User Manual for *gevolution* v1.2

Julian Adamek

August 9, 2022

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 parallelisation based on the MPI protocol. However, very little knowledge about parallel computing is required for usage, 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 parallelisation 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. Some extensions, such as massive neutrinos or a phenomenological dark energy model, are already included in 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 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 underpinnings of *gevolution* can be found in the main code paper [5], while certain aspects related to relativistic species are discussed in [6, 7]. If you use *gevolution* for scientific work, we kindly ask you to cite [8] in your publications.

Copyright © 2015–2022 Julian Adamek (Université de Genève & Observatoire de Paris & Queen Mary University of London & Universität Zürich)

License. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

The Software is provided "as is", without warranty of any kind, expressed or implied, including but not limited to the warranties of merchantability, fitness for a particular purpose and noninfringement. In no event shall the authors or copyright holders be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the Software or the use or other dealings in the Software.

1.1 Development since v1.1

The major addition in this code release is the new light-cone feature of *gevolution*. In addition to producing output for equal-time hypersurfaces one can now also specify any number of observers and survey geometries for which data can be recorded directly on the past light cone. Particle data is stored in a standard format similar to particle snapshots while metric data is pixelised using the *HEALPix* method [9]. There are also minor improvements to the *CLASS* interface, in particular a new option to include linear fluids. This can be useful, for instance, for studying the w_0-w_a-c_s^2 parametrisation of dark energy.

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-1.2.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.

`-DORIGINALMETRIC` In v1.1 of *gevolution* the convention for the scalar potentials ϕ and ψ **has changed at second order** compared to the original code release. The new convention is the one of [6]. With the option `-DORIGINALMETRIC` the original convention of [5] is restored.

`-DCHECK_B` With this option set, the code will compute a separate copy of the spin-1 component of the metric, B_i. This copy is computed using the alternative of the method chosen for the simulation, e.g. it will use the momentum constraint if that equation is *not* used in the simulation. This can be useful to check consistency but it requires additional memory and computations. The option should therefore primarily be used for diagnostic purposes.

`-DEXACT_OUTPUT_REDSHIFTS` While output can only be written at discrete time steps, this option employs a linear interpolation between the two nearest time steps in order to produce output at exactly the redshift that was requested. This only applies to power spectra and to *Gadget-2* binary particle snapshots – metric snapshots are still taken at approximate redshifts.

`-DVELOCITY` With this option set, the code computes a coarse-grained velocity field from which a divergence (θ) and curl (ω) part can be extracted and analysed, see [10] for details. The additional projections slightly increase the computational cost and memory footprint of a simulation.

`-DHAVE_CLASS` Use this option to link *CLASS* as a library. This allows computing linear transfer functions "on the fly" without having to pass them as input files to the IC generator.

`-DHAVE_HEALPIX` Use this option to link the *HEALPix* C-library required for the production of pixelised light-cone output.

`-DICGEN_PREVOLUTION` This option enables an additional IC generator called `prevolution` which can be used to set up – or "pre-evolve" – the phase space of non-cold species, e.g. massive neutrinos, according to a procedure proposed in [11], see section 2.4 of [7] for some details. Note that this generator needs to call *CLASS*.

`-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.

`-DCOLORTERMINAL` This option toggles colour highlighting for the terminal output. The ANSI escape codes are not properly interpreted by many text editors, however.

`-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 `HAVE_CLASS` flag, the user can pass a *CLASS* precision parameter file with the optional command-line parameter `-p`. If omitted, the code will work with default parameters when calling *CLASS*.

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* [12], see section 3.2 for more details.

`IC generator = basic` Selects the IC generator to be called at initialization. For efficiency reasons, *gevolution* can generate initial data "on the fly" which is much faster than reading them from disk. This version of the code comes with two different IC generators. The default choice is called `basic` because it uses linear theory without much sophistication. The other generator, called `prevolution`, is only relevant for setting up the initial phase space of non-cold species (if present). A third option is `read from disk`, which will prompt *gevolution* to read an initial snapshot from provided files. 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. If more than one particle species is present a comma-separated list specifies a separate file for each N-body ensemble. The templates are assigned in following order: CDM, baryons (if present as separate N-body ensemble), non-CDM species (e.g. massive neutrinos).

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 $N_{\rm part} = N_{\rm template} \times (\texttt{tiling factor})^3$. If more than one particle species is present a comma-separated list specifies a separate number for each N-body ensemble. The assignment is as in `template file`. Note that a value of zero is allowed for any non-CDM species (e.g. massive neutrinos), and no N-body ensemble is created in this case. Such species can then still be treated with the linear radiation module (see below).

`PSD samples` Since *gevolution* v1.1 this parameter is **obsolete**. The size of each N-body ensemble is entirely controlled with `tiling factor`.

`particle file = ic_cdm.h5` If initial data is to be read from disk instead of being generated on the fly (i.e. if `IC generator = read from disk`) a file containing the initial particle configuration has to be specified. If more than one particle species is present a comma-separated list specifies a separate file for each N-body ensemble. The assignment is as in `template file`.

Note that the particle phase space coordinates should be in Poisson gauge when running a relativistic simulation (i.e. `gravity theory = GR`)! When you are running a Newtonian simulation (i.e. `gravity theory = Newton`) the coordinates should be in N-body gauge instead [13].

`metric file = ic_phi.h5` If initial data is to be read from disk instead of being generated on the fly (i.e. if `IC generator = read from disk`) and when running a relativistic simulation (i.e. `gravity theory = GR`) a file containing the initial potential ϕ can be provided. The format is assumed to be *gevolution*'s HDF5 output format at the correct resolution. If left unspecified the code will derive the initial ϕ from the particle distribution assuming a linear approximation.

`baryon treatment = blend` Specifies the treatment of the baryonic component. Possible choices are `ignore`, `sample`, `blend` and `hybrid`. Choosing `ignore` means that the IC generator will simply ignore the baryon transfer functions (if present in the `Tk file`) and assume that baryons behave exactly as CDM. No separate N-body ensemble is created. With `sample` the baryons will be present as separate N-body ensemble with their own initial perturbations. The option `blend` prompts the IC generator to treat CDM and baryons as single species, however, the initial perturbations are given by a weighted average of the respective transfer functions. No separate N-body ensemble is created. Choosing `hybrid` amounts to a mixture of `sample` and `blend`. No separate N-body ensemble is created. The weighted average depends on particle IDs such that one out of eight particles behaves maximally as if it were a baryon particle while the remaining particles have an appropriately increased weight of the CDM behaviour.

`Tk file = class_tk.dat` Specifies a file (ASCII format) containing the tabulated linear transfer functions (for δ and θ) 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 the transfer functions should be specified in Newtonian gauge! Additionally the primordial power spectrum (of the gauge invariant curvature perturbation) needs to be specified by setting the parameters `A_s` and `n_s`.

`mPk file = class_pk.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. This option is provided as *alternative* for `Tk file` for backwards-compatibility. While being a bit simpler to use this method has several drawbacks: it does not allow to simulate several species with different initial perturbation amplitudes, and since no velocity power spectrum is specified the velocities are initialized on the Zel'dovich approximation for matter domination.

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

Link *CLASS* as a library: If neither `Tk file` nor `mPk file` are specified, and you compiled with `-DHAVE_CLASS`, the `basic` IC generator will simply run *CLASS* directly to compute the required transfer functions "on the fly."

`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 realisation 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 (for CDM and baryons only). Usually this will produce a more accurate density field at small scales. However, this option *requires* that the discrete symmetry group of the lattice is a symmetry subgroup of the homogeneous particle template (crystal), i.e. the template is invariant under lattice translations, rotations or reflections. If this requirement is not met the IC generator may crash or misbehave.

`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.

If one uses `prevolution` to generate initial data, the relaxation redshift should be taken somewhat *larger* than the initial redshift. This IC generator initializes the particle ensembles for non-cold species at a very high redshift (specified with a parameter called `prevolution redshift`) and evolves them in the linear realisation of the potentials (obtained from transfer functions that are computed at runtime using *CLASS*). When the redshift reaches the value specified by `relaxation redshift`, the IC generator will initialize all remaining particle species and evolves everything down to the "initial" redshift of the (non-linear) simulation. Between the relaxation redshift and the initial redshift, the potentials are interpolations between the linear solutions (obtained from *CLASS*) and the non-linear ones (obtained from *gevolution*), and the frame dragging is still neglected.

`prevolution redshift = 5000` This parameter is only relevant for the `prevolution` IC generator and sets the redshift at which the particle ensembles for non-cold species (e.g. massive neutrinos) are initialized. It is assumed that all non-cold species are ultra-relativistic at this redshift, and that their perturbations can be described by the first two moments of their phase space distributions.

`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 $\Delta x =$ `boxsize` / `Ngrid`. Due to the way the distributed FFT is implemented in *LATfield2*, `Ngrid` has to fulfill certain constraints with respect to the process layout that is given by the two command-line parameters `n` and `m`. These are `Ngrid` / `n` = *integer*, and `Ngrid` / `m` = *even integer*.

`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 and baryon 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 = 8.0` 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.

`vector method = parabolic` Possible choices are `parabolic` or `elliptic`. In the former case the vector perturbations of the metric are computed using the spin-1 projection of the space-space part of Einstein's equations, which gives a parabolic equation. This method is the default choice as it is computationally more efficient. However, if the source term varies rapidly (e.g. if relativistic degrees of freedom are present) this method can have convergence or stability issues. With the `elliptic` method, the code will compute the vector perturbations of the metric using the spin-1 projection of the momentum constraint which is elliptic but requires the computation of additional particle-to-mesh projections. Note that the makefile option `-DCHECK_B` will ensure that an additional copy of B_i is computed using the alternative method (this copy is computed for diagnostic purposes and is never used in the dynamics).

`radiation treatment = CLASS` Use this option to enable the linear radiation module based on *CLASS*. It uses the linear transfer functions from the Boltzmann code to generate a realisation of the perturbed radiation field [6]. This method can also be used to obtain a linear approximation to the effect of massive neutrinos similar to how it is done in [14]. Note that the linear realisation is going to be based on the random number sequence given by `seed`. It is therefore consistent with the initial data if it was prepared by one of the IC generators of *gevolution*.

`fluid treatment = CLASS` When running simulations with an additional cosmological fluid ($\Omega_{\rm fld} > 0$), use this option to enable a linear treatment of the fluid perturbations based on *CLASS*. This method works exactly like the linear radiation module, see [15] for more details. Note that *CLASS* is not designed to compute transfer functions beyond redshift zero, and *gevolution* therefore has to turn off the fluid perturbations at that point.[1]

`switch delta_rad = 15` If the linear radiation module is enabled, this parameter specifies a redshift below which the linear density perturbation of radiation (photons and other ultra-relativistic species) can safely be ignored.

`switch delta_ncdm = 15, 10` If the linear radiation module is enabled, a redshift value can be specified for each non-CDM particle species. Above that redshift the density field used in the metric evolution is going to be the linear approximation obtained from the radiation module. At lower redshift the code will compute the density according to the usual particle-mesh projection from the N-body ensemble of the species. If left unspecified, only the latter method is used.

`switch B ncdm = 15, 10` For each non-CDM particle species one can specify a redshift value below which it is safe to use the curl part of its N-body momentum field to source the frame-dragging potential. Above that redshift value the contribution of the non-CDM particle species to the vector constraint is ignored. This is useful in cases where the perturbations are still in the linear regime and the contribution would therefore be dominated by shot noise. If left unspecified, the redshift values of `switch delta_ncdm` are used.

`switch linear chi = 15` If the linear radiation module is enabled, this parameter specifies a redshift above which the linear approximation should be used for the contributions to χ of all ultra-relativistic and non-CDM species. At lower redshift the contributions of non-CDM species will be obtained by appropriate particle-mesh projections, whereas the contributions of ultra-relativistic species will be ignored. If left unspecified, it will be set to the highest value given in `switch delta_ncdm`.

`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 nor light cones nor hibernation points (e.g. diagnostic files, background data etc.).

[1] This can also cause spurious effects in power spectra when the code is compiled with `-DEXACT_OUTPUT_REDSHIFTS` and the interpolation uses values beyond redshift zero.

`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.

`lightcone file base = lcdm_lightcone` Specifies a file base for light-cone files.

`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 (exception: *Gadget-2* binaries when using the compilation flag `-DEXACT_OUTPUT_REDSHIFTS`).

`snapshot outputs = phi, B, chi, Gadget2` Specifies the datasets to be written at snapshot output. Possible choices are ϕ (`phi`), $\chi = \phi$-ψ (`chi`), B_i (`B`), h_{ij} (`hij`), T^0_0 (`T00`), T^i_j (`Tij`), momentum density (`p`), velocity (`v`, only available if the makefile option `-DVELOCITY` is used) and particles (`pcls`). Furthermore, some Newtonian quantities can also be computed separately from the particle ensemble, namely ρ_N (`rhoN`)[2] 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`, or `multi-Gadget2` if the snapshots should be split into multiple files – the number of files in this case will be equal to the command-line parameter `m` from the process layout). 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, T^i_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 second one for baryon particles if they are present, 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* files (including light-cone outputs). 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. A value of 0 indicates that the corresponding particle species should be omitted from the output entirely.

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.

`downgrade factor = 1` This parameter allows to save only downgraded snapshots of *Fields*. Each point in a downgraded snapshot corresponds to the average over (`downgrade factor`)^3 vertices of the lattice. Note that `Ngrid` and the layout of MPI processes have to be chosen in such a way that the downgrading can be performed on each domain independently. The `downgrade factor` is not applied to particle snapshots (use `tracer factor` instead).

`Pk redshifts = 50, 30, 10, 3, 1, 0` Specifies the (approximate, unless the optional compilation flag `-DEXACT_OUTPUT_REDSHIFTS` is set) 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 spectra related to densities there are two possible choices for the normalisation. The spectra of `T00` and `rhoN` are in units of proper density squared[3], whereas the choices `delta` and `deltaN` correspond to the respective dimensionless density contrasts, defined e.g. as $\delta = \delta T^0_0/\bar{T}^0_0$. For any of these choices, the code will compute not only the total power spectra but also the power spectra for each individual species. Moreover, if baryons and/or several non-CDM species are present, the code will compute some cross-spectra if the dataset `X-spectra` is specified[4]. If velocity spectra are requested the code also computes spectra for the divergence θ and

[2] With *gevolution* v1.1 the option δ_N (`deltaN`) is **obsolete** and has been replaced by ρ_N (`rhoN`). See, however, `Pk outputs`.

[3] Note that *gevolution* v1.0 used comoving densities instead, which are also still used in the snapshot format.

[4] Even without this option the code automatically computes the cross-spectra between following non-CDM species (if present): 0×1, 2×3 and 4×5. This default behaviour facilitates the implementation of shot noise suppression, see [5] for more details.

curl ω, both in units of Mpc^{-2}. The velocity field is derived from the total stress-energy tensor, and in empty regions is estimated using the *RESCALED* method detailed in [10].

`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.

`lightcone outputs = phi, Gadget2` Specifies the datasets to be written for light-cone output. Possible choices are *Gadget-2* (`Gadget2`) for particle data and ϕ (`phi`), $\chi = \phi$-ψ (`chi`), B_i (`B`) and h_{ij} (`hij`) for metric data. The latter require the *HEALPix* C-library to be linked (`-DHAVE_HEALPIX`).

Writing multiple light cones: In some cases it can be useful to define more than one light-cone geometry, e.g. to gather statistics for multiple observers or to mimic complicated survey footprints that consist of fields with different areas and depths. To specify parameters for individual light-cone geometries, the light cones can be enumerated by inserting a numeral after the word `lightcone`, e.g. `lightcone 0 outputs = ...`, `lightcone 1 outputs = ...`. Default settings for all light cones are set if the numeral is omitted.

`lightcone vertex = 0, 0, 0` Specifies the location of the observer in comoving coordinates (in units of Mpc/h).

`lightcone redshift = 0` Specifies the time of observation. If left unspecified, the default $z = 0$ is assumed, corresponding to observations taken today.

`lightcone distance = 20, 500` Specifies the minimum and maximum comoving look-back distance (in units of Mpc/h) for which light-cone data is to be recorded.

`lightcone opening half-angle = 20` Specifies the field of view in terms of a maximal angle (in degrees) with respect to a polar axis. If left unspecified, 180° (full sky) is assumed.

`lightcone direction = 0, 0, 1` Specifies the direction vector of the polar axis with respect to which the field of view is defined. Cartesian (x^1, x^2, x^3) or polar coordinates (ϑ, φ) are both acceptable, and in the latter case the two angles are given in degrees. If left unspecified, the direction vector is aligned with the x^3-axis. Angular coordinates in the *HEALPix* basis are related to the Cartesian basis of the simulation box by an Euler rotation of type x^3-x^2-x^3, where the last Euler angle is assumed to be zero.

`lightcone Nside = 4, 1024` Specifies the minimum and maximum values of $N_{\rm side}$ that determine the angular resolution of pixelised data. If only one value is given, $N_{\rm side}$ is fixed to that value. Otherwise an appropriate value within the available range is chosen for each map, depending on the setting of the `pixel factor`.

`lightcone pixel factor = 0.5` Specifies the minimum inverse area of a pixel in terms of the comoving spatial resolution of the simulation. Explicitly, $N_{\rm side}$ is chosen as the smallest allowed value that satisfies (`pixel factor`) $\times A_{\rm pix} < \Delta x^2$, where $A_{\rm pix}$ is the area of a pixel for the given radius of a shell. I left unspecified, the value 0.5 is assumed which means that the ratio $A_{\rm pix}$ / Δx^2 lies between 0.5 and 2.

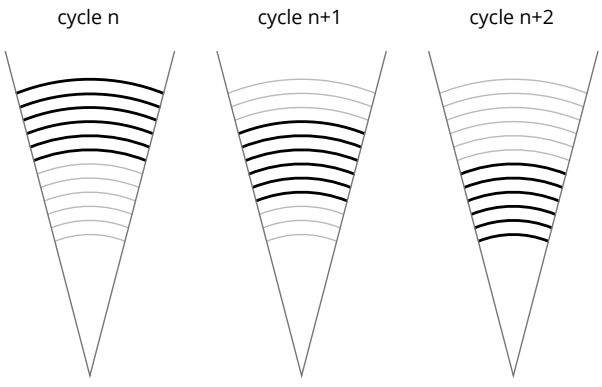

Figure 1: The sketch illustrates which shells (thick black) are used for pixelised light-cone data in consecutive time steps (cycles) of the simulation. In the present example, the `lightcone covering` is 2, which means that a shell will appear in two consecutive outputs – enough to compute first time-derivatives of the data.

`lightcone shell factor = 1.0` Specifies the sampling rate of pixelised data along the radial direction, i.e. the number of shells per look-back distance interval Δx. If left unspecified, the value 1.0 is assumed such that the sampling rate matches the spatial resolution of the simulation.

`lightcone covering = 2.0` This parameter controls the amount of *spatial* overlap between consecutive outputs of pixelised light-cone data, see figure 1 for illustration. A value of 1 corresponds to no overlap (each shell is written only once). If left unspecified a value slightly larger than 2 is used, which ensures that each shell is written (at least) for two consecutive time steps. The parameter has no implications for particle data.

`hibernation file base = lcdm_restart` Specifies a file base for hibernation files.

`hibernation redshifts = 10, 1` Specifies the (approximate) redshifts at which hibernation points should be written to disk (in the same fashion as `snapshot redshifts` works for snapshots).

`hibernation wallclock limit = 72000` Specifies the time (in seconds) after which the code should write a hibernation point and quit. This limit should be somewhat below the time limit of the job to allow for the hibernation to complete. If left unspecified, no time limit is assumed.

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. 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 $\Omega_{\rm cdm}$ (`Omega_cdm`) or $\omega_{\rm cdm}$ (`omega_cdm`). In the latter case the code computes $\Omega_{\rm cdm}$ using the given (or default) value of h.

`omega_b = 0.022032` The density parameter of baryons can be given either as $\Omega_{\rm b}$ (`Omega_b`) or $\omega_{\rm b}$ (`omega_b`). In the latter case the code computes $\Omega_{\rm b}$ using the given (or default) value of h. Note that unless one sets `baryon treatment = sample` the baryons will not be present as separate N-body ensemble and will be lumped together with CDM, i.e. the density parameter for the "cold" particle species in the code is given by $\Omega_{\rm cdm} + \Omega_{\rm b}$.

`omega_fld = 0` A cosmological fluid with parametrised equation of state can be added to the background equations by setting a non-zero value for its density parameter $\Omega_{\rm fld}$ (`Omega_fld`) or $\omega_{\rm fld}$ (`omega_fld`).

`w0_fld = -1.0` The equation of state of the additional cosmological fluid (if present) is parametrised as $w(\mathrm{a}) = w_0 + w_{\rm a}$ (1 - a), where w_0 (`w0_fld`) is the present-day value.

`wa_fld = 0` A time-dependence can be given to the equation of state of the additional cosmological fluid (if present) by setting a non-zero value of $w_{\rm a}$ (`wa_fld`) as indicated above.

`cs2_fld = 1` The (square of the) effective speed of sound of perturbations in the additional cosmological fluid (if present) can be specified. This parameter appears in the evolution equations of the fluid perturbations for which a linear solution can be obtained with *CLASS*. If the appropriate flag is set (`fluid treatment = CLASS`) such a linear contribution is included in the stress-energy used in the simulation.

`A_s = 2.215e-9` Initial amplitude of the primordial power spectrum of the gauge invariant curvature perturbation at pivot scale $k_{\rm p}$ (`k_pivot`). If left unspecified, the code will assume the standard value $A_{\rm s} = 2.215\times10^{-9}$.

`k_pivot = 0.05` Pivot scale in units of Mpc^{-1}. If left unspecified, the code will assume the standard value $k_{\rm p} = 0.05\ \mathrm{Mpc}^{-1}$.

`n_s = 0.9619` Scalar spectral index. If left unspecified, the code will assume the standard value n_s = 0.9619.

`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 Ω_γ = 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, $N_{\rm ur}$ (or $N_{\rm eff}$ if this notation is preferred). In this case the code assumes the standard neutrino scenario and computes $\Omega_{\rm ur} = N_{\rm ur} \times \frac{7}{8} \times \left(\frac{4}{11}\right)^{4/3} \times \Omega_\gamma$. Alternatively, $\Omega_{\rm ur}$ (`Omega_ur`) or $\omega_{\rm ur}$ (`omega_ur`) can be specified (the latter is converted to $\Omega_{\rm ur}$ using h). If left unspecified, the code will assume the standard value $N_{\rm ur}$ = 3.046.

`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 unless the respective `tiling factor` is set to zero. If left unspecified, the number of species is inferred from the number of mass parameters (see `m_ncdm`).

`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 `tiling factor` parameter.

`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 $\omega^{(\rm i)}_{\rm ncdm}$ = `m_ncdm`$^{(\rm i)}$ $\times$ `deg_ncdm`$^{(\rm i)}$ / 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 $\Omega_{\rm K}$ nor Ω_Λ can be specified. Instead, $\Omega_{\rm K}$ = 0 is always assumed and Ω_Λ is then determined from all the other density parameters, Ω_Λ = 1 - $\Omega_{\rm cdm}$ - $\Omega_{\rm b}$ - Ω_γ - $\Omega_{\rm ur}$ - $\Omega_{\rm ncdm}$ - $\Omega_{\rm fld}$.

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, as well as phenomenological dark energy models using the w_0-$w_{\rm a}$-$c_{\rm s}^2$ parametrisation. The `basic` generator will assume that the initial conditions (ICs) are linear, adiabatic, and Gaussian. Therefore it is enough to specify the primordial power spectrum and the linear transfer functions (at initial redshift) in order to generate a random realisation of initial data. *gevolution* accepts tabulated transfer functions in the format provided by *CLASS* (the respective parameter file entry in *CLASS* is `output = mTk, vTk`; 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 transfer functions are in Newtonian gauge (in *CLASS* set `gauge = Newtonian`)!

For simulations with non-cold species (in particular massive neutrinos) *gevolution* (since the release of v1.1) provides an alternative IC generator called `prevolution`. Interfacing other IC generators with *gevolution* should not be difficult. An experimental interface exists for the open source IC generator *FalconIC* [16], and users will eventually implement their own routines according to the models which they want to study.

In order to relieve the user from familiarising 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 four types of output generated by the code: background data, power spectra, snapshots and light cones.

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 columns are *cycle number*, *conformal age of the Universe* (in units of the boxsize), *scale factor* (normalised to 1 today), *conformal Hubble rate* (in units of H_0), $\bar{\phi}$ (the homogeneous mode of the metric perturbation ϕ), and $-3H_0^2 a^3 [\bar{\mathrm{T}}^0_0]_{\text{N-body}}/8\pi G$ (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[5] up to the units of the field in question) 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 discretisation effects. The spectra therefore become distorted near the Nyqvist frequency. For ρ_{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 snapshots of the particle configuration also in the binary format of *Gadget-2*. With the release of v1.2, *gevolution* can distribute those over multiple files, eliminating a limitation that was present for runs with more than 3.5×10^8 particles.

Light cones represent information on a *null* hypersurface that corresponds to the past light cone of a specified *observation event*. Output can be requested for any number of observation events, each specified by a *vertex* location (in comoving coordinates) and an observation time (usually taken to be redshift zero, but other choices are possible). In addition, a *survey geometry* can be specified in terms of a *distance interval* (minimum and maximum look-back distances away from the vertex) and a *solid angle* that could either be the full sky or a pencil beam. In the latter case, the survey area is further specified by a direction vector and an opening half-angle, i.e. the field of view with respect to the direction vector (see figure 2).

Two different output formats are supported. Particle positions and velocities can be recorded on the light cone using the *Gadget-2* binary format.[6] In each simulation time step a shell centered on the current look-back distance, with a radial extent corresponding to the current step size, is recorded for each light cone. Note that the positions and velocities are updated by fractional time steps in order to move them as close as possible to the null hypersurface. This eliminates artefacts of the time discretisation at the shell boundaries. Furthermore, the code keeps track of the IDs of particles that have been written in the previous time step, in order to guarantee that the shells can later be catenated without duplicating

[5] As defined in [8]. Note that we use the symmetric Fourier convention, such that our dimensionless *P(k)* corresponds to the $\Delta(k)$ of [17].

[6] Note that the velocities are *not* rescaled by the usual $\sqrt{a}$-factor since the scale factor a is not a constant on a light cone.

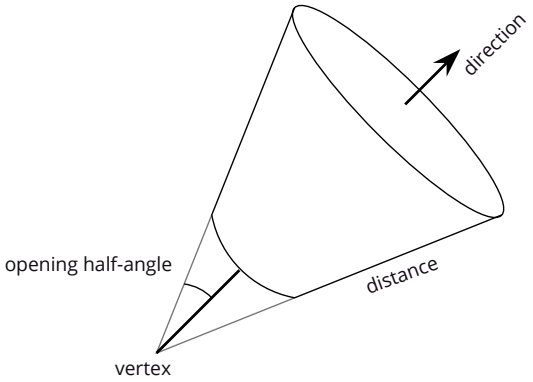

Figure 2: Illustration of a generic light-cone geometry, given by a vertex location and observation time, a look-back distance interval, and a field of view specified through a direction vector and an opening half-angle.

or missing any particles. A simple post-processing tool (*lccat*) is provided for this purpose. While *gevolution* can handle light-cone geometries that make use of the periodic boundary conditions, the user is responsible for ensuring that the light cones do not intersect with each other (or with themselves). If they do, particles may only appear in one of the duplicated regions.

In order to record metric fields on a light cone, each field is resampled on spherical shells that are pixelised using the *HEALPix* framework. A radial sampling rate (distance between shells) and a map resolution (either a fixed angular resolution, or an adaptive angular resolution that approximately maintains a fixed *spatial* resolution between shells of different radius) can be chosen. At each time step, a certain number of shells with radius in the vicinity of the current look-back distance is recorded. The number depends on a *covering* factor that effectively determines the finite time interval (in terms of simulation time steps) around the null hypersurface for which metric data should be recorded, e.g. in order to perform spacetime interpolations or take time-derivatives. A covering factor of one means that a given shell will be recorded once, at the simulation time step at which the look-back distance is closest to the shell radius. A covering factor of two (see figure 1) means that each shell will be recorded twice, i.e. for the two simulation time steps with the closest look-back distances, and so on.

For each metric component, the collection of shells belonging to one time step (cycle) is saved into a single file that uses a custom binary format that has been optimised for I/O performance. A simple post-processing tool (*lcmap*) is provided in order to do some pixel-based data reduction and produce the standard *FITS* format that is commonly used for *HEALPix* data.

A small *info* file is written for each light cone, summarising the geometric parameters and a per-cycle breakdown of the look-back distance intervals (in units of the boxsize) for which data are recorded.

3.4 Hibernation, restarting a run

Interrupting a run (hibernation) and restarting it from a snapshot or restart file is a feature supported since *gevolution* v1.1. When the code is prompted to write a hibernation point, it simply writes a full-resolution snapshot of all quantities that are needed to exactly recreate the current state of the simulation. In addition, a settings file (in the standard ASCII format) is created that contains the simulation settings and a few metadata items that can be parsed through the normal user interface of *gevolution*.

To restart the simulation, simply pass the settings file of the hibernation point in the run command, for instance

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

The settings are such that the code will use the `read from disk` option for initial data, taking into account a few additional metadata items (such as time step, cycle etc.) that are only relevant for restart operations. These provisions ensure that the simulation should proceed exactly as if no hibernation had occurred at all, up to effects due to finite machine precision when doing some unit conversions.

It is possible to modify the settings file of a hibernation point to change the future behaviour of the simulation, for instance in order to modify the output. However, this should be done with great care. Note also that the code will overwrite any output that may have been produced beyond the hibernation point in a previous run.

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 3). 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 in the *FFTW* library (in particular, the FFT is unnormalised, meaning that a forward and backward transform together multiply a field by $\texttt{Ngrid}^3$). 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.

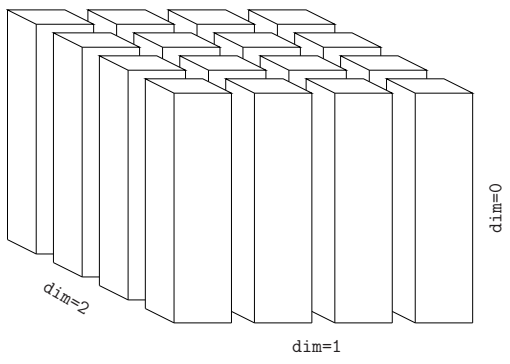

Figure 3: Sketch of the domain decomposition performed by *LATfield2* to distribute the work over the MPI processes. The data is stored in row-major order such that `dim=0` is *contiguous* and *local*, i.e. the non-contiguous dimensions `dim=1,2` are scattered over the process grid. This is essential 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.

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.

`class_tools.hpp` The interface with the Boltzmann code *CLASS* is found in this header. It is only available if *CLASS* is linked as a library and the compilation flag `-DHAVE_CLASS` is specified.

`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).

`hibernation.hpp` This header contains the routines needed for writing a hibernation point to disk.

`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.

`ic_prevolution.hpp` This header contains the `prevolution` IC generator. It is only included when the compilation flag `-DICGEN_PREVOLUTION` is specified.

`ic_read.hpp` This header contains a function to read initial data from disk.

`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. Most numerical constants are also defined in this header.

`output.hpp` This header contains the routines which produce output in form of snapshots, light cones or power spectra.

`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 recognise them.

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

`radiation.hpp` This header contains the linear radiation module that also treats other linear sources. It is only enabled if the code is compiled with `-DHAVE_CLASS`, linking *CLASS* as a library.

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

`velocity.hpp` This header contains the implementation of the *RESCALED* method to estimate the velocity field, see [10] for details.

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

- Initialisation and parsing of settings file
- Generation of initial conditions "on the fly" (lines 295-311)
- Main loop (line 368)
  - Construction of the stress-energy tensor of linear sources (line 377)
  - Construction of the stress-energy tensor of the N-body ensembles (lines 379-446)
  - Update ϕ (lines 453-514)
  - Output of *background data* (lines 517-532)
  - Compute χ (lines 535-564)
  - Update[7] B_i (lines 566-595)
  - Output of *light cones* (line 604)
  - Output of *snapshots* (lines 613-627)
  - Output of *power spectra* (lines 635-675)
  - Check if further output is scheduled, otherwise exit main loop (lines 681-689)
  - Determine time step subdivision for non-cold species (lines 692-697)
  - Update N-body particles (momenta and positions) of non-cold species using leap-frog with subdivided time steps (lines 728-764)
  - Update N-body particles (momenta and positions) of CDM and baryons using leap-frog (lines 767-818) – the background is also evolved in this code section (lines 788 and 810)
  - τ += $\Delta\tau$ (line 821)
  - If required, write hibernation point (lines 823-861)
- The main loop terminates when no more output is scheduled
- Clean-up

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. Note that the vector perturbation is internally stored as $[a^2\, B_i]$ and rescaled for outputs. 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 $-a^3 T^0_0$ and $a^3 T^i_j$, i.e. the background expansion is scaled out. Also, the mass units are chosen such that, in the homogeneous limit and for

[7] Note that a copy of $[a^2\, B_i]$ is kept in Fourier space in order for the `parabolic` solver to work (for all other fields and for the `elliptic` solver, the Fourier image can usually be discarded after the backwards transform)

non-relativistic matter, the $-a^3\bar{T}^0_0$ is equal its density parameter Ω at redshift $z = 0$. For the momentum density the code computes $a^4 T^0_i$ which has the correct number of factors of a for the momentum constraint that gives $[a^2\, B_i]$, $\Delta^2\, [a^2\, B_i] = 16\, \pi\, G\, \nabla\times\nabla\times [a^4 T^0_i]$.

When output power spectra are computed, the code always writes the dimensionless spectra as defined in [8]. However, k is converted back to physical units, h/Mpc. Particle snapshots in *Gadget-2* format use length units of kpc/h, mass units of $10^{10}\, M_\odot/h$ and velocities in km/s. These can be adjusted with the compiler flags `-DGADGET_LENGTH_CONVERSION=<value>`, `-DGADGET_MASS_CONVERSION=<value>` and `-DGADGET_VELOCITY_CONVERSION=<value>`. 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 (August 2019)

- Fixed some bugs in `Particles_gevolution.hpp` and `ic_basic.hpp` which could lead to issues in the particle initialisation under certain circumstances.
- A bug was also discovered and fixed in `ic_read.hpp` which prevented the code from reading *Gadget-2* binaries if they where split into multiple files.

Patch 2 (November 2019)

- The *CLASS* interface has been made compatible with *CLASS* version 2.8, which also means that it may no longer work for some older versions.

Patch 3 (March 2020)

- Slightly improved velocity interpolation for particle light cones, to avoid sharp features in the velocity distribution.

Patch 4 (September 2021)

- Fixed a bug in `Particles_gevolution.hpp` that could cause some duplicate particles to appear on the light cone. In addition, the interpolation of particle positions on the light cone has been improved slightly, adding a small correction due to line-of-sight velocities.

Patch 5 (August 2022)

- The *CLASS* interface has been made compatible with *CLASS* version 3.2, which also means that it may no longer work for some older versions.
- Introduces new compiler flag `-DGRADIENT_ORDER=<value>` to enable higher-order gradient computation (default is `GRADIENT_ORDER=1` which is the original low-order gradient used in all previous versions of *gevolution*).

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," *JCAP* **1607** no. 07, (2016) 053, arXiv:1604.06065 [astro-ph.CO].

[6] J. Adamek, J. Brandbyge, C. Fidler, S. Hannestad, C. Rampf, and T. Tram, "The effect of early radiation in N-body simulations of cosmic structure formation," *Mon. Not. Roy. Astron. Soc.* **470** no. 1, (2017) 303–313, arXiv:1703.08585 [astro-ph.CO].

[7] J. Adamek, R. Durrer, and M. Kunz, "Relativistic N-body simulations with massive neutrinos," *JCAP* **1711** no. 11, (2017) 004, arXiv:1707.06938 [astro-ph.CO].

[8] 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].

[9] K. M. Górski, E. Hivon, A. J. Banday, B. D. Wandelt, F. K. Hansen, M. Reinecke, and M. Bartelman, "HEALPix - A Framework for high resolution discretization, and fast analysis of data distributed on the sphere," *Astrophys. J.* **622** (2005) 759–771, arXiv:astro-ph/0409513 [astro-ph].

[10] G. Jelic-Cizmek, F. Lepori, J. Adamek, and R. Durrer, "The generation of vorticity in cosmological N-body simulations," *JCAP* **1809** no. 09, (2018) 006, arXiv:1806.05146 [astro-ph.CO].

[11] C.-P. Ma and E. Bertschinger, "A Calculation of the full neutrino phase space in cold + hot dark matter models," *Astrophys. J.* **429** (1994) 22, arXiv:astro-ph/9308006 [astro-ph].

[12] 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].

[13] C. Fidler, C. Rampf, T. Tram, R. Crittenden, K. Koyama, and D. Wands, "General relativistic corrections to N-body simulations and the Zel'dovich approximation," *Phys. Rev.* **D92** no. 12, (2015) 123517, arXiv:1505.04756 [astro-ph.CO].

[14] J. Brandbyge and S. Hannestad, "Grid Based Linear Neutrino Perturbations in Cosmological N-body Simulations," *JCAP* **0905** (2009) 002, arXiv:0812.3149 [astro-ph].

[15] F. Hassani, J. Adamek, M. Kunz, and F. Vernizzi, "k-evolution: a relativistic N-body code for clustering dark energy," *JCAP* **1912** no. 12, (2019) 011, arXiv:1910.01104 [astro-ph.CO].

[16] 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./.

[17] 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].