****

FPP DOCUMENTATION

****

(Source:fpp.doc)

(equivalently: transp$:[cmsref.fppdoc]fpp.doc)

Greg Hammett

Aug. 4, 1986

Last revised 31 July 1990

This document describes the generic features of the FPP Fokker-Planck Program which apply to both the standalone and the TRANSP versions of FPP. Features which are more specific to the TRANSP version are described in the file source:trfpp.doc.

****

I. INTRODUCTION

****

FPPRF is a computer program which solves a Fokker-Planck equation to simulate the fast ions produced by neutral beam injection or ICRF heating. It has evolved from an original version written by Rob Goldston in 1977 to study neutral beam injection, and has been extensively modified by Greg Hammett to handle ICRF heating, compression, and other improvements. It has been coupled with Dave Smithe's 3D ICRF wave code SPRUCE.

Two versions of FPP are supported from the same fortran source: a standalone version where the thermal plasma parameters (but not the fast ions) are assumed to be fixed in time, and a complete version coupled to

the time-dependent transport analsysis code TRANSP. This file documents the standalone version of FPP. TRFPP.DOC documents the TRANSP version.

The basic physics models and assumptions are outlined in the next section. References which contain more information are given at the end. FPPRF is a research tool, and is therefore always under development and testing. Caveat Emptor.

****

Physics models and assumptions

****

FPPRF solves the bounce-averaged Fokker-Planck equation to find the fast ion distribution function f(E,mu,r,t), as a function of energy E, magnetic moment (or equivalently, pitch angle) mu, minor radius r (generalized to non-circular flux surface shapes), and time t:

$$\frac{d\, f(E,mu,r,t)}{d\, t} = C(f) + Q(f) + S + R(f).$$

C is the bounce-averaged and linearized collision operator (self-collisions are ignored, and the background species are assumed to be Maxwellian) and includes energy slowing down, energy scattering, and pitch angle scattering. Q is the complete bounce-averaged quasilinear operator (similar to the operator used by Kerbel and McCoy) including E_plus and E_minus, finite k_parallel, and full k_perp rho effects in the Bessel Functions. It includes Airy function and k_parallel corrections to the wave-particle correlation time. S represents sources and sinks due to

neutral beam injection, charge exchange losses, and thermalization. R is a radial transport operator and has options for modelling radial transport of fast ions due to ripple transport and neoclassical transport, or an arbitrary transport level can be specified. Once f is known, the program then calculates such interesting quantities as charge-exchange spectra and neutron production rates, as well as collisional heating of the thermal ions and electrons.

FPP uses an operator-splitting algorithm (similar to the ADI, or alternating difference implicit, algorithm) to solve the 2-D convection-diffusion equation on each flux surace. It uses simple upwind-differencing of the drag terms, which results in a slight enhancement of the "temperature" of the fast ions of order dE/E, where dE/E is the relative spacing of the energy grid (dE/E = 10% is a typical value used for an FPP run). The cross-derivative terms (d/dE d/dmu) in the quasilinear operator are solved using a flux-limited algorithm related to Boris's flux-corrected transport algorithm.

FPP is coupled to Dave Smithe's full wave ICRF package SPRUCE. SPRUCE is an improvement over the earlier Colestock-Smithe code SHOOT, including general geometry plasma and vacuum vessel and improved numerics. SPRUCE solves a contracted second-order differential equation for the fast wave including the effects of fundamental and second harmonic ion damping, electron damping, and mode conversion. The power mode-converted to the Bernstein wave is estimated using the order-reduction scheme outlined in Dave Smithe's paper. It is assumed that the mode-converted power is absorbed immediately by the electrons before propagating elsewhere. SPRUCE uses a spectral representation in the poloidal direction and a finite difference method (which is numerically much more stable than the former shooting method) in the radial direction. The results can then be summed over kz to obtain a

full 3-dimensional solution. However, at the moment we only use a single kz (which is assumed to be the dominant kz in the spectrum).

SPRUCE assumes that all ion species are Maxwellian and expands all of the Bessel functions in the dielectric tensor through second order in (kperp*rho)**2. The minority ions are treated as a Maxwellian with the same parallel temperature as calculated by FPP because the Doppler-broadening of the parallel resonance is the dominant effect.

****

Computational time

****

A simple NBI run to equilibrium (300 msecs of tokamak time) in about 20-40 time steps on an 82 energy * 50 pitch angle * 10 radii can be done in 5-20 minutes on a Vax-8700, depending on the complexity of the beams. A run with 20 radii including radial diffusion and 10 iterations with the ICRF full wave code takes about 2-3 hours on a Vax-8700, and about 30-60 minutes if radial diffusion is ignored.

****

II. SETTING UP TO RUN FPPRF.

****

1. Create a subdirectory to contain your FPPRF work:

    $create/directory [YOURNAME.fpp]

2. Copy the following files to your new [YOURNAME.fpp] area:

transp$:[greg.fpp.test]testnb02.in - a sample NBI input file

transp$:[greg.fpp.test]testcs05.in - a sample ICRF input file

These sample namelist files can be used as a guide. You will need to modify the input variables to match your specific case.

transp$:[greg.fpp]fpbasic.dft contains default values for the namelist input variables, and a description of all of them.

3. Define some logical names and commands needed to run FPP (it may be useful to put this in your login.com file):

$ @transp$:[greg.fpp]fpprf_ini.com

4. Type "show process/quota" and make sure your paging file quota is at least 20000. If not, ask Judy Benson to raise your paging file quota.

****

III. Creating input files for FPPRF from SNAP'ed TFTR data:

****

$ run transp$:[greg.fpp.data]fpprep

This will prompt you for an FPPRF run name, a TFTR shot #, and a snap try #. It then reads the data in the snap archives, and produces the *.in namelist file needed as input for FPPRF. You may then edit this file to tailor the FPP run for your purposes.

****

IV. Executing FPPRF as a batch job:

****

First move to your fpp working directory:

$set default [YOURNAME.fpp]

Then use one of the following three commands, which are functionally equivalent:

1. $ @fpprun RUNNAM !starts execution of FPPRF at your terminal.

!This method is HIGHLY DISCOURAGED because it runs a compute
!intensive job at a high interactive priority. Running on the
!vaxes can get frustratingly slow, and people do check to see
!if anyone is hogging computer time with an interactive job.
!This method should only be used for interactive debugging.

2. $ submit/notify/noprint/log=[YOURNAME.fpp] fpprun/param=(RUNNAM)

!submits FPPRF as a batch job, with the log file put in your
![.fpp]area instead of your top level area or printed.
!you must add the command "$set default [YOURNAME.fpp]"
!to the beginning of FPPRUN.COM for this to work right.

3. $ batch fpprun RUNNAM

! This accomplishes the same result as the above submit
! command, but in a conciser format, using the BATCH
! command written by Charles Karney. Type "$setup batch"
! to initialize it, or "$help batch" for more info.

FPPRF needs the following input file:

RUNNAM.in !this is a namelist format input file.

FPPRF produces the following output files:

RUNNAM.plt !a plot file which can be seen by typing "xplot RUNNAM.plt"
RUNNAM.log !the batch job log file which contains some interesting info
RUNNAM_xxx.fcx !a 2-d ufile containing the CX signal ln f(Energy,time) at the
!detector tangency radius xxx cm's. This file can be plotted using
!Doug McCune's interactive UGRAF2 program.
RUNNAM.in2 An output copy of all of the input namelist parameters,
including those read from the default namelist.

V. How to prepare input files for simulating compression.

(To be added at some future date.)

****
VII. Modifying FPPRF
****

You are encouraged to add new features and capabilities to FPP.
However, this needs to be coordinated with me to ensure that a single
official version of FPP is maintained and to avoid the proliferation of
thousands of versions and the duplication of effort that produces. The
procedure is to:

1. Tell me what you want to change and what your plans are. I may be
able to give you some advice about how to achieve your goals most easily

and which routines need to be modified.

2. Copy the routines you are going to modify out of SOURCE: and into your own area. Modify these routines to your satisfaction.

3. Compile and link your own version of FPP with your newly modified routines (the procedures fpprf_lnk.com and fppdb_lnk.com in transp$:[greg.fpp] will do this), and check out your new routines to make sure they work right.

4. Let me know that your new routines are ready. I will put them back in the SOURCE: library where the OFFICIAL FPP routines are stored and will recompile and relink the official FPPRF.EXE.

Because FPP is part of TRANSP and is run on the CRAY at Livermore and the IBM at JET as well as on the VAX, you need to stick to standard fortran when you change the code. In particular, avoid DO-ENDDO blocks, keep character names to 8 characters or less, and don't use "_" or "$" in variable names. Any subroutines which access the TRANSP common block (TRCOM, or source:port.for) need to use local variable names which start with i, j, or z. The main FPP common block FPPCOM should not have any variables which start with i, j, or z, and the variable names must not be identical to any variable names in in the TRANSP common block, because both FPPCOM and TRCOM are included in some interface routines between FPP and TRANSP.

Because of intrinsic differences between the CRAYs and VAXES, we sometimes are forced to use machine-specific instructions. In these cases, the lines which end in "!@" are for the vax, while lines which start in "c@" are for the CRAY. The preprocesser program transp$:[cra]vaxtoc is used to convert programs to run on the cray.

The BUILD procedure in fphome .cray will automatically build a complete fortran source for FPP to be shipped to the CRAYs.

With the exception of some library routines from Harry Towner, all of the sources needed to compile and link FPPRF are in SOURCE: (which points to transp$:[cmsref.*]). SOURCE: contains the most recent version of every file in TRANSP (so it contains hundreds of files). SOURCE: is a list of special areas maintained by the CMS library system. To get a directory of all of the FPP files, take the following steps:

```
$cms set library transp$:[cms_v3]   !points to the TRANSP CMS library
$cms show element fpp_standalone
```

The transp$:[greg.fpp...] areas contain a number of different types of files, some of which are copies (sometimes out-of-date copies) of things which are in SOURCE:

transp$:[greg.fpp.test]

*.in	Sample namelist inputs for various FPP runs.
*.plt	Corresponding output plot files.

transp$:[greg.fpp]

FPP.DOC	This file.
FPPRUN.COM	Command procedure which runs fpprf.
*.inf	various information files of interest
*.ind (and others)	LISMAK files for finding local and global variables and renaming.
FPPRF.EXE	FPPRF program ready to run.

FPPDB.EXE FPPRF program ready to run with debug compiled subroutines.

FPPLIB.OLB Object library (compiled) of every FPPRF subroutine.

FPPLIBDB.OLB DEBUGgable object library of every FPPRF subroutine.

@FPPRF_MAK Will recompile the whole FPPLIB.OLB library.

@FPPDB_MAK Will recompile/debug the whole FPPLIBDB.OLB library.

@FPPRF_LNK Will relink a new FPPRF.EXE.

@FPPDB_LNK Will relink a new FPPDB.EXE.

@FPPRF_INI defines fpprf_inc: needed to recompile FPP routines.

@update subnam will update only subroutine subnam in FPPLIB.OLB.

@updatedb subnam will update only subroutine subnam in FPPLIBDB.OLB.

transp$:[greg.fpp.test] results of standard test cases of FPP

transp$:[greg.fpp.test.testold] older FPP test cases

transp$:[greg.fpp.now] copy of the just the FPP related files in SOURCE:

*.for Subroutines called by FPPRF.

*.cmn common blocks which are included by various subroutines.

Listing some of the more important files in SOURCE:

FPPIN.NML 'include' file specifying the namelist input block.

FPPIN.OLY Olympus format definitions of all of the variables in the namelist block FPPIN.NML.

FPPRF.FOR	The main FPPRF subroutine.
FPPTOP.FOR	Driver program for the standalone (no TRANSP) version of FPPRF.
FIMAIN.FOR	TRANSP/FPP interface driver (former routines NBMAIN FPMAIN and FPICHMN have been combined)
TRTOFP.FOR	Transfers TRANSP information to FPP inputs.
FPTOTR.FOR	Transfers FPP output to the TRANSP common block.
FPPORT.OLY	Olympus style definitions of FPP specific common block in TRANSP's PORT.FOR and therefore TRCOM.
FPPORT.CMN	A fortran version of the FPPORT.FOR which can be used in the standalone version (without TRANSP) of FPP.
FPPCOM.CMN	Logical name FPPCOM points to this file, which includes most of FPP's own (separate from TRANSP) common blocks.
*.cmn	There are a few other common blocks which have to be included separately from FPPCOM.CMN. In particular, GNEUT.FOR can't use FPPCOM.CMN because of a name conflict with the FRANTIC common block, and so must use its own copy of GNEUT1.CMN. There are also a few common blocks which are not included via *.cmn files but are already in the *.for files.
*.for	all of these contain a single subroutines except for
RATES.FOR	which contains a set of replacement cross-section routines

used to make FRANTIC work for helium plasmas as well as hydrogen plasmas, and

FPPONLY.FOR which contains some routines for the standalone version of FPP only (and so are not used in the TRANSP version.)

****

VIII. References

****

There are extensive comments in the source code. Also, the *.inf files contain comments about various modifications to the code.

References which describe how the code works, what numerical techniques are used, derivations and definitions of the collision and quasilinear operators can be found in:

Hammett86: Gregory Wayne Hammett, "Fast Ion Studies of Ion Cyclotron Heating in the PLT Tokamak", Ph.D. Dissertation (Princeton, 1986), University Microfilms International No. GAX86-12694.

Stix92: Stix documented my approach to bounce-averaging the quasilinear operator in Chapter 18 of the 1992 edition of his book: T.H. Stix, Waves in Plasmas, (American Institute of Physics, New York, 1992).

Smithe87: D.N. Smithe, P.L. Colestock, R.J. Kashuba, T. Kammash, "An Algorithm for the calculation of three-dimensional ICRF fields in tokamak geometry," Nuclear Fusion, Vol. 27 (1987) p. 1319.

Smithe89: D.N. Smithe, C.K. Phillips, G.W. Hammett, P.L. Colestock,

"SNARF analysis of ICRF heating on TFTR", in the proceedings of the Eighth Topical Conference on Radio-Frequency Power in Plasmas, (Irvine, CA, 1989), Roger McWilliams, editor, (AIP Conference Proceedings 190).

See also the papers by G.W. Hammett, C.K. Phillips, and P.L. Colestock in this conference for examples of how FPP and SPRUCE are used.

A recent example of the use of the code to investigate radial transport of fast ions can be found in W.W. Heidbrink et.al., "The Diffusion of Fast Ions Following Short Neutral Beam Pulses in Ohmic TFTR Discharges", submitted to Phys. Rev. Letters.

To get an understanding of the physics underlying ICRF heating and quasilinear theory which FPP is trying to model, one should read the classic work by Stix (much of which now appears in Chapter 18 of the 1992 edition of his book cited above):

Stix75: T.H. Stix, "Fast Wave Heating of a Two-Component Plasma," Nucl. Fusion V. 15 (1975) p. 737.

There are a few additional references which might not be found in my dissertation, but which might be useful:

Rob Goldston's Ph.D. thesis (Princeton, 1977) contains a lot of the foundation. Some of Goldston's thesis was published in R.J. Goldston, Nucl. Fusion 15, p. 651 (1975), but the actual 2D-velocity bounce-averaged finite-difference Fokker-Planck code wasn't written until after the thesis and this paper.

J.G. Cordey, NF 16 p.499 (1976) describes the basic bounce-averaging approach.

FPP was built on an earlier code by Rob Goldston which is described in J.D. Strachan, P.L. Colestock, S.L. Davis, et. al. Nucl. Fus. 21, p.67 (1981), and D.K. Bhadra, et. al. Nucl. Fus. 22, p. 763 (1982). Though somewhat cursory, these two papers are apparently the best published descriptions of Goldston's early version of the code. Much of the code used for neutral beam injection still has the same conceptual structure as Goldston's original code, though I have made extensive improvements virtually rewriting most of the code.

FPPRF uses a conservative differencing scheme which is described in McCoy, Mirin, and Killeen, CPC 24 p.37 (1981). That the RF operator should be written in conservative form was first made clear upon reading Kerbel and McCoy's "Bounce-Averaged ICRH for Toroidal Devices", 1983 Sherwood meeting.