 3, ...$ coefs for $l = 0, 2, 4, ...$, each group having $l/2 + 1$ coefficients for $m = 0, 2, ..l$. Right-click on a cell displays some information in the message area.
Anisotropy shows the value of β for each radial shell (horizontal axis is the shell number).
Orbit weights are shown with the horizontal axis being the orbit number; regular and chaotic orbits are colored blue and red. Right-click on a point does the same as in frequency map plot.



Distribution of orbit weights should be ideally close to uniform; in the example above, the innermost orbit weights are distributed quite well (many orbits with nonzero weights), while the outermost parts of model are overconstrained, which results in quite a few orbits with large weight. This situation may be remedied by using more orbits for the same number of constraints.
Weight histogram displays these weights as a cumulative distribution function. The flatter is it, the better...

3 Console scripting

The console variant may perform the same set of operations either using interactive commands entered in the command prompt, or feeding them as a script from a text file (by running `smilec script_file`). Below follows the command reference (spelling is case-insensitive).

`Exit`. Obvious.

`ReadIni("file.ini")` normally should be the first command in a session (unless one is satisfied with default parameters loaded from `smile.ini`).

`ImportOrbits("orbitsfile")` loads orbit library from text file; `ExportOrbits("orbitsfile")` stores orbit library (after building frequency map, etc.). Equivalent to the **Import/Export** buttons on the *Frequency map* page, but unlike GUI version, this does not automatically load or store configuration in accompanying `orbitsfile.ini` file!

`ImportOrbitsCell("orbitsfile")`, `ExportOrbitsCell("orbitsfile")` – same as above, but also load/stores binary files `orbitsfile.ocw` (with cell occupation times and velocity dispersion data) and `orbitsfile.sam` (with sampling points from trajectory), see Section 4.3. Equivalent to **Import/Export** buttons on *Schwarzschild* page.

`ExportPotential("potentialfile")` creates a text file with potential data described in Section 4.4.4. If an orbit library was loaded, the potential/forces/density is also sampled at the location of points of the library and stored in a separate file `<potentialfile>.points`, and if the potential is BSE or Spline, its coefficients are stored in `<potentialfile>.coefs`

`ExportGrid("orbitsfile.grid")` creates a text file with statistics about Schwarzschild model grid (same as eponymous button on *Schwarzschild* page), see Section 4.4.3.

`ExportNbody(NumberOfBodies, "nbodyfile"[, RefineFactor])` creates the N-body representation of model, saving a binary *NEMO* snapshot `nbodyfile.nb`, and optionally text files with list of particles in the form of orbit library `nbodyfile`(if the ini flag `exportNbodyOrbitsFile` is set).

`BuildFreqMap` creates an orbit library with the number of points specified in the `Frequency_map` section of INI file, for the energy given in the `Orbit` section. `BuildFreqMapExist` integrates the orbits with the initial conditions loaded earlier from an orbit library file.

`SchwBuildOrbitLibrary` and `SchwBuildOrbitLibraryExist` do the same except that first a model instance is created (with the grid parameters specified in the `Schwarzschild_model` section of INI file), and the cell occupation numbers, velocity dispersion and sampling points are also recorded. They later can be saved by

`ExportOrbitsCell` command.

`SchwLinearOptimization`, `SchwQuadraticOptimization`, `SchwLucyOptimization` start the corresponding solver routine, after the model has been loaded by `ImportOrbitsCell` or created by `SchwBuildOrbitLibrary`.

4 File formats

Non-bulky data is kept in text files (with tab- or space-separated values). The most important is the orbit library file, which contains initial conditions and results of analysis for a set of orbits. It is accompanied by a configuration (INI) file which contains all the necessary information about this orbit library (potential, integration parameters, chaos criteria, etc.). Other text files are mainly for export purposes.

4.1 INI parameters

Here is the list of all options and parameters in the configuration file `smile.ini` (and their default values).

`[Potential]` – potential properties and integration accuracy.

`Type` – variants: Logarithmic, Harmonic, Dehnen, Scale-free, Scale-free SH, BSE, Spline, N-body.

`Symmetry` – None, Reflection, Triaxial, Axisymmetric, Spherical.

`DensityModel` – for BSE/Spline potentials, this is the underlying density model used to compute coefficients. Variants: Dehnen, Plummer, Perfect Ellipsoid, Isochrone, Nbody. The last option means that coefs are calculated from a set of point masses given in `NbodyFile`.

`NbodyFile` – file with set of point masses (sec. 4.4.1) used to initialize treecode N-body potential or BSE/Spline with DensityModel=Nbody.

`N_dim (3)` – 2 or 3.

`q, p` – y/x and z/x axis ratio, must be $p \le q \le 1$.

`Mbh (0)` – central point mass (supermassive black hole).

`Rc (0)` – core radius of logarithmic potential.

`Gamma (1)` – index of power-law cusp for Dehnen and scale-free potentials.

`Alpha (0)` – shape parameter in BSE potential; 0 means auto-detect, allowed range is $0.5 - \infty$, preferred values are $1 - 2$.

`Ncoefs_radial, Ncoefs_angular (10, 6)` – number of radial and angular terms in BSE, Spline and Scale-free SH potential expansions (for analytic density models which have at least triaxial symmetry, the number of angular coefs l_{max} is even; 0 means spherical model only).

`treecodeEps (-2)` – ϵ, softening length used in frozen-N-body integration. Negative means adaptive softening based on local interparticle distance.

`treecodeTheta (0.5)` – tree opening angle for N-body potential.

`accuracyRelative, accuracyAbsolute (1e-10)` – integrator accuracy parameters for 7/8 order Runge-Kutta (all potentials except N-body).

`accuracyTreeCode (0.25)` – η, factor in timestep selection for leap-frog integrator used for integration in N-body tree-code potential. The timestep is taken to be $\tau =$

$\eta \times \min(l/v, \sqrt{l/a})$, where l – distance to nearest particle, softened (i.e. it is $\sqrt{l_{\text{true}}^2 + \epsilon^2}$), v – particle velocity, a – acceleration.

`[Orbit]` – single orbit integration properties.

`x, y, z, vx, vy, vz` – initial conditions (IC).

`E` – IC energy.

`useE (false)` – whether to use energy or coordinates (in the former case, x is initialized to be long-axis radius for given E. Makes sense for building frequency map at given E).

`intTime (100)` – integration time in units of long-axis period (T_{orb}). Zero value means using per-orbit data stored in orbit library file, when integrating orbit library with pre-loaded initial conditions.

`intTimeStep (50)` – output timesteps per T_{orb}: determines the maximum orbit frequency which can be detected, and the smoothness of rendered orbit. Has nothing to do with integration accuracy.

`calcLyapunov (false)` – whether to compute Lyapunov exponent.

`usePS (false)` – whether to use Poincaré surface of section (makes sense only in 2d).

`intTimeMax (0)` – maximum integration time (in T_{orb}) if Pfenniger's adaptive method is used (only in the context of Schwarzschild modelling; 0 to disable).

`adaptiveTimeThreshold (0.05)` – controls the Pfenniger's method: if difference in cell occupation times between two halves of integration time exceeds this threshold, continue integration further until this difference becomes less or the maximum integration time is reached.

`treecodeSymmetrizeTimestep (false)` – whether to use Hut et al.(1995) algorithm for time-symmetrizing adaptive timestep, to improve energy conservation at the expense of almost twice as slower integration.

`[Frequency_map]`

`numOrbitsStationary, numOrbitsPrincipalPlane, numOrbitsYalpha, numOrbitsRandom` – number of points in corresponding start spaces.

`[Schwarzschild_model]`

`modelType` – Classic, BSE, SHgrid.

`numOrbitsRandom (10000)` – number of orbits in the entire model.

`numShells (25)` – number of radial shells (used to store velocity dispersion information in all variants of *SM*, and to partition configuration space in Classic SM).

`linesPerSegment (4)` – in Classic *SM*, split each shell into $3 \times (\text{linesPerSegment})^2$ cells.

`numRadialCoefs (25)` – in BSE and SHgrid *SM*, number of radial coefs or radial grid points.

`numAngularCoefs (6)` – in BSE and SHgrid *SM*, number of angular coefs (should be even).

`chaoticMinFreqDiff (1e-3)` – threshold in frequency diffusion rate ($\Delta\omega$) separating regular from chaotic orbits. Used to plot them in different colors on the frequency map, and to increase or reduce fraction of chaotic orbits in Schwarzschild model.

`chaoticMinLambda (0)` – same threshold in Lyapunov exponent (Λ). If it is not calculated, this has no effect; if it is, then the default value of 0 just separates orbits with detected signs of chaos ($\Lambda > 0$) from those for which chaotic behaviour was not detected (the latter are assigned $\Lambda = 0$).

`chaoticWeightFactor (0)` – if positive, penalize usage of chaotic orbits; negative – prefer them. May take a continuous spectrum of values, although most of the effect is felt when this factor is of order ±1. For larger values, the penalty for wrong type of orbits may outweight that of violating cell mass constraints, so the model becomes infeasible.

`maxWeight (0)` – max.weight of a single orbit. 0 means no restriction.

`constrainBeta (false)` – whether to constrain velocity anisotropy coefficient β in the solution.

`betaIn (0), betaOut (0.5)` – values of β for the inner and the outer radial shells (linearly interpolated in enclosed mass between these two values).

`numSamplingPoints (1000)` – number of sampling points from each orbit that are stored in `.sam` file. They are drawn randomly from the trajectory after orbit integration is finished, avoiding duplicates if possible (i.e. if total number of points in trajectory, `intTime*intTimeStep`, is larger than `numSamplingPoints`). The sampling points are used in creating N-body model from Schwarzschild model, so if this feature is not used, one may set this number to zero.

`useBPMPD (true)` – whether to use external solver `bpmpd.exe` for linear/quadratic optimization problem. It is a lot faster on large problems than GLPK, and may handle quadratic problems (preferred mode), but the publicly available version is limited to small problems (approx.250 orbits). You may ask the author, Csaba Mészáros, for the unrestricted version (as did I:-). If this option is turned off, or if bpmpd.exe executable is not present in the application dir, then GLPK library is used as the solver.

`[Common] / WorkDir` – default working directory name.

4.2 Orbit library file

This is the main exchange format used for keeping orbit initial conditions and integration/classification results. This file type is also used to export data to N-body model (if this is requested in addition to creation of binary *NEMO* snapshot file), and to load particles representing N-body or BSE potential (in this case only 7 first fields are used). Each line contains the following data:

```
x y z vx vy vz weight timeunit timestep inttime maxtime ...
    ... energy ediff lfx lfy lfz lfdiff lambda description ...
    ... inertx inerty inertz Lxavg Lxvar Lyavg Lyvar Lzavg Lzvar...
    ... L2min L2slope fitscatter fitsignificance L2circ
```

First 7 fields are orbit initial conditions and mass (which means orbit weight in Schwarzschild model, so it may be zero), in the same order as in the simple N-body interchange file (sec. 4.4.1). This is, generally speaking, sufficient to load any text file containing this data as initial conditions for orbit library, although if no `timeunit` is provided, it will be calculated on-the-fly in the corresponding potential, which takes some time if the number of orbits is large. For 2d orbits z and v_z are zero. The other fields are either directly derived from initial conditions or are results of orbit integration and analysis.

`timeunit` is the dynamical time (period of long-axis orbit with the same energy) which serves as the unit of time and frequency for this orbit (same as T_{orb} in GUI); `timestep` is the timestep of trajectory output.

`inttime` is the integration time (in common time units, not in periods), and `maxtime` is upper limit on adaptive integration time (!temporarily defunct!).

`energy` is (initial) total orbit energy and `ediff` is energy conservation error.
`lfx`, `lfy` and `lfz` are the leading frequencies in three coordinates, and `lfdiff` is the Frequency Diffusion coefficient.
`lambda` is the Lyapunov exponent (if it was calculated, otherwise -1).
`description` is the text string containing orbit class and possibly "chaotic" attribute (based on analysis of spectrum, not a reliable estimate, see Section 6). This text line has underscores instead of spaces, in accordance with the requirement that any space- or tab-separated file may be loaded.
`inert{x/y/z}` are the square roots of diagonal components of inertia tensor, basically these quantities measure the extent of an orbit in each direction (average, not maximal).
`L*avg` and `L*var` are average value and mean-square scatter of angular momentum about each axis.
`L2min` and `L2slope` are coefficients in the linear regression fitting the distribution of squared angular momenta at pericenter passages. If `L2min=0` this means that the orbit is centrophilic. The next two parameters assess the quality of fit: `fitscatter` should be $\lesssim 0.5$ for the linear regression to be reliable, and `fitsignificance` measures how far from zero (in std.dev.) is the fit intercept (L2min), the latter is assumed to be zero if it is negative or within 3 std.dev. from zero. `L2circ` is an approximate squared angular momentum of a circular orbit with this energy (useful scaling parameter for L2min).

The orbit data make sense only in conjunction with corresponding potential, so each orbit library file is accompanied by INI file. Upon import, first INI file (if exists) is read, the potential is initialized, then the orbit library is loaded. Exporting orbit library also creates INI file. (This occurs only in the GUI version; in the console one has to do it manually).

4.3 Binary files

There are two kinds of binary files used in Schwarzschild modelling module.

One contains the data for *SM* (its content depends on the variant of *SM* chosen). For classic *SM*, fraction of time each orbit spent in each cell (orbit-cell-weight, `.ocw`), and radial and tangential velocity dispersion of each orbit in each shell, are stored. Contains $n_{shell} \times n_{lines}^2 + 1$ single-precision float values for each orbit: first the weight array, then radial velocity, then tangential velocity. Number of cells in the model is the grid size, +1 comes from the "infinity" cell, containing the rest of space except finite grid (it is not used in modelling, but the data is recorded). This file is necessary to load along with orbit library file when importing Schwarzschild model. [**v2: obsolete!**]

The other file contains sample points from trajectory of each orbit (`.sam`). Each orbit is stored as 4-byte integer holding the number of sample points in trajectory, then 6 arrays of float with that length, for each of coordinates and velocities. (The number of sampling points per orbit may differ for different orbits). This file is not mandatory, its data is used only when exporting to N-body model. If the number of existing points is not sufficient to sample a high-weight orbit, it will be reintegrated during export, creating more sampling points.

4.4 Auxiliary text files

4.4.1 Point file

This is a simple text file for loading/storing point mass sets: each line contains
`x y z vx vy vz m`

It is used as an input data for initializing BSE/Spline/treecode N-body potentials, for exporting N-body model in a text format, and also may be used to load initial conditions for orbit library / Schwarzschild model, since it is a shortened version of orbitlib file (sec. 4.2).

4.4.2 Orbit file

This file type is used to export or import trajectory; each line contains the following data:
`time x y z vx vy vz`

Upon import of such file, the orbit is assigned initial conditions from the first line, and all the relevant parameters (energy, dynamical time T_{orb} and unit of frequency) are calculated from the current potential parameters (which, however, are not stored along with the orbit).

May be useful to import an orbit recorded from N-body simulation, with potential expressed as frozen-N-body or BSE taken from density profile from the same simulation, and check how does this orbit look like if re-integrated in a fixed potential.

4.4.3 Grid file

Used for export only, this file contains information about Schwarzschild model grid (and hence can be created only when a model is created or loaded). Each line contains data for each cell of a model, then at the end, after an empty line, come average values for each shell, one per line. [**v2: obsolete!**]
`x y z shell Eshell rshell Torbx M dM/M NOtotal NOused sigmaR sigmaT beta`
`x, y, z` are the coordinates of cell center;
`shell` is the shell number;
`rshell` is the long-axis radius of this shell;
`Eshell` is the shell energy (potential at this radius);
`Torbx` is the period of long-axis orbit for this energy;
`M` is the cell/shell mass;
`dM/M` is the relative mass difference (should be zero for a feasible model);
`NOtotal` and `NOused` are the number of orbits that pass through this cell and the number of such orbits that are assigned nonzero weight in the model;
`sigmaR`, `sigmaT` and `beta` are the mean-square radial and tangential velocity and the velocity anisotropy coefficient;
`chaoticfrac` is the fraction of cell mass contained in chaotic orbits.
In addition, for the lines describing shells, the orbit population is also printed (five most important orbit families).

4.4.4 Potential sampling file

Used for export only. Each line contains potential (Phi), density (Rho) and forces (F) sampled at a given point.

`x y z Phi Fx Fy Fz Rho`

Points are logarithmically spaced in radius from 0.001 to 1000 and lie along seven lines: principal axes, diagonals of principal planes ($z = 0, y = x$, etc.) and the diagonal $x = y = z$. (NB: if an orbit library was non-empty, `ExportPotential("...")` also creates another file in the same format, but containing potential/force/density sampled at locations of points in orbit library). This data may be useful, for example, to test the accuracy of BSE/Spline approximation.

4.4.5 Potential coefficients file

Stores coefficients for BSE, Spline and scale-free SH potentials. This file is automatically created when a potential is initialized from a point mass set, and later may be used to load the same potential without spending time on computing the coefs. The first few lines are the header:
`BSEcoefs/SHEcoefs` – specifies potential type (BSE or Spline);
`n_radial` – number of radial coefs or radial grid points, correspondingly; `n_angular` – l_{max}, order of angular spherical-harmonic expansion; 0 means just one coefficient for a spherically symmetric model;
`alpha` – BSE parameter (unused in Spline);
`time` – unused;
`#commented out` line (text header for the table below).
The rest of file is the coefficients table, all angular coefficients for a given radial index (BSE) or radius (Spline) are written in one line. First number is radial coefficient index or radius, second is the $l = 0$ coef (spherical part), and so on. If the number of fields in a line is less than $1 + (n_{angular} + 1)^2$, the rest is filled with zeroes; if it is greater then the rest is ignored (so one may adjust the numbers in header without changing the table).

5 Program structure and compilation

The program is written in `C++` using the `Qt` framework (for GUI and for other benefits like inter-object and inter-thread communication), so it should compile wherever `Qt` is supported (at least Linux, Windows, and MacOS).

In addition, it uses a number of other libraries and software: GSL (GNU scientific library), optionally GLPK (GNU linear programming kit, if this option was selected as the optimization routine) or BPMPD solver (as a standalone application). GUI version needs Qwt (version 5.x, not 6) and QwtPlot3d[2] libraries for plotting, and optionally `qdelaunay` program from QHull package (to render an orbit as a solid body).

Build is typical for Qt applications – check the project files `smile.pro` and `smilec.pro` (they are separate for GUI and console executable) for correct paths to libraries (`INCLUDEPATH`, `LIBS`), run `qmake` (from qt4!), then `make`.
On Mac, one might need to run `qmake -spec macx-g++` to use GNU compiler.
`qdelaunay` (compiled separately) and `bpmpd.exe` should reside in the main application folder (the latter is windows-only binary, so it is run using `wine` in Linux/MacOS).

Source files contain the following sections and classes:

[2]On some systems, it may be called qwtplot3d-qt4

- `common.h / common.cpp` – most numerical constants, configuration data, and definition of several abstract classes in the hierarchy.
  - *CBasicSchwModel* – basic class for variants of Schwarzschild model;
  - *CBasicOptimizationSolver* – parent class for solving linear/quadratic optimization problem by various methods;
  - *CBasicOrbitRuntimeFnc* – parent class for various functions which are called at regular intervals during orbit integration (e.g. to store coordinates);
  - *CBasicOrbitRuntimeFncCreator* – utility classes that create instances of corresponding *CBasicOrbitRuntimeFnc*'s for each orbit in the orbit library;
  - *CBasicInformation* – parent class for storing arbitrary information about an orbit, e.g. results of orbit analysis, or data for Schwarzschild modelling, and also for loading/storing this information from data files.
  - *CBasicOrbitFilteringFnc* – parent class for various orbit evaluation functions, which return a float number based on some properties of an orbit.
  - *CPosPoint*, *CPosVelPoint* – aggregate templated classes for three or six floating-point numbers (double or float);
  - *COrbitInitData* – in addition to six phase-space coordinates, also stores pointer to potential, values of timestep and timeunit, and flag for computing Lyapunov exponent.
- `potential.cpp` – class hierarchy for potential models: each model implements functions for density, potential and force evaluation.
  - *CDensity* – basic (abstract) class for a density model without a known corresponding potential. Provides virtual function for evaluation of density, and several utility functions like computing mass within a given raduis, or inner power-law density slope. Derived classes include *CPotential* and
    - *CDensityPlummer*;
    - *CDensityPerfectEllipsoid*;
    - *CDensityIsochrone*;
  - *CPotential* – basic class (derived from *CDensity*) for models with a corresponding potential. Provides virtual functions for computing potential and forces, and also several utility functions like finding intersection with equipotential surface.
    - *CPotentialLog* – logarithmic potential (with optional core);
    - *CPotentialHarmonic* – harmonic;
    - *CPotentialDehnen* – Dehnen potential ($0 \le \gamma \le 2$);
    - *CPotentialScaleFree* – single power-law cusp ($0 \le \gamma < 2$);
    - *CPotentialNB* – tree-code N-body potential (using points from a file);
    - *CPotentialSH* – basic class for all potential expansions for which angular part is represented in spherical harmonics. Derived classes implement a virtual function for computing SH coefficients and their derivatives at a given radius:

        · *CPotentialScaleFreeSH* – SH expansion of the power-law density profile, recommended instead of the exact one;
        · *CPotentialBSE* – basis-set expansion using Zhao(1996) basis set, which includes Hernquist-Ostriker(1992) and Clutton-Brock(1973) sets as special cases.
        · *CPotentialSpline* – SH coefficients are represented by spline functions in radius.

- `orbit.cpp` – orbit class and related paraphernalia:
    - *COrbit* – class for performing computation of a single orbit (with given initial conditions in a given potential), using either Runge-Kutta or leap-frog integrators [TODO: split into separate classes]. Orbit integrator calls one or more "runtime functions" every output timestep (which is not related to integration timestep).
    - *C***Information, COrbitRuntime***, COrbitRuntime***Creator*: triplets of related classes for storing various data about the orbit:
        - *Trajectory* (and *COrbitInformation*) – general data about the orbit, like the orbit class, frequencies and chaotic properties, etc. The trajectory itself is used in computing these properties, but not stored in the information class.
        - *TrajSample* – store a given number of random points from the trajectory, which later may be used in creating N-body model from SM.
        - *Poincaré* – record data for Poincaré surface of section (for 2d orbits), used only for visualization, not stored in files.
        - *Pericenter* – record pericenter passages and fits a linear regression for distribution of squared angular momentum at pericenter.

- `orbitlib.cpp` – classes for managing orbit library:
    - *COrbitDesc* – "interface" class between single orbit and COrbitLibrary; serves mainly to isolate the latter from internal data and methods of COrbit and to reduce memory consumption (each orbit in a library is represented by COrbitDesc). An instance of *COrbit* class is created internally to perform actual computations; *COrbitInitData* is passed to it and various *C***Information*'s is retrieved and stored in this class after finishing orbit integration and disposing of COrbit object.
    - *COrbitLibrary* – orbit library class; contains array of COrbitDesc and methods to generate initial conditions and to create an N-body model from orbit library.
    - *CMassRefinementFnc* – derived from CBasicOrbitFilteringFnc, ranks orbits into several bins, based on another evaluation function passed as a parameter. Used in creating N-body model with unequal masses: one may wish to have a better mass resolution for inner parts of the model, based on orbit energy, or pericenter distance, or anything else.
    - *CEnergyOrbitFilteringFnc* – returns total orbit energy for the above mentioned binned mass refinement.

- `schwarzschild.cpp` – variants of Schwarzschild model: recording necessary data from orbits, and constructing the system of equations to pass to optimization solver. Quadruplets of related classes contain the model class itself, an information container to store data (like the mass in each cell) for each orbit and for the entire model, and pair of runtime function/creator classes to record this data during orbit computation.
  - *CBasicShellSchModel* – basic class for all variants of models where kinematic data (radial and tangential velocity dispersions) is stored in concentric shells in radius, and the density data representation is not specified. Derived classes are:
  - *Classic* – density data is fraction of time an orbit spends in each cell;
  - *SHBSE* – data is coefficients of basis-set expansion for the density of an orbit;
  - *SHGrid* – similar but with spherical-harmonic coefficients evaluated at a certain number of radial grid points.
- `optimization.cpp` – wrappers for linear/quadratic solvers, which are called from Schwarzschild models:
  - *COptimizationSolverBPMPD* – calls external program BPMPD, writes and reads text files for exchanging data with it;
  - *COptimizationSolverGLPK* – calls a solver from GLPK library.
- `fileio.cpp` – reading and writing various data to disk: individual orbit trajectories, orbit library with related data for *SM*, potential coefficients and so on.
  - *CFileInputText* – reading primary text-based internal data formats: single orbit trajectory (sec. 4.4.2); orbit library (sec. 4.2) in text format, with bulk data (*SM* and trajectory sample data) in binary formats (sec. 4.3); potential coefficients (sec. 4.4.5) or initialization of a BSE/Spline/N-body potential from a point mass set (sec. 4.4.1).
  - *CFileOutputText* – writing the same types of data as above, plus writing a point mass set from a *SM* $\rightarrow$ N-body model convertor.
  - *CFileOutputNEMO* – writing a point mass set as a *NEMO* snapshot file, also used in export to N-body model.
- `massmodel.cpp` – an auxiliary class *CMassModel* for computing various properties of a spherically-symmetric isotropic system with given density profile. Used in creating random initial conditions for *SM*.
- `core.cpp` – core class that provides all non-GUI workflow, and several helper thread classes that perform computations in parallel with main program.
  - *COrbitsCore* – Qt class performing most “business logic”: runs various threads, keeps INI file, processes console input or script, dispatches messages, etc. It also owns instances of CPotential, COrbit, COrbitLibrary and CSchwarzschild-Model.

  - *CCalcThread* – simple Qt thread class that runs integration of single COrbit (that is displayed in GUI on the "Orbit" page).
  - *CCalcManyThread* – a thread class doing same task for orbits from COrbitLibrary; several instances are run in parallel, so it has mutex locking mechanism to select the next orbit in queue for computation.
  - *CSchwarzschildThread* – thread that runs methods of CSchwarzschildModel (optimization and N-body export).

- `gui.cpp` – main GUI window class that handles all interactive operations, and a helper thread class that performs Delaunay triangulation to render orbit as a solid body.
  - *CSmileGUI* – Qt class representing main window, displaying data and handling user input.
  - *CTriangThread* – thread that runs external program `QDelaunay` and processes its output to create a triangulated "outer surface" of an orbit.

- `smile.ui` is the Qt form file containing the main window layout.

6 Known bugs, subtleties and limitations

- All analytic density models assume triplanar symmetry w.r.t. change of sign of any coordinate. For the general-purpose expansions (BSE/Spline) and frozen-N-body potential it is not necessary: one may choose the desired level of symmetry for the expansion. If initializing it from a discrete set of points, it is crucial that the density center is at origin (except for N-body potential, however even for it this is necessary in order for orbit analysis to work properly), and principal axes of figure should be directed along x, y, z as longest to shortest: while the spherical-harmonic expansion should be invariant to rotation (at least when symmetry type is downgraded to "Reflection") the correct ordering of axes is important for orbit analysis. None of the potential expansion methods are expected to perform well for highly flattened models. General-purpose expansions and the entire *SM* module are agnostic to mass normalization of model, but the length scale (typical half-mass radius) should be of order unity (i.e. not too much off, say by a factor of 10), because a number of design choices break the invariance. Therefore, it is recommended to work in dimensionless, N-body units, rather than in parsecs, for example.

- BSE/Spline expansions assume power-law density behaviour at $r \to 0$ with the index $\gamma \in [0, 2)$ (they still work pretty well for $\gamma = 2$, but not for steeper profiles which have divergent potential at origin). Inner and outer density asymptotics are assumed to be power-laws, which may introduce some systematic errors for other type of profiles (such as Sérsic), but they are negligible for a sufficient number of terms.

- *Chaotic* attribute in the orbit description should not be relied upon (it is added when there are lines in spectrum that cannot be fitted as a linear combination of no more than `N_dim` fundamental frequencies, but sometimes the accuracy demanded might be too stringent or too weak). A better indicator is the Frequency Diffusion parameter or Lyapunov exponent. Moreover, in the case of two very nearby spectral lines the method often gets confused and assigns "chaotic" attribute (and even a rather high frequency diffusion rate) to a regular orbit. This typically may be overcome by setting a longer integration time, to better resolve these nearby lines.

- Scale-free potential and its BSE approximation are implemented only for $0 \le \gamma < 2$. In addition, variation equation option for computing Lyapunov exponent is not implemented for scale-free potential, only for its BSE variant. This is not a severe restriction, as the approximation works fairly good and much faster, so it is the preferred option.

- Computing Lyapunov exponent by integration of nearby trajectory is possible for all potentials (except N-body, of course), and this is the default compilation option, since it is usually faster than variation equation approach. Only for orbits close to the black hole may it give incorrect results: a regular orbit seems to have nonzero Lyapunov exponent, which is probably due to roundoff errors in keeping these nearby orbits really near. So to study these orbits, one should use "true" variation equation approach (Update: even in this case "chaotic" attribute may be triggered on erroneously for some tightly-bound orbits, requires further study).

- Since floating-point values are stored in text format (in Orbit Library file), sometimes it may happen that re-integration of the same orbit gives a different result. (This definitely may happen if Lyapunov exponent is used, since the deviation vector is initalized randomly). Although some measures were taken to diminish the possible damage of this effect, one should still keep it in mind. Also orbits with the same initial conditions may be different on different machines.

Other TODO include full support for rotating potentials, and possibility to use yet different linear/quadratic solvers (if there are any viable alternatives at all. Tried CVXOPT, it is painfully slow).

7 Version history

1.0 (March 2011): initial release (not advertised anywhere:)

1.1 (July 2011): implemented a broader class of basis-set expansion, fixed some bugs.

2.0 (2012): program structure substantially reworked towards modularity; it should be possible to use only potential or orbit integrator/analysis in other programs by including a subset of source files.
New class of general-purpose potential approximation (Spline spherical-harmonic expansion).
Choice of symmetry degree for BSE/Spline (from "None" through standard "Triaxial" to "Spherical").
Schwarzschild modelling: separated solver (several possible variants) from model itself; different choices for model type (classical grid-based, basis set and spherical harmonic expansion).
Different method of generating initial conditions for *SM* (based on distribution function for an equivalent spherical model, constructed by Eddington inversion formula).

A number of features were scrapped: initial conditions for SM are only generated randomly, no stationary/principal-plane start space anymore (it did not produce good models anyway), no "orbit bunches" (or "dithering", which averages over several nearby orbits: just increase the total number of orbits, and quadratic optimization will drive the orbit weights towards more uniformity). Lucy iterations optimization removed – was too slow on realistic problem sizes and never outperformed quadratic optimization.

8 Code reuse

This program includes code from Hairer et al. `DOP853` Runge-Kutta integrator, tree-force potential solver `hackcode` by J.Barnes from `NEMO` toolbox, and simplified `NEMO` snapshot writer by S.Rodionov. Everything else is written from scratch. You may use any part of the program in any your project.

References

[1] Vasiliev E., 2012 (in preparation)

A Technical details on the algorithms and formulae used

A.1 Frequency analysis

A.2 Spline spherical-harmonic expansion

The radial functions are represented as

$$\begin{aligned} C_{00}(r) &= [\exp(-\tilde{C}_{00}(\xi)) + 1/C_{00}(0)]^{-1}, \quad \xi \equiv \log r \\ C_{lm}(r) &= C_{00}(r)\tilde{C}_{lm}(\zeta), \quad \zeta \equiv \log(1+r) \end{aligned}$$

Here $C_{00}(0)$ is the value at origin, and $\tilde{C}_{00} \equiv -\log[1/C_{00}(r) - 1/C_{00}(0)]$. Tilded functions are approximated by cubic splines. The evaluation of derivatives w.r.t. r is straightforward:

$$\begin{aligned} \frac{dC_{00}}{dr} &= \frac{C_{00}^2 \exp(-\tilde{C}_{00})}{r}\frac{d\tilde{C}_{00}}{d\xi} \\ \frac{d^2C_{00}}{dr^2} &= \frac{C_{00}^2 \exp(-\tilde{C}_{00})}{r^2}\left[\frac{d^2\tilde{C}_{00}}{d\xi^2} - \frac{d\tilde{C}_{00}}{d\xi} + \left(\frac{d\tilde{C}_{00}}{d\xi}\right)^2 (2C_{00}\exp(-\tilde{C}_{00}) - 1)\right] \\ \frac{dC_{lm}}{dr} &= \frac{C_{00}}{1+r}\frac{d\tilde{C}_{lm}}{d\zeta} + \tilde{C}_{lm}\frac{dC_{00}}{dr} \\ \frac{d^2C_{lm}}{dr^2} &= \frac{C_{00}}{(1+r)^2}\left[\frac{d^2\tilde{C}_{lm}}{d\zeta^2} - \frac{d\tilde{C}_{lm}}{d\zeta}\right] + \frac{2}{1+r}\frac{d\tilde{C}_{lm}}{d\zeta}\frac{dC_{00}}{dr} + \tilde{C}_{lm}\frac{d^2C_{00}}{dr^2} \end{aligned}$$

B Additional programs

A number of auxiliary tools are built on *SMILE* core, most using `NEMO` framework to operate on binary snapshot data.

B.1 mkspherical

Creates spherical mass model from a supplied table with $r, m(r)$ values specifying enclosed mass as a function of radius. The same class from `massmodel.cpp` is used to compute a number of parameters as spline-interpolated functions of radius (potential, radial/circular period, distribution function via Eddington inversion formula, etc.) Output is a table with all these parameters and/or a N-body snapshot for an equilibrium spherical model with isotropic velocity dispersion.

B.2 getpotential

Computes and stores in a text file BSE or Spline coefficients from an N-body snapshot, and optionally a `mkspherical`-type table for an equivalent spherical model.

B.3 rendershape

Inverse of the above: from a BSE/Spline potential file, create a visual representation of this potential by populating space with point masses according to local density (with zero velocities).

B.4 snaporbits

Performs orbit analysis for a N-body simulation (input file containing multiple time snapshots), using a given potential model from a text file, or computing it from the first snapshot of the input file. Outputs OrbitLibrary file with orbit properties (i.e. trajectories of individual particles from the snapshot are analyzed). Useful to compute orbit population and/or proportion of centrophilic orbits; chaotic properties are not very meaningful. To analyze what kind of orbits are possible in a smoothed potential of an N-body model, it is better to use the entire *SMILE* machinery.

B.5 getshape

Computes axis ratios and orientation of principal axes of an N-body snapshot, as functions of radius. Use either equidensity ellipsoid axis ratios (if density is decreasing function of radius), or moment of inertia tensor. (Doesn't use *SMILE*).