Advances in Non-Random Discrete Fracture Network Simulation

Eric B. Niven and Clayton V. Deutsch

DFNSIM is a FORTRAN code created for the purpose of simulating discrete fracture networks. This paper reviews improvements made to DFNSIM over the past year. The most notable improvement to DFNSIM is that it is able to honour a target number of fracture intersections as it is now included as a component in the objective function. This article also reviews implementation of an anisotropic distance as an alternative proxy for fracture spacing and compares it to the previously used perpendicular distance between nearby fractures. There are advantages and disadvantages to both distance measures but the perpendicular distance is preferred.

Introduction

As easily accessible oil reserves continue to decline, producers are turning towards more complex and challenging reservoirs such as naturally fractured reservoirs (NFRs). A NFR is defined as "a reservoir in which naturally occurring fractures either have, or are predicted to have, a significant effect on reservoir fluid flow either in the form of increased reservoir permeability and/or reserves or increased permeability anisotropy" (Nelson, 2001). Carbonate reservoirs in the middle east are commonly thought of as NFRs. In order to optimize management of NFRs, detailed information on the behaviour, attributes and properties of the fracture network must be known along with those of the rock matrix.

A discrete fracture network (DFN) is usually created to model the fractures in a reservoir. Large scale faults and some large scale fractures show up on seismic surveys and are explicitly specified in the model. Small and medium scale fractures (those that do not show up on seismic surveys) are modelled probabilistically. Most commercially available DFN computer codes use a Poisson or a non-homogeneous Poisson process to generate fracture locations randomly (Cacas et al. 2001, Gauthier et al. 2002, Hitchmough et al. 2007). These codes also simulate fracture orientation independent of location. This process may lead to fractures from the same joint set that are unrealistically close together, unrealistically far apart or that cross at very low angles (see Niven and Deutsch (2010b) for more discussion on this matter).

At the 2010 annual meeting for the Centre for Computational Geostatistics, Niven and Deutsch (2010a, 2010b) introduced a new approach and algorithm for DFN simulation. DFNSIM works by generating more fractures than are required and finding a subset that matches target distributions of fracture length, intensity, intersections and local fracture spacing and orientation. The early version of DFNSIM showed promise, however several areas for improvement were identified over the course of the past year. Last year's version of DFNSIM:

- could only draw from normal distributions;
    - Some fracture attributes such as size are distributed lognormally while fracture orientation is often distributed according to a Fisher distribution. It is also possible that fracture attributes may not follow a parametric distribution.
- could not honour the number of fracture intersections seen in real data;
- was too slow, limiting the size of fracture networks that could be simulated;
- required the user to select objective function constants in a trial and error process;
    - This could require many lengthy computation runs to achieve a good match between input and target distributions of fracture attributes.
- finds nearby fractures using a "perpendicular distance" (described in more detail in Niven and Deutsch (2010a and b)).
    - The program requires the user to select a bandwidth and a perpendicular distance to search. If these two parameters are chosen poorly the program may incorrectly identify nearby fractures.

This article reviews some of the improvements made to DFNSIM over the past year aimed at addressing the concerns listed above.

Drawing Values From Non-Normal Distributions

The latest version of DFNSIM adds the ability to draw fracture orientations from a Fisher distribution. The Fisher distribution is analogous to the Normal Distribution on a sphere. It is parameterized by angles from the z (upwards positive) and x (east) axes (ϕ and θ, respectively) as well as a dispersion constant, κ:

$$f(\phi',\theta') = \frac{\kappa \sin\phi' e^{\kappa\cos\phi'}}{2\pi(e^{\kappa}-1)};\ 0 \le \theta' \le 2\pi \tag{1}$$

κ can be estimated from the following formula, which is valid if the number of fracture poles is greater than 30.

$$\kappa \cong \frac{N_f}{N_f - |R|} \tag{2}$$

|R| is the magnitude of the vector sum of the unit vectors for orientation.

Figure 1 shows simulated pole vectors using a fisher distribution. Two examples are shown to illustrate the effect of increasing the dispersion parameter. As the parameter, κ, increases, the cluster of points gets smaller around the mean vector. Uniform dispersion across the sphere can be specified by κ= 0.

Fracture length is often thought to be lognormally distributed (Belfield, 1998). Thus, the ability to draw from the lognormal distribution is also implemented in the latest version of DFNSIM.

In practice, few variables are perfectly modeled by parametric distributions. In these situations, the ability to draw values directly from the input data distributions can be valuable. For that reason, DFNSIM can handle non-parametric distributions from which to draw values from. The non-parametric distribution is described with a cumulative distribution function specified in a file with GEO-EAS format (see Figure 2).

Objective Function Changes

Niven and Deutsch (2010c) demonstrated that modelling some natural fracture networks with typical DFN simulation algorithms (specifically where fracture orientation is drawn independently of fracture location) generates too many fracture intersections. Earlier versions of DFNSIM could not honour the number of fracture intersections seen in some natural fracture networks. Additionally, DFNSIM could not honour the target fracture length distribution.

The latest version of DFNSIM adds objective function components for the number of fracture intersections in the DFN and the fracture length distribution (Equation 3). DFNSIM optimizes by any or all of the components of the objective function:

$$\begin{aligned}\text{Obj. Fn.} = {} & C_{spacing}\sum_{i=1}^{sbins}\left(S_{input,i} - S_{activated,i}\right)^2 + C_{Length}\sum_{i=1}^{lbins}\left(L_{input,i} - L_{activated,i}\right)^2 \\ & + C_{Inter-Angle}\sum_{i=1}^{abins}\left(A_{input,i} - A_{activated,i}\right)^2 + C_{Intersections}\left(Inter_{input,i} - Inter_{activated,i}\right)^2 \\ & + C_{Intensity}\sum_{i=1}^{ibins}\left(I_{input,i} - I_{activated,i}\right)^2\end{aligned} \tag{3}$$

Where:

- $S_{input,i}$ and $S_{activated,i}$ are the target (input) fracture spacing distribution and the fracture spacing distribution of the activated DFN, respectively. *sbins* is the number of fracture spacing bins.
- $L_{input,i}$ and $L_{activated,i}$ are the target (input) fracture length distribution and the fracture length distribution of the activated DFN, respectively. ibins is the number of fracture length bins.

- $A_{input,i}$ and $A_{activated,i}$ are the target (input) nearest fracture inter-angle distribution and the nearest fracture inter-angle distribution of the activated DFN, respectively. *abins* is the number of inter-angle bins.
- $Inter_{input,i}$ and $Inter_{activated,i}$ are the target number of fracture intersections and the actual number of fracture intersections in the DFN, respectively.
- $I_{input,i}$ and $I_{activated,i}$ are the target (input) fracture intensity and the fracture intensity of the activated DFN, respectively. *ibins* is the number of intensity bins.
- $\mathrm{C}_{\mathrm{spacing}}, \mathrm{C}_{\mathrm{length}}, \mathrm{C}_{\mathrm{inter\text{-}angle}}, \mathrm{C}_{\mathrm{Intersections}}, \mathrm{C}_{\mathrm{intensity}}$ are coefficients which serve to make the objective function unit-less in order to compare components with different original units.

Previously in DFNSIM, the C-coefficients were set by the user via trial and error. However, finding the right coefficients can be difficult and may require more computation runs than are necessary. The new version of DFNSIM automatically calculates the coefficients using a simple algorithm:

1. First an initial DFN is calculated.
2. Then, a base objective function is calculated based on the initial DFN and the target distributions.

$$O_{base} = C_1 O_1 + C_2 O_2 + C_3 O_3 + C_4 O_4 + C_5 O_5 \tag{4}$$

   Where C_i and O_i (i=1 – 5) refer to the constants and components of the objective shown in Equation 3.
3. Visit a fracture randomly.
4. Change its activation. If the fracture is activated, deactivate it and vice versa.
5. Recalculate the objective function and compare it to the base objective function.

$$O_{i,change} = \left| O_{i,base} - O_{i,j} \right| \tag{5}$$

6. Reject the change, regardless of whether it improved the objective function or not.
7. Go to 3, until N fractures are visited and changed.
8. Calculate the average objective function change.

$$dO_i = \left(1 \middle/ N \right) \sum_{j=1}^{N} \left| O_{i,base} - O_{i,j} \right| \tag{6}$$

9. Calculate objective function constants. $F_{i,scaling}$ is a scaling factor which can be applied to each constant to increase or decrease the importance of that component of the objective function relative to the others.

$$c_i = \frac{1}{dO_i} F_{i,scaling} \tag{7}$$

Calculating Fracture Intersections

As noted, the latest version of DFNSIM calculates and tracks fracture intersections. Intersections are checked in a two-step process. First, two fractures are selected and the intersections of their bounding boxes are checked (Figure 3). Checking the intersection of bounding boxes is extremely fast. If the bounding boxes do not intersect, the program moves on to check other fractures. If the bounding boxes do intersect, then a possible intersection between the actual fractures is checked. If the fracture intersection is detected, it is saved in memory. Additionally, if both fractures are 'activated', the intersection is also stored separately. This allows the program to keep a running total of intersections between activated fractures.

Computation Speed Improvements

The previous version of DFNSIM re-calculated the distance to the nearest fracture (local fracture spacing) each time a fracture was activated or deactivated. The newest version of DFNSIM implements 'local updating' around activated and deactivated fractures. This is achieved by pre-calculating and storing those distances in memory. When a fracture is activated, its distance to the nearest fracture is simply added to the distribution of local fracture spacing and vice versa. This change by resulted in a reduction in computation time by over a magnitude.

Calculating fracture intersections and using them as a component in the objective function results in an increase in computation speed. However, the above-mentioned 'local updating' as well as other coding improvements has resulted in significant speed improvements for DFNSIM. The current version of DFNSIM is approximately two orders of magnitude faster than it was as presented at the 2010 Annual CCG Meeting. This means that DFNSIM is capable of simulating and optimizing a 10 million fracture model in less than one day (depending on processor speed).

Calculating the Distance to the Nearest Fracture

Previous versions of DFNSIM have always used the perpendicular distance to the nearest neighbouring fracture (See Figure 4) as a proxy for fracture spacing. A search strategy is implemented where the superblock search is first used to identify nearby fractures. Then those fractures identified by the superblock search are further screened by a bandwidth and perpendicular distance to search as shown in Figure 5. The fractures that fall within the bandwidth and perpendicular distance to search are then subjected to the perpendicular distance calculation as shown in Figure 4. The DFNSIM algorithm requires the user to specify the bandwidth and perpendicular distance to search.

Using the perpendicular distance to identify nearby fractures works well. However, there are a some drawbacks:

1. Finding enough nearby fractures near the corners of models is an issue if:
    a. The user specifies the retention of a high number of nearby data (identified by the superblock search) for the perpendicular distance calculation.
    b. Either the bandwidth or perpendicular distance to search is too small.
2. There is a tradeoff between the number of fractures that can be identified and stored in memory and the bandwidth that is chosen. i.e. problems will arise if the user wishes to retain more data for the perpendicular distance calculation than can be identified within the bounds of the bandwidth and perpendicular distance to search.
3. Because of points 1 and 2 (above) selecting an appropriate bandwidth and perpendicular distance to search can sometimes be difficult.

Figure 6 illustrates the problem. The figure shows two DFNs. Fractures are shown in 2D as black lines with centroid locations indicated. The red arrows go from one centroid to another indicating which fracture is nearest to another fracture. The arrowhead represents the nearest fracture to the fracture at the tail of the arrow. A wide bandwidth is used for the left side DFN. The problem is that very small perpendicular distances are calculated for fracture centroids that are fairly far apart. Note that the arrows don't indicate perpendicular distance – only which fracture is closest to the other using a perpendicular distance. Clearly using such a wide bandwidth does not result in a fair approximation of fracture spacing. In order to fix this problem the bandwidth is reduced. In the right side of the figure, the bandwidth is approximately equal to the average fracture length. Here the nearest fractures make much more sense when the notion of a perpendicular distance is taken into account.

Another problem with the perpendicular distance and bandwidth concept occurs near corners of a model. If the bandwidth is too narrow, only a few nearby fractures may be identified. Thus, there is a practical lower limit to the choice of bandwidth.

It was thought that using an anisotropic distance in place of the perpendicular distance might alleviate some or all of the limitations mentioned above. In order to use the anisotropic distance, first the fracture is discretized into four points at each corner of the fracture along with its centroid. One of the five points is chosen on each of two fractures. A vector h is calculated between the two points. The vector

h is resolved into 3 components. One component is perpendicular to the fracture, one is in plane and horizontal and one is in plane and vertical. This is illustrated in Figure 7.

Next, an anisotropic distance is calculated as an effective distance as follows:

$$h_{anisotropic} = \sqrt{\left(\frac{h_{\text{horiz, in plane}}}{a_{\text{horiz, in plane}}}\right)^2 + \left(\frac{h_{\text{vert, in plane}}}{a_{\text{vert, in plane}}}\right)^2 + \left(\frac{h_{\text{perpendicular}}}{a_{\text{perpendicular}}}\right)^2} \quad (8)$$

Anisotropy constants, a_{horiz}, a_{vert} and $a_{perpendicular}$, are specified by the user. This is immediately identifiable as a disadvantage since there is little to guide the user in selecting an appropriate values for a.

Figure 8 shows two examples of nearest fractures calculated using the anisotropic distance. In the figure, the arrows proceed from one point on a fracture to the nearest point on another fracture as defined by the effective anisotropic distance. In the example on the left $a_{in\ plane}$ is set to 1.0 and $a_{perpendicular}$ is set to 10. The results appear reasonable and are actually similar to the perpendicular distance method in that the nearest fractures identified are largely similar. The right side of the figure shows an example where $a_{perpendicular}$ is increased to 50. In this case, the anisotropy ratio between perpendicular and in-plane distance is too high. The result is a number of instances where the nearest fractures are much farther away and are not reflective of what reasonable nearest fractures should be.

Figure 9 shows the variation in the distribution of perpendicular distance calculated for various bandwidths. As is shown in the figure, the range of variation is fairly small for perpendicular distances between 70 and 140. Unfortunately it is difficult to directly compare anisotropic distances with different values of $a_{perpendicular}$ since the constants strongly scale the resultant anisotropic distance. As a result, the length of the arrows (Euclidean distance) is calculated and used to represent the anisotropic cases shown in Figure 9. Figure 10 shows the variation in the distribution of the length of the nearest fracture arrows. There is significant variation on the distributions with increasing anisotropy.

The two figures (Figure 9 and 10) indicate that the distances to the nearest fractures are less sensitive to the choice of bandwidth (when using perpendicular distance) than they are to the ratio between $a_{perpendicular}$ and $a_{in\ plane}$.

The question of which distance calculation method is best as a proxy for fracture spacing is not easy to answer. On one hand, the perpendicular distance is faster (marginally), has units and represents a "real" distance and its results are less sensitive to the user's parameter choice (bandwidth). On the other hand, finding enough nearby fractures near the corners of the model can be problematic if the bandwidth is too narrow.

If the anisotropic distance is used as a proxy for fracture spacing, the aforementioned corner effects can be avoided. The calculation is also more robust in that there is no danger of not finding enough nearby fractures (since there is no trimming by bandwidth). However, the calculation is more sensitive to the user's parameter choice (anisotropy ratio).

The first author prefers the perpendicular distance over the anisotropic distance since it is less sensitive to the user's parameter choice.

Conclusions

The current version of DFNSIM implements several code improvements resulting in a decrease in computation time by around two orders of magnitude. Currently DFNSIM is able to simulate and optimize a model with 10 million fractures in less than a day.

An algorithm for automatically calculating objective function constants has been implanted. In theory, this has the advantage of avoiding the trial-and-error manual tuning that previous versions of DFNSIM required. However, preliminary trials has shown that final DFN matches to target distributions are poorer. Scaling factors can be used to improve the match, however determining scaling factors is largely subject to trial and error and the resulting optimized DFNs still don't match the target as well as the previously used manual tuning approach.

An option to use an anisotropic distance was implemented in DFNSIM and compared to the perpendicular distance method. Each method has advantages and disadvantages but since the anisotropic

distance is more sensitive to the user specified parameter (anisotropy ratio), the original perpendicular distance is preferred.

References

Belfield, W.C. 1998. Incorporating spatial distribution into stochastic modelling of fractures: multifractals and levy-stable statistics. Journal of Structural Geology, 20(4). 473-486.

Cacas, M.C., Daniel, J.M., and Letouzey, J. 2001. Nested geological modelling of naturally fractured reservoirs. Petroluem Geoscience, 7(S): S43-S52.

Gauthier, B.D.M., Garcia, M., and Daniel, J.M. 2002. Integrated fractured reservoir characterization: a case study in a North Africa field. SPE Reservoir Evaluation & Engineering. SPE 79105. 284-294.

Hitchmough, A.M., Riley, M.S., Herbert, A.W. and Tellam, J.H. 2007. Estimating the hydraulic properties of the fracture network in a sandstone aquifer. Journal of Contaminant Hydrology. 93(1-4):38-57.

Niven, E.B. and Deutsch, C.V. 2010a. A new approach to DFN simulation. Paper 102, Report 12, Centre for Computational Geostatistics, University of Alberta, Edmonton, Alberta.

Niven, E.B. and Deutsch, C.V. 2010b. DFNSIM: A new program for simulating discrete fracture networks. Paper 402, Report 12, Centre for Computational Geostatistics, University of Alberta, Edmonton, Alberta.

Niven, E.B. and Deutsch, C.V. 2010c. On the Randomness of Natural Fractures. Paper 207, Report 12, Centre for Computational Geostatistics, University of Alberta, Edmonton, Alberta.

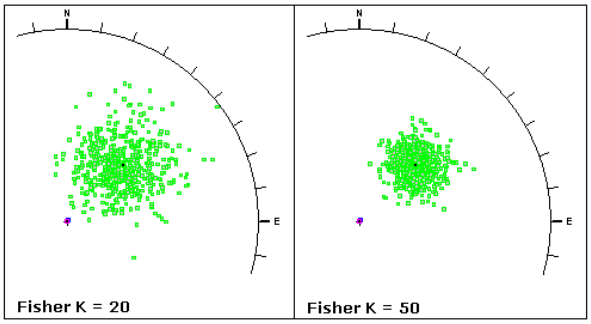
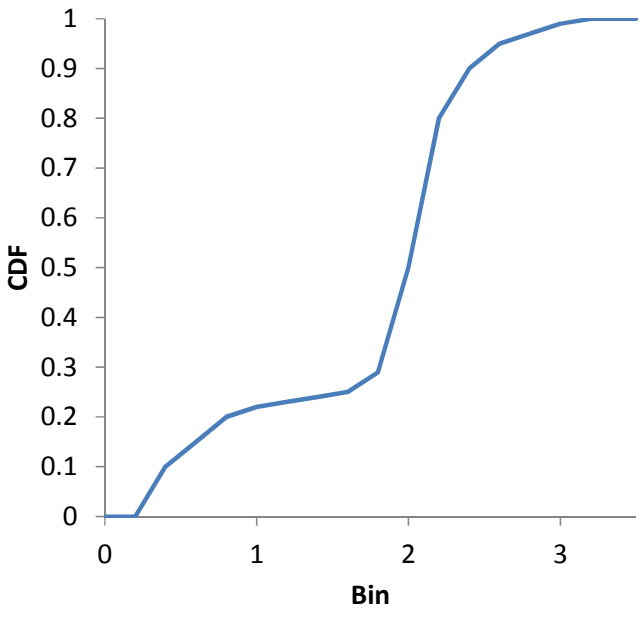


Figure 1: Fracture poles simulated using the fisher distribution for two values of κ.




```
Fracture Length
2
Bin
CDF
0       0
0.2     0
0.4     0.1
0.6     0.15
0.8     0.2
1       0.22
1.2     0.23
1.4     0.24
1.6     0.25
1.8     0.29
2       0.5
2.2     0.8
2.4     0.9
2.6     0.95
2.8     0.97
3       0.99
3.2     1
3.5     1
```

Figure 2: Non-parametric CDF (left) and associated GEO-EAS file format.

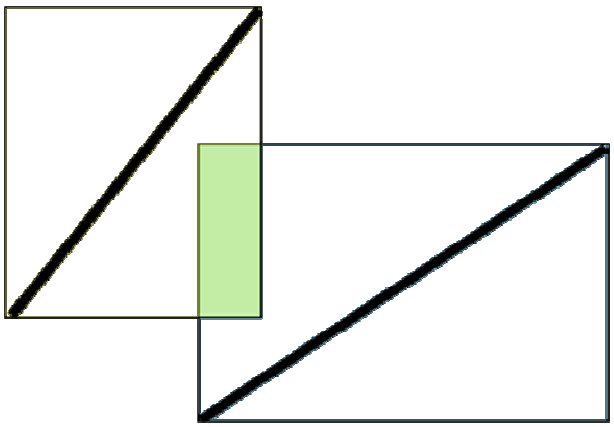


Figure 3: Checking to see if two fractures (represented by thick black lines) intersect. First the fracture bounding boxes are checked for intersection. If the bounding boxes intersect, the fracture planes are checked for a possible intersection.

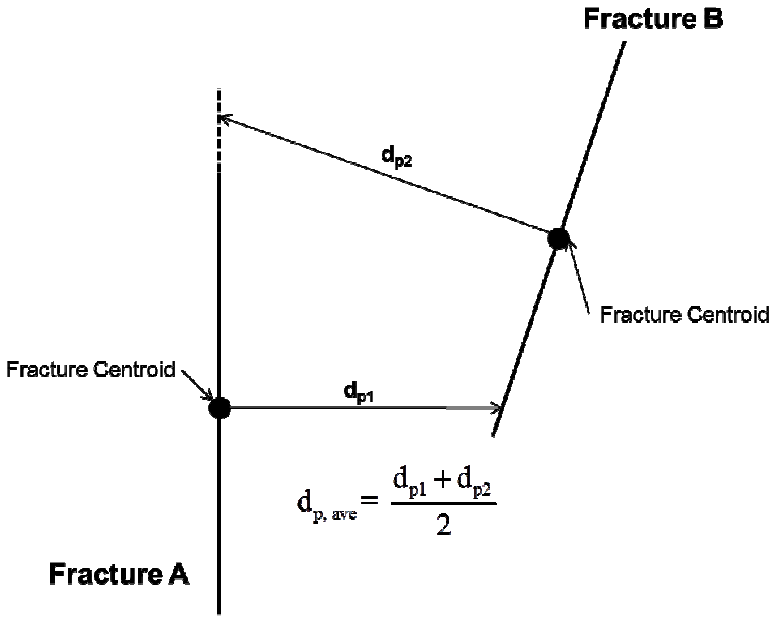


Figure 4: Perpendicular distance to the nearest neighboring fracture calculation.

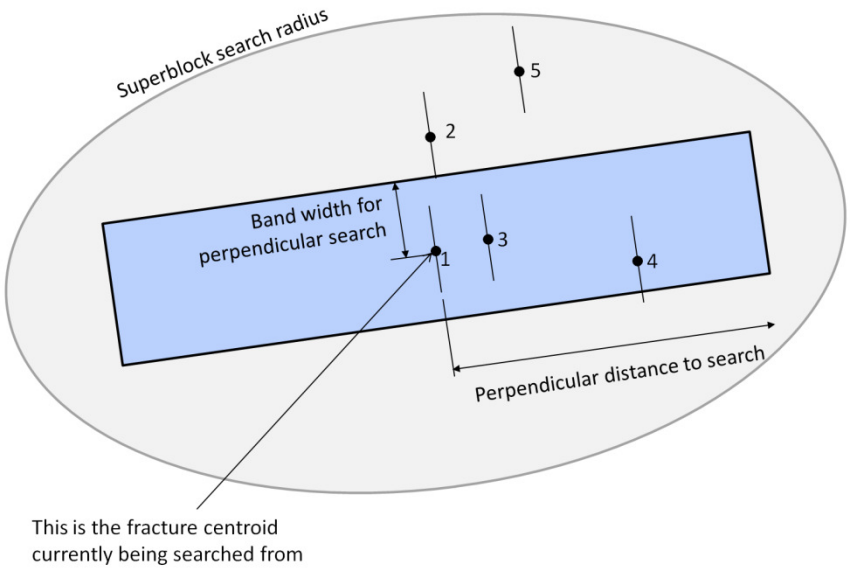


Figure 5: Perpendicular distance search strategy.

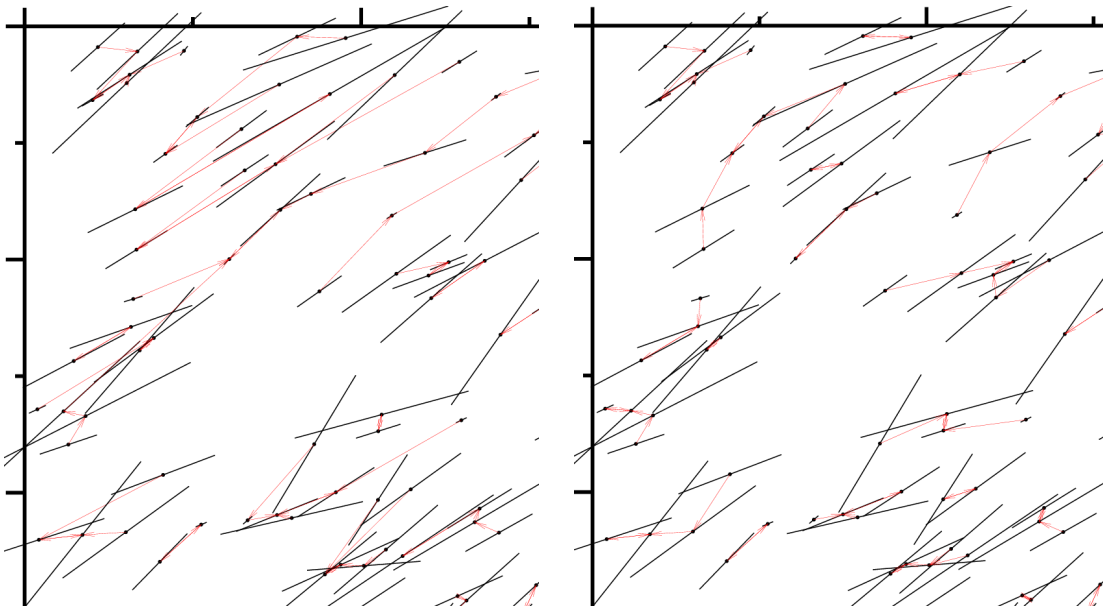


Figure 6: Nearest fractures calculated using a wide bandwidth (left) and nearest fractures calculated using a narrow bandwidth (right).

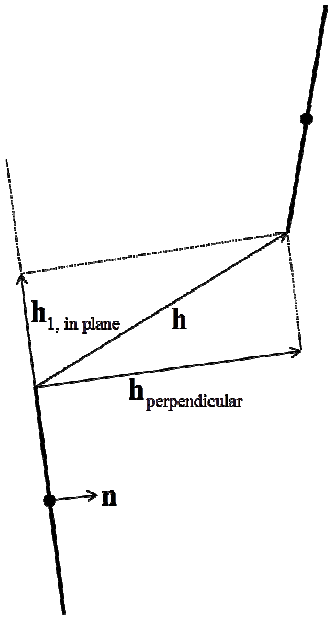


Figure 7: Illustration of anisotropic distance.

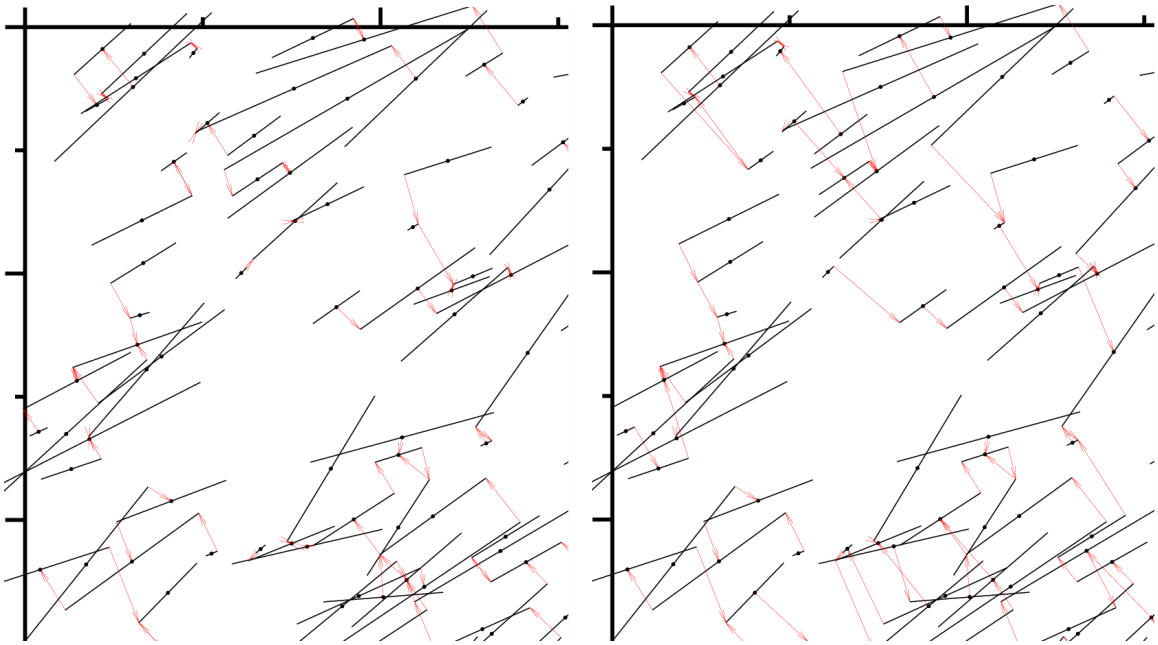


Figure 8: Nearest fractures calculated using an anisotropic distance (ratio = 10 on left and ratio = 50 on right).

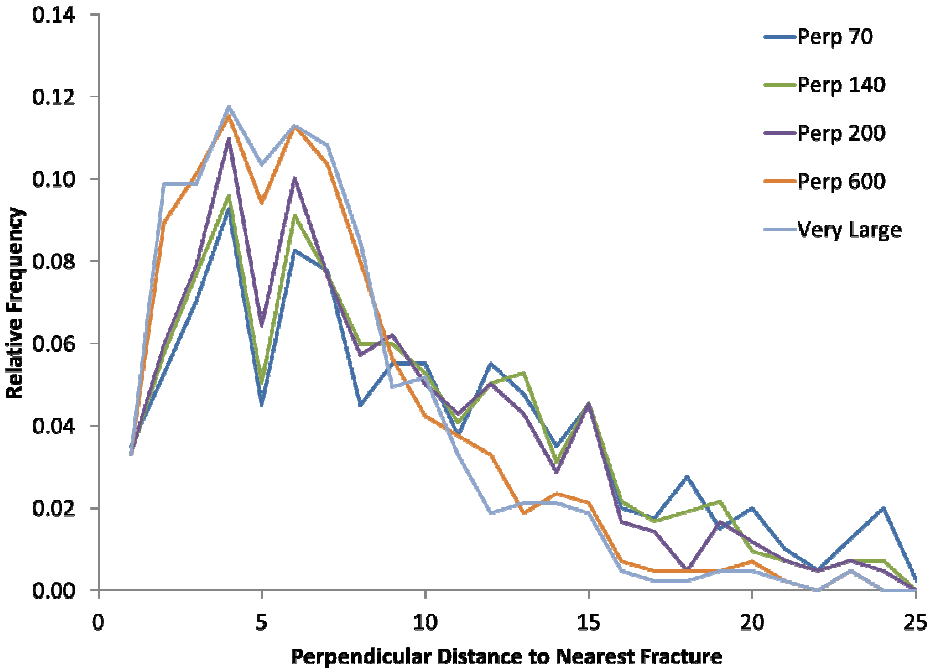


Figure 9: Variation in histograms of fracture spacing with increasing bandwidth.

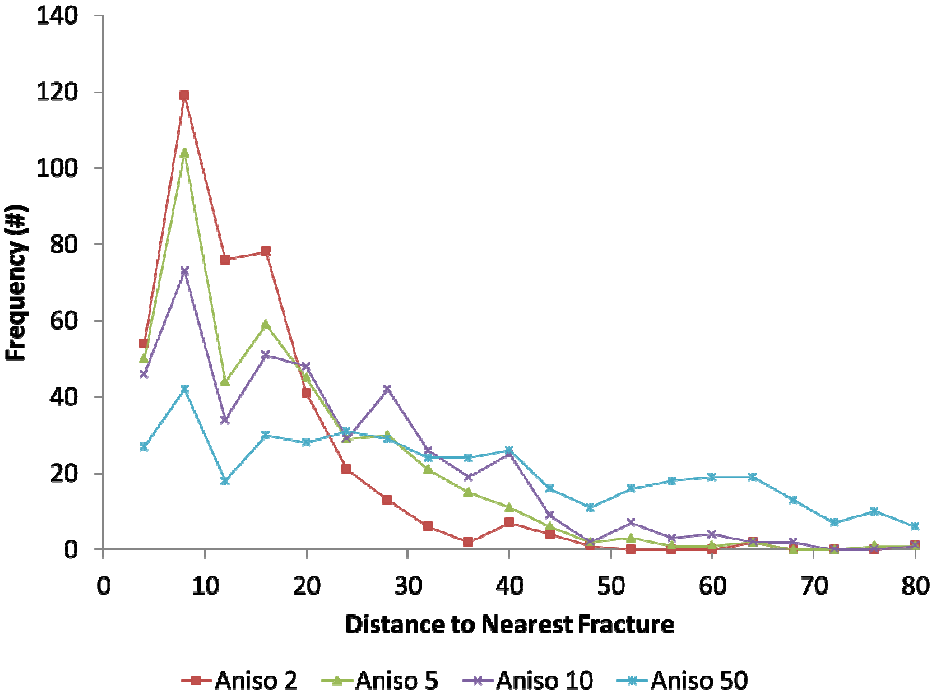


Figure 10: Variation in fracture spacing with increasing anisotropy ratio.