NYX User's Guide

June 20, 2017

Contents

Chapter 1

Getting Started

1.1 Downloading the Code

Nyx is built on top of the amrex framework. In order to run Nyx you must download two separate git modules.

First, make sure that `git` is installed on your machine—we recommend version 1.7.x or higher.

1. Clone/fork the amrex repository – you will need to get this from the private LBL bitbucket repository by emailing ASAlmgren@lbl.gov until the public licensing is complete.

   You will want to periodically update amrex by typing

   ```
   git pull
   ```

   in the `amrex/` directory.

   Note: active development is done on the `development` branch in each repo, and merged into the `master` branch periodically. If you wish to use the Nyx `development` branch, then you should also switch to the `development` branch for amrex.

2. Set the environment variable, `AMREX_HOME`, on your machine to point to the path name where you have put amrex. You can add this to your `.bashrc` as:

   `export AMREX_HOME=` *`/path/to/amrex/`*

   where you replace *`/path/to/amrex/`* will the full path to the `amrex/` directory.

3. Clone/fork the Nyx repository using either HTTP access:

   ```
   git clone https://github.com/BoxLib-Codes/Nyx.git
   ```

   or SSH access if you have it enabled:

   ```
   git clone ssh://git@github.com:/BoxLib-Codes/Nyx.git
   ```

   As with amrex, development on Nyx is done in the `development` branch, so you should work there if you want the latest source.

1.2 Building the Code

1. From the directory in which you checked out Nyx, type

   cd `Nyx/Exec/LyA`

   This will put you into a directory in which you can run a small version of the Santa Barbara test problem.

2. In `Nyx/Exec/LyA`, edit the GNUmakefile, and set

   COMP = your favorite compiler (e.g, gnu, Intel)

   DEBUG = FALSE

   We like COMP = gnu.

3. Now type "make". The resulting executable will look something like "Nyx3d.Linux.gnu.ex", which means this is a 3-d version of the code, made on a Linux machine, with COMP = gnu.

1.3 Running the Code

1. Type "Nyx3d.Linux.gnu.ex inputs.32"

2. You will notice that running the code generates directories that look like plt00000, plt00020, etc, and chk00000, chk00020, etc. These are "plotfiles" and "checkpoint" files. The plotfiles are used for visualization, the checkpoint files are used for restarting the code.

1.4 Visualization of the Results

There are several options for visualizing the data. The popular VisIt package supports the amrex file format natively, as does the yt python package. The standard tool used within the BoxLib-community is Amrvis, which we demonstrate here.

1. Get Amrvis:

   ```
   git clone https://ccse.lbl.gov/pub/Downloads/Amrvis.git
   ```

   Then cd into `Amrvis/`, edit the `GNUmakefile` there to set `DIM = 2`, and again set `COMP` and `FCOMP` to compilers that you have. Leave `DEBUG = FALSE`.

   Type `make` to build, resulting in an executable that looks like `amrvis2d...ex`.

   If you want to build amrvis with `DIM = 3`, you must first download and build `volpack`:

   ```
   git clone https://ccse.lbl.gov/pub/Downloads/volpack.git
   ```

   Then cd into `volpack/` and type `make`.

   Note: Amrvis requires the OSF/Motif libraries and headers. If you don't have these you will need to install the development version of motif through your package manager. `lesstif`

gives some functionality and will allow you to build the amrvis executable, but Amrvis may not run properly.

On most Linux distributions, the motif library is provided by the `openmotif` package, and its header files (like `Xm.h`) are provided by `openmotif-devel`. If those packages are not installed, then use the package management tool to install them, which varies from distribution to distribution, but is straightforward.

You may then want to create an alias to `amrvis2d`, for example

```
alias amrvis2d /tmp/Amrvis/amrvis2d...ex
```

where `/tmp/Amrvis/amrvis2d...ex` is the full path and name of the Amrvis executable.

2. Configure Amrvis:

   Copy the `amrvis.defaults` file to your home directory (you can rename it to `.amrvis.defaults` if you wish). Then edit the file, and change the `palette` line to point to the full path/filename of the `Palette` file that comes with Amrvis.

3. Visualize:

   Return to the `Nyx/Exec/LyA` directory. You should have a number of output files, including some in the form `*pltXXXXX`, where `XXXXX` is a number corresponding to the timestep the file was output. `amrvis2d` *`filename`* to see a single plotfile, or `amrvis2d -a *plt*`, which will animate the sequence of plotfiles.

   Try playing around with this—you can change which variable you are looking at, select a region and click "Dataset" (under View) in order to look at the actual numbers, etc. You can also export the pictures in several different formats under "File/Export".

   Please know that we do have a number of conversion routines to other formats (such as matlab), but it is hard to describe them all. If you would like to display the data in another format, please let us know (again, `asalmgren@lbl.gov`) and we will point you to whatever we have that can help.

Chapter 2

Inputs Files

The Nyx executable reads run-time information from an "inputs" file (which you put on the command line) and from a "probin" file, the name of which is usually defined in the inputs file, but which defaults to "probin". To set the "probin" file name in the inputs file:

amr.probin_file = my_special_probin

for example, has the Fortran code read a file called "my_special_probin"

2.1 Problem Geometry

2.1.1 List of Parameters

Parameter	Definition	Acceptable Values	Default
geometry.prob_lo	physical location of low corner of the domain	Real	must be set
geometry.prob_hi	physical location of high corner of the domain	Real	must be set
geometry.coord_sys	coordinate system	0 = Cartesian, 1 = r-z, 2 = spherical	must be set
geometry.is_periodic	is the domain periodic in this direction	0 if false, 1 if true	0 0 0

2.1.2 Examples of Usage

- **geometry.prob_lo** = 0 0 0
  defines the low corner of the domain at (0,0,0) in physical space.

- **geometry.prob_hi** = 1.e8 2.e8 2.e8
  defines the high corner of the domain at (1.e8,2.e8,2.e8) in physical space.

- **geometry.coord_sys** = 0
  defines the coordinate system as Cartesian

- **geometry.is_periodic** = 0 1 0
  says the domain is periodic in the y-direction only.

2.2 Domain Boundary Conditions

2.2.1 List of Parameters

Parameter	Definition	Acceptable Values	Default
nyx.lo_bc	boundary type of each low face	0,1,2,3,4,5	must be set
nyx.hi_bc	boundary type of each high face	0,1,2,3,4,5	must be set

2.2.2 Notes

Boundary types are:

0 – Interior / Periodic
1 – Inflow
2 – Outflow
3 – Symmetry
4 – Slip Wall
5 – No Slip Wall

Note – **nyx.lo_bc** and **nyx.hi_bc** must be consistent with **geometry.is_periodic** – if the domain is periodic in a particular direction then the low and high bc's must be set to 0 for that direction.

2.2.3 Examples of Usage

- **nyx.lo_bc** = 1 4 0
- **nyx.hi_bc** = 2 4 0
- **geometry.is_periodic** = 0 0 1

would define a problem with inflow (1) in the low-x direction, outflow(2) in the high-x direction, slip wall (4) on the low and high y-faces, and periodic in the z-direction.

2.3 Resolution

2.3.1 List of Parameters

Parameter	Definition	Acceptable Values	Default
amr.n_cell	number of cells in each direction at the coarsest level	Integer > 0	must be set
amr.max_level	number of levels of refinement above the coarsest level	Integer ≥ 0	must be set
amr.ref_ratio	ratio of coarse to fine grid spacing between subsequent levels	2 or 4	must be set
amr.regrid_int	how often to regrid	Integer > 0	must be set
amr.regrid_on_restart	should we regrid immediately after restarting	0 or 1	0

Note: if **amr.max_level** = 0 then you do not need to set **amr.ref_ratio** or **amr.regrid_int**.

2.3.2 Examples of Usage

- **amr.n_cell** = 32 64 64

  would define the domain to have 32 cells in the x-direction, 64 cells in the y-direction, and 64 cells in the z-direction *at the coarsest level*. (If this line appears in a 2D inputs file then the final number will be ignored.)

- **amr.max_level** = 2
  would allow a maximum of 2 refined levels in addition to the coarse level. Note that these additional levels will only be created only if the tagging criteria are such that cells are flagged as needing refinement. The number of refined levels in a calculation must be $\leq$ **amr.max_level**, but can change in time and need not always be equal to **amr.max_level**.

- **amr.ref_ratio** = 2 4
  would set factor 2 refinement between levels 0 and 1, and factor 4 refinement between levels 1 and 2. Note that you must have at least **amr.max_level** values of **amr.ref_ratio** (Additional values may appear in that line and they will be ignored).

- **amr.regrid_int** = 2 2
  tells the code to regrid every 2 steps. Thus in this example, new level 1 grids will be created every 2 level 0 time steps, and new level 2 grids will be created every 2 level 1 time steps.

2.4 Tagging

2.4.1 List of Parameters

Parameter	Definition	Acceptable Values	Default
nyx.allow_untagging	are cells allowed to be "untagged"	0 or 1	0

2.4.2 Notes

- Typically cells at a given level can be tagged as needing refinement by any of a number of criteria, but cannot be "untagged", i.e. once tagged no other criteria can untag them. If we set **nyx.allow_untagging** = 1 then the user is allowed to "untag" cells in the Fortran tagging routines

2.5 Regridding

2.5.1 Overview

The details of the regridding strategy are described in a later section; here we cover how the input parameters can control the gridding.

As described later, the user defines Fortran subroutines which tag individual cells at a given level if they need refinement. This list of tagged cells is sent to a grid generation routine, which uses the Berger-Rigoutsis algorithm to create rectangular grids that contain the tagged cells.

2.5.2 List of Parameters

Parameter	Definition	Acceptable Values	Default
amr.regrid_file	name of file from which to read the grids	text	no file
amr.grid_eff	grid efficiency at coarse level at which grids are created	Real > 0 and < 1	0.7
amr.n_error_buf	radius of additional tagging around already tagged cells	Integer ≥ 0	1
amr.max_grid_size	maximum size of a grid in any direction	Integer > 0	128 in 2D, 32 in 3D
amr.blocking_factor	grid size must be a multiple of this	Integer > 0	2
amr.refine_grid_layout	refine grids more if # of processors $>$ # of grids	0 if false, 1 if true	1

2.5.3 Notes

- **amr.n_error_buf**, **amr.max_grid_size** and **amr.blocking_factor** can be read in as a single value which is assigned to every level, or as multiple values, one for each level
- **amr.max_grid_size** at every level must be even
- **amr.blocking_factor** at every level must be a power of 2
- the domain size must be a multiple of **amr.blocking_factor** at level 0
- **amr.max_grid_size** must be a multiple of **amr.blocking_factor** at every level

2.5.4 Examples of Usage

- **amr.regrid_file** = fixed_grids
  In this case the list of grids at each fine level are contained in the file, *fixed_grids*, which will be read during the gridding procedure. These grids must not violate the **amr.max_grid_size** criterion. The rest of the gridding procedure described below will not occur if **amr.regrid_file** is set.
- **amr.grid_eff** = 0.9
  During the grid creation process, at least 90% of the cells in each grid at the level at which the grid creation occurs must be tagged cells. Note that this is applied at the coarsened level at which the grids are actually made, and before **amr.max_grid_size** is imposed.
- **amr.max_grid_size** = 64
  The final grids will be no longer than 64 cells on a side at every level.
- **amr.max_grid_size** = 64 32 16
  The final grids will be no longer than 64 cells on a side at level 0, 32 cells on a side at level 1, and 16 cells on a side at level 2.
- **amr.blocking_factor** = 32
  The dimensions of all the final grids will be multiples of 32 at all levels.
- **amr.blocking_factor** = 32 16 8
  The dimensions of all the final grids will be multiples of 32 at level 0, multiples of 16 at level 1, and multiples of 8 at level 2..

Having grids that are large enough to coarsen multiple levels in a V-cycle is essential for good multigrid performance in simulations that use self-gravity.

2.5.5 How Grids are Created

The gridding algorithm proceeds in this order:

1. Grids are created using the Berger-Rigoutsis clustering algorithm modified to ensure that all new fine grids are divisible by **amr.blocking_factor**.

2. Next, the grid list is chopped up if any grids are larger than **max_grid_size**. Note that because **amr.max_grid_size** is a multiple of **amr.blocking_factor** the **amr.blocking_factor** criterion is still satisfied.

3. Next, if **amr.refine_grid_layout** = 1 and there are more processors than grids, and

   - if **amr.max_grid_size** / 2 is a multiple of **amr.blocking_factor**

   then the grids will be redefined, at each level independently, so that the maximum length of a grid at level ℓ, in any dimension, is **amr.max_grid_size**[ℓ] / 2.

4. Finally, if **amr.refine_grid_layout** = 1, and there are still more processors than grids, and

   - if **amr.max_grid_size** / 4 is a multiple of **amr.blocking_factor**

   then the grids will be redefined, at each level independently, so that the maximum length of a grid at level ℓ, in any dimension, is **amr.max_grid_size**[ℓ] / 4.

2.6 Simulation Time

2.6.1 List of Parameters

Parameter	Definition	Acceptable Values	Default
max_step	maximum number of level 0 time steps	Integer ≥ 0	-1
stop_time	final simulation time	Real ≥ 0	-1.0
nyx.final_a	if **nyx.use_comoving = t** and positive value then this is final value of a	Real > 0	-1.0
nyx.final_z	if **nyx.use_comoving = t** and positive value then this is final value of z	Real > 0	-1.0

2.6.2 Notes

To control the number of time steps, you can limit by the maximum number of level 0 time steps (**max_step**), or the final simulation time (**stop_time**), or both. The code will stop at whichever criterion comes first. Note that if the code reaches **stop_time** then the final time step will be shortened so as to end exactly at **stop_time**, not pass it.

If running in comoving coordinates you can also set a final value of a by setting **nyx.final_a**, or a final value of z by setting **nyx.final_z**. You may only specify one or the other of these. Once this value of a or z is reached in a time step, the code will stop at the end of this full coarse time step. (Note it does not stop at a (or z) exactly equal to the final value, rather it stops once a is greater than (or z is less than) this value.)

2.6.3 Examples of Usage

- **max_step** = 1000
- **stop_time** = 1.0

will end the calculation when either the simulation time reaches 1.0 or the number of level 0 steps taken equals 1000, whichever comes first.

2.7 Time Step

- If **nyx.do_hydro**= 1, then typically the code chooses a time step based on the CFL number (dt = cfl * dx / max(u+c)).
- If **nyx.do_hydro**= 0 and we are running with dark matter particles, then we use a time step based on the velocity of the particles, i.e. we set Δt so that the particle goes no further than $f\Delta t$ in a coordinate direction where $0 \leq f \leq 1$. The value for f is currently hard-wired in Particles.H, but it will become an inputs parameter.

2.7.1 List of Parameters

Parameter	Definition	Acceptable Values	Default
nyx.cfl	CFL number for hydro	Real > 0 and ≤ 1	0.8
particles.cfl	CFL number for particles	Real > 0 and ≤ 1	0.5
nyx.init_shrink	factor by which to shrink the initial time step	Real > 0 and ≤ 1	1.0
nyx.change_max	factor by which the time step can grow in subsequent steps	Real ≥ 1	1.1
nyx.fixed_dt	level 0 time step regardless of cfl or other settings	Real > 0	unused if not set
nyx.initial_dt	initial level 0 time step regardless of other settings	Real > 0	unused if not set
nyx.dt_cutoff	time step below which calculation will abort	Real > 0	0.0

2.7.2 Examples of Usage

- **nyx.cfl** = 0.9
  defines the timestep as dt = cfl * dx / umax_hydro.
- **particles.cfl** = 0.9
  defines the timestep as dt = cfl * dx / umax_particles where umax_particles is the maximum velocity of any particle in the domain.
- **nyx.init_shrink** = 0.01
  sets the initial time step to 1% of what it would be otherwise.
- **nyx.change_max** = 1.1
  allows the time step to increase by no more than 10% in this case. Note that the time step can shrink by any factor; this only controls the extent to which it can grow.
- **nyx.fixed_dt** = 1.e-4
  sets the level 0 time step to be 1.e-4 for the entire simulation, ignoring the other timestep controls. Note that if **nyx.init_shrink** $\neq$ 1 then the first time step will in fact be **nyx.init_shrink** * **nyx.fixed_dt**.

- **nyx.initial_dt** = 1.e-4
  sets the *initial* level 0 time step to be 1.e-4 regardless of **nyx.cfl** or **nyx.fixed_dt**. The time step can grow in subsequent steps by a factor of **nyx.change_max** each step.

- **nyx.dt_cutoff** = 1.e-20
  tells the code to abort if the time step ever gets below 1.e-20. This is a safety mechanism so that if things go nuts you don't burn through your entire computer allocation because you don't realize the code is misbehaving.

2.8 Subcycling

Nyx supports a number of different modes for subcycling in time.

- If **amr.subcycling_mode**=Auto (default), then the code will run with equal refinement in space and time. In other words, if level $n+1$ is a factor of 2 refinement above level n, then $n+1$ will take 2 steps of half the duration for every level n step.

- If **amr.subcycling_mode**=None, then the code will not refine in time. All levels will advance together with a timestep dictated by the level with the strictest dt. Note that this is identical to the deprecated command **amr.nosub = 1**.

- If **amr.subcycling_mode**=Manual, then the code will subcycle according to the values supplied by **subcycling_iterations**.

2.8.1 List of Parameters

Parameter	Definition	Acceptable Values	Default
amr.subcycling_mode	How shall we subcycle	Auto, None or Manual	Auto
amr.subcycling_iterations	Number of cycles at each level	1 or `ref_ratio`	must be set in Manual mode

2.8.2 Examples of Usage

- **amr.subcycling_mode**=Manual
  Subcycle in manual mode with largest allowable timestep.

- **amr.subcycling_iterations** = 1 2 1 2
  Take 1 level 0 timestep at a time (required). Take 2 level 1 timesteps for each level 0 step, 1 timestep at level 2 for each level 1 step, and take 2 timesteps at level 3 for each level 2 step.

- **amr.subcycling_iterations** = 2
  Alternative form. Subcycle twice at every level (except level 0).

2.9 Restart Capability

Nyx has a standard sort of checkpointing and restarting capability. In the inputs file, the following options control the generation of checkpoint files (which are really directories):

2.9.1 List of Parameters

Parameter	Definition	Acceptable Values	Default
amr.check_file	prefix for restart files	Text	"chk"
amr.check_int	how often (by level 0 time steps) to write restart files	Integer > 0	-1
amr.check_per	how often (by simulation time) to write restart files	Real > 0	-1.0
amr.restart	name of the file (directory) from which to restart	Text	not used if not set
amr.checkpoint_files_output	should we write checkpoint files	0 or 1	1
amr.check_nfiles	how parallel is the writing of the checkpoint files	Integer ≥ 1	64
amr.checkpoint_on_restart	should we write a checkpoint immediately after restarting	0 or 1	0

2.9.2 Notes

- You should specify either **amr.check_int** or **amr.check_per**. Do not try to specify both.
- Note that if **amr.check_per** is used then in order to hit that exact time the code may modify the time step slightly, which will change your results ever so slightly than if you didn't set this flag.
- Note that **amr.plotfile_on_restart** and **amr.checkpoint_on_restart** only take effect if **amr.regrid_on_restart** is in effect.
- See the Software Section for more details on parallel I/O and the **amr.check_nfiles** parameter.
- If you are doing a scaling study then set **amr.checkpoint_files_output** = 0 so you can test scaling of the algorithm without I/O.

2.9.3 Examples of Usage

- **amr.check_file** = chk_run
- **amr.check_int** = 10

  means that restart files (really directories) starting with the prefix "chk_run" will be generated every 10 level 0 time steps. The directory names will be *chk_run00000*, *chk_run00010*, *chk_run00020*, etc.

If instead you specify

- **amr.check_file** = chk_run
- **amr.check_per** = 0.5

  then restart files (really directories) starting with the prefix "chk_run" will be generated every 0.1 units of simulation time. The directory names will be *chk_run00000*, *chk_run00043*, *chk_run00061*, etc, where $t = 0.1$ after 43 level 0 steps, $t = 0.2$ after 61 level 0 steps, etc.

To restart from *chk_run00061*,for example, then set

- **amr.restart** = chk_run00061

2.10 Controlling PlotFile Generation

The main output from Nyx is in the form of plotfiles (which are really directories). The following options in the inputs file control the generation of plotfiles

2.10.1 List of Parameters

Parameter	Definition	Acceptable Values	Default
amr.plot_file	prefix for plotfiles	Text	"plt"
amr.plot_int	how often (by level 0 time steps) to write plot files	Integer > 0	-1
amr.plot_per	how often (by simulation time) to write plot files	Real > 0	-1.0
amr.plot_vars	name of state variables to include in plotfiles	ALL, NONE or list	ALL
amr.derive_plot_vars	name of derived variables to include in plotfiles	ALL, NONE or list	NONE
amr.plot_files_output	should we write plot files	0 or 1	1
amr.plotfile_on_restart	should we write a plotfile immediately after restarting	0 or 1	0
amr.plot_nfiles	how parallel is the writing of the plotfiles	Integer ≥ 1	64
nyx.plot_phiGrav	Should we plot the gravitational potential	0 or 1	0
	plot the gravitational potential	0 or 1	0
particles.write_in_plotfile	Should we write the particles in a file within the plotfile	0 or 1	0
fab.format	Should we write the plotfile in double or single precision	NATIVE or IEEE32	NATIVE

All the options for **amr.derive_plot_vars** are kept in `derive_lst` in Nyx_setup.cpp. Feel free to look at it and see what's there.

2.10.2 Notes

- You should specify either **amr.plot_int** or **amr.plot_per**. Do not try to specify both.
- Note that if **amr.plot_per** is used then in order to hit that exact time the code may modify the time step slightly, which will change your results ever so slightly than if you didn't set this flag.
- See the Software Section for more details on parallel I/O and the **amr.plot_nfiles** parameter.
- If you are doing a scaling study then set **amr.plot_files_output** = 0 so you can test scaling of the algorithm without I/O.
- **nyx.plot_phiGrav** is only relevant if **nyx.do_grav** = 1 and **gravity.gravity_type** = PoissonGrav
- By default, plotfiles are written in double precision (NATIVE format). If you want to save space by writing them in single precision, set the fab.format flag to IEEE32.

2.10.3 Examples of Usage

- **amr.plot_file** = plt_run
- **amr.plot_int** = 10

  means that plot files (really directories) starting with the prefix "plt_run" will be generated every 10 level 0 time steps. The directory names will be *plt_run00000*, *plt_run00010*, *plt_run00020*, etc.

If instead you specify

- **amr.plot_file** = plt_run
- **amr.plot_per** = 0.5

  then restart files (really directories) starting with the prefix "plt_run" will be generated every 0.1 units of simulation time. The directory names will be *plt_run00000*, *plt_run00043*, *plt_run00061*, etc, where $t = 0.1$ after 43 level 0 steps, $t = 0.2$ after 61 level 0 steps, etc.

2.11 Screen Output

2.11.1 List of Parameters

Parameter	Definition	Acceptable Values	Default
amr.v	verbosity of Amr.cpp	0 or 1	0
nyx.v	verbosity of Nyx.cpp	0 or 1	0
gravity.v	verbosity of Gravity.cpp	0 or 1	0
mg.v	verbosity of multigrid solver (for gravity)	0,1,2,3,4	0
particles.v	verbosity of particle-related processes	0,1,2,3,4	0
amr.grid_log	name of the file to which the grids are written	Text	not used if not set
amr.run_log	name of the file to which certain output is written	Text	not used if not set
amr.run_log_terse	name of the file to which certain (terser) output is written	Text	not used if not set
amr.sum_interval	if > 0, how often (in level 0 time steps) to compute and print integral quantities	Integer	-1
nyx.do_special_tagging		0 or 1	1

2.11.2 Notes

- **nyx.do_special_tagging** = 1 allows the user to set a special flag based on user-specified criteria. This can be used, for example, to calculate the bounce time in a core collapse simulation; the bounce time is defined as the first time at which the maximum density in the domain exceeds a user-specified value. This time can then be printed into a special file as a useful diagnostic.

2.11.3 Examples of Usage

- **amr.grid_log** = grdlog
  Every time the code regrids it prints a list of grids at all relevant levels. Here the code will write these grids lists into the file *grdlog*.

- **amr.run_log** = runlog
  Every time step the code prints certain statements to the screen (if **amr.v** = 1), such as
  STEP = 1 TIME = 1.91717746 DT = 1.91717746
  PLOTFILE: file = plt00001
  Here these statements will be written into *runlog* as well.

- **amr.run_log_terse** = runlogterse
  This file, *runlogterse* differs from *runlog*, in that it only contains lines of the form

10 0.2 0.005
in which "10" is the number of steps taken, "0.2" is the simulation time, and "0.005" is the level 0 time step. This file can be plotted very easily to monitor the time step.

- **nyx.sum_interval** = 2
  if **nyx.sum_interval** > 0 then the code computes and prints certain integral quantities, such as total mass, momentum and energy in the domain every **nyx.sum_interval** level 0 steps. In this example the code will print these quantities every two coarse time steps. The print statements have the form
  TIME= 1.91717746 MASS= 1.792410279e+34
  for example. If this line is commented out then it will not compute and print these quanitities.

2.12 Gravity

2.12.1 List of Parameters

Parameter	Definition	Acceptable Values	Default
nyx.do_grav	Include gravity as a forcing term	0 if false, 1 if true	must be set i
gravity.gravity_type	if **nyx.do_grav** = 1, how shall gravity be calculated	CompositeGrav, PoissonGrav	must be set
gravity.no_sync	if **gravity.gravity_type** = PoissonGrav, whether to perform the "sync solve"	0 or 1	0
gravity.no_composite	if **gravity.gravity_type** = PoissonGrav, whether to perform a composite solve	0 or 1	0

2.12.2 Notes

Gravity types are CompositeGrav or PoissonGrav. See the Gravity section for more detail.

- To include gravity you must set
  - USE_GRAV = TRUE in the GNUmakefile
  - **nyx.do_grav** = 1 in the inputs file
- **gravity.gravity_type** is only relevant if **nyx.do_grav** = 1
- **gravity.no_sync** and **gravity.no_composite** are only relevant if **gravity.gravity_type** = PoissonGrav, i.e. the code does a full Poisson solve for self-gravity.

2.13 Physics

2.13.1 List of Parameters

Parameter	Definition	Acceptable Values	Default
nyx.do_hydro	Time-advance the fluid dynamical equations	0 if false, 1 if true	must be set
nyx.do_react	Include reactions	0 if false, 1 if true	must be set
nyx.add_ext_src	Include additional user-specified source term	0 if false, 1 if true	0
nyx.use_const_species	If 1 then read h_species and he_species	0 or 1	0
nyx.h_species	Concentration of H	$0 < X < 1$	0
nyx.he_species	Concentration of He	$0 < X < 1$	0

Chapter 3

Units and Constants

3.1 Units and Constants

We support two different systems of units in Nyx: CGS and Cosmological. All inputs and problem initialization should be specified consistently with one of these sets of units. No internal conversions of units occur within the code, so the output must be interpreted appropriately.

The default is cosmological units.

If you want to use CGS units instead, then set

USE_CGS = TRUE

in your GNUmakefile. This will select the file constants_cgs.f90 instead of constants_cosmo.f90 from the Nyx/constants directory.

Location	Variable	CGS	Cosmological	Conversion Data
inputs file	**geometry.prob_lo** **geometry.prob_hi**	cm cm	Mpc Mpc	1Mpc = 3.08568025e24 cm 1Mpc = 3.08568025e24 cm
Hydro Initialization	density	g / cm^3	$M_\odot$ / Mpc^3	1 ($M_\odot$ / Mpc^3) = .06769624e-39 (g/cm^3)
Hydro Initialization	velocities	cm/s	km/s	1km = 1.e5 cm
Hydro Initialization	momenta	(g/cm^3) (cm/s)	($M_\odot/Mpc^3$) (km/s)	1km = 1.e5 cm 1 ($M_\odot$ / Mpc^3) = .06769624e-39 g/cm^3
Hydro Initialization	temperature	K	K	1
Hydro Initialization	specific energy (e or E)	erg/g= $(cm/s)^2$	$(km/s)^2$	1 $(km/s)^2$ = 1.e10 $(cm/s)^2$
Hydro Initialization	energy (ρe or ρE)	erg / cm^3 = (g/cm^3) $(cm/s)^2$	($M_\odot/Mpc^3$) $(km/s)^2$	1 $(km/s)^2$ = 1.e10 $(cm/s)^2$ 1 ($M_\odot$ / Mpc^3) = .06769624e-39 g/cm^3
Particle Initialization	particle mass	g	$M_\odot$	1 $M_\odot$ = 1.98892e33 g
Particle Initialization	particle locations	cm	Mpc	1 Mpc = 3.08568025e24 cm
Particle Initialization	particle velocities	cm/s	km/s	1 km = 1e5 cm
Output	Pressure	g $(cm/s)^2$ / cm^3	$M_\odot$ $(km/s)^2$ / Mpc^3	1 $M_\odot$ $(km/s)^2$ / Mpc^3 = .06769624e-29 g $(cm/s)^2$ / cm^3
Output	Gravity	(cm/s) / s	$(km/s)^2$ / Mpc	1 $M_\odot$ $(km/s)^2$ / Mpc^3 =
Output	Time	s	(Mpc/km) s	1 Mpc = 3.08568025e19 km

Table 3.1: Units

Constant	CGS	Cosmological	Conversion Data
Gravitational constant (G)	6.67428e-8 cm $(\mathrm{cm/s})^2$ g^{-1}	4.3019425e-9 Mpc $(\mathrm{km/s})^2$ $\mathrm{M}_\odot^{-1}$	
Avogadro's number (n_A)	6.02214129e23 g^{-1}	1.1977558e57 $\mathrm{M}_\odot^{-1}$	1 $\mathrm{M}_\odot$ = 1.98892e33 g
Boltzmann's constant (k_B)	1.3806488e-16 erg / K	0.6941701e-59 $\mathrm{M}_\odot$ $(\mathrm{km/s})^2$ / K	1 $\mathrm{M}_\odot$ $(\mathrm{km/s})^2$ = 1.98892e43 g $(\mathrm{cm/s})^2$
Hubble constant (H)	100 (km/s) / Mpc	32.407764868e-19 s^{-1}	1 Mpc = 3.08568025e19 km

Table 3.2: Constants

The only other place that dimensional numbers are used in the code is in the tracing and Riemann solve. We set three *small* numbers which need to be consistent with the data specified. Each of these can be specified in the inputs file.

- small_dens – small value for density
- small_p – small value for pressure
- small_T – small value for temperature

These are the places that each is used in the code:

- **small_dens**
  - **subroutine enforce_minimum_density** (called after subroutine consup) – if $\rho <$ small_dens then ρ is set to the minimum value of the 26 neighbors. This also modifies momenta, ρE and ρe so that velocties, E and e remain unchanged.
  - **subroutine tracexy / tracez / tracexy_ppm / tracez_ppm**:
    qxp = max(qxp,small_dens)
    qxm = max(qxm,small_dens)
    and analogously for qyp/qym and qzp/qzm. This only modifies density inside the tracing, not the other variables
  - **subroutine riemannus** – we set
    wsmall = small_dens * csmall
    and then
    wl = max(wsmall, sqrt(gaml * pl * rl))
    wr = max(wsmall, sqrt(gamr * pr * rr))
    Also, we set
    ro = max(small_dens,ro)
    where ro = 0.5 * (rl + rr) – this state is only chosen when ustar = 0, and
    rstar = max(small_dens,rstar)
    where rstar = ro + (pstar-po)/co^2
  - **subroutine react_state** – only compute reaction if $\rho >$ small_dens
- **small_temp**:
  - **subroutine ctoprim**: if $\rho e < 0$, then

    call subroutine nyx_eos_given_RTX (e,...,small_temp,...) in order to compute a new energy, e.

    This energy is then used to

call subroutine nyx_eos_given_ReX in order to compute the sound speed, c.

Coming out of this the temperature is equal to small_temp and the energy e has been reset.

- **subroutine react_state**: if $\rho e < 0$, then

  call subroutine nyx_eos_given_RTX (e,...,small_temp,...) in order to compute a new energy, e.

  This energy is then used to proceed with the burner routine.

- **subroutine reset_internal_energy**: if $e < 0$ and $E - ke < 0$ then

  call subroutine nyx_eos_given_RTX (e,...,small_temp,...) in order to compute a new energy, e. This energy is also used to

  define a new $E = e + ke$

- **small_pres**:
  - **subroutine riemannus** – we set
    pstar = max(small_pres,pstar)

    pgdnv = max(small_pres,pgdnv). Note that pgdnv is the pressure explicitly used in the fluxes.
  - **subroutine uflaten** – small_pres is used to keep the denominator away from zero
  - Everywhere we define values of pressure on a face, we set that value to be at least small_pres.

Chapter 4

Equations in Comoving Coordinates

4.1 Hydrodynamic Equations in Comoving Coordinates

4.1.1 Conservative Form

We solve the equations of gas dynamics in a coordinate system that is comoving with the expanding universe, with expansion factor, a, related to the redshift, z, by $a = 1/(1+z)$. The continuity equation is written,

$$\frac{\partial \rho_b}{\partial t} = -\frac{1}{a}\nabla \cdot (\rho_b \mathbf{U}) \ , \tag{4.1}$$

where ρ_b is the comoving baryonic density, related to the proper density by $\rho_b = a^3 \rho_{proper}$, and $\mathbf{U}$ is the proper peculiar baryonic velocity.

The momentum evolution equation can be expressed as

$$\frac{\partial(\rho_b \mathbf{U})}{\partial t} = \frac{1}{a}\left(-\nabla \cdot (\rho_b \mathbf{U}\mathbf{U}) - \nabla p + \rho_b \mathbf{g} + \mathbf{S}_{\rho\mathbf{U}} - \dot{a}\rho_b \mathbf{U}\right) \ , \tag{4.2}$$

or equivalently,

$$\frac{\partial(a\rho_b \mathbf{U})}{\partial t} = -\nabla \cdot (\rho_b \mathbf{U}\mathbf{U}) - \nabla p + \rho_b \mathbf{g} + \mathbf{S}_{\rho\mathbf{U}} \ , \tag{4.3}$$

where the pressure, p, that appears in the evolution equations is related to the proper pressure, p_{proper}, by $p = a^3 p_{proper}$. Here $\mathbf{g} = -\nabla\phi$ is the gravitational acceleration vector, and $\mathbf{S}_{\rho\mathbf{U}}$ represents any external forcing terms.

The energy equation can be written,

$$\frac{\partial(\rho_b E)}{\partial t} = \frac{1}{a}\left[-\nabla \cdot (\rho_b \mathbf{U} E + p\mathbf{U}) + (\rho_b \mathbf{U} \cdot \mathbf{g} + S_{\rho E}) - \dot{a}(3(\gamma-1)\rho_b e + \rho_b(\mathbf{U}\cdot\mathbf{U}))\right] \ . \tag{4.4}$$

or equivalently,

$$\frac{\partial(a^2 \rho_b E)}{\partial t} = a\left[-\nabla \cdot (\rho_b \mathbf{U} E + p\mathbf{U}) + \rho_b \mathbf{U} \cdot \mathbf{g} + S_{\rho E} + \dot{a}(\ (2-3(\gamma-1))\ \rho_b e)\right] \ . \tag{4.5}$$

Here $E = e + \mathbf{U}\cdot\mathbf{U}/2$ is the total energy per unit mass, where e is the specific internal energy. $S_{\rho E} = S_{\rho e} + \mathbf{U}\cdot\mathbf{S}_{\rho\mathbf{U}}$ where $S_{\rho e} = \Lambda^H - \Lambda^C$ represents the heating and cooling terms, respectively. We can write the evolution equation for internal energy as

$$\frac{\partial(\rho_b e)}{\partial t} = \frac{1}{a}\left[-\nabla \cdot (\rho_b \mathbf{U} e) - p\nabla\cdot\mathbf{U} - \dot{a}(3(\gamma-1)\rho_b e) + S_{\rho e}\right] \ . \tag{4.6}$$

or equivalently,

$$\frac{\partial(a^2\rho_b e)}{\partial t} = a\left[-\nabla\cdot(\rho_b\mathbf{U}e) - p\nabla\cdot\mathbf{U} + S_{\rho e} + \dot{a}(\ (2-3(\gamma-1))\ \rho_b e)\right] \ . \tag{4.7}$$

Note that for a gamma-law gas with $\gamma = 5/3$, we can write

$$\frac{\partial(a^2\rho_b E)}{\partial t} = a\left[-\nabla\cdot(\rho_b\mathbf{U}E + p\mathbf{U}) + \rho_b\mathbf{U}\cdot\mathbf{g} + S_{\rho e}\right] \ . \tag{4.8}$$

and

$$\frac{\partial(a^2\rho_b e)}{\partial t} = a\left[-\nabla\cdot(\rho_b\mathbf{U}e) - p\nabla\cdot\mathbf{U} + S_{\rho e}\right] \ . \tag{4.9}$$

4.1.2 Tracing

In order to compute the fluxes on faces, we trace $\rho, \mathbf{U}, \rho e$ and p to the faces.
Thus we must convert the momentum evolution equation into a velocity evolution equation:

$$\frac{\partial\mathbf{U}}{\partial t} = \frac{1}{\rho_b}\left(\frac{\partial(\rho_b\mathbf{U})}{\partial t} - \mathbf{U}\frac{\partial\rho_b}{\partial t}\right) \tag{4.10}$$

$$= \frac{1}{a\rho_b}\left(-\nabla\cdot(\rho_b\mathbf{U}\mathbf{U}) - \nabla p + \rho_b\mathbf{g} + S_{\rho\mathbf{U}} - \dot{a}\rho_b\mathbf{U}\right) + \frac{1}{a}\mathbf{U}\ \nabla\cdot(\rho_b\mathbf{U}) \tag{4.11}$$

$$= \frac{1}{a}\left(-\mathbf{U}\cdot\nabla\mathbf{U} - \frac{1}{\rho_b}\nabla p + \mathbf{g} + \frac{1}{\rho_b}\mathbf{S}_{\rho\mathbf{U}} - \dot{a}\mathbf{U}\right) \ . \tag{4.12}$$

4.2 Subgrid Scale Model in Comoving Coordinates

The fundamental modification to the standard compressible equations is the addition of a SGS turbulence energy variable, K and associated source terms in the equations for the evolution of velocity, total energy, and K [**?**, **?**, **?**]. The set of conservation equations in comoving coordinates (4.1)–(4.5) then becomes [**?**]:

$$\frac{\partial\rho_b}{\partial t} = -\frac{1}{a}\nabla\cdot(\rho_b\mathbf{U}) \ , \tag{4.13}$$

$$\frac{\partial(a\rho_b\mathbf{U})}{\partial t} = -\nabla\cdot(\rho_b\mathbf{U}\mathbf{U}) - \nabla p + \nabla\cdot\boldsymbol{\tau} + \rho_b\mathbf{g} \ , \tag{4.14}$$

$$\frac{\partial(a^2\rho_b E)}{\partial t} = -a\nabla\cdot(\rho_b\mathbf{U}E + p\mathbf{U}) + a\rho_b\mathbf{U}\cdot\mathbf{g} + a\nabla\cdot(\mathbf{U}\cdot\boldsymbol{\tau}) - a^2(\Sigma - \rho_b\varepsilon) + a\dot{a}\left((2-3(\gamma-1))\rho_b e\right) + a^2(\Lambda^H - \Lambda^C) \ , \tag{4.15}$$

$$\frac{\partial(a^2\rho_b K)}{\partial t} = -a\nabla\cdot(\rho_b\mathbf{U}K) + a\nabla\cdot(\rho_b\kappa_{\rm sgs}\nabla K) + a^2(\Sigma - \rho_b\varepsilon) \ . \tag{4.16}$$

The interaction between resolved and unresolved turbulent eddies is described by the SGS turbulence stress tensor $\boldsymbol{\tau}$. Since inertial-range dynamics of turbulence is scale-invariant, we conjecture that $\boldsymbol{\tau}$ in comoving coordinates has the same form as for non-expanding fluids. For compressible turbulence, the following closure is proposed in [**?**]:

$$\tau_{ij} = 2C_1\Delta\rho_b(2K_{\rm sgs})^{1/2}S^*_{ij} - 4C_2\rho_b K\frac{U_{i,k}U_{j,k}}{|\nabla\mathbf{U}|^2} - \frac{2}{3}(1-C_2)\rho_b K\delta_{ij}. \tag{4.17}$$

where $|\nabla \mathbf{U}| := (2U_{i,k}U_{i,k})^{1/2}$ is the norm of the resolved velocity derivative,

$$S^*_{ij} = S_{ij} - \frac{1}{3}\delta_{ij}d = \frac{1}{2}(U_{i,j} + U_{j,i}) - \frac{1}{3}\delta_{ij}U_{k,k} \tag{4.18}$$

is the trace-free rate-of strain, and $\Delta = (\triangle x \ \triangle y \ \triangle z)^{1/3}$ is the grid scale in comoving coordinates. The production and dissipation terms in equation (4.16) are defined as follows:

$$\Sigma = \frac{1}{a}\tau_{ij}S_{ij}, \tag{4.19}$$

$$\varepsilon = \frac{C_\varepsilon K^{3/2}}{a\Delta}, \tag{4.20}$$

and $\kappa_{\rm sgs} = C_\kappa \Delta K^{1/2}$ is the SGS diffusivity. Here we assume that the Reynolds number of turbulence is high such that the damping of turbulent eddies by the microscopic viscosity of the fluid occurs entirely on the subgrid scales. Because of the numerical viscosity of PPM, however, part of the numerically resolved kinetic energy will be dissipated directly into internal energy.

Chapter 5

Gravity

In NYX we always compute gravity by solving a Poisson equation on the mesh hierarchy. To make sure this option is chosen correctly, we must always set

USE_GRAV = TRUE

in the GNUmakefile and

castro.do_grav = 1
gravity.gravity_type = PoissonGrav

in the inputs file.

To define the gravitational vector we set

$$\mathbf{g}(\mathbf{x}, t) = -\nabla\phi \tag{5.1}$$

where

$$\mathbf{\Delta}\phi = \frac{4\pi G}{a}(\rho - \overline{\rho}) \tag{5.2}$$

where $\overline{\rho}$ is the average of ρ over the entire domain if we assume triply periodic boundary conditions, and $a(t)$ is the scale of the universe as a function of time.

Chapter 6

Dark Matter Particles

For the moment, assume that we are running in comoving coordinates, with dark matter particles only (no hydro) and that the particles all exist at level 0. These assumptions are encapsulated in the following lines in the inputs file:

nyx.use_comoving = t
nyx.do_dm_particles = 1
amr.max_level = 0
nyx.do_hydro = 0
nyx.do_react = 0
nyx.do_grav = 1

6.1 Equations

If we define $\mathbf{x}_i$ and $\mathbf{u}_i$ as the location and velocity of particle i, respectively, then we wish to solve

$$\frac{d\mathbf{x}_i}{dt} = \frac{1}{a}\mathbf{u}_i \tag{6.1}$$

$$\frac{d(a\mathbf{u}_i)}{dt} = \mathbf{g}_i \tag{6.2}$$

where $\mathbf{g}_i$ is the gravitational force evaluated at the location of particle i, i.e., $\mathbf{g}_i = \mathbf{g}(\mathbf{x}_i, t)$.

6.2 Initializing the Particles

There are several different ways in which particles can currently be initialized:

6.2.1 Read from an ASCII file

To enable this option, set

nyx.particle_init_type = AsciiFile
nyx.ascii_particle_file =*particle_file*

Here *particle_file* is the user-specified name of the file. The first line in this file is assumed to contain the number of particles. Each line after that contains

x y z mass xdot ydot zdot

Note that the variable that we call the particle velocity, $\mathbf{u} = a\dot{\mathbf{x}}$, so we must multiply $\dot{\mathbf{x}}$, by a when we initialize the particles.

6.2.2 Read from a binary file

To enable this option, set

nyx.particle_init_type = BinaryFile
nyx.binary_particle_file =*particle_file*

As with the ASCII read, the first line in this file is assumed to contain the number of particles. Each line after that contains

x y z mass xdot ydot zdot

Note that the variable that we call the particle velocity, $\mathbf{u} = a\dot{\mathbf{x}}$, so we must multiply $\dot{\mathbf{x}}$, by a when we initialize the particles.

6.2.3 Read from a binary "meta" file

This option allows you to read particles from a series of files rather than just a single file. To enable this option, set

nyx.particle_init_type = BinaryMetaFile
nyx.binary_particle_file =*particle_file*

In this case the *particle_file* you specify is an ASCII file specifying a list of file names with full paths. Each of the files in this list is assumed to be binary and is read sequentially (individual files are read in parallel) in the order listed.

6.2.4 Reading SPH particles

For some applications it is useful to initialize the grid data with SPH-type particles. To enable this option, you must set

nyx.do_santa_barbara = 1
nyx.init_with_sph_particles = 1

The SPH-type particles can then be read in by setting

nyx.sph_particle_file =*sph_particle_file*

where *sph_particle_file* is the user-specified name of the file containing the SPH particles. The type of *sph_particle_file* must be the same (Ascii, Binary or BinaryMeta) as the dark matter particle file as specified by

nyx.particle_init_type =

The SPH particles will be discarded by the code once the grid data has been initialized.

6.2.5 Random placement

To enable this option, set

nyx.nyx.particle_init_type = Random

There are then a number of parameters to set, for example:

nyx.particle_initrandom_count = 100000
nyx.particle_initrandom_mass = 1
nyx.particle_initrandom_iseed = 15

6.2.6 Cosmological

Using cosmological initial conditions is a three step process:

1. Generating a transfer function (e.g. with `camb`)
2. Generating an initial displacement field (with `nyx-ic`)
3. Starting nyx

In the following we will look at each step a bit closer.

6.2.6.1 Generating a transfer function

The transfer function is used in `nyx-ic` to generate the power spectrum. The usual way is to use `camb`[1] to calculate it for the desired universe. A sample `camb.ini` is provided with `nyx-ic`. The important options are:

- **transfer_redshift(1) = 50**
- **transfer_matterpower(1) = tf**

which determine the initial time for the simulation. You should make sure that you catch all necessary wave numbers for the considered box length and resolution.

From the `camb` output you have to note values for `sigma_8` for a redshift of zero and the initial redshift. We need this to compute the right normalization.

[1] See `http://camb.info/`

6.2.6.2 Setting up the initial displacements

We calculate the initial displacements with a stand-alone program called `nyx-ic`. This takes a transfer function and some cosmological parameters as an argument and outputs an "init" directory which basically contains initial displacements for every grid point in an AMReX MultiFAB. Furthermore the mf contains a fourth field containing the density contrast as initial condition for the baryonic matter.
`nyx-ic` is started with an "inputs" file similar to the one from Nyx. A sample one is provided. The options are

```
#Omega_{Matter}
cosmo.omegam = 0.272
#Omega_{Lambda}
cosmo.omegax = 0.728

#equation of state paramater omega_{effective}
cosmo.weff = -0.980

#Omega_{baryon}*Hubble^2
cosmo.ombh2 = 0.0226
#Hubble/100km/s
cosmo.hubble = 0.704
#scalar spectral index
cosmo.enn = 0.963
# initial z
cosmo.z_init = 50

#sidelength of the box (in Mpc)
cosmo.boxside = 90.14
#seed of the rng
cosmo.isd = 100
#resolution of the box
cosmo.gridpoints = 256
#the output file name
cosmo.initDirName = init

#choose the source of the transferfunction
cosmo.transferfunction = CAMB

#some tabulated transferfunction generated with camb (compare camb-ini-file)
cosmo.tabulatedTk = tf
# sigma8 for the input tf at z=0 and initial z (to calc the growthfactor)
cosmo.init_sigma8_0 = 0.7891368
cosmo.init_sigma8_init = 2.0463364E-02
```

The code solves the equation

$$P(k,a) = 2\pi^2 \delta_H^2 \frac{k^n}{H_0^{n+3}} T^2(k) \left(\frac{D(a)}{D(a=1)} \right)^2 \tag{6.3}$$

to calculate P and from that gaussian distributed density perturbations δ following that spectrum. Particle displacements are then calculated as Zel'dovich displacements.

Non-gaussian effects as well as neutrino contributions are planned for the future.

6.2.6.3 Using Nyx with cosmological initial conditions

- **nyx.nyx.particle_init_type = Cosmological**
  set the *right* init type
- **cosmo.initDirName = init**
  set the name of the displacements directory (amrex format)
- **cosmo.particle_mass = 0.19178304E+10**
  sets the mass $[M_\odot]$ of each particle
- **cosmo.omegam = 0.272**
  set Ω_{Matter}
- **cosmo.omegax = 0.728**
  set Ω_Λ
- **cosmo.hubble = 0.704**
  set the reduced hubble constant h

We will generate a particle of mass **particle_mass** in every grid cell displaced from the center by the value found in the **initDirName** for that cell. Velocities are calculated in the Zel'dovich approximation by

$$\vec{v} = \Delta\vec{x} \times 100\text{km/s} \times a\sqrt{\Omega_M/a^3 + \Omega_\Lambda} \times L_{\text{box}} \tag{6.4}$$

where $\Delta\vec{x}$ is the displacement of the particle.

6.3 Time Stepping

There are currently two different ways in which particles can be moved:

6.3.1 Random

To enable this option, set

nyx.particle_move_type = Random

Update the particle positions at the end of each coarse time step using a random number between 0 and 1 multiplied by 0.25 dx.

6.3.2 Motion by Self-Gravity

To enable this option, set

nyx.particle_move_type = Gravitational

6.3.2.1 Move-Kick-Drift Algorithm

In each time step:

- Solve for $\mathbf{g}^n$ (only if multilevel, otherwise use $\mathbf{g}^{n+1}$ from previous step)
- $\mathbf{u}_i^{n+1/2} = \frac{1}{a^{n+1/2}}((a^n \mathbf{u}_i^n) + \frac{\Delta t}{2}\ \mathbf{g}_i^n)$
- $\mathbf{x}_i^{n+1} = \mathbf{x}_i^n + \frac{\Delta t}{a^{n+1/2}} \mathbf{u}_i^{n+1/2}$
- Solve for $\mathbf{g}^{n+1}$ using $\mathbf{x}_i^{n+1}$
- $\mathbf{u}_i^{n+1} = \frac{1}{a^{n+1}}((a^{n+1/2} \mathbf{u}_i^{n+1/2}) + \frac{\Delta t}{2}\ \mathbf{g}_i^{n+1})$

Note that at the end of the timestep $\mathbf{x}_i^{n+1}$ is consistent with $\mathbf{g}^{n+1}$ becasue we have not advanced the positions after computing the new-time gravity. This has the benefit that we perform only one gravity solve per timestep (in a single-level calculation with no hydro) because the particles are only moved once.

6.3.2.2 Computing g

We solve for the gravitational vector as follows:

- Assign the mass of the particles onto the grid in the form of density, ρ_{DM}. The mass of each particle is assumed to be uniformly distributed over a cube of side Δx, centered at what we call the position of the particle. We distribute the mass of each particle to the cells on the grid in proportion to the volume of the intersection of each cell with the particle's cube. We then divide these cell values by Δx^3 so that the right hand side of the Poisson solve will be in units of density rather than mass. Note that this is the *comoving* density.
- Solve $\nabla^2 \phi = \frac{4\pi G}{a} \rho_{DM}$. We discretize with the standard 7-point Laplacian (5-point in 2D) and use multigrid with Gauss-Seidel red-black relaxation to solve the equation for ϕ at cell centers.
- Compute the normal component of $\mathbf{g} = -\nabla \phi$ at cell faces by differencing the adjacent values of ϕ, e.g. if $\mathbf{g} = (g_x, g_y, g_z)$, then we define g_x on cell faces with a normal in the x-direction by computing $g_{x,i-1/2,j,k} = -(\phi_{i,j,k} - \phi_{i-1,j,k})/\Delta x$.
- Interpolate each component of $\mathbf{g}$ from normal cell faces onto each particle position using linear interpolation in the normal direction.

6.4 Output Format

6.4.1 Checkpoint Files

The particle positions and velocities are stored in a binary file in each checkpoint directory. This format is designed for being read by the code at restart rather than for diagnostics.

We note that the value of a is also written in each checkpoint directory, in a separate ASCII file called *comoving_a*, containing only the single value.

6.4.2 Plot Files

If **particles.write_in_plotfile** = 1 in the inputs file then the particle positions and velocities will be written in a binary file in each plotfile directory.

In addition, we can also visualize the particle locations as represented on the grid. There are two "derived quantities" which represent the particles. Setting

amr.derive_plot_vars = particle_count particle_mass_density
amr.plot_vars = NONE

in the inputs file will generate plotfiles with only two variables. **particle_count** represents the number of particles in a grid cell; **particle_mass_density** is the density on the grid resulting from the particles.

We note that the value of a is also written in each plotfile directory, in a separate ASCII file called *comoving_a*, containing only the single value.

6.4.3 ASCII Particle Files

To generate an ASCII file containing the particle positions and velocities, one needs to restart from a checkpoint file but doesn't need to run any steps. For example, if chk00350 exists, then one can set:

amr.restart = chk00350
max_step = 350
particles.particle_output_file = *particle_output*

which would tell the code to restart from chk00350, not to take any further time steps, and to write an ASCII-format file called *particle_output*.

This file has the same format as the ASCII input file:

number of particles
x y z mass xdot ydot zdot

6.4.4 Run-time Data Logs

If you set

amr.data_log = *log_file*

in the inputs file, then at run-time the code will write a log file with entries every coarse grid time step, containing

nstep time dt redshift a

6.4.5 Run-time Screen Output

There are a number of flags that control the verbosity written to the screen at run-time. These are:
amr.v
nyx.v
gravity.v
mg.v
particles.v

These control printing about the state of the calculation (time, value of a, etc) as well as timing information.

Chapter 7

Visualization

The BoxLib format in which NYX output is written can be read by **amrvis**, **VisIt**, and **yt**.

7.1 amrvis

We have a homegrown visualization tool called amrvis. We encourage you to build the amrvis3d executable, and to try using it to visualize your data. A very useful feature is View/Dataset, which allows you to actually view the numbers – this can be handy for debugging. You can modify how many levels of data you want to see, whether you want to see the grid boxes or not, what palette you use, etc.

If you like to have amrvis display a certain variable, at a certain scale, when you first bring up each plotfile (you can always change it once the amrvis window is open), you can modify the amrvis.defaults file in your directory to have amrvis default to these settings every time you run it.

7.2 VisIt

VisIt is also a great visualization tool, and it directly handles our plotfile format (which it calls amrex).

See **http://visit.llnl.gov**

To use the Boxlib3D plugin, select it from File → Open file → Open file as type Boxlib, and then the key is to read the Header file, plt00000/Header, for example, rather than telling to to read plt00000.

7.3 yt

yt also handles BoxLib format and is a great visualization tool for Nyx output.

Here are quick instructions from Matthew Turk:

The directories require that the inputs file be one level up, so that the hierarchy of files looks something like:

data/
data/inputs
data/plt00001
data/plt00002

To load the data in **yt**, you would then do:
from yt.mods import *
pf = load("data/plt00001")

You can also be in the data/ directory and just load plt00001.

See **http://yt.enzotools.org** to download **yt** and for more information.

7.4 Controlling What's in the PlotFile

amr.plot_vars =

and

amr.derive_plot_vars =

are used to control which variables are included in the plotfiles. The default for **amr.plot_vars** is all of the state variables. The default for **amr.derive_plot_vars** is none of the derived variables. So if you include neither of these lines then the plotfile will contain all of the state variables and none of the derived variables.

If you want all of the state variables plus entropy and pressure (both derived quantities), for example, then set

amr.derive_plot_vars = entropy pressure

If you just want density (state variable) and pressure (derived quantity), for example, then set

amr.plot_vars = density

amr.derive_plot_vars = pressure

Recall that we can also control whether the particles are written into a separate file in the plotfile directory by setting

particles.write_in_plotfile = 1

Chapter 8

Post-processing

Nyx interfaces with two post-processing suites, Reeber and Gimlet.

8.1 Reeber

Reeber uses topological methods to construct merge trees of scalar fields. These trees are effectively parameter-independent and contain a complete description of the field topology. In the context of Nyx, the field of interest is the dark matter density. Nyx then queries the merge tree with user-defined runtime parameters in order to locate the volume-averaged center of dark matter halos. The same tree can be queried with any number of such parameters to find halos with different mass/density thresholds.

8.2 Gimlet

Gimlet computes a variety of quantities about the simulation, including optical depths, Lyman-alpha fluxes, power spectra (both 1-D "line-of-sight" as well as fully 3-D), and probability distribution functions. These suites are fully MPI-parallel and can be run either "in situ" or "in-transit," or with a combination of both. A detailed description of their usage is provided in the Nyx User Guide.

8.3 Usage

Nyx can post-process with Gimlet alone, with Reeber alone, or with both simultaneously. To compile with Gimlet, add `GIMLET = TRUE` to the `GNUmakefile`; to compile with Reeber, add `REEBER = TRUE`. Note that these codes are in separate repositories and are not included with Nyx.

Nyx and AMReX provide the capability for the user to execute an arbitrary post-processing workflow either *in situ* or in-transit. An *in situ* workflow is one in which all MPI processes evolving the simulation stop at specified time steps and perform the post-processing before continuing with the simulation. In-transit means that AMReX creates a disjoint group of MPI processes ("sidecars") from the global pool and reserves them exclusively for post-processing. At specified time steps the group of processes evolving the simulation will send the necessary data to the sidecar group, and then will continue with the simulation. The two groups then work independently of one another on their particular tasks.

To run the post-processing workflow *in situ*, one sets the `nSidecars` parameter in the inputs file to `0`. To run the workflow in-transit, one sets `nSidecars > 0`. Note that the sum of all MPI processes is constant for the duration of the simulation, so whatever number the user dedicates to post-processing will be subtracted from the number doing the simulation itself.