SMILE reference

Eugene Vasiliev
Lebedev Physical Institute, Leninsky prospekt 53, Moscow, Russia
email: eugvas@lpi.ru

Version 1.1
July 15, 2011

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. It 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.

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 analysis results)).
- 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: $\Phi(m) = \ln(R_c^2 + m^2)$;
- Triaxial Dehnen: $\rho(m) = \frac{3-\gamma}{4\pi pq} m^{-\gamma}(1+m)^{-(4-\gamma)}$;
- Scale-free: $\rho(m) = m^{-\gamma}$;
- Plummer: $\rho(m) = \frac{3}{4\pi pq}(1+m^2)^{-5/2}$;

[1] Schwarzschild Modelling Interactive expLoratory Environment

- Anisotropic harmonic oscillator: $\Phi(m) = m^2$;
- Basis-set expansion (BSE, see below);
- Frozen-N-body (see below).

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

- $q = y/x$ and $p = z/x$ – axis ratios, should be $p \leq q \leq 1$. 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 $0 \leq \gamma \leq 2$.
- R_c (core radius) – in log.potential denotes region of constant-density core, may be zero.
- N_{radial} and $N_{angular}$ – number of terms in basis-set expansion.
- ϵ (smoothing length), θ (tree opening angle) – parameters for frozen-N-body tree-code
- Nbody file – holds coordinates of particles in frozen-Nbody or generic BSE potentials.

BSE is a potential representation in terms of finite number of basis functions with given coefficients; the number of radial terms is N_{radial} and angular (spherical harmonics) – $N_{angular}(N_{angular}+1)$, only even angular terms are used (which ensures symmetry w.r.t. sign change of any coordinate). It can be used in two variants: as an approximation of mass distribution given by N-body file (generic BSE) or as an alternative to Dehnen, Plummer or scale-free potentials (in fact it is the only option for Plummer). In the latter case only one radial function (power-law) is used, and it is called Scale-free SH (spherical-harmonic).

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 smoothing 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).

In the case of generic BSE and N-body potentials, the file with particle coordinates should be supplied (by pressing the text *Nbody file* and selecting, instead of typing the filename). The axis ratios and cusp slope are not used here in potential evaluation (which is determined by the particle coordinates only), but in creating grid for Schwarzschild modelling (in fact, it assumes that the underlying density profile is still Dehnen. This may be overcome in future). However, an important parameter for the basis set is α (transition strength in the Zhao α-model which is used in the BSE, see [1], Eq. A7).

Note that potential is re-initialized upon any change of its parameters (this can take quite a while for BSE and frozen-N-body).

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 Plummer $E < 0$, 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 4.4.1).

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).

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 $\mathbf{w}$ 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).

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.

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 has the same options as for frequency map (number of points in start spaces, etc.), but the numbers are given *per one radial shell*.

Grid parameters in the middle specify the geometry of configuration-space grid: number of radial shells and number of lines splitting each of three segments of each shell (so the number of cells in shell is $3n_{lines}^2$).

Additionally, one may set the number of **orbits in a bunch**: if $N_b \geq 1$, each orbit in fact is accompanied by $N_b - 1$ adjacent orbits (with slightly perturbed initial conditions), and their cell occupation and trajectory sampling will be averaged. (The orbits text file still contains data only for the "master" orbit of that bunch).

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.

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.

Use superorbits forces averaging of all chaotic orbits within a single shell into one (their weights will be essentially equal after optimization). One may extend this averaging **up to certain shell number** or for entire model (if set to 0), or switch it off completely. Makes sense only for initial conditions from a grid at fixed energy levels, not for random ICs. Again, definition of chaotic orbit comes from settings at the *Frequency map* tab.

Maximal orbit weight is a restriction useful to avoid re-integration while creating N-body model (see below); 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 from β_{in} in the first shell to β_{out} in the last shell.

There are several ways of solving the optimization problem – linear and quadratic programming, and Lucy iterations. The first two are solved with interior-point method (either GLPK library, 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. Lucy

iterations aim to converge to exact solution which is very smooth, but run really slow and cannot include additional constraints, so are not of much use.

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).

Export grid creates a text file with one line per cell, and one per shell in the end. It contains data such as cell mass (required and obtained in the solution), velocity dispersion, orbit population in shells.

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 $N_{\mathrm{sampling\,points}}/N_{\mathrm{bodies}}$, 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 $10\,N_{\mathrm{bodies}}/N_{\mathrm{orbits}}$ 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 optionally may be also exported to text file as an orbit library (if the ini flag `exportNbodyOrbitsFile` is set). Then some statistical quantities are calculated, including virial ratio.

View options switch between various plots:
Grid cell displays the model cells (blue for feasible, red for infeasible, for which the mass difference from the required value is $> 1\%$). They are arranged in a projection on the 2d plane similarly to stationary start-space points. Right-click on a cell displays some information in the message area.
Anisotropy shows the value of β for each cell (horizontal axis is the cell 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.
Weight histogram displays the same quantity as a cumulative distribution function. The flatter is it, the better...

Here also one may show any particular **grid shell** or the entire model.

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 4.4.3.

`ExportGrid("orbitsfile.grid")` creates a text file with statistics about Schwarzschild model grid (same as eponymous button on *Schwarzschild* page), see Section 4.4.2.

`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, BSE, Dehnen BSE, Plummer BSE, Scale-free, Scale-free SH, N-body.

`N_dim (3)` – 2 or 3.

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

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

`Rc (1)` – core radius of logarithmic potential.

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

`Ncoefs_radial, Ncoefs_angular (10, 4)` – number of radial and angular terms in BSE (for angular part only even functions are used, so N_a corresponds to $l_{max} = 2(N_a - 1)$.)

`NbodyFile` – file with particle coordinates for BSE and N-body potentials (7 numbers per line: coordinates, velocities (unused) and mass).

`treecodeEps (0.01)` – ϵ, smoothing length used in frozen-N-body integration.

`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, smoothed (i.e. it is $\sqrt{l_{\rm true}^2 + \epsilon^2}$), 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.

`[Frequency_map]`

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

`[Schwarzschild_model]`

`numOrbitsStationary, numOrbitsPrincipalPlane, numOrbitsYalpha, numOrbitsRandom` – number of points in corresponding start spaces *in each radial shell*.

`linesPerSegment (4)` – split each shell into 3 × (linesPerSegment)^2 cells.

`numShells (20)` – number of radial shells (both in spatial grid and in assigning initial conditions).

`numBunchOrbits (1)` – if > 1, each point cooresponds to several slightly perturbed ICs, and their cell occupation numbers are averaged (although only the 'root' orbit is represented in the orbit library file).

`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 more importantly, 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 $\Lambda = 0$).

`chaoticSuperOrbit (false)` – whether to average all chaotic orbits from the same energy level into one super-orbit (makes sense only for grid-based, not random, initial conditions).

`maxShellSuperOrbit (0)` – if use super-orbits, controls the maximum number of energy level for which this averaging is performed: inner levels are averaged while outer are not. Useful because often a fully averaged model is infeasible, but a model in which only few inner shells are averaged is feasible. 0 corresponds to averaging all energy shells.

`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 between).

`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.

`exportNbodyOrbitsFile (false)` – when creating N-body model, points are written in *NEMO* snapshot format (i.e. coordinates, velocities and mass). If this option is switched on, also write an orbit library file containing all the points (initial conditions are sampled from trajectory, all other fields are taken from corresponding orbit). This requires a lot of memory, about 0.5 kb per orbit, so may fail for large enough number of particles.

`randSigma{X/Y/Z}{In/Out}` – parameters controlling the velocity dispersion in generating random initial conditions (for the whole Schwarzschild model, not for a frequency map at given energy, in which the density/velocity is assigned ergodically and isotropically). The random IC generation is done as follows: first a radius is picked uniformly from inverse $M(r)$ relation (mass contained within this radius varies linearly from 0 to maximum mass of $1 - 1/N_{shell}$), and direction is picked randomly; then the 1d veloc-

ity dispersion in the equivalent spherical model is calculated from known formulae for Dehnen and Plummer models, or as $1/\sqrt{3}$ of escape velocity for anything else. Then it is multiplied by these coefficients in each direction (interpolated between inner and outer radial shells). This is done to favor more radial orbits in the outer parts (lesser velocity dispersion relative to spherical isotropic model), to increase the fraction of orbits that are selected in the solver (otherwise most of them are assigned zero weight, as they are unsuitable to support necessary shape and velocity anisotropy). Guessing the values for these coefficients is a kind of black magic :-(.

`randRefineFrac (0.75), randRefineFact (2.0)` – another means to increase the number of usable orbits in the outer parts of the model, applies to random IC generation. If both of them $\neq 1$, the number of orbits whose radius contains more than `randRefineFrac` of total mass, is increased by a factor of `randRefineFact`.

`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 index shell time timestep timeunit ...
    ... lfx lfy lfz lfdiff lambda description chaotic tmax ...
    ... energy ediff inertx inerty inertz
```

First 7 fields are coordinates and mass (which means orbit weight in Schwarzschild model, so it may be zero). 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.

`index` is the orbit index (initialized as sequential number during generation of initial conditions for frequency map or Schwarzschild model). Its use is mainly for identification of orbits in N-body model that were parented by the same orbit.

`shell` is the index of energy shell in Schwarzschild model grid, to which this initial conditions belong. Used in filtering orbits in *Frequency map* and *Schwarzschild* pages.

`time` is the integration time (in common time units, not in periods), `timestep` is the timestep of trajectory output, and `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).

`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.
`chaotic` boolean attribute is the flag set if either `lfdiff` or `lambda` indicate chaos.
`energy` is the total orbit energy, and `ediff` is the energy conservation error.
`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).

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 fraction of time each orbit spent in each cell (orbit-cell-weight, `.ocw`), and radial and tangential velocity dispersion of each orbit in each cell. Contains $3 \times n_{shell} \times n_{lines}^2 + 1$ 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.

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 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.

4.4.2 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.
`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).

4.4.3 Potential 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 100 and lie along seven lines: principal axes, diagonals of principal planes ($z = 0, y = x$, etc.) and the diagonal $x = y = z$. This data may be useful, for example, to test the accuracy of BSE approximation.

5 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.

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

`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:

- `defines.h` – most numerical constants, macros, and configuration data
- `potential.cpp` – class hierarchy for potential models: each model implements functions for density, potential and force evaluation.
  - *CPotential* – basic (abstract) class, provides virtual functions for evaluation of density, potential, forces, and also convenience 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 ($0\gamma < 2$);
      - *CPotentialScaleFreeBSE* – basis-set (spherical-harmonic) expansion of the above (recommended);
    - *CPotentialBSE* – basis-set expansion of arbitrary potential (using Zhao(1996) basis set, which includes Hernquist-Ostriker(1992) and Clutton-Brock(1973) sets as special cases). May use points from a file to calculate expansion coefficients;
      - *CPotentialDehnenBSE* – BSE for Dehnen potential (prepares coefficients based on exact Dehnen profile);
      - *CPotentialPlummerBSE* – same for Plummer potential;
    - *CPotentialNB* – tree-code N-body potential (using points from a file).
- `orbit.cpp` – orbit and orbit library classes: integration, frequency analysis and orbit classification, generation of initial conditions, import/export of orbit library.
  - *COrbit* – class for performing computation of a single orbit (with given initial conditions in a given potential) and its frequency analysis/classification, including Lyapunov exponent and other quantities.
  - *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). Contains initial data and results of orbit calculation, internally creates COrbit for performing actual computation.
  - *COrbitLibrary* – orbit library; contains array of COrbitDesc and methods to generate initial conditions, load and store library.
- `schwarzschild.cpp` – class for performing Schwarzschild modelling (spatial grid operations, optimization driver routines, export to N-body file).
  - *CSpatialGrid* – manages partitioning of space into cells; has methods to determine cell index from coordinates, and vice versa. Used in COrbit to record time spent in each cell.

  - *CSchwarzschildModel* – Qt class performing two main operations on Schwarzschild model: solving optimization problem (by various methods) and export to N-body model. Also computes mass in cell for spatial grid (it does not know anything about potential-density pair by itself).
  - *CSnapShotWriter* – helper class that writes NEMO-compatible N-body snapshot file.

- `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 CSchwarzschildModel.
  - *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

- *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.

- 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.

- 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.

8 Code reuse

This program includes code from Hairer et al. `DOP853` Runge-Kutta integrator, tree-force potential solver `hackcode` from `NEMO` toolbox by J.Barnes, 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., 2011 (in preparation)