ASTR400B Leach

Colin Leach

Mar 24, 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): 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 aggregate masses 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

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*)

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

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

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

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)

CHAPTER

FIVE

TIMECOURSE CLASS

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

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

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.

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(*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

write_db_tables(*datadir='.'*, *do_com=False*, *do_angmom=False*, *do_totalcom=False*, *do_totalangmom=False*)
Adds data to the *centerofmass*, *angmom* and *totalcom* 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)*)

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*)

Input

R: radius (kpc)

Re: half mass radius (kpc)

n: sersic index

Mtot: total stellar mass

Returns Surface Brightness profile in Lsun/kpc^2

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)

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

g

INDEX

S

T

W