G. V. Pereverzev, P. N. Yushmanov

ASTRA
Automated System for TRansport Analysis

IPP 5/98
February 2002

MAX-PLANCK-INSTITUT FÜR PLASMAPHYSIK

GARCHING BEI MÜNCHEN

ASTRA
Automated **S**ystem for **TR**ansport **A**nalysis

G. V. Pereverzev and P. N. Yushmanov*

* Aspen Technology, Inc.
12730 High Bluff dr.
San Diego, CA 92130
USA

IPP 5/98 February 2002

Contents

1 Introduction

The first transport simulations came up in the late sixties when the problem was formulated by B. B. Kadomtsev and O. P. Pogutse [1] and one of the first transport codes for a tokamak was created by Yu. N. Dnestrovski and D. P. Kostomarov [2]. One of the authors (GVP) of this report was among the users of this first code and is strongly influenced by the ideas implemented there. Further development of transport codes passed through several stages implied by developments in plasma theory, tokamak experiments and computing techniques.

In spite of great progress achieved in tokamak physics during years of research, the transport processes are still far from being really understood. Although theoretically derived transport models are constantly developing and their predictability improves, today's transport modeling is to a large extent empirical and based on scaling studies and trial-and-error approach. The latter assumes that a number of calculations should be accomplished until a reasonable agreement between experimental data and modeling results is achieved.

The transport code used for this goal should be easily tunable and adjustable for new tasks arising during the study. On the other hand, the set of equations to be solved numerically is strongly coupled: the same physical quantity appears in many different places in the code, so that even a minor change can invoke a long chain of related modifications. This feature suggested the value of an automated code builder, which was implemented in the first version of the Astra code at the late eighties in Kurchatov Institute in Moscow.

The current version has benefitted from years of experience in transport modeling, and comprises of the majority of tools which are required for the transport analysis of tokamak experiments. Nevertheless, the practice of transport modeling shows that new requirements are continuously arising. Therefore the code is continuously being developed, and new features are being added increasing its functionality. The ability of Astra to acquire new features, without modifying the background organization and user interface, has made it one of the most popular transport simulation tools in the fusion community. The code is opened for modifications and expansion into new areas. Dozens of people have already contributed their developments into Astra. The authors are thankful to all Astra users, and especially those who have contributed useful suggestions and have participated in the improvement of the code. Particular thanks are due to C. M. Roach, who read the manuscript and essentially contributed to its improvement.

The currently supported version 5.3 of the Astra code runs on Sun, IBM, DEC UNIX–workstations and on IBM PC (under Red Hat Linux). It uses UNIX C shell, Fortran/C compilers, and the X11 graphic libraries, all of which are available on all UNIX systems. Installation requires about 20 MB of system disk space (this is shared by all users in the same file system). In addition, the minimum requirement for each user is 10 MB, which can however be considerably exceeded depending on the size of personal libraries. The code is freely distributed and requires no special licenses.

The ASTRA (Automatic System for TRansport Analysis) code is more than a transport code in the conventional sense. It is a flexible programming system capable of creating numerical codes for predictive or interpretative transport modeling, for stability analysis or for processing experimental data. It is also not restricted to tokamak applications exclusively. Some features are added to versions 3.0 and onwards which allow for 1D stellarator modeling. The Astra system comprises

– an extensive library of modules describing different physical processes and data treatment

– a supervising shell which keeps track of changes in the libraries, processes user's requests and assembles different modules in a required application

– a graphic interface enabling transparent and user-friendly run control and flexible data presentation

– an interface to an experimental database

– a help tool with built-in descriptions and a set of examples which facilitate use of the code.

Transport codes generated by Astra have a modular organization so that each physical process or derived quantity is called by name which represents an object with encapsulated implementation. In the majority of cases, the user deals only with the object names and is not involved in the implementation details. The modular organization allows a variety of processes easily to be included in the code, achieving different levels of the description of experimental device required by the problem under consideration. The encapsulated implementation of objects also decreases the possibility of errors by ensuring that the same expression for a physical quantity is used in all places of the transport code.

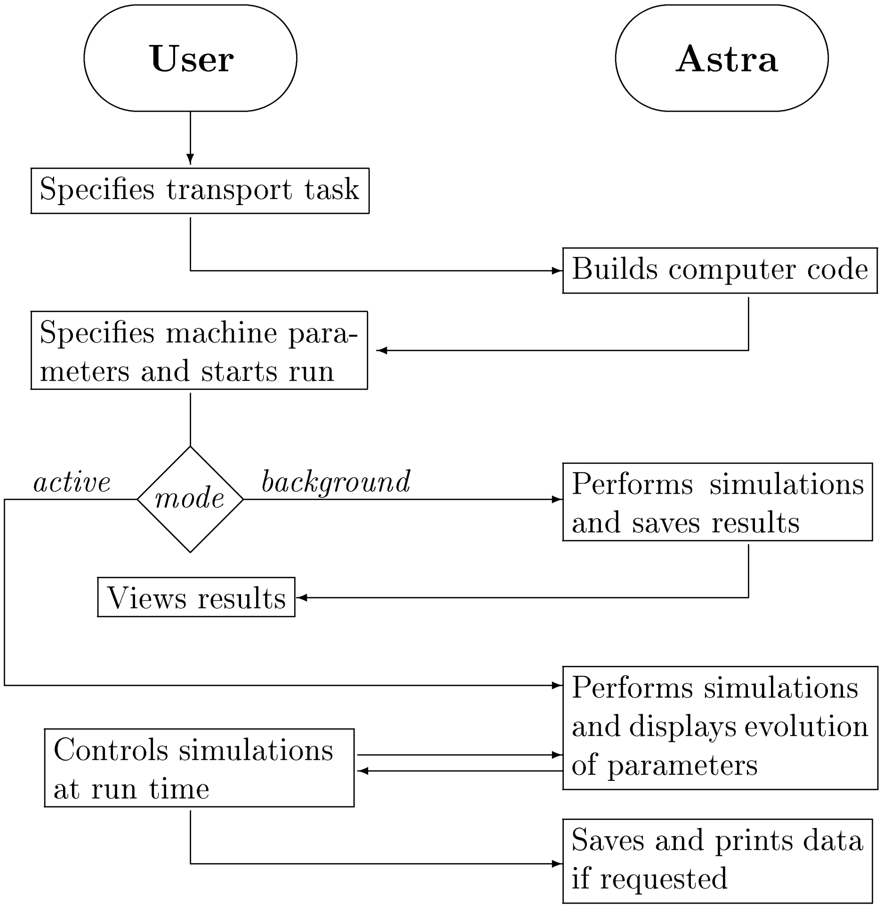
Another significant feature of the system is that it generates interactive codes. The user can observe the time evolution of plasma parameters during the program execution, interrupt it and change data presentation and control parameters influencing the course of the modeling. This enables the testing of different transport hypotheses at run time, and so increases the efficiency of the modeling. The code can also run in background mode, which is useful for routine calculations, or for time consuming cases, e.g. when many additional heating schemes are involved. All output information is then saved and can be retrieved afterwards to study the time evolution of the discharge.

The paper is organized as follows. Section 2 provides an extended introduction to the Astra system: it describes the general structure of the code and gives a bird's eye view of the code. The main physics background is presented in Section 3, which discusses the geometry of magnetic surfaces and related coordinate systems, the 2D equilibrium equation and a system of 1D transport equations and their closure (1.5D set of equations), appropriate boundary and initial conditions, possible energy and particle sources and sinks including an overview of the main schemes of auxiliary heating and current drive. This section also contains a number of formulae which are useful for setting transport modeling tasks. While providing the general background, this section is not directly related to the Astra code, and presents a reference for the detailed specification of transport problems.

Astra code variables and the adopted system of units are introduced in Section 4 (Astra Reference Guide), where the internal representation of the system of transport equations is also formulated. Further discussion in this section is devoted to the generation of user created transport models. Then different modules of the code which are not directly related to transport and the corresponding interfaces with the transport core are discussed. Section 5 (Astra User's Guide) describes the Astra modeling language used for the creation of transport models. This section also includes the description of the Astra experimental database format and the transfer of database information to simulation run. Starting the code, run-time operation and control are also discussed there.

2 Overview of the Astra code

There are several features, which the user has to be aware of, to understand the operation of Astra. First of all, Astra is not a ready-to-run application, but a tool for building customized computer codes for the solution of a variety of transport problems in magnetically confined plasmas. Customization of the code is the reason Astra's for extremely high flexibility and effectiveness. Such a programming paradigm requires two major steps: specification of the transport problem, and performing simulations for selected conditions. In addition simulations may be done in two different ways: depending on the nature of the problem under consideration, the user may select to run simulations in the active control mode or the background mode. The description of the user interaction with Astra is shown in the **high level usage diagram**.


User
Astra
Specifies transport task
Builds computer code
Specifies machine parameters and starts run
active
mode
background
Performs simulations and saves results
Views results
Performs simulations and displays evolution of parameters
Controls simulations at run time
Saves and prints data if requested

Let us now consider each step of operation with Astra in more detail. To set the transport task the user basically has to specify how to perform the simulations and what to display. Some adjustment of these settings may be done through run time control, but the overall simulation tasks of the generated code are determined during the initial transport model specification. The following features of Astra are used to specify the model description:

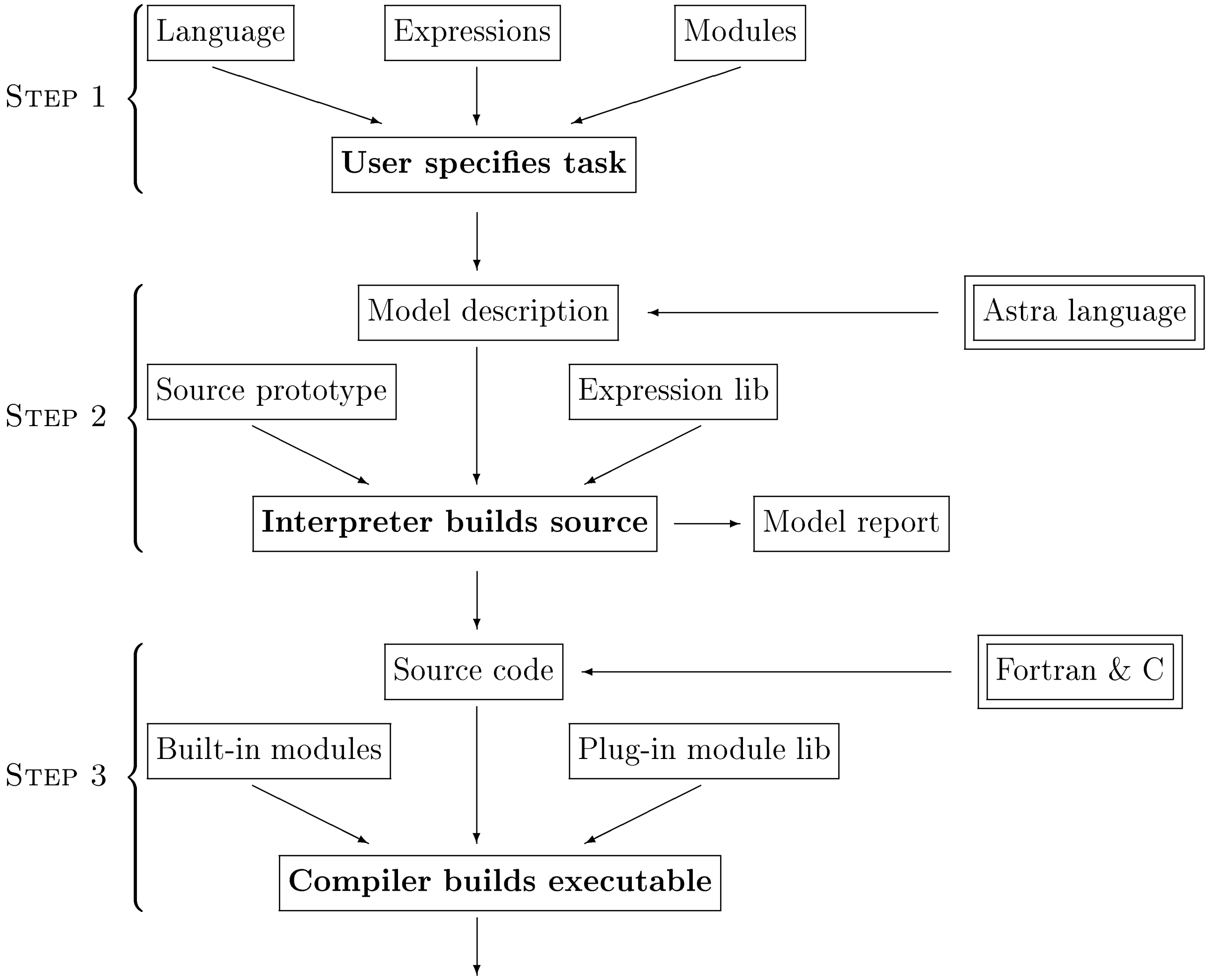
– transport modeling language

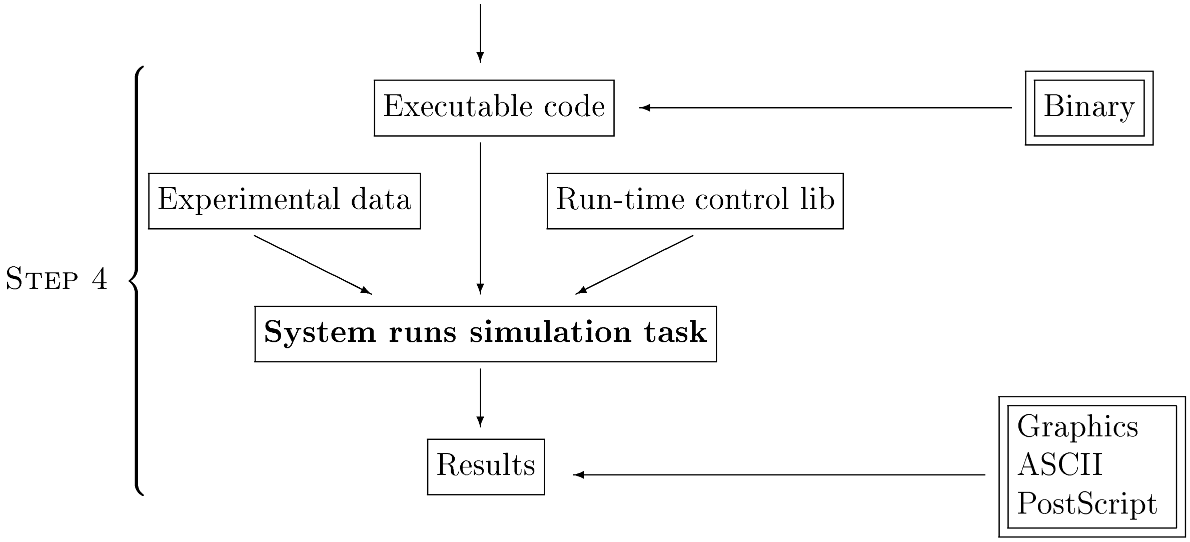
– standard expressions for physical quantities

– modules representing physical processes or features of the experimental device.

The modeling language is described in Section 5.2, standard expressions are listed in Sections 4.7 and 4.8, basic modules are discussed in Section 4.9.

To convert the model description into an executable code, Astra interprets the model description (source prototype) and compiles the generated source code. The sequence of operations and Astra objects involved in the whole process of creating the code and obtaining simulations results are shown in the **flow diagram**.





Specifying the simulation task is still a programming job, but it is simplified greatly by using Astra's high-level transport problem oriented language. To add a specific physical process to the model description the user has only to call it by name in the proper context. Analyzing these requests the Astra interpreter creates a source code where the expressions corresponding to the process names are used in the way specified by the context. These expressions are stored in the expression library and may be viewed or modified by the user. Similar things happen with modules describing auxiliary properties of experimental device such as heating or current-drive systems. Even if these modules involve a lot of functionality and controls, the user may easily add them to or remove them from his model description at his own discretion depending on the specifics of the transport task. Due to this add/remove option these modules are called plug-in. In addition, transport codes generated by Astra contains a number of built-in modules such as the time evolution driver, a graphical representation of output, a supervising control shell, and other features of the transport task which are always included in the simulation model. This part of the code provides the framework for the operation of the customized part and is hidden from the ordinary user. Modification of this part is rarely required, and an increased Astra functionality is available through the use of expression and plug-in module libraries.

A few words have to be said about executable transport codes created by Astra. Though executable code is customized for each specific task it always has the following set of standard features:

- it may be run either in the active or background modes, depending on the type of simulations performed by the user

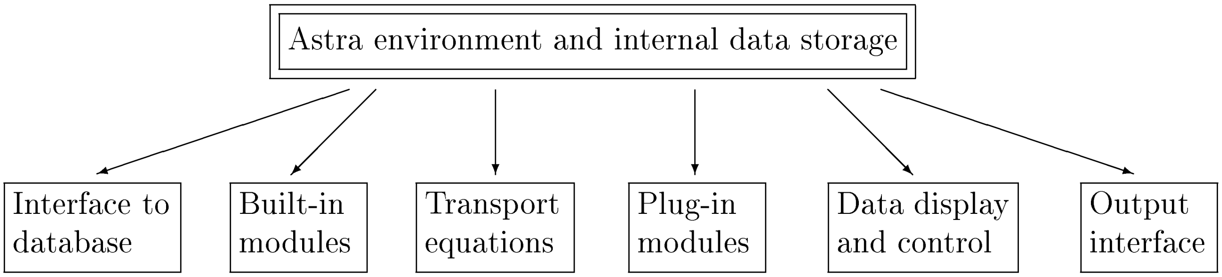
– in the active mode, code enables user specified run time control

– the code interfaces with the experimental database which includes discharges from many machines

– it has a modular organization.

The first three features are represented in the high level usage diagram. They will be discussed in more detail in Section 5. Modular organization is supported by the method of the code creation. Each block necessary for simulations is connected to other blocks through the Astra environment providing uniform control and exchange capability. Plug-in modules are included and connected through the same mechanism. Modular organization allows easy modification of the code and provides an efficient tool for building transport codes to any level of detail. The generated transport code has a transparent image in model description (prototype) and summarized in the model report, which in a compact form and without any omissions represents the whole complexity of the executable transport code.

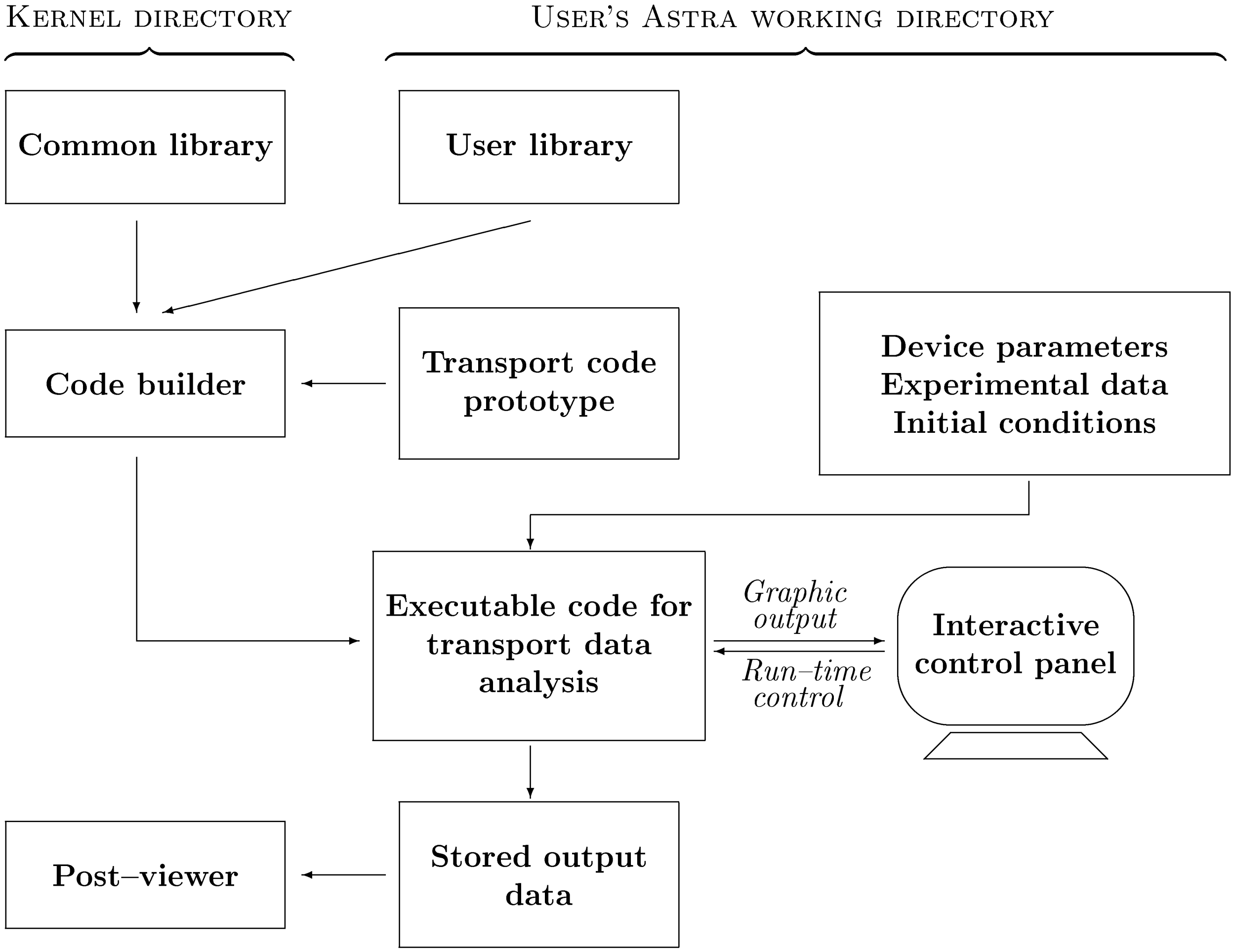
The structure of the Astra code is shown in the **module diagram**.



The other feature of Astra, which has to be discussed, is that Astra is a multi-user distributed system. Generally, there are the code kernel and several working directories for each user. The kernel of the code is shared by all users through a set of symbolic links to the kernel directory. This enables user's access to libraries of expressions, plug-in modules and object modules. On the other hand, all core parts of the code, such as the model interpreter, supervising shell, code building tools and built-in modules are present in the kernel only and consequently cannot be modified by an unauthorized user. This shared configuration saves disk space and more importantly keeps all users with the most recently updated version of the kernel while only one version of the kernel needs to be maintained.

The user owns all variable parts of the code such as model descriptions, executable codes, experimental libraries and the results of modeling. A trickier issue is the ownership of the libraries of standard expressions and physical processes. Basically they belong to the kernel directory and a user has read-only access to the sources files. This allows viewing and linking standard modules in a user created code but forbids any changes in these modules. The latter is however not really restrictive since every user can have his own library and use it together with the common one. Users may also create new modules or expressions for their own particular needs. In particular, a user can copy a common module to his own directory with a new name, then include it in his personal library thus acquiring all rights for subsequent modifications.

Physical locations of different blocks of the Astra system and scheme of their interaction are shown in the **component diagram**.



At the code installation, the user's Astra directory structure is created. We denote the root directory of this structure as `AWD` (Astra Working Directory). This directory is a current working directory during the code execution. In the code, relative addresses only are used, therefore, AWD can have arbitrary location with respect to user's home directory. Moreover, the user can simultaneously handle several versions of the Astra code, each linked to different kernels. However, if another name is not given explicitly then AWD is $HOME/astra . Specific components are stored in subdirectories which are given in Table 2.1:

Table 2.1: Astra working directory structure

Directory	Contents	File owner	User access
AWD/equ/	Transport code prototypes and *.log files	User	Allowed
AWD/exp/	Device parameters, experimental data base	User	Allowed
AWD/udb/	Input and output data in the form of u-files	User	Allowed
AWD/dat/	Computation results (ASCII)	User	Allowed
AWD/fml/	Simple expressions (formulae)	Mixed	Allowed
AWD/fnc/	Extended expressions (FORTRAN functions)	Mixed	Allowed
AWD/sbr/	Plug-in modules (FORTRAN subroutines)	Mixed	Allowed
AWD/tmp/	Included intermediate FORTRAN source files	User	Restricted
AWD/.res/	Computation results (post-viewer binary files)	User	Restricted
AWD/.tsk/	Executable transport codes	User	Restricted
AWD/for/	Service and built-in modules	System	Not allowed
AWD/.lbr/	Common libraries and executable modules	System	Not allowed
AWD/.exe/	Core and built-in modules	Mixed	Not allowed

In most practical work, the user deals mainly with the first 4 directories of this table. All these directories include user owned files in ASCII format only. The directory AWD/equ/ contains transport code prototypes (in what follows "models") and related files. The syntax of model files is discussed in Section 5.2. The directories AWD/exp/ and AWD/udb/ comprise of the experimental data base for interpretative modeling and initial data for predictive simulations. Astra format for input files is described in Section 5.4. Computation output files, either in ASCII or in Post Script format, are stored in AWD/dat/. Output data files have self-evident form and do not require explanations.

More advanced users may also add files to directories AWD/fml/, AWD/fnc/, AWD/sbr/. These directories can include both shared (soft links) and personal files. Every new file appearing in these directories is automatically compiled and added to a user library of object modules, so that it can then be used in the transport code.

The last six directories in the table are used by the Astra system but the unauthorized user has no access to change files in these directories. This part of the Astra system and its kernel are not described in the current report.

3 Background equations and formulae

3.1 Coordinate systems, surface averages and fluxes

In this section we provide a summary of the main relations used in this paper. Detailed derivations and discussions can be found in [3]. For the purpose of the geometry description, we consider the magnetic system as being toroidally symmetric. Small violations of symmetry, which are always present in real systems, are neglected in the equilibrium, but may taken into account in the confinement properties of the plasma.

For the description of tokamak geometry we use two coordinate systems. The first one is the cylindrical coordinate system $\{r, \varphi, z\}$ with the polar axis coinciding with the major axis of a torus. The second $\{a, \theta, \zeta\}$ is related to the magnetic geometry of the tokamak, and uses a "radial" variable a, which is an arbitrary label of a magnetic flux surface, and a poloidal angle θ, the specification of which does not play any role below. In order to make both systems right-handed we choose the toroidal angle ζ as $\zeta = -\varphi$.

We denote a flux surface, defined by the equation $a = \mathrm{Const}$, as S_a and introduce a volume integral over the interior, V, of this flux surface

$$V = \int\limits_{\mathrm{V}} dV = \int\limits_0^a da \int\limits_{\mathrm{S}_a} \frac{dS_a}{|\nabla a|} = \int\limits_0^a da \int\limits_0^{2\pi} d\zeta \int\limits_0^{2\pi} \sqrt{g}\, d\theta = 2\pi \int\limits_0^a da \int\limits_0^{2\pi} \sqrt{g}\, d\theta \tag{1}$$

where g is the determinant of the metric tensor $g = \left(\frac{D(x,y,z)}{D(a,\theta,\zeta)}\right)^2 = (\nabla a \nabla\theta \nabla\zeta)^{-2}$ and V is the volume of V. The flux surface average of an arbitrary function $f(\mathbf{r})$ is defined

$$\langle f \rangle = \frac{\partial}{\partial V} \int\limits_{\mathrm{V}} f\, dV = \frac{\partial}{\partial V} \int\limits_0^a da \int\limits_0^{2\pi} d\zeta \int\limits_0^{2\pi} \sqrt{g} f\, d\theta = 2\pi \frac{\partial a}{\partial V} \int\limits_0^{2\pi} \sqrt{g} f\, d\theta. \tag{2}$$

The symbols for partial derivatives are used here in order to emphasize that all surface functions (i.e. the functions of single space coordinate a) are also functions of time t . Equation (2) can equivalently be represented as

$$\langle f \rangle = \frac{\partial}{\partial V} \int\limits_{\mathrm{V}} f\, dV = \int\limits_{\mathrm{S}_a} f \frac{dS_a}{|\nabla V|} = \frac{\partial \psi}{\partial V} \oint f \frac{dl_\theta}{B_{\mathrm{pol}}} = \oint f \frac{dl_\theta}{B_{\mathrm{pol}}} \Big/ \oint \frac{dl_\theta}{B_{\mathrm{pol}}}, \tag{3}$$

where the poloidal flux ψ and the poloidal component, $B_{\rm pol}$, of the magnetic field $\mathbf{B}$ are determined in the next section. For any axisymmetric function, $f = f(a,\theta)$, one also has

$$\langle \mathbf{B}\cdot\nabla f\rangle = \langle {\rm div}\,(f\mathbf{B})\rangle = \frac{\partial}{\partial V}\int\limits_{\rm V} {\rm div}\,(f\mathbf{B})\,dV = \frac{\partial}{\partial V}\oint\limits_{{\rm S}_a} f\,\mathbf{B}\cdot d\mathbf{s} = 0. \tag{4}$$

Making use of Eq. (3) we find the area of a magnetic surface

$$S_a = \oint\limits_{{\rm S}_a} dS_a = \frac{\partial}{\partial V}\int\limits_{\rm V} |\nabla V| dV = \langle |\nabla V|\rangle = \frac{\partial V}{\partial a}\langle |\nabla a|\rangle\,. \tag{5}$$

Further useful properties of the averaging procedure are

$$\langle F(a) f(a,\theta)\rangle = F(a)\,\langle f\rangle\,, \qquad \frac{\partial V}{\partial a} = 2\pi\int\limits_0^{2\pi}\sqrt{g}\,d\theta, \qquad \langle {\rm div}\,\mathbf{g}\rangle = \frac{\partial}{\partial V}\langle \mathbf{g}\cdot\nabla V\rangle = \frac{\partial \varGamma}{\partial V}. \tag{6}$$

The last identity in Eq. (6) shows that the net flux $\varGamma$ of the vector $\mathbf{g}$ through the whole magnetic surface is given by

$$\varGamma = \int {\rm div}\,\mathbf{g}\,dV = \langle \mathbf{g}\cdot\nabla V\rangle = \frac{\partial V}{\partial a}\langle \mathbf{g}\cdot\nabla a\rangle \tag{7}$$

and the flux density

$$\gamma = \frac{\varGamma}{S_a} = \frac{\langle \mathbf{g}\cdot\nabla a\rangle}{\langle |\nabla a|\rangle}. \tag{8}$$

Assume now that a function of magnetic surface $F(a)$ describes a quantity which satisfies the general diffusion equation

$$\frac{\partial F}{\partial t} + \langle {\rm div}\,\mathbf{g}\rangle = \frac{\partial F}{\partial t} + \frac{\partial \varGamma}{\partial V} = S(a). \tag{9}$$

The local flux $\mathbf{g}$ can be taken in the form

$$\mathbf{g}(a,\theta) = F(a)\tilde{\mathbf{v}}(a,\theta) - \widetilde{D}(a,\theta)\nabla F(a). \tag{10}$$

On substitution in Eq. (9) one has

$$\frac{\partial F}{\partial t} = \frac{\partial}{\partial V}\left(\left\langle (\nabla V)^2\widetilde{D}\right\rangle \frac{\partial F}{\partial V} - F\,\langle \nabla V\cdot\tilde{\mathbf{v}}\rangle\right) + S(a). \tag{11}$$

This result suggests the introduction of two functions of a single argument a

$$D(a) = \frac{\left\langle (\nabla V)^2\widetilde{D}\right\rangle}{\langle (\nabla V)^2\rangle} = \frac{\left\langle (\nabla a)^2\widetilde{D}\right\rangle}{\langle (\nabla a)^2\rangle}, \qquad v(a) = \frac{\langle \nabla V\cdot\tilde{\mathbf{v}}\rangle}{\langle |\nabla V|\rangle} = \frac{\langle \nabla a\cdot\tilde{\mathbf{v}}\rangle}{\langle |\nabla a|\rangle}. \tag{12}$$

These definitions, obviously, do not depend on the particular choice of the magnetic surface label a. Equation (11) then reads

$$\frac{\partial F}{\partial t} = \frac{\partial a}{\partial V}\frac{\partial}{\partial a}\left[\frac{\partial V}{\partial a}\left\langle(\nabla a)^2\right\rangle\left(D\frac{\partial F}{\partial a} - \frac{\langle|\nabla a|\rangle}{\langle(\nabla a)^2\rangle}vF\right)\right] + S(a). \tag{13}$$

This presentation of the diffusion equation is employed in the Astra code. The total flux and the average flux density on a magnetic surface are

$$\mathit{\Gamma}(a) = \frac{\partial V}{\partial a}\left(\langle|\nabla a|\rangle\, vF - \left\langle(\nabla a)^2\right\rangle D\frac{\partial F}{\partial a}\right), \qquad \gamma(a) = vF - \frac{\langle(\nabla a)^2\rangle}{\langle|\nabla a|\rangle}D\frac{\partial F}{\partial a}. \tag{14}$$

If it is needed to introduce an “effective” diffusion coefficient as implied by experimental observations then none of Eq. (14) is appropriate. The coefficient can be derived from Eq. (13) as

$$D_{\text{eff}}(a) = \frac{\displaystyle\int\frac{\partial F}{\partial t}\,dV - \int S\,dV}{\displaystyle\langle(\nabla V)^2\rangle\frac{\partial F}{\partial V}} = \frac{\displaystyle\int\left(\frac{\partial F}{\partial t} - S\right)\frac{\partial V}{\partial a}\,da}{\displaystyle\langle(\nabla a)^2\rangle\frac{\partial V}{\partial a}\frac{\partial F}{\partial a}} \tag{15}$$

which under the additional assumption $v = 0$ coincides with $D(a)$ as defined by Eq. (12).

The definitions (12) are not unique. For instance, it is possible to write the average transport equation in the form

$$\frac{\partial F}{\partial t} = \frac{\partial a}{\partial V}\frac{\partial}{\partial a}\left[\frac{\partial V}{\partial a}\langle|\nabla a|\rangle\left(\widehat{D}\frac{\partial F}{\partial a} - vF\right)\right] + S(a) \tag{16}$$

rather than Eq. (13) with the argument that $\dfrac{\partial V}{\partial a}\langle|\nabla a|\rangle$ is the surface area and that the equation (16) and the flux density $\gamma = vF - \widehat{D}\;\partial F/\partial a$ appear in more “cylindrical” form. The resulting expression for the diffusion coefficient $\widehat{D}_{\text{eff}}(a) = \displaystyle\int(\partial F/\partial t - S)\,dV\Big/(S_a\;\partial F/\partial a)$ looks simpler and, probably, more natural, but it is dependent on a choice of the magnetic surface label “a”. In the Astra code, the invariant definition (15) is used.

We also introduce a local velocity, $\mathbf{u}_a$, of the $a(r,z) = \text{Const}$ surface as defined by the equation

$$\left.\frac{\partial a}{\partial t}\right|_{\vec{r}} + \mathbf{u}_a\cdot\nabla a = 0. \tag{17}$$

The local velocity $\mathbf{u}_f$ of another flux surface labeled by f (where $f = f(a,t)$) may be different from $\mathbf{u}_a$ but both are obviously related as

$$\left.\frac{\partial f}{\partial a}\right|_t(\mathbf{u}_a - \mathbf{u}_f)\cdot\nabla a = \left.\frac{\partial f}{\partial t}\right|_a. \tag{18}$$

Hence $\partial f/\partial t|_a$ expresses the relative motion of the flux surface f with respect to the flux surface a. In other words, a time derivative of a surface function depends on the assumption which quantity is taken as the 'radial' coordinate and considered to be fixed while computing $\partial/\partial t$.

3.2 Formulae for plasma current and magnetic field

Taking advantage of the toroidal symmetry and making use of the Maxwell equations we can express two three-dimensional vectors, the magnetic field $\mathbf{B}$ and the current density $\mathbf{j}$, in terms of two scalar functions $I(a)$ and $\Psi(a)$ (SI units are used):

$$\mathbf{B} = I\nabla\zeta + \frac{1}{2\pi}\left[\nabla\Psi\times\nabla\zeta\right], \tag{19}$$

$$\mathbf{j} = -\frac{\nabla\zeta}{2\pi\mu_0}r^2\mathrm{div}\frac{\nabla\Psi}{r^2} + \frac{1}{\mu_0}\left[\nabla I\times\nabla\zeta\right]. \tag{20}$$

Taking the scalar product of equations (19) and (20) with $\nabla\theta$ and integrating over the volume within magnetic surface using Eq. (1), we obtain[1]:

$$\psi = -\Psi = \frac{1}{2\pi}\int\limits_{\mathrm{V}}\mathbf{B}\cdot\nabla\theta\,d^3x \equiv \int\limits_{\mathrm{S}_\theta}\mathbf{B}\cdot d\mathbf{S}_\theta, \tag{21}$$

$$I = R_0B_0 - \frac{\mu_0}{4\pi^2}\int\limits_{\mathrm{V}}\mathbf{j}\cdot\nabla\theta\,d^3x \equiv R_0B_0 - \frac{\mu_0}{2\pi}\int\limits_{\mathrm{S}_\theta}\mathbf{j}\cdot d\mathbf{S}_\theta. \tag{22}$$

Here R_0 is the distance from the axis of the torus to a fixed point inside the plasma. Usually, one chooses R_0 as the geometric center of the vacuum chamber. B_0 is the vacuum magnetic field at $r = R_0$. While R_0 is constant, the magnetic field at this point may be a function of the time, $B_0 = B_0(t)$. The additive constants in Eqs. (21), (22) are chosen so that the function $\psi = 0$ at the magnetic axis and is physically the poloidal magnetic flux, while the function I is related to the poloidal current inside a magnetic surface. As long as no current flows outside the plasma, it follows that $I \equiv R_0B_0$ everywhere in space between the plasma edge and the toroidal field coils. Equations (21) and (22) can be viewed as definitions for functions ψ and I , respectively. We introduce also the dimensionless quantity

$$J \overset{\mathrm{def}}{=} \frac{I}{R_0B_0}. \tag{23}$$

[1] In what follows we use exclusively the quantity $\psi = -\Psi = -2\pi rA_\zeta$ with A_ζ being the toroidal component of the magnetic vector potential.

As long as poloidal (diamagnetic) plasma current is much smaller than the current in toroidal field coils, the quantity J is very close to unity inside the plasma and $J = 1$ outside it.

Both functions defined in Eqs. (21), (22) are so called surface functions that means that they depend on space coordinates only through the variable a . In evolving plasmas, all surface functions may depend also on time t, e.g., $V = V(a,t)$, $\psi = \psi(a,t)$. Each of them can be used as radial coordinates instead of a . It is convenient to introduce two further surface functions:

$$\Phi \overset{\text{def}}{=} \int\limits_{\mathrm{S}_\zeta} \mathbf{B}\cdot d\mathbf{S}_\zeta = \frac{1}{2\pi}\int\limits_{\mathrm{V}} (\mathbf{B}\cdot\nabla\zeta)\, d^3x = \frac{1}{2\pi}\int\limits_{\mathrm{V}} \frac{I}{r^2} d^3x, \qquad \rho \overset{\text{def}}{=} \sqrt{\frac{\Phi}{\pi B_0}}. \tag{24}$$

The first of them is the toroidal magnetic flux Φ. The second ρ, has the dimensionality of length and represents an effective minor radius.

As seen from Eq. (24) the quantities I and Φ can be expressed in terms of each other. On differentiation of Eq. (24) we have

$$\frac{\partial\Phi}{\partial V} = \frac{I}{2\pi}\left\langle\frac{1}{r^2}\right\rangle \qquad \text{or} \qquad 4\pi^2\rho R_0 = J\frac{\partial V}{\partial\rho}\left\langle\frac{R_0^2}{r^2}\right\rangle \tag{25}$$

which in large aspect ratio (where $\langle R_0^2/r^2\rangle \approx 1$) and at low plasma pressure (where $J \approx 1$) reduces to $\partial V/\partial\rho \approx 4\pi^2\rho R_0$ or $V \approx 2\pi^2\rho^2 R_0$ showing that the variable ρ can indeed be viewed as an effective (cylinder-like) minor radius of a magnetic surface. In what follows, we use ρ as the main radial variable and assume that all surface functions are functions of ρ and t , i.e. $\psi = \psi(\rho,t)$, $V = V(\rho,t)$ and so on. It means that time derivatives are understood as being computed with ρ kept constant:

$$\frac{\partial f}{\partial t} \overset{\text{def}}{=} \frac{\partial f(\rho,t)}{\partial t} \equiv \left.\frac{\partial f}{\partial t}\right|_\rho . \tag{26}$$

unless declared otherwise. Time derivatives taken at constant Φ also play a significant role and are related to the standard time derivative of Eq. (26) by the following relation:

$$\left.\frac{\partial f}{\partial t}\right|_\Phi = \left.\frac{\partial f(\rho(\Phi,t),t)}{\partial t}\right|_\Phi = \left.\frac{\partial f}{\partial t}\right|_\rho + \frac{\partial f}{\partial\rho}\left.\frac{\partial\rho}{\partial t}\right|_\Phi = \frac{\partial f}{\partial t} - \frac{\rho\dot{B}_0}{2B_0}\frac{\partial f}{\partial\rho}. \tag{27}$$

Some flux surface averages are also included in the transport equations and we introduce the following special notations:

$$V' = \frac{\partial V}{\partial\rho}, \quad G_1 \overset{\text{def}}{=} \left\langle(\nabla\rho)^2\right\rangle, \quad G_2 \overset{\text{def}}{=} \frac{V'}{4\pi^2}\left\langle\left(\frac{\nabla\rho}{r}\right)^2\right\rangle, \quad G_3 \overset{\text{def}}{=} \left\langle\frac{R_0^2}{r^2}\right\rangle \equiv \frac{4\pi^2\rho R_0}{JV'} \tag{28}$$

where all G_i are dimensionless.

For some applications the poloidal and toroidal components of the magnetic field **B** are required. They can be found from Eq. (19)

$$B_\mathrm{pol} = \frac{|\nabla\rho|}{2\pi r}\frac{\partial\psi}{\partial\rho}, \qquad\qquad B_\mathrm{tor} = \frac{I}{r}. \tag{29}$$

Both components are poloidally dependent. We introduce also the average cylinder-like poloidal field B_p, the rotational transform μ and the safety factor q as

$$B_\mathrm{p} \overset{\mathrm{def}}{=} \frac{1}{2\pi R_0}\frac{\partial\psi}{\partial\rho}, \qquad \iota \overset{\mathrm{def}}{=} \mu \overset{\mathrm{def}}{=} \frac{\partial\psi}{\partial\Phi} = \frac{1}{2\pi B_0\rho}\frac{\partial\psi}{\partial\rho} = \frac{B_\mathrm{p}R_0}{B_0\rho}, \qquad q \overset{\mathrm{def}}{=} \frac{1}{\mu} = \frac{\partial\Phi}{\partial\psi}. \tag{30}$$

Averaging Eq. (20) and using Eq. (6) we find the toroidal current density

$$j_\mathrm{tor} \overset{\mathrm{def}}{=} R_0\left\langle \mathbf{j}\cdot\nabla\zeta\right\rangle = \frac{2\pi R_0}{\mu_0 V'}\frac{\partial}{\partial\rho}\left(G_2\frac{\partial\psi}{\partial\rho}\right) = \frac{JG_3}{2\pi\mu_0\rho}\frac{\partial}{\partial\rho}\left(G_2\frac{\partial\psi}{\partial\rho}\right) \tag{31}$$

and the parallel current density

$$j_\| \overset{\mathrm{def}}{=} \frac{\langle\mathbf{j}\cdot\mathbf{B}\rangle}{B_0} = \frac{2\pi R_0}{\mu_0 V'}J^2\frac{\partial}{\partial\rho}\left(G_2 J^{-1}\frac{\partial\psi}{\partial\rho}\right). \tag{32}$$

Integration of Eq. (31) gives different relations for the toroidal current

$$\begin{aligned} I_\mathrm{pl} \overset{\mathrm{def}}{=} \int\limits_{\mathrm{S}_\zeta} \mathbf{j}\cdot d\mathbf{S}_\zeta \; &= \frac{1}{2\pi}\int\limits_{\mathrm{V}}(\mathbf{j}\cdot\nabla\zeta)\,d^3x = \frac{1}{2\pi R_0}\int\limits_0^\rho V' j_\mathrm{tor} d\rho = \frac{J}{2\pi R_0}\int\limits_0^\rho \frac{V'}{J^2}j_\| d\rho \\ &= \frac{G_2}{\mu_0}\frac{\partial\psi}{\partial\rho} = \frac{2\pi B_0}{\mu_0}\rho G_2\mu = \frac{2\pi R_0}{\mu_0}G_2 B_\mathrm{p} \end{aligned} \tag{33}$$

and

$$j_\mathrm{tor} = 2\pi R_0\frac{\partial I_\mathrm{pl}}{\partial V}, \qquad j_\| = 2\pi R_0 J^2\frac{\partial}{\partial V}\left(\frac{I_\mathrm{pl}}{J}\right) = J^2\frac{\partial}{\partial V}\int\limits_0^V \frac{j_\mathrm{tor}}{J}dV. \tag{34}$$

Also the formula for the poloidal field energy, which follows from Eqs. (28)-(29), is useful

$$W_\mathrm{I} = \int\limits_{\mathrm{V}} \frac{\mathbf{B}^2_\mathrm{pol}}{2\mu_0}d^3x = \frac{1}{2\mu_0}\int\limits_0^\rho \left(\frac{\partial\psi}{\partial\rho}\right)^2 G_2 d\rho. \tag{35}$$

The internal inductance L^W_i of the plasma current can be found from Eqs. (33), (35) as $L^W_\mathrm{i} = 2W_\mathrm{I}/I^2_\mathrm{pl}$. The frequently used dimensionless quantity which is related to the internal inductance per unit length, l_i, is given by $l_\mathrm{i} = \frac{4\pi}{\mu_0}\frac{L^W_\mathrm{i}}{2\pi R_0} = \frac{2L^W_\mathrm{i}}{\mu_0 R_0}$. Sometimes the internal inductance is defined as $L^\psi_\mathrm{i} = [\psi(a_\mathrm{pl}) - \psi(0)]/I_\mathrm{pl}$ which is obviously different from L^W_i.

3.3 Equilibrium equation

Plasma configuration in a tokamak is determined by the Grad-Shafranov equilibrium equation

$$\Delta^*\psi = r^2 \mathrm{div}\frac{\nabla\psi}{r^2} = -4\pi^2\left(\mu_0 r^2\frac{\partial p}{\partial\psi} + I\frac{\partial I}{\partial\psi}\right). \tag{36}$$

The first term on the right hand side contains the plasma pressure $p = p(\rho, t)$ where the contributions from all plasma species are included. The second term describes the poloidal diamagnetic current, which can be readily expressed in terms of the average toroidal current density. Eliminating $\Delta^*\psi$ from Eqs. (19), (20) and (36) we find

$$\mathbf{j} = -2\pi r^2\frac{\partial p}{\partial\psi}\nabla\zeta - \frac{2\pi}{\mu_0}\frac{\partial I}{\partial\psi}\mathbf{B}. \tag{37}$$

As mentioned above, the transport equations include parallel component of the current density. So we need to represent the quantity $\partial I/\partial\psi$ in terms of the parallel current density $j_\parallel$. A scalar product of Eqs. (19) and (37) gives

$$j_\parallel = -\frac{2\pi}{B_0}\left(I\frac{\partial p}{\partial\psi} + \frac{\langle B^2\rangle}{\mu_0}\frac{\partial I}{\partial\psi}\right), \tag{38}$$

where

$$\frac{\langle B^2\rangle}{B_0^2} = G_3 J^2 + \frac{4\pi^2 G_2\rho^2\mu^2}{V'}. \tag{39}$$

We can now express the right hand side of Eq. (36) in terms of functions which are provided by the transport equations

$$\begin{aligned}\Delta^*\psi &= \frac{2\pi\mu_0 R_0 J}{\langle B^2/B_0^2\rangle}j_\parallel + \frac{2\pi\mu_0 R_0^2}{B_0\rho\mu}\left(\frac{J^2}{\langle B^2/B_0^2\rangle} - \frac{r^2}{R_0^2}\right)\frac{\partial p}{\partial\rho} \\ &= 2\pi\mu_0 R_0\left[\frac{J}{\langle B^2/B_0^2\rangle}\left(j_\parallel + \frac{R_0 J}{B_0\rho\mu}\frac{\partial p}{\partial\rho}\right) - \frac{r^2}{B_0 R_0\rho\mu}\frac{\partial p}{\partial\rho}\right].\end{aligned} \tag{40}$$

The equilibrium equation (40) depends on t parametrically but not explicitly. Physically this means that we assume that the plasma is always in equilibrium, and we do not consider relaxation processes. The justification for this is that the relaxation to equilibrium is several orders of magnitude faster than all transport processes. We consider the coupling of this equation to the time-dependent transport equations in Section 3.11.

3.4 Toroidal electric field

For the toroidal vortex electric field $\mathbf{E}$ we have local (poloidally dependent) relation

$$\mathbf{E} = -\frac{1}{2\pi}\left.\frac{\partial\Psi}{\partial t}\right|_{\mathbf{r}}\nabla\zeta = \frac{1}{2\pi}\left.\frac{\partial\psi}{\partial t}\right|_{\mathbf{r}}\nabla\zeta. \tag{41}$$

The toroidal loop voltage $U_{\rm tor}$ is usually understood as the quantity measured by a fixed toroidally symmetric loop of a constant major radius $\mathbf{r}_l = (r_l, z_l)$ which is given by

$$U_{\rm tor} = 2\pi r\left(\mathbf{E}\cdot r\nabla\zeta\right) = \left.\frac{\partial\psi}{\partial t}\right|_{\mathbf{r}_l} = -\left.\left(\mathbf{u}_\psi\cdot\nabla\psi\right)\right|_{\mathbf{r}_l} \tag{42}$$

where $\mathbf{u}_\psi$ a local velocity of the constant- ψ surface (see Eq. (17)). Different kinds of motion contribute to $\mathbf{u}_\psi$ and, hence, to $U_{\rm tor}$.

First of all, we have

$$U_{\rm tor} = \left.\frac{\partial\psi(\Phi,t)}{\partial t}\right|_{\mathbf{r}_l} = \left.\frac{\partial\psi}{\partial t}\right|_{\Phi} + \mu\left.\frac{\partial\Phi}{\partial t}\right|_{\mathbf{r}_l}. \tag{43}$$

It can be shown [3] that the first term on the right hand side of Eq. (43) is related to an average longitudinal electric field $E_\parallel$ on the magnetic surface Φ

$$U_\parallel \stackrel{\rm def}{=} 2\pi R_0 E_\parallel \stackrel{\rm def}{=} \frac{2\pi R_0}{B_0}\langle\mathbf{E}\cdot\mathbf{B}\rangle = JG_3\left.\frac{\partial\psi}{\partial t}\right|_{\Phi} = JG_3\left(\left.\frac{\partial\psi}{\partial t}\right|_{\rho} - \pi\rho^2\mu\dot{B}_0\right). \tag{44}$$

In ideally conducting plasmas $E_\parallel = 0$ and Eq. (44) describes freezing of the fluxes ψ and Φ in one another. Using Eqs. (18), (25) and (42) we rewrite Eq. (43) as

$$\mu\left(\mathbf{u}_\psi - \mathbf{u}_\Phi\right)\cdot\nabla\Phi + \frac{V'}{4\pi^2 R_0\rho}U_\parallel = 0. \tag{45}$$

We conclude that the poloidal flux ψ moves through the toroidal flux Φ only in proportion to its resistive dissipation within the flux surface. The quantity $U_\parallel$ (or $E_\parallel$) is a measure of such dissipation. It is the quantity $U_\parallel$ which directly participates in and is provided by a transport code. It describes irreversible diffusive flow of the poloidal flux ψ through the toroidal flux Φ.

Making use of Eq. (44) we introduce another flux surface quantity, $U_{\rm pl}$, which is the voltage as seen in the coordinate system moving together with a ρ –surface:

$$U_{\rm pl} = \left.\frac{\partial\psi}{\partial t}\right|_{\rho} = \frac{U_\parallel}{JG_3} + \pi\rho^2\mu\dot{B}_0 = \frac{U_\parallel}{JG_3} + 2\pi\rho\mu B_0\left(\mathbf{u}_\rho - \mathbf{u}_\Phi\right)\cdot\nabla\rho, \tag{46}$$

where Eq. (18) and the definition Eq. (24) have also been used. The last equality in Eq. (46) shows that the relative motion of the Φ –surface with respect to the ρ –surface appears only due to variation of the external toroidal magnetic field B_0 . In most present day tokamaks the vacuum magnetic field can be viewed as time-independent, $\dot{B}_0 = 0$, and both flux surface labels, ρ and Φ, are physically equivalent. Usually, the coefficient in front of $U_{\|}$ in Eq. (46) is close to unity and, when $\dot{B}_0 = 0$, $U_{\rm pl}$ is close to $U_{\|}$. However, the difference between $U_{\|}$ and $U_{\rm pl}$ can become significant in a small aspect ratio tokamak and at high plasma pressure. For practical applications, the quantity $U_{\rm pl}$ is more convenient than $U_{\|}$ because, *in steady state*, $U_{\rm pl}$ does not vary over the minor radius ρ :

$$\frac{\partial U_{\rm pl}}{\partial \rho} = 0. \tag{47}$$

We consider now the second term on the right hand side of Eq. (43). First of all, we note that the pressure of the toroidal magnetic field in a tokamak is much higher than the plasma pressure. For this reason the toroidal flux Φ cannot be changed noticeably by any internal processes in the plasma. Consequently, the second term on the right hand side of Eq. (43) can only become appreciable when the external magnetic fields vary. For instance, varying of the poloidal control field causes plasma movement as a whole (compression in major radius), while increase of the toroidal field results in compression in minor radius. When any of these movements is made faster than characteristic transport times then the plasma is compressed adiabatically. We can further consider the equality

$$-\mathbf{u}_\rho \cdot \nabla\rho = \left.\frac{\partial \rho}{\partial t}\right|_{\mathbf{r}_l} = \frac{\partial \rho}{\partial V}\left.\frac{\partial V}{\partial t}\right|_{\mathbf{r}_l} + \left.\frac{\partial \rho}{\partial t}\right|_{V} = \frac{\partial \rho}{\partial V}\left(\left.\frac{\partial V}{\partial t}\right|_{\mathbf{r}_l} - \left.\frac{\partial V}{\partial t}\right|_{\rho}\right) \tag{48}$$

relating the “magnetic” variable ρ to the “geometric” variable V . We conclude that $\mathbf{u}_\rho$ comprises the plasma motion in space as a whole and a motion of the ρ –surface because of changes in the equilibrium (first and second terms in the brackets on the right hand side of Eq. (48), respectively).

In general case, using Eqs. (43)–(48) we can write the toroidal loop voltage as

$$U_{\rm tor} = \frac{1}{JG_3}U_{\|} + 2\pi\rho\mu B_0 \left.\frac{\partial \rho}{\partial t}\right|_{V} + \pi\rho^2\mu\dot{B}_0 + \frac{2\pi\rho\mu B_0}{V'}\left.\frac{\partial V}{\partial t}\right|_{\mathbf{r}_l} . \tag{49}$$

All terms here have clear physical significance. The left hand side of Eq. (49) is the loop voltage as measured in an experiment. The first term on the right hand side gives a resistive component. The second term describes evolution of the plasma surface due to a variation in plasma para- or dia-magnetism. It is of order $d\beta_{\rm tor}/dt$ and in a tokamak is usually negligibly small. The remaining two terms could be associated with the adiabatic compression in minor and major radii, respectively.

In order to express the measured loop voltage $U_{\rm tor}$ in terms of the calculated quantity $U_{\rm pl}$ we rewrite Eq. (49) as

$$U_{\rm tor} = U_{\rm pl} + 2\pi\rho\mu B_0 \left.\frac{\partial\rho}{\partial t}\right|_V + \frac{2\pi\rho\mu B_0}{V'} \left.\frac{\partial V}{\partial t}\right|_{\mathbf{r}_l} \tag{50}$$

and evaluate it at some point $\mathbf{r}_l = \mathbf{r}_B$ assuming that the point $\mathbf{r}_B$ belongs to the edge flux surface with a fixed (for instance, due to the boundary conditions) volume $V = V_B$ so that $\partial V_B/\partial t = 0$.

$$U_{\rm plB} = U_{\rm tor}\Big|_{\rho_B} = U_{\rm pl}(\rho_B) + 2\pi B_0\rho_B\mu(\rho_B)\frac{\partial\rho_B}{\partial t} = \left.\frac{\partial\psi}{\partial t}\right|_{\rho_B} + 2\pi B_0\rho_B\mu(\rho_B)\frac{\partial\rho_B}{\partial t}. \tag{51}$$

The last term on the right hand side of Eq. (51) is nonzero (although quite small) because the toroidal flux inside the plasma changes with any change in the profiles of plasma pressure or current density which gives rise to $\rho_B = \rho_B(t)$. This term is provided by a solution of the Grad-Shafranov equation. Evaluation of $U_{\rm tor}$ at arbitrary point $\mathbf{r}_l = (r_l, z_l)$ outside the plasma requires solution of the equilibrium equation in the gap between the plasma boundary and the position of measurement $\mathbf{r}_l$. This problem is not considered here and, in what follows, we assume that the edge loop voltage is calculated according to Eq. (51).

3.5 Joule heating and Ohm's law

Dissipation of the magnetic field energy within a flux surface is defined by

$$\begin{aligned}\int\limits_{\rm V} (\mathbf{j}\cdot\mathbf{E})\, d^3x &= \frac{1}{4\pi^2\mu_0}\int\limits_{\rm V} \left.\frac{\partial\psi}{\partial t}\right|_{\mathbf{r}} {\rm div}\frac{\nabla\psi}{r^2}\, d^3x = \frac{1}{4\pi^2\mu_0}\int\limits_{\rm V}\left(\left.\frac{\partial\psi}{\partial t}\right|_{\rho} + \frac{\partial\psi}{\partial\rho}\left.\frac{\partial\rho}{\partial t}\right|_{\mathbf{r}}\right){\rm div}\frac{\nabla\psi}{r^2}\, d^3x \\ &= \int\limits_0^\rho U_{\rm pl}\frac{\partial I_{\rm pl}}{\partial\rho}d\rho - \frac{B_0}{2\pi\mu_0}\int\limits_{\rm V}\rho\mu\left(\mathbf{u}_\rho\cdot\nabla\rho\right){\rm div}\frac{\nabla\psi}{r^2}\, d^3x.\end{aligned} \tag{52}$$

The first term on the right hand side of Eq. (52) can be transformed as

$$Q_{\rm J} = \int\limits_0^\rho U_{\rm pl}\frac{\partial I_{\rm pl}}{\partial\rho}d\rho = I_{\rm pl}U_{\rm pl} - \frac{1}{\mu_0}\int\limits_0^\rho \frac{\partial\psi}{\partial\rho}\frac{\partial^2\psi}{\partial t\partial\rho}G_2 d\rho = I_{\rm pl}U_{\rm pl} - \frac{1}{2\mu_0}\frac{\partial}{\partial t}\int\limits_0^\rho\left(\frac{\partial\psi}{\partial\rho}\right)^2 G_2 d\rho \tag{53}$$

which on account of the definition (35) can be represented as the conservation law for the energy of the poloidal magnetic field in the coordinate system moving together with a ρ – surface

$$\frac{\partial W_{\mathrm{I}}}{\partial t} = I_{\mathrm{pl}} U_{\mathrm{pl}} - Q_{\mathrm{J}}. \tag{54}$$

Here $I_{\mathrm{pl}} U_{\mathrm{pl}}$ is the Pointing flux relative to the $\rho = \mathrm{Const}$ flux surface.

We can also rewrite Eq. (53) as

$$Q_{\mathrm{J}} = \int\limits_0^\rho U_{\mathrm{pl}} \frac{\partial I_{\mathrm{pl}}}{\partial \rho} d\rho = \frac{1}{2\pi R_0} \int\limits_0^\rho U_{\mathrm{pl}} j_{\mathrm{tor}} V' d\rho = \frac{1}{2\pi R_0} \int\limits_0^V U_{\mathrm{pl}} j_{\mathrm{tor}} dV, \tag{55}$$

thus finding the density of magnetic energy dissipation as

$$P_{\mathrm{J}} = \frac{1}{2\pi R_0} U_{\mathrm{pl}} j_{\mathrm{tor}}. \tag{56}$$

This expression does not take into account the work done by the moving surface against the plasma pressure gradient described by the second term on the right hand side of Eq. (52). As discussed in the previous section this contribution is usually small unless the fast processes as the adiabatic compression are involved [3].

The longitudinal Ohm law is assumed to have the form

$$j_{\|} = \sigma_{\|} E_{\|} + j_{\mathrm{BS}} + j_{\mathrm{CD}} \tag{57}$$

which together with relations Eq. (32) and Eq. (44) gives a transport equation for the poloidal flux ψ considered in the next section. The averages of the bootstrap current density, $\mathbf{j}_{\mathrm{BS}}$, and the density of the current driven by external sources, $\mathbf{j}_{\mathrm{CD}}$, are determined similar to $j_{\|}$ in Eq. (32)

$$j_{\mathrm{BS}} = \frac{1}{B_0} \langle \mathbf{j}_{\mathrm{BS}} \cdot \mathbf{B} \rangle, \qquad j_{\mathrm{CD}} = \frac{1}{B_0} \langle \mathbf{j}_{\mathrm{CD}} \cdot \mathbf{B} \rangle. \tag{58}$$

3.6 Transport equations

The tokamak simulation code Astra like other codes comprises of a system of 1D diffusion equations for densities and temperatures of different plasma components, a 2D equilibrium equation, and, a variety of other modules to describe additional heating, current drive and other non-diffusive processes in tokamak plasmas. A set of the transport equations for a tokamak is derived in [3]. In this section, we discuss the set of equations which are implemented in the Astra code which differs from that presented in [3] by allowance for varying toroidal field B_0 and, consequently, adiabatic compression in minor radius.

The basic set of transport equations in the Astra code includes equations for the electron density n_e, electron temperature, T_e, ion temperature, T_i, and the poloidal flux ψ

$$\begin{cases} \dfrac{1}{V'}\left(\dfrac{\partial}{\partial t} - \dfrac{\dot{B}_0}{2B_0}\dfrac{\partial}{\partial\rho}\rho\right)(V'n_e) + \dfrac{1}{V'}\dfrac{\partial}{\partial\rho}\Gamma_e = S_e, \\[2ex] \dfrac{3}{2}(V')^{-5/3}\left(\dfrac{\partial}{\partial t} - \dfrac{\dot{B}_0}{2B_0}\dfrac{\partial}{\partial\rho}\rho\right)\left[(V')^{5/3}n_eT_e\right] + \dfrac{1}{V'}\dfrac{\partial}{\partial\rho}\left(q_e + \dfrac{5}{2}T_e\Gamma_e\right) = P_e, \\[2ex] \dfrac{3}{2}(V')^{-5/3}\left(\dfrac{\partial}{\partial t} - \dfrac{\dot{B}_0}{2B_0}\dfrac{\partial}{\partial\rho}\rho\right)\left[(V')^{5/3}n_iT_i\right] + \dfrac{1}{V'}\dfrac{\partial}{\partial\rho}\left(q_i + \dfrac{5}{2}T_i\Gamma_i\right) = P_i, \\[2ex] \sigma_\parallel\left(\dfrac{\partial\psi}{\partial t} - \dfrac{\rho\dot{B}_0}{2B_0}\dfrac{\partial\psi}{\partial\rho}\right) = \dfrac{J^2R_0}{\mu_0\rho}\dfrac{\partial}{\partial\rho}\left(\dfrac{G_2}{J}\dfrac{\partial\psi}{\partial\rho}\right) - \dfrac{V'}{2\pi\rho}\left(j_{BS} + j_{CD}\right). \end{cases} \tag{59}$$

If the main ion density n_i and its flux Γ_i are not explicitly defined (see Section 4) then it is assumed that $n_i = n_e/Z_i$ and $\Gamma_i = \Gamma_e/Z_i$. (Note that this approximation neglects impurities.) In line with the discussion in Section 2.1 all fluxes included in Eq. (59), i.e. the electron flux Γ_e , the electron heat flux q_e and the ion heat flux q_i , are considered as total fluxes through a flux surface $\rho = Const$. In order to close the system of equations the fluxes should be expressed in terms of thermodynamic forces taken as derivatives with respect to ρ . In a simulation, the fluxes as well as the conductivity $\sigma_\parallel$ and the bootstrap current density j_{BS} can either be taken from a transport theory or from experimental observations in a tokamak. Independently of the origin of the formulae used for the representation of the fluxes, it is assumed in the Astra code that the transport matrix has the following form

$$\begin{pmatrix} \dfrac{\Gamma_e}{n_e} \\ \dfrac{q_e}{n_eT_e} \\ \dfrac{q_i}{n_iT_i} \\ V'G_1\dfrac{\mu_0 j_{BS}}{B_p} \end{pmatrix} = -V'G_1 \begin{pmatrix} D_n & D_e & D_i & D_E \\ \chi_n^e & \chi_e & \chi_i^e & \chi_E^e \\ \chi_n^i & \chi_e^i & \chi_i & \chi_E^i \\ C_n & C_e & C_i & 0 \end{pmatrix} \cdot \begin{pmatrix} \dfrac{1}{n_e}\dfrac{\partial n_e}{\partial\rho} \\ \dfrac{1}{T_e}\dfrac{\partial T_e}{\partial\rho} \\ \dfrac{1}{T_i}\dfrac{\partial T_i}{\partial\rho} \\ \dfrac{E_\parallel}{B_p} \end{pmatrix}. \tag{60}$$

All the coefficients in the upper left third-order minor of the transport matrix have dimensions $[m^2/s]$ while the right column and the bottom row are dimensionless. The average poloidal magnetic field B_p is given by Eq. (30).

In practice it is rare to use all the terms on the right hand side of the transport system of equations Eq. (59) and the full transport matrix Eq. (60) are used in transport modeling. Moreover, often the transport set of four equations Eq. (59) is excessive. For instance, computation of electron density n_e can often be replaced with experimentally measured data. The Astra code provides an easy way to retain only those terms and equations of Eqs. (59), (60) which are necessary for the specific problem under consideration. This is coded as a set of special instructions and will be discussed further in Section 5.2. On the other hand, the four transport equations of Eq. (59) can be insufficient for some problems of interest, such as minority ion heating or helium ash removal. In such cases, the code can be supplemented with additional diffusion equations as will be described in Section 3.12.

In addition to the main set of the transport equations (Eq. (59)) the Astra code provides a variety of other modules for the description of different processes in tokamak plasmas and to compute the transport coefficients in Eqs. (59)-(60). Processes modeled by these modules include additional heating and current drive, minority species behavior, the plasma periphery and SOL plasma, MHD stability analysis etc.

3.7 Sources and sinks

The source of electrons S_e in the first of Eqs. (59) is

$$S_e = s_{ion}^{(e)} N n_e + s_{ion}^{(i)} N n_i - s_{rec} n_i n_e \tag{61}$$

where N is the density of neutral atoms of the working gas, s_{rec} is the recombination rate, and $s_{ion}^{(e)}$ and $s_{ion}^{(i)}$ are the rates of the impact ionization by electrons and ions, respectively. The sources of the electron P_e and of the ion P_i energy are defined as

$$\begin{aligned} P_e &= P_{OH} - P_\Gamma - P_{ei} - P_e^{RAD} - P_e^N + P_e^H + P^{FUS}, \\ P_i &= P_\Gamma + P_{ei} + P_i^N + P_i^H, \end{aligned} \tag{62}$$

include the Ohmic heating P_{OH} which was defined in Eq. (56), the power lost as radiation P_e^{RAD} , the powers of auxiliary heating of electrons P_e^H and ions P_i^H , heat exchange between electron and ion components

$$P_\Gamma = \left\langle (\nabla\rho)^2 \right\rangle \frac{\Gamma_e}{n_e} \frac{\partial\,(n_i T_i)}{\partial\rho}, \qquad P_{ei} = \frac{3m_e}{m_i}\frac{n_e}{\tau_e}(T_e - T_i) \tag{63}$$

and losses due to the atomic processes

$$\begin{aligned} P_e^N &= \left(s_{ion}^{(e)} N \mathcal{E}_{ion} + s_{rad} N \mathcal{E}_1 + \tfrac{3}{2} s_{rec} n_i T_e \right) n_e, \\ P_i^N &= \tfrac{3}{2}\left(s_{ion} N T_N + s_{cx} N (T_N - T_i) - s_{rec} n_e T_i \right) n_i, \end{aligned} \tag{64}$$

where T_N is the temperature of the neutral atoms, s_{cx} and s_{rad} are the charge exchange and radiation rates, respectively, $\mathcal{E}_{ion} = 13.6$ eV, $\mathcal{E}_1 = 10.2$ eV.

Although the bootstrap current j_{BS} is expressed by Eq. (60) in terms of thermodynamic forces, the bootstrap current together with the driven current j_{CD} can also be viewed as sources of poloidal flux. The driven current density j_{CD} as well as N , T_N , P_e^H and P_i^H must be provided by separate modules which are included in the code. Sawtooth oscillations and plasma behavior in a divertor region are further examples of processes which require separate treatment. Modules for these processes are also included in the code, but detailed discussion of all these modules falls outside the scope of this report. Below we discuss ways of adding new modules or replacing modules which already exist in the code.

To close the system of transport equations Eq. (59) we need, in addition to the sources and sinks of Eqs. (61)–(62), the transport matrix of Eq. (60), the electrical conductivity $\sigma_\parallel$ and the driven current j_{CD} , specify also the surface functions V', I, G_1, G_2, G_3 . Then with appropriate initial conditions and boundary conditions, we can determine the time evolution of the radial distributions of the electron density n_e , temperatures $T_{e,i}$ and current density.

3.8 Initial conditions

Initial conditions do not usually play a significant role in tokamak transport modeling, because normally steady-state discharge conditions are under study. However, for the analysis of transient processes, such as plasma current ramp-up, initial conditions may play an essential role. When available, measured experimental quantities may be used to supply distributions, but these are not always available, especially for the poloidal flux ψ and the corresponding current density. The simulation of some processes can be sensitive to the initial choice of the current density profile, and this choice may require care. It is usually sufficient to set initial conditions self-consistently, to give smooth plasma evolution during the initial moments of the run.

For the first three equations in Eq. (59) the natural initial conditions are

$$\begin{aligned} n_e(\rho,t)\big|_{t=0} &= n_{e0}(\rho), \\ T_e(\rho,t)\big|_{t=0} &= T_{e0}(\rho), \\ T_i(\rho,t)\big|_{t=0} &= T_{i0}(\rho). \end{aligned} \tag{65}$$

A similar initial condition for the poloidal flux would read $\psi(\rho,t)\big|_{t=0} = \psi_0(\rho)$. However, this is of no practical use because it is usually more convenient to prescribe the current density $j_\parallel$ or the rotational transform $\mu,$ rather than the poloidal flux ψ itself. Thus, appropriate initial conditions are alternative

$$\text{either} \qquad j_\parallel(\rho,t)\big|_{t=0} = j_0(\rho) \qquad \text{or} \qquad \mu(\rho,t)\big|_{t=0} = \bar{\mu}(\rho). \tag{66}$$

These conditions involve either the second or the first derivative of the unknown function $\psi(\rho)$, respectively. With either initial condition, $\psi(\rho)$ must be determined using either Eq. (32) or Eq. (30). A further well-defined quantity is the total plasma current $I_{\rm pl},$ which is always measured in experiments, while the current distribution during the early phase of a tokamak discharge is usually not known. Therefore it is reasonable to require that $I_{\rm pl}$ takes a given value and the normalizations of the current density or rotational transform profiles are adjusted to be self–consistent using

$$\begin{cases} j_\parallel(\rho,t)\big|_{t=0} = j_0(\rho) \\ \int\limits_0^{\rho_B} J^{-2} j_0 V' d\rho = 2\pi R_0 I_{\rm pl} \end{cases} \tag{67}$$

or

$$\mu(\rho,t)\big|_{t=0} = \frac{\bar{\mu}(\rho)}{\bar{\mu}(\rho_B)} \frac{\mu_0 I_{\rm pl}}{2\pi B_0 \rho_B G_2(\rho_B)}, \tag{68}$$

respectively. The boundary condition $J(\rho_B) = 1$ is used in Eq. (67).

Also convenient and useful is the initial condition with the plasma current distributed according to the steady-state condition $\dfrac{\partial^2\psi}{\partial t \partial\rho} = 0 \Rightarrow \dfrac{\partial\psi}{\partial t} = U_{\rm pl}(\rho) = Const$. Using the parallel Ohm's law Eq. (57) and assuming $\dot{B}_0(t=0) = 0$ we re-write the condition as

$$\left(j_\parallel - j_{BS} - j_{CD}\right)\Big|_{t=0} = \frac{2\pi\rho}{V'}\sigma_\parallel(\rho) U_{\rm pl} = C_\sigma \frac{\rho}{V'}\sigma_\parallel(\rho). \tag{69}$$

Similar to Eqs. (67), (68) the factor C_σ in Eq. (69) should be found from the condition that the total current is $I_{\rm pl}$. Although the current relaxation time is long in a tokamak, the stationary initial condition Eq. (69) is widely used when the direct measurements of the current density are not available. The condition of Eq. (69) is also physically relevant when non-diffusive processes cause fast current relaxation, or when the preceding long phase of relaxation is of no interest.

3.9 Boundary conditions for densities and temperatures

The boundary conditions for Eq. (59) at the magnetic axis $\rho = 0$ are imposed by the geometry of problem and the requirement that all fluxes should vanish at $\rho = 0$

$$\Gamma_e\big|_{\rho=0} = q_e\big|_{\rho=0} = q_i\big|_{\rho=0} = 0. \tag{70}$$

Boundary conditions at $\rho = \rho_B$ for the first three equations of the system Eq. (59) usually take one of two forms

$$\begin{aligned} n_e(\rho_B) &= n_{eB}(t) \qquad &\text{or}\qquad && \Gamma_e(\rho_B) &= \Gamma_{eB}(t),\\ T_e(\rho_B) &= T_{eB}(t) \qquad &\text{or}\qquad && q_e(\rho_B) &= q_{eB}(t),\\ T_i(\rho_B) &= T_{iB}(t) \qquad &\text{or}\qquad && q_i(\rho_B) &= q_{iB}(t). \end{aligned} \tag{71}$$

All the functions on the right hand sides in Eq. (71), besides explicitly depending on time t, can also depend on all other plasma parameters. The boundary conditions of Eq. (71) can be imposed in any combination as described in Section 4.4.

It worth noting now that, in general, the question of boundary conditions is not quite straightforward. Indeed, in the configuration space, the plasma boundary is supposed to coincide with some flux surface S_B. This boundary surface serves as an input to a solver of the Grad-Shafranov equilibrium equation. The solver returns a position of this surface in flux coordinate ρ_B. Even in the simplest case, when the plasma boundary S_B is fixed in space and does not move, the corresponding value ρ_B can vary in time depending on the plasma pressure and current density distributions. Therefore, the plasma boundary $\rho_B(t)$ moves in the frame of the Lagrangian flux coordinate ρ. Thus the problem under consideration is always a moving boundary problem and this is discussed in more detail in Section 3.11.

It is also possible that the required boundary conditions cannot be expressed explicitly in any of forms given in Eq. (71). This can be the case when more elaborated boundary conditions are imposed by considering SOL models. Correct description of these cases could require the use of MHD or kinetic theory. In the Astra code, these situations are implemented by calling special subroutines as described in Section 4.

3.10 Boundary condition for the poloidal flux

The left boundary condition for the fourth equation in Eq. (59) is straightforward, $\left.\dfrac{\partial\psi}{\partial\rho}\right|_{\rho=0} = 0.$ The boundary condition at $\rho = \rho_B$ is more complicated. Generally, the condition should be obtained by solving a magnetostatic problem in the exterior region (with respect to the plasma) taking into account all poloidal currents with their mutual inductances, magnetization of the iron core, currents induced in a vacuum chamber and so on. However, this would unreasonably complicate the problems under consideration. It is more practical to use one of the three possibilities described below.

3.10.1 Prescribed plasma current

This is the most frequently used boundary condition for transport modeling. Making use of Eq. (33) we write the condition as

$$\left.\frac{\partial\psi}{\partial\rho}\right|_{\rho=\rho_B} = \frac{\mu_0}{G_2(\rho_B)} I_{\rm pl}(t) \tag{72}$$

where $I_{\rm pl}(t)$ is the total plasma current with a prescribed time dependence.

3.10.2 Prescribed loop voltage

This is usually not a good choice of boundary condition. For instance, in Ohmic plasmas this boundary condition gives rise to a thermal instability. On the other hand, this instability is very slow and can be easily stabilized. Therefore, this condition is also included in the code as a possible option. In the simplest case where the plasma boundary is fixed (see discussion in Section 3.4) it reads

$$\left.\frac{\partial\psi}{\partial t}\right|_{\rho=\rho_B} = U_{\rm plB}(t) \tag{73}$$

with $U_{\rm plB}$ being the prescribed boundary loop voltage.

3.10.3 External circuit equation

This provides a boundary condition which includes the two previous conditions as particular cases. It uses a simplified description of an external circuit with the single equation

$$U_{\rm ext} = \frac{d}{dt}\left(L_{\rm ext} I_{\rm pl}\right) + U_{\rm plB} \tag{74}$$

where $L_{\rm ext}$ is the external inductance of the plasma, and $U_{\rm plB} = U_{\rm pl}(\rho = \rho_B)$ is the loop voltage calculated at the plasma edge Eq. (51). $U_{\rm ext}$ is a given function of time associated with the voltage produced by the primary coil.

3.11 Closure of the equilibrium and transport equations

The four equations of Eq. (59) do not comprise a closed system, even when all the fluxes and right hand sides are determined. Indeed, the functions $V(\rho,t)$, $J(\rho,t)$, $G_{1,2}(\rho,t)$ included in Eq. (59) still remain unknown. Moreover, the independent variable ρ is also an unknown function of space coordinates and time: $\rho = \rho(r,z,t)$. It is the solution of the Grad-Shafranov equation Eq. (36) which gives the instant relations of all flux coordinates, such as ρ , V , J , etc., to one another and to the laboratory coordinate system $\{r,z\}$. These relations can vary in time due to time evolution of plasma parameters as described by Eq. (59). Hence to study the plasma evolution in a tokamak we should solve the set of one-dimensional transport equations (59), (60) simultaneously with the two-dimensional equilibrium equation (40). [2]

First of all, we limit our consideration to the internal equilibrium problem with a prescribed plasma boundary. This boundary is not supposed to be fixed in time so that the entire plasma can move according to a given law. However, we do not consider the external fields and currents which are required to provide this boundary evolution and the plasma movement because the solving of external equilibrium problem is beyond the scope of the Astra code.

The closing procedure can be seen as follows. At each time step, the solution to the transport equations (59), (60) provides the functions $p(\rho)$ and $j_\parallel(\rho)$ on the right hand side of Eq. (40). These functions are determined on the interval $0 \le \rho \le \rho_B$. On

[2] Because of this combination of 1D and 2D equations the transport codes are often referred to as 1.5D codes.

the other hand, the solution of Eq. (40) is sought within the prescribed plasma boundary S_B. However, the requirement that this boundary S_B must coincide with the flux surface $\rho = \rho_B$ would overdetermine the problem. According to the particular experimental conditions, it is possible to adopt different ways of treating this contradiction. For instance, the overdetermination can be removed if the surface S_B is given by a set of points $\{r_i, z_i\}$ while in between it is allowed to be arbitrary. This approach is practically useless because it suppose a corrugated plasma surface which most probably is not compatible with the external magnetic system.

More practical would be to consider the whole set of boundary parameters $\{\rho_B, r_i, z_i\}$ as approximate which has to be fitted with some accuracy. If $\{r_i, z_i\}$ is firmly prescribed and ρ_B is free then the formulation will be referred as the prescribed plasma boundary. The case with fixed ρ_B and flexible $\{r_i, z_i\}$ will be called the adjustable plasma boundary. These two approaches are discussed below.

3.11.1 Prescribed plasma boundary

Presently, the following procedure is implemented in the Astra code. The equilibrium equation is solved requiring that S_B coincides with one of the ρ -surfaces, say $\rho = \hat{\rho}_B$, which can be either external or internal with respect to ρ_B . In the case $\rho_B < \hat{\rho}_B$, there is a gap between the current carrying plasma and the presumed boundary S_B . In the case $\rho_B > \hat{\rho}_B$ the excessive plasma layer should be scraped off. In both cases, the functions $p(\rho)$ and $j_{\|}(\rho)$ are re-defined on the interval between ρ_B and $\hat{\rho}_B$ so that in the region $\min(\rho_B, \hat{\rho}_B) \le \rho \le \max(\rho_B, \hat{\rho}_B)$: $p(\rho) = p(\rho_B)$ and $j_{\|}(\rho) = 0$. In addition, a surface skin current is added at $\rho = \hat{\rho}_B$ in order to keep the total plasma current unchanged. Solving Eq. (40) is then iteratively repeated until $\hat{\rho}_B$ remains constant with sufficient accuracy. The new boundary position is selected as $\rho = \hat{\rho}_B$. The described procedure does not change when a plasma boundary moves or varies its shape.

In practical calculations, at each time step it is sufficient to use the value of $\hat{\rho}_B$ returned by the equilibrium solver after one iteration as a new position of the boundary point ρ_B . Indeed, as it has already been discussed, modifications in current density, pressure and the corresponding changes in plasma diamagnetism cause time variation of the plasma size. Consequently, the time variation of $\rho_B(t)$ is of the order $\dot{\beta}_{tor}\Delta t$ and hence is very small.

Typically, the relative variation of ρ_B is as small as 10^{-3} per time step.

3.11.2 Adjustable boundary

More relevant would be to consider the prescribed boundary shape as approximate which has to be fitted with some accuracy. Then the different points on the boundary can be matched with different admissible errors according to their confidence weight. A trivial example of such an approach would be to fix the innermost point on the boundary (associated with internal limiter) and, adjusting the rest of the boundary S_B, allow the entire plasma to shift in order to avoid a contact with a limiter and preserve ρ_B unchanged.

This option assumes that the plasma boundary is given with one free parameter, which may be the boundary elongation, plasma shift, etc. When plasma is expanding then the freedom is used in order to fulfill the condition that the plasma remains within the flux surface ρ_B and thus avoids plasma contact with a limiter. All other features of the code remain the same as in the case with prescribed boundary shape. This version is easier for a numerical implementation and, probably, more relevant from the physical point of view. However, it requires a specification of the admissible freedom. We note in conclusion that all the details have significance which is more formal than practical, because the real variation in $\rho_B(t)$ is negligibly small.

3.12 Auxiliary transport equations

Usually, a tokamak plasma contains a variety of different species. These might be hydrogen isotopes, helium ash, or other impurities. Each of them can be characterized by its own density and temperature. Non-Maxwellian populations of the main plasma components can also be treated as independent species. Densities, temperatures, velocities of rotation and other thermodynamic characteristics of plasmas are believed to obey the law of collisional diffusion. For treatment of all these processes not included in Eq. (59) the Astra code provides a set of additional transport equations. The present practice of transport tokamak modeling never uses the transport matrix of Eq. (60) in full and most present-day tokamaks work with a time-independent toroidal magnetic field. Therefore, these additional equations have a truncated form

$$\frac{\partial}{\partial t}(V'f_j) = \frac{\partial}{\partial \rho}\left[V'\left\langle(\nabla\rho)^2\right\rangle\left(D_j\frac{\partial f_j}{\partial \rho} - v_j f_j\right)\right] + V'S_j \qquad j = 1, 2, 3. \qquad (75)$$

In version 5.0 of the Astra code the number of equations (75) cannot exceed 3, but this restriction can easily be lifted. The initial and boundary conditions for Eq. (75) are similar to those for the first three equations in Eq. (59)

$$f_j(\rho,t)\big|_{t=0} = f_{j0}(\rho) \tag{76}$$

and

$$\begin{cases} \left(D_j\dfrac{\partial f_j}{\partial\rho} - v_jf_j\right)\Bigg|_{\rho=0} = 0, \\ f_j(\rho_B,t) = f_{jB}(t) \qquad \text{or} \qquad \left(-D_j\dfrac{\partial f_j}{\partial\rho} + v_jf_j\right)\Bigg|_{\rho_B} = \Gamma_{jB}(t). \end{cases} \tag{77}$$

3.13 Equation for gas puff neutrals

The subroutine solves a kinetic equation for a neutral distribution function f_N

$$v\frac{\partial f_N}{\partial x} + \left(s_{ion}^{(e)}n_e + s_{ion}^{(i)}n_i + s_{cx}n_i\right)f_N = \frac{\sqrt{3}}{2}n_i\left(s_{cx}N + s_{rec}n_e\right)\delta\left(v \pm \frac{v_{Ti}}{\sqrt{3}}\right), \tag{78}$$

where $v_{Ti} = \sqrt{2T_i/m_i}$. The factor $\sqrt{3}$ is included in Eq. (78) in order to describe isotropic velocity distribution of the charge exchange neutrals. The equation (78) is solved in a slab geometry assuming that the plasma slab thickness is equal to twice the minor radius in the equatorial cross-section $2a_B$. This approximation is reasonable when the mean free path of a neutral is much smaller than a_B . In present-day tokamaks, the condition is usually fulfilled with a large margin. As long as the transit time of the neutral atoms is quite small the problem is treated as steady state.

It is assumed that the incoming neutral particles have a velocity distribution

$$\begin{aligned} f_N(x=-a_B,v)\Big|_{v>0} &= N_1\delta\left(v-v_1\right) + N_2\delta\left(v-v_2\right), \\ f_N(x=a_B,v)\Big|_{v<0} &= N_1\delta\left(v+v_1\right) + N_2\delta\left(v+v_2\right), \end{aligned} \tag{79}$$

with $v_{1,2} = \sqrt{2E_{1,2}/m_i}$. Eq. (78) is solved as described in [2] and the neutral density and temperature are then found as

$$N(x) = \int f_N(x,v)dv \qquad \text{and} \qquad T_N(x) = \frac{1}{N(x)}\int\frac{m_iv^2}{2}f_N(x,v)dv. \tag{80}$$

3.14 Other Astra compatible packages available by request

A number of other modules are incorporated in the Astra code. Some of them are quite complicated and time consuming. The code is continually supplemented with newer ones. Only few of these additional packages are included in the standard Astra version. Sometimes the interfaces only are provided, while the packages can be obtained directly from their authors. A description of all these modules is out of the scope of this report. Only a list of the most useful modules with the names of the contact persons is presented here.

3.14.1 Additional heating and current drive

Neutral beam heating and current drive [7]. The NBI Heating and CD package of the Astra code was developed by A. R. Polevoi (polevoy@nfi.kiae.su). A simplified time independent version of the package is distributed with the Astra code. It is briefly described in Section 4.6.2. A more advanced time dependent version is available.

Electron cyclotron heating and current drive [8]. The wave equation for the electromagnetic waves in a cold plasma is solved in the Gaussian beam approximation. The wave absorption at the frequency of the electron cyclotron resonance (ECR) and its harmonics is calculated for a Maxwellian plasma in the weakly relativistic approximation. The driven current is estimated using the adjoint technique. This package is developed by E. Poli (Emanuele.Poli@ipp.mpg.de, see also http://www.aug.ipp.mpg.de/˜emp/torbeam.html).

Lower hybrid heating and current drive [9]. The wave equation is solved for the lower-hybrid frequency range in the cold plasma approximation using the geometric optics technique. The Fokker–Planck equation is solved in the 1D approximation. Ion and $\alpha-$ particle absorption is also included. (A. N. Saveliev, saveliev@ans.ioffe.rssi.ru).

Ion cyclotron heating [10]. The wave equation for the IC frequency range is solved in the eikonal approximation. The code takes into account minority and ion cyclotron harmonic heating, ion-ion hybrid resonance heating and wave absorption due to coupling to the acoustic and ion-Bernstein waves. The code also includes a module for the antenna-plasma coupling which provides the boundary conditions for the ray tracing. (M. Brambilla, Marco.Brambilla@ipp.mpg.de).

3.14.2 Transport coefficients

The Astra library includes about one hundred different neoclassical and anomalous transport coefficients which can be expressed as formulae. Recently, more complicated routines for evaluation of the transport coefficients were developed. Some of them can be included in the Astra code as separate modules.

Neoclassical transport `NCLASS` [11]. A matrix of neoclassical transport coefficients for a multi-species tokamak plasma is calculated. (W.A. Houlberg, houlbergwa@ornl.gov)

Turbulent transport The following three transport models provide transport matrix due to toroidal ion temperature gradient (ITG) modes and trapped electron modes (TEM).

- Weiland-Nordman model [12]. (J. Weiland, elfjw@elmagn.chalmers.se).
- IFS/PPPL model [13]. (W. Dorland, bdorland@pppl.gov).
- GLF23 model [14]. (J. Kinsey, kinsey@apollo.gat.com).

3.14.3 Impurity transport and radiation

The subroutine `STRAHL` [16] describing transport and radiation of impurity ions (R. Dux, Dux@ipp.mpg.de, http://www.aug.ipp.mpg.de/˜Ralph.Dux/home.html) can be linked with the code Astra. No cross influence of impurities on the main plasma transport is included.

3.14.4 Stability analysis

The code provides interfaces to stability codes as `PEST, MAST, CASTOR, GARBO`. For this codes, a special input file is written for specified time slices which can be used for a post-run stability analysis. In addition, more simplified routines are available for on line evaluation of instability induced transport.

Sawtooth oscillations (subroutine `MIXINT`**.)** Sawtooth description [18] based on the Kadomtsev theory for magnetic field line reconnection [17, 18] is included in the standard Astra distribution.

Tearing modes (subroutine `ISLAND`**.)** Simplified Δ' analysis for saturated island width in a cylindrical plasma is provided (A.N.Chudnovskij, chdn@wowa.net.kiae.su).

4 Reference guide

4.1 Units and variables

The following set of units, based on SI with some exceptions, is adopted in the Astra code.

Table 4.1. System of units

Unit	Quantity
m	length
m^2	area
m^3	volume
s	time
m/s	velocity
m^2/s	diffusion
$10^{19}m^{-3}$	particle density
$10^{19}m^{-3}s^{-1}$	particle source
$10^{19}s^{-1}$	particle flux
$10^{19}m^{-2}s^{-1}$	particle flux density
keV	temperature
s^{-1}	collision frequency

Units	Quantity
MW	power, energy flux
$MW{\cdot}m^{-3}$	power density
$MW{\cdot}m^{-2}$	density of energy flux
MJ	energy content
T	magnetic induction
V·s	magnetic flux
V	voltage
MA	current
$MA{\cdot}m^{-2}$	current density
μHn	inductance
$(\mu Ohm{\cdot}m)^{-1}$=MS/m	conductivity
GHz	RF frequency

The main set of the simple variables in the Astra code includes

Table 4.2. List of main device and plasma parameters

Quantity	Variable name	Units	Description
R_0	`RTOR`	m	major radius
a_M	`AB`	m	maximum value of a_B allowed
a_B	`ABC`	m	minor radius of LCMS in the mid-plane
Δ_B	`SHIFT`	m	Shafranov shift of the plasma boundary
λ_B	`ELONG`	–	elongation of the plasma boundary
δ	`TRIAN`	–	triangularity of the plasma boundary
Δ_Z	`UPDWN`	m	vertical shift of the plasma boundary
ρ_B	`ROC`	m	effective minor radius
A_i/m_p	`AMJ`	–	main ion mass number (m_p is the proton mass)
Z_i	`ZMJ`	–	electric charge of the main ion species
B_0	`BTOR`	T	vacuum magnetic field at R_0
$I_{\rm pl}$	`IPL`	MA	total plasma current

The first two quantities in this table cannot depend on time, all others can be time dependent.

Table 4.3 shows the list of main radially and time dependent quantities (arrays) which are continuously stored in memory at two time slices. Except for n_i and V', this list coincides with the list of unknown quantities in the system of transport equations.

Table 4.3. List of radial functions (arrays) stored at two time slices

Quantity	Code variable at the current time t	Code variable at the previous time $t-\tau$	Description
n_e	`NE`	`NEO`	electron density
n_i	`NI`	`NIO`	ion density
T_e	`TE`	`TEO`	electron temperature
T_i	`TI`	`TIO`	ion temperature
ψ	`FP`	`FPO`	poloidal flux
f_1	`F1`	`F1O`	1st function in Eq. (75)
f_2	`F2`	`F2O`	2nd function in Eq. (75)
f_3	`F3`	`F3O`	3rd function in Eq. (75)
V'	`VR`	`VRO`	volume derivative

All other quantities are available for the current time, t, only. The list of main arrays is given in Table 4.4.

Table 4.4. Some other radial functions

Quantity	Array name	Units	Description
ρ	`RHO`	m	main magnetic surface label
a	`AMETR`	m	minor radius as in Eq. (88)
Δ	`SHIF`	m	Shafranov shift Eq. (88)
λ	`ELON`	–	elongation Eq. (88)
δ	`TRIA`	–	triangularity Eq. (88)
$j_\parallel$	`CU`	A/m^2	current density
j_{BS}	`CUBS`	A/m^2	bootstrap current density
j_{CD}	`CD`	A/m^2	driven current density
$\sigma_\parallel$	`CC`	$(\mu$Ohm·m$)^{-1}$	conductivity
μ	`MU`	–	rotational transform
μ_V	`MV`	–	vacuum rotational transform
ψ_V	`FV`	V·s	poloidal flux for vacuum magnetic field
$E_\parallel/B_p$	`VP`	m/s	pinch velocity, $E_\parallel = U_\parallel/(2\pi R_0)$
$U_\parallel$	`ULON`	V	longitudinal loop voltage
$U_{\rm pl}$	`UPL`	V	toroidal loop voltage
Z_{eff}	`ZEF`	–	effective charge
Γ_e	`QN`	10^{19}s^{-1}	particle flux
q_e	`QE`	MW	electron heat flux
q_i	`QI`	MW	ion heat flux
J	`IPOL`	–	normalized poloidal current
$V'G_1$	`G11`	m^2	defined in Eq. (28)
R_0G_2/J	`G22`	m	defined in Eq. (28)
G_3	`G33`	–	defined in Eq. (28)

The complete list of radially dependent variables includes more than 200 quantities. Most of them are discussed in Section 4.6.2.

4.2 Transport equations in Astra notations

For convenience in working with the Astra code, we re-write the set of transport equations (59)-(60) in the newly introduced system of units using Astra variable notation. First, we define the operators

$$\begin{gathered}\hat{D}_T[f] \overset{\text{def}}{=} \frac{3}{2}\frac{1}{(V')^{5/3}}\left(\frac{\partial}{\partial t} - \frac{\dot{B}_0}{2B_0}\frac{\partial}{\partial\rho}\rho\right)\left[(V')^{5/3} f\right], \\ \hat{D}_n[f] \overset{\text{def}}{=} \frac{1}{V'}\left(\frac{\partial}{\partial t} - \frac{\dot{B}_0}{2B_0}\frac{\partial}{\partial\rho}\rho\right)\left[V' f\right], \qquad \hat{D}_\psi[f] \overset{\text{def}}{=} \frac{\partial f}{\partial t} - \frac{\rho\dot{B}_0}{2B_0}\frac{\partial f}{\partial\rho}.\end{gathered} \tag{81}$$

Now Eq. (59) takes the form

$$\begin{gathered}\hat{D}_n\left[\mathtt{NE}\right] + \frac{\partial}{\partial V}\left(\mathtt{QN}\right) = \mathtt{SN} + \mathtt{SNN} * \mathtt{NE}, \\ \hat{D}_T\left[\mathtt{NE} * \mathtt{TE}\right] + \frac{\partial}{\partial V}\left(625 * \mathtt{QE} + \frac{5}{2}{*}\gamma_e{*}\mathtt{TE} * \mathtt{QN}\right) = 625 * (\mathtt{PE} + \mathtt{PET} * \mathtt{TE}), \\ \hat{D}_T\left[\mathtt{NI} * \mathtt{TI}\right] + \frac{\partial}{\partial V}\left(625 * \mathtt{QI} + \frac{5}{2}{*}\gamma_i\,\frac{\mathtt{NI} * \mathtt{TI}}{\mathtt{NE}}\,\mathtt{QN}\right) = 625 * (\mathtt{PI} + \mathtt{PIT} * \mathtt{TI}), \\ 2\pi\rho{*}\mathtt{CC}{*}\hat{D}_\psi\left[\mathtt{FP}\right] + V'{*}\left(\mathtt{CUBS} + \mathtt{CD}\right) = 5 * \mathtt{IPOL}^2\frac{\partial}{\partial\rho}\left[\mathtt{G22}\frac{\partial}{\partial\rho}\left(\mathtt{FP}\right)\right].\end{gathered} \tag{82}$$

Two additional factors γ_e, γ_i are introduced in Eq. (82) to allow modification of the contribution of the convective component to the heat flux. The corresponding code variable names are $\mathtt{GN2E} = \frac{5}{2}\gamma_e$ and $\mathtt{GN2I} = \frac{5}{2}\gamma_i$. When not determined explicitly these factors take the default values $\mathtt{GN2E} = \mathtt{GN2I} = 0$. The full energy fluxes through a whole magnetic surface appear in the form

$$Q_e = q_e + \frac{5}{2}\gamma_e T_e \Gamma_e = \mathtt{QE} + 1.6 \times 10^{-3}\,\mathtt{GN2E} * \mathtt{TE} * \mathtt{QN} \tag{83}$$

and

$$Q_i = q_i + \frac{5}{2}\gamma_i T_i \Gamma_i = \mathtt{QI} + 1.6 \times 10^{-3}\,\frac{\mathtt{NI}}{\mathtt{NE}}\,\mathtt{GN2I} * \mathtt{TI} * \mathtt{QN} \tag{84}$$

where the factor $1.6 \times 10^{-3} = 1/625$ emerges because the mixed system of units (keV and MJ) is adopted.

The set of equations (75) acquires the form

$$\frac{1}{V'}\frac{\partial}{\partial t}\left(V'{*}\mathtt{Fj}\right)+\frac{\partial}{\partial V}\left(\mathtt{QFj}\right)=\mathtt{SFj}+\mathtt{SFFj}*\mathtt{Fj}, \tag{85}$$

with

$$\mathtt{QFj}=\mathtt{G11}{*}\left(\mathtt{VFj}*\mathtt{Fj}-\mathtt{DFj}\,\frac{\partial}{\partial\rho}\left(\mathtt{Fj}\right)\right). \tag{86}$$

Here the letter j should be replaced with one of the numbers: 1, 2, 3.

The transport matrix Eq. (60), in the Astra notation, takes the form

$$\begin{pmatrix} \dfrac{\mathtt{QN}-\mathtt{SLAT}*\mathtt{GNX}}{\mathtt{G11}*\mathtt{NE}} \\[2ex] \dfrac{\mathtt{625}*\mathtt{QE}}{\mathtt{G11}*\mathtt{NE}*\mathtt{TE}} \\[2ex] \dfrac{\mathtt{625}*\mathtt{QI}}{\mathtt{G11}*\mathtt{NI}*\mathtt{TI}} \\[2ex] \dfrac{0.4\pi}{B_p}{*}\mathtt{CUBS} \end{pmatrix} = - \begin{pmatrix} \mathtt{DN} & \mathtt{HN} & \mathtt{XN} & \mathtt{CN} \\[2ex] \mathtt{DE} & \mathtt{HE} & \mathtt{XE} & \mathtt{CE} \\[2ex] \mathtt{DI} & \mathtt{HI} & \mathtt{XI} & \mathtt{CI} \\[2ex] \mathtt{DC} & \mathtt{HC} & \mathtt{XC} & 0 \end{pmatrix} \cdot \begin{pmatrix} \dfrac{1}{\mathtt{NE}}\dfrac{\partial\mathtt{NE}}{\partial\rho} \\[2ex] \dfrac{1}{\mathtt{TE}}\dfrac{\partial\mathtt{TE}}{\partial\rho} \\[2ex] \dfrac{1}{\mathtt{TI}}\dfrac{\partial\mathtt{TI}}{\partial\rho} \\[2ex] -1 \end{pmatrix}. \tag{87}$$

The reason for the additional term -SLAT*GNX in the first row on the left hand side of Eq. (87) will be discussed in Section 4.9.1. Note also that the last element of the column vector on the right hand side of Eq. (60) does not correspond to that in Eq. (87): namely, the quantity “ $E_{\|}/B_p$ ” in Eq. (60) is replaced with “1” in Eq. (87). This allows the inclusion of particle or energy fluxes which are not necessarily related to the toroidal electric field.

Table 4.5. Notations for transport coefficients in the Astra code

Code variable	**Units**	Notations in Eqs. (60),(75)
DN, HN, XN	m/s^2	D_n, D_e, D_i
DE, HE, XE	m/s^2	χ_n^e, χ_e, χ_i^e
DI, HI, XI	m/s^2	χ_n^i, χ_e^i, χ_i
CN, CE, CI	m/s	C_n, C_e, C_i
DC, HC, XC	–	D_E, χ_E^e, χ_E^i
DF1, DF2, DF3	m/s^2	D_1, D_2, D_3
VF1, VF2, VF3	m/s	v_1, v_2, v_3
CC	MS/m	$\sigma_{\|}$

The sources on the right hand side in Eqs. (82) and (85) are split into two parts. The part which is linear with respect to the unknown variable in each equation is written separately, in order to allow the choice of either an implicit or an explicit numerical scheme. For instance, the same particle source S_e can be defined as $\mathtt{SN} = s_{ion}Nn_e$ or as $\mathtt{SNN} = s_{ion}N$. In the first case, it would be approximated explicitly using the electron density $n_e = \mathtt{NEO}$ from the previous time slice, $S_e = s_{ion}N * \mathtt{NEO}$; while, in the second case, $S_e = s_{ion}N * \mathtt{NE}$. We summarize usage of the sources and sinks in Table 4.6.

Table 4.6. Notations for the right hand sides of transport equations

Code variable	Units	Definition
SN	$10^{19}/(\mathrm{m}^3\,\mathrm{s})$	$S_e = \mathtt{SN} + \mathtt{SNN} * \mathtt{NE}$
SNN	1/s	
PE, PI,	$\mathrm{MW/m^3}$	$P_e = \mathtt{PE} + \mathtt{PET} * \mathtt{TE}$
PET, PIT,	$\mathrm{MW/(m^3\,keV)}$	$P_i = \mathtt{PI} + \mathtt{PIT} * \mathtt{TI}$
SF1, SF2, SF3	[Fj]/s	$S_j = \mathtt{SFj} + \mathtt{SFFj} * \mathtt{Fj}$
SFF1, SFF2, SFF3	1/s	
CUBS, CD	$\mathrm{MA/m^2}$	$j_{BS} = \mathtt{CUBS}$, $j_{CD} = \mathtt{CD}$

The transport set of equations will be completely defined if

(1) device and plasma parameters (Table 4.2) are set,

(2) all quantities in Tables 4.5 and 4.6 are determined,

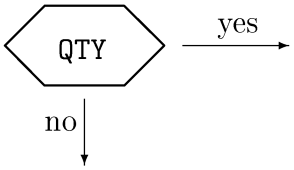
(3) a rule for calculation of metric coefficients is defined and

(4) initial and boundary conditions are prescribed.

Calculation of the metric coefficients is provided by the equilibrium solver, and a link between transport and equilibrium parts of the code is discussed in Section 4.5. The technical details for setting initial and boundary conditions in the code are described in Section 5, while in the following two sections the general logic of the procedure is discussed.

4.3 Initial conditions

In this Section and below we use the following flowchart notation



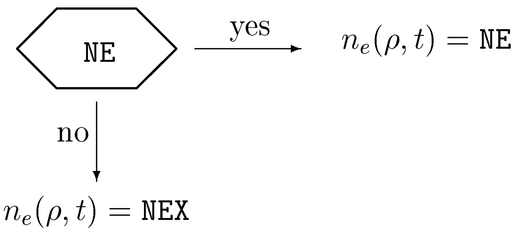
which means a test of whether the quantity QTY is defined in a model or not. If it appears on the left hand side of any assignment instruction in a model (see Section 5.2) at least once as

`QTY = expression;`

then we say that it is defined, the answer to the question is positive and the path labeled ”yes” is taken.

4.3.1 Initial conditions for NE, TE, TI, Fj

Setting of the initial conditions for the electron density is quite straightforward:



Firstly, the Astra compiler checks whether the diffusion equation for n_e is to be solved or not. Suppose first that the density evolution will be prescribed, i.e. the particle diffusion equation is to be suppressed. The Astra compiler continues by checking whether `NE` is assigned in the model. If yes, then the density will evolve according to this assignment and no further checks are made. If `NE` is not defined in the model, then the next check (not shown in the diagram) is made as to whether `NEX` is set in the data file (Section 5.4). If this is the case, then `NE` is set to `NEX`. If it happens that `NEX` does not appear in the data file either, then the default setting $n_e(\rho, t) = 10^{18}$ m^{-3} is applied. The default setting is constant in time and space, while all other means of setting `NE` can have arbitrary dependences.

A similar scheme is applied when the density $n_e(\rho, t)$ is to be defined by the transport equation. The difference is that, in this case, the assignment $n_e(\rho, t_0) = \ldots$ is made only once, and that this is understood as the initial condition. Any time t_0 (code variable `TSTART`) can be selected as a starting time for the simulation. The following evolution of n_e at $t > t_0$ will be defined by the transport equation. Finally, it has to be mentioned that there is an option (see Section 5.2.5) to assign `NE` only once at $t = t_0 =$ `TSTART` even if `NE` or `NEX` are defined as time dependent.

Initial conditions for T_e, T_i and f_j are defined in similar ways:

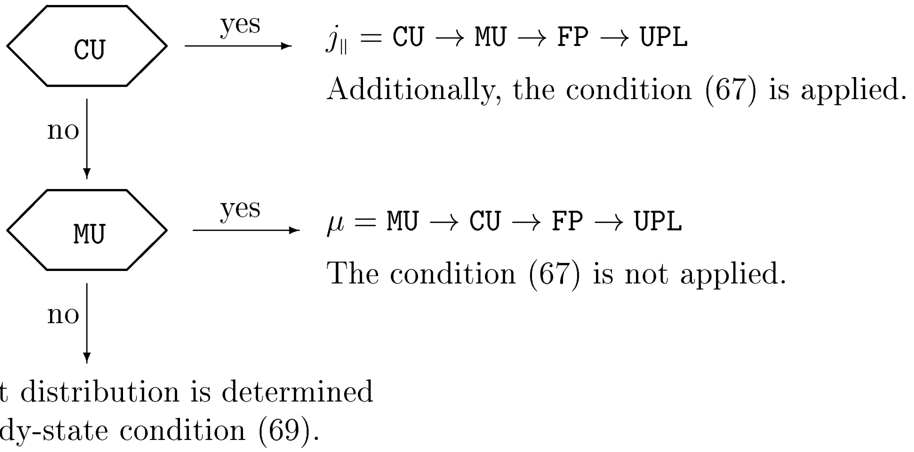
Table 4.7. Summary of the setting of initial conditions

Variable	Name in the code	1st check	2nd check	Default value
n_e	NE	NE $=?$	NEX $=?$	$10^{18}\mathrm{m}^{-3}$
T_e	TE	TE $=?$	TEX $=?$	10 eV
T_i	TI	TI $=?$	TIX $=?$	10 eV
f_1	F1	F1 $=?$	F1X $=?$	0
f_2	F2	F2 $=?$	F2X $=?$	0
f_3	F3	F3 $=?$	F3X $=?$	0

For every quantity in this table, the Astra compiler finds the appropriate mode of calculating the corresponding variable: via equation, formula or data file.

4.3.2 Initial conditions for the poloidal flux $\psi =$ FP

The similar procedure is applied for the poloidal flux. The scheme below can define the initial conditions only (then the equation for ψ is solved), or describe the prescribed time evolution of ψ and all related quantities. Usually, it is more convenient to prescribe the plasma current density $j_{\|}$ rather than the poloidal flux ψ.



$j_{\|} =$ CU $\rightarrow$ MU $\rightarrow$ FP $\rightarrow$ UPL

Additionally, the condition (67) is applied.

$\mu =$ MU $\rightarrow$ CU $\rightarrow$ FP $\rightarrow$ UPL

The condition (67) is not applied.

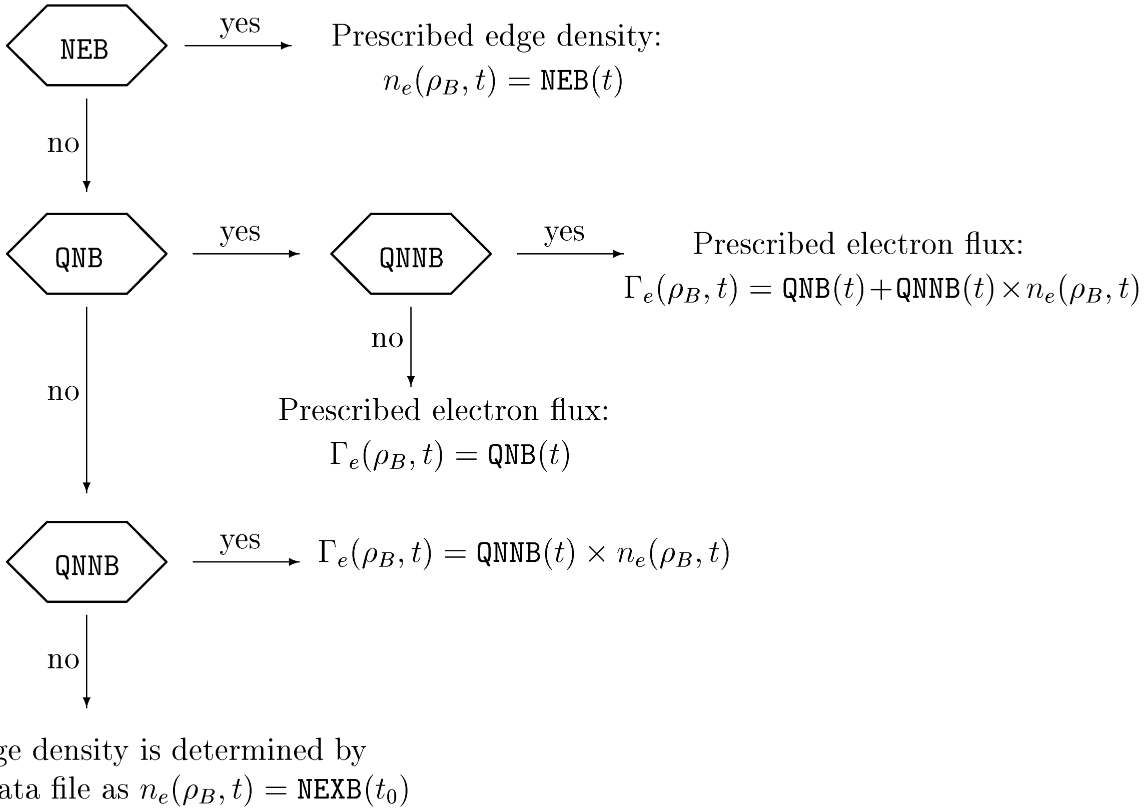
The current distribution is determined by the steady-state condition (69).

After one of the quantities $j_{\|}$ or μ is defined, all other quantities in the list $j_{\|}, \mu, \psi, U_{\rm pl}$ are calculated using Eqs. (33), (34) and Eq. (57). Note that, in case when an assignment of MU is active, no normalization is applied although $\mu(\rho_B)$ is consistent with the total plasma current. That is, in general, $\mu(\rho)$ has a jump (surface current) at the plasma edge. In all other cases, the additional normalization condition (67) is applied so that an assignment of CU is valid within to a multiplicative constant.

4.4 Boundary conditions

4.4.1 Boundary conditions for $n_e, \; T_e, \; T_i, \; f_j$

Besides the initial condition, the boundary values $n_e(\rho_B,t), \; T_e(\rho_B,t), \; T_i(\rho_B,t), \; f_j(\rho_B,t)$ must be prescribed as functions of time. Electron density at the plasma edge $n_e(\rho=\rho_B,t)$ is defined according to


NEB
yes
Prescribed edge density:
$n_e(\rho_B,t) = \mathtt{NEB}(t)$
no
QNB
yes
QNNB
yes
Prescribed electron flux:
$\Gamma_e(\rho_B,t) = \mathtt{QNB}(t)+\mathtt{QNNB}(t)\times n_e(\rho_B,t)$
no
Prescribed electron flux:
$\Gamma_e(\rho_B,t) = \mathtt{QNB}(t)$
no
QNNB
yes
$\Gamma_e(\rho_B,t) = \mathtt{QNNB}(t) \times n_e(\rho_B,t)$
no
Edge density is determined by
a data file as $n_e(\rho_B,t) = \mathtt{NEXB}(t_0)$

It is seen that giving $\mathtt{NEB} = \mathtt{1}$ one obtains the boundary condition $n_e(\rho_B,t) = 10^{19}\mathrm{m}^{-3}$. In such a case, the setting of `QNB` or `QNNB` does not influence $n_e(\rho_B,t)$. Otherwise, if the particle flux, rather than the particle density, at the plasma edge is to be prescribed, then any instructions of the type $\mathtt{NEB} = \cdots$ must not be present.

Any combination of explicit or implicit approximation for the edge particle flux can be used. For instance, both instructions $\mathtt{QNB} = \mathtt{10} * \mathtt{NE}$ and $\mathtt{QNNB} = \mathtt{10}$ describe the same boundary condition (to within the numerical accuracy). However, in the first case, the numerical approximation reads $\Gamma_e(\rho_B,t) = 10n_e(\rho_B,t-\Delta t)$, while in the second, $\Gamma_e(\rho_B,t) = 10n_e(\rho_B,t)$.

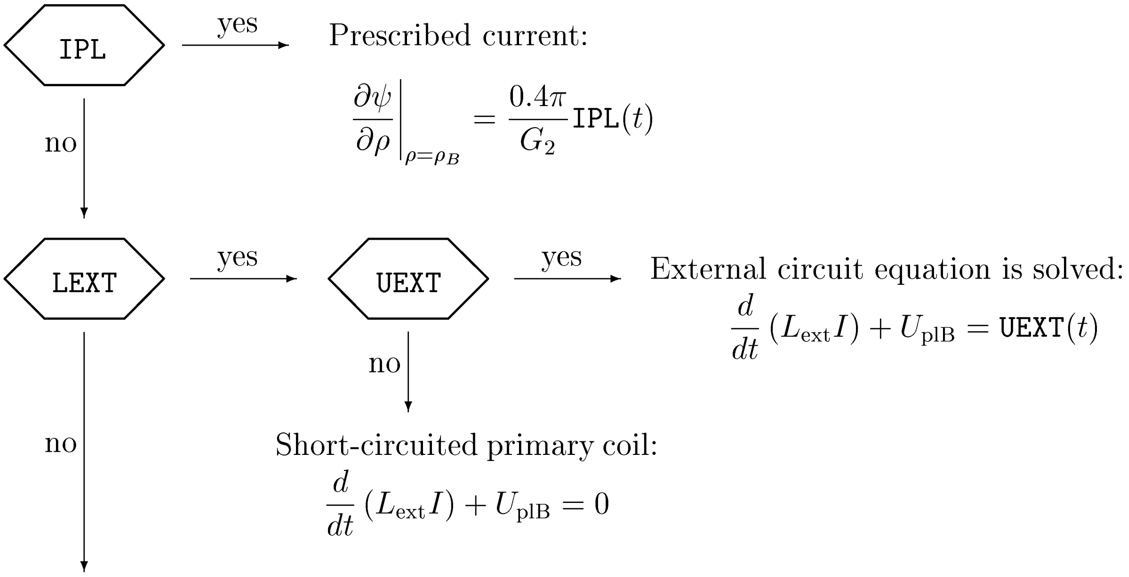
Other boundary conditions can be defined in a parallel fashion, as illustrated in Table 4.8. In addition to this table, we note that if none of the control parameters listed there appears in a model, then a boundary value from the data file is used. However, there is a difference between the following two cases: (i) where the assignment NEB = NEXB is explicitly given in a model, and (ii) where the same assignment is used because none of the control parameters appears in a model. In the first case, the assignment is considered as time dependent while in the second case it is executed only at the initial time. Finally, if a data file does not define the quantity either, then the boundary value is set to one of the defaults given in Table 4.7.

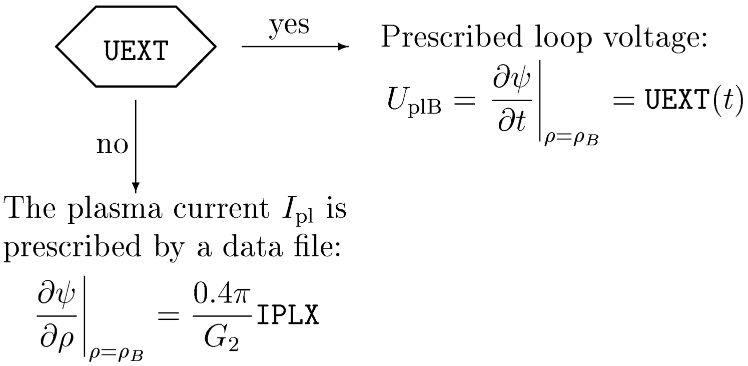
Table 4.8. Summary of boundary condition setting

Quantity/Flux	Code name	Control parameters / Units		
n_e / Γ_e	NE / QN	NEB / $10^{19}\mathrm{m}^{-3}$	QNB / $10^{19}\mathrm{s}^{-1}$	QNNB / m^3/s
T_e / q_e	TE / QE	TEB / keV	QEB / MJ	QETB / MJ/keV
T_i / q_e	TI / QI	TIB / keV	QIB / MJ	QITB / MJ/keV
f_1 / Γ_1	F1 / QF1	F1B / $[f_1]$	QF1B / $[f_1]$ m^3/s	QFF1B / m^3/s
f_2 / Γ_2	F2 / QF2	F2B / $[f_2]$	QF2B / $[f_2]$ m^3/s	QFF2B / m^3/s
f_3 / Γ_3	F3 / QF3	F3B / $[f_3]$	QF3B / $[f_3]$ m^3/s	QFF3B / m^3/s

4.4.2 Boundary condition for the poloidal flux $\psi =$ FP

The boundary condition for the poloidal flux is defined according to the following diagram


IPL
yes
Prescribed current:
$\left.\frac{\partial\psi}{\partial\rho}\right|_{\rho=\rho_B} = \frac{0.4\pi}{G_2}\mathrm{IPL}(t)$
no
LEXT
yes
UEXT
yes
External circuit equation is solved:
$\frac{d}{dt}\left(L_{\mathrm{ext}}I\right) + U_{\mathrm{plB}} = \mathrm{UEXT}(t)$
no
no
Short-circuited primary coil:
$\frac{d}{dt}\left(L_{\mathrm{ext}}I\right) + U_{\mathrm{plB}} = 0$



Prescribed loop voltage:

$$U_{\rm plB} = \left.\frac{\partial\psi}{\partial t}\right|_{\rho=\rho_B} = \mathtt{UEXT}(t)$$

The plasma current $I_{\rm pl}$ is prescribed by a data file:

$$\left.\frac{\partial\psi}{\partial\rho}\right|_{\rho=\rho_B} = \frac{0.4\pi}{G_2}\mathtt{IPLX}$$

In the simplified external circuit equation in this diagram $L_{ext} = \mathtt{LEXT}$ can be understood as the external inductance between plasma and primary coil. Note also that in the case of prescribed loop voltage the parameter UEXT defines a loop voltage at the plasma surface and coincides with $U_{\rm plB} = \mathtt{UPLB}$, whereas in the circuit equation UEXT denotes a voltage provided by the external source (primary coil). In the code, the circuit equation is represented as

$$\frac{L_{\rm ext}G_2}{0.4\pi}\left.\frac{\partial\psi}{\partial\rho}\right|_{\rho_B} + \psi|_{\rho_B} = \int\limits_0^t U_{\rm ext}dt$$

with $L_{\rm ext}$ measured in μHn.

4.5 Link between the transport and equilibrium parts of Astra

Two versions of the equilibrium solver are presently provided by the Astra system. The more simple version [5] solves the equilibrium equation Eq. (40) using 3-moment approach. It is assumed that the plasma configuration is up-down symmetric and each magnetic surface can be parameterized as

$$\left\{ \begin{array}{l} r(a,\theta) = R_0 + \Delta(a) + a\left(\cos\theta - \delta(a)\sin^2\theta\right) \\ z(a,\theta) = \Delta_Z + a\lambda(a)\sin\theta \end{array} \right. \tag{88}$$

where a is a minor radius of every magnetic surface in the equatorial mid-plane. From now on a will denote this magnetic surface function. Δ is the Shafranov shift, λ is the elongation and δ is the triangularity of each flux surface. The parameter Δ_Z is not included in the equilibrium solver because the problem has translational symmetry in the z –direction. However, the plasma up– or down-shift may be relevant for the calculation of heat and particle sources. In this representation the boundary surface S_B is specified with five parameters: R_0 , a_B , $\Delta(a_B)$, $\lambda(a_B)$, $\delta(a_B)$. This is equivalent to a prescription of four characteristic points on the boundary. These are two points in the mid–plane $\{r = R_0 + \Delta(a_B) \pm a_B,\ z = \Delta_Z\}$ and the two points most remote from the plane $\{r = R_0 + \Delta(a_B) - a_B\delta(a_B),\ z = \Delta_Z \pm a_B\lambda(a_B)\}$.

A more advanced equilibrium solver called ESC[3] (Equilibrium and Stability Code) is also available [6] which allows for arbitrary shape of the plasma boundary. In this version, the plasma boundary S_B is specified by a set of $N_B \le 121$ points $\{r_i, z_i\}$ while the configuration is sought in the form

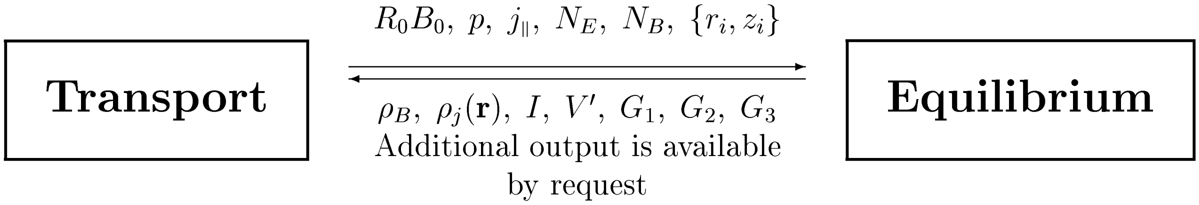
$$\begin{cases} r(\rho,\tau) = r_0^c(\rho) + 2\sum\limits_{k=1}^{K}\left[r_k^c(\rho)\cos k\tau + r_k^s(\rho)\sin k\tau\right], \\ z(\rho,\tau) = \Delta_Z(\rho) + \rho\lambda(\rho)\sin\tau. \end{cases} \tag{89}$$

ESC also provides an analysis of plasma stability with respect to tearing and ballooning modes.

Now we will describe the interface between the equilibrium and the transport packages of the code. If the number N_B does not exceed 4 and the boundary is up down symmetric then it is convenient to use shift, elongation and triangularity of the boundary. Two radially dependent functions on the right hand side of the Grad-Shafranov equation, namely the pressure and current density distributions, are supplied at a ρ_j -grid of N_E points (variable name `NEQUIL`) which is different from the transport grid. In the current version, $N_E =$ `NEQUIL` ≤ 41 and in addition to its main meaning it is also used as a control parameter:

$N_E < 0$	No equilibrium solver is called. Pre-calculated metric is taken from a data file as described in Section 5.3.2.
$N_E = 0$	No equilibrium solver is called. Parabolic distributions for $\Delta(\rho), \lambda(\rho), \delta(\rho)$ are used and $a(\rho) = \rho/\sqrt{\lambda}$.
$0 < N_E \le 41$	3-moment equilibrium solver is called with N_E-point grid.
$N_E > 41$	ESC is called.

Parameter exchange is shown in the following diagram and explained in the table below.


$R_0B_0,\ p,\ j_\parallel,\ N_E,\ N_B,\ \{r_i, z_i\}$
Transport
Equilibrium
$\rho_B,\ \rho_j(\mathbf{r}),\ I,\ V',\ G_1,\ G_2,\ G_3$
Additional output is available by request

[3] http://w3.pppl.gov/topdac/esc.htm

Input to the equilibrium module		**Output** from the equilibrium module	
R_0B_0	boundary value of I	ρ_B	boundary value of ρ
N_B	boundary-grid size ($N_B \leq 12$)	$\rho_j(\mathbf{r})$	shape of every flux surface
$\{r_i, z_i\}$	boundary shape ($1 \leq i \leq N_B$)	$I(\rho_j)$	poloidal current
N_E	ρ-grid size and type of solver	$V'(\rho_j)$	volume derivative
ρ_j	ρ-grid ($1 \leq j \leq N_E$)	$G_1(\rho_j)$	metric characteristics
$p(\rho_j)$	pressure profile	$G_2(\rho_j)$	
$j_{\parallel}(\rho_j)$	current density profile	$G_3(\rho_j)$	

4.6 Astra variables

In this section, we describe Astra internal variables dividing them into groups according to their meaning and limiting the discussion to those variables which can be accessed (or corrupted) by a user of the code. These variables are held in Fortran common blocks and seen from every part of the code. First of all, there are variables describing different physical quantities: some of them have to be defined by a user, others are calculated by the code. The second group consists of control parameters for the numerical schemes used in the code and for tuning the output. Finally, there is a set of global variables without any predefined meaning, which are entirely at the disposal of the user. Employing these variables is recommended if any new quantity is needed in a simulation. Although it is not forbidden to introduce new variables in a user's model, no check is made for dangerous possible name conflicts. Therefore, it is supposed that in introducing a new variable (see Section 5.2.8), the user will check that the variable name does not coincide with any of the Astra global variables. Complete lists of the Astra global variables can be found in the files: [4]

```
AWD/for/parameter.inc,    AWD/for/const.inc,     AWD/for/status.inc,
AWD/for/outcmn.inc,       AWD/tmp/declar.fml,    AWD/tmp/declar.fnc.
```

It is obvious that there is a minimal set of variables which must be defined by a user in order to start an Astra simulation. In the Astra code we have tried to minimize this burden

[4] We remind that `AWD` stands here for the Astra working directory which can be different for each installation.

on the user by setting as many variables as possible according to common sense. We will use the following symbols to mark different types of the code variables.

!	The variable must be defined by a user.
+	The variable can be defined by a user. Otherwise, it takes a default value.
∗	The variable cannot be defined by a user. It is always defined by the code.

As mentioned, there is an additional type of variables which are exclusively defined by the user and can be used for exchanging information between user routines.

There are several ways for determining variables in the Astra code and the default definition is only applied if the variable was not explicitly set by the user. More about possible ways of setting code variables and about their default values can be found in Section 5.3.

4.6.1 Scalar (time dependent) variables

We start with a set of variables which define the plasma configuration. The five geometrical variables `AB, ELONM, TRICH` and `AWALL, RTOR` are time independent. The first three of them are the maximum possible values for the time dependent quantities `ABC, ELONG` and `TRIANG`, respectively. The variable `AWALL` is presently used only for a radial grid allocation

Table 4.9. Configuration parameters

Type	Variable	Units	Description
!	`AB`	m	Limiter position in the mid-plane, a_B
+	`ABC`	m	Transport grid boundary in the variable a
+	`AWALL`	m	Wall position in the mid-plane
!	`RTOR`	m	Major radius of the vacuum chamber
+	`SHIFT`	m	Horizontal shift of the edge magnetic surface
+	`UPDWN`	m	Vertical shift of the edge magnetic surface
+	`ELONG`	–	Elongation of the edge magnetic surface
+	`ELONM`	–	Vacuum chamber elongation (used for drawing)
+	`TRIAN`	–	Triangularity of the boundary surface
+	`TRICH`	–	Vacuum chamber triangularity

Table 4.10. Main plasma parameters

Type	Variable	Units	Description
!	`BTOR`	T	Vacuum toroidal field at the position `RTOR`
!	`IPL`	MA	Plasma current
+	`AMJ`	m_p	Main ion atom mass in units of the proton mass
+	`ZMJ`	e_p	Main ion charge in units of the proton charge

and usually it does not require an explicit definition. The major radius RTOR corresponds to the position where the vacuum toroidal magnetic BTOR is given. Although RTOR cannot vary in time, plasma motion can be defined by varying SHIFT and UPDWN. The main ion mass and charge can also be given as radial arrays AMAIN and ZMAIN.

Table 4.11. Other scalar plasma parameters

Type	Variable	Units	Description
+	GN2E	–	Factor in convective electron heat flux
+	GN2I	–	Factor in convective ion heat flux
+	AIM1,AIM2,AIM3	m_p	Impurity species mass in units of the proton mass
+	LEXT	μH	External inductance
+	UEXT	V	External loop voltage
+	WNE, WTE, WTI	m	Width of exponential decay for n_e, T_e, T_i at $a > a_B$
$*$	RHOW	m	Limiter position in the variable ρ: $a(\mathtt{RHOW}) = \mathtt{AB}$
$*$	ROC	m	ρ_B, grid edge in the variable ρ: $a(\mathtt{ROC}) = \mathtt{ABC}$
$*$	VOLUME	m^3	Plasma volume inside ρ_B =ROC
$*$	VSB	Vs	Poloidal flux at the plasma edge
$*$	VSC	Vs	Poloidal flux at the magnetic axis

As seen from Table 4.11, up to three different ion impurity species can be defined in the code by their masses, charges and densities [1]. The impurities are not directly included in the main set of transport equations, but they are required in the NBI heating subroutine, NCLASS, STRAHL and some other external modules. The parameters WNE, WTE, WTI are used to describe the plasma parameters NE, TE, TI in the scrape-off-layer outside the main transport grid. In the present version, they do not play any role in the simulation and can be seen only in a special drawing mode when radial profiles are plotted beyond the transport boundary.

As long as the memory requirement for the Astra code is quite moderate the code does not use dynamic memory allocation. All arrays have fixed dimensions and any change of those requires re-installation of the code. Some of the code constants are given in the Table 4.12. Normally, an error is reported when any of these array limits is exceeded.

The parameter NRD allocates the maximum possible length of all radial arrays. However, the parameter which really determines the radial grid size is NB1 $\leq$ NRD. The transport core of the code is quite fast so that large values of NB1 do not affect its speed significantly.

[1] Unlike the masses, the impurity charges and densities are described by radial arrays ZIM1, ZIM2, ZIM3 and NIZ1,NIZ2,NIZ3 (Section 4.6.2).

Table 4.12. Internal constants

Type	Constant	Value	Description
$*$	NRD	501	Maximum size of the radial grid
$*$	NRDX	200	Maximum No. of radial points for input from a data file
$*$	NRW	96	Maximum No. of radial / time curves for output
$*$	NTIMES	1024	Dimension of arrays for output in time modes 6 and 7
$*$	NTVAR	2000	Total No. of time slices for all variables in a data file
$*$	NTARR	5000	Total No. of time slices for all arrays in a data file
$*$	GP	π	
$*$	GP2	2π	

However, some additional modules which do not use a separate grid can noticeably decelerate the calculations. The essential feature of the parameter NB1 is that it must be defined at the earliest phase of the code execution and cannot be redefined later. It means that, unlike most other parameters in the Astra code which can be set in many different ways, the only way of defining the grid size NB1 is input from a data file (Section 5.3.1). The parameter NEQUIL defines a radial grid for the 3-moment equilibrium solver and serves also as a switch to the ESC solver. In the latter case, a radial grid is selected automatically. Two parameters XFLAG and NBND define a type of the plasma boundary and a number of points on it, respectively.

Table 4.13. Parameters for radial grid control

Type	Variable	Units	Description
$+$	NB1	–	No. of points of the main transport grid NB1 $\le$ NRD
$+$	NEQUIL	–	3-moment equilibrium solver grid size and control switch
$+$	NB2EQL	–	Factor for NB pressure contribution to equilibrium eqn
$+$	NBND	–	No. of points on the boundary
$+$	XFLAG	–	X-point flag
$+$	NUF	–	No. of radial points to be written in a U-file
$+$	XOUT	–	Type of abscissa for radial graphic output (modes 1, 2, 4, 5)
$+$	XINPUT	–	Type of abscissa for radial profile input
$*$	HRO	m	Radial grid step in the variable ρ
$*$	HROA	m	Edge step of the radial grid: RHO(NA1)-RHO(NA)
$*$	NA	–	= NA1-1
$*$	NA1	–	Edge grid point number: ROC=RHO(NA1)
$*$	NAB	–	NAB $\ge$ NA1, AB = AMETR(NAB) $\ge$ ABC = AMETR(NA1)

The parameter NUF defines a number of radial grid points for an output in the format of U-file. A type of the created U-file depends on the current radial graphic mode (Section 5.5.3)

– 1D (radial coordinate) in graphic modes 1, 2, 3,

– 2D (radial and time coordinates) in the graphic mode 4,

– 1D (time coordinate) in the graphic mode 6.

In addition, one should also have in mind that the radial coordinate in the U-file produced, is the same as for the current graphs on the screen. Therefore, it is controlled by parameter `XOUT` which defines an abscissa type for radial profiles in the current plot. If the parameter `XOUT` takes values 0 or 1 (default value is 1) then the flux surface label a is used as a radial coordinate. If $\texttt{XOUT} = 2$ or 3 then ρ or ψ are used in place of a. The parameter `XINPUT` can take the same values, but it defines an abscissa of radial profiles for input in cases when the radial grid is equidistant but the coordinate is not defined explicitly. The meaning of these numbers is similar to that for the parameter `GRIDTYPE` which is explained in more detail in Section 5.4.4, and summarized in Table 5.5. All other parameters in Table 4.13 cannot be changed by a user. They may be used for output only.

Table 4.14. Parameters for output control

Type	Variable	Units	Description
+	DPOUT	s	Time interval for profile storing (modes 4, 5 and Review)
+	DROUT	s	Time interval for profile redrawing (modes 1, 2 and 3)
+	DTOUT	s	Time step for time curve output (modes 6 and 7)
+	TPAUSE	s	Time for switching on the “step” mode
+	TEND	s	End of run time
+	TINIT	s	Time axis left label (modes 6 and 7)
+	TSCALE	s	Time axis scale length (modes 6 and 7)

Table 4.14 contains a number of other control parameters for graphic output. All these parameters are described in Section 5.5.3. They can be set in a model or adjusted interactively. If the parameters `DPOUT`, `DROUT`, `DTOUT`, `TSCALE` and also the parameters `TAUMIN, TAUMAX` from the Table 4.15 are not given by the user they are set by the code to be proportional to the plasma volume.

The control parameters given in Table 4.15 define the numerical algorithm used in the transport part of the code. Usually, the time step is selected automatically by the following condition. If a maximum relative change in any of variables `TE, TI, NE, F1, F2, F3` is larger than `DELVAR` then the time step $\tau = \texttt{TAU}$ is decreased. Otherwise, it is multiplied by `TAUINC` which when not defined explicitly is equal to 1.1. However, a user can fix the time step by setting TAUMIN = TAUMAX, or modify the algorithm (see Section 4.9.8). An increase of the parameter ITEREX (default value 1) can sometimes improve the numerical stability in strongly nonlinear problems. Another iteration parameter ITERIN cannot be changed by a user. It is selected by the code to ensure that the numerical approximation of electron–ion heat exchange does not violate energy conservation.

Table 4.15. Parameters for accuracy control

Type	Variable	Units	Description
+	TAUMAX	s	Maximum time step: $\tau \le$ TAUMAX
+	TAUMIN	s	Minimum time step: $\tau \ge$ TAUMIN
+	TAUINC	–	Maximum time step increment: TAU$(t+\tau)$/TAU(t)
+	DELVAR	–	Maximum relative change of variables at time step
+	ITEREX	–	No. of iterations in the external loop
$*$	ITERIN	–	No. of iterations in the internal loop
$*$	TIME	s	Current time of the simulation
$*$	TAU	s	Current time step

Exchange variables. A group of parameters, which are in all respects similar to those discussed above, is reserved for particular uses in the different external modules rather than in the Astra core. If, in a particular model, these modules are not involved then the parameters can be used quite arbitrarily independently of their names and significance. If the modules are in use, however, then the parameters can only be used in the predefined context.

Table 4.16. Input/output parameters for the subroutine NEUT

Name	Units	Description
NNCL / NNWM	10^{19}/m^{-3}	Density of incoming cold / warm neutrals
ENCL / ENWM	keV	Energy of incoming cold / warm neutrals
NNCX	–	No. of iterations in the neutral solver
ALBPL	–	Plasma albedo (Neutral flux outward)/(Neutral flux inward)

The scalar input parameters for the external module NEUT are listed in Table 4.16. Two pairs {NNCL, ENCL} and {NNWM, ENWM} are available. Each pair describes a density and energy of one mono-energetic incident neutral component. Although it is not quite correct, the parameter NNCX can be viewed as a number of charge exchange neutral generations. The complete list of parameters related to this subroutine which includes also output is discussed in Section 4.11.

A number of control parameters are used in different heating and current drive modules.

Most of them are built into the appropriate packages, and can be changed by the user according to the usual rules. Only a few of them are associated with the total heating power (QECR, QFW, QICR, QLH, QNBI) and these belong to the list of main Astra parameters. As already mentioned, the meaning of these parameters is defined inside the corresponding packages and can even change if one heating module is replaced by another.

All scalar variables discussed so far can be defined with maximum flexibility (except for those marked with ‘∗’). They can be defined by all the methods described in Section 5.3, i.e. in a model, in a data file (with few exceptions) and interactively. Other scalars described below have fewer restrictions in their usage, but have less flexibility in how they are initialized.

Dummy variables. As mentioned in the beginning of this section there are a number of parameters which are reserved for user's needs and can be used arbitrarily in a model. All these scalar variables have no predefined meanings and mnemonics of names (if any) should not be viewed as binding. For instance, CMHD1 need not be associated with MHD phenomena as well as CIMP1 may have nothing to do with the impurity behavior. They can be used for communication between different user's subroutines, in intermediate calculations or for output. Two different groups are described below. Variables in the both groups can be time dependent but the time dependence is to be set in different ways.

Table 4.17. Parameters for a run control (C-parameters)

CF1	CF2	CF3	CF4	CF5	CF6	CF7	CF8
CF9	CF10	CF11	CF12	CF13	CF14	CF15	CF16
CV1	CV2	CV3	CV4	CV5	CV6	CV7	CV8
CV9	CV10	CV11	CV12	CV13	CV14	CV15	CV16
CHE1	CHE2	CHE3	CHE4	CHI1	CHI2	CHI3	CHI4
CNB1	CNB2	CNB3	CNB4	CNBI1	CNBI2	CNBI3	CNBI4
CCD1	CCD2	CCD3	CCD4	CRF1	CRF2	CRF3	CRF4
CNEUT1	CNEUT2	CNEUT3	CNEUT4	CPEL1	CPEL2	CPEL3	CPEL4
CBND1	CBND2	CBND3	CBND4	CFUS1	CFUS2	CFUS3	CFUS4
CIMP1	CIMP2	CIMP3	CIMP4	CMHD1	CMHD2	CMHD3	CMHD4
CRAD1	CRAD2	CRAD3	CRAD4	CSOL1	CSOL2	CSOL3	CSOL4

The group of parameters listed in Table 4.17 will be called C-parameters. They can be used for run control because the C-parameters can be changed interactively during a code run. They can also be saved during a run (Section 5.3.2), then the subsequent run will be started with the recently adjusted values of these control parameters. If the C-parameters are not determined explicitly in one of the ways described in Section 5.3 then they take default values which are 0 for the 16 variables starting with `CV` (from `CV1` till `CV16`) and 1 for all others. A limitation is that the C-parameters cannot be read from a data file.

A second set of dummy parameters given in Table 4.18 can be defined by an input file. These parameters can be used to input non-standard experimental characteristics for which (unlike `BTOR` or `IPL`) no special variable is allocated. Typical examples are measured $q = 2$ radius or a diamagnetic signal. Default value for all these parameters is 0 and they cannot be changed interactively. This set of parameters is coupled with the next set by the rules discussed in Section 5.3.3.

Table 4.18. Variables readable from a data file

ZRD1X	ZRD2X	ZRD3X	ZRD4X	ZRD5X	ZRD6X	ZRD7X	ZRD8X
ZRD9X	ZRD10X	ZRD11X	ZRD12X	ZRD13X	ZRD14X	ZRD15X	ZRD16X
ZRD17X	ZRD18X	ZRD19X	ZRD20X	ZRD21X	ZRD22X	ZRD23X	ZRD24X
ZRD25X	ZRD26X	ZRD27X	ZRD28X	ZRD29X	ZRD30X	ZRD31X	ZRD32X
ZRD33X	ZRD34X	ZRD35X	ZRD36X	ZRD37X	ZRD38X	ZRD39X	ZRD40X
ZRD41X	ZRD42X	ZRD43X	ZRD44X	ZRD45X	ZRD46X	ZRD47X	ZRD48X

Table 4.19. Z-parameters

ZRD1	ZRD2	ZRD3	ZRD4	ZRD5	ZRD6	ZRD7	ZRD8
ZRD9	ZRD10	ZRD11	ZRD12	ZRD13	ZRD14	ZRD15	ZRD16
ZRD17	ZRD18	ZRD19	ZRD20	ZRD21	ZRD22	ZRD23	ZRD24
ZRD25	ZRD26	ZRD27	ZRD28	ZRD29	ZRD30	ZRD31	ZRD32
ZRD33	ZRD34	ZRD35	ZRD36	ZRD37	ZRD38	ZRD39	ZRD40
ZRD41	ZRD42	ZRD43	ZRD44	ZRD45	ZRD46	ZRD47	ZRD48

4.6.2 Vector (radially and time dependent) variables

A few hundreds of internal Astra variables describe radially and time dependent quantities. Here we discuss those which are implemented as Fortran arrays, i.e. those which are calculated once at each time step at all points on a radial grid. Other radially dependent quantities will be considered in the next section.

A complete alphabetical list of all Astra arrays is given in the file `AWD/for/status.inc`. In this section, for the sake of convenience, the arrays are combined in groups according to their physical meaning. A group of vector variables related to the plasma geometry is given in Table 4.20. The array RHO $= \rho_j = (j - 0.5)$ HRO, $(j = 1, 2, 3, \dots,$ NA$)$ is the main radial grid used in the transport problem. The main set of unknown functions in Eqs. (59), (75) n_e, T_e, T_i, ψ, f_j and most of other arrays are defined on this grid. Fluxes and other quantities which are derivatives of quantities defined on the main grid, for instance, the rotational transform $\mu =$ MU, are defined on the intermediate grid $\rho_j^{aux} = j$ HRO.

Table 4.20. Arrays describing plasma configuration

Type	Variable	Units	Description	
$*$	AMETR	m	a	Magnetic surface radius in a mid–plane
$*$	RHO	m	ρ	Equivalent radius of a magnetic surface
$+$	SHIF / SHX	m	Δ	Shafranov shift of a magnetic surface
$+$	SHIV	m	Δ_Z	Vertical shift of a magnetic surface
$+$	ELON / ELX		λ	Elongation of a magnetic surface
$+$	TRIA / TRX		δ	Triangularity of a magnetic surface
$*$	SQEPS		$\sqrt{\varepsilon}$	$\varepsilon = a/R$ being the inverse aspect ratio
$*$	SLAT	m^2	S_a	Toroidal surface area
$*$	VOLUM	m^3	V	Volume within a magnetic flux surface
$*$ / $+$	VR / VRX	m^2	V'	on the main grid at t
$*$	VRO	m^2	V'	on the main grid at $t - \tau$
$*$	VRS	m^2	V'	on the intermediate grid at t

Some variable names for arrays have the character 'X' at the end. These X-arrays can be imported from an input file as described in Section 5.3. For instance, these external data can be experimentally measured radial profiles, and can be read in for on-line comparison with the results of simulation during the run. They could also be radial functions, pre-calculated by stand-alone codes for subsequent usage in the Astra code, e.g., heat deposition profiles or equilibrium evolution characteristics calculated elsewhere. Although all X-arrays have names associated with predefined names for some physical quantities, most of them are independent from their prototypes. However, there are a few situations when some unexpected side effects are possible. It usually can happen when any explicit definition is missing and the code uses default assignment as described in Section 5.3.1. For instance, if the transport equation (85) for F1 is involved but no initial condition for F1 is specified, then the Astra compiler will use the default assignment F1=F1X. Then F1X can be used in this sense only. Otherwise, the variable F1X can be used for reading arbitrary radially and time dependent function from a data file.

In the next two tables 4.21 and 4.22, the main plasma parameters, corresponding fluxes and sources are collected. Note that the quantities SNTOT, PETOT, PITOT, QN, QE, QI are calculated by the code. Any user's attempt to define them will be overridden.

Table 4.21. Arrays for the main plasma parameters

Type	Variable	Units	Description
$+ / * / +$	NE / NEO / NEX	$10^{19}\mathrm{m}^{-3}$	$n_e(t)$ / $n_e(t-\tau)$ / $n_e^{exp}(t)$
$+ / * / +$	NI / NIO / NIX	$10^{19}\mathrm{m}^{-3}$	$n_i(t)$ / $n_i(t-\tau)$ / $n_i^{exp}(t)$
$+ / * / +$	TE / TEO / TEX	keV	$T_e(t)$ / $T_e(t-\tau)$ / $T_e^{exp}(t)$
$+ / * / +$	TI / TIO / TIX	keV	$T_i(t)$ / $T_i(t-\tau)$ / $T_i^{exp}(t)$
$+$	AMAIN / ZMAIN	m_p / e_p	main ion mass / main ion charge
$+$	ZEF / ZEFX		Z_{eff} / Z_{eff}^{exp}

Table 4.22. Fluxes and sources (see also Table 4.6)

Type	Variable	Units	Description
$* / +$	SNTOT / SNX	$10^{19}\mathrm{m}^{-3}\mathrm{s}^{-1}$	Particle source SNTOT = SN + SNN $*$ NE
$* / +$	PETOT / PEX	$\mathrm{MW\,m}^{-3}$	Electron heat source PETOT = PE + PET $*$ TE
$* / +$	PITOT / PIX	$\mathrm{MW\,m}^{-3}$	Ion heat source PITOT = PI + PIT $*$ TI
$* / +$	GN / GNX	$10^{19}\mathrm{m}^{-2}\mathrm{s}^{-1}$	GN = QN/G11, GNX is usually defined in GNXSRC
$* / +$	QN / QNX	$10^{19}\mathrm{s}^{-1}$	Electron flux through the entire magnetic surface
$* / *$	QE / QI	MW	Electron heat flux / Ion heat flux

Table 4.23 describes quantities related to the diffusion of the poloidal field. The quantities FV, MV, CV are introduced for simulation of a stellarator. For tokamak modelling they are not needed. If MV (or CV) is set then the external (vacuum) poloidal field is added

Table 4.23. Poloidal flux and related quantities

Type	Variable	Units	Description
$+ / *$	FP / FPO	Vs	Poloidal flux: $\psi(t)$ / $\psi(t-\tau)$
$+$	MU / MUX		Inverse safety factor ($\mu = \iota = 1/q$)
$+$	CU / CUX	$\mathrm{MA\,m}^{-2}$	$j_{\|}$, longitudinal current density, Eq. (32)
$+$	CD	$\mathrm{MA\,m}^{-2}$	j_{CD}, driven current, Eq. (58)
$+$	CUBS	$\mathrm{MA\,m}^{-2}$	j_{BS}, bootstrap current, Eq. (58)
$+$	CUTOR	$\mathrm{MA\,m}^{-2}$	j_{tor}, toroidal current density, Eq. (31)
$+$	ULON	V	$U_{\|}$, parallel loop voltage, Eq. (44)
$+$	UPL	V	U_{pl}, toroidal loop voltage, Eq. (46)
$+$	FV	Vs	Vacuum poloidal flux created by external coils
$+$	MV / MVX		Vacuum rotational transform for a stellarator (iota)
$+$	CV	$\mathrm{MA\,m}^{-2}$	Current density equivalent to MV

to the field of the plasma current. In this case, the latter can also be zero. All current densities in Tables 4.23 and 4.24 one are assumed to be parallel to the magnetic field **B** because they are supposed to be used in the parallel Ohm law Eq. (57). The only exception is `CUTOR` which participates in the Joule heating term Eq. (55).

Table 4.24. Right hand sides of transport equations

Type	Variable	Units	Description
+	`CUBM`	$\mathrm{MA\,m^{-2}}$	NB driven current
+	`CUECR`	$\mathrm{MA\,m^{-2}}$	EC driven current
+	`CUFI`	$\mathrm{MA\,m^{-2}}$	Fast ion current
+	`CUFW`	$\mathrm{MA\,m^{-2}}$	FW driven current
+	`CUICR`	$\mathrm{MA\,m^{-2}}$	IC driven current
+	`CULH`	$\mathrm{MA\,m^{-2}}$	LH driven current
+	`PRAD / PRADX`	$\mathrm{MW\,m^{-3}}$	Radiation power
+	`PEECR`	$\mathrm{MW\,m^{-3}}$	Electron heating due to ECR
+	`PEFW`	$\mathrm{MW\,m^{-3}}$	Electron heating due to FW
+	`PEICR`	$\mathrm{MW\,m^{-3}}$	Electron heating due to ICR
+	`PELH`	$\mathrm{MW\,m^{-3}}$	Electron heating due to LHCD
+	`PIFW`	$\mathrm{MW\,m^{-3}}$	Ion heating due to FW
+	`PIICR`	$\mathrm{MW\,m^{-3}}$	Ion heating due to ICR
+	`PBEAM`	$\mathrm{MW\,m^{-3}}$	Total beam power source
+	`SCUBM`	$\mathrm{kg\ m/s^2/m^3}$	Toroidal momentum source due to NB
+	`SNEBM`	$10^{19}\,\mathrm{m^{-3}\,s^{-1}}$	Electron source due to NB
+	`SNNBM`	$10^{19}\,\mathrm{m^{-3}\,s^{-1}}$	Warm neutral source due to NB
+	`SNIBMj`	$10^{19}\,\mathrm{m^{-3}\,s^{-1}}$	Source of ions with the energy `EBEAM/j, j`=1,2,3

Some frequently used quantities are given in Table 4.25. Density and temperature distributions of the ‘cold’ neutral atoms are calculated by the subroutine `NEUT` as described

Table 4.25. Auxiliary plasma parameters

Type	Variable	Units	Description
$*$	`VP`	m/s	$U_\parallel/(d\psi/d\rho) = E_\parallel/B_\mathrm{p}$, pinch velocity
$*$	`VPFP`	m/s	$0.5\rho\dot{B}/B$, flux surface velocity
+	`ER`	V/m	Radial electric field
+	`VPOL / VPOLX`	m/s	Poloidal rotation velocity
+	`VTOR / VTORX`	m/s	Toroidal rotation velocity
+	`NN`	–	Relative neutral density, $N(\rho)/N(\rho_B)$
+	`TN`	keV	Neutral temperature, $T_N(\rho)$
+	`PBLON`	$10^{19}\,\mathrm{keV/m^3}$	Longitudinal pressure of fast ions
+	`PBPER`	$10^{19}\,\mathrm{keV/m^3}$	Perpendicular pressure of fast ions
+	`PELON`	$10^{19}\,\mathrm{keV/m^3}$	Longitudinal electron pressure
+	`PEPER`	$10^{19}\,\mathrm{keV/m^3}$	Perpendicular electron pressure

in Section 4.11.

Table 4.26 contains parameters characterizing the tokamak magnetic configuration. The quantities `EQFF` and `EQPF` give the right hand side of the Grad-Shafranov equation Eq. (36). All others are obtained as a result of solving this equation. Different average magnetic field characteristics are used in transport coefficients while purely geometric quantities are included in the transport set of equations as metric coefficients.

Table 4.26. Geometric and magnetic field characteristics

Type	Variable	Units	Description
+	EQFF / EQPF	$\mathrm{MA\,m^{-2}}$	$j_\zeta(r,z) = R_0/r * \mathtt{EQFF} + r/R_0 * \mathtt{EQPF}$
+	BMAXT / BMINT	T	Maximum / minimum B field on a surface
+	B0DB2 / BDB02		$< B_0^2/B^2 >$ / $< B^2/B_0^2 >$
+	BDB0		$< B/B_0 >$
+	FOFB		$< B_0^2/B^2(1. - \sqrt{1 - B/B_{max}}(1 + .5B/B_{max}) >$
+	GRADRO		$< \lvert\nabla\rho\rvert >$, lateral area S_a =VR*GRADRO
+	DRODA / DRODAX		$\mathrm{d}\rho/\mathrm{da}$
+	G11 / G11X	m^2	$< (\nabla\rho)^2 > V'$
+	G22 / G22X	m	$V' R_0/(4\pi^2 J) < (\nabla(\rho)/r)^2 >$
+	G33 / G33X		$< R_0^2/r^2 >$
+	IPOL / IPOLX		J, Normalized poloidal current Eq. (23)

The next group of parameters (Table 4.27) describes plasma isotope composition and impurity species. These quantities are used in subroutines `NCLASS, NBI, STRAHL` and in many other code modules dealing with the transport properties, additional heating and so on.

Table 4.27. Plasma isotope and impurity composition

Type	Variable			Units	Description
+	NHYDR	NDEUT	NTRIT	$10^{19}\mathrm{m}^{-3}$	Hydrogen isotope density
+	NHE3	NALF		$10^{19}\mathrm{m}^{-3}$	Helium isotope density
+	NIZ1	NIZ2	NIZ3	$10^{19}\mathrm{m}^{-3}$	Impurity species density
+	ZIM1	ZIM2	ZIM3	e_p	Impurity species charge
+	ZEF1	ZEF2	ZEF3		Impurity contribution to Z_{eff}
+	DIMP1	DIMP2	DIMP3	m^2/s	Diffusion coefficient
+	VIMP1	VIMP2	VIMP3	m/s	Impurity pinch velocity
+	PBOL1	PBOL2	PBOL3	$\mathrm{MW\,m^{-3}}$	Contribution to bolometric power
+	PSXR1	PSXR2	PSXR3	$\mathrm{MW\,m^{-3}}$	Contribution to soft X-ray radiation

The group of variables used in the additional transport equations (85), (86) is given in

Table 4.28. They are analogous to the variables used for the main plasma parameters such as NE or TE.

Table 4.28. Arrays associated with dummy unknowns F1, F2, F3

Type	Variable			Units	Description
+	F1	F1O	F1X	$[f_1]$	$f_1(t)$ / $f_1(t-\tau)$ / $f_1^{exp}(t)$
+	F2	F2O	F2X	$[f_2]$	$f_2(t)$ / $f_2(t-\tau)$ / $f_2^{exp}(t)$
+	F3	F3O	F3X	$[f_3]$	$f_3(t)$ / $f_3(t-\tau)$ / $f_3^{exp}(t)$
+	DF1	DF2	DF3	m^2/s	Diffusion for the variable f_j
+	VF1	VF2	VF3	m/s	Pinch velocity for the variable f_j
+	QF1	QF2	QF3	$[f_j]\mathrm{m}^3/\mathrm{s}$	Entire flux for the variable f_j
+	SF1	SF2	SF3	$[f_j]/\mathrm{s}$	Explicit rhs for the variable f_j
+	SFF1	SFF2	SFF3	1/s	Implicit rhs for the variable f_j
+	SF1TOT	SF2TOT	SF3TOT	$[f_j]/\mathrm{s}$	Total source for the variable f_j

Finally, two groups of vector variables without predetermined meanings are defined. All can be used similarly to the scalar variables listed in Table 4.17. Additionally, those of them with an 'X' at the end can be read in from the data file (see Section 5.3).

Table 4.29. Dummy arrays

CAR1	CAR2	CAR3	CAR4	CAR5	CAR6	CAR7	CAR8
CAR9	CAR10	CAR11	CAR12	CAR13	CAR14	CAR15	CAR16
CAR17	CAR18	CAR19	CAR20	CAR21	CAR22	CAR23	CAR24
CAR25	CAR26	CAR27	CAR28	CAR29	CAR30	CAR31	CAR32
CAR1X	CAR2X	CAR3X	CAR4X	CAR5X	CAR6X	CAR7X	CAR8X
CAR9X	CAR10X	CAR11X	CAR12X	CAR13X	CAR14X	CAR15X	CAR16X

4.7 Astra expressions

4.7.1 Types of expressions

Astra expressions provide a convenient notation for defining physical quantities or discharge characteristics which can be expressed by a formula. Each formula/expression can be referred to by a name and used in other expressions as described in Section 5.2. Typical examples are transport coefficients, power and particle sources and sinks, cross-sections of atomic processes and so on. Once these expressions are programmed they are stored in an object library or as Fortran text files. The Astra compiler then ensures the appropriate treatment of all modules and includes them into the resulting code according to their meaning, implementation and application. The whole concept is rather flexible: the expressions can be radially and/or time dependent, they can include integrals and derivatives, depend on other expressions and plasma parameters. In many cases, the Astra expressions are self-documented because they contain comments, examples of usage in the code, and references to papers from which they originate.

On the one hand, the number of Astra expressions is continuously increasing, on the other hand, some which have become obsolete are retained for backward compatibility. Therefore, instead of comprehensively describing all Astra expressions, the goal of this section is to give a general representation of the Astra expression library, to describe its structure, and to demonstrate how to find out more.

Astra supports two different kinds of expressions: formulae and functions. A beginning user of the code does not need to discriminate between functions and formulae and can treat them as identical. Moreover, the names of Astra expressions can coincide with the names of the vector variables described in the previous section. The only essential difference for a user is that, unlike arrays, expressions can be modified and new ones can be added. A user can delete any formula (including those in the Astra kernel) and replace it with his own. A function provided with the code cannot be deleted, however, this is not very restrictive because a user can always create his own function with a new name. Although no detailed explanation of the expression syntax is provided here, a number of expressions in the code can be used as templates for designing new ones.

The actual difference formulae and functions is that the formulae are included in a source code with the `INCLUDE` statement, while functions are conventional Fortran functions. Although there is no clear distinction, functions tend to be used for programming more complicated expressions than formulae.[1] Nevertheless, many expressions are represented both by functions and by formulae. Usage of formulae is preferable because it does not require rebuilding libraries and usually results in a faster executable code than using functions. As already mentioned, the proper usage is handled by the Astra compiler so that a user need not go into these details.

In this description, only the overview and usage of Astra expressions are presented. Usually, a name of variable/expression has a mnemonic meaning related to its use in the code. So the expressions supposed to be used as the matrix elements or the right hand sides

[1] This sequence can be continued by the Astra subroutines which are described in Section 4.9 and related to the processes described by equations rather than by expressions.

in Eqs. (82)–(87) start with the same letters as the corresponding terms of the equations. With a few exceptions the first letter of an expression name has the following meaning

`B`	– contributions to β's,
`C`	– conductivity,
`D`	– diffusivity,
`F`	– predefined radial functions,
`H` or `X`	– heat conductivity, (`'H'` is used for electron, `'X'` for ion heat conductivity),
`I`	– surface integral (applied for current density),
`L` or `R`	– plasma inductance or characteristic length,
`N`	– plasma collision frequencies,
`P`	– heat source or sink,
`Q` or `W`	– volume integral of a source or of energy content, respectively,
`S`	– particle source,
`SV`	– $< \sigma v >$ for collision processes,
`V`	– velocity.

More detailed information about every particular expression can be found in its source file. The source files for Astra formulae are placed in the directory `AWD/fml/`, functions in the directory `AWD/fnc/`, arrays are listed in the file `AWD/for/status.inc`.

4.7.2 List of expressions

In an Astra model (see Section 5.2) both upper- and lowercase letters can be used. Below we write all Astra expressions in uppercase but emphasize that all file names for Astra expressions consist of lowercase letters only. For instance, a source file for the expression `QDT` appears as `AWD/fml/qdt` (formula) and as `AWD/fnc/qdt.f` (function). Other expressions can be presented either by a formula or by a function only. A user should also be aware that all variables associated with Astra expressions are interpreted by the Astra code as Fortran real variables independently of the first letter of the name.

Conductivity and CD efficiency

CCSP	CCSPX	CNHH	CNHR	CCNEU	CCNEU	CHOTF	EFLHN	EFLHW

Bootstrap current density

DCHA	HCHA	XCHA	DCHH	HCHH	XCHH	DCHR	HCHR	XCHR
DCKIM	HCKIM	XCKIM	DCKM1	HCKM1	XCKM1	DHKIM		

Transport coefficients

CERL	DBOHM	DNDIF	DNEXP	ENHH0	ENHH1	ENHH2	FOWC	HAALC
HABM	HABMS	HABOM	HACTE	HAED	HAETI	HAETS	HAGB	HAGBS
HAGBS1	HAITF	HAITR	HAMM	HANAG	HANAL	HAPA	HAPUE	HAPYU
HAQ1	HAQ1C	HARL	HARLS	HARNQ	HARPL	HASCL	HATL	HATLI
HATLS	HBJET	HCHII	HCHGP	HEEFF	HEGN	HETAI	HETIS	HEXP
HGBEJ	HGBIJ	HIT89	HNCHI	HNGSB	HNGSE	HNGSI	HNGSP	HNPSI
TECRL	XCH86	XCHR	XEXP	XIGN	XIEFF	XIRL		

Power sources and sinks

PAION	PBDHE	PBEIE	PBICX	PBRAD	PDT	PEDT	PEDT1	PEGN
PEHCL	PEI	PEICL	PEIGN	PENEU	PENLI	PETSL	PHICL	PICX
PIDT	PIDT1	PIGN	PINEU	PIONZ	PIREC	PITCX	PITSL	PJOUL
POH	PRCAR	PRFER	PRNEO	PRNIT	PROXI	PROXY	PRWOL	PSYNC

Particle sources and sinks

SNNEU	SNNIE	SNNII	SNNR

Volume integrals of power and particle sources

QBREM	QDT	QEDT	QEGN	QEICL	QEIGN	QENEU	QETOT	QEX
QIDT	QIGN	QINEU	QITOT	QIX	QJOUL	QNTOT	QNX	QOH
QRAD	QRADX	QSYNC	QTOT	QBTOT	QEDWT	QIDWT	QNDNT	

Volume average quantities

NEAV	NEXAV	TEAV	TEXAV	TIAV	TIXAV	TENDN	TINDN	
WALF	WBPOL	WE	WEX	WI	WIX	WTOT	WTOTX	ZNDN

Confinement times

TAUE	TAUEE	TAUEI	TAUG	TAUNA	TAUP	TAU89	TITER

Averaged cross-sections of atomic and nuclear processes

SVCX	SVCXX	SVD1	SVD2	SVDBH	SVDHE	SVDT	SVIE
SVIEP	SVIEX	SVII	SVREC	SVRECX			

Collisions

COULG	NUE	NUEE	NUES	NUI	NUIS	NUPP

Velocities

CS	VDIA	VSI	VTE	VTI

Pressure and β ’s

BETE	BETI	BETPL	BETR	BETAJ	BETBM	BETT
ALMHD	PRESE	PRESI	PREST			

Integrals of current density

IBM	IBS	ICD	IOHM	ITOT	IX
IECR	IFI	IFW	IICR	ILH	

Properties of the current profile

LICD	LINT	QBM	QBS	QCD	QNIND	QMIN	SHEAR	XQMIN

Characteristic lengths and derived quantities

LNE	LNI	LNZ1	LTE	LTI	ETAN	ETAE	ETAI
RLTCR	RLTCZ	RLTWN	RLI	RLS			

Average Z in coronal approximation

ZICAR	ZIFER	ZINEO	ZINIT	ZIOXI	ZIWOL

Other plasma parameters

EPAR	EPL	EDR	D2TI	SQZ	DIDT	FTE	FTLLM	TPF
ROTSH	NECH	NELA	CCMHD	CUOHM	HMHD1	HMHD2	XMHD1	XMHD2

Radial functions (See Section 4.8.3)

FA	FLIN	FPA	FPR	FR	FRS	FX

4.8 Built-in functions

Below we list some of the modules (Fortran subroutines and functions) incorporated in the package. Headers of each module are equipped with a brief description and examples of their usage. Fortran sources for most of the functions discussed in this section are available in the file `AWD/for/intern.f`. [5] The general rules of calling Astra subroutines and functions are described in Section 5.2.

[5] Changes in this file are not allowed for a non-authorized user.

The following notations for the arguments of Astra functions and subroutines are adopted below:

R is a vector variable, $R = R(\rho, t)$,

S is a scalar variable or an Astra vector at a given radius, $S = S(t)$,

G is a generic name for both, $R(\rho, t)$ and $S(t)$,

t_0 is the initial time of the simulation run,

t is the current time, $t \geq t_0$,

t_1 is arbitrary time.

In the case where a built-in function allows a generic argument G the result has the same type as the argument. As a rule, only Astra variables (scalars and vectors) can be used as arguments of Astra built-in functions. To overcome this restriction one has to perform two steps as in the following example:

```
CAR1=WTOT;        CAR2=TIMDER(CAR1)
```

Because `WTOT` is an Astra formula, rather than an array, it cannot be directly used as an argument of the built-in function `TIMDER` (time derivative). Therefore, the first statement assigns the kinetic energy density $W_{\rm kin}(\rho)$ to the vector variable `CAR1`, then the second computes the energy derivative as the vector `CAR2`:

$$\texttt{CAR1} = \texttt{WTOT} = W_{\rm kin} = 0.0024 \int_0^\rho (n_e T_e + n_i T_i)\, V' d\rho \quad [\mathrm{MJ}], \qquad \texttt{CAR2} = \frac{\partial \mathrm{W}_{\rm kin}}{\partial \mathrm{t}} \quad [\mathrm{MW}].$$

In the next example

```
CV1=TIMDER(IPL);        CV2=TIMINT(QNB)
```

both results are scalars, where $\texttt{CV1} = dI_{\rm pl}/dt$ is the plasma current ramp rate and $\texttt{CV2} = \int_{t_0}^{t} \texttt{QN}(\rho_{\rm B}, t) dt$ is the total particle flux through the outermost magnetic surface.

The rule restricting the argument type has two exceptions. First, the functions `CUT`, `STEP` (see the end of Section 4.8.2) and Fortran intrinsic functions (Section 4.8.3) allow arbitrary expressions as arguments. Secondly, the values of Astra arrays at fixed radii (e.g. `QNB` in the example above) can also be arguments of the built-in functions.

4.8.1 Time dependent functions

Time derivative (function `TIMDER`).

$$\texttt{TIMDER}(G) = \left(\frac{\partial G}{\partial t}\right)\bigg|_\rho .$$

Time integral (function `TIMINT`).

$$\texttt{TIMINT}(G) = \int_{t_0}^{t} G([\rho,]\,t)dt.$$

Usage of functions `TIMDER` and `TIMINT` has an additional limitation. They can only be used in an Astra model on the right hand side of an assignment statement. This means that the constructions

```
      CV1=TIMDER(VOLUME);       dVdt_CV1
```

and

```
      CAR1=TIMDER(FP);          U_pl\CAR1
```

are correct while the result of `dVdt_TIMDER(VOLUME)` will be wrong.

Heaviside function of time (function `FJUMP`).

$$\texttt{FJUMP}(t_1) = H(t-t_1)\begin{cases} 0, & \text{if} \quad t < t_1 \\ 1, & \text{if} \quad t_1 < t. \end{cases}$$

Linear ramp in time (function `FRAMP`).

$$\texttt{FRAMP}(t_1, t_2) = \begin{cases} 0, & \text{if} \quad t \le t_1 \\ (t-t_1)/(t_2-t_1), & \text{if} \quad t_1 < t < t_2 \\ 1, & \text{if} \quad t_2 \le t. \end{cases}$$

Storing of a variable value at a given time (function `FIXVAL`).

$$\texttt{FIXVAL}(G, t_1) = \begin{cases} G(t), & \text{if} \quad t < t_1 \\ G(t_1), & \text{if} \quad t_1 < t. \end{cases}$$

Sliding time average. Function of two parameters $\texttt{FTAV}(G, \Delta t)$ provides smoothing of the variable $G(t)$ over the time interval Δt so that

$$\texttt{FTAV(G,}\Delta t)\Big|_t = \texttt{FTAV(G,}\Delta t)\Big|_{t-\tau} \exp\left(-\frac{\tau}{\Delta t}\right) + \texttt{G}(t)\left[1 - \exp\left(-\frac{\tau}{\Delta t}\right)\right],$$

with τ being the time step. In particular,

$$\begin{cases} \texttt{FTAV}(t) \approx G(t), & \text{if} \quad \tau \gg \Delta t \\ \texttt{FTAV}(t) \approx \texttt{FTAV}(t-\tau), & \text{if} \quad \tau \ll \Delta t. \end{cases}$$

Minimum/maximum value in time of a function G (FTMIN, FTMAX).

$$\texttt{FTMIN}(G) = \min_{t_0 \le \theta \le t} \left[G(\theta)\right], \qquad \texttt{FTMAX}(G) = \max_{t_0 \le \theta \le t} \left[G(\theta)\right].$$

Here $t_0 = \texttt{TSTART}$ is the initial time and $t = \texttt{TIME}$ is the current time of the run.

Examples:

`FJUMP(1.)` provides a time dependent function with the unity step at the time $t = 1$ s. The actual width of the transition phase is no less than the current time step $\tau = \texttt{TAU}$.

`FRAMP(1.,1.01)` extends the transition phase up to 10 ms (provided $\tau < 10$ ms, otherwise τ).

`MU - FIXVAL(MU,1.)` returns the finite difference $\Delta\mu(\rho,t) = \mu(\rho,t) - \mu(\rho,1.)$, if $t > 1$ s, and 0 otherwise.

`FTAV(UPL,.01)` returns a vector variable which is the loop voltage $U_{\rm pl}(\rho,t)$ averaged over the recent 10 ms (assuming that the time step $\tau \ll 10$ ms).

`FTAV(UPLB,.01)` is a scalar giving the average edge loop voltage $U_{\rm pl}(\rho_{\rm B},t)$.

`FTMIN(MU)` and `FTMAX(MU)` give a range of variation of $\texttt{MU} = 1/q$ during a run.

! `FTAV(HEXP,.01)` this usage is forbidden because `HEXP` is not an Astra array. Two statements `CAR1=HEXP; CAR2=FTAV(CAR1,.01)` should be used instead.

`TIMINT(UPL)` is a vector variable which gives the poloidal flux $\psi(\rho,t) = \psi(\rho,t_0) + \int_{t_0}^{t} U_{\rm pl}(\rho,t)dt$ up to an additive constant.

`TIMDER(FPB)` is a scalar variable and within numerical errors coincides with $U_{\rm pl}(\rho_{\rm B},t)$.

4.8.2 Radially dependent functions

Linear radial functions (FA, FX, FLIN, FR, FRS).

All functions listed above are implemented as Astra formulae. The function `FR` (array synonym `RHO`) gives the nodes of the main transport radial grid ρ_j

$$\rho_j = \begin{cases} h(j-0.5), & \text{if} \quad 1 \le j < \texttt{NA1} \\ \rho_B, & \text{if} \quad j = \texttt{NA1}. \end{cases}$$

Similarly, FA (array synonym AMETR) gives values of the variable a, i.e. $a_j = a(\rho_j)$ on this grid. Most frequently used functions are defined on this grid. Functions FX and FLIN both return the normalized radius ρ_j/ρ_B which can also be obtained as RHO/ROC.

The intermediate grid in ρ is given by the function $\texttt{FRS} = jh$. The functions listed in Table 4.5, all fluxes and some others are defined on this latter grid.

Parabolic profiles (FPR, FPA). The Astra formulae

$$\texttt{FPR} = 1 - (\rho/\rho_B)^2 \qquad \text{and} \qquad \texttt{FPA} = 1 - (a/a_B)^2$$

describe parabolic profiles in variables ρ and a, respectively.

Gaussian distribution (subroutine FGAUSS, function GAUSS).
The subroutine FGAUSS (ρ_0, Δ_ρ, R) returns a radial Gaussian profile in the variable ρ

$$R_i = N \exp\left[-\left(\frac{\rho_i - \rho_0}{\Delta_\rho}\right)^2\right],$$

where the factor N (of dimension m^{-3}) is determined by the normalization condition

$$\int_0^{\rho_B} R(\rho) dV = 1.$$

This means that the statement "FGAUSS(.5,.2,PEX):; " defines the vector variable PEX as a Gaussian profile, centered in $\rho_0 = 0.5$ m with the width $\Delta\rho = 0.2$ m. If the quantity PEX is defined as a power, then the total power deposited would be 1 MW.

The function GAUSS (ρ_0, Δ_ρ) can be used similarly: $\texttt{PEX} = \texttt{GAUSS}(0.5, 0.2)$, however, the function GAUSS can be called in a model only once. !!

Minimum/maximum value of a vector variable $R(\rho)$ (FRMIN, FRMAX).

$$\texttt{FRMIN}(R) = \min_{0\le\rho\le\rho_B}[R(\rho)], \qquad \texttt{FRMAX}(R) = \max_{0\le\rho\le\rho_B}[R(\rho)].$$

Position of minimum/maximum of a vector variable $R(\rho)$ (RFMIN, RFMAX).
The functions RFMIN, RFMAX return

$$\texttt{RFMIN}(R) = \rho_{\min}, \quad \text{where} \quad R(\rho_{\min}) = \min_{0\le\rho\le\rho_B}[R(\rho)],$$

$$\texttt{RFMAX}(R) = \rho_{\max}, \quad \text{where} \quad R(\rho_{\max}) = \max_{0\le\rho\le\rho_B}[R(\rho)].$$

Root of the algebraic equation $R(\rho) = R_0$ (RFVAL).

$$\mathtt{RFVAL}(R, R_0) = \rho_0, \quad \text{where} \quad R(\rho = \rho_0) = R_0.$$

Maximum of all possible solutions is returned, $\rho_0 = 0$ if no root found.

Root of the algebraic equation $R(a) = R_0$ (AFVAL).

$$\mathtt{AFVAL}(R, R_0) = a_0, \quad \text{where} \quad R(a = a_0) = R_0.$$

Maximum of all solutions is returned, 0 if no root found.

Gradient of a function $R(\rho)$ (GRAD).

$$\mathtt{GRAD}(R) = \frac{\partial R(\rho)}{\partial \rho}.$$

One should beware of the approximate character of this operation. The function GRAD is intended for auxiliary purposes such as graphical presentation or estimates only. For instance, it makes no distinction between the main and intermediate radial grids. Therefore, the safety factor q calculated as GP2*BTOR*RHO/GRAD(FP) shows significant deviation from the exact (in a finite difference sense) quantity $q = 2\pi B_0 \rho\, \partial\rho/\partial\psi = 1/\mu = \mathtt{1/MU}$ at the plasma centre. Even the improved evaluation $\mathtt{GP2} * \mathtt{BTOR} * \mathtt{FRS/GRAD(FP)}$ is not everywhere exact because it does not take into account the centering of different grid variables appropriately.

Volume integral of a function $R(\rho)$ (VINT).

$$\mathtt{VINT}(R) = \int\limits_0^{\rho} R(\rho) V' d\rho.$$

In Astra models the function VINT can be used in several different ways:

$$\mathtt{CAR1} = \mathtt{VINT(NE)}; \qquad \mathtt{CV1} = \mathtt{VINT(NEB)}; \qquad \mathtt{CF1} = \mathtt{VINT(NE, 0.5)};$$

The first statement in this example produces a vector, others scalar output. Note that the '0.5' as the second argument of VINT (in the 3rd statement) refers to the minor radius a measured in meters and introduced by Eq. (88). Therefore, this gives the following results:

$$\mathtt{CAR1} = \int\limits_0^{\rho} \mathtt{NE}(\rho) V' d\rho, \qquad \mathtt{CV1} = \int\limits_0^{a_B} \mathtt{NE}(a) dV = \int\limits_0^{\rho_B} \mathtt{NE}(\rho) V' d\rho, \qquad \mathtt{CF1} = \int\limits_0^{0.5\,\mathrm{m}} \mathtt{NE}(a) dV.$$

For instance, the density averaged over a volume inside the current radius ρ can be submitted to the radial graphics output by the two instructions:

$$\mathtt{CAR1} = \mathtt{1}; \qquad < \mathtt{ne} > \backslash\mathtt{VINT(NE)/VINT(CAR1)};$$

In the last example, `VINT(CAR1)` is equivalent to `VOLUM`. Moreover, the entire expression for $< n_e >$ can be obtained making use of already defined Astra function `NEAV`.

Exactly the same rules are applicable to the next Astra function `IINT` which also calculates an integral over the poloidal cross-section rather then over the volume of a magnetic surface.

Surface integral of a function $R(\rho)$ in a poloidal plane (`IINT`).

$$\mathtt{IINT}(R) = \frac{J}{2\pi R_0}\int_0^\rho R(\rho)\frac{V'}{J^2}d\rho.$$

This function is used for integration of the longitudinal current density (see Eq.(33)). Similar to `VINT` it allows expressions like `IINT(CUECR)`, `IINT(CUECRB)` and `IINT(CUECR,0.2)`.

As a general remark, we note that constructions involving `VINT` (or `IINT`) are rarely needed because Astra expressions are available for most physical quantities relevant to transport analysis. We also remind the reader that `VINT(1)` or `VINT(NE*TE)` are not allowed because the argument should be a radial array (Astra vector variable).

Heaviside function of the radial variable a (`ASTEP`).

$$\mathtt{ASTEP}(a_0) = H(a - a_0)$$

Heaviside function of the radial variable ρ (`RSTEP`).

$$\mathtt{RSTEP}(\rho_0) = H(\rho - \rho_0)$$

Heaviside function of the radial variable $x = \rho/\rho_B$ (`XSTEP`).

$$\mathtt{XSTEP}(x_0) = H(\rho/\rho_B - x_0)$$

Heaviside function of any dimensionless argument A (`STEP`).

$$\texttt{STEP}(A) = H(A) = \begin{cases} 1, & \text{if} \quad A \ge 0, \\ 0, & \text{if} \quad A < 0. \end{cases}$$

An argument A of the function `STEP` can be any expression allowed by the Astra rules. The same is valid also for the next function.

Cut-off function (`CUT`). Sometimes, a function for the graphic output varies by many order of magnitude. Because of a very limited word length (1 byte) of the data stored in an Astra post-view file (Section 5.5.4) essential information can be lost and saved data will have very low resolution. To avoid this one can cut off too large values of no interest:

$$\texttt{CUT}(a_0, A) = \begin{cases} -a_0, & \text{if} \quad A \le -a_0, \\ A, & \text{if} \quad -a_0 \le A \le a_0, \\ a_0, & \text{if} \quad a_0 \le A. \end{cases}$$

Examples:

`RSTEP(.5)` can also be obtained as `STEP(RHO-.5)` , `XSTEP(.2)` as `STEP(RHO/ROC-.2)` , `ASTEP(.1)` is equivalent to `STEP(AMETR-.1)` . Note that the numerical argument of `XSTEP` (0.2 in this example) is dimensionless, all others are measured in meters.

`CUT(100.,NUES)` can be useful for drawing a radial profile of the collision frequency which can be very large near the magnetic axis and at the plasma edge. The same effect can be achieved in a more complicated way as `100-(100-NUES)*STEP(100-NUES)`.

`STEP(SIN(GP2*100*TIME)-0.5)` shows another example of allowed usage for this function which gives a time dependent square-wave signal with a period 0.01 s (frequency $f = 100$ Hz) and a duty factor $(\pi/2 - \arcsin(0.5))/\pi = 33\%$.

`FRMAX(MU)` returns the minimum value of the safety factor $q(\rho) = 1/\mu(\rho) = $ `1/MU`,

`RFMAX(MU)` returns the $\rho-$ position of this minimum,

`RFVAL(MU,0.5)` returns the position of the $q = 2$ resonance surface in the coordinate ρ,

`AFVAL(MU,0.5)` returns the position of the same surface with respect to the coordinate a. The last result can also be obtained as `AMETR(RFVAL(MU,0.5))`.

4.8.3 Functions for mapping one flux label to another

Several Astra functions are available for recalculating one 'radial' coordinate to another: `RFA`, `XFA`, `AFR`, `AFX`, `RFAN`, `XFAN` . The function `RFA(0.5)` returns a value of the coordinate ρ (in meters) at the magnetic surface labeled by the mid-plane radius $a = 0.5$ m (see Eq. (88)). The function `XFA`(a) is equivalent to `RFA`(a)/`ROC` and returns the normalized value of the flux label $x(a) = \rho(a)/\rho_B$. The functions `AFR` and `AFX` are inverses of `RFA` and `XFA`, respectively. Finally, `RFAN(0.5)` gives ρ as a function of the dimensionless argument $a_N = a/a_B = 0.5$, similarly, `XFAN`$(a_N) =$ `RFAN`$(a_N)/\rho_B$.

Vector variable at the given radius can be computed using one of the two functions `ATX` and `ATR`. The difference between both is that the first one refers to the normalized radius ρ/ρ_B and the second to the dimensional radius ρ. For instance, `ATX(TI,0.5)` returns a value of the ion temperature at $\rho = 0.5\rho_B$ while `ATR(TI,0.5)` at $\rho = 0.5$ m. Other options are discussed in Section 5.2.3.

4.8.4 Fortran intrinsic functions

In addition to special functions described above one can use also the standard set of Fortran functions: `SIN`, `COS`, `TAN`, `ASIN`, `ATAN`, `EXP`, `ABS`, `MIN`, `MAX`, `SQRT`, `ALOG`, `ALOG10`, `ANINT`, `REAL` and `SIGN`. The type of the argument can be anything. In turn, it defines a type of the result so that `SIN(CV1)` is a scalar, while `SQRT(NE*TE)` and `ALOG(LTE)` are vectors.

4.9 Plug-in subroutines

All subroutines described in this section can be found under the name `AWD/sbr/xxx.f` where `'xxx'` should be replaced by a name referred below in the lowercase. As a matter of fact, the directory `AWD/sbr/` includes many more subroutines which can be useful in different applications. We describe here only few of those.

4.9.1 Ionization particle flux (`GNEX, GNXSRC`).

Both subroutines have no parameters and communicate with the Astra core through common blocks. The subroutine `GNEX` calculates the particle flux `GNX` caused by ionization of neutral atoms in the plasma volume and also due to evolution of the electron density

$$\mathtt{GNX} = \frac{1}{S_a}\int\limits_0^\rho \left(V'S_e - \frac{\partial V'n_e}{\partial t}\right) d\rho,$$

where the toroidal surface area $S_a =$ `SLAT` is defined by Eq. (5). It is assumed that the density of wall neutrals `NN` and a source due to the beam neutrals `SNEBM` are calculated independently (for instance, by calling subroutines `NEUT` and `NBI`). `GNXSRC` gives the same quantity but without allowance for the neutrals from NBI.

The quantity `GNX` is *always* included in Eq. (82). It can be useful when the particle diffusion equation is switched off and the density evolution is taken from an experiment and is a prescribed function of time and radius. Then the 'experimental' particle flux `QN=SLAT*GNX` will be included in the power balance in accordance with Eq. (82),(87) unless $\gamma_e = \gamma_i = 0$. On the other hand, the user should be aware that calling any of the subroutines `GNEX` or `GNXSRC` together with solving the diffusion equation will cause a side effect and include the same particle flux twice.

4.9.2 Upper and lower function boundaries (MINMAX).

A call to the subroutine `MINMAX`(R, R_{min}, R_{max}) returns two quantities

$$R_{min}(\rho) = \min_{t_1\le\theta\le t_2} R(\rho,\theta) \qquad \text{and} \qquad R_{max}(\rho) = \max_{t_1\le\theta\le t_2} R(\rho,\theta).$$

Although the same result can be obtained by calling the two functions `FTMIN` and `FTMAX` using `MINMAX` is more flexible because it finds maxima and minima in a prescribed time interval $t_1 < t < t_2$. So the statement

```
    MINMAX(CU,CAR1,CAR2)::0.5:0.6
```

gives range of variation of the current density for 100 ms inside the time interval 0.5 s $<$ $t <$ 0.6 s.

4.9.3 Sawtooth oscillations (internal disruption) (subroutine MIXINT).

This subroutine models profile evolution of $T_e, T_i, n_e, n_i, j_\parallel, \mu, \psi$ in the process of magnetic field line reconnection according to Kadomtsev's theory of sawtooth oscillations [17]. The current version of the code allows for the internal reconnection at the resonance $q = m/n = 1$ only. In a process of the full reconnection final distributions of all plasma parameters are uniquely determined by the conservation laws and by the initial distributions. The only physical characteristic missing in this theory is the trigger of the internal disruption. Therefore, the subroutine `MIXINT` requires one physical input parameter which can be either the sawtooth period or the reconnection radius. Additionally, it is possible to use a disruption condition based on the double-tearing-mode reconnection model [18]. In this case no additional parameters are required. A statement calling the subroutine reads

```
MIXINT(OPTION,RECON):
```

Both input parameters must be given as real Fortran variables or constants and have the following significance:

`OPTION` $= 0$: if a reconnection takes place at $t = t_1$ then the next one occurs not earlier than at $t = t_1 +$ `RECON` and as soon as the resonance surface $q = 1$ is found,

`OPTION` $= 1$: a reconnection occurs as soon as the radius ρ_1 of the resonance surface $q = 1$ satisfies the condition $\rho_1/\rho_\mathrm{B} >$ `RECON`,

`OPTION` $= 2$: a reconnection occurs according to the model of [18]. The second parameter `RECON` is ignored.

The header of the subroutine `MIXINT` describes other options which provide extended information about every sawtooth event.

Reconnection mixes up all plasma parameters, however, in the code, only those quantities are redistributed which evolution is described by a transport equation. For instance, if the equation for `NE` is not solved then the profiles of `NE` and `NI` do not change in the course of the reconnection described by `MIXINT`. Similarly `FP, MU, CU` do not change if the current density distribution is prescribed.

4.9.4 Gas puff neutrals (subroutine `NEUT`).

The subroutine `NEUT` solves an equation for the distribution of ‘cold’ (to distinguish from ‘fast’ neutral particles from NBI) neutral atoms in the plasma volume. The solution method first replaces the differential kinetic equation (78) with an integral equation [2] then the integral equation is solved iteratively. The parameter input/output for the subroutine can be schematically depicted as

$$\begin{tikzcd}\boxed{\textbf{Transport}} \arrow[r, shift left, "a_B,\ A_i,\ Z_i,\ n_e,\ T_{e,i};\ N_{1,2},\ E_{1,2},\ N_{iter}"] & \boxed{\texttt{NEUT}} \arrow[l, shift left, "N(\rho),\ T_N(\rho),\ \alpha_{\rm pl}"] \end{tikzcd}$$

Most of the exchange parameters were already discussed. Those which are directly related to the neutral atom distribution are listed in the following table.

Parameter	**Code name**	**Units**	**Description**
N_1	NNCL	$10^{19}\mathrm{m}^{-3}$	Density of incoming neutrals (1st component)
N_2	NNWM	$10^{19}\mathrm{m}^{-3}$	Density of incoming neutrals (2nd component)
E_1	ENCL	keV	Energy of incoming neutrals (1st component)
E_2	ENWM	keV	Energy of incoming neutrals (2nd component)
N_{iter}	NNCX		Number of iterations in NEUT
$N(\rho)$	NN*(NNCL+NNWM)	$10^{19}\mathrm{m}^{-3}$	Density of neutral particles
$T_N(\rho)$	TN	keV	Temperature of neutral particles
$\alpha_{\rm pl}$	ALBPL		Plasma albedo

The subroutine allows for two mono-energetic components of incoming neutral particles which are characterized with their densities N_1, N_2 and energies E_1, E_2 (NNCL,NNWM and ENCL,ENWM), respectively. It is assumed that the neutral mass coincides with a mass of the working gas given by the variable AMJ. A control parameter NNCX (default value 20) describes a number of iterations for solving the integral equation.

Output of the subroutine NEUT provides the normalized neutral density NN and the temperature TN distributions of the neutral particles in plasma and, additionally, the plasma albedo ALBPL. Optionally, it is also possible to calculate a distribution function of the outgoing neutral particles. The quantity NN is normalized in such a way that, at the plasma edge, the density of incoming neutrals is $N_1 + N_2 = \mathtt{NNCL} + \mathtt{NNWM}$ and the density of outgoing neutrals is $N_{\rm B} - N_1 - N_2 = (\mathtt{NNCL} + \mathtt{NNWM}) * (\mathtt{NNB} - 1)$, where $N_{\rm B} = N(\rho_{\rm B})$ and $\mathtt{NNB} = \mathtt{NN}(\rho_{\rm B})$. Consequently, the average energy of incoming neutrals is

$$E_{\rm in} = \frac{N_1E_1 + N_2E_2}{N_1 + N_2} = \frac{\mathtt{NNCL} * \mathtt{ENCL} + \mathtt{NNWM} * \mathtt{ENWM}}{\mathtt{NNCL} + \mathtt{NNWM}}$$

and outgoing

$$E_{\rm out} = \frac{N_{\rm B}T_{\rm NB} - N_1E_1 - N_2E_2}{N_{\rm B} - N_1 - N_2} = \frac{\mathtt{NNB} * \mathtt{TNB} - \mathtt{NNCL} * \mathtt{ENCL} - \mathtt{NNWM} * \mathtt{ENWM}}{(\mathtt{NNCL} + \mathtt{NNWM}) * (\mathtt{NNB} - 1)}.$$

Finally, the inward flux is given by

$$\Gamma_{\text{in}} = 4.44 \times 10^{24} \frac{\left(N_1\sqrt{E_1} + N_2\sqrt{E_2}\right)}{\sqrt{A_i}}, \qquad \text{m}^{-2}\text{s}^{-1}$$

and the outward flux by

$$\Gamma_{\text{out}} = \alpha_{\text{pl}}\Gamma_{\text{in}}.$$

4.9.5 Density adjustment (SETNAV).

This subroutine adjusts the density of incoming cold neutrals in order to provide a required volume average electron density. The following algorithm is adopted. At each time step the particle confinement time τ_n is evaluated $\tau_n =< n_e >/(S - d < n_e >/dt)$. Then the density of the cold neutrals (parameter `NNCL`) is scaled in order to provide $S_{\text{req}} =< n_{e,\text{req}} > /\tau_n$. The resulting density evolution is approximately described by the 0D equation

$$\frac{d < n_e >}{dt} = \frac{< n_{e,\text{req}} > - < n_e >}{\tau_n} + S - S_{\text{req}},$$

where S_{req}, τ_n and $< n_{e,\text{req}} >$ are taken from the previous time step. If initially the actual density $< n_e >$ is far from the requested value $< n_{e,\text{req}} >$ the described procedure can give an unreasonably large neutral influx which subsequently crashes the code. In order to avoid this, an additional control parameter τ_N is introduced which restricts the rate of variation `NNCL` $= N_1$ by the condition $N_1/(N_1/dt) < \tau_N$. This parameter should be selected as $\tau_N \geq \tau_n$. As another option, τ_N can be set negative. Then an automatic adjustment will be enabled which works reasonably if the difference $|< n_{e,\text{req}} > - < n_e >|$ is not too large.

Example:

The part of the model related to `SETNAV` reads:

```
CF1=5+5*FRAMP(.5,1);       ! Prescribed volume average density
SETNAV(CF1,.01):;          ! τN = 0.01 s
```

This will result in a ramp of the electron density as prescribed by the first statement.

4.9.6 Smoothing a function (subroutine SMEARR).

The subroutine $\texttt{SMEARR}(\alpha, R_{\text{in}}, R_{\text{out}})$ finds a vector function $R_{\text{out}}(\rho)$ as an extremum of the functional

$$\min \int_0^{\rho_B} \left[\alpha \left(\frac{\partial R_{\text{out}}}{\partial \rho} \right)^2 + \left(R_{\text{in}}(\rho) - R_{\text{out}}(\rho) \right)^2 \right] d\rho$$

with the boundary conditions

$$\left.\frac{\partial R_{\text{out}}}{\partial \rho}\right|_{\rho=0} = 0, \qquad R_{\text{out}}(\rho_B) = R_{\text{in}}(\rho_B)$$

where $R_{\text{in}}(\rho_j)$ is a given grid function, α is a control parameter and R_{out} is the output function. Reasonable values of the control parameter α should be around 0.001. The value $\alpha = 0.01$ results in a very strong smoothing of the result.

Example:

```
CAR1=HEEFF;       SMEARR(0.002,CAR1,CAR2)::;
```

This gives a radially smoothed effective electron heat conductivity

$\chi_{e,eff} = \int (P_e - \partial W_e/\partial t)\, dV/V' < (\nabla\rho)^2 > n_e$.

4.9.7 Store array / variable value at a given time (STARR / STVAR).

The subroutine STARR(R, t_1, R_1) (STVAR(S, t_1, S_1)) stores a vector (simple) variable at a given time t_1 :

$$R_1(\rho) = R(\rho, t = t_1), \qquad\qquad S_1 = S(t = t_1).$$

Note that in recent versions of the code both subroutines can be replaced by the function FIXVAL.

4.9.8 User defined time step control (subroutine TSCTRL).

The standard procedure of automatic time step control implemented in the Astra code restricts the one-time-step variation in the main unknowns TE, TI, NE, Fj . In some cases, for instance for stiff transport models, more strict control is required.

Such a control can be achieved by calling the subroutine TSCTRL$(R_1, R_2, R_3, \Delta_{\max})$. First, a variation in every vector input quantity R_i, where $i = 1, 2, 3$,

$$\Delta_{Ri} = \max_{\rho} \left| \frac{R_i(\rho, t) - R_i(\rho, t - \tau)}{R_i(\rho, t)} \right|$$

is evaluated. Then the time step τ is multiplied by $\max(\Delta_{\max}/\Delta_{Ri}, \tau_{\text{inc}})$. The quantity $\tau_{\text{inc}} =$ TAUINC is described in Table 4.15 and $\Delta_{\max}$ is the real input parameter which can vary between 0.1 and 10. depending on the controlled quantities R_i. In all cases, the time step τ is bounded by TAUMIN $< \tau <$ TAUMAX.

A limitation is that the subroutine TSCTRL can be called from a model only once. To monitor several array parameters it allows up to three different vector arguments.

Examples:

1) In a stiff transport model small variation in gradient can cause huge variation in flux. Therefore, plasma parameters like density or temperatures cannot provide reliable time step control. More appropriate regulation can be provided by

```
TSCTRL(QI,QE,QN,1.):;
```

2) Another way to achieve a similar result is

```
TSCTRL(XI,XI,XI,CF1):;
```

Here variation of a single quantity `XI` is checked. The strength of the control can be adjusted interactively by changing `CF1`.

4.10 Interfaces to additional heating and CD modules

4.10.1 General features.

The Astra system provides additional subroutines developed by different authors for the main methods of additional plasma heating in tokamaks. Each of the subroutines is, in fact, a large package and we do not pretend to describe them in this report. We discuss here the main features of the interfaces only, referring to the original descriptions for more detail.

Input. The following set of input data is usually required for additional heating and current drive modules:

Configuration: R_0 (`RTOR`), a_B (`AB` or `ABC`), $a(\rho)$ (`AMETR`), $\Delta(\rho)$ (`SHIF`), $\lambda(\rho)$ (`ELON`), $\delta(\rho)$ (`TRIA`), metric coefficients.

Plasma parameters: A_j (`AMJ` or `AMAIN`), $Z_{eff}(\rho)$ (`ZEF`), Z_j (`ZMJ` or `ZMAIN, ZIM1, ZIM2, ZIM3`), $n_{e,j}(\rho)$ (`NE, NI, NHYDR, NDEUT,... NIZ1, NIZ2, NIZ3`), $T_{e,j}(\rho)$ (`TE, TI`). Here index j extends over all ion species.

Current and **magnetic field:** $I_{\rm pl}$ (`IPL`), B_0 (`BTOR`), $\mu(\rho)$ (`MU`), $j_{||}(\rho)$ (`CU`).

Launcher or **neutral beam parameters.**

Control parameters for the numerical algorithm.

Output. All modules return power deposition profiles, $P^H_{e,i}(\rho)$, for electron and ion plasma components and, when applicable, driven current profile, j_{CD}. Many other characteristics as injected momentum, distribution functions, antenna coupling or shine through and so on are usually also available.

The set of input parameters can be split into two groups. The first one includes those parameters, which are time-dependent or can change from shot to shot. The second group describes construction characteristics of a particular tokamak or its heating system. In most cases these parameters do not often change and only require be set once. It is natural to exclude these parameters from the Astra core and localize them inside the modules for additional heating. The input parameters of the first group and the main output parameters are treated as standard Astra control parameters which means that they are accessible from the Astra code, can be defined, changed, plotted using the standard Astra service as described in Sections 5.2 and 5.3.

Different heating and current drive packages briefly described in Section 3.14.1 can be obtained by special request. The only exception is the NBI package which is always included in the Astra code. For this reason the interface to the NB heating package is described in more detail in the next Section.

4.10.2 NB heating and current drive (subroutine `NBI` by A. R. Polevoi)

A neutral beam installation is represented in the NBI package by a set of "pencil beams". Each pencil beam is characterized by its geometry and the power distribution across the beam axis which in turn is described by a number of thin parallel beams. Presently, up to 16 pencil beams (or beam sources with the different geometry) is allowed.

The NBI package considers (i) attenuation of the neutral beam during its passage through a plasma due to ionization and charge-exchange, (ii) capture and losses (including ripple losses) of the new-born ions by analysis of the ion drift trajectories, (iii) thermalization of the supra-thermal ions and their contribution to plasma heating, current drive and toroidal rotation. The latter problem requires solving the Fokker-Planck equation. In the fast standard version, an analytic solution of the steady-state equation is used. A time dependent version is also available.

Interface between the Astra core and the neutral beam package is provided by the subroutine AWD/sbr/nbi.f . As all other Astra plug-in modules this subroutine can be modified by the user and, therefore, the description below should be viewed as a framework providing rather guidelines than a comprehensive manual.

The usual set of general discharge and device parameters (see Section 4.10.1) are transferred to the subroutine NBI through common blocks and is not discussed here. Additionally, a number of specific parameters describe the properties of the NBI solver and the geometric characteristics of every pencil beam. First of all, 8 control parameters are used to set different features of the NBI solver. The control parameters are described in the Table 4.30 as they are used in the subroutine AWD/sbr/nbi.f although, in general, every C-parameter (see Table 4.17) can be used to perform every control function.

Table 4.30. Control parameters for the subroutine NBI

Name	Description
CNB1	A number of NBI sources (a number of pencil beams)
CNB2	Explicit (=1) or implicit (=0) account for the charge exchange losses
CNB3	Space grid for NBI routine N1=(NA1-1)/CNB3+1
CNB4	Steady state (=1) or time dependent (=2) Fokker-Planck solver
CNBI1	Fast ion charge exchange losses due to cold neutrals
CNBI2	Fast ion charge exchange losses due to NBI neutrals
CNBI3	FP solver time step: $\tau_{NBI} = \texttt{CNBI2} * \texttt{TAU}$
CNBI4	FP solver velocity grid size: IV1=80/CNBI4+1

The parameter CNB1 gives a number of pencil beams which have to be taken into account. It can also take negative values which invoke an interactive mode for modifying the beam geometry and will be discussed later.

The parameter CNB2 defines a way of describing the ion thermal losses due to the charge exchange between the beam neutrals and plasma ions. These losses can be computed with the formula PBICX=-0.0024*SNNBM*TI where the source of thermal neutrals SNNBM is calculated by the NBI package. The corresponding losses can either be included into the ion heating source PIBM or calculated elsewhere. If the parameter CNB2 is positive then the quantity PBICX is subtracted from the ion heating source PIBM. Otherwise, the losses should be taken into account in a model as shown below

Model 1 (recommended)	Model 2	Model 3
CNB2=0	CNB2=0	CNB2=1
PI=PIBM+...	PI=PIBM-PBICX+...	PI=PIBM+...
PIT=-0.0024*SNNBM+...		

Models 2 and 3 in this example are fully equivalent, however, the ion heating term `PIBM` is different in these two cases. The model 1 enables implicit treatment of the charge exchange losses which can result in more stable numerical scheme when these losses are high enough.

The parameter `CNB3` allows to run NBI routine on the reduced radial grid in order to save computing time. This is reasonable when the fast ion drift trajectories are wide and spread over many Astra radial grid steps. Finally, the parameter `CNB4` defines a type of the Fokker-Planck solver to be used. The C-parameters `CNBI1` ÷ `CNBI4` are effective if `CNB4=2` only. Otherwise they can be anything.

As mentioned every beam source is represented by a pencil beam. A sort of neutral atoms in the beam, their energy composition and power distribution across the beam, geometry and some other characteristics (altogether about 20 parameters) should be set separately for every pencil beam. These parameter are stored in an NBI configuration file which can be either attributed to every input data file (Section 5.1) or to every experimental setup. It is done in the following way. Assume that the Astra code is started with the data input file `aug10000`. When the `NBI` routine is called the code checks whether the file `AWD/exp/aug10000.nbi` exists. If yes, then this file, otherwise, the default configuration file `AWD/exp/aug.nbi` will be used. The latter can also serve as a template for creating new configuration files.

The NBI configuration file can start with arbitrary number of comment lines. A comment line is a line with the sign `'!'` in the first column. The comment lines are allowed in the beginning of the configuration file only. A meaningful part of the file consists of `CNB1` groups, one group for every pencil beam. Each group consists of 21 fields. Each field occupies 12 positions and can be filled either with a name of C-parameter (Table 4.17) or Z-parameter (Tables 4.18 and 4.19) or with a number in the free format as shown in the example below:

```
 1
 ZRD1       0.0000E+00  2.0000E+00  1.0000E+00  6.0000E+01
 5.0000E-01  3.0000E-01  2.0000E-01  1.0000E+00    CF1                  (90)
-1.9570E-01  6.5000E-01  4.0000E-01  8.5500E-02  1.0000E+00
 1.0000E+00  2.0000E+00  1.0000E+00  2.0000E+00  0.0000E+00
```

The first line in every group is also ignored by the code. It gives the ordinal number of the group (or of the corresponding pencil beam) and is useful for manual creating or editing the configuration file only. The other parameters have the following significance

Table 4.31. Parameters describing each pencil beam

No.	Units	Description
1	[MW]	Pencil beam power
2	–	Counter injection fraction (0 for co-, 1 for counter-injection)
3	–	Mass of the beam ions in the proton mass (can be 1,2,3)
4	–	Charge of the beam ions in the proton charge units
5	–	Maximum beam energy
6	–	Power fraction of the full (maximum) energy component
7	–	Power fraction of the one half energy component
8	–	Power fraction of the one third energy component
9	–	Type of averaging over the fast ion orbits
10	–	Number of thin beams in the horizontal plane
11	[m]	Vertical shift in the beam footprint centre
12	[m]	Maximum major radius in the beam footprint
13	[m]	Minimum major radius in the beam footprint
14	–	$\tan\alpha$, α being the angle between the beam axis and the midplane
15	–	Footprint aspect ratio
16	–	These 4 parameters prescribe the beam power distribution across the
17	–	pencil beam. The parameters are transferred to the user defined functions
18	–	`NBFHZ` and `NBFRY` (see the file `AWD/sbr/nbuser.f`) under the names
19	–	`CVER1, CVER2, CHOR1, CHOR2`, respectively
20	–	Unused parameter

The first parameter gives the power injected by each beam source. This quantity is usually time dependent, therefore, it is convenient to set it as a variable rather than as a number. In the example (90), this variable is `ZRD1` $\equiv$ `ZRD1X` and its time evolution can be prescribed in the data file (Section 5.3.3). Similarly, the number of partial thin beams[6] which represent the pencil beam is an adjustable parameter that can be varied interactively (`CF1` in example (90)).

The 9-th parameter in Table 4.31 also requires more explanations. This parameter defines how the power of a fast ion is deposited in the plasma:

$= 0$ no averaging (deposition at the birth point),

$= 1$ averaging with a finite orbit width,

$= 2$ averaging with zero orbit width.

[6] The parameter No. 10 in Table 4.31 gives the number of beams in the horizontal direction, N_h. The total number of constituent thin beams is evaluated as $N_{tot} = N_h N_v$ while N_v is calculated by the code as a number of flux surfaces in the vertical aperture of each pencil beam in the footprint crossection.

The last 10 parameters in Table 4.31 define the geometry of every pencil beam. The neutral beam is characterized, first of all, by its "footprint", i.e. the cross-section of the beam with the plane including the major torus axis and orthogonal to the beam axis projection onto the midplane. In turn, the footprint represented as a rectangle is given by four parameters: the rectangle center (parameter No. 11), major radii of two vertical sides (parameters 12 and 13) and by the aspect ratio that is defined as ratio of the rectangle height to its width. Secondly, by the inclination of the beam axis with respect to the midplane (parameter No. 14), and, thirdly,[7] by the power distribution across the beam axis. The power distribution is described by two Fortran functions, `NBFHZ` and `NBFRY`, in the vertical and in the horizontal cross-sections, respectively. Each function has two free parameters which are the first four parameters in the last line of example (90) (or parameters 16, 17, 18, 19 in Table 4.31). Both functions are included in the file `AWD/sbr/nbuser.f` and can be arbitrarily modified by the user to prescribe any power distribution required.

The NBI configuration file can either be created by the user from scratch or adjusted by editing one of the files of this type provided with the code. Another option is interactive creating/modification during the Astra run. The code enters in the interactive regime whenever

– the NBI configuration file does not exist,
– the parameter `CNB1` takes a negative value,
– the parameter `CNB1` changes its value during the run.

In all cases, a dialog window pops up and the user can adjust all input parameters for all beam sources. As before, assume that the input data file is `aug10000`, (the full name of the file being `AWD/exp/aug10000`). If neither `aug10000.nbi` nor `aug.nbi` exist in the directory `AWD/exp/` then a new file `AWD/exp/aug10000.nbi` will be created on exit from the dialog window. The new (or modified) data file will then be used in the subsequent simulations. The total number of sources is defined as the absolute value |`CNB1`|. Note also that if the actual number of records in the file is larger than expected due to |`CNB1`| then the excessive records are ignored. The opposite case and `CNB1` $= 0$ are treated as an error.

[7] In the current version of the code, the beam divergence is not taken into account.

The NBI routine provides the following output.

Table 4.32. Output from `NBI` routine

Parameter	Units	Description
`CUBM`	$[\mathrm{MA\,m^{-2}}]$	NB driven current
`CUFI`	$[\mathrm{MA\,m^{-2}}]$	Current of the fast ions
`NIBM`	$[10^{19}\mathrm{m^{-3}}]$	Density of the suprathermal ions
`NNBM1`	$[10^{19}\mathrm{m^{-3}}]$	Density of neutrals with the full energy `EBEAM`
`NNBM2`	$[10^{19}\mathrm{m^{-3}}]$	Density of neutrals with the half energy `EBEAM`/2
`NNBM3`	$[10^{19}\mathrm{m^{-3}}]$	Density of neutrals with the energy `EBEAM`/3
`PBEAM`	$[\mathrm{MWm^{-3}}]$	Total beam power absorbed
`PBLON`	$[10^{19}\mathrm{keVm^{-3}}]$	Longitudinal pressure due to fast ions
`PBPER`	$[10^{19}\mathrm{keVm^{-3}}]$	Perpendicular pressure due to fast ions
`PEBM`	$[\mathrm{MWm^{-3}}]$	Beam power absorbed by electrons
`PIBM`	$[\mathrm{MWm^{-3}}]$	Beam power absorbed by ions
`SCUBM`	$[\mathrm{kg\ s^{-2}m^{-2}}]$	Source of the toroidal momentum
`SNEBM`	$[10^{19}\mathrm{m^{-3}s^{-1}}]$	Source of electrons due to NBI
`SNIBM1`	$[10^{19}\mathrm{m^{-3}s^{-1}}]$	Birth rate for ions with the energy `EBEAM`
`SNIBM2`	$[10^{19}\mathrm{m^{-3}s^{-1}}]$	Birth rate for ions with the energy `EBEAM` /2
`SNIBM3`	$[10^{19}\mathrm{m^{-3}s^{-1}}]$	Birth rate for ions with the energy `EBEAM` /3
`SNNBM`	$[10^{19}\mathrm{m^{-3}s^{-1}}]$	Source of thermal neutrals

More detailed description of the algorithm and a way of calculating all quantities in Table 4.32 are described in [7].

5 User's guide

5.1 General concept

As discussed in Section 3 the set of transport equations for a tokamak varies depending on a problem to be solved. The system Eq. (59) can be either shortened or appended with additional equations. Right hand sides of the system can also have quite different forms for different applications. However, the uncertainty in the transport matrix Eq. (60) is incomparably higher. It would not be an overstatement to say that there are hundreds of choices for transport coefficients in the matrix Eq. (60). Therefore, to be efficient the transport simulation of a tokamak requires a convenient and flexible tool for creating numerical codes based on different transport models. The programming system Astra was designed for this purpose.

Proceeding from this requirement the Astra code considers the set Eqs. (59)-(60) as a pattern which can be filled in or left empty according to user's request. Astra is not a numerical program in the conventional sense. A pre-determined program exists for the service parts of the Astra system only. The transport kernel of the code is created every time by a code builder and includes only those equations, transport coefficients, sources, sinks and additional modules that were explicitly requested by the user in the description of transport task. It does not mean, however, that any change requires re-building the program. On the contrary, the code is designed for an interactive work and provides a lot of techniques for computation control and correction. In particular, the interactive mode of operation is convenient for an interpretative modeling or for analysis of experimental data.

Developing the concept of the code we tried to meet two contradictory requirements. The first one is minimizing user's efforts for learning the code and for starting any reasonable simulation. This implies that the user has to take only those decisions and supply only this information which cannot be obtained elsewhere. For instance, the type of device (tokamak or stellarator), its geometrical size, magnetic field, plasma current have to be specified by the user. It is also exclusively left at user's discretion to select a way of data treatment and data presentation. The second requirement is to avoid any hidden assumptions which can result in unexpected or difficult for understanding results and thus make usage of the code even more complicated if possible at all. As a compromise, we adopted a number of default transformations which are in effect only if no explicit request is provided by the user.

The following questions should be answered before creating any transport code:

1. What subset of Eqs. (40), (59), (75) is to be solved?

2. What boundary conditions should be imposed?

3. What quantities and in which form should appear as an output?

4. What parameters (major and minor radii of a torus, magnetic field, plasma current, etc.) and what initial conditions should be used?

In the Astra system the fourth question is separated from the others. It is reasonable because the first three items specify the simulation model, i.e. a set of equations to be solved and a form of presentation of the results, whereas the forth specifies where to apply the simulation model. By providing the answers to the first three questions the user completely defines how he wants to perform a simulation. Then Astra automatically builds the source code for the specified transport problem, compiles and loads it. We will call the generated program code a transport model or simply a model.

The answer to the fourth question fills a model with machine and shot specific contents. Although the machine and plasma parameters can also be included in a model it is reasonable not to do this. Then the same simulation model can be applied to different devices and different parameters that are read from a data file when the program starts execution. In addition to the global device parameters and the initial data for transport simulation, this file can also contain the full available information of plasma parameters, e.g. the time evolution of plasma current, plasma density n_e, the electron T_e and ion T_i temperatures, plasma position, Z_{eff}, impurity densities, radiated power, auxiliary heating power, driven current and many others. Essentially, it stores the whole set of experimental data for a specific shot. In addition, database shot record may include results of calculation from other codes thus providing implicit link between Astra and other codes, when external calculations may be made in advance. All this informations along with specifying discharge parameters can be used in the interpretative transport modeling for comparison with the simulation results in order to refine and develop the transport model. In the analysis mode, the experimental information is used for different kinds of data analysis and processing.

Thus, **the model** is a transport code created for the system of transport equations Eq.(59) with specified transport coefficients Eq.(60), initial and boundary conditions. It can include also a variety of other equations (e.g. different current drive and heating schemes, cold neutrals, SOL model, impurity diffusion), formulae and all possible combinations and processing the obtained data. In addition to the transport analysis, the Astra system provides a powerful apparatus for data processing and presentation.

The data file is a set of time and radially dependent data related to a specific shot in a given device and containing available information about the shot. The data from the record are read by a model and can be used as the initial and boundary conditions, for comparison of stored experimental data with the results of simulation, for different treatments of experimental data and all other types of data analysis.

A number of different models and data records exist in the Astra system independently. The rules for the creation of a model are described in Section 5.2 The data file format is described in Section 5.4.

5.2 Setting a model

5.2.1 Model file

The main step in creation of a simulation code within the Astra system is a formulation of the transport problem to be solved and selection of the form for presentation of the results. This formulation is put into a special file where all requests for transport simulation, data manipulation and result presentation are specified using Astra model description language. After the code starts the file is submitted to the Astra compiler which processes all the instructions contained in the file and creates first a Fortran/C program and then the executable transport code. In most cases, these steps remain hidden for the Astra user who deals exclusively with the model or prototype file. In what follows, we will not make a distinction between the transport code as such and the prototype file. Moreover, for brevity the prototype file will be also called **a model file** or just a model.

Each model file contains a comprehensive and compact (typical length is 3 kB) information about a corresponding transport code (approximate length of all source files 3 MB, binary executable 10 MB) created by the Astra system. Because the model file is a unique image of the transport code, only these files need to be retained in order to repeat any simulation when such a necessity arises. All later Astra versions maintain compatibility with model files of preceding versions. The user can have simultaneously a number of model files which are all stored in the directory `AWD/equ/`.[8] Few syntax rules are adopted in order to present user’s request for a transport code in the most clear, visible and concise form.

5.2.2 General rules

A model file is a text (ASCII) file. Although a length of this file is restricted the restriction practically never occurs. All instructions in a model are case insensitive. Maximal length of each line in the file is 132 symbols including spaces and tabulations which, however, are ignored and can be used to make a file more readable. The last line in the file should have a symbol `<CR>` (carriage return) on the end otherwise it is ignored during processing.

Now we define a set of symbols, which have a special meaning and serve to describe Astra instructions or commands. Those are

$$\texttt{:} \qquad \texttt{=} \qquad \backslash \qquad \texttt{\_} \qquad \texttt{;} \qquad \texttt{!} \tag{91}$$

Every line in a model is considered as a command if anyone of the four first symbols in Eq. (91), i.e. `’:’, ’=’, ’\’, ’_’` is present in this line. Several commands can be joined in one line if they are separated from one another with a semicolon `’;’`.

All other lines are ignored by the Astra compiler and, therefore, they can be used as commentaries. A command line can also be converted to a comment if the first symbol in it is an exclamation sign `’!’`. In other words, the sign `’!’` switches off all special symbols on the right of itself until the end of line (`<CR>`) or the nearest `’;’`. Additionally, if the 1-st symbol in a line is `’!’` then the whole line is ignored.

Example:

```
              Boundary condition for the density
!NEB=NEX;  QNB=100;  QNNB=30;  This line is switched off by the leading ’!’
␣! NEB=3;  QNB=500;  !QNNB=700;  Here the second instruction is effective
```

This example contains only one command: `QNB=500`. All other instructions are suppressed.

[8] See footnote in the page 49.

Order of operators. When creating an executable code the Astra compiler analyzes each command in a model file and then creates a sequence of corresponding Fortran operators. (Sometimes, several operators correspond to one command in a model file.) During a compilation all commands are split in classes. The first class consists of assignment statements (Section 5.2.3) which have a scalar variable on the left hand side. The second class includes assignments of vector variables. Finally, the third class is composed of external modules (plug-in modules, Section 5.2.4) and transport equations (Section 5.2.5). These three classes are treated after one another. With a few exceptions, an order of operators in each class is retained the same as in the original model file.

5.2.3 Arithmetic expressions

Arithmetic expressions in the Astra code basically obey the Fortran syntax. Operands of an expression can be constants, scalar and vector variables, Astra formulae and functions. All arithmetic operations and Fortran intrinsic functions can be used in the expressions.

As in Fortran, arithmetic expressions appear mainly in **assignment statements** i.e. those command lines which include the sign `'='`. On the left of assignment statements can be the Astra scalars and vectors only. If the left hand side of the assignment statement is a scalar than the right hand side of it must be a scalar otherwise an error is reported. If the left hand side is a vector than the right hand side can be either a scalar or a vector.

Examples:

```
       Power source of 1 MW with the Gaussian distribution in radius
CAR1=exp(((RHO/ROC-0.5)/0.1)**2);        CF1=VINT(CAR1B);
                       and a rectangular modulation in time (100Hz)
QECR=anint(.1*sin(100*GP2*TIME)+.5);  PEX=QECR*CAR1/CF1;
```

This example illustrates using expressions in a model. Argument of the exponent in the first command shows that constants, scalars and vectors can be used on the right of assignment statement. The result is a vector. The second command calculates the scalar `CF1` so that `CAR1/CF1` is normalized to 1 MW. The third command uses two intrinsic Fortran functions and, finally, the forth one assigns the vector variable `PEX` that later can be used as heating source in a transport equation.

The next example shows that predefined Astra expressions can be used in a model to construct quite complicated quantities.

```
          Simplified critical gradient model
CAR1=RTOR*grad(TI)/TI-RLTCR;       CAR1=sqrt(step(CAR1)*CAR1);
CAR1=CAR1-CV1*ROTSH/GITGO;         XI=HNCHI+CHE1*step(CAR1)*CAR1;
```

Here the following model for the ion heat conductivity is implemented

$$\eta_i = \frac{R_0 T_i'}{T_i}, \qquad \gamma^{ITG} = \max\left(0, \gamma_0\sqrt{\eta_i - \eta_{i,cr}}\right), \qquad \chi_i^{ITG} = \frac{\chi_0}{\gamma_0}\max\left(0, \gamma^{ITG} - \omega_{E\times B}\right),$$

$$\chi_i = \chi_i^{PS} + \chi_i^{neo} + \chi_i^{ITG}$$

and the following quantities from the Astra library (Astra formulae) are involved

RLTCR $\eta_{i,cr} = \left(\frac{R_0 \nabla T_i}{T_i}\right)_{cr}$, critical ion temperature gradient (ITG),

ROTSH $\omega_{E\times B} = \frac{B_p}{B_0}\left(\frac{E_\rho}{B_p}\right)'$, rotational shear,

GITGO $= \gamma_0$ is related to the linear increment of the ITG instability γ^{ITG} ,

HNCHI $\chi_i^{PS} + \chi_i^{neo}$ is the neoclassical heat conductivity [19].

Two free parameters are included in the model: the stiffness $\chi_0 =$ CHE1 that characterizes transport properties of the nonlinear phase of the instability and the parameter CV1 which enables switching on and off transport (or instability) suppression due to the plasma rotation.

To summarize the interpretation rules for Astra expressions we say that a type of the result is determined by the left hand side of the assignment statement. An assignment of a scalar to a vector variable is not considered as an error so that the vector on the left retaining its type is viewed as radially independent. If a vector is attempted to be assigned to a scalar then a warning message is reported although the error is not considered as severe.

Vector variables at given radii. As discussed, the Astra compiler discriminates different types of variables and allows using them in common context without explicit indication of the variable type. However, sometimes a need arises to evaluate vector quantities at given radial positions. This can be done in the conventional way. For instance, `TE(0)` will be understood as the electron temperature at the magnetic axis $T_e(0)$, `TE(ABC)` as the electron temperature at the plasma edge and `TE(0.3)` as the electron temperature at the radius $a = 0.3$ m. More complicated constructions are also allowed. For instance, `TE(AFVAL(MU,0.5))` gives a value of T_e at the resonance surface $q = 2$.

A reference to the coordinate a is selected as most convenient for associating it with the laboratory reference frame because this coordinate can be thought of as the minor radius in the mid-plane (see Eq. (88)). However, Astra vectors can be calculated also as functions of other flux coordinates using auxiliary functions described in Section 4.8.3. So `NI(AFR(0.3))` gives the ion density $n_i(\rho = 0.3\mathrm{m})$, and `NI(AFX(0.3))` at $\rho_N = \rho/\rho_B = 0.3$. One can also use `NI(0.3*ABC)` to obtain n_i at $a_N = a/a_B = 0.3$ or, what is the same, at $a = 0.3\,a_B = 0.3 * \mathtt{ABC}$.

Shortcuts are provided for vector quantities evaluated at the magnetic axis and at the plasma edge. Namely, adding `C` to a vector variable name one obtains its value at the centre and adding `B` at the edge [9]. It means that `TEC` stands for `TE(0)`, `MUC` for `MU(0)`, `TEB` for $\mathtt{TE}|_{a=a_B} = \mathtt{TE}|_{\rho=\rho_B} = \mathtt{TE(ROC)}$, `MUB` for `MU(ROC)` and so on.

Astra expressions which are implemented as functions (but not as formulae) can also be calculated at given radii according to the same syntax rules. Thus `NEAVB` yields the electron density averaged over the whole plasma volume while `NECHC` gives the line averaged density along the central chord. Finally, one can use two Astra functions `ATR` and `ATX` (see Section 4.8.3) in order to evaluate Astra vectors as functions of the variables ρ and ρ_N, respectively.

5.2.4 Connection of plug-in modules

The next subject to be discussed is a connection of auxiliary packages to the Astra code. On one hand, development of external packages provides the most powerful way to include various physical processes in the code. In this way the user can also get an access to almost any part of the code and make nearly any changes and additions to the resulting simulation model. On the other hand, it is completely at user's discretion to trace possible consequences of such an interference in the code flow. Many plug-in modules are already available and were discussed in the previous section. Others can be developed using one or few of the existing modules as templates.

Front end of any plug-in module is a subroutine called by Astra. Their inclusion in a

[9] One exception of this rule is discussed at the end of Section 5.2.5

simulation model is implemented through the instruction of the type[10].

$$\texttt{Name}(\textit{Argument list})\texttt{:\{}\Delta_t : \texttt{t}_{\texttt{start}} : \texttt{t}_{\texttt{end}}\texttt{:Letter\}} \qquad (92)$$

Only the first colon ':' is required in this line all others can be omitted together with the corresponding control parameters. However, if the i-th $(i = 1, 2, 3, 4)$ parameter is present it should be preceded with i colons. This command results in calling a subroutine with the name "Name". An argument list of the subroutine (if present) is directly converted into a Fortran line without any transformations which are normally performed with the Astra expressions. Therefore, the actual arguments can be constants, variable names, or expressions including constants and scalar variables only. For instance, the instruction

```
    NEUT:;
```

corresponds to the Fortran statement

```
    call NEUT
```

which calls the subroutine NEUT calculating the neutral density NN and temperature TN (see Section 4.9.4).

In the last example, the subroutine will be called at each time step of the simulation run. This can be changed by involving control parameters in the command line Eq. (92). They have the following meaning

Parameter	**Significance**	**Default value**
Δ_t	Time interval between calls	0
$\texttt{t}_{\texttt{start}}$	First time of calling	$-\infty$
$\texttt{t}_{\texttt{end}}$	Last time of calling	∞
Letter	Control key	None

If Δ_t is not specified a subroutine is called at each time step. Parameters $\texttt{t}_{\texttt{start}}$ and $\texttt{t}_{\texttt{end}}$ define start and end time when the module is activated. No calls are made beyond the specified time interval $\texttt{t}_{\texttt{start}} \leq t \leq \texttt{t}_{\texttt{end}}$. If one of these parameters is not specified then corresponding conditions do not apply. Parameter Letter is used to provide manual activation of the module in the interactive mode of operation. So, the instruction

```
    NEUT:0.1:::N;
```

results in calling NEUT every 100 ms and additionally each time when the key combination < Ctrl > N is pressed. The control key Letter can acquire any capital alphabetic value except for 'M'.

[10] Here and below the braces {...} in a command description enclose an optional part of the command.

5.2.5 Transport equations

As described in Section 4, the Astra code can solve up to seven transport equations for plasma density, electron and ion temperatures, current density and three unspecified functions. Each equation can be involved by an explicit request only. Alternatively, a rule can be provided for calculating any of those quantities as functions of radius and time. A type of evolution of a variable is specified by the instruction of the form

```
Name:{Type}{[Tag,Expr1]}{:Expr2}
```
(93)

where every term after the first `':'` can be omitted.

`Name` is one of the variables on the list: `NE`, `TE`, `TI`, `CU`, `F1`, `F2`, `F3`.

`Type` is one of the two qualifiers `Equation` or `Assignment` (each can be truncated up to one letter), default value is `Type=Equation`.

`[Tag,Expr1]` Not applicable if `Name=CU`. Otherwise, `Expr1` defines the right boundary of the interval where the transport equation will be solved. An integer `Tag` specifies a type of the radial coordinate defined by the expression `Expr1`

`Tag=0` dimensional coordinate a (may be omitted),
`Tag=1` dimensionless coordinate a/a_B,
`Tag=2` dimensionless coordinate $\rho_N = \rho/\rho_B$.

`Expr2` is another expression that can be used if `Name` is one of `TE` or `TI` only. The expression then determines γ_e or γ_i in Eq. (82). By default, $\gamma_e = \gamma_i = 0$.

If the qualifier `Type` is omitted then it is assumed to take the value `Type=Equation`. In case `Type=Assignment`, the rest of the command string is ignored. The new feature introduced in the version 5.3 is a possibility to set the different right boundaries for different transport equations. However, some restrictions are imposed.

1. If the right boundary is not given then it is set to $\rho_B =$ `ROC` (or $a_B =$ `ABC`). This maintains compatibility with the previous versions.
2. For the poloidal flux (i.e. for `Name=CU`) the right hand boundary cannot be shifted and is always ρ_B.
3. For all other equations it cannot exceed the value ρ_B.

4. Unlike the outermost boundary position, ρ_B, which can be located arbitrary (continuously varying) on the difference grid, all "internal" boundaries are attributed to the nearest grid point in ρ (jumping).

When a model is converted into a Fortran code, the transport equations and the plug-in modules are put in the same order as they are requested in the model. Exception is made for the subroutines MIXINT and TSCTRL which are always placed after all transport equations. Although, in most cases, the order of transport equations is not essential, sometimes, one has to take this convention into account.

The following example shows setting transport equation for the plasma density

```
NEUT:;
DN=1+FPR**2;             CN=-DN*FX;               SNN=SNNEU          (94)
NE:Equation;             NE=10*sqrt(FPR)+3;
```

The first line calls the subroutine NEUT at each time step. The second line defines the diffusion coefficient DN as a parabolic function, the pinch velocity CN as a product of this parabola and a linear function (the minus sign corresponds to the inward pinch). The source term SNN describes the ionization and recombination processes making use of the formula SNNEU. We remind that the same source term could be defined as SN=SNNEU*NE. The only difference between these two representations is in the numerical implementation, implicit or explicit, respectively. If there is no particular requirement then the implicit approximation is preferable. Finally, the third line says that the diffusion equation for NE should be solved with the initial condition defined by the last command. The boundary condition is not set here explicitly and, therefore, the value assigned by the initial condition at the point ρ_B will be used. Effectively, it means that the boundary condition NEB=3 is applied. Other options for the boundary condition with a prescribed particle flux or more complicated are shown in the example in Section 5.2.2.

Exactly the same result can be obtained by means of any auxiliary transport equation Eq. (85).

```
NEUT:;
DF1=1+FPR**2;            VF1=-DF1*FX;             SFF1=SNNEU
F1:Equation;             F1=10*sqrt(FPR)+3;
```

Here a diffusion equation is constructed for the dummy variable `F1` which in all details reproduces the diffusion equation for `NE` created by the model Eq. (94). This example shows how the transport equations for the variables `F1, F2, F3` can be used for a description of multi-species plasma.

Consider now five different models

Model 1	Model 2	Model 3	Model 4	Model 5
	TE:As	TE=TEX	TE=TEX	TE=TEX
			TE:As	TE:Eq

(95)

and assume that a time evolving electron temperature `TEX` is stored in a data file which will be used together with each of these models. The model 1 is empty. Nevertheless, nonempty executable code will be created for this model which makes a number of assignments including those described in Section 5.3.1. In particular, the electron temperature will be defined as $\mathtt{TE}(\rho) = \mathtt{TEX}(\rho, t_{start})$ where t_{start} is the initial time of simulation. The model 2 produces the same effect. The model 3 makes the assignment $\mathtt{TE}(\rho, t) = \mathtt{TEX}(\rho, t)$ at each time so that the complete time evolution of `TEX` will be stored in `TE`. A result of the model 4 is exactly the same as for the model 3. In the model 5, solving the transport equation is requested, however, neither transport coefficients nor the power sources are defined. For this model, Astra will build a code where the first line `TE=TEX` is interpreted as the initial condition $\mathtt{TE}(\rho) = \mathtt{TEX}(\rho, t_{start})$ and the equation $\partial T_e/\partial t = 0$ will be solved (we assume here that the magnetic field B_0, the plasma density n_e and the plasma size do not change in time). Thus, a result of the model 5 will coincide with that of the models 1 and 2.

One can conclude from this example, that the instruction of the type `Name:As` is inessential. It is really so in most cases, although a formal difference can appear between model 3 and model 4 if other commands of the type (93) are present. Namely, with explicit instructions of the type `Name:As`, order of assignment operators in a code is the same as in a model. If all these instructions are implicit then the assignment operators will follow one another as in the sequence `NE, TE, TI, CU, F1, F2, F3`. The logic discussed implies that if the `Type` of evolution is set to `Assignment` then an assignment expression for this variable is expected. Although an absence of such an assignment is not considered as an error, a warning message is printed for models like the model 2. Another case when the explicit request `CU:A` is necessary is discussed below in the example (96).

The format of equation command line is modified in the most recent version 5.3 in comparison to the previous versions of the code. The old form was

```
Name:Type{:Expr2}
```

which can be compared with (93). The qualifier Type is here obligatory. In all other respects the model generated by the old-type equation command line fully coincides with that generated by the new one. However, there is a difference in treatment when an equation command line is absent. Namely, in old versions, an assignment of time evolution to all main variables NE,TE,TI,... is activated only if the equation command line with Type=Assignment is present. Coming back to the example (95) it means that the models 1, 2, 3 and 5 give the same result while only the model 4 gives the time evolving temperature TE $=$ TEX(ρ, t).

The next example sets a transport equation for the poloidal flux

```
             Current diffusion equation
CU:;                                       !Equation
CC=CCHR;                                   !Conductivity [20]
DC=DCHR;     HC=HCHR;     XC=XCHR;         !Bootstrap current [21]
CD=CUX;                                    !Pre-calculated driven current
CU=1;        MU=.2;                        !Initial distribution
UEXT=0.2;    IPL=0.2+0.8*FRAMP(0.,1.);     !Boundary condition
```

The first line in this model requests the current diffusion equation to be solved by the code. The second line specifies the conductivity [20] by making use of the formula CCHR. The third line defines the bootstrap current in a similar way. The driven current density is supposed to be stored in a data file under the name CUX. The assignment command CU=1 is understood as the initial condition. It sets the flat current distribution normalized to the total current IPL. The second command in this line contradicts to the first one. Therefore, according to the diagram in Section 4.3.2 it is ignored. The last line includes two contradicting boundary conditions for the poloidal flux. According to the priority rule of Section 4.4.2 the second one for the total plasma current IPL is applied and no notice of UEXT=0.2 is taken here.

We discuss now different ways of setting the current density profile in case when the current diffusion equation is switched off.

```
Model 1  |  Model 2   |  Model 3   |  Model 4
CU:AS    |  CU=FPR    |  CU:As     |  CU:A                 (96)
CU=FPR   |            |  MU=MUX    |
```

The first model prescribes parabolic current density and is fully equivalent to the second model. In other words, the command CU:AS can be omitted without changing the result.

The same is also applicable to the third model. This model prescribes a rotational transform profile as stored in a data file. All these three models can also describe time dependent profiles. The last model 4 gives the current density profile which at every time step is defined by the steady state condition Eq. (69) as if the current penetration time (skin time) is much shorter than all the other characteristic times. In this case, the command `CU:A` cannot be omitted.

We remind (see Section 4.3.2) that there is one essential difference in a way of defining initial current density profile between Model 1 and Model 3. In both cases, the initial setting must be consistent with the total plasma current. But, in the first case, the entire current density profile is adjusted, in the second case, the boundary value of $\mu(\rho_B)$ at the edge only. In other words, the current density in Model 1 is continuous, in Model 2, it has a surface skin current.

Initial conditions. As follows from the examples considered above, assignment statements are used for setting initial conditions. This implies that an assignment instruction is treated differently depending on the context. Namely, if the left hand side of an instruction is one of main variables $\{\mathtt{NE}, \mathtt{TE}, \mathtt{TI}, \mathtt{CU}, \mathtt{F1}, \mathtt{F2}, \mathtt{F3}\}$ and the corresponding equation is absent then the assignment is processed as all other assignment statements with a vector variable on the left hand side.

Otherwise, if the corresponding equation is present then the time evolution of the quantity is defined by the equation while the assignment statement is executed only once at the start of simulation. Moreover, all main variables and few others (see Section 5.3.1) are defined even if they do not appear in any assignment statement in a model. In the last-mentioned case, data stored in the start file are used. If no appropriate data are stored there and no assignment is made then the code results are not relevant. In all cases, when an implicit assignment is involved it is applied just once for the initial time only.

In addition, the initial conditions can be set by a user developed subroutine. In this case, one should beware that an explicit definition in a model has higher priority than that in a subroutine. In other words, if two contradicting settings in a subroutine and in a model are present then the latter will be used by the code.

Boundary conditions. A similar rule is applied to the boundary conditions. The boundary condition can be set by an explicit assignment command or by an implicit (Section 4.4) definition. As in the case of initial conditions, an explicit assignment can be time dependent while an implicit one is always time independent and assigns a boundary value at the initial time of the simulation only. Once assigned this value does not change afterwards.

This property can be used for more complicated boundary conditions that cannot be expressed by a simple formula. Suppose that a subroutine SOLAY calculates the electron density at the edge NE(NA1) taking the particle flux at the same point QN(NA1) as an input. This can be implemented in a model as the boundary condition

```
NEUT:;               SOLAY(QN(NA1),NE(NA1)):
DN=.1;               CN=-VP*VRHH;          SNN=SNNEU         (97)
NE:;                 NE=NEX;
```

This model describes a time evolution of the plasma density profile assuming a constant diffusion and the neoclassical Ware pinch. Note that the argument list of a subroutine is not processed by the Astra compiler. Therefore, the abbreviated notations QNB and NEB can not be used as arguments of the subroutine. If an additional instruction like NEB=NEXB is present in the model (97) then it will override the edge density defined in the subroutine SOLAY.

Shifted boundary conditions. Additional possibility for setting boundary conditions is provided by the term in square brackets in (93). It enables solving different transport equations on different radial intervals. This option can be useful when the core and periphery transport have different physical nature and shifting the boundary condition inside allows to exclude the periphery transport out of consideration. It can also be useful when the plasma parameters are not known at the plasma edge and the boundary condition is set at the outermost measured point.

According to the syntax of the command (93) the all three instructions

```
TI:[0.8*ABC]        |   TI:[1,0.8]      |   TI:[2,XFAN(0.8)]
```

are equivalent and require solving the heat conductivity equation for T_i on the interval $0 < a < 0.8a_B = 0.8 * \texttt{ABC}$. Similarly, the instructions

```
TI:[2,0.8]          |   TI:[AFX(0.8)]   |   TI:[1,AFX(0.8)/ABC]
```

require solving the same equation on the interval $0 < \rho < 0.8\rho_B$.

The next model

```
IFSPPL(CAR1,CAR2,CAR3,CAR4,CAR5,CAR6):
XI=HNCHI+CAR5;               PI=PEICL+PIX                    (98)
TI:Eq[1,0.9];                TI=TIX
```

produces a transport code solving the heat conductivity equation for T_i on the interval $0 \leq a \leq 0.9a_B$. Here the ion heat conductivity $\chi_i =$ XI is defined as a sum of neoclassical and anomalous terms where the latter (along with many other quantities) is calculated by the subroutine IFSPPL. The heating source PI includes the electron-ion heat exchange PEICL and the auxiliary heating PIX defined in a data file.

The command TI=TIX bears here a double meaning. Firstly, it defines the ion temperature $T_i(a, t_0) = T_{i,exp}(a, t_0)$ at the initial time t_0 on the whole radial interval $0 \leq a \leq a_B$. The subsequent time evolution of $T_i(a, t > t_0)$ on the partial interval $0 < a < 0.9a_B$ is defined by the heat conductivity equation. Thus, on the segment $0 \leq a \leq 0.9a_B$, the command TI=TIX provides the initial and the boundary condition[11] for the corresponding transport equation. Secondly, it defines the time evolution of the ion temperature $T_i(a, t)$ on the interval $0.9a_B < a \leq a_B$ for $t > t_0$ that is not covered by the transport equation. Therefore, the command TI:[1,0.9] in the last line of the model (98) can be thought of as if it combines the definition of Type=Equation for the interval $0 < a < 0.9a_B$ and of Type=Assignment outside it.

Similarly to the case of non-shifted boundary conditions, the boundary value $T_i(0.9a_b)$ implicitly imposed by the command TI=... is time independent. In order to set a time dependent boundary condition one has either to use an explicit assignment statement in any of the forms

```
TIB=...          |     QIB=...        |     QITB=...
```

or to implement an implicit boundary setting with an external subroutine similar to that of the example (97).

It is significant to note that, in the particular case of the shifted boundary conditions, the abbreviation TIB is interpreted distinctively depending of the context. Namely, whenever

[11] Like in the case of non-shifted boundary, this assignment is made only once at the start time $t = t_0$. Moreover, it applies only when no explicit boundary setting is provided.

`TIB` appears on the right hand side of any expression, it is replaced by the Astra code compiler with `TI(NA1)` which is equivalent to $T_i(a_B)$. However, if (and only if) any of the notations `NEB, QNB, QNNB, TEB` and so on[12] stands on the left hand side of an assignment command then this command is interpreted as the boundary condition assignment and it is attributed to the boundary point declared by a corresponding equation instruction of the type (93).

For instance, if the command `TIB=TIXB` is used in the model (98) it will result in the Fortran statement `TI(NA1I)=TIX(NA1)`, where the array index `NA1I` points to the grid node nearest to $0.9\,a_B$ while the index `NA1` to the edge grid point $a_B =$ `ABC`. Most probably, it is not the result expected by the user. The correct assignment can be implemented as `TIB=TIX(0.9*ABC)`.

5.2.6 Radial output

First of all we note that there is no default output in Astra and therefore quantities of interest must be explicitly specified in the model definition. Request for a radial output has the form

`Name\{Expression}{\\Xdata}{\Scale}` (99)

Here `Name` specifies a legend (a string with a length up to 4 characters) associated with the radial profile given by any regular arithmetic `Expression`. Every part of this command shown in braces is optional and can be omitted, the group preceded by `\\` can either be omitted or repeated several times.

Up to 96 commands of the type (99) can be present in a model and, consequently, up to 96 radial profiles can be plotted (or stored) in a simulation. Each radial profile defined by the corresponding `Expression` together with its `Name` and `Scale` will appear in a graphic window when the code starts execution. The code provides a few different graphic modes which allow combination of several curves in one box and several boxes in one window. The curves are sequentially placed in boxes in the same order as the curve descriptions (99) appear in the model. If a number of curves is larger than the number of boxes then the next curve is again put in the first box, then in the second and so on until all boxes in the current window acquire the maximal for this mode number of curves. Then the next

[12] The complete list is given in Table 4.8.

(hidden) window is filled and the process continues until the list of the curves submitted to output is exhausted.

In other words, one can say that each radial profile defined with (99) receives its sequence number which is an ordinal number of the output command as they appear in the model. The curves are placed in graphic windows according to their sequence numbers. Because the position of the curves on the display it is essential for the convenience of viewing results, it is important to pay attention to the sequence of the output commands. Nevertheless, the user can change the sequence numbers and other curve attributes interactively during the code execution. More detailed description of this feature will be given in Section 5.5.3.

The parameter `Scale` defines the scale for plotting the curve given by `Expression`. If `Scale` is zero or absent then the scale of the output profile is automatically selected by the code. If `Scale` is given as a positive number then this number specifies the actual scale for the corresponding radial profile. Negative integer value of `Scale` allows setting the same automatic scale for all profiles in the group with the same negative scale value. It is useful for representation of several different curves of the same meaning, e.g. the input powers of different heating methods.

There is one more peculiarity in setting automatic group format. Normally auto-scale is selected based on the maximum absolute value in the entire group. But the user can enforce the code to make automatic scale basing upon a single specified profile in the group. It happens if the absolute value of the negative scale coincides with the sequence number of the output. We illustrate the discussed rules for radial output with an example:

Example:

```
Te\TE\-9;        Ti\TI\-9;        j\CU\-2;          Utor\UPL\-1;
ne\NE\10;        \;               jOhm\CUOHM\-2;    iota\MU;
Texp\TEX\-9;     Tixp\TIX\-9;     jBS\CUBS\-2;      Ulng\ULON\-1;
nexp\NEX\10;     \;               sigm\CC;          q\1/MU;
```

Assume that this set of data is presented in the default radial output mode which allows up to 16 curves in a window. The window includes 8 boxes placed in two rows and four columns with two curves in a box as schematically shown in this sketch.

1 `1 & 9`	**2** `2 & 10`	**3** `3 & 11`	**4** `4 & 12`
5 `5 & 13`	**6** `6 & 14`	**7** `7 & 15`	**8** `8 & 16`

Hear the large digits in the left upper corner of each box show the box numbers and the curve sequence number are shown in ordinary type. Then the first eight profiles will be shown in red and the next eight in blue so that each box will contain one red and one blue curve. Consequently, the first and the third lines in the example enter in the first row of boxes while the second and the forth lines occupy the second row. The empty instructions `\;` are used in this example in order to skip the sixth box and put all quantities related to the current density one over or under another for a more convenient comparison.

Thus the first box contains calculated and experimentally measured electron temperatures, `TE` and `TEX`, respectively. The former is labeled on the plot as `Te` and the latter as `Texp`. Both will have the same scale defined by the data `TEX` (sequence number 9). Similarly, the next box contains two ion temperatures with the same scale. The third box includes the total current density `CU` and the bootstrap current density `CUBS` each with a common scale. The last box in the first row encloses the toroidal `UPL` and the longitudinal `ULON` loop voltages in the same scale defined by the largest of the two. The fifth box shows two density profiles with the prescribed scale. The next box (No. 6) is empty. The seventh box shows the Ohmic current density and conductivity profiles (which must be proportional in the steady state). Finally, the last in this mode, eighth box presents the rotational transform `MU` and the safety factor.

Presentation of experimental data. So far we discussed the output of radial profiles presented as curves. In the example above, even so called "experimental" data as `TEX` or `NEX` are smoothed and transferred to the fine transport grid with a typical size of 100 nodes. Transferring input experimental data to a radial grid of the transport code is a problem which has no universal solution. Although the Astra code provides a way of doing this transfer it could be unsatisfactory in some cases, for instance, it destroys a significant information while smoothing data or, in opposite, add artificial features while extrapolation. In order to provide a way of visual control of data treatment the Astra code enables displaying the original input data as the radial plots.

Such a plot can be requested by a command line of the format (99) making use of the control group `"\\"`. A quantity `Xdata` will be shown on a plot as it is stored in the input file (Section 5.4). To be interpreted properly, the control word `Xdata` must be one of the array names given in Table 5.4. For instance, the instruction

```
Tex\TEX\\TEX\10
```

plots the electron temperature `TEX`, firstly, as a smoothed curve, secondly, by dots as the original data are stored in the input file. The curve and the dots will appear in the same box with the same scale 10 keV. One can plot the dots only

```
Tex\\TEX
```

or combine several measurements in one plot

```
Tex\TEX\\TEX\\CAR1X
```

The latter command plots `TEX` as a curve together with two sets of data stored in the input file under the names `TEX` (e.g. Thomson scattering) and `CAR1X` (e.g. ECE data). The data for `TEX` and `CAR1X` will be shown by different symbols.

5.2.7 Time output

Radial and time output commands have a similar format. The difference is that the backslash symbol `"\"` is replaced by the underscore `"_"`. This format reads

```
{Name}_{Expression}{_Scale}                                   (100)
```

Similar to the radial output, any allowed arithmetic expression can be used for output. However, a natural restriction is applied that only scalar (radially independent) expressions may be used in the time output instructions. The user can prescribe an output scale, ask for a common scale for several plots or take advantage of automatic scale adjustment. Up to 96 time dependences can be processed by the current version of the code.

5.2.8 Advanced options

User defined variables. Variables used in a transport model basically belong to the Astra internal variable list which includes (a) a set of physically meaningful predefined variables and (b) a set of holder scalar and vector variables without specific meaning. It is not considered as an error if the user employs his own scalar variables in a model. Though there is no support for this variables and they are not defined in most of Astra modules except for if explicitly passed to a corresponding subroutine.

For instance, one can define a time dependent scalar quantity

```
STAIRS=FJUMP(0.1)+FJUMP(0.2)+FJUMP(0.3)+FJUMP(0.4)+FJUMP(0.5)
```

and later use it as a part of another assignment statement or as a parameter of a subroutine, however, the quantity STAIRS cannot be plotted with the standard time output instruction (100) because it is not available in the module concerned with the output. In order to overcome this limitation the user has two options. The standard and recommended way is to replace the name STAIR with one of the C-parameters (Table 4.17). Another option is to describe the variable STAIRS in a common block inside the file[13]

```
AWD/tmp/declar.usr
```

This file is included by the Fortran statement INCLUDE in all model dependent modules of the Astra kernel.

Verbatim reproduction. Processing a model file the Astra compiler transforms it into a Fortran source code according to some rules. However, sometimes these rules are too restrictive because they are invented exclusively for dealing with Astra variables. There is a possibility to override these rules, switch off all transformations and require a verbatim record into a source code. It can be done by quoting the appropriate string. For instance, the expression

```
rhNA_"RHO(NA)"
```

yields a value of the NA-th element of the radial grid array RHO, i.e. the position of the grid point adjacent to the edge one $\rho_B =$ RHO(NA1). Similarly, the instruction

```
wrk3\"WORK(j,3)"
```

allows to plot the data stored in the 3rd column of the two-dimensional array WORK.

It is not out of place to stress here that a care should be taken when using this feature. The user should beware that the Astra compiler can put the quoted string in unexpected place of the code or even being placed in a proper position the string can get in conflict with Fortran rules. Therefore, it is recommended to check if the effect of such a nonstandard operation provides the desired result.

[13] See footnote on the page 49.

In addition we add a remark concerning the two working arrays `WORK` and `WORK1`. Both arrays are described as real two-dimensional arrays and placed in a common block[14]

```
      COMMON  /WORKAR/ WORK1(NRD,2*NRD+7), WORK(NRD,2*NRD)
```

in the file `AWD/for/status.inc` so that they can be used in every module including this file. The arrays are not being used by any routine of the Astra kernel. It is supposed that the first array `WORK1` is used as a working area for plug-in subroutines without any usage outside these subroutines. Another array `WORK` may be used either as a working area or as a tool for data exchange between user's subroutine and other parts of the code. In the second case, it remains at user's discretion to trace possible conflicts between different user developed modules.

Post-compiler editing. A possibility to interfere in the pre-defined sequence of Astra operations is given by the command

```
> Astra -t
```

This suspends the code execution after the second step (see the flow diagram at page 10). At this point, the user can introduce any changes into the changeable part of the code. This part consists of intermediate Fortran source files located in the directory `AWD/tmp/`. When the appropriate adjustments are performed the code execution will be resumed with account for all recent modifications.

5.2.9 Several examples

In this section we consider several examples of building different models in the Astra system. We assume that all models discussed below are run with an appropriate input file that provides all data required by a model. Therefore, some properties of data setting are used already in this section although they are first introduced in Section 5.3.

We start with a discussion of simple plasma characteristics which can be calculated on the basis of measured plasma density and temperature profiles.

```
Wth\WTOT;     LTi\LTI;      Heff\HEEFF;   Vpol\VPSWW;
tauE\TAUE;    LTic\RLTCR;   Hiff\XIEFF;   rhos\RLS;
```

As a result of this model the following quantities will be plotted one under another (in the 1st

[14] In the standard installation `NRD` = 501.

mode, their placement on a screen is the same as in the printout above): the plasma thermal energy content (possible contribution from fast particles is not included) and the energy confinement time; inverse normalized length of the ion temperature mapped to the mid-plane, $R_0/L_{Ti} = R_0|\partial T_i/\partial a|/T_i$; and a critical value for the same quantity with respect to the ion temperature gradient instability; effective (i.e. experimentally measured) heat conductivities for the electron and ion thermal components; and, finally, the neo-classical poloidal rotation velocity and $\rho_s = \sqrt{T_e/m_i}$. All they will be plotted as radial functions.

This code will also work if some quantities needed for calculations are not defined, however, then the results can be irrelevant. For instance, if the heating power is not defined then the heat conductivities and the confinement time will be negative. In order to get appropriate values for these quantities one has to define the input heating power in any allowed way, e.g. as `PE=POH; PI=PIX`.

Another essential remark is that the radial functions plotted by this model are time independent even if the correspondent data file contains time dependent data. This happens in line with the syntax rules described in Section 5.2.2. Indeed, without the explicit definition the assignment as `TE=TEX` and similar are fulfilled only once at $t = t_0$, consequently, all quantities calculated on the basis of n_e, T_e and so on are time independent. One can view the difference running the model

```
Tex\TEX\\TEX\-9;    2\;   3\;   4\;                 5\;   6\;   7\;   8\;
Te\TE\-9;
```

Two graphs will show the quantities `TE` (in blue) and `TEX` (in red) as smoothed curves in the same scale in the upper left box. Additionally, the original data for `TEX` will be shown with red crosses. The curve for `TEX` will evolve as it is written in the data file, while `TE` will coincide with `TEX` at the first instant only.

In order to remove all differences, one has to include in the model the explicit definitions of main plasma parameters.

```
NE=NEX;   TE=TEX;   TI=TIX;       NEQUIL=41;
```

Here the command `NEQUIL=41` requires that the Astra built-in 3-moment equilibrium solver is employed for the equilibrium reconstruction.[15]

[15] We remind that, if not defined explicitly, the current density distribution is calculated from the steady

As an example of the time output consider

```
ABC_ABC;      kapp_ELONG;      delt_TRIAN;      shif_SHIFT;
Tex0_TEXC;    Tix0_TIXC;       Ipl_IPL;         Btor_BTOR;
Wth_WTOTB;    pk1_NEC/NEAVB;   pk2_NEC/NECHC;   H-89_TAUEB/(TITER+1.e-6);
q0_1/MUC_5;   qa_1/MUB_5;      qmin_QMINB_5;    qmin_1/FRMAX(MU)_5;
Xmin_XQMINB_1;    q2a_AFVAL(MU,.5)/ABC_1;       q3a_AFVAL(MU,.333)/ABC_1;
```

The first line shows the geometrical parameters of the outermost magnetic surface: the minor radius ABC, elongation ELONG, triangularity TRIAN and shift with respect to the major radius RTOR. The second line gives the central values of electron and ion temperatures then the plasma current and magnetic field. After that follow: the total plasma energy content, the density peaking factor calculated as the central value divided by the volume averaged and by the chord averaged densities, enhancement factor with respect to the ITER-89 scaling. The next two lines show different quantities characterizing the safety factor q distribution. The first of them yields the central $q(0)$ and the edge $q(a_B)$ values, then the minimum q_{min} value calculated in two different ways. Finally, the last line gives the radial positions (in the normalized coordinate $x = a/a_B$) of the points where $q(x)$ reaches its minimum value, where $q(x) = 2$ and $q(x) = 3$.

Unlike the radial output most of the time dependencies above are plotted as they are stored in the data file, i.e. if a time dependence of any Astra simple variable (e.g. ABC, IPL, BTOR, ...) is written in the data file then it will be shown as time varying also in the Astra output. This can be not the case for the derived quantities. For instance, NEC and NEAVB are calculated making use of NE, therefore, for those quantities, the same rules are applicable as for NE.

The next example shows a model which reads in the experimental data for n_e, T_e, T_i and calculates the current density evolution assuming the neoclassical conductivity and bootstrap current.

```
NE=NEX;     TE=TEX;     TI=TIX;  !  measured plasma parameters
AMAIN=2;    ZMAIN=1;             !  mass & charge of the main ion species
AIM1=12;    ZIM1=6;              !  mass & charge of the main impurity
```

state condition Eq. (47).

```
NIZ1=NE*(ZEF-1)/(ZIM1-1)/ZIM1;   !  density of the impurity
NI=NE*(ZIM1-ZEF+1)/ZIM1;         !  total density of all plasma ions
NDEUT=NI-NIZ1;                   !  density of deuterium plasma ions
NEQUIL=41;                       !  3-moment equilibrium solver
HC=HCKIM;   DC=DCKIM;   XC=XCKIM;!  Bootstrap due to Kim [22]
CU:EQ;      MU=.33*FPR+.33;      !  current density Eq & initial condition
CC=CNHR;    CD=CUBM;             !  conductivity [20]  & NB driven current
MIXINT(CV1,CF1):;                !  Kadomtsev reconnection
NBI:;         NEUT:;             !  NB injection and cold neutrals
                Radial output
Tex\TEX\\TEX;     q\1./MU\10;  jtot\CU\2;     Upl\UPL;
nex\NEX\\NEX;     cneo\CC;     jNB\CUBM\2;    shir\SHEAR;
Tix\TIX\\TIX;     iota\MU;     jOH\CUOHM\2;   ULON\ULON;
Zeff\ZEFX\\ZEFX;  cSp\CCSP;    jBS\CUBS\2;    psiN\(FP-FPC)/(FPB-FPC);
                Time output
Upl_UPLB_2;   betj_BETAJB;   bett_BETTB;     li_LINTB*RTOR/(RTOR+SHIFT)_1;
Ipl_IPL_2;    Ibs_IBSB_2;    Iohm_IOHMB_2;   Inb_IBMB_2;
```

This model can be used for simulating NBI assisted current ramp-up phase in a tokamak. Therefore, most of output quantities are related to the current density distribution. In particular, the radial output shows the neoclassical and Spitzer conductivities (6th box), the third column (3rd and 7th boxes) includes different contributions to the current density, the forth box exhibits a difference between the toroidal and longitudinal loop voltages, the shear and the normalized poloidal flux are displayed in the 8th box. The time output presents the edge loop (toroidal) voltage, poloidal and toroidal betas, internal inductance per unit length, the total plasma current as comprised of the bootstrap, Ohmic and beam driven currents.

The next example illustrates simulation of a plasma density build-up.

```
      Equilibrium, electron, ion temperatures and poloidal flux
NEQUIL=41;       TE=2*FPR**2+.1;      TI=TE/2;      CC=CCSP;
      Particle source (wall neutrals + pellets)
NEUT:;           SNN=SNNEU;             !  Gas puffing
```

```
CV1=anint(.1*sin(100*GP2*TIME)+.405);   !  Square wave-form oscillator
SN=CF3*30000*CV1*GAUSS(.7,.1);          !  Pellet evaporation profile
        Density eqn with initial and boundary conditions
NE:EQ;           NE=10*FPR;         QNNB=CF2*10000;
CAR1=CF1*(1+3*FX**2);                   !  Prescribed function
DN=CAR1*(1-XSTEP(.85))+.5*XSTEP(.85);   !  Diffusion coefficient
CN=-VP*VRHH;                            !  Neoclassical pinch
TAUMAX=.0001;                           !  Time step limitation:  < 1 ms
         Auxiliary quantities
CV16=VINT(SNTOTB);                      !  The same as QNTOTB
CV15=TIMINT(CV16);                      !  Total particle source integral
CV14=NEAVB;                             !  Volume average density <ne>
CV13=-TIMDER(CV14);                     !  d<ne>/dt
CV2=TIMINT(CV1);                        !  Total evaporation time
         Radial output
D\DN;         gn\GN;       flux\QN;        etai\LNE/LTI\100;
ne\NE;        NN\NN\1;     nue*\NUES\1;    Spuf\SNN*NE;
Vin\-CN;      V/D\-CN/DN;  I(S)\QNTOT;     Lne\LNE;
Stot\SNTOT;   TN\TN;       Ware\VP*VRHH;   Spel\SN;
        Time output
n(0)_NEC;       tauP_TAUPB_.1;      nlin_NECHC;         Gin_CV16;
n(a)_NEB;       tau_CV14/CV13_.1;   <ne>_CV14;          Gout_QNB;
<n>G_2.7*IPL/ABC**2;   NNCL_NNCL;   ntot_VOLUME*CV14;   NNWM_NNWM;
Sinp_CV15;             albe_ALBPL;  1or0_CV1;           duty_CV2/TIME;
```

Only the density diffusion equation is solved in this model. The pinch is assumed to be neoclassical while the particle diffusion is taken as a parabolic function inside $\rho < 0.85\rho_B$ (plasma core) and as a constant $0.5\ \mathrm{m}^2/\mathrm{s}$ outside this region (pedestal zone). The boundary condition for the density is given by the requirement that the total particle flux through the plasma boundary is $Q_n = V < n_e > /\tau_P = \mathtt{QNNB}{*}n_e(a)$. For ASDEX-Upgrade parameters this gives $Q_n \approx 3 \times 10^{22}\ \mathrm{s}^{-1}$.

The particle source consists of two comparable parts: gas puffing and pellet injection. The former is described in Section 4.9.4. The latter is assumed to be periodic ($1/T =$ 100 Hz) with evaporation time $\Delta t \approx 1$ ms. The pellet ablation profile is represented by a Gaussian function with the prescribed width $\Delta_\rho = 0.1\rho_B$ and the maximum position $\rho_0 = 0.7\rho_B$. The amplitude of this source (3×10^{23} s^{-1} in our example) can be evaluated as $Q_n T/\Delta t$. Note that no interaction between the wall and pellet neutrals is taken into account in this simplified model.

The radial output in this model gives only the quantities related to the particle behavior. It shows the diffusion coefficient DN, the pinch velocity CN (positive sign means outward flux), the total particles source SNTOT and the partial sources due to pellets SN and cold neutrals SNN*NE. The quantity $\nu_e^* =$ NUES which characterize collisions is shown together with the Ware pinch velocity VP*VRHH. Also of interest could be a comparison of the two quantities QN, collisional particle flux, and QNTOT, ionization particle source. In the steady state, this two quantities must coincide. In this simulation, they are always strongly different.

The time output shows central, edge, volume and line average densities. The particle confinement time is calculated in two different ways. The Greenwald limiting density is shown under the name <n>G. The two components of incoming cold neutrals are characterized by the parameters NNCL and NNWM.

We conclude this section with a simulation model for electron and ion power balance based on the IFS/PPPL transport model [13]. For more convenient discussion we split the model in two pieces.

```
!=============== ITG instability based transport model (part 1) =========
NEQUIL=41;             !   3-moment solver for the Grad-Shafranov equation
!-------------------- Electron and ion densities ------------------------
NE:AS;        NE=NEX;     ZEF=ZEFX;
AMAIN=AMJ;    ZMAIN=ZMJ;            ! mass & charge of main ion species
AIM1=12.;     ZIM1=6.;              ! mass & charge of 1st impurity
ZEF1=ZIM1*(ZEF-1.)/(ZIM1-1.);       ! 1st impurity contribution to Zeff
NIZ1=NE*(ZEF-1.)/(ZIM1-1.)/ZIM1;    ! density of 1st impurity
NI=NE*(ZIM1-ZEF+1.)/ZIM1;           ! density of all ions
NDEUT=NI-NIZ1-NIBM;                 ! density of deuterium plasma ions
```

```
!-------------------- Auxiliary heating ---------------------------------
NBI:0.01;                           ! Neutral beam injection
!-------------------- Plasma rotation -----------------------------------
VTOR=VTORX;      VPOL=VPSWW;          ! VTOR from exp., VPOL from neocl.
ER=BTOR*(FRS*MU*VTOR/RTOR+VDIA-VPOL);  ! Radial electric field
!-------------------- ISF/PPPL transport model --------------------------
!                         Output of IFSPPL:
!  (1) R/L_Tcrit for ITG mode,  (2) R/L_Tcrit for C branch,  (3) not used,
!  (4) Mode increment,          (5) CAR24=chi_i,           (6) CAR23=chi_e
IFSPPL(CAR19,CAR20,CAR21,CAR22,CAR24,CAR23):;
SMEARR(CV1,CAR23,CAR25):;  CAR17=FTAV(CAR25,CV2);  ! Space and time
SMEARR(CV1,CAR24,CAR26):;  CAR18=FTAV(CAR26,CV2);  !   averaging
HE=CAR17+HNGSE;            XI=CAR18+FOWC*HNCHI;    ! Heat conductivity
!-------------------- Heat transport equations --------------------------
PET=-PEI;    PE=POH+PEBM-PET*TI-PRADX;
PIT=-PEI;    PI=PIBM-PIT*TE;
TE:EQ;       TE=TEX;        TEB=TEXB;
TI:EQ;       TI=TIX;        TIB=TIXB;
!-------------------- Poloidal field equation ---------------------------
HC=HCKIM;    DC=DCKIM;      XC=XCKIM;
CU:EQ;       CU=FPR;        CC=CNHR;     CD=CUBM;
!-------------------- Additional time step control ----------------------
CAR27=RTOR/LTI-CAR19;       TSCTRL(CAR27,CAR23,CAR24,CF4):;
!=============== ITG transport model (end of the 1st part) ==============
```

The beginning of the model is similar to already discussed. It prescribes different density components and additional NBI heating. Then the radial electric field is calculated which will be later used for computing shear of the plasma rotation velocity `ROTSH`. Two velocity components are calculated by the model. These are the diamagnetic velocity `VDIA` and the poloidal velocity `VPOL` taken from the neoclassical theory [23]. The toroidal velocity `VTOR` is assumed to be measured in the experiment and stored in the data file as `VTORX`. Note, however, that if the data for `VTORX` are absent the code will work substituting `VTOR` with zero. The rotational shear `ROTSH` is believed to reduce the effective increment of the ITG instability and, consequently, the anomalous transport. This effect is taken into account inside the calling subroutine `IFSPPL`.

Because of very "stiff" character of the IFS/PPPL model a special care should be taken to obtain a stable solution. This is implemented in two lines after the subroutine call. The subroutine returns electron and ion heat conductivities in two Astra vectors `CAR23` and `CAR24`, respectively. These two vectors are then successively processed with two subroutines `SMOOTH` and `FTAV` performing space and time averaging, respectively, and yielding two smoothed vectors `CAR17` and `CAR18`. As will be seen from the second part of the model, the radial output allows to view both original (i.e. calculated by `IFSPPL`) and smoothed quantities in the same plot and thus to control the quality of the procedure applied. Adjusting the two control parameters `CV1` and `CV2` the user can regulate each type of averaging and achieve nearly the complete coincidence between the original and smoothed heat conductivities in the steady state.

On the other hand, a caution should be made against using this scheme for fast processes such as the heat wave propagation. If any of the characteristic length (for the space smoothing) or time (for the time averaging) is larger than the wavelength or the period of the heat wave under consideration then an essential physic information can be lost in course of averaging and the applicability of results should be additionally inspected.

Another possibility of the accuracy control is provided by the Astra subroutine `TSCTRL` (time step control). This subroutine checks if a relative variation in any of three quantities

`CAR27` being a difference between the instability threshold and the actual gradient,

`CAR23` $=\chi_e^{ITG}$ being the electron heat conductivity as output from `IFSPPL`,

`CAR24` $=\chi_i^{ITG}$ being the ion heat conductivity as output from `IFSPPL`,

exceeds a confidence level given by the parameter `CF4`. If yes, then the time step is automatically reduced. Finally, the user can reduce the time step manually until a very low value when no numerical instability takes place and check which of the described procedures is most suitable in every particular case[16]. To conclude this discussion we note that this numerical instability is not very dangerous and the simulation can safely continue also when

[16] However, we have to warn the user that this check is possible only when the code is installed in double precision version. Otherwise, the time step cannot be reduced below a certain value which is limited due to truncation errors in $U_{pl} = \partial\psi/\partial t$.

a developed instability is clearly seen as oscillations in $\chi_{e,i}^{ITG}$. Even then the electron and ion temperatures show quite quiet behavior being hardly affected by a noise in the heat conductivity.

Other terms of heat conductivity equations and the equation for the poloidal field do not have any special features, therefore, we consider now the rest of this model.

```
!=============== ITG instability based transport model (cont.)  ==========
!-------------------- Auxiliary quantities ------------------------------
CV5=0.8E-3*VINT(PBLONB);        ! Parallel and perpendicular
CV6=1.6E-3*VINT(PBPERB);        ! energy contents in the fast ions
CAR28=BTOR*FRS*MU*VTOR/RTOR;
CAR29=HEXP;         SMEARR(CV3,CAR29,CAR30):;
CAR31=XEXP;         SMEARR(CV3,CAR31,CAR32):;
!====================== Profile output ==================================
Tex\TEX\\TEX\-5;   RLTi\RTOR/LTI\-6;    Hex\HEXP\5;     Hix\XEXP\5;
Tix\TIX\\TIX\-3;   RLTi\RTOR/LTI\-2;    KDes\CAR17\5;   KDis\CAR18\5;
Te\TE\-5;          RLTC\CAR20\-6;       ke\HE\5;        ki\XI\5;
Ti\TI\-3;          RLTD\CAR19\-2;       KDe\CAR23\5;    KDi\CAR24\5;
                  Contributions to Er
ErVd\BTOR*VDIA\-1; gamm\CAR22\-8;       Er\ER;          vpol\VPOL;
Er\ER\-1;          Pecr\PEECR;          Pe\PETOT;       PNBe\PEBM;
ErVp\-VPOL*BTOR\-1;wExB\ROTSH\-8;       shat\SHEAR;     vtor\VTOR\\VTORX;
ErVt\CAR28\-1;     betj\BETAJ;          Pi\PITOT;       PNBi\PIBM;
             Current density
j\CU\-4;        jIC\CUICR;        nuis\NUIS;    Vtor\UPL;
jNB\CUBM\-4;    Zeff\ZEF\\ZEFX;   tpf\TPF;      mu\MU\1;
joh\CUOHM\-4;   jEC\CUECR;        nues\NUES;    V||\ULON;
jBS\CUBS\-4;    sigm\CC;          betj\BETAJ;   q\1./MU\5;
             Heat conductivity
ncCH\HNCHI\10;   GSba\HNGSB\10;   grVt\grad(VTORX);    Hex\HEXP\10;
ncPS\HNPSI\10;   KDes\CAR17\10;   KDis\CAR18\10;       prad\PRADX;
ncGS\HNGSI\10;   GSpl\HNGSP\10;   HeGS\HNGSE;          Hix\CAR32\10;
```

```
ncfw\FOWC*HNCHI\10; KDe\CAR23\10; KDi\CAR24\10;        PN\PENEU-PINEU;
            Other quantities
ne\NE\-11;        dLT1\CAR27;      RLT\RTOR/LTI\-2;    nB\NIBM\-11
pNBl\PBLON;       RLNI\RTOR/LNI;   RLT\RTOR/LTI\-6;    nB1\NNBM1
ni\NI\-11;        shat\SHEAR\2.;   RLTD\CAR19\-2;      nD\NDEUT\-11
pNBp\PBPER;       RLNE\RTOR/LNE;   RLTC\CAR20\-6;      Zeff\ZEF\\ZEFX
!======================= Time output =======================================
Tix0_TIXC_-2;     Ipl_IPL;         Tex0_TEXC_-1;   <Te>_TEAVB;
Ti0_TIC_-2;       PNBI_QNBI;       Te0_TEC_-1;     <Ti>_TIAVB;
ne0_NEC;          Wthm_WTOTB;      <ne>_NEAVB; Wtot_WTOTB+CV5+CV6;
nlc_NECHC;        betp_BETAJB;     Zef0_ZEFC;  Wequ_WTOTB+0.75*CV6+1.5*CV5
TiB_TIB;          Ipl_IPL;         iter_TITER;     INB_IBMB;
TeB_TEB;          Ulc_UPLB;        tauE_TAUEB;     IBS_IBSB;
Ptot_QTOTB;       POH_QOHB;        Pe_QETOTB;      PNBA_QBTOTB;
Pi_QITOTB;        PNB_QNBI;        shin_QNBI-QBTOTB;  Prad_QRADB;
!========================= ITG model end ===================================
```

After calculation of few auxiliary quantities which are used for output only the model defines 80 vector and 32 scalar output signals. The signals are ordered in a way that groups of associated quantities are placed as far as possible in the same box and on the same screen. Some of signals are repeated that allows to present them in appropriate context.

Measurable quantities (as T_e or T_i) are plotted in the same box with the calculated ones. If corresponding experimental data are not available it is not considered as an error in a command like `Tex\TEX\\TEX`. However, Astra vectors `XEXP` and `HEXP` which show "effective" heat conductivities include gradients of measured quantities in a denominator that can cause division by zero.

The second group of 16 radial curves includes information concerning the plasma rotation. The second box here plots the ITG instability increment `CAR22` together with the rotational shear `ROTSH` (see page 91). Other radial output screens present current related quantities, different contributions to the heat conductivity, density distributions and some gradients of plasma parameters which are essential for the ITG stability.

Different local and global quantities are shown as time dependent signals. One can see here central and edge temperatures, energy contents calculated in three different ways, line and volume averaged densities, energy confinement time, partial components of the total current and, finally, volume integrated power sources and sinks.

5.3 Data setting

5.3.1 Hierarchy of data initialization

A chain of initialization procedures is invoked each time when the code is started. First of all, all variables with very few exceptions have pre-defined (default) values. The default value of a variable is used only if the variable is not defined explicitly in any other way. The most straightforward and natural way of data assignment is using a start (or data) file. A name of this file is given by the user as a second parameter in the Astra command line (see Section 5.5.2). Another possibility is assignment instruction in a model as described in the previous section. One more option is an interactive change during the run time. A sequence of assignments for different types of variables is shown in the following table.

Table 5.1. Order of variable setting

Variable type	**Default**	**`.log` file**	**Data file**	**Model**	**Run time**
Device/plasma parameters	+	+	+	+	+
Adjustable (dummy) variables	+	+	−	+	±
Grid/time/accuracy control	+	+	−	+	+
Color control	+	+	−	−	+

Every assignment type overrides all others which have more left position in this table. For instance, if a quantity is redefined in the run time then every other definitions are disabled. The symbol “ − ” means that this type of variable cannot be assigned in this way. The marking ± will be discussed in Section 5.3.4.

There are few exceptions from the table above. Namely the following quantities are used for the radial grid allocation and, therefore, they have to be defined at earlier phase of

`AB`	(none)	maximum minor radius in a horizontal plane,
`RTOR`	(none)	major radius at the same plane,
`ELONM`	(1)	maximum elongation,
`TRICH`	(0)	chamber triangularity (used for drawing only),

$\mathtt{AWALL}\quad(\mathtt{AB})\qquad \rho$ –grid size is $\rho_W = 1.1{*}\mathtt{AWALL}{*}\sqrt{\max(\mathtt{ELONM,ELONG})}$,

$\mathtt{NB1}\quad(41)\qquad \rho$ –grid step is $h = \rho_W/(\mathtt{NB1} - 0.5)$

the code execution and cannot be redefined later. It means that these six quantities can be defined either by a '.log' file (see Section 5.3.2) or by a data file and they cannot be set in a model or changed during the run. This limitation seems to be not very restrictive while it arises quite naturally from the requirement that these basic variables describing geometry of a particular device cannot depend on time. When applicable, the quantities acquire default values which are shown in brackets. However, it is recommended that the first four of these quantities are defined explicitly. In opposite, the default definition for AWALL is in most cases well sufficient, therefore, it should be defined by the user only if Astra provides a diagnostic that the default value is not appropriate. When setting the grid size NB1, the user should take into account that the grid size (variable NA1 which will be automatically selected by the code) actually used in a simulation is usually a fraction ($\geq 80\%$) of NB1. The rest of the radial grid is reserved for a description of the scrape-off layer and for a possible increase of the Lagrangian variable $\rho_B = \mathtt{RHO(NA1)} < \rho_W$.

We continue a list of variables which should be defined by the user. In addition to already mentioned AB, RTOR, ELONM, TRICH, those are (default values are shown in parentheses)

ABC(AB), SHIFT(0), ELONG(ELONM), TRIAN(0), BTOR(none), ZMJ(1), AMJ(2).

Finally, at least one of the three quantities

IPL(none), UEXT(none), LEXT(none)

also must be defined in order to avoid a possible crash during the code start-up. However, all these quantities can be defined by every method listed in the first line of Table 5.1.

Default array setting. Unlike simple variables arrays can be defined either via a data file or in a model. Following is the list of vector variables (arrays) which can aquire default definitions

$$\begin{array}{llll}
\mathtt{NE}(\rho) = \mathtt{NEX}(\rho), & \mathtt{F1}(\rho) = \mathtt{F1X}(\rho), & \mathtt{AMAIN}(\rho) = \mathtt{AMJ}, & \mathtt{CC}(\rho) = \mathtt{CCSP}(\rho), \\
\mathtt{TE}(\rho) = \mathtt{TEX}(\rho), & \mathtt{F2}(\rho) = \mathtt{F2X}(\rho), & \mathtt{ZMAIN}(\rho) = \mathtt{ZMJ}, & \mathtt{CU}(\rho) \propto \mathtt{CC}(\rho), \\
\mathtt{TI}(\rho) = \mathtt{TIX}(\rho), & \mathtt{F3}(\rho) = \mathtt{F3X}(\rho), & \mathtt{NI}(\rho) = \mathtt{NE}(\rho)/\mathtt{ZMJ}, & \mathtt{ZEF}(\rho) = \max(1., \mathtt{ZEFX}(\rho)).
\end{array}$$

These definitions are effective only if the left hand sides are not defined explicitly in a model. For instance, if `NE` appears on the left hand side in an assignment instruction in a model then the default definition `NE=NEX` is not involved. On the other hand, if any of the quantities `NE, TE, TI` is not assigned in a model then it must be defined in the data file. Otherwise, a run time error could be encountered. This requirement is not applied to the quantities `F1, F2, F3`. Because their usage is optional they need not be defined unless they are directly used in a model.

5.3.2 Reading from a "`.log`" file

Usually a large amount of tuning parameters participate in a transport modelling. Often these parameters are not known in advance, therefore, an adjustment in course of simulation is a most reasonable way for finding appropriate values. The Astra code provides a possibility to store all internal variables adjusted during the current run. This happens when user selects the option "Save const" (or the hot key `'I'`) from the run time menu (Section 5.5.3). The current values of adjustable variables are saved in a file which is read each time when the code starts up so that all variable values tailored and saved in a previous simulation will be used in the subsequent one. A name of the file is composed by adding the extension "`.log`" to a name of the running model. This convention implies that only one `.log` file, and, consequently, one set of stored values corresponds to each model file. These files can also be edited manually.

5.3.3 Input from a data file

Unlike the "`.log`" file which belongs to a corresponding model there can be an arbitrary number of input files which describe an experimental setup and are not related to any particular model. Reading input data from a file is a conventional way of data definition. The name of this file is specified as a first parameter in a command line starting execution of the Astra code. The format of this file is described in Section 5.4. Here we discuss only the main principles of data input by making use of the data file.

First of all, note that starting from this level of data initialization (see Table 5.1) every scalar or vector variable can be defined not only as a time constant but also as a time dependent function. On one hand, this adds a flexibility to the data setting. On the other hand, it results in a restriction that only a limited number of specially treated variables can be read from a data file.

Scalar variables readable from a data file. We start with a note that in future versions of the Astra code the set simple variables readable from a data file may be changed but the updated list (except for the group ZRD1, ZRD2, . . . and the three variables NB1, TSTART, TEND) can always be viewed during the code run by pressing the key 'V' or by clicking mouse in the field "Variables". In the version 5.3 of the code this set comprises

Table 5.2. List of Astra scalars definable by a data file

AB	ABC	AIM1	AIM2	AIM3	AMJ	AWALL	BTOR
ELONG	ELONM	ENCL	ENWM	FECR	FFW	FICR	FLH
GN2E	GN2I	IPL	LEXT	NNCL	NNWM	QECR	QFW
QICR	QLH	QNBI	RTOR	SHIFT	TRIAN	TRICH	UEXT
UPDWN	WNE	WTE	WTI	ZMJ	NB1	TSTART	TEND
ZRD1	ZRD2	ZRD3	ZRD4	ZRD5	ZRD6	ZRD7	ZRD8
ZRD9	ZRD10	ZRD11	ZRD12	ZRD13	ZRD14	ZRD15	ZRD16
ZRD17	ZRD18	ZRD19	ZRD20	ZRD21	ZRD22	ZRD23	ZRD24
ZRD25	ZRD26	ZRD27	ZRD28	ZRD29	ZRD30	ZRD31	ZRD32
ZRD33	ZRD34	ZRD35	ZRD36	ZRD37	ZRD38	ZRD39	ZRD40
ZRD41	ZRD42	ZRD43	ZRD44	ZRD45	ZRD46	ZRD47	ZRD48

Each variable from this list is mirrored with another variable with a similar name but with trailing "X"(which stands for eXperimental), X-scalars.

Table 5.3. List of X-scalars readable from a data file

ABX	ABCX	AIM1X	AIM2X	AIM3X	AMJX	AWALLX	BTORX
ELONGX	ELONMX	ENCLX	ENWMX	FECRX	FFWX	FICRX	FLHX
GN2EX	GN2IX	IPLX	LEXTX	NNCLX	NNWMX	QECRX	QFWX
QICRX	QLHX	QNBIX	RTORX	SHIFTX	TRIANX	TRICHX	UEXTX
UPDWNX	WNEX	WTEX	WTIX	ZMJX			
ZRD1X	ZRD2X	ZRD3X	ZRD4X	ZRD5X	ZRD6X	ZRD7X	ZRD8X
ZRD9X	ZRD10X	ZRD11X	ZRD12X	ZRD13X	ZRD14X	ZRD15X	ZRD16X
ZRD17X	ZRD18X	ZRD19X	ZRD20X	ZRD21X	ZRD22X	ZRD23X	ZRD24X
ZRD25X	ZRD26X	ZRD27X	ZRD28X	ZRD29X	ZRD30X	ZRD31X	ZRD32X
ZRD33X	ZRD34X	ZRD35X	ZRD36X	ZRD37X	ZRD38X	ZRD39X	ZRD40X
ZRD41X	ZRD42X	ZRD43X	ZRD44X	ZRD45X	ZRD46X	ZRD47X	ZRD48X

In particular, it means that there are couples of independent variables, AB and ABX , ABC and ABCX and so on, which can be defined and used quite independently with one significant exception. Namely, if any of quantities from the second list is not defined explicitly in a model then, by default, it takes value of its experimental prototype. For a practical work it means that if the quantity `IPLX` is defined by a data file and the quantity `IPL` is not mentioned in a model then there is no difference between both. In this case, the resulting code is fully equivalent to the code obtained with the explicit instruction `IPL=IPLX`. However, if the quantity `IPL` is defined in some independent way (for instance, by the boundary condition `UEXT=0`) then `IPLX` and `IPL` are fully independent.

At first sight, this convention could seem too complicated but it works so that a novice user of the code need not know anything about two different sets of data. This allows to omit in a model multiple instructions of type `ABC=ABCX`. However, an advanced user can take advantage of that keeping trace on `IPLX` as defined by experimental measurements and comparing it with `IPL` calculated in the code. Note also, that if more complicated condition for `IPL` is used, for instance, if `IPL` is calculated inside user's subroutine then the Astra code builder cannot recognize this hidden definition and will try to set `IPL=IPLX` in conflict with user's intention. A simple way to overcome this contradiction is using a dummy instruction `IPL=IPL` in a model.

Three Astra control variables `NB1, TSTART` and `TEND` are used for influencing a simulation flow. They have no X-counterpart, nevertheless, it is often useful to define these quantities by a data file. Therefore they are added to the list of variable in Table 5.2.

Vector variables readable from a data file. Only a limited number of basic vector variables (`X`-vectors) can be defined by an input file.

Table 5.4. List of X-vectors readable from a data file

TEX	TIX	NEX	NIX	CUX	ZEFX	VPOLX	VTORX
MUX	MVX	GNX	SNX	PEX	PIX	PRADX	IPOLX
SHX	ELX	TRX	VRX	G11X	G22X	G33X	DRODAX
F1X	F2X	F3X	CAR1X	CAR2X	CAR3X	CAR4X	CAR5X
CAR6X	CAR7X	CAR8X	CAR9X	CAR10X	CAR11X	CAR12X	CAR13X
CAR14X	CAR15X	CAR16X					

The list includes

- the main plasma characteristics which can be measured in an experiment (e.g. electron temperature `TEX`, density `NEX,` effective charge `ZEFX` and so on),

– calculated by external codes (e.g. quantities supplied by equilibrium, heating and current drive codes),

– a set of 19 dummy arrays `(F1X, F2X, F3X, CAR1X, ..., CAR16X)` which can acquire any meaning depending on user's will.

Any array of Table 5.4 will be defined if its name appears in a data file according to the rules discussed in Section 5.4.3 and this data file is used for the current run.

As already discussed in Section 5.3.1, a default definition of the type $\mathtt{NE}(\rho) = \mathtt{NEX}(\rho)$ is applied to the following seven `X`-arrays `NEX, TEX, TIX, F1X, F2X, F3X, ZEFX` only. The reason is that undefined `NE, TE, TI, ZEF` can cause a simulation crash and `F1, F2, F3` are added in order to keep similarity between the main and auxiliary transport equations, Eq. (59) and Eq. (75), respectively.

Usually, the default assignments are effective in absence of corresponding explicit definition only. However, there is one exception. If the equilibrium control parameter `NEQUIL` (default value 0) is set as

```
NEQUIL=-1
```

then it is assumed that all the geometry and configuration related quantities are defined by means of a data file. In this case, nine `X`-arrays `SNX, ELX, TRX, VRX, DRODAX, IPOLX, G11X, G22X, G33X` must be pre-calculated and stored in the data file. These nine quantities will then be used to calculate the corresponding set of Astra vector variables as

$$\begin{array}{lll}
\mathtt{SHIF}(\rho) = \mathtt{SHIFT} * \mathtt{SHX}(\rho), & \mathtt{ELON}(\rho) = \mathtt{ELONG} * \mathtt{ELX}(\rho), & \mathtt{TRIA}(\rho) = \mathtt{TRIAN} * \mathtt{TRX}{*}(\rho/\rho_a)^2, \\
\mathtt{VR}(\rho) = 4\pi^2\rho\, {*}\mathtt{RTOR} * \mathtt{VRX}(\rho), & \mathtt{DRODA}(\rho) = \mathtt{DRODAX}(\rho), & \mathtt{IPOL}(\rho) = \mathtt{IPOLX}(\rho), \\
\mathtt{G11}(\rho) = \mathtt{VRX}(\rho){*}\mathtt{G11X}(\rho), & \mathtt{G22}(\rho) = \rho\, {*}\mathtt{G22X}(\rho), & \mathtt{G33}(\rho) = \mathtt{G33X}(\rho).
\end{array}$$

As one can see, these `X`-arrays are selected in such a way that for concentric non-shifted circle magnetic surfaces they all are unitary. These assignments are always operative when $\mathtt{NEQUIL} = -1$ and cannot be overwritten by any assignments in a model. On the contrary, when $\mathtt{NEQUIL} \neq \mathtt{-1}$ these nine `X`-arrays have nothing to do with the corresponding left hand sides and can be used for any other goals.

Similarly, all other `X`-arrays have no pre-defined meaning and can be used for input of any radial profile. For instance, if the vacuum rotational transform `MV` should be taken from a data file then the assignment `MV=MVX` has to be made explicitly in a model. Without this assignment `MVX` can be anything. The `X`-arrays can be used in expressions according to usual rules

```
PE=PEX-PRADX+POH-PEICL
```

or appear on the left hand side of an assignment command, e.g., `CAR1X=FPR**1.5`. However, the latter construction is not recommended because if the same array `CAR1X` is occasionally defined in a data file then a result is unpredictable.

5.3.4 Run time variable setting

At the run time most of variables are accessible for a direct control by the user. There are two ways of doing this. First of all, it can be done interactively by pressing keys 'A', 'C', 'D' and 'V' or by mouse selecting proper fields from the run time menu. Another way is defining some quantities in user's subroutine. This method can be used when a quantity has to be calculated on a basis of complicated formula or equation.

As follows from Table 5.1, definition of any quantity during the code execution has the highest priority and overrides any other definitions. In particular, once any plasma or device parameter is interactively changed by the user its previous assignments in a data file or in a model are discarded.

Unfortunately, it is more difficult to fulfill the same property for a C-parameter. One should beware that a result of interactive setting C-parameter can be guaranteed only if it is not assigned in a model. If, nevertheless, a C-parameter is defined in the model by a time-independent instruction like `CV1=2` then a run-time modification will work properly, however, it will not in case of more complicated time-dependent assignment as `CV1=FJUMP(0.1)`.

A similar difficulty arises when a variable or C-parameter is defined in a user subroutine. If it is simultaneously defined in a model or changed interactively then a result is unpredictable. Therefore, it is recommended to avoid any conflicting definitions of those types.

5.4 Data file format

During installation of the Astra system the user is supplied with sample data files. New data files can be obtained by editing the original files retaining their structure. Therefore, this section can be omitted during the first reading. For more elaborated data setting we describe syntax of a data file in this section.

5.4.1 General requirements

In what follows we will distinguish between input of simple variables and vectors. The former means that a scalar function of time will be defined by the input. The latter defines a time dependent vector function of a magnetic surface while the “radial” coordinate can be selected by the user from a wide set of different options. It is also possible to define this function using two spatial coordinates (r, z) in the poloidal plane. This 2D function is then mapped to a 1D function of magnetic surfaces using the time dependent magnetic configuration of the particular Astra run.

The first two lines in a data file are reserved for a commentary. They are not used by the code itself but the first 16 positions of the first line appear in the output graphic window during the Astra run. For instance, they can be used for displaying a device name and a shot number under consideration. A length of meaningful string in a data file cannot exceed 132 characters. There is no limit for a length of commentary string.

During reading a data file each string is checked whether its first 6 characters match one of keywords. The keywords are all the names listed in Tables 5.3 or 5.4 and, additionally, the following seven control words:

`NAMEXP, NTIMES(1), FACTOR(1), FILTER(0.001), GRIDTYPE, POINTS, END.`

A variable corresponds to each of the first 6 control words. An input value of this variable should follow after the control word. Once defined these values are valid until they are redefined by another control string. The control variables `FACTOR` (being the unit conversion factor), `NTIMES` (number of time slices) and `FILTER` (defining a transformation rule), if not defined explicitly are set to a default value shown in parentheses. When the control word `END` is found then reading the data file is ceased, otherwise, it is continued until the file end. A meaning of other variables will be discussed later. All names and control words can appear in low or upper case.

If a string is started with one of the control words it is treated as **a control string**. When a control string is encountered it is parsed and an interpretation regime for a group of several subsequent lines is set. When the group is exhausted then the following string is again checked against the matching condition. If first 6 characters of the string do not match any of the keywords then the string is skipped and the next string is processed similarly. For instance, any non-alphabetic symbol in the first position of a parsed string results in ignoring the string by the code. This property can be used for writing commentaries in the input file.

For a sake of brevity the character `'X'` on the end of every name in the data file can be omitted. Then the `'X'` is added by the code so that presence or absence of `'X'`in a name does not change the result. For instance, there is no matter whether a name from Table 5.3 or Table 5.2 is used. But in any case, a data file defines a variable from Table 5.3 (or 5.4) only. Whether the corresponding variable without the trailing `'X'` is defined or not depends on the contents of a model file and interpretation rules discussed in the previous section.

When reading input data is started the code is in a scalar input mode. It means that at first all scalar variables should be defined. After one of the control words `POINTS, GRIDTYPE, FILTER` is encountered the code is switched to the vector input mode. Now the vector variables (Table 5.4) have to be defined. If a simple variable appears on the input now it will be ignored.

There are several different ways of data input in the Astra system. It can be either a direct reading data from a start file or redirecting input to so called "U-file" where the data are represented in a U-file standard developed in PPPL for storage of experimental data. The U-file standard is used also for the ITER data base. This standard is not described in this paper but it is quite transparent and can be easily understood from few examples attached to the Astra code.

Different input formats can be combined in one input file. Moreover, mixing different formats is permitted for different time slices of the same input quantity. Definition of a time dependent scalar or vector can consist of several subsequent groups for different time slices. However, one restriction should be mentioned here. These groups can use different grids but the subsequent times should be given in the increasing order and all groups related to the same variable should be contiguous. In other words, when a time evolution for one variable is defined and the input sequence is switched to another variable then return to the first one is forbidden.

5.4.2 Reading data from a U-file.

First of all, the first 6 characters of every line are considered. If they are recognized as a name of variable then the next word in the string is analyzed. If this word is `"U-file"` (case insensitive) then the string is interpreted as a reference (pointer) to a U-file where a data set for the current variable is stored. The name of the U-file should be separated from the word `U-file` with a colon `‘:’`. Optionally, a multiplicative factor can be given after the second `‘:’`.

The following 5 lines show different allowed formats of a pointer to U-file:

```
IPL  ⟶|U-file ⟶|: ⟶|A00000.aaa ⟶|Factor: ⟶|0.000001
IPL  ⟶|u-file ⟶|:A00000.aaa ⟶|factor:1.E-6
IPLX ⟶|U-FILE ⟶|:A00000.aaa ⟶|:1.E-6
iplx ⟶|U-File ⟶|:A00000.aaa:1.E-6
ipl  ⟶|u-file:A00000.aaa:1.E-6
```

All these lines have the same effect and mean that the variable describing the plasma current, `IPLX`, should be taken from the U-file `AWD/udb/A00000.aaa` (`‘AWD’` denotes the Astra working directory) where it is stored in amperes and multiplied by 10^{-6} because the Astra code employs MA as a current unit. The word `"Factor"` can be omitted. The symbol ⟶| shows a delimiter, i.e. any combination of spaces and horizontal tabulations.

Note that for this input format

– only a U-file name is case sensitive,

– only one line is allowed for every variable,

– the word “factor” and the factor value can be omitted,

– if the factor value is omitted it is set to 1,

– a delimiter “ ⟶| ” is required only

  – between a name of variable and the word “U-file”,

  – between a name of U-file and the word “factor” if the latter is present.

The U-file name should be given with a path relative to the directory `AWD/udb/`. For instance, if the name is `../exp/A00000.aaa` then the U-file containing data should be in the subdirectory `AWD/exp/`. Every permissible U-file name is allowed but the file must exist and contain data. For input of a scalar variable it should be a 0D or 1D U-file. Moreover, in the latter case it should have time as independent variable. For input of a radial array it should be 1D or 2D U-file.

In the ITER data base every U-file consists of a number of merged separate U-files for different signals. In order to select an appropriate record one can use a specifier `'<'` as shown below

```
IPL   U-file:JET/jet_40847_1D < IP  :1.E-6
ABC   U-file:JET/jet_40847_1D < AMIN:1.E-6
BTOR  U-file:JET/jet_40847_1D < BT  :1.E-6
ZRD1  U-file:JET/jet_40847_1D < LI  :1.E-6
```

In this example, all four Astra variables `IPLX`, `ABCX`, `BTORX` and `ZRD1X` are read from the same U-file `AWD/udb/JET/jet_40847` where they are stored under the names `IP`, `AMIN`, `BT` and `LI`, respectively.

We also remind that, first of all, scalar variables have to be defined and then the input mode can be switched to the vector input. Thus, the group

```
IPL    ⟶| u-file:A00000.aaa:1.E-6
FILTER ⟶| 0.002
TEX    ⟶| u-file:B00000.bbb:factor ⟶| 1.E-3
MV     ⟶| u-file:C00000.ccc
```

first defines the simple variable `IPLX` as discussed before. Then the control word `FILTER` switches the input regime to the vector mode. (It could also be the words `POINTS` or `GRIDTYPE` but these two words do not affect the input from a U-file.) Then the radial array `TEX` is taken from the U-file `AWD/udb/B00000.bbb` and all array values are multiplied by the factor 10^{-3}. Finally, the array `MVX` will be defined.

5.4.3 Fixed format input

If the first word in a string is recognized as a name from Table 5.3 or Table 5.4 and the second word is not `"U-file"` then the rest of the string will be parsed in **positional or fixed format** mode because all symbols will be interpreted according to their positions in the string. This format was used in old versions (before 3.0) of the Astra code. Although it is quite restrictive, for compatibility it is also supported in more recent versions.

Fixed format for a scalar input. In the scalar input mode all positions in a string have the meaning specified by the table:

Field position	1-6	9-14	17-22
Field meaning	`NAME`	Time	Value

`NAME` is one of the names listed in Table 5.2 (or Table 5.3). "Time" and "Value" are numbers which obey Fortran rules for integer or real constants and cannot use a place beyond the reserved fields of 6 positions. Non-used positions (except for those on the end of a string) within the reserved fields and the positions 7, 8, 15 and 16 should be filled with spaces. In the fixed format, the tabulations cannot be used as a separator between meaningful fields. For a time-independent variable the second field "Time" can be skipped (filled with spaces). For instance, the string (spaces are shown with a symbol '_')

```
BTOR____________3.5
```

is equivalent to the Fortran statement

```
        BTORX=3.5
```

A variable evolving in time can be given as

```
Name____Time____Value
123456__901234__789012
IPL_____.0______.1
IPL_____1.______1
IPL_____2_______1.0
IPL_____2.5______2.E-1
```

The first two lines will be interpreted by the Astra code as comment strings. The whole group determines the internal variable `IPLX` as a function of time

$$\mathtt{IPLX}\,(t) = \begin{cases} 0.1 & \text{if} \quad t \le 0 \\ 0.1 + 0.9t & \text{if} \quad 0 \le t \le 1 \\ 1 & \text{if} \quad 1 \le t \le 2 \\ 1 - 0.8(t-2) & \text{if} \quad 2 \le t \le 2.5 \\ 0.2 & \text{if} \quad t \ge 2.5 \end{cases} \tag{101}$$

As can be easily seen the time evolution is obtained by a linear interpolation between subsequent times. Outside the specified time interval the quantity is extended as a constant.

Fixed format for arrays. As already mentioned, the Astra code starts reading data file being in the scalar input mode. In order to invoke vector mode the variable `POINTS` must be defined. It can be done by a line as

```
POINTS ⟶| 20
```

This line switches the input mode to reading vector variables and additionally declares that the vector dimensionality is 20. Then the first 6 characters in a line will be compared with array names of Table 5.4. If it fits one of those name and the next word is not `"U-file"` then the string should have the following fixed format:

Field position	1-6	7-11	12-16	17-21	...	`7+5*j-11+5*j`	...
Field meaning	`NAME`	Time	Value	Value	...	Value	...

All positions which are not used must be filled with spaces. On the other hand, no spaces is required between the different fields. The total number of fields in a line is `POINTS` $+ 2$ and $1 \le j \le$ `POINTS`.

As in the case of scalar variables the first two fields in the line give a variable name and a time value. If the variable is time independent then the second field can be empty (filled with spaces). Remaining fields define radial dependence of the vector assuming that its values are given on the equidistant grid in a radial variable. By default this radial variable is $0 \le a \le a_B =$ `ABC`. This default convention can be changed by redetermining variable `XINPUT` (see Section 4.6.1).

A time dependent vector can be defined similar to a time dependent scalar. In the example below, the time evolving density `NEX` is determined.

```
POINTS ⟶| 10
123456789012345678901234567890123456789012345678901234567890
Name__Time_ValueValueValueValueValueValueValueValueValueValue
NE____.0___2.30_2.27_2.23_2.15_2.02_1.8__1.49_1.13_0.8__.522
NE____0.2002.3842.4302.4562.4102.3022.1281.9031.6471.3601.150
NE____.4___2.66_2.77_2.85_2.86_2.76_2.58_2.33_2.03_1.68_1.43
TE ⟶| U-file:T12345.ECE:.001
TI_________3.22_3.14_3.07_2.64_2.07_1.57_1.14_0.71__.25__.12
END
```

The first line in this example sets value `POINTS=10` for all subsequent lines until it will be changed by another command line. The second and third lines label positions. As long as they include no valid name in the first six positions they are ignored by the code. As shown in the second line for "NEX" no delimiter is required between the different fields. Note that a U-file T12345.ECE can have a number of radial points different from 10 but it will not affect the current value of POINTS. Because the length of every input line is limited by 132 positions no more than 24 array elements can be used in this format.

5.4.4 Free format input.

The control words NAMEXP, NTIMES, FACTOR can be used for input both scalar and vector variables. Moreover, any of the control words POINTS, GRIDTYPE, FILTER switches the input regime from scalar to the vector mode. An order and case of the control words in a line are inessential. Once defined, each control variable value is valid until it is redefined by another control string. If the six leading characters in an input string match any of the control words then the following rules are applied:

- an order of control words in a control line is arbitrary.
- the word NAMEXP must be present in the line, and followed by a name from Table 5.2 or 5.4,
- the words NTIMES, GRIDTYPE and POINTS (if present) should be followed by an integer number,
- the words FACTOR and FILTER (if present) should be followed by a real number,
- the words NTIMES, FACTOR and FILTER can be omitted, then they acquire default values 1, 1. and 0.001, respectively.

A number after the word NTIMES defines the number of time slices in the input. These time values should always follow the control line. A real number after the word FACTOR defines the multiplication (e.g. unit conversion) factor between the input data and in the Astra variable. In the free format, no restrictions apart from those required by the Fortran syntax are imposed.

Free format for reading scalars. In the scalar variable input mode each control line should be followed by a group of 2*NTIMES real numbers. First NTIMES of them give time and the rest give variable values. For instance, the previous example Eq.(101) can be written as

```
NAMEXP IPL    NTIMES 4    FACTOR 1.E-6
 .0         1.         2        2.5
.1e6     1.E+6     1.E6      2.E5
```

The next example sets a box-like time dependence for the quantity QECRX

```
NTIMES 4     NAMEXP QECR
0.299     0.3       1.5       1.501
0.0       1.0       1.0       0.0
```

It switches on from zero to 1 MW at $t = 0.3$ s and switches off at $t = 1.5$ s.

Free format for reading vector variables. The control words POINTS, GRIDTYPE, FILTER switching the input regime to the vector mode have the following meaning

– POINTS is a dimension of the input vector,

– GRIDTYPE is an integer number specifying the radial variable,

– FILTER is a real number determining a method of data transfer.

All control variables can be redefined after any input group as many times as needed. Otherwise they extend their meaning to all subsequent groups with one exception. The grid type and size, as determined by the variables GRIDTYPE and POINTS, are not applied to the input from a U-file. The reason is that the input grid is defined independently in each U-file and this definition is effective inside this U-file only. When processing the U-file is over the variables GRIDTYPE and POINTS retrieve their values. The action of the variable FILTER is extended to all input formats. The control words NAMEXP, NTIMES, FACTOR and their values have the same significance and obey the same rules as for simple variables. Input data should be split in strings no longer than 132 character each.

The variable POINTS has no pre-defined value. It must appear in a control string at least once. It sets the number of grid points to be used in the subsequent input group. This grid has nothing to do with the grid used in the transport equation solver. Therefore, a rule for data transfer from one grid to another is required. Variables GRIDTYPE and FILTER define these rules.

The quantity GRIDTYPE defines the radial variable and a type of the input grid. In the current version of the code it can take the following values.

Table 5.5: Types of radial grid for input arrays

Value	Radial coordinate	Type of the grid	Units	Comment
0	$a_j = \mathtt{AB}{*}\kappa_j$	Equidistant	[m]	
1	$a_j = \mathtt{ABC}{*}\kappa_j$	Equidistant	[m]	Default input mode
2	$\rho_{N,j} = \kappa_j$	Equidistant	[d/l]	
3	$\rho_{\psi,j} = \kappa_j$	Equidistant	[d/l]	
4	$\rho_{V,j}$	Equidistant	[m]	Unimplemented
5	$\psi_{N,j}$	Equidistant	[d/l]	Unimplemented
6	Φ_j	Equidistant	[Wb]	Unimplemented
10	$a_j \in [0, \mathtt{AB}]$	Arbitrary	[m]	
11	$a_j \in [0, \mathtt{ABC}]$	Arbitrary	[m]	
12	$\rho_{N,j} \in [0, 1]$	Arbitrary	[d/l]	
13	$\rho_{\psi,j} \in [0, 1]$	Arbitrary	[d/l]	
14	$\rho_{V,j}$	Arbitrary	[m]	
15	$\psi_{N,j}]$	Arbitrary	[d/l]	
16	Φ_j	Arbitrary	[Wb]	
18	$\{r_j, z_0\}$	Arbitrary	[m]	Vertical chord
19	$\{r_0, z_j\}$	Arbitrary	[m]	Horizontal chord
20	$\{r_j, z_j\}$	Arbitrary	[m]	2D grid

where $\kappa_j = \dfrac{j-1}{\mathtt{POINTS}-1}, \quad 1 \le j \le \mathtt{POINTS}$ and

$$\rho_N = \frac{\rho}{\rho_B}, \qquad \rho_V = \sqrt{\frac{V(\rho)-V(0)}{V(\rho_B)}}\ , \qquad \psi_N = \frac{\psi(\rho)-\psi(0)}{\psi(\rho_B)-\psi(0)}, \qquad \rho_\psi = \sqrt{\psi_N}.$$

When GRIDTYPE < 10, an equidistant grid is selected with respect to the radial coordinate used. Note that equidistant grids on different radial variables do not coincide with one another.

When GRIDTYPE ≥ 10, the grid is arbitrary and is expected to be supplied by the user. In this case, the grid can be extended also beyond the interval specified by the second column in Table 5.5 but all data beyond this interval will be ignored. If the innermost point of the input grid does not coincide with the magnetic axis $\rho = 0$ then the input quantity at $\rho = 0$ is set to the nearest value provided. The same is done for the outer side. In other words, every quantity is extended by a constant outside the region where it is set by the input data if this region is smaller than the region of definition.

After that the input grid is mapped to the Astra transport grid by the quadratic interpolation while the data are transferred to the transport grid making use of the procedure described in Section 4.9.6 where $\alpha =$ FILTER.

We consider now few examples. The first one reads:

```
******
NAMEXP  PRAD  NTIMES  2  POINTS  10  GRIDTYPE  1
0.   0.03
1.683E-02  1.835E-02  2.118E-02  2.511E-02  3.063E-02
3.730E-02  4.366E-02  4.700E-02  4.493E-02  3.684E-02
1.10  0.95  0.78  0.63  0.51  0.45  0.37  0.25  0.13  0.04
******
POINTS  11 GRIDTYPE  10  NAMEXP  PRAD
 .2
 .0  .05  .1  .15  .2  .25  .3  .35  .4  .45  .5
1.613903E-2  1.683072E-2  1.835108E-2  2.118123E-2  0.02511  3.063486E-2
3.730636E-2  4.366064E-2  4.700778E-2  4.493699E-2  3.684750E-2
******
```

Here stars separate successive input groups and are ignored by the code. The first control line declares that the input data should be written in the array with the name PRADX on the radial grid of 10 points which is equidistant in the variable a on the segment $[0,\mathtt{ABC}]$. Two time slices ($t = 0$ s and $t = 0.03$ s) given in the next line are followed with 20 numbers which give the function values. The second control line says that the input should be continued using a new grid of 11 nodes. Then the time $t = 0.2$ s is given, then 11 data for $\{a_j\}$ and 11 values of the function. Note that although the control pair "NTIMES 1" is omitted the corresponding time data must be present.

The second example:

```
********************************* CAR3X *********************************
>>>> Data are given along a vertical chord
POINTS  10  GRIDTYPE  18  NAMEXP  CAR3
 0.3
 1.5
```

```
-0.81 -0.64 -0.47 -0.30 -0.13  0.04  0.21  0.38  0.55  0.72
 0.04  0.31  0.56  0.78  0.93  0.96  0.87  0.68  0.44  0.17
********************************* CAR1X *********************************
>>>> Data are given along a downshifted horizontal chord
POINTS  11  GRIDTYPE  19  NAMEXP  CAR1
 0.3
-0.25
 1.10  1.20  1.30  1.40  1.50  1.60  1.70  1.80  1.90  2.00  2.10
-0.09  0.26  0.53  0.72  0.83  0.87  0.83  0.72  0.53  0.26 -0.09
********************************* CAR2X *********************************
>>>> Data are given along an inclined chord
NAMEXP  CAR2X  GRIDTYPE  20
 0.3
 1.10  1.20  1.30  1.40  1.50  1.60  1.70  1.80  1.90  2.00  2.10
-0.30 -0.28 -0.27 -0.25 -0.24 -0.23 -0.21 -0.20 -0.18 -0.17 -0.15
-0.12  0.24  0.51  0.72  0.84  0.89  0.87  0.76  0.58  0.31 -0.03
*************************************************************************
```

defines three different quantities at the same time $t = 0.3$ s which is specified by the first number after each control line. In the first input group, GRIDTYPE=18, therefore the time is followed by the distance to the torus axis $r_0 = 1.5$ m, then $\{z_j\}$ (also in meters) and $\{f_j\}$ are given, where $1 \le j \le 10$. The second control line switches the name, the grid type and size. It determines the horizontal chord with $z_0 = -0.25$ m, $\{r_j\}$ (in meters) and then gives the function values $\{f_j\}$ for $1 \le j \le 11$. The third group inherits POINTS=11 and after the time value gives 33 numbers for $\{r_j\}$, $\{z_j\}$ and $\{f_j\}$, with $1 \le j \le 11$.

5.4.5 Input format for the plasma boundary.

When the three-moment equilibrium solver (NEQUIL $\le$ 41) is used the plasma boundary should be described by a set of scalar variables RTOR, ABC, SHIFT, ELONG, TRIANG, UPDWN. This boundary setting can also be used for more complicated equilibrium solvers, however, in the latter case, a more complicated boundary shape is usually necessary.

A special input format is provided for setting the plasma boundary of arbitrary shape.

The format is similar to the free format discussed above. A new keyword `BND` (or `BNDX`) is introduced which declares that the subsequent group of numbers will describe the plasma boundary. One should beware that the boundary input switches the input mode from scalar to vector type.

Plasma boundary input is illustrated with an example:

```
NAMEXP  BND  NTIMES  2  POINTS  8
0.263   0.313
2.7750  2.7000  1.7631  1.7695
2.6283  2.6371 -1.5053 -1.5104
3.8605  3.8605  0.2516  0.2516
1.9041  1.9167  0.2516  0.2516
3.3000  3.2250  1.5363  1.5932
2.2748  2.2500  1.5172  1.5014
2.2515  2.2604 -1.0141 -1.0141
3.0728  3.0750 -1.1406 -1.1526
```

Here the time evolving plasma boundary is given by pairs of coordinates $\{r_j, z_j\}$ where $1 \le j \le \texttt{POINTS} = 8$. First two time values $t_1 = 0.263$ s and $t_2 = 0.313$ s are given. Then follow $2 \times \texttt{NTIMES} \times \texttt{POINTS}$ coordinates $r_1(t_1)$, $r_1(t_2)$, $z_1(t_1)$, $z_1(t_2)$, $r_2(t_1)$, $r_2(t_2)$, ... , $r_8(t_1)$, $r_8(t_2)$, $z_8(t_1)$, $z_8(t_2)$. It is understood that the boundary coordinates are linearly interpolated if $t_1 < t < t_2$ and do not evolve otherwise.

In the current version of the code all data for the boundary input should be given in one group (multiple appearance of the combination `"NAMEXP BND"` is not allowed) and the the total number of data in this group $(2 \times \texttt{POINTS} + 1) \times \texttt{NTIMES}$ cannot exceed 25000.

In conclusion, we outline a few **common rules for a data file**:

– The first two lines should be used for a device/shot identification and cannot contain a variable input.

– No meaningful (non-commentary) lines can use more than 132 positions.

– If different time slices for the same quantity follow one another then the time sequence should be put in increasing order.

– A sequence of records for every input quantity should be contiguous, i.e. records for one quantity cannot alternate with records for another.

– At first, all simple variables should be determined.

– A control line containing one of the words `POINTS, GRIDTYPE, FILTER` switches the input to the vector mode. After such a line radial profiles can be defined only.

– A total number of input values for all simple variables and all time slices cannot exceed a value of the parameter `NTVAR` which is set during the code installation. An actual value of the parameter is defined in the file `AWD/for/parameter.inc` (by default `NTVAR=2000`). For radial profiles the same limitation is given by the parameter `NTARR` (default value 5000).

– If the control word `END` is encountered then the subsequent contents of the data file is ignored. Otherwise, all records until the end of the file will be processed.

5.5 Brief operation guide

5.5.1 Installing the code

To install the Astra code the user is supplied with two files. One of them is an archived file containing all source files for the code. Another is a script file `"InstallAstra"`. Both files should be put in the same directory and then the code is installed by running the script file. The script provides many options and can accept several control parameters. For instance, there are options to install the code for multiple users with a shared kernel, options which specify names for the directory to put the code in and for the kernel to be linked to. Only a simplest option is considered here.

Running the script

```
InstallAstra
```

without any control parameters should install the Astra kernel with the name `"Astra"` and the Astra user directory with the name `"astra"`. However, for an unknown platform some additional adjustments could be requested. If an error is encountered a message is printed. Then after a correction `InstallAstra` should be run repeatedly. Usually, after the Astra code is successfully installed it offers to set two aliases

```
Astra = AWD/.exe/astra
```

and

```
    Review = AWD/.exe/view
```

where AWD is replaced by the actual Astra working directory name. Both alias names are arbitrary but for convenience these shorthands will be used in what follows.

5.5.2 Starting the code

After the Astra code is installed a syntax of the starting command and the main contents of this section can be displayed by the command

```
    Astra help
```

or

```
    Astra -h
```

The code is started by the command

```
    Astra [data_file_name] [model_name] [start_time] [end_time]
          [post_view_file_name] [test]
```

which can take up to 6 parameters separated by spaces. Any of them is optional and can be omitted. However, a meaning of every parameter is determined by its ordinal position in the command line, therefore, if the dropped parameter is not the last one in the line it should be replaced with a comma. Significance of the parameters should be obvious:

data_file_name	is the data file name in the directory AWD/exp/ ,
model_name	is the model name in the directory AWD/equ/ ,
start_time	is the initial time of the simulation.

If one of these parameters is not defined then the value from the previous run is used. For instance, the command

```
    Astra readme showdata
```

starts Astra run of a model described by the file showdata (full name AWD/equ/showdata) with initial data from readme and initial time $t_0 = 0$ (default value). To repeat the same run one needs to enter just

```
    Astra
```

The command

```
    Astra , analysis
```

can then be used to run the model analysis with the same data file readme .

Remaining parameters:

`end_time` is the time when the simulation has to be stopped,

`post_view_file_name` is the name for a file where the results will be saved.

These two parameters are of use when interactive control is not required and the code is run in the background. Then the 5-th parameter can be used for a post-run viewing the results of modelling as described in Section 5.5.4. Normally, Astra writes all results of calculations in a file so that, after the run, one can inspect the results of simulation as if they are watched interactively. These results are stored in a special post-view file and each new run destroys the previous one. Moreover, if several Astra processes run simultaneously they all will write into the same file and corrupt it. In order to avoid this corruption during few simultaneous Astra runs the user has to employ the 5-th input parameter and submit the output of different Astra processes to different files.

Finally, if the 6-th parameter `"test"` or `"TEST"` is set, then the code execution is suspended after the all Fortran source files are created (after Step 2 in the flow diagram on page 10). At this moment, the user can manually introduce any code modifications which are not foreseen in the Astra compiler and resume execution of the modified code.

A superseding version of this command reads

```
Astra [-htVM] [-v data_file_name] [-m model_name] [-s start_time]
      [-e end_time] [-r post_view_file_name]
```

The command

```
Astra -t
```

is equivalent to

```
Astra , , , , , test
```

The commands

```
Astra -V
```

and

```
Astra -M
```

list all available data files and models, respectively.

5.5.3 Run time control

When the Astra code is started it sets up an interactive mode. In this mode, the user can control the simulation flow using menu boxes displayed in the main window. If a menu box is selected with a mouse (any button) then either a message is printed about the fulfilled action or the run is suspended and a dialog window appears. After carrying out required changes and exiting the dialog window the run continues.

The key `<Esc>` always means exit from the active window. For a **dialog window** pressing `<Esc>` applies all changes, closes the dialog window and returns cursor and focus to the main window. When pressed in the **main window** `<Esc>` **stops** the code. A dialog window displays a set of boxes containing numbers and words. Only one box shown in bold (highlighted) is active at a time. The user can select a box either with the mouse or with the keyboard arrows. Data in the selected box can be modified in order to change the run flow as required. In a dialog window, useful is the key `‘?‘` which prints an information about the quantity in the highlighted box.

All menu options of the main window are duplicated by keyboard keys (letters are case insensitive). Most of menu items do not require much explanations therefore we restrict this section to description of the main features of interactive control. Instead of boring reading we encourage the user to test different options and find the most suitable mode of operation.

On-line help and information. The list of operational keys can be obtained by pressing `‘H‘` or `‘?‘` or by selecting the menu box `"Help"`. The key `‘L‘` types the results of model analysis by the Astra compiler. The current radial grid is explained by `‘X‘`. Correspondence between the menu options and the hot keys of this group is shown in the table below.

Menu option	`Help`	`Type model`	`What X-axis?`
Hot key	`H or ?`	`L`	`X`

Execution control is performed by using the following menu options:

Menu option	`Variables`	`Constants`	`Grids`	`Save tuning`
Hot key	`V`	`C`	`D`	`I`

The keys `‘V‘`, `‘D‘` and `‘C‘` invoke dialog windows which allow to change Astra variables listed in Tables 4.9–4.11, 4.13–4.16 and 4.17, respectively. The key `‘I‘` stores the current values of all these variables in a file with the extension `.log` as described in Section 5.3.2.

The set of variables accessible via a key `‘D‘` includes also a set of control parameters defined by the command line Eq. (92). In this dialog window, one can change the control times for calling every subroutine. There is an operation which can be done by the keyboard only. It is usage of a control key as described in Section 5.24. Pressing `<Ctrl>` together with the predefined in the model key one can call the corresponding subroutine out of order. If such a combination matching one of the subroutine calling symbols is pressed when code is in waiting mode the one time step is made.

Waiting and run modes are two alternative states of the code. Two keys `<Space>` and `<Return>` are reciprocal and set the code in waiting and run modes, respectively.

Menu option	Run	Step	Quit	Calling subroutine
Hot key	<Return>	<Space>	<Esc>	<Ctrl><Letter>

The run mode is the default one when the calculations are performed and the results presented on the screen are periodically renewed. In the waiting mode, code does nothing although all keys are operable as in the run mode. In this mode, it is possible to advance calculation step by step pressing `<Space>` repeatedly. Additionally, this mode can be used for mapping different radial coordinates to one another because the cursor position is digitized and displayed showing several flux labels simultaneously.

Presentation control. Numerous keys are available for changing presentation of simulation results. First of all, we will distinguish 9 different presentation modes. Every of them can be set up pressing the corresponding digit-key. The first three modes plot radially dependent functions:

Menu option	16*f(#)	8*f(#)	8*f(psi)
Hot key	1	2	3

In the first line, the number in front of `*` shows a number of curves plotted simultaneously at one screen. Instead of the mark `#` in the symbolic notation above the Astra code shows the actual radial coordinate which is defined by the parameter `XOUT` of Table 4.13 and can be one of:

– a introduced by Eq. (88), $\texttt{XOUT} = 0$ or $\texttt{XOUT} = 1$,

– ρ introduced by Eq. (24), $\texttt{XOUT} = 2$,

– ψ introduced by Eq. (21), $\texttt{XOUT} = 3$.

The parameter XOUT appears in the dialog window ‘D‘ under the name Xaxis. It can be changed at any time and stored by pressing ‘I‘.

The first (default) mode which simultaneously presents 16 radial functions was briefly considered in Section 5.2.6 and depicted there schematically as

1 1 & 9	**2** 2 & 10	**3** 3 & 11	**4** 4 & 12
5 5 & 13	**6** 6 & 14	**7** 7 & 15	**8** 8 & 16

where the large numbers numerate boxes containing two radial profiles each. The small digits are the sequence numbers of these radial profiles as they appear in a model file. If the key ‘N‘ is pressed then the next 16 curves will be displayed as shown below.

9 17 & 25	**10** 18 & 26	**11** 19 & 27	**12** 20 & 28
13 21 & 29	**14** 22 & 30	**15** 23 & 31	**16** 24 & 32

If ‘N‘ is pressed several times so that the list of profiles requested for the radial output is exhausted then the first screen appears again. The scan back can be carried out by pressing ‘B‘.

Menu option	Colors	Backward	Next	Refresh	Appearance
Hot key	A	B	N	R	.

These keys operate similarly in all graphic modes. Colors can be changed in a dialog window opened by pressing ‘A‘. The key ‘R‘ redraws the current screen if it was corrupted for any reason. The hot key ‘.‘ (dot) is especially useful for black and white monitors or for plotting data on black and white printers. It changes appearance of the curves either making them dashed or marking them with different symbols.

There are few other possibilities to change the appearance of curves.

Menu option	Select	Scales	Windows	Y-shift
Hot key	M	S	W	Y

The user can change scales for every curve ‘S‘, Y-offsets ‘Y‘ and also redirect any curve in another box ‘W‘. The key ‘M‘ (option "Select") in modes 1, 2, 3, 6, 7 combines actions of all three keys ‘S‘, ‘Y‘ and ‘W‘.

The second mode (hot key ‘2‘) shows the same profiles in larger scale. Repeated pressing of the key ‘2‘ switches between two different frames for the curves. In one case, the zero-ordinate line is in the bottom of a window, in another, in the middle of it.[17] This can be used for plotting curves with alternating sign. Schematically the second mode can be shown as

1	**2**
1 & 5 & 9 & 13	2 & 6 & 10 & 14

It is obtained if the lower row of boxes of the first mode is overlapped with the upper row. Consequently, the screen obtained by pressing ‘N‘ can be represented as

3	**4**
3 & 7 & 11 & 15	4 & 8 & 12 & 16

In other respects, the mode 2 is similar to the mode 1.

The mode 3 is obsolete because it always uses the coordinate ψ for the X-axis and, therefore, it is a particular case of the mode 2. It will be removed in future versions.

The next three modes allow to plot time dependences

Menu option	2*f(a,t)	2*f(R,t)	8*f(t)
Hot key	4	5	6

The mode 6 shows eight time dependent curves placed as

1 1 & 3 & 5 & 7
2 2 & 4 & 6 & 8

The time axis is defined by two parameters TINIT and TSCALE included in the dialog window ‘D‘ . The first parameter defines position of the origin and the second one the total length of the scale. If in process of long simulation the current time t becomes greater than TINIT + TSCALE then the current evolution of parameters is not seen any more. In

[17] A similar feature is also available for modes 3, 4, 5 and 6.

this situation, one can interactively change the parameters `TINIT` or `TSCALE` and make the evolution visible again but most probably it will require a new correction quite soon. Sometimes, a better solution would be to set `TSCALE` negative. Then abs (`TSCALE`) will have the same sense as before but the t -axis will become floating. In other words, `TINIT` will be ignored and alignment of graphs will be performed with respect to their right rather than left edge.

If the key ‘`6`‘ is pressed in the sixth mode once more then the frame will be changed to four boxes with two curves in each box

1 `1 & 5`
2 `2 & 6`
3 `3 & 7`
4 `4 & 8`

The next pressing ‘`6`‘ will put all eight curves in one box and, finally, the forth pressing ‘`6`‘ returns the drawing into the original state.

The modes 4 and 5 are similar and differ by the X-axis only. Both show two radial profiles as functions of a coordinate in the mid-plane. In the mode 4, this coordinate is a (which is close to the minor radius), in the mode 5, it is the major radius R. In these two modes the radial profiles can be plotted for several time slices simultaneously. When the mode is switched on all available radial profiles are shown with shadow color. Pressing the key ‘`M`‘ one can select those profiles which should be plotted and remove others. In other respects, these two modes are similar to the mode 2.

The mode 7 can also be attributed to modes plotting time dependences. It shows the same quantities as the mode 6 but as function one another. In other words, it gives simulation trajectory in a phase space.

Menu option	`Phase space`	`Equilibrium`	`User graph`	`No graphics`
Hot key	`7`	`8`	`9`	`0`

The mode 8 presents the plasma configuration (as a result of the equilibrium solver) and some other informations as NBI geometry or resonance position and so on. The mode ‘9‘ is reserved for plotting user’s graphs. It calls the subroutine `AWD/sbr/mydraw.f` which can be edited by the user and include any graphics not foreseen in the Astra code. The key ‘0‘ suppresses the graphic output.

Output files. The user can save the figure which is currently on the screen as a PostScript file either in portrait (‘G‘) or in landscape (‘Q‘) orientation. The data can also be saved in three different text formats. So the keys ‘F‘ and ‘P‘ results in writing ASCII files which include either all radial profiles (if any of modes 1, 2, 3, 4 or 5 is active) or all time dependences (in mode 6). PostScript and ASCII files are written into directory AWD/dat/ . Their names are generated automatically and reported in the command window. The same data as are stored in a file by pressing ‘F‘ can be typed to the screen with the key ‘T‘ .

Menu option	Port_PS	Land_PS	Write data	Type data	U-files
Hot key	G	Q	F & P	T	U

It is also possible to save data in the U-file format. When the key ‘U‘ is pressed a dialog window is opened where a name of the U-file to be stored should be given. Note that a type of the U-file depends on the current mode of data presentation. In the radial modes 1, 2 and 3, 1D U-files are created with the radial variable which is currently on the screen. Similarly, in the time mode 6, time dependent 1D U-files are written. Finally, in the mode 4, a 2D U-file with both radial and time dependences is created.

5.5.4 Post-run viewer

Although the Astra code is basically designed for interactive work it can also be run in background and the results may be inspected afterwards. To enable this option the Astra code always saves the main results of every run in a (post-view) file and provides a tool for its viewing. A typical length of this file is few MB, therefore, if the file has not been saved by an explicit command it will be overwritten by the next run and lost. Therefore, the user should take care and issue the appropriate command if results of the run should be saved. We remind that the simulation results can be submitted to a named file directly by the Astra starting command (Section 5.5.1).

Review command syntax. The corresponding command in the full format reads

```
Review [-hl] [-d file_name] [-s file_name] [-r file_name] [file_name]
```

The command

```
Review -h                    or       Review help
```

prints the short description of the Astra post-run viewer. When called without parameters

```
Review
```

the command retrieves the most recent Astra run. A name of the post-view file to be inspected can be given explicitly

```
Review file_name
```

A list of all available post-view files can be obtained by

```
Review -l                    or       Review @
```

The saving command is

```
Review -s file_name          or       Review file_name save
```

An existing post-view file can be removed with the command

```
Review -d file_name          or       Review file_name del[ete]
```

Operation control. The `Review` command opens a graphic window which is very similar in image and operation to the Astra run window. Similar to the run mode, there is a selection of menu boxes duplicated with control keys which mostly coincide in both modes. Moreover, some additional keys (arrows and `<Home>, <End>, <PgUp>, <PgDn>`) are enabled. These keys serve to change the current viewing time (shown in the right upper corner) thus selecting appropriate time slices for radial profile plotting. The arrows advance the current time either in small (← and →) or in large (↑ and ↓) steps scanning the simulation time up and down, respectively. `<Home>` (`<End>`) moves the viewing time to the beginning (end) of the record. Two additional keys `<PgUp>` and `<PgDn>` can be used if the review file is too long and can be loaded by parts only.

On the other hand, some limitations for changing control parameters are added because no calculation is made in the view mode. For instance, the keys `‘V‘` and `‘C‘` can be used for inspecting the current values of control or plasma parameters which cannot be changed in the view mode. The full list of active keys is printed when the key `‘H‘` is pressed.

Model retrieval. Transport modelling is controlled by a huge amount of plasma and tuning parameters. Tracking all changes in these parameters which are continuously varying is not easy. Therefore, a problem of reproducing a simulation run performed a while ago could be quite messy. To this end, Astra review file includes also a complete information about the model and tuning parameters for every saved run.

The model file and the corresponding start (`.log`) file can be recovered with the command

```
Review -r file_name
```

Here `file_name` is the post-view file name. The retrieved model and `.log` files will be created in the Astra working directory `AWD/` under the names `model.tmp` and `model.log`, respectively. Then the user will be prompted to save these files under unique names in order not to destroy them by the subsequent operations.

Note, however, that the initial data file is not stored. Therefore, if it was changed in the meantime then the exact reproduction could be not possible. A similar problem arises if the tuning parameters have been modified during the run. Although the history of parameter variation can be restored by a careful examination of the post-view file there is no automatic procedure for doing this. Moreover, this history is stored in a post-view file with some discreteness, therefore, this information still is incomplete.

References

[1] B. B. Kadomtsev and O. P. Pogutse, in Reviews of Plasma Physics ed. by M. A. Leontovich, Vol. 5, Consultants Bureau, NY-London, 1963, p.249.

[2] Yu. N. Dnestrovski and D. P. Kostomarov, International Conference on Plasma Confinement in Closed Systems, Dubna, 1969, Abstracts of Contributed Papers, Moscow, 1969, p.41.

[3] F. L. Hinton and R. D. Hazeltine, Reviews of Modern Physics, **48**, No. 2, Part I, April 1978 (239-308).

[4] G. V. Pereverzev, P. N. Yushmanov, A. Yu. Dnestrovskii, A. R. Polevoi, K. N. Tarasjan, L. E. Zakharov, ASTRA, An Automatic System for Transport Analysis in a Tokamak, Report IPP 5/42, August 1991.

[5] L. E. Zakharov and V. D. Shafranov, in Reviews of Plasma Physics ed. by M. A. Leontovich, Vol. 11, Consultants Bureau, NY-London, 1986, p. 153.

[6] L. E. Zakharov and A. Pletzer, Phys. Plasmas, **6**, No. 12, December 1999, (4693-4704).

[7] A. Polevoi, H. Shirai and T. Takizuka, JAERI-Data/Code 97-014, March 1997.

[8] E. Poli, A. G. Peeters, G. V. Pereverzev, Computer Physics Communications, **136**, 2001 (90-104).

[9] A. Esterkin, A. D. Piliya, Nuclear Fusion, **36**, No. 11, p.1501-1512, (1996). A. D. Piliya, A. N. Saveliev, JET-R(98)01, preprint JET, February 1998.

[10] M. Brambilla, RAYIC, a numerical code for the study of IC heating of large tokamak plasmas, Report IPP 4/216, Februar 1984

[11] W. A. Houlberg, K. C. Shaing, S. P. Hirshman, M. C. Zarnstorff, Phys. Plasmas **4**, 1997 (3230-3242).

[12] H. Nordman, J. Weiland, A. Jarmèn, Nuclear Fusion **30**, No. 6, 1990 (983-996).

[13] M. Kotschenreuther, W. Dorland, M. A. Beer and G. W. Hammet, Phys. Plasmas **2**, (1995) 2381-2389.

[14] R. E. Waltz, G.M.Staebler, W. Dorland, G. W. Hammet, M. Kotschenreuther and J. A. Konings, Phys. Plasmas **4**, No. 7, July 1997 (2482-2496).

[15] S.-I. Itoh, K. Itoh, M. Yagi and A. Fukuyama, Plasma Phys. Contr. Fusion **38**, 1996 (1743-1762).

[16] K. Behringer, Description of the Impurity Transport Code STRAHL, JET, 1987, Report JET-R(87)08.

[17] B. B. Kadomtsev, Sov. Journ. of Plasma Physics, **1**, 1975, p. 710.

[18] V. V. Parail, G. V. Pereverzev, Sov. Journ. of Plasma Physics, **6**, 1980, p. 27.

[19] C. S. Chang and F. L. Hinton, Phys. Fluids, **29**, (1986) 3314.

[20] S. P. Hirshman, R. J. Hawryluk, B. Birge, Nuclear Fusion **17**, No.3 (1977) 611.

[21] S. P. Hirshman, Phys. Fluids, **31** (1988) 3150.

[22] Y. B. Kim, Phys. Fluids, V.**B3**, (1991) 32050.

[23] G. M. Staebler, R. E. Waltz, J. C. Wiley, Nuclear Fusion **37**, No.3, (1997) pp.287-291.