Hyb 3D. Technical report and developer notes

M.Mancini,R.Modolo, S. Hess

February 3, 2012

Abstract

This document contains the description of the **hyb_3D** code: its philosophy, its routines and the manual to implements new environments.

1 The philosophy

2 The structure

The structure of the code is such that a minimum number of files should be edited/created by the user. Namely, general parameters for the simulation are defined in defs_parameters.F90, a env_*name*.F90 file must be created in not already existing, and pointers to routines in this file may be set in environment.F90.

2.1 Module naming convention

Because of FORTRAN limitations, modules and routines cannot share the same name. This issue was dodged by adding "m_" in front of the module name. However, a better usage could be done of this mandatory prefix. Hence prefixes are now:
• "defs_", if the module mainly contains structures or variable definitions.
• "diag_", if the module is related to diagnostics.
• "env_", if the module contains the environment of a given planet.
• "atm_", for the modules containing the subroutines shared by the planet environement files (photoproduction, ionosphere,...). **atm_ modules should only be called by env_ modules**.
• "field_", if the module is related to fields.
• "particle_", if the module is related to particles.
• "m_", for the micellaneous modules which are directly related to the main program.
All other modules contains mostly general routines which cannot be classified into one of the groups mentioned above. Note that in addition to hyb_3d.F90, which contains the main program, initialisation.F90 and time_schedule.F90, which do the initialization and the iterations have no prefix, as they can be understood as part of the main program. Figure 1 shows a summary of all the

modules and the kind (Bloc) they belong to. Note that this division in blocs is not only related to the physical objects they deal with, but that modules in the same bloc are more closely related (through calls) than modules in different blocs.

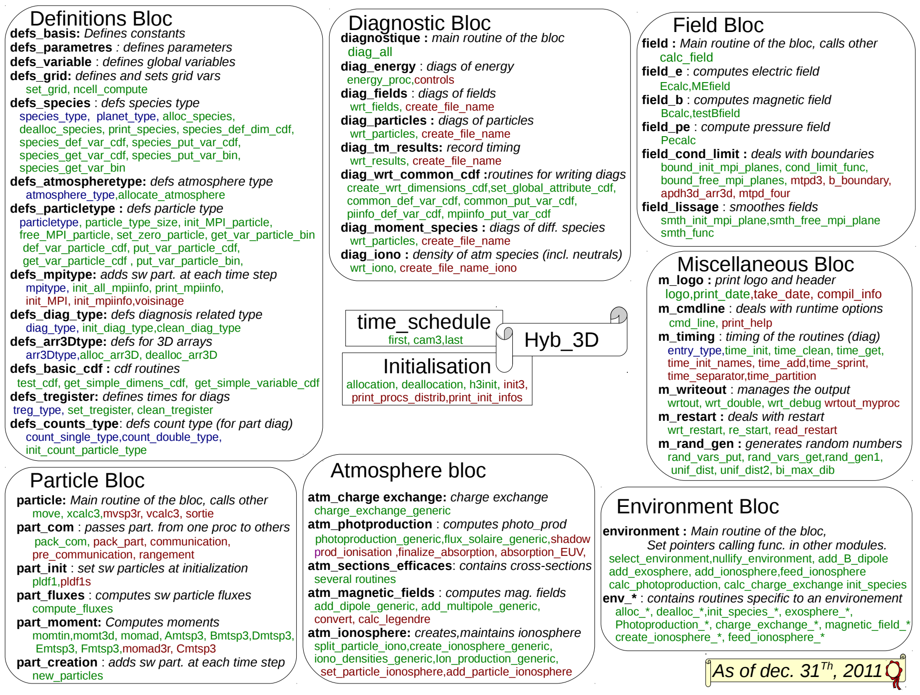
Definitions Bloc
defs_basis: Defines constants
defs_parametres : defines parameters
defs_variable : defines global variables
defs_grid: defines and sets grid vars
set_grid, ncell_compute
defs_species : defs species type
species_type, planet_type, alloc_species, dealloc_species, print_species, species_def_dim_cdf, species_def_var_cdf, species_put_var_cdf, species_get_var_cdf, species_put_var_bin, species_get_var_bin
defs_atmospheretype: defs atmosphere type
atmosphere_type,allocate_atmosphere
defs_particletype : defs particle type
particletype, particle_type_size, init_MPI_particle, free_MPI_particle, set_zero_particle, get_var_particle_bin def_var_particle_cdf, put_var_particle_cdf, get_var_particle_cdf , put_var_particle_bin,
defs_mpitype: adds sw part. at each time step
mpitype, init_all_mpiinfo, print_mpiinfo, init_MPI, init_mpiinfo,voisinage
defs_diag_type: defs diagnosis related type
diag_type, init_diag_type,clean_diag_type
defs_arr3Dtype: defs for 3D arrays
arr3Dtype,alloc_arr3D, dealloc_arr3D
defs_basic_cdf : cdf routines
test_cdf, get_simple_dimens_cdf, get_simple_variable_cdf
defs_tregister: defines times for diags
treg_type, set_tregister, clean_tregister
defs_counts_type: defs count type (for part diag)
count_single_type,count_double_type, init_count_particle_type
Diagnostic Bloc
diagnostique : main routine of the bloc
diag_all
diag_energy : diags of energy
energy_proc,controls
diag_fields : diags of fields
wrt_fields, create_file_name
diag_particles : diags of particles
wrt_particles, create_file_name
diag_tm_results: record timing
wrt_results, create_file_name
diag_wrt_common_cdf :routines for writing diags
create_wrt_dimensions_cdf,set_global_attribute_cdf, common_def_var_cdf, common_put_var_cdf, piinfo_def_var_cdf, mpiinfo_put_var_cdf
diag_moment_species : diags of diff. species
wrt_particles, create_file_name
diag_iono : density of atm species (incl. neutrals)
wrt_iono, create_file_name_iono
Field Bloc
field : Main routine of the bloc, calls other
calc_field
field_e : computes electric field
Ecalc,MEfield
field_b : computes magnetic field
Bcalc,testBfield
field_pe : compute pressure field
Pecalc
field_cond_limit : deals with boundaries
bound_init_mpi_planes, cond_limit_func, bound_free_mpi_planes, mtpd3, b_boundary, apdh3d_arr3d, mtpd_four
field_lissage : smoothes fields
smth_init_mpi_plane,smth_free_mpi_plane smth_func
time_schedule
first, cam3,last
Hyb_3D
Initialisation
allocation, deallocation, h3init, init3, print_procs_distrib,print_init_infos
Miscellaneous Bloc
m_logo : print logo and header
logo,print_date,take_date, compil_info
m_cmdline : deals with runtime options
cmd_line, print_help
m_timing : timing of the routines (diag)
entry_type,time_init, time_clean, time_get, time_init_names, time_add,time_sprint, time_separator,time_partition
m_writeout : manages the output
wrtout, wrt_double, wrt_debug wrtout_myproc
m_restart : deals with restart
wrt_restart, re_start, read_restart
m_rand_gen : generates random numbers
rand_vars_put, rand_vars_get,rand_gen1, unif_dist, unif_dist2, bi_max_dib
Particle Bloc
particle: Main routine of the bloc, calls other
move, xcalc3,mvsp3r, vcalc3, sortie
part_com : passes part. from one proc to others
pack_com, pack_part, communication, pre_communication, rangement
part_init : set sw particles at initialization
pldf1,pldf1s
part_fluxes : computes sw particle fluxes
compute_fluxes
part_moment: Computes moments
momtin,momt3d, momad, Amtsp3, Bmtsp3,Dmtsp3, Emtsp3, Fmtsp3,momad3r, Cmtsp3
part_creation : adds sw part. at each time step
new_particles
Atmosphere bloc
atm_charge exchange: charge exchange
charge_exchange_generic
atm_photproduction : computes photo_prod
photoproduction_generic,flux_solaire_generic,shadow prod_ionisation ,finalize_absorption, absorption_EUV,
atm_sections_efficaces: contains cross-sections
several routines
atm_magnetic_fields : computes mag. fields
add_dipole_generic, add_multipole_generic, convert, calc_legendre
atm_ionosphere: creates,maintains ionosphere
split_particle_iono,create_ionosphere_generic, iono_densities_generic,Ion_production_generic, set_particle_ionosphere,add_particle_ionosphere
Environment Bloc
environment : Main routine of the bloc, Set pointers calling func. in other modules.
select_environment,nullify_environment, add_B_dipole add_exosphere, add_ionosphere,feed_ionosphere calc_photoproduction, calc_charge_exchange init_species
env_* : contains routines specific to an environement
alloc_*, dealloc_*,init_species_*, exosphere_*, Photoproduction_*, charge_exchange_*, magnetic_field_* create_ionosphere_*, feed_ionosphere_*
As of dec. 31Th, 2011

Figure 1: Organization of the modules, routines in the modules are shown in green for public ones and red for private ones. Data types are also shown in the module which defines them, in blue.

3 Defs_parametre.F90

This file define most of the parameters of the simulation which are not specifically related to the environment. In particular, all the parameters which can be modified through the command line are defined here.

3.1 Space and time (relate them here)

• ncx0,ncy0,ncz0. Size of the grid, in cell number. Note that the code may automatically resize the grid so to be commensurable with the number of processes.
• gstep(3). Size of a cell, in units of the inertail length of the main ionic specie.
• dt. Time step

• nhm. Number of iterations

3.2 IMF and Planet

The IMF direction is defined with the spherical variables psi (Z versus XY) and phi (X versus Y). The module of the IMF is defined in the environment file.

The environment file which will be used depends on planetname.

3.3 Fields and Ionosphere additional settings

Some particular settings can be set to describe how the ionosphere interacts we the surrounding plasma. These settings are defined in defs_parametre.F90:
• absorp_surface. If set to 1, the particles reaching the planet surface are absorbed, otherwise they are lost.
• t_iono_release. Can be used to freeze the ionosphere up to a given time, for a smoother initialization.
• ipe. Defines the polytropic law : 0 = adiabatic; 1 = isothermal; 2 = adiabatic in solar wind and hydrostatic in the ionosphere ($\gamma \propto 1/\log(density)$); 3= isothermal in solar wind and hydrostatic in the ionosphere.
• dn_lim_inf_conduc. Above this density (in simulation units, i.e. must be multiplied by the solar wind density to have physical units), the conductivity of the ionosphere is assumed to be infinity: the $v_i \times B$ term is set to 0. Use this to cancel a abnormal diffusion of negative E_y field in the ionosphere which leads to the ejection of a large part of the ionosphere.
• idisp, iresis. Set to 1 to use dissipation or resistive terms in the electric field computation.
• resis. Set the value of the resistivity.
• dmin. Minimum density for the computation of the electric field.

3.4 Optimization

As the simulation box is a cube, with a planet or satellite in its center, some regions on the corners of the simulation box may not play any role in the simulation result (they just see an unperturbed flow of plasma). Moreover, if the planet/satellite as an ionosphere/exosphere, there will be more macroparticles close to it than in the corners of the simulation box. If the simulation box is splitted between the different process in sub-boxes of same sizes, this will lead to (1) processes processing more particles than others, (2) processes simulating nothing interesting. To limit this, it is possible to define a region of interest (ROI) in defs_parametres.F90. This ROI must be defined manually, through 6 variables:
• y_min, y_max: define the position of the ROI in y, in fraction of the total box size in y (y_min=0.33 and y_max=0.67 correspond to a ROI occupying the central third of the box).

• z_min, z_max: define the position of the ROI in z, in fraction of the total box size in z.
• excess : define by how much the size of the box in the ROI must be shrinked (0 means no changes).
• excess_part : allow to inject more particle in the ROI (0 means no changes).

4 Environment.F90

This module serves as an interface between the global part of the code and the code specific to the environment simulated. This is done through the definition of a few pointers to the function specific to the environment. This pointers are associated with functions which: (1) make computations specific to an environment, (2) fill the to structures which defines the property of the plasma flow (species structure) and of the exosphere/ionosphere/magnetic field/anything associated with the planet (atmosphere).

To define an environment, a new env_*name*.F90 file must be created and the following pointers should be associated with routines in this file to enable the related physical processes:

4.1 planet_species =>

This pointer must be associated with a routine which initializes the species structure defined in defs_species.F90.

4.2 dealloc_planet =>

This pointer must be associated with a routine which deallocates arrays definied in the env_*name*.F90 file

4.3 exosphere =>

This pointer must be associated with a routine which computes the density of the neutral species in the exosphere.

4.4 photoproduction =>

This pointer must be associated with a routine which computes the photoproduction rates of some species. Note that if the cross-sections of the photoproduction reactions have previously been stored in the atmosphere variable, this pointer can directly be associated with photoproduction_generic which is a generic function.

4.5 charge_exchange =>

This pointer must be associated with a routine which implementes the charge exchange processes. Note that if the cross-sections of the charge exchange reactions are constant and have previously been stored in the atmosphere variable, this pointer can directly be associated with charge_exchange_generic which is a generic function.

4.6 ionosphere =>

This pointer must be associated with a routine which creates an ionosphere at initialization.

4.7 b_dipole =>

This pointer must be associated with a routine which compute the internal magnetic field of the planet.

4.8 feed_production =>

This pointer must be associated with a routine which computes the rates of ionization of the plasma through the simulation domain and create the corresponding macro-particles.

Important Does not change the number and the order of dummy arguments of the subroutines. The procedure pointers are sensible to the dummy argument list so if change the dummy argument order into `m_exo_future.o` you have to do that for all environment-dependent exosphere calculations.

5 The Species structure

This structure contains most of the information about the incident plasma flow and the obstacle. It should be filled in the routines associated to the planet_species pointer in environment.F90. before the call to planet_species, the number of incident species ns must be defined. This structure contains:

5.1 Reference quantities

These quantities are used to normalize the simulation variables.

- species%ref%density.Physical density of reference, which normalizes density (m^{-3})
- species%ref%c_omegapi. Ions inertial length, which normalizes lengths (km)
- species%ref%mag. Magnetic Field, which normalizes fields (Tesla)
- species%ref%alfvenspeed. Alfvén speed, which normalizes velocity ($km.s^{-1}$)
- species%ref%inv_gyro. Inverse of the gyrofrequency, which normalizes time (s).

• species%ref%maxabs. maximum absorption length (km).

5.2 Obstacle parameters

• species%planetname. Name of the environment.
All the following distances or lengths are in simulation units:
• species%P%centr. Position of the planet center.
• species%P%radius. Radius of the planet.
• species%P%r_exo. Radius of the exobase.
• species%P%r_lim. Radius of the bottom limit of the ionosphere.
• species%P%r_iono. Radius of the top limit of the ionosphere (when loaded explicitely).
• species%P%speed. Planet velocity.

5.3 Incident plasma parameters

• species%S(:)%name. Names of the incident species.
• species%S(:)%ng. Number of particles per cell for each species.
• species%S(:)%vxs; ...%vys; ...%vzs. Directed velocities along x, y and z.
• species%S(:)%percent. Ratio of each species in the plasma flow. Total must be **1**.
• species%S(:)%rcharge. Species charges (ratio to the dominant specie).
• species%S(:)%rmass. Species masses (ratio to the dominant specie).
• species%S(:)%rtemp. Species temperatures (ratio to the dominant specie).
• species%S(:)%betas. β of each species.
• species%betae. β of the electrons.
• species%S(:)%rspeed. Ratio $\frac{v_\parallel}{v_\perp}$ for each species.
• species%S(:)%rvth. Species thermal velocities (ratio to the dominant specie).
• species%S(:)%rmds. Species macroparticle weights (ratio to the dominant specie).
• species%S(:)%qms. Species charge over mass ratio (ratio to the dominant specie).
• species%S(:)%vth1. Species parallel velocities in simulation units.
• species%S(:)%vth2. Species perpendicular velocities in simulation units.
• species%S(:)%sm. Mass of the macroparticles of each species.
• species%S(:)%sq. Charge of the macroparticles of each species.

5.4 Ionospheric plasma parameters

• species%tempe_ratio. Temperature of the ionospheric electrons (ratio to the incident plasma electrons).
• species%viscosity. Collision frequency of the ions in the ionosphere (in $\mathrm{cm}^3.\mathrm{s}^{-1}$).

6 The Atmosphere structure

The exophere is primarily defined through the use of the `density_exo` and `prod_pp` arrays defined in `environment.F90`. From here, there are two ways of loading exospheric particles in the simulation: The first one is "by hand", through customized routines. You can use the `exophere`, `photoproduction`, `ionosphere` and `charge_exchange` pointers in `environment.F90` to call your routines. The second possibility is to use generic functions (those with the "atm_" prefix) and an *atmosphere* structure. The following addresses the second option.

6.1 Global environment variables

The *atmosphere* structure is defined through the call, in environment.F90, of a subroutine alloc_*planetname* which must exist in your env_*planetname*.F90 file. You must then define:
• the number of exospheric species (neutrals and ions): n_species.
• the number of charge exchange reactions : n_exc.
• the number of electron impact ionization reactions : n_ei.
• the number of photoproduction reactions : n_pp.
• the number of species produced by the photoproduction reactions (if two reactions produce the same ion) : n_spe_pp.
• the number of EUV wavelength for the photoproduction computations: nb_lo.
• the number of photoproduction reactions whose rate does not depends on the wavelength : n_pp_freq_fixed.

The *atmosphere* structure is obtained by calling:
allocate_atmosphere(ncm,atmosphere,density,prod_pp). Example:

```
atmosphere%n_species=7 !number of exospheric species (neutral and ions)
atmosphere%n_pp=4 !number of photoproduction reactions
atmosphere%n_spe_pp=3 !number of species obtained by photoproduction
atmosphere%n_pp_freq_fixed=0 !number of photoproduction reactions with fixed rates
atmosphere%nb_lo=37 !number of wavelength in the EUV spectrum
atmosphere%n_exc=4 !number of charge exchange photoreactions
atmosphere%n_ei=2 !number opf electron impact reactions
call allocate_atmosphere(ncm,atmosphere,density,prod_pp)
```

Each species has then to be defined, one should precise:
• the species mass.
• the species charge.
• if the species opacity in EUV must be used (.TRUE. or .FALSE.).
• if the species must be load as ionospheric particle (.TRUE. or .FALSE.).
• the pointer to the photoproduction array for photoproduced species (only).

• the species name (no blank in front).
Examples:

```
atmosphere%species(1)%name = "O+ "
atmosphere%species(1)%mass = 16._dp
atmosphere%species(1)%charge= one
atmosphere%species(1)%opaque= .FALSE.
atmosphere%species(1)%iono = .TRUE.
atmosphere%species(1)%prod => prod_pp(:,:,:,1)


atmosphere%species(2)%name = "O "
atmosphere%species(2)%mass = 16._dp
atmosphere%species(2)%charge= zero
atmosphere%species(2)%opaque= .TRUE.
atmosphere%species(2)%iono = .FALSE.

atmosphere%species(3)%name = "H "
atmosphere%species(3)%mass = 1._dp
atmosphere%species(3)%charge= zero
atmosphere%species(3)%opaque= .TRUE.
atmosphere%species(3)%iono = .FALSE.
```

6.2 Photoproduction reactions

The photoproduction has to be defined in the alloc_*planetname* subroutine. It is defined in the *atmosphere* structure, by setting pointers to the mother and daughter species. Example:

```
!O->O+
atmosphere%photo_reactions(1)%mother => atmosphere%species(2) !O species
atmosphere%photo_reactions(1)%daughter => atmosphere%species(1) !O+ species
```

The fields atmosphere%EUVFLX (EUV flux), atmosphere%ion_abs (absorption cross-section) and atmosphere%ion_react (ionization cross-section) will have to be filled before calling the subroutine photoproduction_generic(Spe,ncm,gstep, s_min_loc,atmosphere), which computes the prod_pp arrays in a generic way. Note that if these values (generally obtained through calls to routines of the atm_sections_efficaces.F90 module) are stored by alloc_*planetname*, you do not need to write a specific photoproduction routine and can compute the photoproduction rate by associating directly the photoproduction pointer to the photoproduction_generic routine in `environment.F90`.

6.3 Charge exchange reactions

The charge exchange reactions have to be defined in the alloc_*planetname* subroutine. they are defined in the *atmosphere* structure, by setting pointers to the mother neutral and ion species as well as the charge/mass ratio of the mother ion specie and the cross-section of the reaction (if constant). Example:

```
!H +O+ ->H+ + O
atmosphere%exc_reactions(1)%qsm = one/16._dp
atmosphere%exc_reactions(1)%ion =>atmosphere%species(1)
atmosphere%exc_reactions(1)%neutral =>atmosphere%species(3)
atmosphere%exc_reactions(1)%cross_section = 9.E-20
```

The field atmosphere%cross_section will have to be filled before calling the subroutine charge_exchange_generic(nn,n2,pickup,qsm,irand, ijk,s_p,v_p,w,Spe,particule,atmosphere), which implementes the charge exchanges in a generic way. Note that if this valueis constant and is stored by alloc_*planetname*, you do not need to write a specific charge exchange routine routine and can compute the charge exchange rate by associating directly the charge_exchange pointer to the charge_exchange_generic routine in `environment.F90`.

6.4 Electron impact reactions

The electron impact reactions have to be defined in the alloc_*planetname* subroutine. they are defined in the *atmosphere* structure, by setting pointers to the mother (neutral) and daughter (ion) species as well as the cross-section of the reaction defined by $\sigma = \sum_{i=1}^{n} \mathsf{coeff}(i) * \ln(T_e(K))^{i-1}$. Example:

```
!O + e-> O+ atmosphere%ei_reactions(2)%ion =>atmosphere%species(1)
atmosphere%ei_reactions(2)%neutral =>atmosphere%species(2)
atmosphere%ei_reactions(2)%n = 5
atmosphere%ei_reactions(2)%coeff(1:5) = (/-1233.29,347.764,-37.4128,1.79337,-0.032277/)
```

The electron impact reactions are automatically implemented if defined in atmosphere, without needing further declaration in environment.F90.

7 The Makefile

The `Makefile` is a gnu Makefile and permits to compile easily the `quiet_plasma` and `diag` binaries.

7.1 Compiler choice

It contains the COMPILER flag where you have to put the path toward the mpi fortran. Then there are three flags which permit to select the compile option for anyone of the three compiler tested:

IFORTFLAG Compilation flags for intel fortran (tested with version 10.1 and 11.1).

GFORTFLAG Compilation flags for gfortran (tested for versions ≥ 4.0).

PGFORTFLAG Compilation flags for Portland Group fortran (work for version ≥ 10).

To select the good compiler flags you have to change the value of the FFLAGS variable accordingly with the compiler chosen.

The source files are compiled by "make" command. The corresponding object files are listed in the `Makefile` by respecting the dependencies between the modules. At the moment the program structure and the object file list permits to compile the program in parallel with two threads. So with the command "make -j 2" you can compile `quiet_plasma` very quickly.

7.2 Compilation Options

The `quiet_plasma` program have some compilation options which enabling or not some *PREPROCESSOR MACROS* permits to compile the program for some specific purposes.

HAVE_TIMING Complete time analysis.

HAVE_NETCDF Read and Write of diagnostic files in NetCDF format.

HAVE_DEBUG Enable debug. For any called routine, the begin and the end are annotated. More controls on the quantities. Add also the good path to headers and libraries in the INC_CDF and LIB_CDF flags.

HAVE_DOUBLE_PRECISION Use double precision real instead of simple precision.

HAVE_NO_PLANET No planet is not considered, only the environment.

HAVE_WAVE_TEST The input plasma is a Alfven wave. Use with HAVE_NO_PLANET.

HAVE_MAXSIZEWRITE If in writing, the system have some limitation in the buffer size, then this option permits to write some big quantities (like particles) not at once.

To activate one of more of the compilation option add -DHAVE_"option" to the compilation flag in use.