The Gyrokinetic Plasma Turbulence Code GENE: User Manual

GENE Development Team

June 20, 2020

Contents

1 Introduction

1.1 A few words on the code

The gyrokinetic plasma turbulence code Gene (this acronym stands for Gyrokinetic Electromagnetic Numerical Experiment) is a software package dedicated to solving the nonlinear gyrokinetic equations in a flux-tube or radially global domain. Alternatively, it can be operated in a linear mode, thus calculating the properties (like complex frequencies, parallel mode structures, and quasilinear transport coefficients) of the microinstabilities driving the turbulence. Furthermore, a solver for the neoclassical equilibrium is implemented. Gene can be run on a large number of different computer architectures, including Linux clusters and various massively parallel systems, using anything between a single and tens of thousands of processors. While the code itself is written in `Fortran2008`, the package also includes an `IDL` based tool for data visualization and analysis and a python user interface for setting up simulations. Gene has been developed by a team of people (the Gene Development Team, led by F. Jenko) for almost two decades, and is freely available to anyone accepting the general guidelines stated in the "User Agreement" document.

1.2 A few words on the basic equations

In the magnetic confinement fusion (MCF) community, it is widely accepted that *ab initio* simulations of turbulence in the core of tokamaks and stellarators are to be based on the nonlinear gyrokinetic equations as first derived in the 1980s by various authors [1, 2, 3, 4] (see also the recent review, Ref. [5]). As it turns out, the fast gyromotion can be removed from the basic equations analytically, such that each particle species is described by a time-dependent distribution function in a five-dimensional phase space:

- the electron distribution function, $f_e(\mathbf{R}, v_\parallel, \mu, t)$,
- the (main) ion distribution function, $f_i(\mathbf{R}, v_\parallel, \mu, t)$,
- the distribution functions of other – active or passive – particle species.

Here, $\mathbf{R}$ describes the position of the gyrocenter in real space, and for the velocity space coordinates, we have chosen the parallel velocity $v_\parallel$ and the magnetic moment μ (other choices would also be possible, slightly affecting the form of the basic equations).

In addition, one has to advance two purely spatial, scalar quantities characterizing the electromagnetic fields by solving modified versions of Maxwell's equations. These quantities are:

- the electrostatic potential, $\phi(\mathbf{x}, t)$,
- the parallel component of the vector potential, $A_{\|}(\mathbf{x}, t)$, which is linked to the perpendicular magnetic field perturbations.
- the parallel component of the magnetic field perturbations, $B_{\|}(\mathbf{x}, t)$.

The nonlinear gyrokinetic equations as used in the GENE code may be found, e.g., in Refs. [6, 7] and in the PhD theses being available at Ref. [8]. The code currently contains:

- an arbitrary number of fully gyrokinetic particle species, active or passive
- magnetic field fluctuations perpendicular and parallel to the background field
- collision operators for each particle species, including energy scattering
- general (tokamak or stellarator) MHD equilibria, a consistent circular model, the standard $\hat{s} - \alpha$ geometry, and Miller local equilibria

This list is not exhaustive, however, and the code is still being extended and generalized. From time to time, new GENE releases will therefore be provided.

Neoclassical equilibrium: Often, turbulent processes dominate the transport of energy, particles, and momentum in the core of fusion devices. However,in some cases, neoclassical effects may play a role. In the flux-tube limit, the neoclassical equation is contained in the gyrokinetic equation as its perturbed, but non-fluctuating part and it decouples from turbulence. The neoclassical equilibrium can thus be solved separately, which is featured by GENE, whereby no electric/magnetic field perturbations or background electric fields are considered.

1.3 A few words on magnetic geometry

As is explained in Ref. [9], one can save 2-3 orders of magnitude in computational effort – compared to more traditional ways of treating the geometry – by using a coordinate system which is aligned to the background magnetic field. This is because the turbulent fluctuations have very large correlation lengths along the field lines (here, the particle motion is practically free) but very small ones in the perpendicular directions (due to the slowness of the gyrocenter drifts). Consequently, the parallel structure of the fluctuations can be represented on a very coarse grid in the field-line following direction, an advantage which does not exist in other, non-field-aligned coordinate systems. Concerning the simulation volume, it turns out to be most natural and efficient to employ a flux-tube domain which follows the field lines and whose perpendicular widths exceed the respective correlation lengths [9]. This approach helps to minimize the computational cost and allows for arbitrarily long simulation times (there is no profile relaxation, provided one uses periodic radial boundary conditions).

As a result of the use of field-aligned, non-cartesian coordinates, several differential and integral operators occurring in the gyrokinetic equations have to be expressed in terms of the metric coefficients [10]. Here, one can currently choose between two options when running GENE. If one is interested in more basic tokamak physics issues, it might be sufficient to employ a simple $\hat{s}$-α MHD equilibrium for which all the relevant metric quantities can be calculated analytically. It describes a large aspect ratio tokamak with circular cross section, but allows for finite Shafranov shifts. One should be aware however that this model contains some order $\epsilon = r/R$ inconsistencies; for simulations with large values of ϵ (typical greater than 0.15), a corrected circular concentric flux surface model is available. Shaping studies can be carried out by using the local Miller equilibrium model. If one aims at even more realism (e.g., shaped equilibria) or at doing stellarator work, however, one should rather provide an appropriate input file for GENE containing the required geometrical information. This can be done by means of software tools which are also provided within the GENE package, post-processing MHD equilibrium data. Another option is to provide an EFIT g-eqdsk file, from which the code will automatically extract the appropriate geometric information.

1.4 A few words on algorithmic issues

For the solution of the nonlinear gyrokinetic equations, the Gene code employs a fixed grid in phase space. It may therefore be called a "continuum" or "Vlasov" or "Eulerian" code – in contrast to particle-in-cell (PIC) or semi-Lagrangian codes which are based on computing the characteristic curves in phase space. Some of the key features of the Eulerian approach are the following:

- it allows for arbitrarily long simulation times
- it exhibits very good convergence properties
- it is very flexible with respect to algorithms
- it can lead to highly parallelizable codes

The discretization scheme used in Gene is based on the so-called "method of lines." This means that the phase space (differential and integral) operators are discretized first, and that an appropriate time stepping scheme is chosen independently in a second step. The differential operators involving the field-line following coordinate z or the parallel velocity $v_\|$ are discretized with the help of a fourth-order Arakawa scheme [11], while the perpendicular x and y directions are treated in the framework of a pseudospectral approach. The latter implies that all linear terms (as well as spatial derivatives) are evaluated in k_x-k_y space, while the nonlinearities are computed in real space for better efficiency. In this context, one has to employ Fourier transformations, of course, as well as an appropriate dealiasing method to avoid spurious couplings of the smallest and largest scales. The velocity space integrations are done via Gauss and trapezoidal rules in μ and $v_\|$ space, respectively. Collisions are modeled in GENE using a linearized Landau-Boltzmann operator, which is discretized by means of a finite volume scheme. Most of the above numerical methods are explained in standard text books on applied mathematics or computational fluid dynamics (CFD).

The time stepping, on the other hand, is done by means of a fourth-order explicit Runge-Kutta method which operates very close to the theoretical optimum (in terms of minimum computational cost per time step unit) of any explicit scheme for our given problem. Unless the code user decides to pick another value, the initial time step is chosen automatically by Gene such that it marginally satisfies the linear stability limit – often set by the parallel electron dynamics in multi-species runs. This is done by an eigenvalue solver which is invoked as part of the initialization process. However, even when starting a simulation at sufficiently low fluctuation amplitudes, the latter are likely to get big enough in the course of the simulation such that the time step needs to be adapted (by the code itself) in order to account for Courant-Lévy-Friedrichs type time step restrictions set by the nonlinear terms. Such nonlinear limits would even have to be respected in a purely implicit code (which does not exist), since they are also to be regarded as a crucial indicator of *accuracy* in the context of turbulent plasma dynamics. Thus, when doing nonlinear runs with Gene, one is usually operating close to the point of maximum efficiency. Starting with GENE release 1.6, the time stepping of the collisional dynamics is separated from the rest of the Vlasov dynamics by means of an adaptive operator scheme, usually reducing the number of evaluations per timestep for the collision operator.

Alternative to time stepping, which solves the gyrokinetic equation as an initial value problem, the linearized equation allows for eigenvalue computations. To that aim, interfaces to the PETSc/SLEPc package and several other linear algebra packages are implemented. The neoclassical equilibrium can be obtained by initial value computations or by an algebraic solver. Gene features an interface to the PETSc package for that purpose.

1.5 A few words on computer resources

Gene has been parallelized and ported to a large number of different computer architectures. The latter include various Linux clusters, Mac OS, as well as massively parallel systems of the following type:

- IBM iDataPlex systems
- IBM Power architectures
- IBM BlueGene/P
- Cray XE6, Cray XC30 (Cascade)
- various Bull and Intel based clusters

It must be stressed here, however, that this list is neither exhaustive nor final. Thus, if any (potential) user of the code would like to know if Gene also runs on his or her favorite architecture not mentioned above, the GENE Development Team should be contacted (support@genecode.org).

While GENE is highly parallelizable and has already been run on tens of thousands of processors, there are many interesting applications which do not require such enormous resources. For example, linear runs – determining properties of the microinstabilities driving the turbulence (like complex frequencies or quasilinear transport coefficients) – can be done even on a single processor and do not take much time. Moreover, for certain studies, it might be possible to slightly change some physical or numerical parameters (like the plasma beta, the ion-to-electron mass ratio, the temperature ratio, the simulation box size, or the phase space resolution) such that the computational effort can be reduced significantly without affecting the results too much. This clearly needs to checked, however, on a case-by-case basis (one such example is mentioned in Ref. [7]). If it indeed turns out to be feasible to do 'reduced simulations' of this sort in any given context, the main load of a project may thus be carried, e.g., by a Linux cluster or another small-size parallel platform. Otherwise, one needs to resort to more powerful machines. Finally, there are other applications – like nonlinear multi-scale or stellarator simulations – which *always* require a very significant amount of computer resources. Projects involving such demanding runs can only be initiated if access to adequate high performance computers is provided.

2 Installing Gene on your machine

2.1 Getting the source code

GENE is stored in a password protected `git` repository hosted at (`https://gitlab.mpcdf.mpg.de/GENE/gene`). If you do not have an account and password already, please visit `http://genecode.org` and fill in the form in the *Get GENE* section. To download GENE to your local system, make sure that you have `git` installed, go to the directory in which you want to install the code (e.g. `<genedir>`) and type

```
git clone https://<USER>@gitlab.mpcdf.mpg.de/GENE/gene.git
-b release-<X.Y> <genedir>
```

Here, `<USER>` should be replaced by your `git` user id. As a result, you should get the release files containing:

- the GENE source code located in the `./src` directory
- the GENE documentation in `./doc`
- the GENE test suite in `./testsuite`
- the Python-based launcher tool in your `gene` root directory
- the `IDL`-based postprocessing routines in `./diagnostics`
- some experimental `python`-based postprocessing routines in `./python-diag`
- various helpful sripts and tools in `./tools`

as well as the `newprob` script, a generic `makefile`, and several machine-specific makefiles (in `./makefiles`). To receive patches later on, simply go to your GENE base directory (here: `<genedir>`) and type `git pull`. More information about `git` commands can be found at
`http://git-scm.com/doc`.

2.2 Creating a new machine-specific makefile

Machine-specific makefiles and batch scripts for the massively parallel systems mentioned in Sec. 1.5 and for a number of Linux clusters (at MPCDF, PPPL, SPC etc.) are provided automatically. In such case, a call to `make` `mach_wrapper` in the GENE base directory should return the proper machine name and `make get_required_modules` may provide a list of modules to be loaded. The header of the machine-specific makefile in

```
./makefiles/<machine\_name>/<machine\_name>.mk
```

should also be consulted for necessary environment settings. If you are planning to use GENE on a different (unknown) architecture, you will have to create a new machine-specific `makefile`. For this purpose, the template provided in `./makefiles/new_machine` should be self-explanatory. Note that the makefile structure has been changed with release 1.6 to provide common definitions/flags for various compilers (to be found in `./makefiles/compilers`), and also a separate file containing general compiling rules (`./makefiles/rules.mk`). The external software you definitely need to specify in order to compile GENE are:

- a `Fortran2008` conform compiler
- the `MPI2` message passing interface
- an FFT routine (FFTW, MKL, or ESSL)
- the BLAS/LAPACK library (also contained in MKL and ESSL)

Optional software packages that can extend the functionality of GENE are

- for eigenvalue computations and a precise determination of the maximum linear time step, the ***complex*** version of the (free) PETSC/SLEPC package [14, 15, 16], available at `http://www.mcs.anl.gov/petsc` and `www.grycap.upv.es/slepc` **(highly recommended!)** [1]
- for full eigenvalue spectra, the ScaLAPACK library, available at `http://www.netlib.org/scalapack/`
- for the HDF5 output format the corresponding library, see `http://www.hdfgroup.org/HDF5/`

[1]set `--with-scalar-type=complex` at configuration and furthermore `--with-precision=double` if GENE is compiled with double precision

In order for the `makefile` to work properly, choose a name for your machine (⟨*my_machine*⟩ in the following) and rename the directory and `makefile` from
`./makefiles/new_machine/new_machine.mk`
to
`./makefiles/`⟨*`my_machine`*⟩`/`⟨*`my_machine`*⟩`.mk`
You can also save submit script templates for your machine in the ⟨*my_machine*⟩/ folder; they will then be available in every problem folder you create.

NOTE: The machines indicated above are automatically recognized by the values of certain predefined environment variables. If you have written a new machine-specific makefile, you either have to extend these routines (located in `./makefiles/machine.def`) or to set an environment variable MACHINE to ⟨*my_machine*⟩ before you can compile or test your installation.

2.3 Testing your installation

Once you have a (preliminary) makefile for your machine, go to the `./testsuite` directory and check whether you can compile with
`gmake -f ../makefile`
Note that the first call to gmake will generate a directory named `./bin`, which contains the machine-specific makefile and, after compilation, an executable named `gene_`⟨*`my_machine`*⟩. This local version of the makefile can then be improved and tested before it is finally transferred back to `./makefiles/`⟨*`my_machine`*⟩.
After you manage to compile without errors, type
`./testsuite`
This will compute test problems with increasing complexity and compare the results and time needed for the computation with reference values. Per default, the maximum number of MPI processes used is 8, this can be decreased by the option -mpiprocs # MPI processes. The OMP parallelization (if supported by your machine and compiler) can be tested by simply by setting the OMP_NUM_THREADS environment variable to a value greater 1 (you may want to limit the number of MPI processes in that case, since the total number of processes used is OMP_NUM_THREADS times mpiprocs).
Since release 1.5, it is possible to compile also from the GENE root directory by typing `gmake`. When compiling from within a `prob` directory, the position of the main makefile has to be given by typing `gmake -f ../makefile`, as before. With this new structure, only one copy of the GENE executable will reside in `./bin`, instead of separate ones for each problem directory.

Note, that changes to the machine specific makefiles in `./makefiles/`⟨*my_machine*⟩ (e.g., after an update) need to be transferred by hand (either by simply calling `gmake distclean`, by copying the ⟨*my_machine*⟩`.mk` file or removing the `./bin` directory) in order to allow the user to keep care of local modifications.

2.4 Using the CHEASE interface module

The metric and equilibrium quantities used in GENE can be obtained using an interface with the MHD equilibrium code CHEASE. This interface requires the library FUTILS, which will automatically be downloaded from an svn repository, as well as the HDF5 library. More information on these two libraries can be found in `external/futils-gene/docs/futils.pdf`. To run GENE with this interface the precompilation option `FUTILS=yes` should be set in the machine.mk file.

3 Setting up a computation

To create a problem folder (`./prob01/`) in which the parameters for a computation are defined, execute
`./newprob`
in the GENE base directory. Successive calls to `newprob` will generate new problem folders: (`./prob02/`, `./prob03/` etc.). The problem folders may be renamed, the directory structure must not be changed, however.

The next step is to adapt the switches for libraries, precision, optimization etc. in the machine-specific makefile (if necessary). These switches have to be set before compilation – later changes are only effective after a recompilation of the code. Compile GENE by typing
`gmake -f ../makefile`
in the problem directory (or by typing `gmake` in the `./` directory. This action should create an executable called ./bin/`gene_`⟨*`my_machine`*⟩, which can then be used for both linear and nonlinear runs. Unless one wants to change or update the source files, there is usually no need to recompile the code.

The other input file to be adapted is the `parameters` file, in which various physical and numerical parameters can be specified. Since release 1.4, a tool named `GENE-GUI.py`, which is described in the next section, is available to simplify this task.

3.1 The GENE launcher

In order to provide easier user access to the GENE code, a graphical user interface is available that allows the user to read, edit and write 'parameters' files, while providing consistency checks to prevent common errors. The interface requires a working installation of Python 2.6+ and the Python/Tkinter widget package, which is contained in many distributions of Python (including the standard one from `www.python.org`). To open the program, just call './GENE-GUI.py' from the GENE root directory.

3.1.1 Description of the interface

The launcher window is divided into several tabs which sort the input parameters according to their purpose:

Operation: This tab allows the user to adjust the GENE code's operation modes and should always be visited first, since changing between linear/nonlinear, initial value/eigenvalue mode or neoclassical solver will affect the options that can be selected in other tabs.

Input/Output: Here the user can specify output directories and the frequency with which the code's output files will be written to disk.

Domain: This tab allows the user to set the grid sizes and resolutions for the simulation.

General: The 'General' tab contains parameters that affect the runtime of the simulations, as well as timestep-related parameters. Furthermore, adjustments for eigenvalue runs and various physics parameters can be made.

Collisions: Collisionality and options for the collision operator can be adjusted here.

Geometry: The 'Geometry' tab contains all geometry-related parameters.

Species: In the 'Species' tab, the user can adjust the number of simulated species as well as their parameters and whether they should be treated as passive tracers. In addition, the settings for various profile interfaces can be adjusted here.

Ref. Values: This tab allows the user to specify the reference quantities used for normalization. Once entered, an automatic calculation of dimensionless parameters like β, the collisionality or the debye length is possible. GENE passes the reference values to the output parameters file, from where they may be used to convert heat flux values to SI units in the diagnostic tool.

Development: Here the user can specify parameters which are not hardwired in the Launcher tool. For each parameter, one can choose the namelist it should be appended to.

In the top row of the interface, numbered buttons allow the user to manage multiple jobs in the same launcher instance by changing between the numbered jobs. All parameter settings that have been made are saved in memory and can be reloaded by returning to the same job number again.

When hovering over labels in the user interface, tooltips will often be displayed to help the user decide how to adjust the respective parameter. In addition to this, the tooltip contains the name of the parameter as it appears in the `parameters` file, should the user wish to edit the latter by hand.

Command buttons:

Read parameters: Reads existing parameters files from the 'prob' directories as well as those created by GENE.

New 'prob' dir.: Creates a new 'prob' directory.

Write parameters: Overwrites an existing parameters file with one created by the launcher tool. The user must select an existing 'prob' directory, in which the `parameters` file is to be written.

Save as: This command allows saving the parameters to an arbitrary file name.

Check: Performs some consistency checks on the input values the user has given. This is also the case when the 'Write' button is clicked.

Submit: Submits a single run taking into account the number of processes. This function relies on the existence of a script named `launcher.cmd` in the `makefiles/⟨my_machine⟩/` directory. This file should contain a job submission script appropriate for your machine, with certain parameters (like the number of MPI processes) replaced by keywords. The latter will then be replaced by the Launcher with the required values. See, e.g., `makefiles/bob_cluster/launcher.cmd` for an example file, and the next paragraph for a list of keywords.

Keywords for job submission: The following keywords are machine-specific input parameters which are required to be able to submit jobs automatically through the Launcher.

MAXWT: Defines the maximum wallclock time in hours.

PROCSPERNODE: Maximum number of MPI processes per node.

SUBMITCMD: The command required to submit a job.

These keywords have to be uppercase and defined in the form `### <keyword> <value>`.
The following keywords are intended for output and will be replaced with their appropriate values:

NMPIPROCS: Number of MPI processes

NPROCSDIV4: Number of MPI processes, divided by four (required e.g. on BlueGenes in Virtual node mode)

PPN: Processes per node (not required by all machines)

NODES: Number of nodes (not required by all machines)

WALLTIME: The wallclock time limit for the simulation to be submitted (can be entered in a dialog which appears after clicking the 'Submit' button).

JOBNAME: Name of the job (can also be entered in a dialog).

In case additional keywords, or a different treatment of the existing ones, are required to set up job submission for a given machine, the Gene development team should be notified (support@genecode.org).

Parameter scans: Scans over one or more parameters can be performed by selecting the option 'Parameter scan' in the 'Operation' frame and giving an entry like "1.0 !scan: 0.1,0.1,1.0" (the first number is necessary) to the parameter which should be scanned. This entry will be copied to the parameters file, so that all options the scanscript offers are available also in the launcher (see section 3.4 for a description of the scanscript).

Further options: The 'clear form' button deletes all parameters entered into the user interface, while the 'default values' button fills the entries so that a standard cyclone base case run (using, however, the corrected circular model) can be started. Note that the latter button delivers different default values depending on whether the nonlinear or linear mode is selected. Finally, a checkbutton labeled 'Advanced options' allows the user to set additional parameters that are hidden otherwise, while the button labeled '?' brings up the Gene documentation.

3.2 The `parameters` file

The `parameters` file consists of namelists containing key-value pairs. The keywords and their meanings are given in the following list. In parentheses, the data type ([int, real, str, bool] for integer, real, string and logical) and default value (if any) is given. Note that strings have to be provided using quotation marks. Parameters with default values don't have to be specified in the `parameters` file. A newly created problem folder contains a `parameters` file for a "Cyclone base case" scenario described in Ref. [13].

3.2.1 The `parallelization` namelist

`n_procs_s [int 1]:` number of MPI processes over which the species are distributed; to ensure load balancing, the parameter `n_spec` (see below) should be equal or at least divisible by this number; if set to a number smaller than one, GENE tries to find the best combination using up to as many MPI processes as are left in the MPI_COMM_WORLD. The range can be restricted by additionally providing lower and upper integer-valued boundaries via `min_nps` and `max_nps`.

`n_procs_w [int 1]:` number of MPI processes over which the μ (magnetic moment) grid is distributed; to ensure load balancing, the parameter `nw0` (see below) should be divisible by this number; if set to a number smaller than one, GENE tries to find the best combination using up to as many MPI processes as are left in the MPI_COMM_WORLD. The range can be restricted by additionally providing lower and upper integer-valued boundaries via `min_npw` and `max_npw`.

`n_procs_v [int 1]:` number of MPI processes over which the $v_{||}$ (parallel velocity) grid is distributed; to ensure load balancing, the parameter `nv0` (see below) should be divisible by this number; note that this parallelization does not reduce the size of the gyroaveraging matrix per processor (which is independent of $v_{||}$) in global simulations - hence, it should be kept low for this type of operation; if set to a number smaller than one, GENE tries to find the best combination using up to as many MPI processes as are left in the MPI_COMM_WORLD. The range can be restricted by additionally providing lower and upper integer-valued boundaries via `min_npv` and `max_npv`.

`n_procs_x [int 1]:` global code specific parameter which controls the number of MPI processes over which the x ('radial direction') grid is distributed; to ensure load balancing, the parameter `nx0` (see below) should be divisible by this number; note that x- and y-parallization cannot be used in combination; if set to a number smaller than one, GENE tries to find the best combination using up to as many MPI processes as are left in the MPI_COMM_WORLD. The range can be restricted by additionally providing lower and upper integer-valued boundaries via `min_npx` and `max_npx`.

`n_procs_y [int 1]:` number of MPI processes over which the k_y ('poloidal wavenumber') grid is distributed; to ensure load balancing, the parameter `nky0` (see below) should be divisible by this number; furthermore, due to array distributions, `nx0` (see below) should be divisible by two times this number; if no thread-safety is provided, OMP_NUM_THREADS and `n_procs_y` cannot be chosen to be greater than unity simultaneously; finally, x- and y-parallization cannot be used in combination in global simulations. if set to a number smaller than one, GENE tries to find the best combination using up to as many MPI processes as are left in the MPI_COMM_WORLD. The range can be restricted by additionally providing lower and upper integer-valued boundaries via `min_npy` and `max_npy`.

`n_procs_z [int 1]:` number of MPI processes over which the z (field-line following coordinate) grid is distributed; to ensure load balancing, the parameter `nz0` (see below) should be divisible by this number; if set to a number smaller than one, GENE tries to find the best combination using up to as many MPI processes as are left in the MPI_COMM_WORLD. The range can be restricted by additionally providing lower and upper integer-valued boundaries via `min_npz` and `max_npz`.

3.2.2 The `box` namelist

`n_spec [int]:` number of particle species to be used in the computation; if only a single ion (electron) species is employed, GENE assumes that the electrons (ions) are adiabatic; note that there is no upper limit to the number of particle species to be specified, and that every one of them is treated gyrokinetically, retaining finite gyroradius effects `nx0 [int]`: number of grid points in the x direction (x is the flux-surface label, i.e., the radial coordinate); should be a product of low prime numbers for efficient FFTs along x; for nonlinear runs, typical values are 128 or 256, translating into a radial resolution of at least (less than) one ion gyroradius; numerical convergence tests with higher numbers are encouraged.

`nky0 [int]`: number of Fourier modes in the y direction (y is the field-line label within a given flux surface, i.e., the bi-normal coordinate); should be a power of low prime numbers for efficient FFTs and parallelization along y; for nonlinear runs, typical values are 16, 32, or 64; usually, it is desirable that the maximum value of k_y (in normalized units) is at least of the order of unity – otherwise, the run might not be converged, resulting, e.g., in wrong spectral decays and wrong transport levels; for linear runs, `nky0=1` together with scans (see Sec. 3.4) over `kymin` are recommended

`nz0 [int]`: number of grid points in the z direction (z is the field-line following coordinate, i.e., the parallel coordinate); should be even; typical values are in the range between 16 and 32 for tokamak runs, and of the order of 100 or even more for stellarators like Wendelstein 7-X

`nv0 [int]`: number of grid points in the $v_\parallel$ direction ($v_\parallel$ is the parallel velocity); should be even; typical values are in the range between 32 and 64, but in some cases, larger values might be required - particularly global simulations with strongly radially varying temperature profiles will be more demanding since the normalization is taken at the reference flux surface at `x0`.

`nw0 [int]`: number of grid points in the μ direction (μ is the magnetic moment); typical values are in the range between 8 and 16, but in some cases, larger values might be required - particularly global simulations with strongly radially varying temperature profiles will be more demanding since the normalization is taken at the reference flux surface at `x0`.

`lx [real]`: extension of the simulation box in the x direction, in units of gyroradii (calculated by using the units of mass and temperature); this value is adjusted automatically for finite magnetic shear ($\hat{s} \neq 0$) such that the quantization rule $\texttt{kymin} \cdot \texttt{shat} \cdot \texttt{lx} \in \mathbb{N}$ (see, e.g., Ref. [9]; this rule is necessary to make the parallel and radial boundary conditions consistent; `kymin` and `shat` are defined below) is satisfied – and it is neglected if the parameter `nexc` (see below) is specified

`lx_a [real]`: global code specific parameter which can be used instead of `lx` to define the radial box length as fraction of the minor radius

`x0 [real]`: For the 'tracer_efit'/'chease' geometry interfaces or when reading profile files, this parameter determines which flux surface (following the x coordinate definition, e.g., normalized ρ_{tor} for `'tracer_efit'` geometries) will be selected for simulation. When running with parallel flow shear and 'tracer' geometry, this parameter also has to be specified.

`kymin [real]`: minimum value of k_y, in units of inverse gyroradii; this corresponds to an extension of the simulation box in the y direction of $2\pi/\texttt{kymin}$, in units of gyroradii; typically, values of the order of 0.1 or less are used for this parameter, corresponding to box sizes of at least 60 gyroradii; inspection of the Gene diagnostics (possibly together with convergence tests) are usually necessary to find out if a given run may be considered converged in `kymin` (if not, it has to be lowered)

`lv [real]`: extension of the simulation box in the $v_\parallel$ direction, in units of the thermal velocity (note the factor of 2 in this definition) $v_{Tj} = (2T_{j0}/m_j)^{1/2}$ of the respective species j; in other words, the parallel velocity points span the range from $-\texttt{lv} \cdot v_{Tj}$ to $\texttt{lv} \cdot v_{Tj}$; typically, this parameter should be at least in the range of 3 in order to make sure that the integral of the discretized, normalized Maxwellian is sufficiently close to unity; in radially global simulations, the thermal velocity at position `x0` is taken for normalization – depending on the variations of the temperature profile, the box size thus needs to be increased significantly.

`lw [real]`: upper end of the simulation box in the μ direction, in units of T_{j0}/B_{ref} of the respective species j; typically, this parameter should be at least in the range of 9 in order to make sure that the integral of the discretized, normalized Maxwellian is sufficiently close to unity; in radially global simulations, the thermal velocity at position `x0` is taken for normalization – depending on the variations of the temperature profile, the box size thus needs to be increased significantly.

`adapt_lx [bool T]`: sets the optimal lx, so that the relation $\texttt{lx} = 1/(\texttt{ky} \cdot \texttt{shat})$ holds for all ky. This maximizes the number of poloidal connections for a given nx0, and means that only one linearly independent mode is considered. This parameter is ignored for nonlinear runs.

optional:

`nexc [int]`: can be used in lieu of `lx` to define the radial box width for finite magnetic shear ($\hat{s} \neq 0$) via the relation $\texttt{lx} = \texttt{nexc}/(\texttt{kymin} \cdot \texttt{shat})$; if specified, the value for `lx` is overwritten

`ky0_ind [int 1]`: shift of the k_y range for linear runs; the smallest value of k_y in the computation is then $\texttt{ky0\_ind} \cdot \texttt{kymin}$ instead of `kymin`; however, if $\texttt{nky0} > 1$, neighboring k_y values are still separated by `kymin`; is neglected for nonlinear simulations

`kx_center [real 0.0]`: shift of the k_x range for linear runs, useful to investigate the growth rate spectrum at finite k_x; this parameter is neglected for nonlinear simulations

`n0_global [int]`: toroidal mode number of the `kymin` mode (is taken into account in the parallel boundary condition and effectively shifts the mode rational surfaces radially; typically, not important for local simulations themselves but for post-processing, e.g., a proper mapping on a toroidal geometry); set to -1111 to automatically evaluate `n0_global = nint(kymin*C_y/(rho_ref/L_ref))` next to the given `kymin`; note that `rhostar` is required.

`adapt_ly [bool F]`: adapts `ly`/`kymin` to be consistent with `n0_global` quantization explained above which might be necessary for post-processing (in the local code, you are allowed to select `kymin` values which do not translate to integer valued mode numbers; `adapt_ly` is always enabled in global simulations)

`mu_grid_type [str 'gau_lag']`: defines the grid in μ direction. Default is Gauss-Laguerre ('gau_lag'), other options are, e.g., equidistant in $v_\perp$ ('eq_vperp', recommended for collision_op = 'sugama'), or Gauss-Legendre ('gau_leg')

3.2.3 The `in_out` namelist

`diagdir [str]`: directory in which the output files should be written; it is sufficient to provide the path relative to the directory in which the executable resides; note that quotation marks have to be used here to identify the input as a string

`chptdir [str]`: directory in which the checkpoint file should be written; it is sufficient to provide the path relative to the directory in which the executable resides; note that quotation marks have to be used here to identify the input as a string

`istep_field [int 0]`: number of time steps between two consecutive entries into the `field.dat` output file (see below); this number should probably not be too small (e.g., in the range of 100-1000)

`istep_mom [int 0]`: number of time steps between two consecutive entries into the `mom.dat` output file (see below); this number should probably not be too small (e.g., in the range of 100-1000); multiples of `istep_field` are recommended

`istep_nrg [int 10]`: number of time steps between two consecutive entries into the `nrg.dat` output file (see below); this number can be rather small (e.g., in the range of 10) since this particular file will usually not get too big, anyways

`istep_omega [int 20]`: number of time steps after which frequency and growth rate are calculated and checked for achievement of desired precision (see `omega_prec`); istep_omega is set to 0 for nonlinear simulations or linear simulations with adapt_lx=.f.; the results will be written to `omega.dat` and - if the desired precision has been reached - to standard output

`istep_vsp [int 0]`: number of time steps between two consecutive entries into the `vsp.dat` output file (see below); this number should probably not be too small (e.g., in the range of 100-1000)

`istep_neoclass [int 0]`: time steps between two entries in the `neoclass.dat` output file (see below); (typical value in the range of 10) `istep_energy [int 100]:` time steps between two entries in the `energy.dat` output file (see below); (typical value in the range of 10)

`istep_energy3d [int 0]:` time steps between two entries in the `energy3d.dat` output file (see below); (typical value in the range of 100-1000)

`istep_dfout [int 0]:` time steps between two entries in the `df_ky{4.4i}kx{4.4i}.dat` output file (typical value in the range of 100-1000)

`istep_triplet [int 0]:` time steps between two entries in the `triplet{}.dat` output file (see below); (typical value in the range of 50-200)

`read_checkpoint [bool F]:` this switch specifies whether to start the simulation by reading an existing checkpoint file (i.e., doing a continuation run) or by initializing it according to one of the options that can be specified via `init_cond`

`write_checkpoint [bool T]:` this switch specifies whether to write out a `checkpoint` file at the end of the run; this allows one to do continuation run in the future; existing checkpoint files in the `chptdir` directory are overwritten

`istep_schpt [int 0]:` number of time steps between two consecutive write outs of a security checkpoint file `s_checkpoint`; this feature should probably only be used if one is running on a quite unstable platform – in this case, part of a run can be reused even if the system crashes by renaming the `s_checkpoint` to `checkpoint`; existing security checkpoint files in the `chptdir` directory are constantly overwritten

`istep_g1 [int 0]:` number of time steps between two consecutive write outs of the g_1 distribution function in the checkpoint-style output file `g1` in `chptdir`. The main difference compared to default checkpoint files is that new entries will be appended instead of overwriting the previous content. Therefore, these files become quite large and should only be written for special applications. Note that only time stamp and g_1 and no further header information will be written.

`istep_prof [int 0]:` global code specific parameter which sets the number of time steps between two consecutive write outs of the current radial temperature and density profiles.

`istep_srcmom [int 0]`: global code specific parameter which sets the number of time steps between two consecutive write outs of various flux surface averaged velocity space moments of the source and sink operators.

`write_h5 [bool F]`: switches all binary output (e.g., `mom` and checkpoint files) to hdf5; *however, due to current restrictions in an external library, it is not possible to use this features if more than GENE simulation is run in parallel - for instance, in parameters scans. Furthermore, linking of the* `FUTILS` *library is obligatory.*

`chpt_read_h5 [bool F]`: enforces hdf5 format for all `checkpoint` file input (can be used to, e.g., convert an hdf5 checkpoint into a binary checkpoint); *aforementioned restrictions apply*

`chpt_write_h5 [bool F]`: enforces hdf5 format for all `checkpoint` file output (can be used to, e.g., convert a binary checkpoint into an hdf5 checkpoint); *aforementioned restrictions apply*

`chpt_read_hac [bool F]`: enforces `ADIOS` format for all `checkpoint` file input (can be used to, e.g., convert a hac checkpoint into a binary checkpoint) *requires* **H***lst*-**Adions**-**C***eckpoint interface to be linked*

`chpt_write_hac [bool F]`: enforces hdf5 format for all `checkpoint` file output (can be used to, e.g., convert a binary checkpoint into a hac checkpoint) *requires* **H***lst*-**A***dios*-**C***eckpoint interface to be linked*

`many_chpts [bool F]`: write a binary checkpoint file per process; to be used on systems with broken/deprecated MPI-I/O; *does not support changes in resolution, MPI mapping or total number of processes - thus not recommended*

`iterdb_file [str '']`: optional string defining a 2d iterdb- or DIII-D-iterdb/onetwo-style file which can be used to extract the physical input parameters; see also the description of the `species` and `units` namelists.

`iterdb_time [real -1]`: optional real value defining the closest time step (default: first) to be read from the data base file being declared by iterdb_file `momentum_flux [bool T]:` writes the radial flux of linear parallel momentum to the nrg file (includes the electrostatic and $A_{||}$ components, but not the $B_{||}$ part).

`tor_ang_mom_flux [bool F]:` writes the linear parallel momentum flux and the radial flux of toroidal angular momentum to the nrg file (includes only the electrostatic component).

3.2.4 The `general` namelist

type of operation

`comp_type [str]:` can be 'EV', 'IV' or 'NC' to select between eigenvalue or initial value computations and the algebraic solver for the neoclassical equilibrium. 'EV' is only possible for linear problems, 'NC' requires to set `include_f0_contr` (see below).

`nonlinear [bool]:` this switch specifies whether to do a nonlinear run (if set to `.t.`) or a linear one (if set to `.f.`); since nonlinear runs require many more modes (which are all coupled) than linear runs, they are much more expensive

`x_local [bool T]:` run fluxtube (local) simulations if true and (radially) global if false

`perf_vec [int 0 0 0 0 0 0 0 0 0]:` the nine-element *performance vector* which defines the routines to be used for the simulation. Zero valued elements allow for choosing the best alternative for the corresponding code part automatically while positive numbers fix the set-up. As the `perf_vec` is correlated with the chosen parallelization, both should optimally only be fixed at once.
The actually chosen `perf_vec` value is printed in the output `parameters.dat` file and can thus be taken for continuation runs.

settings for initial value runs

`ntimesteps [int 1e9]:` upper limit for the number of time steps to be computed (optional; mainly required for testing only)

`timelim [int]:` wall clock time limit for the GENE computation in seconds; this number should be set to a value which is slightly below the time limit of your batch queue (or interactive node) in order to make GENE stop in a controlled way (write checkpoint files etc.) even if `ntimesteps` has not been reached yet

`simtimelim [real]`: GENE is stopped in a controlled way at this *simulation time [in units of $L_{\mathrm{ref}}/c_{\mathrm{ref}}$]*; in addition, all active diagnostics will be called (optional; may be required for automated GENE runs, e.g., in coupled framework)

`omega_prec [real 1e-3]`: desired *absolute* target precision of growth rate and frequency for linear computations, the simulation is stopped once this *absolute* precision is reached. Only considered if `istep_omega` is greater than zero

`overflow_limit [real 1e30]`: defines the overflow limit with respect to the maximum value of nrg moments

`underflow_limit [real 1e-8]`: defines the underflow limit with respect to the minimum value of nrg moments (important if no microinstability is present)

`dt_max [real]`: (maximum) time step to be used in the simulation; this value must, of course, conform to the linear stability condition; an optimized value can be determined by an eigenvalue computation (`which_ev='largest_magnitude'`), an approximate value is suggested by the GENE standard output otherwise; note that in linear runs, the time step is always kept constant, while in nonlinear runs, it is reduced if the nonlinearities demand this

`calc_dt [bool F]`: switches on automatic determination of dt_max. If SLEPc is installed, this determination is exact (sometimes n_ev has to be increased, though), without SLEPc, dt_max is determined from some analytical approximations

`courant [real 1.25]`: factor multiplying the estimate for the maximum time step due to the ExB advection in nonlinear simulations; can be lowered if numerical instabilities are observed in the time trace of nonlinear runs; A value smaller than 0.95 should never be necessary. Some cases are even stable with courant=2.

timescheme [str 'RK4']: time scheme used for initial value calculations. Choices are 'RK3', 'RK4', 'RK4M', (and 'IE1p' if PETSc is available, 'IE1s' if SCALAPACK is available), where RK3 and RK4 are 3rd and 4th order Runge-Kutta schemes with 3 and 4 stages respectively, and RK4M is a 4th order Runge-Kutta scheme with 6 stages, where the additional stages have been used to obtain an almost optimal stability range along the imaginary axis. (IE1p/s are fully implicit 1st order Euler scheme relying on PETSc/SCALAPACK; it is implemented for linear runs only, the numerical errors are of the growth rate and frequency are corrected if istep_omega is sufficiently low)

coll_split_scheme [str 'RKCa']: time scheme used for collisions with operator splitting. Choices are 'EE1' for explicit Euler, 'RKC2', 'RKC3', 'RKC4', 'RKCa' for 2-4 stage Runge-Kutta-Chebychev schemes with optimally extended stability along the negative real axis. 'RKCa' is adaptive. Set to 'none' to include collisions in the regular timescheme without operator splitting.

settings for eigenvalue runs

ev_prec [real 1e-3]: desired precision of growth rate and frequency

which_ev [str 'jd']: can only be used if external eigenvalue solver is installed, valid choices are 'largest_real', 'smallest_real', 'largest_magnitude', 'shift_invert', 'harmonic', 'gd' or 'jd' (from SLEPc 3.1) for finding the eigenvalues closest to ev_shift; for solvers that use additive Schwartz preconditioning (like 'jd') the `parallelization` namelist settings `n_procs_z`≤`nz0`/12 `n_procs_v`≤`nv0`/12 (or -1 for automatic) are recommended;
additionally, two direct solvers 'all_lapack' (not parallelized!) or 'all_mpl' (only with SCALAPACK) to compute all eigenvalues;

n_ev [int 1]: number of eigenvalues to be computed

ev_max_it [int]: maximum number of iterations used in SLEPc eigenvalue computation, if no value is specified the SLEPc default value is used

ev_shift [cmplx (10.0,0.0)]: shift in the complex plane used for the shift and invert spectral transform

settings for neoclassical computations `include_f0_contr [bool F]`: include the f0 contribution (the neoclassical drive term) in the kinetic equation. This is required for neoclassical runs, regardless if `comp_type`='IV' or 'NC' is chosen. In order to separate the neoclassical equilibrium from plasma instabilities one choses `ky0_ind`=0 and `nx0`=1. In 'IV' neoclassical runs, one must ensure a sufficiently long simulation time for the equilibrium to be found, and set `istep_neoclass`>0. A timestep estimate can be precomputed with calc_dt.

initialization

`init_cond [str 'ppj']`: this switch specifies the initialization of the distribution function (for `read_checkpoint=.f.`); one can choose from the following options: `'sw'` ('start wave') corresponds to a single k_x-k_y mode, `'fb'` ('Fourier blob') to a Gaussian distribution in k_x, k_y, and z, `'alm'` ('all modes') to an excitation of all k_x and k_y modes (using powers of the Jacobian in the parallel direction); `'gam'` ('GAM') to initialize $\langle\phi\rangle_{\rm FS}=1$; `'zero'` to initialize zero fluctuations; besides that, there are also the options `'ppg'`, `'ppj'`, and `'mmj'`

`init_aux_x [real]`: optional control parameter in the context of setting up the initial condition in the x direction; its meaning and default value depends on the choice of `init_cond` (e.g., mode number, power law exponent)

`init_aux_y [real]`: same as above, but for the y direction

`init_aux_z [real]`: same as above, but for the z direction

species independent physical input parameters

`beta [real]`: plasma beta as defined by $\beta_{\rm ref}=8\pi n_{\rm e0}T_{\rm ref}/B_{\rm ref}^2$ in cgs; respectively $\beta_{\rm ref}\approx 403\cdot 10^{-5}n_{\rm e19}T_{\rm ref,keV}/B_{{\rm ref},T}^2$ (electron density n_{e19} in $10^{19}\frac{1}{m^3}$, $T_{\rm ref,keV}$ in units of keV and $B_{\rm ref}$ in Tesla); set to -1 for consistent evaluation from reference values if all of them are given in the `units` namelist;
if `beta` has a finite value, magnetic fluctuations are included in the calculation, independent of the choice of the geometry (and the α parameter); note that often, a small but finite β value can help to increase the time step significantly (this is due to the fact that for $\beta\to 0$, the

shear Alfvén wave transitions into a high-frequency electrostatic wave which needs to be resolved in time)

`coll [real]`: only considered if a valid `collision_op` has been selected. Normalized collision frequency defined in Gaussian units as

$$\nu_c = \pi \ln\Lambda \, e^4 n_{\mathrm{ref}} L_{\mathrm{ref}} / \left(2^{3/2} T_{\mathrm{ref}}^2\right)$$

which can be simplified to $\nu_c = 2.3031 \cdot 10^{-5} \left(\ln\Lambda L_{\mathrm{ref}} n_{\mathrm{ref}}\right) / \left(T_{\mathrm{ref}}^2\right)$ with $L_{\mathrm{ref}}[m]$, $n_{\mathrm{ref}}[10^{19}/m^3]$ and $T_{\mathrm{ref}}[keV]$ as in the `units` namelist; set `coll` = -1 for consistent evaluation from reference values if all of them are given in the `units` namelist.
Following Ref. [17], the Coulomb logarithm can be expressed as $\ln\Lambda = 24. - \ln(\sqrt{n_{\mathrm{ref}} \cdot 10^{13}}/(10^3 T_{\mathrm{ref}}))$, which is about 10 to 15, usually;
For reference, the conversion to the electron-ion collision rate of Hinton and Hazeltine[18] [their Eqs. (4.12) and (4.36)] is given by

$$\nu_{ei}(v) = \frac{2^{1/2}\pi Z^2 e^4 n_i \ln\Lambda}{m_e^{1/2} T_e^{3/2}} \frac{v_{Te}^3}{v^3} = 4Z^2 \frac{n_i}{n_{\mathrm{ref}}} \frac{T_{\mathrm{ref}}^2}{T_e^2} \frac{v_{Te}^3}{v^3} \frac{v_{te}}{L_{\mathrm{ref}}} \nu_c$$

with the thermal velocity $v_{te} = \sqrt{T_e/m_e}$ and the ion charge Z. Note: the thermal collision rate is obtained for $v = v_{Te} = \sqrt{2T_e/m_e}$, i.e. differing from v_{te} by a factor of $\sqrt{2}$.
The dimensionless collisionality of the main ion species $\nu_i^* = \frac{\sqrt{2}aB_0}{B_{p0} v_{T,i} \epsilon^{3/2} \tau_i}$ (ion version of Eq. 6.87 in Ref. [18]) can be expressed in terms of GENE parameters for a circular equilibrium as:

$$\nu_i^* = \frac{2^{7/2}}{3\sqrt{\pi}} \frac{qZ^4}{\epsilon^{3/2}} \frac{R}{L_{\mathrm{ref}}} \frac{n_i}{n_{\mathrm{ref}}} \frac{T_{\mathrm{ref}}^2}{T_i^2} \nu_c$$

with R/L_{ref} =`major_R`, ϵ =`trpeps` and q =`q0` from the `geometry` namelist and Z is the ion charge.
In a similar fashion, we obtain the electron version

$$\nu_e^* = \frac{16}{3\sqrt{\pi}} \frac{qZ^2}{\epsilon^{3/2}} \frac{R}{L_{\mathrm{ref}}} \frac{n_i}{n_{\mathrm{ref}}} \frac{T_{\mathrm{ref}}^2}{T_e^2} \nu_c$$

from Ref. [18].

`zeff [real 1.0]`: effective ion charge `zeff`$=n_e^{-1}\sum_i n_i q_i^2$ (only used in collisions); set to -1 to read from iterdb file with `prof_type`=-2 or -3; values of `zeff`$\neq 1$ are only valid, if no active impurity ion species is included; note that the effect of impurities can only partially be modeled by `zeff`

`debye2 [real 0.0]` squared debye wavelength normalized to ρ_{ref}
debye2 $= (\lambda_{db}/\rho_{\text{ref}})^2 = 5.2936 \cdot 10^{-4} \frac{B_{\text{ref}}^2}{n_{\text{ref},19}} \frac{m_p}{m_{\text{ref}}}$
($n_{\text{ref},19}$ in $10^{19}\frac{1}{m^3}$, B_{ref} in Tesla);
set to -1 for consistent evaluation from reference values if all of them are given in the `units` namelist

additional collision operator settings

`collision_op [str 'none']`: Collision operator, can be 'pitch-angle', 'landau' for Landau-Boltzmann, or 'sugama' for a Sugama-type collison operator (`mu_grid_type` ='eq_vperp' is recommended for the latter)

`coll_cons_model [str 'default']`: model for the back-reaction term of the collision operator conserving energy and momentum, can be 'self_adj' for a self-adjoint form, 'default' (='xu_rosenbluth', a little faster), or 'none' (not recommended). For collision_op = 'sugama', the only option is 'nakata'.

`coll_FLR [bool F]`: switch on/off FLR corrections in the 'sugama' collision operator. For 'landau' collisions, only spatial diffusion will be considered.

`coll_f_fm_on [bool F]`: treat f_1/F_0 as a single variable in the collision operator. Recommended for 'sugama' collisions.

`coll_on_h [bool F]`: apply collisions to the h distribution rather than f_1. Recommended for 'sugama' collisions.

hyper diffusion settings

`hyp_x [real 0.0]`: this parameter specifies the strength of the numerical dissipation (hyperdiffusion) in the radial direction (see also FAQ); by default, it is set to zero; (radially) global nonlinear simulation typically require a finite but small value `hyp_x_order [int 4]:` exponent of the hyperdiffusion operator in the radial direction; any even number is allowed

`hyp_y [real 0.0]:` this parameter specifies the strength of the numerical dissipation (hyperdiffusion) in the binormal direction (see also FAQ); by default, it is set to zero

`hyp_y_order [int 4]:` exponent of the hyperdiffusion operator in the binormal direction; any even number is allowed

`GyroLES [bool F]:` switches on/off an adaptive x and y hyperdiffusion which aims at mimicking the energy transfer to smaller, unresolved scales and is hence typically much more realistic and superior to manually chosen `hyp_x` and `hyp_y` in *nonlinear fluxtube* simulations not fully resolving the driven wave number range; see Refs. [24, 25, 26] for more details

`hyp_z [real 0.0]:` this parameter specifies the strength of the numerical dissipation (hyperdiffusion) in the parallel direction[21] (see also FAQ); by default, it is set to zero.

`hyp_z_order [int 4]:` exponent n of the hyperdiffusion operator D_n in the parallel direction; the latter is defined by $D_n = -i^n \, \texttt{hyp\_z} \, (\Delta z/2)^n \, (d/dz)^n$; one may choose between $n = 2$ and $n = 4$

`hyp_v [real 0.0]:` this parameter specifies the strength of the numerical dissipation (fourth-order hyperdiffusion) in the parallel velocity coordinate[21] (see also FAQ); it should not be used in the presence of a finite collision amplitude; by default, it is set to zero.

miscellaneous settings

`bpar [bool F]:` switches on/off the consideration of parallel magnetic fluctuations if `beta` is larger than zero by setting it to `.t./.f.`

`delzonal [bool F]:` this parameter may be used to artificially suppress all zonal components of ϕ (zonal flows) by setting it to `.t.`

`delzonal_fields [bool F]:` this parameter may be used to artificially suppress all zonal components of $A_\parallel$ (zonal fields) by setting it to `.t.`

`delzonal_bpar [bool F]:` this parameter may be set to true (T) to artificially suppress all zonal components of $B_\parallel$

`add_zonal_phi [real 0.0]`: this parameter may be used to define the amplitude of an artificial zonal component which is added to the electrostatic potential at the end of each field solve. In combination with `init_cond = 'zero'`, this may for instance be used to study residual zonal flow levels as predicted by Rosenbluth/Hinton and others.

`avgflux_stime [real -1.0]`: activates computation of time and volume averaged fluxes starting at time `avgflux_stime` if set to 0.0 or larger; this diagnostic is called within the `nrg` diagnostic, i.e. every `istep_nrg` time step.

`parscheme [str 'c4th']`: this parameter defines the finite differencing scheme in the parallel direction; one can choose between `'c4th'` (fourth-order centered) and `'u3rd'` (third-order upwind - requires `arakawa_zv= F` and `hyp_z`= 0); under normal circumstances, the first of these options is recommended but the second may be interesting in case of substantial wiggles in the parallel profiles even at high numbers of `hyp_z`.

`arakawa_zv [bool T]`: casts parallel (z) and parallel velocity ($v_\parallel$) derivatives into a Poisson bracket structure being solved with an Arakawa scheme[11] if set to true; separate finite-difference operators are used else.

`arakawa_zv_order [int 4]`: Determines the order of the Arakawa scheme used in z and $v_\parallel$ direction.

`arakawa_cons_bc [bool T]`: Switches on/off conservative boundary conditions in ballooning space. *Note: Dissipative boundary conditions are currently only implemented for* `arakawa_zv_order`= 2.

`check_qn_gradients [bool T]`: by default, the input density gradients will be required to fulfill quasi-neutrality. Set to false to avoid this check.

`perf_tsteps [int 3]`: defines the number of simulation time steps to be computed for the time measurements in the automatic `perf_vec` determination.

antenna drive settings

`antenna_type [int 0]`: Determines the type of antenna used to drive externally imposed currents. Four settings are possible:

0 – antenna off

1 – sinusoidal antenna applied to all modes

2 – Langevin antenna, included in the modified distribution function g_{1j}

3 – Langevin antenna, not included in g_{1j}, with explicit $\partial A_{1\|}/\partial t$ term (recommended)

`Apar0_antenna [real 0.0]`: Amplitude for type 1 antenna.

`omega0_antenna [real 0.0]`: Frequency for type 1 antenna.

`lv_antenna_modes [integer*N*3 (0,0,0)*N]`: Defines the modes driven by the Langevin antenna (see TenBarge et al. (CPC 2014). For each mode tuple (k_x,k_y,k_z), the amplitude is evolved using a separate Langevin equation. Specify modes in the form i_1,j_1,k_1,i_2,j_2,k_2 etc., where i, j, k are the indices of the k_x, k_y, k_z modes.

`lv_antenna_amp [complex*N (0,0)]`: Sets the amplitude for Langevin-type antenna drive. Amplitudes must be given for each mode tuple. For continuation runs, it is necessary to use the amplitude from the output parameters file (amplitude at the end of the last simulation) to enable a smooth continuation. In this case, the parameter `lv_antenna_initamp` *must* be specified to allow for an equal treatment of the random contribution to the Langevin equation.

`lv_antenna_initamp [complex*N (0,0)]`: Specifies the initial (=target) amplitude for Langevin-type antenna drive. This parameter is only required for continuation runs in order to allow a consistent setup of the random number contribution to the Langevin equation.

`lv_antenna_freq [complex*N (0,0)]`: Defines the frequency and decorrelation rates for each Langevin antenna. Decorrelation rates have to be explicitly specified as negative values, otherwise the antenna amplitudes will grow exponentially.

3.2.5 The `extended_diags` namelist

This namelist controls the extended diagnostic suite: the dfout diagnostic outputs δg at specific wavenumbers, the svd diagnostic reads these files and saves proper orthogonal decompositions, and the triplet diagnostic tracks wavenumber and distribution function structure ($g(z, v_\parallel, \mu, n)$) resolved free energy transfer. The dfout diagnostic saves the time step and the distribution function at a wavenumber every istep_dfout timesteps. The svd (singular value decomposition) diagnostic does not run a full GENE simulation, and instead takes these dfout files and outputs the singular value decomposition at that wavenumber. These runs must be set up as linear simulations with only a single k_y, and the same kymin as the corresponding nonlinear simulation. The triplet diagnostic takes distribution functions from files named EV_df_ky%4.4ikx%4.4i % (kxind, kyind), which contain mode numbers followed by the mode structure $g(z, v_\parallel, \mu, n)$. The triplet diagnostic will attempt to read n_dfs_tplts of modes from these files but will work if there are fewer or the file doesn't exist. Triplets are specified in a seperate file named 'triplets' in the problem directory consisting of lines per triplet specified by $k_x k'_x k''_x k_y k'_y k''_y$, where these are the indices of the wavenumbers to investigate transfer between. Negative k_x values should be referred to with negative indices, and negative k_y values should be avoided. The svd diagnostic produces files in this format named SVD_df_ky%4.4ikx%4.4i.

`n_tplts [integer 0]`: number of triplets to resolve energy transfer between modes

`split_em_tplt [logical True]`: output energy transfer involving ϕ and $A_\parallel$ seperately. Only applicable if n_fields 1.

`n_dfs_tplts [integer 1]`: calculate energy transfer between supplied distribution function structures (from svd or eigenvectors)

`SVD_df_file_path [str './']`: location of modes for use in the triplet diagnostic, and for dfout files for SVD diagnostic

`num_ky_modes [integer 0]`: number of independent wavenumbers to record for dfout diagnostic

`num_kx_modes [integer 0]`: number of connections in dfout files

`which_ky [integer*N 0]`: k_y index for dfout

`which_kx_center [integer*N 0]`: central k_x index for dfout

`SVD_proj [logical False]`: enables SVD projection routine

`SVD_f0_flag [logical True]:` normalizes inner product for SVD with F0 to be more like free energy

`SVD_kx_ind [integer*N 0]:` k_x wavenumbers index for SVD

`SVD_ky_ind [integer*N 0]:` k_y wavenumbers index for SVD

`SVD_nkx0 [integer*N 0]:` number of parallel connections for SVD

`n_svd_ind [integer*N 0]:` how many SVDs to calculate

`SVD_df_n_time [integer*N 0]:` number of timesteps to use

`SVD_start_time [real*N 0]:` time to start from dfout files

`SVD_sparse_factor [integer*N 0]:` skip every SVD_sparse_factor

`SVD_df_file_suffix [string*N 0]:` e.g. files named '*_1' '*_2' ...

`SVD_n_restarts [integer*N 0]:` number of files to use dfouts from

3.2.6 The `external_contr` namelist

This namelist is used to enter externally imposed equilibrium flows and stationary temperature and density profiles.

`ExBrate [real 0.0]:` This parameter defines a radially constant ExB shearing rate (determined by toroidal rotation), which is activated after exceeding `ExB_stime` and implemented by shifting the radial Fourier mode grid in time (see Ref. [19, 20]). The parameter is given in normalized units by `ExBrate`$= -\sigma_{I_p}\sigma_\Omega \frac{x_0}{|q_0|}\frac{\partial|\Omega_{\rm tor}|}{\partial x}\frac{L_{\rm ref}}{c_{\rm ref}}$, where x and x_0 are the radial coordinate and reference fluxsurface (simulation domain center) position in the GENE coordinate system ($\rho_{\rm tor}$ for realistic geometry, $r/L_{\rm ref}$ for circular/s-α/miller geometry), q_0 is the safety factor, $\partial\Omega_{\rm tor}/\partial x$ the radial derivative of the ***toroidal angular velocity*** $\Omega_{\rm tor}$ and σ_{I_p} and σ_Ω are 1/-1 if the plasma current/rotation is clockwise/anti-clockwise when viewed from above. A realistic value for a conventional tokamak would be e.g. `ExBrate`$=$ 0.05;
set to -1111.0 for consistent evaluation from reference values if all of them are given in the `units` namelist and an iterdb_file containing `VROT` has been specified.

`pfsrate [real 0.0]`: This parameter defines a shearing rate to model the parallel flow shear drive that appears due to toroidal rotation. Typically, it is used in combination with finite `ExBrate` and defined in the same way (additional geometric factors are added automatically by GENE). Hence, the recommended setting for this parameter is `pfsrate`$= -1111$, which will set the parallel flow shear equal to the `ExBrate` as it is given for a purely toroidal flow. Note that `pfsrate` is not affected by `ExB_stime`.

`Omega0_tor [real 0.0]`: Angular toroidal rotation in units of $c_{\text{ref}}/L_{\text{ref}}$; if set to finite values, simulations will be run in an accordingly rotating co-moving frame with Coriolis, Centrifugal and further comoving frame effects activated unless explicitly altered by `with_coriolis`, `with_centrifugal`, and `with_comoving_other`; use -1111.0 for consistent evaluation from reference value `omegatorref` if the latter is provided in the `units` namelist. Note that the sign may be overwritten if `sign_Omega_CW` is explicitly given.

`ExB_stime [real 0.0]`: With this parameter, the user can set the time (in simulation units) when ExB flow shear (determined by `ExBrate`) is activated - for instance, to a time where the unsheared turbulence has reached a quasi-stationary state. This is especially recommended for strong flow shear which would otherwise often lead to very long saturation times or an initial suppression of turbulence. Note that parallel flow shear is not affected by this parameter and will be active from the beginning of the simulation. `ExB_stime` does not have any impact on `pfsrate`.

`<var>0_ext [real 0.0]`: These parameters specify the amplitude for sinusoidal contributions to the electrostatic potential, the temperature profile or the density profile (set `<var>` to `phi`, `omt`, `omn` or `apar`, respectively).

`kxind_<var>_ext [int -1]`: These parameters define the wavenumbers of the sinusoidal contributions activated by the `<var>0_ext` parameters. The possible definitions of `<var>` are given above.

`phase_<var>_ext [int -1]`: These parameters define radial shifts (in rad) of the sinusoidal contributions activated by the `<var>0_ext` parameters. See above for possible definitions of `<var>`.

Optional switches for detailed comoving frame effect studies:

`with_coriolis [bool]`: Switch on/off Coriolis effects in the Vlasov equation; default is true for finite `Omega0_tor` and false else

`with_centrifugal [bool]`: Switch on/off Centrifugal effects in the Vlasov equation; default is true for finite `Omega0_tor` and false else

`with_comoving_other [bool]`: Switch on/off all implemented comoving frame effects beyond Coriolis and Centrifugal forces, e.g. the formation of an equilibrium potential in combination with poloidal equilibrium density asymmetries and the modification of the trapping and linear drive terms; default is true for finite `Omega0_tor` and false else
note: a reference position for the density normalization has to be defined via `R0_tor` if poloidal asymmetries are considered!

`R0_tor [real -1]`: Reference radial position for the equilibrium density normalization if poloidal asymmetries are taken into account; set to magnetic axis (-1), low field side radius (-2) or specify positive valued position by hand

`dR0_tor_dx [real]`: Radial derivative of `R0_tor` if the latter is positive valued

3.2.7 The `geometry` namelist

`magn_geometry [str]`: this parameter defines the magnetic geometry used in the computation. The analytical models supported are 'slab' (basic slab model), 'slab_curv' (slab with curvature), 's_alpha', 'circular' (circular concentric flux surface; $x = r$), 'zpinch', and 'miller'; to use the geometric information from (numerical) MHD equilibria choose 'chease', 'dipole', 'tracer_efit' or 'gist'; GENE geometry output files being defined by `geomfile` can be reread by choosing 'gene' (starting from release 1.6)

additional parameters for analytical equilibrium models 's_alpha', 'circular', and 'miller':

`q0 [real]`: safety factor q at the central radial position of the flux tube - note that the actually used value is `sign_Ip_CW` $\cdot$ `sign_Bt_CW` $\cdot$ $|q_0|$ to ensure consistency between the safety factor and the orientation of plasma current and toroidal field `shat [real 0.0]:` magnetic shear parameter as defined by $\hat{s} = (\rho/q)(dq/d\rho)$ with flux surface label ρ ($\rho = r$ for most models); important for parallel boundary condition; the value is overwritten in case of numerical equilibria

`amhd [real 0.0]:` α parameter as used in magnetohydrodynamics and defined by $\alpha_{\rm MHD} = -q^2 R\,(d\beta/dr)$ where $\beta = 8\pi p/B_{\rm ref}^2$; for `amhd=-1`, it is set automatically such that it is consistent with β and the temperature and density gradients of *active* species; otherwise, any other value can be chosen in order to study Shafranov shift effects in the geometry independently of *dynamical* finite β effects (associated with magnetic field fluctuations); note: 'circular' always assumes concentric ($\alpha_{\rm MHD} = 0$) flux surfaces

`major_R [real 1.0]:` major radius ($R/L_{\rm ref}$) of the device–for Miller geometry, of the flux surface–in units of the macroscopic length scale $L_{\rm ref}$

`major_Z [real 0.0]:` vertical elevation of the magnetic axis in units of the macroscopic length scale $L_{\rm ref}$

`minor_r:` minor radius of the device in units of the macroscopic length scale ($a/L_{\rm ref}$); only required for 'circular' and 'miller'

`trpeps [real 0.0]:` inverse aspect ratio at the flux tube position, $\epsilon = r/R$

`q_coeffs [real(6) 0.0]:` global code specific real valued array which specifies the safety factor profile as polynomial function up to 5th order. The first element is the coefficient of the x^0 term, the second element is the coefficient of the linear term, and so on. Currently only available in the `circular` equilibrium. The sign of the safety factor profile will be adjusted to be consisted with toroidal field and plasma current orientation.

additional parameters for 'miller':

`kappa [real 1.0]:` elongation κ

`delta [real 0.0]:` triangularity δ

`zeta [real 0.0]:` squareness ζ

`s_kappa [real 0.0]`: $s_\kappa = \frac{r}{\kappa}\frac{\partial \kappa}{\partial r}$

`s_delta [real 0.0]`: $s_\delta = \frac{r}{\sqrt{1-\delta^2}}\frac{\partial \delta}{\partial r}$; warning: other codes use $r\frac{\partial \delta}{\partial r}$ instead

`s_zeta [real 0.0]`: $s_\zeta = r\frac{\partial \zeta}{\partial r}$

`drR [real 0.0]`: shift of the major radius of the flux surface $\frac{\partial R}{\partial r}$

Note: The GENE implementation of Miller geometry uses $x = r$ as the radial coordinate. Thus, all input gradients (including `amhd`, `shat`, `ExBrate`) are expected to be given in terms of $\frac{d}{dr}$ derivatives. A Python script for an automatic determination of the Miller parameters from a G-EQDSK (EFIT) file can be found in `./tools/python/extract_miller_from_eqdsk.py`. The script will write a block of input parameters to standard output, which can be copied directly to a GENE parameters file. Note that presently, automatic profile input from files is not possible for Miller geometry. Contrary to most geometry interfaces, the Miller toroidal B_{ref} is not taken on axis but at the center of the flux surface (typically the difference is quite small).

additional parameters for 'miller_general':

This is a generalized version of the Miller equilibrium[27] where the flux surface shape is given by parametrizing the distance of the poloidal flux surface contour $a_N(r,\theta) = \sqrt{(R(r,\theta)-R_0)^2 - (Z(r,\theta)-Z_0)^2}/L_{\text{ref}} = \sum_N c_N(r)\cos(n\theta) + s_N(r)\sin(n\theta)$ and its radial derivative with respect to the reference point $(R_0, Z_0)/L_{\text{ref}} = (\texttt{major\_R}, \texttt{major\_Z})$. Here, θ is understood as poloidal angle running counter-clockwise in a poloidal plain to the right of the magnetic axis (COCOS=2/12) and $r = \frac{1}{2}\left[\max R(r,\theta) - \min R(r,\theta)\right]$.

`cN_m [real(32) 0.0]`: Cosine coefficients $c_N(r)$ for $a_N(r,\theta)$ (up to 32 comma-separated entries)

`sN_m [real(32) 0.0]`: Sine coefficients $s_N(r)$ for $a_N(r,\theta)$

`cNdr_m [real(32) 0.0]`: Cosine coefficients $\partial_r a_N(r,\theta)$

`sNdr_m [real(23) 0.0]`: Sine coefficients $\partial_r a_N(r,\theta)$

additional parameters for 'slab', 'slab_curv':

`q0 [real]`: safety factor q at the central radial position of the flux tube `shat [real 0.0]:` magnetic shear parameter as defined by $\hat{s} = (\rho/q)(dq/d\rho)$ with flux surface label ρ for sheared slab applications; important for parallel boundary condition

`parscale [real 1.0]:` defines the parallel scale length in units of L_{ref} for slab geometry
note: (a) this parameter enters only into the Jacobian and - in case of 'slab_curv' in the curvature drift; (b) for comparisons with [21] (diffusion study of slab ITG), one needs to rescale the gradient lengths and the growth rate with $\alpha_{\mathrm{i}}/\sqrt{2}$; (c) set `parscale` to `q0` for direct comparison with 's_alpha' at `trpeps`$\ll 1$ or renormalize results otherwise

additional parameters for 'dipole':

`r0_dipole [real 0.4]:` radius of the current ring [m]

`Icurr_dipole [real 0.4]:` current in the current ring [A]

`c_surf_dipole [real 1.2]:` surface to put the flux tube (measured in r0)

`c_dt_dipole [real 0.00001]:` time step for the tracing

additional parameters for 'tracer', 'tracer_efit' or 'gist':

`geomdir [str './']:` directory from which the geometry file is read

`geomfile [str]:` name of the geometry file; note that for 'tracer' a file generated by the TRACER code is required, whereas 'tracer_efit' will read an EFIT file, which will then be traced by a GENE-internal version of TRACER.

additional parameters for 'chease':

`geomfile [str]:` name of the geometry file (in hdf5 format), if no file name is given, the code looks for a file called ogyropsi.h5.

`x_def [str 'arho_t']:` Select definition of x variable. It can be set to 'arho_p' $= a\sqrt{\Psi_p/\Psi_{p\ \mathrm{edge}}}$ where Ψ_p is the poloidal flux, 'arho_t' $= a\sqrt{\Psi_t/\Psi_{t\ \mathrm{edge}}}$ where Ψ_t is the toroidal flux or 'arho_v' $= a\sqrt{V/V_{\mathrm{edge}}}$ where V is the volume inside a flux surface, a is the minor radius. It can also be set to 'C_psi' $= q0/(r_0 B_0)\Psi_p$ where r_0 and q_0 are the local minor radius and safety factor, and B_0 the magnetic field at the axis. The choice of x_def, should be considered when setting omn=$-L_{\mathrm{ref}}\, d\ln n/dx$ and omt=$-L_{\mathrm{ref}}\, d\ln T/dx$.

Miscellaneous:

`dpdx_pm [real]`: amplitude of the *negative* equilibrium pressure gradient $-\nabla p/p_m$ normalized to the reference magnetic pressure $p_m = B_{\mathrm{ref}}^2/(2\mu_0)$ or $B_{\mathrm{ref}}^2/(8\pi)$, respectively. Set to

≥ 0 for user-defined value

-1 for automatic computation from density and temperature gradients of *active* species via `dpdx_pm` $= \beta_{\mathrm{ref}} \sum_\sigma \texttt{dens}_\sigma \texttt{temp}_\sigma (\frac{L_{\mathrm{ref}}}{L_{T_\sigma}} + \frac{L_{\mathrm{ref}}}{L_{n_\sigma}})$

-2 for automatic computation from MHD equilibrium (*recommended* if available!)

`dpdx_term [str]`: defines the way how the pressure gradient is considered or neglected in the combined curvature and ∇B drift $\mathbf{v}_{\mathrm{curv}+\nabla B}$

- `'full_drift'`: (*recommended* for simulations with $B_\parallel$ fluctuations); consider full drift velocity

$$\begin{aligned}\mathbf{v}_{\mathrm{curv}+\nabla B} &= \mathbf{b} \times \left[\left(v_\parallel^2 + \frac{v_\perp^2}{2} \right) \frac{\nabla B}{B} + v_\parallel^2 \frac{4\pi}{B^2} \nabla p \right] /\Omega \\ &= \mathbf{b} \times \left[\left(v_\parallel^2 + \frac{v_\perp^2}{2} \right) (\mathbf{b}\cdot\nabla)\mathbf{b} - \frac{v_\perp^2}{2} \frac{4\pi}{B^2} \nabla p \right] /\Omega\end{aligned}$$

  requires `dpdx_pm` to be set

- `'gradB_eq_curv'`: (often *recommended* for simulations without $B_\parallel$ fluctuations); erase pressure gradient from ∇B drift, i.e.

$$\mathbf{v}_{\mathrm{curv}+\nabla B} \sim \mathbf{b} \times \left[\left(v_\parallel^2 + \frac{v_\perp^2}{2} \right) \frac{\nabla B}{B} + \left(v_\parallel^2 + \frac{v_\perp^2}{2} \right) \frac{4\pi}{B^2} \nabla p \right] /\Omega$$

  in the $\nabla B/B$ formulation employed in `Gene`; requires `dpdx_pm` to be set

- `'curv_eq_gradB'`: erase pressure gradient from curvature drift (formerly the default in GENE if `pressure_term = F` had been set)

$$\mathbf{v}_{\mathrm{curv}+\nabla B} \sim \mathbf{b} \times \left[\left(v_\parallel^2 + \frac{v_\perp^2}{2} \right) \frac{\nabla B}{B} \right] / \Omega$$

  `dpdx_pm` is not required

`sign_Ip_CW [int]`: defines the sign of the plasma current (clockwise=1, counter-clockwise=-1) in a right-handed $(\rho, \theta_{\mathrm{pol}}, \phi_{\mathrm{tor}})$ coordinate system such that `magn_geometry` = [`s_alpha`, `circular`, `miller`, `tracer_efit`] will be adjusted accordingly. The default for these geometries is 1 (clockwise).

`sign_Bt_CW [int]`: defines the sign of the toroidal magnetic field component (clockwise=1, counter-clockwise=-1) in a right-handed $(\rho, \theta_{\mathrm{pol}}, \phi_{\mathrm{tor}})$ coordinate system such that `magn_geometry` = [`s_alpha`, `circular`, `miller`, `tracer_efit`] will be adjusted accordingly. The default for these geometries is 1 (clockwise).

Optional:

`sign_Omega_CW [int 0]`: set to $1/-1$ to overwrite the sign of the toroidal angular velocity such that clockwise/counter-clockwise rotation is ensured in a in a right-handed $(\rho, \theta_{\mathrm{pol}}, \phi_{\mathrm{tor}})$ coordinate system. Requires `sign_Ip_sign` and `sign_Bt_sign` to be defined.

`norm_flux_projection [bool F]`: Use normalized/unnormalized radial projections when computing radial fluxes in the `nrg` file (see Sec. 4.1).

`rhostar [real]`: defined as $\rho^* = \rho_{\mathrm{ref}}/a$ with minor radius a being set by `minor_r` or the numerical equilibrium interfaces; set to -1 for consistent evaluation from reference values if all of them are given in the `units` namelist; note that `rhostar` is not required in fluxtube simulations unless certain parameters like `n0_global` shall be set consistently w.r.t. ρ_{ref} and a.

`edge_opt [real 0.0]`: Only available for 'miller' and 'tracer_efit' geometry types, this parameter allows to redistribute the parallel grid points to concentrate more strongly on the outboard side (around $z = 0$), where instabilities typically peak. This is particularly useful in strongly shaped geomtries such as in the tokamak edge. The input parameter is a real number that adjusts the degree of redistribution smoothly. Typical numbers for this parameter range between 1 and 4. With the default edge_opt=0, the grid points are equidistant in the straight field line angle. See Chapter 4.3 of Ref. [23] for more details.

3.2.8 The `species` namelist

`name [str]:` name of the particle species in the present namelist (may be chosen freely)

`passive [bool F]:` if this parameter is set to `.t.`, the particle species is only treated as a passive tracer, i.e., no fields generated by this species are computed and there are no back-reactions on the turbulence

`omn [real 0.0]:` normalized density gradient of the present particle species, $\omega_n = -(L_{\rm ref}/n)\,(dn/dx)$; if one chooses `major_R`=1.0, one has $\omega_n = R/L_n$

`omt [real 0.0]:` normalized temperature gradient of the present particle species, $\omega_T = -(L_{\rm ref}/T)\,(dT/dx)$; if one chooses `major_R`=1.0, one has $\omega_T = R/L_T$

`mass [real]:` mass of the present particle species, normalized to an arbitrary mass scale $m_{\rm ref}$

`charge [real]:` signed charge of the present particle species, normalized to the elementary charge (electrons: `charge=-1`)

`temp [real 1.0]:` temperature of the present particle species, normalized to an arbitrary temperature scale $T_{\rm ref}$

`dens [real 1.0]:` density of the present particle species, normalized to the electron density; in order to ensure quasineutrality, the condition $\sum \texttt{charge} \cdot \texttt{dens} = 0$ (sum over all *active* species) must be satisfied

`prof_type [int 0]:` the following options are available:

0 use the aforementioned user-defined `omn`, `omt`, `temp` and `dens` settings; for local code only

-1 use profile data provided in a file named `profiles_<species name>` for temperature, density and their gradients; for further information on the actual file format, see `src/profiles.F90`, subroutine `read_tempdens_profile`

-2 use profile data provided in the `iterdb_file` (`u-file`-like) for temperature, density and their gradients; the relevant entries read by GENE are `NE`, `TE`, `NM1` (,`NM2`), `TI`, `VROT` and `ZEFFR`; the file format is discussed here `http://tokamak-profiledb.ccfe.ac.uk/`. A GENE specific extension is the possibility to use different ion species temperatures by adding a corresponding index, i.e. `TI1`, `TI2`, etc.

-3 use DIII-D iterdb (onetwo) style profile data provided in the `iterdb_file` for temperature, density and their gradients; the relevant entries read by GENE are `nj`, `electron temperature`, `ion temperatue` (sic), `electron density`, `primary ion density` (, `impurity ion density`), `angular rotation speed`, and `zeff profile`

$1-5$ analytical profiles for *global* simulations

Be careful to use *consistent* radial coordinates in your profile files and the magnetic geometry. In most cases, the normalized ρ_{tor} is employed for this purpose.

For each particle species, a separate `species` namelist has to be provided; GENE will then use the first `nspec` such namelists, ignoring others (if present).

3.2.9 The `units` namelist (optional)

The original purpose of this namelist was to allow post-processing tools to convert the normalized observables to SI units as those values had just been passed to the output `parameters.dat` file. However, the reference values being described in the following can now also be used to automatically compute physical input variables like `beta` or `coll` if **all** of them are either explicitly set or consistently evaluated from given profile files or interfaces (see below).

`Tref [real 0.0]:` reference temperature in keV; set to -1 to automatically evaluate and consider T_e at radial domain center if `prof_type` is properly set (< 0) in *each* species namelist.

`nref [real 0.0]:` reference density in $10^{19}\,\mathrm{m}^{-3}$; set to -1 to automatically evaluate and consider n_e at radial domain center if `prof_type` is properly set (< 0) in *each* species namelist.

`mref [real 0.0]:` reference mass in units of the proton mass; set to -1 for deuterium.

`Bref [real 0.0]:` reference magnetic field in T; will always be filled automatically if the magn_geometry is set to `tracer`, `tracer_efit` or `chease`.

`Lref [real 0.0]:` reference length in m; will always be filled automatically if the magn_geometry is set to `tracer`, `tracer_efit` or `chease`.

`omegatorref [real 0.0]:` reference toroidal angular velocity in rad/s; set to -1 to automatically evaluate from data base if `prof_type` is properly set (< 0) in *each* species namelist; needed for Doppler shifted post-processing.

3.2.10 The `nonlocal_x` namelist (global code only)

This namelist can be omitted in fluxtube (local) simulations as it contains parameters controlling heat/particles sources and sinks, buffer zone definitions and further features that are specific to (radially) global simulations, i.e. if `x_local` is set to false.

radial boundary condition

`rad_bc_type [int 1]:` tested options

- `0` periodic (should not be used with radial profiles; implemented for numerical reason)
- `1` Dirichlet boundary condition: $\partial_x f_1|_{x\in\mathcal{B}} = 0$ at boundary $\mathcal{B}$ for all fluctuating quantities.

buffer zone specifications

Particularly in case of Dirichlet boundary conditions, fluctuations should be small close to the boundaries for consistency. This can be achieved, for instance, by a simple Krook operator $-\nu_{\mathrm{Krook}}(x)g_1$ with $\nu_{\mathrm{Krook}}(x) \geq 0$ being a polynomial function which is increasing towards the boundary starting from a predefined position determined by the portion of the simulation domain devoted to the buffer zone, see below.

`l_buffer_size [real 0.0]:` portion of the simulation domain devoted to the inner buffer zone. A typical value would be ~ 0.1.

`u_buffer_size [real 0.0]:` portion of the simulation domain devoted to the outer buffer zone. A typical value would be ~ 0.1.

`lcoef_krook [real 0.0]:` amplitude for the inner krook buffer operator at the boundary - should be larger than the maximum linear growth rate.

`ucoef_krook [real 0.0]:` amplitude for the outer krook buffer operator at at the boundary - should be larger than the maximum linear growth rate.

`lpow_krook [int 4]:` polynomial order for the inner krook buffer operator.

`upow_krook [int 4]:` polynomial order for the outer krook buffer operator.

`buffer_on_ky0 [bool F]:` applies buffer operator only to $n = 0 (k_y = 0)$ mode if set to true and to all modes else

`explicit_buffer [bool]:` use explicit treatment of buffer operator (default for`buffer_on_ky0 = T`) for consideration in the `srcmom` diagnostic or to ensure numerical stability; otherwise, the treatment will be implicit to avoid constraints on the (linear) time step

krook-type heat and particle source/sinks

These operators keep temperature and density profiles fixed on average and therefore allow gradient-driven simulations (in contrast to profile relaxation simulations without any additional source or sink or flux-driven simulations with localized sources/sinks and floating boundary conditions). Ideally, only long-term, long-wavelength dynamics is affected.

`ck_heat [real 0.0]:` amplitude of the krook-type heat source, see Refs. [22, 7, 23], in $c_{\mathrm{ref}}/L_{\mathrm{ref}}$. If no further filters are applied, the value should be significantly smaller than the linear growth rates in order to affect only the long-time dynamics.

`ck_part [real 0.0]`: amplitude of the krook-type particle source[23] in $c_{\mathrm{ref}}/L_{\mathrm{ref}}$ which is needed in case of gyrokinetically treated ions and electrons. If no further filters are applied, the value should be significantly smaller than the linear growth rates in order to affect only the long-time dynamics.

`psource_type [int 2]`: implemented options:

- `1` any additional heat source contribution stemming from the particle source will be removed by a corresponding intrinsic correction of the heat source amplitude
- `2` slightly different particle source which does not enforce/consider the aforementioned correction [needs further explanation]

`ck_heat_smooth [real 0.0]`: width of a radial smoothing window for the krook-type heat source relative to the total radial simulation domain (spatial filtering). Default: no smoothing

`ck_part_smooth [real 0.0]`: width of a radial smoothing window for the krook-type particle source relative to the total radial simulation domain (spatial filtering). Default: no smoothing

`ck_filter_type [int 0]`: type of smoothing operator being employed for spatial filtering; implemented options:

- `0` boxcar average
- `1` Gaussian filtering

`intsource_heat_coeff [real 0.0]`: amplitude of time-integrated krook-type heat source in $c_{\mathrm{ref}}/L_{\mathrm{ref}}$ which can be used alternatively to `ck_heat`.

`intsource_part_coeff [real 0.0]`: amplitude of time-integrated krook-type particle source in $c_{\mathrm{ref}}/L_{\mathrm{ref}}$ which can be used alternatively to `ck_part`.

`intsource_time [real 1000.0]`: characteristic scale for time integrator in $L_{\mathrm{ref}}/c_r f$; should be on the order of a few correlation times

buffers for krook-type heat and particle source/sinks [optional]
In case of interference between the krook buffer operator and the krook-type

heat and particle sources, the latter can be smoothly turned off towards the boundaries using the following parameters.

`lck_buffer_size [real 0.0]:` innermost portion of the simulation domain where the heat and particle sources are artificially reduced near the boundary; typically similar to `l_buffer_size`

`uck_buffer_size [real 0.0]:` outermost portion of the simulation domain where the heat and particle sources are artificially reduced near the boundary; typically similar to `u_buffer_size`

`lckcoef_krook [real 0.0]:` krook-type heat and particle source amplitude at inner boundary (i.e., after reduction)

`uckcoef_krook [real 0.0]:` krook-type heat and particle source amplitude at outer boundary (i.e., after reduction)

`lckpow_krook [int 4]:` polynomial order for the inner heat and particle sources buffer operator.

`uckpow_krook [int 4]:` polynomial order for the outer heat and particle sources buffer operator.

miscellaneous

`shifted_metric [bool F]:` enables shifted metric approach instead of default parallel boundary condition if set to true; see Ref. [23] for a discussion

`drive_buffer [bool F]:` apply exponential smoothing with krook buffer function $\nu_{\mathrm{krook}}(x)$ to temperature and density profiles if set to true – this way, the gradient drive itself is reduced near the boundaries.

`ga_spatial_var [bool F]:` account for slight spatial variations of the metric coefficients in the gyro-average operator if set to true

3.2.11 The `bsgrid` namelist (global code only)

This namelist can be omitted in fluxtube (local) simulations. It is only relevant for radially global simulations with substantial temperature and hence thermal velocity changes where block-structured grids (BSG) in the velocity space[28] may reduce the numerical effort significantly. Many of the parameters below strongly depend on the particular profiles. It is therefore recommended to use the follwing workflow invoking a dedicated pre-processor rather than setting the parameters by hand

1. create a `gauss_quadrature.so` by calling
   `f2py -c --fcompiler=intelem -m gauss_quadrature gauss_quadrature.F90`
   in the `tools/bsg_prepost/grids` subdirectory
   (try, e.g., `fcompiler=gfortran` if `intelem` doesn't work)

2. run the grid-generator on a given input file via
   `python $GENEDIR/tools/bsg_prepost/bsgrid.py cons parameters --plt`
   (the flag `--plt` is optional for plotting, `$GENEDIR` should be replaced by the GENE base directory)

3. clean up and adjust parameters file

It is important to note that the BSG-grid optimizer is currently only considering the first species. Hence, you may want to manually compare the resulting grids from switched species namelists.
Furthermore, please note that `prof_type`< -1 is not supported directly. Instead, one may call Gene once with `ntimesteps`$= 0$ to obtain Gene-style `profile_<species name>` files. These may then be copied into the same directory as the `parameters` file to be adjusted and taken into account by temporarily setting `prof_type = -1` while launching the `bsgrid.py` tool.

block-structured grid parameters

`is_bsg [bool F]`: switch on/off block-structured velocity space grids in global simulations (`x_local = F`)

`is_nv0_fixed [bool]`: flag to fix `nv0` (number of $v_\parallel$ grid points)

`is_nw0_fixed [bool]`: flag to fix `nw0` (number of μ grid points)

`bbfdscheme [str]`: FD scheme for grid block boundaries interpolation

`gtype [str]`: type of ghost cell communication: 'nonblocking', 'onesided'

`nblks [int]`: number of blocks for a block structured r-($v_\parallel, \mu$) subgrid

`vp_std [real]`: width of the $v_\parallel$ grid in terms of standard deviations `blk_mks_r [real(nblocks+1)]:` block marks in radial direction

`lv_mks [real(nblocks+1)]:` block-range marks in $v_{\|}$-direction

`lw_mks [real(nblocks+1)]:` block-range marks in μ-direction

`nblocks_bsg [int]:` number of blocks for BSG-x-exchange

`is_nblocks_bsg_fixed [bool]:` flag whether `nblocks_bsg` should be fixed (no optimization)

3.3 Running the code

After modifying the `parameters` file (make sure that input files and output directories exist), GENE can either be started interactively using the appropriate command (mpiexec, mpirun, poe, aprun etc. depending on your machine) or be submitted to a batch queue. In any case, you have to specify the total number of MPI processes, which is given by $N_{\mathrm{MPI}} = \texttt{n\_procs}_{\texttt{s}} \cdot \texttt{n\_procs}_{\texttt{w}} \cdot \texttt{n\_procs}_{\texttt{v}} \cdot \texttt{n\_procs}_{\texttt{y}} \cdot \texttt{n\_procs}_{\texttt{z}}$. For shared memory architectures, the number of OMP processes can be set independently by setting the environment variable OMP_NUM_THREADS; note that the total number of processors needed is then N_{MPI}*OMP_NUM_THREADS. At the moment, OMP_NUM_THREADS = 1 is recommended.

3.3.1 Continuing a simulation

Particularly nonlinear simulations may not have reached sufficient statistics, e.g., when hitting the wall clock time limit (see `timelim`). In such cases, one or more follow-up runs may be necessary which requires the following steps:

- Rename the GENE output files in the chosen `diagdir` *after each individual run* as the new data will **not be appended** but rather overwrite the existing one. The advantage are smaller chunks of data, e.g., when transferring GENE outputs to other machines, and better tracking/identification of follow-up runs. You may consider using the `runassign` script. For more information, visit the tools section. Please note that `checkpoint` files are typically not renamed due to their large file sizes. In case of expensive simulations requiring many follow-ups, manual backups of that file are, however, encouraged.
- Set `read_checkpoint` in your GENE input file to true (`T`).
- Note that you may *restart your simulation with a different number of processes* within the constraints set by your `parallelization namelist` choices.
- If a continuation with exactly the same number of processes, grid numbers and box sizes is intended *and* the initialization took a substantial time of the initial run, you may copy the `n_procs_*`, `perf_vec`, and `nblocks` entries from the output. The same holds for `dt_max`.

- The `checkpoint` interface supports interpolation along all phase space coordinates but not in the number of species. You may therefore change grid numbers, e.g., when moving to another architecture where different factors are required for parallelization or when gradually increasing resolution within a convergence check. Significant changes might, however, trigger a new transient phase. Note that the post-processing routine may not be able to cope with follow-up runs having different resolution and you may be forced to separate analyses instead.

- When running (nonlinear) sensitivity scans w.r.t to the physics inputs, you may start with the checkpoint of the nominal parameter set. This way, you might be able to avoid transient phenomena like an overshoot and the associated (temporary) reduction of the nonlinear time step. Particular care has to be taken to ensure that the new simulation is sufficiently long to indeed establish the new turbulent state.

- Changes in the box sizes and parameters affecting the latter (e.g., `shat`) are not encouraged. The distribution function will not be inter- or extrapolated such that the results may strongly differ. Still, this approach may be convenient compared to restarting from one of the available initial conditions if the box size changes are rather small.

3.4 The scan script

For Gene parameter variations, use of the scan script is recommended. It can be found in the `prob` directory. Instructions for specification of parameter ranges are given below. Multidimensional parameter scans are possible. The scanscript also enables one to let parameters depend on other parameters. For each scan a new directory named `scanfiles##` in the specified `diagdir` is created, where `##` is an incremented number. If, for example, the directory `"scanfiles0000"` exists, `"scanfiles0001"` is created. The output files are stored in this `scanfiles` directory. Each parameter combination simulated is assigned a run number, which acts as a file ending of all the corresponding output files. In addition, a file called `scan.log` is created at the end of the scan, briefly summarizing the results (run number, the changed parameters, and growth rate / frequency data). This (and other) data can be analyzed using the scan tab in the diagnostic tool. Incomplete scans may be continued by adding the `--cs` option when calling the script (details below). Similar to the `scan.log`, the `neo.log` file is created for neoclassical computations, summarizing the last lines of `neoclass` output files.

It is strongly recommended to familiarize with the GENE option of running multiple GENE subroutines simultaneously to avoid waste of computational resources. Please read Sec. 3.4.3 for correct use. The point is that automatic settings may lead to idling processors.

To initiate a GENE scan, one has to change the lines in the `parameters` file (or the launcher tool) as specified below. The scanscript is executed by replacing the last line of the submit script (the GENE call command) by the following line ./scanscript `--np` ⟨`np`⟩ `--mps` ⟨`mps`⟩ , where ⟨`np`⟩ is the number of processes of the MPI environment, ⟨`mps`⟩ is the maximum number of parallel scans. For more options see Sec. 3.4.2.

3.4.1 How to specify scan parameters:

- ⟨par⟩ = ⟨value⟩ !scanrange: ⟨start⟩, ⟨step⟩,⟨end⟩

  The last three values (start, step, and end) specify the interval to be scanned. The value after the "=" will be replaced by the actual values when running the script. In this fashion, all parameter lines that are to be scanned over need to be adjusted.

  Example:

  ```
  omn = 3.0 !scanrange:  4.0,1.0,10.0
  ```

- ⟨par⟩ = ⟨value⟩ !scan: ⟨start⟩, ⟨step⟩,⟨end⟩

  Performs the same operation as !scanrange: but is not to be confused with the following functionality.

- ⟨par⟩ = ⟨value⟩ !scan: F(⟨pars⟩(⟨#⟩))

  A parameter can depend on other variable or fixed parameters. Scanscript evaluates the function F before every GENE start and assigns the result to the parameter ⟨par⟩. After every argument of F, a bracketed number is mandatory: if the argument parameter under consideration is part of a species namelist, this is the species number, otherwise it is "1". This also works in functions and lists.

  Example:

  ```
  omn = 3.0 !scan:  omn(1)+2*omt(2)-beta(1)
  ```

- ⟨par⟩ = ⟨value⟩ !scanlist: ⟨value1⟩,⟨value2⟩,...

  The scanscript also accepts comma separated lists for scans.

  Examples:

  ```
  nz0 = 1.0 !scanlist:  24,32,48
  ```

  List elements are also allowed to depend on other parameters:

  ```
  omt = 1.0 !scanlist:  omn(1)+1,4,3,8
  ```

  List elements can be character strings :

  ```
  which_ev = 'jd' !scanlist:  "'jd'","'harmonic'"
  ```

- ⟨par⟩ = ⟨value⟩ !scanfunc: ⟨maxsteps⟩,⟨f(xi)⟩,⟨end⟩

  One can use variable steps by specifying a (strictly monotonic) function f(xi), a maximum number of steps maxsteps, and a maximum for f(xi) above which no further steps are taken. If one of these two conditions (exceeded number of steps or maximum value) are fulfilled, the scan terminates at that point. The scan always starts with xi = 1 and increments 1 with every step. Possible operators in the function are `+, -, *, /, %, e, **, abs, sqrt, int, exp, log, sin, cos`. Parameters can be inserted as described further down.

  Example:

  ```
  omn = 3.0 !scanfunc:  10,log(xi)+omn(1),100
  ```

  One has to take care that if there are parameters in f(xi) that are changed during the scan, the number of steps in this dimensions could vary. Also, in multidimensional scans, if f(xi) changes with the change of another dimension, some analyses of the `scan.log` file in the diagnostic may be affected.

  Should the steps width of 1 in x not be sufficient, one can also create a dummy parameter which can be changed arbitrarily. Please take note that the dummy parameter's name has to start with an exclamation mark ("!") in order to be recognized by GENE as such.

  Example:

  ```
  !dummy = 0 !scan:  0, 0.1, 5
  ```

  ```
  omn = 0 !scan:  omn(1)+2*omt(2)-!dummy(1)
  ```

- ⟨par⟩ = 'str_L1('strs') !replace: L2(⟨pars⟩(⟨#⟩))

  For string valued parameters, a '_' separated list (L1) may be replaced with the values of a ',' separated parameter list (L2). The lengths of L1 and L2 must match.

  Example:

  `geomfile = 'tracer_efit_XX_YY' !replace:  x0(1),nz0(1)`

  Here, for `x0=0.5` and `nz0=32`, the geometry file 'tracer_efit_0.5_32' must exist.

3.4.2 Options (short form in brackets):

- `--n_pes=⟨#⟩ (--np=⟨#⟩)` It is required to set ⟨#⟩ the total number of processes the script is started with (MPI processes $\times$ openMP processes)

- `--procs_per_node =⟨#⟩ (--ppn=⟨#⟩)`

  (recommended) ⟨#⟩ is the number of processors per compute node of your machine, used for parallel efficiency optimization.

- `--max_n_par_sims =⟨#⟩ (--mps=⟨#⟩)`

  (recommended) ⟨#⟩ is the maximum number of parallel simulations, used for parallel efficiency optimization.

- `--help`

  Prints a short help text

- `--syscall`

  Set the command for starting GENE , if the default does not work. This depends on the machine.

- `--test`

  Test the scanscript settings without executing GENE

- `--force (--f)`

  Continue even when errors occur (`--test --force` is useful).

- `--continue_scan (--cs)`

  If the script was terminated prematurely, it can be restarted using this option. Make sure that the entries of the `&scan` namelist are correct:

  `scan_dims` is a vector giving the dimensions of the scan

  `par_in_dir` is the subdirectory `in_par` of `diagdir`. This directory contains all parameter files GENE is started with.

  These parameters are automatically set by the scanscript at the first submission of the scan.

- `--mk_scanlog (--mks)`

  Creates the scan.log from existing data in the scanfiles directory without running GENE
  (execute in probdir from command line without specifying `--n_pes`).

- `--efficiency (--eff)`

  Test the parallel efficiency for your problem. All scans are ignored. The maximum number of processes is the value given with `--n_pes`. GENE autoparallelization should be used.
  It is recommended to use the option `--procs_per_node` (processes per node of the machine) and `--min_procs` (minimum number of processes that should be tested).
  depending on the parameter `comp_type`, either initial value computations are run 10 timesteps with `dt_max`$= 1e-6$ or a single eigenvalue solver iteration is performed to test the speed.
  An efficiency.log file is created in the scan output directory. This file contains the resulting time per step and cpu-time per step. Having access to gnuplot, you can plot the efficiency data using the effplot script belonging to the GENE tools, which also creates a `.ps` file:

  ```
  effplot efficiency.log
  ```

  Note that using too many processors can not only be inefficient, but also can slow down the computation in real time!

- `--noeff`

  Suppresses the parallel efficiency test, all relevant parameters have to be set manually.

3.4.3 GENE parallelization over parameter sets

GENE is able to call multiple instances of a GENE subroutine simultaneously, so that different points in the specified parameter space can be computed in parallel. This is particularly useful for large parameter scans, since parallel efficiency of a single GENE run typically decreases with the number of processors.

For this feature, two additional parameters in the "&parallelization" namelist exist.

`n_procs_sim` is the total number of mpi processes one instance of GENE uses

`n_parallel_sims` is the number of GENE subroutines running in parallel

The product of `n_procs_sim` and `n_parallel_sims` must equal the total number of processors of the MPI environment (`--n_pes`).

If you are not sure which values of `n_procs_sim` and `n_parallel_sims` you should use, leave them blank to let the scanscript perform a parallelization efficiency test for your problem. In this case you must use the autoparallelization functionality by setting some or all other entries of the `&parallelization` namelist to < 1. The result is found in the efficiency.log in the output directory. However, low dimensional parameter scans should use `n_parallel_sims`=1 to make sure that processors do not idle at the end of the scan. For the same reason, `n_parallel_sims`=1 is also recommended, when each single point is computationally expensive.

A waste of resources occurs in the following example, the parameter $ky0$ is to take 9 values in the scan. Each run performs best at a total processor number of 16. The scan is submitted on 128 processors. Then, 8 runs will start simultaneously and finish likely about the same time. The remaining run is then starting while 112 processors have to idle! It would in this case be better to either submit on less nodes, or to use more processes per gene run (32 maybe) although parallel efficiency is not optimal. Consider to set `--min_procs` of `--max_n_par_sims` for the initial efficiency test.

3.4.4 Autoparallelization and scanscript

A performance optimization is done before the actual scan. The result is then used for all following calls of GENE . Even at fixed values of the `parallelization` namelist, the parameters `n_blocks` and `perf_vec` are determined.

The optimal combination of parallelization and `perf_vec`, however, changes under the following conditions:

- Box parameters `nx0, nky0, nz0, nw0, nv0` are scanned over.
- `beta=0`(`coll=0`) occurs in a `beta`(`coll`) scan.

In these cases, the scanscript lets the autoparallelization settings given in the parameters file unchanged.

3.4.5 Checkpoint reading in scans

Reading checkpoints is useful in the following cases

- An unfinished scan is restarted with the `--continue_scan` option. Then the scanscript automatically lets GENE try to read `s_checkpoint` files.
- Checkpoints of finished GENE simulations of a running scan are to be used as initial conditions for following simulations. This speeds up computation, in particular for linear eigenvalue runs, but is not recommended for initial value simulations. Set `read_checkpoint` = T to use this feature.
- One checkpoint is to be read as an input for all runs of a scan. Set `chpt_in = '<full path>'` in the `&scan` namelist and `read_checkpoint` = T in the `&in_out` namelist.

4 Output files

The main output files are the `nrg`, the `field` and the `mom` files. The first contains the timetrace information, such as density, temperatures and the transport fluxes, all spatially averaged. The `field` and the `mom` files contain 3D information about fields and moments of the distribution function and become rather large. For this reason they are only written at some time steps (typically all 100-500). In addition, some more output files and diagnostics exist which will be explained below.

4.1 The `nrg` file

The entries in this file are sequentially written in ASCII for each time step divisible by `istep_nrg`. An `nrg` entry looks like

```
596.505348
2.4862E+01   4.4911E+00   2.1926E+01   1.2346E+02   1.8177E+00   1.1874E-02   ...
3.5781E+01   1.7368E+02   2.3356E+01   1.2562E+01   1.8177E+00   1.1874E-02   ...
```

The first line of each data block contains the corresponding simulation time, followed by as many lines and in the same order as species are defined in the `parameters` file. The columns refer to the spatially averaged[2], normalized fluctuating quantities

$$\frac{\langle|n_1|^2\rangle}{(n_0\rho^*_{\rm ref})^2} \quad \frac{\langle|u_{1\|}|^2\rangle}{(v_T\rho^*_{\rm ref})^2/2} \quad \frac{\langle|T_{1\|}|^2\rangle}{(T_0\rho^*_{\rm ref})^2} \quad \frac{\langle|T_{1\perp}|^2\rangle}{(T_0\rho^*_{\rm ref})^2} \quad \frac{\langle\Gamma^x_{\rm es}\rangle}{\Gamma_{\rm gb}} \quad \frac{\langle\Gamma^x_{\rm em}\rangle}{\Gamma_{\rm gb}} \quad \frac{\langle Q^x_{\rm es}\rangle}{Q_{\rm gb}} \quad \frac{\langle Q^x_{\rm em}\rangle}{Q_{\rm gb}} \quad \frac{\langle\Pi^x_{\rm es}\rangle}{\Pi_{\rm gb}} \quad \frac{\langle\Pi^x_{\rm em}\rangle}{\Pi_{\rm gb}}$$

with units being detailed in Sec. A. The velocity space moments of the fluctuating part of the *particle* distribution function $f^{(pc)}$ and the particle, heat parallel momentum, and toroidal angular momentum fluxes are defined as

$$n_1 = \int d^3v \; f_1^{(pc)} \qquad \mathbf{\Gamma} = \int d^3v \; f_1^{(pc)} \; \mathbf{v}_D$$

$$u_{1\|} = \frac{1}{n_0}\int d^3v \; v_\| f_1^{(pc)} \qquad \mathbf{Q} = \int d^3v \; \frac{1}{2}mv^2 f_1^{(pc)} \; \mathbf{v}_D$$

$$T_{1\|} = \frac{m}{n_0}\int d^3v \; (v_\| - u_{1\|})^2 f_1^{(pc)} - T_0\frac{n_1}{n_0} \qquad \mathbf{\Pi} = \int d^3v \; mv_\| f_1^{(pc)} \; \mathbf{v}_D$$

$$T_{1\perp} = \frac{m}{2n_0}\int d^3v \; v_\perp^2 f_1^{(pc)} - T_0\frac{n_1}{n_0} \qquad \mathbf{\Pi}_t = \int d^3v \; mvR f_1^{(pc)} \; \mathbf{v}_D$$

Here, the drift velocity $\mathbf{v}_D$ is approximated by the generalized $\mathbf{E}\times\mathbf{B}$ velocity. The fluxes are furthermore split into their electrostatic (es) and electromagnetic (em) fractions and projected along the radial direction. The choice of the corresponding contravariant basis vector depends on the `norm_flux_projection`

[2]with respect to the full simulation volume

switch and can either be ∇x or the unit vector $\nabla x/|\nabla x|$. The advantage of the latter option (default in prerelease-1.6 only) is the independence of the radial coordinate while the former option is preferable in the context of transport equations where, e.g., the continuity equation yields

$$\begin{aligned}\dot{N} &= -\int_V \frac{\partial n}{\partial t}\mathrm{d}V = \int_S \mathbf{\Gamma}\cdot\nabla x/|\nabla x|\mathrm{d}S \\ &= \int_S \mathbf{\Gamma}\cdot\nabla x J\mathrm{d}y\mathrm{d}z\end{aligned}$$

and hence $\int \dot{N}\mathrm{d}x = -V\langle\Gamma^x\rangle$ since $\langle\ldots\rangle = 1/V\int\ldots J\mathrm{d}x\mathrm{d}y\mathrm{d}z$. Note that both alternatives can always be restored in the post-processing using the 3D data - the statistics might, however, be less accurate.

4.2 The `field` file

In contrast to the `nrg` file, the `field` files are binary files. All entries consist of the real valued step time followed by the three dimensional (k_x, k_y and z space), complex valued fields

$$\frac{\phi_1}{(T_{\mathrm{ref}}/e)\rho^*_{\mathrm{ref}}} \quad \left\{\frac{A_{1\|}}{B_{\mathrm{ref}}\rho_{\mathrm{ref}}\rho^*_{\mathrm{ref}}} \quad \left[\frac{B_{1\|}}{B_{\mathrm{ref}}\rho^*_{\mathrm{ref}}}\right]\right\}$$

Note that $A_{1\|}$ and $B_{1\|}$ are only appended if actually being computed, i.e. `beta` has to be larger than zero for both and parallel magnetic fluctuations need to be switched on via `bpar` for the latter.
The total number of fields `n_fields` can also be found in the info namelist of the GENE output `parameter` file.

4.3 The `mom` file

GENE produces a `mom` file for each species which can be distinguished by the appended species name defined in the corresponding parameter namelist.
The `mom` file structure is very similar to the `field` file structure. After the real valued time step information the following three dimensional, complex valued *velocity space moments*

$$\frac{n_1}{n_0\rho^*_{\mathrm{ref}}} \quad \frac{T_{1\|}}{T_0\rho^*_{\mathrm{ref}}} \quad \frac{T_{1\perp}}{T_0\rho^*_{\mathrm{ref}}} \quad \frac{q_{1\|}+1.5p_0u_{1\|}}{p_0c_{\mathrm{ref}}\rho^*_{\mathrm{ref}}} \quad \frac{q_{1\perp}+p_0u_{1\|}}{p_0c_{\mathrm{ref}}\rho^*_{\mathrm{ref}}} \quad \frac{u_{1\|}}{c_{\mathrm{ref}}\rho^*_{\mathrm{ref}}}$$

are written to the file. The combinations of the average velocity and the parallel and perpendicular components of the parallel heat current density are defined as

$$q_{1\|} + 1.5 p_0 u_{1\|} = \frac{m}{2} \int \mathrm{d}^3v \; v_\|^3 f^{(pc)}$$
$$q_{1\perp} + p_0 u_{1\|} = \frac{m}{2} \int \mathrm{d}^3v \; v_\| v_\perp^2 f^{(pc)}$$

where higher orders in perturbed quantities have been neglected.

4.4 The `vsp` file

In contrast to `mom` and `field` files where information about the three space dimensions are stored, the `vsp` file contains velocity space ($v_\|$, μ) data dependent on the parallel (z) coordinate. A data block consists of the real valued time step followed by the particle and heat fluxes Γ_{es}, Γ_{em}, Q_{es}, Q_{em} and the distribution function $\langle |f| \rangle_{k_x,k_y}^{1/2}$ averaged over the perpendicular directions.

4.5 The `neoclass` file

The entries of this file are sequentially written in ASCII for each time step divisible by `istep_neoclass` or at the end of a comp_type='NC' run. One entry looks like

```
100.0
2.4862E+01   4.4911E+00   2.1926E+01   1.2346E+02
3.5781E+01   1.7368E+02   2.3356E+01   1.2562E+01
```

The first line of each data block presents the corresponding simulation time, followed by as many lines and in the same order as species are defined in the `parameters` file. The columns refer to the flux-surface averaged, normalized neoclassical equilibrium quantities

$$\frac{\langle \Gamma^{\mathrm{neo}} \rangle}{\Gamma_{\mathrm{gb}}} \quad \frac{\langle Q^{\mathrm{neo}} \rangle}{Q_{\mathrm{gb}}} \quad \frac{\langle \Pi^{\mathrm{neo}} \rangle}{\Pi_{\mathrm{gb}}} \quad \frac{\langle j_b \rangle}{(n_{\mathrm{ref}} c_{\mathrm{ref}} B_{\mathrm{ref}} \rho_{\mathrm{ref}}^*)}$$

with units being detailed in Sec. A. The neoclassical fluxes (and the bootstrap current) are computed as velocity space moments of the *nonfluctuating* part $f_1^{(nc)}$ of the perturbed distribution function, which has the Fourier component $k_x = k_y = 0$. They are defined as

$\Gamma^{\rm neo} = \int d^3v\ f_1^{(nc)}\ \mathbf{v}_d^r$ $\qquad\qquad$ $\mathbf{\Pi}^{\rm neo} = \int d^3v\ mv_\parallel f_1^{(nc)}\ \mathbf{v}_d^r$

$\mathbf{Q}^{\rm neo} = \int d^3v\ \frac{1}{2}mv^2 f_1^{(nc)}\ \mathbf{v}_d^r$ $\qquad\qquad$ $j_b = \int d^3v\ B_0 v_\parallel f_1^{(nc)}$

Here, $\mathbf{v}_d$ denotes (the radial component of) the combined curvature and ∇B drifts. Fluctuating fields, as well as the Maxwellian background are not considered. The contravariant basis vector used for radial projection depends on the `norm_flux_projection` switch, as in the `nrg` file.

4.6 The `energy` file

This file contains the time evolution of free energy quantities written in ASCII for each time step divisible by `istep_energy`.

We define the free energy operator as the volume average
$\mathcal{E}[A] = \sum_j \mathrm{Re}\{\langle n_{0j}T_{0j} \int d\mu dv_\parallel h_j/F_{0j}\ A_j\rangle_{xyz}\}$ summed over species and given in normalized units.
Here, $h_{1j} = F_{1j} + (q_j/T_{0j}\bar{\phi}_1 + \mu\bar{B}_{1\parallel})F_{0j}$ is a modified distribution function. The argument A_j can be the distribution function F_{1j} or contributions to its time derivative $\partial_t F_{1j}$ in the gyrokinetic equation.

The energy file has has 14 columns, that enable to diagnose free energy balance in the simulation. The first column gives the simulation time t in units of $L_{\rm ref}/c_{\rm ref}$, the second column is the total free energy ($A_j = F_{1j}$) in units of $E_{\rm ref} \equiv n_{\rm ref}T_{\rm ref}(\rho^*_{\rm ref})^2$ and the following columns contain contributions to the free energy balance (A_j of type $\partial_t F_{1j}$) in units of $E_{\rm ref}/(L_{\rm ref}/c_{\rm ref})$. Details on the normalization are given in Sec. A. In summary, the energy columns are:

1	t	simulation time
2	E_{tot}	total free energy
3	$(dE/dt)_{tot}$	total change of free energy
4	$(dE/dt)_{drive}$	contribution of the drive term that includes ∇_T and ∇_n
5	$(dE/dt)_{source}$	heat and particle source terms (zero in local simulations)
6	$(dE/dt)_{coll}$	collisional dissipation
7	$(dE/dt)_{Dz}$	dissipation by z hyperdiffusion
8	$(dE/dt)_{Dv}$	dissipation by v hyperdiffusion
9	$(dE/dt)_{Dxy}$	dissipation by x and y hyperdiffusion
10	$(dE/dt)_{nl}$	contribution of the nonlinear term (zero to machine precision due to xy integration)
11	$(dE/dt)_{zv}$	contribution of the parallel(zv) Poisson bracket (zero to machine precision unless dissipative z boundary cond. or non-standard derivative methods are used in z and/or $v\|$)
12	$(dE/dt)_{rest}$	curvature and remaining terms (zero to machine precision in local simulations)
13	$(dE/dt)_{check}$	check for conservation: $((dE/dt)_{zv+nl+rest})/(dE/dt)_{drive}$
14	$(dE/dt)_{tot,2}$	$(dE/dt)_{tot}$ computed from two consecutive E_{tot} values (should be similar to second column, depending on `istep_energy` and the time integration scheme)

If the parameter `istep_energy3d` is set, energy terms are written into another (binary) energy3d file before they are volume-averaged in $\{x, y, z\}$ space, when the time index is divisible by `istep_energy3d`. Some of the above terms are combined to keep the file smaller. Each entry begins with the time (a real number) followed by six (real-valued) energy quantities in (k_x, k_y, z) space: (i) E_{tot}, (ii) $(dE/dt)_{noncons}$ (combining non-conservative terms 4-11, except 10), (iii) $(dE/dt)_{drive}$, (iv) $(dE/dt)_{coll}$, (v) $(dE/dt)_{diss}$ (combining hyperdiffusion terms 7-9) (vi)$(dE/dt)_{nl}$.They can be analyzed with the GENE diagnostics tool.

4.7 Triplet Diagnostic files

Free energy (see above) is conservatively transfered in wavenumber triplets satisfying the condition $k-k' = k''$. The triplet diagnostic calculates wavenumber-resolved free-energy transfer, optionally resolved in distribution function modes and into electrostatic/electromagnetic transfer. Energy transfer is given by:

$$T_{k,k'} = 2Re\Big\{ \int dz dv_{\|} d\mu \frac{J(z)\pi n_0 T_0 B_0}{F_0} \Big[g_k + \frac{qF_0}{T_0}\chi_k\Big]^* \Big[(k\times k')\chi(k')g(k-k')\Big]\Big\}$$

There are two default files, triplet_es.dat and triplet_ke_es.dat, tracking the terms proportional to $g_k\chi_{k'}g_{k''}$ and $\chi_k\chi_{k'}g_{k''}$ respectively. The triplet_es.dat file consists of the timestep and an array of transfers for each triplet, wavenumber within the triplet, and distribution function structure loaded in or the remainder of energy transfer for the last index at each wavenumber. The triplet_ke_es.dat file is similar but does not split transfer between distribution function structures. If there are fewer distribution function modes at a wavenumber than n_dfs_tplts, the file has zero for the later entries. For electromagnetic runs with split_em_tplt on there are also files triplet_em.dat and triplet_ke_em.dat, which record transfer by $\chi_{k'}$ due to the $A_{\|}$ contribution.

4.8 The checkpoint files

Depending on the `write_checkpoint` and `istep_schpt` settings a `checkpoint` file and/or a `s_checkpoint` (secure checkpoint) file will be written to the `chptdir` directory (for further explanation see section 3.2.3).
The files contain the full 5D distribution function and information about the used precision and resolutions and will be binary files written with MPI-IO unless chosen otherwise (see again section 3.2.3). Another file with very similar properties but containing multiple snapshots of the distribution function is the `g1` file which is controlled by `istep_g1`.

4.9 The `omega` file

The `omega` file will be written in linear initial value simulations if `istep_omega` is greater than zero and consists of a table with three columns

$$k_y\rho_{\rm ref} \quad \frac{\gamma}{c_{\rm ref}/L_{\rm ref}} \quad \frac{\omega}{c_{\rm ref}/L_{\rm ref}}$$

for as many k_y values as considered in the simulation (typically one). If a mode fails to converge up to the precision given by `omega_prec`, growth rate and real frequency will be set to zero.

4.10 The `eigenvalues` file

The `eigenvalues` file will be written at the end of linear eigenvalue simulations and consists of a header line specifying the considered k_y value in units of ρ_{ref} and as many lines as eigenvalues have been found - at least as many as specified by `n_ev` if the code has converged. The output format is

$$\frac{\gamma}{c_{\text{ref}}/L_{\text{ref}}} \qquad \frac{\omega}{c_{\text{ref}}/L_{\text{ref}}}$$

4.11 The geometry output `<magn_geometry>.dat` file

All information necessary to describe the field-aligned simulation domain is stored in a file named `<magn_geometry>.dat` where `<magn_geometry>` will be replaced by the magnetic equilibrium interface chosen in your simulation, see the input parameter `magn_geometry`.

The header namelist contains 1D information such as the number of `gridpoints` (corresponds to `nz0`), the employed magnetic equilibrium, the reference values set by the geometry (`Lref, Bref`) as well flux surface quantities like safety factor `q0`, magnetic shear `shat`, minor and major radius of the device, and the inverse aspect ratio of the flux surface (`trpeps`$= r/R_0$). Additionally, `beta` and `my_dpdx` (corresponds to `dpdx_pm`) are provided to monitor pressure effects and `s0`=`x0`^{2} is written for compatibility with particular interfaces. Finally, `Cy` and `Cxy` denote the normalized versions of the parameters C_y and C_{xy} which define the y and (more indirectly) x coordinates via

$$y = C_y\,(q\,\texttt{sign\_Ip\_CW}\,\theta_{\text{sf}} - \phi) \tag{1}$$

$$\mathbf{B} = C_{xy}\,\nabla x \times \nabla y. \tag{2}$$

Here, θ_{sf} denotes the straight field line and ϕ the toroidal angle. The `Cy`, `Bref` and `Lref` values are particularly relevant when comparing Gene results obtained with different magnetic interfaces for an identical equilibrium or when benchmarking with other codes. The normalized y wave number is, for instance, given by

$$k_y\rho_{\text{ref}} = n\,\frac{1}{\texttt{Lref}\,\texttt{Cy}}\,c_{\text{ref}}\,\frac{m_{\text{ref}}c}{e\,\texttt{Bref}} \tag{3}$$

where n is the toroidal mode number.

All higher dimensional geometry data is provided in the remainder of the geometry file. In fluxtube simulations, the data is aligned in a table where the 17 columns correspond to

$$g^{xx} \quad g^{xy} \quad g^{xz} \quad g^{yy} \quad g^{yz} \quad g^{zz} \quad B_0 \quad \partial_x B_0 \quad \partial_y B_0 \quad \partial_z B_0 \quad J \quad R \quad \phi \quad Z \quad \frac{\partial x}{\partial R} \quad \frac{\partial x}{\partial Z}$$

while the rows represent the values along the `nz0` parallel grid points. More information on the derivation of the metric coefficients, the normalization of the magnetic field $B_0(z)$ and the Jacobian $J(z)$ can, for instance, be found in Ref. [23]. The cylindrical coordinates R, ϕ, and Z may not be available for all magnetic geometry interfaces but are given in m and rad else. These and the $\frac{\partial x}{\partial R}$ and $\frac{\partial x}{\partial Z}$ derivatives may be required for visualization or synthetic diagnostics purposes.

4.12 The `autopar` file

The `autopar` file will be written upon start-up when GENE is checking the performance for all possible `MPI` setups and the efficiency of different implementations with respect to cache sizes. A typical entry is

```
parallelization:    s   v   w   x   y   z
parallelization:    1   1   8   1   1   1  nblocks:    64
   1 1 1 1 1 1 1 1 1 :    4.8712 MB, t =  5.629E-02 sec
   2 1 1 1 1 1 1 1 1 :    7.8877 MB, t =  5.634E-02 sec
   (...)
   1 2 3 1 2 2 2 1 2 :    4.8184 MB, t =  5.332E-02 sec
   best perf_vec      min. WCT     mem_min  mem_max deficit(byte) data segment
   1 2 3 1 2 2 2 1 1  4.4962E-02     3.57     7.89      238032    23.043
parallelization:    1   1   4   1   1   2  nblocks:    64
   (...)
```

where the first two lines contain information about the parallelization and the block size (for strip mining) being considered in the following section. Afterwards, the wall clock time for each individual `perf_vec` setting is printed together with a memory estimate for those routines. For each parallelization, the best choice is finally redisplayed and supplemented by the minimum and maximum memory estimates as well as the actually used work space (this, however, might not be available/accurate on every machine).

4.13 The `codemods` file

When compiling GENE, if the code was obtained through `git`, the modifications of files in the `src/` directory relative to the repository versions are stored in the executable. Upon running GENE, a `codemods` file is created in the diagdir, which contains said modifications. As such, it expands on the information of GIT_BRANCH/GIT_MASTER and allows for better reproducibility of simulations. In this regard, a useful approach is to copy the codemods file to a different GENE base directory and run

```
$ patch -p0 -i [codemods]
```

Thus, the modifications used to produce the codemods file are applied to another GENE source. It needs to be stressed that codemods functionality relies on git diff—therefore, downloaded GENE tarballs cannot make use of it, and codemods does not include any information contained in source code files not part of the (local) GENE git repository.

5 The post-processing tools

For visualizing one's Gene data, be it the fields, moments, velocity space, or the nrg file data, one can employ bundled post-processing routines. The traditional and currently most comprehensive set is written in IDL but python routines are shipped as well and will be extended in the future.

5.1 The IDL-based Diagnostic

The publicly available version is able to run under the free IDL Virtual Machine (`www.ittvis.com/idlvm`) without an IDL license, and comes with a graphical user interface and an assortment of standard visualization diagnostics.

5.1.1 The graphical user interface

On a machine from where the data which is to be analyzed can be accessed, enter the `diagnostics` directory and start IDL VM with
`idl -vm=vm_diag.sav` to open the graphical user interface (GUI). The latter contains a display area on the top left, a run specification area next to it, the diagnostics table on the bottom, and the command buttons on the far right.

In the data path field, enter the (relative or full) path to the directory in which your run data (`nrg_[run#]` etc.) is saved; the buttons next to the data path field can be set to default paths for your convenience (for this, you will be asked to provide your choice when clicking the buttons for the first time; alternatively you can modify the paths in the `internal/guiform` file). The next field specifies the directory in which the diagnostic output (postscript/data/HDF5 files) is stored. If no path is specified, the default `output` directory is chosen.

Next, enter the run number in the runs field; if more than one number is to be specified (i.e., the first run has follow-ups), either separate the numbers by a comma (e.g. "50,51,58"), or supply ranges (e.g. "50-54"), or use a combination of both. If follow-ups are marked with concluding letters in alphabetical order (e.g. "50,50a,50b"), they will automatically be inserted by using a plus sign (e.g."50+") after the base run number. In order to analyze a run which has not been assigned a run number, simply enter "act"; this works in combination with the above methods, e.g. "50,51+,57-59,60b-d,act". After entering the run numbers, it is recommended that you press the return key.

Please note that the following file name convention is expected: "act" expects unchanged file names, e.g. `parameters.dat`, `field.dat`. Runs are labeled by renaming to `[base]\_[number]` (e.g. "field_50") or `[number]\_[base]` (e.g. "50_field"). In general, one can use letters instead of numbers also, if the regular expression "act" is avoided.

To specify the time frame which is to be analyzed, use the start and end fields. For convenience, the first and last buttons look up the first and last time step, respectively, that is available from the runs entered in the runs field.

The particle button group is redrawn if one changes the runs field, for the number and names of the species may change. Select the species which you wish to analyze here. Next to it is the normalization drop down menu where different normalizations can be chosen. L_perp and R are always available (others depend on the species), and stand for the standard GENE length scale and the major plasma radius, respectively. Lastly, there is the output format button group, where you can choose to create a data file, a postscript, or both.

The other buttons next to the run specification area have the following functionalities:

- show series info: writes some content of the parameters file to the display area, e.g. the resolution, gradients, and whether the run is non-linear
- show nrg data: writes the most relevant data from the nrg file to the display area
- start nrg diag: creates a postscript file with an nrg data plot
- show geometry: plots geometry coefficients used in your GENE calculation

Below, the diagnostics table lists all bundled diagnostic programs for a specific data loop selected by the tab - e.g. to calculate heat and particle flux spectra or to draw contour plots of certain variables for mom/field data. By clicking on the question mark next to the name, one gets a popup window with a description of the diagnostic and its parameters. The column to the right of the name shows which diagnostics are activated and will be run once the save and start button is clicked – toggle the on/off status by clicking in this column. All the other columns contain the diagnostics parameters. If a diagnostic asks which variables from the moments and field files to analyze, that information is to be entered here. Some parameters are on/off switches which can be toggled by clicking the corresponding field. Please refer to the diagnostic programmer's guide placed in `diagnostics/doc/` if you plan to write and include own diagnostic programs.

Again, a range of buttons is provided next to the diagnostics table:

- save and start: saves the state of the GUI and starts reading the run data from the moments and field files and feeds it to the selected diagnostics; no user action is permitted until the "finished diagnosing" dialog window has popped up
- load form: loads the most recently saved GUI form, e.g. to undo changes
- save form: saves the current gui form, e.g. to save the GUI state before exiting
- clear form: clears all fields in both the run specification area and the diagnostics table
- show variable list: writes all available moments and fields with their corresponding variable index to the display area
- show custom diags: this button is not available if you are using the IDL Virtual Machine version of the GENE diagnostic. Clicking it will open an additional window in which one can activate custom diagnostics. Such diagnostics are to be put into the custom folder and will be recompiled every time the user clicks the save and start button. An example is provided with the distribution, and it is strictly recommended to follow the nomenclature.
- recent ps: opens the most recently created postscript files in ghost view by spawning a `gv` command in the shell process where the diagnostic has been started
- ps files: opens a file open dialog where the user can choose to open any ps files from the output directory
- exits the diagnostic without saving the GUI state

Besides displaying data and information, the display can also be used to abort the diagnosing by clicking and holding the left mouse button until the next time step is read.

5.1.2 The configuration file

To customize certain functions, a file named `diag.cfg` can be edited which is found in the main diagnostics directory. It contains entries of the form `key = value`. The following keys are available:

- `ps_viewer [gv]`: one's choice of a viewer for the postscript output (it is used when one clicks on `recent ps` or `ps files`)
- `vm_lowcase [no]`: may be set to `yes` to use lowercase filenames for Virtual Machine output files
- `short_gui [no]`: may be set to `yes` to slightly reduce the GUI window height—useful in conjunction with certain screen resolutions
- `speedup_gui [no]`: on certain machines (e.g., HPCFF), when changing the run number, it takes a long time to refresh the species namelist; set to `yes` to significantly reduce that delay
- `info_str [2]`: setting for the info string on the bottom of the output pages (for most diagnostics); 0: no string; 1: time info only; 2: time info and diag variable settings info
- `bg_color [gray]`: widget background color, requires IDL restart (it changes a file in the home directory to modify the Motif settings); see `internal/global_vars.pro` for other color examples
- `fg_color [black]`: widget foreground color, see above

If you encounter errors while using the GENE Diagnostics Tool, if you have modified any of the above entries, please indicate such modifications when submitting error reports.

5.2 The python-based postprocessing routines

Acknowledging the almost universal availability and increasing potential of python, first attempts at establishing corresponding post-processing routines have been launched already some time ago. While not fully completed yet, the current (**python3**-based) solutions may already be useful for several applications and can be found in the `python-diag` subdirectory after calling `git submodule init` to set up the corresponding git submodule (one-time action only) and `git submodule update` to retrieve the latest linked version. Please note that for running some of the routines from different directories (e.g., `diagdir`), you need to add this directory to your `PYTHONPATH` environment (e.g., `setenv PYTHONPATH "$HOME/$GENEDIR/python-diag"` for tcsh or `export PYTHONPATH="$HOME/$GENEDIR/python-diag"` for bash shells).

5.2.1 The pydiag tool

The `python-diag/pydiag` folder hosts a first implementation of a command-line tool `pydiagc.py` which allows triggering several diagnostics (mode structures, spectra, etc). If this subdirectory is part of your `$PATH` environment, you may simply call `pydiagc.py` in those directories where the simulation data resides (`diagdir`). A typical call of this script should specify the diagnostic (`--diagname`) to be run (see `pydiagc.py --help` for a full list), the time window (`start:end`), and the suffix of the files containing the simulation data (e.g., `.dat`).

Examples:

- Visualize the time traces in the file `nrg_1` from 350 time units to the last (`l`) time step and print averages etc. to the terminal:
  `pydiagc.py --nrg 350:l _1`
- Create interactive plots of the eigenmode structure at the end of the simulation with suffix _0001:
  `pydiagc.py --ballooning l:l _0001`
- Write a GKDB json file (here: data.dat.json) containing information for all computed eigenvalues (`f:l`) for a given linear simulation (in `*.dat`):
  `pydiagc.py --gkdb f:l .dat`

Further details may be retrieved by calling `pydiagc.py --help`.

5.2.2 The python GUI

Currently, a revised version of the above `pydiag` tool is developed which also includes a graphical user interface. Those interested may already have a look at `python-diag/GENE_gui_python`. All others should kindly note that the `pydiag` may be replaced in the intermediate future.

6 Tools

Located in the `/tools/` directory (and subdirectories) you can find some scripts based on perl, bash, or python which might help you to save time, e.g.

- `runassign` to rename the suffix of just generated GENE output files (`*.dat`) to (`*_<string>`) a string provided as second argument; though the latter can be arbitrarily chosen, our post-processing routines are particularly able to support number number+letter (`'1+'= 1,1a,1b,1c,...`) or number ranges (e.g., `1-7`);

- `runrename` to rename the suffix from a string given by the first argument to a string passed to the script as second argument

- `rundelete` to delete a run number range

- `runshift` to shift a run number range

- `diffnrg` to compare nrg files which for instance have been generated on different machines with the same parameters file. In contrast to the usual unix diff command `diffnrg` will ignore differences smaller than 10^{-15}.

- `gplot` for a simple and quick gnuplot visualization of nrg data. Type `gplot --help` for more information.

In the `/tools/python/` subdirectory, there are some Python-based scripts, e.g.

- `efit-tools.py` to generate plots and print selective data from an arbitrary EFIT/g-EQDSK file.

- `extract_miller_from_eqdsk.py)` to analyze an EFIT file and generate the parameters for a Miller geometry description. The script will produce a block of parameters that can directly be copied into a GENE input parameter file.

- `make_aug_eqdsk.py` to create an EFIT magnetic geometry file for an arbitrary ASDEX Upgrade discharge, if run from a machine with access to the ASDEX Upgrade shotfile database.

- `make_aug_iterdb.py` to create an IterDB-style file from an ASDEX Upgrade shotfile (by default, from output data of a TRANSP run). As with AUG EQDSSK script, you need to be on a machine with access to the ASDEX Upgrade shotfile database.
- `show_eqdsk.py` to create additional EQDSK related plots.
- `summarize_probs.py` to obtain a quick overview of all the GENE problem directories contained in a given GENE tree, and the simulations that have been performed there.

7 Reporting bugs and submitting changes

Please report bugs or source code modification requests to support@genecode.org. The latter (and possible bug fixes) may be communicated using patch files which can be created by calling

```
git diff > patch_release_1_7_0b09dc6.diff
```

in your Gene source directory. Please make sure that your file name contains the name of the branch (unstable or release X.Y) and the git hash tag (see `git rev-parse --short HEAD`).

8 Frequently Asked Questions

What should I set the diffusivities to?

If a simulation makes use of an adiabatic response, is electrostatic, or has no (or very small) shear, `hyp_z` should be set to the approximate linear growth rate; otherwise, higher values are required and it is recommended to scale up `hyp_z` with increasing resolution. This is achieved by setting negative values, where `hyp_z`$= -0.5$ is a good starting point (larger absolute value increases the dissipation). Especially, `hyp_z`$= -1$ is equivalent to `hyp_z`$= 4/(3\Delta z)$, which mimics 3rd order upwind dissipation, however neglecting the advection prefactor. In simulations with collisions, these provide a physical velocity space diffusion; if the collisionality is large enough, `hyp_v` may thus be reduced or set to zero. Otherwise, a small amount (`hyp_v`~ 0.2) may be necessary. Perpendicular hyperdiffusion is not needed in fluxtube simulations unless the maximum wave numbers do not resolve the full driving range and a spectral pile-up may occur. In that case, the `GyroLES` method is recommended.

How can I reset my forgotten password for the GENE repository?

Please send an email with a corresponding request and your user id to support@genecode.org.

Repeating a nonlinear simulation with identical initial condition

I did this exercise and got different time traces. Is this a bug?
Both, GENE and FFTW, run some internal optimization before actually starting the simulation. Hence, the cache or MPI distribution might not be completely identical but might yield differences on the last digits. However, in a turbulence simulation even tiny deviations in the initial state can have a visible influence after some time. Hence, as long as the time averaged observables are similar there is no need for any concern. For a rigorous test, you need to fix the parallelization, the `perf_vec`, the `nblocks` value and switch off any optimization in the FFT library, e.g., by using MKL instead of FFTW.

A Normalizations

GENE's implementation of the gyrokinetic set of equations and hence all native code output is dimensionless and thus normalized to appropriate quantities. In this context, the following abbreviations/quantities might appear:

A.1 species dependent

n_0	equilibrium density of the *corresponding species*
T_0	equilibrium temperature of the *corresponding species*
p_0	equilibrium 'pressure' of the *corresponding species*(!), i.e. $p_0 = n_0 T_0$
m	mass of the *corresponding species*
v_T	thermal velocity $v_T = \sqrt{\mathbf{2}T_0/m}$ of the *corresponding species*

A.2 reference quantities

$n_{\rm ref}$	density of the *reference species* (i.e. species with `dens = 1.0`)
$T_{\rm ref}$	reference temperature (temperature of species with `temp = 1.0`)
$m_{\rm ref}$	reference mass (mass of species with `mass = 1.0`)
$L_{\rm ref}$	reference length (depends on the chosen geometry)
$B_{\rm ref}$	reference magnetic field strength (typically on axis; miller: center of flux surface)
$q_{\rm ref}$	reference charge $q_{\rm ref} = e$
$\Omega_{\rm tor,ref}$	reference toroidal angular velocity

These reference values are **not** *required for* GENE *simulations themselves* but can be set in the `&units` namelist (see Sec. 3.2.9) for post-processing purposes or automatic computation of `beta`, `coll`, `rhostar` etc.

A.2.1 derived reference quantities

$p_{\rm ref}$	reference pressure $p_{\rm ref} = n_{\rm ref} T_{\rm ref}$
$c_{\rm ref}$	reference velocity $c_{\rm ref} = \sqrt{T_{\rm ref}/m_{\rm ref}}$ (without $\sqrt{2}$!)
$\Omega_{\rm ref}$	reference gyrofrequency $\Omega_{\rm ref} = q_{\rm ref} B_{\rm ref}/(m_{\rm ref} c)$
$\rho_{\rm ref}$	reference gyroradius $\rho_{\rm ref} = c_{\rm ref}/\Omega_{\rm ref}$
$\rho^*_{\rm ref}$	reference gyroradius-to-machine-size ratio $\rho^*_{\rm ref} = \rho_{\rm ref}/L_{\rm ref}$
$\Gamma_{\rm gb}$	particle flux GyroBohm units $\Gamma_{\rm gb} = c_{\rm ref} n_{\rm ref} (\rho^*_{\rm ref})^2$
$Q_{\rm gb}$	heat flux GyroBohm units $Q_{\rm gb} = c_{\rm ref} p_{\rm ref} (\rho^*_{\rm ref})^2$
$\Pi_{\rm gb}$	momentum flux GyroBohm units $\Pi_{\rm gb} = c^2_{\rm ref} m_{\rm ref} n_{\rm ref} (\rho^*_{\rm ref})^2$

References

[1] E. A. Frieman and L. Chen, Phys. Fluids **25**, 502 (1982)

[2] T. S. Hahm, Phys. Fluids **31**, 2670 (1988)

[3] T. S. Hahm, W. W. Lee, and A. Brizard, Phys. Fluids **31**, 1940 (1988)

[4] A. Brizard, Phys. Fluids B **1**, 1381 (1989)

[5] A. Brizard and T. S. Hahm, Rev. Mod. Phys. **79**, 421 (2007)

[6] F. Jenko *et al.*, Phys. Plasmas **7**, 1904 (2000)

[7] T. Görler *et al.*, J. Comput. Phys. **230**, 7053 (2011)

[8] F. Jenko and The GENE development team, The GENE code. URL `http://genecode.org`

[9] M. A. Beer, Ph.D. Thesis, Princeton (1995)

[10] P. Xanthopoulos and F. Jenko, Phys. Plasmas **13**, 092301 (2006)

[11] A. Arakawa, J. Comput. Phys. **1**, 119 (1966)

[12] P. Xanthopoulos *et al.*, Phys. Plasmas **16**, 082303 (2009)

[13] A. M. Dimits *et al.*, Phys. Plasmas **7**, 969 (2000)

[14] S. Balay, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith and Hong Zhang, PETSc Users Manual, ANL-95/11 - Revision 2.1.5 (2004)

[15] S. Balay, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith and Hong Zhang, PETSc Web page, URL `http://www.mcs.anl.gov/petsc` (2001)

[16] V. Hernandez, J. E. Roman, and V. Vidal, ACM Transactions on Mathematical Software, **31**, 351 (2005)

[17] J. D. Huba, NRL report, URL `http://wwwppd.nrl.navy.mil/nrlformulary` (2011)

[18] F. L. Hinton and R. D. Hazeltine,Rev. Mod. Phys. **48**, 239 (1976)

[19] G.W. Hammett, W. Dorland, N. Loureiro, T. Tatsuno, APS-DPP Meeting, Philadelphia, Oct. 30 - Nov. 3, 2006.

[20] B.F. McMillan, J. Ball, and S. Brunner, PPCF **61**, 055006 (2019); `https://dx.doi.org/10.1088/1361-6587/ab06a4`

[21] M.J. Pueschel, T. Dannert, and F. Jenko, Comput. Phys. Commun. **181**, 1428 (2010)

[22] X. Lapillonne, *Local and Global Eulerian Gyrokinetic Simulations of Microturbulence in Realistic Geometry with Applications to the TCV Tokamak*, PhD thesis, EPFL Lausanne, 2010 (available at gene.rzg.mpg.de)

[23] D. Told, *Gyrokinetic Microturbulence in Transport Barriers*, Universität Ulm, 2012 (available at genecode.org)

[24] P. Morel *et al.*, Phys. Plasmas **18**, 072301 (2011); `http://dx.doi.org/10.1063/1.3601053`

[25] A. Bañón Navarro, *Gyrokinetic Large Eddy Simulations*, Université Libre de Bruxelles, 2012 (available at genecode.org)

[26] A. Bañón Navarro *et al.*, Phys. Plasmas **21**, 032304 (2014)

[27] J. Candy, Plasma Phys. Control. Fusion **51**, 105009 (2009)

[28] D. Jarema, H.J. Bungartz, T. Görler, F. Jenko, T. Neckel, D. Told, *Block-Structured Grids in Full Velocity Space for Eulerian Gyrokinetic Simulations*, Comput. Phys. Commun. **215**, 49 (2017); `http://dx.doi.org/10.1016/j.cpc.2017.02.005`