README: PTDFN_control.dat file

Nataliia Makedonska, LANL, EES-16, nataliia@lanl.gov

PTDFN_control.dat file made up for setting of all necessary parameters to run particle tracking in DFNWorks. Below is a short explanation of each parameter setting on one PTDFN_control.dat example.

```
/*************************************************************************/
/*  CONTROL FILE FOR PARTICLE TRACKING IN DISCRETE FRACTURE NETWORK */
/*************************************************************************/
```

1. Define paths of files that were generated by DFNGEN and contain all the information about computational grid.

```
/*********************  INPUT FILES: grid ****************************/
/**** input files with grid of DFN, mainly it's output of DFNGen ******/
param: params.txt //generated by Mathematica code, contains the normal vectors for each fracture

inp: full_mesh.inp  // AVS file of full mesh of DFN: positions of vertices and connectivity list

stor: tri_fracture.stor // produced by LaGriT: the matrix of all connected nearest neighbors faces of Voronoi polygons and their geometrical coefficients

boundary: allboundaries.zone //list of nodes that located on domain boundaries
/* boundary conditions: reading the nodes that belong to in-flow and
out-flow boundaries. Should be consistent with those applied to obtain
steady state pressure solution (PFLOTRAN)   */
/*1 - top;  2 - bottom;  3 - left_w;  4 - front_s;  5 - right_e;  6 - back_n */
in-flow-boundary: 3
out-flow-boundary: 5
```

The allboundaries.dat file can be modified according to task. For example, inflow boundaries are left and top, and out flow is a bottom side. In this case, left and top list of boundary nodes should be combined together in allboundaries.dat and only one number given (for example 1). DFNTrans code can't read more than one defined number for in-flow or out-flow boundaries.

2. Input of Flow Solver results

```
/*************** INPUT FILES: PFLOTRAN flow solution ******************/
PFLOTRAN: yes //yes – the PFLOTRAN flow solver is used
```

PFLOTRAN_vel: darcyvel.dat // the Darcy flux for each Control Volume face of the full mesh of DFN
PFLOTRAN_cell: cellinfo.dat
PFLOTRAN_uge: lagrit_pflotran_dat/lagrit_pflotran_corrected.uge //connectivity list and area of each Control Volume face of the full mesh of DFN

The current version of DFNWorks uses PFLOTRAN
/**************** INPUT FILES: FEHM flow solution **********************/
/*currently we are using PFLOTRAN , but the code would work with FEHM, too */
FEHM: no
FEHM_fin: tri_frac.fin

3. Settings of OUTPUT files with particle tracking results.

/********************** OUTPUT FILES ******************************/
/* initial grid info structure output, useful for debugging */
out_grid: yes // yes means that the output ASCII files, named "nodes" and "fractures", will be generated. If you don't want this output - type "no" instead of "yes".

/* flow field: 3D Darcy velocities: output ASCII file "Velocity3D" has an each nodes position and its Darcy velocity, reconstructed from fluxes */
out_3dflow: yes

/* out initial positions of particles into separate ASCII file "initpos" */
out_init: yes

/*************** output options for particles trajectories ***************/
/* output frequency is set according to trajectories curvature. We check the curvature of particles trajectory each segment, from intersection to intersection. If it's like a straight line, then the output is less frequent (in case of "out_curv:yes", if "no", the output file will contain every time step) */
out_curv: no

/* output into avs file (GMV visualization, Paraview visualization) */
out_avs: no

/* output into trajectories ascii files (veloc+posit+cell+fract+time) */
out_traj: no

/************* output directories ***********************************/
out_dir: ptresults1 /* path and name of directory where all the particle
tracking results will be written, including those defined above*/

```
out_path: trajectories /*name of directory where all particle
                trajectories will be saved, in out_dir path */

/* name of resultant file (in out_dir path), which contains total travel time and
                                    final positions of particles */
out_time: partime
```

4. Options for Particles Initial Positions in DFN

```
/*************** PARTICLES INITIAL POSITIONS ****************************/
/****init_nf: if yes - the same number of particles (init_partn) will be placed
    on every boundary fracture edge on in-flow boundary,
    equidistant from each other ****/
init_nf: yes
init_partn: 5

/****init_eqd: if yes - particles will be placed on the same distance from
    each other on all over in-flow boundary edges ***********************/
// The difference of the current option and init_nf: here the total length of fracture edges on in-flow boundaries will be calculated. Then, according to init_npart given number of particles, the particles will be distributed equidistant over all fracture edges on in-flow boundaries.  In init_nf option, the init_partn number of particles will be equidist in each edge of fracture on in-inflow boundaries. In this case distance between two neighboring particles in one fracture will not be the same as distance between two particles in other fracture.
init_eqd: no
//maximum number of particles that user expects on one boundary edge
init_npart: 1

/*** all particles start from the same region at in-flow boundary, in a range
   {in_xmin, in_xmax,in_ymin, in_ymax, in_zmin, in_zmax} **************/
//  In this option, the region on in-flow boundary should be defined according to x, y, and z coordination of the domain. Then particles will be placed equidistant in those part of fracture edges that cross the defined region. If there are no fracture edge found there, the program will be terminated, and user should redefine the region.
init_oneregion: no
in_partn: 10
in_xmin: -20.0
in_xmax: 20.0
in_ymin: -20.0
in_ymax: 20.0
in_zmin: 499.0
```

in_zmax: 501.0

/**** all particles are placed randomly over all fracture surface
(not only on boundary edges!) ***********************************/
// In this option the particles will be placed on the center of randomly chosen Control Volume cell over all cells in DFN mesh (not only on in-flow boundary).
init_random: no
// total number of particles
in_randpart: 110000

/**** all particles are seed randomly over matrix,
they will start travel in DFN from the node/cell that is closest to
their initial position in rock matrix ***************************/
// In this case, the python code ParticleTracking/RandomPositGener.py should run first. The required number of particles will be placed all over domain (not only on fracture surface). Then, using LaGriT, the closest fracture cell ID will be defined for each particle. The initial time, the time that is required for particle to travel from it's initial position to closest fracture is estimated using the probability function:

$$t = \frac{z^2}{4\sqrt{\varphi D}}\left(\frac{1}{erfc^{-1}(R)}\right)^2$$

where R is a randomly chosen number between 0 and 1, D is a diffusion coefficient (defined by user below), and ϕ is a porosity of rock matrix (defined by user below).
init_matrix: no
// to obtain these files, run python script RandomPositGener.py
inm_coord: ParticleInitCoordR.dat
inm_nodeID: ClosestNodeR.inp
inm_porosity: 0.02
inm_diffcoeff: 1.0e-12

5. Flow and Fracture Parameters

/***************** FLOW AND FRACTURE PARAMETERS **********************/
porosity: 0.25 // porosity
density: 997.73 //fluid density
satur: 1.0
thickness: 1.0 //DFN aperture (used in case of no aperture file provided)

aperture: yes //DFN aperture
aperture_type: frac //aperture is giving per cell (type "cell")
// or per fracture (type "frac")
// In the current version of DFNWorks the only an aperture per fracture option is given.
aperture_file: aperture.dat // The ASCII file aperture.dat is produced by Mathematica in DFNGen

```
/******************  TIME ****************************************/
timesteps: 2000000
//units of time (years, days, hours, minutes)
time_units:  seconds

/**** flux weighted particles*/
// in all the options of initial positions of particles, particles can be weighted either by input flux  )in case of placing particles in in-flow boundary) of by current cell aperture (in case of randomly defined initial positions).
flux_weight: yes

/* random generator seed */
seed: 337799
```

6. Control Plane Output

```
/*********************  Control Plane/Cylinder Output ********************/
// Here is another option for output. The control Planes can be defined on any position along the flow direction. For example, if fluid flow goes from top to bottom along Z direction (flowdir=2), the imaginary control planes will be parallel  to x-y plane and placed each 1 m (delta_Control: 1). Each particle will have it's data output (location, current velocity) every time it crosses control plane.

/*** virtual Control planes will be build in the direction of flow.
Once particle crosses the control plane, it's position, velocity, time
will output to an ascii file. ****/
ControlPlane: no

/* the path and directory name with all particles output files */
control_out: outcontroldir

/* Delta Control Plane - the distance between control planes */
delta_Control: 1

/* ControlPlane: direction of flow: x-0; y-1; z-2 */
flowdir: 1

/*************************************************************************/
END
```