Solve-MOND-MM

Release e

Claudio Llinares

Nov 05, 2020

CONTENTS:

CHAPTER

ONE

FOLDER STRUCTURE

Here is the folder structure of the code:

```
.
├── bin
├── gravity
│   ├── commons_multigrid
│   ├── mond_mass
│   └── newton
│       └── fft_solver
├── grid
├── libs
├── pm
├── tests
└── time_evolution
```

Details:

- `bin` contains the Makefile
- `gravity` contains the gravity solvers.
- `gravity/commons_multigrid` suport routines for multigrid solvers which do not depend on the model.
- `gravity/mond_mass` constains the multigrid solver for the model we are dealing with (MOND with mass terms).
- `gravity/newton` contains an FFT based solver for Poisson's equation (needed for standard simulations).
- `grid` all routines related to the grid including allocation routines and initialization of the FFTW variables.
- `libs` miscelaneous including cosmology, utility files for allocating 3D arrays and routines for reading parameter files.
- `pm` (particle mesh). Everything related to the particles at their connection to the grid. Here we allocate memory for particles and transfer information from the particles trough the grid (for calculatign density in the grid) and viceversa (for calculating force in the particles).
- `tests` has routines for testing. Here you can find the routines for the spherical test.
- `time_evolution` contains the main time loop.

CHAPTER

TWO

DATA STRUCTURE

We have the following structures defined in the code:

- `params`: defined in solve.h. It contains input parameters and some physical constants (including expansion factor).
- `parts`: defined in `pm/init_parts.h`. It contains particles' positions and velocities, which are allocated in `pm/init_parts.c`
- `grids`: defined in `grid/init_grid.h`. If contains all the fields and space for doing FFTs. Eveything is allocated in `grid/init_grid.h`. Note that there are additional grids allocated by the multigrid solver.
- `grids_gravity_mm`: defined in `gravity/mond_mass/solve_multigrid_mm.h`. It contains everything related to the MOND-MM multigrid solver (including the coarse grids, which are allocated in `gravity/mond_mass/solve_multigrid_mm.c`.

Detailed definitions follow:

type params
Structure with input parameters and fields

double **H0**

double **a**
Expansion factor.

double **box**
Mpc/h and then converted to Mpc in init_physics.

double **cte_poisson**

char [1000] file_in_parts

char [1000] file_out

int **gravity**
0 -> Newton 1 -> MOND with mass term.

int **grid**
Grid size (the code is not tested with this number being different than the number of particles).

int **grid_p**
Grid of particles (cubic root of total number of particles).

double **hsmall**

double **mm_KB**
MOND constant.

double **mm_M**
MOND mass in h/Mpc and converted to 1/Mpc.

double **mm_a0**
MOND a0 in km/sec^2 and converted to Mpc/Gyr^2.

int **n_outputs**

int **nparts**
Total number of particles.

int **nsteps**
Number of time steps.

int **nthreads**
Number of threads. (interesting only if OPENMP).

double **omegal**

double **omegam**

int **static_test**

double [1000] t_of_a

int **verbose**

int **verbose_mm**

int **verbose_mm_coarse**

double **z_end**

double **z_init**

type particles

double ***pos_1**

double ***pos_2**

double ***pos_3**

double ***vel_1**

double ***vel_2**

double ***vel_3**

type grids

double ****chi_bar_mm**

fftw_complex ***data**

double ****delta**

double ****force_1**

double ****force_2**

double ****force_3**

double ****force_chi_bar_mm_1**

double ****force_chi_bar_mm_2**

double ****force_chi_bar_mm_3**

double ****force_phi_bar_mm_1**

double ***force_phi_bar_mm_2**

double ***force_phi_bar_mm_3**

fftw_plan **p_backward**

fftw_plan **p_forward**

double ***phi**

double ***phi_bar_mm**

double ***source**

type grids_gravity_mm

Main data structure of the multigrid solver.

Comments:

- Note that here we have a copy of the main parameters of the code. That is to avoid having to accessing too many structures in some of the loops in the solver.

double **KB**
MOND parameter KB

double **a2M2**
MOND parameter M^2 (multipled by a^2).

double **aa0**
MOND parameter a0

double ***chi**[20]
Solution for the MOND equation ( ).

double **cte_poisson_over_a**

double **fourh**[20]
This one is optimization

int **g**
Grid in which we are working. 0 = finest (so what people call V cycle, is actually a Lambda).

double **h**[20]
Spacing of each grid.

int **n**[20]
Size of each grid.

int **num_grids**
Number of grids according to the finnest.

struct *params* ***par**
Pointer to the parameter structure of the main code (see solve.h for details).

struct params

double ***phi**[20]
Solution for the Newtonia potential .

double ***res_chi**[20]
Residual of the equation for chi.

double ***res_phi**[20]
Residual of the equation for phi.

double ***rho[20]
Density (or overdensity) (see how it is used in gs routine)).

double *[20] source_chi**

double ***source_phi[20]
Source of the equation for coarse grids in the FAS algorithm

double *[20] temp2_chi**

double *[20] temp2_phi**

double *[20] temp_chi**

double *[20] temp_phi**

double ***trunc_chi[20]
truncation error in all cells all grids.

double ***trunc_phi[20]
truncation error in all cells all grids.

int **verbose**

int **verbose_coarse**

CHAPTER

THREE

FILES

3.1 gravity

3.1.1 mond_mass

cycles_mm.c

Here is the V cycle of multigrid iterations.

Since our first grid is number zero, we go to coarse grids by going up, so the cycle is inverted.

Converge criteria:

- See box in page 157 of Trottenberg, Oosterlee, Schuller, "Multigrid" for details.
- The convergence criteria are based on the mean value of the residual
- What we mean by "mean value" is defined in `grid/get_mean_max_two_fields`.
- **We have three criteria:**
  - Absolute value of the residual (which has units, so it is not good).
  - Ratio of residual of latest step and first step. This is the one implemented in Ramses. It depends on how good or bad the initial guess is, so it makes no sence.
  - Comparison with truncation error.

int **cycle_v_mm**(**struct** *grids_gravity_mm* *gg, double *a*)

void **cycle_v_mm_recursive**(**struct** *grids_gravity_mm* *gg, double *a*)

void **do_one_v_cycle**(**struct** *grids_gravity_mm* *gg, double *a*)

void **verbose_fine**(**struct** *grids_gravity_mm* *gg, double *a*, int *write_pot*)

differential_operators_mm.c

Definition of the differential operator for both equations (Poisson like and MOND like).

void **L_mm**(double ***phi*, double ***chi*, double *aa0*, double *a2M2*, double *h*, double *fourh*, int *i*, int *j*, int *k*, int *ip1*, int *im1*, int *jp1*, int *jm1*, int *kp1*, int *km1*, double **operator_phi*, double **operator_chi*)
Evaluates differential operators on a node

void **evaluate_differential_operator_mm**(**struct** *grids_gravity_mm* *gg, int *grid*, double ***phi*, double ***chi*, double ***operator_phi*, double ***operator_chi*)
Evaluates differential operators on the grid given by the second argument. Note that the grid does not have to

necessarily be the grid gg->g. we do not store the result in the correponding variables in gg (instead we store it in the arguments res_phi, res_chi). This is in case you wat to store it somewhere else.

double **mu** (double *x*)

go_up_and_down_mm.c

Two routines for going to coarse (up) and fine (down) grids.

void **go_down_mm** (**struct** *grids_gravity_mm* **gg*, double *a*)

Moves everything to the next finer grid

Comments:

- gg->g is the coarse.
- gg->g-1 fine

void **go_up_mm** (**struct** *params* **par*, **struct** *grids_gravity_mm* **gg*, double *a*)

Moves everything to the next coarser grid.

- Calculates source in coarse grid.
- Transfer density and solution.

See box in Trottenberg, pag 157.

The source is calculated with: S = operator(R(phi), R(rho), R(S)) - R(residual(phi, rho, S))

For first term:

- Restrict three fields (including the source in this level, which is zero in the fine grid).
- Evaluate operator.

For the second term:

- Calculate the residual in the level where we are.
- Restrict that residual.

gs_residual_truncation_error_mm.c

Routines for doing one Gauss-Seidel sweep, calculating residual and truncation error (all in one grid).

Note: This is basically the core of the solver where the equations are included. See also `differential_operators_mm.c`.

void **gs_mm** (**struct** *grids_gravity_mm* **gg*)

void **gs_node_mm** (double *rho*, double *source_phi*, double *source_chi*, double ***phi*, double ***chi*, double *cte_poisson_over_a*, double *aa0*, double *a2M2*, double *h*, double *fourh*, int *i*, int *j*, int *k*, int *ip1*, int *im1*, int *jp1*, int *jm1*, int *kp1*, int *km1*)

This routine does Gauss-Seidel (GS) iteration in one node.

Comments:

- There are a lot of arguments to prevent the routine from accessing the pointer to `grids_gravity_mm` and make it more efficient.
- Whatever you put here is crucial for the performance of the code. In particular, "if" statements can kill the performance.

void **residual_mm**(**struct** *grids_gravity_mm* *gg, int *grid*, double ***res_phi*, double ***res_chi*)
Calculates residual on the grid "grid". Comments:

- Note that the grid does not have to necessarily be the grid gg->g.
- We do not store the result by default in the correponding variables in gg (instead we store it in the arguments res_phi, res_chi). This is in case you wat to store it somewhere else.

sx_first_mm

sy_first_mm

sz_first_mm

void **trunc_err_mm**(**struct** *grids_gravity_mm* *gg, int *grid*, double ***trunc_phi*, double ***trunc_chi*)
Estimates truncation error (i.e. error associated to the discretizatio of the differential equation).

Comments:

- Note that the grid does not have to necessarily be the grid gg->g.
- The estimator is: tau = P[D(R(phi)) - R(D(phi))]

initial_guess_mm.c

Initial guess for multigrid solver.

Comments:

- Use this if you do not have a better solution (such as the solution is previous time step).
- This must be called before you call the multigrid solver.
- For the MOND case, zero is not good enough (that's why we put a random number).
- A better initial guesses can be otained by what we have in `solve_spherical_mond`.

void **initial_guess_mm**(**struct** *params* *par, double ***chi_bar*, double ***phi_bar*)

solve_multigrid_mm.c

Multigrid solver for the non-linear Bekenstein-Milgrom MM equation in a 3D uniform grid given a source. See Brent 1977, Trottenberg 2000 or Wesseling 1992.

Comments:

- User should give and initial condition in phi for the iterations (zero is wrong).
- Grids start from zero, meaning that the V cycle looks more like an inverted V (i.e. when we go up, we go to coarse grids).

void **allocate_grids_mm**(**struct** *grids_gravity_mm* *gg, double ***dens*, double ***phi*, double ***chi*)
Allocate additional coarse grids for the multigrid solver. For the fine grid, we do not make a copy, but just keep a pointer to the variables in the main structure par.

void **deallocate_grids_mm**(**struct** *grids_gravity_mm* *gg, double ***dens*, double ***phi*, double ***chi*)
Clean up the coarse grids.

Comments:

- We do not clean up the fine grid because we only have a pointer to the main variables in the main code (in the structure `grid`).

- We could call this after obtaining a solution. Which means that we have to allocate everything again in the following time step. I am not sure if that is the best thing to do, but in any case, the overhead is negligible compare with everything else.

int **solve_multigrid_mm**(**struct** *params* *par, double ***dens, double ***phi, double ***chi, double *box, double *grid, double *a*)
Main routine of the multigrid solver for MOND-MM potentials

solve_multigrid_mm.h

GO_UP_FIRST
default=1. If 0 then solver does not go up in the first multigrid level (i.e. one grid scheme).

GO_UP_SECOND

MAX_GRIDS
Number of pointers to grids (not necessarily allocated). (i.e. not important unless you want to have more than 20 levels).

MAX_ITERATION
Maximun number of complete V cycles before accepting that the solver will not converge.

MINUS
For evaluating the MOND operator.

NGS
Number of Gauss-Seidel iterations per level.

PLUS
For evaluating the MOND operator.

X
For evaluating the MOND operator.

Y
For evaluating the MOND operator.

Z
For evaluating the MOND operator.

struct grids_gravity_mm
Main data structure of the multigrid solver.

Comments:

- Note that here we have a copy of the main parameters of the code. That is to avoid having to accessing too many structures in some of the loops in the solver.

double **KB**
MOND parameter KB

double **a2M2**
MOND parameter M^2 (multipled by a^2).

double **aa0**
MOND parameter a0

double ***chi**[20]
Solution for the MOND equation ( ).

double **cte_poisson_over_a**

double **fourh**[20]
This one is optimization

int **g**
Grid in which we are working. 0 = finest (so what people call V cycle, is actually a Lambda).

double **h**[20]
Spacing of each grid.

int **n**[20]
Size of each grid.

int **num_grids**
Number of grids according to the finnest.

struct *params* ***par**
Pointer to the parameter structure of the main code (see solve.h for details).

struct params

double ***phi**[20]
Solution for the Newtonia potential .

double ***res_chi**[20]
Residual of the equation for chi.

double ***res_phi**[20]
Residual of the equation for phi.

double ***rho**[20]
Density (or overdensity) (see how it is used in gs routine)).

double *[20] source_chi**

double ***source_phi**[20]
Source of the equation for coarse grids in the FAS algorithm

double *[20] temp2_chi**

double *[20] temp2_phi**

double *[20] temp_chi**

double *[20] temp_phi**

double ***trunc_chi**[20]
truncation error in all cells all grids.

double ***trunc_phi**[20]
truncation error in all cells all grids.

int **verbose**

int **verbose_coarse**

extern int **solve_multigrid_mm**(**struct** *params* **par*, double ***dens*, double ***chi*, double ***phi*, double *box*, double *grid*, double *a*)

solve_spherical_mond.c

Routines for solving for the spherical mond potential given by the standard poisson's equation. The calculation of the source and everything is made here. - See appendix in Llinares' thesis 2010.

pow2 (*x*)

pow3 (*x*)

pow4 (*x*)

pow5 (*x*)

pow6 (*x*)

int **solve_spherical_mond**(double ***phi*, double *a*, fftw_complex **data*, fftw_plan *p_forward*, fftw_plan *p_backward*, double *power_a0*)

This routine comes from the thesis. It has to be merged into the new code. It is not included in the Makefile.

The routine solves an equation for the spherical mond potential which can be obtained by calculating the derivative of the integral (yes, that's what I wrote) of the MOND equation. The method is:

- get newtonian forces.
- get spherical mond forces.
- take divergence.
- solve poisson's equation with that source.

Parameters

- **phi = newtonian potential.** (–) –
- **a = expansion factor.** (–) –

Returns

- phi = spherical mondian potential.

Comments:

- No units (they come with whatever you put as input).
- THIS ROUTINE REWITES PHI.

3.1.2 newton

fft_solver

solve_poisson_fftw.c

Solve Poisson's equation in a 3D uniform grid given a source.

Comments:

- No 4 pi G involved (it should come with the source if there is any).
- You can get the 4 pi G for standard gravity by calling `calc_newt_source`.
- No units (again, they should come with the density).

void **anti_transforms**(double ***gr*, double ***gi*, double *box*, int *grid*, fftw_complex **data*, fftw_plan *p_forward*)

index (*i*, *j*, *k*)

int **solve_poisson_fftw** (**struct** *params* *par, **struct** *grids* *grid, **struct** *units* *unit)

void **transforms** (double ***gr, double ***gi, double *box, int *grid, fftw_complex *data, fftw_plan *p_backward*)

calc_newt_source.c

Calculates the source of standard Poisson's equation. Comments:

- This routine can probably be simplified, but having this separate from the solver allow us to used the FFT solver for many things (for instance, for calculating initial conditions for the MOND equation.

void **calc_newt_source** (**struct** *params* *par, **struct** *grids* *grid, **struct** *units* *unit, double *a*)

3.1.3 prolonge.c

Prologation routines which interpolate a field from a coarse grid back to a fine grid.

Comments:

- See Trottenberg, Oosterlee, Schuller, "Multigrid" for details.
- These routines are needed by the multigrid solver to move between grids.
- There are several routines that use different interpolation order. Higher order is not necessarily better.
- The only public routine is `prolonge`. The other routines are accessed through it. Uncomment the call you want to use.

pow2 (*x*)

void **prolonge** (double ***old, double ***new, int *n_old*, double *h_old*)

void **prolonge_first_order** (double ***old, double ***new, int *n_old*, int *n_new*, double *h_old*, double *h_new*)

void **prolonge_second_order** (double ***old, double ***new, int *n_old*, int *n_new*, double *h_old*, double *h_new*)

void **prolonge_trilinear** (double ***old, double ***new, int *n_old*, int *n_new*, double *h_old*, double *h_new*)

3.1.4 restrict.c

Restriction routines which interpolate a field from a fina grid to a coarse grid (with half the number of points per dimension).

Comments:

- See Trottenberg, Oosterlee, Schuller, "Multigrid" for details.
- These routines are needed by the multigrid solver to move between grids.
- There are several routines that use different interpolation order. Higher order is not necessarily better.
- The only public routine is `restrict_grid`. The other routines are accessed through it. Uncomment the call you want to use.

void **restrict_grid** (double ***old, double ***new, long *n_old*)

void **restrict_grid_cospatial** (double ***old, double ***new, long *n_old*)

void **restrict_grid_original** (double ***old*, double ***new*, long *n_old*)

3.1.5 get_forces.c

Calculates force in the grid by deriving a potential.

void **forces** (**struct** *params* *par, double ***phi, double ***dphi_dx, double ***dphi_dy, double ***dphi_dz)

Derives a potential.

Parameters

- **par = pointer to params structure** (–) –
- **phi = pointer to the potential.** (–) –
- **dphi_dx... = derivatives.** (–) –

pow2 (*x*)

3.1.6 get_gravity.c

Calculates forces on the grid from scratch (including calculation of the density and potential).

Comments:

- The result is returned in the corresponding variable in the grid structure.
- The density and potential will also end up in the coresponding variables in the structure grid.

void **get_gravity** (**struct** *params* *par, **struct** *grids* *grid, **struct** *particles* *part, **struct** *units* *unit, double *t*)

3.1.7 get_lap.c

void **calc_lap** (**struct** *params* *par, double ***phi, double ***lap_phi*)

3.2 grid

3.2.1 add_two_fields.c

void **add_two_fields** (double ***field1*, double ***field2*, int *n*, double ***result*)

3.2.2 clean_up_grid.c

void **clean_up_grid** (**struct** *params* *par, **struct** *grids* *grid)

3.2.3 get_mean_max_two_fields.c

void **get_mean_max_two_fields**(double ***field1*, double ***field2*, int *n*, double **mean*, double **maximum*)

new_max(*x*, *y*)

3.2.4 init_fftw.c

void **init_fftw**(**struct** *params* **par*, **struct** *grids* **grid*)

3.2.5 init_grid.c

void **init_grid**(**struct** *params* **par*, **struct** *grids* **grid*)

3.2.6 init_grid.h

struct grids

double ***chi_bar_mm**

fftw_complex ***data**

double ***delta**

double ***force_1**

double ***force_2**

double ***force_3**

double ***force_chi_bar_mm_1**

double ***force_chi_bar_mm_2**

double ***force_chi_bar_mm_3**

double ***force_phi_bar_mm_1**

double ***force_phi_bar_mm_2**

double ***force_phi_bar_mm_3**

fftw_plan **p_backward**

fftw_plan **p_forward**

double ***phi**

double ***phi_bar_mm**

double ***source**

extern void **init_grid**(**struct** *params* **par*, **struct** *grids* **grid*)

3.2.7 write_phi.c

void **write_phi**(**struct** *params* *par, **struct** *grids* *grid)

3.3 libs

3.3.1 allocator.c

void **alloc_char_2**(char ***a, int *d1*, int *d2*)

void **alloc_char_3**(char ****a, int *d1*, int *d2*, int *d3*)

void **alloc_char_4**(char *****a, int *d1*, int *d2*, int *d3*, int *d4*)

void **alloc_double_2**(double ***a, int *d1*, int *d2*)

void **alloc_double_3**(double ****a, int *d1*, int *d2*, int *d3*)

void **alloc_double_4**(double *****a, int *d1*, int *d2*, int *d3*, int *d4*)

void **alloc_double_5**(double ******a, int *d1*, int *d2*, int *d3*, int *d4*, int *d5*)

void **alloc_float_2**(float ***a, int *d1*, int *d2*)

void **alloc_float_3**(float ****a, int *d1*, int *d2*, int *d3*)

void **alloc_float_4**(float *****a, int *d1*, int *d2*, int *d3*, int *d4*)

void **alloc_int_2**(int ***a, int *d1*, int *d2*)

void **alloc_int_3**(int ****a, int *d1*, int *d2*, int *d3*)

void **alloc_int_4**(int *****a, int *d1*, int *d2*, int *d3*, int *d4*)

void **alloc_long_2**(long ***a, int *d1*, int *d2*)

void **alloc_long_3**(long ****a, int *d1*, int *d2*, int *d3*)

void **alloc_long_4**(long *****a, int *d1*, int *d2*, int *d3*, int *d4*)

3.3.2 a_of_time.c

int **a_dot_supercom_for_rk**(double *t*, **const** double *y*[], double *f*[], void **params*)

double **a_of_supercom_time**(**struct** *params* *par, **struct** *units* *unit, double *tau*)

int **jac_a_dot_supercom_for_rk**(double *t*, **const** double *y*[], double **dfdy*, double *dfdt*[], void **params*)

pow2(*x*)

pow3(*x*)

pow4(*x*)

void **test_a_of_time**(**struct** *params* *par, **struct** *units* *unit)

double **time_of_a_supercom**(**struct** *params* *par, **struct** *units* *unit, double *a_init*)

3.3.3 cosmology.c

double **A**(**struct** *params* *par, **struct** *units* *unit, double *a*)

double **F**(**struct** *params* *par, **struct** *units* *unit, double *a*)

double **a_dot**(**struct** *params* *par, **struct** *units* *unit, double *a*)

double **a_of_t**(**struct** *params* *par, double *t*)

double **hubble**(**struct** *params* *par, **struct** *units* *unit, double *a*)

double **hubble2**(**struct** *params* *par, **struct** *units* *unit, double *a*)

double **hubble_dot**(**struct** *params* *par, **struct** *units* *unit, double *a*)

double **integrand_time**(double *a*, void *parameters)

double **integrate_time_of_a**(**struct** *params* *par, double *a*)

struct params_int

struct *params* ***par**

void **tabulate_t_of_a**(**struct** *params* *par)

3.3.4 ran3.c

FAC

MBIG

MSEED

MZ

float **ran3**(long **idum*)

3.3.5 read_params.c

/// file read_params.c

void **check_keyword**(char *keywords*[50][1000], void **values*[50], char *type*[50][10], int **found*, char **key*, char **val*)

void **delete_spaces**(char **str*)

void **init_keywords**(**struct** *params* *par, char *keywords*[50][1000], void **values*[50], char *type*[50][10], int **found*)

void **read_params**(**struct** *params* *par)

3.3.6 releaser.c

void **free_char_2** (char **a)

void **free_char_3** (char ***a)

void **free_char_4** (char ****a)

void **free_double_2** (double **a)

void **free_double_3** (double ***a)

void **free_double_4** (double ****a)

void **free_float_2** (float **a)

void **free_float_3** (float ***a)

void **free_float_4** (float ****a)

void **free_int_2** (int **a)

void **free_int_3** (int ***a)

void **free_int_4** (int ****a)

void **free_long_2** (long **a)

void **free_long_3** (long ***a)

void **free_long_4** (long ****a)

3.3.7 utils.c

void **change_system** (double *vec*[3][3], double *x, double *y, double *z)

double **dist** (double x, double y, double z)

double **euclides** (double x, double y, double z)

double **euclides2** (double x, double y, double z)

FILE ***fopen_check** (char **file*, char **type*)

void **index_1d_to_3d** (double q, int *i, int *j, int *k, int n)

double **modulus** (double x, double y)

void **sort_3_values** (double *a)

3.4 pm

3.4.1 calc_delta_cic.c

Calculates overdensity on the grid from the particle's positions using cloud in cell (CIC) smoothing. Comments:

- Details in the CIC scheme can be found in Hockney R. W., Eastwood J. W., 1988, Computer simulation using particles.

void **calc_delta_cic**(**struct** *params* *par, **struct** *units* *unit, double *pos_1, double *pos_2, double *pos_3, double ***delta*)

Calculates overdensity on the grid from the particle's positions using cloud in cell (CIC) smoothing..

Note: This is a note for testign Sphinx.

Parameters

- **par** – par structure.
- **unit** – units structure.
- **pos_1** – X component of position vector.
- **pos_2** – X component of position vector.
- **pos_3** – X component of position vector.
- **delta** – to return overdensity.

Returns Overdensity in the parameter delta, which should be allocated.

3.4.2 init_parts.c

Here we allocate memory for the particles and read initial conditions.

void **init_parts**(**struct** *params* *par, **struct** *particles* *part, **struct** *units* *unit)

3.4.3 init_parts.h

extern void **init_parts**(**struct** *params* *par, **struct** *particles* *part, **struct** *units* *unit)

struct particles

double ***pos_1**

double ***pos_2**

double ***pos_3**

double ***vel_1**

double ***vel_2**

double ***vel_3**

3.4.4 input_particles.c

Here we read initial conditions for particles. Comments:

- Units in the input file are expected to be Mpc/h for the positions and km/sec for the velocities.
- Units are converted later to Mpc and Mpc/Gyr.

void **read_particles**(**struct** *params* *par, **struct** *units* *unit, double *pos_1, double *pos_2, double *pos_3, double *vel_1, double *vel_2, double *vel_3, char *file_name)

3.4.5 interp_force_cic.c

Interpolates a force vector from the grid to the particles usign CIC scheme. Comments:

- Details in the CIC scheme can be found in Hockney R. W., Eastwood J. W., 1988, Computer simulation using particles.

void **interp_force_cic**(**struct** *params* *par, **struct** *grids* *grid, **struct** *particles* *part, double ***force_1, double ***force_2, double ***force_3, double *move_part*)

3.4.6 write_particles.c

void **write_particles**(**struct** *params* *par, **struct** *units* *unit, double *pos_1, double *pos_2, double *pos_3, double *vel_1, double *vel_2, double *vel_3, int *num*)

3.5 tests

3.5.1 static_test.c

This is a static test. If the corresponding flag is on in the parameter file, then the code will read initial conditions, calculate all the relevant fields, write them in a file and stops execution. Comments:

- There is a comparison between solutions obtained here and analytic solutions in the directory tests.

void **static_test**(**struct** *params* *par, **struct** *grids* *grid, **struct** *particles* *part, **struct** *units* *unit*)

3.6 time_evolution

3.6.1 time_evolution_standard_variable.c

Here is the main time loop.

void **leap_frog_variable**(**struct** *params* *par, **struct** *grids* *grid, **struct** *particles* *part, **struct** *units* *unit*)

3.7 init_physics.c

void **init_physics**(**struct** *params* *par, **struct** *units* *unit*)

Initializes a lot of physical constants including the factor in front of the density in the Poisson's equation (which somehow contains the effective gravitational constant). So if you want to change G, this is probably the place to do it. Although, keep in mind that additional changes are made later on to include the dependence with expansion factor.

Parameters

- **par** – parameter structure.
- **unit** – units structure.

Returns Changes are included in the two structures that you input.

3.8 init_physics.h

extern void **init_physics**(**struct** *params* *par, **struct** *units* *unit)

3.9 solve.c

Here is the main function.

Units:

input:

- [x] = Mpc/h
- [v] = km sec^-1

internal:

- [x] = Mpc
- [v] = Mpc/Gyr
- [source] = 1/Gyr^2
- [phi] = Mpc^2/Gyr^2 (it takes units from distances)

Comments:

- The program can NOT restart.

int **main**(int *argc*, char **argv*[])
Main routine. It initializes everything (physics, fine grid and particles) and gives control to the routine that does the time evolution.

3.10 solve.h

struct params
Structure with input parameters and fields

double **H0**

double **a**
Expansion factor.

double **box**
Mpc/h and then converted to Mpc in init_physics.

double **cte_poisson**

char [1000] file_in_parts

char [1000] file_out

int **gravity**
0 -> Newton 1 -> MOND with mass term.

int **grid**
Grid size (the code is not tested with this number being different than the number of particles).

int **grid_p**
Grid of particles (cubic root of total number of particles).

double **hsmall**

double **mm_KB**
MOND constant.

double **mm_M**
MOND mass in h/Mpc and converted to 1/Mpc.

double **mm_a0**
MOND a0 in km/sec^2 and converted to Mpc/Gyr^2.

int **n_outputs**

int **nparts**
Total number of particles.

int **nsteps**
Number of time steps.

int **nthreads**
Number of threads. (interesting only if OPENMP).

double **omegal**

double **omegam**

int **static_test**

double [1000] t_of_a

int **verbose**

int **verbose_mm**

int **verbose_mm_coarse**

double **z_end**

double **z_init**

pow2 (*x*)

pow3 (*x*)

pow4 (*x*)

pow5 (*x*)

pow6 (*x*)

pow7 (*x*)

struct units

double **H0**

double **clight**

double **cm2mpc**

double **cte_poisson**

double **gyr2sec**

double **kg2msun**

double **km2mpc**

double **m2mpc**

double **mpc2km**

double **mpc2km_over_gyr2sec**

double **mpc2m**

double **pi**

double **sec2gyr**

double **twopi**

CHAPTER

FOUR

INDICES AND TABLES

- genindex

INDEX

H

I

J

L

M

N

P

R

S

T

U

V

W

X

Y

Z