

NUT-0 User Manual

Prashant M. Valanju
Institute for Fusion Studies
The University of Texas at Austin

Version 0.1, NTCC, March 2003
512-471-1350
pvalanju@mail.utexas.edu

OBJECTIVES OF THIS MANUAL:

To facilitate quick as well as long-term use, evaluation, and user-driven modifications of NUT-0 - the initial test version of the NUT transport code.

ORGANIZATION OF THIS MANUAL

This manual is organized as follows to reach the above objectives:

- [QUICKSTART FOR BEGINNERS](#) who want to get started fast:
 - How to download Nut0.tar from NTCC modules webpage
 - How to install nut0, make a demo run, plot results etc.
 - How to make nut0 library and stand-alone exe for your geometry.
 - Instructions on calling NUT-0 from user's routine (e.g., Nut_demo.f),
- [THE CONTEXT](#): What [NUT-0 code](#) calculates and how:
 - The semi-analytic NUT algorithm, its advantages, and limitations,
 - NUT-0 scope: what is and is not in the first version NUT-0 and why,
 - What NUT-0 can be used for and some benchmarks and run-times,
 - Directions of further NUT developments, "wish list", and priorities.
- [USER MANUAL](#): Help on how to use NUT-0 is given at three user levels:
 - [For regular users](#) who are familiar with basic NUT-0 use:
 - Instructions on reconfiguring [NUT-0 geometry](#) by themselves,
 - An interactive help routine to guide geometry setup,
 - A large collection of auxiliary routines for special cases,
 - [For advanced users](#) who wish to help expand NUT-0
 - Code description (logic, structure, and execution),
 - Changes to NUT-0 that are being made at present.
 - Our priority-ordered "wish list" for NUT improvements.
 - Improvements to the atomic physics used in NUT-0.

QUICK WORD ABOUT NUT

NUT is a **very fast, semi-analytic algorithm** for **3-D neutral transport in 3-D plasma**. Analytic integration over short scales yields very high speed, as compared to Monte-Carlo methods, without compromising accuracy or requiring symmetries. A typical 3-d tokamak neutral simulation (TEXT or TFTR) takes **< 1 sec** on a PC.

REQUEST FOR [FEEDBACK](#) ON NUT-0: THE INITIAL TEST VERSION OF NUT

NUT-0 is only the first step in developing an **NTCC-compliant fast neutral 3-D code**. Getting feedback from actual users of NUT-0 will decide future priorities:

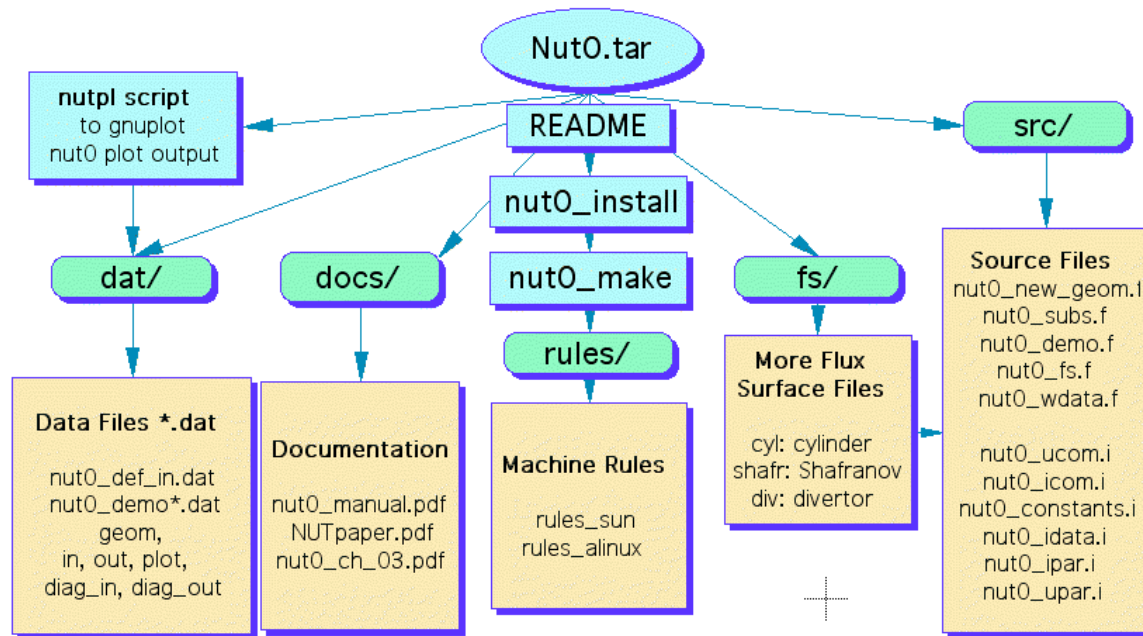
- Are NUT-0 capabilities already adequate? What features to add first?
- Which features should be dropped? What part gets used most?

Note for version 0.1: All file names now start with nut0_. I may not have changed all nut to nut0_ in this manual. All subroutine & variable names are still nut_.

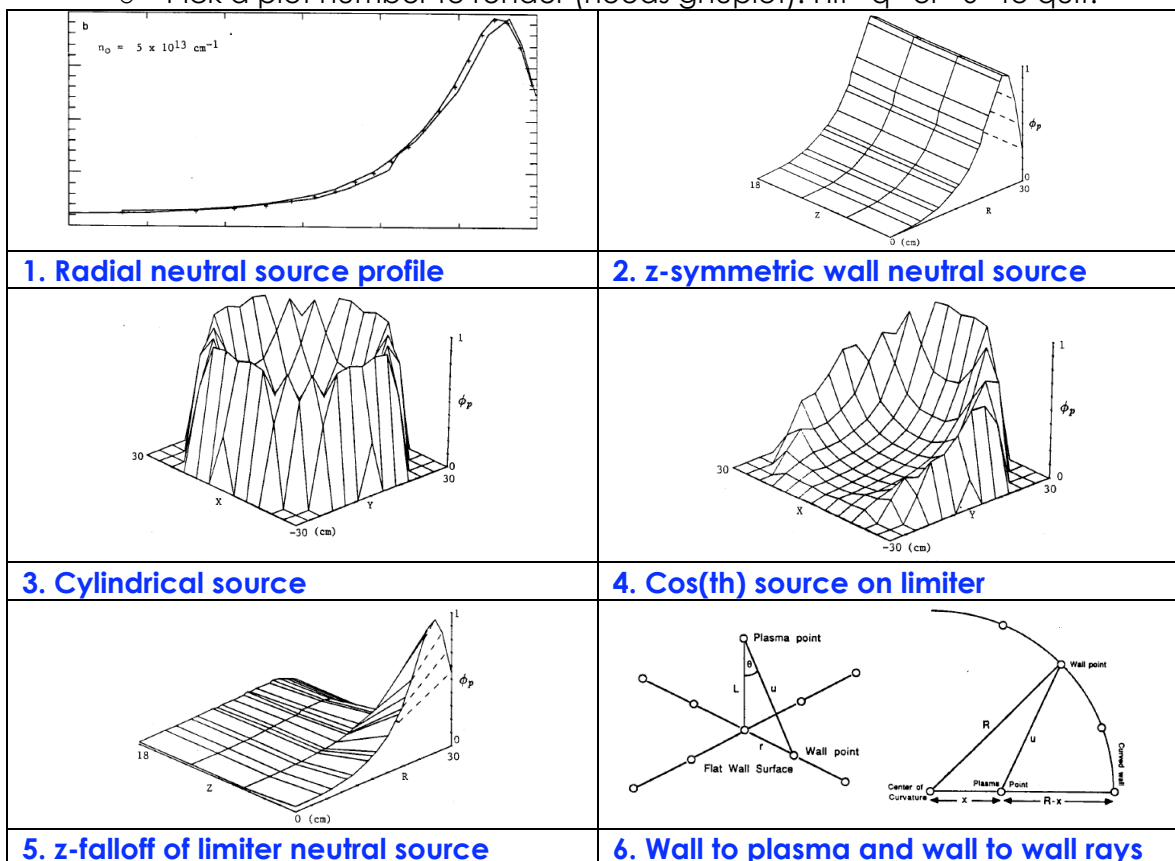
QUICK START FOR BEGINNERS:

- **Distribution:** NUT-0 is downloaded as a single tar file **nut0.tar** from the NTCC archive. Or you can email pvalanju@mail.utexas.edu for a copy and help.
- **Download Nut0.tar from NTCC modules webpage**
<http://w3.pppl.gov/rib/repositories/NTCC/catalog/Asset/nut.html>,
 - **Download nut0.tar from NTCC modules webpage**
<http://w3.pppl.gov/rib/repositories/NTCC/catalog/Asset/nut.html>,
 - Type **tar -xvf nut0.tar** to “untars” into a new directory **nut0.1/**
 - Type **cd nut0.1** to go to the new directory **nut0.1/**
 - Type **“nut0_install demo”** to make **nut0_demo** in **demo/**
 - Say **“y”** to run **nut0_demo**, if everything goes well so far
 - It will finish one full run in well under one second. It is fast!
 - Say **“all”** to plot all demo plots into file **nut_demo_plot_out.dat**
 - Type **“nutpl nut_demo_plot_out.dat”** to see, with **gnuplot**, the sample **nut0** output plots you just made.
- **Contents of NUT-0 distribution:** in directory **nut0.1/** you will find:
 1. **README** file for online unix use
 2. **nut0_install** a script for installing any version (including demo) of NUT-0,
 3. **nut0_make** standard makefile for nut0
 4. **nutpl** a script for rendering nut0 output plots with gnuplot
 5. **rules/** directory containing rules files for different machines (e.g. sun,linux,alpha etc.)
 6. **src/** All runtime NUT-0 subroutines in one folder. Contains
 - All *.f files and *.i include files
 - Main nut0 subs are all in nut)_subs.f
 - **Nut0_demo.f:** Stand-alone demo program to run all NUT-0.
 7. **docs/**
 - **Nut0Manual.pdf:** This user manual in pdf format
 - **NUTPaper.pdf:** pdf version of original NUT paper
 8. **dat/** data files
Demo files: To demo calling NUT-0 from your program:
 - **nut_demo_in.dat:** Runtime input data file for demo run,
 - **nut_demo_out.dat**
 - **nut_demo_plot_out.dat**
 - **nut_diag_demo_in.dat**
 - **nut_diag_demo_out.dat**
 9. **fs/** Sample flux surface shape *.f files
 - **nut_cyl_fs.f**
 - **nut_div_fs.f**
 - **nut_shafr_fs.f**
 10. **gifs/** contains some gifs used in nut0 manual

[Back to top](#)



- When you make a new nut case using `nut0_install`, a new folder is created under `nut0.1`, all necessary files are put there, and a new version is compiled.
- To see some nut0 plots before you start: type **"nutpl demoplots"**.
 - Get **gnuplot 3.7** (free) or **"module load gnuplot"** if you do not have it.
 - Pick a plot number to render (needs gnuplot). Hit "q" or "0" to quit.



- **Compiling and running NUT-0 demo (or any other case):**
First compile Nut0 case “demo” by typing: **“nut0_install demo”**
The script “nut0_install” does the following (for any case <case>):
 1. Makes a new directory <case>, copies all necessary files to it
 2. Runs the correct “makefile” for your machine and <case>.
 3. Makes the nut geometry precompiler and runs it to make *.i include files.
 4. Makes stand-alone, executable **nut0_<case> (e.g., nut0_demo)**.
 5. Asks you if you wish to **run it**. If you type “y”, it runs **nut0_<case>**, then the output plotter, so you can interactively plot any of the answers. At the prompt > nutplot cmd : type help to see plot options. I suggest you type “All” to see a standard plot group.
 6. The run generates following output files:
 - a. nut0_demo_out.dat: All NUT answers in a namelist format.
 - b. nut0_demo_plot_out.dat: Plots you made in gnuplot (table) format.
 7. Use the script **nutpl nut_<case>_plot_out.dat** to see plots for <case>.
- **Calling NUT-0 from your routine. e.g., see Nut0_demo.f:**
All data connection between your program and NUT is strictly via subroutine arguments, not commons. **DO NOT EVER USE the NUT common blocks.**
 1. Include ‘**nut0_ucom.i**’ for the dimensions and types of all variables to pass to and from nut subroutines. All variable names in this file start with nut_ to avoid conflict. A typical call sequence is:
 2. **call nut_start** to initialize the nut0 system. **Compulsory call , needed only one time.** Sets up machine geometry and global parameters.
 3. The next steps can be done in a **loop without repeating step 1**, say during a time-dependent simulation. **Every time, need to reset only things that have changed (plasma or external source).**
 - **nut_wall_source** or **nut_arb_source**: Set external neutral source.
 - **nut_plasma**: Set new plasma parameters.
 - **nut_calculate**: Core nut calculation.
 4. **call nut_plot**: [optional call] This interactively makes a file for gnuplot. That’s it! **You have successfully compiled and run NUT-0 (in < 1/2 sec !).**
- **List of nut subroutines, variables** and their types and meanings.
(see the include file ‘nut0_ucom.i’ for how they are dimensioned.)

Contents of geometry file used to build a new nut case:

&NUTGEOM

```

VERSION = 0.0000000E+00,
PERCENT = 0.5000000 ,           % error in equating locations
MX   =      6,                  # of half grid points in x direction
MY   =      6,                  # of half grid points in y direction
MZ   =      4,                  # of half grid points in z direction
MS   =      6,                  # of secondary points on each ray
MR   =      0,                  # of radial fs pts, 0: all distinct fs's
MF   =      2,                  # of subgrid points in each flux surface
MW   =     24,                  # of wall points
MWW  =      4,                  # of wall-to-wall ray points

```

```

WALCNTRL      =      0,           Method of wall specification
FSCNTRL = -1.000000 ,           Method of flux surface specification
FSPARM = 10*0.0000000000000000E+000 ,           Parameters for flux surface shape
MSEG      =      1,           Diagnostioc: # of site-line segments
MBEAM     =      1,           Diagnostioc: # of neutral beam lines
MBOL      =      1,           Diagnostioc: # of bolometer chords
MDCX      =      0,           Diagnostioc: # of double cx chords
MHAL      =      1,           Diagnostioc: # of halpha chords
XWAL      = 1024*0.0000000000000000E+000 ,           X array of wall points
YWAL      = 1024*0.0000000000000000E+000 ,           Y array of wall points
THWAL     = 1024*0.0000000000000000E+000           Theta array of wall points
/

```

Note about all the "diagnostic routines" in nut:

Original nut was developed for TEXT and TFTR use,

It was used closely coupled to the experiments, including diagnostic neutral beam, halpha measurements, cx diagnostics, bolometers, etc. Many auxiliary "diagnostic" routines were added to facilitate its use.

However, they are not in the "core" nut algorithm.

I had to decide whether to remove them, even though some may be useful.

For now, I have left all the "diagnostic" branch in nut0 as is,

I have not removed it, but I have also not upgraded it.

I suggest that you should not use the "diagnostic" branch in nut0 for now.

If you see something you like, let me know and we can activate it.

[Back to top](#)

Typical input data file to run NUT0 stand-alone:

(NOT USED if you call nut0 as a subroutine from your program)

```

&NUTDFLT
PROMPT = 1.           0:Prompts off, >1 more detailed
UNITS  = 'MKS-19-EV'   Units used, MKS, den in 10^19, E in eV
USERNAME = 'PMV'       Your user name
fnames(1)='nut_demo_in.dat'   Names of i/o files to use
fnames(2)='nut_demo_out.dat'
fnames(3)='nut_demo_fs_in.dat'
fnames(4)='nut_demo_xy_in.dat'
fnames(5)='nut_demo_diag_in.dat'
fnames(6)='nut_demo_plot_out.dat'
fnums(1)=21           Unit numbers of i/o files to use
fnums(2)=22
fnums(3)=23
fnums(4)=24
fnums(5)=25
fnums(6)=26

```

MACHINE = 'TEXT(CYL)'	Machine name
DISTYP = 'LIMITER'	Discharge type
TITLE = ''	Run Title
LEGEND = ''	Plot legend
RWAL = 0.30	Wall radius (m)
RLIM = 0.26	Limiter radius (m)
RMAJ = 1.0	Major radius (m)
TYPLAS = 0	Plasma profile type
TYPWAL = 3.	Wall type
RCOEF = 0.0	Wall reflection coeff
APLAS = 1.	Plasma atomic weight
ZPLAS = 1.	Plasma atomic number
AWAL = 52.	Wall atomic weight
ZWAL = 26.	Wall atomic number
ABEAM = 1.	Beam atomic weight
ZBEAM = 1.	Beam atomic number
BENERGY = 25000.	Beam main energy (eV)
DIO = 5.0	Central plasma density
DIW = 0.05	Plasma density at wall
DIA = 1.	Plasma density profile coeff A
DIB = 0.	Plasma density profile coeff B
TIO = 800.	Same for ion temperature
TIW = 5.	
TIA = 2.	
TIB = 0.	
TEO = 1000.	Same for electron temperature
TEW = 5.	
TEA = 2.	
TEB = 0.	
TAUP = 6.E-3	Particle confinement time (s)
TAUE = 1.	Energy confinement time (s)
SLIM = 1.	Source at limiter
SWAL = 0.01	Source on wall
SDIV = 0.	Source in divertor
SMEAN = 1.	Average source
BETASRC = 0.9	Source angular distribution cos(beta)
/	

[Back to top](#)

NUT0 subroutines to call from your program and their arguments:

They are roughly in the order in which they should be called

1. Initialize whole nut system with:

```
Subroutine nut_start(
& nut_fnames, nut_fnums, nut_prompt,
& nut_names, nut_values, nut_delta, nut_r, nut_rxy,
& nut_xwal, nut_ywal, nut_rwal, nut_thwal, nut_error )
```

Note: All arguments are potentially both input and output
 This allows NUT0 to guide you if anything is not set right
 by returning the default or giving a suggested value.

User declares their dimensions either by using the include file
 nut0_ucom.i that appears when you create a new nut0 geometry,
 or can allocate them or whatever.
 All names in nut0_ucom.i start with nut_ so as to avoid conflicts.
 Not an elegant solution, I know, but will do for now.

inputs :

- nut_fnames(5) : nut i/o file names and
- nut_fnums(5) : their corresponding unit numbers

List of files that nut0 potentially uses nut_fnames(i):

1. Input data file for nut (e.g., nut0_demo_in.dat)
2. Output data file for nut (e.g., nut0_demo_out.dat)
3. Input Flux surface profiles (e.g. nut0_demo_fs_in.dat)
4. Input XY profiles (e.g. nut0_demo_xy_in.dat)
5. Input file for auxiliary diagnostics. (e.g. nut0_demo_diag_in.dat).

Note: These unit numbers will be used in "open" statements
 to open corresponding files.
 USER must make sure they do not conflict with any
 file unit numbers in his/her main or other subroutines.
 ANY CONFLICT IS NOT A NUT PROBLEM!

nut_prompt = 0 : silent mode, no prompts (i or o) from nut0
 1 : Only major prompts
 2 and higher : more detailed diagnostic prompts

nut_names() : list of nut parameter names and
 nut_values() : their values that user wishes to set.
 Note: These are lists of any nut parameters and values you
 may wish to set from your program.
 If you do not set any, the values read from the files
 will be returned to you so you can use them in your program.

Allowed nut_names are (upto 40 names):
 'rwal','rmaj','rlim','typlas','typwal',
 'rcoef','percent','prompt','aplas','zplas','awal','zwal',
 'abeam','zbeam','benergy','di0','diw','dia','dib','ti0',
 'tiw','tia','tib','te0','tew','tea','teb','taup','taue',
 'slim','swal','sdiv','smean','betasrc','delx','dely',
 'rplas','zeff','dnbcur','dnbvolt'

output : nut_delta: grid constant,
 wall radius nut_rwal
 Geometrical arrays (dimensions from nut0_ucom.i):
 nut_r, nut_rxy, nut_xwal, nut_ywal, nut_thwal

```

c      nut_error = 0  All is well
c      = 1  Bad
c      = 2  Very bad, etc.
c      Note: Nut0 is not terminated even if nut_error is not zero
c      So YOU have to take proper action in your program.
c
c.....

```

[Back to top](#)

2. Set plasma profiles for nut with:

**Subroutine nut_plasma(nut_di, nut_te, nut_ti,
& nut_di2, nut_te2, nut_ti2, nut_error)**

```

c      purpose: get plasma profiles from proper source and put them in
c      form usable by nut, then perform all preliminary
c      calculations which require these profiles.
c      relevant control parameters :
c      typlas : plasma type control switch (see below)
c      profile parameters di0,diw,dia,dib, ti0,tiw,tia,tib
c      plasma coefficient file, data file, or database names
c      if typlas = 1 or -1, profiles are given by user
c      if typlas = 1, 1d profiles are given by user
c      if typlas = -1, 2d profiles are given by user
c      otherwise, this routine gets the profiles and returns them to you
c      if typlas = 0, 1d profiles are made from coefficients
c      if typlas = 2, 1d profiles are read from data file
c      if typlas = -2, 2d profiles are read from data file
c      if typlas = 3, 1d profiles are read from data base
c      if typlas = -3, 2d profiles are read from data base
c      nut_error if plasma is not found
c.....

```

3. Set wall source profile with:

**Subroutine nut_wall_source(nut_ewal, nut_swal,
& nut_phiw0, nut_smean)**

```

c      purpose: Get/set given wall source into nut internal arrays
c      inputs : limiter and wall energies and sources
c      nut_ewal() = wall energy array
c      nut_swal() = wall source array
c      nut_beta = coeff of beta for wall/limiter source
c      output : these values get calculated here, passed back to user,
c      and are also internally passed to nut.
c      nut_phiw0(mw,0:mz,mwe) = wall source
c      nut_smean = average wall source (for ANTIC comparison)
c.....

```

3. Or set arbitrary source profile with:

Subroutine nut_arb_source(nut_hip0, nut_phiw0)


```

c    purpose: set arbitray external sources
c    inputs : user supplied arbitrary external sources
c          nut_phip0(-mx:mx,-my:my,0:mz) : source in plasma
c          nut_phiw0(mw,0:mz,mwe) :external source on wall at all energies
c    output : none. these sources are just given to nut internal arrays
c.....

```

4. Perform one nut calculation with:

**Subroutine nut_calculate(nut_iter,
& nut_phip, nut_d0, nut_p0, nut_q0)**

```

c    purpose: calculate total neutral source by iteration
c    inputs : number of iterations nut_iter,
c          if nut_iter>0 start new iteration, else continue old one
c    params : total particle confinement time taup,
c          if taup>0, integral of source is matched to it
c          if taup<0, maximum of source is matched to it
c    output : calculated values of nut_phip,phiw,d0,p0,q0
c-----

```

5. Calls 3,4, and 5 can be looped as needed.

For example, loop calls 4 and 5 if plasma does not change but source does.

[Back to top](#)

```

c-----

```

Auxiliary routines, not essential, but useful:

Subroutine nut_save_main_par(fname)

```

c
c    action : saves all parameters and names in file fname
c    inputs : filename fname where default is to be saved
c    output : none
c    note  : there is no inverse routine nut_read_default beacuse
c          you have to reset all internal quantities if you change
c          default settings. it is like starting over, so it can
c          only be done in nut_start.
c.....

```

Subroutine nut_change_one_param(name, nut_values, value)

```

c    purpose: change one nut parameter only
c    inputs : name of parameter to be changed and new value
c          array of parameters
c    output : none. the value is changed if name matches
c.....

```

Subroutine nut_change_one_filename(name, nut_names, value)

```

c    purpose: change one nut file name

```

c inputs : name of name to be changed and new value
 c array of names
 c output : none. the value is changed if name matches
 C.....

Subroutine nut_get_one_param(name, value)

c purpose: get one nut parameter by name
 c inputs : name of parameter to get
 c output : value if name matches
 C.....

Subroutine nut_get_one_filename(name, value)

c purpose: get one nut name by name
 c inputs : name of name to get
 c output : value if name matches
 C.....

Optional diagnostic initialization call, not needed for core nut use.
 For now, I have left the "diagnostic" branch in nut0,
 but I have not upgraded it.
 I suggest that you should not use the "diagnostic" branch in nut0 for now.

call nut_start_diag
 (nut_dinf, nut_plotpar, nut_beampar, nut_bolpar, nut_halpar)
 nut_dinf,
 nut_plotpar,
 nut_beampar,
 nut_bolpar,
 nut_halpar

- **Note** that following extras are provided for your convenience only. They are NOT an essential part of NUT core, but you can use them if needed. See Appendix A to identify core and auxiliary routines.
 1. plot script Nutpl and plot output routines to generate nut_plots
 2. Reading and writing routines for standard nut I/O files.
 3. NUT-0 stand-alone driver, demo, and diagnostic routines for nut.

[Back to top](#)

THE CONTEXT: NUT ALGORITHM AND NUT-0 CODE

THE FAST, SEMI-ANALYTIC, 3-D NUT ALGORITHM FOR NEUTRAL TRANSPORT:

In the NUT integral equation method, the neutral particle distribution in position and energy space depends on the neutral source $\Gamma(r, E)$, which is the number of neutrals produced at a given point r (in three dimensions) at energy E . The source can have many components representing different chemical species. Under very general conditions including short and long mean free paths, this source is given by the sum of the externally injected source Γ_0 and secondary contributions from other points in the system

$$\Gamma(r, E) = \Gamma_0(r, E) + \int dE' \int dr' G(r, r') \Gamma(r', E'), \quad (1)$$

where the kernel $G(r, r')$ represents the probability of bringing a neutral created at r' to r and generating a secondary neutral from it. It has the general form

$$G(r, r') = a_{creation}(r, r') \frac{\exp[-\int_r^{r'} \Gamma(r'', r'')]}{4\pi |r - r'|^2}, \quad (2)$$

where $a_{creation}$ is the probability of secondary neutral generation, the inverse square factor comes from geometrical attenuation, and the physical attenuation integral is given by

$$\int_r^{r'} \Gamma(r'', r'') = \int_{r'}^r dr'' \Gamma_{total}(r'', r''). \quad (3)$$

where the inverse mean free path (IMF) a_{total} can involve physical and chemical processes. The volume integral in (1) can be broken into integrals over rays of the form

$$I = \int_0^s ds \exp[-\int_0^s \Gamma_{total}(s, s')] \Gamma(s). \quad (4)$$

It can be broken into segments over cells in which the background plasma is nearly constant, and each segment can be evaluated analytically (even for a short mean free path) by using a Taylor expansion (to any desired order) of the IMF's within each cell. *This analytical step allows the grid size to be much larger than the smallest mean free path, resulting in high speed with no loss of accuracy.* The grid has to adequately resolve only the background plasma. Since the integrals in (4) are linear in Γ , Eq. (1) becomes an algebraic matrix equation

$$\Gamma_i = \Gamma_i^0 + G_{i,j} \Gamma_j, \quad (5)$$

where the index i represents the cell, species, and energy bin. Boundary conditions at the wall are handled by modifying the kernel G to

$$K_{i,j} = G_{p,p} + G_{p,w} \left[I \otimes G_{w,w} \right]^{n-1} G_{w,p}, \quad (6)$$

where p and w indicate points in the plasma and on the wall, and changing the external source Γ^0 to

$$\Gamma^0 = \Gamma_p^0 + G_{p,w} \left[I \otimes G_{w,w} \right]^{n-1} \Gamma_w^0. \quad (7)$$

Equation (5) is quickly solved either by iteration or by inverting the operator $[I - G]$. Once the neutral source is found, various quantities such as neutral density, power loss etc. can be calculated by simple integrations.

[Back to top](#)

NUT-0 IMPLEMENTATION OF THE MORE GENERAL NUT ALGORITHM:

NUT-0 is the first version of NUT. It implements a restricted form of the general NUT algorithm. These restrictions can and will be removed in next versions of NUT:

- Uses fixed, cubic grid, simple atomic physics and wall reflection model.
- NUT-0 is a 2.5-D code in the sense that:
 - Only the plasma has to be 2-D, i.e., constant in z-direction, though
 - the neutral sources (wall, beam etc.) can be 3-D.
 - Wall shape can be fully 3-D.

QUICK NUT-0 OVERVIEW:

[Back to top](#)

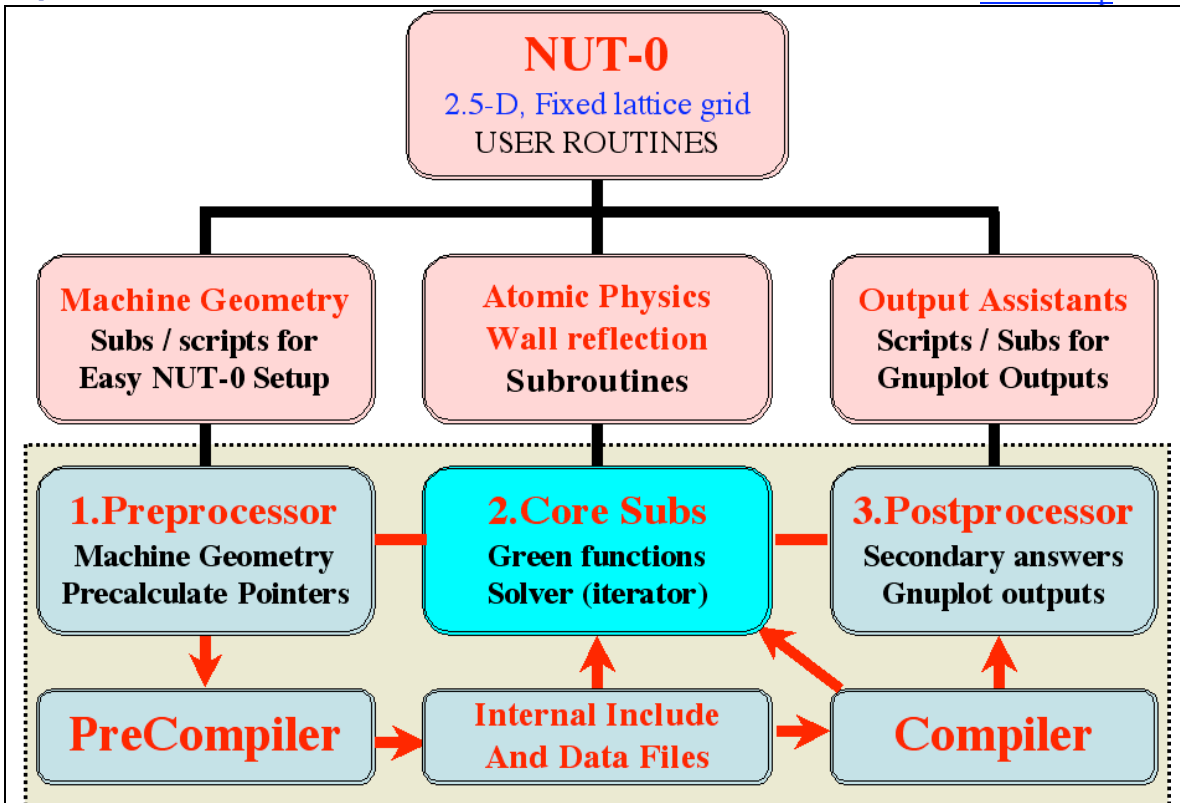


Fig. 1 NUT-0 outline.

Codes in dotted area are internal to NUT-0, do NOT call them directly.

- Users can run NUT-0 by using the `nut_demo.f` routine, or they can use NUT-0 in their own codes by calling any of the `nut_user` routines (in pink boxes).
- User codes can access NUT-0 input/output data via the following methods:
 - Calling `Nut_***_user(**)` routines in pink boxes, with proper arguments,
 - Or using ONE common block in `nut_user_com.i` file (all names `nut_***`)
- Geometrical preprocessor has to be run **ONLY** one time for each machine geometry to be simulated. This fixes **ONLY** the material walls etc. and **NOT** the plasma shape – same geometry is used as plasma shape changes.
- User's code can set its own inputs to NUT-0, or use the **`nut_start`** routine.
- The core is run with a single call to **`nut_calculate`**.
- After calling NUT-0, user's code can use and write out its own output, or use the output **`nut_plot`** routine to generate different (gnu)plots & files.
- Various auxiliary scripts are provided as demos to make these steps easier.

- **Learning about NUT-0 from the demo:**

After successfully running `nut_demo`, you should find out:

1. What simulation did demo run? (Tokamak with limiter source).
2. What do plasma, neutral source etc profiles look vs r , or r and z .
3. Hint: demo is essentially the case shown in Ref. 1.
4. How the geometry was set up for this case.
5. What the grid dimension settings mean.
6. Why we could get away with such a coarse grid.
7. How the wall shape is set up in the geometry input file.
8. How the external neutral source is set up using standard forms.
9. How an arbitrary 3-d external neutral source can be set up.
10. How the plasma n_e , T_e , T_i are set up using "flux surfaces".
11. How to set up arbitrary plasma shape without "flux surfaces".
12. How to plot various outputs and what they mean.
13. How `nut_start` sends all this input info to NUT-0.
14. How `nut_calculate` is called and what it does.
15. How `nut_plot` shows the outputs, and the structure of plot files.

Note: NUT-0 has many other diagnostic capabilities (multiple neutral beam sources, $H\alpha$ and bolometer signal profiles for multiple detectors, double-charge-exchange profiles, etc.) built-in, but you should not try them at this point. These allow `nut-0` answers to be directly compared to data. If you are interested only in NUT-0 as a subroutine for your plasma code, you will not need these diagnostic capabilities.

This exercise will quickly increase your familiarity with the NUT-0 code. Now you are ready to try making a NUT-0 geometry on your own.

THE NUT-0 GRID (CALCULATIONAL DOMAIN):

[Back to top](#)

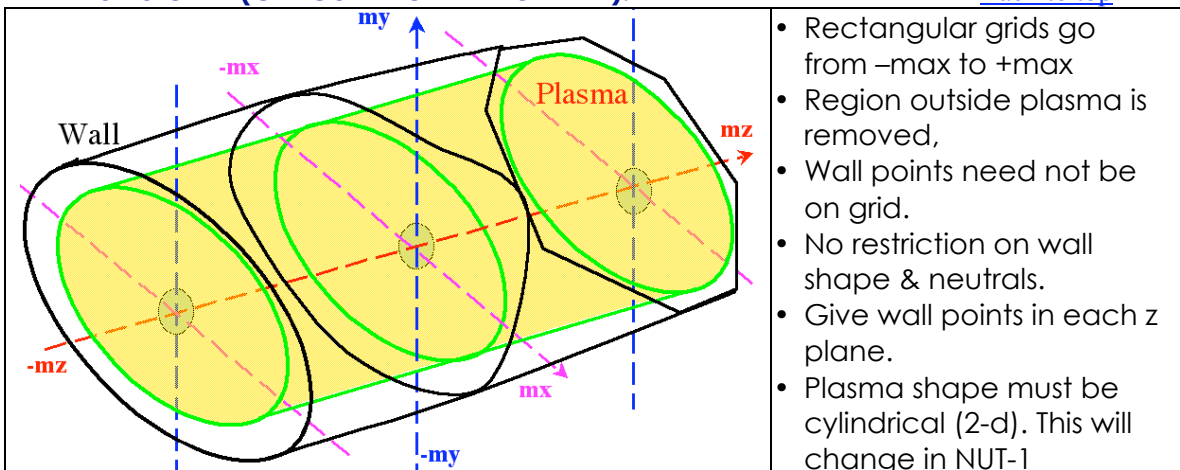


Fig. 2 The x,y,z region in which a NUT-0 simulation is fitted. Note that for modeling the neutral profile in a tokamak, for example, one can make use of the fact that the neutral mean free paths are \ll major radius R , so that a small section of plasma can be modeled (say near a limiter structure, as is done in the demo). Hence, for most plasmas, the z -symmetry restriction in NUT-0 is ok (and faster).

NUT CODE IMPROVEMENTS: CURRENTLY UNDERWAY AND FUTURE

KNOWN UNDESIRABLE FEATURES FROM OLD CODE STILL IN NUT-0:

Following known undesirable “features” in NUT_0 have not been fixed yet.

Note: NUT-0 goal is only to exhume the old NUT code from VAX-VMS to UNIX.

- **Rectangular grid** is used. Grid is **not adaptive**. Plasma map is awkward.
- Z-symmetry is assumed for plasma.
- Atomic physics and wall reflection model are weak and old.
- No “allocatables” are used, so code **MUST** be recompiled for each case.
- “Common blocks” are used internally (**ugh!**).
- Transportability of code and scripts has not been tested fully yet.
- Only “real” **precision** is used, no proviso for double precision.
- Old code has many **historical fossils** that will be removed in NUT-1.

LOGICAL IMPROVEMENTS TO BE MADE STARTING FROM NUT-0:

- Remove Z-symmetry requirement on plasma.
- Non-rectangular grid, preferably same as used in master transport code, so that the plasma mapping is trivial and simple.
- Allow multiple plasma (neutral) species to run simultaneously.
- Allow ionization from partially excited neutrals within the Green function, instead of making user change cross sections.
- Add neutral gas feed and pump models, rather than user specifying the “external” neutral source.
- Adaptive grid for additional speed, if necessary.
- Explore the use of svd inversion rather than iterator for Green’s function.

STRUCTURAL IMPROVEMENTS TO BE MADE:

- Use allocatables to get rid of step 1, where dimensions get hard-wired.
- Replace all common blocks with modules. Note: this step **REQUIRES** the previous step where parameter statements are written off.
- Centralize precision specification to one module so that user can change between single and double (or any other) specification.
- Add more machine-checks to “nut_install” and “Makefile” scripts.

EXECUTION PLAN, PRIORITIES, AND RESOURCE ALLOCATION:

Before anything else, we MUST get NUT-0 working on many platforms.

- Improve and test transportability of code and scripts.

Only then, we proceed with first level of improvements, one step at a time.

- Use allocatables and put everything in modules to simplify use.
- Replace all common blocks with modules.
- Remove Z-symmetry requirement on plasma.
- Improve atomic physics and wall reflection model.

Then these more difficult tasks to widen NUT applications scope.

- Use non-rectangular grid. This one is high priority.
- Add neutral gas feed and neutral pumping models.
- Incorporate neutral-neutral interactions via a fluid model.

Finally, we may become cautiously ambitious, but only as needed.

- Use adaptive grid, but **ONLY IF** speed is deemed inadequate. Given the inherent speed of NUT, adaptive grid may not be necessary. [Back to top](#)

USER MANUAL: FOR REGULAR NUT USERS:

Once you have mastered the basic use of NUT-0, you can try the following:

- **Compiling NUT-0 for your own machine/simulation:**

The steps are essentially the same as for the demo, the only difference is you start with a different grid dimension control file called **nut_<casename>_dim.dat**. (in place of nut_demo_dim.dat)

Step 1. Copy nut_demo_dim.dat and edit it for your case, say nut_tftr.dat.

Note that the grid should be chosen as coarse as possible. Typically, you need to resolve only the plasma scales, not the neutral scales.

Always try to get away with as coarse a grid as possible (speed~□⁶).

Note: This step will be removed in the next NUT version by using allocatables.

The nut_<casename>_dim.dat file contains the following namelist:

\$nutstart

XYZ Grid array sizes:

mx	= 6,	(from -mx to +mx)
my	= 6,	(from -my to +my)
mz	= 4,	(from -mz to +mz)

Secondary ray settings:

ms	= 6,	(maximum # of secondary points on a "ray")
----	------	--

Wall settings:

mw	= 24,	(number of wall points in each z plane)
mww	= 4,	(maximum length of a "ray" along wall)
xwal	= 24*0.0	(x locations of wall points)
ywal	= 24*0.0	(y locations of wall points)
thwal	= 24*0.0	(angles of wall segments)
walcntrl	= 0.0	(type of wall, only for advanced users)

The following settings may be left alone for now (for advanced users)

Auxiliary settings for "flux surfaces"

mf	= 2,
fscntrl	= -1.0
fsparm	= 0.0
mr	= 0
r	= 0.0

Auxiliary settings for "neutral beams"

mseg	= 13
mbeam	= 1

Auxiliary settings for "Bolometer and other detector arrays"

mbol	= 10
mdcx	= 10
mhal	= 10

percent	= 0.5	!positional accuracy
---------	-------	----------------------

\$end

[Back to User Manual Top](#)

Step 2. (This step is optional, do this ONLY if you need a new plasma shape)

Copy nut_demo_fs.f to nut_<casename>_fs.f and edit it for your case.
The NUT0 distribution (nut0.tar) comes with a few **samples**: Nut_cyl_fs.f (cylindrical), Nut_shafr_fs.f (Shafranov-shifted fs), and nut_div_fs.f (divertor).

This is the only external routine that a non-advanced NUT user may supply.

- This routine allows you to define any arbitrary “flux surface shape”.
- Each flux surface is identified by a “radial” coordinate r , and you can set the plasma as a 1-d profile on these flux surfaces.
- You can have many such routines and pick the one you wish to use when recompiling NUT-0.
- When modifying this routine, YOU MUST KEEP THE SAME NAME, CALL PARAMETERS AND DIMENSION STATEMENTS. You can use the “FSPARM” array to pass any number of parameters to your routine.
- Your routine MUST convert (XY to RTH) or (RTH to XY), CONTROLLED BY “GIVEN”
- X,Y and R, Th are single real values, not arrays.

Below is an example of cylindrical flux surfaces with no shafranov shift.

```
C      USER SUPPLIED FLUX SURFACE SHAPE ROUTINE FOR NUT
C      IF YOU MODIFY THIS ONE, YOU MUST RECOMPILE NUT COMPLETELY.

C      THIS ROUTINE SPECIFIES FS SHAPE BY GIVING BOTH (XY TO RTH) AND
C      (RTH TO XY) TRANSFORMATIONS. CONTROLLED BY "GIVEN"
C      NOTE THAT THE LENGTHS X,Y,R ARE DIMENSIONLESS, NOT IN METERS !
C      THETA IS IN RADIANS

C      IN THIS ONE, SINCE FS SHAPE IS CIRCULAR, FSPARM() ARE NOT USED,
C      HOWEVER, IF THE FS SHAPE WERE DIFFERENT, YOU COULD USE THEM,
C      FOR EXAMPLE, TO SET THE ELLIPTICITY, YOU COULD USE FSPARM(1)

SUBROUTINE NUT_USER_FS_SHAPE( R, TH, X, Y, GIVEN, FSPARM)
REAL FSPARM(1)
CHARACTER*(*) GIVEN

C      "GIVEN" decides what is given:
IF((GIVEN.EQ.'XY').OR.(GIVEN.EQ.'xy')) THEN      !XY TO RTH
    R = SQRT(X*X + Y*Y)                          !RADIUS OF FS AT IX,IY
    IF((X.EQ.0).AND.(Y.EQ.0)) THEN
        TH = 0.0                                !CENTER HAS THETA = 0
    ELSE                                           !THETA GOES FROM -180 TO 180 DEGREES
        TH = ATAN2(Y,X)
    END IF
ELSE
    X = R * COS(TH)                              !RTH TO XY
    Y = R * SIN(TH)
END IF

RETURN
END
```

[Back to User Manual Top](#)

Step 3. Copy nut_demo.dat to nut_<casename>.dat and edit it,

This file is read by NUT at runtime, so

The settings in this file can be changed later without recompiling NUT.

Here is the nut_demo.dat file:

```
$NUTDFLT
PROMPT = 1.           Higher number gives more interactive prompts, 0 for none.
USERNAME = 'PMV'
MACHINE = 'TEXT(CYL)'
DISTYP = 'LIMITER'    Type of discharge
UNITS = 'MKS-19-EV'
TYPLAS = 1            Plasma type, 0: arbitrary, 1: tokamak
TYPWAL = 3.           Wall type,
RWAL = 0.30           Wall radius
RLIM = 0.26           Limiter radius
RMAJ = 1.0            Major radius of plasma
RCOEF = 0.0           Overall wall reflection coeff
APLAS = 1.            Plasma ion atomic weight
ZPLAS = 1.            Plasma Zeffective
AWAL = 52.            Wall material A (Iron)
ZWAL = 26.            Wall material Z
ABEAM = 1.            Beam ion A
ZBEAM = 1.            Beam ion Z
BENERGY = 25000.      Beam energy (eV)
DIO = 5.0             Central plasma density (10^13/cc)
DIW = 0.05            Edge plasma density (10^13/cc)
DIA = 1.              Plasma radial profile coefficients (see routine)
DIB = 0.
TIO = 800.            Central ion temperature (eV)
TIW = 5.
TIA = 2.
TIB = 0.
TEO = 1000.           Central electron temperature (eV)
TEW = 5.
TEA = 2.
TEB = 0.
TAUP = 6.E-3          Particle confinement time (sec)
TAUE = 1.             Energy confinement time (sec)
SLIM = 1.             Limiter source
SWAL = 0.01           Wall source
SDIV = 0.             Divertor source
SMEAN = 1.            Average source
BETASRC = 0.9         Source angular distribution 1+beta*cos(theta)
$END
```

Step 4. Type “nut_install <casename>” to install your new version of NUT-0.

This will create a new executable called **nut0_<casename>**,

an archive of all compiled Nut0 routines called **nut0_<casename>.a**

and ask you if you want to run nut (you are expected to say yes :-)

Note: At this point, you can look at the “nut_install” script to see what it does. Some auxiliary features from old NUT will be removed/replaced in the next version. Although the core NUT routines are self-contained, the auxiliary routines are useful during **initial and stand-alone use** of NUT – they allow easy setup of standard cases, plotting outputs etc. [Back to User Manual Top](#)

- **Running NUT_0 for your machine/simulation/transport_code:**

Typical sequence of steps when running **nut0_<casename>**:

1. Start nut, i.e, set up all pointers, defaults etc.
2. Start nut diagnostic section (if needed).
3. Give nut the plasma parameters.
4. Give nut the external sources (wall, beam etc).
5. Calculate by iterations.
6. Calculate secondary diagnostics (bolometers, halpha etc).
7. Plot/print/save answers.
8. Change nut single parameters and/or arrays.
9. If plasma stays same but external sources change, goto 3.
10. If plasma changes, goto 2.

This translates into the following calling sequence (see nut_demo.f):

1. call **nut_start**(nut_inf, nut_param, nut_names, delta,nut_r, nut_rxy,
nut_xwal, nut_ywal, nut_rwal, nut_thwal, nut_error)
2. call **nut_start_diag**(nut_dinf, nut_plotpar, nut_beampar, nut_bolpar,
nut_halpar)
3. call **nut_plasma**(nut_di, nut_te, nut_ti, nut_di2, nut_te2, nut_ti2, nut_error)
4. call **nut_wall_source**(nut_ewal, nut_swal, betasrc, nut_hip0, nut_phiw0,
nut_smean)
5. call **nut_calculate**(nut_iter, nut_hip, nut_d0, nut_p0, nut_q0)
6. call **nut_start_plot**(nut_plfnum, 'nut_demo_plot.dat')
7. call **nut_plot**(nut_plfnum, nut_plnum, nut_rxy, nut_hip0, nut_hip,
nut_phiw0, nut_d0,nut_p0, nut_q0, nut_bolsig, nut_halsig)

Notes:

STEP # 5 (IN RED) IS THE CORE NUT CALCULATION. Everything else is auxiliary.

- Detailed information on each routine is in the program file nut0_subs.f
- The nut_plot routine allows interactive plotting of nut answers.

Total time for a complete simulation of fully non-symmetric (in ϕ and z) neutral source in a tokamak (TEXT) is ~ 0.2 seconds on a 600 MHz Linux PC. This simulation used grid spacing of 5 cm, but got accurate neutral profiles.

[Back to User Manual Top](#)

USER MANUAL: FOR ADVANCED USERS (NUTS :-)

This section shows you how to modify some internal NUT-0 routines to suit your needs. Of course, before doing this, I strongly advise that you should:

1. Identify your specific needs that cannot be met with existing NUT-0 features,
2. Check with me if I am already implementing similar modifications, and
3. Be familiar with NUT-0 code structure (see NUT-0 detail section below).

Note : At any stage, if you get (`nut_error > 0`) do not proceed. You can see what the error means by "call `nut_error_message(nut_error)`"

Nut atomic physics (cross-sections) is all in one routine:

This is the ONLY atomic physics routine in NUT.

This is done make it easy to change this routine.

It will be upgraded in later versions.

You can change this if you wish.

subroutine `nut_imfp (eneu, diu, teu, tiu, delu, a, ur, ud, ut, at, ax, ai)`

```
c      calculate inverse mean free path at one point and one energy
c      any change in <physics> can only go here
c      all lengths converted from user units to units of del (grid size)
c      ref 1 : s.tamor, antic : j. comput. phys., 40, 104 (1981)
c      note : all quantities are converted to mks_ev units using the
c              conversion factors ur,ud,ut, and then the cross sections
c              are computed.
c      note : the formulae 19-22 in antic are in cm. we use m. so the
c              cross sections are reduced by 4 orders (from cm**2 to m**2)
c              the inverse mean free paths are returned here in
c              dimensionless units by multiplying by length delta.
c      input : energy of incoming neutral eneu
c              target plasma parameters : diu, teu, tiu
c              delu : lattice length unit
c      note : all above are in user's favourite units
c              ur : converts lengths from user units to meters
c              ud : converts density from user units to 1/m**3
c              ut : converts temp/ene from user units to ev
c              a = atomic weight of neutral
c      output: inverse mean free paths are calculated in 1/m for
c              neutrals of energy ene (converted to ev). they are multiplied by
c              delta (converted to m from user units) and returned as
c              dimensionless numbers.
c              at : total inverse mean free path (imfp) (in units of del)
c              ax : charge exchange collision imfp (in units of del)
c              aii : ion-ion collision ionization imfp (in units of del)
c              aie : ion-elec collision ionization imfp (in units of del)
c              ai : total ionization inverse mean free path (imfp)
c      if((eneu*diu*teu*tiu).eq.0.0) return !all must be non zero
```

```

di = ud * diu          !convert from user units to mks-ev
te = ut * teu
ti = ut * tiu
ene = ut * eneu
del = ur * delu

pi = 3.1415926585
epls = (ene + 4.*ti/pi)/a          !eq.19, ref.1
eplog= alog10(epls)
aoe = sqrt(a/ene)
w   = aoe * sqrt(epls)
telog= alog10(te)

c   rate of ionization by cx with ions * density of ions = imfp(cx)
ax = del* di* 6.94e-19 * w * ((1. -0.155*eplog)**2) /
$   (1. + 1.12e-15* epls**3.3)          !eq.20, ref.1

c   rate of ionization by electron impact * electron density = imfp
c   (electron density is same as ion density (charge neutrality) )
aei = del* di * 7.25e-11 * aoe *
$   ( 10**(-( 0.515*telog +2.56/telog +5.231)) ) !eq.21, ref.1

c   rate of ionization by ion impact * density of ions = imfp(ion-ion)
aai = del* di * 1.51e-20 * w *
$   ( 10**(-(1.15* ((4.4-eplog)**2))) )          !eq.22, ref.1

c   total mfp is smaller than its parts, so add the inverse mfp's
ai = aei + aai          !ionization inverse mean free path
at = ax + ai          !total inverse mean free path

return
end subroutine

```

More details about NUT USER callable routines and calling order:

Note : All common blocks will go away in next NUT version. However, for now, after each routine is called, all nut routines know the answers calculated by that routine. This is done via internal common blocks /nuti1com/, /nuti2com/ etc, contained in nuticom.for. You must not use these names for your common blocks. Do not include nuticom.for in your routine unless you know all details of nut system.

c User callable main subroutines of the nut system :

- c all these get/set all information directly via the arguments
- c and not via any common blocks. you get what you see.

- c any common blocks you see in them when you look closely
- c are for internal nut communications. they do not (and must not)
- c communicate with the outside world. they all start with /nut../

- c so you should avoid this name conflict.
 - c these routines do not have any diagnostic modules. they are all in
 - c "nutdiag.f". these only solve the nut problem.
 - c for each routine, there are :
 - c a list of direct inputs in arguments,
 - c a list of relevant parameters passed to it internally by common
-

Steps in calling NUT (details):

Step 1. Before calling any routines in "nut", some initializations have to be done, and some geometrical information has to be calculated. to do this make the following call :

**subroutine nut_start(fname, nut_param, nut_names, nut_delta,
& nut_r, nut_rxy, nut_xwal, nut_ywal, nut_rwal, nut_thwal, nut_error)**

- c note that this call has to be made before all other "nut" calls.
- c starts the whole nut system
- c inputs : default data filename fname
- c params : none.
- c output : values of parameters and names, grid constant delta,
- c geometrical arrays nut_delta, nut_xwal, nut_ywal, nut_rwal, nut_thwal
- c if default file not found, nut_error=1 is returned so
- c operation can be terminated by your program, or whatever

Note: If you plan to use any diagnostics in nut, you first need to call:

**subroutine nut_start_diag(fname, nut_plotpar,
& nut_beampar, nut_bolpar, nut_halpar)**

- c get diagnostic parameters for nut and give them to user.
 - c call this routine only if you wish to use any diagnostic
 - c facilities of nut such as plot, beam, bolometer, halpha etc.
-

Step 2. Next, the plasma profiles are given to nut by calling

**subroutine nut_plasma(nut_di, nut_te, nut_ti,
& nut_di2, nut_te2, nut_ti2, nut_error)**

- c purpose: get plasma profiles from proper source and put them in
- c form usable by nut, then perform all preliminary
- c calculations which require these profiles.
- c relevant control parameters :
- c typlas : plasma type control switch (see below)
- c profile parameters di0, diw, dia, dib, ti0, tiw, tia, tib
- c plasma coefficient file, data file, or database names

```

c      if typlas = 1 or -1, profiles are given by user
c          if typlas = 1, 1d profiles are given by user
c          if typlas = -1, 2d profiles are given by user
c      otherwise, this routine gets the profiles and returns them to you
c          if typlas = 0, 1d profiles are made from coefficients
c          if typlas = 2, 1d profiles are read from data file
c          if typlas = -2, 2d profiles are read from data file
c          if typlas = 3, 1d profiles are read from data base
c          if typlas = -3, 2d profiles are read from data base
c          nut_error if plasma is not found
c      After this call, all NUT routines know the plasma profiles

```

Step 3. Specify the external sources (plas and wall) (phip0,phiw0) for nut

Since each of the following keep adding to the existing source already given to nut,
You can (and should) zero the nut source at the beginning by calling

call nut_zero_source(nut_phip0, nut_phiw0)

you can either add an arbitrary source by calling

call nut_arb_source(nut_phip0, nut_phiw0)

```

c      purpose: set arbitray external sources
c      inputs : user supplied arbitrary external sources
c          nut_phip0(-mx:mx,-my:my,0:mz) : source in plasma
c          nut_phiw0(mw,0:mz,mwe) :external source on wall at all energies
c      output : none. these sources are just given to nut internal arrays

```

or you can add a wall source by calling

call nut_wall_source(nut_ewal, nut_swal, nut_phiw0)

```

c      purpose: set given wall source into nut internal arrays
c      inputs : limiter and wall energies and sources
c      output : nut_phiw0(mw,0:mz,mwe). these values are passed to nut.
c          and average wall source

```

or you can add a beam source by calling

**call nut_beam_source(nut_bcur, nut_phip0,
& nut_beamamp, nut_bneusrc, nut_bionsrc)**

```

c      purpose: to insert a neutral beam source and give it to nut
c      inputs : beam currents nut_bcur(mbeam) in each beam chord (amps)
c          if no nut_bcur is given, it uses parameter(7) or (8)
c      meaning of beam parameters (same table at top of this file) :
c          1,2,3 : x,y,z of beam entry point into plasma
c          4,5,6 : x,y,z of beam exit point from plasma
c          7,8   : beam current, voltage
c          9,10  : beam z and atomic weight
c          11    : track step size (calculated by nut)
c          12    : beam power in watts (if pow=0, current is used)
c      output : nut_phip0 with beam sources are passed to nut

```

Step 4. Calculate the source by calling the core iterating routine This step is the only core nut call, all others are auxiliary.

- call nut_calculate(nut_iter, nut_phip, nut_phiw)**
- c purpose: calculate total neutral source by iteration
- c inputs : number of iterations nut_iter,
- c if nut_iter>0 start new iteration, else continue old one
- c params : total particle confinement time taup,
- c if taup>0, integral of source is matched to it
- c if taup<0, maximum of source is matched to it
- c output : calculated values of nut_phip,phiw,d0,p0,q0
1. All **green's functions depending on plasma are calculated** if it has not been done already.
 2. Then a fixed number (nut_iter) of iterations are done on the basic equation $\phi = \phi_0 + g * \phi$. If nut_iter is set negative, the current iteration sequence is continued. If it is + a new iteration is started by setting $\phi = \phi_0$ at the first step
 3. Finally the secondary answers nut_d0, nut_p0, nut_q0 are then calculated.

Note: Before you call any following diagnostic routines, you must call (see step 1):

**subroutine nut_start_diag(fname, nut_plotpar,
& nut_beampar, nut_bolpar, nut_halpar)**

Step 5. Calculate various diagnostic signals to compare with data, such as

- a. Bolometer signals in all bolometer detectors

call nut_bolometer(nut_bolsig)

c purpose: calculate signals seen in bolometers

c inputs : none

c output : nut_bolsig(mbol) in bolometer detectors
- b. Halpha signals in all halpha detectors

call nut_halpha(nut_halsig)

c purpose: calculate halpha signals seen

c inputs : none

c output : nut_halsig(mhal) in halpha detectors

Step 6. To see nut outputs, there is a menu driven plot routine.

- call nut_plot(nut_rxy, nut_phip0,nut_phip, nut_phiw0,
& nut_d0,nut_p0,nut_q0, nut_bolsig,nut_halsig)**
- c purpose: interactively plot many nut plots
- c inputs : none
- c output : produces a gnuplots file that can be rendered with
- c scripts GetPlot or Plot on screen
-

Extra steps (recommended only for advanced users).

You can interactively change/view/save nut parameters or names by

```
*****
call nut_modify_parms( nut_param)
call nut_modify_names( nut_names)
call nut_save_default( nut_defile)
*****
```

or you can change/view one nut parameter or name by calling

```
*****
call nut_change_one_param( name, nut_param, value)
call nut_change_one_filename( name, nut_names, value)
*****
```

the list of parameters and names is given at the end of this file
The core NUT routine nut_calculate can be called in a loop inside a plasma simulation. Only the plasma has to be reset every time. The machine and diagnostics geometry remain fixed.

Extra Notes:

1. You have a **choice of units**. The default units are mks-ev, which means all quantities in mks, except energies and temperatures in ev. You can choose other units by setting the (character type) parameter units = 'mks-kev', 'mks-19-ev', 'mks', 'cgs', 'cgs-13', 'cgs-13-ev' etc. i.e, any combinations (in any order, with or without the dash -) of mks, cgs, ev, kev, 13, and 19. The 13 and 19 stand for 1e13 or 1e19 factors for the densities in cgs or mks. For example, if you say mks-19-kev, the temperatures will be in kev, densities in units of 1e19 / m**3, and length in meters etc.

Of course, you better give the data in the same units as you declare !

nut can do this easily because it calculates all lengths in dimensionless units by dividing by the lattice spacing delta. Thus, the only routine requiring care is the one that calculates cross sections

2. The **physical cross sections** are calculated by the internal subroutine call **nut_cross_sections** which calls **imfp**(e, di,te,ti, at,ax,ai, delta) (imfp == inverse mean free path) this "imfp" is the only official subroutine making these calculations to be consistent, always use this routine for cross sections. Any changes in physics should be made to this one only.

3. The **standard physical constants** such as pi are in the include file "**constants.i**" in the form of data statements. Use the same file for consistency.

C Standard values of physical and mathematical constants for NUT (mks)

```
data pi/3.141592654/
data kboltz/1.3807e-23/
data vlight/2.9979e8/
data emass/9.1095e-31/
data pmass/1.6726e-27/
data amu0/1.2566e-6/
data echarge/1.6022e-19/
data epsilon0/8.8542e-12/
data denmks/1.0e19/
data dencgs/1.0e13/
```


4. The names of **internal parameters** used by nut

To access them by name use these nut routines :

You can interactively change/view/save ALL nut parameters or names by calling

CALL NUT_MODIFY_PARM(NUT_PARAM)

CALL NUT_MODIFY_NAMES(NUT_NAMES)

CALL NUT_SAVE_DEFAULT(NUT_DEFILE)

Or change/view ONE nut parameter or name by calling

CALL NUT_CHANGE_ONE_PARAM(NAME, NUT_PARAM, VALUE)

CALL NUT_CHANGE_ONE_FILENAME(NAME, NUT_NAMES, VALUE)

C NUT_PARM(100) ARE PARAMETERS USED IN NUT.

C # | NAME | DESCRIPTION OF PARAMETER

C|.....|.....

C 1 TYPLAS PLASMA INPUT TYPE

C 2 TYPWAL WALL CALCULATION TYPE

C 3 DIO CENTRAL DENSITY

C 4 DIW WALL DENSITY

C 5 DIA DENSITY ALPHA

C 6 DIB DENSITY BETA

C 7 TIO CENTRAL TION

C 8 TIW WALL TION

C 9 TIA TION ALPHA

C 10 TIB TION BETA

C 11 TEO CENTRAL TELECTRON

C 12 TEW WALL TELECTRON

C 13 TEA TELECTRON ALPHA

C 14 TEB TELECTRON BETA

C 15 APLAS PLASMA ATOMIC NUMBER

C 16 WPLAS PLASMA ATOMIC WEIGHT

C 17 TAUP PARTICLE CONFINEMENT TIME (MS)

C 18 TAUE ENERGY CONFINEMENT TIME (MS)

C 19 SLIM EXTERNAL SOURCE AT LIMITER

C 20 SWALL EXTERNAL SOURCE AT WALL

C 21 SDIV EXTERNAL SOURCE AT DIVERTOR

C 22 SMEAN AVERAGE EXTERNAL SOURCE AT WALL

C 23 NUMXPL NUMBER OF POINTS ON X AXIS

C 24 NUMYPL NUMBER OF POINTS ON Y AXIS

C 25 XPLMIN MIN X

C 26 XPLMAX MAX X

C 27 XPLTIC XTICKS

C 28 YPLMIN MIN Y

C 29 YPLMAX MAX Y

C 30 YPLTIC YTICKS

C 31 ZPLMIN ZMIN

C 32 ZPLMAX ZMAX

C 33 ZPLTIC ZTICKS

C 34 XPLSCL SCALE ON X AXIS (X(I)/XSCL IS PLOTTED)

C 35 YPLSCL SCALE ON Y

C 36 ZPLSCL SCALE ON Z

C 37 XPLEVL DRAW A LINE AT THIS X LEVEL

C 38 YPLEVL

C 39 ZPLEVL

C 40 PERCENT PERCENT ACCURACY NEEDED IN ALL CALCULATIONS

C 41 NUMBOL NUMBER OF BOLOMETER DETECTORS

C 41 NUMHAL NUMBER OF HALPHA DETECTORS

C 42 NUMDCX NUMBER OF DCX DETECTORS

C 43 NUMBCH NUMBER OF BEAM INPUT CHORDS

C 44 BENERGY BEAM ENERGY IN KV

C 45 ABEAM ATOMIC NUMBER OF BEAM

```

C  NUT_NAMES(100) ARE CHARACTER*100 CONSTANTS USED IN NUT.
C  # | NAME      | DESCRIPTION OF NAME PARAMETER
C  ....|.....|.....
C  1  DFLTFIL -   - NUT DEFAULT FILENAME
C  2  USERNAME-   - USER NAME
C  3  MACHINE -   - MACHINE NAME
C  4  DISTYPE -   - DISCHARGE TYPE (LIMITER OR DIVERTOR)
C  5  PLCOEF -   - FILE FOR PLASMA COEFFICIENTS
C  6  FSPROF -   - FILE FOR FLUX SURFACE (R) PLASMA PROFILES
C  7  XYPROF -   - FILE FOR XY PLASMA PROFILES
C  8  DATABASE-   - NAME OF DATABASE
C  9  BOLINF -   - FILE FOR BOLOMETER INPUT
C 10  BOLOUT -   - FILE FOR BOLOMETER OUTPUT
C 11  DCXINF -   - FILE FOR DCX INPUT
C 12  DCXOUT -   - FILE FOR DCX OUTPUT
C 13  HALINF -   - FILE FOR HALPHA INPUT
C 14  HALOUT -   - FILE FOR HALPHA OUTPUT
C 15  BEAMINF -   - FILE FOR BEAM INPUT
C 16  BEAMOUT -   - FILE FOR BEAM OUTPUT
C 17  ANTINF -   - FILE FOR ANTIC INPUT
C 18  ANTOUT -   - FILE FOR ANTIC OUTPUT
C 19  PLOTINF -   - FILE FOR PLOT INPUT
C 20  PLOUTOUT - - FILE FOR PLOT OUTPUT
C 21  XAXIS  -   - XAXIS NAME
C 22  YAXIS  -   - YAXIS NAME
C 23  ZAXIS  -   - ZAXIS NAME
C 24  TITLE  -   - TITLE ON PLOT
C 25  LEGEND1 -   - LEGEND 1 ON PLOT
C 26  LEGEND2 -   -
C 27  LEGEND3 -   -
C 28  UNIT   -   - UNITS TO BE USED
C-----

```

5. Customizing "nut" for the geometrical shape of your machine :

You can customize the "nut" system to suit the geometry of your machine. Then, with that geometry hardwired, all "nut" routines have to be recompiled into a library called "nut.a". Once this is done, any nut routines can be called by your program.

this is done by :

1. Editing the data file "nut_demo_dim.dat" ,
2. Editing the subroutine "nut_demo_fs.for" ,
3. Running the make

Step 1. Edit the data file "nut_demo_dim.dat". You can name it anything else.

use that name when you run the command file nut. For example, if you name it nut_55_dim.dat (for a 5 x 5 grid), then type "nut_install 55".

This make <casename> = "55"

I will henceforth refer to this file as nut_demo_dim.dat

It specifies the following numbers which are read into a namelist/nutstart/, which is what you supply in the above file.

the namelist format makes editing nut_demo_dim.dat easy and understandable.

note : for later use, you may want to save your current version of

nut_demo_dim.dat before editing it. Give it a good name you can relate to.

note : all lengths at this point are dimensionless. The actual sizes

will be specified only when you run the program, not when you compile it

this section only needs to know the shapes, not sizes.

C data file nut_demo_dim.dat contains the namelist
 C namelist/nutstart/ mx,my,mz, ms, percent,
 C & mr,mf,r, rmaj,rwal,rpw,walcntrl,mw,mww,xwal,ywal,thwal,
 C & fscntrl,fsparm, mbch,mbol,mdcx,mhal
 C
 C note : z refers to the toroidal direction, xy to poloidal plane
 C
 C mx, my, mz : half grid sizes in x,y, and z directions (5/20/6)
 C all grid arrays are (-mx:mx, -my:my, 0:mz)
 C ms : max number of secondary points on each ray (5/20/6)
 C the ray points go from 0 to ms. (0 is the target)
 C
 C percent : percent accuracy in geometrical specs
 C
 C fscntrl : flux surface input information format switch
 C
 C if fscntrl=0, routine assumes mr and r(mr) are given
 C in the data file.
 C
 C note : if fscntrl < 0, mr is set by the program
 C else it uses the mr given.
 C
 C if fscntrl=1, routine finds all distinct radii for each
 C xy using delta and the user supplied routine
 C "nutfsuser.for". It then chooses mr radii out of these
 C so that the cross sectional area of each zone is same.
 C
 C if fscntrl=-1, routine finds all distinct radii for each
 C xy using delta and the user supplied routine
 C "nutfsuser.for" and uses all of them. It sets mr.
 C
 C note : for the above cases where abs(fscntrl) < 2 ,
 C the user supplied routine "nutfsuser.for" is needed
 C
 C if fscntrl=2, routine assumes that an array
 C rxy(-mx:mx,-my:my) has been given in the above data
 C file and uses it to set mr flux surfaces.
 C
 C if fscntrl=-2, routine assumes that an array
 C rxy(-mx:mx,-my:my) has been given in the above data
 C file and uses all distinct radii for flux surfaces.
 C it sets mr in this case
 C
 C fsparm : array of flux surface shape (not size)
 C parameters. This array is passed to the user supplied
 C fs_shape routine "nut_user_fs_shape".
 C you can use these numbers to pass shape information
 C such as ellipticity to nut. Remember, this array should
 C contain only shape, not size information.
 C fsparm(1) contains number of parameters. If fsparm(1) = 0,
 C there are no flux shape parameters, i.e., it is cylindrical.

```

C
C   mr   : number of radial flux surface zones.
C   mf   : number of subzones for each zone      (1/10/5)
C         this gives extra resolution for plasma,acx,and atot
C   r(mr) : array of flux surface radii. R(mr) (dimensionless)
C   rxy   : array of radii at xy point. (-mx:mx,-my:my) array
C
C wall parameters :
C   walcntrl : control parameter for wall
C             if walcntrl = 0, wall is outermost flux surface with
C                 equal angular spacing  $2 * \pi / mw$ 
C             if walcntrl = 1, wall is outermost flux surface with
C                 angular locations thwal(mw) given by user
C             if walcntrl = 2, wall point xy are given by user
C   mw      : number of wall points              (8/256/32)
C   mww     : max wall to wall distance         (2/64/8)
C   xwal    : array of x co-ordinates of wall points: xwall(mw)
C   ywal    : array of y co-ordinates of wall points: ywall(mw)
C   thwal   : array of th co-ordinates of wall points: thwall(mw)
C
C parameters for auxiliary calculations :
C   mbe     : number of beam energy components      (1/10/4)
C   mbch    : number of beam chords                (1/9/100)
C   mbol    : number of bolometer detectors         (1/100/20)
C   mdcx    : number of double-charge-exchange detectors (1/100/20)
C   mhal    : number of halpha detectors            (1/100/20)
C.....

```

Step 2. The subroutine "nut_demo_fs.f" defines shape of the flux surfaces
it is used only if flux surface shape is given by a function, i.e.,
only if "fsparm(1)" is set to 1 or -1. It can be edited by user as long as the form of the
input/output arguments is maintained

Step 3. A. Run "nutdimen.exe" to create "nutstdim.for" and then
b. Run "nutstart.exe" to create the three include files,
nutucom.for, nuticom.for and nutidata.for
c. Recompile all routines in nutsubs.for into the library nut.olb
or your own library.

After this step, the geometry is hardwired into the code, and can not
be changed. Essentially, this means the maps between the xy grid and
the flux surfaces are fixed at this point. You have now customized
"nut" for your machine.

This geometrical hardwiring is at the core of nut. It allows nut
to turn all multidimensional arrays into single dimensional ones
and use vectorizable do loops for all calculations. That means speed.

Of course, if you want to change the geometry, you need to recompile.
that is the price you pay for speed. However, once you settle on
a good choice of geometry for your machine, you need not
recompile nut.

NUT_0 CODE DETAILS FOR ADVANCED USERS AND NUT CODE WRITERS :

- All internal units are mks, all temperatures are in electron volts (ev), user can pass data in many different units. (cgs,kev,10**13,10**19 etc)
- All input/ outputs numbers are real (single precision), and
- All names are character*80 strings.
- All runtime "nut" routines start with "nut0_". They are in one file nut_subs.f
- All user callable routines are at the beginning, followed by all internal routines and then incidental routines. Each is marked in a comment at top.
- Do not try to call any internal "nut" routines directly unless you know what you are doing.
- All dimensioned arrays needed to call "nut" are together in file "nut0_ucom.i" which you can include in your routine. All names in this file start with nut_ so that you can avoid conflicts with your programs.
- There are **NO USER ACCESSIBLE COMMON BLOCKS INSIDE NUT**. The internal common blocks of "nut" all start with nut_, so avoid this name for your common blocks. Try not to interact with nut via common. If you do, I will not help you sort out what happened.
- Whatever common blocks are used in this version were inherited from old NUT, and will be removed to be replaced by modules in the next version.
- All routines are written in fortran90, though I have not yet changed some of the old stuff to fortran90 style (mainly common blocks). Each routine contains a list of inputs, outputs, and some comments.

[Back to top](#)

APPENDIX A: NUT-0 core and auxiliary routines, commons, etc.

NUT-0 core stuff:

NUT-0 core routines and their arguments:

NUT-0 core common blocks (user should not use them):

NUT-0 auxiliary stuff:

NUT-0 input routines and their arguments:

NUT-0 output routines and their arguments:

NUT-0 plot routines and their arguments:

NUT-0 auxiliary diagnostic routines and their arguments:

APPENDIX B: NTCC Module Standards Compliance Checklist

Every attempt has been made to assure that NUT-0 is fully compliant with following NTCC module standards. This has been kindly checked by John Mandrekas, who I wish to thank for making numerous helpful suggestions that have now been incorporated. Please let me know if you run into any NTCC compliance issues for NUT.

Module Standards (from <http://w3.pppl.gov/NTCC/standards/standards.html>)
Version Date: Feb. 26, 1999

The tables below list ``Standards'' and ``Goals''. The ``Standards'' described below will be enforced for the modules submitted to the library; whereas, ``Goals'' are strongly encouraged and may become future standards for the module library.

General Standards:

- Standard: Provide source code for each physics module or code. (done)
- Standard: Provide test case(s) with driver program(s) with input and output data and their documentation. (done)
- 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. (done)
- 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. (done)
- 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. (done)
- Standard: Eliminate graphics calls embedded in physics modules. (done)
- Standard: The source code files (e.g., *.f, *.c or *.cpp files) should be submitted rather than requiring extraction from another file. (done)
- Standard: Authors may upgrade their modules with approval of the current chairperson of the NTCC modules committee. If the upgrade is extensive, the chairperson can require that the upgrade be subject to a full review. (first upgrade under progress)
- Goal: Portability (code should run on multiple platforms and under different operating systems). (done on 2 platforms)
- Goal: Offer single and double precision versions or offer user control of precision at compile time. (partially done but not fully)
- Goal: Multi-platform (code should run on different computers). (done on 2 platforms)
- Goal: Provide error checking (but not stops). (done)
- Goal: Minimize external dependencies that cost money(i.e., avoid using expensive proprietary licenses). (done)

- Standard: Supply warnings in the documentation when the above goal has not been met. (Not Applicable)
- Goal: Arrays should be dynamically allocated. (partially done)
- Standard: The characteristics of the I/O should be clearly documented (i.e., the implementation I/O unit numbers, if any). (done)
- 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. (done)
- Standard:

Documentation Standards:

- Standard: Provide Name of contact person for support. (done)
- Standard: Provide Date of last revision. (done)
- Standard: Provide at least comments describing module or code, citations to publications (if any), and range of validity. (done)
- Standard: Specify the precision of floating point calculations. (done *)
- Standard: Provide index of input-output variables for each module (include type of variable, dimensions, units). (done *)
- Standard: List dependencies -- names of external routines called. (done)
- Standard: Provide statement of known bugs. (done)
- Goal: Index of modules, routines, variables. (not done)
- Goal: Publication of code or module in journal (such as Computer Physics Communications). (not done)
- Goal: Online hyper-text reference documentation. (done)
- Goal: Interactive online help menus. (done)
- Standard:

Data Standards:

- Goal: Provide interface routines to data. (done)
- Goal: Use self-describing data files (such as NetCDF). (not done)
- Goal: Use public domain, portable, available and well-documented data file formats. (not done)
- Goal: Establish standards for variable names, units, dimensions, independent variables and grid descriptions as they appear in the module interfaces. (done)
- Standard:

Standards for Users:

- Standard: Users of NTCC codes and modules must acknowledge the author(s) and any modifications made when publishing or presenting results.
- Standard: Users who find bugs in NTCC codes or modules must report back to the contact person first.
- Standard: Users who make improvements to a module's source code, makefile, or documentation, will submit these improvements to the module's author or support person, who may choose to incorporate these improvements in a future release of the module.

APPENDIX C: Some “gotchas” on different machines

This list will keep growing as NUT is used on different machines.

Please see the **nut_install** script to find out which machines have so far been tried.

I provide a collection of makefiles in folder **makefiles**, (alpha, Killeen, winnt etc).

On your machine generally you only need to change following:

FC=name of f90 compiler (e.g., fort on alpha, f90 on Killeen, xfl on ...)

FFLAGS=(simplest) set of flags for fortran compiling

I am sure there will be other fortran compiler issues with specific flags.

The routine is vectorizable, but I have not tried that feature out yet on killeen, say.

The other issue is availability of gnuplot. Version 3.7 or higher works best. 3.5 also works with a small modification in nutpl (see plotscripts/Killeen).

Here are some “gotchas” I have found so far. Please email any insights you find to pvalanju@mail.utexas.edu, and I will include them here. Thanks.

On some machines, you may need to:

- Use **gmake** rather than **make** to use my makefiles.
- Activate **gnuplot**, say by typing **module load gnuplot**. [Back to top](#)

APPENDIX D: Some changes made from nut0 to nut0.1

Changes made to nut0 from Jan-Mar 03 by Prashant Valanju,
With many important suggestions from John Mandrekas and Jim Wiley.
Thanks to both. - P. Valanju, March 2003

Summary:

1. Combined the two steps in geometrical pre-processor to one step.
2. Restructured input parameters to separate geometry from physics.
3. Simplified wall and plasma geometry specifications.
4. Removed all unit numbers in open file statements in nut0.
5. Used allocatables in pre-processor. Not yet in core NUT0.
6. Reduced i/o and include files and streamlined their names.
7. Reduced number of include files to 3:
nut0_ucom.i and for user, and nut0_prec.i to set precision
nut0_icom.i for general internal use.
Others are used only in specific routines.
8. Rewrote install -> nut0_install. Used "rules" for different machines.
9. Made dependancies explicit. nut0_install now quits if error happens.
10. Rewrote many parts of manual to reflect these changes.
11. Added and streamlined a lot of internal documentation in code.
12. Streamlined online nut0 webpage.
13. Added file structure, install steps, calling steps, and program flow diagrams.
14. All names (files, user variables, user commons, user subroutines etc)
now start with nut0_
15. Provision for changing precision (file nut0_prec.i).
16. Checked all items in NTCC standards list for nut0 compliance.
17. Added NTCC standards checklist to web and pdf manual.

New version is in nut0.1.tar

List of files in nut0.1.tar

(tar -xvf nut0.1.tar makes a new directory nut0):

nut0.1/

nut0.1/README

nut0_install Script to install nut0 in nut0/

nut0.1/nut0_make Standard makefile for nut0.1

nutpl Script to plot nut plots with gnuplot

nut0.1/demoplots Output demo plots, use with nutpl demoplots

*.dat data files in nut0/dat:

nut0.1/dat/

nut0.1/dat/nut0_demo_diag_in.dat

nut0.1/dat/nut0_demo_diag_out.dat

nut0.1/dat/nut0_demo_geom.dat

nut0.1/dat/nut0_demo_in.dat
nut0.1/dat/nut0_demo_out.dat

nut0.1/docs/

nut0.1/docs/nut0_ch_303
nut0.1/docs/NUT0Manual.pdf
nut0.1/docs/NUTPaper.pdf

nut0.1/fs/

nut0.1/fs/nut0_cyl_fs.f
nut0.1/fs/nut0_div_fs.f
nut0.1/fs/nut0_shafr_fs.f

nut0.1/rules/

nut0.1/rules/rules_alpha
nut0.1/rules/rules_linux
nut0.1/rules/rules_sun

***.f fortran code files in nut0/src/:**

nut0/src/
nut0/src/nut0_new_geom.f
nut0/src/nut0_subs.f
nut0/src/nut0_demo.f
nut0/src/nut0_fs.f
nut0/src/nut0_wdata.f

***.i include files in nut0/src/:**

nut0/src/
nut0/src/nut0_ucom.i
nut0/src/nut0_icom.i
- These use following includes:
nut0/src/nut0_constants.i
nut0/src/nut0_idata.i
nut0/src/nut0_ipar.i
nut0/src/nut0_upar.i

Details of changes, comments, and notes (as changes are made):

1. In nut0_install (renamed install to nut0_install to avoid inadvertant conflict between install and ./install)
 - a. Clarified usage, and meaning of each parameter
 - b. Removed killeen
 - c. Changed all names to start with nut0_
 - d. Fixed test ! -f
 - e. Added exits if error at each step.
2. In makefile
 - a. Added explicit dependencies for each *.f file

- b. Isolated all statements specific to each machine in
 RULES = rules/rules_sherlock
 include \${RULES}
 - c. Each RULES file includes flags FC, LD, FFLAGS, LDFLAGS, ARFLAGS
 - d. Have 2 RULES files so far: for SUN and Alpha Linux
3. New routine nut0_new_geom now combines two steps on nutstart and nutgeom
- a. Allocatables used.
 - b. New geometry parameters streamlined: No plasma parameters are here now.
4. In nut0_subs.f
- a. Made sure all user variable names are nut0_*
 - b. Removed all fixed open file unit numbers.
 Now user sets file names and numbers through 2 arrays:
 nut_fnames(5) : nut i/o file names and
 nut_fnums(5) : their corresponding unit numbers
 List of files that nut0 potentially uses nut_fnames(i):
 1. Input data file for nut (e.g., nut0_demo_in.dat)
 2. Output data file for nut (e.g., nut0_demo_out.dat)
 3. Input Flux surface profiles (e.g. nut0_demo_fs_in.dat)
 4. Input XY profiles (e.g. nut0_demo_xy_in.dat)
 5. Input file for auxiliary diagnostics.
 (e.g. nut0_demo_diag_in.dat).
 - c. Provided nut_prompt = 0 : silent mode, no prompts (i or o) from nut0
 - c 1 : Only major prompts
 - c 2 and higher : more detailed diagnostic prompts
 - d. Removed all interactive read/print calls.
 - e. Isolated all graphics calls.

REFERENCES

1. P. M. Valanju, *NUT: A Fast 3-Dimensional Neutral Transport Code*, J. Comp. Phys. **88**, 114 (1990).
2. NTCC Team, *NTCC Demonstration Project Web Site*, <http://electrojet.colorado.edu/wwwntc/> (1998).
3. D. Heifetz, D. Post, M. Petravic, J. Weisheit, and G. Bateman, *A Monte-Carlo Model of Neutral-Particle Transport in Diverted Plasmas*, J. Comp. Phys. **46**, 309 (1982).
4. W. L. Rowan, C. C. Klepper, D. M. Patterson, and B. Richards, *Recycling measurements in the Texas Experimental Tokamak*, J. Nucl. Mater. **145**, 562 (1987).
5. W. L. Rowan, C. C. Klepper, C. P. Ritz, R. D. Bengtson, K. W. Gentle, P. E. Phillips, T. L. Rhodes, B. Richards, and A. J. Wootton, *Global Particle Confinement in the Texas Experimental Tokamak*, Nucl. Fusion **27**, 1105 (1987).
6. R. D. Bengtson, P. M. Valanju, A. Ouroua, and W. L. Rowan, *Measurement of Neutral Density Profile in TEXT Using a Diagnostic Neutral Beam.*, Rev. Sci. Instrum. **61**, 3110 (1990).
7. J. A. Boedo, R. D. Bengtson, A. Ouroua, and P. M. Valanju, *Measurements of Neutral Density Profiles on the TEXT Tokamak*, Rev. Sci. Instrum. **59**, 1494 (1988).
8. B. C. Stratton, R. J. Fonck, A. T. Ramsey, E. J. Synakowski, B. Grek, K. W. Hill, D. W. Johnson, D. K. Mansfield, H. Park, G. Taylor, and P. M. Valanju, *Charge Exchange Recombination Spectroscopy Measurements in the Extreme Ultraviolet Region of Central Carbon Concentrations During High Power Neutral Beam Heating in TFTR*, Nucl. Fusion **30**, 675 (1990). [Back to top](#)