How to use d3d-dt programs

A.Wingen 4/27/09

Contents:

1. Parameter files

All programs need the diiidsup.in file as well as the g-files. All other '.in' files have been replaced by the parameter file described below.

ilt = 360 and dpinit = 1 **cannot** be changed outside the code itself. All trip3d output files are **no** longer used and will **not** be created.

The path to the g-files is set in the header file *d3d-drift.hxx*. To change the path, the code has to be recompiled (sorry!)

A new parameter file is necessary:

```
# Parameterfile for DIII-D Programs
# Shot: 129194      Time: 3000ms
free_Parameter=     500
itt=   100
rmin=  0.42
rmax=  0.57
thmin= 0
thmax= 0
N=     31
phistart(deg)=      0
MapDirection= 1
R0=    1.757
Z0=    0.0003
target(0=verical,1=45°,2=horizontal,3=shelf)=   1
createPoints(0=set,1=random,2=target)=   0
useFcoil(0=no,1=yes)= 1
useCcoil(0=no,1=yes)= 1
useIcoil(0=no,1=yes)= 1
ParticleDirection(1=co-pass,-1=count-pass,0=field-lines)=    0
ParticleCharge(-1=electrons,>=1=ions)=   1
Ekin[keV]=    100
lambda=       0.1
useFilament(0=no)=  1
pi=    3.141592653589793
2*pi=  6.283185307179586
```

Their filenames always start with an underscore '_' and end with '.dat'. Example: '_plot.dat' The first line is just a comment. In the second line, the shot and the time have to be specified, although it is marked as acomment line. All comment lines have to start with '#'. An arbitrary number of comment lines can be added after the second line of the file, but **no** comments are allowed in between or after the data.
The name of a parameter can have arbitrary length as long as no spaces are used. The name has no influence on the way the data is read. The data is read by its position in the data list. The name is followed by a tab and the parameter value.

- free_Parameter
  it is not comonly defined and is used in different ways by the different programs. See the respective program for details.
- itt
  Number of toroidal iterations to perform by the program
- rmin, rmax, thmin, thmax
  Range of initial conditions in real toroidal coordinates *r* and *theta*. The latter has to be between *0* and *2*pi*.
  For CreatePoints=2 case: *r = t* and *theta = phi* is used
- N
  Number of initial conditions
- phistart
  toroidal angle in degrees *(0°..360°)* for the Poincaré section
- MapDirection: +1 or -1 (0 for d3dlaminar too)
  +1/-1 → iteration in positive/negative phi direction
- R0, Z0
  unimportant Input! Position of the magnetic axis in real machine coordinates *(R,Z)* this data is now directly acquired from the g-file
- target
  specifies the target plate to choose the initial conditions on. Only necessary if CreatePoints = 2 is chosen. For details see d3dfoot.
- CreatePoins
  Choose method to get initial conditions.
  0 = Points are taken from a regular grid specified by rmin, rmax, thmin, thmax and N (sqrt(N) points in r and theta direction respectively).
  1 = random generator chooses N initial conditions within the specified range
  2 = initial conditions are chosen on target plate from a regular grid
- useFcoil, useCcoil, useIcoil
  specifies, whether to use these coils (=1) or not (=0)
- ParticleDirection, ParticleCharge
  The code now includes particle-drift effects. For field lines only, use ParticleDirection=0. ParticleCharge has no effects on field lines
- Ekin, lambda
  kinetic energie of particles and radial part of energie (in percent, only estimate)
  no effect on field lines
- useFilament
  includes current filaments. 0 = none, >0 = all
  see filament section for more detail
- pi, 2*pi are not read by any program or necessary at all, they are just helpfull to set e.g. thmax by copy and paste

2. Outputfile structure

All output files have a header which includes all relevant parameters for the respective run. Example:

```
# d3dtracer
#------------------------------------------------
### Parameterfile: _3000plot.dat
#------------------------------------------------
### Switches:
# Target (0=vertical, 1=45°, 2=horizontal, 3=shelf): 1
# Create Points (0=grid, 1=random, 2=target): 0
#------------------------------------------------
### Global Parameters:
# Steps till Output (ilt): 360
# Step size (dpinit): 1
# Boundary Rmin: 1.016
# Boundary Rmax: 2.4
# Boundary Zmin: -1.367
# Boundary Zmax: 1.36
# Magnetic Axis: R0: 1.757
# Magnetic Axis: Z0: -0.004
#------------------------------------------------
### additional Parameters:
# Max. Iterations: 300
# Points: 100
# rmin: 0.46
# rmax: 0.54
# thmin: 0
# thmax: 0
# phistart: 225
# MapDirection: 1
#------------------------------------------------
### Data:
# theta[rad]     r[m]     phi[deg]     psi
#
2.783539745069958 0.5935466234728225      585   0.7916423537837542
4.551706379276331 0.8405327650042975      945   0.790854672604256
2.144874365875646 0.7848006675379068      1305  0.7938579085980841
4.139298920881824 0.813733863778083 1665  0.7958004296252812
```

Again all comment lines start with ‘#’, the default comment character of gnuplot. The data in the columns has up to 16 didgets after the dot and the columns are separated by tabs. The kind of data is named in the last header line. In this example the first column is the poloidal angle in rad, the second is the minor radius with respect to the magnetic axis, the third is the toroidal angle in degrees and the last is the normalized flux (psiwp in the trip3d code). The structure is optimized for usage with gnuplot.

3. Programs

3.1 dtplot

```
// Program calculates Poincaré Plot for D3D
// Fortran Subroutines for Perturbation are used
//
// Input: 1: Parameterfile   2: praefix(optional)
// Output:  Poincaré field line data file
//          log-file
```

The program needs the name of the parameter file as an input and gives the opportunity to use some arbitrary string (called praefix here) to be added to the output filename. It gives back an ASCII file with the data and a log file which includes various informations like startup parameters, progress reports during runtime and error messages.

Example:
> d3dtracer _plot.dat tryit
Output:
trip3d_plot_tryit.dat and log_d3dtracer_tryit.dat (log-file)

Exceptions: free_Parameter is **not** used in this program

3.2 dtfix

```
// Program searches for periodic fixed points
// Fortran Subroutines for Perturbation are used
//
// Input: 1: Parameterfile   2: period of fixed point
//          3: praefix (optional)
// Output:  fixed points
//          log-file
```

In principal the same as d3dtracer. But the second input is now the period of the fixed points, that are searched for.

Example:
> d3dfix _3000fix.dat 1 all
Output:
d3dfix_3000fix_1_all.dat and log_d3dfix_1_all.dat (log-file)

Exceptions: free_Parameter, itt, target and CreatePoints are **not** used in this program

3.3 dtman

```
// Program calculates unstable manifold of a hyp. fixed point for D3D
// For stable manifold use reverse Integration (see MapDirection)
// For left or right hand-sided set sign of 'verschieb' in Parameterfile to
// - or + respectively
// Fortran Subroutines for Perturbation are used
//
// Input: 1: Parameterfile   2: File with fixed points
//          3:praefix(optional)
// fixed points have to be in toroidal coordinates, not cylindrical!!!
// Output:  one file for each of the fixed points specified in the input-
//          file, giving the respective manifold
//          log-file
```

d3dman reads the output file of d3dfix and calculates a manifold for each of the fixed points specified in the d3dfix output file. Usually these fieles contain the same points several times (and elliptical points as well). So it is necessary to copy and paste the required points to a separate file, or delete the unneccessary points from the original file. This cannot be done automatically, because the way fixed points are found by d3dfix is unpredictable.

Example:
> d3dman _3000fix.dat d3dfix_3000fix_1_selected.dat

Output:
d3dman_3000fix_unstr1_0.dat and log_d3dman_unstr.dat (log-file)

In the example the file 'd3dfix_3000fix_1_selected.dat' is read. It contains only one point (the intersection point of the separatrix). Note that no praefix was used.
unst = unstable manifold → st = stable
r = right-hand sided → l = left
1 = period of fixed point
0 = first outputfile. All output files are numbered according to the number of fixed points in the input file

Exeptions: itt, rmin, rmax, thmin,thmax, N, target and CreatePoints are **not** used in this program.
free_Parameter(verschieb) is a suggestion of a necessary shift out of the position of the fixed point. **Typical value: 1e-5.** Its sign determines if the right-hand sided, positive, or left-hand sided, negative, manifold is calculated.
MapDirection determines if the stable or unstable manifold is calculated.

3.4 dtfoot

```
// Program calculates connection length and penetration depth for D3D
// Fortran Subroutines for Perturbation are used
//
// Input: 1: Parameterfile    2: praefix(optional)
// Output:  2d footprint data for colored contour plot
//          log-file
```

Since the data is for a 3d plot, it has to be prepared for plotting afterwards (e.g. with matlab) The data is written in columns, although it would be matrices. The first column is varied first.

Example:
> d3dfoot_2250foot.dat scan
Output:
foot_2250foot_scan.dat and log_d3dfoot_scan.dat (log-file)

Exceptions: phistart, MapDirection and CreatePoints are **not** used in this program
free_Parameter(Np) = number of *phi* grid points
rmin, rmax = tmin, tmax: grid boundarys in length parameter *t*
thmin, thmax = phimin, phimax: grid boundarys in toroidal angle *phi*
N = Nt: number of *t* grid points

Targets: (positions in m)
vertical (0): P1=(1.0161,-1.22884) ←→ *t* = 0 P2=(1.0161,-1.034873) ←→ *t* = -1
length=19.3967cm
45° (1): P1=(1.0161,-1.22884) ←→ *t* = 0 P2=(1.15285,-1.3664) ←→ *t* = 1
length=19.3967cm
horizontal (2): P1=(1.15285,-1.3664) ←→ *t* = 0 P2=(1.372,-1.3664) ←→ *t* = 1
length= 21.915cm
shelf (3): P1=(1.372,-1.25) ←→ *t* = 0 P2=(1.59115,-1.25) ←→ *t* = 1
length= 21.915cm
If target 1 (45°) is chosen, negative t values are allowed, which automatically are related to positions on target 0 (vertical wall). Note that target 0 and 1 have the same length which guarantees correct scaling of *t*. Note further that target 2 and 3 scale differently to 0 and 1.

3.5 dtlaminar

```
// Program calculates connection length and penetration depth for D3D
// inside the plasma volume
// Fortran Subroutines for Perturbation are used
// A.Wingen                                2.04.08

// Input: 1: Parameterfile    2: praefix(optional)
// Output:  2d connection length data for colored contour plot
//          log-file
```

similar to d3dfoot, but program works in the machine coordinate system *(R,Z)*, otherwise the colored contour plot would not be rectangular.

Exceptions: phistart, MapDirection and CreatePoints are **not** used in this program
free_Parameter(NZ) = number of *Z* grid points
rmin, rmax = Rmin, Rmax: grid boundarys in *R*
thmin, thmax = Zmin, Zmax: grid boundarys in *Z*
N = NR: number of *R* grid points

3.6 dtstructure

```
// Program calculates path of field lines in 10 deg steps. Used for 3-D
// Visualization for D3D
// Fortran Subroutines are used for perturbations

// Input: 1: Parameterfile    2: Initial value file (optional; if not, enter
//        x instead) 3: praefix (optional)
// Output:  paths of individual field lines in 10 deg steps between the
//          target plates
//          log-file
```

works similar to dtplot, but creates output every 10 degrees. But the output is given as the following data set, first the carthesian , then the cylindrical coordinates
X [m] Y [m] Z [m] R [m] phi [rad]
Output is stored in asscending order with respect to the toroidal angle phi.

Exceptions: see dtplot

3.7 additional programs

The following programs help to visualize the data.

target

```
// Program calculates targets, wall and boundary box, to plot.
// Necessary input: Shot# and Time

// Input:   1. Parameterfile (optional)
// Output:  target0.dat       -> vertical shelf above 45° target
//               target1.dat       -> 45° target
//               target2.dat       -> horizontal target
//               target3.dat       -> horizontal shelf above pump to
//                                    horizontal target
//               wall.dat          -> entire wall
//               boundary.dat      -> boundary box
// All outputfiles include theta, r, R, Z coordinates
```

Code requires either the parameter file or asks for the shot# and time.

RZ

```
// Program transforms from (theta,r) to (R,Z) coordinates
// First two columns are replaced accordingly, all other columns (up to
// total number of 5) are copied unchanged

// Input:   1. File to transform    2. total number of
//          columns (optional, Warning: automated column count can be
//          wrong)
// Output:  to (R,Z) coordinates transformed file; it gets the additional
//          praefix _RZ
```

merge_foot_files

```
// Program merges footprint files that have been manually divided for
// parallel computation
// Files have to have a continuing number at the end of the filename
// (before the .dat)
// Input filename has to be the first file to be read. All following files
// are read automatically
// Number of the first file has to be a single digit (not necessarily 1)

// Input:   1. first file to be merged    2. total number of columns
//                                        (optional, Warning: automated
//                                              column count can be wrong)
// Output:  File which contains all data from all files written among one
//          another
```

3.8 include current filaments

To include current filaments the following procedure has to be done as a preperation. The path of the filament in the machine has to be calculated using dtstructure. The following file has to be created using the data from dtstructure:

filament1.in

```
# struct_large.dat
#-----------------------------------------------
### Current[A]: 168
#-----------------------------------------------
### Data:
# X[m]     Y[m]     Z[m]     R[m]     phi[rad]
#
```

(data from dtstructure file)

The first line just names the original dtstructure output file. The third line is important! Here the current is specified.

To include more than one filament, similar filament files: *filament2.in*, *filament3.in*, etc. have to be created.

In the next step call fi_prepare:

```
// Program precalculates total magnetic field on EFIT grid and 360 toroidal
// slices of current filaments
// for D3D-Drift with current filament perturbations

// Input: 1: Parameterfile    2: praefix(optional)
// Output:  filament field file
```

In the parameter file the number of filaments, you want to include, has to be specified.
Example: useFilament = 2 then *filament1.in* and *filament2.in* are read by fi_prepare

fi-prepare creates an output file called *filament_all.in*. This file is then used by all other programs, as long as useFilament > 0 is set in parameter file. *filament_all.in* contains the magnetic field on the EFIT grid (129x129) and 360 toroidal slices (dpinit=1 is assumed!). In the code the field in between in interpolated by bicubic interpolation.

4. How to compile the source code

Three scripts are available to compile the certain source codes.

1. cxx
   this script is used for all only c++ codes: RZ.cxx, target.cxx, man_strike.cxx
   Example:
   > cxx RZ this compiles RZ.cxx and creates executable RZ
2. cxx_ifort
   this script is used for all c++/fortran combined codes (all files which start with d3d!)
   Example:
   > cxx_ifort d3dtracer this compiles d3dtracer.cxx and creates executable d3dtracer
3. cxx_dtall
   this script compiles all six d3d files, it calls cxx_ifort successively for all files

recomended folder structure:
~/src /bin
/include
/lib
~/lasys

src contains the source code and the compile scripts
bin contains the executables, compiling with the scripts automatically replaces executable(s) here
include contains necessary header files
lib contains necessary library files
lasys contains blitz library folders

blitz library is required for all the codes described here. See documantation of blitz library for details and install instructions

trip3d library is required: all trip3d fortran subroutines are stored in the library *libtrip3d.a*. This library has to be renewed for any changes of trip3d to be effective in the codes described here.
To do this: compile fortran code with e.g. ifort like you do for creating trip3d. Execute the script *create_libtrip3d* in the trip3d source folder. This creates the library from the object files (.o) and stores it in ~/src/lib. The previous library is replaced.

The header files contain all subroutines which are used in more than one program. d3d.hxx is the specific header for all d3d programs which use fortran subroutines.

5. Subroutines used from trip3d

Only five subroutines from the fortran trip3d library are called: `d3pfgeom, d3igeom, d3cgeom, d3pferrb` and `polygonb`. I have **not** checked, if these two call other subroutines! The common blocks used for the subroutines are created and initialized in the main header file d3d-drift.hxx.