# ESERCIZIO FINALE 1
# PRIMO MODULO DEL LABORATORIO DI PROGRAMMAZIONE AVANZATA
# ANNO ACCADEMICO 2019/2020

Data la “snapshot”  snap_le_134, output finale di una simulazione cosmologica di formazione ed evoluzione di una galassia a disco, eseguita con il codice PM-Tree-SPH GADGET-3, si scriva un codice che:

1 – legga posizioni, velocita’, massa ed ID delle particelle di dark matter (Type 1) gas (Type 0) e stelle (Type 4). Il formato binario utilizzato e’ fornito in appendice;

2 – generi tre outputs in formato testo (ASCII), che contengano rispettivamente i tre tipi di particelle, entro 30 kpc dalla posizione del centro della galassia che in questo caso si trova in **x=50.307465  y=50.323699  z=50.511895 ;**

3 – calcoli il profilo radiale di densita’ delle tre componenti. Si ricorda che il profilo di densita’ si ottiene calcolando la massa che cade in un dato numero di gusci sferici, posti a distanza crescente dal centro, e dividendo tale valore per il volume di ciascun guscio sferico;

4 – calcoli il profilo radiale delle velocita’ radiali Vr, pesate in massa. Si ricorda che il profilo delle velocita’ radiali si ottiene utilizzando gusci sferici come nel caso precedente, ma moltiplicando ogni velocita’ radiale per la massa della particella di cui e’ stata calcolata, e dividendo poi il risultato per la massa totale presente nel guscio sferico.

Si scrivano anche opportune procedure per visualizzare i risultati. Le procedure possono essere per python, gnuplot, o per qualunque altro programma grafico disponibile su linux. Si noti che i profili radiali sono di solito visualizzati in coordinate logaritmiche, e per tale motivo andrebbero anche calcolati in gusci sferici equidistanziati logaritmicamente e non linearmente.
Preferibilmente, le procedure andrebbero chiamate dallo stesso codice, dopo aver terminato i calcoli.

La snapshot sara’ fornita via email sotto forma di un link ad un google drive, a causa delle sue grandi dimensioni.

# APPENDICE – FORMATO BINARIO DI GADGET

Gadget output file brief documentation, Antonio Ragagnin
<antonio.ragagnin@inaf.it>

## Gadget Data

Gadget particles are subdivided into 6 particle types that go from 0 to 6.
- Type 0 particles are always gas particles.
- Type 1,2,3 are typically dark matter particles with different, decreasing resolutions.
- Type 4 typically are  stars and type 5 typically are black holes.

Particles have properties (e.g. position, velocity, mass, ecc..), all particles within the same particle type have the same kind of properties, e.g. all (and only) gas particles (particle type 0) have the internal energy property.
Some properties are present in all particle types (thus all particles), this is the case of the position.

This will be important later: typically dark matter particles of a given particle type (e.g. type 1), have all the same value of mass (for instance, 1e9Msun/h).

Gadget stores data in code units:

- Masses are in 1e10 Msun/h
- Positions are in Mpc/h
- Velocities are in sqrt(a)*km/s

Note: following the above conversions, the gravitational constant is G = 43.0071 [code units].

Temperatures are in Kelvin.

Properties will be stored inside the output files in the format of the so called "block". Each block is identified by an array of four characters (note: it is not a C string),
- positions are stored in the block "POS " (note the space at the end)
- velocities are stored in the block "VEL " (note the space at the end)
- masses are stored in the block "MASS"

Each block contains the array of properties of all particles that have such property. For instance, since all particle types have the position property, then the block "POS " will be the concatenation of all type0 positions, then all type1 positions and so on.

One exception is made by the "MASS" block:
since dark matter particles typically have the same mass, for each particle type it is possible to specify it in the header of the gadget output, and, if such quantity is specified, then it is not concatenated in the block "MASS".

For instance, if we have a simulation with only gas and dark matter, and one specifies in the header that the dark matter has a mass of "0.038" then block "MASS" will contains only the masses of gas particles.

Gadget will output a snapshot of particles many times during the simulation.
The naming convention is the following:

- each snapshot has a base name, tipically "snap_"
- after the basename there is a "%3d" integer suffix that identify the *i*th timeslice

The output of Gadget data can be splitted into different files, in this case, there is an additional suffix to each snapshot, that is ".%d", where "%d" is an integer indicating the slice number of the snapshot. Note: this is **not** the case for this exercize.

## Gadget Output Specifics
===========================

Gadget output files are in binary format.
They are formed by a concatenation of so-called blocks.
Each blocks is composed by the concatenation of (i) its header, (ii) the content.

Each block header is

```
struct gadget_block_info{
  unsigned int dummy1;
  char name[4];
  unsigned int  size;
};
```

where name[4] is a 4-characters alphanumeric identification array of the block name.
For instance, "POS " for the block of positions.
The size variable contains the length in bytes of the content.

The content of the block, which follows directly the header is in the following form:
- 4 bytes of "rest"
- the data
- 4 bytes of "rest"

Note that the variable size inside the header includes the total size of the content part, including the initial and final 4bytes. So you have to remove 8 bytes to know the data length.

The block data whose block_info's name[4] is "HEAD" can be stored in the following struct:
(always remember  to skip the initial 4 bytes after the block_info to reach for the data)

```
struct gadget_header
{
  int npart[6];                 /*!< number of particles of each type in this
file */
  double mass[6];               /*!< mass of particles of each type. If 0, then
the masses are explicitly
                                   stored in the mass-block of the snapshot
file, otherwise they are omitted */
  double time;                  /*!< time of snapshot file */
  double redshift;              /*!< redshift of snapshot file */
  int flag_sfr;                 /*!< flags whether the simulation was including
star formation */
  int flag_feedback;            /*!< flags whether feedback was included
(obsolete) */
  unsigned int npartTotal[6];   /*!< total number of particles of each type in
this snapshot. This can be
```

```
                                   different from npart if one is dealing with a
multi-file snapshot. */
  int flag_cooling;            /*!< flags whether cooling was included  */
  int num_files;               /*!< number of files in multi-file snapshot */
  double BoxSize;              /*!< box-size of simulation in case periodic
boundaries were used */
  double Omega0;               /*!< matter density in units of critical density
*/
  double OmegaLambda;          /*!< cosmological constant parameter */
  double HubbleParam;          /*!< Hubble parameter in units of 100 km/sec/Mpc
*/
char fill[Nbytes]
}
```

The array npart[6] contains the number of particles in this file, for each of the six particle types.

The array mass[6] gives the possibility to specify the mass of particles within its particle type.
A value of zero in mass[ptype] means that the mass of particle type "ptype" is stored in the "MASS" block.
The array npartTotal[6] is the sum of the number of particles of all files of the current timeslice.
*Note that Nbytes must be such that the total number of bytes of the structure sums up to 256.*

The other blocks will contain an array of data.

the "POS " and "VEL " block data are made by an array of 3d floats.
the "MASS" block data is made by an array of  floats.
the "ID  " block data is made by an array of unsigned int.

For example, if the total number of particles is N, reading all IDs in the block (a vector defined as: unsigned int ID[N]) will result in:

```
fread(ID,sizeof(int), N file)
```

while, if positions are defined as float pos[N][3], reading them will be:

```
fread(pos, sizeof(float), 3*N, file)
```

Note these instructions just read the data of the block. You must manage names and sizes by yourself.

Also note that the snapshot contains many blocks; you will need to only read the relevant ones.