Available online at www.sciencedirect.com
SCIENCE DIRECT®
Computer Physics Communications 159 (2004) 157–184
Computer Physics Communications
www.elsevier.com/locate/cpc

The tokamak Monte Carlo fast ion module NUBEAM in the National Transport Code Collaboration library ☆

Alexei Pankin [a,*], Douglas McCune [b], Robert Andre [b], Glenn Bateman [a], Arnold Kritz [a]

[a] *Lehigh University, Physics Department, Bethlehem, PA 18015, USA*
[b] *Princeton Plasma Physics Laboratory, Princeton University, Princeton, NJ 08543, USA*

Received 27 June 2003; accepted 17 November 2003

Abstract

The NUBEAM module is a comprehensive computational model for Neutral Beam Injection (NBI) in tokamaks. It is used to compute power deposition, driven current, momentum transfer, fueling, and other profiles in tokamak plasmas due to NBI. NUBEAM computes the time-dependent deposition and slowing down of the fast ions produced by NBI, taking into consideration beam geometry and composition, ion-neutral interactions (atomic physics), anomalous diffusion of fast ions, the effects of large scale instabilities, the effect of magnetic ripple, and finite Larmor radius effects. The NUBEAM module can also treat fusion product ions that contribute to alpha heating and ash accumulation, whether or not NBI is present. These physical phenomena are important in simulations of present day tokamaks and projections to future devices such as ITER. The NUBEAM module was extracted from the TRANSP integrated modeling code, using standards of the National Transport Code Collaboration (NTCC), and was submitted to the NTCC module library (http://w3.pppl.gov/NTCC). This paper describes the physical processes computed in the NUBEAM module, together with a summary of the numerical techniques that are used. The structure of the NUBEAM module is described, including its dependence on other NTCC library modules. Finally, a description of the procedure for setting up input data for the NUBEAM module and making use of the output is outlined.

PACS: 52.40.Mj; 52.50.Gj; 52.65.Cc; 52.65.Pp

Keywords: Neutral Beam Injection; NBI; Tokamak; Heating; Monte Carlo

1. Introduction

The National Transport Code Collaboration (NTCC) project [1,2] is developing a new approach to tokamak and stellarator transport codes. This approach facilitates code use by non-experts and eases the implementation of large multi-physics integrated models. A modular approach allows the development of plug-in modules for physics and numerical packages, to have steerable applications, and to access experimental data from local files or over

☆ Supported by U.S. DOE contracts DE-AC02-76CH03073 and DE-FG02-92-ER-5414.
* Corresponding author.
E-mail address: pankin@haven.adnc.net (A. Pankin).

0010-4655/$ – see front matter
doi:10.1016/j.cpc.2003.11.002

the Internet through a uniform interface, while still reusing legacy Fortran code internally. The NTCC project established module library standards in order to promote sharing and community ownership of modules required for predictive integrated modeling. An essential element of predictive integrated modeling involves a description of the heating mechanisms that input energy into the plasma.

Neutral beam injection (NBI) heating, ohmic heating and heating by high-frequency waves are the major heating methods used in the modern magnetic fusion experiments. NBI is the major heating mechanism at the Joint European Torus (JET), the largest tokamak device in the world, as well as at many other important tokamak fusion experiments.

The basic concept of NBI heating is straightforward. Injected energetic neutral particles (usually isotopes of Hydrogen or Helium) are captured in the target plasma by atomic physics processes—charge exchange and ionization. This leads to the creation of an energetic ion population, which is confined in the tokamak's magnetic field and subsequently transfers its energy to the bulk target plasma through Coulomb collisions [3–5].

Neutral beams are created by passing an ion beam through a neutralizer chamber; the resulting neutral atoms are injected across the strong confining magnetic field into the target plasma. Conventional positive ion-based neutral beams have particles whose energies are limited by the fall-off of charge exchange neutralizer chamber efficiency to about 100 KeV. Negative ion-based neutral beams, which use an electron stripping neutralizer, can create neutral beams of significantly higher energy, as high as several MeV. In either case, the injected neutral atoms are deposited as ions in the target plasma through impact ionizations and charge exchange processes. A large fraction of the resulting fast ions become trapped in the confinement region while the remainder leave the plasma region and hit the wall or the limiter. Depending on beam geometry and point of capture, the deposited fast ions carry a greater or lesser portion of their velocity parallel or anti-parallel to the magnetic field. Those particles with sufficient velocity along the magnetic field lines follow "passing orbits" and travel roughly parallel to the magnetic field lines. Ions with insufficient parallel velocity are mirror-trapped in the tokamak's nonuniform magnetic field and their orbits follow a characteristic banana shaped trajectory. All confined fast ions transfer their energy and momentum to thermal electrons and ions through collisions and, eventually, they become thermalized. A schematic view of the NBI system based on the negative ion source for the JT-60U tokamak is given in Fig. 1 [6]. The calculation of the complete dynamics of NBI heating requires a comprehensive computer code.

The TRANSP code [7–10] is a Princeton Plasma Physics Laboratory (PPPL) transport code that has continued to develop and advance since the early 1970s. TRANSP is one of the primary codes used in the fusion community for time dependent analysis of tokamak experimental data. An essential element in the TRANSP code is the Monte Carlo package for NBI physics. The NBI treatment includes the physics of neutral beam deposition, fast ion two-dimensional orbiting, power deposition, beam driven current and momentum transfer. The NBI treatment accounts for particle collisions, charge exchange loss and recapture, and transport of beam particles. The treatment of neutral beam injection in the TRANSP code has been tested using experimental data from numerous tokamaks and is widely acknowledged to be very accurate [10,11]. For example, Fig. 2 shows the match between measurement and TRANSP simulation for collimated profile measurements of Deuterium–Tritium neutron emission in the JET tokamak. However, the NBI coding, which was originally written in Fortran-77, was developed as an integral part of the TRANSP code and could not be transferred easily for use by other modeling codes.

In order to make the TRANSP NBI package available to the wider fusion community, the NTCC module library standards were employed in extracting the NBI coding from the TRANSP code. The NTCC standards reflect general tendencies in modern computational physics that are focused on portability issues. The new NBI module that has been developed, called NUBEAM, has been submitted to the NTCC module library.

The original TRANSP Monte Carlo fast ion code was coupled to the rest of the TRANSP code through enormous shared Fortran "common" blocks, as is typical for older Fortran codes. This is an inherently non-modular and error-prone communications method, but early versions of Fortran offered few practical alternatives.

Static analysis of the TRANSP code before extraction revealed that there were a total number of more than 6000 TRANSP common block scalar and array variables, and 1117 of these variables were used by the NUBEAM package. Approximately 370 variables were input, 365 were output, and 55 variables were both input and output.

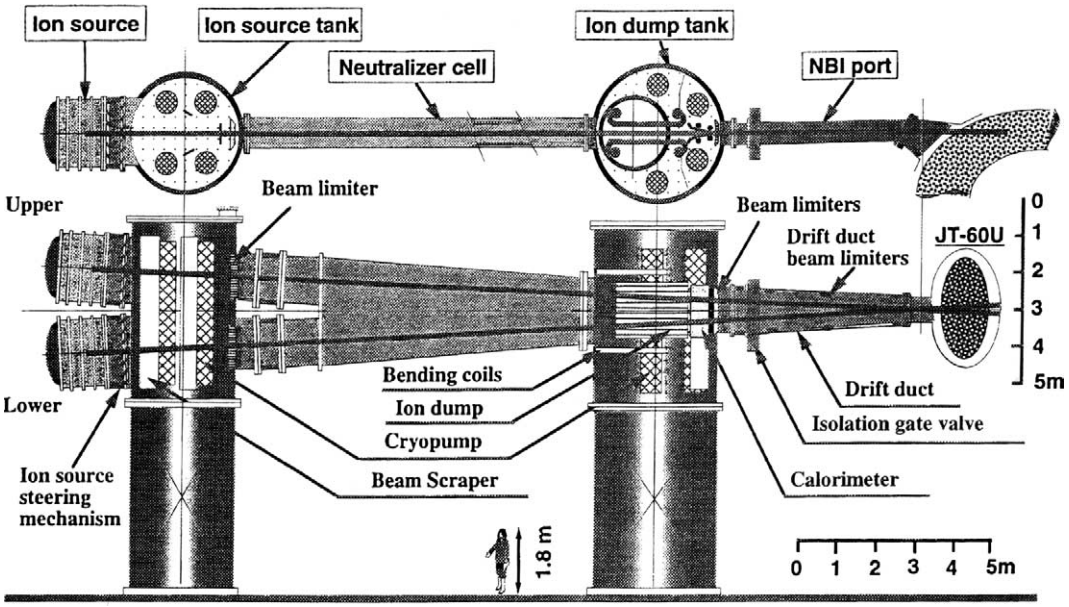

Fig. 1. Schematic view of the NBI system for the JT-60U tokamak described in Ref. [6].

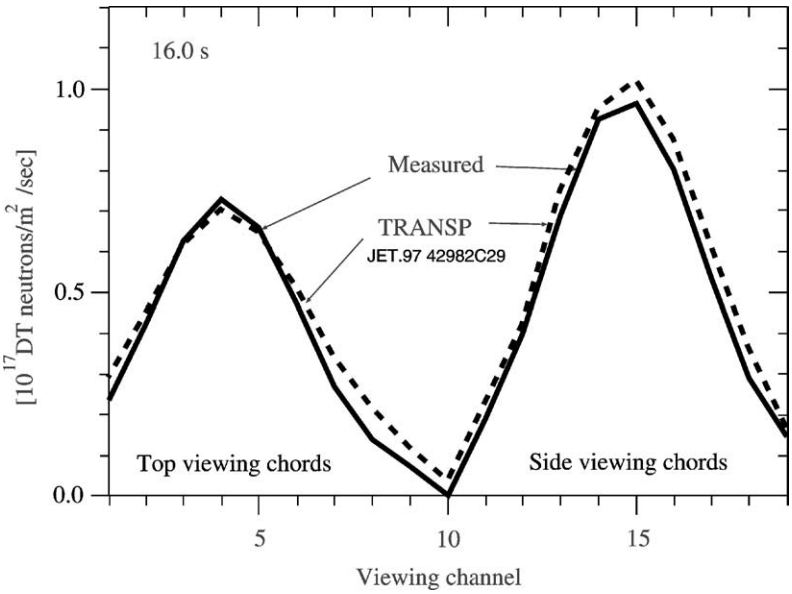

Fig. 2. Measurement and TRANSP simulation of collimated profile of Deuterium–Tritium neutron emission on JET discharge 42982 Ref. [11].

The remaining 327 variables were found to be internal to NUBEAM and not needed by the rest of the TRANSP code. The number of Fortran subroutines in NUBEAM was more than 250. The physical meaning of all the external variables has been analyzed and each variable has been assigned to one of a number of Fortran-90 compound data structures. The NUBEAM code's internal communications were rebuilt around a set of Fortran-90 modules, with all external communications handled through the exchange of instances of compound data types. A Python script

code generator was developed to help with the development of the interface, greatly simplifying the procedure for making modifications. NUBEAM's internal spatial grid was made independent of that of the main TRANSP code, and the XPLASMA NTCC module was invoked to handle the resulting interpolations needed for passing profile information between the disparate grids. Finally, external dependencies that existed in the original TRANSP implementation of NUBEAM were either included in the NUBEAM module or replaced with modules available in the NTCC module library. The structure of the resulting NUBEAM module is described in this paper.

This paper is organized in the following manner. In Section 2, the underlying physics used in the NUBEAM module is described. In particular, beam deposition and fast ion orbiting are emphasized. In Section 3, the procedure for initializing the NUBEAM module is described and some necessary information is included about other NTCC modules that are used in the NUBEAM module. Section 4 contains a description of the NUBEAM module output. Conclusions are presented in Section 5.

The NUBEAM source code and associated modules are available for download at the NTCC modules library website, http://w3.pppl.gov/NTCC.

2. NUBEAM module underlying physics

The NUBEAM module contains a Monte Carlo package for time dependent modeling of fast ion species in an axisymmetric tokamak. This Monte Carlo package represents the fast ion slowing down distribution function as a discreet set of N weighted model ions selected by random processes using probabilities dictated by the underlying physics. An advantage of the Monte Carlo method is that the representation of "smoothly varying" complex physics is relatively straight-forward. A disadvantage lies in the computational cost of reducing the statistical variance or "noise" in the model results. Generally, N^2 model ions need to be followed in order to reduce the statistical variance by a factor of N. Such cost considerations make it impractical to model the entire thermal continuum by such methods. Consequently, the NUBEAM module uses the Monte Carlo method for the fast ions only. The NUBEAM module stops following ions that slow down below $(3/2)T_i$, where T_i is the temperature of the thermal ions. These ions are then considered as "thermalized" ions and are described in terms of a thermalization source function provided as an output of the NUBEAM module.

The NUBEAM module takes into account multiple beamlines, all beamline geometries, and beam composition by isotope and energy fraction. Neutral beam stopping atomic physics, including collisions with partially slowed down fast ion species, are taken into account, with an option for a neutral beam excitation correction. After deposition of fast ions in the plasma, the modeling of the slowing down includes anomalous diffusion of fast ions, the effects of large scale instabilities, the effects of magnetic ripple, and the effects of finite Larmor radius. The modeling also includes charge exchange loss and recapture of slowing down fast ions. The NUBEAM module computes the trajectories of neutral atoms and fast ion orbits. The module accounts for multiple fast ion species that can be present, either due to beam injection of energetic neutral particles or as a result of the product of nuclear fusion reactions. The 2-D beam–beam, beam–target, and thermonuclear reaction rate profiles are computed. The basic elements are described in this section. A complete listing of the elements of the NUBEAM physics is given in Appendix A.

The original NUBEAM library, developed for the TRANSP code, is described in Ref. [12]. While the basic ideas and methods used in the module remain the same, many new elements have been added and existing elements improved during the past two decades. The major new and improved elements of the NUBEAM code are described below:

(1) *Angular momentum balance and the effects of target plasma rotation*—Toroidal rotation of the thermal target plasma can effect the plasma frame energy of newly injected beam particles, thus affecting their deposition, slowing down, and beam–target fusion rates. Angular momentum transport is also an important plasma physics

research topic; the coupling of the NUBEAM model's angular momentum source terms to angular momentum transport equations is described in detail in Ref. [13].

(2) *Generalized, time varying axisymmetric numerical MHD equilibrium and fields*—Slowing down beam ions carried from a prior NUBEAM timestep re-enter orbiting in a changed plasma field configuration. The ion guiding center positions are reconstructed on the same toroidal flux surface as close as possible to the guiding center's final $[R, Z]$ position in the prior NUBEAM timestep. Then, the guiding center's velocity vector is reconstituted by asserting conservation of magnetic moment, $v_\perp^2/B$, and canonical momentum, $P_\phi = -m_b R v_\phi - q_b \Psi/c$, where m_b and q_b is the mass and charge of particle species b, and Ψ is the poloidal flux. This leads to adjustments of the ion's kinetic energy and momentum, usually small, which mimic the effects of adiabatic magnetic (de)compression.

(3) *Finite Larmor radius* (*FLR*) *corrections*—Deposition, charge-exchange, and recapture involve FLR steps from an actual particle position to a guiding center, and back again. Also, for each evaluation of the atomic physics and collision operators on slowing down ions, as well as beam–target fusion rates, a random gyrophase angle is taken to determine the direction of the Larmor displacement from the ion guiding center to the point of interaction with the target thermal plasma.

(4) *Fusion product species, with simultaneous treatment of multiple fast species*—Self-consistent treatment of the fusion product ions allows for the simulation of alpha particle effects, which can be potentially important for 'next-step' tokamaks, such as ITER. The resulting fast alpha particle profiles from the NUBEAM module were recently used as input to codes that calculate the effects of the toroidal Alfvén eigenmode instability, MHD stability and micro-turbulence [14].

(5) *Anomalous diffusion, sawtooth mixing, and toroidal field ripple transport models*—The modeling of these physical phenomena have also been improved.

Particular features of the physics included in the NUBEAM module, with the emphasis on aspects not described elsewhere, and in particular not described in reference [12], are summarized in the subsections below.

2.1. Beamline geometry

All of the parts of a neutral beam injector that are located outside of the tokamak (or magnetically confined plasma device) vacuum vessel are referred to as a "neutral beamline". A neutral beamline consists of the following parts: a plasma discharge that provides a source of ions; a series of electrically charged grids that accelerate the ions and focus them into a beam; a large neutralization chamber designed to convert as much of the ion beam as possible into a neutral beam; strong magnets and a beam dump to remove any fast ions that remain in the beam; and finally, valves and a flange for attaching the beam line to the vacuum vessel. In the NUBEAM module, each beamline is represented by a probability weighted randomly chosen collection of neutral tracks characterized by parameters that are input to the NUBEAM module. These input parameters include the tangency radius of the beam center line relative to the axis of symmetry of the tokamak; the height and angle of the beam center line relative to the midplane of the tokamak; the size and shape of the ion source grid, the height and width of the aperture; and the vertical and horizontal divergence and focal length of the beam. These parameters constitute an "engineering description" which must be provided separately for each beamline on a tokamak experiment. These parameters are important and should be accurately specified.

The formation of each beam starts with the extraction of a beam from the ion source [15]. Most of the neutral beam injection systems use positive ion sources. Since the cross-section for charge exchange of positive hydrogenic ions is small for energies larger than 100 keV/amu, the energy of the ion is limited by the neutralization efficiency. The energy of beam species can be increased to more than 1000 keV/amu by using negative ion sources. The ions extracted from the plasma source consist of atomic ions mixed together with diatomic and triatomic molecular ions. After these ions are accelerated to the uniform (full) energy, each nucleus in each diatomic molecular ion has half of the full energy and each nucleus in each triatomic molecular ion has one third of the full energy. The input

parameters for each beamlet in the NUBEAM module include the magnitude of the full energy of the beam ions, the fractions of current (i.e., the number of ions per second) in the full, half, and third energy components of the beamlet, and the power injected through the vacuum wall of the tokamak as a function of time.

The other components of the NUBEAM module are used to compute the physical processes that occur as the fast neutrals pass through and interact with the magnetically confined plasma. These physical processes are described in the next four sections.

2.2. Beam deposition

Techniques, such as the Monte Carlo method [16] or the pencil-beamlet method [17,18], have been developed for the calculation of beam deposition profiles within the magnetically confined plasma. The NUBEAM module uses the Monte Carlo method. While the pencil beamlet method is computationally faster, the Monte Carlo method provides a more detailed treatment of the physics. "Monte Carlo" particles that represent ensembles of physical particles with similar velocities are introduced [12]. The number of physical particles represented by each model particle is denoted by its "weight". Experience using the module within TRANSP has shown that NUBEAM will calculate reasonably well behaved particle and energy source profiles if a statistical ensemble of a few thousand weighted Monte Carlo particles per fast ion species is maintained throughout the simulation.

The process of maintaining a fixed number of Monte Carlo particles per fast ion species is known in NUBEAM as "constant census", which works as follows. The physical quantities, N_{inj}, the physical number of newly injected particles during timestep $[t, t+\Delta t]$, and N_{old}, the physical number of unthermalized ions circulating in the plasma at time t, are defined from the beam input data and the results of the simulation up to the current time:

$$N_{\mathrm{inj}} = \int_{t}^{t+\Delta t} \left[I_b\left(E_0, \hat{t}\right) + I_b\left(E_0/2, \hat{t}\right) + I_b\left(E_0/3, \hat{t}\right) \right] \mathrm{d}\hat{t} \quad \text{and} \quad N_{\mathrm{old}} = \int_{V} n_b(\rho, t)\, \mathrm{d}V, \tag{1}$$

where Δt is the timestep, $I_b(E,t)$ is the input neutral beam current (particles per second) as a function of energy E and time t, and E_0 is the full energy of a particle in the beam. The numerical control, $\mathcal{N}_{\mathrm{tot}}$, the "constant census" number of Monte Carlo particles to maintain per fast ion species (control input to NUBEAM), is provided by the user. Then, the following target Monte Carlo census numbers are defined:

$$\mathcal{N}^*_{\mathrm{inj}} = \frac{\mathcal{N}_{\mathrm{tot}} N_{\mathrm{inj}}}{N_{\mathrm{inj}} + N_{\mathrm{old}}} \tag{2}$$

and

$$\mathcal{N}^*_{\mathrm{old}} = \frac{\mathcal{N}_{\mathrm{tot}} N_{\mathrm{old}}}{N_{\mathrm{inj}} + N_{\mathrm{old}}}, \tag{3}$$

the number of new Monte Carlo particles to inject, and the number of old Monte Carlo particles to retain, respectively. Due to time variation of input power and the time variability of loss processes, the quantity $\mathcal{N}_{\mathrm{old}}$, the actual number of unthermalized Monte Carlo ions stored from the previous timestep, will not precisely match $\mathcal{N}^*_{\mathrm{old}}$. Therefore, the quantity

$$\mathcal{N}_{\mathrm{adjust}} = \mathcal{N}^*_{\mathrm{old}} - \mathcal{N}_{\mathrm{old}} \tag{4}$$

is computed, representing the necessary adjustment to the Monte Carlo ensemble representing the old ions. If $|\mathcal{N}_{\mathrm{adjust}}|$ exceeds a certain threshold ($\mathcal{N}^*_{\mathrm{inj}}/5$), then, Russian Roulette or Splitting are used to subtract or add $\mathcal{N}_{\mathrm{adjust}}$ particles to the Monte Carlo ensemble representing the surviving population, with adjustment of weight so that N_{old} is conserved, and then $\mathcal{N}^*_{\mathrm{inj}}$ new particles are deposited. If the threshold is not exceeded, the surviving population

is not adjusted, and $\mathcal{N}_{\rm inj} = \mathcal{N}_{\rm tot} - \mathcal{N}_{\rm old}$ new particles are deposited. In either case, the number of particles after deposition is the constant census $\mathcal{N}_{\rm tot}$.

The only time that the Monte Carlo "constant census" is not maintained, is when the new particle source goes to zero—for example, after the shut-down of neutral beam heating late in a discharge. During this end phase of the simulation, the Monte Carlo population is allowed to decline at the natural rate dictated by thermalization and/or other loss processes, until $\mathcal{N}_{\rm old}$ reaches zero, signaling the end of the simulation.

New beam ions are injected into the simulation with energies assigned by a random process consistent with the measured beam voltage and full, half, and one-third energy fractions. If multiple beams are present, their representation in the Monte Carlo deposition process is proportional to the measured total power on each beam line.

In NUBEAM, $\mathcal{N}_{\rm tot}$ for beam ions is denoted as `nptcls`; $\mathcal{N}_{\rm tot}$ for fusion product ions is denoted as `nptclf`; these are set by user input. The amount of computer time used by NUBEAM is very nearly proportional to the number of Monte Carlo model ions $\mathcal{N}_{\rm tot}$ used. Normally, NUBEAM assigns equal weight to each newly deposited Monte Carlo particle in each timestep. However, situations can arise where it is desirable to modify this behavior. For example, in simulations with high beam ion density fraction n_b/n_e, a Monte Carlo fluctuation in the central density might cause the unphysical result $n_b/n_e > 1$. Since the central radial zones represent smaller targets for Monte Carlo orbit trajectories, it could be desirable to increase the number of Monte Carlo particles representing n_b in the core region, at the expense of the number of particles representing the fast ions in the edge region, without modifying the constant census $\mathcal{N}_{\rm tot}$. The NUBEAM control `wghta` allows this adjustment to be made. If `wghta=1` (the default) is set, there is no radial adjustment of weight. As the value of `wghta` is increased (to a maximum of n-zones, the user-selected number of radial bins in the Monte Carlo model), an ever larger number of Monte Carlo particles each of reduced weight are used to represent the population in the central region, while corresponding fewer particles of increased weight are used for the edge region. Statistical variance in model results are accordingly reduced (increased) depending on their relative dependence of model populations in the core (edge) regions. The effects of `wghta` adjustment on a test case, which is based on a DIII-D discharge at 1.95 seconds, is shown in Table 1. The table shows the average values and root-mean-square (rms) variations of number of physical, N and Monte Carlo, $\mathcal{N}$ fast ions with orbit averaged flux radial coordinate, $\hat{\rho}$, in the intervals $[0, 0.1]$ and $[0.75, 1]$, the central Deuterium beam ion density, and the beam electron heating, integrated over the edge region $0.75 \leqslant \hat{\rho} \leqslant 1$.

Evaluation of the beam deposition takes into account the expected full range of atomic processes affecting beam stopping in a hot target plasma. For a beam neutral that has atomic number A_0, energy E_0, and velocity $\vec{v}_0$, the

Table 1
The effects of `wghta` adjustment

	`wghta=1`		`wghta=20`	
	average value	rms deviation	average value	rms deviation
$\mathcal{N}\|_{\hat{\rho}\leqslant 0.1}$	111	9%	251	5%
$N\|_{\hat{\rho}\leqslant 0.1}$	1.3×10^{18}	9%	1.4×10^{18}	6%
$\mathcal{N}\|_{\hat{\rho}\geqslant 0.75}$	180	9%	93	11%
$N\|_{\hat{\rho}\geqslant 0.75}$	2.2×10^{18}	8%	2.3×10^{18}	11%
$n_b(0)$ (cm^{-3})	6.4×10^{12}	5%	6.5×10^{12}	4%
$\int_{V(\hat{\rho}\geqslant 0.75)} P_{be}(\hat{\rho})\,\mathrm{d}V$ (W)	1.9×10^{5}	5%	1.9×10^{5}	8%

Data based on run to equilibrium against a fixed target plasma ($n_e(0) = 6.5 \times 10^{13}$ cm^{-3}, $T_i(0) = 8$ keV, $T_e(0) = 4$ keV, total injected power is 9.2 MW, full energy of injected Deuterium neutral beam atoms is 80 keV). 4000 Monte Carlo ions were used; the vast majority have $0.1 < \hat{\rho} < 0.75$. After the beam ion distribution equilibrated, the run was extended for 15 0.01 second timesteps. The table values show the average value and rms variation of the indicated quantities from these 15 timesteps. The quantities $\mathcal{N}$ and N are measured in each NUBEAM timestep after deposition, but before orbiting. The physical quantities $n_b(0)$ and the P_{be} integrated in the edge region are the results of sums of all particle orbit trajectories through the indicated regions over the duration of the timestep.

expectation value of the flight time is

$$\tau_{\mathrm{fl}} = \left[\sum_j n_j \langle \sigma_j v_{\mathrm{rel}} \rangle \right]^{-1}. \tag{5}$$

The summation is performed over all the j beam stopping reactions; n_j is the density of the charged particle population driving the reaction; σ_j is the cross-section of the reaction; $v_{\mathrm{rel}} = |\vec{v}_i - \vec{v}_0|$; $\vec{v}_0$ is the beam neutral particle velocity; $\vec{v}_i$ is the charged particle velocity; and

$$\langle \sigma_j v_{\mathrm{rel}} \rangle >= \frac{\int_{\vec{v}_i} \mathrm{d}\vec{v}_i \, \sigma_j(v_{\mathrm{rel}}) v_{\mathrm{rel}} f(\vec{v}_i)}{\int_{\vec{v}_i} \mathrm{d}\vec{v}_i \, f(\vec{v}_i)} \tag{6}$$

yields the appropriate target species distribution averaged reaction rate coefficient for the jth stopping reaction.

The stopping reactions included in the model are:

(1) electron impact ionization,
(2) thermal ion charge-exchange,
(3) thermal ion impact ionization,
(4) high-Z thermal ion stopping (impact ionization and non-neutralizing charge exchange),
(5) charge exchange with slowing down fast ions,
(6) impact ionization on slowing down fast ions.

Electron impact ionization is approximated using the Maxwellian averaged rate coefficients from standard tables (p. 27 in Ref. [19]), using the adjusted electron temperature

$$T_e^* = T_e + 2/3(m_e/m_p)(E_0/A_0). \tag{7}$$

The required electron impact ionization data tables are supplied by the NTCC PREACT module.

For the thermal ion interactions (Hydrogen–Hydrogen, Hydrogen–Helium, and Helium–Helium), ground-state charge exchange cross-sections are used from the ORNL "red book" [20]. The beam–target Maxwellian averages are pretabulated on a grid to represent the two-dimensional functions $\langle \sigma_j v_{\mathrm{rel}} \rangle (E_0/A_0, T_j/A_j)$ by fast piecewise bilinear interpolation, again using the NTCC PREACT module. Here, T_j and A_j are the target species temperature and atomic number, respectively, and the neutral energy E_0 is taken in the reference frame of the flowing thermal plasma, for which NUBEAM allows a toroidal angular velocity to be specified.

For the impurities, approximate expressions for $\langle \sigma_j v_{\mathrm{rel}} \rangle$ are taken for Carbon and Oxygen from the standard ORNL "red book" [21], and a Z-scaling is applied for other impurity species; this data is also available in the NTCC PREACT module. The impurity stopping data has known deficiencies [22], and a collaboration with the Oak Ridge atomic data center is under way to upgrade the available cross sections, which will be reflected in a future version of PREACT.

For the interactions with slowing down fast ion species, an explicit Monte Carlo integral of $\langle \sigma_j v_{\mathrm{rel}} \rangle$ is calculated during Monte Carlo orbiting, and is retained from the preceding timestep of the NUBEAM calculation for subsequent use by the deposition model. The deposition model is used to compute the representative average velocity vector $\vec{v}_0$ in each zone on a two-dimensional spatial grid, separately for the ingoing and outgoing trajectories of each energy component of each beamline. Each such portion of the injected beam neutral population is treated as mono-energetic, for which a separate $\langle \sigma_j v_{\mathrm{rel}} \rangle$ Monte Carlo sum is evaluated. These sums yield the appropriate averaged rate coefficients for "beam–beam" stopping by charge exchange and impact ionization, which are important terms affecting deposition in many beam heated tokamak experiments at lower densities, where the beam ion density is 10% or more of the electron density.

The average flight time τ_{fl} for a beam neutral is a function of the neutral's position in the target plasma. For reasons of efficiency, all rate coefficients for thermal plasma and beam–beam stopping reactions are pretabulated

for all neutral energies at each plasma zone. The NUBEAM's Fortran-90 module, `nbatom_mod`, collects these one-dimensional lookup tables, which are faster to use than the two-dimensional tables provided directly by the general purpose atomic physics NTCC PREACT module.

For any given beam neutral trajectory with velocity v_0, the probability of "shine-through" loss is

$$P_{\text{shine}} = \exp\left(-\int_{\mathcal{L}} \frac{\mathrm{d}l/v_0}{\tau_{\text{fl}}} \right), \tag{8}$$

where the integration is performed over the entire beam path $\mathcal{L}$, while the probability of the neutral surviving to distance L along its flight path is

$$P(L) = \exp\left(-\int_0^L \frac{\mathrm{d}l/v_0}{\tau_{\text{fl}}} \right). \tag{9}$$

In order to generate the initial condition for a Monte Carlo ion from a sample beam neutral trajectory, a random choice of the deposition point is taken from a perturbed probability distribution, which incorporates a statistical adjustment variable, `wghta`, allowing the model to generate, if desired, a greater number of Monte Carlo ions of reduced weight for improved statistics in the core region at the expense of the edge.

It should be noted that the atomic physics data provided by the NTCC PREACT module supports only a ground state model for deposition. In high density target plasmas, multi-step ionization can be significant. Multi-step ionization occurs when an initial collision puts the neutral beam atom's electron into an excited state, and a subsequent collision completely removes the electron before it has a chance to decay back to the ground state. NUBEAM supports an excited states correction [19].

The beam neutrals that are involved in ionization and charge exchange collisions determine the particle source rates of ions, electrons and thermal neutral atoms, and they contribute to charge exchange loss rates for thermal ions. These source/sink profiles are accumulated both as flux surface averages and as poloidally resolved 2-D profiles, available as code output, which can be used as input for transport equations and which can provide data for a thermal neutral gas transport model.

In the case of deposition of fusion product ions, beam–beam and beam–target fusion rates from the previous timestep are added to thermonuclear rates from the current timestep, to determine the spatial distribution of the fast ion source. In a simplification of the physics, the rest frame of all fusion reactions is assumed to be coincident with the rotating thermal plasma reference frame, so that the fusion product ion source is isotropic and mono-energetic in the plasma frame (e.g., 3.5 MeV for DT alphas). As with neutral beam deposition, a "constant census" algorithm is used: a population of `nptclf` Monte Carlo model ions of roughly equal weight is maintained for each fusion product species. Splitting or Russian Roulette are used as necessary so that the number of newly deposited Monte Carlo model ions relative to the number continuing from earlier timesteps will match the physical ratio of new fusion product ions deposited relative to the total number of slowing down fusion product ions. The `wghta` profile statistics control for beam ions is not available for fusion product ions.

2.3. Fast ion orbits

The Monte Carlo initial conditions for each orbiting ion follow from a finite Larmor radius step that is displaced from the ion's deposition point. The orbits of beam ions are advanced by solving the guiding center drift orbit equations using standard techniques [23]. For reasons of numerical performance and convenience, the method was generalized to allow the use of magnetic coordinates, in which the calling code provides the plasma geometry and fields; a numerical Jacobian, $\mathcal{J}$, is generated, rather than requiring a coordinate system in which $\mathcal{J}$ is proportional to $1/B^2$. The drift orbit equations are also generalized to incorporate a radial electrostatic field, which is generally present in the case of toroidally rotating plasmas.

The effects of drag, energy diffusion, and pitch angle scattering are taken into account using alternating time steps in the drift equations with a "collision operator" based on Fokker-Plank coefficients that characterize the relevant rates [12,24]. A toroidal electric field imparts a modest acceleration to the orbiting ions.

The collision operator is evaluated in the frame of reference of the toroidally rotating plasma. Strong toroidal rotation can significantly affect the plasma frame deposition energy of beam particles, affecting their slowing down times and the splitting of beam heating between thermal ions and electrons. Heating profiles, torques, and thermalization sources (particles, momentum, energy), as well as beam–target fusion rates are also evaluated in the collision operator. The beam–target fusion rates $\langle \sigma v_{\mathrm{rel}} \rangle$ are taken from the Maxwellian averaged beam-target rate tables in the NTCC PREACT module, based on standard fusion cross-sections [25]. The use of lookup tables of Maxwellian averaged beam–target fusion rate coefficients reflects the same method applied in the treatment of beam-target atomic physics for neutral beam deposition.

During orbiting, Monte Carlo sums are also used to calculate the fast ion density, average toroidal angular velocity, and average energy densities parallel to and perpendicular to the magnetic field, for each fast ion species. In fact, an entire 4-dimensional phase space fast ion distribution $f(E, v_{\parallel}/v, \rho, \theta)$ is calculated by Monte Carlo summation. This fast ion distribution function uses a spatial (ρ, θ) grid, described further in Section 3.2. The spatial grid, representing magnetic coordinates, has an evenly spaced set of ρ zone "rows", with a variable number of θ zones per row: fewer θ zones near the axis, more toward the edge. Typically, 10 or 20 zone rows are used; enough θ zones are used to give similar spatial resolution in both the radial and the poloidal directions. For the energy grid, the default numbers for pitch angle $v_{\parallel}/v$ and energy E grid zones are 50 and 100 correspondingly. For beam ions, the maximum energy for the energy grid is selected to be larger than beam injection energies in order to allow energy diffusion and is typically in the range 120–160 keV. For fusion products, the maximum energy is set as follows:

- fusion Tritons: 1.25 MeV
- fusion He3: 1.00 MeV
- fusion Alphas: 5.00 MeV

which are safely above the birth energies for common plasma fusion reactions. The spatially two-dimensional distribution function $f(E, v_{\parallel}/v, \rho, \theta)$ is used to compute the fusion reaction rate between fast ions and is also retained for simulations of diagnostics based on chord integrals, such as charge exchange flux spectra. The number of Monte Carlo particles necessary to produce reasonably smooth fast ion densities and heating profiles, is generally not sufficient to produce a locally well-behaved Monte Carlo summed fast ion distribution function. Since reduction in statistical variance scales only as the square root of the number of Monte Carlo particles (and hence computer time) used, it is expensive to reduce the local variance. Nevertheless, chordal and volume convolutions over noisy Monte Carlo distribution functions have been shown to work well.

The accumulation of Monte Carlo fast ion distribution functions during orbiting allows NUBEAM to perform the retrospective evaluation of the fusion reaction between fast ions, by convolving the distribution functions in a Monte Carlo integral. Based on the standard fusion cross-sections [25], the NTCC PREACT module provides the necessary gyro-averaged fusion rate coefficients $\langle \sigma v_{\mathrm{rel}} \rangle$ for this calculation.

During the slowing down of fast ions, an atomic physics operator is evaluated to model fast ion charge exchange losses. The atomic physics operator is also used to compute fast-ion beam stopping coefficients for the next deposition timestep, as well as profiles of charge exchange and impact ionization rate coefficients for thermal energy background neutrals. The latter profiles are important because they play a dominant role in core thermal neutral gas transport.

Direct integration of the ion drift equation contributes substantially to the computational burden in the NUBEAM module because the characteristic times of the fast ion orbiting and of ion thermalization are very different (the ion orbit bounce time is usually many orders of magnitude shorter than the ion slowing down time). Consequently, a number of acceleration techniques are used in the NUBEAM module. In particular, the numerical

control variables, `fppcon` and `cxpcon`, are introduced. The first variable, `fppcon`, indicates the number of collision operator evaluations to perform per orbit half-bounce. The second variable, `cxpcon`, indicates the number of atomic physics evaluations per orbit half-bounce. During the evaluation of the collision and atomic physics operators, characteristic slowing down, pitch angle scattering, energy diffusion and charge exchange loss times are estimated, based on an average over each orbit half-bounce. The ratio of the shortest of these characteristic physical timescales to the orbit bounce time is compared to the numerical control variable `goocon`, which specifies the number of orbit bounces to evaluate per shortest characteristic competing timescale. A "numerical" orbit timestep multiplier, `goose`, is set accordingly, to artificially accelerate the collision and charge exchange operators so that only `goocon` orbits need be evaluated per selected characteristic physical timescale. This reduces, by two to three orders of magnitude, the number of orbit timesteps that are necessary to complete each model calculation, without significantly affecting the results.

The procedure for updating `goose` during the fast ion slowing down process has been generalized. The traditional NUBEAM procedure measures orbit half bounces by successive midplane crossings. However, it has been recognized that there exist tokamak field equilibria which support classes of orbits that never reach the plasma midplane. For example, spherical tokamak equilibria can have off-midplane local field minima, in which ions with low parallel velocity can be trapped. Even in conventional aspect ratio tokamaks where each flux surface field minimum occurs at or near the outer flux surface midplane intercept, collisional orbits near the trapped passing boundary can travel for long periods of time without a midplane crossing. Therefore, a "backup" goose update mechanism has been introduced in the code. This is based on a zero-banana-width estimate of expected bounce time for trapped orbits, localized to each flux surface. If, at any time, an orbit's actual travel time (since last goose update) exceeds more than twice the zero-banana-width bounce time estimate, the ion's goose factor is updated immediately, without waiting for the next midplane crossing; the goose factor is also reduced by a factor of two, to empirically compensate for the relative infrequency of updates of the goose factor, and, the midplane goosing controls (which require two successive crossings) are re-initialized.

The details are as follows. On each flux surface, ρ, $B_{\min}$ and $B_{\max}$ are found. For a given trapped orbit on this surface, B_{refl}, between $B_{\min}$ and $B_{\max}$, is the "reflection point" where $v_\perp = v$ and $v_\parallel = 0$. A handful of B_{refl} values are chosen as a basis for numerical evaluation of the function $\tau_b(\rho, B_{\mathrm{refl}})$ using the integral

$$\tau_b = 2 \int_{\theta^-(B_{\mathrm{refl}})}^{\theta^+(B_{\mathrm{refl}})} \mathrm{d}\theta \frac{\mathrm{d}l_p}{\mathrm{d}\theta} \frac{B}{B_p} \frac{1}{v_\parallel} \tag{10}$$

where l_p is the poloidal path length, B/B_p is the ratio of total magnetic field to poloidal component of magnetic field, and $\theta^-(B_{\mathrm{refl}})$ and $\theta^+(B_{\mathrm{refl}})$ are, respectively, the θ values nearest to $\theta(B_{\min})$ satisfying $B(\theta, \rho) = B_{\mathrm{refl}}$. These values are located numerically with a root finder.

In the zero-banana-width approximation, v and magnetic moment $v_\perp^2/B$ are constants of motion. For a collisionless orbit these considerations lead to

$$v_\parallel = v\sqrt{1 - \frac{v_\perp^2}{v^2}} = v\sqrt{1 - \frac{B}{B_{\mathrm{refl}}}}. \tag{11}$$

This allows the v dependence to be factored out of the above integral, so that

$$\tau_b = 2\frac{L_b}{v} \quad \text{and} \quad L_b = \int_{\theta^-(B_{\mathrm{refl}})}^{\theta^+(B_{\mathrm{refl}})} \mathrm{d}\theta \frac{\mathrm{d}l_p}{\mathrm{d}\theta} \frac{B}{B_p} \frac{1}{\sqrt{(1 - B/B_{\mathrm{refl}})}} \tag{12}$$

which only needs to be evaluated at a handful of representative values of B_{refl} on each flux surface. The integrand is singular at the end points, but the singularity is integrable unless $\mathrm{d}B/\mathrm{d}\theta$ is zero at the end points, which happens

classically, e.g., at $B_{\rm refl} = B_{\rm max}$, corresponding to the "usual" zero-banana-width trapped-passing boundary. But in the general case, for flux surfaces with multiple local field maxima, multiple non-integrable L_b values can arise.

Two measures are taken to avoid the singularities. First, in the numerical evaluations of L_b integrals, $(1 - B/B_{\rm refl})$ is replaced by $\max(10^{-8}, (1 - B/B_{\rm refl}))$. Second, the representative $B_{\rm refl}$ values are chosen at

$$\begin{aligned}
&B_{\rm refl}(1) = B_{\rm max} - \delta(B),\\
&B_{\rm refl}(2:3) = B_{\rm min} + 0.75(B_{\rm max} - B_{\rm min}) \pm \delta(B),\\
&B_{\rm refl}(4:5) = B_{\rm min} + 0.50(B_{\rm max} - B_{\rm min}) \pm \delta(B),\\
&B_{\rm refl}(6:7) = B_{\rm min} + 0.25(B_{\rm max} - B_{\rm min}) \pm \delta(B),
\end{aligned} \tag{13}$$

where $\delta(B) = (B_{\rm max} - B_{\rm min})/100$. Then, $L_b(B_{\rm refl}(1))$ is evaluated and assigned as $L_b^*(B_{\rm max})$; $\min(L_b(B_{\rm refl}(2)), B_{\rm refl}(3))$ is evaluated and assigned as $L_b^*(B_{\rm min} + 0.75(B_{\rm max} - B_{\rm min}))$, and similarly for $L_b^*(B_{\rm min} + 0.50(B_{\rm max} - B_{\rm min}))$ and $L_b^*(B_{\rm min} + 0.25(B_{\rm max} - B_{\rm min}))$. The choose of the minimum of the evaluation of a pair of values avoids the singularity on any flux surface for which $B(\theta)$ has only a small number of local maxima, i.e., all cases as seen in practice. This procedure results in a handful of numerical integrations on each flux surface, which are very fast to evaluate.

For use during orbiting, L_b^* is organized as a function of flux surface label ρ and dimensionless coordinate x_B, defined by the relation

$$B_{\rm refl} = B_{\rm min} + x_B(B_{\rm max} - B_{\rm min}). \tag{14}$$

During orbiting, the two quantities

$$\Delta t\,(\text{orbit}) = [\text{time since last goose evaluation}]$$

and

$$B_{\rm max}(\text{orbit}) = [\text{maximum } B \text{ seen by particle since last goose evaluation}] \tag{15}$$

are tracked. Then, for a particle of velocity v on flux surface ρ,

$$x_B = \min\left(1, \max\left(0.25, \frac{B_{\rm max}(\text{orbit}) - B_{\rm min}(\rho)}{B_{\rm max}(\rho) - B_{\rm min}(\rho)}\right)\right) \quad \text{and} \quad \tau_b = \frac{2}{v} L_b^*(x_B, \rho) \tag{16}$$

are computed at very low computational cost. The goose factor is updated without waiting for the next midplane crossing, if the condition

$$\Delta t\,(\text{orbit}) > 2\tau_b \tag{17}$$

is satisfied.

When tested on a conventional tokamak test case (a TFTR supershot), fewer than one orbit in a thousand experienced goose updates is triggered by the alternative method. Even in NSTX cases with off-midplane field minima in the equilibrium, due to the beam injection geometry, Monte Carlo ion trajectories which trigger the backup goose update method are quite rare.

The overall computational overhead of supporting this alternate goose update method lies within the error bars of CPU time measurement, in brief tests, and is believed to be less than 1%.

Hypothetically, in a simulation where neutrals are injected directly into off-midplane trapped orbits, the absence of the backup goose update method could leave large numbers of highly collisionless particles orbiting ungoosed, resulting in a major loss of computational efficiency plus some accumulation of orbit integration error. However, such a case has not yet been observed in practice.

2.4. Effects of sawtooth oscillations, fishbone instabilities, and magnetic ripple

Sawtooth oscillations [26], fishbone instabilities [27–29], and magnetic ripple [30] cause perturbations of the magnetic field and distortions of the fast ions paths, which result in either rearrangement of fast ions within the plasma or ejection of the ions from the plasma region [30]. Magnetic ripple is the non-axisymmetric variation in the strength of magnetic field and the non-axisymmetric displacement of magnetic field lines around the toroidal circumference of tokamak as a result of the discrete toroidal field coils. Fishbone and sawtooth oscillations are both large scale MHD instabilities in tokamak plasmas. Fishbone instabilities causes bursts of fast ion losses, while sawtooth oscillations generally mix fast ions within the central region of the plasma.

The NUBEAM module supports a method to model the effects of sawtooth oscillations based on the Kadomtsev mixing model [31]. If the sawtooth flags are set, to indicate that a sawtooth crash may occur, the results of Kadomtsev mixing are precomputed and stored. This includes profiles of the "potential" post-sawtooth fast ion density and energy content, which the calling code should interpolate to its own grid. If the calling code determines that a sawtooth crash has indeed taken place prior to the next NUBEAM timestep, then, NUBEAM's subroutine `sawnbi` is called. This updates the internal state of NUBEAM to reflect the occurrence of the sawtooth. Also, the calling code updates its own representations of the fast ion density and other profiles, using the data provided. It is recommended that NUBEAM timestep boundaries be synchronized (on a transport timescale) with "discontinuous" events such as sawtooth crashes, as well as plasma fueling by neutral pellet injection.

The NUBEAM module calculates losses of fast ions due to fishbone oscillations and magnetic ripple [32]. Ions are checked for possible loss due to interactions with fishbone oscillations when they cross the plasma midplane. If the time corresponds to a fishbone event, and if the ion parameters fall within the specified range, the ion is declared lost from the plasma.

A stochastic magnetic ripple loss criterion [33] is computed in the NUBEAM module for every ion at every timestep. The parameter that determines whether or not a particle has been lost is

$$\delta_s = \alpha_{\text{anom}} \left(\frac{\varepsilon}{N \pi q} \right)^{3/2} \frac{1}{\rho_g} \left(\frac{\mathrm{d}q}{\mathrm{d}\psi} |\nabla \psi| \right)^{-1}, \tag{18}$$

where $\varepsilon = r/R$ is the inverse aspect ratio, N is the number of coils, q is the plasma safety factor, and ρ_g is the Larmor ion gyro radius. This criterion can be adjusted by a user-supplied anomaly factor α_{anom}. The particle is lost if the toroidal magnetic field ripple δ at the bounce point is greater than δ_s. This model also calculates the power loss, particle loss, and momentum loss caused by magnetic ripple, and the corresponding $\vec{J} \times \vec{B}$ torque and rotational energy change of the plasma.

In practice, the ripple loss anomaly factor α_{anom} has been found for select cases by timeslice comparison of TRANSP results with results from a more detailed and time consuming TF orbit ripple loss calculation, such as ORBIT [23]. This method has been used to study beam ion and alpha ion losses in TFTR [34].

2.5. Anomalous fast ion diffusion

The NUBEAM module has an option to calculate an anomalous fast ion diffusion operator. This anomalous diffusion model can be applied to beam ions and fusion products separately, or to all the Monte Carlo particles. The calling code sets the anomalous diffusivity, which can be a time varying radial profile.

The NUBEAM code allows fast ion anomalous diffusion to be specified as a function of the fast ion energy. The method is to specify, via an input parameter list, a piecewise linear function giving the energy dependence of the fast ion anomalous diffusion coefficients. This function is independent of all other coordinates such as space, time, and the species' indices.

2.6. Finite Larmor radius adjustment

NUBEAM's guiding center drift orbit integrator computes the trajectory of fast ion guiding centers, but, actual particle positions are separated from their guiding centers by a Larmor radius. By choosing a random gyrophase angle, NUBEAM's Finite Larmor Radius (FLR) adjustment causes particle collisions, beam–target fusion, and atomic physics reactions, and fast ion loss to the limiter or wall, all to be calculated out at the particle position, rather than at the guiding center.

NUBEAM supports two FLR adjustment models. For a fast ion of atomic weight A_b and number Z_b, the "traditional" model approximates the Larmor radius

$$\rho_g = A_b m_p v_\perp / Z_b e B_g \tag{19}$$

as independent of gyrophase angle, based on the magnetic field B_g at the orbit guiding center. After selection of a random gyrophase angle, the particle displacement, normal to the guiding center field, is calculated directly in $[R, Z]$ coordinates and mapped to back to flux coordinates by a fast bilinear map. Plasma parameters are then found at the chosen particle position for the calculation of collision operator, charge exchange, etc.

While the traditional model is computationally efficient, and adequate for most conventional tokamak applications, use of a gyrophase-invariant Larmor radius is not sufficient in certain low field spherical tokamak configurations. Therefore, a "generalized" FLR model has been added to the code.

In general, the FLR effect introduces a displacement $\Delta\vec{x}$ of each fast ion within the gyro plane

$$\Delta\vec{x} = \rho_g\big[a(\vec{e}_g \cos\alpha + \vec{e}_\perp \sin\alpha) + b\,\vec{e}_\parallel\big], \tag{20}$$

where ρ_g is the guiding center Larmor radius of the fast ion, $\vec{e}_\parallel = \vec{B}_g/B_g$ is a unit vector defining the direction of the magnetic field at the guiding center, $\vec{B}_g$ is the magnetic field at the guiding center in contrast with $\vec{B}_\ell$, which is the magnetic field at the location of a fast ion (gyro point ℓ), $\vec{e}_g$ is the unit vector normal to the flux surface at the guiding center, $\vec{e}_\perp = \vec{e}_g \times \vec{e}_\parallel$, α is the gyro angle that defines a plane within which the gyro point ℓ is located, and a and b are coefficients that describe the displacement caused by the change of magnetic field within a Larmor radius. If the change of magnetic field within a Larmor radius is small, $\delta B/B \ll 1$, or in other words, the Larmor radius ρ_g is much smaller than the characteristic scale length of the magnetic field gradient L_B, $\rho_g \ll L_B$, the displacement from the guiding center is based on the magnitude and direction of the $\vec{B}_g$ field at the guiding center and the coefficients a and b go to 1 and 0 correspondingly. These settings are inherent to the traditional FLR model in the NUBEAM module and are used by default. These assumptions for the magnetic field are not valid for discharges with high β or low aspect ratio, where the magnetic field can be small inside the diamagnetic well and orbit losses are overestimated [35]. The new FLR model uses the displacement from the guiding center, which is based on the magnitude and direction of the magnetic field at the gyro point, instead of on the magnetic field at the guiding center. Then, the coefficients a and b are to be found to satisfy the energy and canonical angular momentum conservation conditions in the axisymmetric (R, Z) coordinate system:

$$v_\ell^2 = {v_g}^2 - \frac{2q_b}{em_b}(\Phi_\ell - \Phi_g), \tag{21}$$

$$v_{\varphi_\ell} = \frac{R_g}{R_\ell} v_{\varphi_g} + \frac{q_b}{m_b R_\ell c}(\psi_\ell - \psi_g), \tag{22}$$

where the indexes g and ℓ represent the gyro center and gyro point correspondingly, $R_{g,\ell}$ is the radial coordinate, $\psi_{g,\ell}$ is the poloidal flux, $\Phi_{g,\ell}$ is the electrostatic potential, $\vec{v}_{g,\ell}$ is the velocity of the particle, and $v_{\varphi_g} = (\vec{v}_g \cdot \vec{e}_\parallel) B_{\varphi_g}/B_g$.

Assuming that the displacement $\Delta\vec{x}$ is orthogonal to the magnetic field $\vec{B}_\ell$ at the gyro point, one can show that the function $V(a,b)$ defined as

$$V(a,b) \equiv \left\| \frac{Z_b e B_\ell}{mcv_\ell}(\Delta\vec{x} \times \vec{e}_\parallel) \right\|^2 + \frac{1}{(\hat{\varphi}_\ell \cdot \vec{e}_\parallel)^2}\left(\frac{v_{\varphi_\ell}}{v_\ell} - (\Delta\vec{x} \times \vec{e}_\parallel) \cdot \hat{\varphi}_\ell \right)^2, \tag{23}$$

is unity at the gyro point

$$V(a,b) = 1. \tag{24}$$

The latter equation together with the orthogonality condition for the displacement $\Delta\vec{x}$

$$X(a,b) \equiv (\Delta\vec{x} \cdot \vec{e}_\parallel) = 0 \tag{25}$$

provide the complete set of equation for the coefficients a and b that define the displacement $\Delta\vec{x}$ in Eq. (20). Once the displacement is calculated, the NUBEAM module checks whether or not the particle is lost to the limiter (leaves the plasma).

A two-dimensional Newton iteration method is employed to solve Eqs. (24) and (25). This requires computation of the Jacobian $(\partial(X,V)/\partial(a,b))$ within the iteration. A numerical computation of the Jacobian using central finite differences is performed. The NTCC XPLASMA module is used for computing the $\vec{B}$, ψ and Φ functions, and so the computational cost of computing the Jacobian using five points in the (a,b) plane is reduced due to the vectorized nature of the XPLASMA module. As an alternative, a two-dimensional secant method can be used, which starts with a numerically computed Jacobian, and then the Jacobian is updated along with the solution in subsequent iterations. The first iteration requires five points in the (a,b) plane but the following iterations require only a single point unless the iteration scheme needs to be restarted. There are fewer function evaluations per iteration but, because the secant method converges more slowly, there are more iterations. The secant method appears to have a slight benefit in computation time over the Newton method though it should be noted that the first step, which is the same for both the Newton and secant methods, typically takes the starting point fairly close to the final solution. For both the Newton and secant methods, the iteration scheme is started at the $(a=1, b=0)$ point, which corresponds to the point on the gyro orbit returned by the older FLR model.

In both FLR models, the guiding center of a fast ion is found at deposition time by making a Larmor displacement based on the magnetic field at the point of ionization. The guiding center is then incrementally advanced for each ion. The tracking of the high energy, large gyro radii ions present in a spherical tokamak by means of their guiding centers has been shown to be valid by Mikkelsen et al. [35].

After advancement of the guiding center for an ion, a random gyro phase is chosen and the position of the ion on the gyro orbit is computed from a Larmor displacement using the traditional FLR model or through the iteration method of the new FLR model. The space and velocity distribution of the fast ions can be computed at this point as a density in the guiding centers or a density of the ions on the gyros. Due to the random gyro phase displacement, the fast ion distribution on the gyro tends to have more Monte Carlo noise then the distribution at the guiding center.

Table 2 compares the total limiter and charge exchange power loss, beam heating power, neutron production, and Monte Carlo CPU time for two NSTX cases computed with the traditional and new FLR models. The first case corresponds to the NSTX discharge 109070, 0.4 s after beam turn on from current NSTX data. The second case is based on profiles predicted for high beta plasma with a markedly different current profile than that of 109070. The simulation of NSTX discharge 109070 shows little difference between the new and traditional FLR model, while the simulation of the high beta shot shows much smaller fast ion losses outside the plasma boundary with a corresponding increase in beam heating and neutron production. Both cases indicate a severe runtime penalty in this section of code when using the new FLR model due to the search scheme inherent in this model. The traditional FLR model is therefore generally used for new runs while the new FLR model is used for checking for possible large Larmor radius effects. The inclusion of a full orbit code into NUBEAM, which can later be parallelized, is being investigated.

Table 2
Beam heating power, total limiter and charge exchange power loss, neutron production, and Monte Carlo CPU time with new and traditional FLR models for two NSTX discharges

	NSTX 109070		NSTX high β	
	Traditional FLR	New FLR	Traditional FLR	New FLR
Limiter and charge exchange loss [W]	5.2×10^5	5.4×10^5	3.5×10^6	2.6×10^6
Beam heating power [W]	2×10^6	2×10^6	5×10^5	9×10^5
Neutron production [1/s]	4.0×10^{14}	4.0×10^{14}	1×10^{14}	3×10^{14}
Monte Carlo CPU time [hours]	0.5	4.0	0.1	0.5

3. Using the NUBEAM module

3.1. Initialization of the NUBEAM module

Almost all of the NUBEAM input and output parameters are passed to the NUBEAM module as elements of Fortran-90 compound data types. More detailed up-to-date descriptions of these structures can also be found on the Internet in the appendices of the NUBEAM User Guide on the NUBEAM webpage under the NTCC website, http://w3.pppl.gov/NTCC [2]. The input data structures are listed in Appendix B and the output data structures are listed in Appendix C. The total collection of compound data type definitions are contained in the Fortran-90 module named `nbi_types.f90`, which is imported into the user's software by employing the statement

```
use nbi_types
```

The `nbi_types` module defines data types only—it does not contain any actual data. To communicate with NUBEAM, the user's code declares an instance of each appropriate data type. When setting up input for the NUBEAM module, the following steps need to be implemented: An instance of type `nbitype_sys`, for example, is created; the corresponding NUBEAM routine `nbi_init_sys` is called to set default values for the data elements in this instance; the user's routine can modify individual elements as desired; and then the NUBEAM routine `nbi_set_sys` is called to pass the instance of the compound data type as input to the model. The following code example illustrates this process:

```
use nbi_types

type(nbitype_sys) :: zsys      ! declaration of variable zsys for
                               ! basic NUBEAM system inputs

call nbi_init_sys(zsys)        ! load defaults in zsys

! user needs to set only those data elements for which non-default
! values are desired. For example:

zsys%runid = my_runid          ! set RUNID string
zsys%nonlin = my_msg_lun       ! set Fortran unit number for
                               ! messages
zsys%nseed = my_seed           ! set random number seed

call nbi_set_sys(zsys)         ! system input parameters are passed
                               ! to the NUBEAM module.
```

This method (the use of Fortran-90 compound data types for specification of inputs) has some significant maintenance advantages. In particular, the NUBEAM developer can add new input options to the code without breaking existing user installations, provided intelligent default settings are specified.

There are some ordering and update restrictions on how inputs are presented to NUBEAM. More precisely, certain array dimensions must be given first because they control the dynamic allocation of arrays for NUBEAM's internal memory; an instance of the type `nbitype_dims` is used to set this array dimension information. Then, there are a set of inputs that can only be specified once at the beginning of a run, and cannot be modified subsequently: geometries of individual beam-lines are an example of this class of inputs. Such inputs are maintained in NUBEAM's state file; if a code is restarted, the NUBEAM module will find them in the state file and NUBEAM does not need to receive them a second time from the integrated modeling code. Finally, there are a set of input parameters, such as the powers and voltages on each beamline, which are updatable and which are expected to vary in time.

Complete definitions of the compound data types used for input are found in `nbspec.dat`, which is distributed with the NUBEAM module and is described in Appendix D. This file, which serves as input to a Python code generator script that actually builds the NUBEAM interface layer, is therefore functionally guaranteed to be up-to-date.

3.2. Use of the NTCC XPLASMA module

The NTCC XPLASMA module is a general tool for representing and sharing plasma geometry and profiles, and for interpolating the profiles between disparate grids. The XPLASMA module at present is restricted to axisymmetric configurations. The way that the XPLASMA module is used within the NUBEAM module is described in this subsection.

In setting up the call of the NUBEAM module in a new application, the plasma MHD equilibrium and various plasma parameter profiles must be loaded into the XPLASMA module. The equilibrium can either be loaded directly via the appropriate XPLASMA calls, or it can be loaded from a standard data source such as EFIT [36] or from TRANSP MDSplus trees [37,38], which are available from various tokamak experimental databases. The NTCC module I2MEX can be used to load XPLASMA with numerical axisymmetric MHD equilibria from various sources; the NTCC module TRXPLIB can load both equilibria and TRANSP profiles such as temperatures and densities into the XPLASMA module. Equilibrium and plasma parameter profiles are defined as a function of the standard flux surface label coordinate $\rho = \sqrt{\Phi/\Phi_0}$, where Φ is the toroidal flux enclosed within a given flux surface and Φ_0 is the toroidal flux within the outer plasma boundary. The flux surface coordinate ρ is a dimensionless variable ranging from 0 at the magnetic axis to 1 at the plasma boundary. In addition, XPLASMA allows equilibria to be numerically extrapolated in order to provide a nested set of nominal flux surfaces beyond the edge of the plasma at $\rho = 1$. These surfaces with $\rho > 1$ are numerical surfaces only; the relation to the physical toroidal flux does not apply (or only applies very roughly) beyond $\rho = 1$.

The XPLASMA module allows the specification of an axisymmetric wall or limiter, which is a closed path encompassing a region that includes the entire cross-section of the core plasma. This limiter or wall is used to define orbits that are lost to the plasma: a NUBEAM Monte Carlo ion is not considered lost, even if it crosses the plasma boundary, unless it comes within a Larmor radius of a physical limiter. Therefore, the NUBEAM module requires that XPLASMA be given a wall configuration, and that the equilibrium be extrapolated to give a nested toroidal flux surface system large enough to fill the rectangle $[R_{\min}, R_{\max}]$ by $[Z_{\min}, Z_{\max}]$ defined by the minimum and maximum major radius and elevation of the given wall configuration. The extrapolation can be achieved by calling the XPLASMA routine `eqm_brz` after the wall coordinates have been loaded. If the equilibrium is built directly from EFIT results that already cover the exterior vacuum region, the extrapolation is not required.

In addition, XPLASMA supports the definition of an irregular 2-D grid, illustrated in Fig. 3, that is used by various TRANSP models for fast ions. This grid consists of radial zones that are subdivided poloidally, with fewer subdivisions near the center of the plasma and more subdivisions toward the edge of the plasma, yielding a set of subzones with roughly equal cross-sectional area. This 2-D grid is created by calling the XPLASMA routine

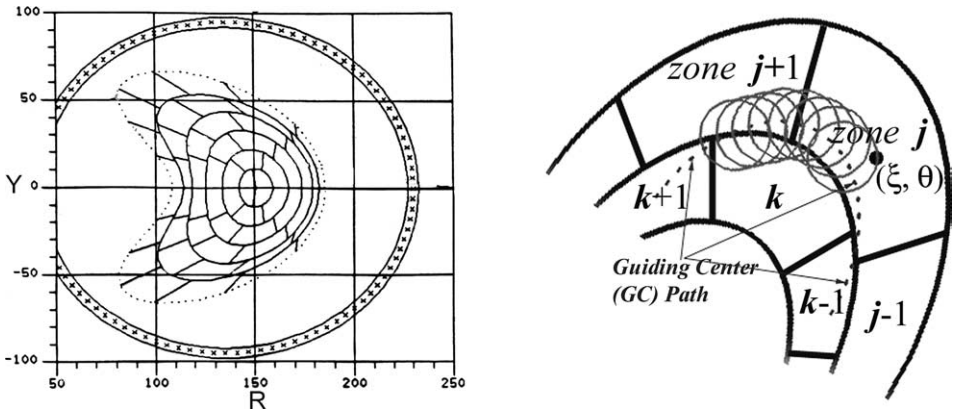

Fig. 3. Irregular 2-D grid.

`mcgrid_define`. Within each radial zone, the poloidal zones are spaced equally in the θ dimension, which is the poloidal angle coordinate implicitly defined by the supplied 2-D MHD equilibrium.

In general, XPLASMA allows for the definition of profiles in any of the forms $f(\rho)$, $g(\rho,\theta)$, $h(R,Z)$. Associated with each such profile is a unique name and a unique integer "XPLASMA ID", which is assigned automatically by the XPLASMA module. NUBEAM acquires the input profiles it needs by receiving "XPLASMA ID" integers specified as members of a compound data type (Fortran-90 type) used for input, as described in the previous Section 3.1. Almost all NUBEAM profile inputs are of the form $f(\rho)$, with ρ in the range $[0,1]$. Extrapolations of profiles beyond the plasma boundary are not required. When NUBEAM sets up a timestep within its calculation, it uses its own XPLASMA calls to remap the profiles from the caller's grid to NUBEAM's own internal grid.

Similarly, on output, the NUBEAM defines a large set of XPLASMA profiles, which are each identified by a unique integer ID. These profiles are returned to the caller as members of compound data structures, which are fetched with calls to the appropriate data retrieval routines after the NUBEAM timestep calculation is completed. The caller can then use XPLASMA interpolation routines to remap the data to the caller's grid.

There is also a set of spatially 2-D data defined on the XPLASMA irregular 2-D grid. These profiles include 2-D neutral source and neutral sink data, and the full distribution functions for each fast ion species.

The NUBEAM input/output interfaces are written by using a Python code generator. The generator input specification file is an ASCII file that is designed to be understandable by the user. It is a useful document in its own right, as is described in the previous section and in Appendix D. The python generator also writes an HTML file that fully describes the generated components of the NUBEAM input and output interfaces—both subroutines and Fortran-90 compound data types. This webpage is posted with the NUBEAM module webpage as documentation, under the NTCC website [2].

The XPLASMA module is one of the eight NTCC nodules that are used by the NUBEAM module. All eight NTCC modules are listed in Appendix E, with a brief description of each module.

3.3. Running the NUBEAM module in a transport code

A typical sequence of calls for the time-dependent use of NUBEAM fast ion simulations includes the following steps once the NUBEAM module is initialized with an initial set of control inputs and array dimensions:

(1) Initialize the XPLASMA MHD equilibrium and associated profiles;
(2) Update inputs to the NUBEAM module, such as the power injected with each beam; and "XPLASMA ID" integers for the plasma parameter profiles at the current timestep;

(3) Have NUBEAM interpolate XPLASMA profiles to its internal grid with error checking of the profile data (call NBI_INTERP_PROFILES);
(4) Initialize NUBEAM timestep and perform further error checks on the input data (call NBSTART);
(5) Compute beam and fusion product deposition (call DEPALL);
(6) Compute orbiting and slowing down (call ORBALL);
(7) Finish timestep (call NBFINISH). All output profile data are stored in the XPLASMA module. Scalar data and XPLASMA IDs can be fetched through compound data types using calls to NUBEAM's generated `nbo_get_*` routines;
(8) Get NUBEAM outputs.

An example of the Fortran-90 user code needed to advance the NUBEAM module through a timestep is given below:

```
CALL <my_xplasma_setup>   ! get XPLASMA MHD equilibrium and
                          ! profiles ready
CALL <my_NUBEAM_setup>    ! set/update inputs to NUBEAM: XPLASMA
                          ! profile ids, power injected with each
                          ! beam, etc.
CALL NBI_INTERP_PROFILES(ierr)
                          ! interpolate XPLASMA profiles to NUBEAM
                          ! internal grid. Check error code (0=OK).

! at this point all input data has been loaded into NUBEAM.

CALL NBSTART(ierr)        ! NUBEAM call-- initialize timestep.
if(ierr.ne.0) then [... implement error handling ...]

CALL DEPALL(ierr)         ! NUBEAM call-- beam/fusion product
                          ! deposition
if(ierr.ne.0) then [... implement error handling ...]
CALL ORBALL(iorbtot, ierr)! NUBEAM call-- orbiting and slowing down
                          ! integer, intent(out) :: iorbtot returns
                          ! zero if there were no orbits to follow.
if(ierr.ne.0) then [... implement error handling ...]
CALL NBFINISH(ierr)       ! NUBEAM call-- finish timestep.
if(ierr.ne.0) then [... implement error handling ...]

! at this point all output profile data are in XPLASMA. Scalar data
! and XPLASMA id's can be fetched through compound data types using
! NUBEAM's generated nbo_get_* calls.

CALL <my_output_extractor> ! get NUBEAM outputs.
```

The implementation of the NUBEAM module in the TRANSP code allows an estimate of time required by the NUBEAM module for a typical run simulating a neutral beam heated tokamak plasma. For example, a single processor computer time using an Intel Pentium IV 2 GHz chip running under Redhat Linux 7.2 operation system was used for a test run. Fig. 4 shows the CPU time distribution between different modules for a simulation of TFTR discharge 37 065, which is used as a test case for the TRANSP code. The simulation is started at 3.0 sec after the

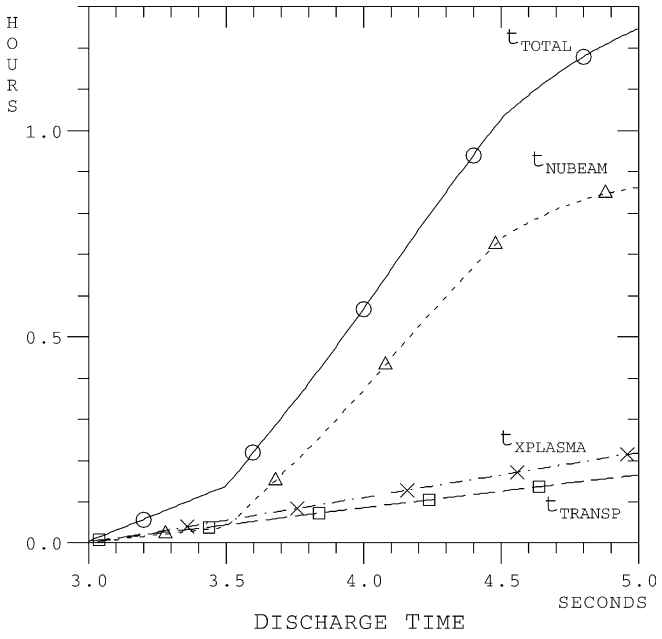

Fig. 4. CPU time distribution.

discharge is initiated and the beam deposition begins at 3.5 s. Timesteps in the TRANSP code are adjustable and the NUBEAM code is not called each timestep. Based on the experience gained from intensive use of the NUBEAM package in the TRANSP code, it has been found that a sufficient time interval between calls to the NUBEAM module is 0.01 s for most current tokamak discharges (the NUBEAM timestep should be short compared to typical slowing down times of newly deposited fast ions). The value of time interval used in our simulation is 0.01 s. The CPU time used by the NUBEAM module t_{NUBEAM}, the XPLASMA module t_{XPLASMA}, and the remainder of the TRANSP code t_{TRANSP} add up to the total CPU time t_{TOTAL}:

$$t_{\mathrm{TOTAL}} = t_{\mathrm{NUBEAM}} + t_{\mathrm{XPLASMA}} + t_{\mathrm{TRANSP}}.$$

In this particular example, the NUBEAM module uses about two thirds of the total CPU time in a TRANSP simulation. The average time used by the NUBEAM module per call is about 25 s during the NBI phase of the discharge.

4. NUBEAM module output

The output from the NUBEAM module at a given timestep can be divided into three categories: scalars or arrays of scalars; 1-D profiles $f_1(\rho)$; and 2-D profiles $f_2(\rho,\theta)$ on the 2-D irregular grid that is described in Section 3.2.

In the first category, the variables contain 0-D information such as contributions to the global energy balance, momentum balance, or particle balance for each individual fast ion species. These are typically arrays of numbers dimensioned by the fast ion index array dimension. The 1-D profiles are defined on the NUBEAM internal grid but stored as XPLASMA objects, allowing interpolation to the caller's own radial grid as a function of ρ. For each 1-D output profile, a unique integer XPLASMA ID code is returned.

Output variables are organized into several blocks, each of which corresponds to a compound data type definable in the caller's code by the statement

```
use nbi_types
```

The caller's code creates an instance of each output data type, such as

```
type (nbotype_power_balance) :: zpbal
type (nbotype_powers) :: zpowers
```

Then, after the beam code timestep has been completed, the caller code must issue a call, such as:

```
call nbo_get_power_balance(zpbal) ! getting scalar power balance from
                                  ! the NUBEAM code
```

after which individual elements can be extracted into the caller's data structures. This is illustrated in the following example

```
do isi=1,nsfast     ! loop over transport code's fast specie list

   itype=0          ! transp code determines fast ion type:
                    ! Fokker-Planck modeled RF fast ions,
                    ! not handled by NUBEAM, could be present.
   if(...[NUBEAM beam ion]...) itype=1
   if(...[NUBEAM fusion product]...) itype=3

   if(itype.gt.0) then
      isb=index_nbfi(Z(isi),A(isi),itype) ! get NUBEAM's specie index
                                          ! all arguments: integer

      pinjs(isi) = zpbal%pinjs(isb)       ! copy out NUBEAM compound type
      pshine(isi) = zpbal%bpshins(isb)    ! data members. These are
      ...                                 ! scalar power balance terms.
   endif
enddo
```

The definitions of the compound data types in `nbspec.dat` are described in Appendix D.

To retrieve the heating profiles, the user code employs:

```
use nbi_types
type (nbotype_powers) :: zpowers   ! declaration of variable
                                   ! zpowers, which has Fortran-90
                                   ! compound data type nbotype_powers
                                   ! and contains ids of all 1-D
                                   ! power profiles
...
call nbo_get_powers(zpowers)       ! get profile ids
```

```
id = zpowers%io_pbi                ! interpolate PBI to user grid
call my_xplasma_extractor(id, my_pbi_array, ...)

id = zpowers%io_pbe                ! interpolate PBE to user grid
call my_xplasma_extractor(id, my_pbe_array, ...)
```

In the last lines of this example, the subroutine `my_xplasma_extractor`, is a user written routine that uses XPLASMA to interpolate the NUBEAM profile data to the caller's own radial grid in ρ, with possible normalizations or transformations of units applied. The majority of NUBEAM output profiles are integrated particle, power, torque, source or energy density profiles in MKS units. The typical method employed to reconstruct the local density on the user's radial grid is to use the XPLASMA module to interpolate the integrated particle profile to the boundaries of the user's rho grid, and then compute the particle density using

$$n_j = \frac{N_{j+1/2} - N_{j-1/2}}{\mathrm{d}V_j}$$

where n_j is the density in the user's zone j; $N_{j\pm1/2}$ is the integrated particle profile interpolated to the boundaries of zone j; and $\mathrm{d}V_j$ is the volume of zone j. The variable $N_{j\pm1/2}$ is computed using the XPLASMA interpolation from the corresponding NUBEAM variable.

The NUBEAM module creates several outputs on the irregular 2-D (ρ, θ) grid that is illustrated in Fig. 3. These arrays are transferred from NUBEAM to the XPLASMA module. Although XPLASMA routines exist for fetching both the data and a description of the irregular grid, the interfaces are still being developed and, consequently, will not be described here. Documentation will be provided in a future release of the NUBEAM module.

The following outputs are defined over the 2-D grid:

(1) volumes of 2-D irregular grid zones;
(2) density profile for each NUBEAM fast ion species;
(3) profile of average fast ion perpendicular energy, for each NUBEAM fast species;
(4) profile of average fast ion parallel energy density, for each NUBEAM fast ion species;
(5) target fusion reaction rates between fast ions and thermal ions for each fusion reaction channel; reactions involving beam injected ions are counted separately from reactions involving the fusion products, Tritium or Helium-3 ions;
(6) fast-ion–fast-ion fusion reaction rates for each fusion reaction channel;
(7) neutral source profile for each thermal species, as driven by each NUBEAM fast ion species;
(8) charge exchange neutral sink rate estimate for each species of thermal neutrals due to each NUBEAM fast ion species;
(9) impact ionization neutral sink rate estimate for each species of thermal neutrals due to each NUBEAM fast ion species;
(10) Monte Carlo summed fast ion distribution function as a function of 2-D grid zone index, velocity pitch, and energy, for each NUBEAM fast ion species.

5. Summary

The NUBEAM module, which computes the power deposition and other source profiles that are consequence of Neutral Beam Injection (NBI) in magnetically confined plasmas, has been extracted from the TRANSP integrated modeling code [7–10] and modified to meet the standards of the National Transport Code Collaboration (NTCC) module library [1,2]. The NUBEAM module provides a comprehensive computation of the effects of NBI in

tokamak plasmas. The centerpiece of the NUBEAM module is a Monte Carlo computation of the trajectories of neutral atoms and fast ion orbits within the magnetically confined plasma, together with the associated atomic physics, collisional interactions with the thermal target plasma, and predicted nuclear reaction rates. The module includes options to compute the effects of large scale instabilities, such as sawtooth oscillations and fishbone instabilities, as well as the effects of magnetic ripple. All NUBEAM dependencies are resolved within the NTCC library; NUBEAM uses eight other NTCC modules (see Appendix E).

The main challenge encountered with extracting this module was caused by the fact that the original NUBEAM package shared more than one thousand variables that were stored in the large common blocks of the TRANSP code. As the NUBEAM code was turned into a module, these variables were organized into 370 input, 365 output, 55 input and output variables, and the remainder as variables that are internal to the NUBEAM module. All of the input and output variables were organized into publicly defined Fortran-90 compound data structures. The interface for these structures is written with a Python-script code generator. Default values are assigned to all of the input variables in order to minimize the number of variables that have to be set by the user. The NTCC XPLASMA module [1,2] is used to interpolate all of the profiles from the user's spatial grid to the NUBEAM spatial grids and back again. The result of all these changes is the transformation of a large legacy code into a portable, reusable, and well documented module with encapsulated data, physics, and interpolation methods. Moreover, the NUBEAM module described in this paper has now been introduced back into the TRANSP code and is used to carry out the NBI computations in the TRANSP code. This has allowed validation of the results produced by the NUBEAM module and facilitates future improvements of the treatment of the NBI physics in the TRANSP code as well as in other transport codes in which the NUBEAM module is being installed.

Appendix A. Physical elements of the NUBEAM module

The NUBEAM module self-consistently takes into account the following physical processes:

(1) Beamline geometry and beam composition by isotope, with time dependent specification of power, voltage, and energy fractions for each beamline;
(2) Trajectory of neutral atoms passing through the plasma, with deposition of neutral beams and charge-exchange loss and recapture of partially slowed down fast ions;
(3) Guiding center fast ion orbits: trajectories of fast ions within the plasma, including banana orbits and the loss of the ions to the walls;
(4) Collision operator and thermal plasma source terms: heating rates, momentum sources, current source, particle sources;
(5) Anomalous diffusion of fast ions;
(6) Neutral particle—fast ion reactions: ionization, charge exchange, and a model for multi-step ionization as a result of excitation effects;
(7) Multiple fast ion species including fusion product fast ions treated as separate Monte Carlo species; multi-species target plasma;
(8) Effect of magnetic ripple;
(9) Effect of large scale instabilities, such as fishbone instabilities and sawtooth oscillations;
(10) Finite Larmor radius corrections, collisions or losses at the actual particle position, not at the orbit guiding center.
(11) Estimates of fusion reaction rates, separated not only by reaction but also by reagent types, e.g., beam–target and beam–beam reactions counted separately; and
(12) Numerical estimate of the entire fast ion distribution function, suitable for subsequent use e.g., by diagnostic simulation models.

Appendix B. NUBEAM module Fortran-90 input structures

	Structure	Description
1	sys	basic system information for Monte Carlo code
2	times	start and stop times for the current timestep
3	grid	basic grid information
4	beams	the beams and the thermal plasma species
5	impurity	atomic weight and atomic number of the impurity species
6	minority	RF minority species
7	powers	beam powers, voltages, and energy fractions (at full, half, and one third energies)
8	fusion	fusion products
9	fpp	Monte Carlo code support for Fokker-Planck model
10	num	numerical controls
11	atomic	atomic physics controls
12	collid	collision operator controls
13	flr	finite Larmor radius corrections
14	saw	sawtooth model controls
15	adif	description of anomalous diffusion effect
16	ripple	description of magnetic ripple effect
17	outcon	code output control options
18	fishbone	specification of fishbone model
19	box	beam-in-box neutral density calculation controls
20	misc	miscellaneous options and parameters, such as options for calculation of fast ion driven current
21	profiles	"XPLASMA IDs" of the profile inputs to the NUBEAM module
22	fusion	reaction rate parameters to be included
23	density	arrays associated with sawtooth mixing

Appendix C. NUBEAM module Fortran-90 output structures

	Structure	Description
1	deposition	energy and particle deposition profiles
2	n0_fast	flux surface averaged fast neutral density profiles
3	trap_fraction	fraction of "banana trapped" ions
4	erngfi_output	densities and trapping fractions
5	average_energies	average parallel and perpendicular energies, average energy
6	excited_states	excited states correction profiles (flux surface averaged)
7	mc_statistics	statistics that are computed after deposition and before slowing down
8	rotation	approximate toroidal angular velocity of ions, before and after sawtooth crash
9	compression	heating profiles resulting from compression
10	powers	heating profiles inside flux surface
11	currents	driven current profiles
12	neutral_sources	neutral sources
13	recapture	profiles showing processes for recapture of charge exchange fast neutrals
14	neutral_sinks	flux surface averaged sink rates for thermal neutral gases
15	sources	electron and ion density source profiles
16	fokker_planck	profiles related to the collision operator
17	radial_current	radial current profiles through boundaries
18	torques	torques applied to thermal plasma species
19	hi_z_beams	average charge state of orbiting heavy ions for $Z > 2$
20	power_balance	data related to scalar power balance for beam species
21	momentum_balance	quantities related to scalar angular momentum balance
22	torque_work	power associated with work done by the beam on a rotating target plasma
23	particle_balance	information related to fast ion particle balance

Appendix D. Use of Python code generating scripts to build input and output structures

A Python code generator, `nbigen.py`, is used to maintain the NUBEAM interface and state file input/output facility. The use of a code generator makes it much easier to add new input and output options to the NUBEAM module. By using compound data types with defaults, newly added input options can be safely defaulted without breaking the existing user code. In particular, a new option can be added to NUBEAM, and the updated NUBEAM code distributed, without breaking user implementations that might not use of the new option.

The NUBEAM source is distributed with `nbigen.py` together with the file `nbspec.dat`, which is the NUBEAM module data specification file. When `nbigen.py` is run, it reads the `nbspec.dat` file and uses the specifications to write interface routines such as `nbi_init_sys` and `nbi_set_sys`, as well as NUBEAM's internal data modules `nbi_dimensions` and `nbi_com`.

The `nbspec.dat` file is important because it defines the actual input and output data for the entire NUBEAM module and the module's internal data structures, as well as defining elements of the state of the time-dependent NUBEAM calculation. The `nbspec.dat` file is well commented, and the user is able to refer to it for many details of NUBEAM input/output. Those comments in the `nbspec.dat` file that start with "!" pertain to NUBEAM data elements; comments that start with "#" pertain to `nbspec.dat` format and details of the python code generator.

Each element of each compound data type is defined in the data structure file `nbspec.dat` in lines of the form:

```
<data-type>[A][S][code] <name>[(dims,...)] [! comments...]
```

where <data-type> is a Fortran data type code (R=REAL, D=REAL*8, I=INTEGER, L=LOGICAL), [A] is an optional indicator that the variable is an array, and [S] is an optional flag that indicates whether the variable is used during next timestep by the NUBEAM module and should be saved in the NUBEAM state file, <name> is the name of variable, and [(dims,...)] are optional dimensions of the variable. There is also an optional symbolic [code] specification, which indicates a profile subtype and is defined as follows:

[code]	Description
~	Indicates a variable such as "average energy" which is not integrated from the axis. These variables require smoothing.
^	Indicates a variable such as "slowing down time" which does not require smoothing.
|	Indicates a boundary oriented "flow" variable. Internally these are Monte Carlo sums of flows across boundaries. These variables do not require smoothing.
@	Indicates profiles, which specify an enclosed toroidal current as a function of flux surface label. These profiles are integrated using the cross-sectional area rather than the volume of plasma flux zones and are smoothed.

As an example of a scalar data definition, the Fortran-90 compound data structure `power_balance` contains the specifications:

```
DA  pinjs(mibs)  ! watts
                 ! injected power (or fusion product source power),
                 ! by species.

D  pftota        ! watts
                 ! grand total power in fusion product source
```

The 1-D profile outputs are similarly declared, but are identifiable by the presence of the array dimension `mj`, which is the radial grid dimension used in the NUBEAM module. Thus, for example, the Fortran-90 compound data structure `powers` contains:

```
DA pbe(mj)          ! watts
                    ! power to electrons from beam heating
                    ! for electron power balance
DA pbis(mj, mibs)   ! watts
                    ! power to ions from fast ion heating, by species
```

These declarations define a set of XPLASMA output profiles. The corresponding compound data type for each of these items contains an integer scalar or array which is formed by deleting the radial dimension from the above specifications. For example, in this case:

```
integer :: id_pbe, id_pbis(mibs)
```

which are members of a compound Fortran-90 structure of type `nbotype_powers`. The array dimension symbols `mj` and `mibs` indicate the radial ρ grid and the fast ion species indexing, respectively; these are defined near the beginning of `nbspec.dat`.

In summary, the code generator reads

```
nbspec.dat -- NUBEAM i/o and module data specification file
              basic data types are:
              R -- REAL,
              I -- INTEGER,
              L -- LOGICAL,
              D -- REAL*8,
              C*n -- CHARACTER*n.
              An instance of each specified item is generated
              in NUBEAM's internal Fortran-90 module; in addition,
              input/output items are made members of publicly
              declared compound data types used for transfer of
              information in and out of the NUBEAM internal module.
```

and the code generator writes the following NUBEAM input component files:

- `nbi_dimensions_mod.f90` and `nbi_com_mod.f90`. NUBEAM's internal Fortran-90 modules. These should not be referenced directly by user code, but they contain copies of comments from nbspec.dat, and may be useful as documentation.
- `nbi_types.f90`. Public Fortran-90 module defining input/output data structures.
- `nbi_alloc.f90`. Allocate most NUBEAM data structures based on user specified grid sizes—called by user after a successful `nbi_set_dims` call...
- `nbi_alloc2.f90`. Allocate remainder of NUBEAM data structures based on additional user input (e.g., desired number of Monte Carlo particles in simulation). The user does not call this directly.
- `nbi_dalloc.f90`. Deallocate NUBEAM data structures. Since the NUBEAM simulation has state, this should not be done unless all NUBEAM fast ions have thermalized, and NUBEAM is never again to be called.
- `nbi_init.f90`. Set defaults for input data structures.

- `nbi_set.f90`. Pass inputs to NUBEAM. Since NUBEAM assumes state, this is generally done ONLY ONCE per run per input data structure.
- `nbi_get.f90`. Fetch current settings of input data structures.
- `nbi_update.f90`. Update input. Most input data elements are not updatable. For example, array dimensions can only be set once at the beginning of a run. Attempts to update non-updatable members are ignored. See the Inputs section for more details.
- `nbi_state_io.f90`. Subroutines for NUBEAM state save/restore operations. User callable routine: see `nbi_states.f90`.
- `nbi_ascii.f90`. ASCII dump of NUBEAM input and output quantities, for debugging.
- `nbo_get.f90`. Retrieve NUBEAM output (into output data structures).

Appendix E. NTCC modules required by the NUBEAM module

The NUBEAM module is a module of the NTCC module library [1,2]. The NUBEAM module together with its driver program and test cases can be downloaded from the NTCC module library web site [2]. The NUBEAM module uses other NTCC modules, which are briefly described below.

- The NTCC PREACT module performs lookups and interpolation of the rate (weighted product of cross-section and velocity) of various charge exchange, ionization, and fusion reactions.
- The NTCC PORTLIB module provides a number of “operating systems support” functions, for which a standard interface is not available: access to the shell, access to command line arguments, access to environment variables or previously used logical names, elapsed CPU time, and so on.
- The NTCC XPLASMA module provides a standard representation for plasma MHD equilibria and parameter profiles. The module includes routines for setting up the equilibrium, profiles, scrape-off region definition and limiter specification and routines for coordinate transformations.
- The NTCC PSPLINE module contains a collection of Spline and Hermite interpolation tools for 1D, 2D, and 3D datasets on rectilinear grids. The spline routines yield full control over boundary conditions.
- The NTCC EZCDF module provides an easy interface for netCDF routines.
- The NTCC KDSAW module is the Kadomtsev-style MHD sawtooth “mixing model”.
- The NTCC R8SLATEC module is a library of mathematical subroutines.
- The NTCC RANDOM module is a portable, parallelizable, high quality random number generator.

More detailed information about these modules is available at the NTCC module library web site [2].

References

[1] A. Kritz, et al., The National Transport Code Collaboration module library, Comput. Phys. Comm., 2004, in press.
[2] NTCC module library, http://w3.pppl.gov/NTCC.
[3] K. Miyamoto, Plasma Physics for Nuclear Fusion, MIT Press, Cambridge, MA, 1987.
[4] R.A. Gross, Fusion Energy, Wiley, New York, 1984.
[5] E. Speth, Rep. Prog. Phys. 52 (1989) 57.
[6] L. Hu et al., Power loading on the beamline components and beam divergence on the negative-ion based NBI system for jt-60u, Technical Report JAERI-Tech 99-057, Japan Atomic Energy Research Institute, 1999.
[7] R.J. Hawryluk, An empirical approach to tokamak transport, in: Physics of Plasmas Close to Thermonuclear Conditions, vol. 1, CEC, Brussels, 1980, p. 19.
[8] TRANSP home page, http://w3.pppl.gov/transp.
[9] J. Ongena, M. Evrard, D. McCune, Trans. Fusion Tech. 33 (1998) 181.
[10] R.V. Budny, Nucl. Fusion 34 (1994) 1247.

[11] R.V. Budny, et al., Phys. Plasmas 7 (1994) 5038.
[12] R.J. Goldston, et al., J. Comput. Phys. 43 (1981) 61.
[13] R.J. Goldston, Basic Physical Processes of Toroidal Fusion Plasmas (Proc. Course and Workshop Varenna, 1985), vol. 1, CEC, Brussels, 1986.
[14] R.V. Budny, Nucl. Fluids 42 (2002) 1382.
[15] G. Düsing, et al., Fusion Tech. 11 (1987) 163.
[16] C.G. Lister, D.E. Post, R. Goldston, Computer simulation of neutral beam injection into tokamaks using Monte Carlo techniques, in: Proc. of the 3rd Symposium on Plasma Heating in Toroidal Devices (Varenna, Italy, 1976), Euroatom, 1976, p. 303.
[17] J.A. Rome, J.D. Callen, J.F. Clarke, Nucl. Fusion 14 (1974) 141.
[18] S.E. Attenberger, W.A. Houlberg, S.P. Hirshman, J. Comput. Phys. 72 (1987) 435.
[19] R.K. Janev, W.D. Langer, K. Evans, D.E. Post, Elementary Processes in Hydrogen-Helium Plasmas, Springer-Verlag, Berlin/New York, 1987.
[20] C.F. Barnett, I. Alvarez, C. Cisneros, R.A. Phaneuf, et al., Collisions of H, H_2, He and Li atoms and ions with atoms and molecules, Technical Report ORNL-6086/V1, Oak Ridge National Laboratory, 1990.
[21] R.A. Phaneuf, R.K. Janev, M.S. Pindzola, Collisions of carbon and oxygen ions with electrons, H, H_2, and He, Technical Report ORNL-6090/V5, Oak Ridge National Laboratory, 1987.
[22] D. Schultz, P. Krstic, private communication.
[23] R.B. White, M.S. Chance, Phys. Fluids 27 (1984) 2455.
[24] N. Byrne, H. Klein, J. Comput. Phys. 26 (1978) 352.
[25] H.-S. Bosch, Review of data and formulas for fusion cross-sections, Technical Report IPP I/252, Max-Planck-Institut für Plasmaphysik, 1990.
[26] S. von Goeler, W. Stodiek, N. Sauthoff, Phys. Rev. Lett. 33 (1974) 1201.
[27] K. McGuire, et al., Phys. Rev. Lett. 50 (1983) 891.
[28] R. Kaita, et al., Phys. Fluids B 2 (1990) 1584.
[29] W.W. Heidbrink, G. Sager, Nucl. Fusion 30 (1990) 1015.
[30] ITER Physics Expert Group on Energetic Particles, Heating and Current Drive, ITER Physics Basis Editors, Nucl. Fusion 39 (1999) 2471.
[31] B. Kadomtsev, Soviet J. Plasma Phys. 1 (1975) 295.
[32] R.B. White, R. Goldston, M.H. Redi, R.V. Budny, Phys. Plasmas 3 (1996) 3043.
[33] R.J. Goldston, R.B. White, A.H. Boozer, Phys. Rev. Lett. 47 (1981) 647.
[34] M.H. Redi, et al., Nucl. Fusion 35 (1995) 1509.
[35] D.R. Mikkelsen, et al., Phys. Plasmas 4 (1997) 3667.
[36] L.L. Lao, H.S. John, R.D. Stambaugh, A.G. Kellman, W. Pfeiffer, Nucl. Fluids 25 (1985) 1611.
[37] T.W. Fredian, J.A. Stillerman, Fusion Engrg. and Design 60 (2002) 229.
[38] W. Davis, P. Roney, T. Carroll, T. Gibney, D. Mastrovito, Fusion Engrg. and Design 60 (2002) 247.