PYGadgetReader

Author: Robert Thompson

E-mail: rthompsonj@gmail.com

Contents

Summary

This module contains a function for reading SPH particle data into python. The relevant functions are described in the following sections. It currently supports these file types:

```
- Gadget binaries (type 1)
- TIPSY Binaries  (bin,aux,envira,future)
- HDF5 Gadget outputs
- PStarGroupFinder property files
```

REQUIREMENTS

```
- python2.7.x (not tested with other versions)
- numpy
- c compiler
- mercurial
- HDF5 for HDF5-read support
```

Obtaining the code

The easiest way to download the code and stay up to date is to clone a version from bitbucket to your local computer via Mercurial (hg). The repository is hosted here:

```
https://bitbucket.org/rthompson/pygadgetreader
```

Customization

Before we build the module, there are a few additional parameters that you may want to adjust.

1) In order to enable HDF5 support you must modify *setup.py* and indicate the location of your HDF5 installation. To enable HDF5 support you must set *HDF5_PRESENT* to 1, and modify *HDF5INCL* & *HDF5LIB* to point to your HDF5 installation.

```
HDF5_PRESENT = 1
HDF5INCL     = "/Users/bob/local/hdf5/include"
HDF5LIB      = "/Users/bob/local/hdf5/lib"
```

2) *pygadgetreader* allows the user to specify at compile time the default type of file one will be reading in. These options must be set before compilation, and by default they are commented out; they are located in *modules/vars.h*.

```
//#define KENCODE       //default = off, only uncomment this if you are using
K.Nagamine's Gadget.
//#define TIPSY         //default = off, uncomment this to read TIPSY files by
default
//#define ENABLE_HDF5   //default = off, uncomment if you want HDF5 support
//define HDF5_DEFAULT   //default = off, uncomment this to read HDF5 files by
default
```

KENCODE is mainly legacy support for Gadget Binary type 1 files, and defines a differnet *skips.h* file to allow for the proper reading of a different block format. If your version of gadget has a different block structure you are more than welcome to define your own *skips.h* file to suit your needs. If you have any questions regarding this proceedure feel free to email me.

TIPSY allows for the omission of tipsy=1 in your read commands (below). By uncommenting this the code will automatically try and read TIPSY files by default; one can still read GADGET files via passing tipsy=0 to the read commands.

ENABLE_HDF5 tells the code to include <hdf5.h> during compilation. This is needed if you want HDF5 support. HDF5_DEFAULT allows for the omission of hdf5=1 in your read commands (below). By uncommenting this the code will automatically try and read HDF5 files by default; one can still read GADGET files via passing hdf5=0 to the read commands.

Installation

Once the code is downloaded there are two methods of installation depending on your access rights. If you have write access to your python distribution, then the preferred method is to execute the following commands:

```
python setup.py build      ## this builds the module
python setup.py install    ## this installs the module, may require sudo
```

If you do *not* have write access to your python install we can build the module in place via:

```
python setup.py build_ext --inplace
```

this will produce a *readgadget.so* file in the root directory of pygadget reader. If you elect to use this method then you should add the location of the *readgadget.so* file to your PYTHONPATH environment variable.

Usage

There are multiple functions contained within this module, I will go through each and its options below. In order to use this module in your python scripts one should add the following to the top of their scripts:

```
from readgadget import *
```

All functions have a few universal arguments:

```
units=0             //converts some units to more physical realizations
nth_particle=1      //only reads in ever Nth particle (readsnap() only)
tipsy=0             //reads in tipsy binary format
future=0            //specifies a specific future file (TIPSY ONLY)
hdf5=0              //reads in HDF5 binary format
debug=0             //prints out debug statements for reading
supress_output=0    //supresses output messages
numfiles=1          //depreciated, now read from the header
```

readhead( )

This function reads in the header and returns values of interest. The values it can read in are as follows:

```
 time          - scale factor of the snapshot
 redshift      - redshift of the snapshot
 boxsize       - boxsize if present in units of kpc/h
 O0             - Omega_0 (Omega_dm+Omega_m)
 Ol             - Omega_Lambda
 h              - hubble parameter
 gascount      - total # of gas   particles [type 0]
 dmcount       - total # of DM    particles [type 1]
 diskcount     - total # of disk  particles [type 2]
 bulgecount    - total # of bulge particles [type 3]
 starcount     - total # of star  particles [type 4]
 bndrycount    - total # of bndry particles [type 5]
 f_sfr         - Star Formation Rate flag    0=off 1=on
 f_fb          - Feedback flag               0=off 1=on
 f_cooling     - Cooling flag                0=off 1=on
 f_age         - Stellar Age tracking flag   0=off 1=on
 f_metals      - Metal tracking flag         0=off 1=on


Definition:   readhead('a','b',tipsy=0,debug=0)

      Parameters
      ----------
      a : Input file.
          Must be input as a string and enclosed in ' ' - see examples.
      b : Value of interest from the above list.
          Must be input as a string and enclosed in ' ' - see examples.

      Optional
      --------
      numfiles: DEPRECIATED!  No longer needed, but still here for backwards
compatability with previous scripts:
      Number of files the snapshot is broken up into. Assumed to be 1 if it is
not included.


Example:
      z=readhead('snap_001','redshift')
            - reads redshift value and assigns it to the z variable

      h=readhead('snap_005','h')
      boxsize=(readhead('snap_005','boxsize',numfiles=2)/1000/h)**3
            - reads hubble param and assigns it to the h variable, then reads in
the boxsize,
              converts it to Mpc^3.
```

readsnap( )

This function does the bulk of the work. It reads data blocks from the snapshot file and returns the requested data for a specified particle type.

```
   Supported data blocks are:

   --GADGET & TIPSY--
   pos          - (all)         Position data
   vel          - (all)         Velocity data in km/s
   pid          - (all)         Particle ids
   mass         - (all)         Particle masses
   u            - (gas)         Internal energy
   rho          - (gas)         Density
   ne           - (gas)         Number density of free electrons
   nh           - (gas)         Number density of neutral hydrogen
   hsml         - (gas)         Smoothing length of SPH particles
   sfr          - (gas)         Star formation rate in Msun/year
   delaytime    - (gas)         DelayTime (>0 member of wind)
   fH2          - (gas)         Fractional Abundance of molecular hydrogen
   Sigma        - (gas)         Approximate surface density @ each particle
(HIdensity * scale_height)
   age          - (stars)       Formation time of stellar particles (in terms of
the scale factor)
   z            - (gas & stars) Metallicty of gas & star particles (returns total
Z)
   tmax         - (gas & stars) Maximum temp
   nspawn       - (gas & stars) Number of star particles spawned
   potential    - (all)         Potential of particles

   metals       - (gas & stars) NMETALS array [C,O,Si,Fe]

   --TIPSY--
   s_age        - (aux)    stellar age

   mhalo        - (envira) mass of parent SKID halo
   windage      - (envira) time since last launched in a wind for gas
   rvir         - (envira) ??
   vvir         - (envira) ??

   starfrac     - (future) ??
   relaunch     - (future) ??
   rho          - (future) gad density in the future
   u            - (future) gas particle temp in the future
   z            - (future) metal value of particle in the future
```

```
   Supported particle types (note tipsy only returns gas/dm/star particles):
   gas         - Gas
   dm          - Dark Matter
   disk        - Disk particles
   bulge       - Bulge particles
   star/stars  - Star particles
   bndry       - Boundary particles


Definition:
readsnap('a','b','c',numfiles=0,units=0,tipsy=0,future=0,debug=0,nth_Particle =
1)

    Parameters
    ----------
    a: Input file.
       Must be input as a string and enclosed in ' ' - see examples.
    b: Data block you are interested in (see above list)
       Must be input as a string and enclosed in ' ' - see examples.
    c: Particle type you are interested in (see above list)
       Must be input as a string and enclosed in ' ' - see examples.

    Optional
    --------
    numfiles: Number of files the snapshot is broken up into. Assumed to be 1 if
it is not included. (obsolete)
       units: Can either be 0 for code units or 1 for real units.  Assumed to be
0 if not included.
              This parameter allows for the data to be returned in real units(1)
rather than code units(0).
              Currently only active for density (rho), internal energy
              (u - returns temperature in K), Mass (returns Msun), and Sigma
(returns g/cm^2).
       tipsy: Must be set to 1 if reading in tipsy binary/aux/envira files.
      future: integer value for the future file.
        hdf5: Must be set to 1 if reading in HDF5 binary files.
       debug: Shows debug information
nth_Particle: Allows the user to read in a subset of data rather than the full
dataset


Example:
    DMpos=readsnap('snap_001','pos','dm')
    DMx,DMy,DMz=hsplit(DMpos,3)
        - reads in dark matter data and returns an Nx3 array containing
positions.
```

```
          hsplit is then used to split the array into x,y,z positions.

     grho=readsnap('snap_005','rho','gas',numfiles=2,units=1)
    gtemp=readsnap('snap_005','u','gas',numfiles=2,units=1)
        - reads a multi-file snapshot (2) and returns density and temperature in
cgs units.
```

galprop( )

This function reads in property files output by P-StarGroupFinder. It returns one of the following:

```
 mstar         - Mass of the stars within a group
 bmag          - B-magnitude of each group
 imag          - I-magnitude of each group
 vmag          - V-magnitude of each group
 kmag          - K-magnitude of each group
 cm            - Center of mass positions of each group
 sfr           - Instantanious star formation rate of each group
 mgas          - Mass of the gas within a group
 zstar         - Stellar metallicity of each group
 zgas          - Gas metallicity of each group


Definitions:    galprop('a',b,'c',units=0)

    Parameters
    ----------
    a: Input directory (location of the property file)
       Must be input as a string and enclosed in ' ' - see examples.
    b: Snapshot number
    c: Value of interest (see above list)
       Must be input as a string and enclosed in ' ' - see examples.

    Optional
    --------
    untits: Can either be 0 for code units or 1 for real units.  Assumed to be 0
if not included.
            This is only active for Gas & Star Masses (1 returns units of Msun)

Example:    group_mstar=galprop('../', 7, 'mstar',units=1)
            - returns the total stellar mass of each group in units of Msun

            group_cm=galprop('../', 7, 'cm')
            gx,gy,gz=hsplit(group_cm,3)
            - reads in group center of mass positions and returns an NgroupX3
array.
              hsplit is then used to split the array into gx,gy,gz positions.
```

galdata( )

Returns data from all baryonic particles contained within a specified galprop group. The data returned is an array with the following particle properties at each index:

```
 xpos          - x-position of the particle (index 0)
 ypos          - y-position of the particle (index 1)
 zpos          - z-position of the particle (index 2)
 INDEX         - index of the particle (in relation to ALL particles from the
simulation) (index 3)
 TYPE          - TYPE of particle: 0=gas, 4=star (index 4)


Definitions:    galdata('a','b',c,d)

    Parameters
    ----------
    a: Snapshot file.
       Must be input as a string and enclosed in ' ' - see examples.
    b: Input directory (location of the property files)
       Must be input as a string and enclosed in ' ' - see examples.
    c: Snapshot number
    d: Galaxy number of interest - see examples.

Example:    galaxy=galdata('snap_001', '../', 7, 5)
             - returns an array of data with the above information for galaxy 5
in the galprop list

To match data to the snapshot:
            indexes = galaxy[:,3]           #unfortunately this returns an array
of doubles
            indexes = indexes.astype(int)  #convert that array to integers

            - now you can access the data for this particlular galaxy using
these indexes on the full dataset
            for example:  if you read in the density value via:             rho
= readsnap(snap,'rho','gas')
            then rho[indexes] will return the density values of only the
particles within
            the target galaxy.
```

This code is heavily inspired by gadgetpyio written by Matthew Becker, Matthew Turk, & Peter Teuben - thanks for putting the source to your code online.

Any comments or suggestsions feel free to contact me:

Robert Thompson

rthompsonj@gmail.com