DFN Statistics

January 11, 2016

```
WARNING: you've entered a value of {0.25, 0.25, 0.25, 0.25} for the variable fa
randomly generated fractures you specified in nShape. The program has
automatically changed the value of famProb for you to famProb={};
```

This file contains data regarding the results of the DFN generation process.

Contents

1 General Info

```
Desired number of fractures (4) accepted.
Final fractures number: 4
Final fractures total surface area (Total fracture Area * 2) = 7.3
Final fractures total volume (Fractures Area *Fractures Aperture) = 0.0365
Final fracture density (No of fractures per unit volume), P30 = 3.4188
Final fracture intensity (Area of fractures per unit volume), P32 = 3.11966
Final fracture porosity (Volume of fractures per unit volume), P33 = 0.0311966
Number of families for each shape:
*0 families of ellipses;
*0 families of rectangles;
*0 user defined ellipses;
*4 user defined rectangles;
Domain (m) = {-0.9<=x<=0.9,  -0.5<=y<=0.5,  -0.325<=z<=0.325}
Domain Volume (m^3)= 1.17
h (m)= 0.1
Attempted fractures (number of polygons created during the program
while attempting to reach the desired number of fractures): 4
Final number of fractures: 4.
Accepted number of fractures (number of fractures accepted, but
that may have been eliminated because they were isolated or weren't
part of a network that connected two of the boundary faces: 4.
Number of fractures rejected because they were isolate or not part of
a network connecting two boundary faces: 0.
Number of fracture networks (in the final mesh): 1.
Sizes of fracture networks/groups: {4}.
```

```
Accepted family distribution:  {0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1}
Attempted family distribution: {0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1}
Adjacency matrix elements and size: SparseArray[<6>, {4, 4}]
```

2 Mesh Info

```
The final mesh of DFN consists of:
1063 triangular elements;
554  nodes / control volume cells;
3786 geometrical coefficients / control volume faces.
```

3 Time Profile

```
0.028957   Create fractures
0.000598   Find intersections
0.020099   Check single intersection
0.002447   Check triple intersection
0.000016   Construct disjoint sets
0.006015   Read/write sparse matrix
0.001404   Find fractures in the largest set
0.000388   Construct A (adjacency matrix used in\

>    generatingPoints`CreateIntersectionPoints)
0.023948   Discretize intersections
0.010146   Write output files
Max memory used (bytes) before output: 25,003,392
```

4 Rejection Reasons

```
Accepted and not deformed:          0
Accepted and truncated:             4
Accepted and deformed:              0
Outside rejections:                 0
Short intersections rejections:     0
Close to edge rejections:           0
Close to vertex rejections:         0
Int. point close to edge rejections: 0
Triple intersection rejections:     0
Not in the largest set rejections: 0
```

5 Mathematica Input File

```
(* ::Package:: *)

(* Mathematica input *)
(* For more explanation of each parameter, see the User Manual *)
Begin["Global`"];

(*=========================================================================*)
(*=========================================================================*)
(*Mandatory Mathematica/Generating Fracture Network Parameters: *)

nPoly = 5; (* Total number of fractures you would like to have
                        in the domain you defined. The program will complete
                        once you have nPoly number of fractures,
                        maxPoly number of polygon/fracture rejections,
                    rejPoly number of rejections in a row, or reach a
                    specified fracture cluster size if using
                    stoppingParameter = -largestSize  *)

domainSize = {1.8,1,0.65}; (* Mandatory Parameter.
                domainSize = {xlength, ylength, zlength}
                Creates a domain with dimension x*y*z centered at the origin.*)

h = 0.1; (* Minimum fracture length scale(meters)
          Any fracture intersection length of less than h will be rejected. *)



(*=========================================================================*)
(*Optional Mathematica/Generating Fracture Network Parameters:
 If these parameters are commented out, defaults will be used *)

multipleClusters = 0; (*Options: 0 or 1
                                        0: Keep only largest cluster
                                        1: Keep all clusters *)

visualizationMode = 0; (*Options: 0 or 1 *)
(* is used during meshing: 0: creates a fine mesh, according to h parameter;
   1: produce only first round of triangulations. In this case no
   modeling of flow and transport is possible. *)

productionMode = 1;
(* 0:"debugging mode" - keeps data of all attempted fractures
during fracture generations as well as all files during mesh producing;
   1:"production mode" - discards data of attempted but not
   accepted fractures and cleans up the file produced by lagrit.
   Default: 1  *)
```

```
SeedRandom[123457]; (* Seed for random generators.
                      Will be different each run if commented out *)

domainIncrease4Random=.0; (*optional user specified domain size
                               temporary size increase for inserting fracture*)
boundaryFaces={3,5};
(*         1 = top = positive z direction (0,0,1)
           2 = bottom = bottom = negative z direction (0,0,-1)
           3 = left / west = negative x direction (-1,0,0)
           4 = front / south = positive y direction (0,1,0)
           5 = right / east = positive x direction (1,0,0)
           6 = back / north = negative y direction (0,-1,0)

           Program will keep only clusters which connect to every specified face
           Default: boundaryFaces will keep any and all two connected faces
*)

maxPoly=1000*nPoly; (* max number of polygons/fractures the program will attempt
                     before program stops (accepted+rejected). Must be greater or
                     equal to nPoly
                     Default: maxPoly = 100*nPoly *)

rejPoly= 100*nPoly; (*Number of rejections in a row per polygon/fracture
                    before program stops
                    Default: rejPoly = 100 *)


(*=========================================================================*)
(* Mandatory Shape and Probability Parameters *)

nShape = {0,0,0,4}; (* {ellipses, rectangles, user-ellipses, user-recantles}
         Enter number of families for each defined shape
         you would like in the domain. The order is:
         {ellipses, rectangles, user-ellipses, user-rectangles}
         NOTE: Only one user-ellipse/user-rectangle are put
         in the domain per user-rectangle or user-ellipse family *)

famProb={0.25, 0.25, 0.25,  0.25 };
(* Probability of occurrence of each family of random distrubution rectangles
   and ellipses (first two dimensions of nShape).
   User-ellipses and user-rectangles insertion will be attempted with 100%
   likelihood, but with possability they may be rejected.
   The famProb elements should add up to 1.0 (for %100).
   The probabilities are listed in order of families.
   For example: If nShape={2,1,0,0} and famProb={.4,.5,.1}, then the first
   two famProb numbers are the probabilities of incurring a polygon from
   the two ellipses' families, and the third number is the probability of
   incurring a polygon of the rectangular family.*)
```

```
(*===========================================================================*)
(* Elliptical Fracture Options                                            *)
(*NOTE: Number of elements must match number of ellipse families  *)
(* first number in nShape input parameter                                 *)
(*===========================================================================*)
(*edist is a mandatory parameter if using statistically generated ellipses *)
edistr={1, 3};   (* Ellipse statistical distribution options:
                       1 - lognormal distribution
                       2 - truncated power law distribution
                       3 - exponential distribution
                       4 - constant *)
(*===========================================================================*)
(* Parameters used by both lognormal and truncated power law options: *)
(* Mandatory Parameters if using statistically generated ellipses  *)

easpect = {1, 1};  (* Aspect ratio. Used for lognormal and truncated
                   power law distribution. *)

enumPoints = {15, 15}; (* Number of vertices used in creating each elliptical
                     fracture family. Number of elements must match number
                     of ellipse families (first number in nShape) *)

etheta = {Pi/2, 0 };     (* Ellipse fracture orientation.
                     The angle the normal vector makes with the z-axis *)

ephi = {0, 0};  (* Ellipse fracture orientation.
                The angle the projection of the normal onto the x-y plane
                makes with the x-axis *)

ekappa = {20, 20};  (* Parameter for the fisher distribnShaprutions. The bigger,
                the more similar (less diverging) are the elliptical familiy's
                  normal vectors  *)

(*===========================================================================*)
(* Options Specific For Ellipse Lognormal Distribution (edistr=1): *)
(* Mandatory Parameters if using ellispes with lognormal distribution *)

(*NOTE: Number of elements must match number of
  ellipse families (first number in nShape) *)

emean = {Log[2], Log[2]};  (* Mean value For Lognormal Distribution.
*)


esd = {.5, 0.5};  (* Standard deviation for lognormal distributions of ellipses

(*===========================================================================*)
(* Options Specific For Ellipse Exponential Distribution (edistr=3): *)
(* Mandatory Parameters if using ellispes with exponential distribution *)
```

```
ellmean = {3.0, 2.0};  (* Mean value for Exponential  Distribution.
*)

(*=======================================================================*)
(* Options Specific For Constant Size of ellipses (edistr=4): *)

econst = {2, 2};  (* Constant radius for all the ellipses in the DFN
*)

(*=======================================================================*)
(* Options Specific For Ellipse Truncated Power-Law Distribution (edistr=2)*)
(* Mandatory Parameters if using ellipses with truncated power-law dist. *)

(* NOTE: Number of elements must match number
   of ellipse families (first number in nShape)*)

emin = {1, 1}; (* Minimum radius for each ellipse family.
                        For power law distributions. *)

emax = {3, 3};   (* Maximum radius for each ellipse family.
                        For power law distributions. *)

ealpha = {1.5,1.5}; (* Alpha. Used in truncated power-law
                         distribution calculation*)

epowerLawReject={50, 50};   (* Maximum number of rejections of one fracture of
                         given radius before its radius is decreased *)


(*=======================================================================*)
(* Rctangular Fractures Options
*)
(* NOTE: Number of elements must match number of rectangle families
*)
(*      (second number in nShape parameter)
*)
(*=======================================================================*)
(*rdist is a mandatory parameter if using statistically generated rectangles *)
rdistr={4,1};    (*  Rectangle statistical distribution options:
                 1 - lognormal distribution
                 2 - truncated power law distribution
                 3 - exponential distribution
                 4 - constant *)
(*=======================================================================*)
(* Parameters used by both lognormal and truncated power law options: *)
(* Mandatory Parameters if using statistically generated rectangles
*)
```

```
raspect = {1,1};  (* Aspect ratio. Used for lognormal and truncated
                          power law distribution. *)

rtheta = {1.3285,  1.213 };  (*Rectangle fracture orientation.
                         The angle the normal vector makes with the z-axis *)

rphi = { 5.93826, 3.355 }; (* Rectangle fracture orientation.
                       The angle the projection of the normal onto the x-y
                       plane makes with the x-axis *)

rkappa = { 12.99, 10.41};  (*Parameter for the fisher distributions. The bigger,
                   the more similar (less diverging) are the rectangle
                   familiy's normal vectors  *)

(*================================================================================*)
(* Options Specific For Rectangle Lognormal Distribution (rdistr=1): *)
(* Mandatory Parameters if using rectangles with lognormal distribution *)

rmean = {Log[2], Log[2]};    (* For Lognormal Distribution.
                                   Mean radius (1/2 rectangle length) in
                                   lognormal distribution for rectangles.
                                   The number inside the log is the mean value.
                                   So, if you want a mean fracture radius of 5,
                                   you must input Log[5]*)

rsd = {.3, .3};   (* Standard deviation for lognormal distributions of
                        rectangles *)

(*================================================================================*)
(* Options Specific For Rectangle Power-Law Distribution (rdistr=2): *)
(* Mandatory Parameters if using rectangles with power-law distribution *)

 rmin = { 1, 1 };      (* Minimum radius for each rectangle family.
                        For power law distributions. *)

 rmax = { 3, 3};     (* Maximum radius for each rectangle family.
                       For power law distributions. *)

 ralpha = {1.5, 1.5};   (* Maximum radius for each ellipse family.
                          For power law distributions. *)

rpowerLawReject={50, 50}; (* Maximum number of rejections of one fracture of giv
                       radius before its radius is decreased *)

(*================================================================================*)
(* Options Specific For Rectangle Exponential Distribution (edistr=3): *)
(* Mandatory Parameters if using rectangules with exponential distribution *)

rellmean = {3, 2};  (* Mean value for Exponential  Distribution.   *)
```

```
(*=========================================================================*)
(* Options Specific For Constant Size of rectangles (edistr=4): *)

rconst = {7,3};  (* Constant length for all the rectangles in the DFN
*)

(*=========================================================================*)
(*=========================================================================*)
(* User-Specified Ellipses                                            *)
(* Mandatory Parameters if using user-ellipses                        *)
(* NOTE: Number of elements must match number of user-ellipse families*)
(*(third number in nShape parameter)                                 *)
(*=========================================================================*)

ueb = {0.7, 0.7,  0.7, 0.7};    (* Radius for each user-ellipse *)

ueaspect = {1, 0.65,  1, 1};   (* Aspect ratio for each user-ellipse *)

uetanslation = {{-0.4,0,0}, {0,0,0},{0.4,0.,0.2},{0.4,0.,-0.2} };
        (* {x,y,z} Location of the center of each user ellipse *)
uenormal =     {{0,0,1},{1,0,0}, {0,0,1}, {0,0,1}};
        (* 3d normal vector direction for each user-ellipse*)
uenumPoints = {19, 19, 19, 19}; (* Number of verticies for each user-ellipse *)


(*=========================================================================*)
(* User-Specified Rectangles                                          *)
(* Mandatory Parameters if using user-rectangles                      *)
(* NOTE: Number of elements must match number of user-ellipse families*)
(* (fourth number in nShape parameter)                               *)
(*=========================================================================*)

urb = {1,  1,  1,  1};   (* Radius (1/2 rectangle length) *)

uraspect = {1, 0.65,  1, 1};  (* Aspect ratio for each user-rectangle *)

urtanslation = { {-0.9,0,0}, {0,0,0},{0.9,0.,0.2},{0.9,0.,-0.2}};
        (* {x,y,z} Location of the center of each user rectangle *)

urnormal = {{0,0,1},{1,0,0}, {0,0,1}, {0,0,1}};  (* 3d normal vector direction f
                                      each user-rectangle *)

(* If you would like to input user specified rectangles according to their
  coordinates, you can use the parameter userDefCoordRec. In that case, all
  of the user specified rectangles will have to be according to coordinates.
  You cannot specify user defined rectangles according to both
  coordinates and to parameters such as urb, uraspect, etc.*)
```

```
(* userDefCoordRec = { { {1,0,0},{6,0,10},{11,0,0},{6,0,-10} },
                    {{1,0,7},{7,6,7},{12,0,7},{7,-6,7}}}; *)
(* Format:*)
(* {{{x11, y11, z11},{x12, y12, z12}, {x13, y13, z13}, {x14, y14, z14}},
  {{x21, y21, z21},{x22, y22, z22}, {x23, y23, z23}, {x24, y24, z24}}};*)

(*WARNING: userDefCoordRec can cause LaGriT errors because the polygon
vertices are not put in clockwise/counter-clockwise order.
If errors, try to reorder the points till u get it right.*)

(*=====================================================================*)
(* Aperture [m]*)
(* Mandatory parameter, and can be specified in several ways:
- 1)meanAperture and stdAperture for using LogNormal distribution.
- 2)apertureFromTransmissivity, first transmissivity is defined, and then,
  using a cubic law, the aperture is calcuylated;
- 3)constantAperture, all fractures, regardless of their size, will have
  the same aperture value;
- 4)lengthCorrelatedAperture, aperture is defined as a function of fracture size

(*NOTE: Only one aperture type may be used at a time *)

(**** 1)meanAperture and stdAperture for using LogNormal distribution.********)
(*meanAperture =  Log[10^-3] ; *) (*Mean value for aperture using
                              normal distribution *)
(*stdAperture = 0.8; *) (*Standard deviation  *)

(****** 2)apertureFromTransmissivity, first transmissivity is defined,
  and then, using a cubic law, the aperture is calcuylated;***************)
(*apertureFromTransmissivity={1.6*10^-9, 0.8};*)
        (* Transmissivity is calculated as transmissivity = F*R^k,
           where F is a first element in aperturefromTransmissivity,
           k is a second element and R is a mean radius of a polygon.
           Aperture is calculated according to cubic law as
           b=(transmissivity*12)^1/3 *)

(******* 3)constantAperture, all fractures, regardless of their size,
   will have the same aperture value;   **********************************)

constantAperture = 10^-2; (* Sets constant aperture for all fractures *)

(******** 4)lengthCorrelatedAperture, aperture is defined as a function of
       fracture size *******************)

(*lengthCorrelatedAperture={5*10^-5, 0.5};   *)
        (*Length Correlated Aperture Option:
          Aperture is calculated by: b
=F*R^k,
          where F is a first element in lengthCorrelatedAperture,
        k is a second element and R is a mean radius of a polygon.*)


(*=====================================================================*)
(* Permeability *)
(* Options:
- Permeability of each fracture is a function of fracture aperture,
     given by k=(b^2)/12, where b is an aperture and k is permeability
- Or, you can specify a constant permeabilty for all the fractures*)

constantPermeability=2.5*10^-14;  (* Constant permeability for all fractures *)



(*=====================================================================*)
(* Mandatory Meshing Grid Refinement Parameters: *)

slope = 2; (* Factor used when refining fractures according to distance
        from fractures' intersection lines. The bigger this number
        the more refined the mesh will be closer to the intersection
        line (multiplies the reference field in message2
        in the files DFNgenerator/dfn/final_dfn.py or final_dfn _un.py). *)

refineDist = 0.5;(* Factor used when refidning fractures according to distance
        from fractures' intersection lines. The bigger this number
        the more refined the mesh will be closer to the intersection
        line (adds a value to the reference field in massage2
        in the files DFNgenerator/dfn/final_dfn.py or final_dfn _un.py). *)

(*=====================================================================*)
createVisReport = -1; (*  createVisReport=-1 produces report with no figures,
                        recommended on UBUNTU runs, or when no vis sets for
                        mathematica;
                        createVisReport= 0 produces only a few figures for
                        report;
                        createVisReport= 1 - produces full statistical report
                        stat.pdf
                        by default, createVisReport= -1. *)



End[]
```