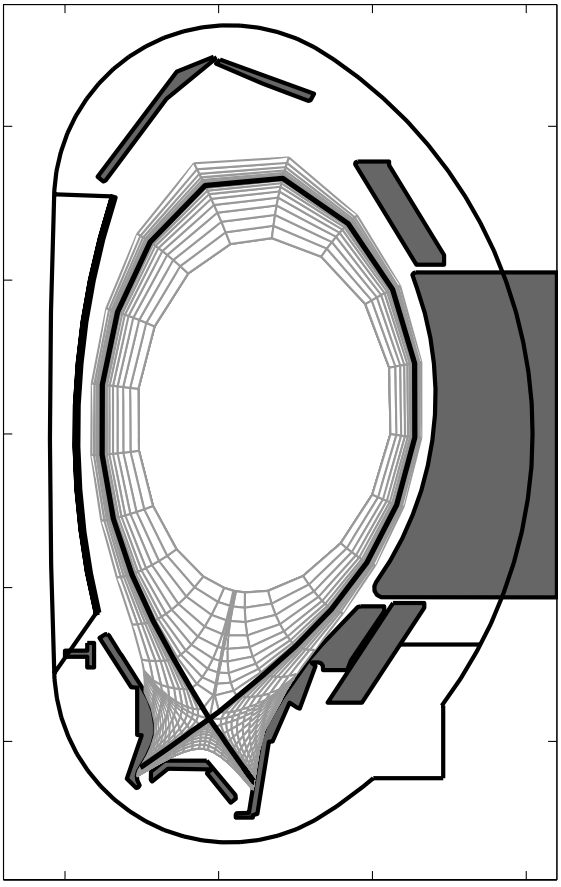


Max-Planck-Institut
für Plasmaphysik

Division of Plasma Edge and Wall

SOLPS-ITER



CONTENTS

1 - The SOLPS-ITER code package 5

2 - Physics modules in SOLPS-ITER 9

3 - Basic UNIX commands and computing 11

4 - Download, installation and compilation of the code 14

5 - Structure of the directories 17

6 - Computational domain of SOLPS-ITER 20

7 - Definition of the physical variables 26

8 - Constructing a SOLPS-ITER grid 38

9 - Setting up the input parameters of B2.5 64

10 - The EIRENE input file 90

11 - Running SOLPS-ITER 114

12 - Checking the results of a run 119

13 - Visualizing the results of a run 128

14 - Post-processing tools for graphical output 136

15 - Post-processing tools for numerical balances

16 - Using MDSplus

1 - The SOLPS-ITER code package

SOLPS-ITER is the newest version of the SOLPS (Scrape-Off Layer Plasma Simulator) code package, one of the most sophisticated tools for plasma edge modelling, developed in order to make numerical simulations of plasma behavior in the scrape-off layer (SOL) region of fusion plasmas. It has been widely used for both interpretative and predictive purposes in the major currently operating tokamaks, such as ASDEX Upgrade, JET, Alcator C-Mod and DIII-D; moreover, it is currently the most employed tool to assess the performances of future devices, as well as to perform studies on the plasma-facing components.

SOLPS-ITER consists in the coupling of a 2D multi-fluid plasma transport code and a 3D kinetic Monte Carlo neutral transport code. The main modules contained in SOLPS-ITER are:

- DivGeo, a graphical tool used for developing the input plasma grid starting from the material structure and the magnetic equilibrium configuration of the scenario to be modelled, originally written with about 35000 lines of C code. This requires the upload of the two-dimensional poloidal cross section of the main plasma-facing components in the considered experimental setup. Such cross sections can be available in the SOLPS databases, or can be generated using DivGeo itself starting from a technical drawing file. Then, the equilibrium poloidal magnetic flux surfaces of the considered experiment must be also uploaded. The grid resolution should be higher when strong gradients in plasma parameters are expected to exist, but can be relatively coarse in regions of small or negligible gradients. Grids usually therefore have higher spatial resolution in the poloidal direction close to the targets and around the X-point, whilst the highest spatial resolution is requested around the separatrix in the perpendicular direction.

- Carre, a mesh generating program, written with about 20000 lines of FORTRAN 77 code. This uses the output files created by DivGeo to generate the curvilinear mesh used in the simulations, which aligns strictly with the magnetic field lines. Numerical calculations are then performed on a topologically rectangular mesh, i.e. the computational domain. Transformation between the physical geometry and this mesh is provided through metric coefficients that are included in the output of Carre.

- B2.5, a computational fluid code for the plasma, written with about 90000 lines of FORTRAN 90 code. This solves the plasma transport equations (precisely the modified Braginskii equations for multi-species plasmas, in which an arbitrary number of ionic species can be included). At each iteration corresponding to a single time step, volumetric and surface source terms are computed, solving for momentum conservation, continuity, energy conservation and finally again for the continuity equation. The above procedure is repeated for a number of internal iterations (usually 10) to relax the equation solutions before proceeding to the next time step (external iteration step). The process is repeated until convergence is reached. Steady state is achieved if, independently of the time step chosen, no 'major' variation of plasma parameters is observed. This must include steady state of the densities, temperatures, energy and particle fluxes at various locations in addition to the total particle and energy content of the plasma. B2.5 includes the treatment of currents and drifts too, even if they are not included in the most basic version of the program.

- **EIRENE**, a Monte-Carlo kinetic code for neutrals transport, written with about 105000 lines of FORTRAN 90 code. This solves linear transport equations for test particles in a triangular mesh of arbitrary complexity by means of Monte-Carlo methods, in order to compute statistical expectation values of complex processes between such test particles and the plasma, following the history of a number of test particles launched in the plasma following a prescribed distribution. All events which can occur on trajectories have probabilities depending on their nuclear cross sections. The lengths of the trajectories are randomly generated, starting from the mean free path of the particles, and each test particle is followed along its trajectory until it is either absorbed or ionized; a number of histories are created, from which averaged values may be derived. The input data in EIRENE, that are vessel geometry, grid and atomic/molecular data, are specified though a formatted input file. The version of EIRENE included in SOLPS-ITER is parallelizable (since stories of different neutrals are independent one from each other), so that it is possible to carry out coupled runs with through full parallel computing schemes.

Historically, there have been many implementations of the SOLPS code package, according to which version of the fluid plasma code (B2 or B2.5) and of the EIRENE code are used. We distinguish between SOLPS 4.x which uses B2 and SOLPS 5.x which uses B2.5. SOLPS-ITER, in particular contains the B2.5 plasma kernel and the most recent version of EIRENE.
The current SOLPS-ITER distribution, as provided by a GIT repository maintained at the ITER organization (https://git.iter.org/), includes all the source code and documentation associated with the various executables. Because this package is made available to a wide variety of users operating in different computing environments, the source code used must adhere to strict FORTRAN and C standards and the compilation files must allow for this diversity. This is achieved by determining, at session start-up, the current location and compiler choice and passing that information using environment variables at compilation time.

SOLPS-ITER could work both in **standalone mode** (that is with the fluid module only running) or in **coupled mode** (that is with the kinetic module running as well). In the first case, since it is made use of the fluid module only, the fluid equations are solved for both charged plasma particles and neutral particles. In the second case, instead, plasma particles are described using the fluid module, while the transport of neutral species and their interaction with the formers and the material surfaces is described by the Monte Carlo module; in detail, in this case the coupling acts with the surface and volumetric sources/sinks for the plasma in B2.5 due to the interactions with neutrals being computed by EIRENE. The main advantage of the standalone mode is its much faster achievement of the results and the avoidance of the typical Monte Carlo noise in the results; however, simulations done in coupled mode are quite more accurate, especially for cases in which the neutral density is relatively high.
The coupling procedure, actuated at each time step of B2.5, presents the following scheme:

1. When a plasma background has been prepared by B2.5, it is sent to EIRENE.
2. A number of neutral particle histories is simulated by EIRENE in the plasma background.
3. The neutral-plasma interactions modelled by EIRENE are transferred again to B2.5 in the form of sources and sinks for particles, momentum and energy.
4. B2.5 re-computes the plasma solution by solving the conservation equations foe electrons and ions, after which a new plasma background is prepared, which will be used in the successive iteration.

The main goal of SOLPS-ITER is to fully capture a steady-state situation, obtained at the end of a time-dependent convergence process in which the plasma evolution is followed by solving the equations included in it, consistently with the just described inputs. Once this has been reached, it is possible to extract all the information of interest for the modelled scenario.
Usually, some aspects of the physics are most easily understood if physics in the code is simplified: this means that runs are often performed with pure hydrogen, neglecting any drifts of plasma particles and

implementing the simplest choice for the anomalous transport model. For quantitative comparison with experiments, runs might be expanded to include impurities, which can be intrinsic or extrinsic (in this former case a production model, e.g. physical and/or chemical sputtering, needs to be added), and/or to include drifts terms in the solved conservation equations, and/or to fit the experimental data in providing an empirical anomalous transport model.

SOLPS-ITER solves the equations describing the behavior of a multi-fluid plasma in the edge region of magnetic fusion devices, consisting in a thin annulus of the outer core plasma and the entire scrape-off layer, fully including private flux region and divertor region. For doing this, it takes as input:

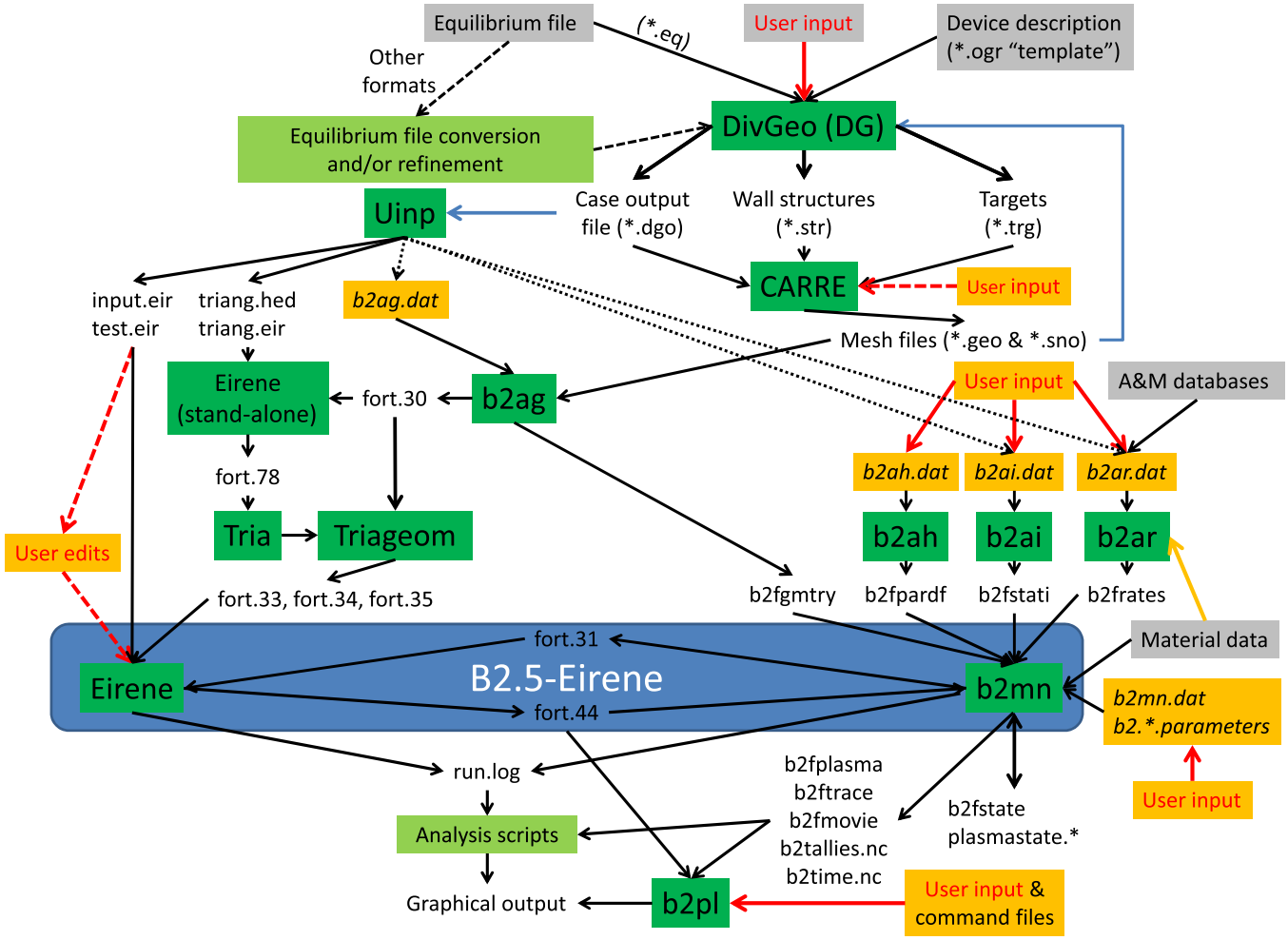
- A **computational grid**, derived from the discretization of the 2D poloidal cross section of the plasma following the magnetic equilibrium configuration for a particular experimental scenario, in which the fluid and kinetic equations are solved through finite-difference calculations.
- A set of **boundary conditions** about physical inputs, e.g. considered atomic species, density regime and input power, and properties of the plasma-facing materials, e.g. geometry of the material surfaces and models for the plasma-material interactions.
- A choice of empirically evaluated **anomalous transport coefficients**, such as particle diffusivities and electron/ion thermal diffusivities.

The main steps required in building a SOLPS-ITER run are:

- **Geometry definition**: using the DivGeo utility we establish the location of the material surfaces inside the vacuum vessel (including main wall, divertor targets, limiters etc.), their properties, the pumping locations, the topology of the equilibrium magnetic flux surfaces, and the present plasma species.
- **Plasma mesh generation**: using the Carre mesh generator we prepare a structured quadrangular mesh to be user by the plasma solver, which covers an area going from the edge of the confined region to the line of first tangency with a limiting wall structure.
- **Input file generation**: using the data just provided, the Uinp utility is used to create the corresponding EIRENE and B2.5 input files.
- **Neutral mesh generation**: with such input files, the Triang programs create an unstructured triangular grid containing the area gridded by Carre plus the outer SOL domain up to the outest vacuum vessel wall.
- **Physics input**: other plasma parameters are translated by means of the programs b2ag (geometry), b2ah (transport and boundary conditions), b2ar (atomic processes databases) and b2ai (initial state) into the internal format used by B2.5. These are supplemented by a list of run switches and additional namelists provided as stand-alone human-readable input files.

With the input files stablished, it is then possible launch runs of either EIRENE or B2.5 in stand-alone mode, or coupled B2.5/EIRENE runs.

A basic SOLPS-ITER run is obtained by following the workflow in figure from top to bottom, where the main plasma solver is represented by the blue box; then, each of the green boxes is another separate executable, grey boxes represent data provided by external databases, and orange boxes represent the user inputs.



Everything above the blue box is input preparation (which need not to be repeated for each run, apart from eventual input parameter changes). Below the blue box, instead, we find the post-processing phase, consisting in the analysis of the final results by means of opportune scripts to work on the output files, and eventually the graphical output of the same.

2 - Physics modules in SOLPS-ITER

As already mentioned, **B2.5** is the fluid module of SOLPS-ITER, completely written in FORTRAN 90, which computes the evolution of the charged plasma particles. Its first version, consisting in the earliest version of the B2 program, was originally developed by B. Braams in 1986; this version was then improved in the geometry and enlarged by adding several terms in the fluid equations which were firstly neglected, such as drifts and currents. However, these earliest versions still contained major simplifications. In the following years, other physical processes originally neglected have been added, and the coupling with EIRENE was successfully achieved.

In particular, it solves the fluid transport equations (precisely a modified version of the **Braginskii equations**) for the electrons and an arbitrary number of ion species, the last ones sharing a same temperature, basing on a finite volume discretization method .
At each numerical iteration, corresponding to a single time step, volumetric and surface source terms are computed, and the conservation equations (particles, momentum and energy, in addition to potential if currents are also considered) are solved. This is repeated for a number of "internal iterations" to relax the equation solutions before proceeding to the next time step, or "external iteration".
The equations are solved for the necessary number of time steps until the convergence is reached, which is monitored by analyzing the norm of the residuals of each conservation equation. Steady state is achieved if, independently of the time step chosen, no major variation of the main plasma parameters is observed. Currently, B2.5 can be run with time steps of $10^{-3} - 10^{-4}$ s when no steep time gradients are expected to occur during the convergence procedure (e.g. when drifts and currents are neglected); otherwise, time steps down to $10^{-5} - 10^{-6}$ s could be required, slowing down considerably the convergence process. Depending on the plasma conditions to simulate, a new case may require about 10^4 to 10^5 iterations before an acceptably converged steady state is obtained.

The equations are solved on a **curvilinear grid**, covering the entire plasma edge region, which strictly aligns with the magnetic field lines; for this reason, this grid must be prepared by importing the magnetic equilibrium configuration of the particular scenario to be modelled. The numerical calculations are performed on a topologically rectangular mesh: the transformation between the physical geometry and the computational mesh is provided in each cell through a set of metric coefficients. The resulting curvilinear orthogonal system is characterized by a x direction representing the orthogonalized **poloidal coordinate** θ (with positive direction from the inner to the outer divertor target) and a y direction representing the orthogonalized **radial coordinate** r (with positive direction from the core outwards).
The grid resolution is generally adjusted so that it results higher in directions where strong gradients in plasma parameters are expected to exist, but it is left relatively coarse in directions of small or negligible gradients. Higher resolutions are then required in the divertor region along the poloidal direction and around the separatrix in radial direction.
During a run, **boundary conditions** must be specified for each conservation equation. These are usually specified in terms of fluxes or derivatives of the fluxes at the external grid boundaries, in order to act as sources or sinks for the equations at the outermost grid cell. Furthermore, it is also possible to specify these in form of feedback schemes, i.e. adapting the magnitude of external physical outputs so that the desired values of some plasma quantities are achieved; for example, one can specify to control the external gas puff in this way so that a desired value of density at the outer midplane separatrix is obtained.

Then, **EIRENE** is the module dedicated to the modelling of neutral species, also written in FORTRAN 90. Its first version was developed by D. Reiter in 1984, but several updates have been introduced since then. For example, the most recent version EIRENE-2010, which is the one included in SOLPS-ITER, includes the enlargement of the considered atomic and molecular processes, the possibility to simulate radiation losses and neutral-neutral and photon-neutral collisions, and the possibility to run it in parallel mode (that is following more histories for each step).

In particular, it solves the linear **kinetic transport equations** for the neutrals as **test particles** in 3D volumes by means of Monte Carlo methods; namely, it computes the statistical expectation values for the processes between such test particles and the plasma, following the history of a number of these when launched in the plasma.
The operating principle of EIRENE is as follows. A number of test particles is launched in a background medium from some particle source, with a given distribution in directions and velocities; in case of a coupled SOLPS-ITER run, such background medium is just the plasma as computed at each time step by B2.5. The lengths of the trajectories travelled by these are randomly generated, although proportional to the neutrals mean free path. Each test particle is followed, along its trajectory through the medium, until it is either absorbed at a material surface or ionized becoming a particle of the plasma background. All the events which can occur on the trajectories of test particles have probabilities depending on the cross sections of the related atomic process. At the end, a number of histories of test particles will have been created, from which averaged values for the related physical quantities in each point are derived.
Rate coefficients for the atomic processes which test particles can undergo when interacting with the particles of the plasma background are provided by external atomic physics databases such as AMJUEL and HYDHEL.
Computations of EIRENE are done on tetrahedral volumes in a full 3D geometry, which reduces to a triangular grid in the poloidal plane. This extends over the entire grid of B2.5 and even further, up to the material boundaries of the main chamber wall. So, the grid boundaries coinciding with the material surfaces can be characterized by some absorption and reflection probabilities to take into account the interactions of the neutrals with the solid surfaces.

3 - Basic UNIX commands and computing

For using SOLPS-ITER it is necessary to use an **UNIX environment**. For this reason, a brief overview of the most important commands to navigate in a UNIX terminal is now given.

- `cd`

Changes directory to the home directory of the system.

- `cd dir1`

Enter the subdirectory "`dir1`" located in the current directory.

- `cd ..`

Go back to the parent directory of the current directory.

- `cp file1 file2`

Copies the content of the file "`file1`" in a file "`file2`" (eventually overwriting it if it already exists).

- `cp file1 dir1`

Copies the file "`file1`" in the already existing directory "`dir1`".

- `cp -r dir1 dir2`

Recursively copies the directory "`dir1`", together with all the files/subdirectories in it, as a subdirectory of the directory "`dir2`"; if "`dir2`" does not already exist, it is created containing all the files/subdirectories of "`dir1`".

- `date`

Displays the currant date and time.

- `diff file1 file2`

Reports line-by-line the differences between the text files "`file1`" and "`file2`".

- `diff dir1 dir2`

Sorts the contents of the directories "`dir1`" and "`dir2`" by name, and then run diff on the text files which differ.

- `file file1`

Tells at which category the file "`file1`" belongs (source program, executable program, shell script, empty file, directory, library).

- `find . -name '*.f' -print`

searches the current directory and all its subdirectories for files ending for example in `.f` and writes their name to the standard output.

- `find /dir1 -name file1 -user user1 -print`

searches the directory "`dir1`" and all its subdirectories for files called "`file1`" belonging to the user "`user1`" and writes their full names to the standard output.

- `finger`

gives information on all user currently logged onto the used machine.

- `finger user1`

gives further information about the user "`user1`" currently logged.

- `gedit`

opens a text editor.

- `grep pattern1 file1`

searches for lines containing the pattern "`pattern1`" in the file "`file1`" and writes them in the output

- `ls`

Lists the content of the current directory.

- `ls dir1`

Lists the content of the directory "`dir1`" located in the current directory.

- `ls -ltr`

Lists the content of the current directory in order of time the files were last altered.

- `ls -l file1`

Gives detail for the access permissions for the file "`file1`", its size in kB, and the time it was last altered.

- `man command1`

Displays the manual page for the command "`command1`", with all the information about it and its options.

- `mkdir dir1`

Creates the new subdirectory "`dir1`" in the current directory

- `mkdir -p dir1/dir2`

Created the new subdirectory "`dir2`" in its parent directory "`dir1`" if this does not already exist.

- `mv file1 file2`

Changes the name of the file "`file1`" into "`file2`" unless "`file2`" already exists, in which case "`file1`" will overwrite "`file2`".

- `mv dir1 dir2`

Changes the name of the directory "`dir1`" to "`dir2`" unless "`dir2`" already exists, in which case "`dir1`" will be moved into "`dir2`".

- `mv file1 dir1`

Moves the file "`file1`" into the directory "`dir1`".

- `pwd`

Gives the complete address of the current directory.

- `rm file1`

Deletes the file "`file1`" located in the current directory.

- `rm -r dir1`

Recursively deletes the directory "`dir1`" and all the files/subdirectories in it.

- `ssh machine_name`

Logs into the machine `machine_name`.

- `exit`

The first time for eventually leaving the machine in which we are currently logged, another time for closing the terminal.

For performing computing at the **Max-Planck-Institut für Plasmaphysik** in Garching, it is necessary to connect to the **TOK Linux cluster**, which is possible through 8 different **login nodes** (`tok01`, `tok02`... `tok08`). There are more than one hundred **computing nodes**, each one supported by two Intel Xeon Gold 6130 CPUs, each one with 16 cores and 2.1 GHz base clock speed (thus 32 CPU cores per node) and 192 GB memory. The operating system is Suse Linux Enterprise Server 15.

It is possible to access to the TOK cluster from any machine, in the **terminal** window. If the machine is already connected to the IPP network, it is simply possible to access by entering, for example,

- `ssh tok01`

From any other network connection, instead, this is possible after connecting to the IPP VPN service and then entering the complete address, that is, for example,

- `ssh {USER}@tok01.bc.rzg.mpg.de`

Several programs are available on the TOK cluster, such as visualization tools, compiling tools, scientific programming tools, etc. In order to see which programs are available, we can enter

- `module avail`

while, for opening one particular program, we can enter

- `module load {MODULE}`
- `{MODULE}`

Data are stored in the AFS **global file system**, accessible from any machine. The path to the home directory is `/afs/ipp-garching.mpg.de/home/{U}/{USER}/`. This is immediately accessible, once logged in the TOK cluster, through

- `cd /afs/ipp-garching.mpg.de/home/{U}/{USER}/`

For computing processes, two different file systems are available, one for series computing and one for parallel computing.

- `/toks/`, which is intended for **series computing** (usually devoted then to light works). The possible paths are `/toks/work/{USER}/` and `/toks/scratch/{USER}/`.
- `/tokp/`, which is intended for **parallel computing** (usually devoted to heavier works). The possible paths are `/tokp/work/{USER}/` and `/tokp/scratch/{USER}/`.

For each file system, the path `/work/` is intended for long term file storage; it has a quota of 2 TB, and is backed up automatically. The path `/scratch/`, instead, is intended for short term file storage; it has an unlimited quota, but is not backed up.
So, for example, to access the `/work/` path of the `/toks/` file system, we must enter

- `cd /toks/work/{USER}/`

while to access the `/work/` path of the `/tokp/` file system, we must enter

- `cd /tokp/work/{USER}/`

In the cluster three different job types can be submitted:

- **Serial jobs** (on the `/toks/` file system), which use one CPU core for each. A standard job can last up to 36 hours, and one single user can use up to 400 CPU ores simultaneously.
- **Single-node parallel jobs** (on the `/tokp/` file system), which use more CPU cores for each (up to 32 cores), each one on a single computing node. A standard job can last up to 24 hours, and one single user can use up to 256 CPU cores simultaneously.
- **Several-nodes parallel jobs** (on the `/tokp/` file system), which use more entire computing nodes for each (up to 24 nodes). A standard job can last up to 24 hours, and one single used can use up to 24 computing nodes (i.e. 768 CPU cores) simultaneously.

Among all the computing nodes present in the system, currently one third are reserved for serial jobs (partition `s.tok`), single-node parallel jobs (partition `p.tok.openmp`) and several-nodes parallel jobs (partition `p.tok`) each.

4 - Download, installation and compilation of the code

SOLPS-ITER is provided by a GIT repository maintained at the ITER organization, specifically at https://git.iter.org/projects/BND/repos/solps-iter/browse. Since these repositories are restricted, one will first need to authenticate with the personal ITER username and password. After having obtained the access to a SOLPS-ITER account, then for being able to access the GIT repository it is sufficient to upload a copy of the own public ssh key in the SOLPS-ITER account settings on the website.

First of all we access the cluster environment entering, for example,

- `ssh tok01`

Then, we must go to the directory, in the series or parallel computing branch of the cluster, where we want to locate the main SOLPS-ITER directory, as

- `cd /toks/work/{USER}`

or

- `cd /tokp/work/{USER}`

In this location, we can start the procedure for downloading and installing the code. In particular, we have to simply clone a copy of SOLPS-ITER here, by entering

- `module load git`
- `git clone ssh://git@git.iter.org/bnd/solps-iter.git`

This command will create a directory `solps-iter/` in the current location and will link it with the SOLPS-ITER GIT repository. Then, we enter the just created directory, as

- `cd solps-iter`

and we download each submodule, by entering

- `git submodule init`
- `git submodule update`

Finally, when we are still in the main SOLPS-ITER directory, we have to position oneself to the master branch of each of the just created subdirectories, by entering the sequence of commands:

- `git checkout master`
- `cd modules/B2.5`
- `git checkout master`
- `cd ../Eirene`
- `git checkout master`
- `cd ../DivGeo`
- `git checkout master`
- `cd ../Carre`
- `git checkout master`
- `cd ../adas`
- `git checkout master`

Now the installation is complete.

In each moment, if we want to update the code to its last master (i.e. official) version, we must go to the main SOLPS-ITER directory and then executing the automatic update script by entering

- `solps-iter_update`

If, instead, we want to update the code to its last development version, the must enter

- `solps-iter_update_develop`

This is possible only in the installation is in a clean state, i.e. with no locally modified files, unless they are committed to local branches.

For each local implementation of SOLPS-ITER, the two environmental variables {HOST_NAME} and {COMPILER} are used to designate the location in which the various libraries must be found, and the preferred compiler to be used. These variables are then propagates throughout the entire code structures.
In particular, the {HOST_NAME} variable is deduced by the whereami script, which can be found in the solps-iter/ directory, which checks the address of the system in which we are logged. For example, if we are logged in the TOK Linux cluster at IPP, then {HOST_NAME} = IPP. Then, the default_compiler script, which also can be found in the solps-iter/ directory, deduces the compiler platform to be used according to the system in which we are logged. Currently, the supported compiler platforms are:

- pgf90: Portland Group 32-bit FORTRAN compiler, used at JET and ORNL.
- ifort64: Intel 64-bit FORTRAN compiler, used at ITER and IPP.
- gfortran: GNU FORTRAN compiler, used at ZFJ and KUL.
- g95: FORTRAN95 compiler, used on the Marconi Gateway.

For example, in our case, then {COMPILER} = ifort64.
The different settings for each supported combination of these variables are specified in the files config.{HOST_NAME}.{COMPILER}, which can be found in the folder solps-iter/SETUP/.

So, once these variable are set, to start a SOLPS-ITER session, every time we must go into the solps-iter/ directory, and then enter

➢ source setup.csh

Through this setup command, the set of the corresponding environmental variables are loaded from the opportune one among the various setup.csh.{HOST_NAME}.{COMPILER} files, contained in the directory solps-iter/SETUP/.
Each time the setup command will also show where we are running the code and which compiler we are using. If desired, the default choice for the compiler for any location could be overridden by giving the desired compiler name as an argument following the setup command, i.e. entering

➢ source setup.csh {COMPILER}

For example, one of the environmental variables which are set up in this way is the {DEVICE} variable, which indicates the name of the fusion device which is most likely to be simulated for our {HOST_NAME}. The possible values are, for example:

- cmod: Alcator C-Mod.
- d3d: DIII-D.
- iter: ITER.
- jet: JET.
- upgrade: ASDEX Upgrade.

In our case, then, we load by default {DEVICE} = upgrade. Nevertheless, it is possible to change by default choice by entering

➢ setenv DEVICE {DEVICE}

If this is the first installation in our personal file space, then we should enter

➢ first setup

This will, if they are not already present, create the various config.{HOST_NAME}.{COMPILER} files for each of the SOLPS-ITER submodules.
Then, once the configuration files have been filled with the appropriate information for the local implementation desired, the code must be opportunely compiled. For this purpose, it is made use of a master Makefile, located in the solps-iter/ directory. Among all the supported options for the compilation, we have:

- `solps`: Compiles the necessary components for a full SOLPS-ITER implementation (to be done in the toks file system).
- `solps_mpi`: Compiles the necessary components for a full SOLPS-ITER implementation with MPI parallelization options (to be done in the tokp file system).
- `solps_openmp`: Compiles the necessary components for a full SOLPS-ITER implementation with OpenMP parallelization options (to be done in the tokp file system).
- `all`: The same as `solps`, but with in addition the B2.5 standalone module and the EIRENE standalone program (to be done in the toks file system).
- `all_mpi`: The same as `solps_mpi`, but with in addition the B2.5 standalone module and the EIRENE standalone program (to be done in the tokp file system).
- `all_openmp`: The same as `solps_openmp`, but with in addition the B2.5 standalone module and the EIRENE standalone program (to be done in the tokp file system).

For compiling the code in the opportune way, we just enter (while we are in the `solps-iter/` directory):

- `gmake listobj`
- `gmake tags`
- `gmake depend`
- `gmake {MODE}`

where `{MODE}` where is equal to one of the options specified above.
It is important to note that, after each update, the update script automatically recompiles the code to its standard version (i.e. without parallelization option); so, if we desire to activate the parallelization option, it is necessary to recompile it in the proper way after each update.

Now SOLPS-ITER is ready to be used. Before each session, to start the code is necessary to go to the `solps-iter/` directory, through

- `cd /toks/work/{USER}/solps-iter`

or

- `cd /tokp/work/{USER} /solps-iter`

and then enter

- `source setup.csh`

5 - Structure of the directories

Here the structure of the SOLPS-ITER directories is briefly described, focusing on the most important subdirectories.

At the **top directory** (`solps-iter/`) of SOLPS-ITER we found the file `setup.csh`, which we have to execute as first thing for loading the SOLPS-ITER environment in the terminal, and the folders:

- `doc/`
- `modules/`
- `runs/`
- `scripts/`
- `scripts.local/`

The **`doc/`** folder contains some useful pdf documents, like manuals, guides, theses, etc.

The **`modules/`** folder contains the various modules presents in the code, that are:

- `amds/`: automated generator of atomic physics blocks for the Monte Carlo code. It contains the links to the datafiles describing the various reaction rates, some default `.amd` files for reaction sets and the corresponding `.amo` output files. Moreover, we find:
  - the subfolder `amds/builds/`, which contains the compiled files for building the amds executable program, for the combination of hostname and compiler that we are using.
  - the subfolder `amds/config/`, which contains the default option files for each combination of hostname and compiler.
  - the subfolder `amds/src/`, which contains the source code files.
- `B2.5/`: source directories for the fluid code B2.5. It contains:
  - the subfolder `B2.5/builds/`, which contains the compiled files for building the B2.5 executable programs, for the combination of hostname and compiler that we are using.
  - the subfolder `B2.5/config/`, which contains the default option files for each combination of hostname and compiler.
  - the subfolder `B2.5/Database/`, which contains the data files used for computations (such as reaction rates, for example).
  - the subfolder `B2.5/src/`, which contains the source code files.
  - the subfolder `B2.5/src.local/`, which contains, eventually, the locally modified source code files.
- `Carre/`: grid generator for the fluid code. It contains:
  - the subfolder `Carre/builds/`, which contains the compiled files for building the Carre executable program for the combination of hostname and compiler that we are using.
  - the subfolder `Carre/config/`, which contains the default option files for each combination of hostname and compiler.
  - the subfolder `Carre/meshes/`, which contains, for each devices, the `.geo` mesh files which have been created for any simulated cases.
  - the subfolder `Carre/src/`, which contains the source code files.
  - the subfolder `Carre/src.local/`, which contains, eventually, the locally modified source code files.

- `DivGeo/`: graphical interface to the grid preprocessor. It contains:
  - the subfolder `DivGeo/builds/`, which contains the compiled files for building the DivGeo executable program, for the combination of hostname and compiler that we are using.
  - the subfolder `DivGeo/config/`, which contains the default option files for each combination of hostname and compiler.
  - the subfolder `DivGeo/device/`, which contains, for each devices, the `.sno` mesh files which have been created for any simulated cases.
  - the subfolder `DivGeo/src/`, which contains the source code files.
  - the subfolder `DivGeo/src.local/`, which contains, eventually, the locally modified source code files.
  - the subfolder `DivGeo/topologies/`, which contains some default files describing any possible magnetic topology for characterizing the equilibrium of the case to be simulated.
- `EIRENE/`: source directories for the Monte Carlo code EIRENE. It contains:
  - the subfolder `EIRENE/builds/`, which contains the compiled files for building the EIRENE executable programs, for the combination of hostname and compiler that we are using.
  - the subfolder `EIRENE/config/`, which contains the default option files for each combination of hostname and compiler.
  - the subfolder `EIRENE/Database/`, which contains the datafiles used for computations (such as the datafiles AMJUEL, HYDHEL, METHANE, SPUTER etc.).
  - the subfolder `EIRENE/src/`, which contains the source code files.
  - the subfolder `EIRENE/src.local/`, which contains, eventually, the locally modified source code files.
- `Triang/`: triangular mesh generator for the Monte Carlo code. It contains:
  - the subfolder `Triang/builds/`, which contains the compiled files for building the Triang executable program, for the combination of hostname and compiler that we are using.
  - the subfolder `Triang/config/`, which contains the default option files for each combination of hostname and compiler.
  - the subfolder `Triang/src/`, which contains the source code files.
  - the subfolder `Triang/src.local/`, which contains, eventually, the locally modified source code files.
  - the subfolder `Triang/triangles/`, which contains, for each device, the files describing the triangular meshes which have been constructed for any simulated cases.
- `Uinp/`: Uinp input file generator. It contains:
  - the subfolder `Uinp/builds/`, which contains the compiled files for building the Uinp executable program, for the combination of hostname and compiler that we are using.
  - the subfolder `Uinp /config/`, which contains the default option files for each combination of hostname and compiler.
  - the subfolder `Uinp /src/`, which contains the source code files.
  - the subfolder `Uinp /src.local/`, which contains, eventually, the locally modified source code files.

The **`runs/`** folder contains the actual runs of the various simulations. Here further directories can be created to separate categories of runs, with arbitrary depth of subdirectories. The `runs/` folder also contains a folder of example cases.
When we are in subdirectory if interest of a considered case, it will contain a `baserun/` folder, containing the defaults for all of the other directories at its same level, and, at the same level, the actual **run directories**.

The **scripts/** folder contains the various scripts included in the code. The most important of these serve for computing the residuals and making plots of the output values.
In each case, for reading the lines of code of which each of these scripts are written, we can just execute

- `vim 'script_name'`

The **scripts.local/** folder contains, eventually, the locally created scripts.

There are some aliases which are useful for navigating in the SOLPS-ITER directory structure with which we can go directly to some important places. These are:

- **stop**

takes us the `solps-iter/` top directory.

- **sbb**

takes us to the main `B2.5/` directory.

- **ssc**

takes us to the main `Carre/` directory.

- **ssd**

takes us to the main `DivGeo/` directory.

- **sei**

takes us to the main `Eirene/` directory.

- **sst**

takes us to the main `Triang/` directory.

- **ssu**

takes us to the main `Uinp/` directory.

- **sbr**

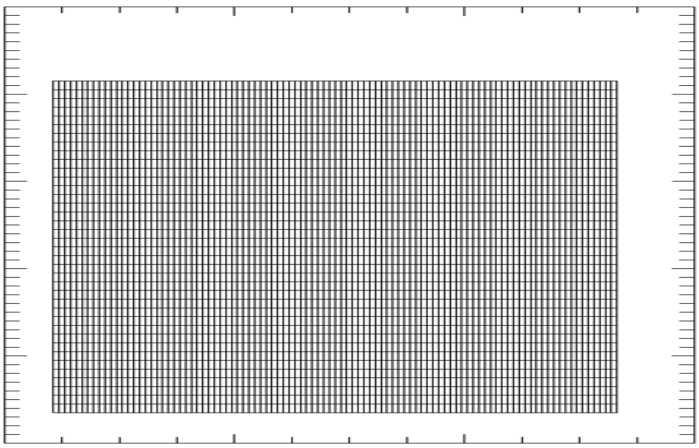
takes us to the `runs/` directory.

- **scr**

takes us to the `scripts/` directory.

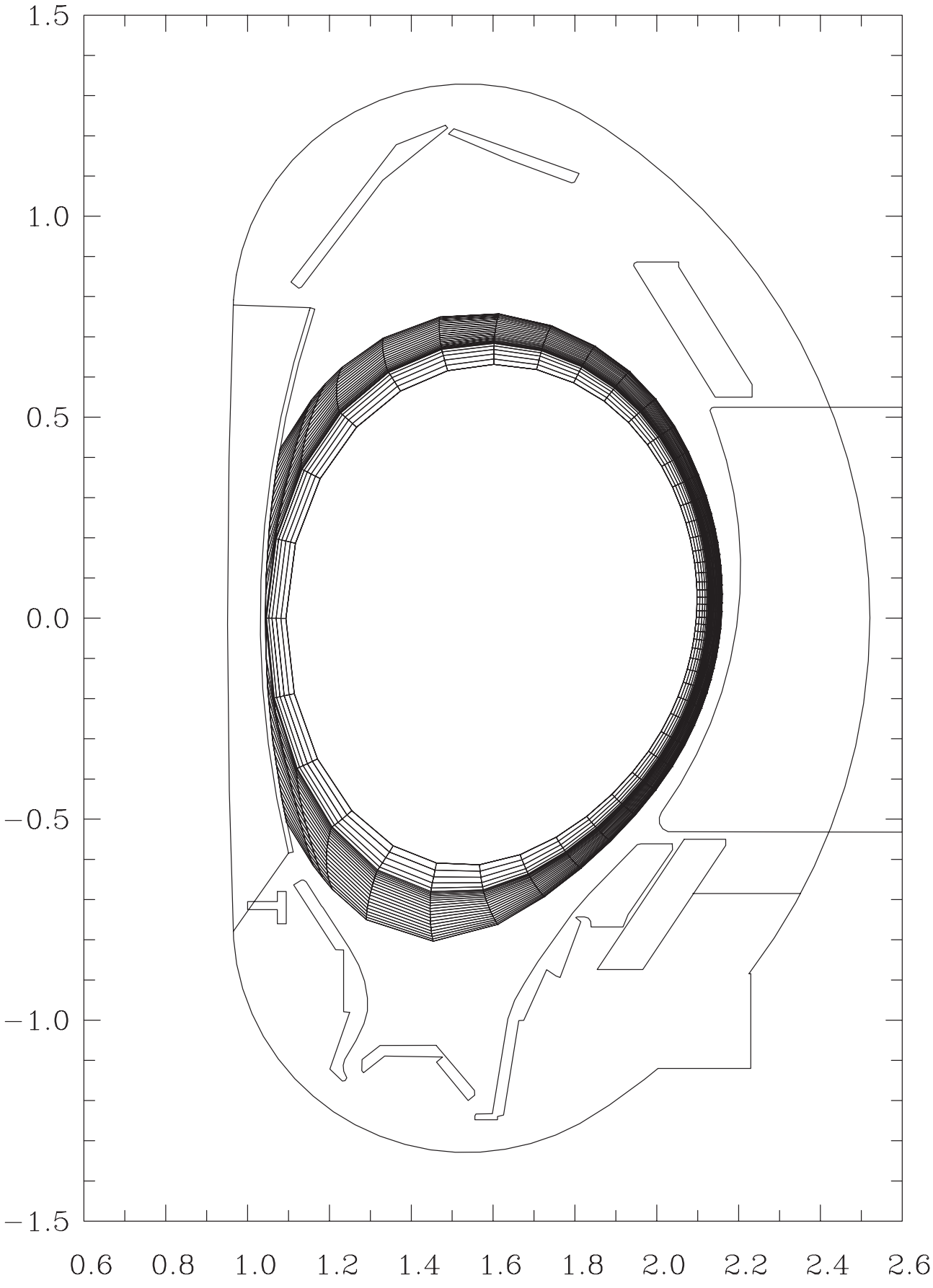
6 - Computational domain of SOLPS-ITER

The physical region of interest for plasma edge modelling with SOLPS-ITER consists of the outer section region of a poloidal cross section of a tokamak, resulting in a **computational domain** which includes the entire SOL (including the whole divertor region), the entire PFR and an annulus of the outer core plasma extending for some cm inside the separatrix. SOLPS-ITER, in particular, requires a mesh consisting of quadrilaterals, with two of the sides aligned along the magnetic flux surfaces of the considered scenario. The presence of steep gradients (which mostly occur close to the target plates and near the separatrix) defines the high-resolution regions for the grids.
In choosing the number and the distribution of cells we must take into account that the time-scale for convergence is often determined by the radial diffusion time in the core (being the slowest time-scale), and that the CPU time per iteration scales quadratically with the number of grid points. It is in fact important to balance the costs of the calculation with the physics which is desired to be captured, so that domains with high resolutions are used only for simulating physical aspects which require it.
After the knowledge of the considered magnetic equilibrium configuration, the grid generator code included in the package outputs the 2D coordinate of each cell corner in the (R,z) coordinates, and the values at the cell centers of the poloidal θ (varying along magnetic flux surfaces) and radial r (orthogonal to the magnetic flux surfaces) coordinates. This also outputs some metric coefficients for each cell grid, which describe the orthogonalization of the physical domain to a topologically rectangular domain, i.e. the coordinate transformation (θ,r) --> (x,y), obtained through conformal mapping of the magnetic configuration to be modelled.
In this way, the actual computations, performed as finite difference calculations, are done on such a topologically rectangular mesh with orthogonal cell coordinates (x,y), related to the physical geometry after having specified for each cell the discrete metric functions characterizing the coordinate transformation. The advantage of the orthogonality of the coordinate system is, indeed, that it ensures that all the fluxes out of a volume element can be specified between the cell itself and those neighboring its faces, rather than involving other cells adjoining just its edges. Orthogonality therefore simplifies the discretized equations and reduces the computing times.



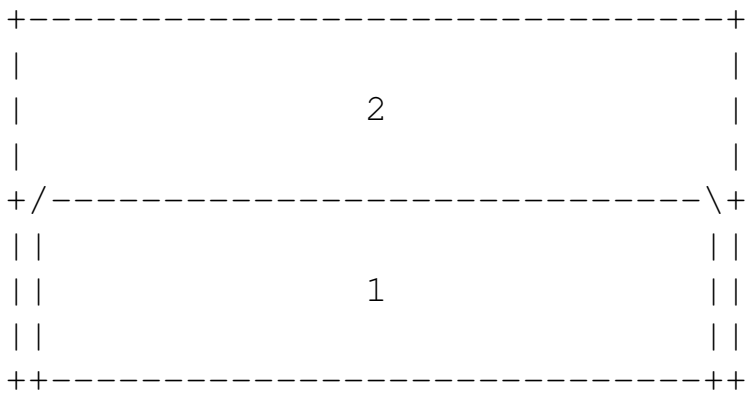
SOLPS-ITER has the flexibility to simulate a wide range of experimental magnetic topologies, being able to consider both diverted and non-diverted plasmas and, in the first case, the presence of one or more X-points.
In all the possible configurations, the **x direction** always represents the orthogonalized **poloidal coordinate**, while the **y direction** always represents the orthogonalized **radial coordinate**.
Then, according to the chosen configuration, the computational domain is subdivided in a number or default **regions**, whose number is defined by the parameter **NREG**. The classification of the **boundaries** between the various regions follows a NSWE classification, with west/east boundaries occurring along the x direction and the north/south boundaries occurring along the y direction.

In the case of **limiter geometry** there are no X-points, and the SOL plasma is directly limited by the main chamber wall. The separatrix here is defined as the magnetic flux surface which is tangent to wall, and the targets are defined as the portions of the wall on which the open magnetic field lines strike. Here is how the physical domain appears.



In this case we have `NREG = 2` so that all the variables in B2.5 which need to be specified for each region will form strings of `NREG + 1 = 2 + 1` values, in the form `(0:NREG)`.
In the computational domain we have the following region indexing and definitions:

1. Core
2. SOL



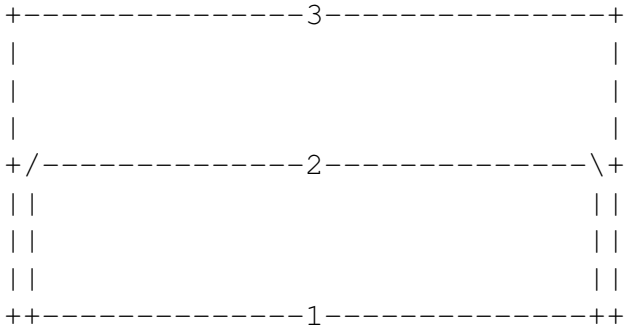
The resulting boundaries indexing and definitions, then, is:

- **west/east boundaries** (x-directed):



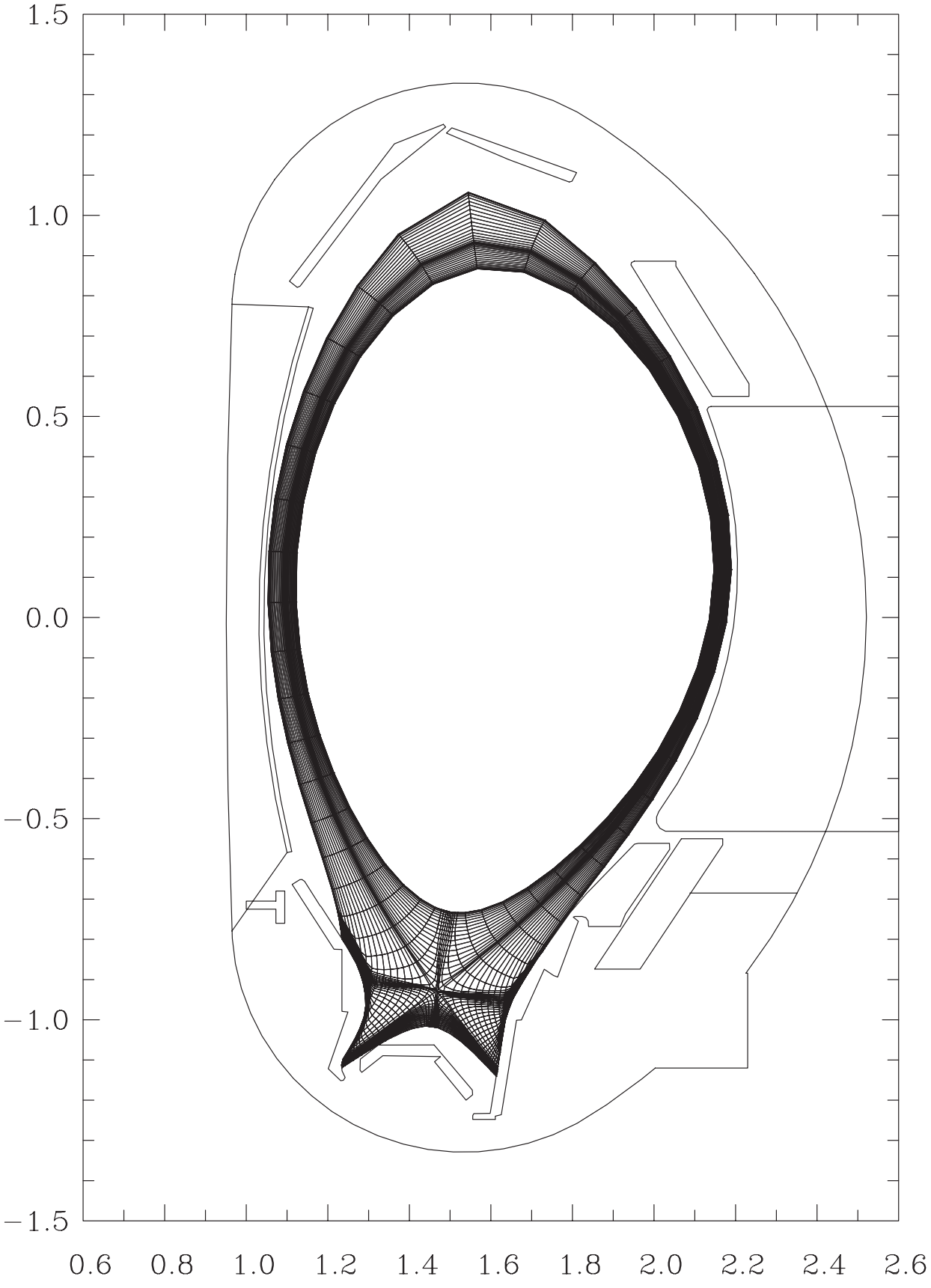
1: left wall target
2: right wall target
3: connection between both sides of core

- **north/south boundaries** (y-directed):

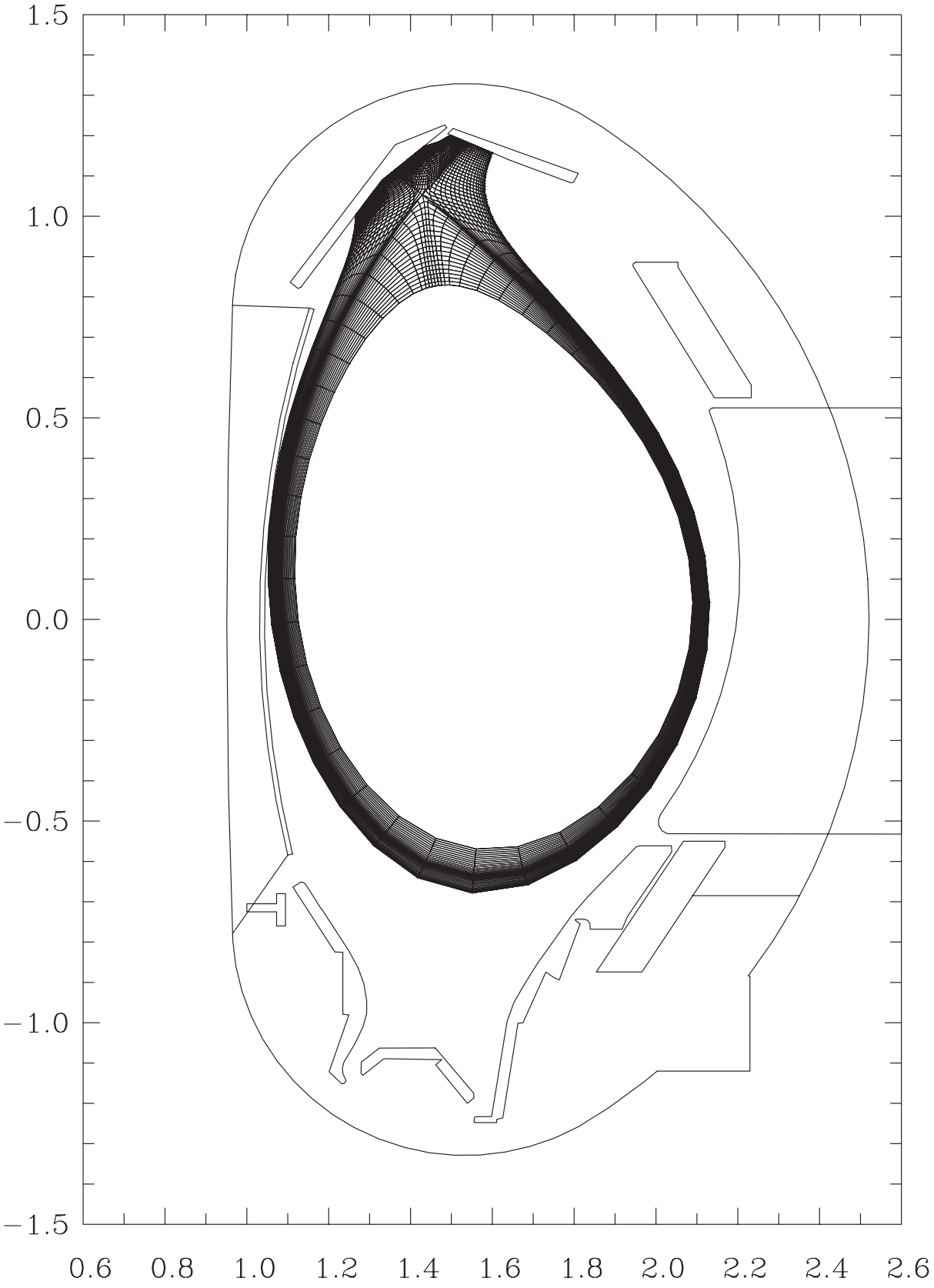


1: core boundary
2: separatrix
3: SOL to main chamber wall boundary

In the case of **single-null geometry** there is one X-point, and the SOL plasma is limited by the divertor targets. The separatrix here is defined as the boundary magnetic flux surface between closed- and open-field-line regions, and the targets are defined as the portions of the divertor plates on which the open magnetic field lines strike.
The first possibility (which is the most common one) consists in the **lower single-null** geometry, in which the X-point lies below the core plasma. Here is how the physical domain appears.

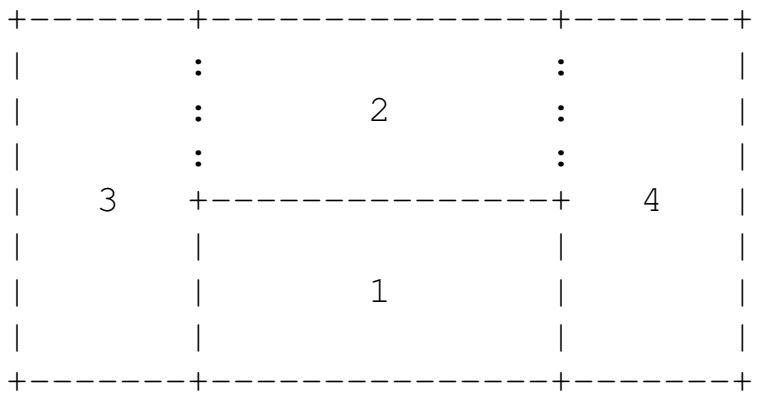


A second possibility, instead, consists in the **upper single-null** geometry, in which the X-point lies above the core plasma. Here is how the physical domain appears.



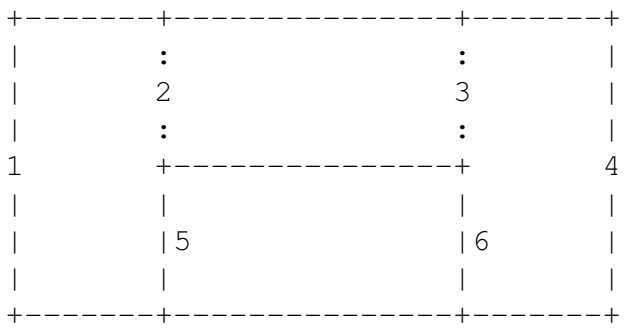
In this case we have `NREG = 4` so that all the variables in B2.5 which need to be specified for each region will form strings of `NREG + 1 = 4 + 1` values, in the form `(0:NREG)`.
In the computational domain we have the following region indexing and definitions:

1. Core
2. SOL
3. Inner divertor region
4. Outer divertor region



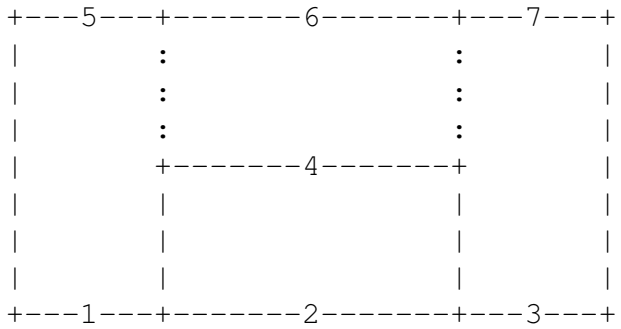
The resulting boundaries indexing and definitions, then, is:

- **west/east boundaries** (x-directed):



1: inner divertor target
2: entrance to inner divertor
3: entrance to outer divertor
4: outer divertor target
5: connection between both sides of the core
6: connection between both sides of the divertor

- **north/south boundaries** (y-directed):



1: inner divertor private flux wall boundary
2: core boundary
3: outer divertor private flux wall boundary
4: separatrix
5: inner divertor to main chamber wall boundary
6: SOL to main chamber wall boundary
7: outer divertor to main chamber wall boundary

7 - Definition of the physical variables

Here we specify the names of the various **physical variables** as used as read by various code scripts and routines of B2.5 and EIRENE.

The variables describing the **geometry** are:

- **nx**, **ny**: integers.
  These specifies the **number of cells** (interior cells, precisely) along the poloidal and radial coordinates, respectively, with cells indexed by `(-1:nx,-1:ny)`.
  Usually, we have `(nx,ny)=(48,18)` or `(nx,ny)=(96,36)`.
  In this way, considering the also the "unused" cells and external borders of the domain, consisting in one additional line of cells per border, the total number of cells in the poloidal coordinate is `nx+2` and in the radial coordinate is `ny+2`.

- **bb**: real, indexed by `(-1:nx,-1:ny,0:3)`; an array of four numbers for each cell.
  For the cell `(ix,iy)`, the four numbers in the array `bb(ix,iy,0:3)` specify the **magnetic field** at the center of the cell. In particular:
  - the number `bx(,)=bb(,,0)` specifies the x-component of the magnetic field.
  - the number `by(,)=bb(,,1)` specifies the y-component of the magnetic field.
  - the number `bz(,)=bb(,,2)` specifies the z-component of the magnetic field.
  - the number `bb(,,3)` specifies the absolute magnetic field intensity.

  In this way, the following identity will be held:
  - `bb(,,3)=sqrt(bb(,,0)**2+bb(,,1)**2+bb(,,2)**2)`

- **crx**, **cry**: reals, indexed by `(-1:nx,-1:ny,0:3)`; an array of eight numbers for each cell.
  For the cell `(ix,iy)`, the four numbers in the array `crx(ix,iy,0:3)` and the four number in the array `cry(ix,iy,0:3)` specify the **horizontal physical coordinates** and the **vertical physical coordinates**, respectively, of the cell in the physical domain (the poloidal cross section of the plasma).
  In particular:
  - the numbers `(,,0)` give the position of the lower left corner of each cell.
  - the numbers `(,,1)` give the position of the lower right corner of each cell.
  - the numbers `(,,2)` give the position of the upper left corner of each cell.
  - the numbers `(,,3)` give the position of the upper right corner of each cell.

  In this way, the following identities will be held:
  - `crx(ix,iy,0)=crx(ix-1,iy,1)=crx(ix,iy-1,2)=crx(ix-1,iy-1,3)`
  - `cry(ix,iy,0)=cry(ix-1,iy,1)=cry(ix,iy-1,2)=cry(ix-1,iy-1,3)`

  Since the cells are quadrilaterals, these variables completely specify the computational grid for B2.5.

- **gs**: real, indexed by `(-1:nx,-1:ny,0:2)`; an array of three numbers for each cell.
  For the cell `(ix,iy)`, the three numbers in the array `gs(ix,iy,0:2)` specify the **contact areas** of the cell with its neighbors. In particular:
  - the number `gsx(,)=gs(,,0)` specifies the contact area of the face between a cell in `(0:nx,-1:ny)` and its left neighbor.
  - the number `gsy(,)=gs(,,1)` specifies the contact area of the face between a cell in `(-1:nx,0:ny)` and its bottom neighbor.
  - the number `gsz(,)=gs(,,2)` specifies the area of a cell in `(-1:nx,-1:ny)` normal to the third coordinate.

- **pbs**: real, indexed by `(-1:nx,-1:ny,0:1)`; an array of two numbers for each cell.
  For the cell `(ix,iy)`, the two numbers in the array `pbs(ix,iy,0:1)` specify the **effective contact areas** (that is the parallel contact areas) of the cell with its neighbors. In particular:
  - the number `pbsx(,)=pbs(,,0)` specifies the parallel contact area of the face between a cell in `(0:nx,-1:ny)` and its left neighbor, that is `(bx/bb)*gsx(,)`.
  - the number `pbsy(,)=pbs(,,1)` specifies the parallel contact area of the face between a cell in `(-1:nx,0:ny)` and its bottom neighbor, that is `(by/bb)*gsy(,)`.

- **qz**: real, indexed by `(-1:nx,-1:ny,0:1)`; an array of two numbers for each cell.
  For the cell `(ix,iy)`, the two numbers in the array `qz(ix,iy,0:1)` specify the **relative orientation** between the two coordinate-oriented sides of the cell, in terms of the complementary angle `t` between the x direction and the y direction of the cell (which is in the range from `-pi/2` to `+pi/2`). In particular:
  - the number `qz(,,0)` specifies `sin(t)`.
  - the number `qz(,,1)` specifies `cos(t)`.

- **hx**, **hy**, **hz**: real, indexed by `(-1:nx,-1:ny)`; an array of three numbers for each cell.
  For the cell `(ix,iy)`, the three numbers `hx(ix,iy)`, `hy(ix,iy)`, `hz(ix,iy)` specify the **metric coefficients** of the cell.

- **vol**: real, indexed by `(-1:nx,-1:ny)`; one number for each cell.
  For the cell `(ix,iy)`, the number `vol(ix,iy)` specifies the **volume** of the cell.

- **dspar**: real, indexed by `(-1:nx,-1:ny)`; one number for each cell.
  For the cell `(ix,iy)`, the number `dspar(ix,iy)` specifies the **parallel distance** (i.e. the parallel connection length) of such cell calculated from the inner divertor target.

- **dspol**: real, indexed by `(-1:nx,-1:ny)`; one number for each cell.
  For the cell `(ix,iy)`, the number `dspol(ix,iy)` specifies the **poloidal distance** (i.e. the poloidal connection length) of such cell calculated from the inner divertor target.

The variables describing the **basic physical parameters** are:

- **/b2cmpa/**: block, common for all the cells, containing the real numbers:
  - `me` = **electron mass**, in kg, that is $9,109 \cdot 10^{-31}$.
  - `mp` = **proton mass**, in kg, that is $1,673 \cdot 10^{-27}$.
  - `ev` = **electronvolt** in J, that is $1,602 \cdot 10^{-19}$.
  - `qe` = **elementary charge**, in C, that is $1,602 \cdot 10^{-19}$.
  - `eps0` = **vacuum dielectric permittivity**, according to the SI, that is $8,854 \cdot 10^{-12}$.
  - `mu0` = **vacuum magnetic permeability**, according to the SI, that is $1,257 \cdot 10^{-6}$.

The variables describing the **plasma state** are:

- **ns**: integer, indexed by (0:ns-1).
  This specifies the **number of atomic species** considered in the calculations, with species indexed by (0:ns-1), where am(is) is the atomic mass of the atomic species is. In the simplest case with only one atomic species present (e.g. hydrogen H or deuterium D) together with its single-charged ion (e.g. H^+ or D^+), then is=0 refers to the neutral atom and is=1 refers to the ion.

- **ne**: real, indexed by (-1:nx,-1:ny); one number for each cell.
  For the cell (ix,iy), the number ne(ix,iy) specifies the **electron density** on the cell. With the assumption of quasi-neutrality and single-charged ions, it will be held that ne(,)=sum(na(,,ions)), with the sum extended on all the atom species representing ions.

- **na**: real, indexed by (-1:nx,-1:ny,0:ns-1); an array of ns numbers for each cell.
  For the cell (ix,iy), the ns numbers in the array na(ix,iy,0:ns-1) specify the B2.5 **atomic densities** of each species on the cell. In the simplest case with only one atomic species present (e.g. hydrogen/deuterium H/D) together with its single-charged ion (e.g. H^+/D^+), then na(,,0) is the B2.5 **neutral density**, while na(,,1) is the B2.5 **ion density**.

- **ni**: real, indexed by (-1:nx,-1:ny); one number for each cell.
  For the cell (ix,iy), the number ni(ix,iy) specifies the (total) **ion density** on the cell, that is ni(,)=sum(na(,,ions)).

- **dab2**: real, indexed by (-1:nx,-1:ny); one number for each cell.
  For the cell (ix,iy), the number dab2(ix,iy) specifies the EIRENE **atomic neutral density** on the cell.

- **dmb2**: real, indexed by (-1:nx,-1:ny); one number for each cell.
  For the cell (ix,iy), the number dmb2(ix,iy) specifies the EIRENE **molecular neutral density** on the cell.

- **te**: real, indexed by (-1:nx,-1:ny); one number for each cell.
  For the cell (ix,iy), the number te(ix,iy) specifies the **electron temperature** on the cell.

- **ti**: real, indexed by (-1:nx,-1:ny); one number for each cell.
  For the cell (ix,iy), the number ti(ix,iy) specifies the (total) **ion temperature** on the cell, supposed equal for all the ion species.

- **tab2**: real, indexed by (-1:nx,-1:ny); one number for each cell.
  For the cell (ix,iy), the number tab2(ix,iy) specifies the EIRENE **atomic neutral temperature** on the cell.

- **tmb2**: real, indexed by (-1:nx,-1:ny); one number for each cell.
  For the cell (ix,iy), the number tmb2(ix,iy) specifies the EIRENE **molecular neutral temperature** on the cell.

- **pr**: real, indexed by (-1:nx,-1:ny); one number for each cell.
  For the cell (ix,iy), the number pr(ix,iy) specifies the plasma **total static pressure** on the cell.

- **p***: real, indexed by (-1:nx,-1:ny); one number for each cell.
  For the cell (ix,iy), the number p*(ix,iy) specifies the plasma **dynamic pressure** on the cell.

- **po**: real, indexed by `(-1:nx,-1:ny)`; one number for each cell.
  For the cell `(ix,iy)`, the number `po(ix,iy)` specifies the **electric potential** on the cell.

- **ua**: real, indexed by `(-1:nx,-1:ny,0:ns-1)`; an array of `ns` numbers for each cell.
  For the cell `(ix,iy)`, the `ns` numbers in the array `ua(ix,iy,0:ns-1)` specify the average **parallel velocities** of each atomic species on the cell, with sign indicating whether it is flowing in the same (+) or opposite (−) direction as the local poloidal magnetic field.

- **fna**: real, indexed by `(-1:nx,-1:ny,0:1,0:ns-1)`; an array of `2*ns` numbers for each cell.
  For the cell `(ix,iy)`, the `2*ns` numbers in the array `fna(ix,iy,0:1,0:ns-1)` specify the **atom particle flux** for the various species through the faces of the cell. In particular:
  - The number `fnax(,,is)=fna(,,0,is)` specifies the poloidal atom flux of the species `is` through the face between a cell in `(0:nx,-1:ny)` and its left neighbor, with sign indicating whether it is directed in the direction in increasing (+) or decreasing (−) poloidal (`ix`) index.
  - The number `fnay(,,is)=fna(,,1,is)` specifies the radial atom flux of the species `is` through the face between a cell in `(-1:nx,0:ny)` and its bottom neighbor, with sign indicating whether it is directed in the direction in increasing (+) or decreasing (−) radial (`iy`) index.

- **fne**: real, indexed by `(-1:nx,-1:ny,0:1)`; an array of two numbers for each cell.
  For the cell `(ix,iy)`, the two numbers in the array `fne(ix,iy,0:1)` specify the **electron particle flux** through the faces of the cell. In particular:
  - The number `fnex(,)=fne(,,0)` specifies the poloidal electron flux through the face between a cell in `(0:nx,-1:ny)` and its left neighbor, with sign indicating whether it is directed in the direction in increasing (+) or decreasing (−) poloidal (`ix`) index.
  - The number `fney(,)=fne(,,1)` specifies the radial electron flux through the face between a cell in `(-1:nx,0:ny)` and its bottom neighbor, with sign indicating whether it is directed in the direction in increasing (+) or decreasing (−) radial (`iy`) index.

- **fni**: real, indexed by `(-1:nx,-1:ny,0:1)`; an array of two numbers for each cell.
  For the cell `(ix,iy)`, the two numbers in the array `fni(ix,iy,0:1)` specify the (total) **ion particle flux** through the faces of the cell. In particular:
  - The number `fnix(,)=fni(,,0)` specifies the poloidal ion flux through the face between a cell in `(0:nx,-1:ny)` and its left neighbor, with sign indicating whether it is directed in the direction in increasing (+) or decreasing (−) poloidal (`ix`) index.
  - The number `fniy(,)=fni(,,1)` specifies the radial ion flux through the face between a cell in `(-1:nx,0:ny)` and its bottom neighbor, with sign indicating whether it is directed in the direction in increasing (+) or decreasing (−) radial (`iy`) index.

- **fhe**: real, indexed by (-1:nx,-1:ny,0:1); an array of two numbers for each cell.
  For the cell (ix,iy), the two numbers in the array fhe(ix,iy,0:1) specify the **electron heat flux** through the faces of the cell. In particular:
  - The number fhex(,)=fhe(,,0) specifies the poloidal electron heat flux through the face between a cell in (0:nx,-1:ny) and its left neighbor, with sign indicating whether it is directed in the direction in increasing (+) or decreasing (-) poloidal (ix) index.
  - The number fhey(,)=fhe(,,1) specifies the radial electron heat flux through the face between a cell in (-1:nx,0:ny) and its bottom neighbor, with sign indicating whether it is directed in the direction in increasing (+) or decreasing (-) radial (iy) index.

- **fhi**: real, indexed by (-1:nx,-1:ny,0:1); an array of two numbers for each cell.
  For the cell (ix,iy), the two numbers in the array fhi(ix,iy,0:1) specify the (total) **ion heat flux** through the faces of the cell. In particular:
  - The number fhix(,)=fhi(,,0) specifies the poloidal ion heat flux through the face between a cell in (0:nx,-1:ny) and its left neighbor, with sign indicating whether it is directed in the direction in increasing (+) or decreasing (-) poloidal (ix) index.
  - The number fhiy(,)=fhi(,,1) specifies the radial ion heat flux through the face between a cell in (-1:nx,0:ny) and its bottom neighbor, with sign indicating whether it is directed in the direction in increasing (+) or decreasing (-) radial (iy) index.
- **fht**: real, indexed by (-1:nx,-1:ny,0:1); an array of two numbers for each cell.
  For the cell (ix,iy), the two numbers in the array fht(ix,iy,0:1) specify the **total energy flux** (electron heat + ion heat + parallel kinetic energy + electrostatic energy + potential energy) through the faces of the cell. In particular:
  - The number fhtx(,)=fht(,,0) specifies the poloidal total energy flux through the face between a cell in (0:nx,-1:ny) and its left neighbor, with sign indicating whether it is directed in the direction in increasing (+) or decreasing (-) poloidal (ix) index.
  - The number fhty(,)=fht(,,1) specifies the radial total energy flux through the face between a cell in (-1:nx,0:ny) and its bottom neighbor, with sign indicating whether it is directed in the direction in increasing (+) or decreasing (-) radial (iy) index.

- **fch**: real, indexed by (-1:nx,-1:ny,0:1); an array of two numbers for each cell.
  For the cell (ix,iy), the two numbers in the array fch(ix,iy,0:1) specify the **electric current** through the faces of the cell. In particular:
  - The number fchx(,)=fch(,,0) specifies the poloidal electric current through the face between a cell in (0:nx,-1:ny) and its left neighbor, with sign indicating whether it is directed in the direction in increasing (+) or decreasing (-) poloidal (ix) index.
  - The number fchy(,)=fch(,,1) specifies the radial electric current through the face between a cell in (-1:nx,0:ny) and its bottom neighbor, with sign indicating whether it is directed in the direction in increasing (+) or decreasing (-) radial (iy) index.

The variables describing the **residuals** for the conservation equations are:

- **resco**: real, indexed by `(-1:nx,-1:ny,0:ns-1)`; an array of `ns` numbers for each cell. For the cell `(ix,iy)`, the `ns` numbers in the array `resco(ix,iy,0:ns-1)` specify the **residuals of the continuity equation** for the various species on the cell.
- **resmo**: real, indexed by `(-1:nx,-1:ny,0:ns-1)`; an array of `ns` numbers for each cell. For the cell `(ix,iy)`, the `ns` numbers in the array `resmo(ix,iy,0:ns-1)` specify the **residuals of the parallel momentum equation** for the various species on the cell.
- **resmt**: real, indexed by `(-1:nx,-1:ny)`; one number for each cell. For the cell `(ix,iy)`, the number `resmt(ix,iy)` specifies the **residual of the total parallel momentum equation** on the cell.
- **reshe**: real, indexed by `(-1:nx,-1:ny)`; one number for each cell. For the cell `(ix,iy)`, the number `reshe(ix,iy)` specifies the **residual of the electron energy equation** on the cell.
- **reshi**: real, indexed by `(-1:nx,-1:ny)`; one number for each cell. For the cell `(ix,iy)`, the number `reshi(ix,iy)` specifies the **residual of the ion energy equation** on the cell.
- **resht**: real, indexed by `(-1:nx,-1:ny)`; one number for each cell. For the cell `(ix,iy)`, the number `resht(ix,iy)` specifies the **residual of the total energy equation** on the cell.
- **respo**: real, indexed by `(-1:nx,-1:ny)`; one number for each cell. For the cell `(ix,iy)`, the number `respo(ix,iy)` specifies the **residual of the electric potential equation** on the cell.

The variables describing the **rate coefficients** for atomic reactions are:

- **/b2cmrc/**: block, common for all the cells, containing:
  the integer numbers:
  - rtnt = number of intervals in the **discretization of the temperature range**.
  - rtnn = number of intervals in the **discretization of the density range**.
  - rtns = total number of atomic species

  the arrays of real numbers:
  - rtt, indexed by (0:rtnt), specifying the sequence of values of the temperature, in eV, for which the rate coefficients are evaluated.
  - rtn, indexed by (0:rtnn), specifying the sequence of values of the density, in m^{-3}, for which the rate coefficients are evaluated.
  - rtlt, indexed by (0:rtnt), specifying the sequence of the logarithms of the values of the temperature for which the rate coefficients are evaluated, that is rtlt()=log(rtt()).
  - rtln, indexed by (0:rtnn), specifying the sequence of the logarithms of the values of the density for which the rate coefficients are evaluated, that is rtln()=log(rtn()).

  the arrays of real numbers specifying the rate coefficients:
  - rtrsa, indexed by (0:rtnt,0:rtnn,0:rtns-1), specifying the **ionization rate coefficients** in m^3/s in any intervals of (electron) temperature, in any intervals of (electron) density, and for the processes is ---> is+1. In the simplest case with only one atomic species present (e.g. hydrogen H) together with its single-charged ion (e.g. H^+), then this is defined only for the species is=0 (H), and describes the process H ---> H^+.
  - rtrra, indexed by (0:rtnt,0:rtnn,0:rtns-1), specifying the **recombination rate coefficients** in m^3/s in any intervals of (electron) temperature, in any intervals of (electron) density, and for the processes is ---> is-1. In the simplest case with only one atomic species present (e.g. hydrogen H) together with its single-charged ion (e.g. H^+), then this is defined only for the species is=1 (H^+), and describes the process H^+ ---> H.
  - rtrcx, indexed by (0:rtnt,0:rtnn,0:rtns-1), specifying the **charge-exchange rate coefficients** in m^3/s in any intervals of (neutral) temperature, in any intervals of (electron) density, and for the processes is ---> is-1. In the simplest case with only one atomic species present (e.g. hydrogen H) together with its single-charged ion (e.g. H^+), then this is defined only for the species is=1 (H^+), and describes the process H^++H ---> H+H^+.

  the arrays of real numbers specifying the logarithms of the rate coefficients:
  - rtlsa, indexed by (0:rtnt,0:rtnn,0:rtns-1), specifying log(rtrsa).
  - rtlra, indexed by (0:rtnt,0:rtnn,0:rtns-1), specifying log(rtrra).
  - rtlcx, indexed by (0:rtnt,0:rtnn,0:rtns-1), specifying log(rtrcx).

- **rsa**: real, indexed by (-1:nx,-1:ny,0:ns-1); an array of ns numbers for each cell.
  For the cell (ix,iy), each number rsa(ix,iy,is) specifies the **rate coefficient for ionization** from the atomic species is to the atomic species is+1 on the cell, in m^3/s, basing on its density and temperature.

- **rra**: real, indexed by (-1:nx,-1:ny,0:ns-1); an array of ns numbers for each cell.
  For the cell (ix,iy), each number rra(ix,iy,is) specifies the **rate coefficient for recombination** from the atomic species is to the atomic species is-1 on the cell, in m^3/s, basing on its density and temperature.

- **rcx**: real, indexed by (-1:nx,-1:ny,0:ns-1); an array of ns numbers for each cell.
  For the cell (ix,iy), each number rqa(ix,iy,is) specifies the **rate coefficient for charge-exchange reaction** of the atomic species is by collisions with neutrals on the cell, in m^3/s, basing on its density and temperature.

The variables describing the **boundary conditions** are:

- **/b2cmbp/**: common block, specifying physics parameters for boundary conditions, with boundary divided up into regions: so, for each region a set of coefficients defines the local conditions.
  It contains the integer numbers:
  - nsdecl: upper bound for the number of species ns.
  - cbnrmx: upper bound for the number of boundary regions nreg.
  - cbirso: index of the first region on the south boundary.
  - cbirno: index of the first region on the north boundary.
  - cbirwe: index of the first region on the west boundary.
  - cbirea: index of the first region on the east boundary.
  - cbnrso: number of regions on the south boundary.
  - cbnrno: number of regions on the north boundary.
  - cbnrwe: number of regions on the west boundary.
  - cbnrea: number of regions on the east boundary.

  and the arrays of real numbers:
  - cbrbrk, indexed by (0:cbnrmx-1): boundaries between the boundary regions.
  - cbsna, indexed by (0:7,0:nsdecl-1,0:cbnrmx-1): the eight coefficients in each array cbsna(0:7,is,ireg) specify standard form boundary conditions for the continuity equation for species is on the ireg boundary region.
  - cbsmo, indexed by (0:7,0:nsdecl-1,0:cbnrmx-1): the eight coefficients in each array cbsmo(0:7,is,ireg) specify standard form boundary conditions for the parallel momentum equation for species is on the ireg boundary region.
  - cbshe, indexed by (0:7,0:cbnrmx-1): the eight coefficients in each array cbshe(0:7,ireg) specify standard form boundary conditions for the electron energy equation on the ireg boundary region.
  - cbshi, indexed by (0:7,0:nsdecl-1,0:cbnrmx-1): the eight coefficients in each array cbshi(0:7,is,ireg) specify standard form boundary conditions for the ion energy equation for species is on the ireg boundary region.
  - cbsch, indexed by (0:7,0:cbnrmx-1): the eight coefficients in each array cbsch(0:7,ireg) specify standard form boundary conditions for the electric potential equation on the ireg boundary region.

The variables describing the **source coefficients** are:

- **sna**: real, indexed by `(-1:nx,-1:ny,0:1,0:ns-1)`: an array of `2*ns` numbers for each cell.
  For the cell `(ix,iy)`, the `2*ns` numbers in the array `sna(ix,iy,0:1,0:ns-1)` specify a linear approximation to the assumed **particle source** for each species, integrated over the cell volume, so that the particle source of species `is` can be expressed as
  ```
  sna(ix,iy,0,is)+
  sna(ix,iy,1,is)*na(ix,iy,is).
  ```

- **sne**: real, indexed by `(-1:nx,-1:ny,0:1)`: an array of two numbers for each cell.
  For the cell `(ix,iy)`, the two numbers in the array `sne(ix,iy,0:1)` specify a linear approximation to the assumed **electron source**, integrated over the cell volume, so that the electron source can be expressed as
  ```
  sne(ix,iy,0)+
  sne(ix,iy,1)*ne(ix,iy).
  ```

- **smo**: real, indexed by `(-1:nx,-1:ny,0:3,0:ns-1)`: an array of `4*ns` numbers for each cell.
  For the cell `(ix,iy)`, the `4*ns` numbers in the array `smo(ix,iy,0:3,0:ns-1)` specify a linear approximation to the assumed **parallel momentum source** for each species, integrated over the cell volume, so that the momentum source for the species `is` can be expressed as
  ```
  smo(ix,iy,0,is)+
  smo(ix,iy,1,is)*ua(ix,iy,is)+
  smo(ix,iy,2,is)*am(is)*mp*na(ix,iy,is)+
  smo(ix,iy,3,is)*am(is)*mp*na(ix,iy,is)*ua(ix,iy,is)
  ```

- **she**: real, indexed by `(-1:nx,-1:ny,0:3)`: an array of four numbers for each cell.
  For the cell `(ix,iy)`, the four numbers in the array `she(ix,iy,0:3)` specify a linear approximation to the assumed **electron heat source**, integrated over the cell volume, so that the electron heat source can be expressed as
  ```
  she(ix,iy,0)+
  she(ix,iy,1)*te(ix,iy)+
  she(ix,iy,2)*ne(ix,iy)+
  she(ix,iy,3)*ne(ix,iy)*te(ix,iy)
  ```

- **shi**: real, indexed by `(-1:nx,-1:ny,0:3)`: an array of four numbers for each cell.
  For the cell `(ix,iy)`, the four numbers in the array `shi(ix,iy,0:3)` specify a linear approximation to the assumed (total) **ion heat source**, integrated over the cell volume, so that the total ion heat source can be expresses as
  ```
  shi(ix,iy,0)+
  shi(ix,iy,1)*ti(ix,iy)+
  shi(ix,iy,2)*ni(ix,iy)+
  shi(ix,iy,3)*ni(ix,iy)*ie(ix,iy)
  ```

- **sch**: real, indexed by `(-1:nx,-1:ny,0:3)`: an array of four numbers for each cell.
  For the cell `(ix,iy)`, the four numbers in the array `sch(ix,iy,0:3)` specify a linear approximation to the assumed **electric charge source**, integrated over the cell volume, so that the electric charge source can be expressed as
  ```
  sch(ix,iy,0)+
  sch(ix,iy,1)*po(ix,iy)+
  sch(ix,iy,2)*ne(ix,iy)+
  sch(ix,iy,3)*ne(ix,iy)*po(ix,iy)
  ```

The variables describing the **transport coefficients** are:

- **dna**: real, indexed by `(-1:nx,-1:ny,0:1,0:ns-1)`: an array of `2*ns` numbers for each cell. For the cell `(ix,iy)`, the `2*ns` numbers in the array `dna(ix,iy,0:1,0:ns-1)` specify the parallel and perpendicular components of the **density-driven diffusivity** for the various species on the cell, which drive a term `-dna*grad(na)` in the particle flux density. In particular:
  - The number `dnax(,,is)=dna(,,0,is)` specifies the x-component of the density-driven diffusivity, that is the parallel diffusivity, of the species `is` on the cell.
  - The number `dnay(,,is)=dna(,,1,is)` specifies the y-component of the density-driven diffusivity, that is the perpendicular diffusivity, of the species `is` on the cell.

- **dpa**: real, indexed by `(-1:nx,-1:ny,0:1,0:ns-1)`: an array of `2*ns` numbers for each cell. For the cell `(ix,iy)`, the `2*ns` numbers in the array `dpa(ix,iy,0:1,0:ns-1)` specify the parallel and perpendicular components of the **pressure-driven diffusivity** for the various species on the cell, which drive a term `-dpa*grad(pa)` in the particle flux density. In particular:
  - The number `dpax(,,is)=dpa(,,0,is)` specifies the x-component of the pressure-driven diffusivity, that is the parallel diffusivity, of the species `is` on the cell.
  - The number `dpay(,,is)=dpa(,,1,is)` specifies the y-component of the pressure-driven diffusivity, that is the perpendicular diffusivity, of the species `is` on the cell.

- **vla**: real, indexed by `(-1:nx,-1:ny,0:1,0:ns-1)`: an array of `2*ns` numbers for each cell. For the cell `(ix,iy)`, the `2*ns` numbers in the array `vla(ix,iy,0:1,0:ns-1)` specify the parallel and perpendicular **anomalous velocities** for the various species on the cell. In particular:
  - The number `vlax(,,is)=vla(,,0,is)` specifies the x-component of the anomalous velocity, that is the parallel anomalous velocity, of the species `is` on the cell.
  - The number `vlay(,,is)=vla(,,1,is)` specifies the y-component of the anomalous velocity, that is the perpendicular anomalous velocity, of the species `is` on the cell.

- **vsa**: real, indexed by `(-1:nx,-1:ny,0:1,0:ns-1)`: an array of `2*ns` numbers for each cell. For the cell `(ix,iy)`, the `2*ns` numbers in the array `vsa(ix,iy,0:1,0:ns-1)` specify the parallel and perpendicular **anomalous viscosities** for the various species on the cell. In particular:
  - The number `vsax(,,is)=vsa(,,0,is)` specifies the x-component of the anomalous viscosity, that is the parallel viscosity, of the species `is` on the cell.
  - The number `vsay(,,is)=vsa(,,1,is)` specifies the y-component of the anomalous viscosity, that is the perpendicular viscosity, of the species `is` on the cell.

- **hce**: real, indexed by `(-1:nx,-1:ny,0:1)`: an array of two numbers for each cell. For the cell `(ix,iy)`, the two numbers in the array `hce(ix,iy,0:1)` specify the parallel and perpendicular components of the **electron thermal diffusivity** on the cell. In particular:
  - The number `hcex(,,)=hce(,,0)` specifies the x-component of the electron thermal diffusivity, that is the parallel electron thermal diffusivity, on the cell.
  - The number `hcey(,,)=hce(,,1)` specifies the y-component of the electron thermal diffusivity, that is the perpendicular electron thermal diffusivity, on the cell.

- **hci**: real, indexed by `(-1:nx,-1:ny,0:1)`: an array of two numbers for each cell. For the cell `(ix,iy)`, the two numbers in the array `hci(ix,iy,0:1)` specify the parallel and perpendicular components of the (total) **ion thermal diffusivity** on the cell. In particular:
  - The number `hcix(,,)=hci(,,0)` specifies the x-component of the ion thermal diffusivity, that is the parallel ion thermal diffusivity, on the cell.
  - The number `hciy(,,)=hci(,,1)` specifies the y-component of the ion thermal diffusivity, that is the perpendicular ion thermal diffusivity, on the cell.

- **sig**: real, indexed by `(-1:nx,-1:ny,0:1)`: an array of two numbers for each cell.
  For the cell `(ix,iy)`, the two numbers in the array `sig(ix,iy,0:1)` specify the parallel and perpendicular components of the **electrical conductivity** on the cell. In particular:
  - The number `sigx(,,)=sig(,,0)` specifies the x-component of the electrical conductivity, that is the parallel electrical conductivity, on the cell.
  - The number `sigy(,,)=sig(,,1)` specifies the y-component of the electrical conductivity, that is the perpendicular electrical conductivity, on the cell.

- **alf**: real, indexed by `(-1:nx,-1:ny,0:1)`: an array of two numbers for each cell.
  For the cell `(ix,iy)`, the two numbers in the array `alf(ix,iy,0:1)` specify the parallel and perpendicular components of the **thermo-electric coefficient** on the cell. In particular:
  - The number `alfx(,,)=alf(,,0)` specifies the x-component of the thermo-electric coefficient, that is the parallel thermo-electric coefficient, on the cell.
  - The number `alfy(,,)=alf(,,1)` specifies the y-component of the thermo-electric coefficient, that is the perpendicular thermo-electric coefficient, on the cell.

- **dna0**: real, indexed by `(-1:nx,-1:ny,0:ns-1)`: an array of `ns` numbers for each cell.
  For the cell `(ix,iy)`, the `ns` numbers in the array `dna0(ix,iy,0:ns-1)` specify the **anomalous density-driven diffusivity** for the various species on the cell.

- **dpa0**: real, indexed by `(-1:nx,-1:ny,0:ns-1)`: an array of `ns` numbers for each cell.
  For the cell `(ix,iy)`, the `ns` numbers in the array `dpa0(ix,iy,0:ns-1)` specify the **anomalous pressure-driven diffusivity** for the various species on the cell.

- **vla0**: real, indexed by `(-1:nx,-1:ny,0:1,0:ns-1)`: an array of `2*ns` numbers for each cell.
  For the cell `(ix,iy)`, the `2*ns` numbers in the arrays `vla0x(ix,iy,0:ns-1)` and `vla0y(ix,iy,0:ns-1)` specify the parallel and perpendicular components of the **anomalous velocities** for the various species on the cell.

- **vsa0**: real, indexed by `(-1:nx,-1:ny,0:ns-1)`: an array of `ns` numbers for each cell.
  For the cell `(ix,iy)`, the `ns` numbers in the array `vsa0(ix,iy,0:ns-1)` specify the **anomalous viscosities** for the various species on the cell.

- **hce0**: real, indexed by `(-1:nx,-1:ny)`: one number for each cell.
  For the cell `(ix,iy)`, the number `hce0(ix,iy)` specifies the **anomalous electron thermal diffusivity** on the cell.

- **hci0**: real, indexed by `(-1:nx,-1:ny)`: one number for each cell.
  For the cell `(ix,iy)`, the number `hci0(ix,iy)` specifies the (total) **anomalous ion thermal diffusivity** on the cell.

- **sig0**: real, indexed by `(-1:nx,-1:ny)`: one number for each cell.
  For the cell `(ix,iy)`, the number `sig0(ix,iy)` specifies the **anomalous electrical conductivity** on the cell.

- **alf0**: real, indexed by `(-1:nx,-1:ny)`: one number for each cell.
  For the cell `(ix,iy)`, the number `alf0(ix,iy)` specifies the **anomalous thermo-electric coefficient** on the cell.

Finally, some of the parameters describing the **numerical computations** are:

- **ntim**: integer, specifies the number of timesteps to be performed in the computation.
- **itim**: integer, specifies the timestep counter.
- **dtim**: real, specifies the timestep for the current computation call.

8 - Constructing a SOLPS-ITER grid

The first step to perform for setting up a SOLPS-ITER run is constructing the discretized **grid** to be used for the numerical calculations. This consists in:

- a mesh formed by quadrilaterals, used for the B2.5 calculations, which extends over an annulus of core plasma and over the SOL (including private flux region and divertor region).
- a mesh formed by triangles, used for the EIRENE calculations, which extends over the entire B2.5 mesh and over the entire remaining regions inside the vacuum vessel.

For doing it is made use of the **DivGeo** program, a graphical interface which is used to import the material structure of the considered device, the magnetic equilibrium configuration of the particular discharge to be modelled, and to set up the specifications for constructing both the B2.5 and EIRENE meshes. Then, these are finally prepared by the **Carre** and **Triang** programs, respectively.

So, the first step to perform is to run DivGeo, with which it is possible to set up a graphical environment for preparing the grid. In this step we can import structure and equilibrium data from an existing device, and the goal is to prepare the input files for the successive programs. DivGeo allows indeed to graphically visualize the model geometry, to modify it, and to specify the necessary geometry-related data.
Moreover, with DivGeo it is also possible to specify:

- the **plasma composition** to be considered in the model, that is the particular ionic species to be modelled (D, D+T, etc., and with or without impurities).
- the **surface properties** of the wall materials, that is the kind of material, its reflection/absorption properties, etc.

Starting constructing a grid from zero requires:

- A **template file**, with `.ogr` extension, containing the **wall shape** made from a CAD drawing or other sources for the considered device, i.e. a list of (R,z) points (in mm) in device coordinates which describe the layout of all the material components of the device, forming a series of closed polygons.
- An **equilibrium file**, with `.eq` extension, specifying the **equilibrium configuration** of the magnetic flux surfaces for the considered experiment. For example, for the case of ASDEX Upgrade, while we are inside the TOK cluster, it is possible by create the desired equilibrium file, related to a particular time of a particular discharge, by entering

➢ `shotfile2dg {DISCHARGE_NUMBER} {TIME}`

First of all we move to the `runs/` directory, and here we create a test case directory by entering

➢ `mkdir {TEST_CASE}`

and then, inside it, we create a `baserun/` directory, by entering

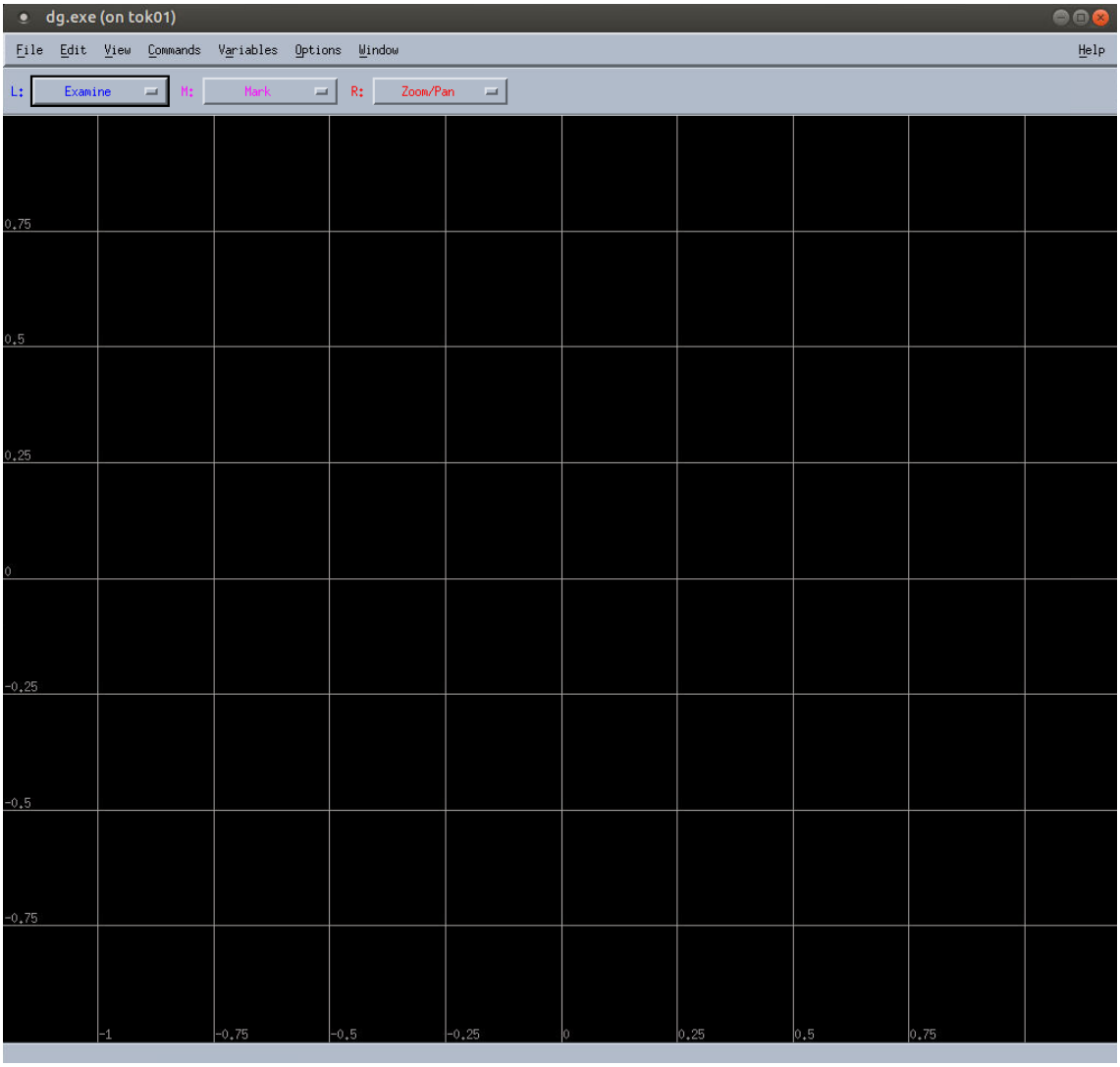
➢ `cd {TEST_CASE}`
➢ `mkdir baserun`

which will contains the main pre-processor input file which will be used to prepare the run. We move inside it by entering

➢ `cd baserun`

and here we paste our `.ogr` and `.eq` files.
Now, we finally start DivGeo by entering

➢ `dg`

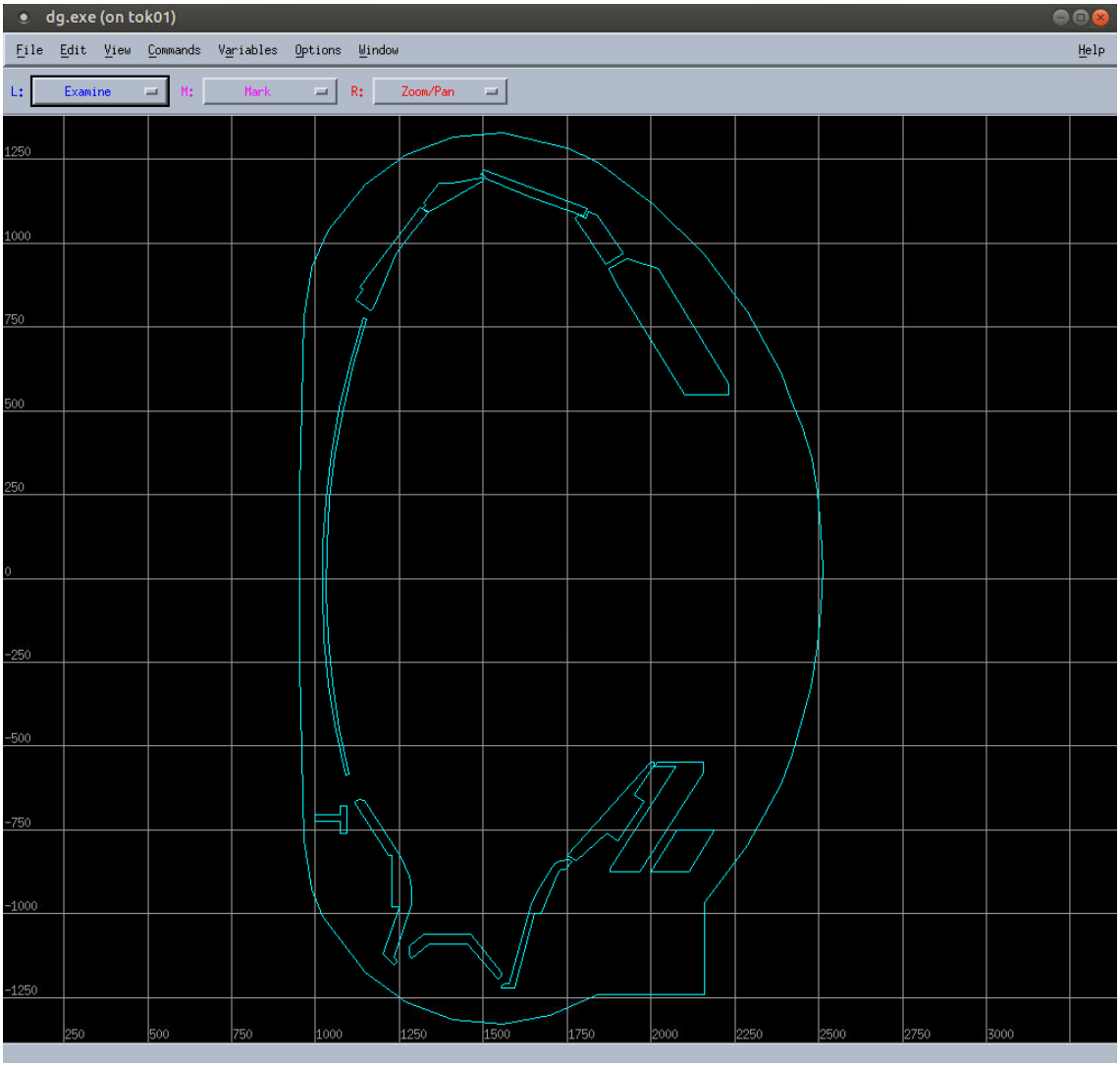


The main window contains a menu bar, giving access to most of the DivGeo functionalities, a toolbar, allowing to assign different tools to each of the mouse buttons, the work area, where the drawing is displayed, and a message line. In the work area, we could create a grid with arbitrary properties by means of the many tools contained in the program.
The work area, in particular, is made of **objects**, among which there are:

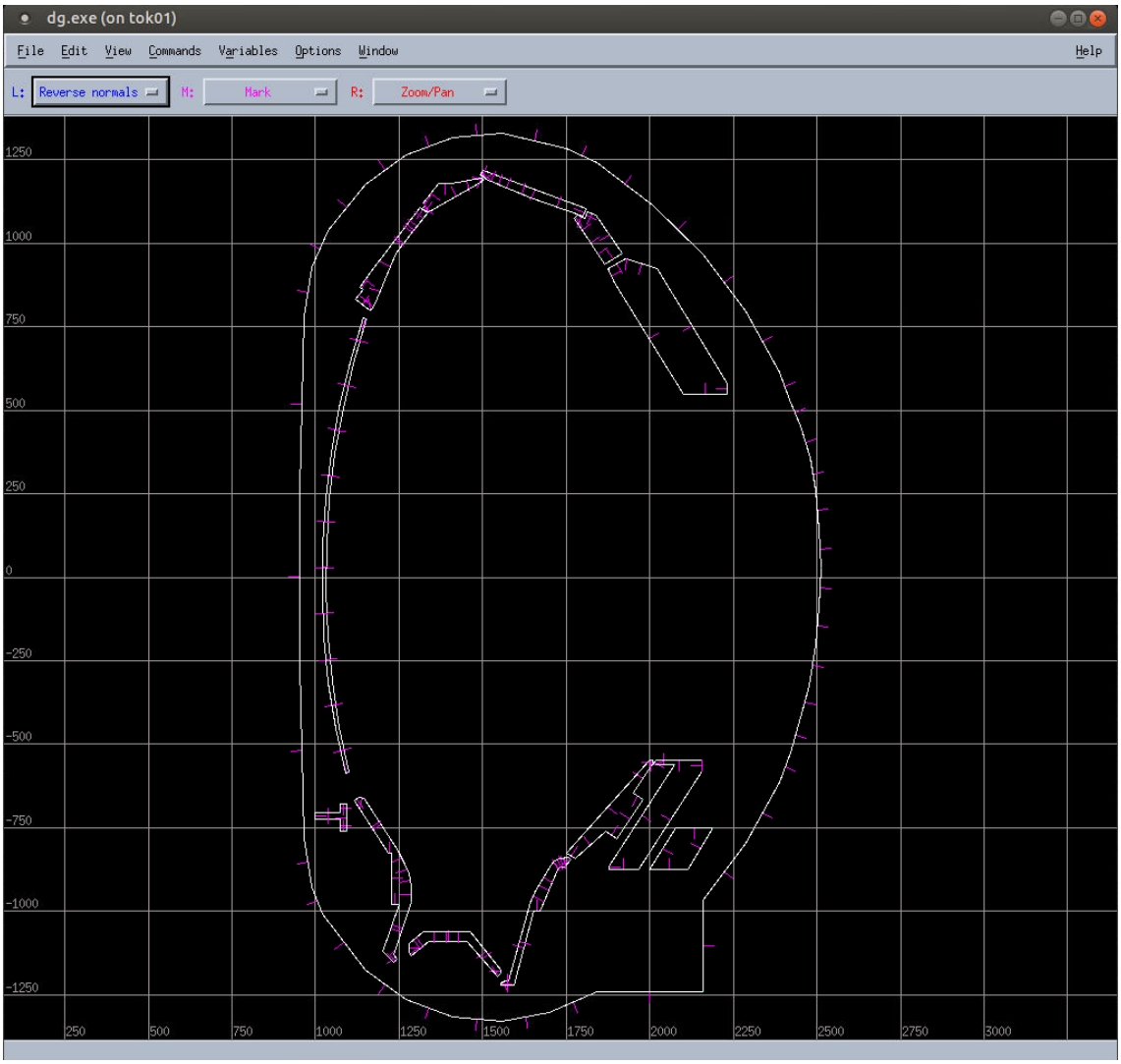
- **Points**, displayed as red squares.
- **Elements**, straight lines connecting points, displayed as white segments.
- **Surfaces**, contours of constant poloidal flux computed from equilibrium data, displayed in yellow. They include the separatrix, displayed as a red line.
- **Grid points**, points in the separatrix which represent partition of the magnetic surfaces in different regions for the successive grid generator, displayed as magenta dashed.
- **Separators**, straight line segments connecting the outer nodes of the computational mesh with the surrounding structure, displayed in cyan.
- **Sources**, special points optionally specifiable, displayed as white asterisks.
- **Equilibrium**, a simple visualization of the equilibrium data, with red and blue dashes corresponding to negative and positive, respectively, values of the poloidal field.
- **Mesh**, computational mesh produced in a grid generator, displayed in blue.

Working and combining these object, we are free to create an arbitrary grid.

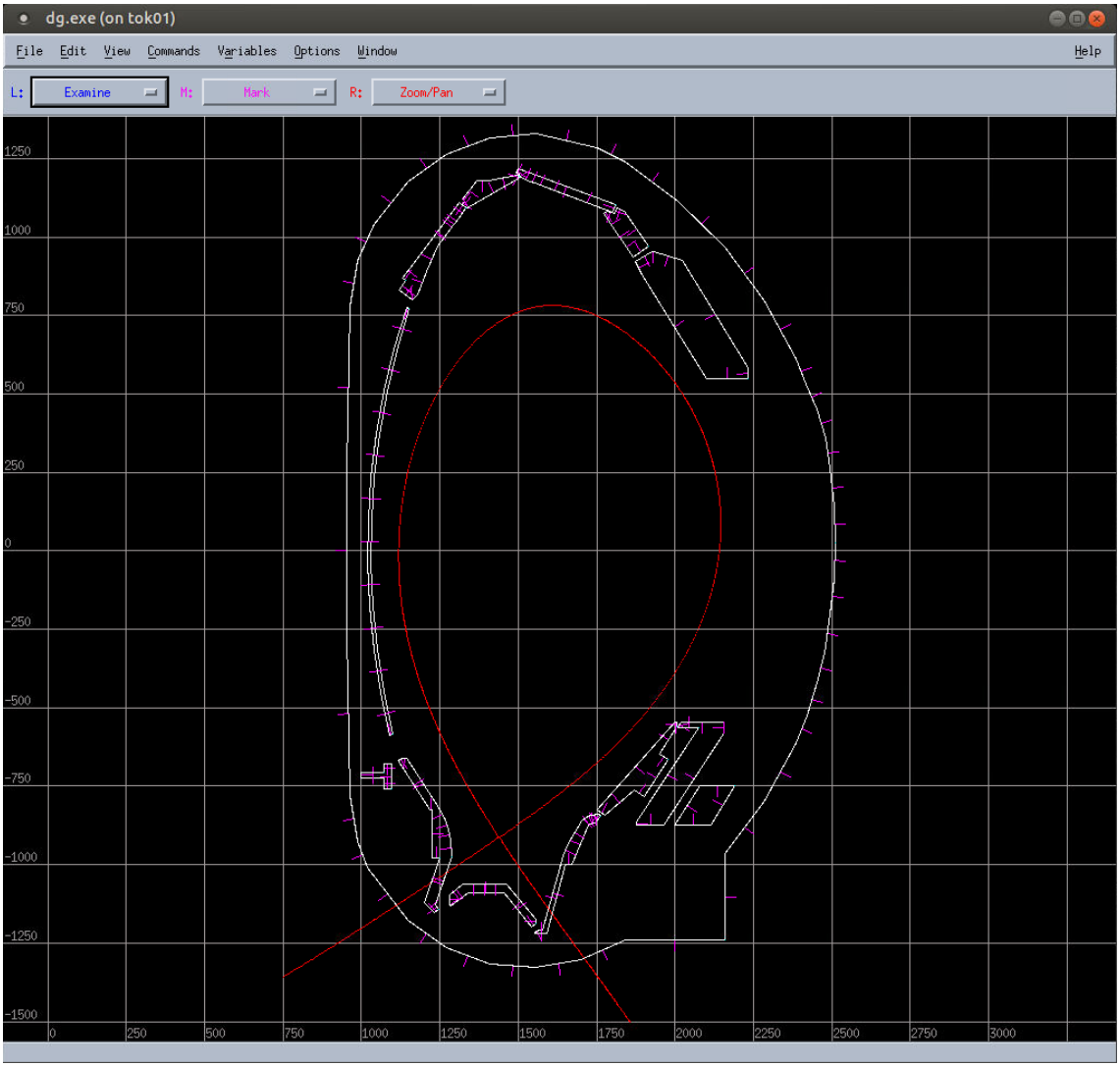
First of all we import the (R,z) data which describe the **geometry of the device**, by importing the `.ogr` template file through `File` → `Import` → `Template`.



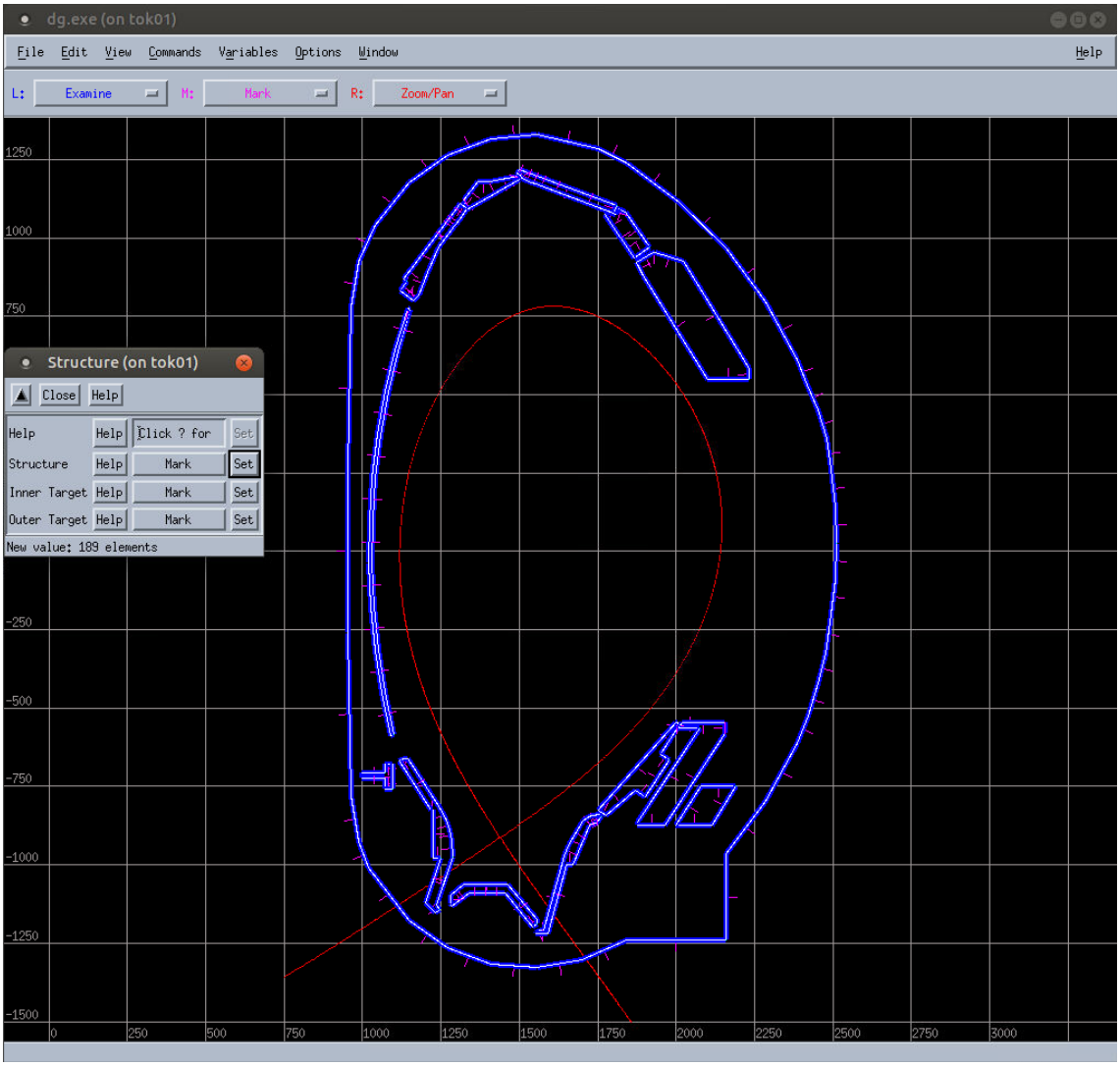
Now, we have to convert the template structure to **geometry elements**, which can be processed by DivGeo, through `Commands` → `Convert` → `Template to elements`. After this, we have a number of white straight lines defining the structures, each one having a short pink line at its center indicating the direction which is defined to be normal to it.
After that, we must make sure that all the **normals** are pointing away from the plasma; this is a convention which is required by later steps in the grid triangulation and input build-up process. If the direction of the normals is not already right, we can reverse it by clicking on them with the command `Reverse normals`.



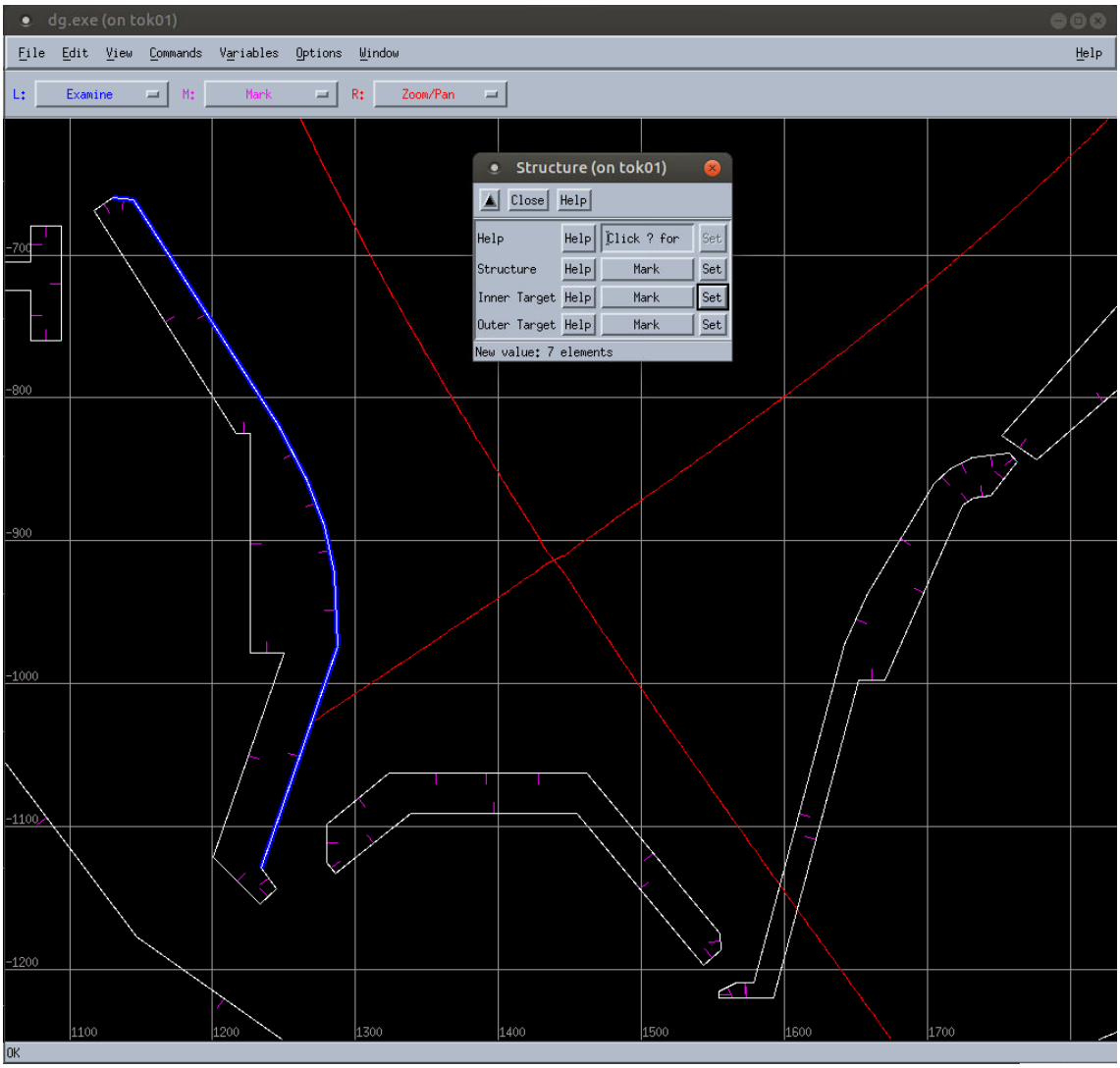
Now, we import the **equilibrium data** of the considered discharge, by importing the `.eq` equilibrium file through `File` → `Import` → `Equilibrium`.
In addition to it, we must explicitly tell DivGeo which **magnetic topology** is to be considered for the modelling grid, by setting it through `File` → `Import` → `Topology`. The choice is between various possibilities like lower single-null (SN), upper single-null (SN-up), connected double-null (CDN), disconnected double-null (DDN), etc.
After that, the region inside the separatrix will be shown in red, while the region outside the separatrix will be shown in blue. In order to clear the display, we can choose what to view through `View` → `Display`. It is advisable to show the separatrix only from here on; in this case, this will be shown as a continuous red line.



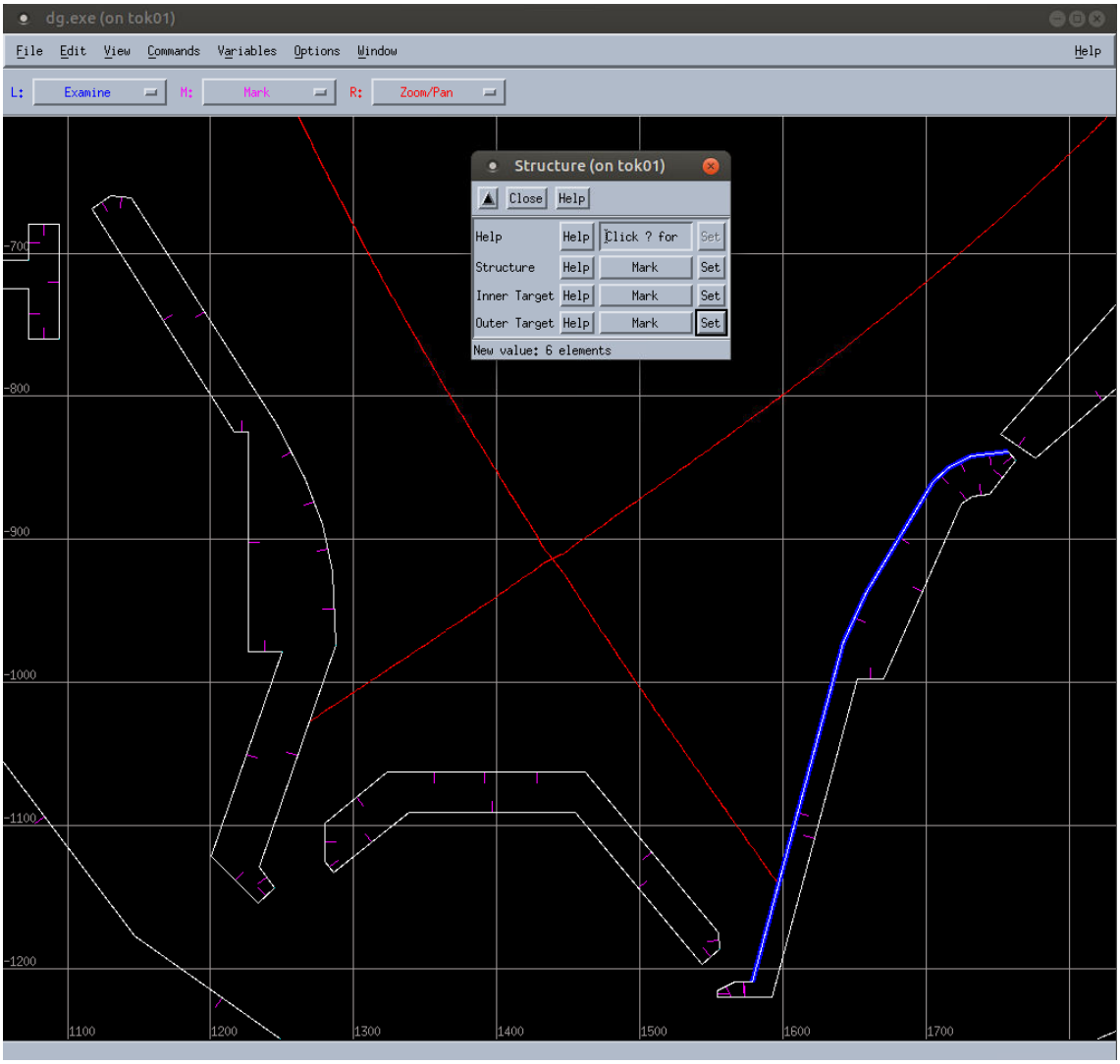
At this point, we must define the adequate variable "**Structure**" for each element. For doing this, we go to `Variables` → `Structure`. We mark all the physical elements, and then we click on "Set" in the first line of the dialogue box to define these elements to belong to the **main structure**.



Then, we must define the extends of the targets. Both the resulting target surfaces will have to consist in at least two elements, and the elements at their ends must be defined as the elements on which the boundaries of the B2.5 grid are expected to strike. This means that the end elements of the two target surfaces will have to be touch only partially the B2.5 grid.
While still in `Variables` → `Structure`, we mark all the physical elements which, according to these rules, must define the inner target, and then we click on "Set" in the second line of the dialogue box to define these elements to belong to the **inner target**.

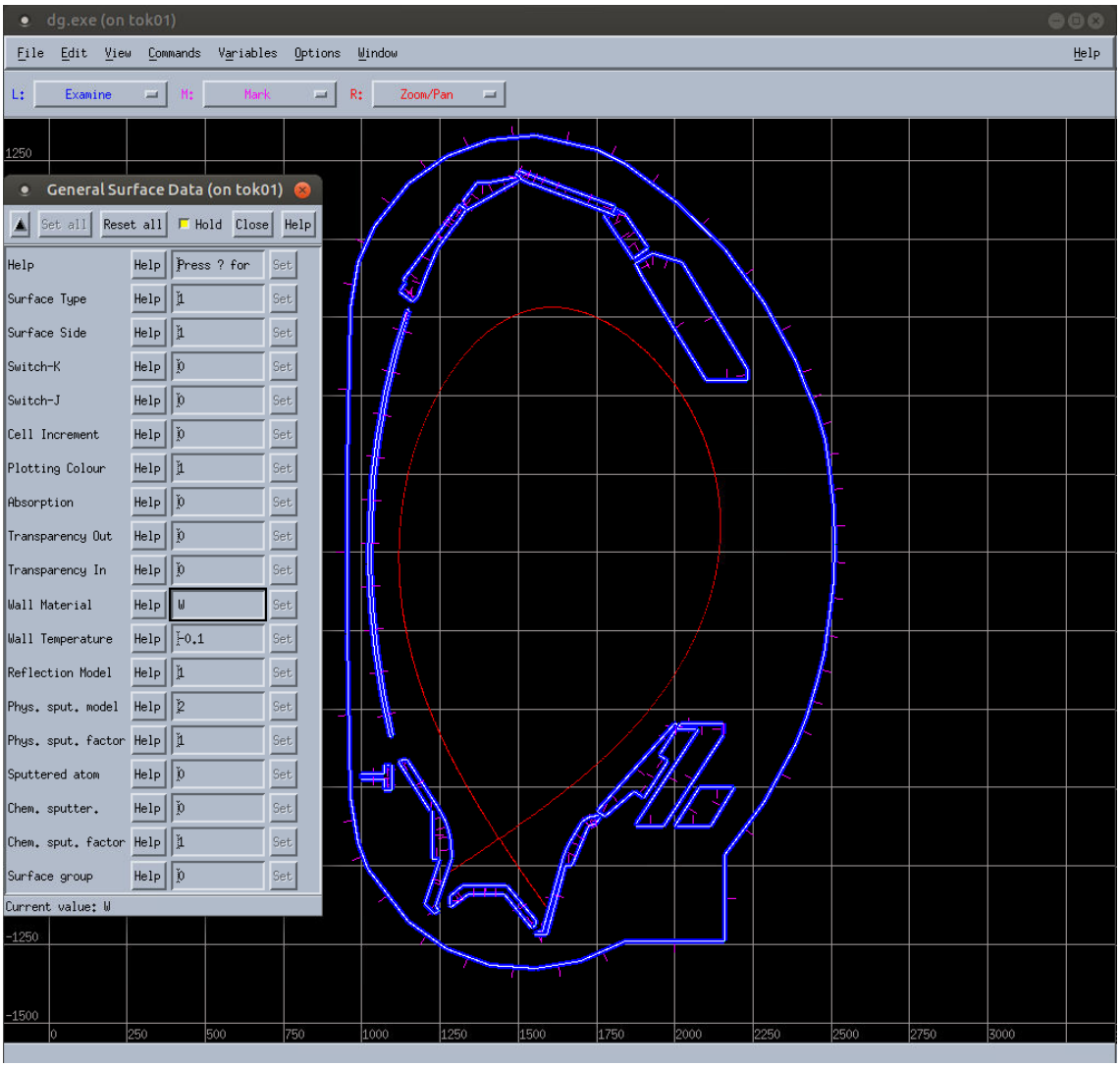


Similarly, for the outer target, while still in `Variables` → `Structure` we mark all the physical elements which must define it, and then we click on "Set" in the third line of the dialogue box to define these elements to belong to the **outer target**.

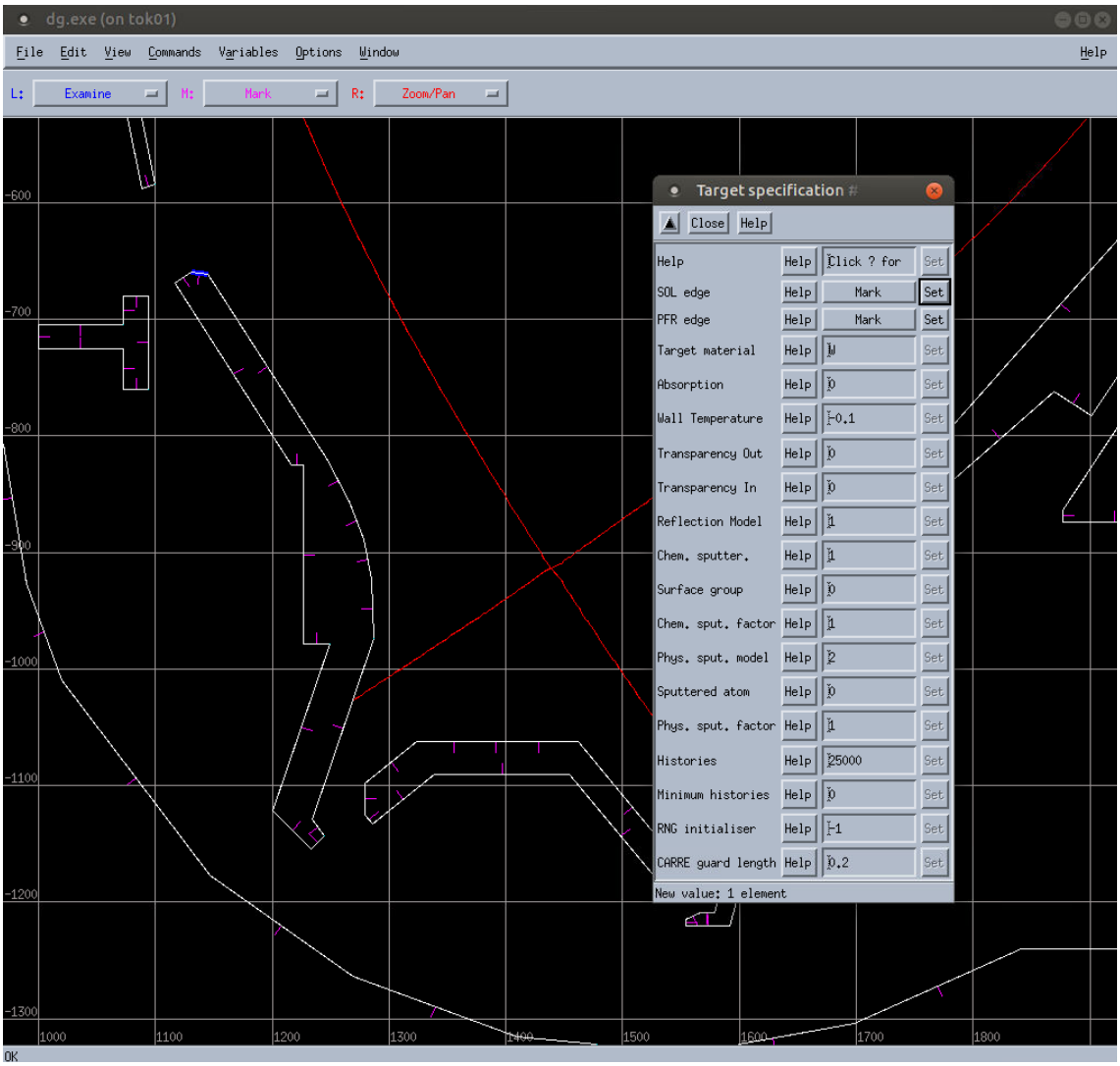


Now, we must define the **general surface data** (i.e. kind of material, physical sputtering model, reflection/absorption model, etc.). For doing this, we go to `Variables` → `General surface data`.

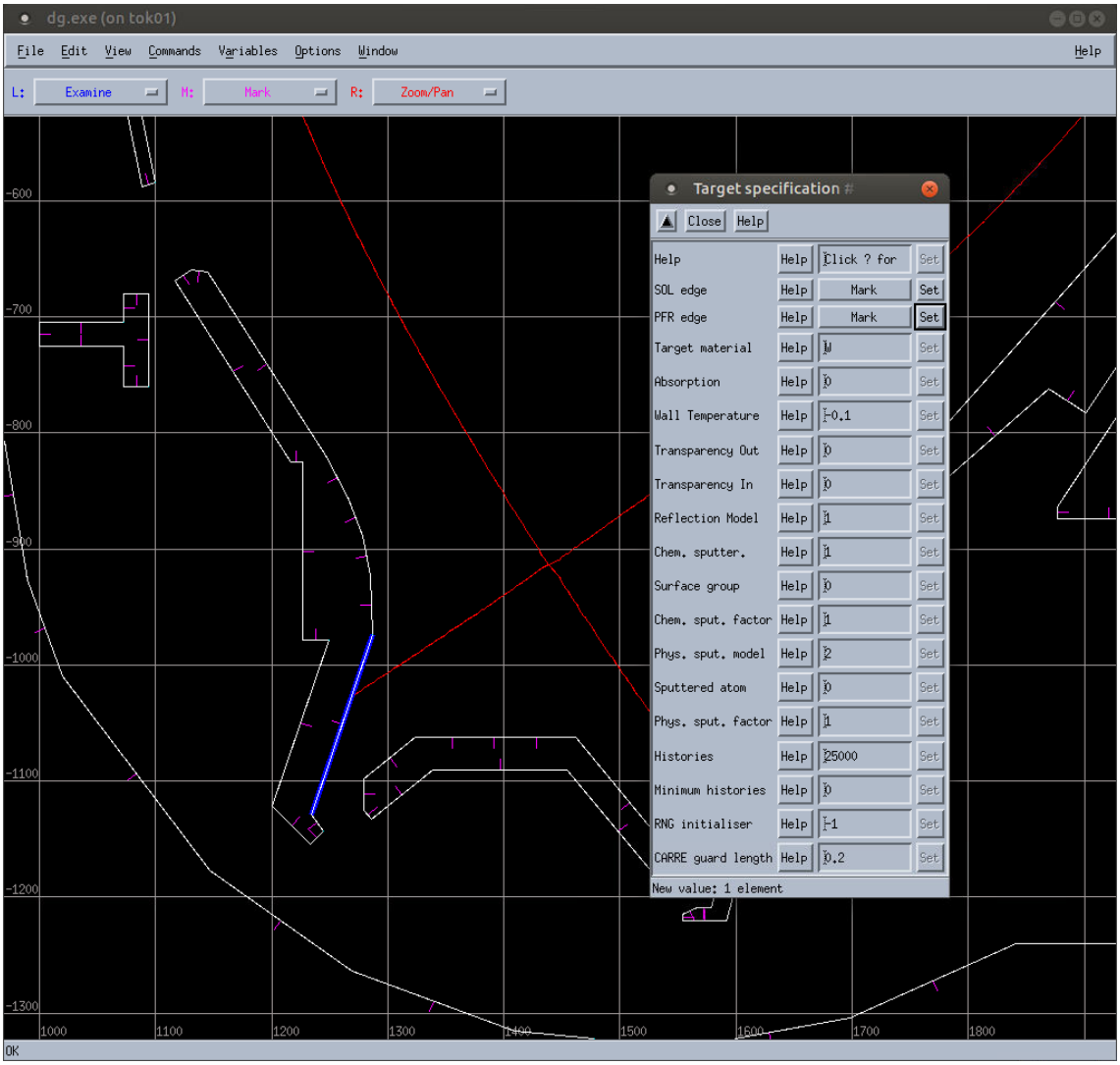
In particular, in the "**Absorption**" line we can define the fraction of incident particles absorbed by the surface (i.e. 1 - fraction of incident particles recycled at that surface), for the elements which represent pumping surfaces. Then, in the "**Wall material**" line we can define the physical material of each surface, choosing between beryllium (Be), Carbon (C), Iron (Fe), Molybdenum (Mo) and Tungsten (W), while in the "**Wall temperature**" line we can set the assumed temperature (in eV) of the surface. Finally, in the "**Phys. sput. model**" line we can define the physical sputtering model to be used (0 for no physical sputtering, 2 for the standard Roth-Bohdansky model), while in the "**Chem. sputter.**" line we can define the chemical sputtering model to be used (0 for no chemical sputtering, 1 for a constant chemical sputtering yield model).



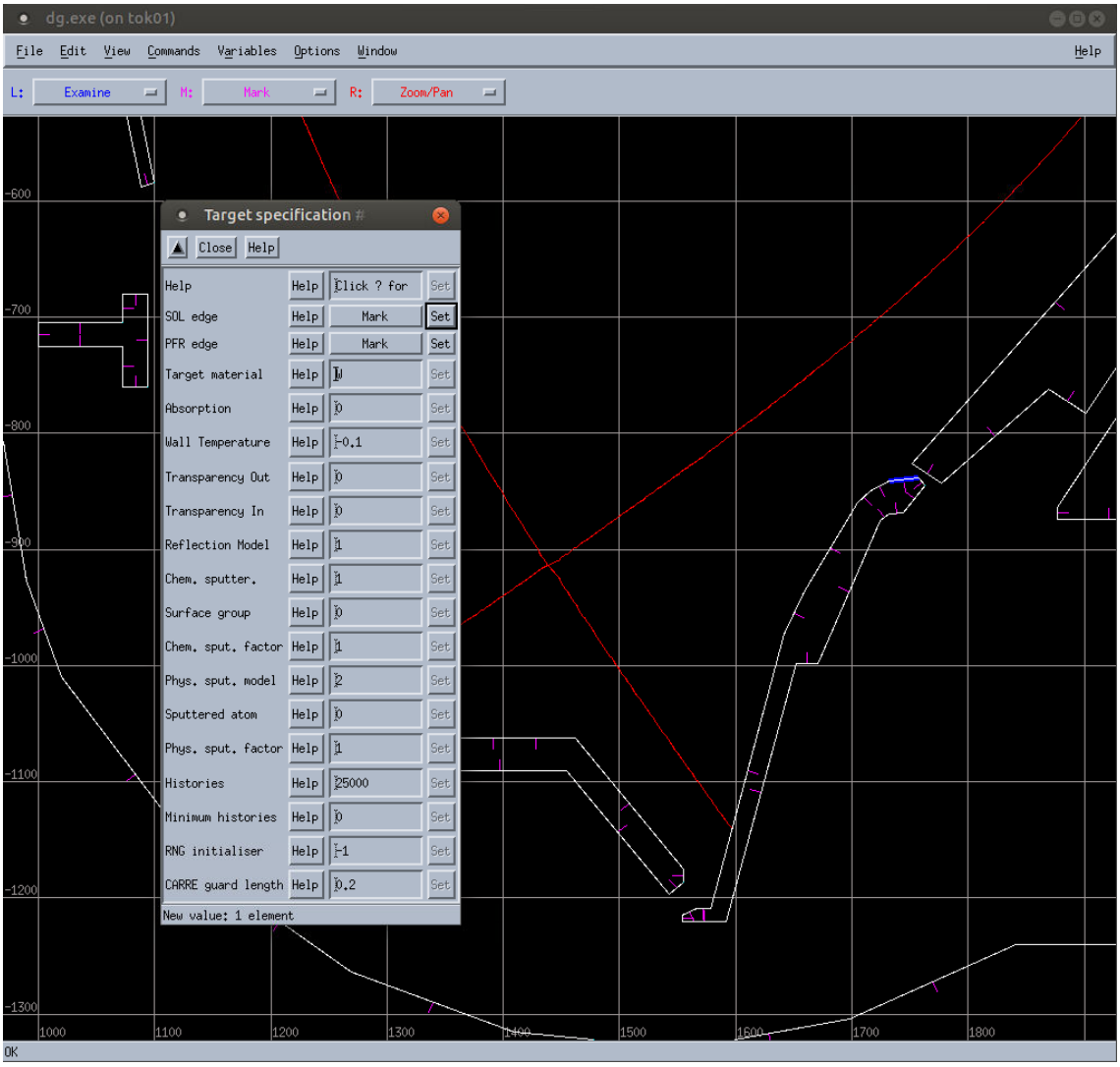
Then, we turn to the **target specifications**, in which we define the general surface data for the targets, and the position of their boundary elements.
For the inner target, we go to `Variables` → `Target specification` → `#1`. After having specified the general surface data, we mark the element on which the boundary of the B2.5 grid in the SOL region is expected to strike, and then we click on "Set" in the "**SOL edge**" line.



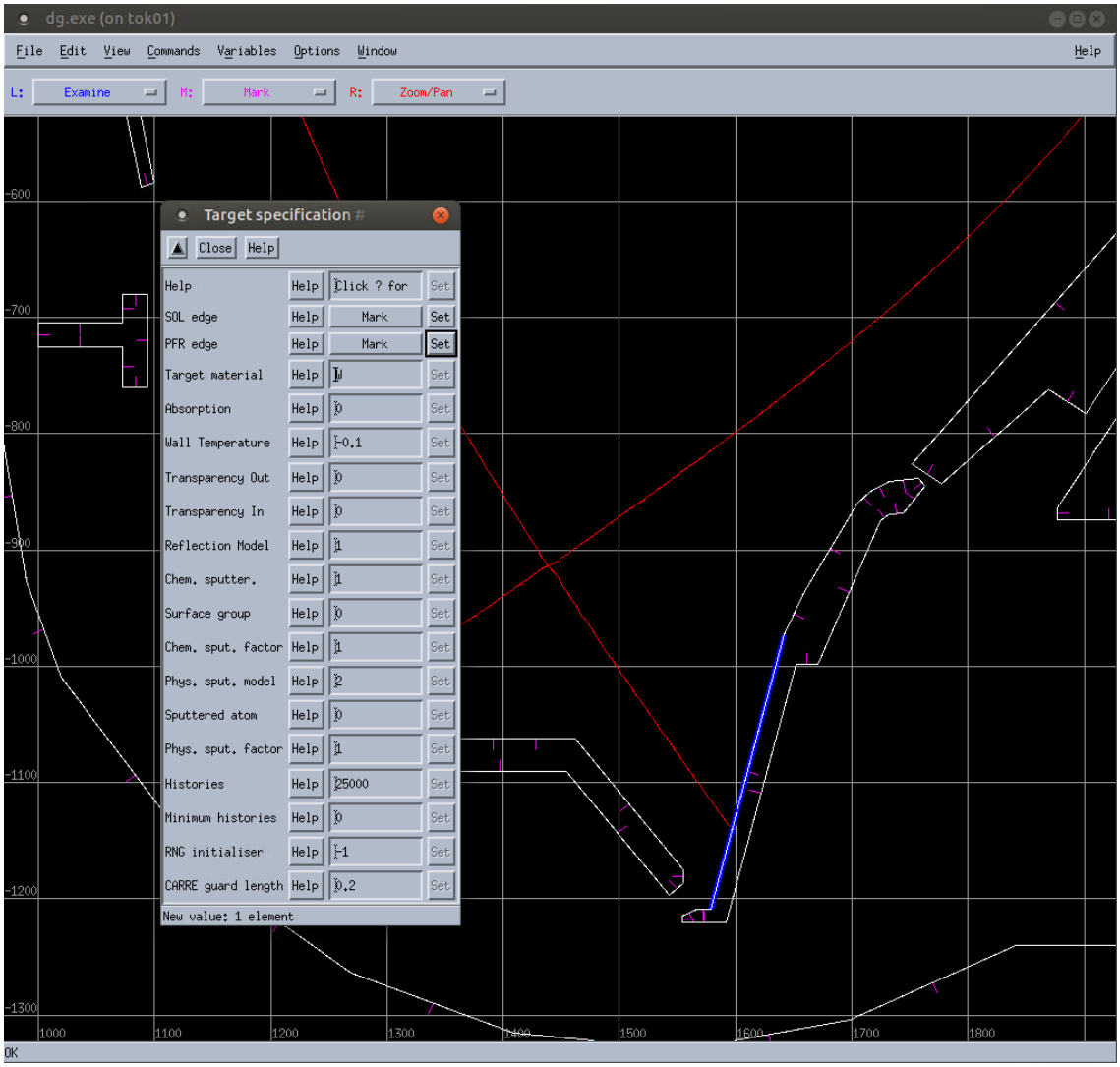
Then, we mark the element on which the boundary of the B2.5 grid in the PFR region is expected to strike, and then we click on “Set” in the “**PFR edge**” line.



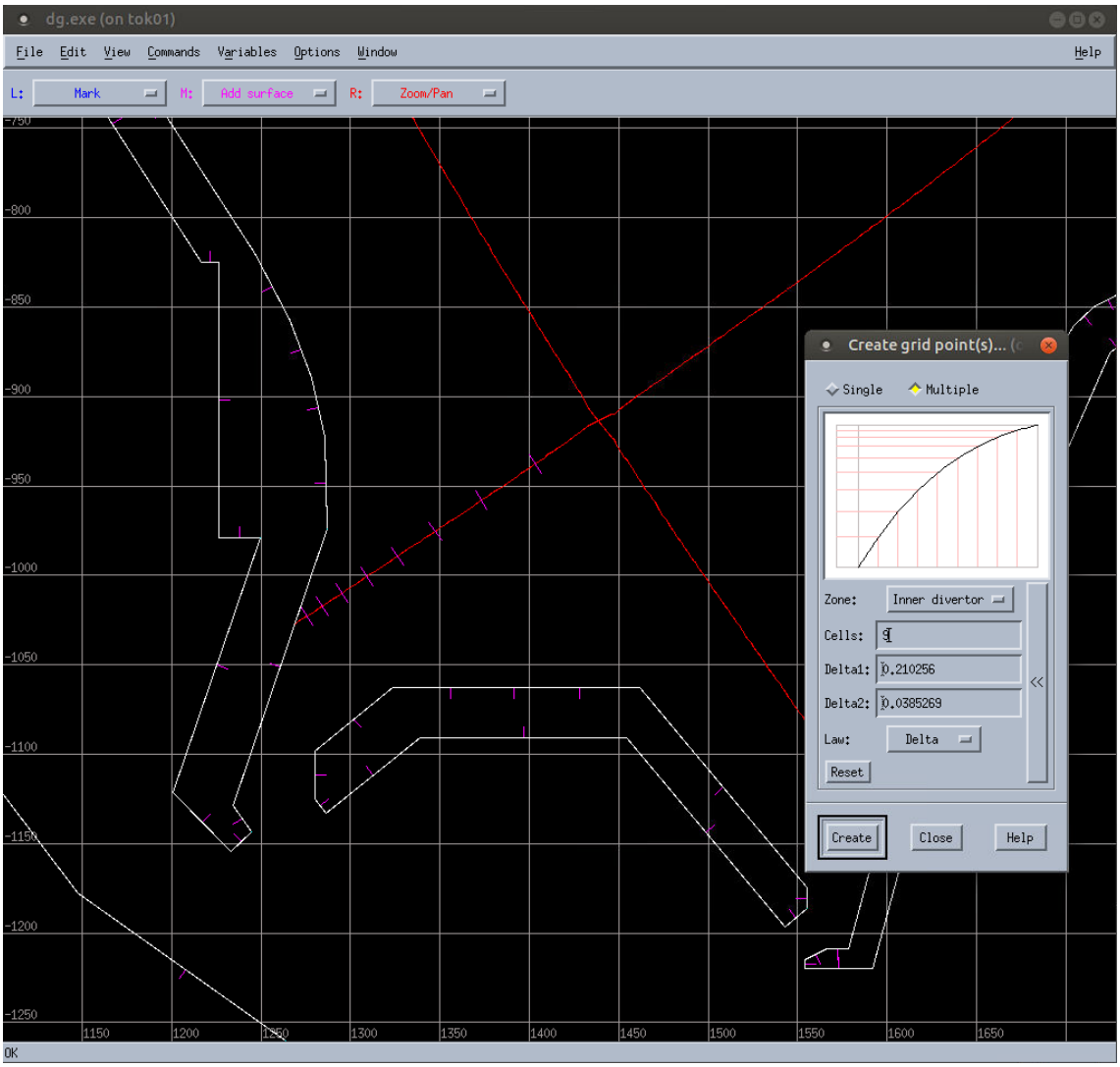
Similarly, for the inner target, we go to `Variables` → `Target specification` → `#2`. After having specified the general surface data, we mark the element on which the boundary of the B2.5 grid in the SOL region is expected to strike, and then we click on "Set" in the "**SOL edge**" line.



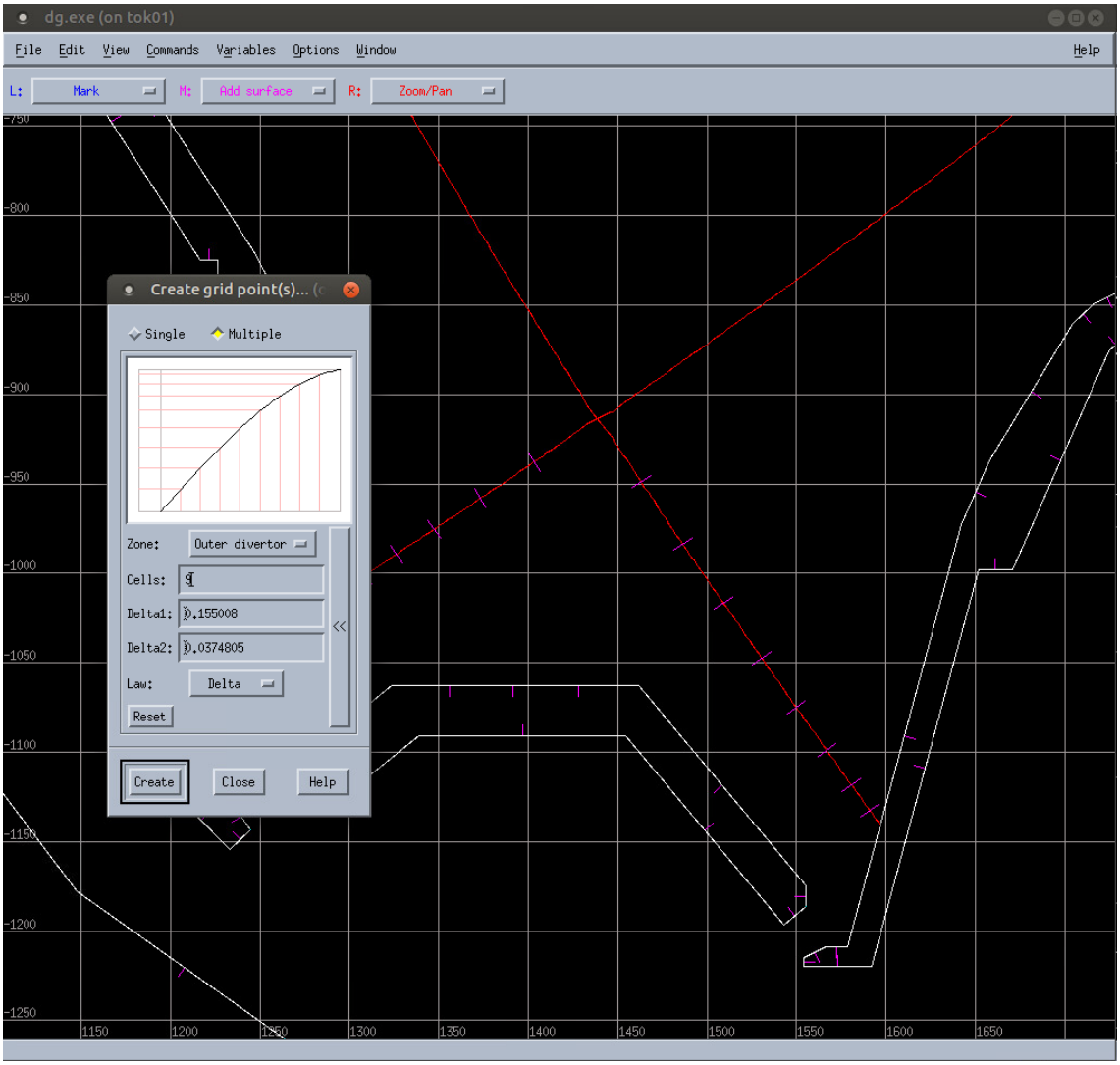
Then, we mark the element on which the boundary of the B2.5 grid in the PFR region is expected to strike, and then we click on “Set” in the “**PFR edge**” line.



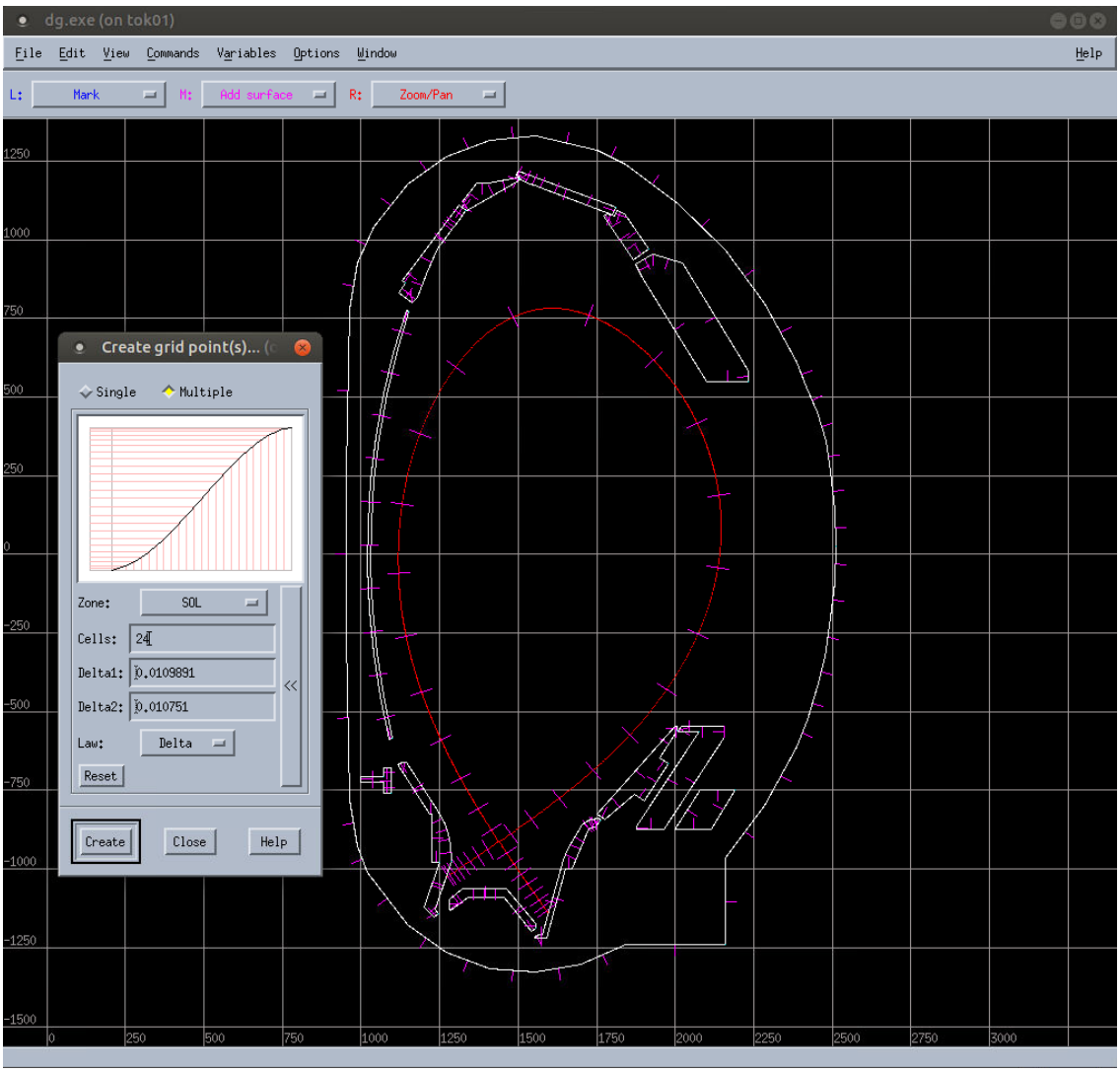
Finally, we start defining which will be the **poloidal distribution of the grid points** in the B2.5 mesh, to be done separately for the two divertor legs and the SOL region.
For doing this we go to `Edit` → `Create` → `Grid points`. First of all we put into the "**Inner divertor**" zone, we select the desired number of cells (typically 12 or 24) and then we adjust the gradient in order to have higher spatial resolution near the target.



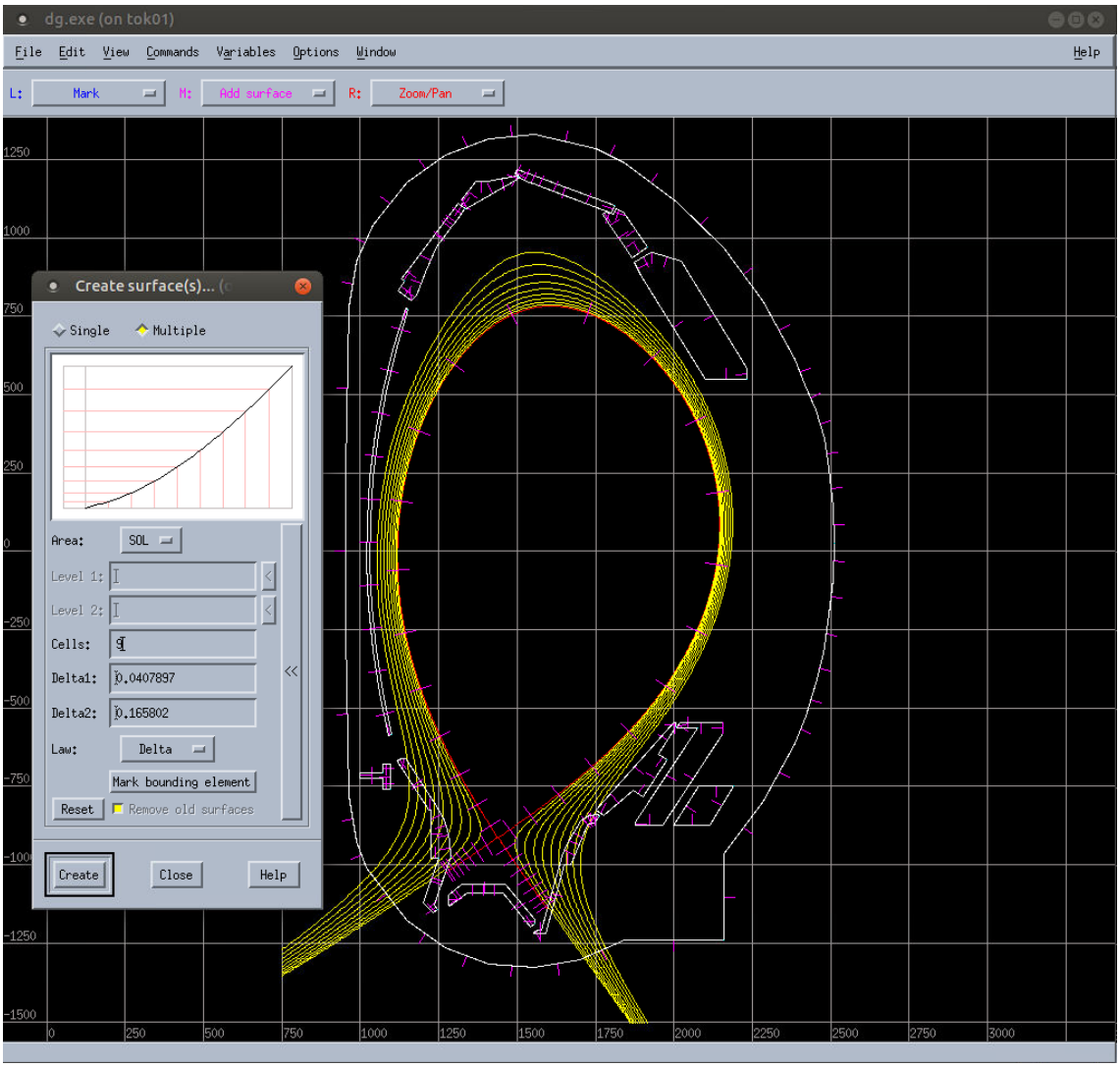
Similarly, while still in `Edit` → `Create` → `Grid points`, we put into the "**Outer divertor**" zone, we select the desired number of cells (typically 12 or 24) and then we adjust the gradient in order to have higher spatial resolution near the target.



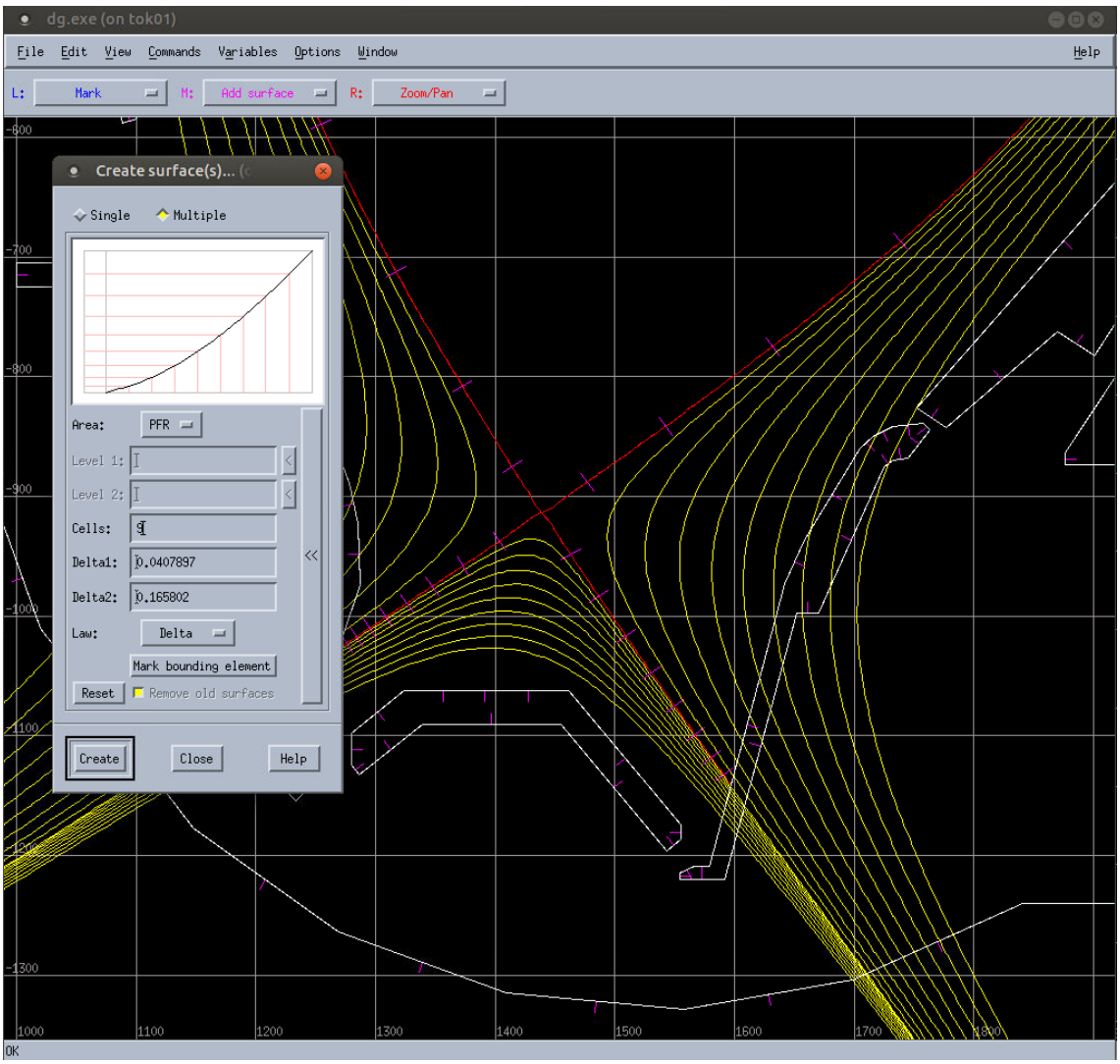
Finally, while still in `Edit` → `Create` → `Grid points`, we put into the "SOL" zone, we select the desired number of cells (typically 24 or 48) and then we adjust the gradient in order to have a roughly uniform distribution of grid points, paying attention to obtain a spacing of grid points more or less symmetric around the X-point.



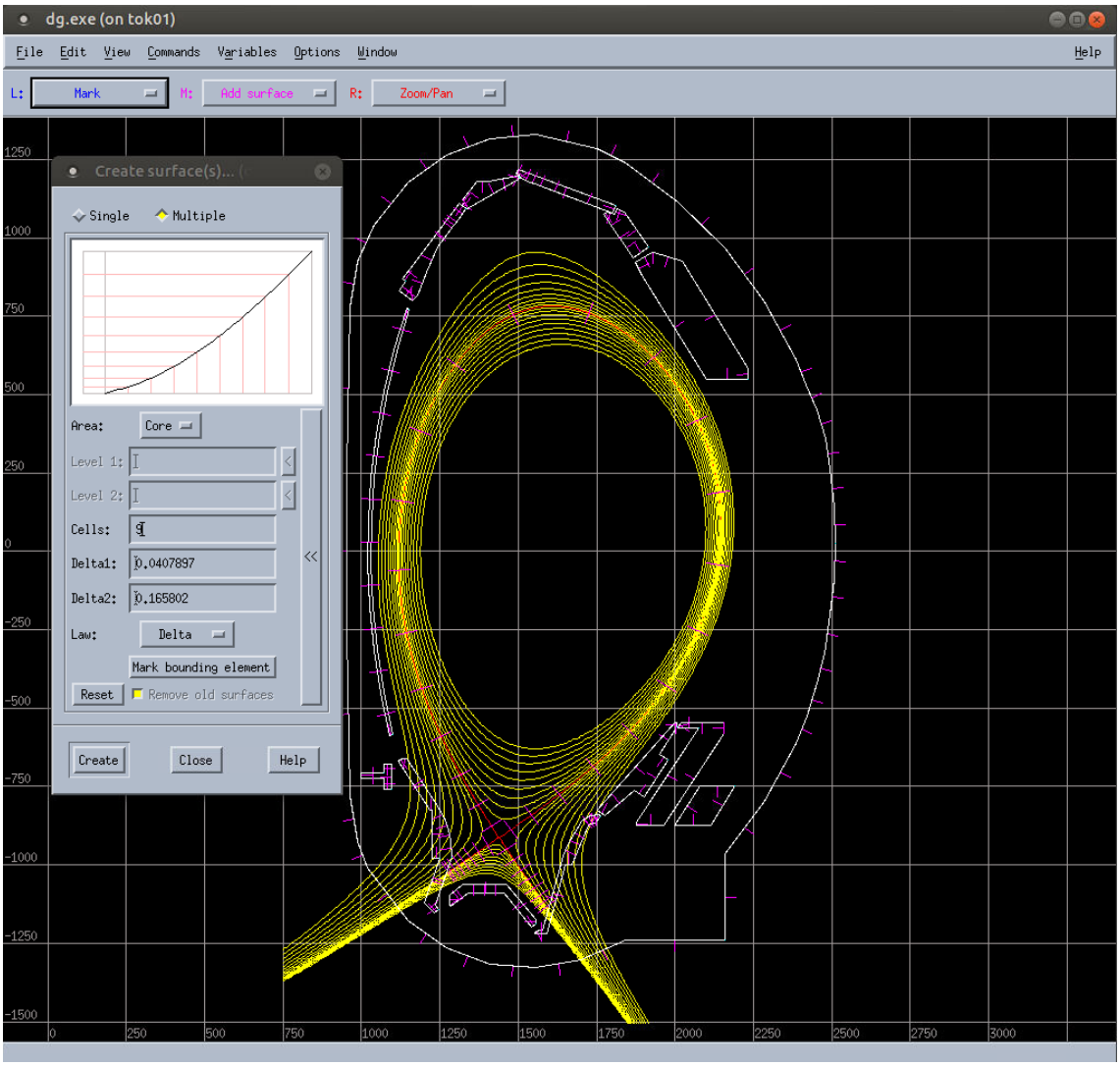
Then, we turn to define the **radial surfaces**, according to the imported magnetic equilibrium, which will define the boundaries between rings on the B2.5 grid.
For doing this we go to `Edit` → `Create` → `Surfaces`. First of all we put into the "**SOL**" region, we set a number of surfaces (typically 9 or 18) and then we adjust the radial distribution in order to obtain higher spatial resolution near the separatrix. The outermost ring will be automatically defined by the position of first tangency with a material surface.



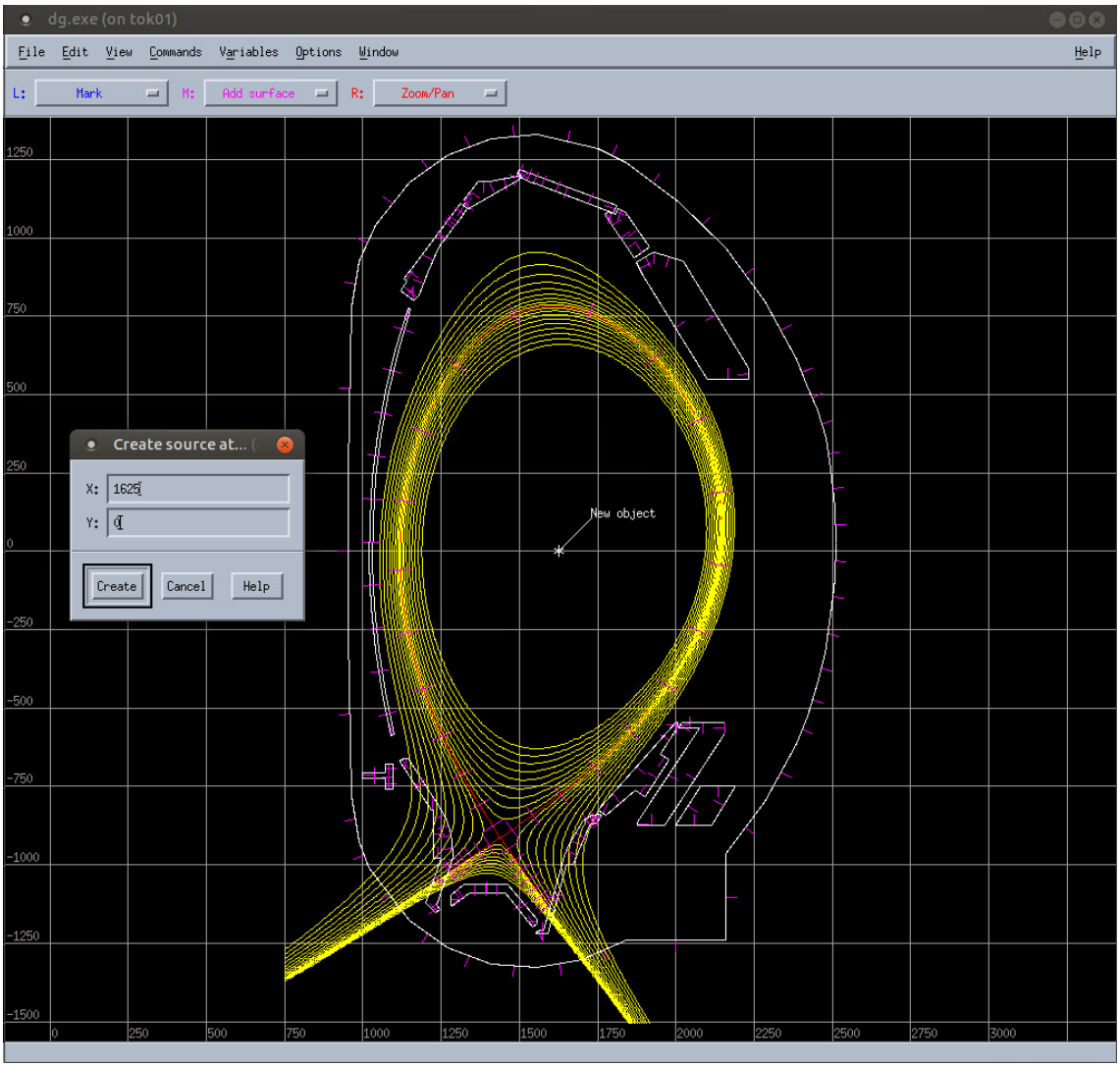
Then, while still in `Edit` → `Create` → `Surfaces`, we put into the "PFR" region, we set a number of surfaces (typically 9 or 18) and then we adjust the radial distribution in order to obtain higher spatial resolution near the separatrix. Again, the outermost ring will be automatically defined by the position of first tangency with a material surface.



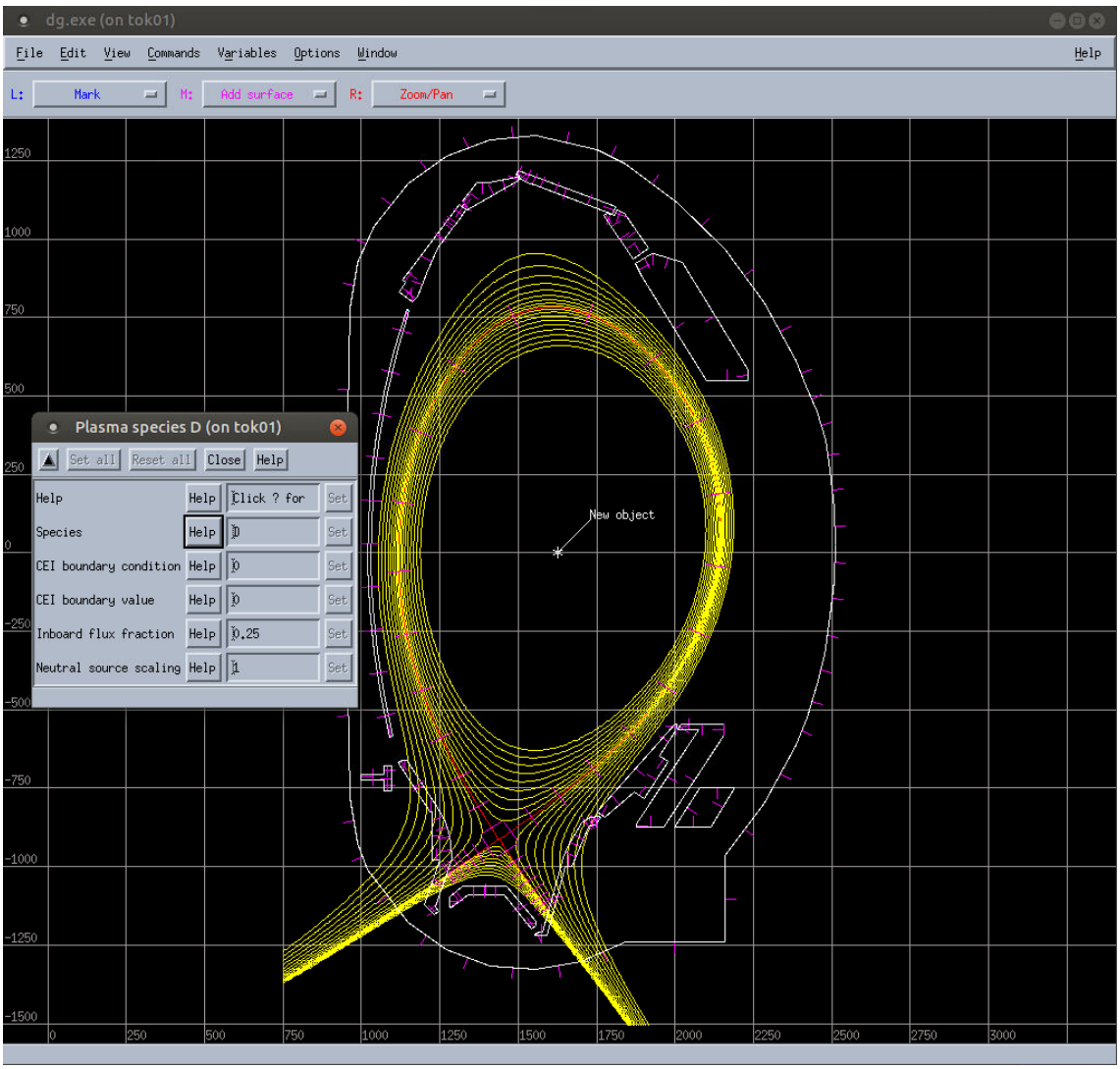
Finally, for what concerns the core region, first of all we must add a surface which will define the extent to which the grid penetrates into the core, through the command `Add surface`. Then, while still in `Edit` → `Create` → `Surfaces`, we put into the "core" region, we set a number of surfaces (typically 9 or 18) and then we adjust the radial distribution in order to obtain higher spatial resolution near the separatrix.



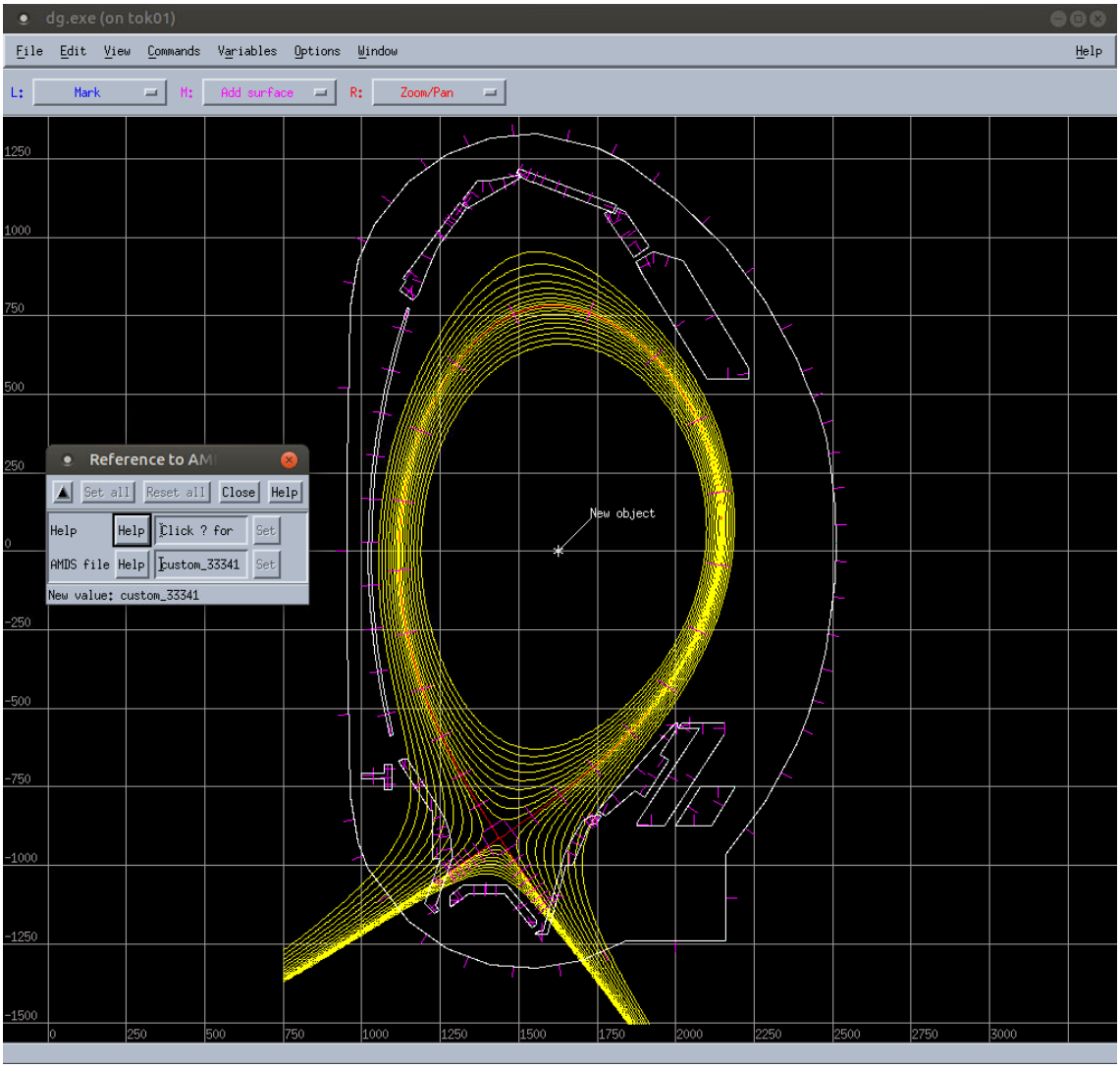
It is also possible to include some **core radiation**, i.e. the power which is experimentally known (for the modelled scenario) to be radiated in the region inside the core which is not covered by the grid. For doing this we go to `Variables` → `Add` → `Radiation sources`, and we enter in the "Radiated Power" line the amount of core radiation in MW.
Then, it is needed to specify the location from where this core radiation is emitted. This is done by providing a set of **point sources**, to be done through `Edit` → `Create` → `Source`. The radiated power will be then spread evenly among the defined point sources.



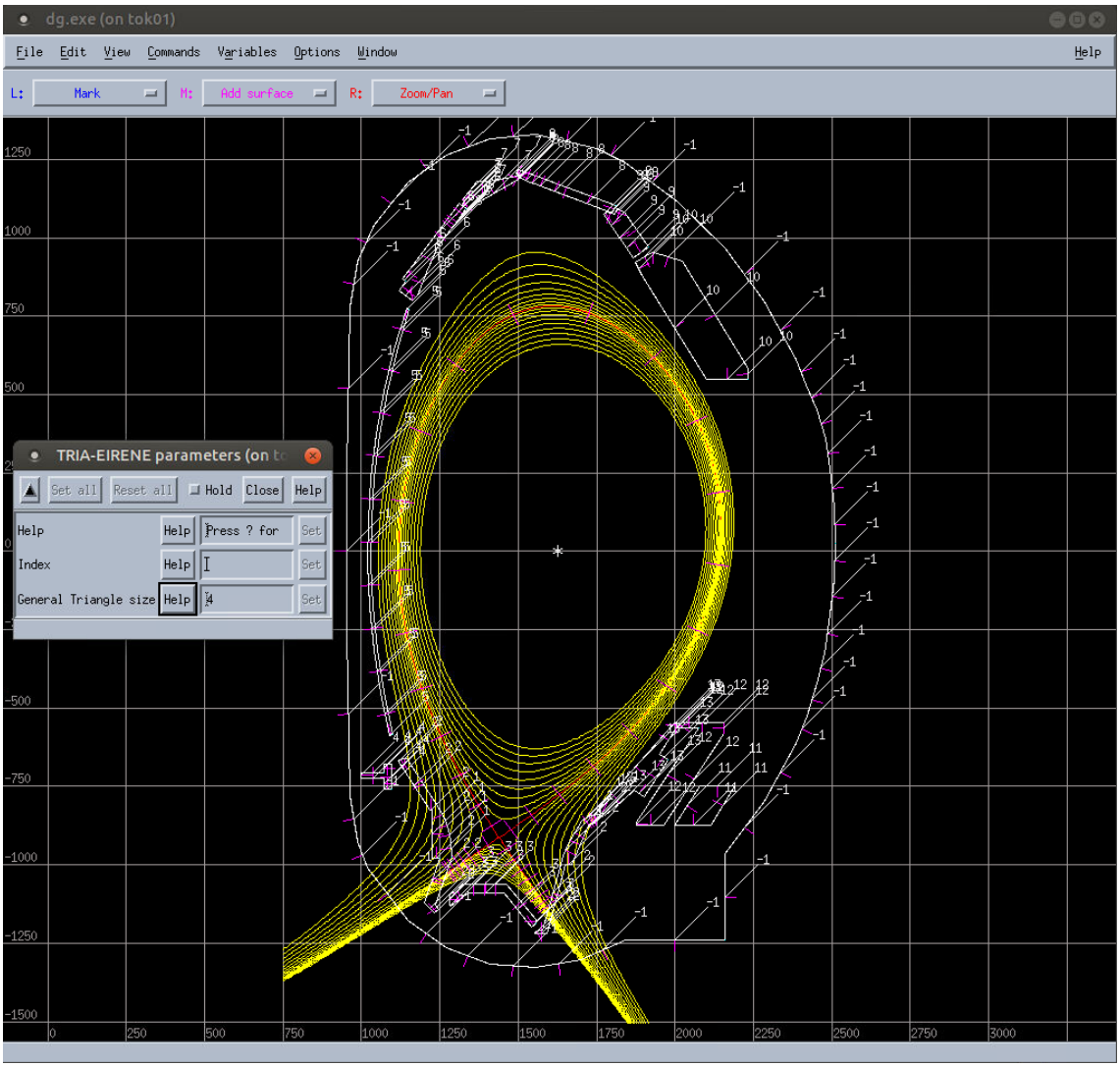
Now, we must finally specify the **plasma species** to be included in the simulations, which can be done by going to `Variables` → `Plasma species D`. The possibilities are hydrogen (H), deuterium (D), tritium (T), helium (He), beryllium (Be), carbon (C), nitrogen (N), neon (Ne), argon (Ar). Moreover, **impurity species** can be added by going to `Variables` → `Add` → `Plasma species`.



The **atomic and molecular processes** to be considered must be chosen from an AMDS file, by typing the name of the `.amo` file which is desired to be considered in `Variables` → `Add` → `Reference to AMDS`.



Now it is the moment to specify some settings for the EIRENE triangular grid. EIRENE, in fact, will use a grid in regions that are outside the B2.5 grid as well, so we must to define precisely the regions in which we want the triangle mesh generator to produce the EIRENE mesh.
This is done going to `Variables` → `Add` → `TRIA-EIRENE parameters`. We must set negative indices in elements such that the mesh should be generated inside the marked region, while positive indices in elements such that the mesh should be generated outside the marked region. In our example case, we put -1 for the boundary of the vacuum vessel. Then, we put 0 for elements belonging to both inner and outer target. Then, we put 2 for the remaining elements belonging to the target polygons. Finally, we put different positive values for all the other closed polygons.
In addition, in the "General triangle size" line, we can specify which should be the average **triangle size** (in cm).
After that, we go to `Variables` → `Global Eirene Data` and we set the "Major Radius" value to a negative real number, such as -1.0, in order to use the toroidal approximation instead of the cylindrical approximation (as it is the case for tokamaks).
At this point, we check if all the values which we assigned to the various variables make sense, through `Commands` → `Check variables`. Hopefully, all variables should have a valid value. Then, we go to `Commands` → `Rebuild Carre objects`.



Now we can save the DG model, in the `baserun/` directory, through `File` → `Save`. This will create the `{MODEL}.dg` file, through which we can still make modifications to our model if desired.
Then, we produce the output files `File` → `Output`. This will create three files:

- The **main DG output file** `{MODEL}.dgo`.
- The **structure file** `{MODEL}.str`, specifying geometry and properties of the main structure for our model.
- The **target file** `{MODEL}.trg`, specifying properties and extension of the targets for our model.

The just produces files must be finally used as input files by the Carre program.
We should also enter the command

➢ `lns {MODEL}`

which will provide a copy of the various DG output files in the standard DivGeo directory `solps-iter/modules/DivGeo/device/{DEVICE}`.

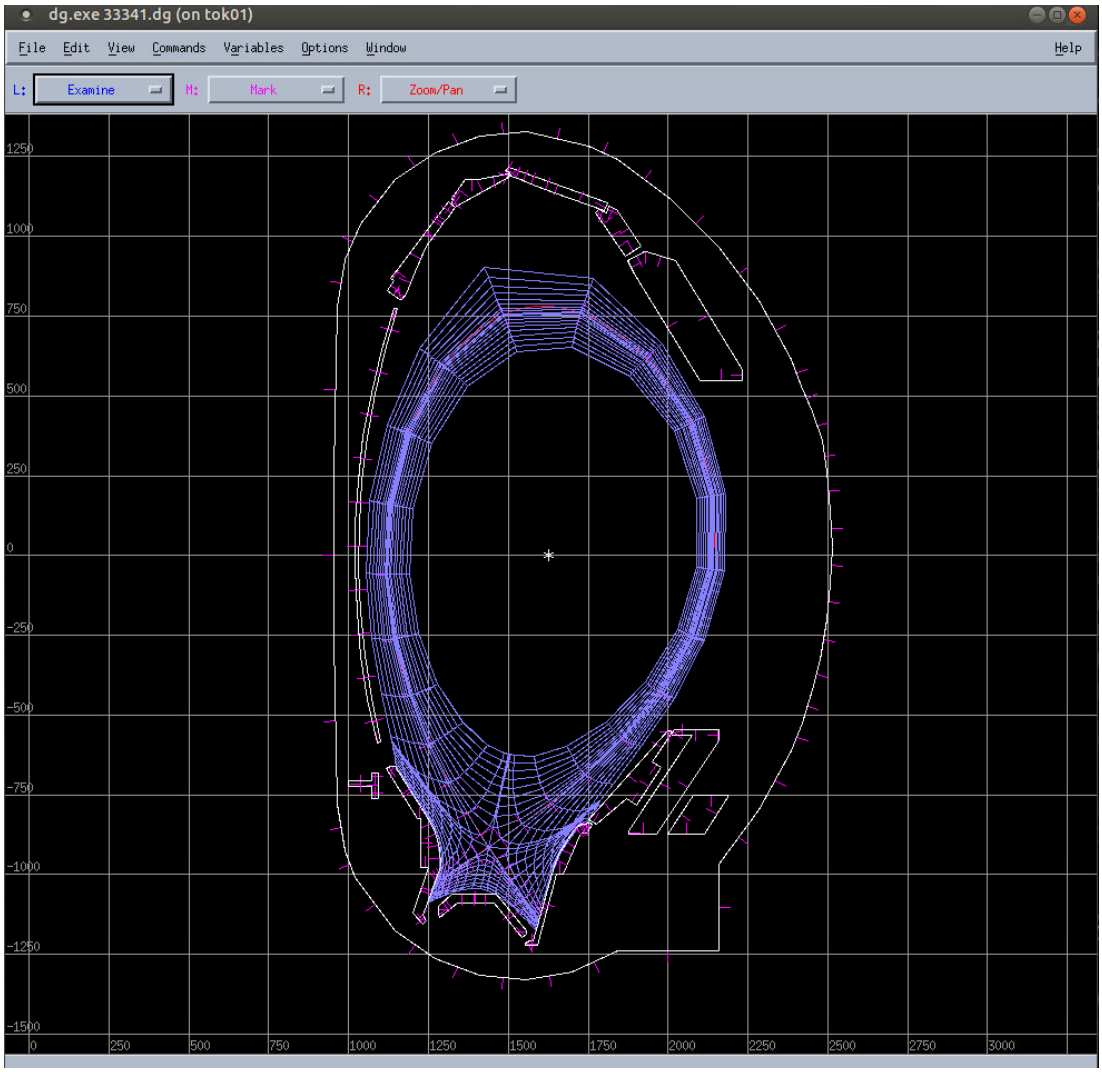
We are ready to proceed with the creation of the B2.5 grid using the **Carre** program. We must enter

➢ `carre -`

and then follow the instructions in the terminal window.
At the end of this process, we have produced the **mesh files**:

- `{MODEL}.v00x.sno`, stored in the standard DivGeo directory `solps-iter/modules/DivGeo/device/{DEVICE}`.
- `{MODEL}.v00x.geo`, stored in the standard Carre directory `solps-iter/modules/Carre/meshes/{DEVICE}`.

We can now import the just produced `.sno` file again in DivGeo, through `File` → `Import` → `Mesh`, to see how the B2.5 grid appears.



If we are happy with the just produced grid, we can finally turn to produce the triangular grid for EIRENE, which is done by starting the triangulation program **Triang** through

- `triang`

Following again the instructions which appear in the terminal window, first of all the **Uinp** script is called to produce a series of input files for the main programs, that is:

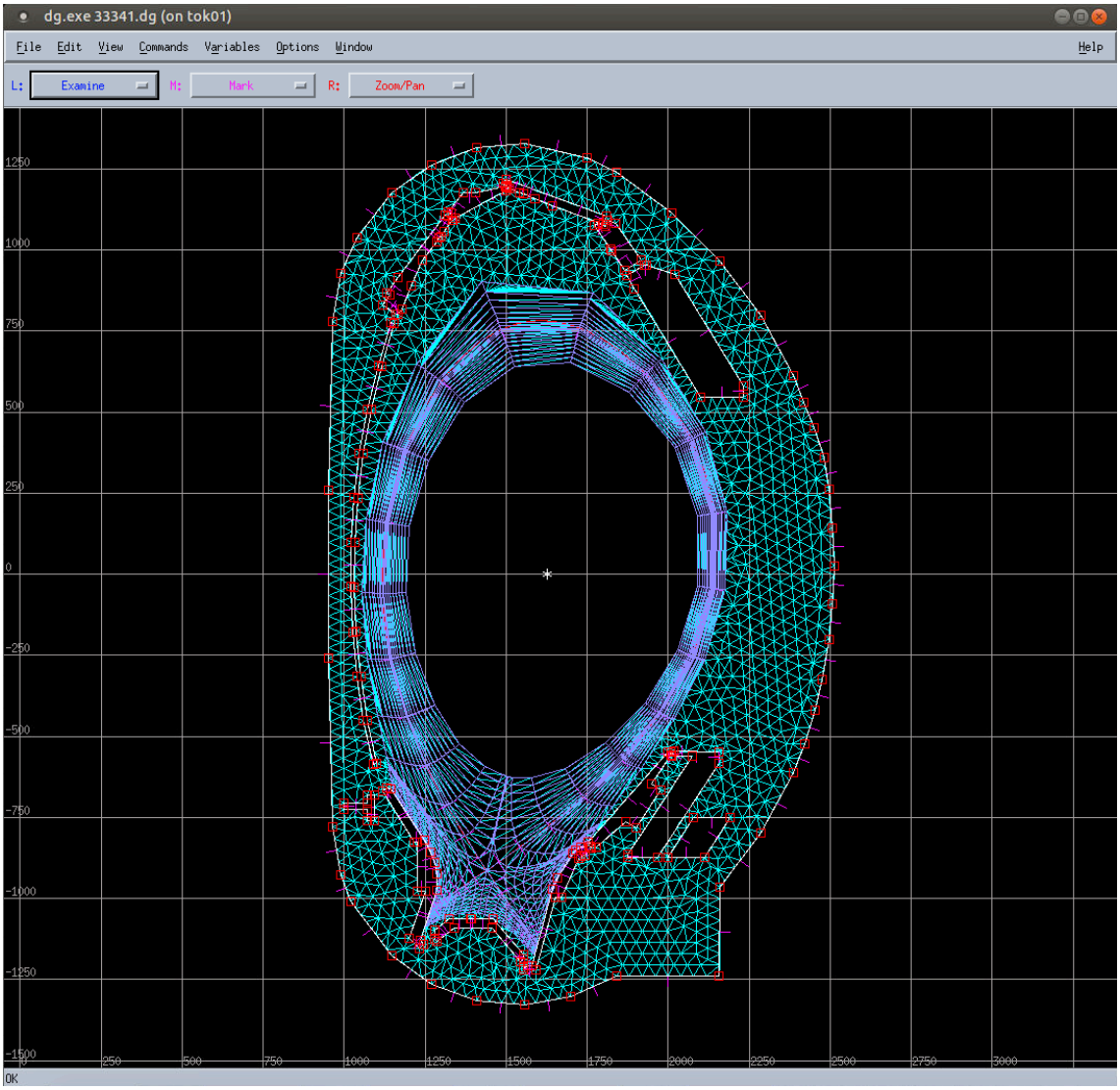
- The **Tria input file** `triang.hed`.
- The **EIRENE input files** `input.eir`, `test.eir`, `triang.eir`.
- The **B2.5 pre-processor input files** `b2ag.dat`, `b2ai.dat`, `b2ar.dat`, `b2ah.dat`.
- The **B2.5 input file** `b2.user.parameter`.
- The **B2plot input file** `mesh.extra`.

After that, keeping following the instructions, the mesh files are used to create a geometry file `fort.30` that provides the mesh geometry to be used by EIRENE. Then, we start the triangulation preparation run, which the full contours that will be used by EIRENE are written in the geometry file `fort.78`. The running script then concatenates this files with the Tria input file `triang.hed`, which contains the information about the desired triangle size, to be used as input for the **Tria** program.
At this point, the actual triangulation is performed by Tria, with the resulting information stored in the three files `tria.nodes`, containing the coordinates of the nodes of the triangles, `tria.elemente`, containing the coordinates of the vertices of the triangles, and `tria.neighbor`, containing information about the connectivity between triangles.
Finally, also the B2.5 grid is triangulated and it is merged with the outer triangles. The final information about the triangular grid will be contained in the just produced **triangle files**

- `{DEVICE}.tria.00x.nodes`
- `{DEVICE}.tria.00x.cells`
- `{DEVICE}.tria.00x.links`

stored in the standard Triang directory `solps-iter/modules/Triang/triangles/{DEVICE}`. These three files are also linked, to the `baserun/` directory, through the `fort.33`, `fort.34` and `fort.35` link files, respectively.

At the very end, the triangles files are converted in two template files which can be read by DivGeo, that is the outer template file `template.a00x.tria` and the grid template file `template.g00x.tria`, stored in the standard DivGeo directory `solps-iter/modules/DivGeo/device/{DEVICE}/templates`, and linked to the `baserun/` directory.
We can now import these just produced `.tria` files into DivGeo, through through `File` → `Import` → `Template`, to see how the triangular EIRENE grid appears.


dg.exe 33341.dg (on tok01)
File Edit View Commands Variables Options Window Help
L: Examine M: Mark R: Zoom/Pan
1250
1000
750
500
250
0
-250
-500
-750
-1000
-1250
-1500
0 250 500 750 1000 1250 1500 1750 2000 2250 2500 2750 3000
OK

9 - Setting up the input parameters of B2.5

Now we can start to deal with the specification of the input parameters for **B2.5**, the fluid module of SOLPS-ITER.
The various input files for B2.5 are made of a number of **switches**, used for activating or disactivating some code functionalities or routines during a run.
In particular, the various switches overall present can be classified as:

- **Geometry switches**: these refer to the plasma geometry, which usually need to be set up only once for a family of runs.
- **Run switches**: these refer to the parameters of the code run, which are likely to change for every different run.
- **Physics switches**: these refer to the physics terms in the code, which are likely to be different within a family of runs.
- **Output switches**: these refer to the level and type of output.
- **Numerics switches**: these refer to the numerical treatment of equations.
- **Atomic physics switches**: these refer to the handling of atomic processes.

First of all, it is explained how to produce the start-up files for B2.5 by running each of the various subprograms.
In particular, each of the subprograms of B2.5 makes use of the **pre-processor input files** `b2ag.dat`, `b2ah.dat`, `b2ar.dat`, `b2ai.dat`, which have been created by Uinp. These are written in ASCII format and therefore are readable and editable, and their content says to the B2.5 programs how to work and with which inputs; compiling such files is then the actual **user input** to the B2.5 code. Among these inputs there are:

- Geometry characteristics of the considered case.
- Boundary conditions for the considered case.
- Physics assumptions about anomalous transport processes.
- Databases for atomic processes.

as well as:

- Settings regarding the numerical computation
- Specification about whether B2.5 must be run in stand-alone mode or coupled with EIRENE.

Therefore, starting a complete new run means first of all modifying or checking the pre-processor input files `b2ag.dat`, `b2ah.dat`, `b2ar.dat`, `b2ai.dat`, located in the `baserun/` directory, and then running the related **pre-processor programs** (`b2ag.exe`, `b2ah.exe`, `b2ar.exe`, `b2ai.exe`), in order to produce the files which will definitely be used as input by the main program. Initially they had been written by Uinp with some default content, which can be nevertheless modified according to the settings which we want to define for the B2.5 input files.
In the following pages the content of each of the four pre-processor input files is described.

The pre-processor input file **b2ag.dat** is used for setting up the **geometry** of the problem. Reading it, we find the strings:

- *dimens: contains four number specifying the **size of the grid** (e.g. 48 18 48 18). The first pair contains the number of cells in x direction (poloidal) and in y direction (radial), respectively. The second pair is the size of the grid that was originally created; each needs to be an integer multiple of the corresponding entry of the first pair.
- *param: we are only interested in the first number here, which must be -1.0 if we want to read the mesh data using the Carre format.
- 'b2agfs_geometry': specifies the .geo **mesh file to be read**, which will be looked in the run directory, in the /baserun directory and in the solps-iter/data/meshes directory.

Eventually we could add other strings, if we want to specify some geometry properties in a different way than the default one; these can be:

- 'b2agfs_nncut': integer, specifies the **number of topological cuts** in the geometry, that is corresponding to the x-point. For a single-null geometry, it is 1 by default.
- 'b2agfs_leftcut': integer, specifies the poloidal index of cells directly to the right of the left branch of the cut. For a single-null geometry and, e.g., a 48x18 grid, it is set to 12 by default.
- 'b2agfs_rightcut': integer, specifies the poloidal index of cells directly to the right of the right branch of the cut. For a single-null geometry and, e.g., a 48x18 grid, it is set to 36 by default.
- 'b2agfs_topcut': integer, specifies the radial index of cells directly above the cut. For a single-null geometry and, e.g., a 48x18 grid, it is set to 9 by default.
- 'b2agfs_bottomcut': integer, specifies the radial index of cells directly below the cut. For a single-null geometry and, e.g., a 48x18 grid, it is set to -1 by default.
- 'b2mwti_jxi': integer, specifies the cell index, on the basis mesh, of the **inner midplane**. For a single-null geometry, its default position in the mesh is given by the value jxi=leftcut+(rightcut-leftcut)/4, that is one quarter of the way between the two cuts in the computational domain. For a single-null geometry and, e.g., a 48x18 grid, will be then 12+(36-12)/4=18 by default.
- 'b2mwti_jxa': integer, specifies the cell index, on the basis mesh, of the **outer midplane**. For a single-null geometry, its default position in the mesh is given by the value jxa=rightcut-(rightcut-leftcut)/4, that is three quarters of the way between the two cuts in the computational domain. For a single-null geometry and, e.g., a 48x18 grid, will be then 36-(36-12)/4=30 by default.
- 'b2mwti_jsep': integer, specifies the cell index, on the basis mesh, of the last poloidal ring before the **separatrix**. For a single-null geometry, its default position in the mesh is given by the value jsep=topcut-1, that is three quarters of the way between the two cuts in the computational domain. For a single-null geometry and, e.g., a 48x18 grid, will be then 9-1=8 by default.

Finally, if we want to modify somehow the global magnetic properties with respect to the information contained in the considered .geo file, we could add the strings:

- 'b2agfs_Bt_rescale': **rescale factor for the magnetic field**. If activated, all components of the magnetic field in each cell are multiplied together by such factor, with respect to the original field magnitude. To be used if we want to simulate a different total magnetic field with respect to the one described in the .geo file.
- 'b2agfs_pit_rescale': **rescale factor for the magnetic field line pitch**. If activated, the poloidal field components in each cell are multiplied together by such factor, with respect to the original field magnitude. To be used if we want to simulate a different plasma current with respect to the one described in the .geo file.

Here is how the b2ag.dat file appears:

```
*dimens          (nx, ny, nx1, ny1; free format)
     48     18     48     18
*param           (param(0:99); free format)
 -1.0, 99*0.0
'b2agfs_min_pitch'    '0.01'
'b2agfs_geometry'       '33341.v001.geo'
```

The pre-processor input file **b2ah.dat** is used for setting up the **default physics parameters**. Reading it, we find the strings:

- *dimens: specifies the **number of charge states.**
- *label: speficies a **label** for the run.

and the blocks of strings:

- *b2cmpa: specifies some **basic parameters** for each species.
  - *specs: contains atomic charge, nuclear charge, atomic mass and atomic charge squared (one line per species).
- *b2cmpb: specifies types of various **boundary conditions**; the first strings specify the number of regions where boundary conditions will be specified, while then we find one block of strings for each of the region, containing the actual types of the boundary conditions which will be imposed by default for each species:
  - *cbsna: boundary conditions for density (one line per species).
  - *cbsmo: boundary conditions for parallel momentum (one line per species).
  - *cbshi: ion/neutral temperature/energy boundary condition (one line per species).
  - *cbshe: electron temperature/heat flux boundary condition.
  - *cbsch: boundary condition for the electric potential.
  - *cbrec: recycling coefficients (one line per species).
- *b2cmpt: specifies the values for the **anomalous transport coefficients** which will be imposed by default for each species.
  - *cfdf0: auxiliary transport quantity (one line per species).
  - *cfdna: anomalous density-driven particle diffusion coefficient (one line per species).
  - *cfdpa: anomalous pressure-driven particle diffusion coefficient (one line per species).
  - *cfvla: anomalous pinch velocity (one line per species).
  - *cfvsa: anomalous transverse momentum diffusivity (one line per species).
  - *cfhci: anomalous ion/neutral heat diffusivity (one line per species).
  - *cfhce: anomalous electron heat diffusivity.
  - *cfsig: diffusivity corresponding to the anomalous electrical conductivity.
  - *cfalf: diffusivity corresponding to the anomalous thermos-electric coefficient.
  - *cflim: coefficients for flux limiting.

Here is how the b2ah.dat file appears:

```
*dimens         (ns; free format)
      2
*label          (lblps, character*60; free format)
  'Default transport parameters for D, all charge states'
*b2cmpa         basic parameters
*specs    zamin    zamax       zn      am
   '0'     0.0      0.0      1.0      2.0
   '1'     1.0      1.0      1.0      2.0
*b2cmpb         boundary conditions
*cbregs
'south'     3    0.2500    0.7500    2.0000
'north'     1    2.0000
'west'      1    2.0000
'east'      1    2.0000
*region
 0  'south, private flux'
*cbsna     (0)       (1)       (2)       (3)       (4)       (5)       (6)       (7)
   '0'  0.00E+00  0.00E+00 -1.00E-02  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
   '1'  0.00E+00  0.00E+00 -1.00E-03  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
*cbsmo     (0)       (1)       (2)       (3)       (4)       (5)       (6)       (7)
   '0'  0.00E+00  0.00E+00  0.00E+00 -1.00E+10  0.00E+00  0.00E+00 -1.00E+00  0.00E+00
   '1'  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00 -1.00E+00  0.00E+00
*cbshi     (0)       (1)       (2)       (3)       (4)       (5)       (6)       (7)
   '0'  1.00E-18 -1.00E+02 -1.00E-01  0.00E+00  0.00E+00  0.00E+00  0.00E+00 -2.50E+00
   '1'  1.00E-18 -1.00E+02 -1.00E-02  0.00E+00  0.00E+00  0.00E+00  0.00E+00 -2.50E+00
*cbshe     (0)       (1)       (2)       (3)       (4)       (5)       (6)       (7)
  '-1'  1.00E-18 -1.00E+02 -1.00E-04  0.00E+00  0.00E+00  0.00E+00  0.00E+00 -2.50E+00
*cbsch     (0)       (1)       (2)       (3)       (4)       (5)       (6)       (7)
  '-1'  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
*cbrec     (0)       (1)       (2)       (3)       (4)       (5)       (6)       (7)
   '0'  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
   '1'  1.00E+00  3.00E-01  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
*cbmsa     (0)       (1)       (2)       (3)       (4)       (5)       (6)       (7)
  '-1'  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
*cbmsb     (0)       (1)       (2)       (3)       (4)       (5)       (6)       (7)
   '0'  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
   '1'  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
*region
 1  'south, core plasma'
*cbsna     (0)       (1)       (2)       (3)       (4)       (5)       (6)       (7)
   '0'  0.00E+00 -1.00E+10  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
   '1'  0.00E+00 -1.00E+10  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
*cbsmo     (0)       (1)       (2)       (3)       (4)       (5)       (6)       (7)
   '0'  0.00E+00  0.00E+00  0.00E+00 -1.00E+10  0.00E+00  0.00E+00  0.00E+00  0.00E+00
   '1'  0.00E+00  0.00E+00  0.00E+00 -1.00E+10  0.00E+00  0.00E+00  0.00E+00  0.00E+00
*cbshi     (0)       (1)       (2)       (3)       (4)       (5)       (6)       (7)
   '0'  4.00E+13 -1.00E+30  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
   '1'  4.00E+13 -1.00E+30  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
*cbshe     (0)       (1)       (2)       (3)       (4)       (5)       (6)       (7)
  '-1'  4.00E+13 -1.00E+30  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
*cbsch     (0)       (1)       (2)       (3)       (4)       (5)       (6)       (7)
  '-1'  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
*cbrec     (0)       (1)       (2)       (3)       (4)       (5)       (6)       (7)
   '0'  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
   '1'  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
*cbmsa     (0)       (1)       (2)       (3)       (4)       (5)       (6)       (7)
  '-1'  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
*cbmsb     (0)       (1)       (2)       (3)       (4)       (5)       (6)       (7)
   '0'  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
   '1'  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
*region
2  'south, private flux'
*cbsna     (0)       (1)       (2)       (3)       (4)       (5)       (6)       (7)
   '0'  0.00E+00  0.00E+00 -1.00E-02  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
   '1'  0.00E+00  0.00E+00 -1.00E-03  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
*cbsmo     (0)       (1)       (2)       (3)       (4)       (5)       (6)       (7)
   '0'  0.00E+00  0.00E+00  0.00E+00 -1.00E+10  0.00E+00  0.00E+00 -1.00E+00  0.00E+00
   '1'  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00 -1.00E+00  0.00E+00
*cbshi     (0)       (1)       (2)       (3)       (4)       (5)       (6)       (7)
   '0'  1.00E-18 -1.00E+02 -1.00E-01  0.00E+00  0.00E+00  0.00E+00  0.00E+00 -2.50E+00
   '1'  1.00E-18 -1.00E+02 -1.00E-02  0.00E+00  0.00E+00  0.00E+00  0.00E+00 -2.50E+00
*cbshe     (0)       (1)       (2)       (3)       (4)       (5)       (6)       (7)
  '-1'  1.00E-18 -1.00E+02 -1.00E-04  0.00E+00  0.00E+00  0.00E+00  0.00E+00 -2.50E+00
```

```
*cbsch     (0)       (1)       (2)       (3)       (4)       (5)       (6)       (7)
  '-1'  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
*cbrec     (0)       (1)       (2)       (3)       (4)       (5)       (6)       (7)
   '0'  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
   '1'  1.00E+00  3.00E-01  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
*cbmsa     (0)       (1)       (2)       (3)       (4)       (5)       (6)       (7)
  '-1'  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
*cbmsb     (0)       (1)       (2)       (3)       (4)       (5)       (6)       (7)
   '0'  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
   '1'  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
*region
 3  'north, SOL/DIV outer wall'
*cbsna     (0)       (1)       (2)       (3)       (4)       (5)       (6)       (7)
   '0'  0.00E+00  0.00E+00 -1.00E-02  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
   '1'  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
*cbsmo     (0)       (1)       (2)       (3)       (4)       (5)       (6)       (7)
   '0'  0.00E+00  0.00E+00  0.00E+00 -1.00E+10  0.00E+00  0.00E+00 -1.00E+00  0.00E+00
   '1'  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00 -1.00E+00  0.00E+00
*cbshi     (0)       (1)       (2)       (3)       (4)       (5)       (6)       (7)
   '0'  1.00E-18 -1.00E+02 -1.00E-01  0.00E+00  0.00E+00  0.00E+00  0.00E+00 -2.50E+00
   '1'  1.00E-18 -1.00E+02 -1.00E-02  0.00E+00  0.00E+00  0.00E+00  0.00E+00 -2.50E+00
*cbshe     (0)       (1)       (2)       (3)       (4)       (5)       (6)       (7)
  '-1'  1.00E-18 -1.00E+02 -1.00E-04  0.00E+00  0.00E+00  0.00E+00  0.00E+00 -2.50E+00
*cbsch     (0)       (1)       (2)       (3)       (4)       (5)       (6)       (7)
  '-1'  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
*cbrec     (0)       (1)       (2)       (3)       (4)       (5)       (6)       (7)
   '0'  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
   '1'  1.00E+00  3.00E-01  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
*cbmsa     (0)       (1)       (2)       (3)       (4)       (5)       (6)       (7)
  '-1'  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
*cbmsb     (0)       (1)       (2)       (3)       (4)       (5)       (6)       (7)
   '0'  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
   '1'  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
*region
 4  'west, target plate'
*cbsna     (0)       (1)       (2)       (3)       (4)       (5)       (6)       (7)
   '0'  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00 -1.00E+00
   '1'  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00 -1.00E+00
*cbsmo     (0)       (1)       (2)       (3)       (4)       (5)       (6)       (7)
   '0'  0.00E+00  0.00E+00  0.00E+00  0.00E+00 -1.00E+10  0.00E+00  0.00E+00  0.00E+00
   '1'  0.00E+00  0.00E+00  1.00E+10  0.00E+00 -1.00E+10  0.00E+00  0.00E+00  0.00E+00
*cbshi     (0)       (1)       (2)       (3)       (4)       (5)       (6)       (7)
   '0'  1.00E-18 -1.00E+02  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00 -1.50E+00
   '1'  1.00E-18 -1.00E+02  0.00E+00 -1.00E-01  0.00E+00  0.00E+00  0.00E+00 -1.50E+00
*cbshe     (0)       (1)       (2)       (3)       (4)       (5)       (6)       (7)
  '-1'  1.00E-18 -1.00E+02  0.00E+00 -1.00E-01  0.00E+00  0.00E+00  0.00E+00 -1.50E+00
*cbsch     (0)       (1)       (2)       (3)       (4)       (5)       (6)       (7)
  '-1'  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  1.00E+00  5.00E-01
*cbrec     (0)       (1)       (2)       (3)       (4)       (5)       (6)       (7)
   '0'  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
   '1'  1.00E+00  3.00E-01  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
*cbmsa     (0)       (1)       (2)       (3)       (4)       (5)       (6)       (7)
  '-1'  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
*cbmsb     (0)       (1)       (2)       (3)       (4)       (5)       (6)       (7)
   '0'  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
   '1'  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
*region
 5  'east, target plate'
*cbsna     (0)       (1)       (2)       (3)       (4)       (5)       (6)       (7)
   '0'  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00 -1.00E+00
   '1'  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00 -1.00E+00
*cbsmo     (0)       (1)       (2)       (3)       (4)       (5)       (6)       (7)
   '0'  0.00E+00  0.00E+00  0.00E+00  0.00E+00 -1.00E+10  0.00E+00  0.00E+00  0.00E+00
   '1'  0.00E+00  0.00E+00  1.00E+10  0.00E+00 -1.00E+10  0.00E+00  0.00E+00  0.00E+00
*cbshi     (0)       (1)       (2)       (3)       (4)       (5)       (6)       (7)
   '0'  1.00E-18 -1.00E+02  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00 -1.50E+00
   '1'  1.00E-18 -1.00E+02  0.00E+00 -1.00E-01  0.00E+00  0.00E+00  0.00E+00 -1.50E+00
*cbshe     (0)       (1)       (2)       (3)       (4)       (5)       (6)       (7)
  '-1'  1.00E-18 -1.00E+02  0.00E+00 -1.00E-01  0.00E+00  0.00E+00  0.00E+00 -1.50E+00
*cbsch     (0)       (1)       (2)       (3)       (4)       (5)       (6)       (7)
  '-1'  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  1.00E+00  5.00E-01
*cbrec     (0)       (1)       (2)       (3)       (4)       (5)       (6)       (7)
   '0'  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
   '1'  1.00E+00  3.00E-01  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
*cbmsa     (0)       (1)       (2)       (3)       (4)       (5)       (6)       (7)
```

```
  '-1'  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
*cbmsb     (0)       (1)       (2)       (3)       (4)       (5)       (6)       (7)
   '0'  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
   '1'  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
*b2cmpt  transport coefficients
*cfdf0     (0)       (1)       (2)       (3)       (4)       (5)       (6)       (7)
   '0'  1.00E+02  0.00E+00  1.00E+00  5.00E-20  1.50E-18  0.00E+00  0.00E+00  0.00E+00
   '1'  5.00E-01  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
*cfdna     (0)       (1)       (2)       (3)       (4)       (5)       (6)       (7)
   '0'  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
   '1'  0.00E+00  0.00E+00  1.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
*cfdpa     (0)       (1)       (2)       (3)       (4)       (5)       (6)       (7)
   '0'  0.00E+00  0.00E+00  1.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
   '1'  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
*cfvla     (0)       (1)       (2)       (3)       (4)       (5)       (6)       (7)
   '0'  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
   '1'  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
*cfvsa     (0)       (1)       (2)       (3)       (4)       (5)       (6)       (7)
   '0'  0.00E+00  0.00E+00  1.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
   '1'  0.00E+00  0.00E+00  4.00E-01  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
*cfhci     (0)       (1)       (2)       (3)       (4)       (5)       (6)       (7)
   '0'  0.00E+00  0.00E+00  1.40E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
   '1'  0.00E+00  0.00E+00  1.40E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
*cfhce     (0)       (1)       (2)       (3)       (4)       (5)       (6)       (7)
  '-1'  7.00E-01  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
*cfsig     (0)       (1)       (2)       (3)       (4)       (5)       (6)       (7)
  '-1'  1.00E-07  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
*cfalf     (0)       (1)       (2)       (3)       (4)       (5)       (6)       (7)
  '-1'  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
*cflim    cflme     cflmi     cflmv     cflal     cflab      ...
  '-1'  1.50E-01  1.50E-01  5.00E-01  5.00E-01  5.00E-01  0.00E+00  0.00E+00  0.00E+00
*endphy
```

The pre-processor input file **b2ar.dat** is used for setting up the **atomic processes rates**. Reading it, we find the strings:

- *tlohi: specifies the **range of temperatures** within which the rates must be read from atomic physics table, in eV.
- *nlohi: specifies the **range of densities** within which the rates must be read from atomic physics table, in m^{-3}.
- *numnuc: specifies the number of distinct isonuclear sequences to be considered.
- *nucspec: specifies nuclear charge, lowest charge state and highest charge state to be considered.
- *flag: specifies which **atomic physics database** to use. The choice is between "ADPAK", "STRAHL", "ADAS" and "AMDS". Each database contains rated for various process for each species, among which: **ionization rates** (both as functions of temperature at fixed densities and as functions of density at fixed temperatures), **recombination rates** (both as functions of temperature at fixed densities and as functions of density at fixed temperatures), etc.

Here is how the b2ar.dat file appears:

```
*tlohi          (tlo, thi; real, free format)
   1.0E-01   1.0E+04
*nlohi          (nlo, nhi; real, free format)
   1.0E+17   1.0E+22
*numnuc         (nnuc; integer, free format)
   1
*nucspec        (nz, izlo, izhi; integers, one triplet per line)
    1  0  1
*flag           ('adpak', 'strahl', 'adas' or 'amds', free format)
  'adas'
*tailep         (real, free format)
   0.08
```

The pre-processor input file **b2ai.dat** is used for setting up the **initial plasma state**. Reading it, we find the strings:

- *dimens: specifies the **number of charge states.**
- *label: specifies a **label** for the run.
- *specs: specifies **atomic charge**, **atomic mass** etc. for each charge state.
- *naini: specifies the **initial densities** for each of the charge states, in m^{-3}.
- *ttini: specifies the **initial ion and electron temperatures**, in eV.

Here is how the b2ai.dat file appears:

```
*dimens         (ns; free format)
      2
*label          (lblps, character*60; free format)
  'Flat initial state for D, all charge states'
*specs    zamin    zamax       zn      am
   '0'     0.0      0.0      1.0      2.0
   '1'     1.0      1.0      1.0      2.0
*naini          (na0(0:ns-1); free format)
   1.0E+12   1.0E+19
*ttini          (tt0; free format)
   1.0e2
```

At this point we can prepare the **start-up files** for B2.5 by using the information contained in pre-processor input files, by running, while still in the `/baserun` directory, the relative pre-processor programs.
The general way of running any of these programs is by running

- `gmake -f $SOLPSTOP/runs/Makefile b2f*****`

A useful shortcut is provided by the script **b2run**, which can be launched executing

- `b2run b2**`

which automatically performs the `gmake` operation.

The programs to be run for producing the start-up files are:

- **b2ag**, which prepares the **geometry file**, **`b2fgmtry`**, reading the pre-processor input file `b2ag.dat`. This is done by running
  - `gmake -f $SOLPSTOP/runs/Makefile b2fgmtry`

  or, best,
  - `b2run b2ag`
- **b2ah**, which prepares the **default physics file**, **`b2fpardf`**, reading the pre-processor input file `b2ah.dat`. This is done by running
  - `gmake -f $SOLPSTOP/runs/Makefile b2fpardf`

  or, best,
  - `b2run b2ah`
- **b2ar**, which prepares the **default atomic processes rates file**, **`b2frates`**, reading the pre-processor input file `b2ar.dat`. This is done by running
  - `gmake -f $SOLPSTOP/runs/Makefile b2frates`

  or, best,
  - `b2run b2ar`
- **b2ai**, which prepares the default **initial plasma state file**, **`b2fstati`**, reading the pre-processor input file `b2ai.dat`. This is done by running
  - `gmake -f $SOLPSTOP/runs/Makefile b2fstati`

  or, best,
  - `b2run b2ai`

So, after having checked that the pre-processor input files have the desired content, we enter:

- `rm b2ag.prt`
- `rm b2ah.prt`
- `rm b2ar.prt`
- `rm b2ai.prt`
- `b2run b2ag`
- `b2run b2ah`
- `b2run b2ar`
- `b2run b2ai`

Now, all the start-up for B2.5, i.e. `b2fgmtry`, `b2fpardf`, `b2frates`, `b2fstati`, are produced, and stored in the `/baserun` directory.
In particular, `b2fstati` will contain all the physical fields characterizing the plasma state with which we want to start a run, i.e. various blocks indicating the values of each physical variable inside each cell of the B2.5 mesh.
The initial state being created will be a homogeneous isothermal static plasma, which is very far from a realistic plasma edge solution. It is therefore always better to start a case using an already converged case of the same topology and containing the same main species. It is possible, by adding the switch `'b2aidr_read_b2fstate` in `b2ai.dat` and setting it to the value 1, to obtain a new initial plasma

state which will include the data from the already computed solution in a locally present final plasma state file, and to which are added fields corresponding to a flat state for new species declared here in the species list.
It is also fundamental that, for a run, these files are chronologically ordered in a precise way, so that to be read in the right order of precedence by B2.5. For doing this, at this point we go to the parent run directory (i.e. one level above the `/baserun` directory) and we execute the script

- `correct_baserun_timestamps`

This is how the `/baserun` directory should at this point.

{MODEL}.dg	**batch_av/**	param.dg@
{MODEL}.dgo	btor.dat	**run_av/**
{MODEL}.str	carre.dat	rzpsi.dat
{MODEL}.trg	carre.log	selptx.inf
{MODEL}.v001.geo@	carre.out	stra.dat@
{EQUILIBRIUM}.eq	curtrngl.dat	structure.dat
AMJUEL@	dg.dgo@	structures.dat
AMMONX@	dg.equ@	template.a001.tria.ogr@
{TEMPLATE}.ogr	dg.str@	template.g001.tria.ogr@
CARRE.HISTORY	dg.trg@	test.eir
H2VIBR@	eirtria.log	**tracing/**
HYDHEL@	fort.21@	traduit.log
METHANE@	fort.22@	tria.cms
PHOTON@	fort.29	tria.elemente
SPUTER@	fort.3	tria.eps
b2.user.parameters	fort.30	tria.in
b2ag.dat	fort.33@	tria.log
b2ag.log	fort.34@	tria.neighbor
b2ag.prt	fort.35@	tria.nodes
b2ah.dat	fort.78	tria.ps
b2ah.prt	fort.85	tria.ps.legend
b2ai.dat	frontier.dat	triageom.eps
b2ai.prt	gmeta	triageom.log
b2ar.dat	graphite_ext.dat@	triang.eir
b2ar.prt	input.dat@	triang.hed
b2fgmtry	input.eir	triangles.dat
b2fpardf	input_avg.eir	triplt.dat
b2frates	input_fin.eir	uinput.dg@
b2fstati	mesh.extra	uinput.trg@
b2yr.dat	mo_ext.dat@	vertex_data.out
b2yt.dat	ncar.cfg	weis.dat@

Now, we are in principle ready to start a run. For doing this, we can create, at the same level of the `/baserun` directory, a number of **run directories**, which will allow for different runs. The various run directories, in fact, can allow different variations to the default run parameters, to be set up inside each of these.
This means that, when performing a run, the code "looks" for certain information first in the actual run directory, and then, only if not found there, in the `baserun/` directory at the same level. Thus, the `baserun/` directory provides a set of default input parameters. contained in the start-up files produced so far, that would be shared between all different cases contained in the parent directory, each one represented by a different run directory.

After having created a run directory, first of all it is necessary to put here an initial plasma state file, which can be the one just created (i.e. flat) or one coming from an already converged run (more realistic).
Then, we must create here the **main user input file**, which we must call **`b2mn.dat`**. The information contained in it will determine the operating regime of the code, by means of declaring a value for a number of possible switches to include in it.

In addition to it, it is also possible to create, always in the run directory, a number of **additional user input files**, which can contain additional input general/geometry/physics/numerics data with respect to the ones contained in the just created startup files. In this way, if desired, we can tell the code to read the corresponding input data from these files, rather than from the startup files, through opportune switches in `b2mn.dat`.
These data belong to a number of different **namelists**, which could be either read or not read, depending if related activation switches are on, can be classified as:

- **Boundary namelist**, to be imposed in an input file which must be called **`b2.boundary.parameters`**, used to specify the boundary conditions for the run.
- **Transport namelist**, to be imposed in an input file which must be called **`b2.transport.parameters`**, used to choose between different anomalous transport models to use in the calculations.
- **Neutrals namelist**, to be imposed in an input file which must be called **`b2.neutrals.parameters`**, used to either set parameters for the B2.5 neutral fluid model or for the EIRENE neutral kinetic model.
- **Numerics namelist**, to be imposed in an input file which must be called **`b2.numerics.parameters`**, used to specify time multiplies per equation and per region for the equation solver of the code.
- **Source profiles namelist**, to be imposed in an input file which must be called **`b2.sources.profile`**, used to eventually set arbitrary profiles of external sources to use in the calculations.
- **Transport profiles namelist**, to be imposed in an input file which must be called **`b2.transport.inputfile`**, used to eventually set arbitrary profiles of anomalous transport coefficients to use in the calculations.

The namelists contained in such files are defined by numerical values (logical, integer or real) and can be written in the form of:

- single values.
- one value for each species (array of size `(0:NS-1)`).
- one value in each region (array of size `(0:NREG)`).
- one value for each species for each species in each region (array of size `(0:NS-1,0:NREG)`).

First of all, as already said, the main user input file **b2mn.dat** determines the **operating regime of the code**. It starts with a label containing the basic description of the case (device, discharge number, etc.). Then, it is possible to impose a great number of switch, that is:

- **Numerics and output** switches:
  - 'b2mndr_ntim': specifies the **number of time steps** to perform in the simulation (usually in the order $10^3/10^4$).
  - 'b2mndr_dtim': specifies the **duration of each time step** (usually the order $10^{-4}/10^{-3}$ s is good, even if, at least changing the most important parameters, we should be able to reach convergence with any time steps from 10^{-6} to 10^{-2} s).
  - 'b2mndr_cpu': specifies the **CPU limit**, the number of CPU-seconds after which of running time after which the calculations must be stopped; if it is 0, no limit is enforced.
  - 'b2mndr_savecpu': specifies the CPU-time interval, in CPU-seconds, after which save files plasmastate.xxxx, are written, useful for the recovery from an eventual system crash.
  - 'b2mndr_b2time': defines the frequency at which the various output scalar quantities are written out in the b2time.nc file, specifying the number of time steps between two writes.
  - 'b2mndr_tally': defines the frequency at which the various tallies are written out in the b2tallies.nc file, specifying the number of time steps between two writes.
  - 'balance_netcdf': if it is set to 1, the file balance.nc is created, containing all the arrays required by the balance post-processing routines in CDF format.
  - 'balance_average': if it is set to 1, the balance arrays written in balance.nc are averaged over all the performed timesteps in the run.
- **Run switches**:
  - 'b2mndr_eirene': turns on coupled runs with the EIRENE neutral code if it is 1, while runs B2.5 in standalone mode if it is 0.
  - 'b2mndr_rescale_neutrals_sources': multiplier to the neutral sources as computed by EIRENE or the corresponding rate coefficients.
  - 'b2mndr_rescale_neutrals': multiplier of the neutral density on the first timestep.
  - 'b2mndr_density_rescale': multiplier of all densities on the first timestep.
  - 'b2mwti_jxi': integer, specifies an arbitrary poloidal cell position defining the inner midplane.
  - 'b2mwti_jxa': integer, specifies an arbitrary poloidal cell position defining the outer midplane.
  - 'b2stbc_boundary_namelist': if it is set to 1 it will read input data from the file named b2.boundary.parameters instead that from b2ah.dat.
  - 'b2tqna_transport_namelist': if it is set to 1 it will read input data from the file named b2.transport.parameters instead that from b2ah.dat.
  - 'b2stbr_neutrals_namelist': if it is set to 1 it will read input data from the file named b2.neutrals.parameters instead that from other files.
  - 'b2srdt_numerics_namelist': if it is set to 1 it will read input data from the file named b2.numerics.parameters instead that from other files.
  - 'b2sral_inputfile': if it is set to 1 it will read data from the file named b2.sources.profile, containing arbitrary profiles of sources, instead that from other files.
  - 'b2tqna_inputfile': if it is set to 1 it will read data from the file named b2.transport.inputfile, containing arbitrary radial profiles of anomalous transport parameters, instead that from b2.transport.parameters.
  - 'b2stbc_feedback': default set to 0. If it is set to 1, it turns on the feedback mode for implementing boundary conditions.

- Feedback switches (to be considered when `'b2stbc_feedback'` is set to 1):
  - `'b2stbc_fheycore'`: imposes the value of electron heat flow radially entering from the core boundary.
  - `'b2stbc_fhiycore'`: imposes the value of ion heat flow radially entering from the core boundary.
  - `'b2stbc_fchycore'`: imposes the value of electric current radially entering from the core boundary.
  - `'b2stbc_nesepm'`: imposes the value of the outer midplane separatrix electron density, controlled through particle input of main plasma species through the core boundary.
  - `'b2stbc_nesepm_sol'`: imposes the value of the outer midplane separatrix electron density, controlled through a gas puff source outside the SOL region, that is through particle input of neutral species through the SOL north boundary.
  - `'b2stbc_nesepm_pfr'`: imposes the value of the outer midplane separatrix electron density, controlled through a gas puff source outside the PFR region, that is through particle input of neutral species through the private flux region boundary.
  - `'b2stbc_private_flux_puff'`: imposes the value of the gas puff particle flow through the private flux region boundary.
- Physics switches:
  - `'b2stbr_sput_phys`: if it is set to 1, it turns on physical sputtering.
  - `'b2stbr_sput_phys_model'`: if it is set to 0, the empirical formula for physical sputtering is used, while if it is set to 1 data are read from the TRIM tables.
  - `'b2stbr_sputter_energy_on'`: if it is set to 1, it turns on the energy contribution to the ion energy equation from the sputtered particles, while if it is set to 0 particles are sputtered cold.
  - `'b2tqce_model'`: model to be used for the classical parallel electron heat conductivity. If it is set to 1, it uses the Balescu formulation, while if it is set to 2, is uses the older Braginskii formulation.
  - `'b2tqca_model'`: model to be used for the classical parallel ion heat conductivity. If it is set to 1, it uses the Balescu formulation, while if it is set to 2, is uses the older Braginskii formulation.
  - `'b2tqna_ballooning'`: if it is set to 1, the ballooning model for transport coefficients is activated, rescaling all transport coefficients in each cells proportionally to the magnetic field, by a factor of `('b2tqna_ballooning_rescale')*abs(bb_ref/bb)`, where `bb_ref` is the arithmetic average of the total magnetic field intensity over the entire computational domain.
  - `'b2tqna_pfr_rescale'`: scaling factor for all electron and ion transport coefficients inside the private flux regions.
  - `'b2tqna_divsol_rescale'`: scaling factor for all electron and ion transport coefficients inside the divertor SOL regions.
  - `'b2sqcx_phm0'`: multiplier to the charge-exchange rate coefficients.
  - `'b2sqel_phm0'`: multiplier to the ionization rate coefficients.
  - `'b2sqel_phm1'`: multiplier to the recombination rate coefficients.

Here is how the b2mn.dat file should appear:

```
label          (lblmn, character*60; free format)
 'Arbitrary label for the case'
*b2cmpa         basic parameters
*b2cmpb         boundary conditions
*b2cmpt         transport coefficients
*cflim    cflme    cflmi    cflmv     ...[3:7 Not in use]
 ' -1'  3.00E-01  1.00E+01  5.00E-01  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00
*endphy
'b2mndr_ntim'                            '1000'
'b2mndr_dtim'                           '1.0e-4'
'b2mndr_cpu'                             '80000'
'b2mndr_savecpu'                         '3600'
'b2mndr_b2time'                             '1'
'b2mndr_tally'                              '1'
'tallies_netcdf'                            '1'
'balance_netcdf'                            '1'
'balance_average'                           '0'
'b2mndr_eirene'                             '1'
'b2mndr_rescale_neutrals_sources'    '1.0e-10'
'b2mndr_rescale_neutrals'                '1.0'
'b2mwti_jxi'                              '19'
'b2mwti_jxa'                              '26'
'b2stbc_boundary_namelist'                  '1'
'b2tqna_transport_namelist'                 '1'
'b2stbr_neutrals_namelist'                  '1'
'b2srdt_numerics_namelist'                  '1'
'b2sral_inputfile'                          '0'
'b2tqna_inputfile'                          '1'
'b2stbc_feedback'                           '1'
'b2stbc_nesepm_sol'                  '1.00e19'
'b2stbr_sput_phys'                       '1.0'
'b2stbr_sput_phys_model'                    '1'
'b2stbr_sputter_energy_on'                  '1'
'b2tqce_model'                              '1'
'b2tqca_model'                              '1'
'b2tqna_ballooning'                        '1.'
'b2tqna_ballooning_rescale'           '0.8345'
'b2tqna_pfr_rescale'                       '1.'
'b2tqna_divsol_rescale'                    '1.'
```

Then, at next pages follows a description of how the additional user input files, to be put optionally in the actual run directory, should be structured.

b2.boundary.parameters specifies the **boundary conditions** on each boundary segment present in the simulated region. Boundary conditions are actually read from this file, instead of being taken from the file created by the b2ah program, only if the switch 'b2stbc_boundary_namelist' is set to 1 in b2mn.dat. It contains the strings:

- NBC: specifies the integer **number of boundary segments**.
- bcchar: specifies the **nature of each of the boundary segments** (north, south, west, east).
- bcpos: for north/south boundaries it specifies the **row index**, while for west/east boundaries it specifies the **column index.**
- bcstart: for north/south boundaries it specifies the **start column index**, while for west/east boundaries it specifies the **start row index.**
- bcend: for north/south boundaries it specifies the **end column index**, while for west/east boundaries it specifies the **end row index.**
- bccon: integer array with length NS*NBC, specifies the type of **density boundary condition** for each species (first index) and for each segment (second index):
  - 0: prescribes no boundary condition.
  - 1: prescribes the value of the density, in m^{-3}, for conpar(,,1).
  - 2: prescribes the value of the density gradient, in m^{-4}, for conpar(,,1).
  - 3: prescribes the sheath conditions at the boundary, with conpar(,,1) not used.
  - 5: prescribes the value of the particle flux density, in $m^{-2}s^{-1}$, for conpar(,,1).
  - 6: prescribes the value of the particle flux, in s^{-1}, with constant density, for conpar(,,1).
  - 8: prescribes the value of the particle flux, in s^{-1}, with constant flux density, for conpar(,,1).
  - 9: prescribes the value of the decay length for the density, in m, for conpar(,,1).
  - 10: prescribes the leakage option for the density, with conpar(,,1) specifying the leakage factor.
  - 11: prescribes the feedback boundary condition for the particle flux, with conpar(,,1) not used.
  - 12: prescribes the feedback boundary condition for the particle flux density, with conpar(,,1) not used.
  - 20: prescribes the feedback boundary condition for a constant density, with conpar(,,1) specifying the density in m^{-3}.
- conpar: real array with length NS*NBC*3, specifies the real values for each species as each boundary segment for a quantity depending on the type of bccon, as above.
- bcmom: integer array with length NS*NBC, specifies the type of **parallel momentum or velocity boundary condition** for each species (first index) and for each segment (second index).
  - 0: prescribes no boundary condition.
  - 1: prescribes the value of the parallel velocity, in m/s, for mompar(,,1).
  - 2: prescribes the value of the parallel velocity gradient, in s^{-1}, for mompar(,,1).
  - 3: prescribes the sheath conditions at the boundary, with mompar(,,1) not used.
  - 5: prescribes the value of the parallel momentum flux density, in N/m^2, for mompar(,,1).
  - 8: prescribes a zero gradient for the Mach number, with mompar(,,1) not used.
  - 9: prescribes the value of the parallel momentum flux, in N, with constant flux density, for mompar(,,1).
  - 10: prescribes the value of the decay length for the parallel momentum, in m, for mompar(,,1).
  - 17: prescribes the leakage option for the parallel momentum, with mompar(,,1) sèecofying the leakage factor.

- mompar: real array with length NS*NBC*2, specifies the real values at each boundary segment for each species for a quantity depending on the type of bcmom, as above.

- bcene: integer array with length NBC, specifies the type of **electron energy/temperature boundary condition** for each segment (first index):
  - 0: prescribes no boundary condition.
  - 1: prescribes the value of the electron temperature, in eV, for enepar(,1).
  - 2: prescribes the value of the electron temperature gradient, in eV/m, for enepar(,1).
  - 3: prescribes the sheath conditions at the boundary, with enepar(,1) specifying the contribution to the energy transmission coefficient.
  - 5: prescribes the value of the electron energy flux density, in W/m^2, for enepar(,1).
  - 8: prescribes the value of the electron energy flux, in W, with constant flux density, for enepar(,1).
  - 9: prescribes the value of the decay length for the electron temperature, in m, for enepar(,1).
  - 12: prescribes the sheath conditions at the boundary, with enepar(,1) specifying an electron energy transmission factor.
  - 14: prescribes the leakage option for the electron energy, with enepar(,1) specifying the leakage factor.
  - 16: prescribes the feedback boundary condition at constant temperature, with enepar(,1) specifying the electron energy flux across the flux surface with index b2stbc_type16_ref, (to be specified in b2mn.dat) and with enepar(,2) specifying the strength of the feedback. Active if the switch b2stbc_bcene_16_style in b2mn.dat is 1.
- enepar: real array with length NS*NBC*2, specifies the real values at each boundary segment for a quantity depending on the type of bcene, as above.

- bceni: integer array with length NBC, specifies the type of **ion energy/temperature boundary condition** for each segment (first index):
  - 0: prescribes no boundary condition.
  - 1: prescribes the value of the ion temperature, in eV, for enipar(,1).
  - 2: prescribes the value of the ion temperature gradient, in eV/m, for enipar(,1).
  - 3: prescribes the sheath conditions at the boundary, with enipar(,1) specifying the contribution to the energy transmission coefficient.
  - 5: prescribes the value of the ion energy flux density, in W/m^2, for enipar(,1).
  - 8: prescribes the value of the ion energy flux, in W, with constant flux density, for enipar(,1).
  - 9: prescribes the value of the decay length for the ion temperature, in m, for enipar(,1).
  - 12: prescribes the sheath conditions at the boundary, with enipar(,1) specifying an ion energy transmission factor.
  - 14: prescribes the leakage option for the ion energy, with enipar(,1) specifying the leakage factor.
  - 16: prescribes the feedback boundary condition at constant temperature, with enipar(,1) specifying the ion energy flux across the flux surface with index b2stbc_type16_ref, (to be specified in b2mn.dat) and with enipar(,2) specifying the strength of the feedback. Active if the switch b2stbc_bceni_16_style in b2mn.dat is 1.
- enipar: real array with length NS*NBC*2, specifies the real values at each boundary segment for a quantity depending on the type of bceni, as above.

- `bcpot`: integer array with length `NBC`, specifies the type of **electric potential or current boundary condition** for each segment (first index):
  - 0: prescribes no boundary condition.
  - 1: prescribes the value of the potential, in V, for `potpar(,1)`.
  - 2: prescribes the value of the potential gradient, in V/m, for `potpar(,1)`.
  - 3: prescribes the sheath conditions at the boundary, with `potpar(,1)` not used.
  - 5: prescribes the value of the electric current flux density, in A/m^2, for `potpar(,1)`.
  - 8: prescribes the value of the electric current flux, in A, with constant flux density, for `potpar(,1)`.
  - 9: prescribes the value of the decay length for the potential, in m, for `potpar(,1)`.
  - 16: prescribes the feedback boundary condition at constant electric potential
- `potpar`: real array with length `NS*NBC*2`, specifies the real values at each boundary segment for a quantity depending on the type of `bcpot`, as above.
- `gammai`: specifies the real value of the **adiabatic coefficient** (ratio of specific heats, usually 1,667).
- `gammae`: specifies the **secondary electron emission coefficient** (usually 0,5).

Here is how the `b2.boundary.parameters` file should appear:

```
BOUNDARY
 NBC= 6,
 bcchar=   'S',  'W',  'E',  'S',  'S',  'N',
 bcpos=     -1,   -1,   48,   -1,   -1,   18,
 bcstart=   12,   -1,   -1,    0,   36,    0,
 bcend=     35,   18,   18,   11,   47,   47,
 bcene=      8,    3,    3,    9,    9,    9,
 bceni=      8,    3,    3,    9,    9,    9,
 bcpot=     12,    3,    3,   13,   13,   13,
 enepar(1,1)=  0.260e+06,  0.900    ,  0.900    ,  1.000e-02,  1.000e-02,  1.000e-02,
 enipar(1,1)=  0.260e+06,   2.50    ,   2.50    ,  0.100    ,  0.100    ,  0.100    ,
 enepar(1,2)=   100.    ,   0.00    ,   0.00    ,   100.    ,   100.    ,   0.00    ,
 enipar(1,2)=   100.    ,   0.00    ,   0.00    ,   100.    ,   100.    ,   0.00    ,
 potpar(1,1)=   0.00    ,   0.00    ,   0.00    ,   0.00    ,   0.00    ,   0.00    ,
 bccon(0,1)=   5,   5,
 bccon(0,2)=   5,   3,
 bccon(0,3)=   5,   3,
 bccon(0,4)=  10,  10,
 bccon(0,5)=  10,  10,
 bccon(0,6)=  11,  10,
 conpar(0,1,1)=   0.00    ,   0.00    ,
 conpar(0,2,1)=   0.00    ,   0.00    ,
 conpar(0,3,1)=   0.00    ,   0.00    ,
 conpar(0,4,1)= -1.000e-02, -1.000e-03,
 conpar(0,5,1)= -1.000e-02, -1.000e-03,
 conpar(0,6,1)= -1.000e-02, -1.000e-03,
 bcmom(0,1)=   2,   2,
 bcmom(0,2)=   1,   3,
 bcmom(0,3)=   1,   3,
 bcmom(0,4)=   8,   8,
 bcmom(0,5)=   8,   8,
 bcmom(0,6)=   8,   8,
 mompar(0,1,1)=   0.00    ,   0.00    ,
 mompar(0,2,1)=   0.00    ,   1.00    ,
 mompar(0,2,2)=   1.00    ,   1.00    ,
 mompar(0,3,1)=   0.00    ,   1.00    ,
 mompar(0,3,2)=   1.00    ,   1.00    ,
 mompar(0,4,1)=   0.00    ,   0.00    ,
 mompar(0,5,1)=   0.00    ,   0.00    ,
 mompar(0,6,1)=   0.00    ,   0.00    ,
 lcbs=  1,
 nniso= 0,
 gammai= 1.6666667, gammae= 0.5000000,
 lbndusr=f, lfeedback=f,
/
```

b2.transport.parameters specifies the various **anomalous transport coefficients** which must be used for each physical transport model. Such coefficients are actually are read from this file, instead of being taken from the file created by the b2ah program, only if the switch 'b2tqna_transport_namelist' is set to 1 in b2mn.dat. It contains the strings:

- flag_dna: integer number, specifies the type of scaling relation to use for the anomalous coefficient of the **density driven particle diffusion model**.
- parm_dna: specifies the real value of the anomalous coefficient of the density driven particle diffusion model, for each species.
- flag_dpa: integer number, specifies the type of scaling relation to use for the anomalous coefficient of the **pressure driven particle diffusion model**.
- parm_dpa: specifies the real value of the anomalous coefficient of the pressure driven particle diffusion model, for each species.
- flag_vla: integer number, specifies the type of scaling relation to use for the anomalous coefficient of the **radial pinch velocity model**.
- parm_vla: specifies the real value of the anomalous coefficient of the radial pinch velocity model, for each species.
- flag_vsa: integer number, specifies the type of scaling relation to use for the anomalous coefficient of the **viscosity model**.
- parm_vsa: specifies the real value of the anomalous coefficient of the viscosity model, for each species.
- flag_hce: integer number, specifies the type of scaling relation to use for the anomalous coefficient of the **electron thermal diffusivity model**.
- parm_hce: specifies the real value of the anomalous coefficient of the electron thermal diffusivity model, for each species.
- flag_hci: integer number, specifies the type of scaling relation to use for the anomalous coefficient of the **ion thermal diffusivity model**.
- parm_hci: specifies the real value of the anomalous coefficient of the ion thermal diffusivity model, for each species.
- flag_sig: integer number, specifies the type of scaling relation to use for the anomalous coefficient of the **electrical conductivity model**.
- parm_sig: specifies the real value of the anomalous coefficient of the electrical conductivity model, for each species.
- flag_alf: integer number, specifies the type of scaling relation to use for the anomalous coefficient of the **thermo-electric current model**.
- parm_alf: specifies the real value of the anomalous coefficient of the thermo-electric current model, for each species.

For each model we can set up a type of **scaling relation** for the related coefficient (0 = use value specified in b2ah.dat, 1 = use constant value specified here, 2,3,4 = use constant value specified here with different scaling relations, respectively scaled by $\frac{10^{20}}{n_e}$, scaled by $\frac{T_e}{16B}$, and flux-scaled).

Here is how the `b2.transport.parameters` file should appear:

```
&transport
 flag_dna=1, parm_dna=2*0.1,
 flag_dpa=0, parm_dpa=2*0,
 flag_vla=0, parm_vla=2*0,
 flag_vsa=0, parm_vsa=2*0,
 flag_hci=1, parm_hci=2*0.5,
 flag_hce=1, parm_hce=0.5,
 flag_sig=0, parm_sig=0,
 flag_alf=0, parm_alf=0,
 /
```

b2.neutrals.parameters specifies data and parameters regarding the **neutrals**, used either as parameters for the B2.5 neutral fluid model or for the EIRENE run. It contains a number of strings, among which:

- NSTRAI: specifies the integer **number of neutral sources**.
- crcstra: specifies the **nature of each of the boundary segments** (north, south, west, east), including also the columns for the **gas puff source** (C), the **volume recombination source** (V) and the **time-dependent source** (T).
- rcpos: for north/south boundaries it specifies the **row index**, while for west/east boundaries it specifies the **column index.**
- rcstart: for north/south boundaries it specifies the **start column index**, while for west/east boundaries it specifies the **start row index.**
- rcend: for north/south boundaries it specifies the **end column index**, while for west/east boundaries it specifies the **end row index.**
- recyc: specifies the **particle recycling coefficients** for each neutral source.
- userfluxparm: specifies the strength of the constant neutrals source by **gas puff**, in s^{-1}.

Here is how the b2.transport.parameters file should appear:

```
&NEUTRALS
NSTRAI=7,
crcstra=        'W',  'E',  'N',  'S',  'S',  'C',  'V',  'T',
rcpos=           -1,   48,   18,   -1,   -1,    0,    0,    0,
rcstart=          0,    0,    0,    0,   36,    0,    0,    0,
rcend=           17,   17,   47,   11,   47,    0,    0,    0,
species_start=    0,    0,    0,    0,    0,    0,    0,    0,
species_end=      1,    1,    1,    1,    1,    1,    1,    1,
eirene_step_cpu=  300.,
eirene_step_dt=   10.0,
recyc(0,1)=  2*1.00,
recyc(0,2)=  2*1.00,
recyc(0,3)=  2*1.00,
recyc(0,4)=  2*1.00,
recyc(0,5)=  2*1.00,
recyc(0,6)=  2*1.00,
recyc(0,7)=  2*1.00,
recyc(0,8)=  2*1.00,
userfluxparm(1,1)=  5*0, 0.0
userfluxparm(1,2)=  6*0,
chemical_sputter_yield(1)=  208*0.00,
chemical_sputter_yield(209)=    0.00,
chemical_sputter_yield(210)=    0.00,
chemical_sputter_yield(211)=    0.00,
chemical_sputter_yield(212)=    0.00,
chemical_sputter_yield(213)=    0.00,
chemical_sputter_yield(214)=    0.00,
fchar_chemical=0,
igass_chemical=0,
itsput_chemical=0,
issput_chemical=0,
l_neutrad=   1,
lmolscl=     1,
mlcmp(1,1)=  2,
lcns=        1,
ltns=    2,  3,
/
```

b2.numerics.parameters specifies the various setting for the **numerical computation**, regarding the equations to be solved in the code, that is **continuity equation** (for each species), **momentum equation** (for each species), **electron energy equation**, **ion energy equation**, **potential equation**. It contains the strings:

- dtco: real numbers, specifies the timestep multiplier for the continuity equation for each species in each region.
- dtmo: real numbers, specifies the timestep multiplier for the momentum equation for each species in each region.
- dtee: real numbers, specifies the timestep multiplier for the electron energy equation in each region.
- dtei: real numbers, specifies the timestep multiplier for the ion energy equation in each region.
- solveco: logical numbers, indicates whether the continuity equations for each species are to be solved in each region.
- solvemo: logical numbers, indicates whether the momentum equations for each species are to be solved in each region.
- solveee: logical numbers, indicates whether the electron energy equation is to be solved in each region.
- solveei: logical numbers, indicates whether the ion energy equations is to be solved in each region.
- solvepo: logical numbers, indicates whether the potential equation is to be solved in each region.

Here is how the b2.numerics.parameters file should appear:

```
&numerics
dtco(0,1)= 12*1.0, dtco(0,2)= 12*1.0, dtco(0,3)= 12*1.0, dtco(0,4)= 12*1.0,
dtmo(0,1)= 12*1.0, dtmo(0,2)= 12*1.0, dtmo(0,3)= 12*1.0, dtmo(0,4)= 12*1.0,
dtee(1)= 1.0, 1.0, 1.0, 1.0,
dtei(1)= 1.0, 1.0, 1.0, 1.0,
/
```

b2.transport.inputfile specifies an eventual arbitrary **radial variation of the anomalous transport coefficients**. This file is actually read only if the switch 'b2tqna_inputfile' is set to 1 in b2mn.dat.
Here there can be more blocks of values: each one starts with the string ndata(1,k,m)=N, defined by three integer numbers (the first one is always 1, the second one, k, is the index for the type of coefficient considered in this block, from 1 to 9, and the third one, m, is a species index, 0 for atoms and 1 for ions in a D only case). The second index must be set to:

- k=1 for a block defining the **anomalous particle density-driven diffusivity** profile.
- k=2 for a block defining the **anomalous particle pressure-driven diffusivity** profile.
- k=3 for a block defining the **anomalous ion thermal diffusivity** profile.
- k=4 for a block defining the **anomalous electron thermal diffusivity** profile.
- k=5 for a block defining the **x-component of the anomalous pinch velocity** profile.
- k=6 for a block defining the **y-component of the anomalous pinch velocity** profile.
- k=7 for a block defining the **anomalous viscosity** profile.
- k=8 for a block defining the **anomalous radial electrical conductivity** profile.
- k=9 for a block defining the **anomalous radial thermo-electric coefficient** profile.

So we can define different values for the considered coefficient for N different arbitrarily defined radial positions (at midplane), measured with respect to the separatrix (- sign for inside the separatrix, + sign for outside the separatrix).
Then, for each block, we have N lines of the kind tdata(i,j,k,m), corresponding to each radial position, in two columns: the first one defines the radial positions, the second one defines the relative radial distribution of the considered coefficient. So, we have: the index i defining the column in the block, with i=1 for the radial positions and i=2 for the values of the coefficients, the index j defining the row in the block, with j spanning from 1 to N, and the index k,m defining the considered coefficient and species for each block.

Here is how the b2.transport.inputfile file should appear:

```
&TRANSPORT
ndata(1, 1 , 1 )= 6 ,
tdata(1, 1 , 1 , 1 )= -4.00E-02 , tdata(2, 1 , 1 , 1 )= 0.50 ,
tdata(1, 2 , 1 , 1 )= -1.50E-02 , tdata(2, 2 , 1 , 1 )= 0.50 ,
tdata(1, 3 , 1 , 1 )= -1.40E-02 , tdata(2, 3 , 1 , 1 )= 0.05 ,
tdata(1, 4 , 1 , 1 )= 5.00E-03 , tdata(2, 4 , 1 , 1 )= 0.05 ,
tdata(1, 5 , 1 , 1 )= 5.50E-03 , tdata(2, 5 , 1 , 1 )= 0.50 ,
tdata(1, 6 , 1 , 1 )= 2.50E-02 , tdata(2, 6 , 1 , 1 )= 0.50 ,
ndata(1, 3 , 1 )= 6 ,
tdata(1, 1 , 3 , 1 )= -4.00E-02 , tdata(2, 1 , 3 , 1 )= 0.70 ,
tdata(1, 2 , 3 , 1 )= -1.50E-02 , tdata(2, 2 , 3 , 1 )= 0.70 ,
tdata(1, 3 , 3 , 1 )= -1.40E-02 , tdata(2, 3 , 3 , 1 )= 0.07 ,
tdata(1, 4 , 3 , 1 )= 5.00E-03 , tdata(2, 4 , 3 , 1 )= 0.07 ,
tdata(1, 5 , 3 , 1 )= 5.50E-03 , tdata(2, 5 , 3 , 1 )= 0.70 ,
tdata(1, 6 , 3 , 1 )= 2.50E-02 , tdata(2, 6 , 3 , 1 )= 0.70 ,
ndata(1, 4 , 1 )= 6 ,
tdata(1, 1 , 4 , 1 )= -4.00E-02 , tdata(2, 1 , 4 , 1 )= 0.70 ,
tdata(1, 2 , 4 , 1 )= -1.50E-02 , tdata(2, 2 , 4 , 1 )= 0.70 ,
tdata(1, 3 , 4 , 1 )= -1.40E-02 , tdata(2, 3 , 4 , 1 )= 0.07 ,
tdata(1, 4 , 4 , 1 )= 5.00E-03 , tdata(2, 4 , 4 , 1 )= 0.07 ,
tdata(1, 5 , 4 , 1 )= 5.50E-03 , tdata(2, 5 , 4 , 1 )= 0.70 ,
tdata(1, 6 , 4 , 1 )= 2.50E-02 , tdata(2, 6 , 4 , 1 )= 0.70 ,
no_pflux=.true.
/
```

b2.sources.profile specifies eventual arbitrary **radially dependent source profiles**. This file is actually read only if the switch 'b2sral_inputfile' is set to 1 in b2mn.dat.
Here there can be more blocks of values: each one starts with the string nsdata(1,k,m)=N, defined be three integer numbers (the first one is always 1, the second one, k, is the index for the type of source considered in this block, from 1 to 6, and the third one, m, is a species index, 0 for atoms and 1 for ions in a D only case). The second index is:

- k=1 for a block defining a **particle source** profile.
- k=2 for a block defining a **parallel momentum** source profile.
- k=3 for a block defining an **electron heat source** profile.
- k=4 for a block defining an **ion heat source** profile.
- k=5 for a block defining an **electric charge source** profile.
- k=6 for a block defining a **non-ambipolar electric particle source** profile.

So we can define different values for the strength of the considered source for N different arbitrarily defined radial positions (at midplane), measured with respect to the separatrix (- sign for inside the separatrix, + sign for outside the separatrix).
Then, for each block, we have N lines of the kind sdata(i,j,k,m), corresponding to each radial position, in two columns: the first one defines the radial positions, the second one defines the relative radial distribution of the considered source. So, we have: the index i defining the column in the block, with i=1 for the radial positions and i=2 for the values of the coefficients, the index j defining the row in the block, with j spanning from 1 to N, and the index k,m defining the considered source and species for each block.

So, before actually running a case, the file `b2mn.dat` and all the `b2.*` files could be modified inside each run directory, depending on eventual different input quantities or different physics assumptions which are desired to charecterize each different case. For doing this, before launching the run, it is necessary to open the files we want to modify with an appropriate editor, e.g. through

➢ `nedit "file_name"`

and then save after having done the desired modifications.

More information about B2.5 can be found in the manual; in particular:

- Appendix A.1: some general documentation about the B2.5 code.
- Appendix A.2: brief statement of the purpose of many subroutines of the B2.5 code.
- Appendix A.3: description of the variables used in the B2.5 code.
- Appendix A.4: information about internal parameters and switches of the B2.5 code.
- Appendix A.5: information about the namelists of the B2.5 code.

B2.5, in particular, works by calling, at appropriate computation times, many **subroutines**, which are source codes written in FORTRAN 90 language, and are then present in form of `.f` files, contained in the directory `solps-iter/modules/B2.5/builds/`.
The naming convention followed by some of the most important routines is:

- `b2a*`: **auxiliary programs**.
  - `b2ag*`: preparation of the geometry.
  - `b2ah*`: preparation of the default physics parameters.
  - `b2ai*`: preparation of the initial plasma state.
  - `b2ar*`: preparation of the atomic rate coefficients.
- `b2m*`: **core routines for the main program**.
  - `b2mndr`: main driver routine.
  - `b2mnds`: routine for initialization.
  - `b2mndt`: routine for time stepping.
  - `b2mw*`: routines for monitoring the calculations.
- `b2n*`: **driver routines for inner iterations**.
  - `b2news`: routines for controlling inner iterations.
  - `b2np*`: routines for the iterations of the individual equations.
- `b2s*`: **routines for evaluating source term coefficients**.
  - `b2si*`: routines for evaluating internal source terms.
  - `b2sp*`: routines for evaluating log-log linearized rate coefficients.
  - `b2sq*`: routines for evaluating rate coefficients.
  - `b2sr*`: routines for evaluating of modifying the total source term coefficients.
  - `b2st*`: routines for evaluating partial source term coefficients.
- `b2t*`: **routines for evaluating transport coefficients and fluxes**.
  - `b2tf*`: routines for evaluating fluxes.
  - `b2tl*`: routines for applying flux limits to transport coefficients.
  - `b2tq*`: routines for evaluating differential transport coefficients.
  - `b2tr*`: routines for evaluating cell-integrated transport coefficients.
- `b2y*`: **post-processing routines**.
  - `b2yg*`: routines for displaying the geometry and magnetic field.
  - `b2yh*`: routines for displaying the table of physics parameters.
  - `b2yi*`: routines for displaying the plasma state.
  - `b2ym*`: routines for displaying the evolution of the plasma state.
  - `b2yn*`: routines for displaying the progress of the inner iterations.
  - `b2yp*`: routines for a comprehensive postprocessing of calculations.
  - `b2yq*`: routines for displaying the evolution of the calculations.
  - `b2yr*`: routines for displaying the table of atomic rate coefficients.
- `b2ys*`: routines for displaying the externally imposed source terms.

10 - The EIRENE input file

It is now explained how the input data of **EIRENE**, in case of a coupled run, are specified. This is done through the only EIRENE input file, called **`input.dat`**, which is a human-readable formatted file.
According to the specifications about plasma species, atomic/molecular processes, material surfaces data etc. which had been set up in DivGeo, a first tentative input file called `input.eir`, was already created by Uinp, and located in the the `baserun/` directory. First of all we have to copy it in the actual run directory with its final name, and then eventually we could modify it, or add features which had been not included in the DivGeo step.

The file contains a large number of lines, subdivided in 14 **blocks**, each one starting with the string
`*** n. "name_block"`
where n is the number of the considered block, from 1 to 14; then, some blocks are also subdivided into sub-blocks, each one starting with the string
`** n. "name_sub-block"`
Instead, a string starting with `*` is recognized as a comment line, i.e. it does not contain any input data.
The content of each block is here summarized:

- 1: **input data for operating mode**
- 2: **input data for standard mesh surfaces**
- 3: **input data for non-default standard surfaces and additional surfaces**
- 4: **input data for species specification and atomic physics module**
- 5: **input data for plasma background**
- 6: **input data for surface interaction models**
- 7: **input data for initial distribution of test particles**
- 8: **additional data for some specific mesh zones**
- 9: **data for statistics and non-analog methods**
- 10: **data for additional volumetric and surface averaged tallies**
- 11: **data for numerical and graphical output**
- 12: **data for plasma diagnostic module**
- 13: **data for nonlinear and time-dependent mode**
- 14: **data for interfacing with external code**

It should be noted that, in `input.dat`, for specifying values for the various quantities, cgs units are used, with two exceptions: all temperature and energies are in eV, while all particle fluxes are in ampere (i.e. in $\#/s \cdot 1{,}602 \cdot 10^{-19}$). Then, all geometrical data lengths are in cm (so that densities are in cm^{-3}, velocities are in cm/s, etc.).
For the various variables to be specified the standard Fortran naming conventions are used, i.e. variables starting with the letters `I`, `J`, `K`, `L`, `M`, `N` are integers, while all the other variables are real.

In the following only the most relevant parts of the input files are described. For a full description see the EIRENE manual.

The **block 1** contains the main input data for controlling the general **operating mode** of EIRENE. The variables in this block control some of the more general options of EIRENE such as overall running time, usage of dump files but also some parameters depending on the simulation model.
The form of the block is:

```
*** 1. Data for operating mode
NPRLL  NMODE  NTCPU     NFILE      NITER0  NITER   NTIME0 NTIME
NOPTIM NOPTM1 NGEOM_USR NCOUP_INPUT NSMSTRA NSTORAM NGSTAL NRTAL NREAC_ADD
NLSCL    NLTEST   NLANA    NLDRFT   NLCRR
NLERG    NLIDENT  NLONE    NLMOVIE  NLDFST
NLOLDRAN NLCASCAD NLOCTREE NLWRMSH
CFILE DBHANDLE DBFNAME
```

The first two rows contains integer flags, controlling some **global options** for the run. The meaning of the various flags is:

- NPRILL: specifies various options for the operated parallelization mode. Default: 1.
- NMODE: specifies the operating mode. It must be >0 for working in coupled mode with B2.5.
- NTCPU: specifies the maximum number of CPU-seconds allowed for a run.
- NFILE: 5-digits integer, regulates the reading/writing of the dump files.
- NITER0: specifies the initial iteration number. Default: 1.
- NITER: specifies the number of iterations, if EIRENE is run in iterative mode.
- NTIME0: specifies the initial time-cycle number. Default: 1.
- NTIME: specifies the total number of iterations in time carried out in one single run.
- NOPTIM: default: -1.
- NOPTIM1: default: 1.
- NGEOM_USR: regulates the use of external geometry packages. Default: 0.
- NCOUP_INPUT: regulates the storage for data transfer via coupling routines. Default: 1.
- NSMSTRA: "sum over strata" disabled for 0, enabled for 1.
- NSTORAM: regulates the storage vs. speed in atomic data evaluation. Minimum storage, slowest computation: 0. Maximum storage, fastest computation: 9.
- NGSTAL: regulates the storage of the surface tallies on non-default standard surfaces. Default: 1.
- NRTAL: regulates the condensation of the mesh cells into fewer larger cells, for output volume tallies. Default: 0.
- NREAC_ADD: default: 1.

The successive rows contain some **global logical switches**. The meaning of the various switches is:

- NLSCL: some volume averaged tallies are re-scaled in order to exactly preserve the total number of particles. Default: T.
- NLTEST: Tests for consistency between cell numbers and geometrical data along the particle tracks which are carried out at each point of collision. Default: F.
- NLANA: De-activates all non-analog sampling distributions. Default: F
- NLDRFT: The assumed distribution in velocity space of the bulk ions is a drifting Maxwellian for volumetric background tallies of bulk particles. Default: T
- NLCRR: Correlated sampling is used. Default: F
- NLERG: The case is automatically reduced to a case for estimating cell volumes by using an "ergodic property". Default: F
- NLIDENT: Forces all processors working on the same stratum to use the same sequence of random numbers and hence to carry out exactly identical work. Default: F

- NLONE: The case is automatically reduced to a single species and "one speed transport" problem. Default: F
- NLMOVIE: Reset a number of model parameters to enable a series of geometry plots for making particle trajectory movies. Default: F
- NLDFST: Used for testing time step options in the Fokker-Planck solver. Default: F

The other switches, then, are for internal use only.

Finally, we find several pathways to **external databases** (such as AMJUEL, HYDHEL, etc.), whose presence is identified by the starting character string CFILE.

- DBHANDLE: Name of the database which is specified in each line.
- DBFNAME: Absolute or relative path of the datafile to be used for this database.

Here is how the block 1 should appear:

```
*** 1. Data for operating mode
     2     1   300 30000     1     1     0     1     0
    -1     1     0     1     1     9     1     0     1
TFFTF FFFFF FFFTF F
CFILE AMJUEL Database/AMdata/amjuel.tex
CFILE HYDHEL Database/AMdata/hydhel.tex
CFILE SPUTER Database/Surfacedata/SPUTER
CFILE TRIM   Database/Surfacedata/TRIM/trim.dat
CFILE gr_ext Database/graphite_ext.dat
CFILE mo_ext Database/mo_ext.dat
```

The **block 2** contains the input data for the **standard mesh surfaces**, i.e. regarding the section of the EIRENE mesh which is included in the B2.5 mesh. It consists of 5 sub-blocks:

- 2a: data about radial coordinate surfaces.
- 2b: data about poloidal grid surfaces.
- 2c: data about toroidal grid surfaces.
- 2d: data about mesh multiplication.
- 2e: data about additional cells outside the standard mesh.

The form of the block is:

```
*** 2. Data for standard mesh
INDGRD(1) INDGRD(2) INDGRD(3)
** 2a. Data for x- or radial co-ordinate surfaces
NLRAD
IF NLRAD=T then
     NLSLB  NLCRC  NLELL  NLTRI  NLPLG  NLFEM  NLTET  NLGEN
     NR1ST  NRPLG  NPPLG  NRKNOT NCOOR
ENDIF
IF INDGRD(1)<=5 THEN
     XPCOR YPCOR ZPCOR
ENDIF
** 2b. Data for y- or poloidal grid surfaces
NLPOL
IF NLPOL=T then
     NLPLY  NLPLA  NLPLP
     NP2ND  NPSEP  NPPLA  NPPER
ENDIF
IF INDGRD(2)<=5 THEN
     YIA YGA YAA YYA
ENDIF
** 2c. Data for z- or toroidal grid surfaces
NLTOR
IF NLTOR=T then
     NLTRZ  NLTRA  NLTRT
     NT3RD  NTSEP  NTTRA  NTPER
ENDIF
IF INDGRD(3)<=5 THEN
     ZIA ZGA ZAA ZZA ROA
ENDIF
** 2d. Data for mesh multiplication
NLMLT
IF NLMLT=T then
     NBMLT
     IF NBMLT>1 then
          VOLCOR
     ENDIF
ENDIF
** 2e. Data for additional cells outside standard mesh
NLADD
IF NLADD=T then
     NRADD
     IF NRADD>0 then
          VOLADD
     ENDIF
ENDIF
```

First of all, INDGRD(1), INDGRD(2), INDGRD(3) are indexes which control the meaning of the input variables of different standard grid options, regarding the radial, poloidal and toroidal direction respectively.

If NLRAD=T, then a **radial grid** is defined. In this case, then we have two rows of cards.
The first row contains some logical switches:

- NLSLB: cartesian geometry.
- NLCRC: cylindrical or toroidal geometry.
- NLELL: mesh of nested elliptical flux surfaces.
- NLTRI: to be written.
- NLPLG: mesh in the (x,y) plane described by polygonal arcs.
- NLFEM: mesh in the (x,y) plane consists of triangles.
- NLTET: 3D discretization of volume by tetrahedrons.
- NLGEN: arbitrary geometrical configuration.

The second row contains some integer flags:

- NR1ST: numbers of polygons for discretization in radial direction.
- NRPLG: number of points per polygon.
- NPPLG: number of valid parts on each polygon.
- NRKNOT: ...
- NKCOOR: ...

Finally, if INDGRD(1) is less than 5, we have a row of three real numbers, defining an eventual shift of the whole mesh in the (x,y) plane.

If NLPOL=T, then a **poloidal grid** is defined. In this case, then we have two rows of cards.
The first row contains some logical switches:

- NLPLY: ...
- LNPLA: ...
- NLPLP: ...

The second row contains some integer flags:

- NP2ND: ...
- NPSEP: ...
- NPPLA: ...
- NPPER: ...

Finally, if INDGRD(2)is less than 5, we have a row of four real numbers.

If NLTOR=T, then a **toroidal grid** is defined. In this case, then we have two rows of cards.
The first row contains some logical switches:

- LNTRZ: the cylindrical approximation is used.
- LNTRA: the toroidal approximation is used.
- LNTRT: it is made use of toroidal coordinates.

The second row contains some integer flags:

- NT3RD: ...
- NTSEP: ...
- NTTRA: ...
- NTPER: ...

Finally, if INDGRD(3) is less than 5, we have a row of five real numbers.

The fourth sub-block is usually irrelevant.

Finally, if NLADD=T, then there are cells in the computational volume which are defined through additional surfaces as cell boundaries. The integer flag NRADD defines the number of additional zones, while the real number VOLADD defines the volume, in cm^{-3}, of each additional zone as seen by the test particles.

Here is how the block 2 should appear:

```
*** 2. Data for standard mesh
     6     1     1
** 2a. Data for x- or radial co-cordinate surfaces
T
FFFFT FFF
    19     1    51     3     0
 0.00000E+00 0.00000E+00 0.00000E+00
** 2b. Data for y- or poloidal grid surfaces
T
FFF
    51     0     0     0
 0.00000E+00 1.00000E+00 0.00000E+00 0.00000E+00
** 2c. Data for z- or toroidal grid surfaces
F
TFF
     0     0     0     0
 0.00000E+00 0.00000E+00 1.06500E+03 0.00000E+00 0.00000E+00
** 2d. Data for mesh multiplication
F
** 2e. Data for additional cells outside standard mesh
T
     1
 1.00000E+00
```

The **block 3a** contains the input data for the **non-default standard surfaces**, i.e. the boundaries of the EIRENE mesh which coincide with the external boundaries of the B2.5 mesh. In particular, the number of non-default standard surfaces is described by the variable NSTSI. For a single-null geometry, e.g., we have **core side**, **inner target**, **outer target**, **inner PFR**, **outer PFR**, **main chamber wall**.
The form of the block is:

```
*** 3a. Data for non-default standard surfaces                    NSTSI
NSTSI                                                             times
-----------------------------------------------------------------------
* TXTSFL
ISTS   IDIMP  INUMP  IRPTA1 IRPTE1 IRPTA2 IRPTE2 IRPTA3 IRPTE3
ILIIN  ILSIDE ILSWCH ILEQUI ILTOR  ILCOL  ILFIT  ILCELL ILBOX  ILPLG
```

TXTSFL is a text to characterize each particular surface.
Then, for each surface, we have two strings of integer flags. The first string identifies the **position of the surface** in the standard mesh:

- ISTS: labelling index (i.e. from 1 to NSTSI).
- IDIMP: identifies the mesh from which the particular surface is taken. 1 stands for x-directed standard mesh (e.g. core side, PFRs, main chamber wall), 2 stands for y-directed standard mesh (e.g. targets), 3 stands for z-directed standard mesh.
- INUMP: identifies the number of the considered surface in the x-directed, y-directed or z-directed standard mesh.
- IRPTAs, IRPTEs: ...

The second string, instead, identifies the **properties of the surface**:

- ILIIN: defines the type of surface.
  >0 stands for non-transparent surface. In particular, 1 stands for reflecting or partly absorbing surface, 2 stands for purely absorbing surface.
  ≤0 stands for transparent surface.
- ILSIDE: 0 means that both sides of the surface act as described by ILIIN, 1 means that particles incident on the surface in the negative direction will be absorbed.
- ILSWCH: 0 by default.
- ILEQUI: 0 by default.
- ILTOR: 0 by default.
- ILCOL: 1 by default
- ILFIT: 0 by default.
- ILCELL: 0 by default.
- ILBOX: 0 by default.
- ILPLG: identifies closed polygonal lines composed of additional surfaces (for each surface it is the same number specified as index in the "TRIA-EIRENE parameters" options in DivGeo).

Most of these surfaces are not actual material surfaces. If instead it is the case, as for example for the target surfaces, it must be linked to the block 6 which assigns the surface interaction properties. The presence of such link is identified by the string SURFMOD followed by a name {MODNAME} identifying the surface.

Here is how the block 3a should appear:

```
*** 3a. Data for non-default standard surfaces
     6
*   1 : Core side    1
     1     1     1    14    38     0     0     0     0
     2     2     0     0     0     6     0     0     0     0
*   2 : Inner Target
     2     2     1     1    19     0     0     0     0
     1     2     0     0     0     6     0     0     0     0
SURFMOD_TARGETS
*   3 : Outer Target
     3     2    51     1    19     0     0     0     0
     1     2     0     0     0     6     0     0     0     0
SURFMOD_TARGETS
*   4 : Inner PFR
     4     1     1     1    13     0     0     0     0
    -3     0     0     0     0     6     0     0     0    10
*   5 : Outer PFR
     5     1     1    39    51     0     0     0     0
    -3     0     0     0     0     6     0     0     0    10
*   6 : Main Chamber Wall
     6     1    19     1    51     0     0     0     0
    -3     0     0     0     0     6     0     0     0    10
```

The **block 3b** contains the input data for the **additional surfaces**, i.e. all the extra physical boundary surfaces for the EIRENE mesh (including the actual material surfaces, e.g. the vessel wall and components). In particular, the number of additional surfaces is described by the variable NLIMI.
In this block it is possible to assign the geometric properties and the general material properties of each surface; in addition, to each of these it is possible to insert a link to the successive block 6, in which the material surfaces can be specified in more detail (e.g. type of material, temperature, reflection properties, absorbing properties, eventual sputtering model, etc.) for a large group of surfaces.
The form of the block is:

```
*** 3b. Data for additional surfaces
NLIMI
------------------------------------------------------------------
* TXTSUR                                                          }
RLBND RLARE RLWMN RLWMX                                          }  NLIMI
ILIIN  ILSIDE ILSWCH ILEQUI ILTOR  ILCOL  ILFIT ILCELL ILBOX ILPLG }  times
XLIMS1 YLIMS1 ZLIMS1 XLIMS2 YLIMS2 ZLIMS2                        }
------------------------------------------------------------------
```

TXTSUR is a text to characterize each particular surface.
Then, for each surface, we have three strings. The first one contains four real numbers:

- RLBND: flag for different options to define the boundary of the surface.
  If it is 0, then no boundary inequalities are specified, i.e. the whole surface is seen by the test particles.
  If it is 1, then only part of the surface, which lies inside the right parallelepiped defines by the vectors (XLIMS1,YLIMS1,ZLIMS1) and (XLIMS2,YLIMS2,ZLIMS2) are seen by the particles.
- RLARE: area (in cm2) of the surface element which his seen by the test particles.
- RLWMN: lower weight limit for space weigh windows for particles crossing the surface in positive direction.
- RLWMX: upper weight limit for space weigh windows for particles crossing the surface in positive direction.

The second string, instead, contains integers and, in the same way as in the block 3a, identifies the **properties of the surface**:

- ILIIN: defines the type of surface.
  > 0 stands for non-transparent surface. In particular, 1 stands for reflecting or partly absorbing surface, 2 stands for purely absorbing surface.
  ≤ 0 stands for transparent surface.
- ILSIDE: 0 means that both sides of the surface act as described by ILIIN, 1 means that particles incident on the surface in the negative direction will be absorbed.
- ILSWCH: 0 by default.
- ILEQUI: 0 by default.
- ILTOR: 0 by default.
- ILCOL: 1 by default
- ILFIT: 0 by default.
- ILCELL: 0 by default.
- ILBOX: 0 by default.
- ILPLG: identifies closed polygonal lines composed of additional surfaces (for each surface it is the same number specified as index in the "TRIA-EIRENE parameters" options in DivGeo).

Finally, we find the vectors (XLIMS1,YLIMS1,ZLIMS1) and (XLIMS2,YLIMS2,ZLIMS2) which define the **extension of the surface**. In particular, on a poloidal cross section, XLIMS1, XLIMS2 and YLIMS1, YLIMS2 represent, respectively, the R coordinates and the Z coordinates of the boundary points of the segment representing the surface.

Normally, the surfaces included in this block are all material or pumping surfaces. So, each of these must be linked to the block 6 which assigns the surface interaction properties. Again, the presence of such link is identified by the string SURFMOD followed by a name {MODNAME} identifying the surface.

Here is how the block 3b should appear:

```
*** 3b. Data for non-default additional surfaces
   12
*   1 :   1
 2.00000E+00 1.00000E+00 1.00000E+00 1.00000E-05
     1     1     0     0     0     1     0     0     0    10
 1.23460E+02-9.78900E+01-1.00000E+20 1.23460E+02-8.25400E+01 1.00000E+20
SURFMOD_1_TUNGSTEN_TARGET
*   2 :   2
 2.00000E+00 1.00000E+00 1.00000E+00 1.00000E-05
     1     1     0     0     0     1     0     0     0    10
 1.23460E+02-8.25400E+01-1.00000E+20 1.21550E+02-8.24400E+01 1.00000E+20
SURFMOD_1_TUNGSTEN_TARGET
*   3 :   3
 2.00000E+00 1.00000E+00 1.00000E+00 1.00000E-05
     1     1     0     0     0     1     0     0     0    10
 1.21550E+02-8.24400E+01-1.00000E+20 1.11250E+02-6.63500E+01 1.00000E+20
SURFMOD_2_IRON
*   4 :   4
 2.00000E+00 1.00000E+00 1.00000E+00 1.00000E-05
     1     1     0     0     0     1     0     0     0    10
 1.11250E+02-6.63500E+01-1.00000E+20 1.13170E+02-6.51600E+01 1.00000E+20
SURFMOD_2_IRON
*   5 :   5
 2.00000E+00 1.00000E+00 1.00000E+00 1.00000E-05
     1     1     0     0     0     1     0     0     0    10
 1.13170E+02-6.51600E+01-1.00000E+20 1.13690E+02-6.51300E+01 1.00000E+20
SURFMOD_3_TUNGSTEN
*   6 :   6
 2.00000E+00 1.00000E+00 1.00000E+00 1.00000E-05
     1     1     0     0     0     1     0     0     0    10
 1.13690E+02-6.51300E+01-1.00000E+20 1.14310E+02-6.55700E+01 1.00000E+20
SURFMOD_3_TUNGSTEN
*   7 :   7
 2.00000E+00 1.00000E+00 1.00000E+00 1.00000E-05
     1     1     0     0     0     1     0     0     0    10
 1.14310E+02-6.55700E+01-1.00000E+20 1.18070E+02-7.16100E+01 1.00000E+20
SURFMOD_4_TURBOPUMP
*   8 :   8
 2.00000E+00 1.00000E+00 1.00000E+00 1.00000E-05
     1     1     0     0     0     1     0     0     0     0
 1.18070E+02-7.16100E+01-1.00000E+20 1.18360E+02-7.18900E+01 1.00000E+20
SURFMOD_4_TURBOPUMP
*   9 :   9
 2.00000E+00 1.00000E+00 1.00000E+00 1.00000E-05
     1     1     0     0     0     1     0     0     0     0
 1.18360E+02-7.18900E+01-1.00000E+20 1.24860E+02-8.21000E+01 1.00000E+20
SURFMOD_5_CRYOPUMP
*  10 :  10
 2.00000E+00 1.00000E+00 1.00000E+00 1.00000E-05
     1     1     0     0     0     1     0     0     0     0
 1.24860E+02-8.21000E+01-1.00000E+20 1.27180E+02-8.63200E+01 1.00000E+20
SURFMOD_5_CRYOPUMP
*  11 :  11
 2.00000E+00 1.00000E+00 1.00000E+00 1.00000E-05
     1     1     0     0     0     1     0     0     0     0
 1.27180E+02-8.63200E+01-1.00000E+20 1.28600E+02-9.02900E+01 1.00000E+20
SURFMOD_6_CONDUCTANCE
*  12 :  12
 2.00000E+00 1.00000E+00 1.00000E+00 1.00000E-05
     1     1     0     0     0     1     0     0     0     0
 1.28600E+02-9.02900E+01-1.00000E+20 1.29300E+02-9.44500E+01 1.00000E+20
SURFMOD_6_CONDUCTANCE
```

The **block 4** contains the input data for **species** and **atomic physics**, i.e. information about the considered atomic species, molecular species, test ion species and atomic/molecular processes. In particular, the number of considered atomic species is described by the variable NATM; the number of considered molecular species is described by the variable NMOL; the number of considered test ion species is described by the variable NION; the number of considered atomic/molecular processes is described by the variable NREAC. Masses assigned to the particle species are in units of the proton mass (i.e. 1,0073 atomic mass units).
The form of the block is:

```
*** 4. Data for species and atomic physics module
** Reactions
NREAC
IR FILNAM H123 REAC CRC MASSP MASST DP RMN1 RMX1          } NREAC times
** 4a. Neutral atom species                                              NATM
NATM                                                                     times
ISPZ TEXT NMASS NCHAR NPRT NCHRG ISRF ISRT ID1 NRC NFOL NGEN NHSTS ID3   }
** 4b. Neutral molecule species                                          NMOL
NMOL                                                                     times
ISPZ TEXT NMASS NCHAR NPRT NCHRG ISRF ISRT ID1 NRC NFOL NGEN NHSTS ID3   }
** 4c. Test ion species                                                  NION
NION                                                                     times
ISPZ TEXT NMASS NCHAR NPRT NCHRG ISRF ISRT ID1 NRC NFOL NGEN NHSTS ID3   }
```

First of all, it is described which **atomic/molecular processes** are activated, and the specification for reading the corresponding atomic/molecular data.
Atomic/molecular processes are always in the following type:

$$a \cdot A + b \cdot B \rightarrow m \cdot M + n \cdot N + \Delta E$$

Here, A, B are the labels for the pre-collision particles, while M, N are the labels for the post-collision particles, and ΔE is the amount of energy which is transferred into (or at the expense of) the kinetic energy of the reaction products.
We have the following meaning for the various inputs:

- IR: labelling index.
- FILNAM: name of the dataset which contains the required reaction (which could be then HYDHEL, AMJUEL, METHANE, etc.).
- H123: identification flag for the type of data, as these are stored in the databases. The possibilities for this are:
  H.1: single-parameter fit for the cross section as function of the laboratory energy.
  H.2: single-parameter fit for the rate coefficient as function of the target temperature.
  H.3: double-parameter fit for the rate coefficient as function of the projectile energy and the target temperature.
  H.4: double-parameter fit for the rate coefficient as function of the target density and temperature.
  H.5: single-parameter fit for the target particle momentum-weighted rate coefficient as function of the target temperature.
  H.6: double-parameter fit for the target particle velocity-weighted rate coefficient as function of projectile energy and target temperature.
  H.7: double-parameter fit for the target particle velocity-weighted rate coefficient as function of target density and temperature.
  H.8: single-parameter fit for the target particle energy-weighted rate coefficient as function of the target temperature.

H.9: double-parameter fit for the target particle energy-weighted rate coefficient as function of projectile energy and target temperature.
H.10: double-parameter fit for the target particle energy-weighted rate coefficient as function of target density and temperature.

- REAC: identification number of the reaction as stored in the considered datafile.
- CRC: identification label for the type of the considered reaction. The possibilities for this are:
  EI: electron-impact collision (i.e. ionization, excitation, de-excitation).
  CX: charge-exchange collision.
  EL: elastic collision.
  DS: dissociation.
  RC: recombination.
- MASSP: mass number (in AMU) of the first particle involved in the reaction.
- MASST: mass number (in AMU) of the second particle involved in the reaction.
- DP: additional energy lost or gained in the reaction, which is not already included in the rates.

Then, the **atomic/molecular species** which are considered are described. This is done through three different sub-blocks, for neutral atomic, neutral molecular and test ion species respectively, each one having the same structure. In particular, we have:

- ISPS: labelling index.
- TEXT: name of the species as it is desired to be plotted in the printout.
- NMASS: mass number.
- NCHAR: nuclear charge number.
- NPRT: flux units carried by one particle of this species (i.e. equivalent atoms per second).
- NCHRG: atomic charge state.
- ISRF: species index flag for the fast particle reflection mode.
- ISRT: species index flag for the thermal particle re-emission model.
- ID1: not in use.
- NRC: total number of reactions to be read for this species.
- NFOL: flag controlling the motion of test particle species.
- NGEN: maximum number of generations for test particle species.
- NHSTS: default to 0.
- ID3: not in use.

For each species, after the first line we find other sub-sub-blocks, one for each reaction to be read for the species, characterizing properties such as velocity distribution of the reaction products, energy loss per collision, etc.

Here is how the block 4 should appear:

```
*** 4. Data for species and atomic physics module
**  Reactions
    14
  1 AMJUEL H.4 2.1.5    EI   0  1 0.00000E+00 0.00000E+00 0.00000E+00
  2 AMJUEL H.102.1.5    EI   0  1 0.00000E+00 0.00000E+00 0.00000E+00
  3 HYDHEL H.1 3.1.8    CX   1  1 0.00000E+00 0.00000E+00 0.00000E+00
  3 HYDHEL H.3 3.1.8    CX   1  1 0.00000E+00 0.00000E+00 0.00000E+00
  4 AMJUEL H.4 2.2.9    EI   0  2 0.00000E+00 0.00000E+00 0.00000E+00
  5 AMJUEL H.4 2.2.5g   DS   0  2 0.00000E+00 0.00000E+00 0.00000E+00
  6 AMJUEL H.4 2.2.10   DS   0  2 0.00000E+00 0.00000E+00 0.00000E+00
  7 AMJUEL H.0 0.3T     EL   1  2 0.00000E+00 0.00000E+00 0.00000E+00
  7 AMJUEL H.1 0.3T     EL   1  2 0.00000E+00 0.00000E+00 0.00000E+00
  7 AMJUEL H.3 0.3T     EL   1  2 0.00000E+00 0.00000E+00 0.00000E+00
  8 AMJUEL H.2 3.2.3    CX   1  2 0.00000E+00 0.00000E+00 0.00000E+00
  9 AMJUEL H.4 2.2.11   EI   0  2 0.00000E+00 0.00000E+00 0.00000E+00
 10 AMJUEL H.4 2.2.12   DS   0  2 0.00000E+00 0.00000E+00 0.00000E+00
 11 AMJUEL H.4 2.2.14   DS   0  2 0.00000E+00 0.00000E+00 0.00000E+00
 12 AMJUEL H.8 2.2.14   DS   0  2 0.00000E+00 0.00000E+00 0.00000E+00
 13 AMJUEL H.4 2.1.8    RC   0  1 0.00000E+00 0.00000E+00 0.00000E+00
 14 AMJUEL H.102.1.8    RC   0  1 1.36000E+01 0.00000E+00 0.00000E+00
** 4a. Neutral atom species
     1
 1 D         2  1  1  0  1 -1  0  2  0  0  0
     1   115   114     0 30000     0     0
 2.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 1.00000E+00 0.00000E+00
     3   114   111   114  1001     0     0
 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 1.00000E+00 0.00000E+00
** 4b. Neutral molecule species
     1
 1 D2        4  2  2  0  0  1  0  5  0  0  0  1
     4   115   113     0     0     0     0
-1.54000E+01 0.00000E+00 0.00000E+00 0.00000E+00 1.00000E+00
     5   115   121     0     0     0     0
-1.05000E+01 0.00000E+00 3.00000E+00 3.00000E+00 1.00000E+00
     6   115   111   114     0     0     0
-2.50000E+01 0.00000E+00 5.00000E+00 5.00000E+00 1.00000E+00
     7   114   114   112  1001     0     0
 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 1.00000E+00
     8   114   111   113  1001     0     0
 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 1.00000E+00
** 4c. Test ion species
     1
 1 D2+       4  2  2  1  0 -1  0  3 -1  0  0  1
     9   115   124     0     0     0     0
-1.55000E+01 0.00000E+00 2.50000E-01 2.50000E-01 1.00000E+00
    10   115   111   114     0     0     0
-1.04000E+01 0.00000E+00 4.30000E+00 4.30000E+00 1.00000E+00
    11   115   121     0 30300     0     0
 1.20000E+01 0.00000E+00 1.20000E+01 0.00000E+00 1.00000E+00
```

The **block 5** contains the input data for the **plasma background**, i.e. information about the considered bulk ion species present in the plasma modelled by B2.5. As it is, for each of these species arrays containing particle density, flow velocity (three components) and temperature are defined, with one value per parameter and per B2.5 grid cell. In particular, the number of the considered bulk ion species is described by the variable NPLS.
The form of the block is:

```
*** 5. Data for plasma background
** 5a. Bulk ion species                                              NPLS times
NPLS
-------------------------------------------------------------------------------
ISPZ TEXT NMASS NCHAR NPRT NCHRG ISRF ISRT ID1 NRC NFOL NGEN NHSTS ID3 }
-------------------------------------------------------------------------------
** 5b. Plasma background data
INDPRO(1)  INDPRO(2)  INDPRO(3)  INDPRO(4)  INDPRO(5)  INDPRO(6)
INDPRO(7)  INDPRO(8)  INDPRO(9)  INDPRO(10) INDPRO(11) INDPRO(12)
IF INDPRO(1)≤5 then
     TE0 TE1 TE2 TE3 TE4 TE5
ENDIF
IF INDPRO(2)≤5 then
-------------------------------
     TI0 TI1 TI2 TI3 TI4 TI5 } NPLS times
-------------------------------
ENDIF
IF INDPRO(3)≤5 then
-------------------------------
     DI0 DI1 DI2 DI3 DI4 DI5 } NPLS times
-------------------------------
ENDIF
IF INDPRO(4)≤5 then
-------------------------------
     VX0 VX1 VX2 VX3 VX4 VX5
     VY0 VY1 VY2 VY3 VY4 VY5 } NPLS times
     VZ0 VZ1 VZ2 VZ3 VZ4 VZ5
-------------------------------
ENDIF
IF INDPRO(5)≤5 then
     B0 B1 B2 B3 B4 B5
ENDIF
IF INDPRO(12)≤ then
     VL0 VL1 VL2 VL3 VL4 VL5
ENDIF
```

First of all, the **bulk ion species** which are considered are described in the sub-block 5a, in a sub-block which has the same structure of the ones present in the blocks 4a, 4b, 4c. In particular, we have:

- ISPS: labelling index.
- TEXT: name of the species as it is desired to be plotted in the printout.
- NMASS: mass number.
- NCHAR: nuclear charge number.
- NPRT: flux units carried by one particle of this species (i.e. equivalent atoms per second).
- NCHRG: atomic charge state.
- ISRF: species index flag for the fast particle reflection mode.
- ISRT: species index flag for the thermal particle re-emission model.
- ID1: not in use.
- NRC: total number of reactions to be read for this species.
- NFOL: flag controlling the motion of test particle species.
- NGEN: maximum number of generations for test particle species.
- NHSTS: default to 0.
- ID3: not in use.

Again, for each species, after the first line we find other sub-sub-blocks, one for each reaction to be read for the species, characterizing properties such as velocity distribution of the reaction products, energy loss per collision, etc.

Then, the block 5b start with an array of 12 flags, each one controlling one particular **input tally**. Namely, it describes how each particular plasma background quantity is read by EIRENE, describing the type of considered profile or how the data are transferred from B2.5 to EIRENE. The quantities associated to each flag are:

- `INDPRO(1)`: electron temperature `TEIN`.
- `INDPRO(2)`: ion temperature `TII` (either one common for all the `NPLS` buck ion species, or one for each species).
- `INDPRO(3)`: density `DIIN` of each of the `NPLS` buck ion species (and of the electrons as well, having supposed quasi-neutrality).
- `INDPRO(4)`: components `VXIN`, `VYIN`, `VZIN` of the ion flow velocity (either one common for all the NPLS buck species, or one for each species).
- `INDPRO(5)`: magnetic field `BIN`.
- `INDPRO(12)`: cell volume `VL`.

For SOLPS-ITER, the value of interest of such flags is 6, which would mean that each of these quantities are directly transferred into EIRENE from the B2.5 output after each iteration of the latter.

Here is how the block 5 should appear:

```
*** 5. Data for plasma background
** 5a. Bulk ion species
     1
 1 D+        2  1  1  1  1 -1  0  1  0  0  0  0            0
    13   115   111     0 30000
 1.40000E+01 0.00000E+00 0.00000E+00 0.00000E+00 1.00000E+00
** 5b. Plasma background data
     6     6     6     6     6     6     0     0     0     0     0     0
 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00 0.00000E+00
```

The **block 6** contains the input data for the **surface interaction models**, i.e. the specification of the material properties of the surfaces, such as type of material, temperature, reflection properties, absorbing properties, eventual sputtering model, etc. It is possible, in particular, to define a number of groups of surfaces, each one defined by a name of the type SURFMOD_{MODNAME}, to be characterized by the same properties, and linked to the surfaces as listed in the block 3.
In detail, the surface interaction model comprises the following information:

- The probability p_f that a particle striking a surface is re-emitted as a fast particle (i.e. **reflection**).
- The probability p_t that a particle striking a surface is re-emitted as a thermal particle with a temperature equal to the surface temperature (i.e. **thermal re-emission**).
- The probability p_a that a particle striking a surface is absorbed by it, so that it disappears (i.e. **absorption**, to be defined for pumping surfaces).
- The probability that a particle striking a surface induces **physical sputtering**.
- The probability that a particle striking a surface induces **chemical sputtering**.
- Energy distribution and emission angle of reflected fast particles and sputtered particles.

All these data may be functions of the incident energy of the striking particle and its angle of incidence against the surface normal, and of the projectile and target species. The three probabilities p_f, p_t and p_a must add up to 1.
The form of the block is:

```
***6. General data for reflection model
NLTRIM
TPATH
A_on_B
DATD
DMLD
DIOD
DPLD
ERMIN ERCUT RPROBF RINTEG EINTEG AINTEG
-------------------------------------------------------
SURFMOD_{MODNAME}                                      }
ILREF   ILSPT   ISRS    ISRC                           }
ZNML    EWALL   EWBIN   TRANSP_1 TRANSP_2 FSHEAT       } as times as the number
RECYCF  RECYCT  RECPRM  EXPPL    EXPEL    EXPIL        }   of different groups
RECYCS  RECYCC  SPTPRM  ESPUTS   ESPUTC                }     of surfaces
-------------------------------------------------------
```

First of all, NLTRIM is a logical switch controlling whether the TRIM database is used. After that, TPATH describes the path to the TRIM database. It is then specified which of the data contained in the TRIM database are read, each one describing a possible projectile-target combination A_on_B. The possible combinations include hydrogen (H), deuterium (D), tritium (T) and helium (He) as projectiles, and carbon (C), iron (Fe), molybdenum (Mo) and tungsten (W) as targets.

Then, the successive lines describe the distribution for sampling the species index of reflected or otherwise emitted particles. Namely, these state the relative probability that each species of atoms, molecules, test ions and bulk ions, respectively, are emitted, rather than another one.

- DATD: describes the distribution of emission for each of the NATM considered species of atoms.
- DMLD: describes the distribution of emission for each of the NMOL considered species of molecules.
- DIOD: describes the distribution of emission for each of the NION considered species of test ions.

- DPLD: describes the distribution of emission for each of the NPLS considered species of bulk ions.

The next flags regulate, so some extent, the particle interaction model in terms of the energy of the incident particle.

- ERMIN: energy of the incident particle such that below it the reflection model is switched off.
- ERCUT: value such that if the energy of the incident particles lies between ERMIN and ERCUT then the probability of reflection is RPROBF.
- RINTEG:
  if > 0, then the particle reflection coefficient is fixed, i.e. independent of energy and angle of incidence, and set as $p_f = min(1 - p_a, RINTEG)$.
  if = 0, then the reflection probability is defined by the chosen reflection model.
  if < 0, then the reflection probability is set to $p_f = 1 - p_a$ (i.e. pure reflection on any non-pumping surfaces).
- EINTEG:
  if > 0, then the energy reflection coefficient is fixed, i.e. independent of energy and angle of incidence, and the energy random sampling procedure in the reflection model is replaced by the assumption $E_{out} = E_{in} \cdot EINTEG$.
  if = 0, then we have the default energy distribution in the reflection model.
  if < 0, then the energy of emission is set to $E_{out} = E_{in}$ (i.e. purely elastic reflection).
- AINTEG:
  if > 0, then the momentum reflection coefficient is fixed, i.e. independent of energy and angle of incidence, and the angle random sampling procedure in the reflection model is replaced by the assumption $\alpha = 1 - AINTEG$.
  if = 0, then we have the default angular distribution in the reflection model.
  if < 0, then the angle of emission is set to $\alpha = 0$ (i.e. specular reflection).

Finally, the next sub-blocks define the **local surface interaction model** for each group of surfaces, each one indicated with the label SURFMOD_{MODNAME}. In this way, many surfaces present in the block 3 may be linked to a same local surface interaction model.
Each local model is firstly characterized by four integer flags:

- ILREF: defines the choice of the local reflection model. If it is 1, then the TRIM database is used. If it is 2, then the modified Behrisch matrix model is used.
- ILSPT: two-digit number, defines the choice of the local sputtering model.
  If the first digit is 0, then physical sputtering is de-activated on this surface. If it is 1, then a constant physical sputtering rate is considered for this surface. If it is 2, then the Roth-Bohdansky formula for the physical sputtering yield is used, Thomson energy distribution and a cosine angular distribution for the emitted particles.
  If the second digit is 0, then chemical sputtering is de-activated on this surface. If it is 1, then a constant chemical sputtering rate is considered for this surface. If it is 2, then the Roth formula for the chemical sputtering yield is used, with a thermal distribution of the emitted particles.
- ISRS: sputtered particle species flag for physical sputtering:
  if > 0, then both sputtered particle and reflected particle (if any) will be followed.
  if = 0, then the sputtered particle is not followed, even if its atomic test particle species could be recognized; only the reflected particle is followed.
- ISRC: sputtered particle species flag for chemical sputtering:
  if > 0, then both sputtered particle and reflected particle (if any) will be followed.
  if = 0, then the sputtered particle is not followed, even if its atomic test particle species could be recognized; only the reflected particle is followed.

After that, we find some lines with the following numbers:

- ZNML: four/five digits; the first two/three digits define the atomic weight of the surface material, while the final two digits define the nuclear charge of the surface material. For example, for carbon we have 1206, for iron we have 5626, for molybdenum we have 9642, for tungsten we have 18474.
- EWALL: surface temperature.
  if < 0, then $T_w = -EWALL$ will be the temperature, in eV, of a maxwellian distribution for the thermally re-emitted particles, meaning that these will have a mean energy $E_{mean} = 2T_w$.
  if > 0, then all thermally re-emitted particles are monoenergetic, with an energy equal to $EWALL$.
  if = 0, then the energy of the thermally re-emitted particles is sampled from a Thomson distribution, with parameter EWBIN.
- EWBIN: parameter for the Thomson distribution is EWALL is 0.
- TRANSP_1: specifies the semi-transparency for particles incident from the positive side. It is 0 by default (i.e. no transparency).
- TRANSP_2: specifies the semi-transparency for particles incident from the negative side. It is by default (i.e. no transparency).
- FSHEAT: surface sheath potential factor.
- RECYCF: multiplier for the reflection probability, which is modified as
  $p_f = min(1 - p_a, RECYCF \cdot RPROBF)$. It is 1 by default.
- RECYCT: recycling coefficient, i.e. the fraction of particles which are not absorbed (meaning that $RECYCT = p_f + p_t$). In this way, a flux $RECYCT \cdot influx$ is re-emitted from the surface for any $influx$ of particles of any species. Then, the fraction $p_a = 1 - RECYCT$ of incident (atomic) flux will be absorbed at the surface.
  1 stands for fully recycling surfaces, i.e. the "real" material surface, while values between 0 and 1 indicate pumping surfaces. In particular, if A is the surface area of the surface as seen by the particles, in cm^2, then this value can be related with the pumping speed S of the surface, in m^3/s, through the relation $S = A \cdot (1 - RECYCT) \cdot 3{,}638 \cdot \sqrt{T/m}$ with T temperature of the particles in K and m their mass in AMU.
- RECPRM: 0 by default.
- EXPPL: if the modified Behrisch matrix model is used (i.e. ILREF is 2), then the incident angular dependence of the particle reflection coefficient has the form
  $p_f(\Phi) \propto 1 - (1 - RPROBF) \cdot [\cos(\Phi)]^{EXPPL}$. It is 1 by default.
- EXPEL: if the modified Behrisch matrix model is used (i.e. ILREF is 2), then the incident angular dependence of the energy reflection coefficient has the form
  $E_{out}(\Phi) \propto 1 - (1 - RPROBF) \cdot [\cos(\Phi)]^{EXPEL}$. It is 0,5 by default.
- EXPIL: if the modified Behrisch matrix model is used (i.e. ILREF is 2), regulates the incident angular dependence of the momentum reflection coefficient. It is 0 by default (i.e. cosine distribution).
- RECYCS: if the physical sputtering is activated (i.e. the first digit of ILSPT is greater than 0) then this defines the constant physical sputtering yield (if that digit was 1) or a multiplier for the theoretical physical sputtering yield (if that digit was 2).
- RECYCC: if the chemical sputtering is activated (i.e. the second digit of ILSPT is greater than 0) then this defines the constant chemical sputtering yield (if that digit was 1) or a multiplier for the theoretical chemical sputtering yield (if that digit was 2).
- SPTPRM: 0 by default
- ESPUTS: 0 by default.
- ESPUTC: 1 by default.

Here is how the block 6 should appear:

```
*** 6. General data for reflection model
T
cpath modules/Eirene/Database/Surfacedata/TRIM/
D_on_Fe
D_on_W
 1.00000E+00
 1.00000E+00
 1.00000E+00
 1.00000E+00
 1.00000E+00 5.00000E+01 1.00000E-01 0.00000E+00 0.00000E+00 0.00000E+00
SURFMOD_1_TUNGSTEN_TARGET
     1    12     0     0     1
 1.84740E+04-1.00000E-01 0.00000E+00 0.00000E+00 0.00000E+00-1.00000E+00
 1.00000E+00 1.00000E+00 1.00000E+00 1.00000E+00 5.00000E-01 1.00000E+00
 1.00000E+00 1.00000E+00 0.00000E+00 0.00000E+00 1.00000E+00
SURFMOD_2_IRON
     1    12     0     0     1
 5.62600E+03-1.00000E-01 0.00000E+00 0.00000E+00 0.00000E+00-1.00000E+00
 1.00000E+00 1.00000E+00 1.00000E+00 1.00000E+00 5.00000E-01 1.00000E+00
 1.00000E+00 1.00000E+00 0.00000E+00 0.00000E+00 1.00000E+00
SURFMOD_3_TUNGSTEN
     1    12     0     0     1
 1.84740E+04-1.00000E-01 0.00000E+00 0.00000E+00 0.00000E+00-1.00000E+00
 1.00000E+00 1.00000E+00 1.00000E+00 1.00000E+00 5.00000E-01 1.00000E+00
 1.00000E+00 1.00000E+00 0.00000E+00 0.00000E+00 1.00000E+00
SURFMOD_4_TURBOPUMP
     1    00     0     0
 5.62600E+03-1.00000E-01 0.00000E+00 0.00000E+00 0.00000E+00 1.00000E+00
 1.00000E+00 9.93000E-01 0.00000E+00 1.00000E+00 5.00000E-01 1.00000E+00
SURFMOD_5_CRYOPUMP
     1    00     0     0
 5.62600E+03-1.00000E-01 0.00000E+00 0.00000E+00 0.00000E+00 1.00000E+00
 1.00000E+00 7.00000E-01 0.00000E+00 1.00000E+00 5.00000E-01 1.00000E+00
SURFMOD_6_CONDUCTANCE
     1    00     0     0
 5.62600E+03-1.00000E-01 0.00000E+00 0.00000E+00 0.00000E+00 1.00000E+00
 1.00000E+00 1.00000E+00 0.00000E+00 1.00000E+00 5.00000E-01 1.00000E+00
 1.00000E+00 1.00000E+00 0.00000E+00
```

The **block 7** contains the input data for the **initial distribution of test particles**, i.e. the specification of how these are created and then launched. These can be result, in fact, from a number NSTRAI of different sources, or "**strata**", which are computed one after the other and are linearly superimposed at the end of each EIRENE run. These can result from **recycling** of the (ion) influx the outer boundaries of the B2.5 mesh (targets, main chamber wall, PFRs), **gas puff** source and **volumetric recombination** source.
The form of the block is:

```
*** 7. Data for primary sources
NSTRAI
INDSRC(1), INDSRC(2), ..., INDSRC(NSTRAI)                          NSTRAI times
ALLOC
-------------------------------------------------------------------------------
* TXTSOU
NLAVRP NLAVRT  NLSYMP NLSYMT
NPTS   NINITL  NEMODS NAMODS NMINPTS
FLUX   SCALV
*Species index distribution
NLATM  NLMOL  NLION  NLPLS  NLPHOT
NSPEZ
*Distribution in physical space and time
NLPNT     NLLNE     NLSRF     NLVOL     NLCNS
NSRFSI
-------------------------------------------------------------------------------
INUM      INDIM     INSOR
INGRDA(1) INGRDE(1) INGRDA(2) INGRDE(2) INGRDA(3) INGRDE(3)   } NSRFSI times
SORWGT    SORLIM    SORIND    SOREXP    SORIFL
NRSOR     NPSOR     NTSOR     NBSOR     NASOR     NISOR
SORAD1    SORAD2    SORAD3    SORAD4    SORAD5    SORAD6
-------------------------------------------------------------------------------
*Distribution in velocity space
SORENI    SORENE    SORVDX    SORVDY    SORVDZ
SORCOS    SORMAX    SORCTX    SORCTY    SORCTZ
-------------------------------------------------------------------------------
```

First of all, each of the NSTRAI strata (i.e. each different source of particles to be launched and followed by EIRENE) is described by the variable INDSRC, which specifies the way in which the input data for each stratum are to be read. Then, ALLOC defines the allocation of CPU-time per stratum, proportionally to the variable NPTS for each of it.

Then, we have NSTRAI sub-blocks, one for each stratum.

At the beginning of each of it, we find the variable TXTSOU, which is the name of the stratum as it is desired to be plotted in the printout, and then the logical switches:

- NLAVRP: ...
- NLAVRT: ...
- NLSYMP: ...
- NLSYMT: ...

Then we find some integer flags controlling the **general properties** for the particles launched from each stratum:

- NPTS: maximum number of test particles, for each stratum, to be launched per history, if it is greater than 0; instead, if it is zero, the stratum is "turned off".
- NINITL: if greater than 0, then seed for the initialization of the random number generator; if equal to 0, than no initialization of the random number is performed.

- NEMODS: four-digit number, flag to select one of the pre-programmed source energy conditional distributions of the launched particles.
  The first digit regulates the energy distribution of the particles launched from the stratum. It can be:
  - 1: mono-energetic source with fixed energy $E_0 = SORENI$.
  - 2: mono-energetic source with energy $E_0 = SORENE \cdot T_e + SORENI \cdot T_i$.
  - 3: as 2, but with added sheath acceleration.
  - 4: mono-energetic source with energy equal to the mean energy of a shifted maxwellian distribution.
  - 5: as 4, but with added sheath acceleration.

  The second digit regulates the choice of temperature to be used for the shifted maxwellian distribution if the first digit was 4 or 5:
  - 0: the temperatures T_e, T_i to be used for the distribution are the local temperatures taken from the input tallies TEIN, TIIN on the grid.
  - 1: the local temperatures to be used for the distribution are replaced by the input costants $T_e = SORENE$ and $T_i = SORENI$.

  The third digit regulates the choice of velocities to be used for the shifted maxwellian distribution if the first digit was 4 or 5:
  - 0: the drift velocity $\boldsymbol{V}$ to be used in the distribution is the local ion drift velocity for bulk ions.
  - 1: the local drift velocity to be used for the distribution is replaced by the vector $\boldsymbol{V} = (SORVDX, SORVDY, SORVDZ)$.

  The fourth digit is 1 by default.
- NAMODS: flag to select one of the pre-programmed conditional source angular distributions of the launched particles. In each case, the reference for the direction of emission of the particle is the reference unit vector $\boldsymbol{C}$ which is defined, by default, as the outward surface normal vector (in case of surface sources) or as $\boldsymbol{C} = (SORAD4, SORAD5, SORAD6)$ (in case of volume sources); alternatively, it can be overwritten and given by the normalization of $(SORCTX, SORCTY, SORCTZ)$.
  It can be:
  - 1: the polar angle against the reference unit vector is sampled from the distribution $f(\theta)d\theta \propto sin\,\theta\,(cos\,\theta)^{SORCOS}d\theta$. In this case the exponent $SORCOS$ will define the distribution (e.g. 0 for isotropic distribution, 1 for cosine distribution), while $SORMAX$ is used as cut-off angle for the polar angle distribution.
  - 2: the polar angle against the reference unit vector is sampled from a gaussian distribution with zero mean value. In this case $SORCOS$ is used as standard deviation of the distribution, while $SORMAX$ is used as cut-off angle for the polar angle distribution.
- NMINPTS: minimum number of test particles enforced for the stratum.

Then, finally, we found the real number FLUX, which is used as scaling factor for the surface- or volume-averaged tallies defining the source intensity of each stratum. This is done only if the flag SCALV is different from zero.

Then, we find some flags for defining the **species** of particles which are launched from each stratum (one and one only of the first line of logical switches must be true):

- NLATM: true if the stratum is a source of atoms.
- NLMOL: true if the stratum is a source of molecules.
- NLION: true if the stratum is a source of test ions.
- NLPLS: true if the stratum is a source of bulk ions.
- NLPHOT: true if the stratum is a source of photons.

- NSPEX: defines the species index of the source particle.

Then, we find some flags for defining **type** and **space distribution** of each stratum (one and one only of the first line of logical switches must be true):

- NLPNT: true if the stratum is a point source.
- LNNLE: true if the stratum is a line source.
- NLSRF: true if the stratum is a surface source.
- NLVOL: true if the stratum is a volume source.
- NLCNS: true if time-dependent mode is activated.
- NSRFSI: total number of different points over which the starting points for the stratum are distributed, defining a number of substrata.

For each substratum, we find then the following cards:

- INUM: labelling index for the substratum.
- INDIM: defines the type of surface on which the stratum is located. It must be:
    - 0 for sources on additional surfaces.
    - 1 for sources on standard surfaces on x-directed mesh (e.g. for recycling surfaces on main chamber wall and PFR boundaries).
    - 2 for sources on standard surfaces on the y-directed mesh (e.g. for recycling surfaces on the targets).
    - 3 for sources on standard surfaces on the z-directed mesh.
    - 4 for sources on surfaces composed of one of more segments of radial and/or poloidal polygons.
- INSOR: defines the number of the considered surfaces on the standard meshes (i.e. as INUMP in the block 3a) or on the additional mesh (i.e. as labelled in the block 3b).
- INGRDAs, INGRDEs: as IRPTAs, IRPTEs in the block 3a for sources on standard mesh surfaces; irrelevant for sources on additional mesh surfaces.
- SORWGT: defines the relative frequency for starting points on the surface labelled INUM.
- SORLIM: three-digit number; the three digits define one of the pre-programmed space distribution of the source in the x-direction, y-direction and z-direction of the surface, respectively. So, if a, b are the boundaries of the source location in the directions x, y or z, the digits can be:
    - 0: the respective coordinate is computed from the two others and from the equation for the surface number INSOR.
    - 1: δ-distribution at $(a+b)/2$.
    - 2: uniform distribution in the interval $[a, b]$.
    - 3: truncated exponential decay on the interval $[a, b]$.
    - 4: step function.
- SORIND: flag to choose on from the various step functions.
- SOREXP: defines decay length in the exponential distribution, in case of some digits of SORLIM equal to 3.
- SORIFL: ...
- NRSOR: ...
- NPSOR: ...
- NTSOR: ...
- NBSOR: ...
- NASOR: ...
- NPSOR: ...

Then, the next line is used to define the sampling intervals in the three coordinate direction x, y, z, i.e. the boundaries a, b of the source location. In case of standard surfaces, there is actually no need to specify manually such intervals, since they are read from the grid; in case of additional surfaces, instead, these data must be compiled.

- SORAD1: left endpoint of the interval $[a, b]$ for the x coordinate.
- SORAD2: right endpoint of the interval $[a, b]$ for the x coordinate.
- SORAD3: left endpoint of the interval $[a, b]$ for the y coordinate.
- SORAD4: right endpoint of the interval $[a, b]$ for the y coordinate.
- SORAD5: left endpoint of the interval $[a, b]$ for the z coordinate.
- SORAD6: right endpoint of the interval $[a, b]$ for the z coordinate.

Finally, we found the additional parameters which are eventually used to define the distribution of the emitted particles in the velocity space, namely the parameters SORENI, SORENE, SORVDX, SORVDY, SORVDZ used for defining the energy distribution of the emitted particles, and the parameters, and the parameters SORCOS, SORMAX, SORCTX, SORCTY, SORCTZ used for defining their angular distribution, whose functions was described before.

Here is how the block 7 should appear:

```
*** 7. Data for primary sources
     7
     0     0     0     0     0     0    -1
  0.5000E 00
* Stratum 1: Recycling, inner target
FFFFF
050000  1001     3     1
  1.0000E 00  1.0000E 00
FFFTF
    -1
FFTFF
     1
     1     2     1     1    19
  1.0000E 00  1.0400E 02  1.0000E 00  0.0000E 00  1.0000E 03
    -1     0     0     0
  0.0000E 00  0.0000E 00  0.0000E 00 -0.0000E 00  0.0000E 00  0.0000E 00
  3.0000E 00  0.5000E 00
  1.0000E 00  9.0000E 01
* Stratum 2: Recycling, outer target
FFFFF
050000  1001     3     1
  1.0000E 00  1.0000E 00
FFFTF
    -1
FFTFF
     1
     1     2    51     1    19
  1.0000E 00  1.0400E 02  2.0000E 00  0.0000E 00  1.0000E 03
    -1     0     0     0
  0.0000E 00  0.0000E 00  0.0000E 00 -0.0000E 00  0.0000E 00  0.0000E 00
  3.0000E 00  0.5000E 00
  1.0000E 00  9.0000E 01
* Stratum 3: Recycling, main chamber wall
FFFFF
050000  3001     2     1
  1.0000E 00  1.0000E 00
FFFTF
    -1
FFTFF
     1
     1     1    19     1    51
  1.0000E 00  1.4000E 02  3.0000E 00  0.0000E 00  1.1110E 03
    -1     0     0     0
  0.0000E 00  0.0000E 00  0.0000E 00 -0.0000E 00  0.0000E 00  0.0000E 00
  2.0000E 00  0.0000E 00
  1.0000E 00  9.0000E 01
```

```
* Stratum 4: Recycling, inner PFR
FFFFF
050000  3001     2     1
  1.0000E 00  1.0000E 00
FFFTF
    -1
FFTFF
     1
     1     1     1     1    13
  1.0000E 00  1.4000E 02  4.0000E 00  0.0000E 00  2.1110E 03
    -1     0     0     0
  0.0000E 00  0.0000E 00  0.0000E 00 -0.0000E 00  0.0000E 00  0.0000E 00
  2.0000E 00  0.0000E 00
  1.0000E 00  9.0000E 01
* Stratum 5: Recycling, outer PFR
FFFFF
050000  3001     2     1
  1.0000E 00  1.0000E 00
FFFTF
    -1
FFTFF
     1
     1     1     1    39    51
  1.0000E 00  1.4000E 02  5.0000E 00  0.0000E 00  2.1110E 03
    -1     0     0     0
  0.0000E 00  0.0000E 00  0.0000E 00 -0.0000E 00  0.0000E 00  0.0000E 00
  2.0000E 00  0.0000E 00
  1.0000E 00  9.0000E 01
* Stratum 6: Gas puff
FFFFF
050000  1001     1     1
  1.0000E 00  1.0000E 00
FTFFF
     1
FFFF
     1
     6     0   225     0     0     0     0     0     0
  1.0000E 00  2.1100E 02  0.0000E 00  0.0000E 00  1.0000E 03
     0     0     0     0     0
  2.4500E+02  2.4500E+02  1.6360E+01  1.7360E+01  0.0000E+00  0.0000E+00
  2.0000E 00  0.0000E 00
  0.0250E 00  4.5000E 01
* Stratum 7: Volume recombination
FFFFF
050000  4001     6     1
  1.0000E 00
FFFTF
     1
FFFTF
     1
     1     0     0
  1.0000E 00  1.4000E 02  1.0000E 00  0.0000E 00  1.1110E 03
    -1     0     0     0
  0.0000E 00  0.0000E 00  0.0000E 00  0.0000E 00  0.0000E 00  0.0000E 00
  3.0000E 00  0.5000E 00
  1.0000E 00  9.0000E 01
```

11 - Running SOLPS-ITER

So, summarizing once again, the steps to correctly set up a case are:

1. Insert/edit the desired **pre-processor parameters** in the pre-processor input files `b2ag.dat`, `b2ah.dat`, `b2ar.dat`, `b2ai.dat`, located in the `baserun/` directory.
2. Run, from the `baserun/` directory, the related **pre-processor programs** `b2ag.exe`, `b2ah.exe`, `b2ar.exe`, `b2ai.exe`, in order to produce the default files `b2fgmtry`, `b2fpardf`, `b2frates`, `b2fstati`, which will be read during the simulation.
3. Insert/edit the desired **user input parameters** in the main input file `b2mn.dat` and, optionally, in the others `b2.*` input files, located in the actual run directory. If the simulation is wanted to be run in coupled mode, also the EIRENE input parameters must be inserted/edited in the EIRENE input file `input.dat`.
4. Run, from the actual run directory, the **main program** `b2mn.exe`, in order to perform the simulation on the local machine.

In order to finally launch the run on the local machine we must enter, while in the run directory of the considered case, the command

- `localsubmit`

If, instead, we want to launch the run in the TOK cluster environment we must enter, while in the run directory of the considered case, the command

- `ippsubmit`

In case we want to submit the job in series mode (i.e. in the toks server), then the above mentioned command is sufficient. If, instead, we want to submit the job in parallel mode (i.e. in the tokp server), then we should add the parallelization options; this can be done by entering

- `ippsubmit -m '-np N'`

where `N` is the number of CPU cores which are intended to be used for the job.
For checking the works which are currently running on the cluster, we must enter

- `squeue | grep 'user'`

which will list all the running works, with job number and time of submission.
Finally, if for some reasons we want to abort a case while it is still running, we must enter

- `qdel 'job number'`

for a hard abord (executed immediately), or

- `touch b2mn.exe.dir/.quit`

in the run directory for a soft abort, which will be executed after that the current iteration has completed, generating a complete final plasma state after that.

Once that a run is successfully finished, an e-mail will be received, indicating start time, end time and running time.
First of all, we must check that all went good, that is that the `b2fstate` file is of correct size and content. This can be done by executing the script

- `b2fstate_OK`

which will give `yes` for a well-concluded run, or `no` if there was some error during it. If something went wrong, we must enter

- `touch b2fstati`

and then submit again, in order to reinitiate the simulation with the same initial state.

As already said, each run starts computations from the full **initial plasma state**, whose physical quantities are contained in the file `b2fstati`, which is produced by the related pre-processor program the first time. At the end of a run, that is after all the timesteps have been performed (and different physical

quantities characterize the plasma state), the main program will have produced the following unformatted **main output files**:

- **b2fparam**: it contains a description of the **physical parameters** (boundary conditions and transport coefficients) used in the run. It will contain various blocks, containing the strings of all the quantities for the various physical variables as named in /b2cmbp/ and in /b2cmpt/.
- **b2fmovie**: it contains a series of **movie frames** of the plasma evolution during the run. It will contain various blocks, each one containing a list of the variations of the values of some fundamental physical variables (density, temperature, etc.) for each cell of the computational domain between initial and final plasma state.
- **b2ftrack**: it contains the **residuals** of the B2.5 equations from the inner iterations during the run.
- **b2ftrace**: it contains some B2.5 **evolution data** during the run.
- **b2fplasma**: it contains a full description of the **plasma solution** and derived quantities from the run. This file, differently from the other output files, is not human-readable.
- **b2fstate**: it contains the full **final plasma state** as at the end of the run, that is all the physical quantities as evolved from the situation contained in b2fstati. It will contain various data blocks, each one containing a list of physical variables (density, temperature, fluxes, etc.) concerning species modelled by B2.5 (i.e. electrons and ions), for each cell of the computational domain.

Moreover, when running in couple mode, the additional **EIRENE output files** are created:

- **fort.44**: it contains the final state of the physical variables (density, temperature, fluxes etc.) concerning species modelled by EIRENE (i.e. atoms, molecules and test ions), for each cell of the computational domain.
- **fort.46**: it contains the final state of some fundamental physical variables concerning species modelled by EIRENE (i.e. atoms, molecules and test ions), for each triangle of the EIRENE triangular grid.

Finally, after the calculations are finished, also some **additional output files** can be produced, if the related switches are activated in b2mn.dat. These files are:

- **b2time.nc**, containing the time evolution of various important scalar/vector quantities computed during the run.
- **b2tallies.nc**, containing the time evolution of various integrated quantities computed during the run.
- **balance.nc**, containing all the arrays required by the balance post-processing routines, useful for the analysis of particle, momentum and energy balance of the plasma.

At the end of the run, all the just mentioned produced output files will be contained in the run directory.

If b2fstate is well-written, we must check whether the computed values have reached **convergence**, meaning that a steady-state situation has been modelled. For doing this we must check:

- The **time trends of the main plasma quantities** (e.g. point values of temperature and density), since the time evolution during the run should not change much in a converged case.
- The **residuals** of the B2.5 fluid equations, which must be small and flat, which can be plotted for:
  a hydrogen only case by executing
  ➢ resall_H
  a deuterium only case by executing
  ➢ resall_D

In both cases, we must however take into account the unavoidable Monte Carlo noise in the values.
It should be taken into account that SOLPS-ITER runs tend to take long time to converge, so that at the end of one particular run an actual steady state situation could be not reached yet. In such case,

we should set up a new initial plasma state starting from the former final plasma state, that is overwriting the b2fstati file with the just produced b2fstate file by entering

➢ cp b2fstate b2fstati

and then submit again the simulation. To make the process faster, auto resubmitting scripts can be used, in which we could set up a desired goal about the convergence.
Of course, this procedure implies that data for the last run will be totally overwritten with the current run data. So, if we want to save the old run data, we could just build a new directory in which to continue the run, instead of running it in the same old directory.

In case of a **system crash** during the run (not it case of the crash of the code itself), in which the final plasma state file b2fstate would be probably corrupted, it may be possible to recover the results and continue the simulation, since the code writes, at regular CPU-seconds intervals (specified in the related switch in b2mn.dat), the short recovery files called plasmastate.xxxx and b2.*_save_parameters.xxxx in the b2mn.exe.dir/ subdirectory located in the run directory, with xxxx numbers indicating the order in which the files were written, starting from 0000 and ranging up. This procedure uses the program **b2co**, which will produce the files necessary to restart the run from the last point at which a good plasma state was saved, according to the recovery CPU-time indicated in b2mn.dat.
So, after an eventual crash, if the plasmastate.xxxx files were present in the b2mn.exe.dir/ subdirectory, we must check the last one in order of creation, and then execute

➢ setenv PLASMASTATE b2mn.exe.dir/plasmastate."last"

and then

➢ b2run b2co

which will create a "good" b2fstate file containing the last good information which were computed and saved about the plasma state. Then, we set up a new initial plasma state starting from this last recovered plasma state, that is overwriting the b2fstati file with the just produced b2fstate file by entering

➢ cp b2fstate b2fstati

after which, we re-launch the run by submitting it again.

Other **additional programs** included in the B2.5 code, providing other useful information if desired, are:

- **b2yg**, executable by running
  ➢ b2run b2yg
  which will produce a b2yg.plt file containing a series of plots giving some details about the geometry, as well as a b2yg.prt ASCII file containing tables of the geometry quantities. The b2yg.plt file could also be converted in a .ps file by executing
  ➢ ctrans -d ps.mono b2yg.plt > b2yg.ps
  which can be readily visualized by running
  ➢ gv b2yg.ps

- **b2yh**, executable by running
  ➢ b2run b2yh
  which will produce a b2yh.prt ASCII file containing a description of the run parameters: a species list, default boundary conditions, and transport coefficients.

- **b2yr**, executable by running
  ➢ b2run b2yr
  which will produce a b2yr.plt file containing a series of plots describing the dependencies of the atomic processes rates on density and temperature for each isonuclear sequence, as well as

a b2yr.prt ASCII file containing tables of these rates. The b2yr.plt file could also be converted in a .ps file by executing

➢ ctrans -d ps.mono b2yr.plt > b2yr.ps

which can be readily visualized by running

➢ gv b2yi.ps

- b2yi, executable by running

  ➢ b2run b2yi

  which will produce a b2yi.plt file containing a series of plots describing the plasma state (extracted by b2fstate), as well as a b2yi.prt ASCII file containing tables of the plasma quantities. The b2yi.plt file could also be converted in a .ps file by executing

  ➢ ctrans -d ps.mono b2yi.plt > b2yi.ps

  which can be readily visualized by running

  ➢ gv b2yi.ps

Another useful tool is the converter program b2yt, which can be used to change from one grid size and/or species to another. In particular, this allows to take a b2fstate solution, with size nx*ny, and transform it into a b2fstatt file with size nx1*ny1, provided the numbers of real cells in both x, y directions in the new solution are divisors of the ones in the original mesh. Other files necessary to proceed with the run under the new geometry will also be produced.
The dimensions of the "new" grid must be provided manually in a file called b2yt.dat which could have the form

```
*dimens         (nx1, ny1; free format)
     96   36
'solps_version' 'iter'
'eirene_format' 'iter'
```

with which is possible to create a solution on a 96x36 grid. For the conversion it is necessary to know the original version of SOLPS with which the various input/output files were created.
So, it a new geometry is requested, the b2yt program should be run after the new geometry file b2fgmtry has been produces and as available in the baserun/ subdirectory. The "old" geometry file should also remain available in the directory as b2fgmtry.old.The program also needs the b2mn.dat and any other used input files (such as input.dat, b2.*.parameters, etc.) from the original run.
It is important, for proper execution of this program, that the necessary files have the expected seniority. In particular, the following relations must be verified:

- b2ag.dat < b2fgmtry < b2fpardf < b2frates < b2fstate.
- b2ah.dat < b2fpardf.
- b2ar.dat < b2frates.

If this order is not met, it is possible to use a series of

➢ touch "filename"

commands to give the files new dates.
Upon successful termination of b2yt, we find a b2fstatt file containing the transformed plasma state, as well as a b2mn.dat.2 file which will contain the updated run switches appropriate for the new geometry. Similarly, the files b2.*.parameters.2 will contain the updated namelist variables.
A summary of the sequence of steps to be followed for a successful conversion is:

1. Create a new case directory.
2. In this new case directory, create a `baserun/` directory, a `convert/` directory and the run directory.
3. Copy from the old case to the new case's `convert/` directory all input and output files which are desired to be converted.
4. Check and correct, as necessary, the relative ages of the input files.
5. Run the b2yt program.
6. Displace the converted case input files to the converted case `baserun/` directory and run directory.

Finally, another useful additional program is **b2uf**, which convert the unformatted `b2fplasma` final plasma solution file into a formatted **`b2fplasmf`** file, containing a full description of the plasma solution and the derived quantities, which can be easily read by post-processing plotting tools. For doing this it is sufficient to run the program

➢ `b2run b2uf`

in the run directory at the end of the run, and the `b2fplasmf` file will be produced.

12 - Checking the results of a run

Once that a run is finished and convergence is finally reached, we are ready to analyze the **final results**, that is the steady-state plasma state following from the particular input parameters with which we have set up the simulation. In the simplest way, this can be done by plotting the content of the output files `b2time.nc` and `b2tallies.nc`, directly or using some scripts.

First of all, we can check the trend, during the simulation time, of some scalar time-dependent quantities, which are stored in the file **`b2time.nc`**, with different names.
Each of these quantities can be plotted as function of time by entering

- `2dt "quantity name"`

Alternatively, in order to plot one quantity versus another quantity for all the values of these, we can enter

- `2d "first_quantity_name" "second_quantity_name"`

If we want to make a file containing all the plots, we must execute

- `2d_plots`

This will prepare the file **`b2time2d.ps`**, containing all the quantities stored in `b2time.nc`; such file can then be read, showing the plots of the various quantities versus the running time, by executing

- `gv b2time2d.ps`

If, instead, we are interested only in the final values (the ones at the last time point), we must execute

- `2d.summarize`

This will prepare the file **`2d.data`**, containing such final values, which can be simply read by executing

- `vim 2d.data`

Such scalar quantities contained in `b2time.nc` are:

POINT VALUES AT MIDPLANE:

1. Density at (outer) midplane [m^{-3}]
2. Electron/ion temperature at (outer) midplane [eV]
3. Potential at (outer) midplane [V]

POINT VALUES AT DIVERTOR TARGETS:

4. Density at inboard/outboard divertor (separatrix value and maximum value) [m^{-3}]
5. Electron temperature at inboard/outboard divertor (separatrix value and maximum value) [eV]
6. Ion temperature at inboard/outboard divertor (separatrix value and maximum value) [eV]
7. Potential at inboard/outboard divertor (separatrix value and maximum value) [V]

RADIALLY INTEGRATED VALUES AT DIVERTOR TARGETS:

8. Total parallel particle flux onto inboard/outboard divertor [s^{-1}]
9. Total parallel electron power flux onto inboard/outboard divertor [W]
10. Total parallel ion power flux onto inboard/outboard divertor [W]
11. Total parallel current onto inboard/outboard divertor [A]

POLOIDALLY INTEGRATED VALUES AT SOL/PFR BOUNDARIES:

12. Total perpendicular particle flux to main chamber and to divertor region [s^{-1}]
13. Total perpendicular electron power flux to main chamber and to divertor region [W]
14. Total perpendicular ion power flux to main chamber and to divertor region [W]
15. Total perpendicular current to main chamber and to divertor region [A]

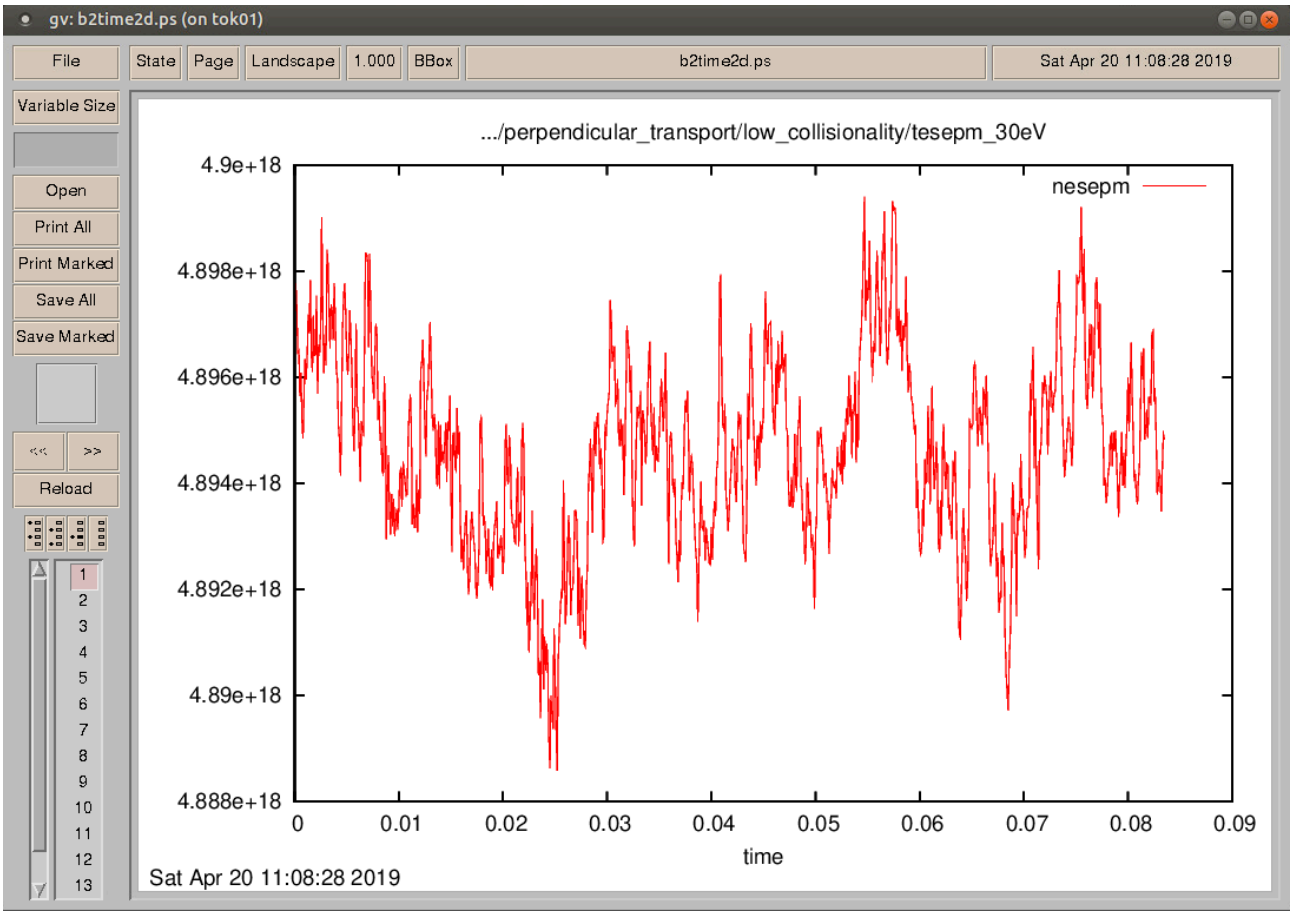
OTHERS:

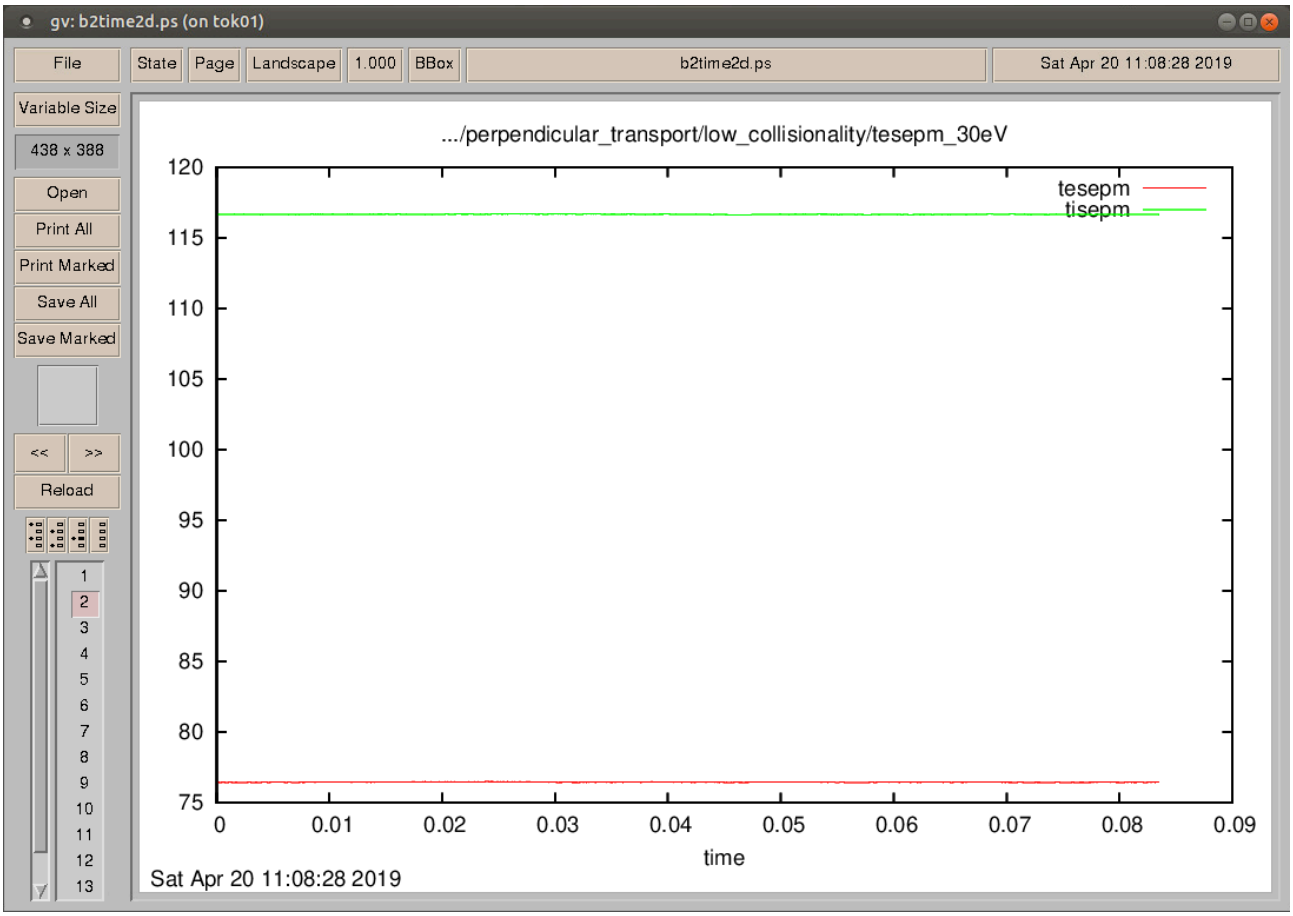
16. Maximum electron+ion power flux density onto inboard/outboard divertor [W/m^2]
17. Total number of particles in the computational domain
18. Total electron/ion energy in the computational domain [eV]

Names of all these scalar quantities as listed in `b2time.nc` are:

1. **nesepm**: outer midplane separatrix (electron) density [m^{-3}]
2. **tesepm**: outer midplane separatrix electron temperature [eV]
   tisepm: outer midplane separatrix ion temperature [eV]
3. **posepm**: outer midplane separatrix potential [V]
4. **nesepi**: inboard divertor separatrix (electron) density [m^{-3}]
   nemxip: inboard divertor maximum (electron) density [m^{-3}]
   nesepa: outboard divertor separatrix (electron) density [m^{-3}]
   nemxap: outboard divertor maximum (electron) density [m^{-3}]
5. **tesepi**: inboard divertor separatrix electron temperature [eV]
   temxip: inboard divertor maximum electron temperature [eV]
   tesepa: outboard divertor separatrix electron temperature [eV]
   temxap: outboard divertor maximum electron temperature [eV]
6. **tisepi**: inboard divertor separatrix ion temperature [eV]
   timxip: inboard divertor maximum ion temperature [eV]
   tisepa: outboard divertor separatrix ion temperature [eV]
   timxap: outboard divertor maximum ion temperature [eV]
7. **posepi**: inboard divertor separatrix potential [V]
   pomxip: inboard divertor maximum potential [V]
   posepa: outboard divertor separatrix potential [V]
   pomxap: outboard divertor maximum potential [V]
8. **fnixip**: inboard divertor integrated poloidal ion particle flux [s^{-1}]
   fnixap: outboard divertor integrated poloidal ion particle flux [s^{-1}]
9. **feexip**: inboard divertor integrated poloidal electron energy flux [W]
   feexap: outboard divertor integrated poloidal electron energy flux [W]
10. **feixip**: inboard divertor integrated poloidal ion energy flux [W]
    feixap: outboard divertor integrated poloidal ion energy flux [W]
11. **fchxip**: inboard divertor integrated poloidal current [A]
    fchxap: outboard divertor integrated poloidal current [A]
12. **fniyip**: main chamber integrated radial ion particle flux [s^{-1}]
    fniyap: divertor region integrated radial ion particle flux [s^{-1}]
13. **feeyip**: main chamber integrated radial electron energy flux [W]
    feeyap: divertor region integrated radial electron energy flux [W]
14. **feiyip**: main chamber integrated radial ion energy flux [W]
    feiyap: divertor region integrated radial ion electron energy flux [W]
15. **fchyip**: main chamber integrated radial current [A]
    fchyap: divertor region integrated radial current [A]
16. **pwmxip**: inboard divertor maximum total power flux density [W/m^2]
    pwmxap: outboard divertor maximum total power flux density [W/m^2]
17. **tmne**: total number of particles
18. **tmte**: total electron energy [eV]
    tmti: total ion energy [eV]

For example, here we see how the time evolution of density and temperatures at the outer midplane separatrix appears, in `b2time2d.ps`, after a run in which convergence has been reached.





Moreover, b2time.nc contains also some vector time-dependent quantities, that are strings containing the values of some quantities along lines of constant x (poloidal) coordinate; that is, they describe the time evolution of **radial profiles** of such quantities at some relevant poloidal coordinate:

- at **inner target** (l)
- at **inner midplane** (i)
- at **outer midplane** (a)
- at **outer target** (r)

The positions of inner and outer midplanes (that is their poloidal coordinate) can be arbitrarily selected in the strings b2mwti_jxi, b2mwti_jxa, respectively, in the b2mn.dat file.
So, entering

➢ 2d_profiles

we create ********.last10*** files, one for each vector quantity contained in b2time.nc. For each quantity, the filename containing its profile at inner target finishes with "l", the one containing its profile at inner midplane finishes with "i", the one containing its profile at outer midplane finishes with "a", and the one containing its profile at outer target finishes with "r".
We can visualize the vector of values contained in each of these files by entering

➢ vim *****.last10

depending on the quantity of interest.

Such vector quantities contained in b2time.nc are:

DENSITIES

1. (Electron) densities [m^{-3}]

TEMPERATURES

2. Electron temperatures [eV]
3. Ion temperatures [eV]

POTENTIALS

4. Potentials [V]

PARALLEL PARTICLE FLUX DENSITIES

5. Electron flux densities [$m^{-2}s^{-1}$]
6. Ion flux densities [$m^{-2}s^{-1}$]

PARALLEL POWER FLUX DENSITIES

7. Electron power flux densities [W/m^2]
8. Ion power flux densities [W/m^2]
9. Total power flux densities [W/m^2]

PARALLEL CURRENT DENSITIES

10. Parallel current densities [A/m]

NEUTRAL ATOMS DENSITIES

11. Neutral atoms densities [m^{-3}]

Names of all these vector quantities as listed in `b2time.nc` are:

	INNER TARGET	INNER MIDPLANE	OUTER MIDPLANE	OUTER TARGET
1. (Electron) densities [m^{-3}]	**ne3dl**	**ne3di**	**ne3da**	**ne3dr**
2. Electron temperatures [eV]	**te3dl**	**te3di**	**te3da**	**te3dr**
3. Ion temperatures [eV]	**ti3dl**	**ti3di**	**ti3da**	**ti3dr**
4. Potentials [V]	**po3dl**	**po3di**	**po3da**	**po3dr**
5. Electron flux densities [$m^{-2}s^{-1}$]	**fl3dl**	-	-	**fl3dr**
6. Ion flux densities [$m^{-2}s^{-1}$]	**fn3dl**	-	-	**fn3dr**
7. Electron power flux densities [W/m^2]	**fe3dl**	-	-	**fe3dr**
8. Ion power flux densities [W/m^2]	**fi3dl**	-	-	**fi3dr**
9. Total power flux densities [W/m^2]	**ft3dl**	-	-	**ft3dr**
10. Parallel current densities [A/m]	**fc3dl**	-	-	**fc3dr**
11. Neutral atoms densities [m^{-3}]	**an3dl**	**an3di**	**an3da**	**an3dr**

It should be noted that the values contained in `b2time.nc` regarding fluxes are actually fluxes per computational cell; anyway, by executing the script `2d_profiles`, it is automatically done a conversion which takes this values "per unit area" considering the physical size of each cell, so that values of flux densities are written in the related `*****.last10` files.

At inner and outer midplane, however, we can plot some transport coefficients, that are:

PARTICLE DIFFUSIVITIES

1. Density-driven particle diffusivities [m^2/s]
   dn3di at inner midplane, **dn3da** at outer midplane
2. Pressure-driven particle diffusivities [m^2/s]
   dp3di at inner midplane, **dp3da** at outer midplane

HEAT DIFFUSIVITIES

3. Electron heat diffusivities [m^2/s]
   ke3di at inner midplane, **ke3da** at outer midplane
4. Ion heat diffusivities [m^2/s]
   ki3di at inner midplane, **ki3da** at outer midplane

VISCOSITIES

5. Viscosities [m^2/s]
   vs3di at inner midplane, **vs3da** at outer midplane

In the following is how these files appear. Here for example we have the radial density profile at the outer midplane `ne3da.last10`:

```
 -6.273312073650993E-002        17177560000000000000
 -5.486785957146419E-002        15600180000000000000
 -4.077171086626574E-002        13615280000000000000
 -2.948488443244221E-002        11575130000000000000
 -2.066692772625169E-002        9533846000000002048
 -1.397828160442636E-002        7871404000000000000
 -9.057576797147374E-003        6768245000000000000
 -5.496817960617455E-003        6031254000000000000
 -2.896949275768723E-003        5509275000000000000
 -8.650881611592456E-004        5090265000000000000
  8.650881611592526E-004        4698607000000000000
  2.646912109142543E-003        4272191000000000000
  4.933274286200579E-003        3753843000000000000
  8.091766228553190E-003        3152461000000000000
  1.249193976604650E-002        2455195000000000000
  1.849546099579198E-002        1709194000000000000
  2.650773654300791E-002        1022149000000000000
  3.695471742242740E-002        503495500000000000
  5.021933571232239E-002        192753200000000000
  5.767766743561852E-002        114501000000000000
```

and here for example we have the radial temperature profiles at the outer midplane `te3da.last10` and `ti3da.last10`, respectively:

```
 -6.273312073650993E-002        410.397
 -5.486785957146419E-002        385.157
 -4.077171086626574E-002        330.71
 -2.948488443244221E-002        280.738
 -2.066692772625169E-002        235.656
 -1.397828160442636E-002        195.274
 -9.057576797147374E-003        159.673
 -5.496817960617455E-003        130.18
 -2.896949275768723E-003        106.322
 -8.650881611592456E-004        85.5873
  8.650881611592526E-004        67.324
  2.646912109142543E-003        54.6607
  4.933274286200579E-003        44.4186
  8.091766228553190E-003        35.888
  1.249193976604650E-002        30.6817
  1.849546099579198E-002        27.4978
  2.650773654300791E-002        24.893
  3.695471742242740E-002        21.9933
  5.021933571232239E-002        16.5164
  5.767766743561852E-002        11.4362
```

```
 -6.273312073650993E-002        517.485
 -5.486785957146419E-002        480.237
 -4.077171086626574E-002        396.262
 -2.948488443244221E-002        329.06
 -2.066692772625169E-002        275.297
 -1.397828160442636E-002        231.83
 -9.057576797147374E-003        196.304
 -5.496817960617455E-003        168.174
 -2.896949275768723E-003        145.686
 -8.650881611592456E-004        125.707
  8.650881611592526E-004        107.575
  2.646912109142543E-003        95.1612
  4.933274286200579E-003        84.6536
  8.091766228553190E-003        74.802
  1.249193976604650E-002        68.5698
  1.849546099579198E-002        64.5166
  2.650773654300791E-002        61.1293
  3.695471742242740E-002        57.8899
  5.021933571232239E-002        53.9844
  5.767766743561852E-002        51.6587
```

Then, at the end of the run, in addition to the various time-dependent quantities contained in `b2time.nc`, we can found also some tallies (volume and surface integrals), which are stored instead in the file **`b2tallies.nc`**, with different names.
In particular, we can have:

- **Volume region tallies**, which contain 5 volumetric quantities, one for each of the four region of the computational domain and one for the whole plasma, that are:
  `0`: **Whole plasma**
  `1`: **Main plasma**
  `2`: **SOL**
  `3`: **Inner divertor region**
  `4`: **Outer divertor region**
  These can be, for example, particle sources, in s^{-1}, power sources, in W, or charge sources, in A, calculated integrating the relative source densities on each region.
- **x-directed region tallies**, which contain 4 surface quantities, one for each of the four surfaces directed in poloidal direction, that are:
  `W`: **inboard divertor target**
  `w`: **entrance to inboard divertor region**
  `e`: **entrance to outboard divertor region**
  `E`: **outboard divertor target**
  These can be, for example, poloidal particle fluxes, in s^{-1}, poloidal power fluxes, in W, or poloidal currents, in A, calculated integrating the relative flux/current densities on each surface.
- **y-directed region tallies**, which contain 7 surface quantities, one for each of the seven surfaces directed in radial direction, that are:
  `sw`: **inboard divertor private flux boundary**
  `core`: **core boundary**
  `se`: **outboard divertor private flux boundary**
  `sep`: **separatrix**
  `nw`: **inboard divertor to main chamber wall boundary**
  `mcw`: **SOL to main chamber wall boundary**
  `ne`: **outboard divertor to main chamber wall boundary**
  These can be, for example, radial particle fluxes, in s^{-1}, radial power fluxes, in W, or radial currents, in A, calculated integrating the relative flux/current densities on each surface.

The region tallies contained in `b2tallies.nc` are:

- **`volreg`**: contains the total volume, for each region.
- **`nireg`**: contains the average ion density, for each region.
- **`nereg`**: contains the average electron density, for each region.
- **`tireg`**: contains the average ion temperature, for each region.
- **`tereg`**: contains the average electron temperature, for each region.
- **`poreg`**: contains the average potential, for each region.
- **`b2stbc_sna_reg`**: contains the boundary particle source term, for each region.
- **`b2stbc_sne_reg`**: contains the boundary electron source term, for each region.
- **`b2stbc_smo_reg`**: contains the boundary parallel momentum source term, for each region.
- **`b2stbc_she_reg`**: contains the boundary electron energy source term, for each region.
- **`b2stbc_shi_reg`**: contains the boundary ion energy source term, for each region.
- **`b2stbc_sch_reg`**: contains the boundary charge source term, for each region.

- **b2stbr_sna_reg**: contains the recycling particle source term, for each region.
- **b2stbr_sne_reg**: contains the recycling electron source term, for each region.
- **b2stbr_smo_reg**: contains the recycling parallel momentum source term, for each region.
- **b2stbr_she_reg**: contains the recycling electron energy source term, for each region.
- **b2stbr_shi_reg**: contains the recycling ion energy source term, for each region.
- **b2stbr_sch_reg**: contains the recycling charge source term, for each region.

- **rcxnareg**: contains the charge-exchange particle source term, for each region.
- **rcxmoreg**: contains the charge-exchange parallel momentum source term, for each region.
- **rcxhireg**: contains the charge-exchange ion energy source term, for each region.

- **rranareg**: contains the recombination particle source term, for each region.
- **rramoreg**: contains the recombination parallel momentum source term, for each region.
- **rrahireg**: contains the recombination ion energy source term, for each region.

- **rsanareg**: contains the ionization particle source term, for each region.
- **rsamoreg**: contains the ionization parallel momentum source term, for each region.
- **rsahireg**: contains the ionization ion energy source term, for each region.

- **rqahereg**: contains the electron cooling rate term, for each region.
- **rqbrmreg**: contains the bremsstrahlung term, for each region.

- **b2divua**: contains the heat source term from ion velocity gradient, for each region.
- **b2divue**: contains the heat source term from electron velocity gradient, for each region.
- **b2visa**: contains the heat source term from ion viscosity, for each region.
- **b2joule**: contains the heat source term from joule heating, for each region.
- **b2fraa**: contains the heat source term from inter-species friction, for each region.

Then, the x- and y- directed region tallies contained in `b2tallies.nc` are:

- **fnaxreg** / **fnayreg**: contain the poloidal particle flux across each poloidal-directed surface, and the radial particle flux across each radial-directed surface.
- **fhexreg** / **fheyreg**: contain the poloidal electron energy flux across each poloidal-directed surface, and the radial electron energy flux across each radial-directed surface.
- **fhixreg** / **fhiyreg**: contain the poloidal ion energy flux across each poloidal-directed surface, and the radial ion energy flux across each radial-directed surface.
- **fhmxreg** / **fhmyreg**: contain the poloidal parallel kinetic energy flux across each poloidal-directed surface, and the radial parallel kinetic energy flux across each radial-directed surface.
- **fhpxreg** / **fhpyreg**: contain the poloidal ionization energy flux across each poloidal-directed surface, and the radial ionization energy flux across each radial-directed surface.
- **fhjxreg** / **fhjyreg**: contain the poloidal electrostatic energy flux across each poloidal-directed surface, and the radial electrostatic energy flux across each radial-directed surface.
- **fhtxreg** / **fhtyreg**: contain the poloidal total energy flux across each poloidal-directed surface, and the radial total energy flux across each radial-directed surface.
- **fchxreg** / **fchyreg**: contain the poloidal current across each poloidal-directed surface, and the radial current across each radial-directed surface.

Plots from each tally can be called by `b2tallies.nc` with scripts, which will show values for the various volumetric/surface quantity for each tally. In particular, there are some interesting scripts involving the plot of energy flows and energy balances, which are:

- `energy_analysis`
  plots the electron+ion thermal energy flows through the various surfaces.
- `energy_analysis_fht`
  plots the total (thermal+kinetic+ionization+electrostatic) energy flows through the various surfaces.
- `energy_balance`
  plots the energy balance for the computational domain, including the boundary+recycling, charge-exchange, recombination, ionization, electron cooling and bremsstrahlung source terms.
- `energy_balance_extended`
  plots the energy balance for the computational domain, including, in addition to terms already present in the former, also the heat source terms from velocity gradient, ion viscosity, joule heating, inter-species friction.

13 - Visualizing the results of a run

Different ways for visualizing the results of a simulation run, in terms of plasma solution, are available. The most important one is the **b2plot** program, which is designed to take the data from both standalone B2.5 runs and B2.5-EIRENE coupled runs, in particular from the output files `b2fstate`, `b2fplasma`, `b2frates` and `b2fgmtry`, and `fort.44` when running with EIRENE, and graphically plot them under interactive control. The basic idea in using b2plot is to specify the data quantities from the plasma solution, some transformations, and then the plot command.
b2plot is stack-based: the various stacks are:

- A **matrix stack** (`M`), which is used to store quantities on the computational grid. These quantities can be stored in form `nx` × `ny` × `ns`, where `nx`, `ny` represent poloidal and radial sizes of the grid, and `ns` eventually ranges over the species.
- An **integer stack** (`I`), which is used to supply arguments for some of the commands.
- A **real stack** (`R`), which is also used to supply arguments for come commands and, more importantly, for storing values to be used in doing arithmetic on the matrix stack.
- A **string stack** (`S`), which is also used to supply arguments for some commands.

As input lines are parsed, each input entity is identified as being a command, an integer value, a real value or a string value. Integers and reals are distinguished by the latter having a decimal point, while strings must be put inside single quote marks. For each command with one or more stack requirements, these should be always put before the command while entering.
The program is simply started entering

➢ **`b2plot`**

in actual run directory of the simulation run we are interested in. After having entered the proper command, the desired plot is generated as a `.ps` file.

The final plot is generated with one single command line, starting with `echo`, in which we can specify what we want to plot and how the plot will appear. All we can write in a command line is described in the following sections.

Page format parameters: here we specify the graphic properties of the plot area of the output `.ps` file.

- `a4l`: sets the plot area in A4 landscape format.
- `a4p`: sets the plot area in A4 portrait format.
- `sq`: sets the plot area in square format.
- `<n_x> <n_y> page`: plots `n_x` graphs in the horizontal direction and `n_y` graphs in the vertical direction of the same page of the output `.ps` files, in case of more plots to be produced.

Graph appearance commands: here we specify some options for the style and the layout of the plot we want to produce.

- `<font_number> font`: sets the font type for the printed text and values, by setting the integer number `font_number` as:
  - `-25` for Times.
  - `-21` for Helvetica.
  - `-29` for `Courier`.
- `<'label'> glbl`: sets the graph label to the string `'label'`.
- `<is> <'species_label'> slbl`: sets the species label of species with index is to the string `'species_label'`.
- `<factor> lbsz`: rescales the size of the plot labels (both text and values) by `factor` times their default size, with `factor` real number.

Data plotting commands: here we specify which kind of plot we want to produce.

- `mesh`: produces a 2D plot of the **computational mesh**.
- `surf`: produces a 2D **patch plot** of the desired matrix data from the plasma solution, where each cell is plotted in the color associated with the value of the plotted quantity.
- `cont`: produces a 2D **contour plot** of the desired matrix data from the plasma solution, where contour lines are produced according to the distribution of values of the plotted quantity.
- `<iy> f.x`: produces a line **poloidal plot** of the desired matrix data along the cells with fixed integer radial index `iy` as a function of the poloidal coordinate x.
- `<ix> f.y`: produces a line **radial plot** of the desired matrix data along the cells with fixed integer poloidal index `ix` as a function of the radial coordinate y.

Graph format commands: here we specify the format of the plot we want to produce.

- `phys`: plots the desired matrix data from the plasma solution, as a 2D patch/contour plot, on the **physical mesh**, that is the actual 2D poloidal cross section (to use when `surf` or `cont` is called).
- `comp`: plots the desired matrix data from the plasma solution, as a 2D patch/contour plot, on the **rectangular mesh**, that is the 2D rectangular (x,y) domain (to use when `surf` or `cont` is called).
- `linf`: sets **linear scalings** for the plotted values from the plasma solution.
- `logf`: sets **logarithmic scalings** for the plotted values from the plasma solution.
- `<n_d> ndec`: sets the number of shown decades to `n_d` for logarithmic plots (to use when `logf` is called).
- `xpar`: plots the desired matrix data in function of the parallel distance for line poloidal plots (to use when `f.x` is called).
- `xpol`: plots the desired matrix data in function of the poloidal distance for line poloidal plots (to use when `f.x` is called).
- `mmid`: maps 1D radial plots at any poloidal location onto the radial coordinate at the outer midplane (to use when `f.y` is called).
- `linx`: sets linear scalings for the abscissae values in line poloidal/radial plots (to use when `f.x` or `f.y` is called).
- `logx`: sets logarithmic scalings for the abscissae values in line poloidal/radial plots (to use when `f.x` or `f.y` is called).
- `<factor> lwid`: rescales the line width in line poloidal/radial plots by `factor` times its default size, with `factor` real number (to use when `f.x` or `f.y` is called).

Plotting ranges: here we specify the real values corresponding to maxima and minima for the plot we want to produce.

- `<value> fmax`: **maximum function value** for plotting the desired matrix data.
- `<value> fmin`: **minimum function value** for plotting the desired matrix data.
- `<value> rmax`: **maximum major radius** for 2D patch/contour plots in the physical domain (to use when `phys` is called).
- `<value> rmin`: **minimum major radius** for 2D patch/contour plots in the physical domain (to use when `phys` is called).
- `<value> zmax`: **maximum height** for 2D patch/contour plots in the physical domain (to use when `phys` is called).
- `<value> zmin`: **minimum height** for 2D patch/contour plots in the physical domain (to use when `phys` is called).

- `<value> xmax`: **maximum poloidal coordinate index** for 2D patch/contour plots in the rectangular domain (to use when `comp` is called).
- `<value> xmin`: **minimum poloidal coordinate index** for 2D patch/contour plots in the rectangular domain (to use when `comp` is called).
- `<value> ymax`: **maximum radial coordinate index** for 2D patch/contour plots in the rectangular domain (to use when `comp` is called).
- `<value> ymin`: **minimum radial coordinate index** for 2D patch/contour plots in the rectangular domain (to use when `comp` is called).
- `<value> omax`: **maximum ordinate** for line poloidal/radial plots (to use when `f.x` or `f.y` is called).
- `<value> omin`: **minimum ordinate** for line poloidal/radial plots (to use when `f.x` or `f.y` is called).

Overlays to 2D and line plots: here we specify some additional graphic elements to show in 2D patch/contour plots and line poloidal/radial plots.

- `sep`: draws the flux surface defining the **separatrix**, in 2D plots.
- `<factor> sepw`: rescales the line width of the separatrix by `factor` times its default size, with `factor` real number.
- `<color_number> scol`: draws the separatrix line with color defined by the integer `color_number`.
- `vesl`: draws the outline of the **vacuum vessel**, in 2D plots.
- `<factor> vwid`: rescales the line width of the vessel outline by `factor` times its default size, with `factor` real number.
- `<color_number> vcol`: draws the vessel outline with color defined by the integer `color_number`.
- `grid`: draws of **overlaid mesh**, in 2D plots.
- `outl`: draws the outline of the **plasma domain**, in 2D plots.
- `<color_number> zcol`: defines the color to be used for the axes of line plots, defined by the integer `color_number`.
- `<color_number> dcol`: defines the color to be used for the lines in line plots, defined by the integer `color_number`.
- `<color_number> bcol`: defines the color to be used for the label bar title, defined by the integer `color_number`.
- `<color_number> gcol`: defines the color to be used for the graph title, defined by the integer `color_number`.

  Available colors are:
  - 1: black
  - 2: lavender
  - 3: dark violet
  - 4: slate blue
  - 5: royal blue
  - 6: deep sky blue
  - 7: aqua
  - 8: green
  - 9: celeste
  - 10: chartreuse
  - 11: green yellow
  - 12: yellow
  - 13: golden
  - 14: orange
  - 15: orange red
  - 16: red
  - 17: orchid

Plotting patch/contour parameters: here we specify some plotting and graphic options about the appearance of 2D patch/contour plots (to use when `surf` or `cont` is called).

- `<level_min> <level_max> <delta> clvd`: starts patch/contour levels at the real value `level_min`, increasing by the real value `delta` up to the real value `level_max`.
- `<level_min> <level_max> <N> clvn`: produces `N` patch/contour levels going from the real value `level_min` to the real value `level_max`.
- `<N> <level_1> … <level_N> clvs`: produces `N` patch/contour levels to the values arbitrarily specified by the real values `label_i`.
- `<'palette_file'> cltb`: sets the palette from which producing the colors for the different patch/contour levels by selecting it in the string `'palette_file'`; some of the available palette files, which will be automatically searched in the opportune directories, are:
  - `'rainbow_16.pal'`, `'rainbow_32.pal'`, `'rainbow_64.pal'`, useful for data fields which remain everywhere positive or negative.
  - `'jet_16.pal'`, `'jet_32.pal'`, `'jet_64.pal'`, useful for data fields which have a zero in the middle.
- `fclv`: sets filled contour levels in contour plots (to use when `cont` is called).
- `eclv`: sets empty contour levels in contour plots (to use when `cont` is called).
- `clvl`: switches off contour level lines to be drawn into filled contour plots (to use when `cont` is called).
- `lbon`: draws labels into contour plots (to use when `cont` is called).
- `lbof`: does not draw labels into contour plots (to use when `cont` is called).

Available plasma quantities: finally, here we specify the plasma quantity, stored on the matrix stack, that we want to plot (or eventually more plasma quantities to combine with arithmetic operations). In case the called matrix has also a `ns` dimension (meaning that the quantity is defined for more species), then more plots will be produced, for each species, in different pages of the output `.ps` file. Some of the most important are:

- Magnetic geometry:
  - `bb`: gives the total magnetic field (in T).
  - `bx`: gives the poloidal magnetic field (in T).
  - `by`: gives the radial magnetic field (in T).
  - `bx`: gives the toroidal magnetic field (in T).
  - `q`: gives the safety factor.
  - `pit`: gives the magnetic pitch.
- Basic plasma quantities:
  - `ne`: gives the electron density (in m^{-3}).
  - `na`: gives ion density, for each species (in m^{-3}).
  - `dab2`: gives the EIRENE atomic neutral density (in m^{-3}).
  - `dmb2`: gives the EIRENE molecular neutral density (in m^{-3}).
  - `te`: gives the electron temperature (in eV).
  - `ti`: gives the ion temperature (in eV).
  - `tab2`: gives the EIRENE atomic neutral temperature (in eV).
  - `tbm2`: gives the EIRENE molecular neutral temperature (in eV).
  - `pr`: gives the total static pressure (in Pa).
  - `p*`: gives the dynamic pressure (in Pa).
  - `po`: gives the electron potential (in V).
  - `ua`: gives the average parallel velocity, for each species (in m/s).
  - `mach`: gives the plasma Mach number.

- Fluxes and currents:
  - fnax: gives the poloidal particle flux, for each species (in s^{-1}).
  - fnay: gives the radial particle flux, for each species (in s^{-1}).
  - fmox: gives the poloidal momentum flux, for each species (in N).
  - fmoy: gives the radial momentum flux, for each species (in N).
  - fhex: gives the poloidal electron heat flux (in W).
  - fhey: gives the radial electron heat flux (in W).
  - fhix: gives the poloidal ion heat flux (in W).
  - fhiy: gives the radial ion heat flux (in W).
  - fhtx: gives the poloidal total energy flux (in W).
  - fhty: gives the radial total energy flux (in W).
  - fchx: gives the poloidal electric current (in A).
  - fchy: gives the radial electric current(in A).
- Boundary and recycling sources:
  - bcna: gives the boundary particle source, for each species (in s^{-1}).
  - bcmo: gives the parallel momentum source, for each species (in N).
  - bche: gives the boundary electron heat source (in W).
  - bchi: gives the boundary ion heat source (in W).
  - bcch: gives the boundary charge source (in A).
  - brna: gives the recycling particle source, for each species (in s^{-1}).
  - brmo: gives the recycling parallel momentum source, for each species (in N).
  - brhe: gives the recycling electron heat source (in W).
  - brhi: gives the recycling ion heat source (in W).
  - brch: gives the recycling charge source (in A).
- Volumetric particle sources/losses:
  - rsan: gives the particle source/loss by ionization (in s^{-1}).
  - rran: gives the particle source/loss by recombination (in s^{-1}).
  - rcxn: gives the particle source/loss by charge-exchange ion-neutrals collisions (in s^{-1}).
- Volumetric momentum sources/losses:
  - smaf: gives the momentum source from friction (in N).
  - smag: gives the momentum source from pressure gradient (in N).
  - rsam: gives the momentum source/loss by ionization (in N).
  - rram: gives the momentum source/loss by recombination (in N).
  - rcxm: gives the momentum source/loss by charge-exchange ion-neutrals collisions (in N).
- Volumetric energy sources/losses:
  - b2ra: gives the line radiation losses (in W).
  - b2br: gives the bremsstrahlung losses (in W).
  - neutrad: gives the neutrals radiation losses (in W).
  - dvue: gives the electron heating due to parallel velocity gradient (in W).
  - dvua: gives the ion heating due to parallel velocity gradient (in W).
  - visa: gives the viscous heating (in W).
  - joul: gives the joule heating (in W).
  - fraa: gives the inter-species friction heating (in W).
  - rsai: gives the energy source/loss by ionization (in W).
  - rrai: gives the energy source/loss by recombination (in W).
  - rcxi: gives the energy source/loss by charge-exchange ion-neutrals collisions (in W).

- Arbitrarily set volumetric sources/losses:
  - `sna`: gives the particle source, for each species (in s^{-1}).
  - `smo`: gives the parallel momentum source, for each species (in N).
  - `she`: gives the electron heat source (in W).
  - `shi`: gives the ion heat source (in W).
  - `sch`: gives the electric charge source (in A).
- Transport coefficients:
  - `d`: gives the total density-driven particle diffusivity (in m^2/s).
  - `dnneo`: gives the neoclassical density-driven particle diffusivity (in m^2/s).
  - `dp`: gives the total pressure-driven particle diffusivity (in m^2/s).
  - `vlax`: gives the total poloidal pinch velocity (in m/s).
  - `vlay`: gives the total radial pinch velocity (in m/s).
  - `vnneo`: gives the neoclassical pinch velocity (in m/s).
  - `visc`: gives the total viscosity (in m/kgs).
  - `cvsx`: gives the neoclassical poloidal viscosity (in m/kgs).
  - `cvsy`: gives the neoclassical radial viscosity (in m/kgs).
  - `kye`: gives the total electron thermal diffusivity (in m^2/s).
  - `kyi`: gives the total ion heat diffusivity (in m^2/s).
  - `chineo`: gives the neoclassical heat diffusivity (in m^2/s).
  - `sig`: gives the total electrical conductivity (in $\Omega^{-1}m^{-1}$).
  - `alf`: gives the total thermo-electric coefficient (in $\Omega^{-1}m^{-1}$).
- Other plasma quantities:
  - `wpe`: gives the plasma frequency (in rad/s).
  - `wce`: gives the electron cyclotron frequency (in rad/s).
  - `wci`: gives the ion cyclotron frequency (in rad/s).
  - `lnlam`: gives the Coulomb logarithm.
  - `nua`: gives the total collision frequency (in s^{-1})
  - `nustar`: gives the normalized collisionality.

Once that matrix data values from the plasma solution have been called, it is also possible to perform some simple analytic calculations on these data through the following **data manipulating functions**:

- `<matrix> ddx`: performs the derivative of the data values in `matrix` with respect to the poloidal coordinate x.
- `<matrix> ddy`: performs the derivative of the data values in `matrix` with respect to the radial coordinate y.
- `<matrix> intx`: performs the integration of the data values in `matrix` along the x direction, starting from the first poloidally indexed cell.
- `<matrix> inty`: performs the integration of the data values in `matrix` along the y direction, starting from the first radially indexed cell.
- `<matrix> psx`: divides the data values in `matrix` by the contact cell area between cells in the x direction.
- `<matrix> psy`: divides the data values in `matrix` by the contact cell area between cells in the y direction.
- `<matrix> pvol`: divides the data values in `matrix` by the cell volumes.
- `<matrix> abs`: takes the absolute value of the data values in `matrix`.
- `<matrix> sqrt`: takes the square root of the data values in `matrix`.
- `<matrix> exp`: takes the exponential of the data values in `matrix`.
- `<matrix> ln`: takes the natural logarithm of the data values in `matrix`.

Finally, if desired, it is possible to perform some **arithmetic commands** for acting on the matrix data values or for combining data values from more matrices:

- `<n> rm*`: multiplies the data values in matrix by the real number `n`.
- `<n> rm/`: divides the data values in matrix by the real number `n`.
- `<n> rm**`: raises the data values in matrix to the `n`-th power.
- `<matrix_1> <matrix_2> m+`: adds the data values of the matrices `matrix_1`, `matrix_2`.
- `<matrix_1> <matrix_2> m-`: subtracts the data values of the matrices `matrix_1`, `matrix_2`.
- `<matrix_1> <matrix_2> m*`: multiplies the data values of the matrices `matrix_1`, `matrix_2`.
- `<matrix_1> <matrix_2> m/`: divides the data values of the matrices `matrix_1`, `matrix_2`.
- `<matrix_1> <matrix_2> mmax`: takes the maximum between the data values of the matrices `matrix_1`, `matrix_2`.
- `<matrix_1> <matrix_2> mmin`: takes the minimum between the data values of the matrices `matrix_1`, `matrix_2`.

In addition to b2plot, we can also use a Python package, named **Solpspy**, for visualizing graphically the results of a run, as well as for extracting actual numerical tables or arrays of quantities from the plasma solution. This is made of various scripts, written in Python language, which produce different types of outputs by reading the information contained in the output files of a run (like `b2fstate`).
Once we are connected to the cluster, the Solpspy package is contained in `repository/python/solpspy/`; here, the `.py` files for all the various scripts are contained.

In order to use this tool for visualizing the results of a run, we must go to the actual directory of the desired run. Here, we can enter in the Python environment by executing

```
➢ ipython
```

After that, we must load in Solpspy the output data of the run by entering

```
➢ from solpspy import *
➢ run = SolpsData()
```

Then:

- In order to extract **tables or arrays of numerical quantities** describing the plasma solution at various points of the computational domain, we must enter:
  `> run.'f'[…,is]`
  for printing a **table** of values of the quantity `'f'` for the species is on each of the `nx+2` poloidal coordinates and on each of the `ny+2` radial coordinates.
  `> run.'f'[ix,…,is]`
  for printing an **array** containing the **radial profile** of the quantity `'f'` for the species `is` at the fixed poloidal coordinate `ix`.
  `> run.'f'[…,iy,is]`
  for printing an **array** containing the **poloidal profile** of the quantity `'f'` for the species `is` at the fixed radial coordinate `iy`.

- In order to draw **line plots of radial profiles**, first of all we must call the Matplotlib library, as
  `> import matplotlib.pyplot as plt`
  Then, for plotting the radial profile of a quantity `'f'` at the poloidal location `ix`, we can produce a plot against the radial cell number as
  `> plt.plot(run.'f'[ix,…,is])`
  or against the radial physical coordinate as
  `> plt.plot(run.dsrad[ix], run.'f'[ix,…,is])`
  In each case, we must enter
  `> run.plot.show()`
  for showing the produced plot, or
  `> run.plot.savefig('….eps')`
  for saving it as an `.eps` file.

- In order to draw **line plots of poloidal profiles**, first of all we must call the Matplotlib library, as
  `> import matplotlib.pyplot as plt`
  Then, for plotting the poloidal profile of a quantity `'f'` at the radial location `iy`, we can produce a plot against the poloidal cell number as
  `> plt.plot(run.'f'[…,iy,is])`
  or against the poloidal physical coordinate as
  `> plt.plot(run.dspol[iy], run.'f'[…,iy,is])`
  In each case, we must enter
  `> run.plot.show()`
  for showing the produced plot, or
  `> run.plot.savefig('….eps')`
  for saving it as an `.eps` file.

- In order to draw **contour plots** in the physical computational domain of the quantity `'f'`, first of all we must call the Matplotlib library, as
  `> import matplotlib.pyplot as plt`
  Then, we must enter
  `> run.plot.grid_data(run.'f'[…,…,is], cmap='…')`
  eventually choosing a particular color map, and then
  `> run.plot.show()`
  for showing the produced plot, or
  `> run.plot.savefig('….eps')`
  for saving it as an `.eps` file.

The quantities from the plasma solution which can be plotted by Solpspy, and the relative names with which these must be called, are the same as for b2plot.

14 - Post-processing tools for graphical output

15 - Post-processing tools for numerical balances

16 - Using MDSPLUS

There exist a routine which allows us to save the output of our simulations to an **MDSplus server** at Max-Planck-Institut für Plasmaphysik in Garching. In this way, simulation runs can be easily shared with other users.
This uses the program **b2md**, which requires as input the file **b2md.dat**, located in the `baserun/` folder corresponding to the considered run directory. This file contains some strings, among which:

- `exp`: it contains the name of the experiment being modelled in the considered case (AUG, JET, ITER, etc.)
- `shot`: it contains the shot number identifying the peculiar characteristics of the considered run.
- `time`: it specifies the time point of the experimental shot being measured.
- `comment`: it can contain free comments on the run.
- `timedep`: logical key; if set to `.true.`, it makes save on the MDSplus server data from `b2time.nc`.
- `tallies`: logical key; if set to `.true.`, it makes save on the MDSplus server data from `b2tallies.nc`.
- `snapshot`: logical key; it set to `.true.`, it makes save on the MDSplus server data from `b2fplasma`.

So, once input data and preferences are set in `b2md.dat`, in order to save a finished run on the MDSplus server, we go to the desired run directory, and here we create a symbolic link from the file `mesh.extra`, contained in the `baserun/` folder, by entering

➢ `ln -s ../baserun/mesh.extra .`

and then we execute

➢ **`save_mds`**

This should create a **`shotnumber.history`** file, located in the subfolder `b2md.exe.dir/` of the run directory containing the "**shot number**" for the run on the server (which enumerates all the runs uploaded here), which can be read just by entering

➢ `vim shotnumber.history`

The command `save_mds` always creates a new run save on the server, with a new shot number.
If, instead, we want to re-save a run with the same shot number, overwriting the run that was saved before, we must execute

➢ **`resave_mds`**

In the MDSplus server, the SOLPS tree consists of the main tree, `SOLPS`, and six-sub-trees:

- **`\SOLPS::TOP.DIAG`**
- **`\SOLPS::TOP.IDENT`**
- **`\SOLPS::TOP.MOVIES`**
- **`\SOLPS::TOP.SNAPSHOT`**
- **`\SOLPS::TOP.TALLIES`**
- **`\SOLPS::TOP.TIMEDEP`**

In particular, `DIAG` contains the simulated experimental diagnostic signals, `IDENT` contains the main information identifying a specific run, `SNAPSHOT` contains the detailed results from the end of the run, extracted from `b2fplasma`, `TALLIES` contains the various tallies extracted from `b2tallies.nc`, and `TIMEDEP` contains the various time-dependent scalar quantities extracted from `b2time.nc`.
The whole list of signals contained in these sub-trees once the run has been saved on MDSplus server can be found in the manual.

So, if we want to access one particular signal in one particular sub-tree for a run saved on the server, we must:

- access the server, whose complete name is `solps-mdsplus.aug.ipp.mpg.de:8001`.
- enter the `solps` tree.
- indicate the shot number of the desired run.
- indicate the desired signal, as `"top"."subtree_name"."signal_name".`

This can be done using standard experimental data viewing programs, for example **cview**, which can be launched just executing

➢ `cview`

and inserting the required data. This will produce the plot of the indicated signal.

Conversely, if a run has been archived into the MDSplus server, it is also possible to extract from it the information necessary to obtain again the necessary files from which to continue the run (that is `b2fstate` and `b2gmtry`), using the B2.5 program **b2rd**.
For doing this, first of all we check for the present of a `shotnumber.history` file in the considered run directory, sign that the run was actually uploaded on the MDSplus server. If present, the fetched run will be the one with the run number specified last in `shotnumber.history`.
At this point, we can execute

➢ **`b2run b2rd`**

which will create a `b2fstate` file using the information archived on the server. Then, for setting a new starting point, we execute

➢ `cp b2fstate b2fstati`
➢ `cp b2fgmtrt b2fgmtry`

after that we can re-launch the run.
So, this procedure will re-start the run from the point at which it was stored in the MDSplus server. However, since the other input files and run settings were not stored (differently from `b2fstate`), we should check that the input data present in all these files in the run directory are correct before re-launching the case.

In each case, at http://solps-mdsplus.aug.ipp.mpg.de/solps/, we could interactively search for all runs saved in the MDSplus server, searching it by:

- Shot number, if known.
- Ranges (minimum and maximum possible values) of important quantities.
- The user who uploaded the run.
- The particular considered experiment (AUG, JET, ITER, etc.)