2021/09/13

User Manual of the TASK/FP Code

Contents

1 Structure

1.1 Environment routines

- fpcomm.f90: Definition of variable module, allocation of arrays
  - **Module** fpcomm_parm: Definition of constants and input parameters including
    - bpsd_kinds
    - bpsd_constants
    - commpi
    - plcomm
    - obcomm_parm
  - **Module** fpcomm: Definition of common variables
  - **Subroutine** fp_allocate: Allocation of common arrays
  - **Subroutine** fp_deallocate: Deallocation of common arrays
  - **Subroutine** fp_allocate_ntg1: Allocation of time history ntg1 array
  - **Subroutine** fp_deallocate_ntg1: Deallocation of time history ntg1 array
  - **Subroutine** fp_adjust_ntg1: Adjust (save, deallocate, allocate larger, recover) time history ntg1 array
  - **Subroutine** fp_allocate_ntg2: Allocation of profile time history ntg2 array
  - **Subroutine** fp_deallocate_ntg2: Deallocation of profile time history ntg2 array
  - **Subroutine** fp_adjust_ntg2: Adjust (save, deallocate, allocate larger, recover) profile time history ntg2 array
- fpmain.f90: Main routine
  - **Program** fp
    - Initialization of MPI and graphics
    - Initialization of input parameters of module, pl, eq, and fp
    - Read input namelist file fpparm
    - Start menu input loop fpmenu
    - Termination of graphics and MPI
- fpinit.f90: **Module** fpinit: Initialization of input parameters
  - **Subroutine** fp_init: Set default values of input parameters
- fpmenu.f90: **Module** fpmenu: Process control
  - **Subroutine** fp_menu: menu command input and execute
    - P or with =: namelist read of input parameters
    - V: view all input parameters

        * R: start new calculation (set initial condition and go)
        * C: continue previous calculation (just go)
        * G: graphic output of calculated results
        * F: file output of calculated results
        * S: save present status of calculation into a file
        * L: load a file to continue previous calculation
        * W: print out interesting quantities
        * Y: debug output [temporal]
        * Z: debug output [transport coefficients]
        * Q: quit menu input and stop

- fpparm.f90: **Module** fpparm: Read and check input parameters
    - **Subroutine** fp_parm: analyze parameter input
        * read one line from standard input and analyze command or namelist
        * read namelist input from a file
        * read one phrase and analyze as a namelist
    - Subroutine fp_nlin: namelist input slave routine
    - Subroutine fp_plst: input parameter list slave routine
    - Subroutine fp_check: input parameter check
    - **Subroutine** fp_broadcast: Broadcast input parameters
    - **Subroutine** fp_view: Show input parameters

1.2 Execution routines

- fpprep.f90: **Module** fpprep: Preparation of run (mesh, initial profile, initialization)
    - **Subroutine** fp_mesh: create mesh quantities
        * set pmax
        * read EQ data (RKAP=1.0 should be removed)
        * set radial mesh
        * read WR data
        * read WM data
        * set approximate poloidal magnetci field (exact poloidal magnetic field should be used)
        * set momentum space mesh
        * set element volume in real space
        * set bounce-average parameters
    - **Subroutine** FPCINI: clear friction and diffusion coefficients
    - **Subroutine** fp_comm_setup: check partitions and setup shadow and commucicators
    - **Subroutine** fp_set_nsa_nsb: set table on nsa and nsb
    - Subroutine FNSP_INIT: initialize distribution functions FNSP and delta f
    - **Subroutine** FNSP_INIT_EDGE: initialize edge distribution functions FS1, FS2
    - **Subroutine** fp_set_normalize_param: set normalized density, temprature and other quantities

  - **Subroutine** `Coulomb_log`: calculate Coulomb logarithm
  - **Subroutine** `fp_continue`: setup for continuation
  - **Subroutine** `fp_set_intial_value_from_f`: record initial values
  - **Subroutine** `fp_prep`:
    * initialize time counter
    * allocate variables
    * setup matrix parameters for MPI
    * setup particle species table (fp_set_nsa_nsa)
    * create mesh (fp_mesh)
    * initialize diffusion coefficients (FPCINI)
    * set parameters (fp_set_normalize_param)
    * initialize distribution functions (FNSP_INIT, FNSP_INIT_EDGE)
    * set background distribution functions (update_fnsb)
    * read FIT3D rsults for NBI (READ_FIT3D, SV_WEIGHT_R)
    * set parallel electric field
    * setup for Legendre expansion and fusion reaction rate (NF_*)
    * record initial values (fp_set_initial_value_from_f)

- `fpbounce.f90`: Preparation of bounce parameters
- `fploop.f90`: Time loop for execution
- `fpexec.f90`: Execution of one time step

1.3 Calculation of coefficients of equations

- `fpcoef.f90`: Calculation of various coefficients
- `fpcalc.f90`: Calculation of collisional term (linear operator)
- `fpcalcn.f90`: Calculation of collisional term (nonlinear operator)
- `fpcalcnr.f90`: Calculation of collisional term (relativistic nonlinear operator)
- `fpcale.f90`: Calculation of static electric field term
- `fpcalr.f90`: Calculation of radial diffusion term
- `fpcalw.f90`: Calculation of quasi-linear term (given wave field)
- `fpcalwm.f90`: Calculation of quasi-linear term (using wm results)
- `fpcalwr.f90`: Calculation of quasi-linear term (using wr results)
- `fpcdbm.f90`: Calculation of CDBM radial diffusion coefficients
- `fpnfrr.f90`: Calculation of fusion reaction term (isotropic distribution)
- `fpnflg.f90`: Calculation of fusion reaction term (anisotropic distribution)
- `fpdisrupt.f90`: Calculation of disruption-related tems

1.4 Calculation of transport coefficients (by Ota)

- `fpcaldeff.f90`: Effective particle diffusion coefficients
- `fpcalchieff.f90`: Effectiv thermal diffusion coefficients
- `fpcaltp.f90`: Particle confinement time, tauP
- `fpcalte.f90`: Energy confinement time, tauE
- `fpchecknc.f90`: Radial diffusion coefficients (neoclassical diffusion)

1.5 File IO routines

- `fpfile.f90`: Save and load restart data
- `fpfout.f90`: File output of graphic data
- `fpoutdata.f90`: File output of intermediate data (by Ota)
- `fpread.f90`: Read FIT3D data from file
- `fpreadeg.f90`: Read Experimental data from file (by Nuga)
- `fpsave.f90`: File output of various data (by Nuga)
- `fpwmin.f90`: File input of full-wave analysis (wm) results
- `fpwrin.f90`: File input of ray/beam tracing analysis (wr) results
- `fpwrite.f90`: File output of trcoef data (by Ota)

1.6 Graphic routines

- `fpgout.f90`: Graphic output for gsaf
- `fpcont.f90`: Graphic subroutines

1.7 Library routines

- `fpmpi.f90`: MPI interface for fp
- `fpsub.f90`: Subroutine library (FPMXWL, FPNEWTON)

1.8 Orbit-averaging routines (by Ota)

- `fowcomm.f90`: Definition of fow variables and allocation
  - **Module** `fowcomm`: define quantities related to orbit-averaging
  - **Subroutine** `fow_allocate`: allocate adjustable arrays
  - **Subroutine** `fow_deallocate`: deallocate adjustable arrays
- `fowprep.f90`: **Module** `fowprep`: Preparation of fow, initialization of variables
  - **Subroutine** `search_pinch_orbit`: calculate pinch orbit
  - **Subroutine** `calculate_jacobian`: calculation of jacobians
- `foworbit.f90`: **Module** `foworbit`: Interface to ob

  - **Subroutine** `fow_set_obparm`
    * ob_init, ob_parm, ob_prep, ob_allocate, and spline psim, Fpsi, B, dradpsi
  - **Subroutine** `fow_orbit`
    * calculated orbits
  - **Subroutine** `fow_cal_local_COMs`
    * calculate thetaml, rhoml, tau_loss
  - Subroutine `construct_orbit`
    * call ob_calc to calculate ob structure
  - Subroutine `construct_orbit_zero`
    * set zero into ob structure
  - Subroutine `save_orbit`
    * save orbit data into file
  - Subroutine `load_orbit`
    * load orbit data from file
  - **Subroutine** `quantities_at_Bminimum`
    * calculate quantities at B_min
  - **Subroutine** `mean_ra_quantities`
    * calculate quantities at average radius

- `fowclassify.f90`: **Module** `fowcllasify`: Orbit classification and data output to file
  - **Subroutine** `output_orbit_cllasify`:
    * prep_orbit_classify
    * pinch_orbit
    * D_orbit(beta_D)
    * stagnation_orbit(beta_stag)
    * stagnation_type(xi_Xtype_boundary_ion)
    * output to file dat/*_obclass.txt
  - Subroutine `prep_orbit_classify`: calculate `theta_m` and `xi`
  - Subroutine `pinch_orbit`: calculate all pinch points
  - Subroutine `get_pinch_point`: calculate momentum of one pinch orbit
  - Subroutine `D_orbit`: calculate maximum momentum of trapped particle
  - Subroutine `stagnation_orbit`: calculate maximum momentum of not-forbidden orbit
  - Subroutine `stagnation_type`:

- `fowdistribution.f90`: Distribution conversion and integrated quantities
  - **Subroutine** `fI_Maxwellian`: calculate Maxwellian distribution
  - **Subroutine** `convert_fI_to_fu`: convert f(I) to f(local)
  - **Subroutine** `moment_0th_order_COM`: calculate 0th order momentum from f(I)
  - **Subroutine** `moment_2nd_order_COM`: calculate 2nd order momentum from f(I)
  - **Subroutine** `particle_flux`: calculate total particle flux
  - **Subroutine** `particle_flux_element`: calculate particle flux from each component

  - **Subroutine** `total_N`: calculate total density
  - **Subroutine** `effective_diffusion_cosfficient`: calculate particle diffusion coefficint from particle flux and density gradient

- `fowloop.f90`: fmodfowloop: Time loop for execution
  - **Subroutine** `fow_loop`:
    * fI_Maxwellian: calculate initial distribution function
    * fow_coef: calculate transport coefficients
    * fow_calculate_source: calculate source
    * loop: call fow_exec, update coef
    * calculate density and temperature
    * output data
  - Subroutine `update_bulk_temperature`: calculate temperature
  - Subroutine `output_data`: output data to file dat/*.txt

- `fowexec.f90`: **Module** `fowexec`: Execution of one time step
  - **Subroutine** `fow_exec`: Execution of one time step
    * `mtx_setup`: initialization of matrix solver
    * `fowweight`: setup weight array
    * `SET_FM_NMA`: setup index array
    * `fowsetm`: calculate matrix coefficients in a row
    * `IBC_pinch, IBC_X_stagnation, IBC_O_stagnation`: set internal boundary conditions
    * `mtx_set_matrix, mtx_set_vector, mtx_set_source`: set matrix coefficients, initial solution vector, and right-hand-side vector
    * `mtx_solve, mtx_get_vector`: solve matrix equation and obtain solution vector
    * `shadow_comm_np, shadow_comm_nr`: gather solution vector
  - Subroutine `SET_FM_NMA`: setup index array
  - Subroutine `fowweight`: setup weight array
  - Function `fowwegh`: elementary weight function
  - Subroutine `fowsetm`: calculate matrix coefficients in a row
  - Function `Dfow`: matrix coefficients
  - Function `w`: weighting
  - Function `check_external_boundary`: check within external boundary
  - Function `get_nma`: calculate matrix position
  - Subroutine `IBC_pinch`: set pinch boundary condition
  - Subroutine `IBC_X_stagnation`: set X stagnation boundary condition
  - Subroutine `IBC_O_stagnation` set O stagnation boundary condition
  - Function `f_grid`: evaluate weight on boundary grid point
  - Function `nma_boundary`: check `nth`, `np`, `nr` within range (update the evaluation)

- `fowcoef.f90`: **Module** `hfowccoef`: Calculation of various coefficients
  - **Subroutine** `fow_coef`:

        * allocate arrays and initialize
        * `convert_fI_to_fu`: set local distribution function
        * `fp_calc`: calculate local coefficients
        * `bounce_average`: bounce average local coefficients
    - Subroutine `bounce_average`: bounce average local coefficients
    - **Subroutine** `transformation_matrix`: calculation of transformation matrix
    - **Subroutine** `make_U_Dxy`: calculation diffusion coefficints in U frame
    - **Subroutine** `interpolate_D_unlessZero`: calculate interporation coefficients
- `fowsource.f90`: **Module** `fowsource`: Calculation of source terms
    - **Subroutine** `fow_calculate_source`:
        * `beam_source`: calculate beam source
        * set `sppb`
    - Subroutine `beam_source`: calculate and normalize beam source
    - Function `construct_beam`: evaluate `construct_beam`
- `fowlib.f90`: **Module** `fowlib`: Library for fow
    - **Subroutine** `solve_quadratic_equation`: quadratic equation solver
    - **Subroutine** `first_order_derivative`: evaluate first-order derivative
    - **Subroutine** `second_order_derivative`: evaluate second-order derivative
    - **Subroutine** `gauss_jordan`: matrix solver
    - **Subroutine** `fow_cal_spl`: 1D spline for fow
    - **Subroutine** `fow_cal_spl2D`: 2D spline for fow