

# User Guide for N-MODY: a parallel code for collisionless N-body simulations in modified Newtonian dynamics.

Pasquale Londrillo<sup>1</sup> and Carlo Nipoti<sup>2\*</sup>

<sup>1</sup>INAF - Osservatorio Astronomico di Bologna, via Ranzani 1, I-40127 Bologna, Italy

<sup>2</sup>Dipartimento di Astronomia, Università di Bologna, via Ranzani 1, 40127 Bologna, Italy

January 11, 2008

## Contents

<b>1</b>	<b>Overview</b>	<b>2</b>
1.1	The code procedures . . . . .	2
1.2	Parallelization . . . . .	2
<b>2</b>	<b>A MOND potential solver in spherical coordinates</b>	<b>3</b>
2.1	The computational grid . . . . .	4
2.2	A Newton-like relaxation procedure . . . . .	4
2.3	Implementation of the relaxation procedure. . . . .	5
2.4	Performances . . . . .	6
<b>3</b>	<b>Particle-Mesh scheme in spherical grids</b>	<b>7</b>
3.1	Assignment of mass density . . . . .	7
3.2	Interpolation of grid acceleration . . . . .	7
3.3	Leap-frog particles time-stepping . . . . .	8
<b>4</b>	<b>Compiling and running N-MODY</b>	<b>8</b>
4.1	Source files . . . . .	9
4.2	Code units . . . . .	9
4.3	Input data . . . . .	9
4.3.1	Simulation parameters . . . . .	9
4.3.2	Particle data . . . . .	11
4.4	Output data . . . . .	12

---

\*E-mail: pasquale.londrillo@oabo.inaf.it, carlo.nipoti@unibo.it

# 1 Overview

Modified Newtonian dynamics (MOND) is a non-linear alternative gravity theory, originally proposed by Milgrom (1983) and Bekenstein & Milgrom (1984) to explain the observed kinematics of galaxies without dark matter. While most N-body codes for simulations of collisionless systems in Newtonian gravity are based on the gridless multipole expansion treecode scheme (Barnes & Hut 1986; see also Dehnen 2002), the non-linearity of the MOND field equation forces one to resort to other methods, such as the particle-mesh technique (see Hockney & Eastwood 1988). In this approach, particles are moved under the action of a gravitational field computed on a grid, with particle-mesh interpolation providing the link between the two representations. N-MODY is a parallel, three-dimensional particle-mesh code to run N-body simulations of collisionless systems in MOND. Note that the potential solver of N-MODY is based on a grid in spherical coordinates, so periodic boundary conditions are not implemented, and the code is best suited for simulation of isolated galaxies. The code and the potential solver have been presented and tested in Ciotti, Nipoti, Londrillo (2006) and Nipoti et al. (2007abc).

## 1.1 The code procedures

N-MODY implements a particle-mesh scheme in spherical coordinates, following these main computational steps:

- 1) for a given distribution N-particle, a grid-based density field is reconstructed by mass deposition using linear or quadratic shape functions. Particles are represented by a six-component array  $(\mathbf{x}_i, \mathbf{v}_i)$  of positions and velocities in Cartesian components.
- 2) For a given density field, the MOND grid-based acceleration and potential fields are then computed. As an alternative, the code also provides a fast Newtonian solver.
- 3) To move Lagrangian particles, the acceleration spherical components are first interpolated using the same linear or quadratic shape function at each particle position and then transformed to Cartesian vector components.
- 4) Finally, particle positions and velocities are advanced in time using a leap-frog scheme. An explicit, three-stage fourth-order leap-frog scheme is also implemented.

## 1.2 Parallelization

In N-MODY, steps 1) and 3) can be executed in parallel using MPI routines. To that purpose, particles are uniformly distributed among the processors (PEs) following the sequential ordering provided by their initial memory addresses and then never exchanged among different PEs. This simple strategy assures a full efficiency in parallel execution, but entails more memory resources than in a standard domain decomposition. In fact, grid data computed on the overall computational domain must now be at disposal of each PE at each time step for particles interpolation and move. It is then a viable parallelization strategy only for reasonable grid size and for sufficiently high particles number per grid cell, in a way that the CPU time spent for particles move and interpolation is dominant or at least comparable to the time needed for grid based computations.

Step 2) is only partially parallelized, following a specific procedure detailed below. This part, containing the MOND potential solver, represents a new contribution to the fast numerical solution of a non-linear elliptic equation, and will be considered first in the following section.

## 2 A MOND potential solver in spherical coordinates

N-MODY solves the non-relativistic field equation (Bekenstein & Milgrom 1984)

$$\nabla \cdot \left[ \mu \left( \frac{g}{a_0} \right) \nabla \phi(\mathbf{x}) \right] = 4\pi G \rho(\mathbf{x}), \quad (1)$$

where  $\phi(\mathbf{x})$  is the gravitational potential produced by the density distribution  $\rho(\mathbf{x})$ ,  $g(\mathbf{x}) = |\mathbf{g}(\mathbf{x})|$ , with  $\mathbf{g}(\mathbf{x}) = -\nabla\phi(\mathbf{x})$  and  $a_0$  is the MOND acceleration parameter. The MOND interpolating function  $\mu(y)$  is a monotonic function such that

$$\mu(y) \sim \begin{cases} y & \text{for } y \ll 1, \\ 1 & \text{for } y \gg 1. \end{cases} \quad (2)$$

In the present version of N-MODY we adopt

$$\mu(y) = \frac{y}{\sqrt{1+y^2}}, \quad (3)$$

but it is clear that a trivial modification of the code allows to implement any other continuous function  $\mu$  with the required asymptotic behaviour<sup>1</sup>.

Alternatively, the user of N-MODY can also choose to simulate a Newtonian system, in which case the Poisson equation is solved instead of equation (1), or a system in the so-called ‘deep MOND regime’, i.e. obeying the equation

$$\nabla \cdot (\|\nabla\phi\| \nabla\phi) = 4\pi G a_0 \rho. \quad (4)$$

We consider only systems of finite mass

$$M = \int \rho(\mathbf{x}) d^3\mathbf{x}, \quad (5)$$

for which the boundary conditions of equation (1) are given by the asymptotic behaviour  $\nabla\phi \rightarrow O(1/r)$  for  $r \equiv \|\mathbf{x}\| \rightarrow \infty$ .

To solve numerically the MOND field equation it is then necessary:

1. to discretize a sufficiently large computational domain, in a way the asymptotic boundary condition can be represented with reasonable accuracy;
2. to use a relaxation method to solve the non-linear elliptic equation (1).

---

<sup>1</sup>In versions of MOND based on Bekenstein’s (2004) covariant theory TeVeS, there is a scalar field  $\phi_s$  obeying equation (1), but with an unbounded interpolating function  $\mu_s$  instead of the bounded function  $\mu$ . We verified that our code can be adapted to solve for  $\phi_s$  (see Famaey et al. 2007).

## 2.1 The computational grid

To accomplish task 1), N-MODY uses a spherical grid  $(r, \vartheta, \varphi)$  with radial coordinate represented by the invertible mapping on the angular  $\xi$  coordinate,  $0 \leq \xi < \pi/2$ ,

$$r(\xi) = L \tan^\alpha \xi, \quad r'(\xi) = \frac{\alpha L \tan^{\alpha-1} \xi}{\cos^2 \xi}, \quad (6)$$

where the mapping index  $\alpha = 1$  or  $\alpha = 2$  and the scale length  $L$  are user provided parameters. In this representation, any unbounded radial range is mapped onto the finite open interval  $[0, \pi/2)$ . The radial derivative is then expressed as

$$\frac{\partial}{\partial r} = \frac{1}{r'(\xi)} \frac{\partial}{\partial \xi}. \quad (7)$$

The  $\xi$  coordinate is discretized into the uniform grid

$$\xi_i = (i + 1/2)\Delta\xi, \quad \Delta\xi = \frac{\pi}{2N_r}, \quad i = 0, 1, \dots, N_r, \quad (8)$$

so the corresponding discretized radial variable  $r_i = r(\xi_i)$  avoids the  $r = 0$  and  $1/r = 0$  singular points. The other coordinates  $(\vartheta, \varphi)$  are discretized in the uniform grids

$$\vartheta_j = (j + 1/2)\Delta\vartheta, \quad \Delta\vartheta = \pi/N_\vartheta, \quad j = 0, 1, \dots, N_\vartheta - 1 \quad (9)$$

to avoid the  $\vartheta = 0, \pi$  singular points, and

$$\varphi_k = k\Delta\varphi, \quad \Delta\varphi = 2\pi/N_\varphi, \quad k = 0, 1, \dots, N_\varphi - 1 \quad (10)$$

Finally, using these uniform discretizations, the corresponding  $(\xi, \vartheta, \varphi)$  numerical derivatives are computed by second-order or fourth-order finite difference central schemes.

## 2.2 A Newton-like relaxation procedure

To solve the non-linear MOND elliptic problem given by the functional equation

$$\hat{M}[\phi(\mathbf{x})] \equiv \nabla \cdot \left[ \mu \left( \frac{g}{a_0} \right) \nabla \phi(\mathbf{x}) \right] - 4\pi G \rho(\mathbf{x}) = 0, \quad g = O(r^{-1}) \text{ for } r \rightarrow \infty, \quad (11)$$

where  $\rho(\mathbf{x})$  is assigned and  $\mathbf{x} = (r, \vartheta, \varphi)$ , N-MODY implements a Newton iterative procedure. Starting with an approximate guess solution  $\phi^{(0)}$  having the required asymptotic behaviour, a sequence of linear problems

$$\hat{R}^{(n)} \delta\phi^{(n)} = -\hat{M}[\phi^{(n)}], \quad n = 0, 1, \dots \quad (12)$$

is then solved for the increments  $\delta\phi^{(n)} = \phi^{(n+1)} - \phi^{(n)}$ , each  $\phi^{(n)}$  provisional solution having the same asymptotic behaviour as the initial guess  $\phi^{(0)}$ . Here the choice of the relaxation linear operator  $\hat{R}^{(n)}$  is based on the requirement that it must assure convergence in the maximum norm  $\|\cdot\|$

$$\|\delta\phi^{(n)}\| = \left\| \left[ \hat{R}^{(n)} \right]^{-1} \hat{M}[\phi^{(n)}] \right\| \leq \|\delta\phi^{(n-1)}\|, \quad n = 1, 2, \dots \quad (13)$$

and it must be easy to invert. In a classical Newton method, one would use  $\hat{R}^{(n)} = \delta\hat{M}^{(n)}$ ,

$$\delta\hat{M}^{(n)}[\delta\phi^{(n)}] = \hat{M}[\phi^{(n+1)}] - \hat{M}[\phi^{(n)}] + O[(\delta\phi^{(n)})^2], \quad (14)$$

which in the case of equation (1) takes the form:

$$\delta\hat{M}^{(n)} = \mu^{(n)}\nabla^2 + \delta\hat{M}_1^{(n)} \quad (15)$$

where

$$\delta\hat{M}_1^{(n)} = (\nabla\mu^{(n)}) \cdot \nabla + \nabla \cdot [\mu^{(n)}\mathbf{g}^{(n)}(\mathbf{g}^{(n)} \cdot \nabla)] \quad (16)$$

and  $\mu'^{(n)} \equiv \mu'(\mathbf{g}^{(n)}/a_0)/g^{(n)}a_0$ . Boundedness of the inverse of the operator  $\delta\hat{M}^{(n)}$  assures quadratic convergence of the scheme for  $\phi^{(0)}$  sufficiently close to the sought solution (e.g. Stoer & Bulirsch 1980). Unfortunately,  $[\delta\hat{M}^{(n)}]^{-1}$  is difficult to compute, so we discretize the simpler linear relaxation operator

$$\hat{R}^{(n)} = \omega\mu^{(n)}\nabla^2, \quad (17)$$

where  $\omega > 1$  is an empirical relaxation parameter. As an approximation of the Newton relaxation operator  $\delta\hat{M}$ , this choice assures a lower convergence rate, but has clear computational advantages. Using equation (12) and the identity

$$\hat{M}(\phi^{(n)}) = \delta\hat{M}^{(n-1)}[\delta\phi^{(n-1)}] + \hat{M}[\phi^{(n-1)}] + O[(\delta\phi^{(n-1)})^2], \quad (18)$$

it follows that the condition for convergence (13) requires

$$\left\| \frac{(\nabla^2)^{-1}\delta\hat{M}^{(n-1)}}{\omega\mu^{(n)}} - \frac{\mu^{(n-1)}}{\mu^{(n)}}I \right\| < 1. \quad (19)$$

### 2.3 Implementation of the relaxation procedure.

N-MODY then solves the sequence of Poisson equations:

$$\nabla^2\delta\phi^{(n)} = \mathcal{S}^n \quad (20)$$

with source term given by

$$\mathcal{S}^n = -\frac{1}{\omega\mu^{(n)}}\hat{M}[\phi^{(n)}] \quad (21)$$

In spherical coordinates, the Laplacian operator has the form

$$\nabla^2 \equiv \frac{1}{r^2} \left[ \frac{\partial}{\partial r} \left( r^2 \frac{\partial}{\partial r} \right) + \hat{L}_\vartheta + \hat{L}_\varphi \right], \quad (22)$$

where

$$\hat{L}_\vartheta \equiv \frac{1}{\sin\vartheta} \frac{\partial}{\partial\vartheta} \left( \sin\vartheta \frac{\partial}{\partial\vartheta} \right), \quad \hat{L}_\varphi \equiv \frac{1}{\sin^2\vartheta} \frac{\partial^2}{\partial\varphi^2}. \quad (23)$$

After expanding the source term  $\mathcal{S}^n(r, \vartheta, \varphi)$  and the unknown function  $\delta\phi^{(n)}(r, \vartheta, \varphi)$  in spherical harmonics (or Fourier-Legendre) components

$$\delta\phi^{(n)}(r, \vartheta, \varphi) = \sum_{l,m} \delta\phi_{l,m}^{(n)}(r) Y_l^m(\vartheta, \varphi) \quad (24)$$

equation (20) takes the simple form

$$\frac{1}{r} \left[ \frac{\partial}{\partial r} \left( r^2 \frac{\partial}{\partial r} \right) - l(l+1) \right] \delta\phi_{l,m}^{(n)}(r) = r\mathcal{S}_{l,m}^{(n)}(r), \quad (25)$$

where we multiplied both sides by  $r$  to avoid the singularity in the source term for the astrophysically relevant case of  $\rho \sim r^{-1}$  central density profiles. Equation (25), involving derivatives only in the radial coordinates, is discretized by central finite differences.

The relaxation scheme in N-MODY is implemented in the following steps:

1. For assigned density  $\rho(r, \vartheta, \varphi)$  the initial guess solution is chosen to be the *exact* spherical symmetric MOND solution corresponding to the angle-averaged density distribution:

$$\rho_{0,0}(r) = \frac{1}{4\pi} \int_0^{2\pi} \int_0^\pi \rho(r, \vartheta, \varphi) \sin(\vartheta) d\vartheta d\varphi. \quad (26)$$

This solution satisfies the boundary condition, providing the values of the potential  $\phi^{(0)}$  and of the radial acceleration  $g_r^{(0)}$  at the boundary grid point  $r_{N_r}$ . In N-body time evolving systems, this choice holds only for the initial time  $t = 0$  solution. At subsequent time steps, the initial guess solution will be provided by the numerical solution found in the previous time step.

2. For assigned acceleration  $\mathbf{g}^{(n)}(r, \vartheta, \varphi)$  at the iteration level  $n = 0, 1, \dots$ , the source term  $r\mathcal{S}^{(n)}(r, \vartheta, \varphi)$  is evaluated, using central finite differences to approximate space derivatives in all the coordinate variables.
3. The source term is then transformed into Fourier-Legendre components by:

$$\mathcal{S}_{l,m}(r_i) = \sum_j \sum_k \mathcal{S}(r_i, \vartheta_j, \varphi_k) e^{-im\varphi_k} P_{l,m}^{-1}(\vartheta_j) \quad (27)$$

where  $P_{l,m}^{-1}(\vartheta_k)$  are the exact inverse of the associated Legendre polynomials, with unit normalization  $\sum_j P_{l,m}(\vartheta_j) P_{l',m}^{-1}(\vartheta_j) = \delta_{l,l'}$ .

4. The operator in equation (25) is discretized in the radial coordinate by using finite differences for first and second derivatives. It results a tri-diagonal or penta-diagonal matrix of order  $N_r + 1$  that can be easily inverted by using a standard Lower-Upper triangular (LU) decomposition, to solve for the  $\delta\phi_{l,m}^{(n)}$  variables with boundary conditions  $\delta\phi_{l,m}^{(n)}(r_{N_r}) = 0$ .
5. Finally, the potential increments  $\delta\phi_{l,m}^{(n)}(r_i)$  are back transformed into the  $(\vartheta, \varphi)$  coordinate space and the corresponding acceleration increments  $\delta\mathbf{g}^{(n)}$  are evaluated using finite differences, to move to the next level solution

$$\mathbf{g}^{(n+1)} = \mathbf{g}^{(n)} + \delta\mathbf{g}^{(n)} \quad (28)$$

until convergence is achieved. We notice that in this relaxation scheme the potential is never used for intermediate solutions, only the potential increments being required. The final acceleration field still remains a potential gradient because the initial guess and all the intermediate solutions keep the irrotational form. The final potential field can be computed (when needed for outputs) from the final acceleration field by numerical inversion of  $\nabla\phi = \mathbf{g}$ .

6. To evaluate convergence, N-MODY adopts the maximum and r.m.s. norms to check for the error condition in the relative acceleration increment  $Err \equiv \|\delta g/g\| < \varepsilon$ , where  $\varepsilon$  is a user defined tolerance parameter.

## 2.4 Performances

By choosing a tolerance parameter  $\varepsilon = 10^{-3}$  for convergence in maximum norm (corresponding to a  $\simeq 10^{-4}$  value in r.m.s. norm), with a relaxation parameter  $\omega^{-1} = 0.3 - 0.5$ , the N-MODY

solver provides the required solution in a few 5–10 iterations. For typical N-body systems, these error bounds correspond to an approximation in maximum norm of the  $\hat{M}(\phi) = 0$  equation, of  $\max |\hat{M}(\phi)| \simeq 0.1$  and  $\text{r.m.s}|\hat{M}(\phi)| \simeq 0.01$ .

Computational time needed in the MOND potential solver scales roughly with the total number of grid points  $N_g = (N_r + 1)N_\vartheta N_\varphi$ . In fact, at the scalar level (only one PE used), MOND solver needs a  $\simeq 10\%$  of the time spent to run particles, in a typical configuration where  $N_p = 10N_g$ . For large grids, and lower number of particles per cell, some advantages are obtained by parallelization of the MOND solver. To that purpose, when running with  $N_{\text{PE}}$  processors, N-MODY adopts a domain decomposition during the iteration cycle, by assigning at each PE only a sector of the  $\vartheta$  coordinate containing  $N_\vartheta/N_{\text{PE}}$  grid points (note that the condition  $N_{\text{PE}} \leq N_\vartheta/4$  must be satisfied). This simple strategy results to be effective, even if several operations still require the full grid, resulting in a 70% of parallelization rate.

### 3 Particle-Mesh scheme in spherical grids

For a given set of  $N_p$  point particles with mass  $m = M/N_p$ , the particles Cartesian positions  $(x_p, y_p, z_p)$ ,  $p = 1, 2, \dots, N_p$  are first converted into radial coordinates by:

$$r_p = \sqrt{x_p^2 + y_p^2 + z_p^2}, \quad \xi_p = \tan^{-1}(r_p/L)^{1/\alpha}, \quad (29)$$

$$\vartheta_p = \cos^{-1}(z_p/r_p), \quad \varphi_p = \tan^{-1}(y_p/x_p). \quad (30)$$

#### 3.1 Assignment of mass density

For mass deposition on the radial grid, using linear or quadratic shape functions  $S(u - u_p)$  of compact support for each  $u$  coordinate, the resulting mass density at the grid point of indices  $(i, j, k)$  is defined by

$$D_{i,j,k} = m \sum_p [S(\xi_i - \xi_p)S(\vartheta_j - \vartheta_p)S(\varphi_k - \varphi_p)] \quad (31)$$

the sum extending only at the particle positions where the shape function is non zero. Here

$$D_{i,j,k} \equiv \left[ \rho r^2 r' \sin \vartheta \Delta \xi \Delta \vartheta \Delta \varphi \right]_{i,j,k} \quad (32)$$

and then, since the mass assignment scheme is conservative,

$$\sum_{i,j,k} D_{i,j,k} = m N_p = M. \quad (33)$$

#### 3.2 Interpolation of grid acceleration

The inverse operation to assign the grid defined acceleration components to each particle is performed along similar lines, where now the same linear or quadratic shape functions act as interpolating functions. To improve momentum conservation, interpolation on particle position  $\mathbf{x}_p$  is first computed on the potential derivatives components  $\tilde{\mathbf{g}} = (g_r, r g_\vartheta, r \sin \vartheta g_\varphi)$

$$\tilde{\mathbf{g}}(\mathbf{x}_p) = \sum_{i,j,k} \tilde{\mathbf{g}}_{i,j,k} [S(\xi_i - \xi_p)S(\vartheta_j - \vartheta_p)S(\varphi_k - \varphi_p)] \quad (34)$$

and then the interpolated spherical components

$$[g_r]_p = [\tilde{\mathbf{g}}_r]_p, [g_\vartheta]_p = [\tilde{\mathbf{g}}_\vartheta/r]_p, [g_\varphi]_p = [\tilde{\mathbf{g}}_\varphi/r \sin \vartheta]_p \quad (35)$$

are combined to get the corresponding Cartesian components of the particle acceleration:

$$g_x = [g_R \cos \varphi - g_\varphi \sin \varphi]_p, \quad g_y = [g_R \sin \varphi + g_\varphi \cos \varphi]_p, \quad g_z = [g_r \cos \vartheta - g_\vartheta \sin \vartheta]_p, \quad (36)$$

where

$$g_R = g_r \sin \vartheta + g_\vartheta \cos \vartheta. \quad (37)$$

### 3.3 Leap-frog particles time-stepping

N-MODY implements a standard leap-frog integration scheme of the form:

- half-step position move:

$$\mathbf{x}_p(t + \Delta t/2) = \mathbf{x}_p(t) + \frac{\Delta t}{2} \mathbf{v}_p(t), \quad (38)$$

- evaluation of the particles acceleration at the current time  $\mathbf{g}_p(t + \Delta t/2)$
- one-step velocity move:

$$\mathbf{v}_p(t + \Delta t) = \mathbf{v}_p(t) + \Delta t \mathbf{g}_p(t + \Delta t/2) \quad (39)$$

- followed by a second half-step position move:

$$\mathbf{x}_p(t + \Delta t) = \mathbf{x}_p(t + \Delta t/2) + \frac{\Delta t}{2} \mathbf{v}_p(t + \Delta t), \quad (40)$$

The implemented fourth order leap-frog scheme is obtained by cycling three times this basic scheme with time step  $\Delta t$  replaced by  $(c_1, c_2, c_1)\Delta t$  respectively, where  $c_1 = 1/(2 - 2^{1/3})$  and  $c_2 = 1 - 2c_1$ .

## 4 Compiling and running N-MODY

The N-MODY code, written in FORTRAN90, is organized in package containing a directory `n_mody` and three subdirectories: `src` (containing the `.f90` source files described in Section 4.1 and template makefiles), `run` (containing the parameter file `input.data`), and `example_init` (containing source and parameter files to generate examples of initial conditions).

In the directory `src`, the `makefile` generates the executable `rmond` and with the command `make mvexec` moves the executable in the subdirectory `run`, where data inputs and outputs are planned to reside.

A scalar version of N-MODY (not using the MPI library) can be obtained by substituting `mond_prll` with `mond_scalar` in the `makefile`.

## 4.1 Source files

N-MODY has seven .f90 source files residing in the `src` subdirectory:

1. “`ftnrmod.f90`” contains routines to execute Fast Fourier Transforms (FFTs). Includes the subroutine `four1` of Press et al. (1997). This routine can be substituted by any user-defined FFT routine.
2. “`mond_prll.f90`” contains interfaces to the MPI library for handling communications among PEs. These routines are then called in all other parts of the code without any reference to MPI definitions.
3. “`mond_scalar.f90`” substitutes `mond_prll.f90` in the scalar version of the code, not using the MPI library.
4. “`mond_lib.f90`” contains all the relevant routines implementing the MOND (or Newtonian) potential solver.
5. “`nbd_lib.f90`” contains all the relevant routines implementing particle-grid operations, i.e. mass deposition and fields interpolation.
6. “`mond_iout.f90`” contains the relevant routines for input-output operations, both on grid-based fields and particles.
7. “`nbd_mond_main.f90`” contains start-up routines and the routines driving input-output and all other procedures for time evolution. Console information on the running process is always collected in the `fort.99` file.

## 4.2 Code units

In the code we use gravitational constant  $G = 1$ . Thus, the code time, velocity and acceleration units are

$$t_u \simeq 4.7 \times 10^6 \left( \frac{l_u}{\text{kpc}} \right)^{3/2} \left( \frac{M_u}{10^{10} M_\odot} \right)^{-1/2} \text{ yr} \quad (41)$$

$$v_u \simeq 207.4 \left( \frac{M_u}{10^{10} M_\odot} \right)^{1/2} \left( \frac{l_u}{\text{kpc}} \right)^{-1/2} \text{ km s}^{-1} \quad (42)$$

$$a_u \simeq 1.39 \times 10^{-9} \left( \frac{M_u}{10^{10} M_\odot} \right) \left( \frac{l_u}{\text{kpc}} \right)^{-2} \text{ m s}^{-2}, \quad (43)$$

where  $M_u$  and  $l_u$  are the the code mass and length units. All the input/output parameters are in these units. In particular, we note that in MOND simulations  $a_0$  must be specified in units of  $a_u$ . See Nipoti et al (2007a) for a detailed discussion of the scaling of MOND simulations.

## 4.3 Input data

### 4.3.1 Simulation parameters

The already included `run/input.data` contains the following user defined control and structure parameters:

`nr`: Is the number of radial grid points  $N_r$ .

**nth**: Is the number of  $\vartheta$  grid points  $N_\vartheta$ . We recall that when running in parallel with  $N_{\text{PE}}$  processors, the condition  $N_\vartheta \geq 4N_{\text{PE}}$  must be satisfied.

**nph**: Is the number of  $\varphi$  grid points  $N_\varphi$ .

**lmax**: Is the maximum value of Legendre mode  $l$ . Typically,  $\text{lmax} = \text{nth}/2$

**model**: **model** = 0 for N-body simulations. **model** > 0 for static models [**model** = 1: Hernquist (1990) sphere; **model** = 2: Miyamoto & Nagai (1975) disk].

**mond\_ind**: Is the gravity parameter. **mond\_ind**=0 for Newtonian gravity; **mond\_ind**=1 for MOND gravity (equation 1); **mond\_ind**=2 for deep-MOND gravity (equation 4);

**spl\_order**: Order of the interpolation. **spl\_order**=1 linear; **spl\_order**=2 quadratic.

**rmap**: Determines the value of  $\alpha = 1$  or  $\alpha = 2$  in the radial grid mapping (equation 6).

**a0**: The value of the acceleration parameter  $a_0$  in code units:  $\text{a0} = a_0/a_u$ . We recall that the code uses  $G = 1$  (see Section 4.2)

**dt\_iter**: Determines the relaxation parameter  $\omega^{-1} = \text{dt\_iter}$  (see equation 17). Typical values are  $\text{dt\_iter} = 0.3 - 0.5$

**scale**: Is the value of the scale length  $L$  (equation 6) in code units. Typically,  $L$  is of the order of the half mass radius of the distribution.

**iter\_max**: Maximum number of iteration in the Newton-relaxation procedure. If convergence is not reached after **iter\_max** iterations a warning message is given in **fort.99**.

**tol**: Determines the value of the tolerance parameter  $\varepsilon = \text{tol}/10^4$  (see Section 2.3).

**new**: To read new particle data (**new**=0) or to restart an evolved simulation (**new**=1)

**id\_new**: Input/output files have names **moutXX.bin**, where **XX**=00,...,99. **id\_new** provides the **XX** integer value to identify the input file.

**nout**: The total number of consecutive outputs in a run. Output data are written in **moutXX.bin**, **poutXX.bin**, **mondXX.bin**, **diagXX.dat** files, for **XX**=**id\_new**+1,...,**id\_new**+**nouts** (see Section 4.4).

**iene**: Data related to conservation laws are computed and stored **iene**+1 times. These data are finally written in the ASCII file **diagXX.dat** (see Section 4.4).

**mrates**: If **mrates** > 0, CPU-timing of the main computational steps are documented in the **fort.99** output file at the rate of **mrates** time steps.

**tmax**: Time length of the simulation in units of the code time unit  $t_u$ . If **tmax** = 0, only one-step force evaluation with some diagnostics on the main computational steps is performed.

**dt\_min**: Minimum time step in units of the code time unit  $t_u$ . If the dynamically computed timestep is shorter than **dt\_min**, **dt\_min** is used as timestep and a warning message is given in **fort.99** standard output.

**cf1**: Dimensionless control parameter for time step evaluation. The code uses the leap-frog stability threshold  $\Delta t = \text{cf1}/\sqrt{\max|\nabla \cdot \mathbf{g}|}$ , and the maximum is evaluated over the grid points. Typical value is **cf1** = 0.3

**lp\_ord**: Order of the leapfrog scheme (**lp\_ord**=2 or **lp\_ord**=4).

### 4.3.2 Particle data

Data on position and velocity of the initial particle distribution are user provided on the input file `/run/moutXX.bin`, where `XX=00,...,99` is an integer parameter. The initial file index `XX` is read as `id_new` on `input.data` file.

The same file name, with consecutive index values (the code internal `iout` variable), is then used also for outputs. In `moutXX.bin` data are stored in binary format, and are read as follows (see `mond_iout.f90` module and `example_init` for more details)

```
read(), ipar(1:5)
read(), rpar(1:5)
do k=1, nbd_tot
read(), pd(k)%posvel(1:6)
end do
```

The parameters `ipar(1:5)` are integer numbers specifying:

- `ipar(1)`: Total number of particles `nbd_tot = Np`.
- `ipar(2)`: Model.
- `ipar(3)`: `mond_ind`. Just for reference. The value of `mond_ind` is determined in `input.data`.
- `ipar(4:5)`: Unused.

The parameters `rpar(1:5)` are (4 byte) real numbers specifying:

- `rpar(1)`: Total mass  $M$  in code units.
- `rpar(2)`: = `tnow`: the current time in units of  $t_u$ . (`tnow` = 0 for initial conditions).
- `rpar(3)`: = `tdyn`: an estimated dynamical time (in units of  $t_u$ ) related to the initial mass distribution. Just for reference, not used by the code!
- `rpar(4:5)`: Unused

The quantities `pd(k)%posvel(1:6)`, where `k=1, Np` is the particle index, are (4 byte) arrays storing

- `pd(k)%posvel(1)`: particle  $x$  position
- `pd(k)%posvel(2)`: particle  $y$  position
- `pd(k)%posvel(3)`: particle  $z$  position
- `pd(k)%posvel(4)`: particle  $x$  velocity component
- `pd(k)%posvel(5)`: particle  $y$  velocity component
- `pd(k)%posvel(6)`: particle  $z$  velocity component

An example `.f90` scalar code generating initial conditions is provided in the `example_init` subdirectory. In that directory, containing also a template `makefile`, the command `make` generates the executable `rungb`, which creates an input file `mout00.bin` according to the input parameters in `newgalaxy.data`. `rungb` generates Newtonian-equilibrium spherical  $\gamma$ -models (Dehnen 1993; Tremaine et al. 1994) with Osipkov-Merritt (Osipkov 1979; Merritt 1985) distribution function.

## 4.4 Output data

On output, the code provides the following files of data: `moutXX.bin`, `poutXX.bin`, `mondXX.bin`, `diagXX.dat`. Other run-time messages on initial condition parameters, allocated memory, fatal errors and control diagnostics are reported in the console output file `fort.99`. Here we summarize the content of the output files:

- `moutXX.bin`: in binary format, storing particle positions and velocities as explained in Section 4.3.2.
- `poutXX.bin`: in binary format, storing particle potential. `poutXX.bin` is written as follows:

```
write()ipar(1:5)
write()rpar(1:5)
do k=1,nbd_tot
write()pot(k)
end do
```

The (positive) potential of the  $k$ -th particle is stored in the (4 byte) real number `pot(k)`. The potential additive constant is such that the potential is zero at the boundary  $r_{N_r}$ .

- `mondXX.bin`: in binary format, storing grid data. `mondXX.bin` is written as follows:

```
write()nfields,nr,nth,nph,nph2,lmax,iter_max,rmap,mond_ind,spl_ord ! integer num-
bers
write()tnow,a0,scale,rh ! (4 byte) real numbers
write()rad(1:nr+1) ! (4 byte) real array
write()th(1:nth) ! (4 byte) real array
write()ph(1:nph2) ! (4 byte) real array
write()den ! (4 byte) real array den(1:nr+1,1:nth,1:nph2)
write()pot ! (4 byte) real array pot(1:nr+1,1:nth,1:nph2)
write()gr ! (4 byte) real array gr(1:nr+1,1:nth,1:nph2)
write()gth ! (4 byte) real array gth(1:nr+1,1:nth,1:nph2)
write()gph ! (4 byte) real array gph(1:nr+1,1:nth,1:nph2)
```

`nfields=5` is the number of fields stored; `nph2=nph` (if `nph > 4`) or `nph2=1` (if `nph ≤ 4`); `rh` is the half mass radius of the distribution; `rad`, `th` and `ph` are the discretized  $r$ ,  $\vartheta$ ,  $\varphi$  coordinates; `pot`, `den`, `gr`, `gth`, `gph` are the density, potential, and acceleration components.

- `diagXX.dat`: in ASCII format, for diagnostics on conservation laws. It contains values of the following quantities computed for a total number of  $k=1,2,\dots,iene+1$  steps:

**Time:** Current time

**Eint:** Absolute value of the trace of the Chandrasekhar potential energy tensor  $W = - \int \rho(\mathbf{x}) \mathbf{x} \cdot \nabla \phi d^3 \mathbf{x}$

**Epot:** Absolute value of the total potential energy  $E_{\text{pot}} = \frac{1}{2} \int \rho(\mathbf{x}) \phi d^3 \mathbf{x}$

**Ekin:** Total kinetic energy  $K$ .

**Enf:** Field energy  $E_{\text{field}} = \int \frac{a_0^2}{8\pi G} \mathcal{F} \left( \frac{\|\nabla\phi\|}{a_0} \right) d^3\mathbf{x}$ , where  $\mathcal{F}(y) \equiv 2 \int_{y_0}^y \mu(\xi)\xi d\xi$   
**D\_max:** Maximum density  
**R\_h:** Half mass radius  
**Ek\_r:** Radial kinetic energy  
**Ek\_th:** Tangential ( $\vartheta$ ) kinetic energy  
**Ek\_ph:** Tangential ( $\varphi$ ) kinetic energy  
**P\_x:**  $x$  linear momentum component  
**P\_y:**  $y$  linear momentum component  
**P\_z:**  $z$  linear momentum component  
**L\_x:**  $x$  angular momentum component  
**L\_y:**  $y$  angular momentum component  
**L\_z:**  $z$  angular momentum component  
**T\_xx:**  $xx$  inertia tensor component  
**T\_yy:**  $yy$  inertia tensor component  
**T\_zz:**  $zz$  inertia tensor component  
**T\_xy:**  $xy$  inertia tensor component  
**T\_xz:**  $xz$  inertia tensor component  
**T\_yz:**  $yz$  inertia tensor component

We recall that energy is conserved in MOND, though the total energy is infinite even for finite mass systems (Bekenstein & Milgrom 1984), so the verification of total energy conservation is not trivial. We refer to Nipoti, Londrillo & Ciotti (2007a) for a detailed discussion of energy conservation in the code. Here we just note that in MOND the quantities  $K + W$  and  $K + E_{\text{pot}}$  are *not* conserved (and are *not* the total energy). However,  $W$  is conserved in deep-MOND regime ( $W = -(2/3)\sqrt{Ga_0M^3}$  for *all* systems of finite total mass  $M$ ; see Nipoti et al. 2007a).

## References

- [1] Barnes J.E., Hut P., 1986, Nature, 324, 446
- [2] Bekenstein J. 2004, Phys. Rev. D, 70, 083509
- [3] Bekenstein J., Milgrom, M. 1984, ApJ, 286, 7
- [4] Ciotti, L., Londrillo, P., & Nipoti, C. 2006, ApJ, 640, 741
- [5] Dehnen W., 1993, MNRAS, 265, 250
- [6] Dehnen, W. 2002, Journal of Computational Physics, 179, 27
- [7] Famaey B., Gentile G., Bruneton J., Zhao H., 2007, Phys. Rev. D, 75, 063002
- [8] Hernquist L., 1990, ApJ, 356, 359
- [9] Hockney R., Eastwood J. 1988, Computer Simulation Using Particles (Bristol: Hilger)

- [10] Merritt D., 1985, *AJ*, 90, 102
- [11] Milgrom, M. 1983, *ApJ*, 270, 365
- [12] Miyamoto M., Nagai R., 1975, *PASJ*, 27, 533
- [13] Nipoti C., Londrillo P., Ciotti L., 2007a, *ApJ*, 660, 256
- [14] Nipoti C., Londrillo P., Zhao H.S., Ciotti L., 2007b, *MNRAS*, 379, 597
- [15] Nipoti C., Londrillo P., Ciotti L., 2007c, *MNRAS*, 381, L104
- [16] Osipkov L.P., 1979, *Soviet Astron. Lett.*, 5, 42
- [17] Press W.H., Teukolsky S.A., Vetterling W.T., Flannery B.P., 1997, “Numerical Recipes: The Art of Parallel Scientific Computing”, Cambridge University Press
- [18] Stoer J., Bulirsch R., 1980, *Introduction to Numerical Analysis* (New York: Springer-Verlag)
- [19] Tremaine S., Richstone D.O., Yong-Ik B., Dressler A., Faber S.M., Grillmair C., Kormendy J., Laurer T.R., 1994, *AJ*, 107, 634