User Manual for *gevolution* v1.0

Julian Adamek

April 21, 2016

Contents

1 Preface

What *gevolution* is and what it isn't. *gevolution* is a particle-mesh (PM) N-body code which is specifically designed to provide a general relativistic (GR) treatment of gravity. In order to allow for large simulations, the code uses massive parallelization based on the MPI protocol. However, very little knowledge about parallel computing is required for the user, as the entire overhead is hidden inside the *LATfield2* library [1]. *LATfield2* provides a framework for managing data, in particular *Fields* on a three dimensional regular *Lattice*. The data is automatically distributed over MPI processes using a rod-decomposition of the three-dimensional domain, i.e. two of the three dimensions are scattered over a two-dimensional process grid. *LATfield2* also has a highly scalable implementation of the three-dimensional Fast Fourier Transform (FFT). In addition, a versatile *Particle* handler is also provided.

As a consequence of relying on *LATfield2* for all the parallelization and data handling, *gevolution* is a relatively lightweight code which should be easy to read, understand, and eventually modify. The relativistic formulation and treatment of the laws of physics allows for the consistent implementation of many theoretical models beyond the baseline ΛCDM cosmological model which comes with this release version of the code. *gevolution* is intended to be a versatile tool for cosmologists who want to explore the consequences of extensions or alternatives to ΛCDM, but it can also be used to address certain questions within the ΛCDM model. However, it is *not* intended as a full replacement of existing "traditional" N-body codes (such as Gadget [2, 3] or RAMSES [4]): the applications (i.e. the phenomena which can be studied) have some overlap, but they also cover different terrain.

There are two main points to mention in this respect. Firstly, *gevolution* works at fixed (comoving) spatial resolution which is owed to its *LATfield2* heritage. Other codes often use either a tree algorithm (e.g. Gadget) or adaptive mesh refinement (e.g. RAMSES) in order to resolve small scale structure inside dense regions. At present, *gevolution* has no means to achieve this. Adaptive mesh refinement could be implemented in the future, but it is not clear that this is the most relevant path of development. Secondly, no baryonic physics (hydrodynamics) is implemented in this first version of *gevolution*. Again, this domain is left to traditional N-body codes. We want to provide a simple tool for studies of new phenomena mainly related to relativistic physics. Once such phenomena are understood separately one can iteratively increase the level of realism by including additional effects of known origin.

Additional information. This user manual provides essential information about the usage of the code (user interface) and its internal structure. More information about the *LATfield2* framework can be found in [1] or on the *LATfield2* web site, `http://latfield.org`. More information about the theoretical underpinning of *gevolution* can be found in the main code paper [5]. If you use *gevolution* for scientific work, we kindly ask you to cite [6] in your publications.

2 Compilation

2.1 Requirements

A copy of the *gevolution* code can be cloned from a public *Git* repository:
`https://github.com/gevolution-code/gevolution-1.0.git`
The code is written in C++ and requires the following additional libraries to be installed.

- *LATfield2* version 1.1 (`https://github.com/daverio/LATfield2.git`)
- FFTW version 3
- GNU Scientific Library (GSL) and the C-language Basic Linear Algebra Subprograms (CBLAS)
- HDF5 (use the parallel version for production runs)

Note also that the code does not support serial execution – it always requires the MPI protocol to run.

2.2 Makefile options

Before calling `make`, you should set the *include* and *library* paths in the *makefile* to point to your installation of *LATfield2* etc., or you should add those paths to the corresponding environmental variables.

Some features of the code are toggled using compiler options.

`-DBENCHMARK` With this option set, the code will output some benchmarking information (CPU time spent on various code components).

`-DPHINONLINEAR` With this option set, the equations will be solved taking into account shortwave corrections. In fact, this option should always be used unless you have a model which sources relativistic corrections at linear order and you know what you are doing.

`-DCHECK_B` With this option set, the code will compute a separate copy of the spin-1 component of the metric, B_i, using the momentum constraint each time an output is written. This can be used to check consistency but it requires additional memory. The option should therefore primarily be used for diagnostic purposes, especially if one is memory limited. Note that you also have to set this option if you want to compute cross-spectra of non-CDM species, as this feature uses the additional memory to store temporary data.

`-DH5_HAVE_PARALLEL` This option enables the parallel output of HDF5 files in the *LATfield2* library. It requires the parallel version of the HDF5 library to be installed on the system. The parallel output mode should always be used if available, especially for production runs.

`-DEXTERNAL_IO` This option enables the I/O server of the *LATfield2* library. Use this capability if you have to write large amounts of data (many snapshots) and only if you know what it means.

In addition, the compiler options `-DFFT3D` and `-DHDF5`, which switch on the corresponding features of *LATfield2*, are mandatory settings for *gevolution*.

Please note that the *LATfield2* library is developed to a large extent independently of *gevolution*. If you encounter problems related to *LATfield2*, in particular if you discover some bugs in the library, please contact the developers of *LATfield2* directly, developers@latfield.org.

3 Running *gevolution*

A typical command to run a simulation looks like this:

```
mpirun -np 16 ./gevolution -n 4 -m 4 -s settings.ini
```

The command-line parameters `-n` and `-m` specify the two-dimensional process grid which should be used by the *LATfield2* library for the decomposition of the three-dimensional domain. The product of the two numbers gives the total number of processes. Note that each number has to be ≥ 2, such that the minimum number of MPI processes is 4. Finally, a settings file has to be passed with the command-line parameter `-s`.

If the code is compiled with the `EXTERNAL_IO` flag, two additional command-line options, `-i` and `-g`, specify the number of I/O processes and the mapping of compute processes onto I/O processes (grouping), respectively.

3.1 Simulation settings

The settings file is a simple ASCII file which specifies simulation parameters in the format

<parameter name> = *<value>* [, *<value2>*, . . .]

The parser ignores empty lines and anything following a `#`-symbol. Note that parameter names are case sensitive and that white spaces within parameter names are allowed – the parameter name extends from the first non-whitespace character to the last non-whitespace character before the equal sign. In fact, the parser is written as to conform with the syntax of *CLASS* [7], see section 3.2 for more details.

`IC generator = basic` Selects the IC generator to be called at initialization. For efficiency reasons, *gevolution* generates initial data "on the fly" which is much faster than reading them from disk. The IC generator which comes with this version of the code is called *basic* because it uses linear theory without much sophistication. Other IC generators (e.g. written by users) should have their own unique name and should prompt the parser to read additional settings if appropriate.

`template file = sc1_crystal.dat` Specifies a file (in *Gadget-2* format) containing a homogeneous particle template. In the simplest case this can be a regular lattice of particles (a "crystal"), but one could also use a "glassy" template, for instance.

There are four template files already available with this version of the code, corresponding to four different crystal structures. `bcc_crystal.dat` and `fcc_crystal.dat` contain a body-centered cubic and face-centered cubic configuration, respectively, with 16 and 32 particles in the template. `sc0_crystal.dat` and `sc1_crystal.dat` contain a simple cubic template with 64 particles ($4\times4\times4$) each. In the first file, a particle sits at each corner of the template, while in the second file the corners are centered between eight particles. This allows one to choose how the particle crystal lattice is aligned with the lattice of the simulation.

`tiling factor = 16` Specifies the number of times the particle template shall be repeated in each direction in order to fill the entire simulation volume. Note that this parameter thereby sets the total particle number, as $N_{\text{part}} = N_{\text{template}} \times (\texttt{tiling factor})^3$.

`N_ncdm = 2` Specifies the number of distinct non-CDM species to be included in the simulation. An additional N-body ensemble is created for each species. If left unspecified, the number of species is inferred from the number of mass parameters (see `m_ncdm`).

`PSD samples = 2, 2` Specifies for each non-CDM species the number of samples to be drawn from the *phase space distribution* at each location where particles are placed. In effect this means that the number of particles for each non-CDM species is multiplied by `PSD samples`. For broad phase space distributions a large number of samples would be desirable in order to suppress shot noise, however, the simulation quickly becomes very expensive as this number increases.

`mPk file = pk-z100.dat` Specifies a file (ASCII format) containing a tabulated matter power spectrum at initial redshift. The IC generator assumes that the table is in the format produced by *CLASS*. Make sure that the table covers all the scales present in the simulation, otherwise the IC generator will crash.

Note that, although *gevolution* works in Poisson gauge, the input matter power spectrum should be provided in synchronous gauge!

`seed = 12345` Seed for random number generator. The IC generator will generate the Fourier modes starting from the lowest k value and then moving systematically towards higher k. This guarantees that two simulations with identical `seed` and `boxsize`, but with different resolution (i.e. different `Ngrid`), will contain the same realization for all modes which they have in common, allowing for resolution studies which are minimally affected by cosmic variance.

`correct displacement = yes` With this option set, the IC generator will try to fold the template pattern into the convolution kernel used for the displacement field. Usually this will produce a more accurate density field at small scales. However, this option should only be used with regular templates (crystals).

`k-domain = sphere` Possible choices are `sphere` or `cube`. With the former option, the IC generator will only generate modes within the largest k-sphere which fits into the Fourier cube – the remaining modes will be zero.

`initial redshift = 100` Specifies the redshift at initialization.

`relaxation redshift = 100` If the code requires some time for transients to decay and the non-Newtonian quantities to become valid, this parameter sets the redshift at which the full dynamics set in. For this version of the code this simply means that for $z >$ `relaxation redshift` the frame dragging force on the particles is ignored. In ΛCDM no relaxation is usually required and the number can safely be taken $\geq$ the initial redshift. If left unspecified, it will be set to the initial redshift.

`boxsize = 320.0` Specifies the comoving size of the simulation box (in units of Mpc/h).

`Ngrid = 64` Specifies the number of lattice points in each dimension. Therefore the (comoving) spatial resolution is given by Δx = `boxsize` / `Ngrid`.

`Courant factor = 48.0` This is the Courant factor for the gravity solver and sets the global time resolution of the simulation. If Δx is the (comoving) spatial resolution and $\Delta\tau$ is the (conformal) time step, the `Courant factor` is given by $\Delta\tau/\Delta x$. In other words, it is the number of grid units which a light signal can travel during one time step. Note that CDM particles (as opposed to non-CDM species) do not have a separate Courant criterion at present. However, if the `Courant` `factor` remains $\lesssim$ 100, the CDM particles will typically not move by more than one grid unit each time step (as their velocity is typically no more than 1% of the speed of light).

`move limit = 25` This specifies the maximum number of grid units a non-CDM particle is allowed to move in one update. If necessary, the global time step is subdivided into several particle updates in order to fulfill this criterion. Note, however, that the gravity solver remains at the time resolution given by the `Courant factor`, which means that the metric is not necessarily evolved for each particle update.

`time step limit = 0.04` Specifies the maximum admissible value for $\Delta\tau$ in units of the current Hubble time. This overrides the `Courant factor` if necessary, for instance to make sure that the background itself is evolved with sufficient accuracy.

`gravity theory = GR` Possible choices are `GR` or `Newton`. The default should of course be `GR`, but one can run a Newtonian simulation by switching to the other option. In this case the equations solved are the Newtonian ones, but the code still computes relativistic quantities for the output.

`output path = output/` Specifies the path where the output should be written. Note that the directory has to be created before running the code, as the code can not create directories and will crash at output if the directories do not exist!

`generic file base = lcdm` Specifies a file base for all files which are neither snapshots nor spectra (e.g. diagnostic files, background data etc.).

`snapshot file base = lcdm_snap` Specifies a file base for snapshot files.

`Pk file base = lcdm_pk` Specifies a file base for power spectra files.

`Pk bins = 1024` Specifies the number of *k*-bins for the spectra. Note that empty bins will not be written, therefore the final number of entries in the file may be less than this number.

`snapshot redshifts = 30, 10, 3, 0` Specifies the (approximate) redshifts at which a snapshot should be saved. Note that the code does not adapt the time step to reach these redshifts precisely – it will rather write the snapshots each time a given redshift has been passed.

`snapshot outputs = phi, B, chi, Gadget2` Specifies the datasets to be written at snapshot output. Possible choices are Φ (`phi`), $\chi = \Phi\text{-}\Psi$ (`chi`), B_i (`B`), h_ij (`hij`), T^0_0 (`T00`), $\mathrm{T}^\mathrm{i}_\mathrm{j}$ (`Tij`), momentum density (`p`) and particles (`pcls`). Furthermore, some Newtonian quantities can also be computed separately from the particle ensemble, namely δ_N (`deltaN`) and ψ_N (`psiN`). All files will be in HDF5 format. However, the code can also output the particle snapshot using the binary format of *Gadget-2* (`Gadget2`). This can be useful if one wants to process the data with tools that where designed for *Gadget*.

Note that the code will only write snapshots for the total T^0_0, $\mathrm{T}^\mathrm{i}_\mathrm{j}$ and momentum density. However, it should be easy to modify the output routine in order to write these quantities for individual constituents separately, as is done e.g. for the power spectra.

`tracer factor = 1, 1, 1` For each particle species (the first number is for CDM particles, the following ones for non-CDM species if appropriate) only particles having IDs which are integer multiples of this number will be saved into the *Gadget* file. For instance, with `tracer factor = 8` only one out of 8 CDM particles will be written. This allows for writing low-resolution snapshots of high-resolution runs. Of course, this type of data compression is lossy! The default is 1, meaning that all particles are saved.

If the particle template is a crystal only certain choices of `tracer factor` will result in a crystal structure for the *tracer particles*. These are 1, 2, 8, 16, 64 for the simple cubic templates, 1, 4, 8, 32 for the face-centered cubic one, and 1, 2, 4, 16 for the body-centered cubic one. For the non-CDM species one has to take into account also the `PSD samples` parameter which multiplies the above numbers.

`Pk redshifts = 50, 30, 10, 3, 1, 0` Specifies the (approximate) redshifts at which power spectra should be computed and saved (in the same fashion as `snapshot redshifts` works for snapshots).

`Pk outputs = phi, B, chi, hij` Specifies the datasets for which spectra should be computed. Possible choices are as in `snapshot outputs`, currently without the options `Tij`, `p`, `pcls`, and `Gadget2`. However, for the choices `T00` and `deltaN`, the code will compute not only the total power spectra but also the power spectra for each individual species. Moreover, if several non-CDM species are present, the code will also compute the cross-spectra between them (this feature requires the compiler option `CHECK_B` to be set since it uses some additional memory provided through this setting).

Cosmological parameters. We adopt the convention that all physical parameters which have a meaning in both *gevolution* and *CLASS* should be specified in the same way, using even the same parameter name in the respective settings files. Ideally it should therefore be possible to run both codes with a single settings file as each code will ignore any unknown parameters. This can facilitate the generation of initial conditions for *gevolution* using *CLASS*.

`h = 0.67556` The Hubble parameter (in units of 100 km/s/Mpc) never occurs in the equations solved by *gevolution* and its value is therefore arbitrary from the point of view of the code. For instance, distances are always given in units of Mpc/h. However, depending on how certain cosmological parameters are specified in the settings file, h may be used for initial unit conversion. If omitted, the code assumes the default value 0.67556.

`omega_cdm = 0.12038` The density parameter of CDM can be given either as Ω_{cdm} (`Omega_cdm`) or ω_{cdm} (`omega_cdm`). In the latter case the code computes Ω_{cdm} using the given (or default) value of h.

`omega_b = 0.022032` The density parameter of baryons can be given either as Ω_{b} (`Omega_b`) or ω_{b} (`omega_b`). In the latter case the code computes Ω_{b} using the given (or default) value of h. Note that this version of *gevolution* does not have a special treatment for baryons. Instead, they are simply lumped together with CDM, i.e. the density parameter for the “cold” particle species in the code is given by $\Omega_{\mathrm{cdm}} + \Omega_{\mathrm{b}}$.

`T_cmb = 2.7255` The density parameter of photons can be given in three different ways. Either through the CMB temperature (`T_cmb`, in units of K), in which case the code uses Planck’s law to compute ω_γ. Or one specifies Ω_γ (`Omega_g`) or ω_γ (`omega_g`). In the end, ω_γ will again be converted to Ω_γ using the given (or default) value of h. Note that the code will assume $\Omega_\gamma = 0$ if the density of photons is left unspecified!

`N_ur = 3.046` The density parameter of additional ultra-relativistic species can be specified in three different ways. Either the number of additional relativistic species is given, N_{ur} (or N_{eff} if this notation is preferred). In this case the code assumes the standard neutrino scenario and computes $\Omega_{\mathrm{ur}} = N_{\mathrm{ur}} \times \frac{7}{8} \times \left(\frac{4}{11}\right)^{4/3} \times \Omega_\gamma$. Alternatively, Ω_{ur} (`Omega_ur`) or ω_{ur} (`omega_ur`) can be specified (the latter is converted to Ω_{ur} using h). If left unspecified, the code will assume the standard value N_{ur} = 3.046.

`m_ncdm = 0.04, 0.04` This parameter specifies the fundamental masses (in units of eV) of additional non-CDM species. The code will create a full N-body ensemble for each massive non-CDM species. By default the initial phase-space distribution is assumed to be the relativistic Fermi-Dirac distribution of a standard neutrino species at the given mass. The sampling of the phase space can be set for each species separately using the `PSD samples` parameter (the default is 1).

`deg_ncdm = 1.0, 1.0` Specifies the degeneracy parameters for the non-CDM species. In practice these numbers, together with the fundamental masses, determine the density parameters for the non-CDM species following the standard formula $\omega_{\mathrm{ncdm}}^{(i)} = \texttt{m\_ncdm}^{(i)} \times \texttt{deg\_ncdm}^{(i)}$ / 93.14 (eV).

`T_ncdm = 0.71611, 0.71611` Specifies the temperature parameter (in units of the photon temperature) for each non-CDM species. This parameter appears in the initial phase-space distribution. If left unspecified the default value for neutrino cosmology is assumed, 0.71611.

Note that neither Ω_K nor Ω_Λ can be specified. Instead, $\Omega_K = 0$ is always assumed and Ω_Λ is then determined from all the other density parameters, $\Omega_\Lambda = 1 - \Omega_{\mathrm{cdm}} - \Omega_{\mathrm{b}} - \Omega_\gamma - \Omega_{\mathrm{ur}} - \Omega_{\mathrm{ncdm}}$.

3.2 Initial data

The initial data required to set up a simulation ultimately depends upon the model one wants to simulate. Within the generator provided with this version (`basic`) one has the option to simulate ΛCDM standard cosmology and its extension including massive neutrinos. The `basic` generator will assume that the initial conditions (ICs) are linear, adiabatic, and Gaussian. Therefore it is enough to specify a total matter power spectrum (at initial redshift) in order to generate a random realization of initial data. *gevolution* accepts a tabulated matter power spectrum in the format provided by *CLASS* (the respective parameter file entry in *CLASS* is `output = mPk`; you should also set `z_pk` to generate a power spectrum at the desired initial redshift for the N-body simulation, and make sure that `P_k_max_h/Mpc` is set to a value large enough to cover the full range of scales present in the simulation). Note that the `basic` generator assumes that the matter power spectrum is in synchronous gauge (and since *gevolution* works in Poisson gauge, the `basic` generator takes care of the appropriate gauge transformation)!

It should be stressed that the `basic` generator is suitable only for simple studies as it uses many approximations, some of which could easily be refined. Interfacing other IC generators with *gevolution* should not be difficult. An experimental interface exists for the open source generator *FalconIC* [8], and users will eventually implement their own routines according to the models which they want to study.

In order to relieve the user from familiarizing themselves with yet another system of units we adopt the convention that physical parameters should be specified exactly as in *CLASS*. Furthermore, the format of the settings file is such that one can write a single settings file which can be parsed by both *gevolution* and *CLASS* at the same time. Physical parameters which have a meaning for both codes should also have the same unique parameter name in the settings. *gevolution* should accept the initial data in the format which is provided by *CLASS*. Note, however, that the IC generator will convert the initial data finally to code units. We made an effort to have a "natural" system of units. Many quantities are dimensionless within the framework; an exception are distance and time, which are both measured in units of the comoving box size (i.e. we set $c = 1$). See section 4.2 for more details.

3.3 Output formats

There are essentially three types of output generated by the code: background data, power spectra and snapshots.

Background data is a table of time-dependent quantities stored in ASCII format. One line is written in each cycle of the time integration scheme. By default, the colums are *cycle number*, *conformal age of the Universe* (in units of the boxsize), *scale factor* (normalized to 1 today), *conformal Hubble rate* (in units of H_0), $\bar{\Phi}$ (the homogeneous mode of the metric perturbation Φ), and $a^3[\bar{T}^0_0]_{\text{N-body}}$ (the homogeneous mode of T^0_0 of the entire particle ensemble, scaled to today's value and in units of the critical density today). The latter two quantities are useful for checking that the assumed background model is accurate. In particular, $|\bar{\Phi}| \ll 1$ is required for internal consistency.

Power spectra are saved as tables of binned data in ASCII format. A separate file is written for each spectrum at each requested output redshift (parameter `Pk redshifts`). The first two columns contain the central values of k (in units of h/Mpc) and *P(k)* (dimensionless[1]) for each bin, the next two columns give the sample standard deviation for those, and the final column gives the number of samples which contributed to the bin. One should be aware that the spectra are estimated simply by averaging the amplitude of k-modes within a certain finite shell on the discrete Fourier lattice, without any measures to control discretization effects. The spectra therefore become distorted near the Nyqvist frequency. For δ_N and T^0_0 the spectra are deconvolved with the cloud-in-cell kernel used for the particle-mesh projection.

Snapshots are representations of the full 3D information on a given equal-time hypersurface. The code uses the I/O routines provided by the *LATfield2* library, which writes HDF5 format. In addition, the code can write a snapshot of the particle configuration also in the binary format of *Gadget-2*.

[1] As defined in [6]. Note that we use the symmetric Fourier convention, such that our dimensionless *P(k)* corresponds to the Δ*(k)* of [9]

3.4 Hibernation, restarting a run

Interrupting a run (hibernation) and restarting it from a snapshot or restart-file is currently *not* supported by *gevolution*. This contributes to the lightweight character of the code and facilitates the implementation of new models. However, it should be clear that the lack of this feature can become a problem when running very large production simulations. We are therefore planning to add hibernation/restart capability to the code in the future.

4 Modifying *gevolution*

If you want to modify the code, you should first understand the basics of the *LATfield2* library – visit `http://latfield.org` for a complete documentation. The most important classes are:

`Lattice` This class encapsulates the geometric information of a structured lattice, including the way how the 3D domains are distributed over parallel processes. It knows, for instance, how many grid points there are in each dimension etc. In *LATfield2*, the first dimension (`dim=0`) is local, while the other two dimensions (`dim=1,2`) are scattered as in a rod-decomposition (see figure 1). Each process only holds the data contained within its designated domain, plus a so-called *halo*, a layer of buffer sites which belong to adjacent domains. The halo is necessary if you want to compute discrete derivatives at the boundary of a domain, for instance.

There are two ways to initialize a `Lattice`: either you specify the geometry, or you define a `Lattice` to be the Fourier image of an existing `Lattice` (the geometry follows automatically in this case).

`Site` This class essentially represents a grid coordinate and provides a simple way to navigate the 3D data. There are two related classes, `rKSite` and `cKSite`, which are used to navigate the Fourier lattices of a real-to-complex or complex-to-complex Fourier transform, respectively.

`Field` This class represents *data* living on a `Lattice`. It could be a scalar field (real or complex) or a vector/tensor field. In *gevolution* all fields live on the same `Lattice` (or its Fourier image).

Important: any time a `Field` is modified locally and this modification should become "effective" globally, you have to call `updateHalo` for that `Field`. This will do the necessary communication between processes holding adjacent domains such that the halo sites have valid values.

`PlanFFT` This class is one of the centerpieces of *LATfield2*. It connects a `Field` defined on a (configuration-space) `Lattice` with its Fourier image (living on the Fourier `Lattice`). A plan can be executed in either direction (`FFT_FORWARD` / `FFT_BACKWARD`), meaning that either the Fourier image or its counterpart will be computed by (inverse) FFT. The conventions for the FFT are exactly as

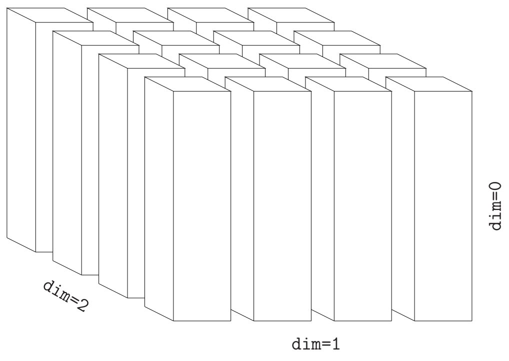

Figure 1: Sketch of the domain decomposition performed by *LATfield2* to distribute the work over the MPI processes. The data is stored in row-major order such that `dim=0` is *contiguous* and *local*, i.e. the non-contiguous dimensions `dim=1,2` are scattered over the process grid. This is essential for the FFT algorithm. In order to perform a 3D FFT, *LATfield2* first computes a real-to-complex FFT for `dim=0`; then it performs a transposition between `dim=0` and `dim=1`, making the latter one the local dimension; then a complex-to-complex FFT is computed for the now-local `dim=1`; a second transposition between `dim=2` and `dim=1` makes `dim=2` finally local, and another complex-to-complex FFT is performed; in a last step, *LATfield2* performs a transposition between the now-local `dim=2` and `dim=0` in order to make the latter one local again. Note that after these operations the ordering between `dim=1` and `dim=2` has been exchanged, but the `rKSite` class is aware of this. The number of *k*-space points in the local `dim=0` is `Ngrid`/2 + 1. The backwards FFT carries out the operations in reverse order.

in the *FFTW* library (in particular, the FFT is unnormalized, meaning that a forward and backward transform together multiply a field by N^3). Note that a FFT does not automatically update the *halo*, so make sure to call `updateHalo` if appropriate.

`Particles` This class handles N-body particles in *LATfield2*. The information about each particle is stored in a structure which contains the six phase-space coordinates (`pos[3]` and `vel[3]`, note that `vel[3]` is not a velocity vector in *gevolution*, but rather a canonical momentum). These coordinates are updated by two functions, `updateVel` and `moveParticles`. Each of these functions loops over the particles and executes an update through a custom procedure passed as function pointer.

Note that *gevolution* uses the derived class `Particles_gevolution` which additionally provides the I/O routine to write binary *Gadget-2* format.

LATfield2 instantiates an object called `parallel` which can be used to manage communication between MPI processes. This object is aware of the layout of the two-dimensional process grid used for the domain decomposition. Note that `dim=2` of the `Lattice` is scattered along `dim=0` of the process grid!

4.1 Navigating the source code

The source code of *gevolution* consists of several *header* files and a *main*. Subroutines and data structures are grouped in the header files according to purpose:

`background.hpp` This header contains auxiliary functions related to the background model. In particular, it provides a fourth-order Runge-Kutta integrator for the scale factor. If the background model is modified, this should be done in this header.

`gevolution.hpp` This header contains the algorithms needed for the evolution of the metric and the solution of the geodesic equations for the N-body particles. If additional equations of motion for new degrees of freedom need to be implemented, one may add them here (or one creates a separate header).

`ic_basic.hpp` This header contains the `basic` IC generator and some auxiliary functions related to IC generation. Every IC generator, e.g. added by users for whatever purpose, should have a separate header file containing its source code.

`metadata.hpp` This header defines some data structures which hold simulation parameters or settings related to IC generation. These data structures may have to be extended to include new parameters which arise when one implements additional physics.

`parser.hpp` This header contains the parser for the settings file. The parser translates the information given in the settings file into the format used by the code and fills the data structures defined in `metadata.hpp` accordingly. Therefore, if new parameters are added, the parser needs to be modified in order to recognize them.

`Particles_gevolution.hpp` This header defines the particle handler used by *gevolution*. It is derived from the `Particles` class provided by *LATfield2*, adding an I/O routine for *Gadget-2* binary format.

`tools.hpp` This header contains auxiliary functions and tools, e.g. algorithms to compute power spectra and generate formatted output.

Main control sequence. The main control sequence of *gevolution* is found in `main.cpp`. It is structured as follows:

- Initialization and parsing of settings file
- Generation of initial conditions "on the fly" (lines `263-281`)
- Main loop (line `324`)
  - Construction of stress-energy tensor of the N-body ensemble (lines `329-349`)

  - Output of *snapshots* (lines `357-593`)
  - Output of *power spectra* (lines `600-826`)
  - Update Φ (lines `853` and `864`, or line `877` for a *Newtonian* run; the backwards FFT is done in line `883` for both cases, followed by `updateHalo` in line `888`)
  - Output of *background data* (lines `890-907`)
  - Compute χ (lines `911` and `922`; the backwards FFT is done in line `929`, followed by `updateHalo` in line `934`)
  - Update B_i (line `924`; the backwards FFT is done in line `944` and `updateHalo` in line `949`) – note that a copy of $[a^2\ \mathrm{B}_\mathrm{i}]$ is kept in Fourier space in order for the solver to work (for all other fields, the Fourier image can usually be discarded after the backwards transform)
  - Update Ψ (lines `938-957`)
  - Determine time step subdivision for non-CDM species (lines `959-975`)
  - Update N-body particles (momenta and positions) using leap-frog (lines `997-1099`) – the background is also evolved in this code section (lines `1063` and `1098`)
  - τ += $\Delta\tau$ and determine $\Delta\tau$ for the next time step (lines `1111-1124`)
- The main loop terminates when no more output is scheduled (line `909`)
- Cleanup

The general procedure for adding new physics is the following. First, one needs to implement a way to handle the additional degrees of freedom – in many cases the `Field` class can be useful for this purpose. Then one needs a procedure to construct the stress-energy tensor for the configuration of the additional degrees of freedom. This new contribution to the stress-energy tensor needs to be added at an appropriate point in the main loop. The code will then proceed to evolve the metric according to the new stress-energy sources. Finally, one needs to implement an algorithm to solve the equations of motion for the new degrees of freedom. This will generally depend on the metric and should therefore be called after the metric is evolved, probably somewhere close to the particle update.

In addition, one will also have to write a new IC generator which includes the new degrees of freedom, and possibly some new output routines if additional quantities are to be studied.

4.2 Code units

The code works in natural units where $c = 1$. Both, conformal distances and conformal time are measured in units of the comoving box size (i.e. spatial coordinates range between 0 and 1). Metric perturbations are dimensionless. The canonical momentum is measured in units of the particle mass in order to make it dimensionless and independent of the mass resolution.

For the stress-energy tensor the code actually computes $a^3\ \mathrm{T}^0_0$ and $a^3\ \mathrm{T}^\mathrm{i}_\mathrm{j}$, i.e. the background expansion is scaled out. Also, the mass units are chosen such that, in the homogeneous limit and for nonrelativistic matter, the $a^3\ \bar{\mathrm{T}}^0_0$ is equal its density parameter Ω at redshift $z = 0$.

When output power spectra are computed, the code always writes the dimensionless spectra as defined in [6]. However, k is converted back to physical units, h/Mpc. Particle snapshots in *Gadget-2* format use length units of kpc/h and velocities in km/s. HDF5 output is produced in code units.

5 Patches

Minor bugs will be addressed with patches on the *master* branch of the repository. Apart from that, the features of each version of the code should remain stable. New features are implemented on a separate branch which may eventually become a new release version of the code.

Patch 1 (April 2016)

- Fixed minor bugs in `parser.hpp` and `ic_basic.hpp` which affected simulations with more than one particle species.
- Changed default batch size for writing particles in *Gadget-2* format from `1024` to `1048576`. This is an optimization parameter which can be manually set by adding `-DPCLBUFFER=`<*value*> as compiler option.
- The particle number in *Gadget-2* format is now interpreted as unsigned integer, allowing for files with up to 4 billion particles (the previous maximum file size was about 2 billion particles).
- Added `-std=c++11` as compiler flag in the `makefile`. This is required by the current version of the *LATfield2* library.

References

[1] D. Daverio, M. Hindmarsh, and N. Bevis, "Latfield2: A c++ library for classical lattice field theory," arXiv:1508.05610 [physics.comp-ph]. http://latfield.org.

[2] V. Springel, N. Yoshida, and S. D. M. White, "GADGET: A Code for collisionless and gasdynamical cosmological simulations," *New Astron.* **6** (2001) 79, arXiv:astro-ph/0003162 [astro-ph].

[3] V. Springel, "The Cosmological simulation code GADGET-2," *Mon. Not. Roy. Astron. Soc.* **364** (2005) 1105–1134, arXiv:astro-ph/0505010 [astro-ph].

[4] R. Teyssier, "Cosmological hydrodynamics with adaptive mesh refinement: a new high resolution code called ramses," *Astron. Astrophys.* **385** (2002) 337–364, arXiv:astro-ph/0111367 [astro-ph].

[5] J. Adamek, D. Daverio, R. Durrer, and M. Kunz, "gevolution: a cosmological N-body code based on General Relativity," arXiv:1604.06065 [astro-ph.CO].

[6] J. Adamek, D. Daverio, R. Durrer, and M. Kunz, "General relativity and cosmic structure formation," *Nature Phys.* **12** (2016) 346–349, arXiv:1509.01699 [astro-ph.CO].

[7] D. Blas, J. Lesgourgues, and T. Tram, "The Cosmic Linear Anisotropy Solving System (CLASS) II: Approximation schemes," *JCAP* **1107** (2011) 034, arXiv:1104.2933 [astro-ph.CO].

[8] W. Valkenburg and B. Hu, "Initial conditions for cosmological N-body simulations of the scalar sector of theories of Newtonian, Relativistic and Modified Gravity," *JCAP* **1509** no. 09, (2015) 054, arXiv:1505.05865 [astro-ph.CO]. http://falconb.org./.

[9] F. Bernardeau, S. Colombi, E. Gaztañaga, and R. Scoccimarro, "Large scale structure of the universe and cosmological perturbation theory," *Phys. Rept.* **367** (2002) 1–248, arXiv:astro-ph/0112551 [astro-ph].