Review of the NUT-0.1 Code

Developer: Prashant M. Valanju, IFS
Reviewer: John Mandrekas, GIT
Date: August 25, 2003

Summary

NUT is a fast, semi-analytic routine for the calculation of neutral transport in plasmas based on integral transport methods. The methodology has been published in *Journal of Comput. Phys*., **88** (1990) 114, and a PDF version of this paper is included in the NUT-0 distribution. While the present version of the code has a number of limitations (simple non-adaptive rectangular grid, limitation to one atomic species, rather elementary atomic physics and wall reflection models, etc) and cannot be compared to more sophisticated neutral transport codes such as DEGAS, EIRENE etc., it is nevertheless a useful replacement of the simple analytic neutral transport models usually found in most 1½-D transport codes. The developer of the code is aware of these limitations (a more detailed list of them is included in page 14 of the User Manual) and a new version of the code (NUT-1) is under development.

The present version of the module would be a useful addition in the NTCC collection of source modules and I therefore recommend its approval.

Review Summary

The review consisted of installing and testing the module on various computing platforms (SUN Workstations running Solaris 8 & 9, HP-UX systems running HP-UX 11i, PC running Windows XP and the Absoft Fortran Compiler 7.5, in addition to the developer's own platform, Alpha Linux with the Fujitsu Fortran compiler) and checking the compliance of the module to the NTCC standards. In addition, the usability of the module was tested by writing small driver programs to call routines of the module and test them.

The original version of the NUT-0 module that had been submitted to NTCC failed to compile on several platforms due to the use of several non-standard Fortran extensions used by the developers. In addition, it violated a number of NTCC standards. Based on my suggestions, Prashant Valanju created a new version of the module, NUT-0.1 in March 2003. This version was re-tested, a few minor problems were fixed so that the code would compile on platforms with strict Fortran compilers (HP F90 v2.6.4 under HP-UX 11i and Absoft Fortran 90/95 v7.5) and the final version was submitted by the developer on August 20, 2003. Although the new version does not include any new physics, the structure of the code and various supporting tools (installation scripts, Makefiles, etc.) are much improved and the updated module now satisfies the NTCC coding standards. A summary of the changes from the original version 0 to version 0.1 is included in the User Manual (Appendix D, page 34).

There is sufficient documentation to get started (README file and a NUT-0 User Manual in PDF format, in addition to the *J. Comput. Phys*. article), however potential users will probably have to consult some of the user-callable subroutines for more

detailed information. Assuming this code will most likely be used to calculate fueling rates in 1½-D core transport simulations, most users will have to develop their own geometry routine to map their flux surface geometry to the form required by the NUT code. There are a few examples for simple cylindrical and Shafranov-shifted geometries (files `nut0_cyl_fs.f` & `nut0_shafr_fs.f`), but anything more complicated will have to be developed by the user. Perhaps other NTCC tools, like XPLASMA, may prove useful for this task.

Detailed report follows:

I. GENERAL STANDARDS

Standard: *Provide source code for each physics module or code.*

OK

Standard: *Provide test case(s) with driver program(s) with input and output data and their documentation.*

A test case (called "demo" is included, along with a "demoplots" file, which can be used with GnuPlot to display the results.

Standard: *Provide script to compile and link (e.g., makefile). The script should make at least some provision for portability to multiple brands of Unix(at minimum). Provide clear documentation (possibly in the README file) on how to use the script or makefile.*

OK: Installation script (nut0_install) and a Makefile (nut0_make) are included. The user should create a "rules" file in the "rules" directory for his/her own computing platform with the desired Fortran compilation options (compiler, linker, flags, etc.). Detailed documentation in the README file and the User Manual is included.

Standard: *Provide a README file giving (a) the name of the module and its authors, (b) the location and form of general module documentation, and (c) information (or pointer to more detailed documentation) enabling a user to build binaries from the source code.*

OK (see comments above)

Standard: *Provide documentation about how the module should be used, for example, whether the module needs to be initialized or used sequentially. Important usability issues, such as the existence of state information in COMMON or other static memory, which persists between calls, must be described.*

OK: Detailed documentation exists in the User Manual, with the list of the user-callable routines and with detailed instructions on how to call NUT-0 from another code. All data communication between the calling program and NUT-0 should be via subroutine arguments and no COMMON blocks.

Standard: *Eliminate graphics calls embedded in physics modules*.

OK: Fixed in updated version of the module (NUT-0.1)

Standard: *The source code files (e.g., *.f, *.c or *.cpp files) should be submitted rather than requiring extraction from another file.*

OK: Source code is included in the package.

Goal: *Portability (code should run on multiple platforms and under different operating systems).*

The reviewer has tested the module on systems running the SUN (Solaris 8 & 9) and the HP-UX 11i operating systems. The code has also been successfully compiled on a PC running Windows XP and using the Absoft Fortran compiler (installation scripts and Makefiles were not tested on the PC platform). In addition, the main platform of the developer is an Alpha Linux system running the Fujitsu compiler.

Goal: *Offer single and double precision versions or offer user control of precision at compile time*.

This has not been implemented in the present version of the code. It is one of the major goals of the next release.

Goal: *Multi-platform (code should run on different computers).*

OK: The code has been tested on four platforms (see portability comments above)

Goal: *Provide error checking (but not stops).*

OK: A number of the user callable routines (e.g., nut_start and nut_plasma) have error flags. It is up to the user of the code to take action depending on the value returned by these flags. NUT will not terminate even if the flags indicate errors.

Goal: Minimize external dependencies that cost money (i.e., avoid using expensive proprietary licenses).

No external libraries are required. The view the demo results, the freely available GnuPlot package is required, but this is optional and the user can substitute a different plotting package.

Standard: *Supply warnings in the documentation when the above goal has not been met.*

N/A

Goal: *Arrays should be dynamically allocated.*

Partially done. Will be implemented fully in next release.

Standard: *The characteristics of the I/O should be clearly documented (i.e., the implementation I/O unit numbers, if any).*

OK: Information about I/O is included in the User Manual.

Goal: Avoid using hard-wired I/O unit numbers. Allow informational output to be switched on or off. Provide a method for rerouting warning or error message output to a user specified file or I/O unit number.

OK

II. DOCUMENTATION STANDARDS:

Standard: *Provide Name of contact person for support.*

OK: Contact information is included in the README file and the User Manual.

Standard: *Provide Date of last revision.*

OK.

Standard: *Provide at least comments describing module or code, citations to publications (if any), and range of validity*.

OK: A copy of a published paper detailing the physics of the method and its implementation is included. A list of limitations of the present version of the code is also included in the User Manual.

Standard: *Specify the precision of floating point calculations*.

OK (See first part of file `nut0_icom.i`)

Standard: *Provide index of input-output variables for each module (include type of variable, dimensions, units).*

Extensive lists of variables and information on units are included in the User Manual. The user may have to consult individual routines for more detailed information.

Standard: *List dependencies -- names of external routines called.*

OK: The User Manual includes a schematic calling tree and other code-structure information.

Standard: *Provide statement of known bugs.*

No specific bugs are listed, but limitations of the methodology and potential pitfalls when building the code in various systems are included in the User Manual.

III. DATA STANDARDS:

Goal: *Provide interface routines to data.*

OK

Goal: *Use self-describing data files (such as NetCDF).*

Not implemented

Goal: *Establish standards for variable names, units, dimensions, independent variables and grid descriptions as they appear in the module interfaces:*

OK: Standards for variable names and units are described in the User Manual