# aarseth

April 25, 2025

## 1 NEMO (PHYS265)

The NEMO package is introduced here, with a focus on the Aarseth N-body codes. This notebook was written as an example for the 2025 PHYS265 final project.

### 1.1 The Aarseth NBODY family of codes

The N-body problem solves the time integration of N interacting particles, where the accelleration on particle $i$ is given by the contributions of all $j \neq i$ particles:

$$\ddot{\mathbf{r}}_i = -G \sum_{j=1;\, j\neq i}^{N} \frac{m_j \, (\mathbf{r}_i - \mathbf{r}_j)}{(r_{ij}^2 + \epsilon^2)^{3/2}}$$

One of the early practicioners of N-body codes was Sverre Aarseth, who has always made his N-body codes available for anybody to use and modify to suite their own needs. His current body of work can still be found at https://people.ast.cam.ac.uk/~sverre/web/pages/nbody.htm as well as an entry in ASCL. A few of these exist in NEMO. We list a few of the programs available here, and will try them out:

- nbody0, nbody00 - version from Binney & Tremaine's "Galactic Dynamics" (1987) book
- nbody1, runbody1 - integrator with variable timestep
- nbody2, runbody2 - Ahmad-Cohen N-body code
- nbody4, runbody4 - hermite N-body code with optional stellar evolution
- firstn - von Hoerners first N-body code (1960)

- mkplummer - create a Plummer (1911) N-Body sphere. Algorithm by Aarseth, Henon and Wielen (1974)
- Sverre's 1999 paper "From NBODY1 to NBODY6: The Growth of an Industry" outlines the history behind this series.

## 2 Loading NEMO

We start by loading NEMO in the Unix shell, `bash` in this case.

As a sanity check we first look to see if $NEMO exist. It should not, unless your shell defined NEMO already.

[1]:
```
# check to see if NEMO exists
echo NEMO=$NEMO
```

```
NEMO=
```

If $NEMO existed already, the next cell could be safely skipped, but would not do any harm.

[2]:
```
# load NEMO    (your location will likely differ).
source $HOME/NEMO/nemo/nemo_start.sh
```

[3]:
```
# show NEMO and some related things with the `nemo` command
nemo
```

```
NEMO:         /home/teuben/NEMO/nemo  - Version:4.5.2
YAPP:         /xs - default yapp plotting device
git:          Branch:docopt     Counter:12481      Date: 2025/04/24_17:13:49
python:       /home/teuben/NEMO/nemo/anaconda3/bin/python  - Python 3.12.4
OS_release:   Linux Description: Pop!_OS 22.04 LTS
```

### 2.0.1 Plotting ?

A minor nuisance of using a bash notebook instead of a python notebook is that you cannot produce the typical interactive matplotlib plots. If we compile NEMO with *yapp_pgplot*, plots can be saved in **png** format and markdown cells can load them after the cell was executed. This requires an extra step, but is still illustrative. In this notebook they have been pre-computed.

## 3 nbody0

This code was published in an Appendix of the 1987 (first) edition of Binney & Tremaine's *Galactic Dynamics*. The source code can be found in **$NEMO/src/nbody/evolve/aarseth/nbody0**, where several derivatives of this *Micky Mouse* (Sverre's words) version are available. We have a NEMO manual page available for this code, which you can also access from the command line with the `man nbody0` command:

[4]:
```
# man nbody0
```

### 3.0.1 Creating initial conditions

By default the FORTRAN code is compiled with space for a maximum of 2048 particles (it's FORTRAN!). We thus create a Plummer (1911) sphere with 2048 particles. We fix the seed to have reproducible results, and integrate a few crossing times to keep the CPU loaded for a few seconds.

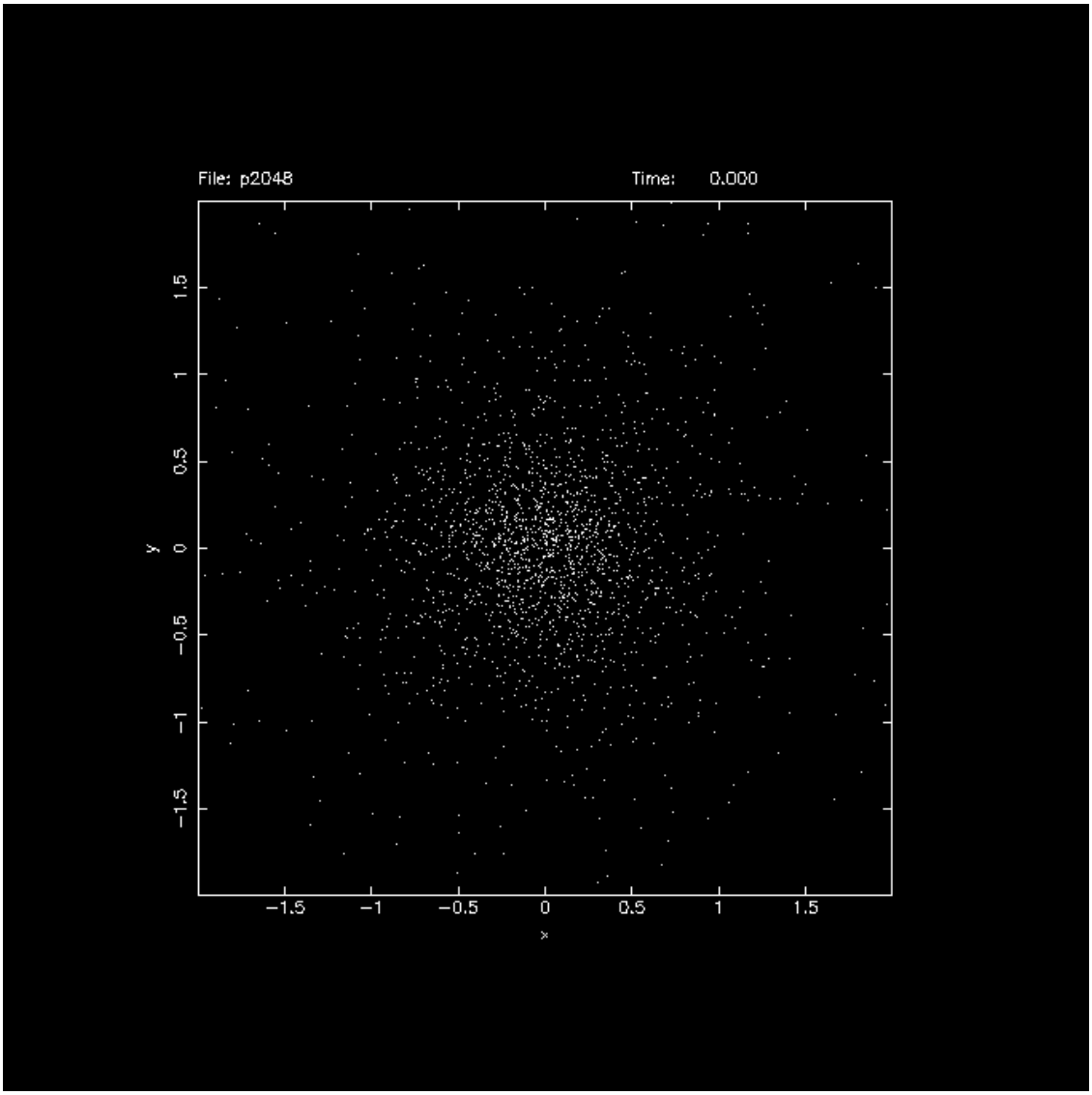
First create the initial conditions mkplummer, and view them with the tsf program:

[5]:
```
rm -f p2048
mkplummer p2048 2048 seed=123
tsf p2048
```

```
char Headline[28] "init_xrandom: seed used 123"
char History[43] "mkplummer p2048 2048 seed=123 VERSION=3.0c"
set SnapShot
  set Parameters
    int Nobj 2048
    double Time 0.00000
  tes
  set Particles
    int CoordSystem 66306
    double Mass[2048] 0.000488281 0.000488281 0.000488281 0.000488281
      0.000488281 0.000488281 0.000488281 0.000488281 0.000488281
      0.000488281 0.000488281 0.000488281 0.000488281 0.000488281
      0.000488281 0.000488281 0.000488281 0.000488281 0.000488281
      . . .
    double PhaseSpace[2048][2][3] -0.570453 -0.0544111 -0.627691
      -0.0761437 -0.0984996 0.337867 4.84828 -0.318906 -1.70248
      0.419009 0.228663 0.171499 0.584347 0.246823 -0.113503 0.0872366
      0.00176906 -0.340730 0.416242 -0.0460422 -0.188560 -0.739448
      . . .
  tes
tes
```

[6]:
```
snapplot p2048 yapp=aarseth_fig1.png/png
```

Plot of a Plummer sphere with 2048 particles, created with the snapplot program.




### 3.0.2 Comparing FORTRAN and C versions

We first compare the performance of the FORTRAN and C versions of *nbody0*. We use **out=.** to not have to write an output file, perhaps saving some overhead. Using a *dot* for a filename is a NEMO feature. The default integration time **tcrit=2** is used, and a total of 161,238 (variable) integration steps will be taken. A typical CPU time will be 2 seconds.

[7]:
```
/usr/bin/time nbody0  p2048 . tcrit=2   # Fortran version
/usr/bin/time nbody00 p2048 . tcrit=2   # C version
```

```
### nemo Debug Info: time = 0   steps = 0   energy = -0.244373 cpu =     0.0015
min
### nemo Debug Info: time = 0.25   steps = 19181   energy = -0.244375 cpu =
0.00517 min
### nemo Debug Info: time = 0.5   steps = 39859   energy = -0.244383 cpu =
0.00933 min
```

```
### nemo Debug Info: time = 0.75   steps = 60485   energy = -0.244384 cpu =
0.0137 min
### nemo Debug Info: time = 1   steps = 80889   energy = -0.244386 cpu =
0.0178 min
### nemo Debug Info: time = 1.25   steps = 100950   energy = -0.244386 cpu =
0.0213 min
### nemo Debug Info: time = 1.5   steps = 121284   energy = -0.244389 cpu =
0.0252 min
### nemo Debug Info: time = 1.75   steps = 141176   energy = -0.244396 cpu =
0.0293 min
### nemo Debug Info: time = 2   steps = 161238   energy = -0.244397 cpu =
0.0333 min
2.00user 0.00system 0:02.00elapsed 99%CPU (0avgtext+0avgdata 2444maxresident)k
352inputs+0outputs (1major+333minor)pagefaults 0swaps
### nemo Debug Info: time = 0   steps = 0   energy = -0.244373 cpu =     0.0015
min
### nemo Debug Info: time = 0.25   steps = 19181   energy = -0.244375 cpu =
0.0055 min
### nemo Debug Info: time = 0.5   steps = 39859   energy = -0.244383 cpu =
0.00983 min
### nemo Debug Info: time = 0.75   steps = 60485   energy = -0.244384 cpu =
0.0143 min
### nemo Debug Info: time = 1   steps = 80889   energy = -0.244386 cpu =
0.0188 min
### nemo Debug Info: time = 1.25   steps = 100950   energy = -0.244386 cpu =
0.0232 min
### nemo Debug Info: time = 1.5   steps = 121284   energy = -0.244389 cpu =
0.0285 min
### nemo Debug Info: time = 1.75   steps = 141176   energy = -0.244396 cpu =
0.0332 min
### nemo Debug Info: time = 2   steps = 161238   energy = -0.244397 cpu =
0.0375 min
### nemo Debug Info: Time spent in searching for next advancement: 0.37
### nemo Debug Info: Energy conservation: -2.47054e-05 / -0.244373 = 0.000101087
### nemo Debug Info: Time resets needed 0 times / 9 dumps
2.25user 0.08system 0:02.35elapsed 99%CPU (0avgtext+0avgdata 2852maxresident)k
376inputs+0outputs (1major+333minor)pagefaults 0swaps
```

#### 3.0.3 Reproducability

If NEMO's random number generator is working correctly, the number of steps and energy at time=2 should be exactly

```
time = 2   steps = 161238   energy = -0.244397
```

and although the CPU time varies per machine, my 2023 "Ultra 7 155H" laptop CPU took about 1.7sec for **nbody0** and 2.0sec for **nbody00**. Also notable is that the C version does use a small amount (4%) of system time, whereas FORTRAN took 0.

### 3.0.4 nbody00_ff

The previous two discussed versions of *nbody0* have a NEMO command line interface, and can read NEMO files. The original pure FORTRAN version does not have a NEMO CLI. It reads a one line header with 6 numbers from the terminal (stdin), followed by **n** (the number of bodies) lines containing the mass, position and velocity (7 values per line). The header contains **n,eta,deltat,tcrit,eps2,reset**. Such an input file can be easily created using basic Unix and NEMO tools:

[8]:
```
# create a fresh Plummer sphere with 5 particles, again with a fixed seed
echo "Creating initial conditions:"
echo "----------------------------"
rm -f p5
mkplummer p5 5 seed=123

# convert the snapshot to the input file that nbody0_ff needs
echo "5 0.02 1.0 10 0.0001 1"                > input5
snapprint p5 m,x,y,z,vx,vy,vz format=%.15g >> input5

# run nbody0_ff
echo "Running nbody0_ff:"
echo "----------------------"
nbody0_ff < input5

# run nbody0, and compare the phase space coordinates at times=10
echo "Running nbody0 to compare:"
echo "------------------------------"
nbody0 p5 - deltat=1 eps=0.01 tcrit=10 | snaptrim - - times=10 | snapprint -
```

```
Creating initial conditions:
----------------------------
### nemo Debug Info: m x y z vx vy vz
Running nbody0_ff:
----------------------
 Enter n,eta,deltat,tcrit,eps2,reset:
      0.20        -1.62     -0.19     -0.08        -0.05     -0.31      0.28
0.0881       1
      0.20         3.80     -0.46     -1.16         0.45      0.01      0.11
0.4797       2
      0.20        -0.46      0.11      0.43         0.12     -0.21     -0.40
0.0343       3
      0.20        -0.63     -0.19      0.36        -0.71      0.55     -0.18
0.0347       4
      0.20        -1.08      0.73      0.46         0.19     -0.03      0.20
0.0815       5
     time =   0.00  steps =     0 energy =   -0.1811

      0.20        -1.46     -0.39      0.26         0.42      0.02      0.36
```

```
0.0416       1
      0.20         4.23     -0.44     -1.04         0.42      0.02      0.12
0.4820       2
      0.20        -1.16      0.06      0.09        -0.99      0.47     -0.08
0.0361       3
      0.20        -0.82      0.44      0.27        -0.17      0.24      0.53
0.0099       4
      0.20        -0.79      0.34      0.42         0.31     -0.74     -0.93
0.0102       5
     time =   1.00  steps =   125 energy =   -0.1811

      0.20        -0.78      0.04      0.34         0.81      0.48     -0.69
0.0195       1
      0.20         4.64     -0.42     -0.92         0.40      0.02      0.12
1.0269       2
      0.20        -1.72      0.42      0.23        -0.13      0.21      0.20
0.0713       3
      0.20        -1.00     -0.24      0.26        -0.63     -0.18      0.10
0.0182       4
      0.20        -1.13      0.20      0.09        -0.45     -0.53      0.27
0.0287       5
     time =   2.00  steps =   315 energy =   -0.1811

      0.20        -0.45      0.38     -0.35         0.10      0.26     -0.61
0.1075       1
      0.20         5.03     -0.40     -0.80         0.38      0.02      0.13
0.3865       2
      0.20        -1.52      0.29      0.41         0.38     -1.05      0.39
0.0058       3
      0.20        -1.59      0.18      0.47        -0.46      0.88     -0.01
0.0054       4
      0.20        -1.46     -0.45      0.26        -0.39     -0.12      0.10
0.0506       5
     time =   3.00  steps =   698 energy =   -0.1812

      0.20        -0.44      0.61     -0.88        -0.05      0.20     -0.47
0.1462       1
      0.20         5.40     -0.38     -0.67         0.36      0.02      0.13
0.5662       2
      0.20        -1.85     -0.03      0.57         0.25     -0.92     -0.57
0.0021       3
      0.20        -1.29     -0.10      0.47        -0.18     -0.60      0.10
0.0471       4
      0.20        -1.82     -0.09      0.51        -0.38      1.29      0.81
0.0021       5
     time =   4.00  steps =   900 energy =   -0.1812

      0.20        -0.52      0.78     -1.30        -0.11      0.15     -0.37
```

```
0.2659       1
      0.20         5.75     -0.36     -0.54         0.34      0.02      0.13
0.9946       2
      0.20        -1.55      0.10      0.38         0.19     -0.21      0.23
0.0519       3
      0.20        -1.92     -0.34      0.64        -0.70      0.38      0.45
0.0204       4
      0.20        -1.76     -0.18      0.82         0.28     -0.34     -0.43
0.0196       5
     time =   5.00  steps =  1186 energy =   -0.1812

      0.20        -0.65      0.91     -1.63        -0.15      0.11     -0.30
0.1796       1
      0.20         6.08     -0.33     -0.41         0.33      0.03      0.13
0.5089       2
      0.20        -1.94     -0.36      0.78        -0.87     -0.07      0.06
0.0333       3
      0.20        -1.66     -0.04      0.74         0.48     -0.39     -0.55
0.0164       4
      0.20        -1.83     -0.18      0.52         0.21      0.32      0.66
0.0171       5
     time =   6.00  steps =  1315 energy =   -0.1812

      0.20        -0.81      1.01     -1.90        -0.17      0.09     -0.24
0.4639       1
      0.20         6.40     -0.31     -0.28         0.32      0.03      0.13
1.2083       2
      0.20        -2.15     -0.32      0.85        -0.53     -0.10      0.36
0.0245       3
      0.20        -2.14     -0.09      0.61         0.20      0.27     -0.17
0.0233       4
      0.20        -1.30     -0.29      0.72         0.18     -0.29     -0.08
0.0595       5
     time =   7.00  steps =  1590 energy =   -0.1812

      0.20        -0.98      1.08     -2.12        -0.18      0.06     -0.19
0.3726       1
      0.20         6.71     -0.28     -0.15         0.30      0.03      0.13
1.7834       2
      0.20        -2.31     -0.11      0.81         0.55      0.32     -0.32
0.0143       3
      0.20        -2.08     -0.17      0.82        -0.44     -0.26      0.45
0.0139       4
      0.20        -1.33     -0.52      0.63        -0.23     -0.15     -0.07
0.1313       5
     time =   8.00  steps =  1655 energy =   -0.1812

      0.20        -1.17      1.14     -2.29        -0.19      0.04     -0.15
```

```
0.4863       1
      0.20         7.01     -0.25     -0.01         0.29      0.03      0.13
1.7834       2
      0.20        -2.15     -0.28      0.95         0.14      0.10     -0.26
0.0380       3
      0.20        -1.90     -0.08      0.72         0.48     -0.44      0.15
0.0293       4
      0.20        -1.79     -0.52      0.63        -0.73      0.26      0.13
0.0289       5
     time =   9.00  steps =  1802 energy =   -0.1812

      0.20        -1.36      1.17     -2.42        -0.20      0.02     -0.10
0.5768       1
      0.20         7.30     -0.23      0.12         0.28      0.03      0.13
0.9696       2
      0.20        -2.39     -0.46      0.73        -0.23      0.08     -0.43
0.0119       3
      0.20        -2.23     -0.47      0.72        -0.73     -0.27      0.38
0.0106       4
      0.20        -1.32     -0.02      0.85         0.87      0.14      0.02
0.0620       5
     time =  10.00  steps =  2351 energy =   -0.1813

Running nbody0 to compare:
------------------------------
### nemo Debug Info: x y z vx vy vz
### nemo Debug Info: time = 0   steps = 0   energy = -0.181136 cpu =          0
min
### nemo Debug Info: time = 1   steps = 125   energy = -0.18114 cpu =          0
min
### nemo Debug Info: time = 2   steps = 315   energy = -0.181137 cpu =
0 min
### nemo Debug Info: time = 3   steps = 698   energy = -0.181227 cpu =
0 min
### nemo Debug Info: time = 4   steps = 900   energy = -0.181224 cpu =
0 min
### nemo Debug Info: time = 5   steps = 1186   energy = -0.181229 cpu =
0 min
### nemo Debug Info: time = 6   steps = 1315   energy = -0.181232 cpu =
0 min
### nemo Debug Info: time = 7   steps = 1590   energy = -0.181231 cpu =
0 min
### nemo Debug Info: time = 8   steps = 1655   energy = -0.181234 cpu =
0 min
### nemo Debug Info: time = 9   steps = 1802   energy = -0.181237 cpu =
0 min
### nemo Debug Info: time = 10   steps = 2351   energy = -0.181269 cpu =
0 min
```

```
### nemo Debug Info: Using timefuzz=1e-05
### nemo Debug Info: time = 10  npart =   1     ndiag =   0     outputing
particles
### nemo Debug Info: copy_item: 4
### nemo Debug Info: copy_item: 8
### nemo Debug Info: copy_item: 4
### nemo Debug Info: copy_item: 40
### nemo Debug Info: copy_item: 240
-1.36279 1.16632 -2.41738 -0.19854 0.021288 -0.104094
7.29869 -0.227201 0.116816 0.283669 0.0275029 0.13166
-2.38877 -0.457732 0.726767 -0.227988 0.0829789 -0.433205
-2.22922 -0.465448 0.721034 -0.727559 -0.272988 0.382557
-1.31639 -0.0168633 0.85151 0.87065 0.141302 0.0230623
```

Did you see something like this?

```
-1.36279 1.16632 -2.41738 -0.19854 0.021288 -0.104094
7.29869 -0.227201 0.116816 0.283669 0.0275029 0.13166
-2.38877 -0.457732 0.726767 -0.227988 0.0829789 -0.433205
-2.22922 -0.465448 0.721034 -0.727559 -0.272988 0.382557
-1.31639 -0.0168633 0.85151 0.87065 0.141302 0.0230623
```

if so, then it's reproducable.

#### 3.0.5 Comparing nbody0 and nbody0_ff

Apart from the limited accuracy that nbody0_ff shows, the comparison is excellent, as well as number of steps taken and the energy in the final snapshot:

```
time = 10   steps = 2351   energy = -0.181269
```

### 3.1 Evolution of a Plummer Sphere

Here we evolve the Plummer sphere for several dynamical times. This should take about 10 seconds for a default softening eps=0.05, which will go faster for more gravitational softening.

[9]:
```
rm -f p2048a.dat
/usr/bin/time nbody00 in=p2048 out=p2048a.dat tcrit=10 deltat=0.1 eps=0.05␣
 ↪debug=-1
```

```
11.30user 0.42system 0:11.72elapsed 99%CPU (0avgtext+0avgdata 2808maxresident)k
0inputs+22688outputs (0major+329minor)pagefaults 0swaps
```

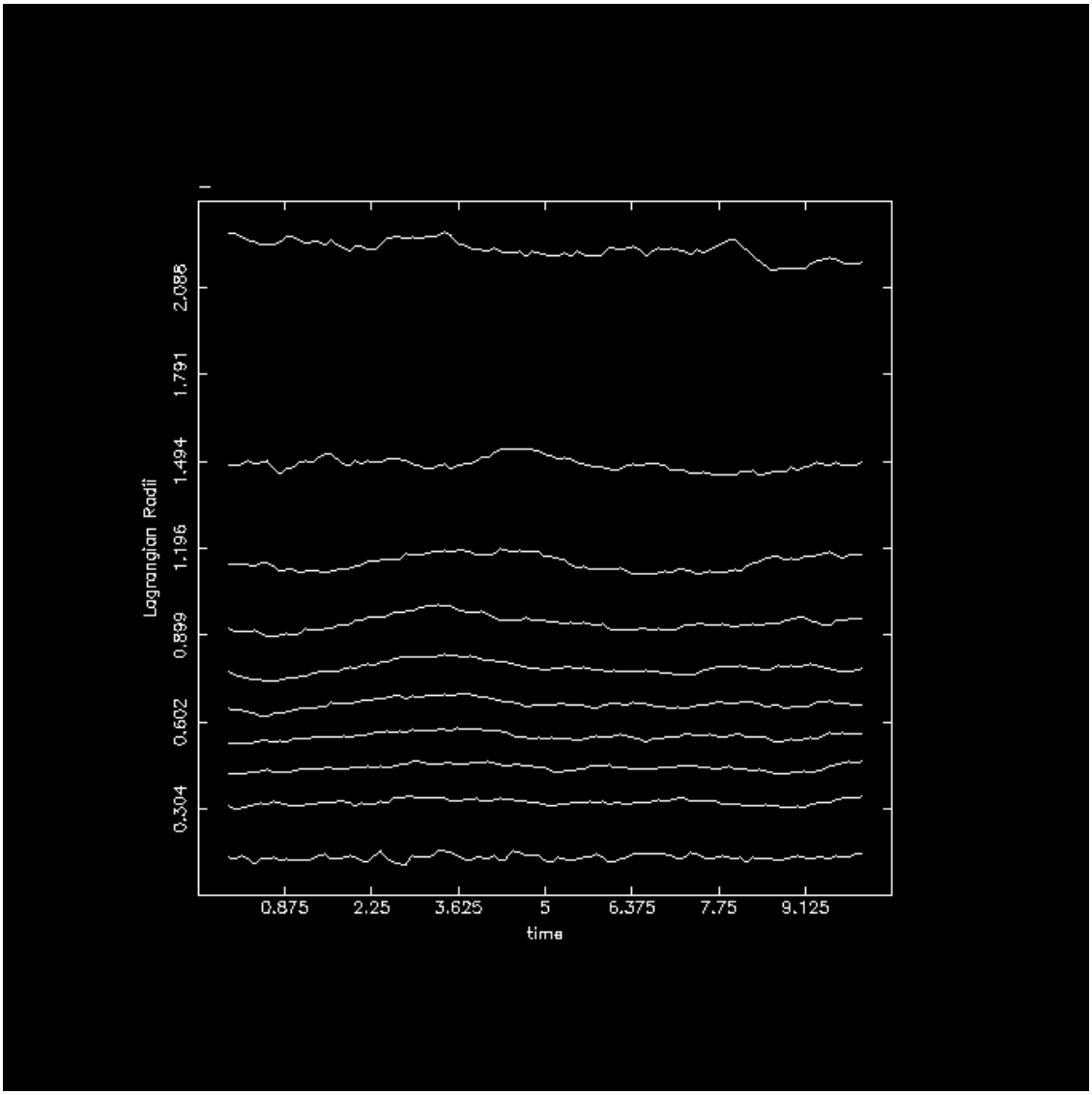
[10]:
```
snapmradii p2048a.dat 0.01,0.1:0.9:0.1,0.99 |\
    tabplot - 1 2:11 line=1,1 xlab=time ylab="Lagrangian Radii"␣
 ↪yapp=aarseth_fig2.png/png
```

```
### nemo Debug Info: read 101 points
### nemo Debug Info: min and max value in xcolumns 1: [0.000000 : 10.000000]
### nemo Debug Info: min and max value in ycolumns 2:11: [0.115023 : 2.277360]
```

```
### nemo Debug Info: X:min and max value reset to : [-0.500000 : 10.500000]
### nemo Debug Info: Y:min and max value reset to : [0.006906 : 2.385477]
```

Plot of time evolution of the lagrangian mass radii for a 2048 Plummer sphere. It is comforting to see that the inner 1% of the particles do not seem to evolve. Maybe a slight indication the 99% mass radius contracts a tiny bit. But in between some oscillation on a dynamical timescale seem present that move outwards. Experimenting with larger softening will give a different picture, and show that the sphere is expanding in an oscillating way, presumably to find a different equilibrium shape.

A table was created with snapmradii, the data were piped into the plotting program tabplot




[ ]: