Connecting to observations with AGAMA

After including `obs.h` one can use several classes that are useful in connection with observational data. The code resides in six files: `obs_base.h`, `obs_dust.h`, `obs_los.h` and the corresponding `.cpp` files. All objects reside in the `obs` namespace, so in code they should carry the prefix `obs::`.

Sky coordinates

`PosSky` encodes (ra,dec) and (ℓ, b) sky coordinates. Possibilities are `PosSky p(40,30)` and `PosSky p(40,30,true)` – the first makes `p` the point with $\ell = 40, b = 30$ while the second makes `p` the point with $(\alpha = 40, \delta = 30)$ (degrees). The first component of `p` is `p.l` regardless of whether this actually refers to ℓ or α, and similarly for the number `p.b`.

`VelSky` encodes a proper motion, so `VelSky v(2,3)` sets the proper motion `v` to $(\dot{\ell}\cos b = 2, \dot{b} = 3)$ (mas/yr), while `VelSky v(2,3,true)` gives $(\dot{\alpha}\cos\delta = 2, \dot{\delta} = 3)$. The components are called `pv.mul` and `pm.mub` regardless of whether they are really (μ_α, μ_δ) or not.

`PosVelSky` encodes full astrometry – `PosVelSky pv(30,40,2,3)` makes `pv` into a struct that has components `pv.pos`, which is a `PosSky` and thus a component `pv.pos.b`, etc, and `pv.pm`, which is a `VelSky` so has a component `pv.pm.mul`. To store astrometry in the equatorial we write `PosVelSky pv(30,40,2,3,true)` – then `pv.pos.b` will be $\delta = 40$, etc. With `p` a `SkyPos` and `v` a `VelSky`, `PosVelSky pv(p,v)` will store in `pv` the astrometry in whichever coordinate system was used for `p` and `v`.

We have a series of translation functions

`PosSky p(from_RAdec(double ra,double dec))  PosSky p(from_RAdec(PosSky peq))` yield (ℓ, b) positions.

`PosSky peq(to_RAdec(double l,double b))  PosSky peq(to_RAdec(p))` yield (α, δ) positions.

`PosVelSky pv(from_RAdec(ra,dec,mura, mudec))` with `ra,dec,mura,mudec` doubles and `PosVelSky pv(from_RAdec(pveq))` with `pveq` (α, δ) astrometry yield astrometry in Galactic coords.

Conversely `PosVelSky pveq(to_RAdec(l,b,mul,mub))` and `PosVelSky pveq(to_RAdec(pv))` with `pv` (ℓ, b) astrometry yield astrometry in equatorial coords.

Heliocentric coordinates

If `intUnits` is an instance of `units::InternalUnits` then `solarShifter sun(intUnits)` creates an object that encodes the position and velocity of the Sun. Invoked thus the Sun's position is $(x, y, z) = (-8.27, 0, 0.025)\,\mathrm{kpc}$ and its Galactocentric velocity is $(v_x, v_y, v_z) = (12, 249, 7)\,\mathrm{km\,s^{-1}}$ but these can be changed to anything by creating a `coord::PosVelCar xv` that gives the Cartesian coordinates of the Sun wrt the Galactic centre and then writing `solarShifter sun(intUnits,&xv)`. Observations of the LMC could be handled by making `xv` the position and velocity of the Sun wrt the centre of the LMC.

`coord::PosCar X(sun.xyz())` stores in `X` the Cartesian position of the Sun (internal units). `coord::VelCar V(sun.Vxyz())` stores in `V` the Cartesian velocity of the Sun (internal units).

A `solarShifter` provides several methods for moving between helio- and galacto-centric systems:

If `PosSky p` is in Galactic coordinates and `double sKpc` is a distance in kpc, then `coord::PosCar xyz(sun.toCar(p,sKpc))` stores in `xyz` the Galactocentric Cartesian coordinates asociated with `p,sKpc`.

Similarly if `v,Vlos` are a proper motion in Galactic coords and a line-of-sight velocity in km/s, we can write `coord::PosVelCar xv(sun.toCar(p,sKpc,v,Vlos))`. We can also write `coord::PosVelCyl Rv(sun.toCyl(p,sKpc,v,Vlos))` to get the corresponding cylindrical coordinates.

Given a 3d point `p` in either cylindrical or Cartesian coords, the inverse `PosSky lb(sun.toSky(p,Skpc))` returns sky position `lb` (degrees) and distance `sKpc` (kpc). Similarly, given 6d coordinates `pv` (in either cylindrical or Cartesian coords) `PosVelSky lbpm(sun.toSky(pv,sKpc,Vlos))` returns (ℓ, b) astrometry, distance and line-of-sight velocity (kpc, km/s). If `pv` gives 6d coords in either cylindrical or Cartesian coords, we can extract just the proper motion and line-of-sight velocity: `VelSky v(sun.toPM(pv,Vlos)))`.

If `p` is a 3d position in either Cylindrical or Cartesian coords, `double s=sun.sKpc(p)` stores in `s` the distance to `p` (kpc).

Dust

A `BaseDustModel` stores a rule for returning the dust density at any location, which probably suffices for observations of external galaxies. `BaseDustModel dsty(5,0.1,norm,12,2,fromKpc)` will create an exponential dust layer with scale length $R_\mathrm{d} = 5\,\mathrm{kpc}$, scale-height $z_0 = 0.1\,\mathrm{kpc}$ that has an integral sign ($m = 1$) warped starting at $R_w = 12\,\mathrm{kpc}$ that reaches to $\pm h_w = 2\,\mathrm{kpc}$. The dust density can be scaled up or down by adjusting the `double norm`, while `fromKpc = intUnits.from_Kpc` is the linear scaling.

If `potential::PtrDensity gasDens` specifies a gas density, then `BaseDustModel dst(gasDens,norm,fromKpc)` makes the dust density `norm` times the gas density.

If `p` is a 3d position in cylindrical coordinates, `double rho = dsty.dens(p)` store in `rho` the dust density at `p`.

From `BaseDustModel` we derive a more complex class `DustModel` for observations of our Galaxy. The creators of this class have a `solarShifter` as an argument and in addition to one of the global dust patterns inherited from by `BaseDustModel` one can add any number of blobs, spirals or clouds.

`DustModel dsty(Rd,z0,dAvds,Rw,Hw,&sun)` or `DustModel dsty(gasDens,dAvds,&sun)` or make `dsty` a `DustModel` that's just the corresponding `BaseDustModel` with a specified location of the Sun and the dust density normalised to `dAvds` A_V mag per kpc at the Sun.

`dsty.dens(p)` with `p` a `PosCyl` returns the dust density at `p`.

`dsty.addBlob(p,Ampl,rad)` adds a Gaussian blob of density centred on `p`, `Ampl` and `rad` being the amplitude and scale radius of the blob:

$$\rho(R, z, \phi) = A \exp\left[\{(R - p_R)^2 + y^2 + (z - p_z)^2\}/2\mathrm{rad}^2\right]$$

where $y \equiv p_R(\phi - p_\phi)$.

`deleteBlob(n)` deletes the nth blob – `deleteBlob()` deletes the blob added last.

`addSpiral(Ampl,phase,alpha,Narms,kz)` adds a global `Narms` log-spiral

$$\rho(R, z, \phi) = A \cos\left[\alpha \ln(R) - N_\mathrm{arms}(\phi - \phi_0)\right] \mathrm{e}^{-k_z|z|}$$

`deleteSpiral(n)` deletes the nth spiral – `deleteSpiral()` deleting the last spiral.

addCloud(Rc,phic,z0,norm,fname) adds a cloud centred on (Rc,phic) with (exponential) scale height z0 and peak density norm. The surface density of the cloud is read from the file fname and stored in a linear interpolator. The first line of the file should contain nx,ny,Q,Xmax,Ymax,amax, the grid size in the radial and azimuthal directions, Toomre's Q, the x, y extents covered by the grid and the peak density. The following numbers should be the density at grid points, the outer loop being over y. A suitable input file is computed by the method of Lulian & Toomre (1966) – see Binney (2020).

deleteCloud(n) deletes the nth cloud, etc.

Lines of sight

The classes for los in our Galaxy and los to distant galaxies are derived from the class BaseLos. A BaseLos provides methods to convert (ℓ, b) astrometry and a distance to a location p, and conversely. Similarly, it provides methods to convert a phase-space location into a distance and line-of-sight velocity. Finally, if a dust model is specified it can give the extinction in the V, B, R, H, K bands at distance s.

With Los an instance of any line of sight, the relevant methods are

Los.xyz(s) returns the PosCar at distance s (intUnits)

Los.Rzphi(s) returns the PosCyl at distance s (intUnits)

Los.s(p) returns the distance (intUnits) to p (PosCar PosCyl)

Los.sKpc(p) returns the distance (kpc) to p (PosCar PosCyl)

Los.sVlos(xv) returns a std::pair<double,double> comprising the distance and $V_{\rm los}$ (both in intUnits) of the phase-space point xv (PosVelCar or PosVelCyl)

Los.A_V(sKpc) returns the V band extinction to sKpc – Los.A_H(sKc) gives the H-band extinction, etc.

extLos is a class of los through distant external galaxies. Since this class is derived from BaseLos, instances of extLos inherit the methods of BaseLos. Distances are measured from the point on the los that's closest to the galaxy's centre, so they are often negative.

extLos Los(xs,ys,incl,D,from_Kpc) makes Los the line of sight that is xs (kpc) parallel to the the apparent major axis, ys (kpc) parallel to the apparent minor axis of a galaxy that has inclination incl (degrees) and is D Mpc from us. Created thus, there is no dust so all extinctions A_V etc vanish.

extLos Los(xs,ys,incl,D,from_Kpc,&dsty) where dsty is a BaseDustModel makes Los a los along which there is extinction. This creation call causes $A_V(s)$ to be tabulated and fitted to a cubic spline.

SunLos is a class for lines of sight from the Sun, normally through our Galaxy, but it could be used for lines of sight to nearby galaxies such as the LMC or M31.

SunLos Los(ps,Sun) with pos a sky position and Sun a solarShifter makes Los a dustless los at *pos*.

SunLos Los(ps,Sun,&dsty) with dsty a dustModel makes Los the corresponding dusty los.

Distribution functions

A new keyword PopFile for .ini files makes it possible to specify the photometric properties of the stellar population described by a distribution function (DF). If the file name given after

PopFile is valid, AGAMA seeks to read from the file the fraction of stars in some band that are brighter than a series of magnitudes (in ever-fainter order) (xx add colours). When a DF is created (DF_factory.cpp), the data are read and stored in a cubic-spline interpolator.

Interfacing with Observables

The following functions use either a los or the ability to specify magnitudes to interface with observations. These functions are specified in galaxymodel_base.h so in code their names should carry the prefix galaxymodel::

computeMomentsLOS(model,&Los,rho,mom1,mom2) with model a GalaxyModel, Los any los, rho and mom1 arrays of doubles and mom2 an array of coord::Vel2Cyl will compute $\langle\rho\rangle(s)$, $\langle\mathbf{v}\rangle(s)$ and $\langle v_iv_j\rangle(s)$ along Los. If Los is a SunLos, $\langle\rho\rangle$ includes an s^2 factor, so it has units of mass per sterad per unit distance. If Los is an extLos there is no s^2 factor and the units are mass per unit volume. The inclusion or not of the s^2 factor affects the amount by which fore- and background stars contribute to the velocity moments but not their units.

computeMomentsLOS has several optional parameters: the first three are uncertainties on the three moments, the next, bool separate, specifies whether the contributions from the components of model (bulge, stellar halo, thin disc, etc) should be reported separately. The next two specify the required relative error and the maximum number of DF evaluations, and the final two, bright and faint, limit the stars to be considered to a range of *apparent* magnitudes. Apparent magnitudes are connected to absolute magnitudes using the distance and extinction provided by Los. For example
computeMomentsLOS(model, los, &dens[0], &Vphi[0], &sigmas[0], NULL, NULL, NULL, true, 1e-3, 1e5, 8, 15); with std::vector<double> dens(nc),Vphi(nc) and std::vector<coord::Vel2Cyl> sigmas(nc) will compute moments, separated into the nc components, accurate to 0.1% for stars between mags 8 and 15.

sampleVelocity(model,Rzphi,N,&Los,8,15) with Rzphi a PosCyl will return (as std::vector<coord::VelCyl>) the velocities of N mock stars at the coord::PosCyl Rzphi that have apparent magnitudes between 8 and 15 given the extinction values provided by Los. The contributions of individual components can be separated by calling this function multiple times with model defined to include only one component's DF. Since disc DFs vanish for $v_\phi < 0$, there is an analogous function sampleHalfVelocity that samples the interesting half of velocity space.

sampleVelocity_LF(model,Rzphi,N,&Los,8,15) returns an apparent magnitude in addition to a velocity for each mock star (as std::pair<coord::VelCyl,double>).

sampleLOS(model,&Los,N,8,15) returns N phase-space locations (std::vector<coord::PosVelCyl>) for stars between apparent mags 8 and 15. The s^2 factor is included when Los is a SunLos. The last two parameters are optional; omitting them will remove restriction on apparent magnitudes.

sampleLOS_LF(model,&Los,N,8,15) returns an apparent magnitude in addition to a phase-space location for each mock star (as std::pair<coord::PosVelCyl,double>).

sampleLOSsVlos(model,&Los,N,8,15) returns just distances and $V_{\rm los}$ values for mock stars (as sd::vector<std::pair<double,double> >).