Galactic dynamics with AGAMA

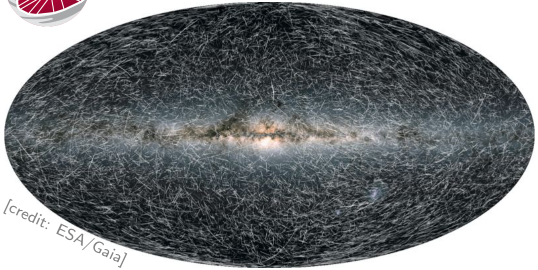
part 1: foundations

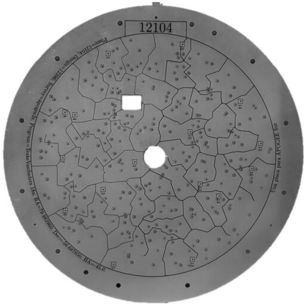
Eugene Vasiliev

Galaxy Modelling & Galactic centre workshop

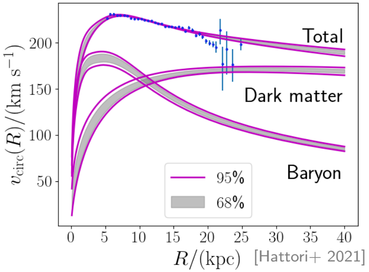
University of Surrey, December 2024

Observational and theoretical context

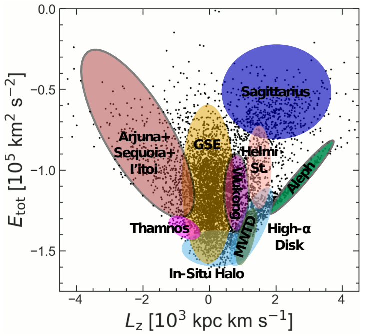




Milky Way rotation curve

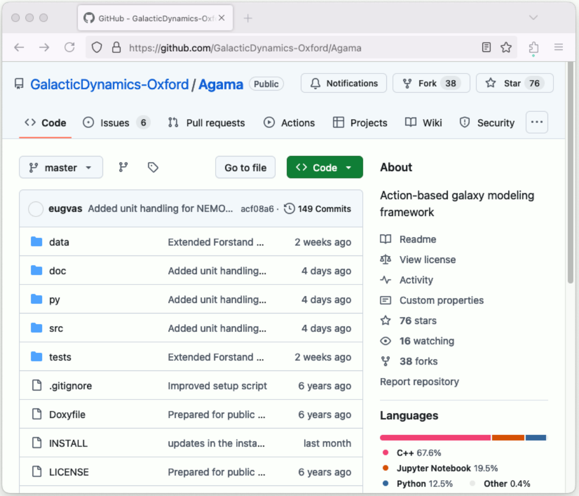


Classification of accreted stars



[Naidu+ 2020]

Agama software package



GitHub - GalacticDynamics-Oxf ×

https://github.com/GalacticDynamics-Oxford/Agama

GalacticDynamics-Oxford / Agama Public

Notifications | Fork 38 | Star 76

Code | Issues 6 | Pull requests | Actions | Projects | Wiki | Security | ···

master | Go to file | Code

eugvas Added unit handling for NEMO... acf08a6 · 149 Commits

```
data        Extended Forstand ...      2 weeks ago
doc         Added unit handling...     4 days ago
py          Added unit handling...     4 days ago
src         Added unit handling...     4 days ago
tests       Extended Forstand ...      2 weeks ago
.gitignore  Improved setup script      6 years ago
Doxyfile    Prepared for public ...    6 years ago
INSTALL     updates in the insta...    last month
LICENSE     Prepared for public ...    6 years ago
```

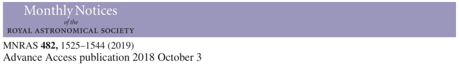
About

Action-based galaxy modeling framework

Readme
View license
Activity
Custom properties
76 stars
16 watching
38 forks
Report repository

Languages

● C++ 67.6%
● Jupyter Notebook 19.5%
● Python 12.5%
● Other 0.4%



Monthly Notices of the ROYAL ASTRONOMICAL SOCIETY

MNRAS **482,** 1525–1544 (2019)
Advance Access publication 2018 October 3

AGAMA: action-based galaxy modelling architecture

Eugene Vasiliev [1,2,3★]

development started in 2015;
code paper published in 2018;
used in $\gtrsim$350 publications.

Main features:

- core library written in C++;
- OpenMP parallelization;
- hand-made Python interface;
- extensible with user-defined functions;
- several dozen tests and example programs in C++ and Python;
- detailed documentation ($\sim$140 pages);
- $\sim$80 000 lines of code.

Structure of the Agama library

some mathsy stuff (C & Fortran)

GSL (C) & Eigen (C++) math libs

CVXOPT quad. opt. solver (Python)

C++ core $\Longrightarrow$ shared library `agama.so`

computationally heavy parts (potentials, actions, orbit integration, etc.);
built-in density, potential and DF models

Python interface (C & C++)
vectorization (operation on arrays);
OpenMP parallelization

external gravity
source interfaces
for *N*-body codes

C, Fortran
and Julia
interfaces

additional Python routines

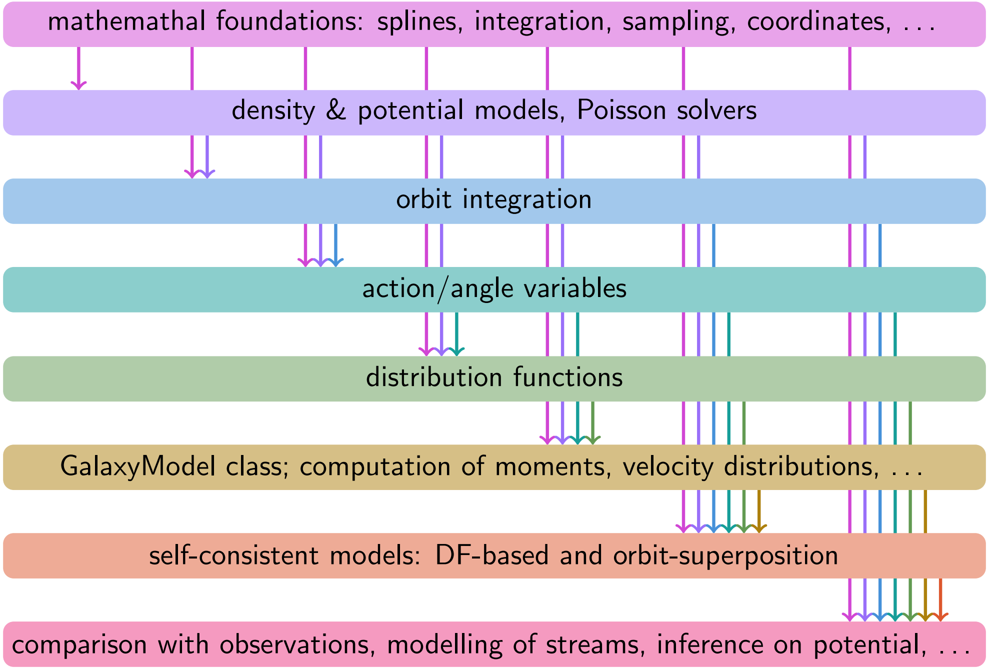
math & coord utility functions;
Forstand orbit-superposition code;
interfaces for Galpy and Gala

NEMO, AMUSE, Gadget4, Arepo

Agama potentials can be used in
Galpy and Gala

user scripts in Python (including custom density, potential & DF models)

Structure of the Agama library


mathemathal foundations: splines, integration, sampling, coordinates, ...
density & potential models, Poisson solvers
orbit integration
action/angle variables
distribution functions
GalaxyModel class; computation of moments, velocity distributions, ...
self-consistent models: DF-based and orbit-superposition
comparison with observations, modelling of streams, inference on potential, ...

Gravitational potential

Task: given the density profile $\rho(\mathbf{x})$, determine the potential $\Phi(\mathbf{x})$ from the Poisson equation: $\nabla^2\Phi = 4\pi\, G\, \rho$.

Example 1: spherical Plummer model

$$\rho(r) = \frac{3\,M}{4\pi\, a^3\, \left(1 + r^2/a^2\right)^{5/2}} \implies \Phi(r) = -\frac{G\,M}{\sqrt{r^2+a^2}}.$$

Example 2: triaxial Hernquist model

$$\rho(x,y,z) = \frac{M}{2\pi\, abc\; s\,(1+s)^3}, \quad s \equiv \sqrt{\frac{x^2}{a^2}+\frac{y^2}{b^2}+\frac{z^2}{c^2}} \implies$$

$$\Phi(x,y,z) = -G\,M \int_0^\infty \mathrm{d}\tau\, \frac{\left(1+\sqrt{\frac{x^2}{a^2+\tau}+\frac{y^2}{b^2+\tau}+\frac{z^2}{c^2+\tau}}\right)^{-2}}{2\sqrt{(a^2+\tau)(b^2+\tau)(c^2+\tau)}}.$$

It gets very complicated very quickly!

Gravitational potential

Commonly used analytic potential–density pairs: `Plummer`, `NFW`, `MiyamotoNagai`, `Dehnen`, `Ferrers` ...

If one needs more flexibility, there are three general-purpose Poisson solvers:

0. Direct integration:

$$\xcancel{\Phi(\mathbf{x}) = -\iiint d^3x'\,\rho(\mathbf{x}') \times \frac{G}{|\mathbf{x}-\mathbf{x}'|}}.$$

(impractical)

1. Azimuthal-harmonic expansion (`CylSpline`):

$$\Phi(R,z,\phi) = \sum_{m=-\infty}^{\infty} \Phi_m(R,z)\,\mathrm{e}^{im\phi}.$$

interpolated functions

2. Spherical-harmonic expansion (`Multipole`):

$$\Phi(r,\theta,\phi) = \sum_{l=0}^{\infty}\sum_{m=-l}^{l} \Phi_{lm}(r)\,Y_l^m(\theta,\phi).$$

3. `BasisSet` expansion (a.k.a. self-consistent field method of Hernquist&Ostriker 1992):

$$\Phi(r,\theta,\phi) = \sum_{n=0}^{\infty}\sum_{l=0}^{\infty}\sum_{m=-l}^{l} \Phi_{nlm}\,A_{nl}(r)\,Y_l^m(\theta,\phi).$$

Gravitational potential

Workflow for the `Multipole` potential expansion:

$$\begin{tikzcd}
\text{original } \rho(r,\theta,\phi) \arrow[d, phantom, "\wr"] & \Phi(r,\theta,\phi) \\
\text{approximate } \sum_{\ell=0}^{\ell_{\max}} \sum_{m=-\ell}^{\ell} \rho_{\ell m}(r)\, Y_\ell^m(\theta,\phi) \arrow[r, bend right, "\text{solve Poisson eqn for each term}"'] & \sum_{\ell=0}^{\ell_{\max}} \sum_{m=-\ell}^{\ell} \Phi_{\ell m}(r)\, Y_\ell^m(\theta,\phi) \arrow[u, phantom, "\shortparallel"]
\end{tikzcd}$$

solve Poisson eqn for each term

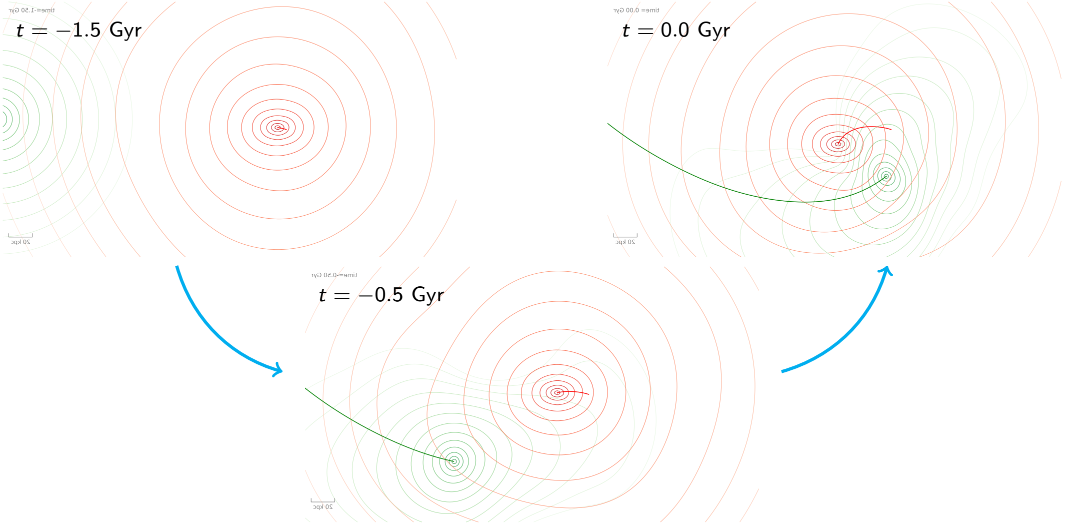
$$\Phi_{\ell m}(r) = -\frac{4\pi\, G}{2\ell+1} \left[r^{-\ell-1} \int_0^r \rho_{\ell m}(s)\, s^{\ell+2}\, \mathrm{d}s + r^{\ell} \int_r^{\infty} \rho_{\ell m}(s)\, s^{1-\ell}\, \mathrm{d}s \right]$$

`Multipole` and `BasisSet` potentials are well-suited for extended but not too flattened profiles; `CylSpline` is ideal for disky profiles, but has a finite spatial extent and doesn't like cusps.

Composite and time-dependent potentials

Potentials can be added, scaled or interpolated with time, rotated (e.g., bar or spiral arms), shifted along a time-dependent trajectory, etc.

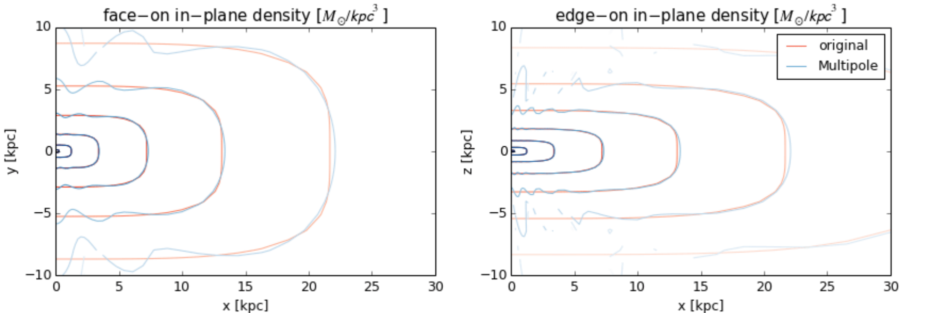
Example: potentials of Milky Way and LMC extracted from an *N*-body simulation.



Gravitational potential: example 1

User-defined density model: a boxy bar $\rho(x,y,z) = \rho_0 \exp\left(-s^{1/n}\right)$,
where $s \equiv \left[(x/a)^k + (y/b)^k + (z/c)^k\right]^{1/k}$ is the generalized ellipsoidal radius (an ordinary ellipsoid has $k = 2$), and n is the Einasto index.

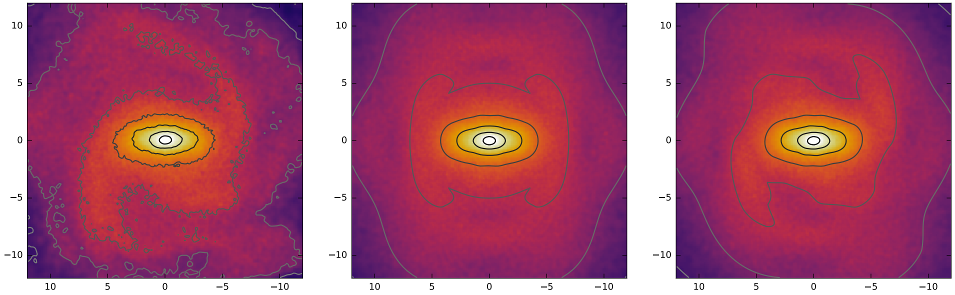
```
def dens_bar(xyz):
    x,y,z = abs(xyz).T
    s = ((x/a)**k + (y/b)**k + (z/c)**k)**(1./k)
    return rho0 * numpy.exp(-s**(1./n))

pot_bar = agama.Potential(type='Multipole', density=dens_bar,
    lmax=20, mmax=10, symmetry='triaxial')
```



Gravitational potential: example 2

One may construct these potential expansions either from an analytic density profile (including any user-defined Python function for ρ or Φ) or from an N-body snapshot, specifying the desired level of symmetry.

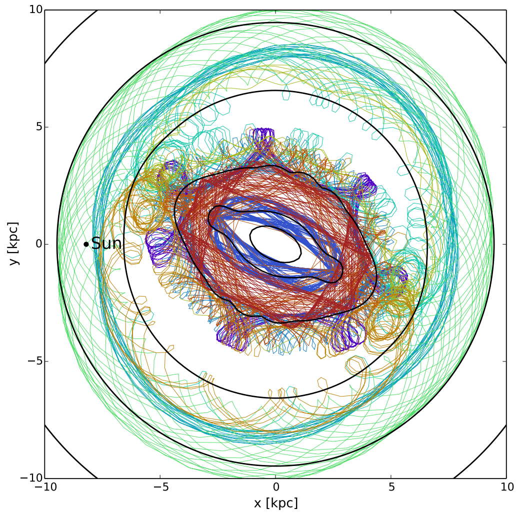
```
pot_nbody = agama.Potential(type='CylSpline',
    particles=(pos,mass), symmetry='bisymmetric')
```

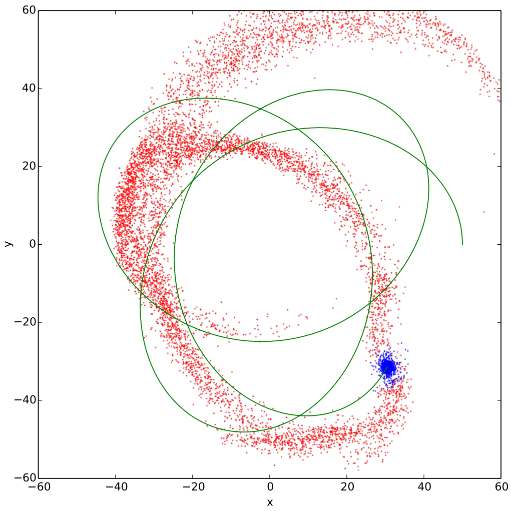


original snapshot triaxial bisymmetric

Numerical integration of orbits

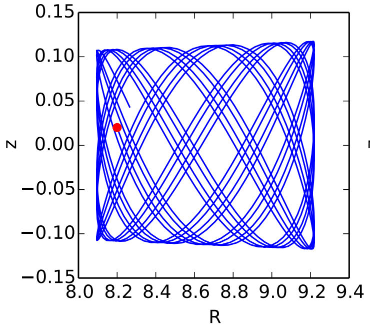
One of the most common tasks in galactic dynamics.
Examples: orbits in the MW bar; test-particle simulations of a tidal stream

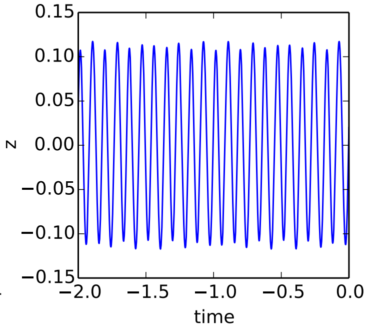


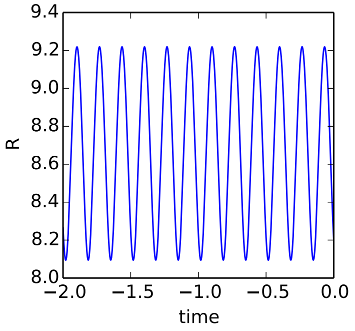


Numerical integration of orbits: example

```
agama.setUnits(length=1, velocity=1, mass=1)
#Note: distances are in kpc, velocities in km/s ⇒ time in kpc/(km/s)=0.978 Gyr
pot_mw = agama.Potential('McMillan17.ini')
time, traj = agama.orbit(potential=pot_mw,
    ic=[-8.2, 0, 0.02, 13, 245, 8], time=-2.0, trajsize=1001)
R = (traj[:,0]**2 + traj[:,1]**2)**0.5
z = traj[:,2]
```

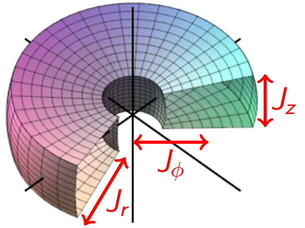






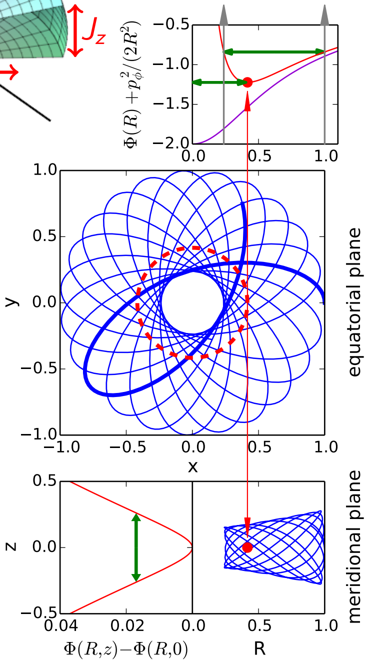
see `tutorial_potential_orbits.ipynb` for much more detail

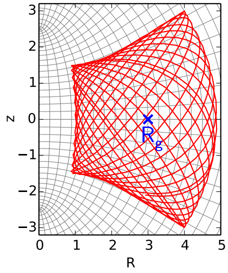
Action–angle variables



Most orbits in axisymmetric potentials look like "rectangular tori" with three parameters defining the shape:
$J_\phi \equiv L_z = R_g\, v_{circ}(R_g)$ determines the overall size of the orbit ("guiding radius" R_g);
J_R determines the extent of radial oscillations;
J_z does the same for vertical oscillations.

Corresponding phase angles $\theta_{\phi,R,z}$ determine the location on the orbit.



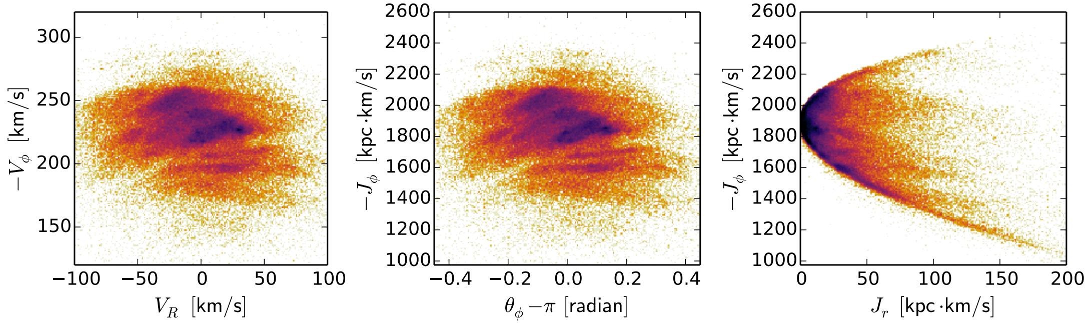


Actions are defined as $J_i \equiv \frac{1}{2\pi}\oint p_i\, dx_i$,
and are computed in the Stäckel approximation [Binney 2012],
using spheroidal coordinates for x_i.

Inverse transformation $\{\mathbf{J},\boldsymbol{\theta}\} \Rightarrow \{\mathbf{x},\mathbf{v}\}$ is provided by the Torus code [Binney & McMillan 2017].

Action–angle variables: example

```
af = agama.ActionFinder(pot_mw)
#posvel is an array of 6d phase-space coords in Cartesian frame
actions, angles = af(posvel, angles=True, frequencies=False)
J_R, J_z, J_phi = actions.T
theta_R, theta_z, theta_phi = angles.T
```



Gaia sample of nearby stars ($D < 100$ pc)

other applications: clustering in the integrals space; study of resonances, ...

Distribution functions

DF $f(\mathbf{x}, \mathbf{v})$ offers a complete description of a stellar population.

Fundamental principle of stellar dynamics (Jeans's theorem):

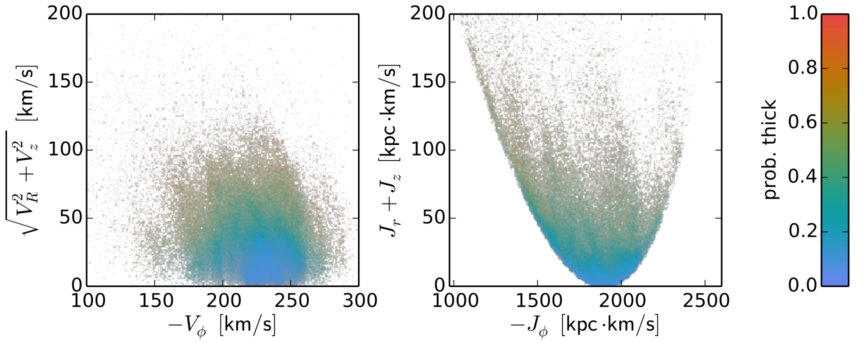
in a steady state, DF must be a function of integrals of motion $f\left(\mathcal{I}(\mathbf{x}, \mathbf{v};\ \Phi)\right)$, and it is often convenient to use actions $\mathbf{J}$ as integrals $\mathcal{I}$.

Two ways of constructing a DF in AGAMA:

- Using the Cuddeford–Eddington [anisotropic] inversion formula to obtain the DF of the form $f(E, L) = \hat{f}\left(E + L^2/(2r_a^2)\right) L^{-2\beta_0}$ corresponding to a given density $\rho(r)$ and potential $\Phi(r)$:
```
df_qs = agama.DistributionFunction(type='QuasiSpherical',
    density=dens, potential=pot, beta0=0.3, r_a=10)
```
  This DF is internally converted to the form $f(\mathbf{J})$.
- Using an *ad hoc* expression for the DF $f(\mathbf{J})$, either one of built-in models (`DoublePowerLaw`, `QuasiIsothermal`, `Exponential`) or a user-defined Python function:
```
df_dp = agama.DistributionFunction(type='DoublePowerLaw',
    J0=J0, slopeIn=Gamma, slopeOut=Beta, mass=1)
```

Distribution function applications: classification

```
df_thin = agama.DistributionFunction(type='QuasiIsothermal',
    potential=pot_mw, mass=0.7, Rdisk=2.5, Hdisk=0.15,
    sigmar0=50, Rsigmar=10)
df_thick = agama.DistributionFunction(type='QuasiIsothermal',
    potential=pot_mw, mass=0.3, Rdisk=2.5, Hdisk=0.40,
    sigmar0=150, Rsigmar=10)
df_total = agama.DistributionFunction(df_thin, df_thick)
prob_thick = df_thick(actions) / df_total(actions)
```



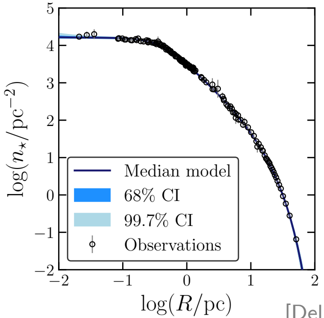
Distribution function applications: potential inference

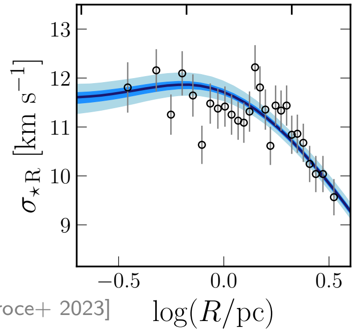
DF is a *probability distribution* for finding a star with a given position and velocity, and it also depends on the potential Φ via the integrals of motion $\mathcal{I}$.

By maximising the likelihood of the observed dataset, one can determine the best-fit parameters $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ of the stellar system, including its mass distribution.

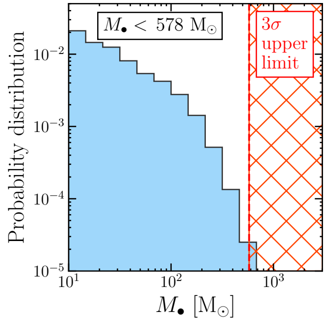
$$\ln \mathcal{L} = \sum_{i=1}^{N_\mathrm{stars}} \ln f\Big(\mathcal{I}\big(\mathbf{x}, \mathbf{v}\,;\ \Phi(\mathbf{x}\,;\ \boldsymbol{\alpha})\big);\ \boldsymbol{\beta}\Big)$$

Example: dynamical modelling of the globular cluster NGC 104 with the goal of constraining the mass of the central IMBH.





[Della Croce+ 2023]



Distribution function moments

The 6d DF $f(\mathbf{x}, \mathbf{v})$ can be reduced to more "easy to grasp" quantities:

- density $\rho(\mathbf{x}) = \int f(\mathbf{x}, \mathbf{v})\, \mathrm{d}^3 v$,
- mean velocity $\overline{\mathbf{v}}(\mathbf{x}) = \frac{1}{\rho(\mathbf{x})} \int \mathbf{v}\, f(\mathbf{x}, \mathbf{v})\, \mathrm{d}^3 v$,
- second moment of velocity $\overline{v_{ij}^2}(\mathbf{x}) = \frac{1}{\rho(\mathbf{x})} \int v_i\, v_j\, f(\mathbf{x}, \mathbf{v})\, \mathrm{d}^3 v$,
- more generally, velocity distribution at a given point
  $\mathfrak{f}(v_1;\, \mathbf{x}) = \frac{1}{\rho(\mathbf{x})} \int f(\mathbf{x}, \mathbf{v})\, \mathrm{d}v_2\, \mathrm{d}v_3$ (it can be strongly non-Gaussian!).

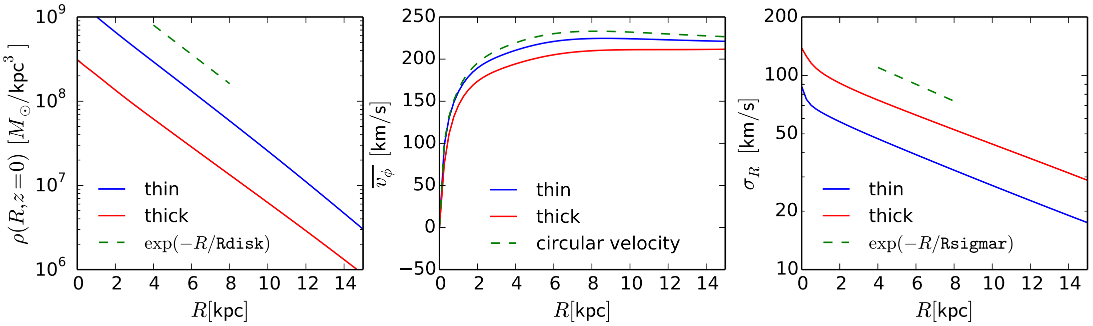
In most cases, we need $f(\mathbf{x}, \mathbf{v})$, but the DF gives us $f(\mathbf{J}) \Longrightarrow$
use an action finder to convert from phase-space to action-space.

The combination of a potential $\Phi(\mathbf{x})$, action finder $\mathbf{J}(\mathbf{x}, \mathbf{v} \mid \Phi)$ and DF $f(\mathbf{J})$ is called `GalaxyModel`.

Distribution function moments

```
gm = agama.GalaxyModel(pot_mw, df_total)
radii = numpy.linspace(0, 15, 61)
rho, vel, vel2 = gm.moments(
    numpy.column_stack([radii, radii*0, radii*0]),
    dens=True, vel=True, vel2=True, separate=True)
ax[0].plot(radii, rho[:,0], color=’b’, label=’rho,thin’)
ax[0].plot(radii, rho[:,1], color=’r’, label=’rho,thick’)
ax[1].plot(radii, vel[:,0,1], color=’b’, label=’mean vphi, thin’)
ax[2].plot(radii, vel2[:,0,0]**0.5, color=’b’, label=’sigma_r, thin’)
```

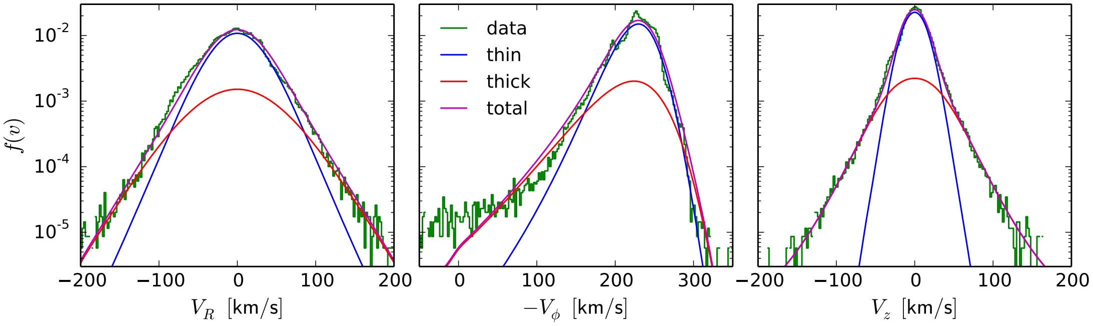
index of DF component ⤴ ⤴ index of velocity dimension: $x, y, \ldots$



Distribution functions in velocity (1d projections)

```
gridv = numpy.linspace(-200, 350)
vdf_vx, vdf_vy, vdf_vz, norm = gm.vdf([8.2, 0, 0],
    gridv=gridv, separate=True, dens=True)
frac0 = norm[0] / sum(norm)
frac1 = 1 - frac0
ax[0].plot(gridv, vdf_vx[0](gridv) * frac0, label='f(v_R), thin')
ax[0].plot(gridv, vdf_vx[1](gridv) * frac1, label='f(v_R), thick')
ax[0].plot(gridv, vdf_vy[0](gridv) * frac0, label='f(v_phi), thin')
ax[0].plot(gridv, vdf_vz[0](gridv) * frac0, label='f(v_z), thin')
```

index of DF component

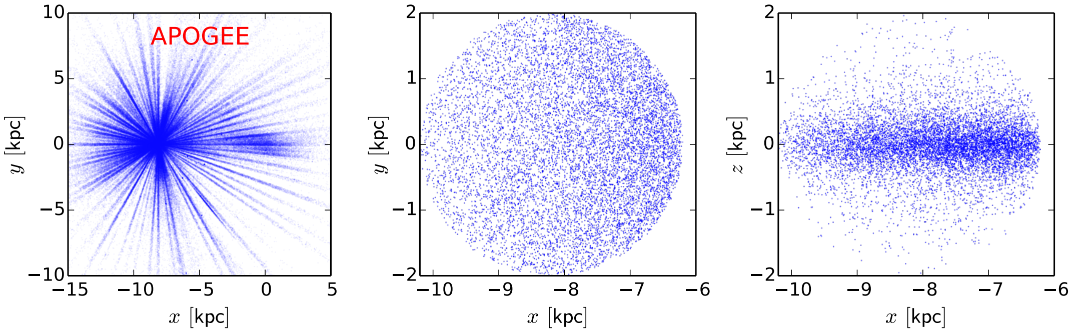


Sampling from a distribution function

One can use a selection function (usually spatial) to restrict the `GalaxyModel` to a limited volume, essentially using DF×SF: $f\left(\mathbf{J}(\mathbf{x},\mathbf{v})\right) \times S(\mathbf{x})$.

```
sf = agama.SelectionFunction([8.2, 0, 0], radius=2.0)
gm1 = agama.GalaxyModel(pot_mw, df_total, sf=sf)
posvel, mass = gm.sample(1000000)
ax[0].scatter(posvel[:10000,0], posvel[:10000,1])  # x,y
ax[0].scatter(posvel[:10000,0], posvel[:10000,2])  # x,z
```

Or sample from the entire model (e.g., to create an N-body representation of it)...



Construction of self-consistent $\boxed{\text{equilibrium}}$ models

Distribution function of stars $f(\mathbf{x}, \mathbf{v}, t)$
satisfies [sometimes] the collisionless Boltzmann equation:

$$\underbrace{\frac{\partial f(\mathbf{x}, \mathbf{v}, t)}{\partial t}}_{\text{not measured directly on human timescales}} + \mathbf{v}\,\frac{\partial f(\mathbf{x}, \mathbf{v}, t)}{\partial \mathbf{x}} - \underbrace{\frac{\partial \Phi(\mathbf{x}, t)}{\partial \mathbf{x}}}_{\text{Potential} \Leftrightarrow \text{mass distribution}}\,\frac{\partial f(\mathbf{x}, \mathbf{v}, t)}{\partial \mathbf{v}} = 0.$$

In order to infer anything about the potential from a time-dependent DF, need to make further assumptions about the initial state of the system, e.g., that the stars belong to a single stream or were perturbed from an equilibrium configuration in a specific way, etc.

Construction of self-consistent $\boxed{\text{equilibrium}}$ models

Distribution function of stars $f(\mathbf{x}, \mathbf{v}, t)$
satisfies [sometimes] the collisionless Boltzmann equation:

$$\mathbf{v}\,\frac{\partial f(\mathbf{x},\mathbf{v}\quad)}{\partial \mathbf{x}} - \frac{\partial \Phi(\mathbf{x}\quad)}{\partial \mathbf{x}}\,\frac{\partial f(\mathbf{x},\mathbf{v}\quad)}{\partial \mathbf{v}} = 0.$$

3D
(want to infer)

Steady-state assumption $\Longrightarrow$ Jeans theorem:

$$f(\mathbf{x},\mathbf{v}) = f\left(\mathcal{I}(\mathbf{x},\mathbf{v};\ \Phi)\right)$$

3D – 6D
(observed)

integrals of motion ($\leq$ 3D?), e.g., $\mathcal{I} = \{E, L, \ldots\}$

Construction of $\boxed{\text{self-consistent}}$ equilibrium models

Definition: a stellar system described by a time-independent DF $f\big(\mathcal{I}(\mathbf{x},\mathbf{v};\ \Phi)\big)$ and potential $\Phi(\mathbf{x})$, which are related by the Poisson equation:

$$\nabla^2\Phi(\mathbf{x}) = 4\pi\, G\, \rho(\mathbf{x}), \quad \text{where}\ \ \rho(\mathbf{x}) = \iiint \mathrm{d}^3\mathbf{v}\; f\big(\mathcal{I}(\mathbf{x},\mathbf{v})\big).$$

Applications:

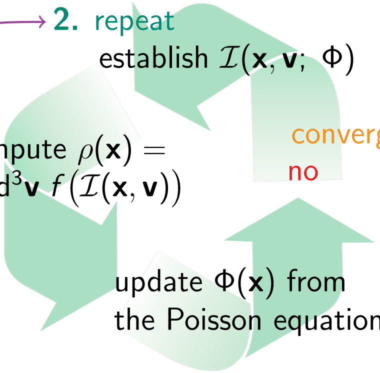
- inference on gravitational potential from stellar kinematics (so-called *dynamical modelling*)
- creation of initial conditions for isolated galaxy simulations

Methods: (non-exhaustive list)

- DF$_1$: $\Phi + \rho \implies f$ (Eddington–Ossipkov–Merritt–Cuddeford inversion) only in spherical systems; `QuasiSpherical` DF with two free params β_0, r_a
- DF$_2$: $f \implies \Phi + \rho$ (iterative method)
- orbit-superposition: $\Phi + \rho + f_{i=1..N} \implies w_i$ (Schwarzschild method)

Iterative construction of DF-based self-consistent models

1. assume $f(\mathcal{I})$ and an initial guess for Φ $\longrightarrow$ **2.** repeat



establish $\mathcal{I}(\mathbf{x},\mathbf{v};\ \Phi)$

compute $\rho(\mathbf{x}) = \iiint d^3\mathbf{v}\ f\big(\mathcal{I}(\mathbf{x},\mathbf{v})\big)$

update $\Phi(\mathbf{x})$ from the Poisson equation

converged?

no

yes $\longrightarrow$ **3.** enjoy!

```
scm_params = dict(
    rminSph=0.01, rmaxSph=100., sizeRadialSph=25, lmaxAngularSph=4)
df = agama.DistributionFunction(**df_params)
comp = agama.Component(df=df, disklike=False, **scm_params)
scm = agama.SelfConsistentModel(components=[comp], **scm_params)
scm.potential = init_potential
for i in range(5):
    scm.iterate()
comp.density.export('final_density.ini')
scm.potential.export('final_potential.ini')
```

Orbit-superposition method for self-consistent models

Introduced by Schwarzschild (1979) as a practical approach
for constructing self-consistent triaxial models with prescribed $\rho(\mathbf{x}) \Leftrightarrow \Phi(\mathbf{x})$.

integrals of motion

To invert the equation $\rho(\mathbf{x}) = \iiint f\left(\mathcal{I}\left(\mathbf{x}, \mathbf{v}\,;\;\Phi\right)\right)\, d^3\mathbf{v}$,

discretize both the density profile and the distribution function:

$\rho(\mathbf{x}) \implies$ cells of a spatial grid;
mass of each cell is $M_c = \iiint\limits_{\mathbf{x}\in V_c} \rho(\mathbf{x})\; d^3x$;

$f(\mathcal{I}) \implies$ collection of orbits with unknown weights:

$$f(\mathcal{I}) = \sum_{k=1}^{N_{\rm orb}} w_k\; \delta(\mathcal{I} - \mathcal{I}_k)$$

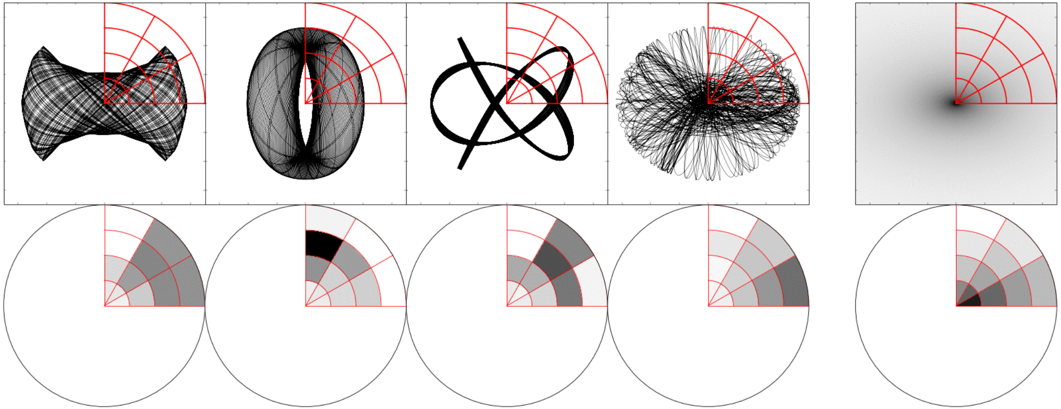
each orbit is a delta-function in the space of integrals of motion

adjustable weight of each orbit [to be determined]

Orbit-superposition method for self-consistent models

orbits in the model

target density



discretized orbit density

(fraction of time t_{kc} that k-th orbit spends in c-th cell)

discretized density

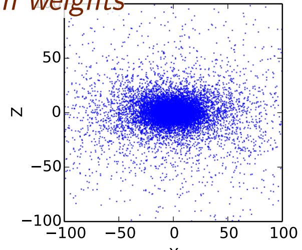
(mass M_c in grid cells)

For each c-th cell we require $\sum_k w_k \, t_{kc} = M_c$, where $w_k \geq 0$ is orbit weight

system of linear equations with nonnegativity constraints $\Rightarrow$ optimization problem

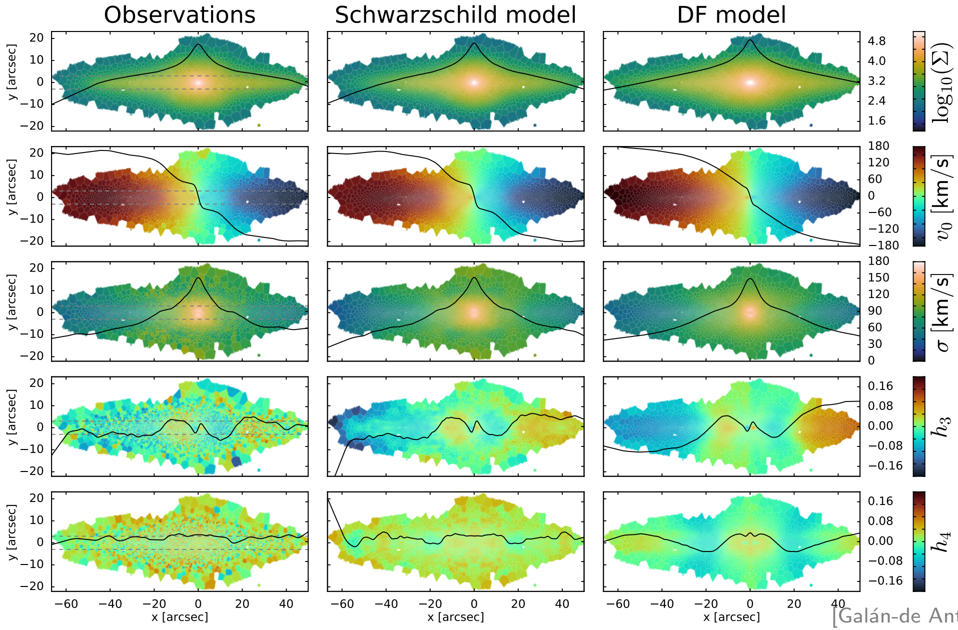
Orbit-superposition method: example

```
dens = agama.Density(type='Spheroid', axisRatioZ=0.5)
pot = agama.Potential(type='Multipole', density=dens)
target = agama.Target(type='DensityClassicLinear',
    gridr=numpy.logspace(-1.5, 2), stripsPerPane=2)
# prepare initial conditions for the orbit library
numOrbits = 10**4
ic,_ = dens.sample(numOrbits, potential=pot)
# compute the orbits and their contribution to the density
matrix, trajs = agama.orbit(potential=pot, ic=ic,
    time=100*pot.Tcirc(ic), dtype=object, targets=[target])
# solve the optimization problem to get the orbit weights
rhs = target(dens)
weights = agama.solveOpt(matrix=matrix.T, rhs=rhs)
# create N-body snapshot from recorded trajectories with weights
nbody = 10**5
_,(xv,m) = agama.sampleOrbitLibrary(
    nbody, trajs, weights)
plt.scatter(xv[:,0], xv[:,2])
# (omitting some non-essential but useful steps)
```



Example of DF- and orbit-based dynamical models

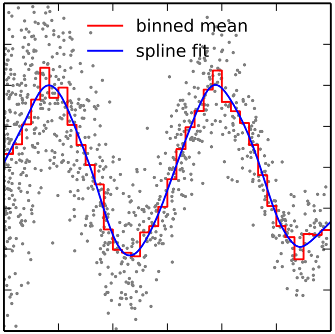
Model of an edge-on S0 galaxy FCC 170 constrained by MUSE IFU kinematics

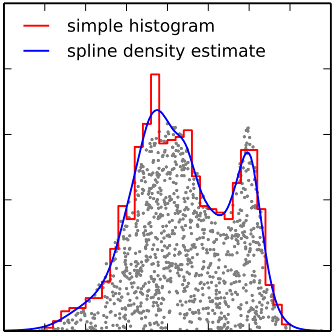


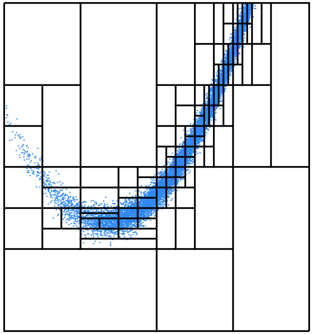
[Galán-de Anta+ 2023]

Various mathematical methods

- spline interpolation (B-spline/cubic/quintic, 1d/2d/3d);
- penalized (i.e., with automatic optimal smoothing) spline fitting and density estimation;
- N-dimensional integration (using cubature or cuba libraries);
- drawing uniform-weight samples from an arbitrary N-dim probability function (rejection sampling with adaptive domain refinement).







Other features, caveats and limitations

- AGAMA is not a general-purpose N-body simulation code, but it can be used in this role to some extent, utilizing orbit integration coupled with the `Multipole` potential expansion (aka BFE).
- It can create initial conditions for galaxy simulations, provide external potential for several simulation codes (NEMO / GYRFALCON, AMUSE, GADGET4, AREPO), and assist in the analysis (e.g., extract potential from snapshots, integrate orbits, etc.)
- User-defined `Python` functions can serve as density, potential, DF and selection functions along with built-in `C++` models, but are [much] less efficient; recommended approach is to approximate density or potential with `C++`-native expansions. A possible future development is to compile them as `Cython` code and use natively from the `C++` core.
- No universal support for differentiable programming (`Jax` autodiff does not propagate into `C++`), but some functions can provide analytic derivatives (orbit integration, DFs, and in the future action-angle transformations).

Software for galactic dynamics

	GALPY [Bovy 2015]	GALA [Price-Whelan 2017]	AGAMA [Vasiliev 2019]
density and potential profiles:			
collection of analytic models	+	+	+
solution of the Poisson equation for an arbitrary $\rho(\mathbf{r})$ or an N-body snapshot	+	+	+
numerical integration of orbits	+	+	+
conversion between position/velocity and action/angle variables	+	+	+
distribution functions and their moments	+	–	+
construction of equilibrium models	–	–	+
modelling of tidal streams	+	+	+
N-body simulations with BFE	–	–	+
integration with ASTROPY	+	+	–
language	Python, C	Cython	C++, Python

AGAMA potentials can be used in GALA & GALPY, although not most efficiently

Summary

AGAMA is a versatile toolbox for stellar dynamics catering to many needs:

- Extensive collection of gravitational potential models (analytic profiles, azimuthal- and spherical-harmonic expansions) constructed from smooth density profiles or N-body snapshots;
- Numerical orbit integration;
- Conversion to/from action/angle variables;
- Self-consistent multicomponent models with action-based DFs;
- Schwarzschild orbit-superposition models;
- Generation of initial conditions for N-body simulations;
- Various math tools: spline-based interpolation, fitting and density estimation, multidimensional sampling;
- Efficient and carefully designed C++ implementation, examples, Python and Fortran interfaces, plugins for Galpy, Gala, NEMO, AMUSE.

`https://github.com/GalacticDynamics-Oxford/Agama`