ASTR400B Leach

Colin Leach

May 22, 2020

CONTENTS:

This is documentation for code written during course ASTR 400B, Theoretical Astrophysics, running at the University of Arizona's Steward Observatory, Spring 2020.

Instructor: Prof. Gurtina Besla, **TA:** Rixin Li

GitHub: https://github.com/colinleach/400B_Leach

> **Warning** This is a student project. I will try to make it as professional as possible, but let's be realistic in our expectations.

CHAPTER

ONE

GALAXY CLASS

This will read in a data file for a given galaxy and snap, returning the data in a variety of formats.

class galaxy.galaxy.**Galaxy**(*name*, *snap=0*, *datadir=None*, *usesql=False*, *ptype=None*, *stride=1*)

A class to find, read and manipulate files for a single galaxy.

Args:

name (str): short name used in filename of type 'name_000.txt', eg 'MW', 'M31'.

snap (int): Snap number, equivalent to time elapsed. Zero is starting conditions.

datadir (str): Directory to search first for the required file. Optional, and a default list of locations will be searched.

usesql (bool): If True, data will be taken from a PostgreSQL database instead of text files.

ptype (int or list): Optional. Restrict data to this particle type, for speed. Only valid with usesql=True.

stride (int): Optional. For stride=n, get every nth row in the table. Only valid with usesql=True.

Class attributes:

filepath (*pathlib.Path* object): directory containing the data file

filename (str): in *name_snap.txt* format, something like 'MW_000.txt'

data (np.ndarray): type, mass, position_xyz, velocity_xyz for each particle

read_db(*ptype*, *stride*)

Get relevant data from a PostgreSQL database and format it to be identical to that read from test files.

Args:

ptype (int): Optional. Restrict data to this particle type.

stride (int): Optional. For stride=n, get every nth row in the table.

Changes: *self.time*, *self.particle_count* and *self.data* are set.

Returns: nothing

get_filepath(*datadir*)

Args: datadir (str): path to search first for the required file

Returns: *pathlib.Path* object. A directory containing the file.

Raises: FileNotFoundError

Pretty boring housekeeping code, but may make things more resilient.

read_file()

Read in a datafile in np.ndarray format, store in *self.data*.

Requires: *self.path* and *self.filename* are already set.

Changes: *self.time*, *self.particle_count* and *self.data* are set.

Returns: nothing

type2name (*particle_type*)
Args: particle_type (int): valid values are 1, 2, or 3

Returns: typename (str): 'DM', 'disk' or 'bulge'

name2type (*typename*)
Args: typename (str): valid values are 'DM', 'disk' or 'bulge'

Returns: particle_type (int): 1, 2, or 3 as used in data files

filter_by_type (*particle_type*, *dataset=None*)
Subsets the data to a single particle type.

Args: particle_type (int): for particles, 1=DM, 2=disk, 3=bulge dataset (array including a type column): defaults to self.data

Kwargs: dataset (np.ndarray): optionally, a starting dataset other than self.data

Returns: np.ndarray: subset data

single_particle_properties (*particle_type=None*, *particle_num=0*)
Calculates distance from the origin and magnitude of the velocity.

Kwargs:

particle_type (int): a subset of the data filtered by 1=DM, 2=disk, 3=bulge

particle_num (int): zero-based index to an array of particles

returns:

3-tuple of Euclidean distance from origin (kpc), Euclidean velocity magnitude (km/s), particle mass (M_sun)

all_particle_properties (*particle_type=None*, *as_table=True*)
Calculates distances from the origin and magnitude of the velocities for all particles (default) or a specied particle type.

Kwargs:

particle_type (int): A subset of the data filtered by 1=DM, 2=disk, 3=bulge

as_table (boolean): Return type. If True, an astropy QTable with units. If False, np.ndarrays for position and velocity

Returns: QTable: The full list, optionally with units, optionally filtered by type.

component_count (*particle_type=None*)

Kwargs: particle_type (int): a subset of the data filtered by 1=DM, 2=disk, 3=bulge

Returns: Quantity: The number of particles in the galaxy of this type

all_component_counts ()

Returns: list: The number of particles of each type in the galaxy Ordered as [halo, disk, bulge]

component_mass (*particle_type=None*)

Kwargs: particle_type (int): a subset of the data filtered by 1=DM, 2=disk, 3=bulge

Returns: Quantity: The aggregate mass of all particles in the galaxy of this type

all_component_masses()

Returns: list: The aggregate masses of particles of each type in the galaxy

get_array()
Returns: all particle data in *np.ndarray* format

Pretty superfluous in Python (which has no private class members)

get_df()
Returns: data as pandas dataframe

get_qtable()
Returns: data as astropy QTable, with units

xyz()
Returns position as a (3,N) array

vxyz()
Returns velocity as a (3,N) array

m()
Conventience method to return array of masses

CHAPTER

TWO

GALAXIES CLASS

This stores and manipulates data for multiple galaxies and snaps.

class galaxy.galaxies.**Galaxies**(*names=('MW', 'M31', 'M33')*, *snaps=(0, 0, 0)*, *datadir=None*, *usesql=False*, *ptype=None*, *stride=1*)

A class to manipulate data for multiple galaxies.

Kwargs:

names (iterable of str): short names used in filename of type 'name_000.txt', eg 'MW', 'M31'.

snaps (iterable of int): Snap number, equivalent to time elapsed. Zero is starting conditions.

datadir (str): Directory to search first for the required file. Optional, and a default list of locations will be searched.

usesql (bool): If True, data will be taken from a PostgreSQL database instead of text files.

ptype (int): Optional. Restrict data to this particle type, for speed. Only valid with usesql=True.

stride (int): Optional. For stride=n, get every nth row in the table. Only valid with usesql=True.

Class attributes:

path (*pathlib.Path* object): directory (probably) containing the data files

filenames (list of str): in *name_snap* format, something like 'MW_000' (no extension)

galaxies (dict): key is filename, value is the corresponding Galaxy object

read_data_files()

Attempts to create a Galaxy object for each name/snap combination set in *self.names* and *self.snaps*

No return value. Sets *self.galaxies*, a dictionary keyed on *name_snap*

get_pivot(*aggfunc*, *values='m'*)

Generic method to make a pandas pivot table from the 9 combinations of galaxy and particle type.

Args: aggfunc (str): 'count', 'sum', etc as aggregation method values (str): column name to aggregate

Returns: pandas dataframe

get_counts_pivot()

Pivots on *count('m)*.

Returns: pandas dataframe

get_masses_pivot()

Pivots on *sum('m)*.

Returns: pandas dataframe

get_full_df()
Combined data for all input files.

Returns: Concatenated pandas dataframe from all galaxies Includes 'name' and 'snap' columns

get_coms(*tolerance=0.1*, *ptypes=(1*, *2*, *3)*)
Center of Mass determination for all galaxies. Defaults to all particle types, but *ptypes=(2,)* may be more useful.

Args: tolerance (float): convergence criterion (kpc)

Returns: QTable with COM positions and velocities colnames: ['name', 'ptype', 'x', 'y', 'z', 'vx', 'vy', 'vz', 'R', 'V']

separations(*g1*, *g2*)
Position and velocity of galaxy g2 COM relative to g1 COM. Uses only disk particles for the COM determination.

Args: g1, g2 (str): galaxies matching entries in self.filenames

Returns: Dictionary containing relative position, distance, velocities in Cartesian and radial coordinates

total_com()
Center of Mass determination for the local group.

Uses all particles of all types. Position and velocity should be conserved quantities, subject to numerical imprecision in the sim.

Returns: position, velocity: 3-vectors

total_angmom(*origin*)
Calculate angular momentum summed over all particles in the local group, abot point *origin*.

Arg: origin (3-vector): x,y,z coordinates

Returns: angular momentum: 3-vector

CHAPTER

THREE

CENTEROFMASS CLASS

Determines position and velocity of the COM for a galaxy/particle type combination.

class galaxy.centerofmass.**CenterOfMass**(*gal*, *ptype=2*)

Class to define COM position and velocity properties of a given galaxy and simulation snapshot

Args:

gal (Galaxy object): The desired galaxy/snap to operate on

ptype (int): for particles, 1=DM/halo, 2=disk, 3=bulge

Throws: ValueError, if there are no particles of this type in this galaxy (typically, halo particles in M33)

com_define(*xyz*, *m*)

Function to compute the center of mass position or velocity generically

Args:

xyz (array with shape (3, N)): (x, y, z) positions or velocities

m (1-D array): particle masses

Returns: 3-element array, the center of mass coordinates

com_p(*delta=0.1*, *vol_dec=2.0*)

Function to specifically return the center of mass position and velocity .

Kwargs: delta (tolerance)

Returns: One 3-vector, coordinates of the center of mass position (kpc)

com_v(*xyz_com*)

Center of Mass velocity

Args: X, Y, Z positions of the COM (no units)

Returns: 3-Vector of COM velocities

center_com(*com_p=None*, *com_v=None*)

Positions and velocities of disk particles relative to the CoM

Returns [two (3, N) arrays] CoM-centric position and velocity

angular_momentum(*com_p=None*, *com_v=None*, *r_lim=None*)

Returns:

L [3-vector as array] The (x,y,x) components of the angular momentum vector about the CoM, summed over all disk particles

pos, v [arrays with shape (3, N)] Position and velocity for each particle

r_lim [float] Radius to include in calculation (implicit kpc, no units)

rotate_frame(*to_axis=None*, *com_p=None*, *com_v=None*, *r_lim=None*)

Arg: to_axis (3-vector) Angular momentum vector will be aligned to this (default z-hat)

Returns: (positions, velocities), two arrays of shape (3, N) New values for every particle. *self.data* remains unchanged.

Based on Rodrigues' rotation formula Ref: https://en.wikipedia.org/wiki/Rodrigues%27_rotation_formula

shell_h(*radii*, *m*, *xyz*, *vxyz*)
Calculate specific angular momentum of spherical shells.

Args:

radii ((M,0) array of float): boundaries of shells (implicit kpc from center)

m ((N,) array of float): masses (implicit Msun)

xyz, vxyz ((3,N) arrays of float): positions and velocities (implicit kpc, km/s)

Returns:

rad ((M-1,) array): midpoints of shells (implicit kpc)

L ((M-1,3) array of float): total angular momentum

h ((M-1,3) array of float): specific angular momentum, i.e. L/m

sphere_h(*radii*, *m*, *xyz*, *vxyz*)
Calculate specific angular momentum within spheres.

Args:

radii ((M,0) array of float): boundaries of shells (implicit kpc from center)

m ((N,) array of float): masses (implicit Msun)

xyz, vxyz ((3,N) arrays of float): positions and velocities (implicit kpc, km/s)

Returns:

L ((M,3) array of float): total angular momentum

h ((M,3) array of float): specific angular momentum, i.e. L/m

disp_by_radius(*x*, *vy*, *xbins*, *binwidth=None*)
Calculate mean velocity and dispersion sigma for a set of radius bins

Args:

x (array of float): lateral distance from CoM

vy (array of float): radial velocity

xbins (array of float) midpoints of equally-spaced bins

binwidth (float): optional and probably useless

CHAPTER

FOUR

MASSPROFILE CLASS

Calculates mass vs. radius relations and rotation curves for a given galaxy.

class galaxy.massprofile.**MassProfile**(*gal*, *com_p=None*)

Class to define mass enclosed as a function of radius and circular velocity profiles for a given galaxy and simulation snapshot

Args:

gal (Galaxy object): The desired galaxy/snap to operate on

com_p (3-vector): Optional. The position of the disk CoM.

mass_enclosed(*radii*, *ptype=None*)

Calculate the mass within a given radius of the CoM for a given type of particle.

Args: radii (array of distances): spheres to integrate over ptype (int): particle type from (1,2,3), or None for total

Returns: array of masses, in units of M_sun

mass_enclosed_total(*radii*)

Calculate the mass within a given radius of the CoM, summed for all types of particle.

Args: radii (array of distances): spheres to integrate over

Returns: array of masses, in units of M_sun

halo_mass()

Utility function to get a parameter for Hernquist mass

hernquist_mass(*r*, *a*, *M_halo=None*)

Calculate the mass enclosed for a theoretical profile

Args:

r (Quantity, units of kpc): distance from center

a (Quantity, units of kpc): scale radius

M_halo (Quantity, units of M_sun): total DM mass (optional)

Returns: Total DM mass enclosed within r (M_sun)

circular_velocity(*radii*, *ptype=None*)

Calculate orbital velocity at a given radius from the CoM for a given type of particle.

Args:

radii (array of distances): circular orbit

ptype (int): particle type from (1,2,3), or None for total

Returns: array of circular speeds, in units of km/s

circular_velocity_total(*radii*)
Syntactic sugar for circular_velocity(radii, ptype=None)

circular_velocity_hernquist(*radii*, *a*, *M_halo=None*)
Theoretical V_circ assuming halo mass follows a Hernquist profile

Args:

radii (array of distances): circular orbit

a (Quantity, units of kpc): scale radius

M_halo (Quantity, units of M_sun): total DM mass (optional)

fit_hernquist_a(*r_inner=1*, *r_outer=30*, *get_details=False*)
Get *scipy.optimize* to do a non-linear least squares fit to find the best scale radius *a* for the Hernquist profile.

Args:

r_inner (numeric): Optional. Minimum radius to consider (implicit kpc). Avoid values < 1 as they cause numeric problems.

r_outer (numeric): Optional. Maximum radius to consider (implicit kpc).

sersic(*R*, *Re*, *n*, *Mtot*)
Function that returns Sersic Profile for an Elliptical System (See in-class lab 6)

Input

R: radius (kpc)

Re: half mass radius (kpc)

n: sersic index

Mtot: total stellar mass

Returns Surface Brightness profile in Lsun/kpc^2

bulge_Re(*R*)
Find the radius enclosing half the bulge mass.

Args:

R (array of Quantity): Radii to consider (kpc)

Returns:

Re (Quantity) : Radius enclosing half light/mass (kpc)

bulge_total (numeric): Mass of entire bulge (M_sun, no units)

bulgeI (array of Quantity): Surface brightness at radii R (kpc^-2), assuming M/L=1

fit_sersic_n(*R*, *Re*, *bulge_total*, *bulgeI*)
Get *scipy.optimize* to do a non-linear least squares fit to find the best value of *n* for a Sersic profile.

Args:

R (array of quantity): Radii at which to calculate fit (kpc)

Re (Quantity) : Radius enclosing half light/mass (kpc)

bulge_total (numeric): Mass of entire bulge (M_sun, no units)

bulgeI (array of Quantity): Surface brightness at radii R (kpc^-2)

Returns: best *n* value and error estimate

density_profile_shell(*radii*, *m*, *xyz*)
Calculates mass density in successive spherical shells

Arg:

radii (array of float): boundary values beteen shells (implicit kpc, no units)

m (shape (N,) array of float): particle masses (implicit Msun, no units)

xyz ((3,N) array of float): particle cartesian coordinates

Returns:

r_annuli: geometric mean of boundaries (array is one shorter than input radii)

rho: densities (Msun/kpc^3) (same length as r_annuli)

density_profile_sphere(*radii*, *m*, *xyz*)
Calculates average mass density within successive spherical radii

Arg:

radii (array of float): boundary values beteen shells (implicit kpc, no units)

m (shape (N,) array of float): particle masses (implicit Msun, no units)

xyz ((3,N) array of float): particle cartesian coordinates

Returns:

rho: densities (Msun/kpc^3) (same length as radii)

virial_radius(*r_min=20*, *r_max=1000*, *rho_c=None*)
Calculates radius where DM density falls to 200x critical density for the universe.

Args:

r_min, r_max (floats) optional, limits for search (implicit kpc, no units)

rho_c (float or Quantity) optional, critical density for chosen cosmology

Returns: r_200 (float): virial radius, implicit kpc

virial_mass(*r_200=None*, *ptype=None*)
Mass enclosed by the virial radius

CHAPTER

FIVE

TIMECOURSE CLASS

Various methods to work with data across a series of snaps (timepoints).

class galaxy.timecourse.**TimeCourse**(*datadir='.'*, *usesql=False*)

A commection of methods for generating, reading and writing summary data for parameters that change over the timecourse of the simulation.

These fall into a few groups:

write_xxx() [] Methods that loop through the raw data, calculate parameters and write the results to file. Can be slow (hours) to run but Only run once. See the *data* folder for the resulting files, one line per snap.

read_xxx_file()` **:** Read in the summary files and return a numpy array. These rely on the generic *read_file()* method.

read_xxx_db() **:** Get the summary data from postgres instead of a text file. The returned array should be identical to the *read_xxx_file()* group.

write_db_tables() **:** Read a text file, insert the contents to a postgres table.

get_one_com() **:** Convenience method to return a single CoM position.

write_com_ang_mom(*galname*, *start=0*, *end=801*, *n=5*, *show_progress=True*)

Function that loops over all the desired snapshots to compute the COM pos and vel as a function of time.

inputs:

galname (str): 'MW', 'M31' or 'M33'

start, end (int): first and last snap numbers to include

n (int): stride length for the sequence

datadir (str): path to the input data

show_progress (bool): prints each snap number as it is processed

returns: Two text files saved to disk.

write_total_com(*start=0*, *end=801*, *n=1*, *show_progress=True*)

Function that loops over all the desired snapshots to compute the overall COM pos and vel as a function of time. Uses all particles in all galaxies.

inputs:

start, end (int): first and last snap numbers to include

n (int): stride length for the sequence

show_progress (bool): prints each snap number as it is processed

output: Text file saved to disk.

write_total_angmom(*start=0*, *end=801*, *n=1*, *show_progress=True*)
Function that loops over all the desired snapshots to compute the overall angular momentum as a function of time. Uses all particles in all galaxies.

inputs:

start, end (int): first and last snap numbers to include

n (int): stride length for the sequence

show_progress (bool): prints each snap number as it is processed

output: Text file saved to disk.

write_vel_disp(*galname*, *start=0*, *end=801*, *n=1*, *show_progress=True*)
Function that loops over all the desired snapshots to compute the veocity dispersion sigma as a function of time.

inputs:

galname (str): 'MW', 'M31' or 'M33'

start, end (int): first and last snap numbers to include

n (int): stride length for the sequence

datadir (str): path to the input data

show_progress (bool): prints each snap number as it is processed

returns: Text file saved to disk.

write_LG_normal(*start=0*, *end=801*)
Calculates the normal to a plane containing the three galaxy CoMs.

Args:

start, end (int): first and last snap numbers to include

output: Text file saved to disk.

write_rel_motion(*start=0*, *end=801*)

read_file(*fullname*)
General method for file input. Note that the format is for summary files, (one line per snap), not the raw per-particle files.

read_com_file(*galaxy*, *datadir='.'*)
Get CoM summary from file.

Args:

galaxy (str): 'MW', 'M31', 'M33'

datadir (str): path to file

Returns: np.array with 802 rows, one per snap

read_angmom_file(*galaxy*, *datadir='.'*)
Get CoM summary from file.

Args:

galaxy (str): 'MW', 'M31', 'M33'

datadir (str): path to file

Returns: np.array with 802 rows, one per snap

read_total_com_file(*datadir='.'*)
Get CoM summary from file.

Args:

datadir (str): path to file

Returns: np.array with 802 rows, one per snap

read_normals_file(*datadir='.'*)
Get normals to plane containing 3 galaxy CoMs from file.

Args:

datadir (str): path to file

Returns: np.array with 802 rows, one per snap

read_relmotion_file(*datadir='.'*)
Get relative CoM distances/velocities from file.

Args:

datadir (str): path to file

Returns: np.array with 802 rows, one per snap

write_db_tables(*datadir='.', do_com=False, do_angmom=False, do_totalcom=False, do_totalangmom=False, do_normals=False, do_sigmas=False, do_relmotion=False*)
Adds data to the various tables in the *galaxy* database

read_com_db(*galaxy=None, snaprange=(0, 801)*)
Retrieves CoM positions from postgres for a range of snaps.

Args:

galaxy (str): Optional, defaults to all. Can be 'MW', 'M31' , 'M33'

snaprange (pair of ints): Optional, defaults to all. First and last snap to include. This is NOT the [first, last+1] convention of Python.

read_angmom_db(*galaxy=None, snaprange=(0, 801)*)
Retrieves disk angular momentum from postgres for a range of snaps.

Args:

galaxy (str): Optional, defaults to all. Can be 'MW, 'M31 , 'M33'

snaprange (pair of ints): Optional, defaults to all. First and last snap to include. This is NOT the [first, last+1] convention of Python.

read_total_com_db(*snaprange=(0, 801)*)
Retrieves total CoM positions from postgres for a range of snaps.

Args:

snaprange (pair of ints): Optional, defaults to all. First and last snap to include. This is NOT the [first, last+1] convention of Python.

get_one_com(*gal, snap*)
Gets a CoM from postgres for the specified galaxy and snap.

Args:

gal (str): Can be 'MW, 'M31 , 'M33'

snap (int): The timepoint.

read_total_angmom_db(*snaprange=(0, 801)*)
Gets the total angular momentum of the 3-galaxy system. In practice, this turns out to be near-zero at all timepoints and can be ignored in future work.

read_normals_db(*snaprange=(0, 801)*)
Gets the normals to the 3-galaxy plane.

read_sigmas_db(*galaxy=None*, *snaprange=(0, 801)*)
Gets the velocity dispersions (km/s) for one galaxy at a range of snaps.

read_relmotion_db(*snaprange=(0, 801)*)
Retrieves relative CoM positions and velocities from postgres for a range of snaps.

Args:

snaprange (pair of ints): Optional, defaults to all. First and last snap to include. This is NOT the [first, last+1] convention of Python.

snap2time(*snap*)

time2snap(*time*)

Arg: time (float): value in Gyr

Returns: List of closest values, often but not reliably just one.

CHAPTER

SIX

UTILITIES MODULE

A collection of useful functions that don't fit into any of the other classes.

galaxy.utilities.**wolf_mass**(*sigma*, *Re*)

Wolf mass estimator from Wolf+ 2010

Args:

sigma [] 1D line of sight velocity dispersion in km/s

Re [] 2D radius enclosing half the stellar mass in pc

Returns: estimate of the dynamical mass within the half light radius in Msun

galaxy.utilities.**sersic**(*R*, *Re*, *n*, *Mtot*)

Function that returns Sersic Profile for an Elliptical System (See in-class lab 6)

Input

R: radius (kpc)

Re: half mass radius (kpc)

n: sersic index

Mtot: total stellar mass

Returns Surface Brightness profile in Lsun/kpc^2

galaxy.utilities.**HernquistM**(*r*, *a=<Quantity 60. kpc>*, *M_halo=<Quantity 1.97e+12 solMass>*)

Args: r (Quantity, units of kpc): distance from center a (Quantity, units of kpc): scale radius M_halo (Quantity, units of M_sun): total DM mass

Returns: Total DM mass enclosed within r (M_sun)

galaxy.utilities.**jacobi_radius**(*r*, *M_host*, *M_sat*)

The Jacobi Radius for a satellite on a circular orbit about an extended host, where the host is assumed to be well modeled as an isothermal sphere halo:

R_j = r * (M_sat / 2 M_host(<r))}^(1/3)

For MW/LMC, the Isothermal Sphere approximation is not a bad one within 50 kpc.

In other contexts, can be called the Roche radius, Roche limit or Hill radius.

Args:

r: distance between stellite and host (kpc)

M_host: host mass enclosed within r (M_sun)

M_sat: satellite mass (M_sun)

returns: Jacobi radius (kpc)

galaxy.utilities.**jacobi_mass**(*Rj*, *r*, *Mhost*)

Function that returns min mass of a satellite given its observed size + distance from a massive host: Msat = (Rj/r)**3 * 2 * Mhost

Args:

Rj: Jacobi radius (approx as observed size) (kpc)

r: distance between stellite and host (kpc)

Mhost: mass enclosed within r (M_sun)

returns: Minimum mass Msat of a satellite given its current size (M_sun)

galaxy.utilities.**rotation_matrix_to_vector**(*old_axis*, *to_axis=None*)

Args:

old_axis (3-vector) Vector to be brought into alignment with *to_axis* by rotation about the origin

to_axis (3-vector) Angular momentum vector will be aligned to this (default z_hat)

Returns: 3x3 rotation matrix

Based on Rodrigues' rotation formula Ref: https://en.wikipedia.org/wiki/Rodrigues%27_rotation_formula

Note that orientation in the plane perpendicular to 'to_axis' is arbitrary

galaxy.utilities.**z_rotation_matrix**(*pt1*, *pt2*)

Rotates about z-axis to line up two given points along the x-axis

Args:

pt1, pt2 (2-component iterables) define points to be placed on the x-axis

Returns: 3x3 rotation matrix

galaxy.utilities.**find_nearest**(*array*, *value*)

Find the entry in *array* which is closest to *value* Modified from https://stackoverflow.com/questions/2566412/find-nearest-value-in-numpy-array

Returns: index and corresponding value

CHAPTER

SEVEN

DB CLASS

A wrapper for connections to the PostgreSQL database

class galaxy.db.**DB**

A simple wrapper class for connecting to the PostgreSQL database.

Takes no arguments. Relies on having connection information in *~/dbconn.yaml*.

read_params()

Needs the yaml parameter file to be in the user's home directory

get_cursor()

A simple getter method

run_query(*query*)

Runs a SQL query (typically SELECT)

Returns results in Python list format (not numpy, which would need a dtype list)

get_xyz(*gal*, *snap*)

CHAPTER

EIGHT

REMNANT CLASS

Treats the post-merger remnant of MW-M31 as a galaxy.

class galaxy.remnant.**Remnant**(*snap=801*, *datadir=None*, *usesql=False*, *stride=1*, *ptype=(2, 3)*)

A class to work with the MW-M31 post-merger remant.

Args:

snap (int): Snap number, equivalent to time elapsed. Defaults to the last timepoint.

datadir (str): Directory to search first for the required file. Optional, and a default list of locations will be searched.

usesql (bool): If True, data will be taken from a PostgreSQL database instead of text files.

stride (int): Optional. For stride=n, get every nth row in the table. Only valid with usesql=True.

ptype (int or iterable of int): Particle type: 1, 2, 3 or a combination of these

Class attributes:

data (np.ndarray): type, mass, position_xyz, velocity_xyz for each particle

read_db(*stride*)

Get relevant data from a PostgreSQL database and format it to be identical to that read from test files.

Ex-disk and ex-bulge particles are included, not DM particles.

Args:

stride (int): Optional. For stride=n, get every nth row in the table.

Changes: *self.time*, *self.particle_count* and *self.data* are set.

Returns: nothing

xyz()

Convenience method to get positions as a np.array of shape (3,N)

vxyz()

Convenience method to get velocities as a np.array of shape (3,N)

I_tensor(*m*, *x*, *y*, *z*)

Args:

m, x, y, z: 1-D arrays with mass and coordinates (no units)

Returns: 3x3 array representing the moment of inertia tensor

ellipsoid_axes(*m*, *x*, *y*, *z*, *r_lim=None*)

Args:

m, x, y, z: 1-D arrays with mass and coordinates (no units)

r_lim [float] Radius to include in calculation (implicit kpc, no units)

Returns: Two 3-tuples: relative semimajor axes and principal axis vectors

sub_mass_enclosed(*radii*, *m*, *xyz*)

Calculate the mass within a given radius of the origin. Based on code in MassProfile, but this version assumes CoM-centric coordinates are supplied.

Args: radii (array of distances): spheres to integrate over m (array of masses): shape (N,) xyz (array of Cartesian coordinates): shape (3,N)

Returns: array of masses, in units of M_sun

hernquist_mass(*r*, *a*, *M_halo*)

Calculate the mass enclosed for a theoretical profile

Args:

r (Quantity, units of kpc): distance from center

a (Quantity, units of kpc): scale radius

M_halo (Quantity, units of M_sun): total DM mass

Returns: Total DM mass enclosed within r (M_sun)

fit_hernquist_a(*m*, *xyz*, *r_inner=1*, *r_outer=100*)

Get *scipy.optimize* to do a non-linear least squares fit to find the best scale radius *a* for the Hernquist profile.

Args:

r_inner (numeric): Optional. Minimum radius to consider (implicit kpc). Avoid values < 1 as they cause numeric problems.

r_outer (numeric): Optional. Maximum radius to consider (implicit kpc).

sersic(*R*, *Re*, *n*, *Mtot*)

Function that returns Sersic Profile for an Elliptical System (See in-class lab 6)

Input

R: radius (kpc)

Re: half mass radius (kpc)

n: sersic index

Mtot: total stellar mass

Returns Surface Brightness profile in Lsun/kpc^2

Re(*R*, *m*, *xyz*)

Find the radius enclosing half the mass.

Args:

R (array of Quantity): Radii to consider (kpc)

Returns:

Re (Quantity) : Radius enclosing half light/mass (kpc)

bulge_total (numeric): Mass of entire bulge (M_sun, no units)

bulgeI (array of Quantity): Surface brightness at radii R (kpc^-2), assuming M/L=1

fit_sersic_n(*R*, *sub_Re*, *sub_total*, *subI*)

Get *scipy.optimize* to do a non-linear least squares fit to find the best value of *n* for a Sersic profile.

Args:

R (array of quantity): Radii at which to calculate fit (kpc)

Re (Quantity) : Radius enclosing half light/mass (kpc)

bulge_total (numeric): Mass of entire bulge (M_sun, no units)

bulgeI (array of Quantity): Surface brightness at radii R (kpc^-2)

Returns: best *n* value and error estimate

CHAPTER

NINE

APPROACHES CLASS

Convenience class to get concatenated data for all galaxies. Mainly used when they are in close proximity and can appear on the same plot.

class galaxy.approaches.**Approaches**(*snap=801*, *datadir=None*, *usesql=False*, *stride=1*, *ptype='lum'*)

A class to work with all 3 galaxies when in close proximity.

Args:

snap (int): Snap number, equivalent to time elapsed. Defaults to the last timepoint.

datadir (str): Directory to search first for the required file. Optional, and a default list of locations will be searched.

usesql (bool): If True, data will be taken from a PostgreSQL database instead of text files.

stride (int): Optional. For stride=n, get every nth row in the table. Only valid with usesql=True.

ptype (str): can be 'lum' for disk+bulge, 'dm' for halo

Class attributes:

data (np.ndarray): type, mass, position_xyz, velocity_xyz for each particle

read_db(*stride*)

Get relevant data from a PostgreSQL database and format it to be identical to that read from test files.

Args:

stride (int): Optional. For stride=n, get every nth row in the table.

Changes: *self.time*, *self.particle_count* and *self.data* are set.

Returns: nothing

xyz()

Convenience method to get positions as a np.array of shape (3,N)

vxyz()

Convenience method to get velocities as a np.array of shape (3,N)

CHAPTER

TEN

SURFACEDENSITYPROFILE CLASS

Calculate the surface density profile of a galactic disk.

class galaxy.surfacedensity.**SurfaceDensityProfile**(*gal*, *snap*, *radii=None*, *r_step=0.5*, *ptype=2*, *usesql=False*)

Calculate the surface density profile of a galaxy at a snapshot. Modified from code supplied by Rixin Li.

plot_xy(*figsize=(8, 8)*, *xlim=(0, 60)*, *ylim=(0, 60)*, *pfc='b'*, *ngout=False*, *fname=None*)

CHAPTER

ELEVEN

VDISP CLASS

To calculate velocity dispersions and rotation curves for the post-merger remnant of MW-M31.

class galaxy.remvdisp.**Vdisp**(*snap=801*, *ptype=(2*, *3)*, *r_lim=60*)

A class to work with rotation and velocity dispersion of the MW-M31 post-merger remant.

Args:

snap (int): Snap number, equivalent to time elapsed. Defaults to the last timepoint.

usesql (bool): If True, data will be taken from a PostgreSQL database instead of text files.

ptype (int or iterable of int): Particle type: 1, 2, 3 or a combination of these

calc_centered()

Sets the CoM position and velocity, plus particle coordinates centered on the CoM.

No args, no return value.

rotate(*r_lim*)

Creates transformed coordinates with the angular momentum vector along z.

Arg:

r_lim (float): Only consider particles within this radius when computing L-hat. Implicit kpc, no units.

calc_v_sigma(*pos_index*, *v_index*)

Calculate mean radial velocities and dispersions for bins along an axis.

Args:

pos_index, v_index (integers in (0,1,2)): Axis numbers for binning and for radial velocity. x=0, y=1, z=2

Returns: Binned v_radial and dispersions, v_max and central dispersion. All implicit km/s, no units.

set_xbins(*xbins*)

Sets new x-boundaries for binning, invalidates any previous calculations.

Args:

xbins (array of float): Distances from CoM along chosen axis (signed, not just radius)

set_yx()

Convenience method to call calc_v_sigma() with y-axis and x-velocities

set_zx()

Convenience method to call calc_v_sigma() with z-axis and x-velocities

plot_yx(*particles='Stellar'*, *xlim=(-40, 40)*, *ylim1=(-120, 120)*, *ylim2=(0, 200)*, *xbins=None*, *pngout=False*, *fname=None*)
Wrapper for Plots.plot_v-sigma()

plot_zx(*particles='Stellar'*, *xlim=(-40, 40)*, *ylim1=(-120, 120)*, *ylim2=(0, 200)*, *xbins=None*, *pngout=False*, *fname=None*)
Wrapper for Plots.plot_v-sigma()

CHAPTER

TWELVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

g

INDEX

H

I

J

M

N

P

R

S

T

V

W

X

Z