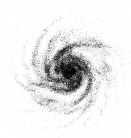
Workshop on N-body simulations

Rubens Machado

UTFPR Curitiba

18–20 February 2020



These tutorials are part of a three-day workshop aimed mainly at undergraduate physics/astronomy students interested in learning the basics of N-body simulations in practice.

Find additional information at http://paginapessoal.utfpr.edu.br/rubensmachado

– version 1.0

Contents

- NEMO
- Gadget-2
- Arepo

NEMO

NEMO

- NEMO is a software environment for stellar dynamics. It has many tools to setup, integrate and analyze *N*-body simulations. Users invoke programs just like any other program within the Unix environment. NEMO development started in 1986 in Princeton by Joshua Barnes, Piet Hut and Peter Teuben.
- old webpage: https://bima.astro.umd.edu/nemo/
- new github repository: https://teuben.github.io/nemo

Before installing NEMO

Packages that will probably be needed:

```
sudo apt install tcsh
sudo apt install git
sudo apt install cmake
sudo apt install build-essential
sudo apt install gcc
sudo apt install g++
sudo apt install gfortran
sudo apt install libx11-dev
sudo apt install libxt-dev
sudo apt install libxext-dev
sudo apt install libcairo2-dev
sudo apt install pgplot5
sudo apt install wcslib-dev
sudo apt install libhdf4-dev libhdf5-dev hdf5-tools
sudo apt install libgsl-dev
```

NEMO installation

Works on Ubuntu 18.04. Local installation (no root privilege needed). A directory `nemo` will be created.

```
csh
wget https://teuben.github.io/nemo/install_nemo
chmod +x install_nemo
./install_nemo nemo=$HOME/nemo
source $HOME/nemo/nemo_start.csh
```

Add an alias to the hidden file `.cshrc` located in your home:

```
echo 'alias nemo "source $HOME/nemo/nemo_start.csh" ' >> $HOME/.cshrc
```

From now on, the NEMO environment can be loaded with:

```
csh
nemo
```

NEMO programs

As a first example, we will create an N-body realization of a Plummer sphere with 1000 particles:

```
mkplummer out=aaa.ic nbody=1000
```

but if the parameters are given in order, this also works:

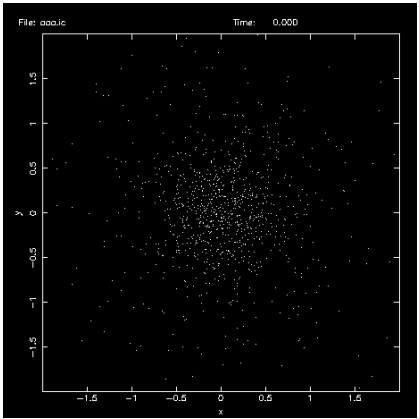
```
mkplummer aaa.ic 1000
```

For most NEMO programs, help will be available via:

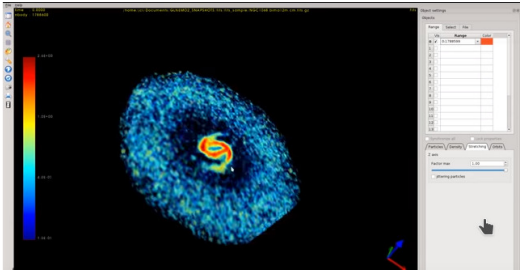
```
mkplummer help
mkplummer help=h
man mkplummer
```

An old-style visualization tool:

```
snapplot aaa.ic
```



glnemo2



- glnemo2 is an interactive 3D visualization program which displays particles of different components (gas, stars, disk, dark mater halo, bulge) of an *N*-body snapshot. It is a very useful tool to visualize simulations from isolated galaxies to cosmological simulations. It was originally developed for the NEMO file format, but currently also supports Gadget-2 and RAMSES files, among others. It was written by Jean-Charles Lambert at the Laboratoire d'Astrophysique de Marseille.
- project page: https://projets.lam.fr/projects/glnemo2
- youtube channel: https://www.youtube.com/user/jcldc/videos

glnemo2 installation

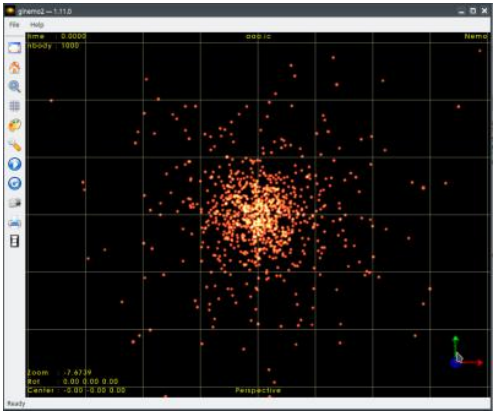
.deb packages are available in the download page:
https://projets.lam.fr/projects/glnemo2/wiki/Download

For Ubuntu 18.04:

```
wget https://projets.lam.fr/attachments/download/5216/glnemo2-1.11.0.ubuntu18.04.x86_64.deb
sudo dpkg -i ./glnemo2-1.11.0.ubuntu18.04.x86_64.deb
sudo apt-get -f install
```

glnemo2 usage

```
glnemo2 aaa.ic all
```



Some useful keyboard shortcuts:

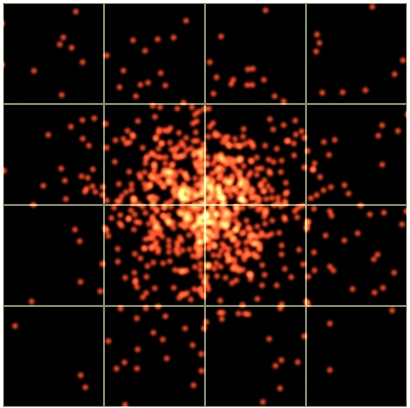
[g] grid
[r] range and colors
[o] options dialog
[p] play
[l] reload

See:

```
man glnemo2
```

glnemo2 screenshots

```
glnemo2 aaa.ic all texture=f point=t psize=10 osd=f grid=t axis=f wsize=400 hsize=400 \
bestzoom=f zoom=-4.828 screenshot=aaa.jpg
```



The zoom parameter is the height of a camera with 45° field of view:

$$\text{zoom} = \frac{R}{\tan(\frac{\pi}{8})}$$

where R is the desired half-width of the square frame (i.e. range $-R : R$). If the frame is rectangular, this applies to the shorter dimension.

Units

Here we will adopt the so-called natural system, where $G = 1$. Quantities may then be regarded as being expressed in internal code units. They are often presented dimensionlessly, since the conversion to physical units is arbitrary, allowing for any desired rescaling. For definiteness, one possible choice would be such that the physical units of length, mass, time and velocity are:

$$\begin{aligned} [L] &= 3.5\,\mathrm{kpc} \\ [M] &= 5 \times 10^{10}\,\mathrm{M}_{\odot} \\ [T] &= 1.4 \times 10^{7}\,\mathrm{yr} \\ [V] &= 248\,\mathrm{km/s} \end{aligned}$$

Thus a galaxy with disk mass $M_{\mathrm{d}} = 1$ and scale length $R_{\mathrm{d}} = 1$ will be comparable to the Milky Way.

Initial conditions for a disk galaxy

The `magalie` program creates initial conditions for an isolated galaxy composed of disk, bulge and halo particles. We will create a bulgeless galaxy with only disk and halo:

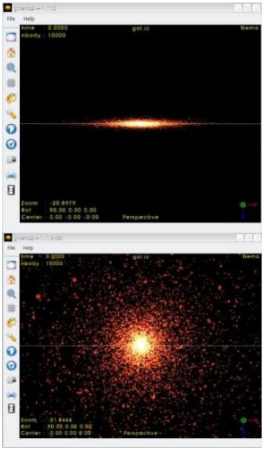
```
magalie gal.ic ndisk=10000 nbulge=0 nhalo=10000
```

The first ten thousand particles (indices 0:9999) are the disk:

```
glnemo2 gal.ic 0:9999 xrot=90
```

The next ten thousand particles (indices 10000:19999) are the halo:

```
glnemo2 gal.ic 10000:19999 xrot=90
```



gyrfalcON

- gyrfalcON is an N-body code whose force algorithm of complexity $\mathcal{O}(N)$ is about 10 times faster than an optimally coded tree code. It was written by Walter Dehnen at the Max-Planck-Institut für Astronomie, Heidelberg.
- extremely fast
- serial (not parallel)
- N-bodies only (no hydrodynamics)
- Dehnen, W., 2000, ApJ, 536, L39
  Dehnen, W., 2001, MNRAS, 324, 273
  Dehnen, W., 2002, JCP, 179, 27

Running gyrfalcON

In order to run the simulation, we need to provide at least the following parameters:

```
gyrfalcON in=??? out=??? tstop=??? step=??? kmax=??? eps=???
```

- `in`: initial conditions file (`in=gal.ic`)
- `out`: output file file (`out=gal.out`). All times are written into one single file.
- `tstop`: final time of the simulation. We will use `tstop=50`, which is 0.7 Gyr.
- `step`: time between outputs; zero means every time step. This sets how often the output will be written. It does not affect the calculations. **Caution**: always estimate the size of the output file before starting the run. Writing every single time step may quickly produce an exceedingly large file. We will use `step=0.1` (1.4 Myr) to obtain 500 frames for the visualization.
- `kmax`: the longest time step is taken to be $\tau = 2^{-\mathtt{kmax}}$. For example, if `kmax=6`, the time step will be $\tau = \frac{1}{2^6}$, i.e. 1 time unit divided by 64, or 0.015625 time units ($\sim$0.2 Myr).
- `eps`: softening length. We will use `eps=0.03` ($\sim$0.1 kpc).

Running gyrfalcON

We can now execute the simulation with:

```
gyrfalcON gal.ic gal.out tstop=50 step=0.1 kmax=6 eps=0.03
```

During the run, the code will print 14 columns of diagnostics that can optionally be sent to a log file. The first column is the simulation time in code units. The last column is the accumulated wallclock time in `hh:mm:ss.ss`, which can be used to estimate the total run time.

Once the run is finished (or before) we can visualize the output by selecting all components:

```
glnemo2 gal.out 0:9999,10000:19999
```

or selecting only the disk particles:

```
glnemo2 gal.out 0:9999
```

Creating an animation with ffmpeg

First we use glnemo2 to produce 500 jpg images in a separate directory. They will be named frame.00000.jpg until frame.00500.jpg.

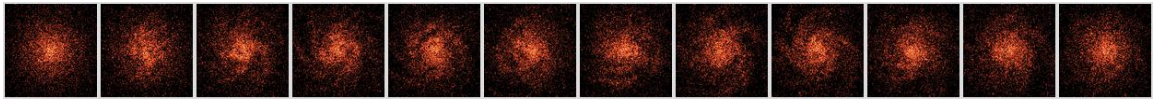
```
mkdir frames
cd frames/
glnemo2 ../gal.out 0:9999 hsize=400 wsize=400 play=t screenshot=frame
```

Install ffmpeg and mplayer:

```
sudo apt install ffmpeg
sudo apt install mplayer
```

Now use ffmpeg to create an mp4 video:

```
ffmpeg -y -i frame.%05d.jpg -c:v libx264 -vf fps=20 -pix_fmt yuv420p video.mp4
mplayer video.mp4
```



Imagemagick

Imagemagick is a software suite for displaying, converting, and editing images. Some useful examples:

```
sudo apt install imagemagick
```

- convert

```
convert in.png out.jpg
```

- resize

```
convert -resize   50% in.jpg out.jpg
convert -resize  720x in.jpg out.jpg
convert -resize x720  in.jpg out.jpg
```

- rotate

```
convert -rotate 90 in.jpg out.jpg
```

- write label

```
convert -fill white -gravity Northeast -pointsize 30 -annotate +5+5 "t=5Gyr" in.jpg out.jpg
```

- draw line

```
convert -stroke white -strokewidth 4 -draw "line 100,30 200,30" in.jpg out.jpg
```

Imagemagick

- montage

```
montage -geometry +0+0 -tile 1x2 -borderwidth 10 -bordercolor white in1.jpg in2.jpg out.jpg
```

- crop white borders

```
convert -trim in.jpg out.jpg
```

- chop slice

```
convert -gravity North -chop 0x50 in.jpg out.jpg
```

- convert to eps

```
convert in.jpg eps3:out.eps
```

- batch convert to jpg

```
mogrify -format jpg *.png
```

- batch resize (**Warning**: overwrites originals!)

```
mogrify -resize 320x *.jpg
```

Imagemagick

- create transparency

```
convert -fuzz 10% -transparent white in.jpg out.gif
```

- overlay transparency

```
composite -dissolve 50% -gravity center transp.gif in.jpg out.jpg
```

- invert colors

```
convert -negate in.jpg out.jpg
```

- convert to graysclale

```
convert -type Grayscale in.jpg out.jpg
```

- create animated gif

```
convert -delay 100 -loop 0 in*.jpg out.gif
```

gyrfalcON exercises

Now explore different parameters and evaluate the consequences:

1) write the output at different intervals

```
step=0, step=0.01, step=1
```

2) smaller and larger time steps

```
kmax=1, kmax=4, kmax=8
```

3) smaller and larger softening lengths

```
eps=0, eps=0.03, eps=1
```

4) number of particles

```
ndisk=1000, ndisk=10000, ndisk=100000
```

5) Toomre parameter (`magalie` parameter)

```
Qtoomre=0.2, Qtoomre=1.0, Qtoomre=2.0
```

6) Halo mass (`magalie` parameter)

```
mhalo=1, mhalo=5, mhalo=25
```

Some more examples

1 - Collapse of a uniform sphere
2 - Bare disk
3 - Colliding galaxies
4 - Satellite galaxy
5 - Barred galaxy

1 - Collapse of a uniform sphere

Create a uniform sphere:

```
mksphere sphere.ic 10000 radii=0:10:0.1 density=1
```

Check the initial conditions:

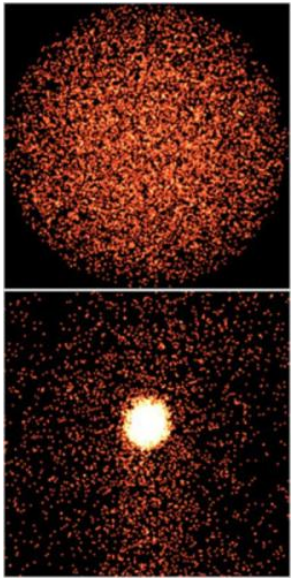
```
glnemo2 sphere.ic all point=t psize=5
```

Run the simulation:

```
gyrfalcON sphere.ic sphere.out tstop=100 kmax=4 step=0.1 eps=0.03
```

View the result:

```
glnemo2 sphere.out all point=t psize=5
```



2 - Bare disk

Create an exponential disk, assuming no other mass distribution is present. **Note**: Such disks are wildly unstable, but fun to play with.

```
mkbaredisk disk.ic 10000
```

Check the initial conditions:

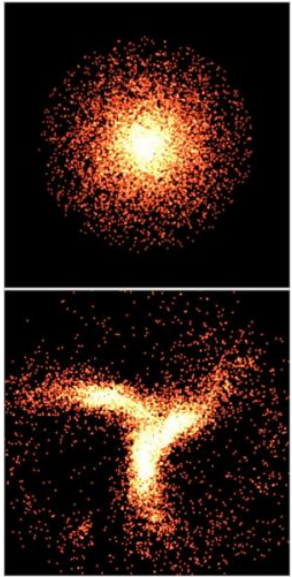
```
glnemo2 disk.ic all point=t psize=5
```

Run the simulation:

```
gyrfalcON disk.ic disk.out tstop=10 kmax=5 step=0 eps=0.03
```

View the result:

```
glnemo2 disk.out all point=t psize=5
```



3 - Colliding galaxies

First we rotate a galaxy:

```
snaprotate gal.ic gal.rot theta=0,90,0 order=xyz
```

Now we join the two galaxies into one snapshot:

```
snapstack gal.ic gal.rot col.ic deltar=10,10,0 deltav=-1,0,0
```

Check the initial conditions (note the disk indices):

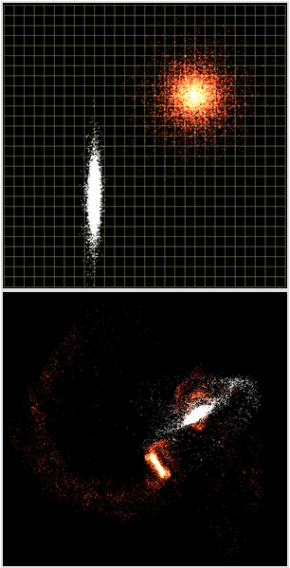
```
glnemo2 col.ic 0:9999,20000:29999
```

Run the collision:

```
gyrfalcON col.ic col.out tstop=50 step=0.1 kmax=6 eps=0.03
```

View the result:

```
glnemo2 col.out 0:9999,20000:29999
```



4 - Satellite galaxy

Create a small spheroidal galaxy:

```
mkdehnen sat.ic 1000 gamma=0 mass=0.05 r_s=0.05
```

Join the two galaxies:

```
snapstack gal.ic sat.ic sat2.ic deltar=10,10,6 deltav=0,0,-0.4
```

Check the initial conditions (note the indices):

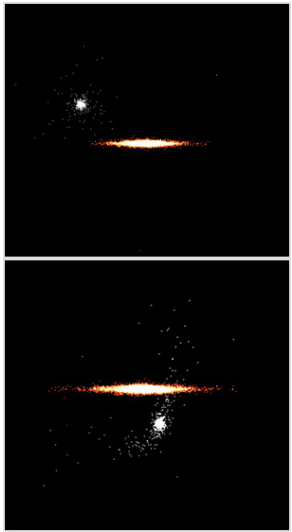
```
glnemo2 sat2.ic 0:9999,20000:20999 xrot=90 grid=f
```

Run the simulation:

```
gyrfalcON sat2.ic sat2.out tstop=300 step=0.1 kmax=5 eps=0.03
```

View the result:

```
glnemo2 sat2.out 0:9999,20000:20999 xrot=90 grid=f
```



5 - Barred galaxy

Create a galaxy with a cold and massive disk:

```
magalie galaxy.ic ndisk=10000 nhalo=10000 nbulge=0 Qtoomre=0.2 \
mhalo=2
```

Check the initial conditions:

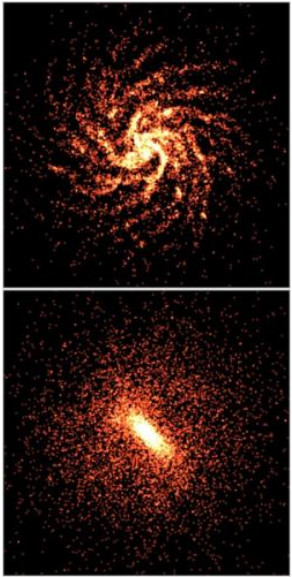
```
glnemo2 galaxy.ic 0:9999 texture=f point=t psize=3
```

Run the simulation:

```
gyrfalcON galaxy.ic galaxy.out tstop=200 kmax=4 step=0.1 eps=0.05
```

View the result:

```
glnemo2 galaxy.out 0:9999 texture=f point=t psize=3
```



Analyzing NEMO output: print columns to txt file

The `snapprint` command can be used to print columns of coordinates into an ascii file.

First, we will create a new initial condition:

```
magalie gal.ic ndisk=1000 nbulge=0 nhalo=1000
```

And then send the masses, positions and velocities to a text file:

```
snapprint gal.ic times=0 options=m,x,y,z,vx,vy,vz > gal0.txt
```

The first 1000 lines are disk particles:

```
head -1000 gal0.txt > disk0.txt
```

The final 1000 lines are halo particles:

```
tail -1000 gal0.txt > halo0.txt
```

Analyzing NEMO output: print columns to txt file

This text file with N lines (one for each particle) and 7 columns (mass, 3 position coordinates and 3 velocity coordinates) can then be read and manipulated elsewhere.

```
# m       x         y         z         vx        vy        vz
0.001 -0.598099 -0.463554 -0.172503  0.195702 -0.771182 -0.380785
0.001 -0.075118 -1.007530 -0.038463  0.683740  0.096297  0.194712
0.001  2.363940  0.247924  0.156649 -0.171309  0.842389 -0.060829
0.001 -0.877326 -0.175230 -0.002608  0.098003 -0.708416  0.036857
0.001  0.261797  0.488046  0.017017 -0.780273  0.560569 -0.292504
0.001 -2.029360 -1.265490 -0.011543  0.414185 -1.029155 -0.057341
0.001 -0.380687 -0.257418  0.108136  0.147553 -0.596774 -0.269792
0.001 -0.330207 -0.082219  0.005551  0.301624 -0.743189  0.106995
0.001  0.367545  0.625392  0.309730 -0.749725  0.451241  0.256248
0.001  0.226765  2.758810 -0.078053 -1.063312 -0.107696 -0.289566
0.001  0.931352 -0.804894  0.273779  0.625324  0.474351  0.062041
0.001  0.360092 -6.168010 -0.020866  0.877723  0.051029  0.014430
...
```

Analyzing NEMO output: gnuplot

We could have a quick look at the data with gnuplot:

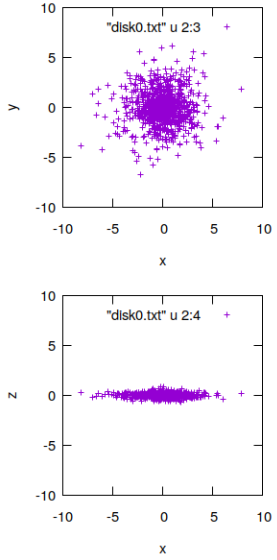
```
gnuplot
```

Check the *xy* disk positions:

```
set size square
set xlabel "x"
set ylabel "y"
plot[-10:10][-10:10] "disk0.txt" using 2:3
```

Check the *xz* disk positions:

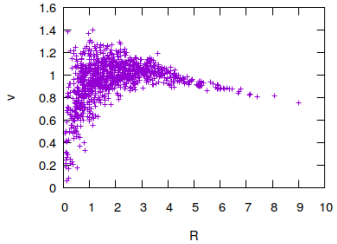
```
set size square
set xlabel "x"
set ylabel "z"
plot[-10:10][-10:10] "disk0.txt" using 2:4
```



Analyzing NEMO output: gnuplot

We might want to plot velocities on the *xy* plane and cylindrical radii:

```
gnuplot
set xlabel "R"
set ylabel "v"
plot "disk0.txt" using (sqrt($2*$2+$3*$3)):(sqrt($5*$5+$6*$6))
```



Analyzing NEMO output: python

We can read the columns with a simple python script containing:

```
from numpy import *
m, x, y, z, vx, vy, vz = loadtxt("disk0.txt", unpack=True)
Mdisk = sum(m)
print Mdisk
```

and run it:

```
python readtxt.py
```

Analyzing NEMO output: python

Other simple calculations in a python script:

```
from numpy import *
m, x, y, z, vx, vy, vz = loadtxt("disk0.txt", unpack=True)
#center of mass coordinates
xcom = sum(m*x) / sum(m)
ycom = sum(m*y) / sum(m)
zcom = sum(m*z) / sum(m)
print "com  = %f %f %f" % (xcom, ycom, zcom)
#minimum and maximum radii
r = sqrt(x**2+y**2+z**2)
rmin = min(r)
rmax = max(r)
print "rmin = %f" % (rmin)
print "rmax = %f" % (rmax)
```

Analyzing NEMO output: addgravity

The command addgravity adds accelerations and potential to the snapshot:

```
addgravity gal.ic gal.grav
```

Check which fields are present in the new snapshot:

```
tsf gal.grav
```

Print also accelerations and potential to a text file:

```
snapprint gal.grav options=m,x,y,z,vx,vy,vz,ax,ay,az,phi > galgrav.txt
```

Gadget-2

Gadget-2

- Gadget-2 is a massively parallel code for N-body/SPH simulations. It was written by Volker Springel at the Max-Planck-Institut für Astrophysik, Garching.
- Webpage: `https://wwwmpa.mpa-garching.mpg.de/gadget/`
- Read the User Guide
- The Mailing List is archived and has useful questions and answers
- Springel V., 2005, MNRAS, 364, 1105
  Springel V., Yoshida N., White S. D. M., 2001, New Astronomy, 6, 51

Gadget-2 requirements

- Message Passing Interface (MPI)
- GNU Scientific Library (GSL)
- Fastest Fourier Transform in the West (FFTW)

MPI installation

MPI can be installed normally via the package repositories:

```
sudo apt install mpich
sudo apt install libmpich-dev
```

GSL installation

Option 1: System installation. After compilation, the libraries will be copied into the default directories. This is convenient in a personal computer where the user has administrator privilege.

```
cd
mkdir gsl
cd gsl
wget ftp://ftp.gnu.org/gnu/gsl/gsl-1.9.tar.gz
tar -xzf gsl-1.9.tar.gz
cd gsl-1.9/
./configure
make
sudo make install
```

GSL installation

Option 2: Local installation. The libraries will be installed at a specified location within the user home. This is useful for installation in clusters, where the user does not have administrator privilege.

```
cd
mkdir gsl
cd gsl
wget ftp://ftp.gnu.org/gnu/gsl/gsl-1.9.tar.gz
tar -xzf gsl-1.9.tar.gz
cd gsl-1.9/
./configure --prefix=/home/rubens/gsl
make
make install
```

FFTW installation

Option 1: System installation. After compilation, the libraries will be copied into the default directories. This is convenient in a personal computer where the user has administrator privilege.

```
cd
mkdir fftw
cd fftw
wget http://www.fftw.org/fftw-2.1.5.tar.gz
tar -xzf fftw-2.1.5.tar.gz
cd fftw-2.1.5
./configure --enable-mpi --enable-type-prefix --enable-float
make
sudo make install
```

FFTW installation

Option 2: Local installation. The libraries will be installed at a specified location within the user home. This is useful for installation in clusters, where the user does not have administrator privilege.

```
cd
mkdir fftw
cd fftw
wget http://www.fftw.org/fftw-2.1.5.tar.gz
tar -xzf fftw-2.1.5.tar.gz
cd fftw-2.1.5
./configure --enable-mpi --enable-type-prefix --enable-float --prefix=/home/rubens/fftw
make
make install
```

Gadget-2 installation

First, download the tar file and extract it:

```
cd
wget http://wwwmpa.mpa-garching.mpg.de/gadget/gadget-2.0.7.tar.gz
tar -xzf gadget-2.0.7.tar.gz
cd Gadget-2.0.7/Gadget2
```

Before compiling the code we need to edit a few options in the Makefile. Comment this line becase our first examples will be non-cosmological (hence no periodic boundary conditions):

```
# OPT   +=  -DPERIODIC
```

Also comment this line since we will not use the HDF5 format:

```
# OPT   +=  -DHAVE_HDF5
```

The TreePM method may be switched off:

```
# OPT   +=  -DPMGRID=128
```

Gadget-2 installation

We should define our own SYSTYPE within the Makefile, telling it where to find the GSL and FFTW libraries. If they were installed in the default system directories, their paths should be:

```
#-------------------------------------- Select target computer
SYSTYPE="Dafis"
#-------------------------------------- Select target computer
ifeq ($(SYSTYPE),"Dafis")
CC = mpicc
OPTIMIZE = -O3 -Wall
GSL_INCL = -I/usr/local/include
GSL_LIBS = -L/usr/local/lib
FFTW_INCL= -I/usr/local/include
FFTW_LIBS= -L/usr/local/lib
MPICHLIB = -L/usr/lib
endif
```

Gadget-2 installation

If GSL and FFTW were installed locally, give their full paths:

```
#--------------------------------------- Select target computer
SYSTYPE="Dafis"
#--------------------------------------- Select target computer
ifeq ($(SYSTYPE),"Dafis")
CC = mpicc
OPTIMIZE = -O3 -Wall
GSL_INCL = -I/home/rubens/gsl/include
GSL_LIBS = -L/home/rubens/gsl/lib
FFTW_INCL= -I/home/rubens/fftw/include
FFTW_LIBS= -L/home/rubens/fftw/lib
MPICHLIB = -L/usr/lib
endif
```

Gadget-2 installation

Having edited the Makefile, we can finally compile the code:

```
make
```

To ensure that it runs, try this quick test using the provided examples:

```
cd ..
mkdir galaxy
mpirun -np 1 Gadget2/Gadget2 Gadget2/parameterfiles/galaxy.param
```

Units

The conventional system of units adopted with Gadget-2 uses $G = 43007.1$, such that the physical units of length, mass, velocity and time are:

$$
\begin{aligned}
[L] &= 1\,\mathrm{kpc}\,h^{-1} \\
[M] &= 10^{10}\,\mathrm{M}_{\odot}\,h^{-1} \\
[V] &= 1\,\mathrm{km/s} \\
[T] &= 0.98\,\mathrm{Gyr}\,h^{-1}
\end{aligned}
$$

Preparing a Gadget-2 run

Let us start from a clean directory and prepare the necessary structure:

```
cd
mkdir simulations
cd simulations
```

Bring only the code itself here:

```
cp -r /home/rubens/Desktop/aaa/ggg/Gadget-2.0.7/Gadget2/ Gadget2/
```

It can be recompiled if necessary:

```
cd Gadget2
make clean
make
cd ..
```

Preparing a Gadget-2 run

Now, apart from the executable, we need three elements:

- the initial conditions
- the parameterfile
- an output directory

Bring the initial conditions here:

```
cp ~/Gadget-2.0.7/ICs/galaxy_littleendian.dat .
```

Bring the parameterfile here:

```
cp ~/Gadget-2.0.7/Gadget2/parameterfiles/galaxy.param param
```

Create a new directory where the output will be written:

```
mkdir output
```

Advice: It is usually good practice to write voluminous outputs elsewhere.

Preparing a Gadget-2 run

In the `param` file, the paths of the initial conditions and of the output directory need to be specified (with respect to the launch point). **Advice**: Specially when working on a cluster and submitting several jobs, it is prudent to give the *full path* of the OutputDir.

```
%  Relevant files
InitCondFile       galaxy_littleendian.dat
OutputDir          output/
```

Finally, to launch the simulation:

```
mpirun -np 2 Gadget2/Gadget2 param
```

The first argument, `-np 2`, is the number of processors. The second argument is the path to the executable. The third argument is the parameterfile.

glnemo2 with a list of snapshots

This example would produce only 6 snapshots. In the parameterfile:

```
TimeBetSnapshot     0.5
TimeMax             3.0
```

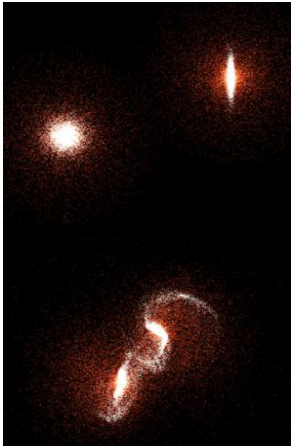
we can decrease the time between snapshots in order to write 100 snapshots, for instance. This does not affect the calculations.

glnemo2 can load one snapshot at a time and it understands the labels of the particle types:

```
cd output
glnemo snapshot_000 disk
```

If we direct a list of snapshot names to a text file named `list`, glnemo2 will load them all sequentially:

```
ls snapshot_* > list ; glnemo2 list disk,halo
```



Numbering of the snapshots files

When the number of snapshots exceeds 999, listing their names in order might become inconvenient for some purposes. One could edit the file `io.c`, replacing the `%03d` with `%04d` in lines 99 and 101. Recompile the code, of course. Now the snapshots will all be numbered with four-digit indices:

```
snapshot_0000
snapshot_0001
...
snapshot_0999
snapshot_1000
```

Gadget-2 file format

The code distinguishes between different particle types and each type may have a different gravitational softening.

Type	Name
0	Gas
1	Halo
2	Disk
3	Bulge
4	Stars
5	Bndry

Essentially, all particles have masses, positions and velocities, but gas particles also have information about density and temperature. See the User Guide.

Gas particles

Apart from the fields MASS, POS, VEL, gas particles also have RHO (density), U (internal energy) and HSML (hydro smoothing length).

The initial conditions do not need to have RHO and HSML necessarily; these fields will be calculated by the code at the first step, based on the positions and masses. **Note**: `glnemo2` needs both these fields in order to render the colors of the gas particles. Initial conditions that lack them may not be displayed properly, but the simulation will run normally and the output will be readable.

The field U has units of energy per unit mass. It is often convenient to convert this to temperature in Kelvin. For an ideal monoatomic gas (adiabatic index $\gamma = 5/3$), the temperature T and internal energy u are related by:

$$u = \frac{3}{2} \frac{k_B T}{\mu m_H},$$

where k_B is the Boltzmann constant and m_H is the proton mass. The mean molecular weight is approximately $\mu = 0.6$ for a fully ionized gas and $\mu = 1.2$ for a fully neutral gas (assuming primordial abundance $X = 0.76$).

pygadgetreader

- pygadgetreader is a useful library to read Gadget-2 data into a python program. Positions, velocities etc are loaded directly into numpy arrays. The author is Robert Thompson.
- https://bitbucket.org/rthompson/pygadgetreader/

pygadgetreader installaion

```
hg clone https://bitbucket.org/rthompson/pygadgetreader
cd pygadgetreader/
python setup.py build
sudo python setup.py install
```

pygadgetreader: reading pos, vel

Basic example of reading positions and velocities in a python script:

```
from numpy import *
from pygadgetreader import *
#file name
snapshot = 'snapshot_0000'
#read halo positions
pos   = readsnap(snapshot,'pos','dm')
xhalo = pos[:, 0]
yhalo = pos[:, 1]
zhalo = pos[:, 2]
#read disk positions
pos   = readsnap(snapshot,'pos','disk')
xdisk = pos[:, 0]
ydisk = pos[:, 1]
zdisk = pos[:, 2]
```

Note: pygadgetreader uses ‘dm’ rather than ‘halo’. The other type labels are standard.

pygadgetreader: reading the header

Example to read the information in the snapshot header (time, numbers of particles):

```
from numpy import *
from pygadgetreader import *
#read header data
header = readheader(snapshot,'header')
Time   = header['time']
Ngas   = header['ngas']
Nhalo  = header['ndm']
Ndisk  = header['ndisk']
```

pygadgetreader: reading masses

For the particle masses, there are two possibilities.

1) If the particles have equal mass, perhaps the mass of each type was specified only once in the header. The array massTable has 6 elements, with the particle mass of each type:

```
from numpy import *
from pygadgetreader import *
#read header data
header = readheader(snapshot,'header')
massTable = header['massTable']
mgas      = massTable[0]
mhalo     = massTable[1]
mdisk     = massTable[2]
mbulge    = massTable[3]
mstars    = massTable[4]
mbndry    = massTable[5]
```

pygadgetreader: reading masses

2) However, masses may have been specified in mass arrays (even if they are all the same). In this case, the corresponding entries in massTable are zero. Then the masses can be read similarly to pos, vel:

```
from numpy import *
from pygadgetreader import *
#read header data
mdisk    = readsnap(snapshot,'mass','disk')
mhalo    = readsnap(snapshot,'mass','dm')
mgas     = readsnap(snapshot,'mass','gas')
```

Load an example snapshot and check the lengths and contents of these arrays. It is simple to calculate in python, for example: the total mass of each component; the minimum, mean and maximum positions of particles; the center of mass, etc.

`clustep`: initial conditions for clusters

- `clustep` creates initial conditions in the Gadget-2 format for a galaxy cluster represented by dark matter particles and gas particles. It was written by Rafael Ruggiero at IAG/USP.
- https://github.com/ruggiero/clustep

clustep installation

Dependencies:

```
sudo apt install python-numpy python-scipy cython python-matplotlib python-argparse python-dev
```

```
git clone https://github.com/ruggiero/clustep.git
cd clustep/clustep
make
```

clustep usage

Inside the clustep directory, simply run:

```
python clustep.py -o snapshot_0000
```

where the -o parameter is the name of the created initial condition file. The parameters of the cluster are inside the text file `params_cluster.ini`. Essentially, we can choose – for both gas and halo – the number of particles, the mass and the scale length. We can also choose the γ parameter of the Dehnen density profile. If $\gamma = 1$, then the profile is equal to a Hernquist profile, featuring a central cusp. If $\gamma = 0$, then the profile features a central core.

- Dehnen, W., 1993, MNRAS, 265, 250
  Hernquist, L., 1990, MNRAS, 356, 359

galstep: initial conditions for galaxies

- galstepcreates initial conditions in the Gadget-2 format for a disk galaxy with the components: gas, halo, disk and bulge. This code uses the algorithm described in Springel, Di Matteo & Hernquist (2005). It was written by Rafael Ruggiero at IAG/USP.
- https://github.com/ruggiero/galstep
- Springel, Di Matteo & Hernquist, 2005, MNRAS, 361, 776

galstep installation

There is no installation. Just download the code.

```
git clone https://github.com/ruggiero/galstep.git
```

galstep usage

Inside the galstep directory, simply run:

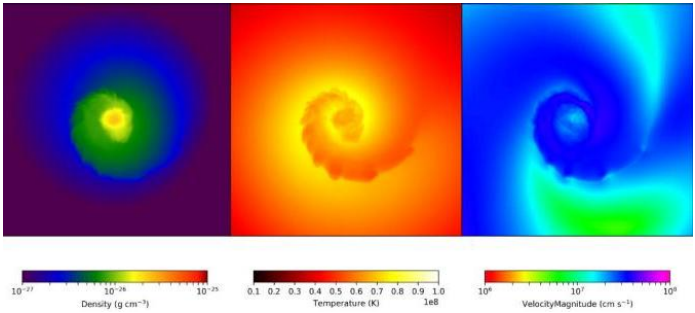
```
python galstep.py -i params_galaxy.ini -o snapshot_0000
```

but see the help for other options. Similarly, the parameters of the galaxy are inside the text file `params_galaxy.ini`.

Other codes to create initial conditions

- **DICE** creates initial conditions for galaxies in the Gadget-2 format, offering several free parameters. It was written by Valentin Perret. https://bitbucket.org/vperret/dice
- **GALIC** creates *N*-body galaxy models in collisionless equilibrium (no gas). Developed by Denis Yurin. https://wwwmpa.mpa-garching.mpg.de/ volker/galic/
- **N-GenIC** creates cosmological initial conditions based on the Zeldovich approximation, in the Gadget-2 format. By Volker Springel. http://www.mpa-garching.mpg.de/gadget/n-genic.tar.gz

Visualization with yt



- yt is a python library for visualization of volumetric data. It supports several simulation formats and is community-developed. It is very useful to visualize specially gas properties, such as density maps and temperature maps.
- https://yt-project.org/
- https://yt-project.org/doc/cookbook/index.html

yt installation

To install via pip:

```
sudo apt install python-pip
pip install yt
```

but see other options on the webpage.

yt and mean molecular weight

In galaxy clusters, the temperature of the ICM is sufficiently high that the gas is fully ionized. It so happens that when computing temperatures from internal energy in Gadget-2 snapshots, `yt` unfortunately assumes zero ionization by default, leading to $\mu = 1.2$ rather than $\mu = 0.6$. Thus all plotted temperatures in the ICM would be *wrong* by a factor of 2. At the moment, a workaround to fix this is to find line 89 in this file:

```
/home/rubens/.local/lib/python2.7/site-packages/yt/frontends/gadget/fields.py
```

and change

```
# Assume zero ionization
mu = 4.0 / (3.0 * x_H + 1.0)
```

to

```
# ASSUME FULL IONIZATION
mu = 4.0 / (5.0 * x_H + 3.0)
```

There is no need to reinstall.

Load Gadget-2 snapshot in yt

To load a Gadget-2 snapshot:

```
import yt
from pygadgetreader import *
import numpy as np
import yt.units as units
import pylab
fname = 'snapshot_0000'
#-------------------------------------------------------
unit_base = {'UnitLength_in_cm'         : 3.08568e+21,
             'UnitMass_in_g'            :   1.989e+43,
             'UnitVelocity_in_cm_per_s' :      100000}
bbox_lim = 1e5 #kpc
bbox = [[-bbox_lim,bbox_lim],
        [-bbox_lim,bbox_lim],
        [-bbox_lim,bbox_lim]]
ds = yt.load (fname,unit_base=unit_base,bounding_box=bbox)
ds.index
ad= ds.all_data()
#-------------------------------------------------------
```

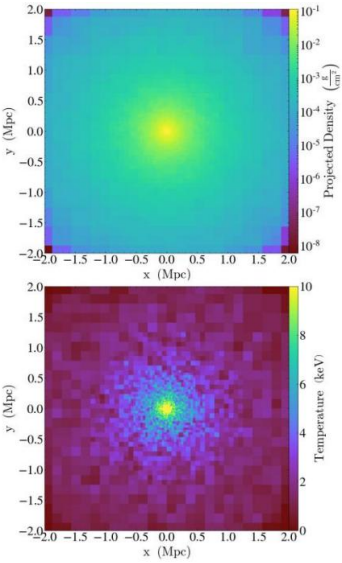
density and temperature maps with yt

And then to plot a density map:

```
d = yt.ProjectionPlot(ds, 'z', 'density', width = (2, 'Mpc'))
d.save('density.png')
```

And a temperature map in keV:

```
t = yt.SlicePlot(ds, 'z', 'temperature', width = (2, 'Mpc'))
t.set_unit('temperature', 'keV', equivalency='thermal')
t.set_log("temperature", False)
t.set_zlim('temperature', 0, 10)
t.save('temperature.png')
```



Arepo

Arepo

- Arepo is a massively parallel code for gravitational N-body systems and magnetohydrodynamics, both on Newtonian as well as cosmological backgrounds. The code uses a moving mesh for the hydrodynamics. It was developed initially by Volker Springel at the Max-Planck-Institut für Astrophysik, Garching.
- https://wwwmpa.mpa-garching.mpg.de/ volker/arepo/
- Srpingel, V., 2010, MNRAS, 401, 791
- A public version was released in September 2019: https://arxiv.org/abs/1909.04667
- Repository, User Guide, etc: https://arepo-code.org/

Some Arepo features

- The general structure of compilation, parameterfile, etc is similar to Gadget-2.
- The recommended format for input and output is the HDF5 format.
- (Both glnemo2 and yt can read HDF5.)
- The old Gadget-2 format can also be used both for input and output.
- If SPH gas particles are present in the initial conditions, an intermediate step is required in order to convert the SPH particles into a cell structure suitable for grid calculations.
- A simple cooling and star formation model is included in the public version.