IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDECINE

Department of Earth Science and Engineering
Centre for Petroleum Studies

Computation of effective dynamic properties of naturally fractured reservoirs:

Comparison and validation of methods

By

Benoit Decroux

A report submitted in partial fulfilment of the requirements for the MSc and/or the DIC.

September 2012

DECLARATION OF OWN WORK

I declare that this thesis

Computation of effective dynamic properties of naturally fractured reservoirs: Comparison and validation of methods

is entirely my own work and that where any material could be construed as the work of others,
it is fully cited and referenced, and/or with appropriate acknowledgement given.

Signature: .....

Name of student: Benoit DECROUX

Names of supervisor: Prof. Olivier GOSSELIN

ACKNOWLEDGEMENT

First, I would like to express my sincere thanks to my supervisor Prof. Olivier Gosselin, from Imperial College, for his support, guidance and advice during this project.

I would also like to express my sincere gratitude to Prof. Sebastian Geiger from Heriot-Watt University, Prof. Stephan K. Matthäi from the University of Leoben, Dr. Adriana Paluszny from Imperial College and Indira Kospanova from Total E&P from their very generous and invaluable help with the use of the software CSMP++.

I express my sincere gratitude to Michel Garcia from Kidova and to Alan Irving from Total to for their precious help with the use of GoFraK.

In addition, I would like to thank all my professors from the MSc Petroleum Engineering who shared their knowledge with me.

I would like to thank all my colleagues and friends from the department of Earth Science and Engineering and especially Stephen Chukwudi, Jhonatan Valdiviano Huertas and Xiao Shuyue for their invaluable help, remarks, advice and encouragement during my project.

Finally, I would like to thank my family, and especially my parents, who allowed me to study at the Imperial College and who supported me all along my studies.

TABLE OF CONTENT

TABLE OF FIGURES

TABLE OF TABLES

Imperial College
London

Computation of effective dynamic properties of naturally fractured reservoirs: Comparison and validation of methods

Student: Benoit Decroux

Imperial College supervisor: Prof. Olivier Gosselin

Abstract

Naturally Fractured Reservoirs (NFR) are very heterogeneous media and therefore are difficult to simulate directly. To reduce the computation time it is common to use either the dual-porosity or the dual-permeability model where fractures are represented as a continuous medium in communication with the rock matrix. The transformation from a Discrete Fracture Network (DFN) to a continuous equivalent medium requires the calculation of effective properties. This procedure is often improperly referred as an upscaling process.

Many calculation methods for the computation of the effective permeability have been proposed. Analytical methods rely on assumptions which are seldom met in practice. Numerical methods are known for being more accurate. They use simulations on either a global Discrete Fracture Network (DFN), defined in the entire reservoir, or on local DFNs defined in each cell.

New simulators using the Discrete Fracture and Matrix (DFM) model which enables to simulate explicitly NFRs have been developed. They offer an opportunity to estimate the accuracy of the various effective permeability calculation methods.

The development of new simulators using the Discrete Fracture and Matrix (DFM) model, which enables to simulate explicitly a NFR, offers a new opportunity to estimate the accuracy of the various effective permeability calculation methods. Explicit simulations realised with the DFM model can be compared with simulation based on effective properties.

In this paper, a fracture network was simulated explicitly on a Cartesian grid using Eclipse .This simulation was compared against simulations based on effective properties. The tested calculation techniques are an analytical method (the Oda's technique); numerical methods using a global DFN with different boundary conditions (impermeable boundaries or linearly varying pressure) and a numerical method using a local DFN where no boundary conditions need to be set (the IBPOS technique developed in the GoFraK plugin of GoCaD).

In a second part a demonstration is done of how the software CSMP++ (which uses the DFM model) can be used to evaluate the effective permeability of a reservoir block and how it can be compared with other calculation techniques.

The results showed the importance of the fracture network connectivity on the effective permeability and highlighted the weakness of techniques such as the Oda's method which do not take this parameter into account.

Introduction

A large proportion of the remaining oil resources resides in Naturally Fractured Reservoirs (NFR). These reservoirs are characterised by a high heterogeneity which makes prediction of flow behaviour more difficult and increases the risk of early water breakthrough. The localisation of injection wells has a key importance since the presence of fractures connecting them to production wells can prevent water to penetrate into the matrix and then dramatically decrease the sweep efficiency and the recovery factor. For this reason the effect of fractures on the reservoir permeability anisotropy should always be taken into account when choosing a development plan for a NFR.

The characterisation of a reservoir as a NFR is not straightforward because fractures are not always detected at the early stage of reservoir development and because, even if they are identified, their effect on flow behaviour is generally unknown (Bourbiaux 2010). Hence, many reservoirs are initially considered as non-fractured and are re-qualified when water breakthrough is observed sooner than what was expected (Wayne et al. 2006).

In order to understand the effect of fractures better, NFR are classified in four categories according to the relative contribution of fractures and matrix to oil storage and permeability (cf. Figure 1) (Nelson 2001). In type I reservoirs the fractures provide the essential storage capacity as well as the permeability in the reservoir. In type II reservoirs the matrix provides the essential storage capacity and the fractures provide the permeability. In type III reservoirs the fractures improve the permeability of a reservoir which had already a good matrix permeability and porosity.

The prediction of the recovery factor for a NFR demands, even more than for other reservoirs, a fully integrated study between geoscientists, geophysicists and reservoir engineers (Astratti et al. 2010). Iterations must be done between the modelling of the fractures and calibration against dynamic data using simulations.

The simulation of NFR is a multi-scale problem since fracture aperture is generally several millimetres wide, while fracture length can range between some centimetres to hundreds of metres and reservoir dimensions are of several kilometres.

Direct simulation of all the fractures in this kind of reservoir, although not totally impossible (Geiger et al. 2009), is very long and difficult. To overcome this difficulty reservoirs are divided in Representative Elementary Volumes (REV) of the fracture network (Long et al. 1982; Kfoury 2004) and for each of them the effective fracture properties (the effective porosity, effective permeability and characteristic block size) are calculated. This process, often improperly referred as an upscaling process, must be done each time the geological model is modified to perform new simulations.

The calculation time necessary for the computation of effective properties must remain reasonable since a lot of

iterations can be done between modifications of the geological model and comparisons with dynamic data (Souche et al. 2009). However, the accuracy of the effective properties has a key importance. If these properties are wrong, simulation results do not represent the behaviour of the defined geological model. So, even if a very accurate geological model has been built, the estimation of the recovery factor for a given production scenario can be wrong if the effective properties are improperly calculated (Ahmed and Geiger 2011).

In practice, the differences between simulation results and real production data come from the uncertainty of the geological model (which results from our lack of information about the reservoir), from the errors in the calculation of the effective properties and from the errors due to the simplification assumed by the chosen simulation model. The role of a reservoir engineer is estimate the uncertainty, to minimise these errors and while keeping a reasonable simulation time.

This paper proposes to estimate the differences in production simulation for different calculation methods of the effective fracture network permeability. In a first part, the usual modelling process of a NFR, the various available simulation models and the effective property calculation methods for NFRs will be presented. In a second part, two simple "Cartesian" fracture networks will be simulated with an explicit simulation used as a reference solution and compared with simulations based on 4 effective permeability calculation techniques (3 implemented in Petrel and one in the GoCad plugin named GoFraK). In the third part, a more realistic fracture network (FRACS2000) will be simulated explicitly using the software named CSMP++ and will be compared with simulations based on effective properties.

Modelling process of a Naturally Fractured Reservoir

Building of the geological model

Fractures in NFR are classified in fracture sets (Cacas et al. 2001; Bourbiaux et al. 2002). These sets represent families of fractures which share a similar orientation (characterised by the dip and the azimuth), length, aperture and permeability. These characteristics are referred as "non-spatial parameters" since they are similar for all fractures of a given set and do not depend on the localisation.

The primary sources of information about the presence of fractures are core data and image logs (Astratti et al. 2010). Fracture sets are identified by regrouping fractures with a similar orientation. The length of the fractures is more difficult to measure and can be estimated from outcrop analogues. The fracture aperture and permeability are linked by the "cubic law" which is the equation of a laminar flow between two parallel plates where the tortuosity is neglected (cf. equation 1).

$$k_f = \frac{a^2}{12} \qquad \text{(Equation 1)}$$

The aperture is estimated by calibration of the fracture permeability using well test analysis or history matching.

Once all the fracture sets have been defined with all their non-spatial parameters, the Fracture Density (FD) of each set must be interpolated in the entire reservoir (Gauthier et al. 2002). The FD can be defined for example as a cumulative surface of fractures per unit volume or a cumulative length of fractures per unit area (Garcia et al. 2007).

The FD is estimated in each well using mainly log images and cores. In order to interpolate it in the rest of the reservoir a relationship must be found between the FD and geological or geophysical parameters known everywhere in the reservoir (Macé 2006). These parameters can be: the type of lithology (or facies), the reservoir horizons curvature, the distance to faults, and the bed thickness.

Once the fracture set spatial (the FD) and non-spatial parameters are established an implicit fracture network is defined. At this point the fracture effective properties can be calculated using some specific techniques. However, most of the effective properties calculation methods require the definition of a Discrete Fracture Network (DFN) where all the fractures are explicitly represented.

In a DFN, to increase the realism of the model, all the fractures of the same set do not have exactly the same non-spatial parameters (length, aperture, orientation and permeability). These parameters are generated using probability laws. It results that many different DFNs can be generated from the same implicit model.

These fracture networks have all the same fracture density but they can have very different network connectivities. To chose which DFN is the most realistic, in GoFraK, a new parameter named the "connectivity factor" must be defined. This parameter is a measure of the number of fractures intersecting one another (Macé 2006; Delorme et al. 2008; Ozkaya 2011). It is estimated from well data calibration and must be preserved for any effective permeability calculation across the whole reservoir.

The fracture length and aperture can be estimated using a normal or log-normal distribution law. The permeability is derived from the aperture using the "cubic law" and the orientation can be calculated using the Von Mises-Ficher distribution law (Vigier 2009).

There is little information related to the shape of the fractures. In Petrel, they are assumed to be rectangular and to have a constant elongation ratio between their length and their width. In GoFraK, the reservoir is layered in so called "mechanical units". Each fracture is assumed to be rectangular and to fully cross at least one mechanical unit in the vertical direction.

Fracture simulation models

Several simulation models can be used to forecast a NFR oil production:

A first one, called the Discrete Fracture Network model, is a model which requires to represent explicitly all the

fractures. In this model, the flow is only simulated in fractures and interactions with the matrix are neglected (Ding et al. 2005). It can be used to simulate reservoirs of type I in the Nelson classification where the matrix has a low permeability and porosity.

An extension of the DFN model is called the Discrete Fractures and Matrix (DFM) model. This model has been implemented in the software named CSMP++ (Matthäi et al. 2005; Belayneh et al. 2006; Matthäi et al. 2007; Geiger et al. 2007; Belayneh et al. 2009; Geiger et al. 2009). In this model the flow is simulated explicitly in the fractures and in the matrix. It is the most accurate model and it can be used for any type of reservoir. It requires the use of an unstructured grid honouring the fracture geometry (see Figure 2). These grids need a prohibitively high number of elements to represent a full field (at least ten millions of cells). Therefore, it remains difficult to use the DFM model for the simulation of an entire reservoir but it can be interesting to simulate small parts of the reservoir (e.g. around a well) and compare the results with other models.

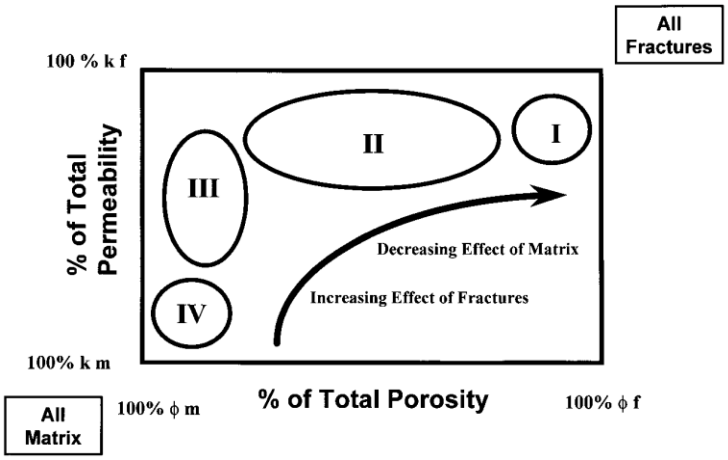

Figure 1: Classification of NFRs according to the relative contribution of fractures and matrix to the oil storage and the permeability (modified from Nelson 2001).

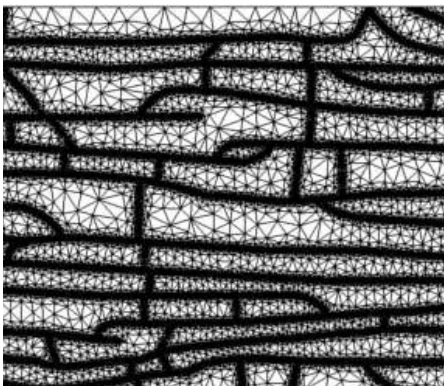

Figure 2: Unstructured grid mesh for a DFM simulation model (from Belayneh et al. 2006)

The currently most used simulation models are all based on effective fracture network properties. In these models fractures are not explicitly represented but an equivalent virtual medium is considered with effective properties (the effective porosity and permeability) representing the dynamic properties of the flowing domain (fractures).

In single-medium models, simulations are performed exactly the same way as for non-fractured reservoirs. The only difference is that block properties (porosity and permeability) are not the ones of the matrix, or of the fractures, but are global effective properties. If a Nelson type I reservoir is considered these effective properties represent only the behaviour of the fractures and the matrix is not simulated. This model, although very simple, is rarely used in the industry because it does not represent accurately the exchanges between the fractures and the matrix when a fluid is injected into the reservoir (Matthäi 2007).

The dual-porosity model (Barenblatt et al. 1960, Warren and Root 1962, Kazemi et al. 1976) can be used to simulate reservoir of type II in the Nelson classification. In this model, the number of reservoir cells is doubled compared with the singe-medium one. Two blocks are present at each location: one block represents the matrix and its properties are calculated the same way as for non-fractured reservoirs. The other block represents the fractures; its properties are calculated using effective property calculation methods. Blocks representing the fractures are connected with their neighbours but the matrix blocks are only connected with their associated fracture block (cf. Figure 3). When creating this last connection the exchange surface area between the fracture network and the matrix must be calculated.

The dual-porosity/dual-permeability model is an extension of the previous model with the difference that matrix blocks are also connected with their neighbours (cf. Figure 4). This model enables to simulate Nelson type III reservoirs since fluids can flow in the matrix domain as well as in the facture domain. Its main drawback is that it increases the simulation time compared with the dual-porosity model (Lange et al. 2004).

Effective property calculation methods

The effective properties of a fracture network are the properties of a virtual homogenous medium which have the same dynamic behaviour during simulation. These properties are the permeability, the porosity and the shape factor (which measures the exchange surface area between the fractures and the matrix).

The effective porosity is calculated in each reservoir block by dividing the volume occupied by all the fractures by the block volume. The main assumption of this calculation is that fractures are constituted of empty space.

The shape factor is calculated using the equation 2 from Kazemi et al. 1976). The parameter Lx, Ly and Lz represent the typical lengths of a matrix block surrounded by fractures. These lengths must be estimated using statistical techniques.

$$\sigma = 4\left(\frac{1}{{L_X}^2} + \frac{1}{{L_Y}^2} + \frac{1}{{L_Z}^2}\right) \qquad \text{(Equation 2)}$$

The effective permeability is represented as a tensor in the form of the Darcy's law presented in the equation 3. This enables to take into account the fracture network anisotropy.

$$\vec{q} = -\frac{\bar{\bar{k}}}{\mu}\,\overrightarrow{grad}(p) \qquad \text{(Equation 3)}$$

This tensor can be represented as a symmetrical matrix once a basis of three vectors $(\vec{e_X}, \vec{e_Y}, \vec{e_Z})$ has been defined (cf. equation 4). This basis is determined by the reservoir grid orientation and can be different in each block (cf. Figure 5).

$$Mat\left(\bar{\bar{k}}, (\vec{e_X}, \vec{e_Y}, \vec{e_Z})\right) = \begin{pmatrix} k_X & k_{XY} & k_{XZ} \\ k_{XY} & k_Y & k_{YZ} \\ k_{XZ} & k_{YZ} & k_Z \end{pmatrix} \qquad \text{(Equation 4)}$$

When a pressure gradient is applied in the reservoir in the direction of $\vec{e_X}$, a flow proportional to k_X is observed in the direction of $\vec{e_X}$, a flow proportional to k_{XY} is observed in the direction of $\vec{e_Y}$ and a flow proportional to k_{XZ} will be observed in the direction of $\vec{e_Z}$. The parameters k_{XY}, k_{XZ} and k_{YZ} are called the "cross-flow terms" of the permeability matrix. They are usually neglected during dynamic simulations and only the flow parallel to the pressure gradient is taken into account. To minimize the error caused by this simplification, the reservoir grid orientation should be optimised so that the cross-flow terms are as small as possible when the effective permeability is calculated.

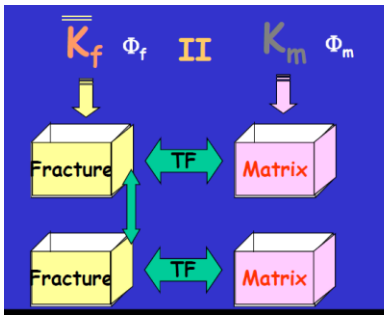

Figure 3: Transfers in the dual-porosity model

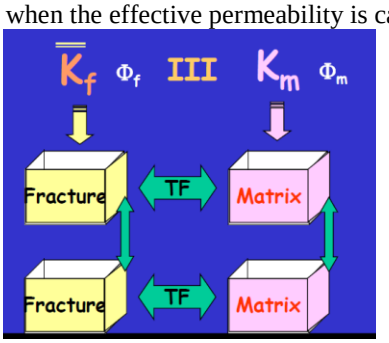

Figure 4: Transfers in the dual-porosity/dual-permeability model

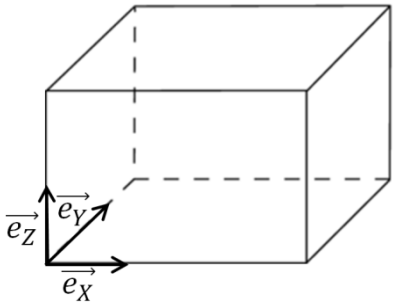

Figure 5: Reservoir grid cell with its associated local basis of three vectors: $(\vec{e_X}, \vec{e_Y}, \vec{e_Z})$

The Oda's method (Snow 1969 and Oda 1985) is the most used calculation technique for the computation of the effective permeability. It is based on an analytical formula which can be used indistinctly on a DFN or an implicit fracture network. This method enables to compute the effective permeability very quickly. Nevertheless it is one of the least accurate techniques. The main assumption behind this technique is that the fracture network has a full connectivity. This is rarely the case in practice and for this reason the Oda's method always over-estimates the effective permeability.

More sophisticated techniques calculating the effective permeability are based on a DFN simulation model (where the flow is explicitly simulated in the fractures but not in the matrix) (Lough et al. 1997). These techniques are called "numerical methods". The principle is to simulate a one-phase flow through a reservoir block by imposing the pressure on two faces. Then, the flow rate can be measured and, since the pressure difference is known, the effective permeability can be calculated using the Darcy's law (Bourbiaux et al. 1997).

Boundary conditions for the other faces must be chosen. One option is to assume they are impermeable: this is called the No Flow boundary condition (cf. Figure 6). A second option, the Linear Pressure boundary condition, assumes the pressure varies linearly between the two faces (cf. Figure 7). With this second option a flow can be observed perpendicularly to the pressure gradient. As a consequence, the cross-flow terms of the permeability matrix can be calculated (Sabathier et al. 1998).

The main advantage of numerical techniques compared with the Oda's method is that they take into account the fracture connectivity. The Linear Pressure boundary condition assumes a higher connectivity in grid block than the No Flow condition since it allows the fluid to flow in more fractures. For this reason the Linear Pressure condition gives generally higher effective permeabilities than the No Flow one.

The drawback of these calculation methods is that they are very slow when they are applied to a full reservoir (approximately 1 day of calculation is required for 120,000 cells with Petrel) (Cottereau et al. 2010; Ahmed-Elfeel and Geiger 2012).

A new method for the calculation of the effective permeability named "Image Based Periodic Object Simulation" (IBPOS) has been developed in the GoCad plugin named GoFraK (Gouth et al. 2006; Garcia et al. 2007; Cottereau et al. 2010). The particularity of this method is that it does not use a global DFN. At each location, and for a given observation scale a virtually infinite periodic network is built where a flow simulation can be performed (cf. Figure 8). The main advantage of this technique is that, since each cell is considered to be infinite, there is no need to choose boundary conditions that could overestimate or underestimate the fracture connectivity. In addition, the calculation time with this technique is much shorter than with other numerical methods (from 1mn to 15mn for 120,000 cells) (Cottereau et al. 2010).

The IBPOS method is only implemented for 2D DFNs, in order to compute the horizontal permeability. In GoFraK fractures are generated inside layers named "mechanical units" and each fracture is assumed to fully cross vertically at least one layer. The IBPOS method is applied for each mechanical unit to calculate the horizontal terms of the permeability matrix (k_X, k_Y and k_{XY}). The vertical inter-layer permeability k_Z (or transmissibility) is calculated with an analytical formulation (Cottereau et al. 2010).

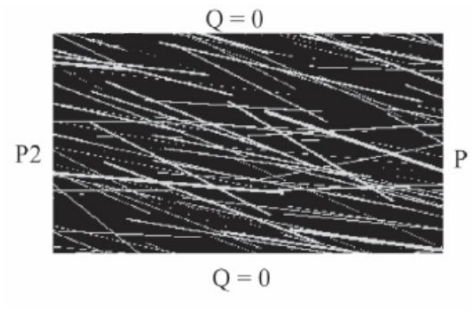

Figure 6: Numerical effective permeability calculation method with No Flow boundary condition (from Ding et al. 2005)

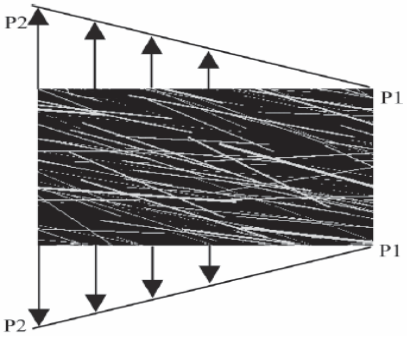

Figure 7: Numerical effective permeability calculation method with Linear Pressure boundary condition (from Ding et al. 2005)

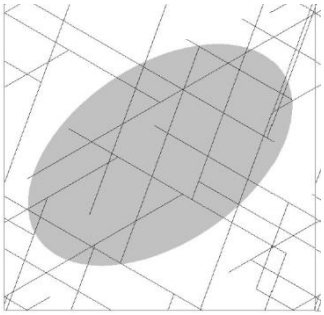

Figure 8: Example of a virtually infinite periodic DFN generated with GoFraK (modified from Garcia et al. 2007)

Cartesian fracture network

To assess the accuracy of the different effective permeability calculation techniques, two DFNs were simulated using simulation models based on effective properties and compared with an explicit simulation performed with Eclipse.

To represent explicit fractures in this software a 2 dimension grid containing large and smaller blocks was created (cf. Figure 9). The large blocks represent parts of the matrix whereas each smaller block can represent either a part of a fracture or of the matrix. The "fracture blocks" are assumed to have a porosity of 100% and a permeability significantly higher than the one of the matrix. All the "matrix blocks" (the large ones as well as the smaller ones) are assumed to have the same porosity and permeability (much lower than fractures) in the entire reservoir. Fractures can be generated in this grid by transforming small blocks representing the matrix into "fracture blocks".

Two DFNs were generated on this grid from implicit networks. A commercial DFN generator could not be used for this operation since this grid imposes constraints on the fracture network which are not usually met in other DFNs:

- The fracture orientation must follow the grid directions. Therefore, only "horizontal" fractures (which follow the x direction) and "vertical" ones (which follow the y direction) are allowed.
- The length of the fractures can only have discrete values since they necessarily start and end at the extremities of a small block. For this reason, a constant length was assumed for all the vertical and all the horizontal fractures.
- The fracture aperture is controlled by the small block width. To avoid calculation convergence problems an unrealistic aperture of 10 cm was assumed for all the fractures.
- The distance between fractures can only have discrete values since the size of the large blocks is constant. No commercial DFN generator can create a network which honours this constraint.

A program generating a DFN from an implicit fracture network respecting all these conditions has been written in the Python language. The non-spatial parameters (facture length, aperture, permeability and orientation) were assumed to be constant so probability density functions were not used.

The generated DFNs are in 2 dimensions. This approach can be compared with the way DFNs are generated in GoFraK. In this software, the reservoir is layered in mechanical units and each fracture is assumed to fully cross vertically one layer. The DFNs studied in this report could represent the fracture network of one reservoir mechanical unit. Only the horizontal permeability of the "layers" has been considered in this project.

Model characteristics

Two DFNs with, for each of them, two fracture sets called "the horizontal set" and "the vertical set" are presented in this paper.

The first DFN "the homogeneous model" was generated assuming a constant fracture length, aperture, permeability and Fracture Density (FD) for the "horizontal" and "vertical" fracture sets in the entire reservoir. Since the chosen fracture aperture (10 cm) is not realistic the permeability was not calculated from it using the "cubic law" but a constant permeability of 2.0 x10^5 mD (which corresponds to an aperture of 50 µm) was assumed (cf. Table 1 and Figure 10).

Homogeneous model		
	"Horizontal" fracture set	**"Vertical" fracture set**
Aperture	0.1 m	0.1 m
Permeability	2.00E+05 mD	2.00E+05 mD
Length	60 m	60 m
Width	10 m	10 m
Constant FD	0.094 m^2/m^3	0.094 m^2/m^3

Table 1: Fracture set properties of the homogeneous model

Heterogeneous model		
	"Horizontal" fracture set	**"Vertical" fracture set**
Aperture	0.1 m	0.1 m
Permeability	1.00E+05 mD	2.00E+05 mD
Length	40 m	80 m
Width	10 m	10 m
FD for facies 1	0.075 m2/m3	0.088 m2/m3
FD for facies 2	0.044 m2/m3	0.15 m2/m3

Table 2: Fracture set properties of the heterogeneous model

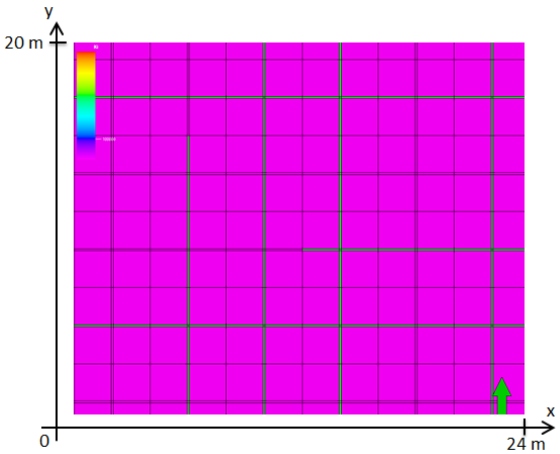

Figure 9: Part of the grid used to perform explicit simulations. The matrix is represented in pink and the fractures in green

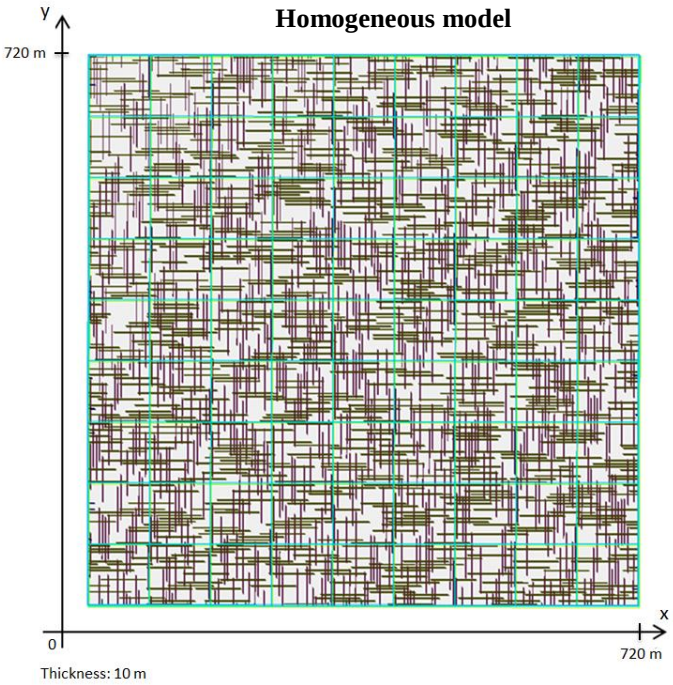

Figure 10: Discrete Fracture Network (DFN) of the Homogeneous model

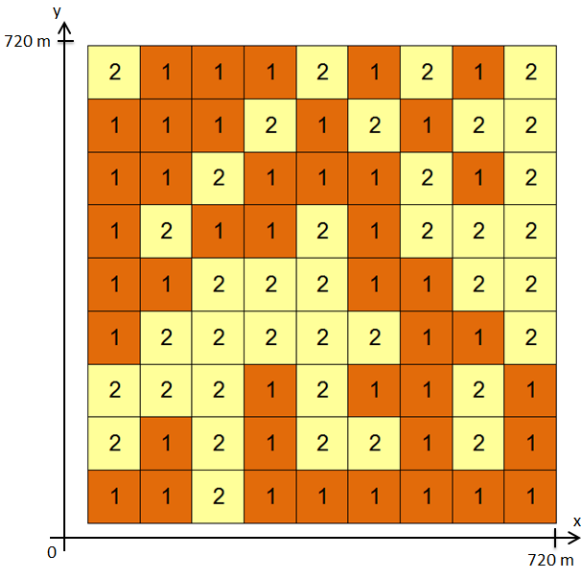

Figure 11: Facies map for the Heterogeneous model

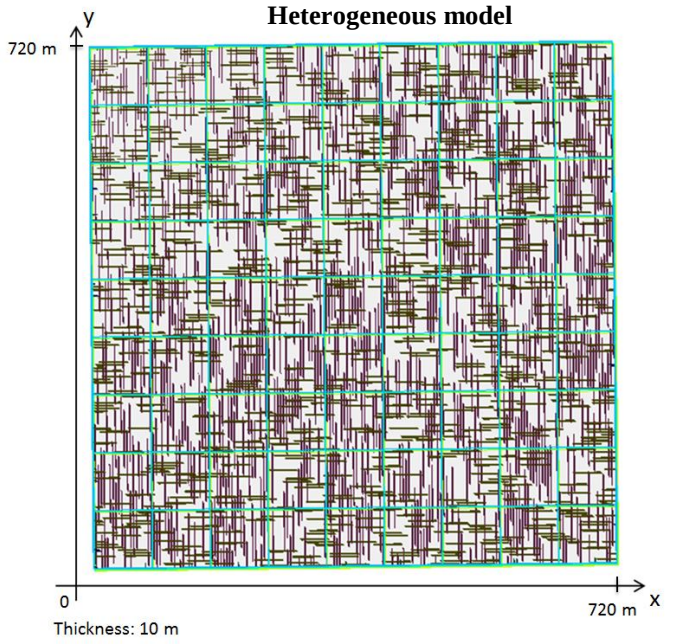

Figure 12: Discrete Fracture Network (DFN) of the Heterogeneous model

In the heterogeneous model the "vertical" fractures have a higher length, permeability and FD than the "horizontal" ones. In this model the fracture set FDs depend on the rock facies. The simulation domain is divided in 81 (9x9) cells and each of them is constituted by one rock type (or facies). Each facies is associated to one FD for each fracture set (cf. Table 2; Figure 11 and Figure 12).

Effective permeability

These two DFNs were imported into Petrel to calculate the effective permeability using the Oda's method, and the numerical methods with the Linear Pressure boundary condition and the No Flow boundary condition. Since GoFraK do not use a global DFN to calculate the effective permeability only the implicit models (the fracture set spatial and non-spatial parameters) were imported into this last software and local DFNs were generated in each cell (cf. Figure 13).

These four methods were used to calculate the k_X and the k_Y componants of the effective permeability matrix. The cross-flow terms and the vertical permeability were ignored.

The effective permeability was computed on two different grids: one containing 81 (9x9) cells and one containing 729 (27x27) cells to estimate the differences during oil production simulations. Previous papers tried to evaluate the effect of the grid size for the simulation of NFRs (Wang et al. 2008; Ahmed and Geiger 2011; Ahmed-Elfeel and Geiger 2012; Leung et al. 2012).

In the homogeneous model the effective permeability in the X direction (k_X) and the one in the Y direction (k_Y) are similar for each calculation method (cf. Table 3; Table 4; Table 5 and Table 6). In the heterogeneous model the permeability in the Y direction (k_Y) is always higher than the one in the X direction (cf. Table 7; Table 8; Table 9 and Table 10)

The effective permeabilities calculated with the Oda's method are systematically higher than the one obtained with numerical techniques. This is due to the fact that the Oda's method always over-estimates the connectivity of the fracture network. The effective permeabilities calculated with the numerical method using the No Flow boundary conditions were generally lower than the one calculated using the Linear Pressure condition since this first technique has a tendency to under-estimate the fracture connectivity. However, the lowest effective permeability is most of the time calculated with the IBPOS method.

The Oda, the Linear Pressure and the No Flow methods were used to calculate the effective permeability on a finer grid containing 729 (27x27) cells. The main effect of increasing the number of cells (i.e. reducing the cell dimensions) is that it increases the standard deviation of the effective permeability distribution in the reservoir. This phenomenon is typical of techniques applied on DFNs. If the size of the cell is too small, each block either contains a fracture and have a high effective permeability; or does not contain any fracture and has zero effective permeability. To avoid this problem several authors (Dershowitz et al. 1998; Bourbiaux 2010; Cottereau et al. 2010; Ahmed-Elfeel and Geiger 2012) have recommended to calculate the effective permeability at the scale of a Representative Elementary Volumes (REV) where the DFN seems to be homogeneous.

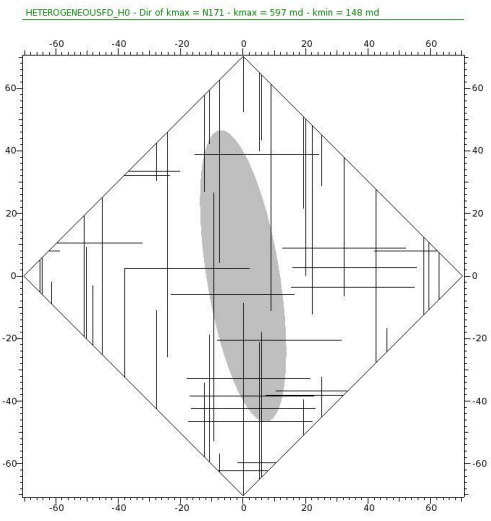

Figure 13: Local periodic DFN generated in one reservoir block of the Heterogeneous model with GoFraK. The ellipse represents the effective permeability tensor of this network.

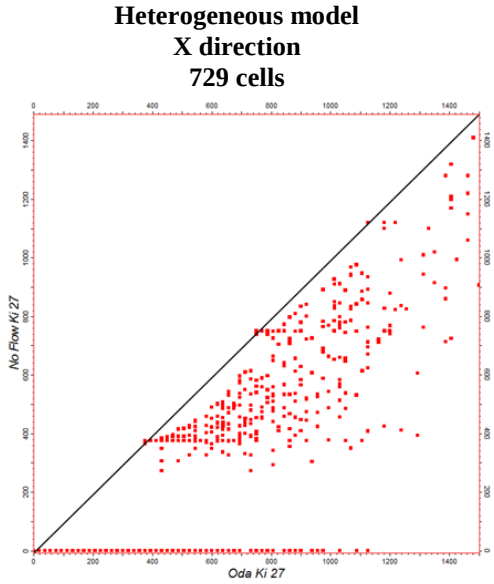

Figure 14: Comparison of the effective permeability in X direction calculated with the No Flow numerical method and the Oda's method for the Heterogeneous model in a grid containing 729 (27x27) cells (in mD)

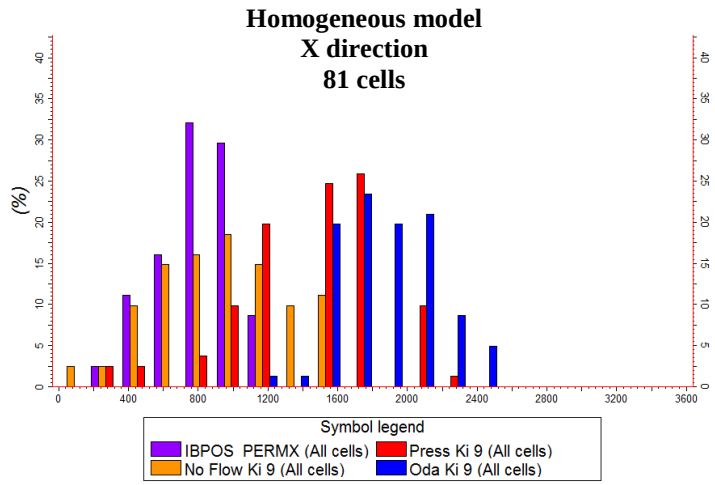

Figure 15: Effective permeability in X direction for the Homogeneous model on the grid containing 81 (9x9) cells calculated with the Oda's method (in blue), the Linear Pressure numerical method (in red), the No Flow numerical method (in orange) and the IBPOS method (in purple) (in mD)

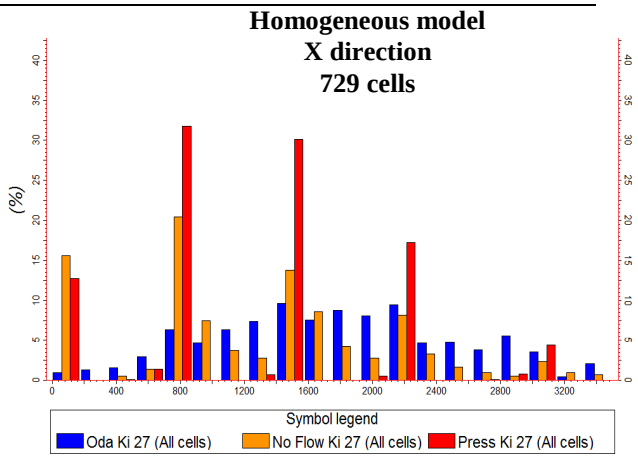

Figure 16: Effective permeability in X direction for the Homogeneous model on the grid containing 729 (27x27) cells calculated with the Oda's method (in blue), the Linear Pressure numerical method (in red) and the No Flow numerical method (in orange) (in mD)

Method	Mean	Standard deviation
Oda's	1,853 mD	268 mD
Linear Pressure	1,440 mD	408 mD
No Flow	934 mD	359 mD
IBPOS	800 mD	215 mD

Table 3: Effective permeability in X direction for the Homogeneous model on the grid containing 81 (9x9) cells

Method	Mean	Standard deviation
Oda's	1,853 mD	822 mD
Linear Pressure	1,283 mD	821 mD
No Flow	1,287 mD	851 mD

Table 4: Effective permeability in X direction for the Homogeneous model on the grid containing 729 (27x27) cells

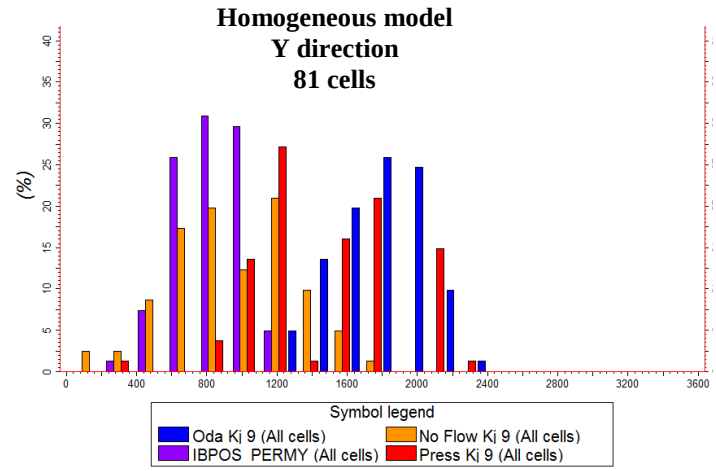

Figure 17: Effective permeability in Y direction for the Homogeneous model with the grid containing 81 (9x9) cells calculated with the Oda's method (in blue), the Linear Pressure numerical method (in red), the No Flow numerical method (in orange) and the IBPOS model (in purple) (in mD)

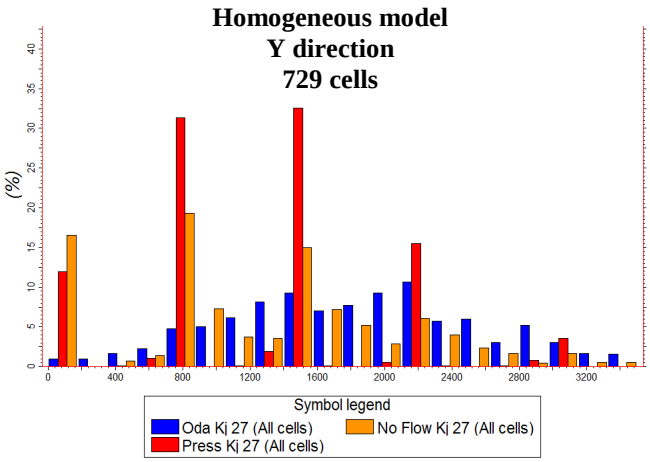

Figure 18: Effective permeability in Y direction for the Homogeneous model with the grid containing 729 (27x27) cells calculated with the Oda's method (in blue), the Linear Pressure numerical method (in red) and the No Flow numerical method (in orange) (in mD)

Method	Mean	Standard deviation
Oda's	1,863.58 mD	245 mD
Linear Pressure	1,458.28 mD	380 mD
No Flow	907.42 mD	352 mD
IBPOS	796.13 mD	190 mD

Table 5: Effective permeability in Y direction for the Homogeneous model on the grid containing 81 (9x9) cells

Method	Mean	Standard deviation
Oda's	1,864 mD	771 mD
Linear Pressure	1,260 mD	771 mD
No Flow	1,261 mD	829 mD

Table 6: Effective permeability in Y direction for the Homogeneous model on the grid containing 729 (27x27) cells

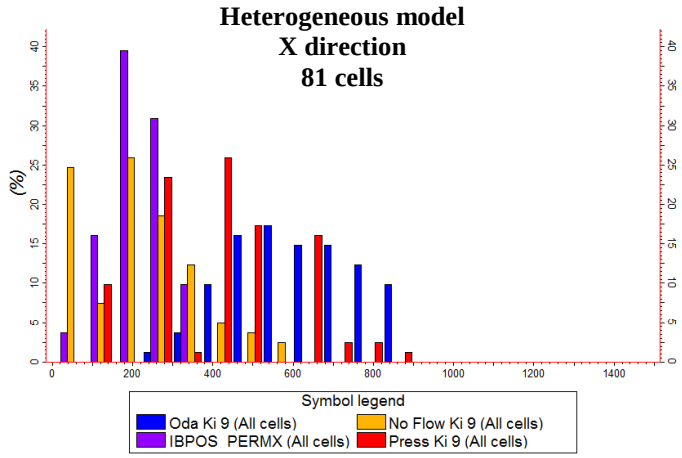

Figure 19: Effective permeability in X direction for the Heterogeneous model on the grid containing 81 (9x9) cells calculated with the Oda's method (in blue), the Linear Pressure numerical method (in red), the No Flow numerical method (in orange) and the IBPOS method (in purple) (in mD)

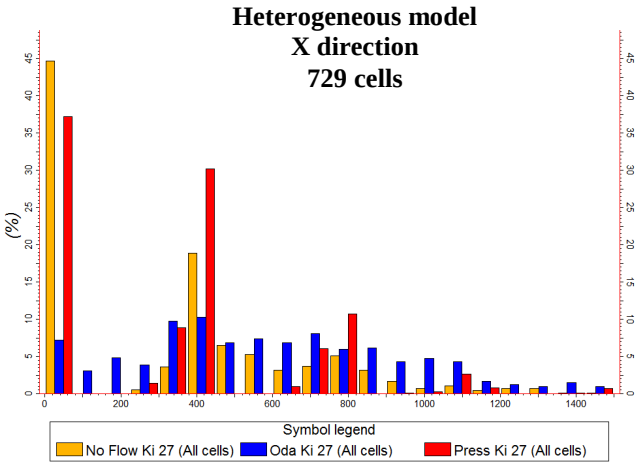

Figure 20: Effective permeability in X direction for the Heterogeneous model on the grid containing 729 (27x27) cells calculated with the Oda's method (in blue), the Linear Pressure numerical method (in red) and the No Flow numerical method (in orange) (in mD)

Method	Mean	Standard deviation
Oda's	611 mD	147 mD
Linear Pressure	410 mD	180 mD
No Flow	201 mD	149 mD
IBPOS	207 mD	70 mD

Table 7: Effective permeability in X direction for the Heterogeneous model on the grid containing 81 (9x9) cells

Method	Mean	Standard deviation
Oda's	611 mD	367 mD
Linear Pressure	335 mD	325 mD
No Flow	328 mD	350 mD

Table 8: Effective permeability in X direction for the Heterogeneous model on the grid containing 729 (27x27) cells

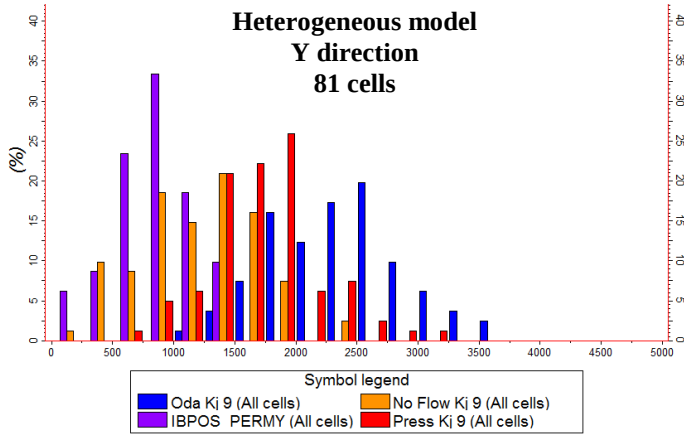

Figure 21: Effective permeability in Y direction for the Heterogeneous model with the grid containing 81 (9x9) cells calculated with the Oda's method (in blue), the Linear Pressure numerical method (in red), the No Flow numerical method (in orange) and the IBPOS method (in purple) (in mD)

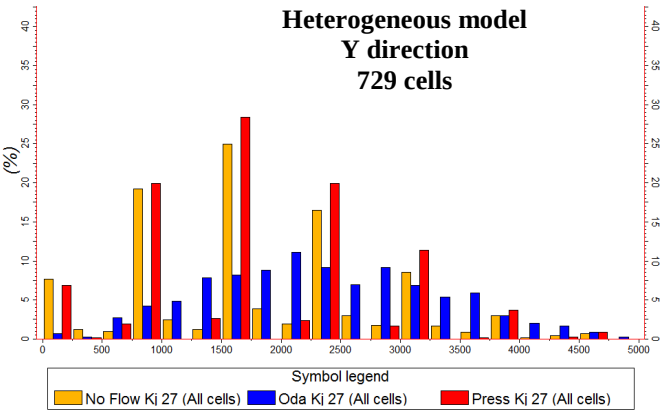

Figure 22: Effective permeability in Y direction for the Heterogeneous model with the grid containing 729 (27x27) cells calculated with the Oda's method (in blue), the Linear Pressure numerical method (in red) and the No Flow numerical method (in orange) (in mD)

Method	Mean	Standard deviation
Oda's	2,360 mD	533 mD
Linear Pressure	1,583 mD	450 mD
No Flow	1,161 mD	484 mD
IBPOS	823 mD	327 mD

Table 9: Effective permeability in Y direction for the Heterogeneous model on the grid containing 81 (9x9) cells

Method	Mean	Standard deviation
Oda's	2,360 mD	959 mD
Linear Pressure	1,711 mD	972 mD
No Flow	1,723 mD	996 mD

Table 10: Effective permeability in Y direction for the Heterogeneous model on the grid containing 729 (27x27) cells

When the grid cell size changes the average effective permeability in the reservoir calculated with the Oda's method remains the same. However, the average effective permeability calculated with the No Flow boundary condition increases and the one calculated with the Linear Pressure boundary condition decreases. This can be explained by the fact that when the size of the cells decreases the difference between these two boundary conditions has less importance. Indeed, when a reservoir cell is very small it generally contains no more than one fracture. In that case, either the fracture crosses the cell and the effective permeability calculated with the numerical methods is the same as the one calculated with the Oda's technique; either the cell does not cross the cell and no connectivity is detected so the cell effective permeability is zero. This behaviour can be seen in the comparison between the Oda's method and the No Flow method for the Heterogeneous model (Figure 14). The effective permeabilities calculated with the Oda, the No Flow and the Linear Pressure methods are relatively similar with the exception that many cells have zero permeability with the No Flow and Linear Pressure methods.

Previous papers (e.g. Delorme et al. 2008) have studied the conditions under which the Linear Pressure and the No Flow methods give the same effective permeability as the Oda's method. It has been proposed to evaluate the DFN connectivity in each grid cell to know if the Oda's method can be used to evaluate the effective permeability or if a numerical method is required. This approach is implemented in the FracMan software from Golder Associates under the name of "Oda Gold" method. More complex techniques trying to take into account the connectivity of the fracture network have also been proposed (e.g. Leung and Zimmerman 2012).

Reservoir production simulation

The homogeneous and the heterogeneous fracture networks were simulated using an explicit model (where all the fractures were represented) and a model based on effective properties. To ignore problems due to flow transfer between the matrix and the fractures, the matrix permeability was assumed to be zero (the reservoir is considered to be of type I in the Nelson's classification). As a consequence, the explicit model and "effective medium" model used for the simulation do not represent the matrix. The explicit simulation model used is called the "DFN model" and the model based on effective properties is the "single-medium model".

The two DFN models (the homogeneous one and the heterogeneous one) were simulated four times with the single-medium model each time using an effective permeability calculated with a different method (the Oda's method, the Linear Pressure numerical method, the No Flow numerical method and the IBPOS method).

In the chosen production scenario one production well is located at the centre of the reservoir and four injections wells are positioned near the edges (cf. Figure 23). The injection well locations were not chosen because they maximise the recovery factor by delaying the water breakthrough as it is normally done for real NFRs but because they enable the oil and the water to flow in the same direction as the grid.

A "well zone" with a homogeneous permeability of 2,700 mD and a homogeneous porosity of 2% has been added around each well to increase their connectivity with the fracture network. This zone could represent the result of an operation such as fracturing performed which increases the well connectivity with the fractures. In the simulations this zone is represented as a square with a side length of 80 m (cf. Figure 23).

The choice of the relative permeability function for NFRs is a complex issue. Theoretically the relative permeability used for the single-medium model should be a pseudo relative permeability which is different from the relative permeability used in the explicit simulation (Matthäi and Nick 2009; Ahmed-Elfeel et al. 2010). In this paper a simple linear permeability (cf. Figure 24) was used both in the explicit simulation and the single-medium model.

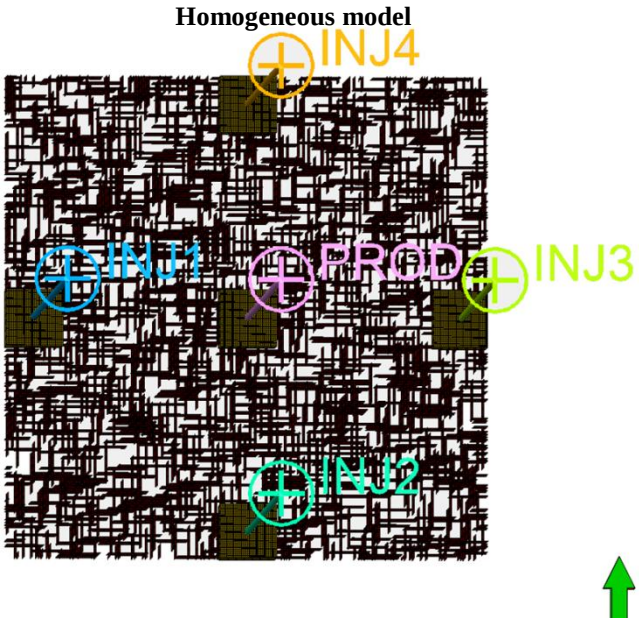

Figure 23: Well and well zone locations on the Homogeneous model (the locations are the sane on the Heterogeneous model)

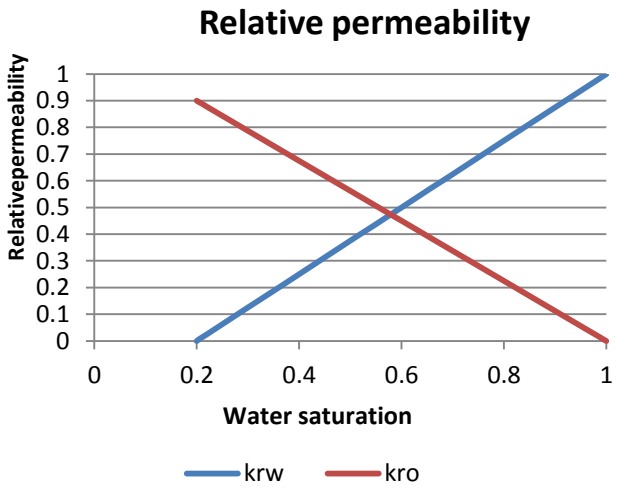

Figure 24: Relative permeability of water (in blue) and oil (in red) against water saturation

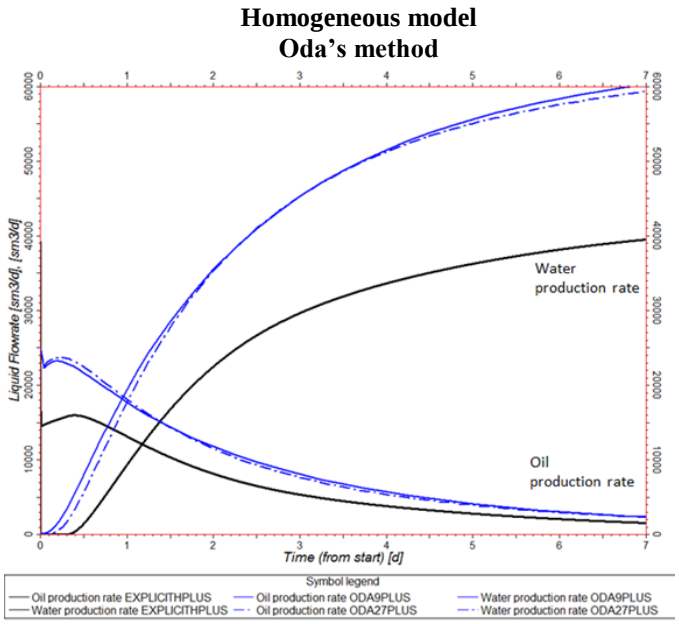

Figure 25: Oil and water production rate on the Homogeneous model calculated with the Explicit simulation (black line) and simulations using effective permeabilities calculated with the Oda's method on a grid with 81 (9x9) cells (solid blue line) and 729 (27x27) cells (dashed blue line)

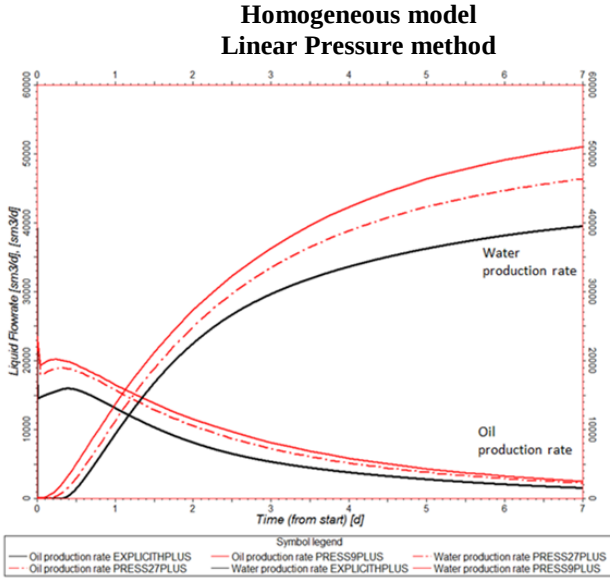

Figure 26: Oil and water production rate on the Homogeneous model calculated with the Explicit simulation (black line) and simulations using effective permeabilities calculated with the Linear Pressure method on a grid with 81 (9x9) cells (solid red line) and 729 (27x27) cells (dashed red line)

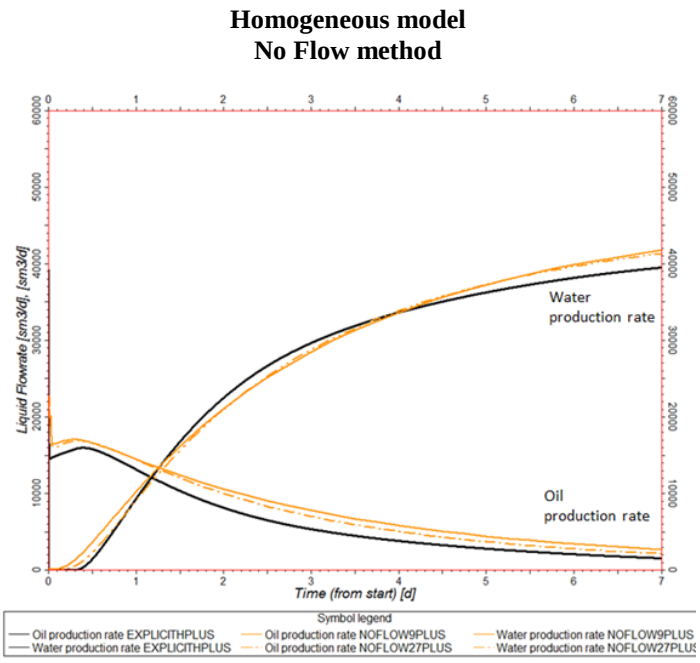

Figure 27: Oil and water production rate on the Homogeneous model calculated with the Explicit simulation (black line) and simulations using effective permeabilities calculated with the No Flow method on a grid with 81 (9x9) cells (solid orange line) and 729 (27x27) cells (dashed orange line)

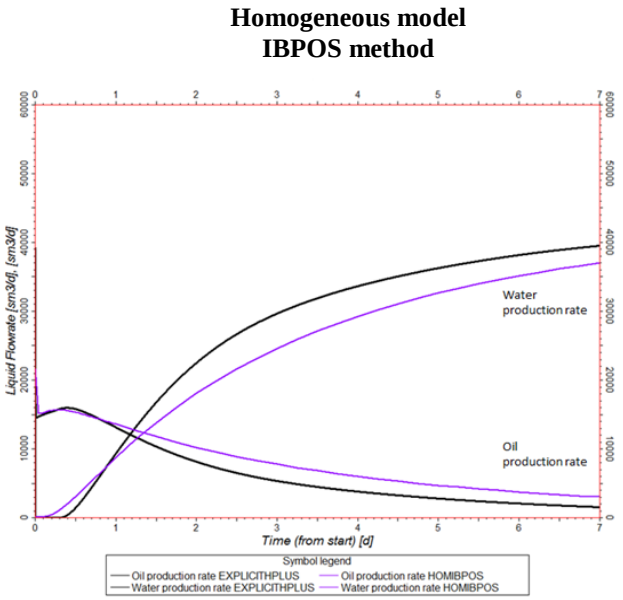

Figure 28: Oil and water production rate on the Homogeneous model calculated with the Explicit simulation (black line) and simulations using effective permeabilities calculated with the IBPOS method on a grid with 81 (9x9) cells (solid purple line)

Heterogeneous model
Oda's method

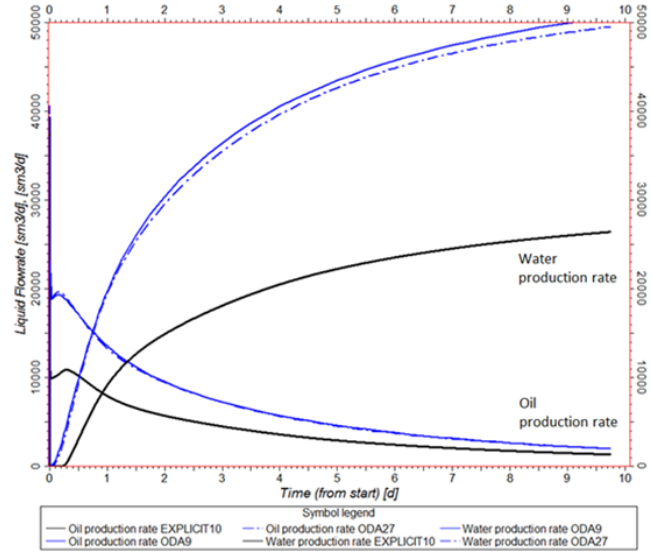

Figure 29: Oil and water production rate on the Heterogeneous model calculated with the Explicit simulation (black line) and simulations using effective permeabilities calculated with the Oda's method on a grid with 81 (9x9) cells (solid blue line) and 729 (27x27) cells (dashed blue line)

Heterogeneous model
Linear Pressure method

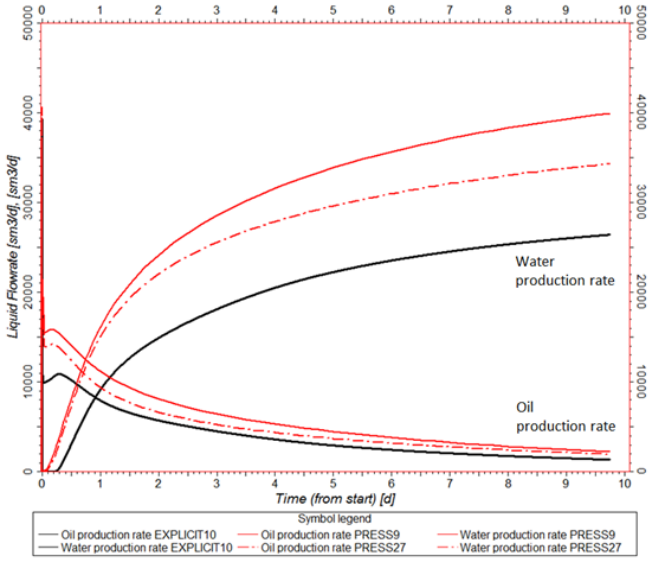

Figure 30: Oil and water production rate on the Heterogeneous model calculated with the Explicit simulation (black line) and simulations using effective permeabilities calculated with the Linear Pressure method on a grid with 81 (9x9) cells (solid red line) and 729 (27x27) cells (dashed red line)

Heterogeneous model
No Flow method

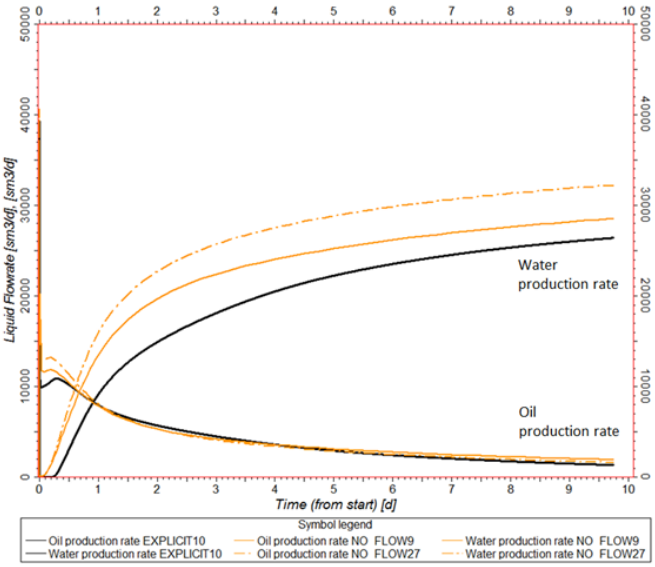

Figure 31: Oil and water production rate on the Heterogeneous model calculated with the Explicit simulation (black line) and simulations using effective permeabilities calculated with the No Flow method on a grid with 81 (9x9) cells (solid orange line) and 729 (27x27) cells (dashed orange line)

Heterogeneous model
IBPOS method

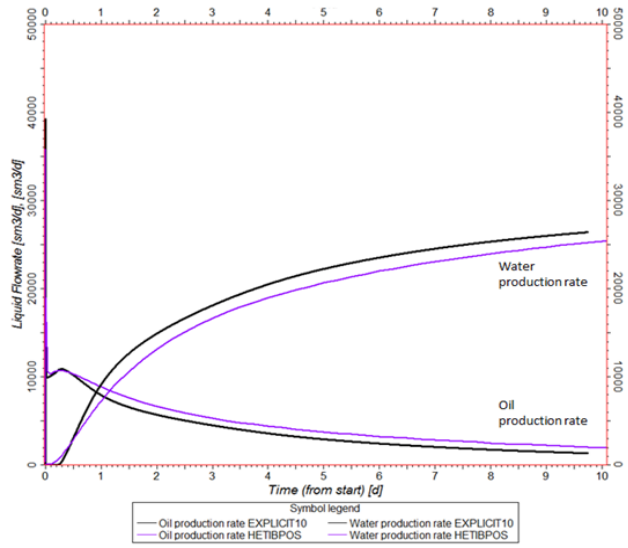

Figure 32: Oil and water production rate on the Heterogeneous model calculated with the Explicit simulation (black line) and simulations using effective permeabilities calculated with the IBPOS method on a grid with 81 (9x9) cells (solid purple line)

At the initial stage the reservoir has a homogeneous water saturation of 20%. The water is injected in the reservoir at the constant pressure of 250 bars and the production well has a constant bottom hole pressure of 1 bar. The compressibility effect of oil, water and rocks is negligible and therefore the volume of water injected is equal to the volume of fluid produced.

Since the matrix is not represented producing the oil from the fractures is very rapid. The homogeneous model is produced during 7 days while a simulation of 10 days is performed for the Heterogeneous model.

The highest oil and water rate where observed for the simulations using the permeabilities calculated with the Oda's method. The simulations using permeability values calculated with the No Flow and the IBPOS methods produced the lowest oil and water rate.

The calculation techniques giving the best match with the explicit simulation are the No Flow and the IBPOS methods. This can be explained by the fact that the considered DFNs have a very low connectivity.

Being able to predict accurately the oil production flow rate does not necessarily mean the water production rate (and hence the water injection rate) are accurately forecast. This can be due to a bad estimation of the reservoir sweep efficiency.

The oil and water production rates calculated with the Oda's method on the two different grid sizes are very similar (cf. Figure 25 and Figure 29). This is coherent with the fact that the average effective permeability calculated with the Oda's method remains the same for any grid size.

The simulations performed on the fine scale grid (the one containing 729 (27x27) cells) with the models using the No Flow and the Linear Pressure calculation techniques give very similar oil and water rates. This is consistent with the fact that their effective permeabilities are very similar on this grid.

Nevertheless, refining the grid does not necessarily increase the accuracy of the simulation since it can lead to an over-estimation of the network connectivity (e.g. Figure 31).

FRACS2000 model

The FRACS2000 model represents a section of the San Andreas Formation in an onshore field in California (Ahmed-Elfeel and Geiger 2012). It has been used in many publications (Matthäi et al. 2005; Geiger et al. 2009; Matthäi and Nick 2009; Ahmed-Elfeel and Geiger 2012).

A single-phase flow was simulated through this model between two faces in the direction of X (cf. Figure 33). This simulation enables to compute the effective permeability in the direction of X similarly to the No Flow calculation method. The main difference with this technique is that the DFM simulation model (which takes into account the matrix) is used.

With the DFM model the matrix permeability cannot be zero. A permeability of 1.01 10^{-2} mD is used in order to represent a reservoir of type I or II in the Nelson's classification.

This DFM simulation produces an effective permeability (Table 12), and can be compared with simulations using effective properties, computed by Petrel where the same FRACS2000 was imported. The flow rate varies during the simulation due to the rock and fluid compressibility (cf. Figure 34). Only the final steady flow rates are compared to each other. The single-medium model has been chosen because using a more complex model had little impact on the final liquid flow rate.

The effective permeabilities were calculated on 3 different grids: One containing 5x5x4 cells (Figure 35; one containing 10x10x4 cells (Figure 36) and one containing 20x20x4 cells (Figure 37). Only techniques usable with DFNs (the Oda's method, the Linear Pressure and No Flow numerical methods) were applied. In each cell the effective permeability in the direction of X, Y and Z must be calculated. When the cell does not contain any fracture its effective permeability is 0 mD.

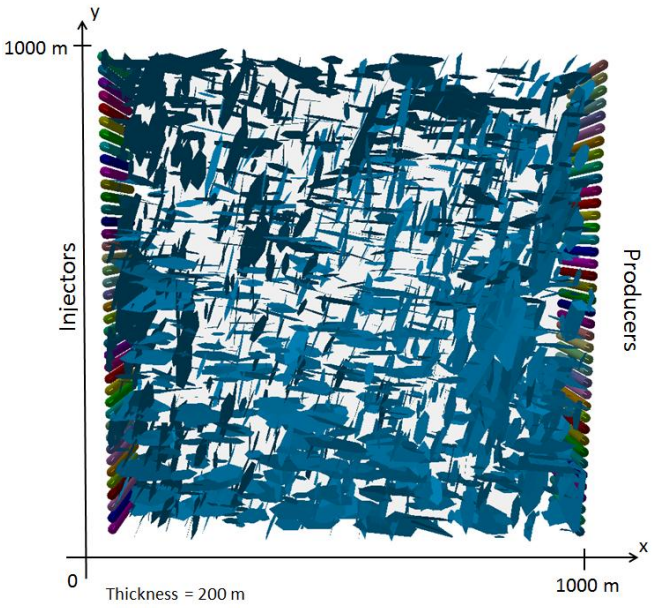

Figure 33: Well location on the FRACS2000 model

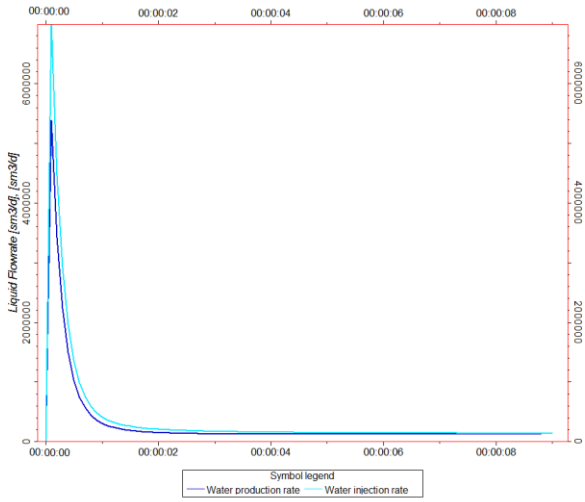

Figure 34: Fluid injection and production flow rate

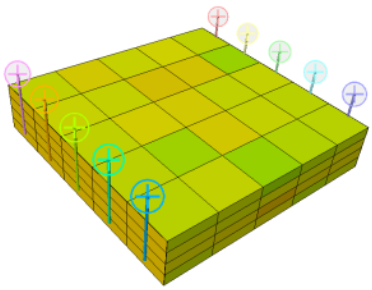

Figure 35: Production scenario for the simulation using effective properties with a grid containing 5x5x4 cells

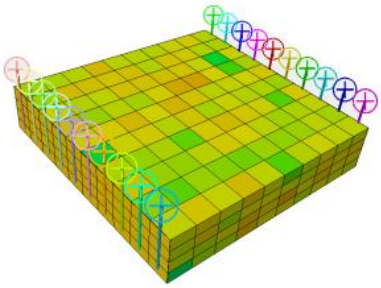

Figure 36: Production scenario for the simulation using effective properties with a grid containing 10x10x4 cells

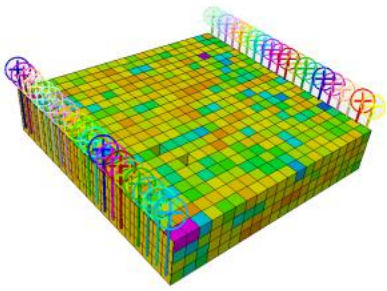

Figure 37: Production scenario for the simulation using effective properties with a grid containing 20x20x4 cells

The Oda's method gave the highest permeability results while the No Flow numerical method gave the lowest one. When the grid cell size is decreased the average of the effective permeability shows little variation but its standard deviation increases dramatically.

No flow rate has been observed on the simulations using effective permeabilities calculated with the No Flow method. This is due to a too high number of cells having a zero effective permeability. Flow is possible with the DFM model because the fluid travels through the matrix to go from one fracture to another one. With the dual-permeability model, although the fluid can flow through the matrix, it cannot do short distances between fractures and therefore the permeability is under-estimated.

Model properties	
Pressure gradient	$9.809\ 10^{-2}$ bar m^{-1}
Section area	$200\ 10^{3}$ m^{2}
Viscosity	1.6 cP
Matrix permeability	$1.01\ 10^{-2}$ mD
Fracture permeability	$8.33\ 10^{7}$ mD
Distance between the inlet and the outlet	1000 m

Table 11: FRACS2000 model properties

DFM Results	
Flow rate	**Effective permeability**
0.289 m^3/s	**239 mD**

Table 12: DFM simulation results

Grid with 5x5x4 cells		
	Flow rate	**Eff perm**
Oda	1.589 m^3/s	1,315 mD
Linear Press	0.680 m^3/s	562 mD
No Flow	0 m^3/s	0 mD

Table 13: Simulation results on the grid with 5x5x4 cells

Grid with 10x10x4 cells		
	Flow rate	**Eff perm**
Oda	1.505 m^3/s	1,245 mD
Linear Press	0.584 m^3/s	483 mD
No Flow	0 m^3/s	0 mD

Table 14: Simulation results on the grid with 10x10x4 cells

Grid with 20x20x4 cells		
	Flow rate	**Eff perm**
Oda	1.236 m^3/s	1,023 mD
Linear Press	0.491 m^3/s	406 mD
No Flow	0 m^3/s	0 mD

Table 15: Simulation results on the grid with 20x20x4 cells

The effective permeabilities presented in the Table 12, Table 13, Table 14 and Table 15 are related to the entire model in the direction of X. They are deduced from the flow rate knowing the pressure difference, the viscosity, the section area, and the distance between the inlet and the outlet. This method of calculating the effective permeability is the same as the No Flow method.

Conclusion

The accuracy of effective permeability calculation techniques have been studied by performing reservoir production simulations. Dynamic simulations realised with the single-medium model (based on effective properties) have been compared to a DFN model in a first part and to a DFM model in a second part. The main information that can be learnt from these experiments is that the calculation methods are most of the time over-estimating the flow rates. This is due to a bad estimation of the fracture network connectivity.

The Oda's method is the most used calculation technique because it is fast and can be applied to any type of fracture network. However, it is also the least accurate technique so it should always be used with caution. The other calculation methods are much more accurate. Nevertheless, the Linear Pressure and the No Flow numerical methods are very slow and

cannot be realistically used on a full reservoir. The IBPOS method is quick and has been used for real reservoirs nevertheless only bed-confined subvertical fracture networks can be simulated.

In the DFN simulation model a fracture is either connected to another one or unconnected while in the DFM model fractures very close from each other can have a partial connection. Therefore numerical upscaling techniques can sometimes underestimate the effective permeability.

The fracture connectivity is never fully determined by the fracture set spatial and non-spatial parameters. Several realisations of the same implicit network give different effective permeabilities. As a consequence, using a calculation method which takes perfectly into account the network connectivity is not necessarily relevant if this connectivity is only controlled by random functions.

This calls for further research in the parameters that can indicate the connectivity degree of fracture networks, DFN generators respecting an input connectivity degree, and "quick" effective permeability calculation methods taking into account the connectivity. This goal can only be achieved by calibration to dynamic data (e.g. well tests), interference tests, and history production data.

Nomenclature

a	Fracture aperture
DFN	Discrete Fracture Network
DFM	Discrete Fractures and Matrix
FD	Fracture Density
IBPOS	Image Based Periodic Object Simulation
$\bar{\bar{k}}$	Effective permeability tensor
k_f	Fracture permeability
NFR	Naturally Fractured Reservoir
q	Darcy velocity
REV	Representative Elementary Volume
σ	Shape factor
μ	Fluid viscosity

References

Ahmed-Elfeel, M., Couples, G., Geiger, S. and Ma, J. (2010). "Upscaled Multi-Phase Flow Properties of Fracture Corridors." SPE 139463. Presented at the SPE Caspian Carbonates Technology Conference held in Atyrau, Kazakhstan, 8–10 November 2010.

Ahmed-Elfeel, M. and Geiger, S. (2012). "Static and Dynamic Assessment of DFN Permeability Upscaling." SPE 154369. Presented at the EAGE Annual Conference & Exhibition incorporating SPE Europec held in Copenhagen, Denmark, 4–7 June 2012.

Ahmed, M. and Geiger, S. (2011). "DFN Permeability Upscaling what works and what doesn't." Presentation at the ITF-ISF phase III meeting Paris, 8 November 2011.

Astratti, D., Souche, L., Faskhoodi, M.M. and Menegatti, P. (2010). "Seismic to Simulation Fracture Characterization of a Green Carbonate Reservoir in Presence of Large Uncertainties." SPE 136829. Presented at the Abu Dhabi International Petroleum Exhibition & Conference held in Abu Dhabi, UAE, 1–4 November 2010.

Barenblatt, G., Zheltov, Iu. and Kochina, I. (1960). "Basic concepts in the theory of seepage of homogeneous liquids in fissured rocks (strata)." PMM Vol. 24, No. 5: pp. 852-864.

Belayneh, M., Geiger, S. and Matthäi, S. (2006). "Numerical simulation of water injection into layered fractured carbonate reservoir analogs." AAPG Bulletin, v. 90, no. 10 (October 2006): pp. 1473-1493.

Belayneh, M., Matthäi, S., Blunt, M. and Rogers, S. (2009). "Comparison of deterministic with stochastic fracture models in water-flooding numerical simulations." AAPG Bulletin, v. 93, no. 11 (November 2009): pp. 1633-1648.

Bourbiaux, B., Basquet, R., Cacas, M. C., Daniel, J. M. and Sarda, S. (2002). "An integrated workflow to account for multi-scale fractures in reservoir simulation models: implementation and benefits." SPE 78489. Presented at the 10th Abu Dhabi International Petroleum Exhibition and Conference, 13-16 October 2002.

Bourbiaux, B. (2010). "Fractured Reservoir Simulation: a Challenging and Rewarding Issue." Oil & Gas Science and Technology – Rev. IFP, Vol. 65 (2010), No. 2: pp. 227-238.

Bourbiaux, B.J., Sarda, M.C. and Sabathier, J.C. (1997). "A fast and efficient methodology to convert fractured reservoir images into a dual-porosity model." SPE 38907. Presented at the 1997 SPE Annual Technical Conference and Exhibition held in San Antonio, Texas, 5-8 October 1997.

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

Cottereau, N., Garcia, M., Gosselin, O. and Vigier, L. (2010). "Effective Fracture Network Permeability: Comparative Study of Calculation Methods." SPE 131126. Presented at the SPE EUROPEC/EAGE Annual Conference and Exhibition held in Barcelona, Spain, 14–17 June 2010.

Delorme, M., Atfeh, B., Allken, V. and Bourbiaux, B. (2008). "Upscaling Improvement for Heterogeneous Fractured Reservoir Using a Geostatistical Connectivity Index." Geostats 2008, Santiago, Chile.

Dershowitz, B., LaPointe, P., Eiben, T. and Wei, L. (1998). "Integration of Discrete Feature Network Methods with Conventional Simulator Approaches." SPE 62498. SPE Reservoir Eval. & Eng. Vol. 3, No. 2, April 2000: pp 165-170. SPE 49069 presented at the 1998 SPE Annual Technical Conference and Exhibition, New Orleans, 27-30 September.

Ding, Y., Basquet, R. and Bourbiaux, B. (2005). "Upscaling Fracture Networks for Simulation of Horizontal Wells using a Dual-Porosity Reservoir Simulator." SPE 92774. Presented at the 2005 SPE Reservoir Simulation Symposium held in Houston, Texas, U.S.A, 31 January - 2 February 2005.

Garcia, M., Gouth, F. and Gosselin, O. (2007). "Fast and efficient modeling and conditioning of naturally fractured reservoir models using static and dynamic data." SPE 107525. Presented at the SPE Europec/EAGE Annual Conference and Exhibition held in London, United Kingdom, 11–14 June 2007.

Gauthier, B., Auzias, V., Garcia, M. and Chiapello, E. (2002). "Static and Dynamic Characterization of Fracture Pattern in the Upper Jurassic Reservoirs of an Offshore Abu Dhabi Field: From Well Data to Full Field Modeling." SPE 78498. Presented at the 10th Abu Dhabi International Petroleum Exhibition and Conference, 13-16 October 2002.

Geiger, S., Matthäi, S., Niessner, J. and Helmig, R. (2007). "Black-Oil Simulations for Three-Component, Three-Phase Flow in Fractured Porous Media." SPE 107485. SPE Journal, June 2009: pp 338-354. Presented at the EUROPEC/EAGE Conference and Exhibition, London, 11-14 June 2007.

Geiger, S., Huangfu, Q., Reid, F., Matthäi, S., Coumou, D., Belayneh, M., Fricke, C. and Schnid, K. (2009). "Massively Parallel Sector Scale Discrete Fracture and Matrix Simulations." SPE 118924. Presented at the 2009 SPE Reservoir Simulation Symposium held in The Woodlands, Texas, USA, 2-4 February 2009.

Gouth, F., Toublanc, A. and Mresah, M. (2006). "Characterisation and modelling of a fractured reservoir using a novel DFN approach." SPE 102165. Presented at the 2006 Abu Dhabi International Petroleum Exhibition and Conference held in Abu Dhabi, U.A.E., 5–8 November 2006.

Kazemi, H., Merrill, L. S., Porterfield, K. L. and Zeman, P. R. (1976). "Numerical Simulation of Water-Oil Flow in Naturally Fractured Reservoirs." SPE 5719. Presented at the SPE-AIME 4th Symposium on Numerical Simulation of Reservoir Performance, held in los Angeles, Feb. 19-20, 1976.

Kfoury, M. (2004). "Changement d'échelle séquentiel pour les milieux fracturés hétérogènes." PhD thesis from the Institut National Polytechnique de Toulouse, France.

Lange, A., Basquet, R. and Bourbiaux, B. (2004). "Hydraulic Characterization of Faults and Fractures Using a Dual Medium Discrete Fracture Network Simulator." SPE 88675. Presented at the 11th Abu Dhabi International Petroleum Exhibition and Conference held in Abu Dhabi, UAE, 10-13 October 2004.

Leung, C., Hoch, A. and Zimmerman, R. (2012). "Comparison of discrete fracture network and equivalent continuum simulations of fluid flow through two-dimensional fracture networks for the DECOVALEX-2011 project." Mineralogical Magazine, special issue on Geological Disposal of Radioactive Waste (2012).

Leung, C. and Zimmerman, R. (2012). "Estimating the Hydraulic Conductivity of Two-Dimensional Fracture Networks Using Network Geometric Properties." Transp Porous Med (2012) Vol. 93: pp. 777–797.

Long, J., Remer, J., Wilson, C. and Witherspoon, P. (1982). "Porous Media Equivalent for Networks of Discontinuous Fractures." Water Resources Research Vol. 18, No. 3, June 1969: pp. 645-658.

Lough, M., Lee, S. and Kamath, J. (1997). "A New Method to Calculate Effective Permeability of Gridblocks Used in the Simulation of Naturally Fractured Reservoirs." SPE 36730. Presented at the 1996 SPE Annual Technical Conference and Exhibition held in Denver, Colorado, 6-9 October.

Macé, L. (2006). "Caractérisation et modélisation numériques tridimensionnelles des réseaux de fractures naturelles." PhD thesis from the Institut National Polytechnique de Lorraine, Vandoeuvre-lès-Nancy, France.

Matthäi, S., Mezentsev, A. and Belayneh, M. (2005). "Finite Element-Node-Centered Finite-Volume Two-Phase-Flow Experiments with Fractured Rock Represented by Unstructured Hybrid-Element Meshes." SPE 93341. SPE Reservoir Evaluation & Engineering, December 2007: pp 740-756. Presented at the SPE Reservoir Simulation Symposium, The Woodlands, Texas, 31 January–2 February 2005.

Matthäi, S. (2007). "The State-of-the-Art in Upscaling of Two Phase Flow in Fractured Rock." Report prepared for United Kingdom Nirex Limited, March 31, 2007.

Matthäi, S., Geiger, S., Roberts, S., Paluszny, A., Belayneh, M., Burri, A., Mezentsev, A., Lu, H., Coumou, D., Driesner, T. and Heinrich, C. (2007). "Numerical simulation of multi-phase fluid flow in structurally complex reservoirs." Geological Society, London, Special Publications 2007, v.292: pp 405-429.

Matthäi, S. and Nick, H. (2009). "Upscaling two-phase flow in naturally fractured reservoirs." AAPG Bulletin, v. 93, no. 11 (November 2009): pp. 1621-1632.

Nelson, R. A. (2001). Geologic Analysis of Naturally Fractured Reservoirs second edition, Gulf Professional Publishing.

Oda, M. (1985). "Permeability tensor for discontinuous rock masses." Géotechnique 35, No. 4: pp 483-495.

Ozkaya, S. (2011). "A Simple Formula to Estimate 2d Fracture Connectivity." SPE 153143. SPE Reservoir Evaluation & Engineering December 2011: pp. 763-775.

Sabathier, J., Bourbiaux, B., Cacas, M. C. and Sarda, S. (1998). "A New Approach of Fractured Reservoirs." SPE 39825. Presented at the 1998 SPE International Petroleum Conference and Exhibition of Mexico held in Villahermosa, Mexico, 3-5 March 1998.

Snow, D. (1969). "Anisotropic permeability of Fractured Media." Water Resources Research Vol. 5, No. 6, December 1969: pp 1273-1289.

Souche, L., Kherroubi, J., Rotschi, M. and Quental, S. (2009). "A Dual Representation for Multiscale Fracture Characterization and Modeling." Presentation at the AAPG Annual Convention and Exhibition, Denver, Colorado, USA, June 7-10, 2009.

Vigier, L. (2009). "Effective fracture network permeability tensor calculation." IFP/Total Master degree internship report.

Wang, H., Forster, C. and Deo, M. (2008). "Simulating Naturally Fractured Reservoirs: Comparing Discrete Fracture Network Models to the Upscaled Equivalents." Presentation at the AAPG Annual Convention, San Antonio, Texas, April 20-23, 2008.

Warren, J. E. and Root, P. J. (1962). "The Behavior of Naturally Fractured Reservoirs." SPE 426. SPE Journal, September 1963: pp 245-255. Presented at the Fall Meeting of the Society of Petroleum Engineers in Los Angeles on Oct. 7-10, 1962.

Wayne, N., Schechter, D. and Thompson, L. (2006). Naturally fractured reservoir characterization, Richardson, TX : Society of Petroleum Engineers, c2006.

APPENDIX A: Critical literature review

Paper references	Year	Title	Authors	Contribution
PMM Vol. 24, No. 5 (1960) pp. 852-864	1960	Basic concepts in the theory of seepage of homogeneous liquids in fissured rocks (strata)	G. Barenblatt, Iu. Zheltov, I. Kochina	First to propose the use of the dual-porosity/dual-permeability model to simulate NFR. First to present the concept of "effective property".
SPE 426	1963	The behavior of naturally fractured reservoirs	J.E. Warren, P.J. Root	First to propose the use of a dual porosity model for the simulation of fracture reservoirs First to use the so-called "Sugar Box Model"
Water Resources Research Vol. 5, No. 6, pp 1273-1289	1969	Anisotropic permeability of fractured media	D.T. Snow	Fist to propose the so-called "Oda's method" for the calculation of the effective permeability of fractures.
Geotechnique Vol 35, No 4, pp 483-495	1985	Permeability tensor for discontinuous rock masses	M. Oda	Generalisation of the Oda's method for implicit fracture networks. Comparison with other techniques for the calculation of the effective permeability.
SPE 36730	1997	A new method to calculate effective permeability of grid blocks used in the simulation of naturally fractured reservoirs	M. Lough S. Lee J. Kamath	First to present a numerical method using a DFN model for the calculation of the fracture effective permeability
Petroleum Geoscience, Vol.7. pp. S43 - S52	2001	Nested geological modelling of naturally fractured reservoirs	M. C. Cacas J. M. Daniel J. Letouzey	Proposed a fully integrated workflow for the modelling of fractures based on fracture sets.
SPE 107525	2007	Fast and efficient modelling and conditioning of naturally fractured reservoir models using static and dynamic data	M. Garcia, F. Gouth, O. Gosselin	First to present a new method for the calculation of the anisotropic permeability tensor of fracture networks distinguishing horizontal and vertical flow (the EPB method).
SPE 118924	2009	Massively parallel sector scale discrete fracture and matrix simulations	S. Geiger, Q. Huangfu, F.Reid, S.Matthäi, D. Coumou, M. Belayneh, C. Fricke, K. Schmid	First to perform a reservoir scale simulation of a fracture reservoir using a DFM model with CSMP++.
SPE 131126	2010	Effective fracture network permeability: comparative study of calculation methods	N. Cottereau, M. Garcia, O. Gosselin, L. Vigier	Comparison of different methods for the calculation of the anisotropic permeability tensor of fracture networks using three pieces of software and performing dynamic simulations.
SPE 154369	2012	Static and dynamic assessment of DFN permeability upscaling	M. Ahmed Elfeel S. Geiger	Propose a comparison between the different calculation methods of the effective permeability and for different grid sizes.
Mineralogical Magazine, special issue on Geological Disposal of Radioactive Waste (2012)	2012	Comparison of discrete fracture network and equivalent continuum simulations of fluid flow through two-dimensional fracture networks for the DECOVALEX-2011 project	C. Leung A. Hoch R. Zimmerman	Comparison between a simulation using the DFN model and simulations using effective permeabilities calculated with Oda's method on different grid sizes.

PMM Vol. 24, No. 5, pp 852-864 (1960)

Basic concepts in the theory of seepage of homogeneous liquids in fissured rocks (strata)

Authors: G. Barenblatt, Iu. Zheltov, I. Kochina

Contribution to the understanding of naturally fractured reservoir simulation:

This paper is the first to propose the use of the dual-porosity/dual-permeability model to simulate NFR. It is the first to present the concept of “effective property”.

Objective of the paper:

To propose a simplified model enabling to simulate fluid flow in a NFR.

Methodology used:

The concept is to assume two different pressures and flow velocities at each point in the reservoir: one for the fractures and one for the matrix. The Darcy’s equation is ten solved for these two domains and a transfer of flow between them is possible.
Several different boundary conditions are presented:

- The pressure is given
- The flow rate is given
- A linear combination of the pressure and the flow rate is given

The result of the equations is calculated for the cases of a steady and a transient flow.

Conclusion reached:

This model enables to take into account the effect of fractures when simulating fluid flow in a NFR. Equations for any type of simulation concerning NFR are presented in this paper and have been resolved.

Comments:

This model does not neglect the flow in the matrix therefore two equations of flow must be resolved. A lot of properties concerning the fractures and the matrix must be known and this article does not explain how to find their value. This model assumes the fracture permeability is isotropic.

SPE 426 (1963)

The Behavior of Naturally Fractured Reservoirs

Authors: J.E. Warren, P.J. Root

Contribution to the understanding of naturally fractured reservoir simulation:

This paper is the first one to propose the use of a simplified model called “the sugar box model” for NFR simulations. It is the first paper to introduce the dual-porosity model and to propose an equation for well-test analysis based on this model.

Objective of the paper:

To propose an idealised model enabling to characterise fracture reservoirs performing a well-test analysis.

Methodology used:

The authors considered an idealized fracture reservoir using the sugar-box model: They considered two media with different permeabilities and porosities. Then, they resolved the equation of flow for a pressure build-up using Darcy’s law and considering oil expansion.
The result was tested considering simple values of porosities and permeabilities in order to compare it with other authors’ solutions.

Conclusion reached:

1. The model considered seems realistic and can be used to characterize a fractured reservoir.
2. Two additional parameters (λ and ω) are sufficient to describe the difference between the pressure build-up curve of a fractured reservoir and the one of a homogeneous reservoir.
3. The build-up curve observed during well test analysis is similar to the one of stratified reservoirs

Comments:

The concept of “dual-medium” is used in the majority of commercial software for naturally fractured reservoir simulation.
In the sugar box model it is assumed that the permeability of the fracture network is isotropic. This is generally not the case in real reservoirs.

Water Resources Research Vol. 5, No. 6, pp 1273-1289 (1969)

Anisotropic permeability of fractured media

Authors: D.T. Snow

Contribution to the understanding of naturally fractured reservoir simulation:

This paper is the first to propose the so-called “Oda’s method” for the calculation of the effective permeability of fractures.

Objective of the paper:

Propose a technique for the calculation of the effective permeability which takes into account the anisotropy of the fracture network.

Methodology used:

The fracture permeability is calculated from the aperture using the cubic law.
The equations related to the new proposed method are presented and resolved for two particular cases:

- Fractures are represented as layers of the reservoir
- Fractures can have different orientations. In this case an extension is proposed to take into account the connectivity between two fractures which depends on the intersection angle.

The author presents data from real reservoirs and explains how to use them to build a simple DFN and apply his technique to calculate the effective permeability.

Conclusion reached:

The effective permeability of fractures depends on their orientation. If the fractures are parallel there is an infinite anisotropy in one direction. If the fractures are intersecting each other orthogonally the reservoir is statistically isotropic.

Comments:

The equations presented by Snow are the one used in the so-called “Oda’s method” for the calculation of the effective permeability on DFN. The improvement brought by Oda enables to use this technique on implicit fracture networks. The method presented for the construction of a DFN is very basic and does not represent accurately the heterogeneity of NFRs.

Geotechnique Vol 35, No 4, pp 483-495 (1985)

Permeability tensor for discontinuous rock masses

Author: M. Oda

Contribution to the understanding of naturally fractured reservoir simulation:

This paper presents the most used upscaling technique to calculate the anisotropic permeability tensor of a fracture network. It is very important because it is one of the first papers that propose to take into account the fracture directions for reservoir flow prediction. It extends the theory of Snow by taking into account fracture sets.

Objective of the paper:

To propose a reliable method to calculate the anisotropic permeability tensor of a fracture network assuming that the fracture geometry is known. In addition, the paper proposes a technique to estimate the fracture directions from wireline logs.

Methodology used:

The author considered a homogenous block cut by several cracks with different directions. The cracks were considered to be void and penny shaped for volume estimation. The equation of flow in the fractures assumed that they were of infinite extent.
The resulting effective permeability calculation method was compared with numerical experiments performed by Long et al. (1982).
The estimation of the crack tensor was performed using geometrical probabilities. Two methods are proposed:

- One assumes the fracture aperture is constant
- Another one assumes the fracture aperture is proportional to the crack size.

Conclusion reached:

1. The method presented gives the right major and minor principal axes of the permeability tensor. They only depend on the fracture geometry.
2. This technique generally overestimates the fracture permeability and connectivity therefore correction factors must be introduced to improve the accuracy of this method

Comments:

The Oda's method is entirely analytic. It is therefore the quickest technique for the estimation of a fracture network effective permeability. However, the assumptions used for it (especially the infinite extent of the fracture and the full connectivity) are generally not verified in real reservoirs.

SPE 36730 (1997)

A new method to calculate effective permeability of grid blocks used in the simulation of naturally fractured reservoirs

Authors: M. Lough, S. Lee, J. Kamath

Contribution to the understanding of naturally fractured reservoir simulation:

This paper is the first to present a numerical method using a DFN model for the calculation of the fracture effective permeability

Objective of the paper:

This paper proposes a new technique for the calculation of the effective permeability taking into account the fracture connectivity.

Methodology used:

The technique is applied for 2 dimensions DFNs. The intersection points between the fractures must be found and the pressure at each of these “nodes” must be calculated. The chosen boundary conditions are the No Flow one.
This technique is applied for the simulation of a part of a real reservoir using the dual-porosity simulation model

Conclusion reached:

This method enables to estimate more accurately than with the Oda’s method the effective permeability of a DFN. The results are very different compared with Oda’s technique when the fracture density is low. This numerical technique is useful to evaluate the cells where the facture connectivity is so poor that the fracture can be neglected during simulation.

Comments:

The main drawback of this technique is that only few fractures can be represented otherwise the computation time is too long for application in real reservoirs. This paper only presents the No Flow boundary condition and only 2D fracture networks can be simulated. This technique has been extended for 3D DFNs in Petrel.

Petroleum Geoscience, Vol. 7. pp. S43-S52 (2001)

Nested geological modelling of naturally fractured reservoirs

Authors: M. C. Cacas, J. M. Daniel, J. Letouzey

Contribution to the understanding of naturally fractured reservoir simulation:

This paper propose a fully integrated workflow for the modelling of fractures using on fracture sets

Objective of the paper:

The objective of this paper is to propose a systematic methodology for the building of a NFR simulation model

Methodology used:

Fractures are segregated between fracture corridors and smaller diffuse fractures. Small fractures are classified in fracture sets. These set are determined from the analysis of borehole data and the non-spatial parameters can be estimated from them. Then, an implicit fracture network (where fracture set non-spatial parameters and Fracture Densities are defined) is created. The fracture density is calculated using seismic and geological data. From this implicit network a DFN can be built and the fracture effective properties can be estimated.

Conclusion reached:

The process presented in this paper can be qualified as an “indirect process” since the DFN is computed from an implicit model. A lot of attention has been placed in the generation of a DFN. The current process is mostly based on statistics. A DFN generation based on a geo-mechanical model is likely to be more accurate.

Comments:

The workflow presented in this paper is the one applied in the IFP in-house software named FracaFlow. This methodology has been adapted in the Total software named GoFraK.

SPE 107525 (2007)

Fast and efficient modelling and conditioning of naturally fractured reservoir models using static and dynamic data

Authors: M. Garcia, F. Gouth, O. Gosselin

Contribution to the understanding of naturally fractured reservoir simulation:

This paper proposes a new method for the generation of a DFN from an implicit network and new numerical method to calculate the effective permeability of fractured reservoirs.

Objective of the paper:

The objective of this paper is to develop a new upscaling method more accurate than the Oda's method and quicker than the one based on DFN simulations. In addition, this paper presents how to perform history matching and well test analysis using this upscaling technique.

Methodology used:

First of all the Discrete Fracture Network is modelled using a geo-statistical technique based on the Fracture Density (FD).
The reservoir is divided in layers where all the fractures are perpendicular to the interlayer surface and are fully crossing the thickness of the layer.
The horizontal within-layer permeability tensor is calculated with a numerical simulation of flow using the DFN model and original boundary conditions: For each element where the effective permeability is computed one assumes that this element is periodically repeated in any direction. These boundary conditions are supposed to give more representative effective permeabilities than the no-flow or the linearly varying pressure boundaries.
The vertical inter-layer permeability is calculated analytically by decoupling the permeability due to persistent fractures and the one due to bedding-terminated ones. These two permeabilities are estimated using equations based on Darcy's law.
The history matching can be done by modifying spatial and non-spatial parameters using optimisation techniques.
A semi-analytic technique is also presented for the simulation of a well test.

Conclusion reached:

1. The new upscaling method proposed is easy to use and quicker than the "classical" techniques based on a DFN.
2. The integration of production history and well test data remains a challenging task

Comments:

The geo-modelling of the fracture network and the upscaling technique presented in this paper are implemented in the Total in-house software named GoFraK.

SPE 118924 (2009)

Massively parallel sector scale discrete fracture and matrix simulations

Authors: S. Geiger, Q. Huangfu, F.Reid, S.Matthäi, D. Coumou, M. Belayneh, C. Fricke, K. Schmid

Contribution to the understanding of naturally fractured reservoir simulation:

The authors were the first to perform a reservoir scale simulation of a fracture reservoir using a DFM model.

Objective of the paper:

This paper presents the techniques that have been used for the simulation of a full reservoir using a DFM model.

Methodology used:

The model FRACS2000 and the model RESERVOIR have been simulated with the CSMP++ software. These models were meshed with an unstructured hybrid-element grid constituted of up to 5 million cells.
To keep the computational time reasonable the resolution of the flow equations has been parallelised in 32, 64 and 128 processors and a study of the computational time depending on the number of processor has been performed. The simulation has been done in two computers at the Edinburgh Parallel Computing Centre at the University of Edinburg called NESS (with 32 processors and 64 GB of memory) and ECDF(with 128 dual-core processor and 236 quad-core processors; each processor has 2GB memory).
A single-phase and three-phase flow simulation was performed on FRACS2000 between two faces and a well-test was simulated on the RESERVOIR model.

Conclusion reached:

1. Simulation that would take weeks in a single processor can be performed in a few hours using parallelisation.

2. Performing this type of simulation can be very useful to estimate the effective permeability of reservoirs and study the flow transfers between fractures and the matrix.

3. Increasing the number of processor for the simulation does not necessarily decreases the computational time as much as expected. This is due to the fact that processors have to communicate between each other and increasing the number of processor increases the time spent in the “communication step”.

Comments:

Although it is now possible to simulate directly NFR it remains very difficult since all the reservoir must be meshed with millions of cells and many processors are necessary to keep the computational time reasonable. It is obvious that the construction of a grid mesh and flow simulations cannot be performed reasonably small time as it is necessary when modifying the geological model during well-test analysis or history matching. Nevertheless, a simulation on a smaller part of the reservoir (e.g. around a well) can be interesting.

SPE 131126 (2010)

Effective Fracture Network Permeability: Comparative Study of Calculation Methods

Authors: N. Cottereau, M.H. Garcia, O.R. Gosselin, L. Vigier

Contribution to the understanding of naturally fractured reservoir simulation:

This paper compares the resulting effective permeability of three different upscaling techniques. Moreover this paper present a new upscaling technique based on Oda's method called **2DILF**

Objective of the paper:

The objective of this paper is to benchmark different upscaling techniques used for the calculation of a fracture network effective anisotropic permeability tensor. The CPU time, the resulting permeability tensor and the impact on dynamic simulation are compared.

Methodology used:

Three different pieces of software (P, F and G) were used and several upscaling calculations were performed:

- **PA** and **FA**: The Oda's method of the pieces of software P and F
- **2DILF**: The 2D analytical method based on Oda's method of the software G
- **PN** and **FN**: The numerical method based on a DFN of P and F
- **GN**: The new IBPOS method presented by Garcia et al. 2007 of the software G.

Two fracture networks (one of 120,000 cells and one of 20,000 cells) were used for simulation. A production scenario with 4 producers and 4 injectors was modelled.
The equivalent permeabilities and the total oil produced were compared for the different upscaled models.

Conclusion reached:

1. The numerical methods based on DFN failed to compute the 120,000 cell reservoir case.
2. The computational time is the smallest for **2DILF** (2D Oda's method) then the classical Oda's method is longer, the IBPOS method is again longer and the classical numerical method is prohibitively time-consuming even for the small model.
3. The new **2DILF** and **IBPOS** methods are reliable and can be used for real reservoir simulations

Comments:

Although the different upscaling techniques were compared to each other their accuracy could not be checked because they were not compared with a fine scale explicit simulation used as a reference solution.

SPE 154369 (2012)

Static and dynamic assessment of DFN permeability upscaling

Authors: M. Ahmed Elfeel, S. Geiger

Contribution to the understanding of naturally fractured reservoir simulation:

This paper proposes a comparison between the different calculation methods of the effective permeability and for different grid sizes.

Objective of the paper:

The objective of this paper is to study the uncertainty in the calculation of the effective permeability depending on the calculation method used and the grid block size.

Methodology used:

A first reservoir located at the Teapot Dome Structure, Wyoming, USA was modelled and then the fracture effective permeability was calculated using three calculation methods (Oda, No Flow numerical method and Linear Pressure numerical method). The resulting effective permeabilities were compared for different grid sizes and the calculation time was measured.
A dynamic simulation was performed using the single-porosity model (the flow from and through the matrix was neglected).
In a second part, the FRACS2000 model was used for the calculation of the effective permeability using different calculation techniques and different grid sizes. No dynamic simulation was performed. The effective permeability of the full model was also simulated using CSMP++ by modelling a single-phase flow between two faces.

Conclusion reached:

1. Oda's method systematically overestimates the effective permeability compared with other methods.

2. Oda's method is very inaccurate for large cells but it is difficult to evaluate the optimum grid size to increase the accuracy.

3. Numerical calculation techniques have a too long computation time to be used in practice.

Comments:

In this paper the effective permeability calculation methods were compared to each other using dynamic simulations but no explicit simulation was used as a reference solution.
One of the main problems seems to be that CSMP++ does not simulate easily a reservoir with a 0 permeability matrix.

Mineralogical Magazine, special issue on Geological Disposal of Radioactive Waste (2012)

Comparison of discrete fracture network and equivalent continuum simulations of fluid flow through two-dimensional fracture networks for the DECOVALEX-2011 project

Authors: C. Leung, A. Hoch, R. Zimmerman

Contribution to the understanding of naturally fractured reservoir simulation:

This paper proposes a comparison between a simulation using the DFN model and simulations using effective permeabilities calculated with Oda's method on different grid sizes.

Objective of the paper:

The objective of this paper is to study the impact of the grid size when computing the effective permeability. The best grid size can be determined by comparison with a DFN model simulation used as a reference solution.

Methodology used:

A flow was simulated between two faces of a 2D fracture network using an explicit simulation (the DFN model) where all the fractures are represented and the model using effective properties. The effective permeability was calculated using the Oda's method and several grids (10x10 cells; 40x40 cells; 100x100 cells; 4000x400 cells) were used. The matrix was assumed to have a 0 mD permeability therefore only flow into fractures was simulated.

The simulation using the DFN model was performed with the software named NAPSAC while the simulation using effective properties was performed with NAMMU.

Conclusion reached:

1. The effective permeability simulation model is relatively accurate for all the grid sizes. A maximum difference of 20% is observed between the different grid sizes.

2. The concept of Representative Elementary Volume is not very interesting since even for very small grid sizes the simulation results are similar.

Comments:

This paper suggests that the concept of Representary Elementary Volume is not relevent when chosing a grid size to compute effective properties. Nevertheless, only the Oda's method was studied and results may be different for other upscaling techniques.
In addition, only one 2D Discrete Fracture Network was simulated. The influence of the grid size may be more important for 3D fracture networks.

APPENDIX B: Oda's effective permeability calculation method

In the demonstration proposed by Oda the fractures are assumed to have a circular form ("penny shaped"). Although the Oda's method can be used for any form of fracture in this demonstration the fracture will be assumed to be rectangular.
Each fracture can be defined by:

- its orientation $\vec{n}$,
- its aperture a (in m),
- its length l (in m),
- its width w (in m),
- the location of its centre.

The fracture orientation $\vec{n}$ is defined as a unit vector normal to the fracture major plane. With this definition it is obvious that $\vec{n}$ and $-\vec{n}$ can both represent the orientation of the same fracture.

Let's consider one fracture set (s). The non-spatial parameters of this set are:

- the probability density function of the fracture length $f^{(s)}(l)$
- the probability density function of the aperture $g^{(s)}(a)$
- the probability density function of the orientation $2E^{(s)}(\vec{n})$
- the elongation ratio (which is assumed to be constant) $el^{(s)}$

$$el = \frac{l}{w} \qquad \text{(Equation B.1)}$$

By definition of probability density functions one can write (Equation B.2), (Equation B.3) and (Equation B.4):

$$\int_0^{+\infty} f^{(s)}(l)\, dl = 1 \qquad \text{(Equation B.2)}$$

$$\int_0^{+\infty} g^{(s)}(a)\, da = 1 \qquad \text{(Equation B.3)}$$

$$\int_{\Omega/2} 2E^{(s)}(\vec{n})\, d\Omega = \int_{\Omega} E^{(s)}(\vec{n})\, d\Omega = 1 \qquad \text{(Equation B.4)}$$

In the (Equation B.4) Ω represents the solid angle formed by the vector $\vec{n}$. This vector must be integrated over the surface of a hemisphere ($\int_{\Omega/2}$) to take into account all the possible orientations. When it is integrated over the surface of a sphere ($\int_{\Omega}$) each orientation is counted twice.

Let's consider a reservoir block (V):

- V_{block} is the volume of the block (in m^3)
- $\rho^{(s)}$ is the total number of fractures from the set (s) whose centre is inside the block (V) divided by the block volume (V_{block}). $\rho^{(s)}$ can be seen as a fracture density.
- $dN^{(\vec{n},l,a)}$ is the number of fractures from the set (s) whose centre is inside (V) with the orientation $\vec{n}$, the aperture a and the length l.

We have:

$$dN^{(\vec{n},l,a)} = V_{block}\ \rho^{(s)}\ 2E^{(s)}(\vec{n})\ g^{(s)}(a)\ f^{(s)}(l)\ d\Omega\ dl\ da \qquad \text{(Equation B.5)}$$

The volume associated with these dN fractures is:

$$dV^{(\vec{n},l,a)} = V_{block}\ \rho^{(s)}\ \frac{1}{el^{(s)}}\ a\ l^2\ 2E^{(s)}(\vec{n})\ g^{(s)}(a)\ f^{(s)}(l)\ d\Omega\ dl\ da \qquad \text{(Equation B.6)}$$

The Darcy's law in an anisotropic reservoir is presented by the (Equation B.7):

$$\vec{q} = -\frac{\bar{\bar{k}}}{\mu}\ \overrightarrow{grad}(p) \qquad \text{(Equation B.7)}$$

$\vec{q}$ is the Darcy velocity (in m/s)
$\bar{\bar{k}}$ is the permeability tensor (in m^2)
μ is the viscosity (in Pa.s)
p is the pressure (in Pa)

Using the Einstein summation convention this equation can be written in the form of the (Equation B.8):

$$q_i = -\frac{k_{ij}}{\mu}\frac{\partial p}{\partial x_j} = \frac{k_{ij}}{\mu} J_j \qquad \text{(Equation B.8)}$$

J_j is defined by the (Equation B.9) and is assumed to be constant in each reservoir block.

$$J_j = -\frac{\partial p}{\partial x_j} \qquad \text{(Equation B.9)}$$

Let's $\vec{J}^{(\vec{n},l,a)}$ be a component of $\vec{J}$ projected in the direction of flow for the dN fractures with the orientation $\vec{n}$, the aperture a and the length l.

$$\vec{J}^{(\vec{n},l,a)} = \vec{J} - \left(\vec{n}\,.\,\vec{J}\right)\vec{n} \qquad \text{(Equation B.10)}$$

$$J_i^{(\vec{n},l,a)} = \left(\delta_{ij} - n_i n_j\right) J_j \qquad \text{(Equation B.11)}$$

Using the cubic law (which assumes the fractures are infinite planes and the flow inside is laminar) we have (Equation B.12):

$$v_i^{(\vec{n},l,a)} = \frac{1}{12}\frac{a^2}{\mu} J_i^{(\vec{n},l,a)} \qquad \text{(Equation B.12)}$$

$$v_i^{(\vec{n},l,a)} = \frac{1}{12}\frac{a^2}{\mu}\left(\delta_{ij} - n_i n_j\right) J_j \qquad \text{(Equation B.13)}$$

$v_i^{(\vec{n},l,a)}$ is the velocity in the the $dN^{(\vec{n},l,a)}$ fractures (in m/s)
a is the fracture aperture (in m)
μ is the viscosity (in Pa.s)
J_j is the pressure gradient (in Pa/m)
δ_{ij} is the Kronecker delta

The average velocity $\overline{v_i}^{(s)}$ in the reservoir block (V) for the fracture set (s) is defined by (Equation B.14). The main assumption behind this equation is that all the fractures are flowing (there is a full connectivity of the fracture network).

$$\overline{v_i}^{(s)} = \frac{1}{V_{block}} \int_{(\vec{n},l,a)} v_i^{(\vec{n},l,a)}\, dV^{(\vec{n},l,a)} \qquad \text{(Equation B.14)}$$

Using (Equation B.6), (Equation B.13) and (Equation B.14) we have:

$$\overline{v_i}^{(s)} = \frac{J_j}{12\,\mu}\,\frac{\rho^{(s)}}{el^{(s)}} \int_0^{+\infty}\int_0^{+\infty}\int_{\Omega} a^3\, l^2 \left(\delta_{ij} - n_i n_j\right) E^{(s)}(\vec{n})\; g^{(s)}(a)\; f^{(s)}(l)\, d\Omega\, dl\, da \qquad \text{(Equation B.15)}$$

The pressure gradient J_j is assumed to be constant in each reservoir block.

The average velocity $\overline{v_i}$ for all the fracture sets in the block (V) is given by (Equation B.16):

$$\overline{v_i} = \sum_s \overline{v_i}^{(s)} \qquad \text{(Equation B.16)}$$

If it is assumed there is no flow in the matrix, the block average velocity $\overline{v_i}$ is equal to the Darcy velocity q_i in (Equation B.8).

From (Equation B.8), (Equation B.15) and (Equation B.16) the effective permeability tensor k_{ij} (in m^2) can be identified:

$$k_{ij} = \frac{1}{12}\left(P_{kk}\,\delta_{ij} - P_{ij}\right) \qquad \text{(Equation B.17)}$$

$$P_{kk} = P_{11} + P_{22} + P_{33} \qquad \text{(Equation B.18)}$$

P_{ij} is called the “crack tensor”. It is defined by (Equation B.19):

$$P_{ij} = \sum_s \frac{\rho^{(s)}}{el^{(s)}} \int_0^{+\infty} a^3\, g^{(s)}(a)\, da \int_0^{+\infty} l^2\, f^{(s)}(l)\, dl \int_{\Omega} n_i n_j\, E^{(s)}(\vec{n})\; d\Omega \qquad \text{(Equation B.19)}$$

This equation can be used in the case of an implicit fracture network.

In the case of a Discrete Fracture Network (DFN) there is no more probability density functions and the crack tensor can be calculated with (Equation B.20):

$$P_{ij} = \frac{1}{V_{block}} \sum_f \frac{{a^{(f)}}^3\, {l^{(f)}}^2}{el^{(f)}}\; {n_i}^{(f)}\, {n_j}^{(f)} \qquad \text{(Equation B.20)}$$

In a more general case for a DFN with:

- $A^{(f)}$ the fracture area (in m^2)
- $a^{(f)}$ the fracture aperture (in m)
- $k^{(f)}$ the fracture permeability (in any unit)
- ${n_i}^{(f)}$ the fracture orientation
- V_{block} the block volume (in m^3)
- k_{ij} the effective permeability tensor (in the same unit as $k^{(f)}$)

We have:

$$k_{ij} = \frac{1}{V_{block}} \sum_f a^{(f)} A^{(f)} \, k^{(f)} \quad \left(\delta_{ij} - \, {n_i}^{(f)} \, {n_j}^{(f)}\right) \qquad \text{(Equation B.21)}$$

APPENDIX C: Homogeneous model properties

1. Model characteristics

Size of the model	
In X direction:	720 m
In Y direction:	720 m
In Z direction:	10 m

Table C.1: Model dimensions

Horizontal fracture set		Vertical fracture set	
Constant aperture	0.1 m	Constant aperture	0.1 m
Constant permeability	2.00E+05 mD	Constant permeability	2.00E+05 mD
Shape	Rectangular	Shape	Rectangular
Constant length	60 m	Constant length	60 m
Constant width	10 m	Constant width	10 m
Constant fracture density	0.094 m^2/m^3	Constant fracture density	0.094 m^2/m^3

Table C.2: Fracture set properties

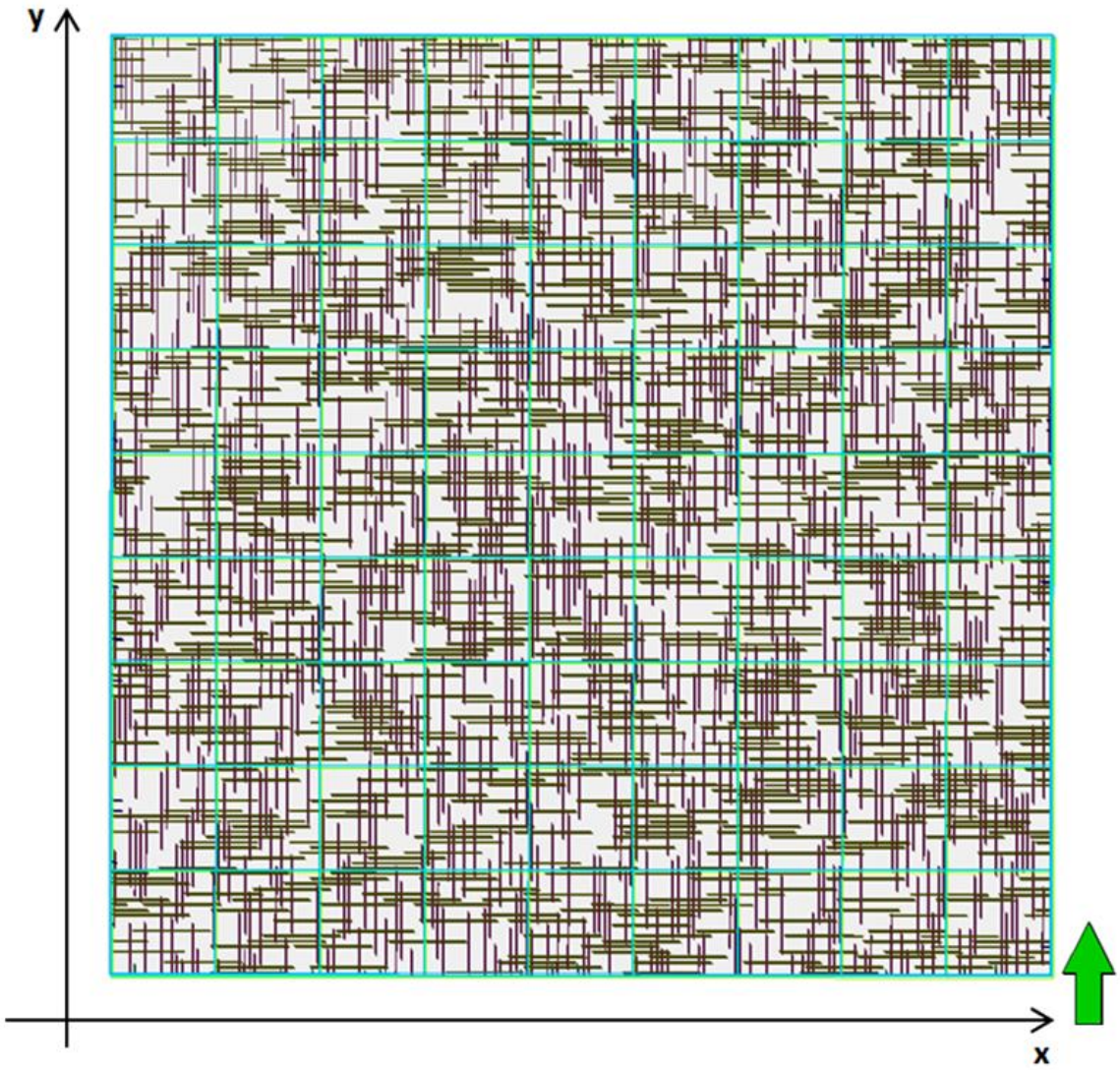

Figure C.1: Discrete Fracture Network

2. Effective properties

2.1. Grid with 9x9 cells

Grid properties	
Cell size in X direction:	80 m
Cell size in Y direction:	80 m
Cell size in Z direction:	10 m
Total number of cells:	9x9 = 81

Table C.3: Grid properties

2.1.1. Effective porosity

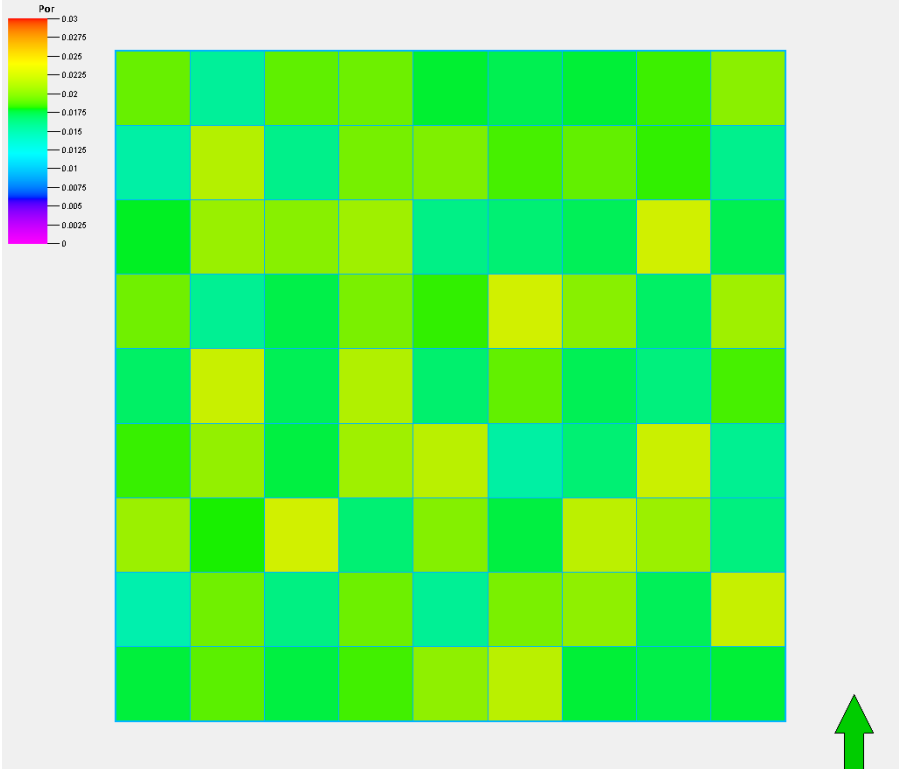

Figure C.2: Effective porosity with 9x9 cells

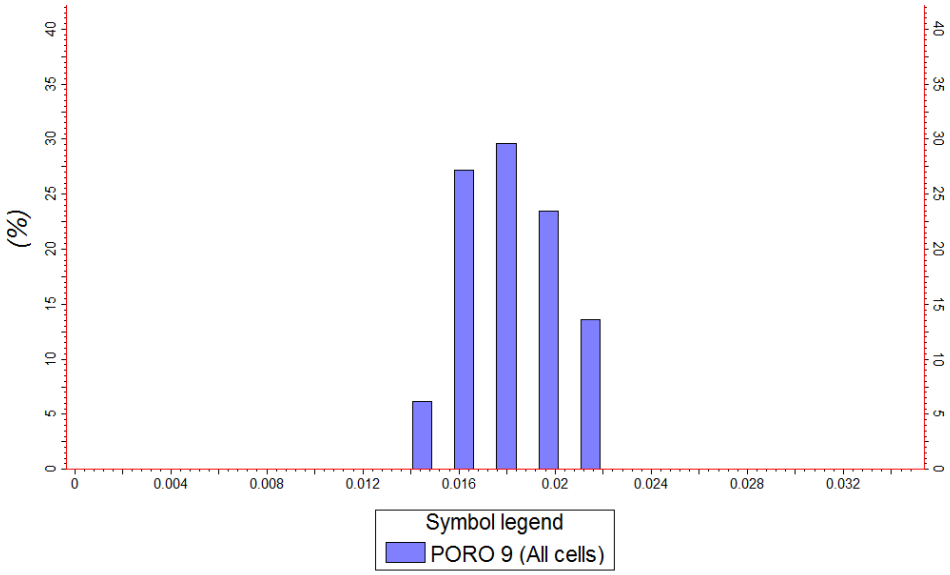

Figure C.3: Effective porosity histogram with 9x9 cells

	Mean	Std. Deviation
Effective porosity	0.0186	0.002

Table C.4: Effective porosity with 9x9 cells statistics

2.1.2. Effective permeability in X direction

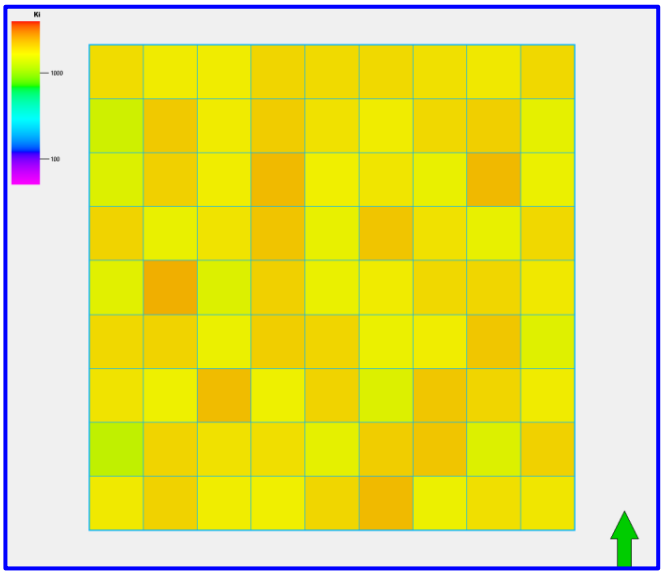

Figure C.4: Effective permeability in X direction with **Oda's** method (in mD)

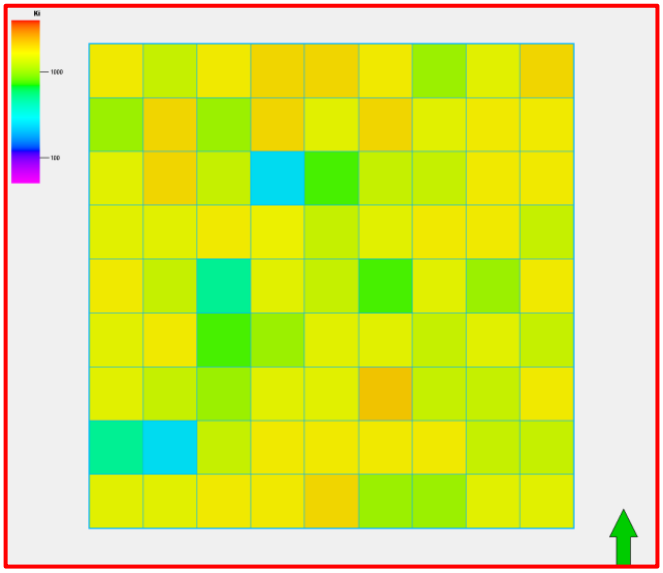

Figure C.5: Effective permeability in X direction with the **Linear Pressure** numerical method (in mD)

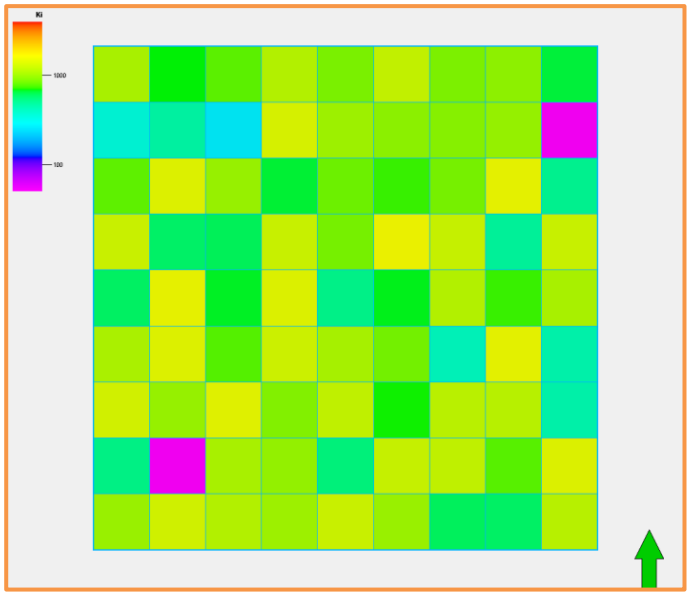

Figure C.6: Effective permeability in X direction with the **No Flow** numerical method (in mD)

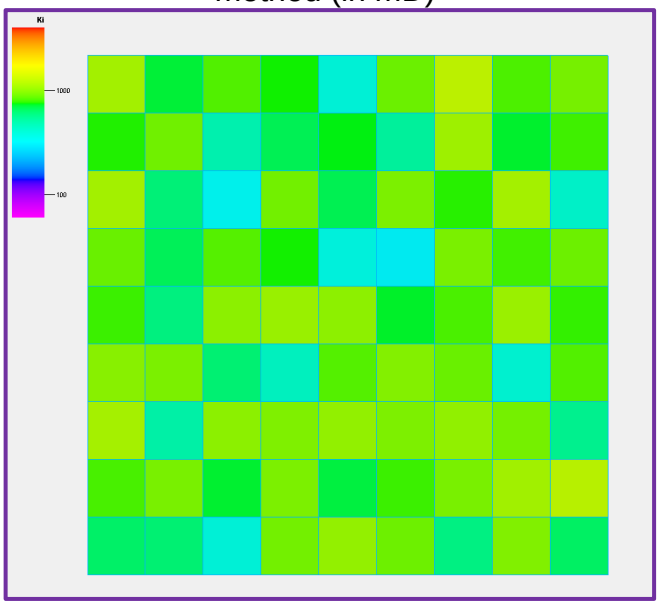

Figure C.7: Effective permeability in X direction with the **IBPOS** numerical method (in mD)

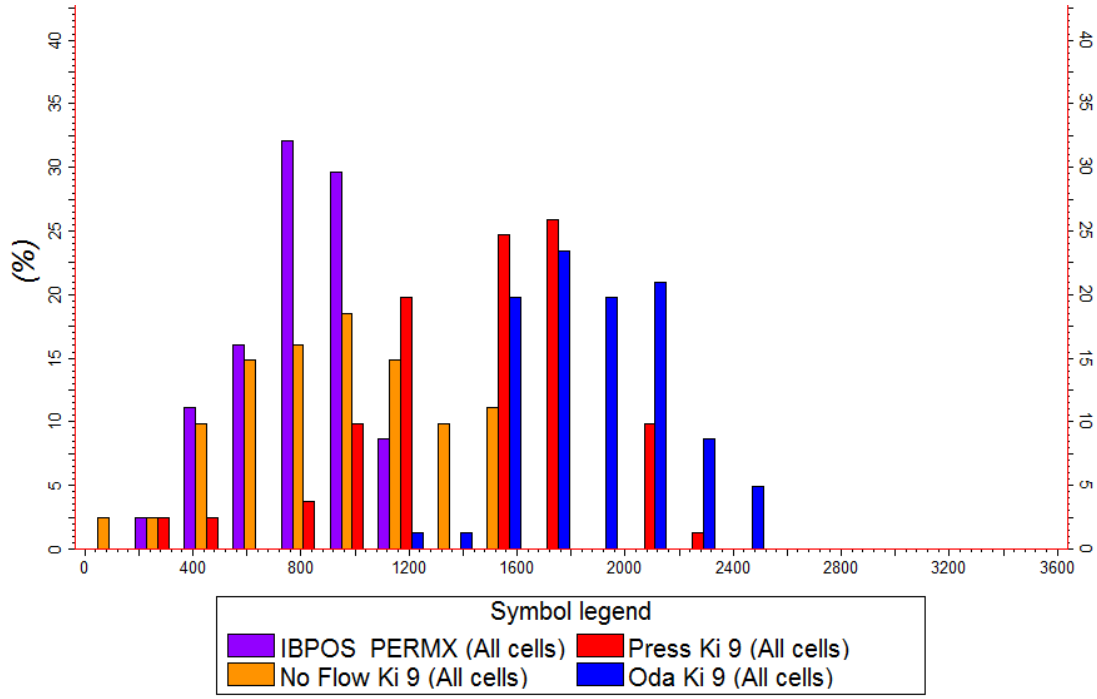

Figure C.8: Effective permeability in X direction histogram for **Oda**'s method (in blue), the **Linear Pressure** numerical method (in red), the **No Flow** numerical method (in orange) and the **IBPOS** method (in purple) (in mD)

	Mean	Std. Deviation
Effective perm. X using **Oda**'s method	1853.09 mD	267.53 mD
Effective perm. X using **Linear Pressure** numerical method	1439.6 mD	408.38 mD
Effective perm. X using **No Flow** numerical method	933.6 mD	358.69 mD
Effective perm. X using **IBPOS** numerical method	800.5 mD	214.76 mD

Table C.5: Effective permeability in X direction with 9x9 cells statistics

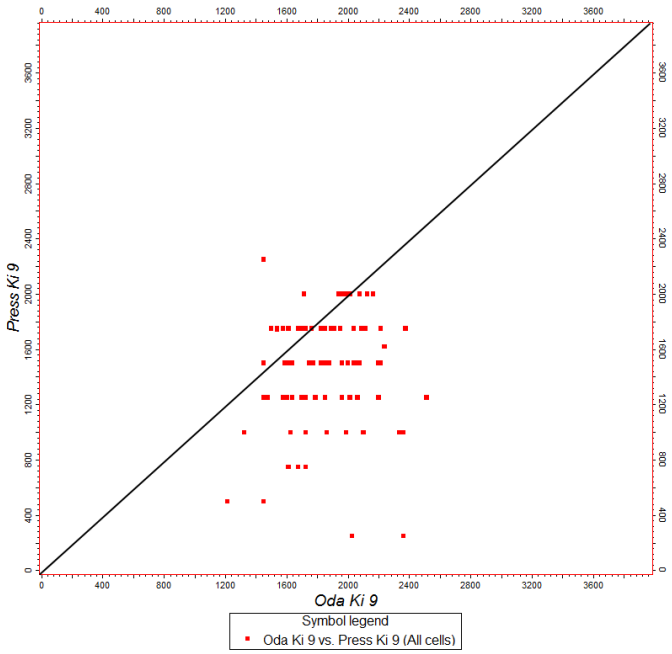

Figure C.9: Comparison of the effective permeability in X direction between the **Linear Pressure** numerical method and the **Oda**'s method (in mD)

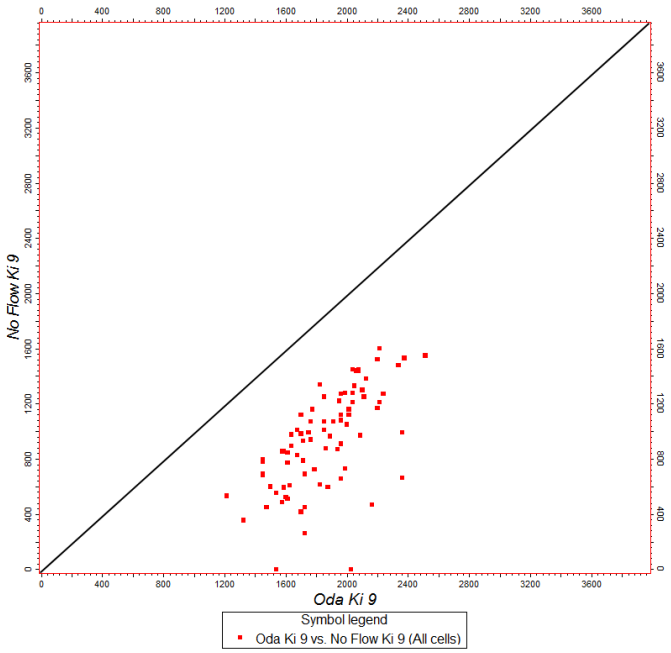

Figure C.10: Comparison of the effective permeability in X direction between the **No Flow** numerical method and the **Oda**'s method (in mD)

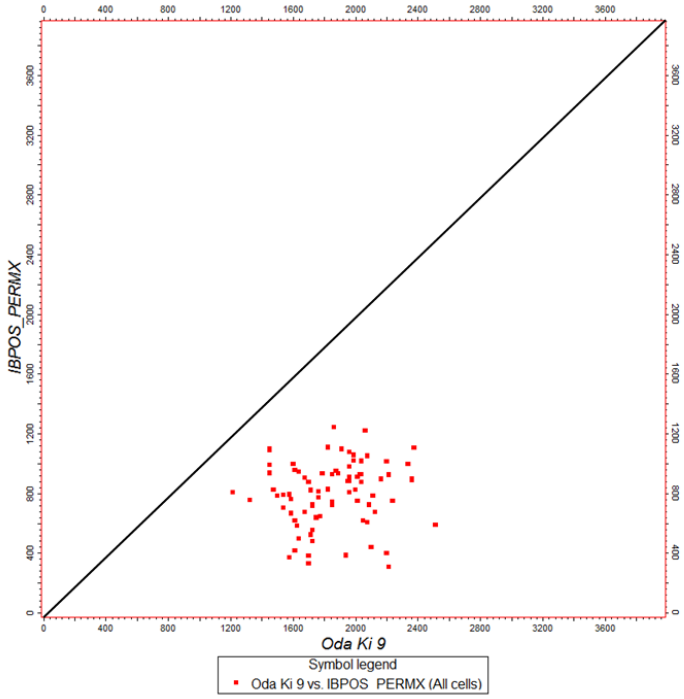

Figure C.11: Comparison of the effective permeability in X direction between the **IBPOS** numerical method and the **Oda**'s method (in mD)

2.1.3. Effective permeability in Y direction

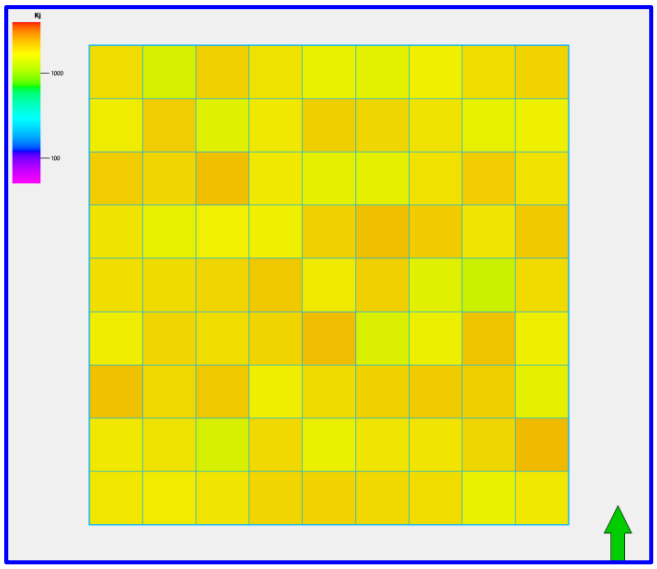

Figure C.12: Effective permeability in Y direction with **Oda's** method (in mD)

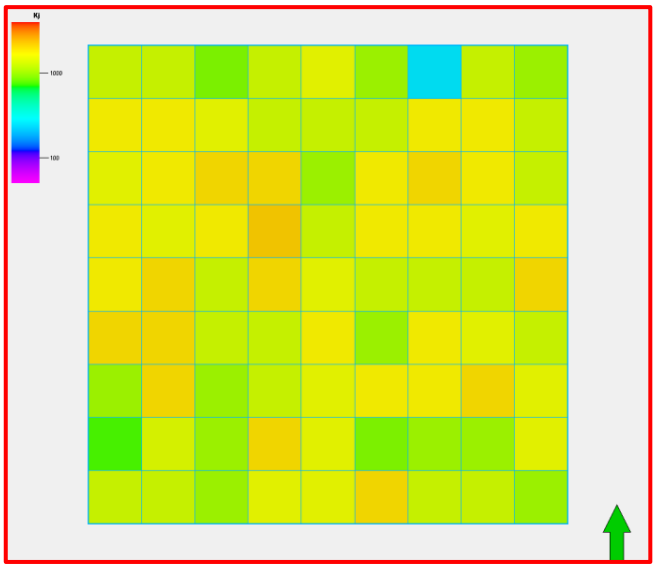

Figure C.13: Effective permeability in Y direction with the **Linear Pressure** numerical method (in mD)

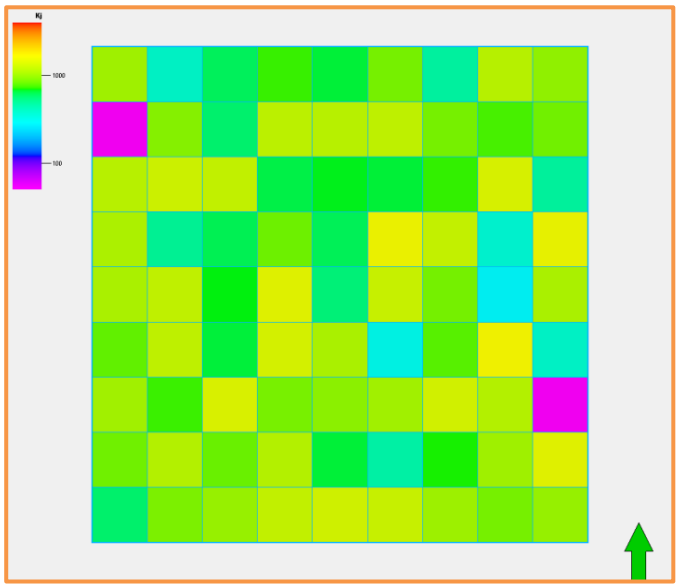

Figure C.14: Effective permeability in Y direction with the **No Flow** numerical method (in mD)

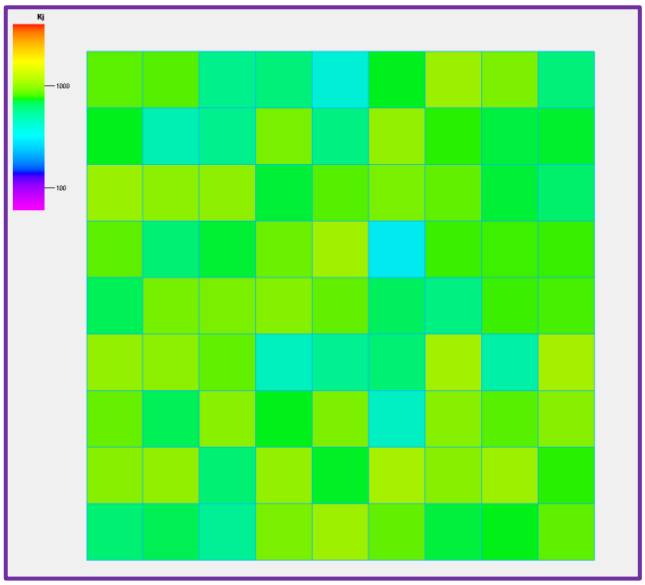

Figure C.15: Effective permeability in Y direction with the **IBPOS** numerical method (in mD)

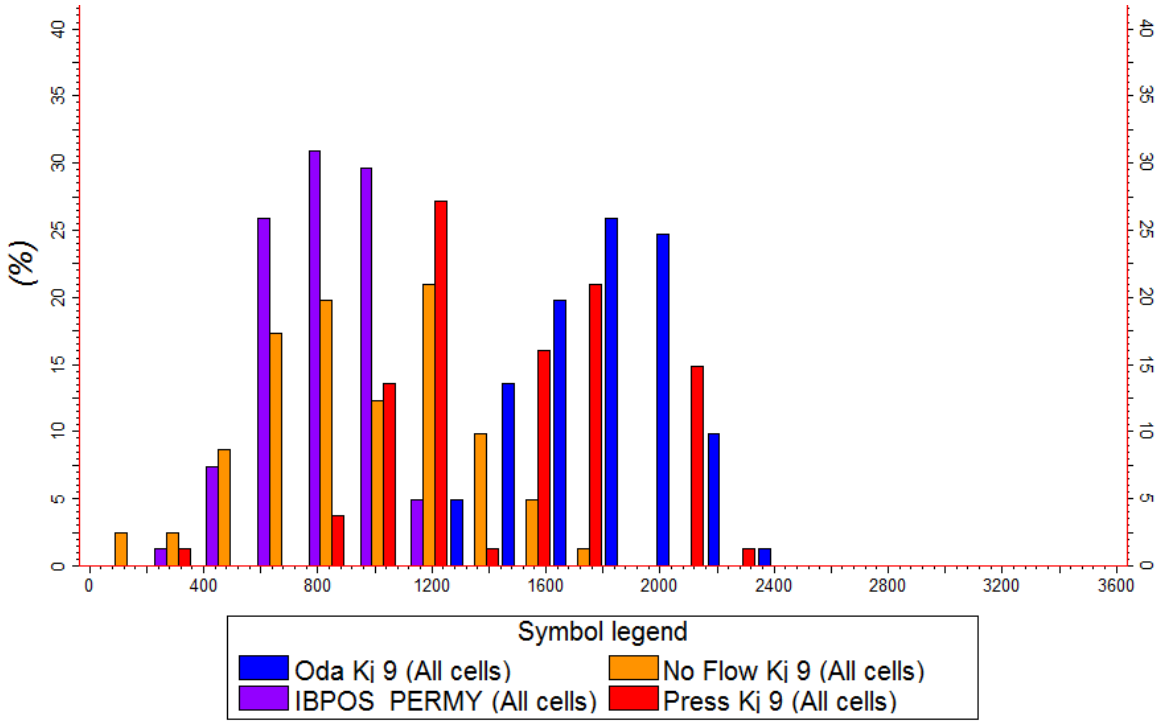

Figure C.16: Effective permeability in Y direction histogram for **Oda**'s method (in blue), the **Linear Pressure** numerical method (in red), the **No Flow** numerical method (in orange) and the **IBPOS** method (in purple) (in mD)

	Mean	Std. Deviation
Effective perm. Y using **Oda**'s method	1853.09 mD	267.53 mD
Effective perm. Y using **Linear Pressure** numerical method	1439.6 mD	408.38 mD
Effective perm. Y using **No Flow** numerical method	933.6 mD	358.69 mD
Effective perm. Y using **IBPOS** numerical method	796.1 mD	189.74 mD

Table C.6: Effective permeability in Y direction with 9x9 cells statistics

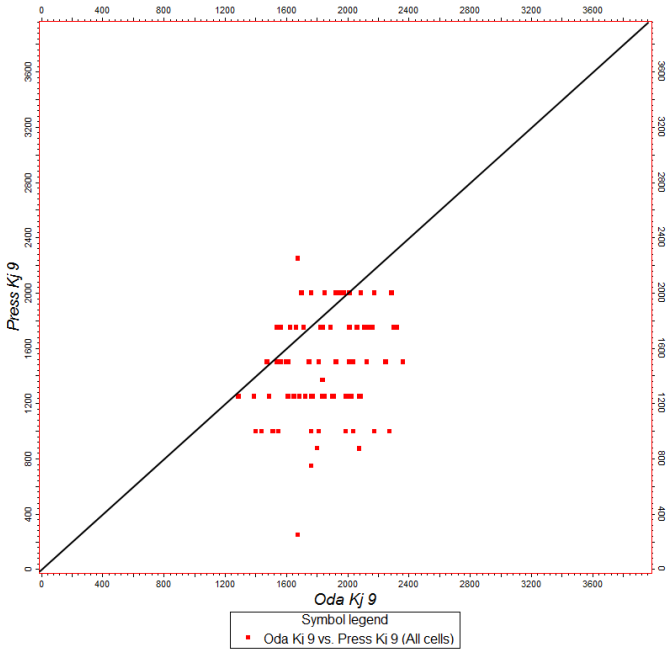

Figure C.17: Comparison of the effective permeability in Y direction between the **Linear Pressure** numerical method and the **Oda**'s method(in mD)

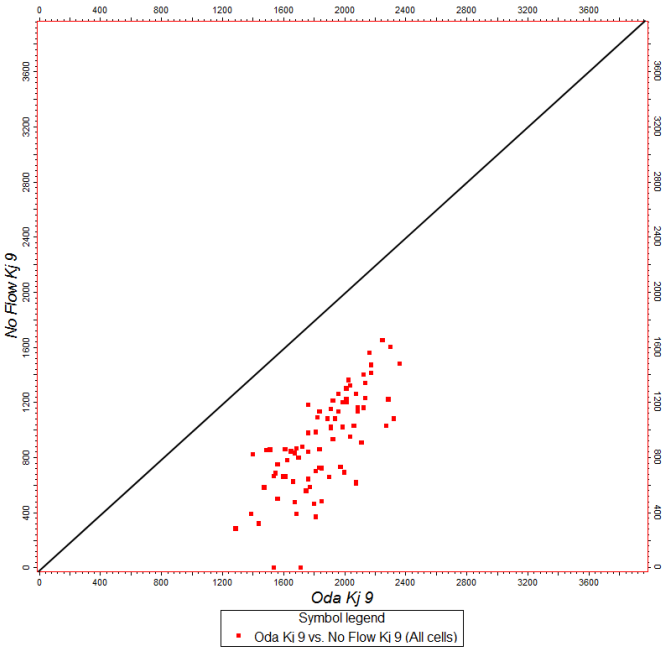

Figure C.18: Comparison of the effective permeability in Y direction between the **No Flow** numerical method and the **Oda**'s method(in mD)

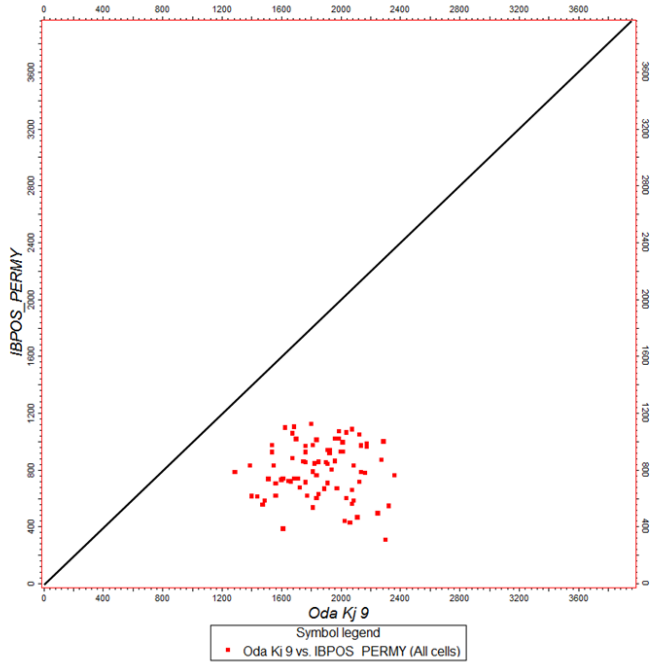

Figure C.19: Comparison of the effective permeability in Y direction between the **IBPOS** numerical method and the **Oda**'s method(in mD)

2.2. Grid with 27x27 cells

Grid properties	
Cell size in X direction:	26.7 m
Cell size in Y direction:	26.7 m
Cell size in Z direction:	10 m
Total number of cells:	27x27 = 729

Table C.7: Grid properties

2.2.1. Effective porosity

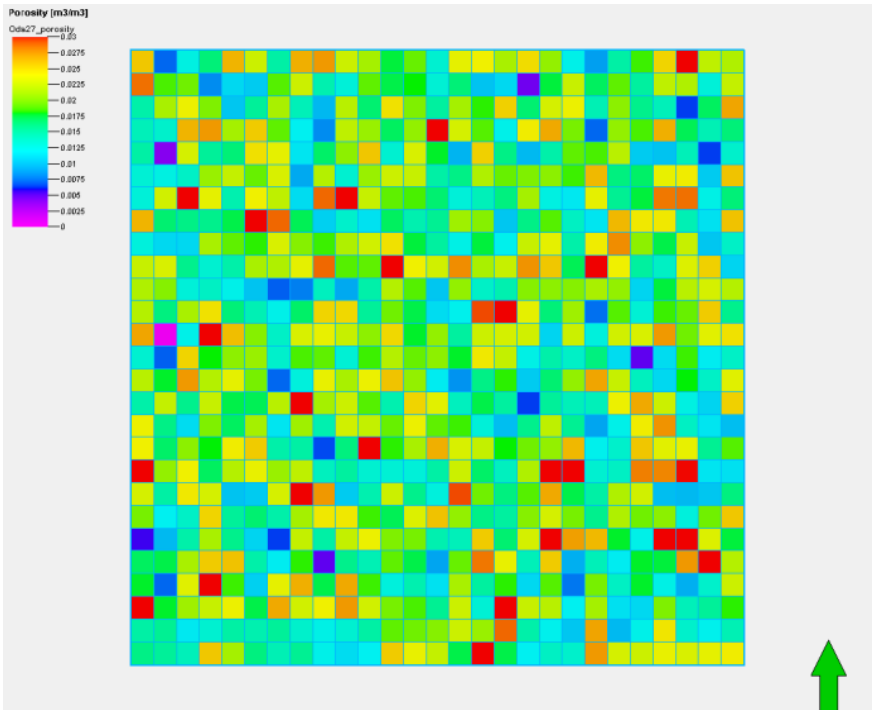

Figure C.20: Effective porosity with 27x27 cells

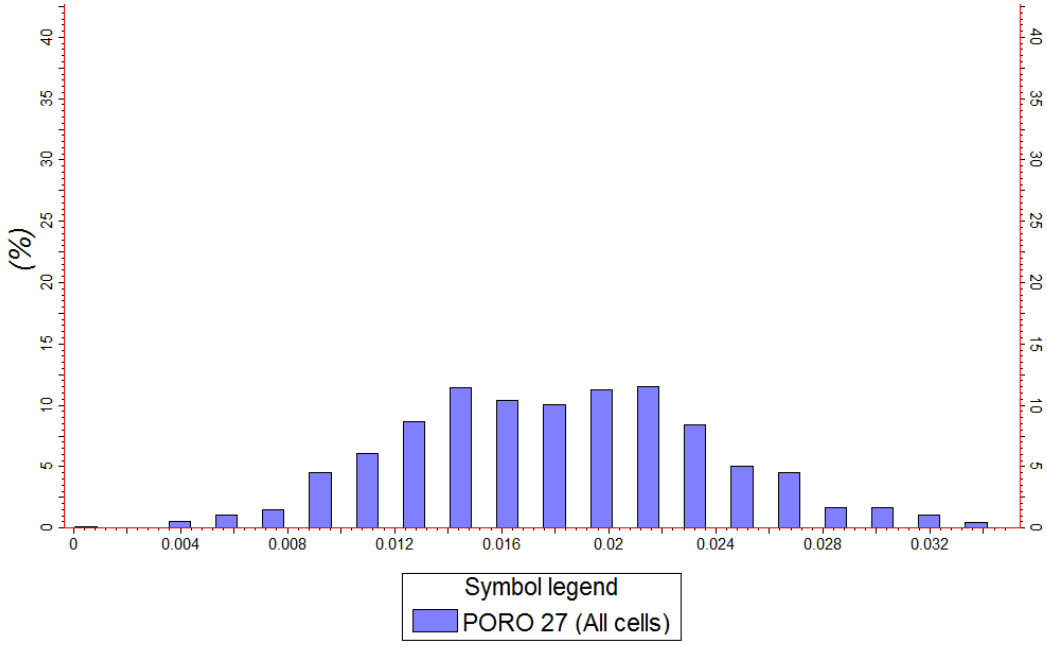

Figure C.21: Effective porosity histogram with 27x27 cells

	Mean	Std. Deviation
Effective porosity	0.0186	0.0058

Table C.8: Effective porosity with 27x27 cells statistics

2.2.2. Effective permeability in X direction

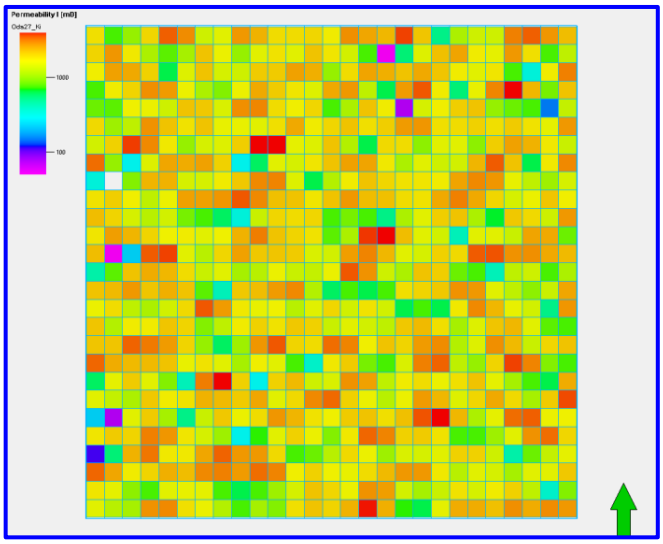

Figure C.22: Effective permeability in X direction with **Oda's** method (in mD)

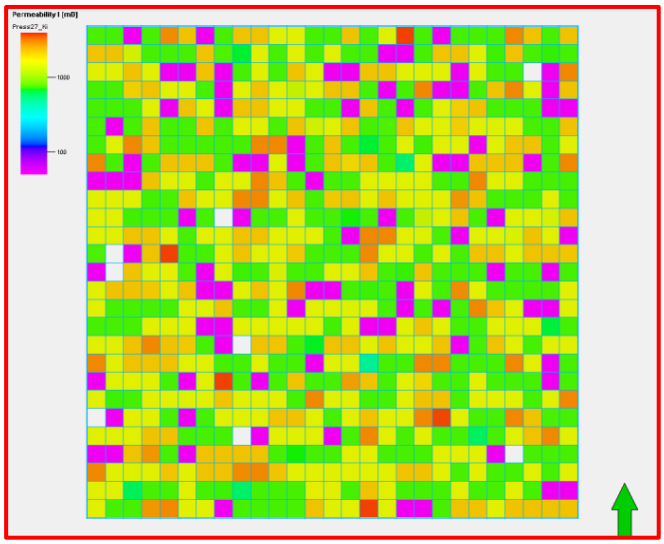

Figure C.23: Effective permeability in X direction with the **Linear Pressure** numerical method (in mD)

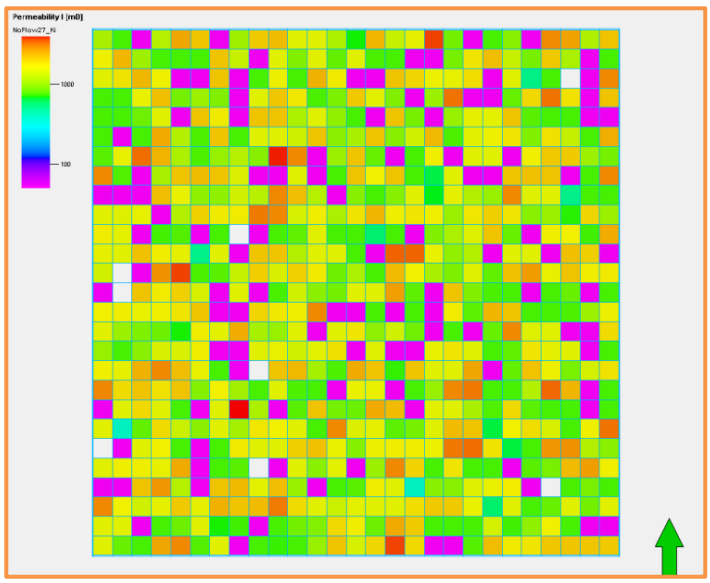

Figure C.24: Effective permeability in X direction with the **No Flow** numerical method (in mD)

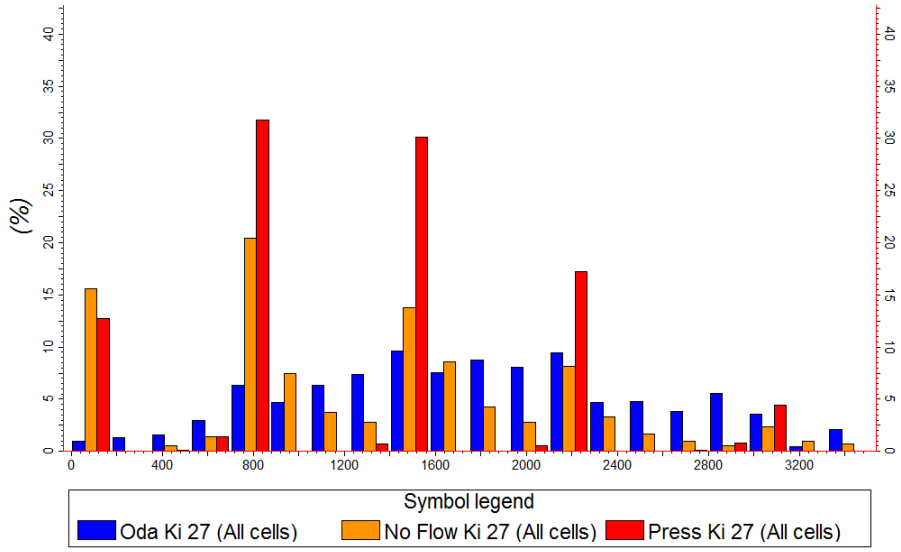

Figure C.25: Effective permeability in X direction histogram for **Oda**'s method (in blue), the **Linear Pressure** numerical method (in red) and the **No Flow** numerical method (in orange) (in mD)

	Mean	Std. Deviation
Effective perm. X using **Oda**'s method	1853.05 mD	822.47 mD
Effective perm. X using **Linear Pressure** numerical method	1283.17 mD	821.41 mD
Effective perm. X using **No Flow** numerical method	1286.68 mD	851.47 mD

Table C.9: Effective permeability in X direction with 9x9 cells statistics

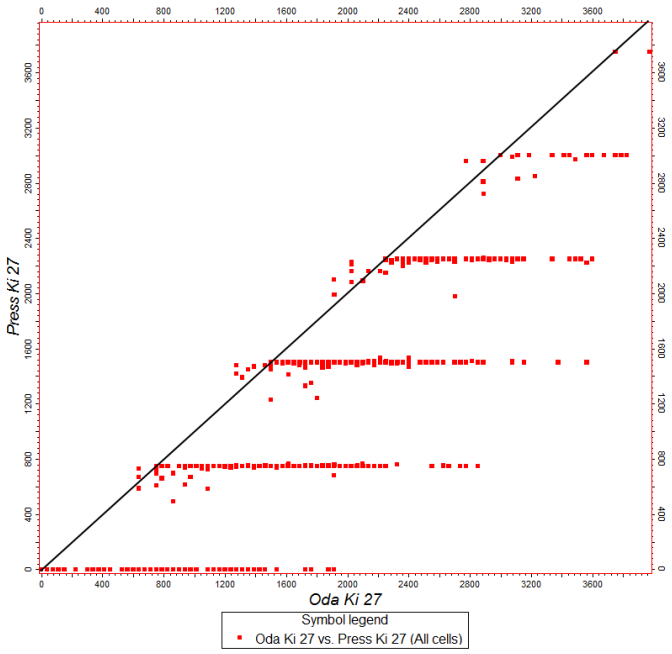

Figure C.26: Comparison of the effective permeability in X direction between the **Linear Pressure** numerical method and the **Oda**'s method(in mD)

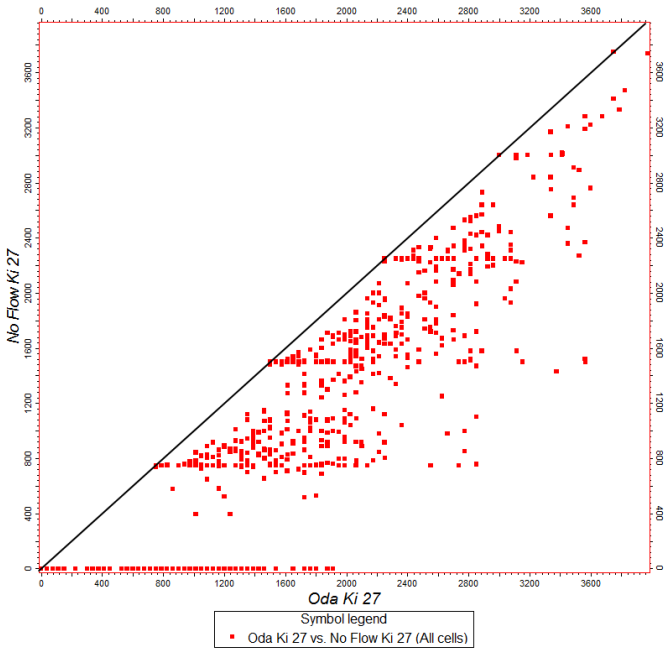

Figure C.27: Comparison of the effective permeability in X direction between the **No Flow** numerical method and the **Oda**'s method(in mD)

2.2.3. Effective permeability in Y direction

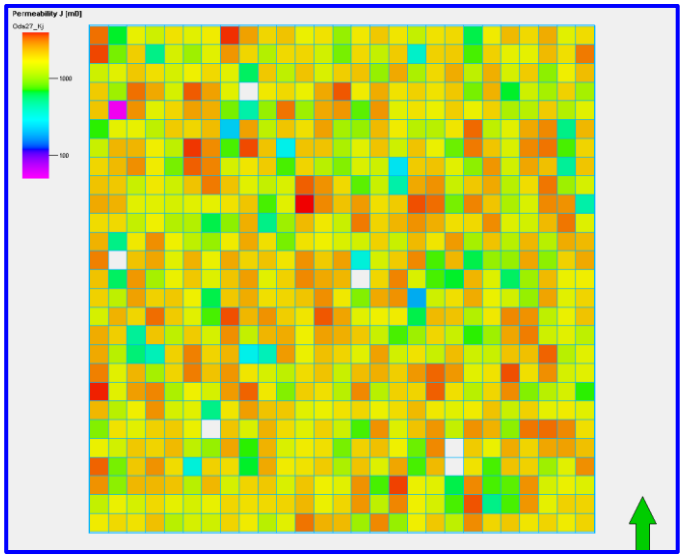

Figure C.28: Effective permeability in Y direction with **Oda's** method (in mD)

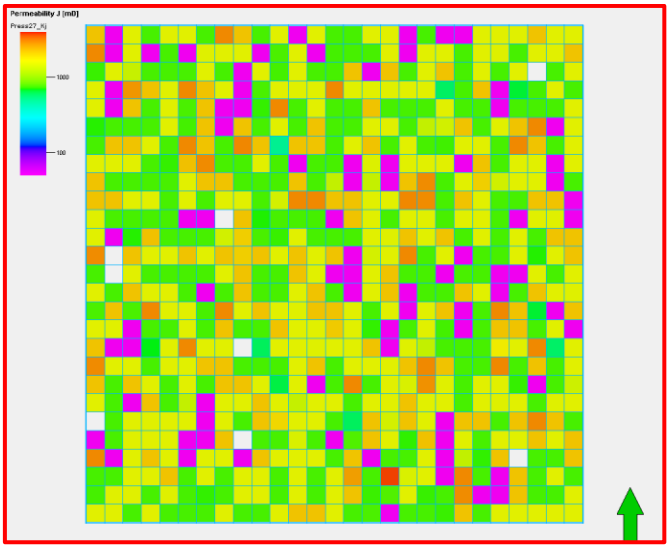

Figure C.29: Effective permeability in Y direction with the **Linear Pressure** numerical method (in mD)

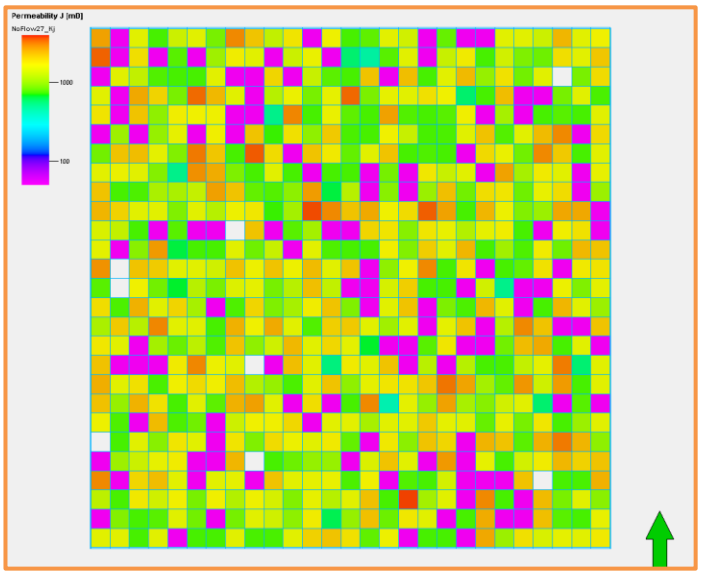

Figure C.30: Effective permeability in Y direction with the **No Flow** numerical method (in mD)

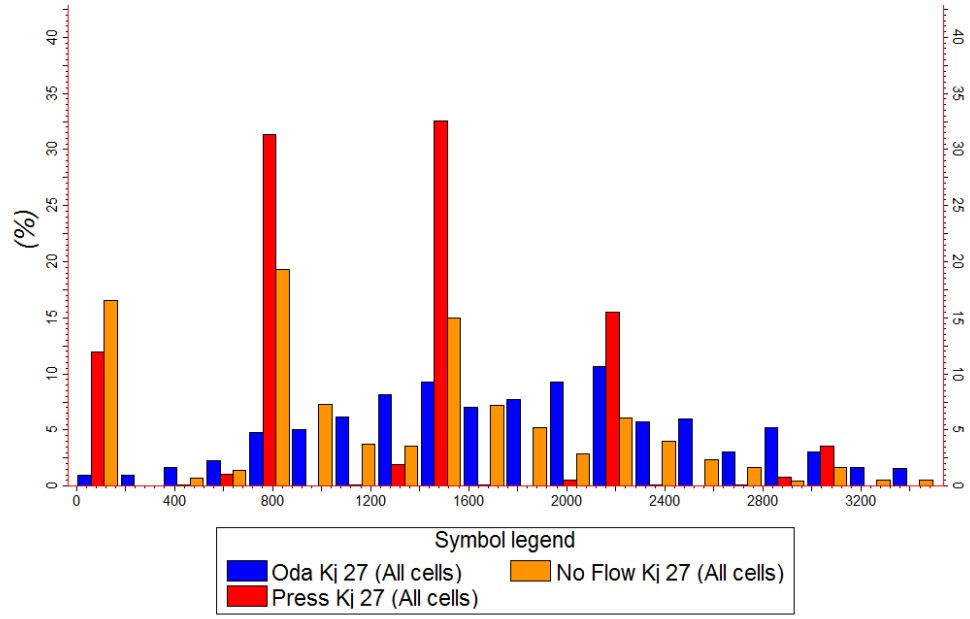

Figure C.31: Effective permeability in Y direction histogram for **Oda**'s method (in blue), the **Linear Pressure** numerical method (in red) and the **No Flow** numerical method (in orange) (in mD)

	Mean	Std. Deviation
Effective perm. Y using **Oda**'s method	1863.55 mD	771.37 mD
Effective perm. Y using **Linear Pressure** numerical method	1260.13 mD	771.34 mD
Effective perm. Y using **No Flow** numerical method	1260.53 mD	828.87 mD

Table C.10: Effective permeability in Y direction with 9x9 cells statistics

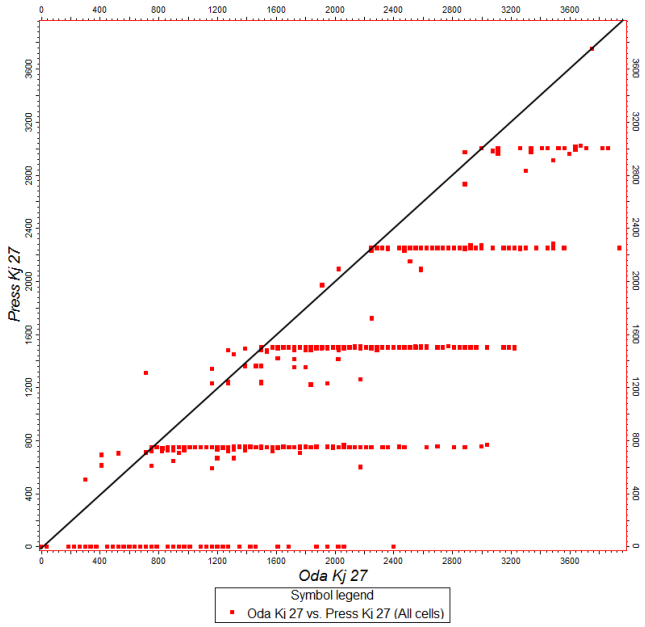

Figure C.32: Comparison of the effective permeability in Y direction between the **Linear Pressure** numerical method and the **Oda**'s method(in mD)

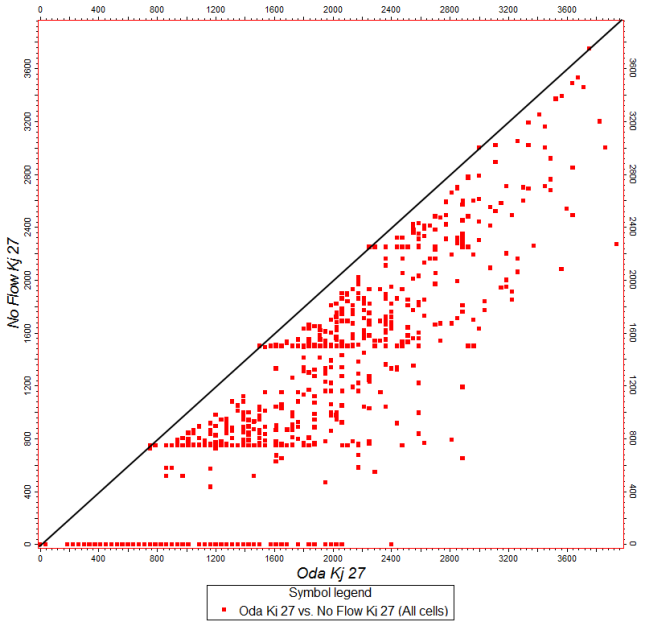

Figure C.33: Comparison of the effective permeability in Y direction between the **No Flow** numerical method and the **Oda**'s method(in mD)

3. Dynamic simulation

3.1. Model characteristics

Matrix rock properties	
Compressibility	4.93E-5 bar^{-1}
Permeability	0.0 mD

Table C.11: Matrix rock properties

	Oil properties	Water properties
Density	897 kg/m^3	1000 kg/m^3
Viscosity	1.4 cP	0.5 cP
Formation Volume Factor	1.0 rm^3/sm^3	1.0 rm^3/sm^3
Compressibility	4.0E-5 bar^{-1}	4.0E-5 bar^{-1}

Table C.12: Fluid properties

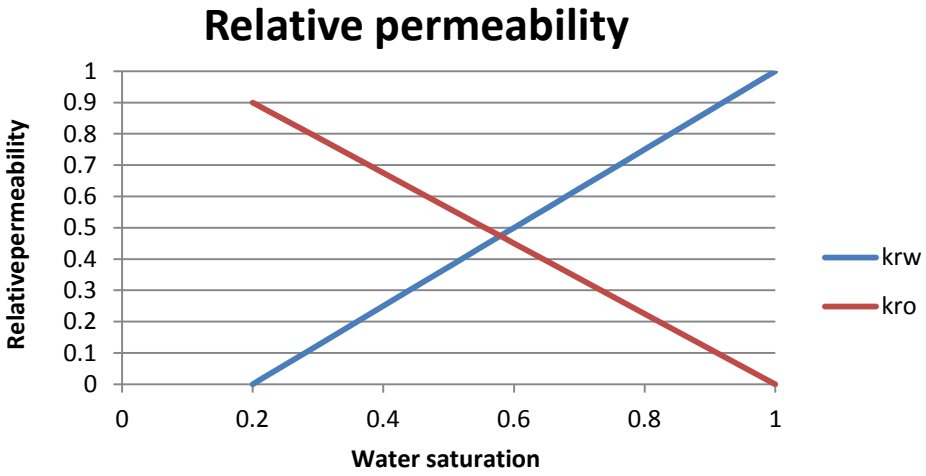

Figure C.34: Relative permeability of water (in blue) and oil (in red) against Water Saturation

Initial Conditions	
Initial Water Saturation	0.2
Initial Pressure	200 bars
Initial oil volume in place	77 000 sm^3

Table C.13: Initial conditions

Production well characteristics	
Number of producer	1
Well diameter	0.2 m
Constant bottom hole pressure	1 bar

Table C.14: Production well characteristics

Injection well characteristics	
Number of injectors	4
Well diameter	0.2 m
Constant injection pressure	250 bars
Fluid injected	Water

Table C.15: Injection well characteristics

Well zone properties	
Effective porosity	0.02
Effective permeability	2700 mD
Size in X direction	80 m
Size in Y direction	80 m
Size in Z direction	10 m

Table C.16: Size and effective properties of the zones around the wells

Simulation properties	
Production duration	7 days

Table C.17: Simulation characteristics

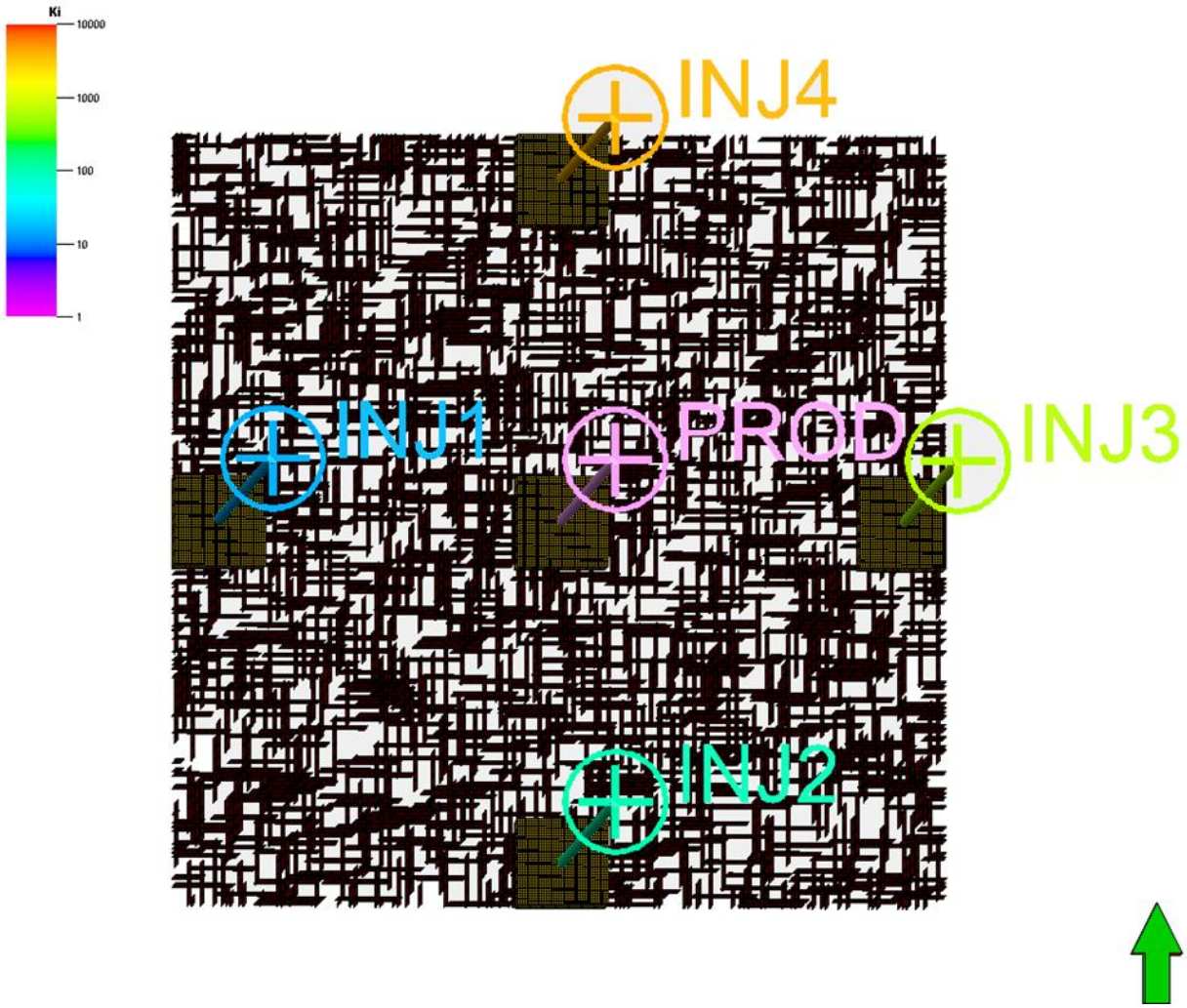

Figure C.35: Well and well zone locations

3.2. Oil production rate and production cumulative

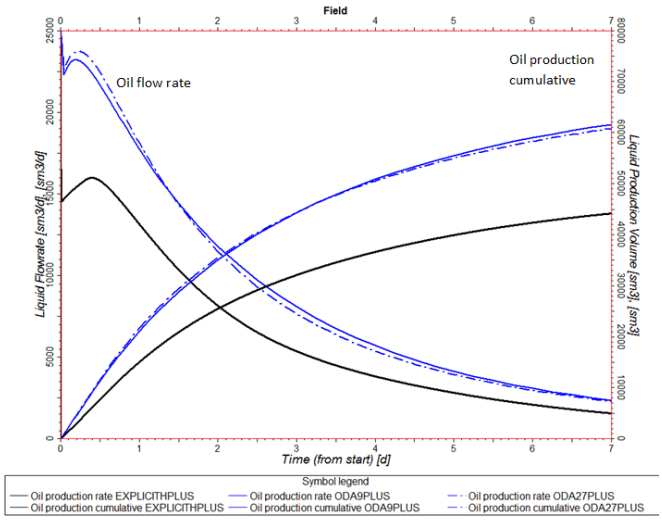

Figure C.36: Oil flow rate and oil production cumulative with the Explicit simulation (black line) and simulations using **Oda**'s method on a grid with 9x9 cells (solid blue line) and 27x27 cells (dashed blue line)

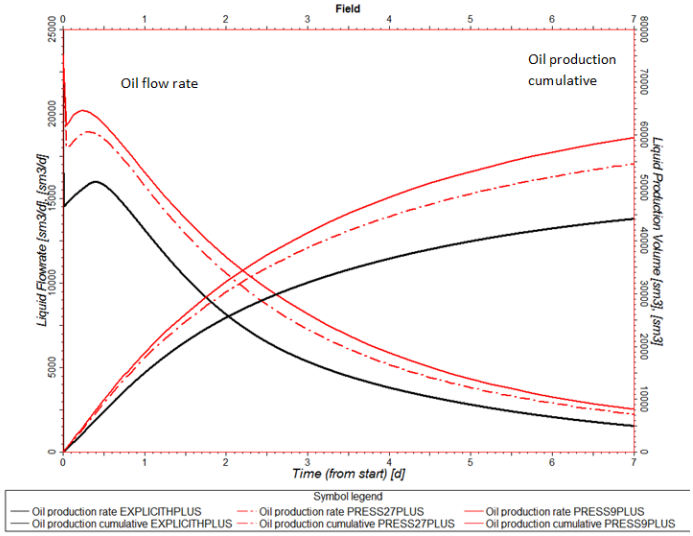

Figure C.37: Oil flow rate and oil production cumulative with the Explicit simulation (black line) and simulations using the **Linear Pressure** numerical method on a grid with 9x9 cells (solid red line) and 27x27 cells (dashed red line)

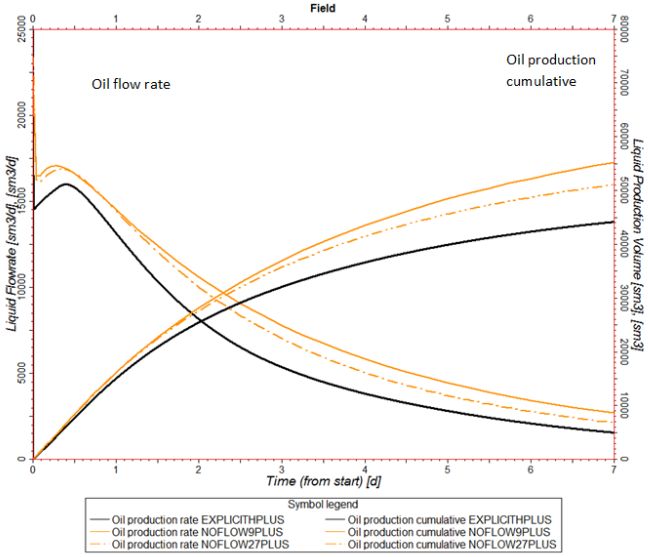

Figure C.38: Oil flow rate and oil production cumulative with the Explicit simulation (black line) and simulations using the **No Flow** numerical method on a grid with 9x9 cells (solid orange line) and 27x27 cells (dashed orange line)

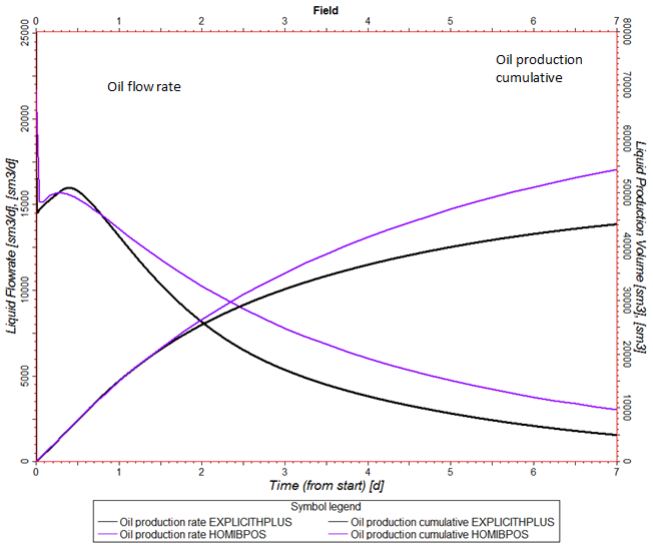

Figure C.39: Oil flow rate and oil production cumulative with the Explicit simulation (black line) and simulations using the **IBPOS** numerical method on a grid with 9x9 cells (solid purple line)

3.3. Oil and water production rate

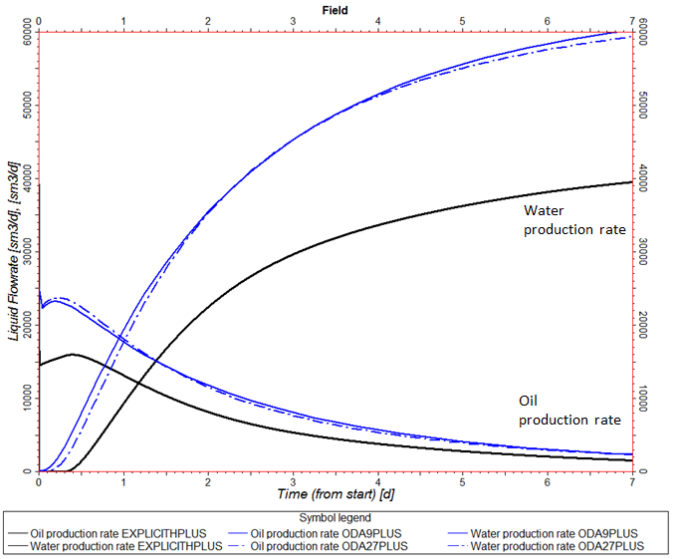

Figure C.40: Oil and water production rate with the Explicit simulation (black line) and simulations using **Oda**'s method on a grid with 9x9 cells (solid blue line) and 27x27 cells (dashed blue line)

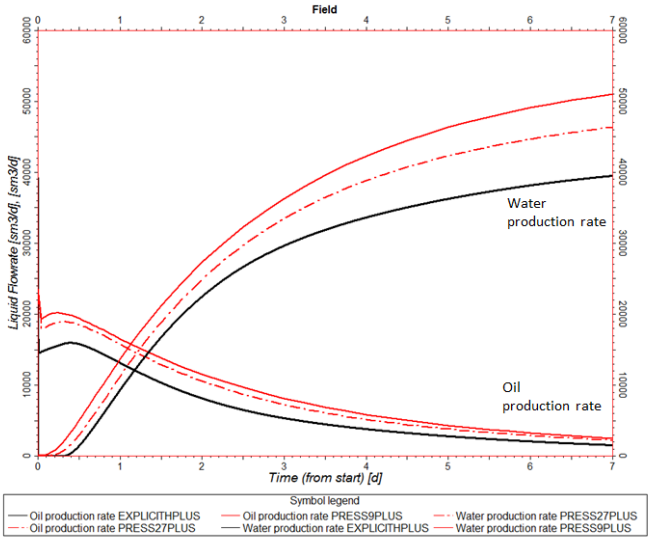

Figure C.41: Oil and water production rate with the Explicit simulation (black line) and simulations using the **Linear Pressure** numerical method on a grid with 9x9 cells (solid red line) and 27x27 cells (dashed red line)

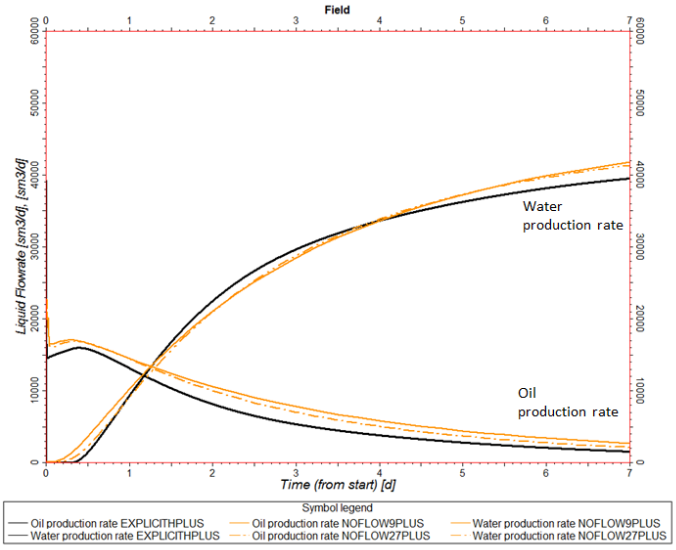

Figure C.42: Oil and water production rate with the Explicit simulation (black line) and simulations using the **No Flow** numerical method on a grid with 9x9 cells (solid orange line) and 27x27 cells (dashed orange line)

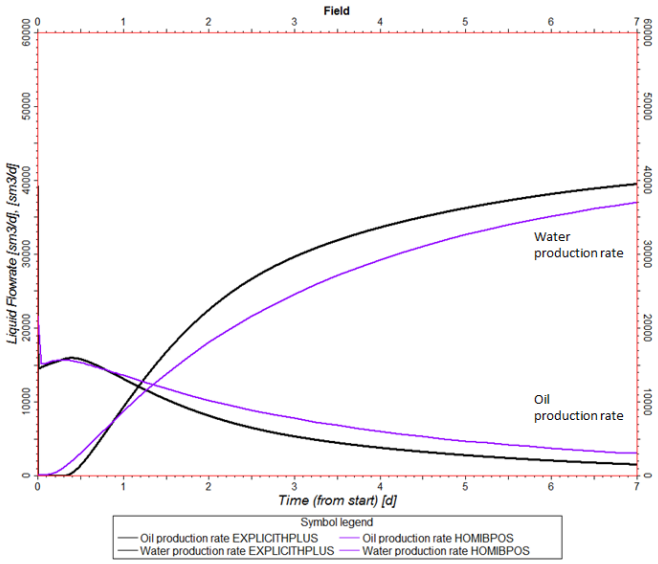

Figure C.43: Oil and water production rate with the Explicit simulation (black line) and simulations using the **IBPOS** numerical method on a grid with 9x9 cells (solid purple line)

3.4. Buckley-Leverett analysis

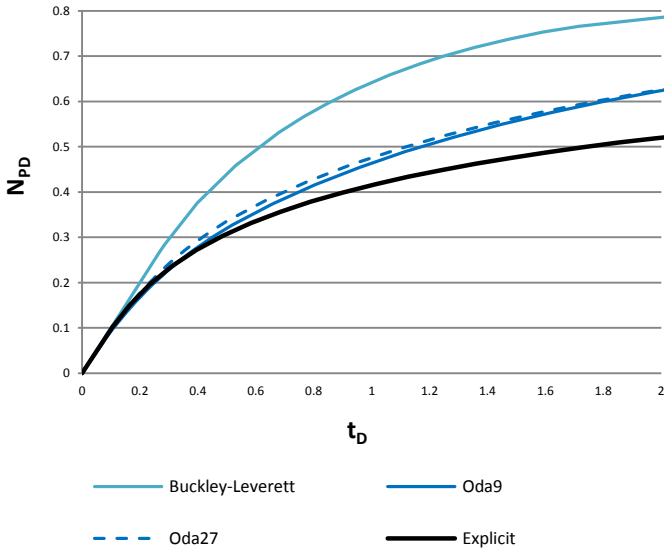

Figure C.44: Dimensionless oil production cumulative (N_{PD}) against dimensionless water injection cumulative (t_D) with the Buckley-Leverett theory (green line), the Explicit simulation (black line) and simulations using **Oda**'s method on a grid with 9x9 cells (solid blue line) and 27x27 cells (dashed blue line)

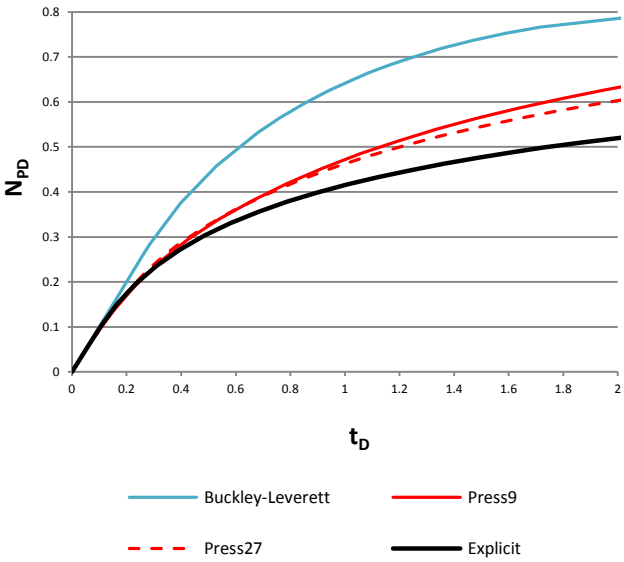

Figure C.45: Dimensionless oil production cumulative (N_{PD}) against dimensionless water injection cumulative (t_D) with the Buckley-Leverett theory (green line), the Explicit simulation (black line) and simulations using the **Linear Pressure** numerical method on a grid with 9x9 cells (solid red line) and 27x27 cells (dashed red line)

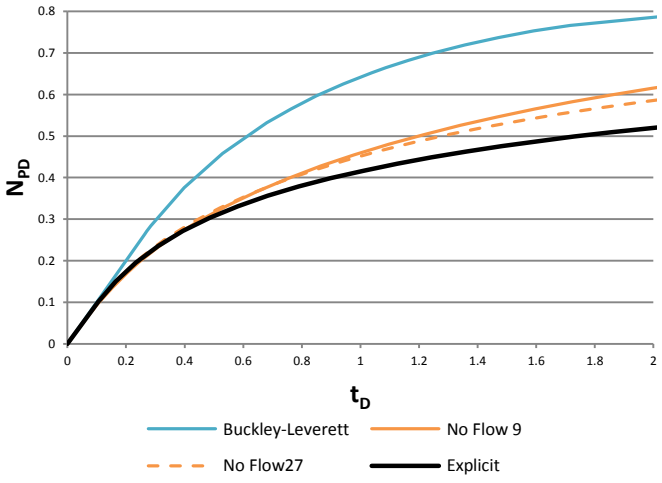

Figure C.46: Dimensionless oil production cumulative (N_{PD}) against dimensionless water injection cumulative (t_D) with the Buckley-Leverett theory (green line), the Explicit simulation (black line) and simulations using the **No Flow** numerical method on a grid with 9x9 cells (solid orange line) and 27x27 cells (dashed orange line)

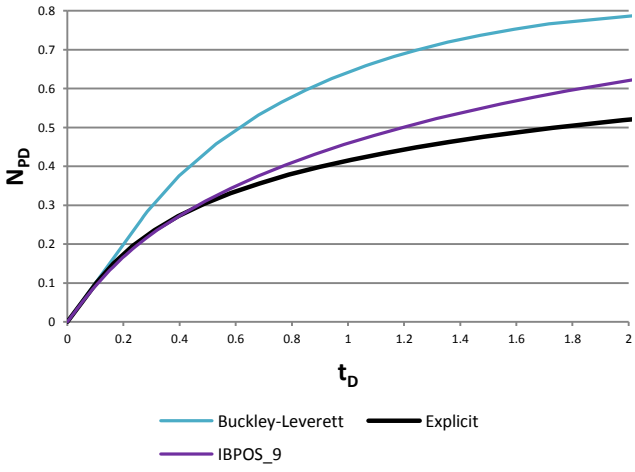

Figure C.47: Dimensionless oil production cumulative (N_{PD}) against dimensionless water injection cumulative (t_D) with the Buckley-Leverett theory (green line), the Explicit simulation (black line) and simulations using the **IBPOS** numerical method on a grid with 9x9 cells (solid purple line)

APPENDIX D: Heterogeneous model properties

1. Model characteristics

Size of the model	
In X direction:	720 m
In Y direction:	720 m
In Z direction:	10 m

Table D.1: Model dimensions

Horizontal fracture set		Vertical fracture set	
Constant aperture	0.1 m	Constant aperture	0.1 m
Constant permeability	1.00E+05 mD	Constant permeability	2.00E+05 mD
Shape	Rectangular	Shape	Rectangular
Constant length	40 m	Constant length	80 m
Constant width	10 m	Constant width	10 m
Constant fracture density for facies 1	0.075 m^2/m^3	Constant fracture density for facies 1	0.088 m^2/m^3
Constant fracture density for facies 2	0.044 m^2/m^3	Constant fracture density for facies 2	0.15 m^2/m^3

Table D.2: Fracture set properties

Facies cell dimensions	
In X direction:	80 m
In Y direction:	80 m
In Z direction:	10 m
Total number of cells	9x9 = 81
Percentage of cells with facies 1	53 %
Percentage of cells with facies 2	47 %

Table D.3: Fracture set properties

2	1	1	1	2	1	2	1	2
1	1	1	2	1	2	1	2	2
1	1	2	1	1	1	2	1	2
1	2	1	1	2	1	2	2	2
1	1	2	2	2	1	1	2	2
1	2	2	2	2	2	1	1	2
2	2	2	1	2	1	1	2	1
2	1	2	1	2	2	1	2	1
1	1	2	1	1	1	1	1	1

Figure D.1: Facies map

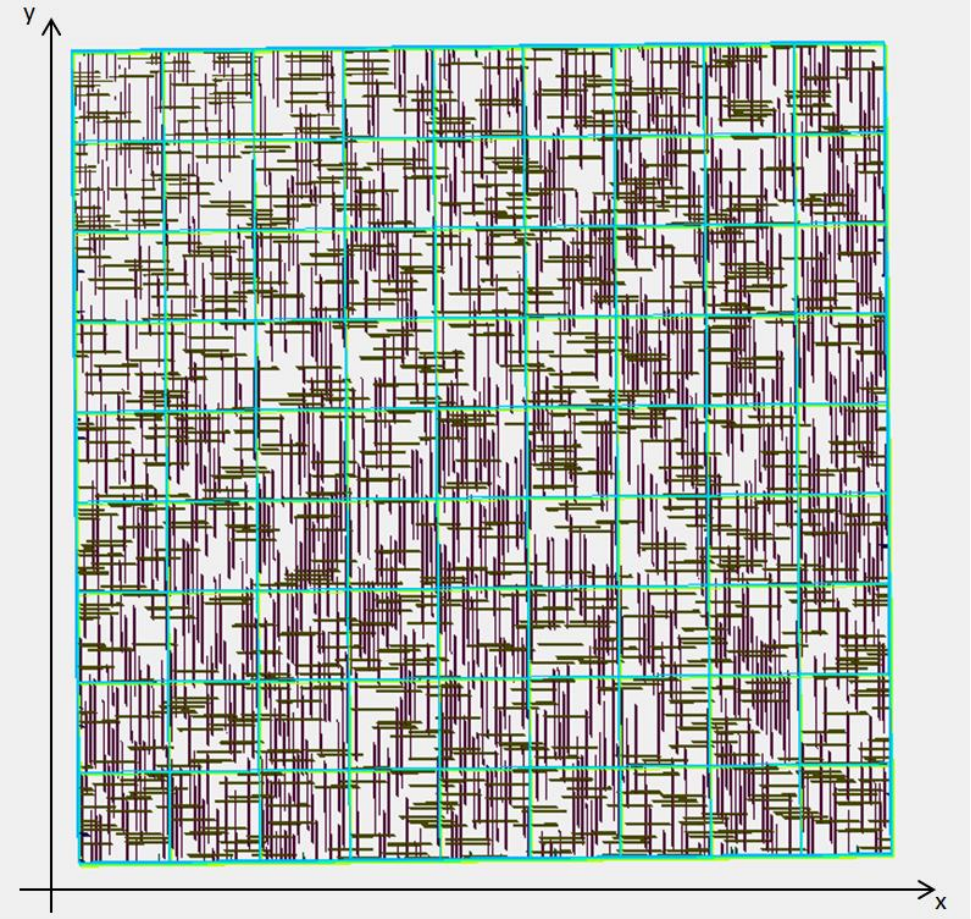

Figure D.2: Discrete Fracture Network

2. Effective properties

2.1. Grid with 9x9 cells

Grid properties	
Cell size in X direction:	80 m
Cell size in Y direction:	80 m
Cell size in Z direction:	10 m
Total number of cells:	9x9 = 81

Table D.4: Grid properties

2.1.1. Effective porosity

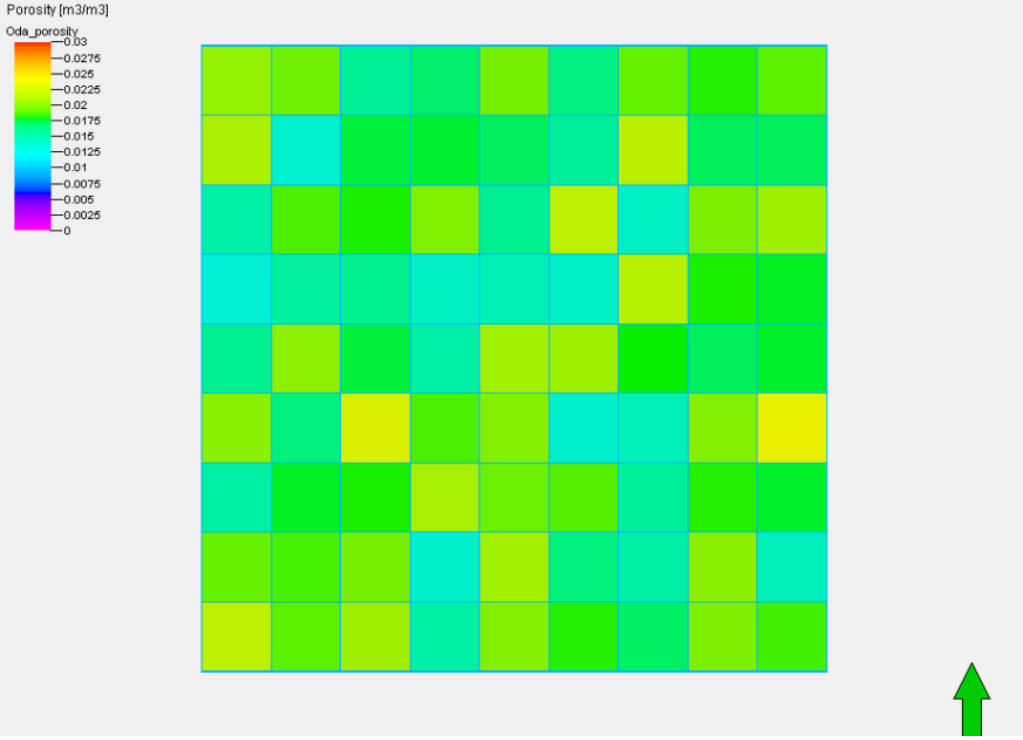

Figure D.3: Effective porosity with 9x9 cells

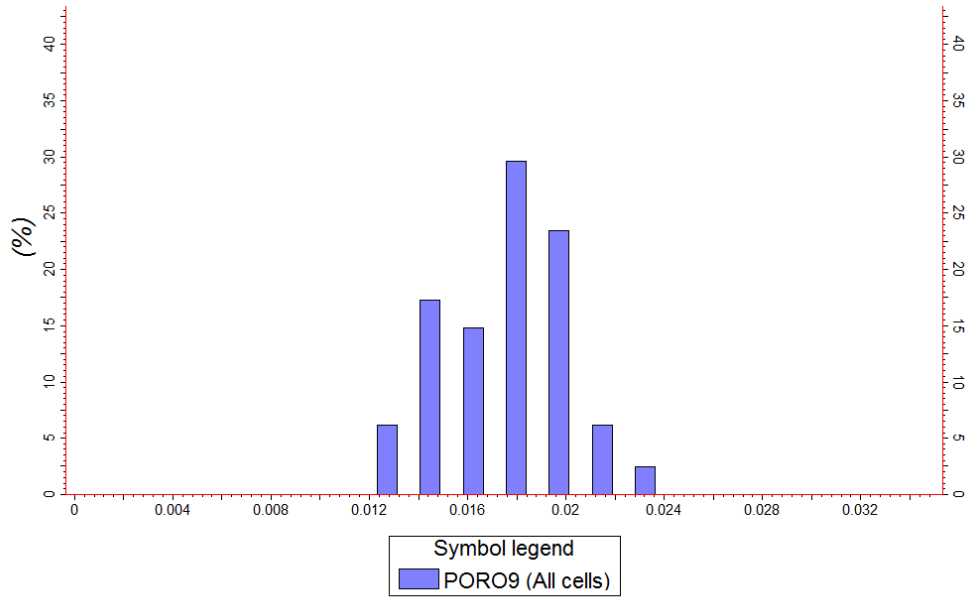

Figure D.4: Effective porosity histogram with 9x9 cells

	Mean	Std. Deviation
Effective porosity	0.0179	0.0024

Table D.5: Effective porosity with 9x9 cells statistics

2.1.2. Effective permeability in X direction

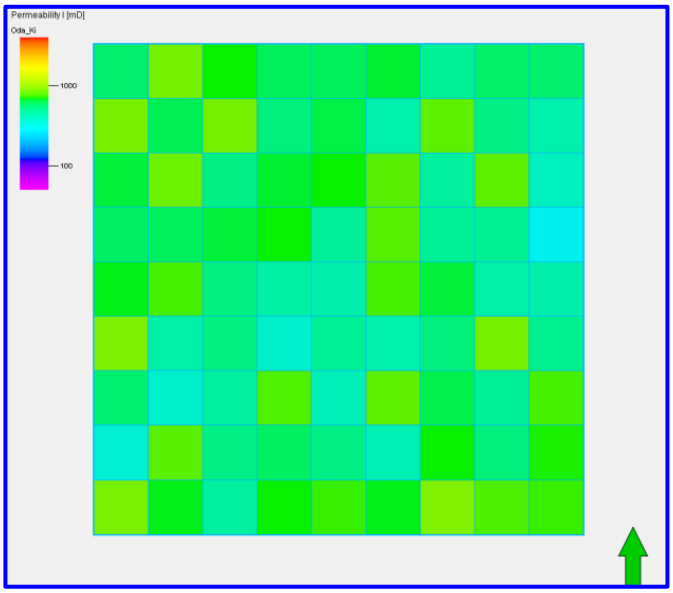

Figure D.5: Effective permeability in X direction with **Oda's** method (in mD)

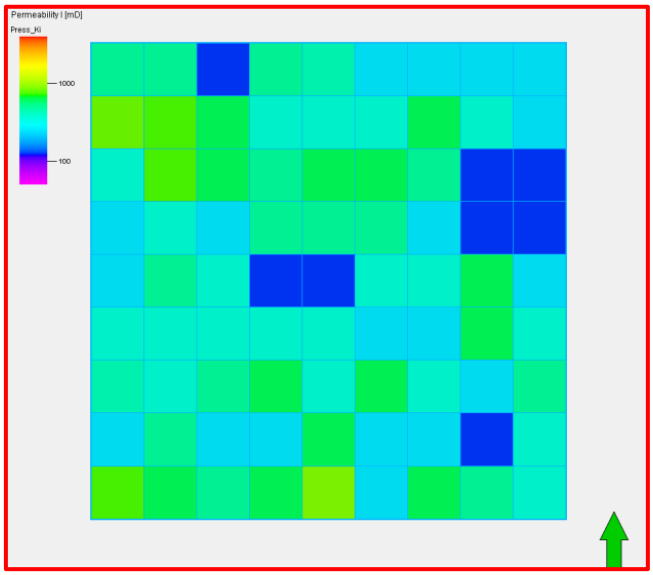

Figure D.6: Effective permeability in X direction with the **Linear Pressure** numerical method (in mD)

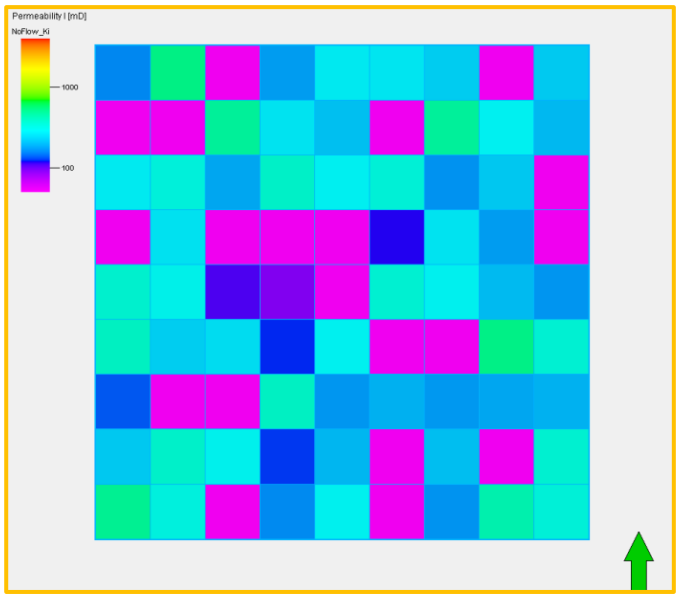

Figure D.7: Effective permeability in X direction with the **No Flow** numerical method (in mD)

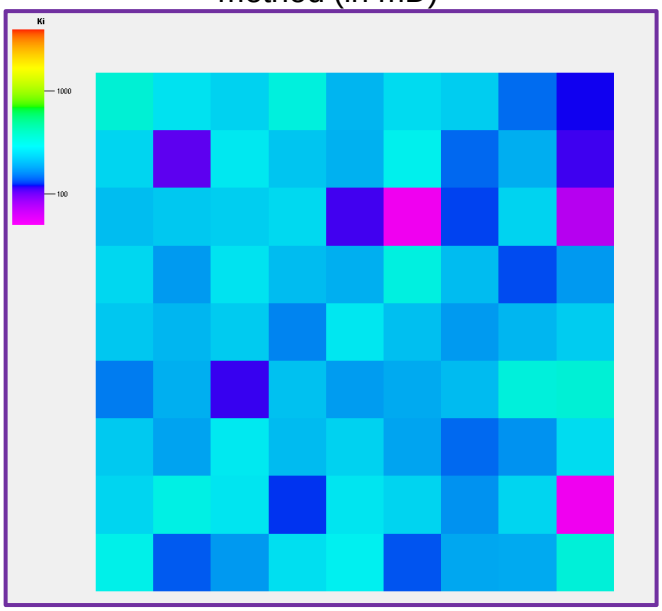

Figure D.8: Effective permeability in X direction with the **IBPOS** numerical method (in mD)

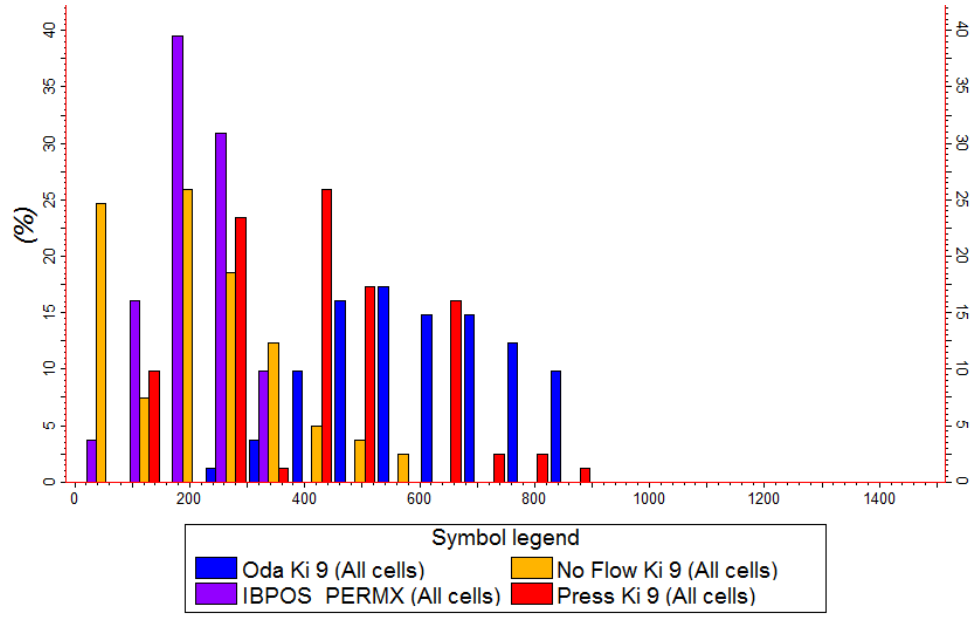

Figure D.9: Effective permeability in X direction histogram for **Oda**'s method (in blue), the **Linear Pressure** numerical method (in red), the **No Flow** numerical method (in orange) and the **IBPOS** numerical method (in purple) (in mD)

	Mean	Std. Deviation
Effective perm. X using **Oda**'s method	610.65 mD	146.56 mD
Effective perm. X using **Linear Pressure** numerical method	409.58 mD	180.22 mD
Effective perm. X using **No Flow** numerical method	201.23 mD	149.37 mD
Effective perm. X using **IBPOS** numerical method	206.85 mD	69.86 mD

Table D.6: Effective permeability in X direction with 9x9 cells statistics

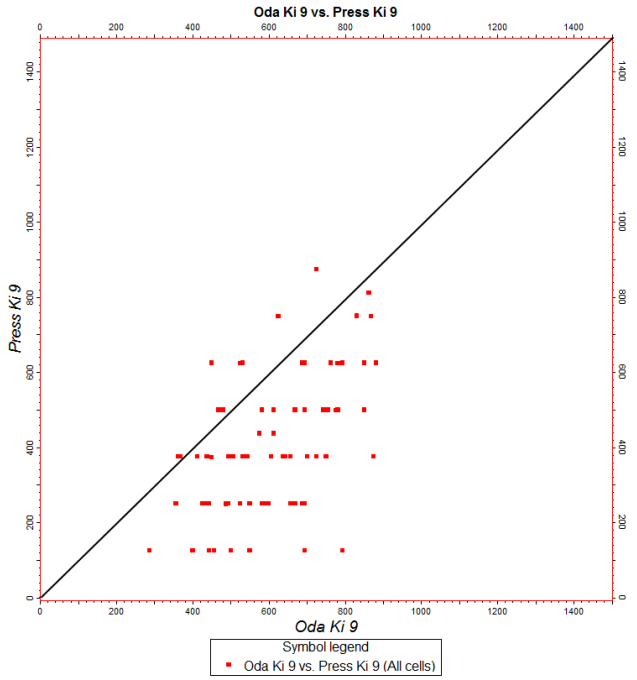

Figure D.10: Comparison of the effective permeability in X direction between the **Linear Pressure** numerical method and the **Oda**'s method (in mD)

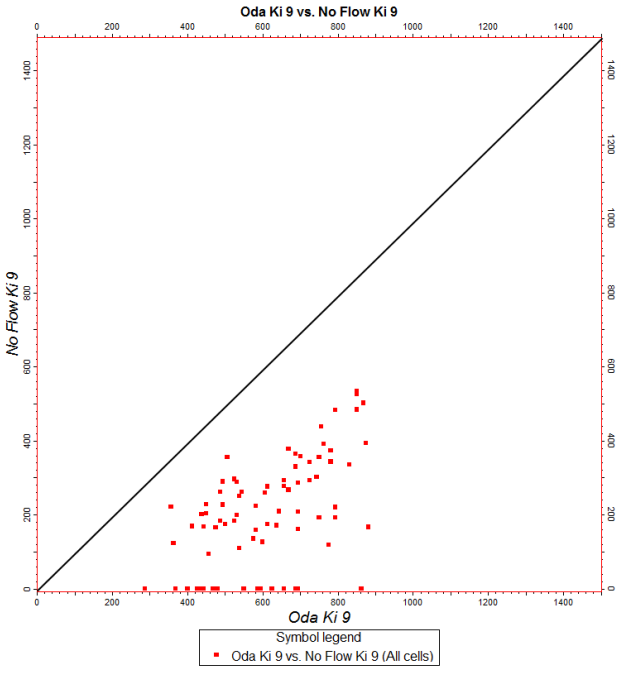

Figure D.11: Comparison of the effective permeability in X direction between the **No Flow** numerical method and the **Oda**'s method (in mD)

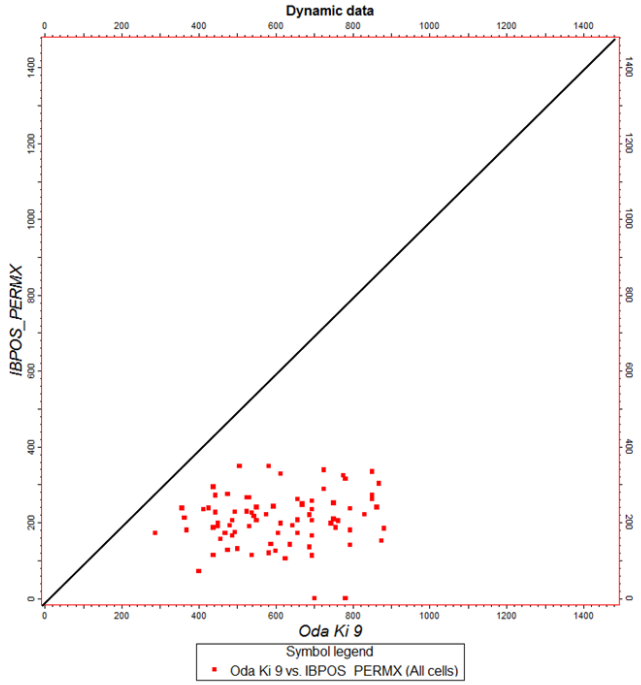

Figure D.12: Comparison of the effective permeability in X direction between the **IBPOS** numerical method and the **Oda**'s method (in mD)

2.1.3. Effective permeability in Y direction

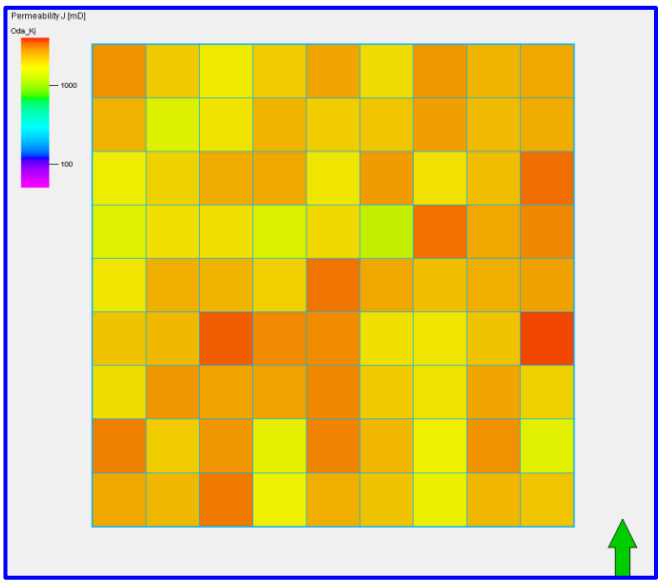

Figure D.13: Effective permeability in Y direction with **Oda's** method (in mD)

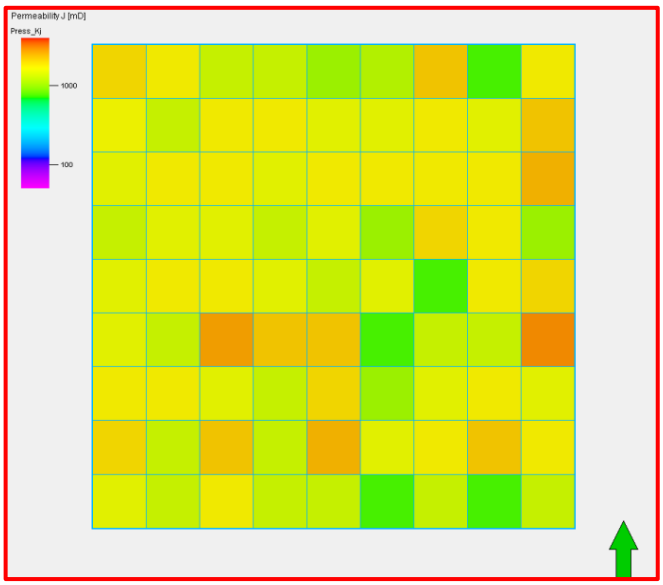

Figure D.14: Effective permeability in Y direction with the **Linear Pressure** numerical method (in mD)

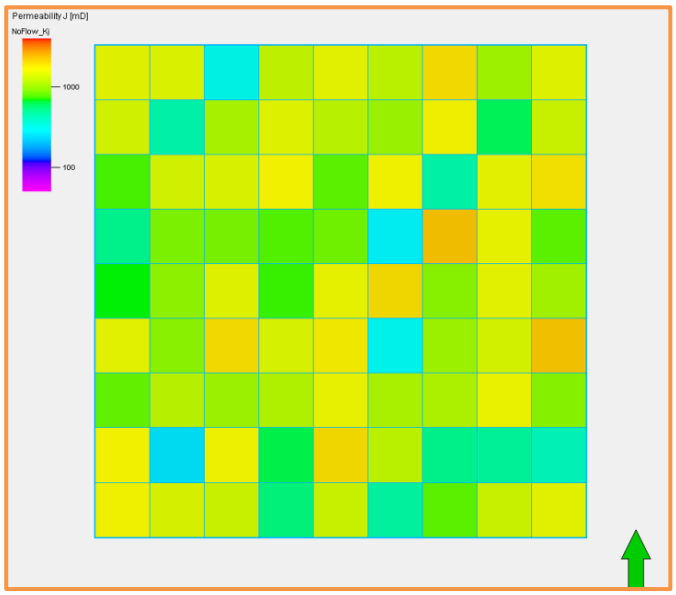

Figure D.15: Effective permeability in Y direction with the **No Flow** numerical method (in mD)

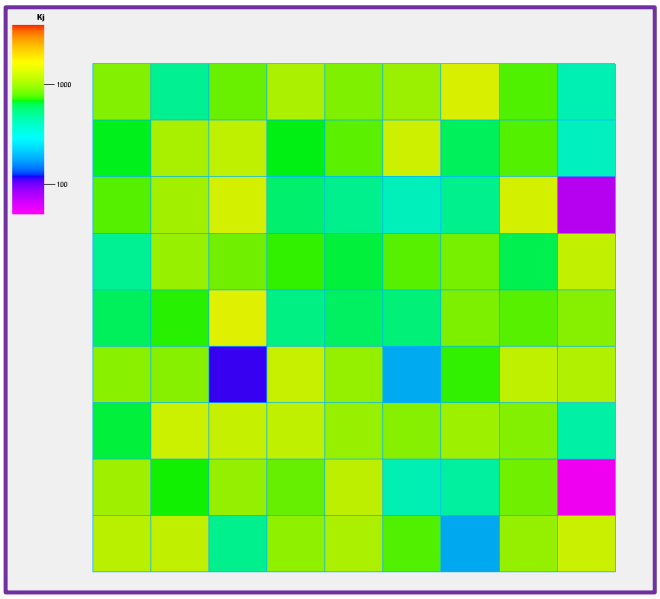

Figure D.16: Effective permeability in Y direction with the **IBPOS** numerical method (in mD)

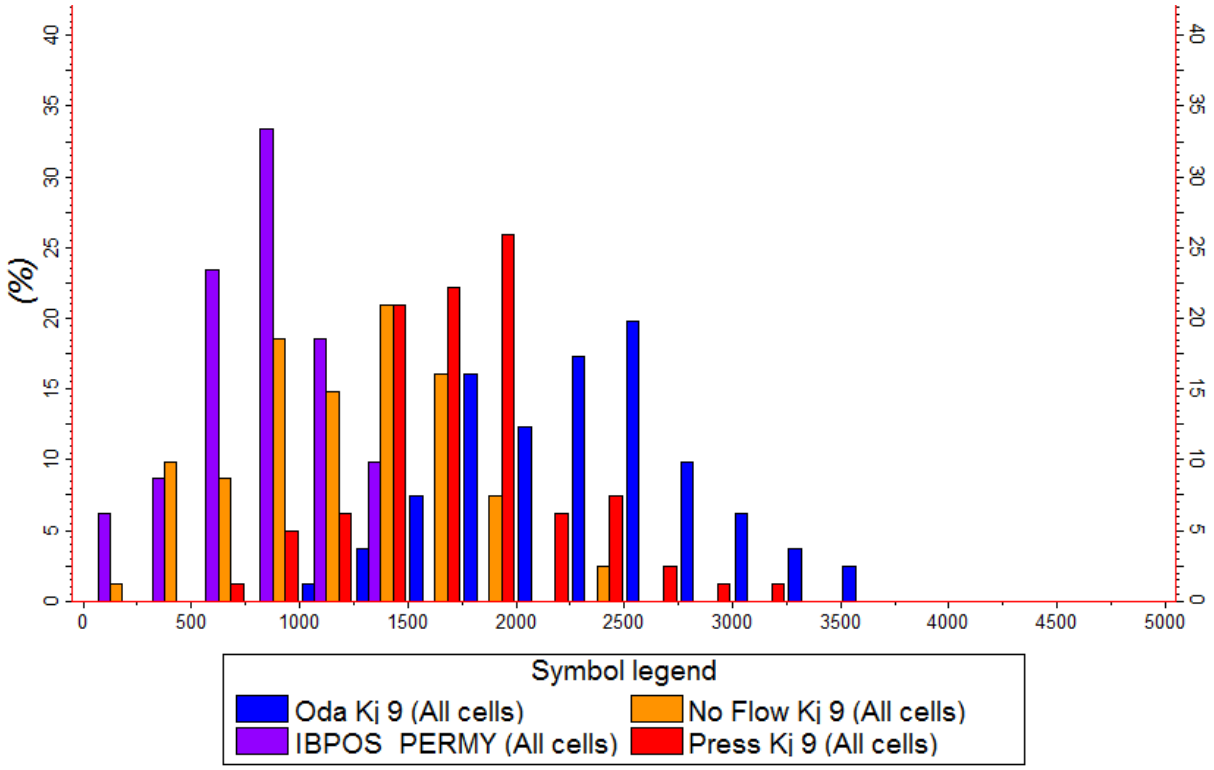

Figure D.17: Effective permeability in Y direction histogram for **Oda**'s method (in blue), the **Linear Pressure** numerical method (in red), the **No Flow** numerical method (in orange) and the **IBPOS** numerical method (in purple) (in mD)

	Mean	Std. Deviation
Effective perm. Y using **Oda**'s method	2359.57 mD	533.6 mD
Effective perm. Y using **Linear Pressure** numerical method	1583.19 mD	450.37 mD
Effective perm. Y using **No Flow** numerical method	1161.41 mD	484.47 mD
Effective perm. Y using **IBPOS** numerical method	823.48 mD	326.78 mD

Table D.7: Effective permeability in Y direction with 9x9 cells statistics

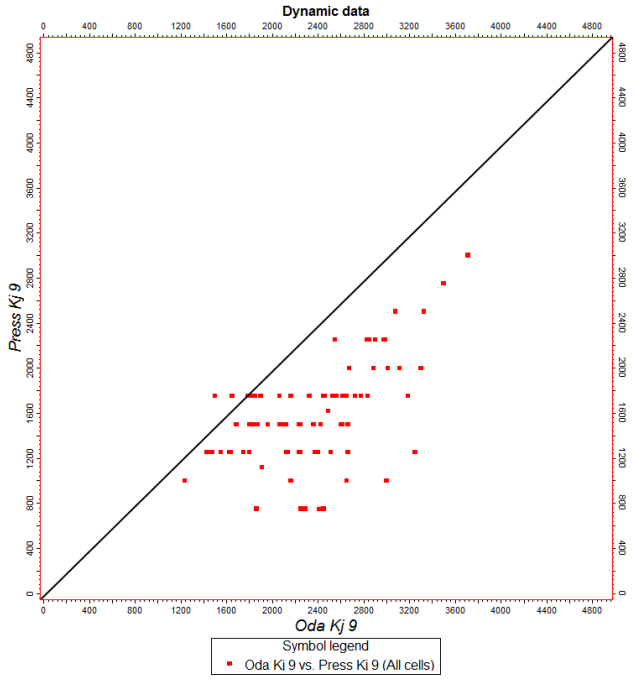

Figure D.18: Comparison of the effective permeability in Y direction between the **Linear Pressure** numerical method and the **Oda**'s method(in mD)

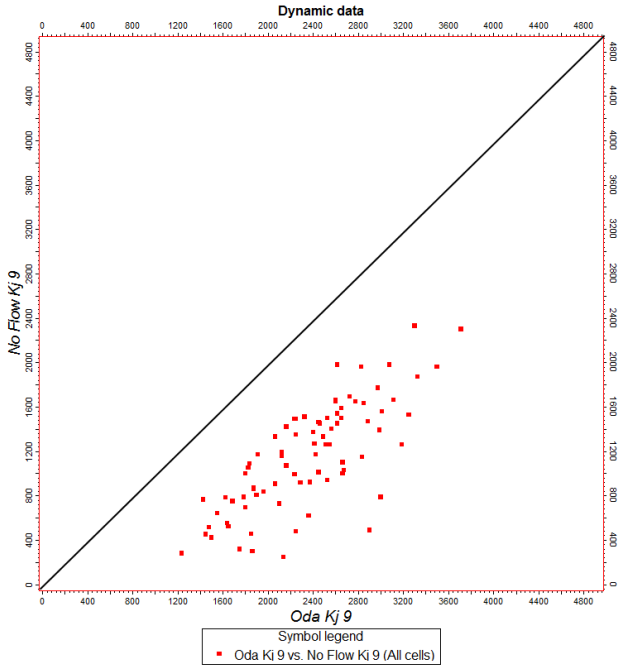

Figure D.19: Comparison of the effective permeability in Y direction between the **No Flow** numerical method and the **Oda**'s method(in mD)

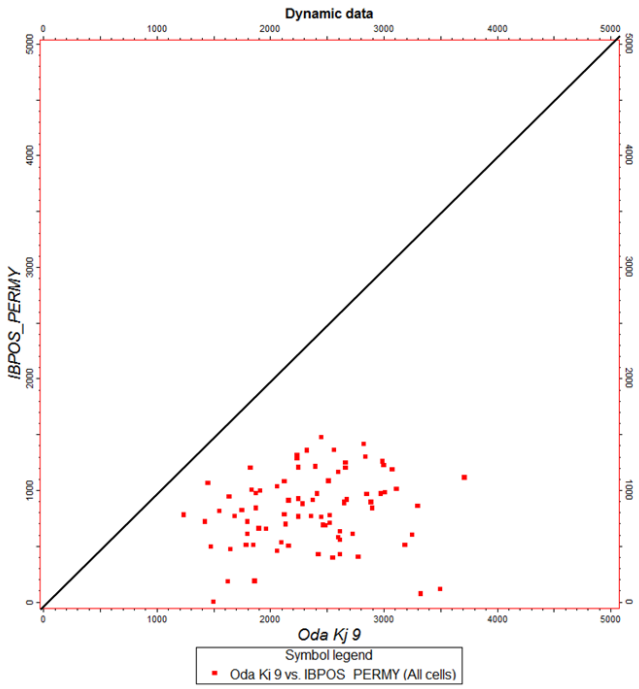

Figure D.20: Comparison of the effective permeability in Y direction between the **IBPOS** numerical method and the **Oda**'s method(in mD)

2.2. Grid with 27x27 cells

Grid properties	
Cell size in X direction:	26.7 m
Cell size in Y direction:	26.7 m
Cell size in Z direction:	10 m
Total number of cells:	27x27 = 729

Table D.8: Grid properties

2.2.1. Effective porosity

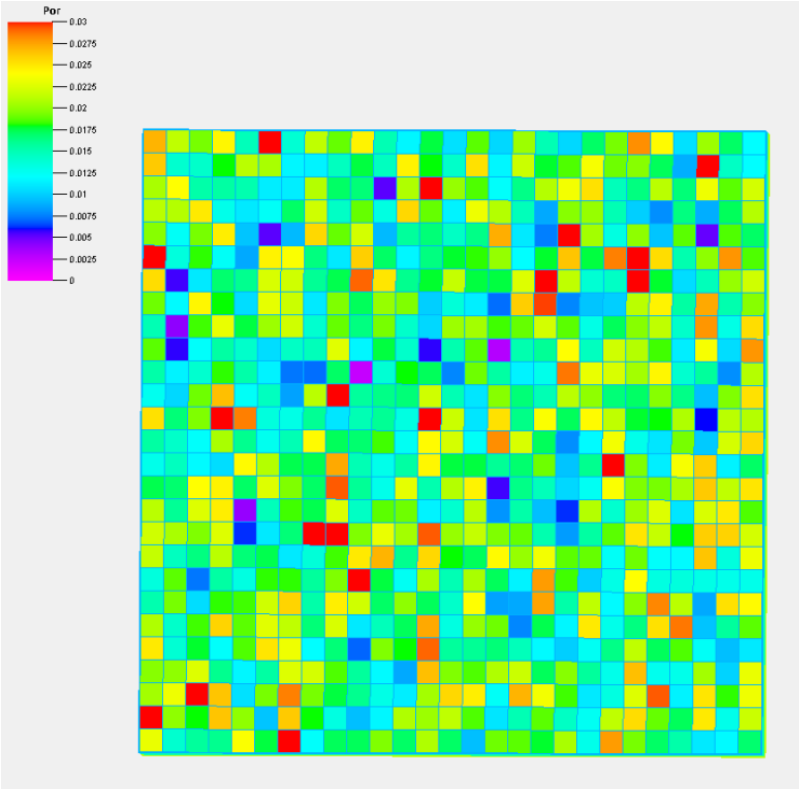

Figure D.21: Effective porosity with 27x27 cells

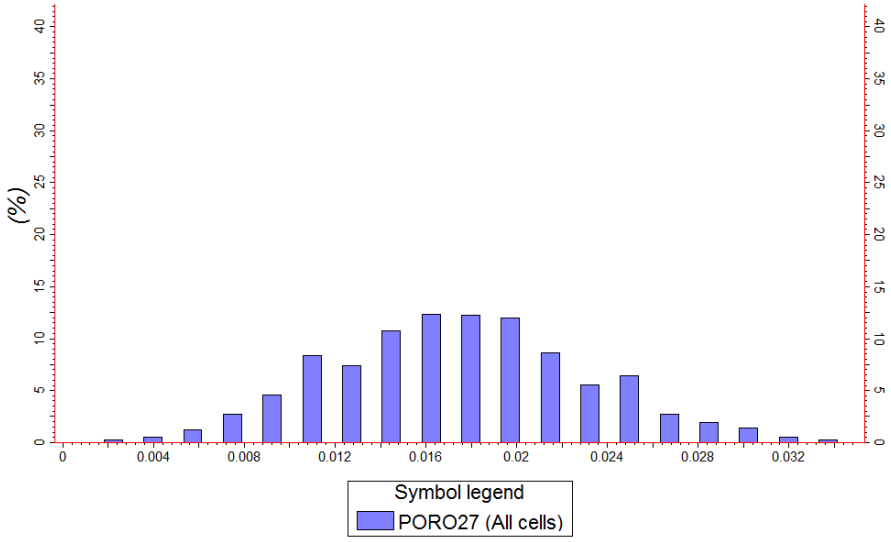

Figure D.22: Effective porosity histogram with 27x27 cells

	Mean	Std. Deviation
Effective porosity	0.0179	0.0057

Table D.9: Effective porosity with 27x27 cells statistics

2.2.2. Effective permeability in X direction

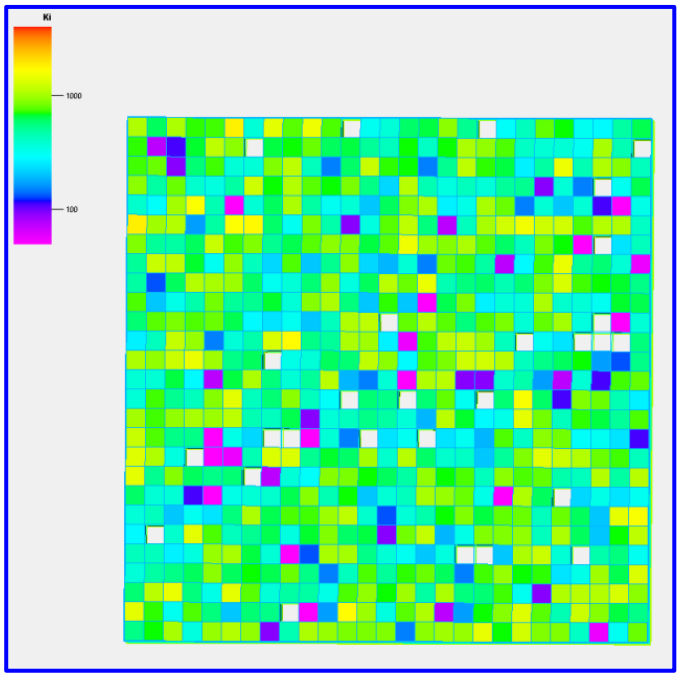

Figure D.23: Effective permeability in X direction with **Oda's** method (in mD)

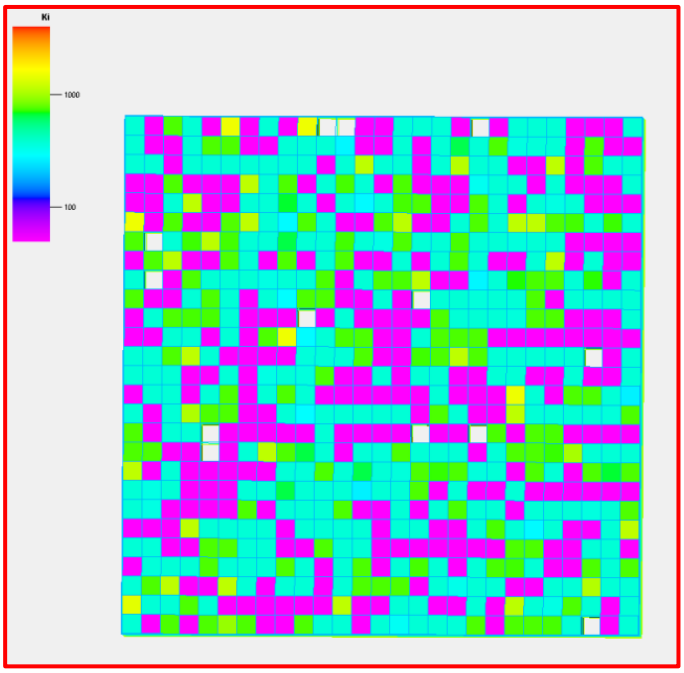

Figure D.24: Effective permeability in X direction with the **Linear Pressure** numerical method (in mD)

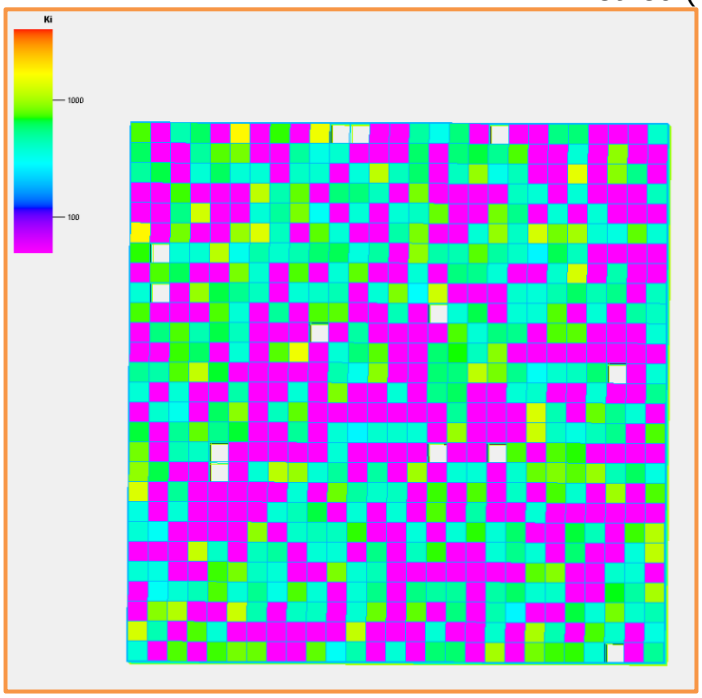

Figure D.25: Effective permeability in X direction with the **No Flow** numerical method (in mD)

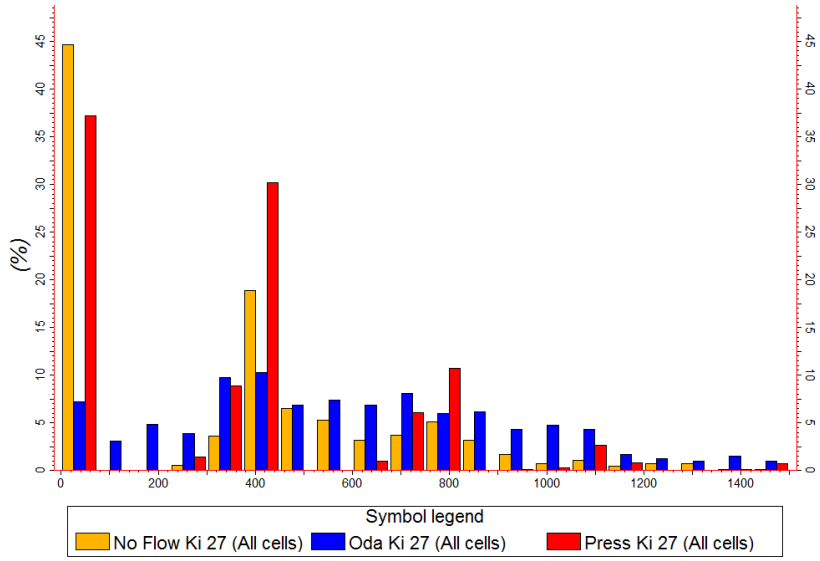

Figure D.26: Effective permeability in X direction histogram for **Oda**'s method (in blue), the **Linear Pressure** numerical method (in red) and the **No Flow** numerical method (in orange) (in mD)

	Mean	Std. Deviation
Effective perm. X using **Oda**'s method	610.63 mD	366.66 mD
Effective perm. X using **Linear Pressure** numerical method	335.43 mD	324.72 mD
Effective perm. X using **No Flow** numerical method	328.07 mD	349.86 mD

Table D.10: Effective permeability in X direction with 9x9 cells statistics

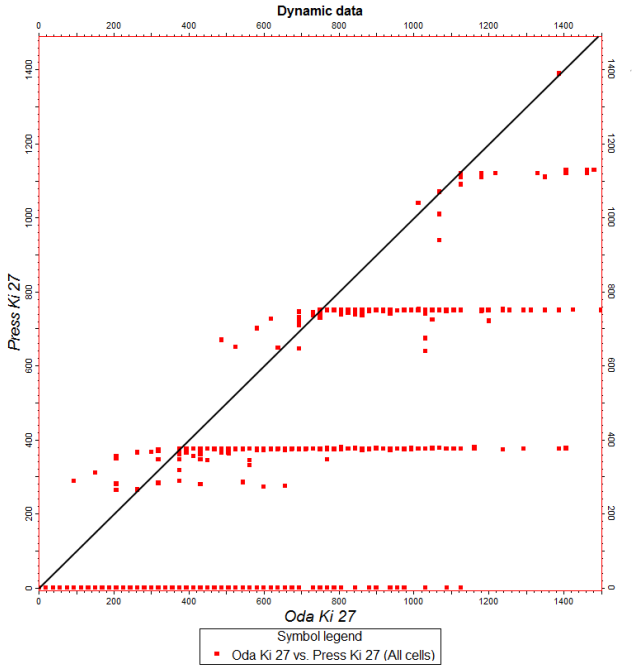

Figure D.27: Comparison of the effective permeability in X direction between the **Linear Pressure** numerical method and the **Oda**'s method(in mD)

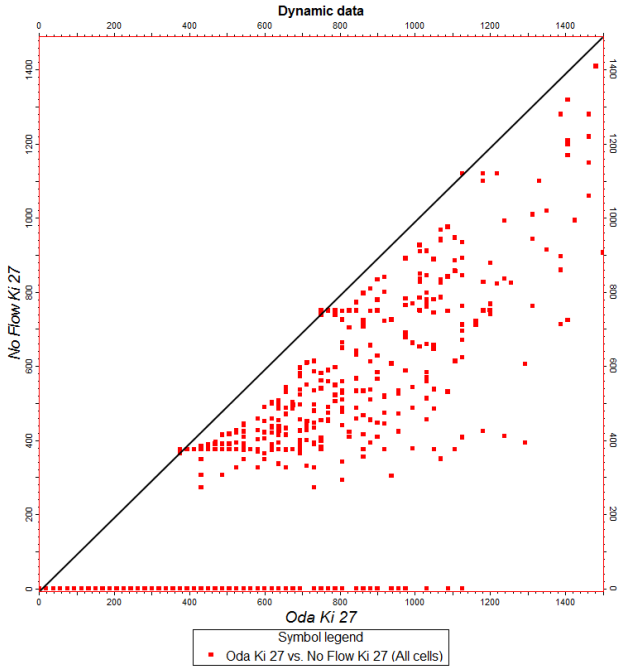

Figure D.28: Comparison of the effective permeability in X direction between the **No Flow** numerical method and the **Oda**'s method(in mD)

2.2.3. Effective permeability in Y direction

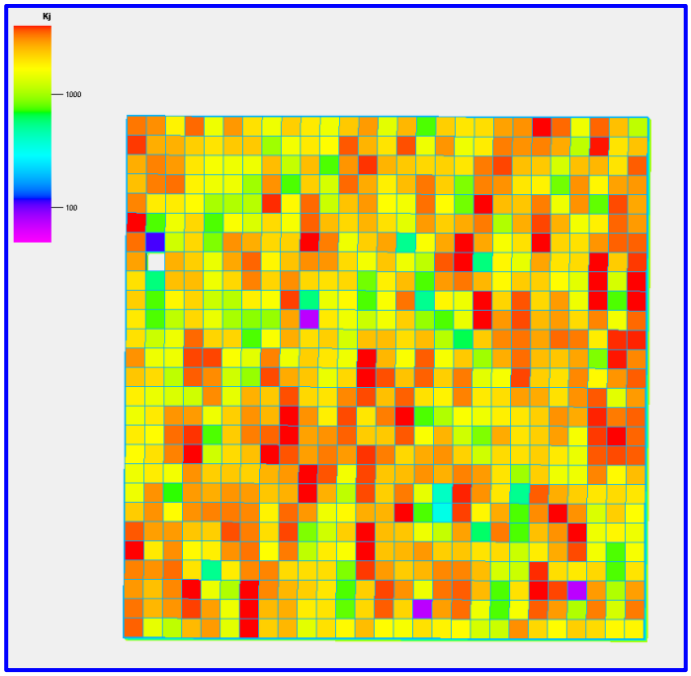

Figure D.29: Effective permeability in Y direction with **Oda's** method (in mD)

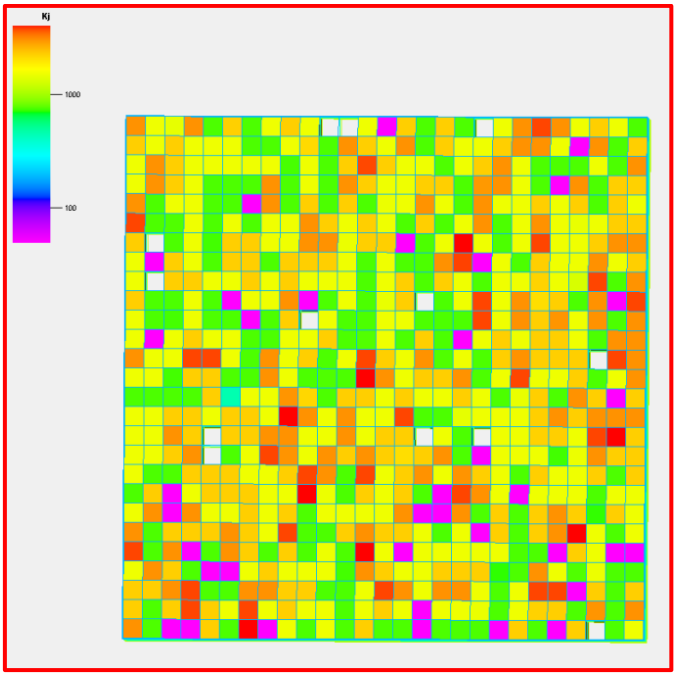

Figure D.30: Effective permeability in Y direction with the **Linear Pressure** numerical method (in mD)

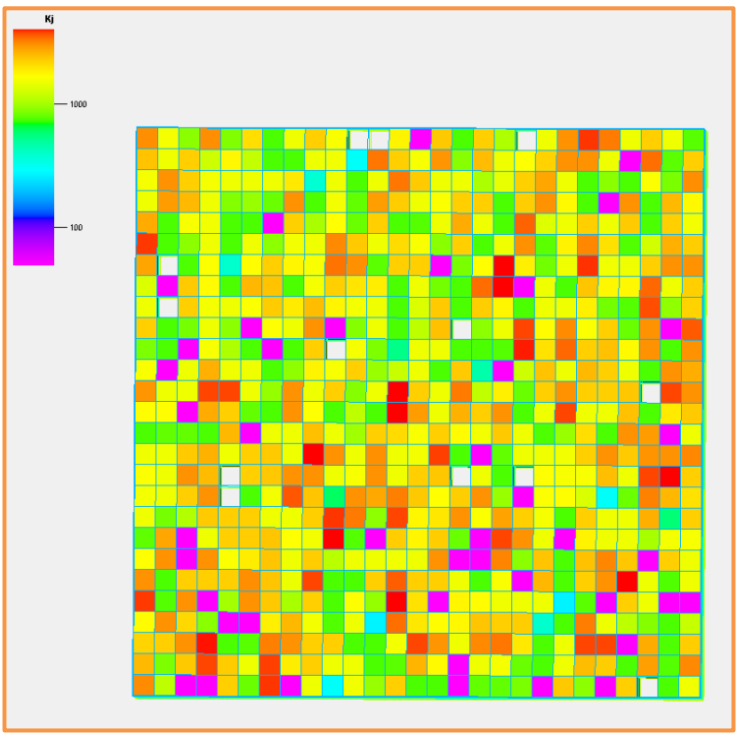

Figure D.31: Effective permeability in Y direction with the **No Flow** numerical method (in mD)

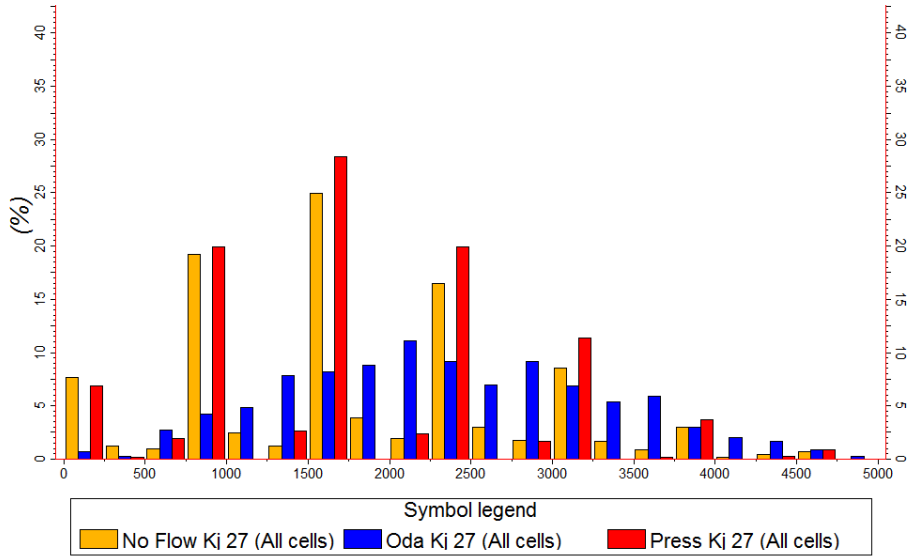

Figure D.32: Effective permeability in Y direction histogram for **Oda**'s method (in blue), the **Linear Pressure** numerical method (in red) and the **No Flow** numerical method (in orange) (in mD)

	Mean	Std. Deviation
Effective perm. Y using **Oda**'s method	2359.55 mD	958.85 mD
Effective perm. Y using **Linear Pressure** numerical method	1710.85 mD	971.84 mD
Effective perm. Y using **No Flow** numerical method	1722.78 mD	995.99 mD

Table D.11: Effective permeability in Y direction with 9x9 cells statistics

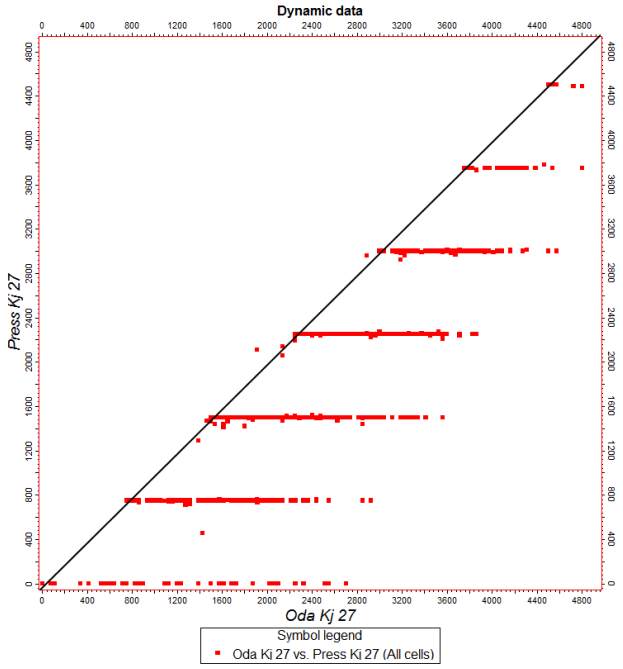

Figure D.33: Comparison of the effective permeability in X direction between the **Linear Pressure** numerical method and the **Oda**'s method(in mD)

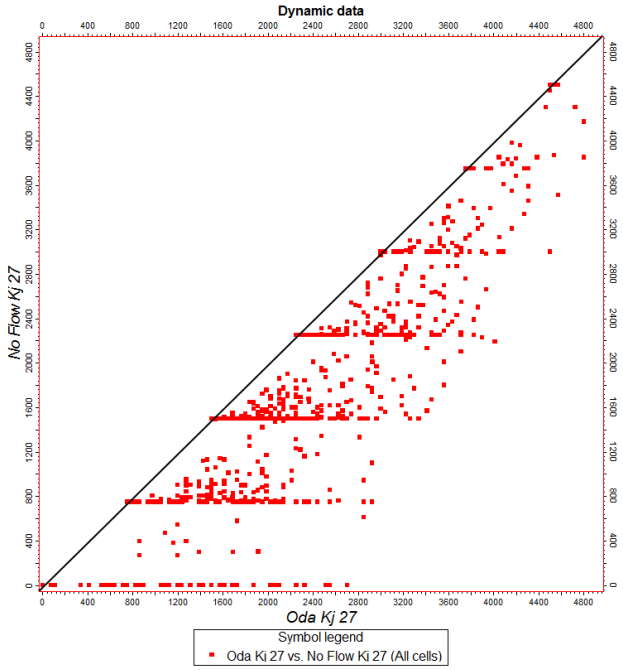

Figure D.34: Comparison of the effective permeability in X direction between the **No Flow** numerical method and the **Oda**'s method(in mD)

3. Dynamic simulation

The model characteristics are the same as Appendix C.

3.1. Oil production rate and production cumulative

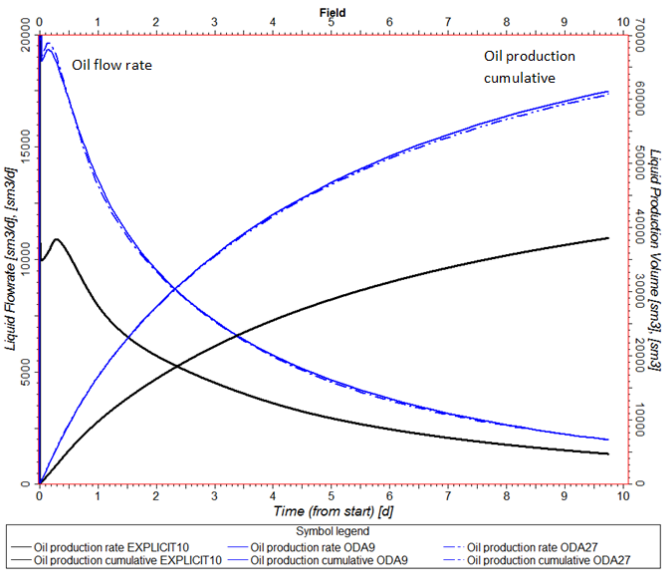

Figure D.35: Oil flow rate and oil production cumulative with the Explicit simulation (black line) and simulations using **Oda**'s method on a grid with 9x9 cells (solid blue line) and 27x27 cells (dashed blue line)

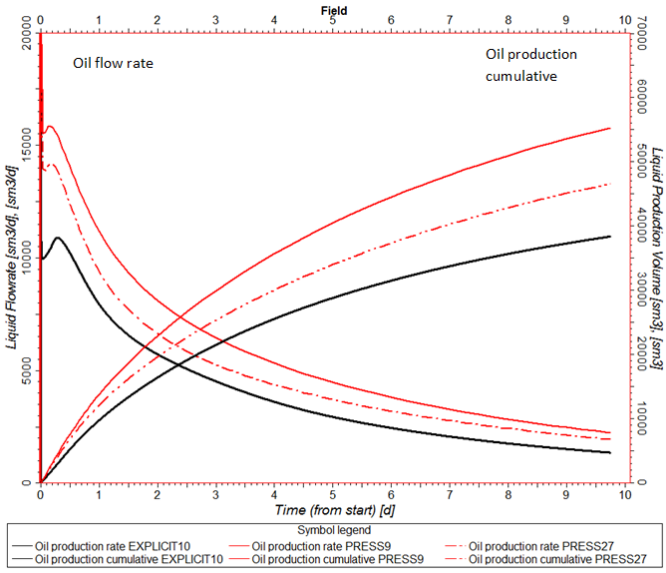

Figure D.36: Oil flow rate and oil production cumulative with the Explicit simulation (black line) and simulations using the **Linear Pressure** numerical method on a grid with 9x9 cells (solid red line) and 27x27 cells (dashed red line)

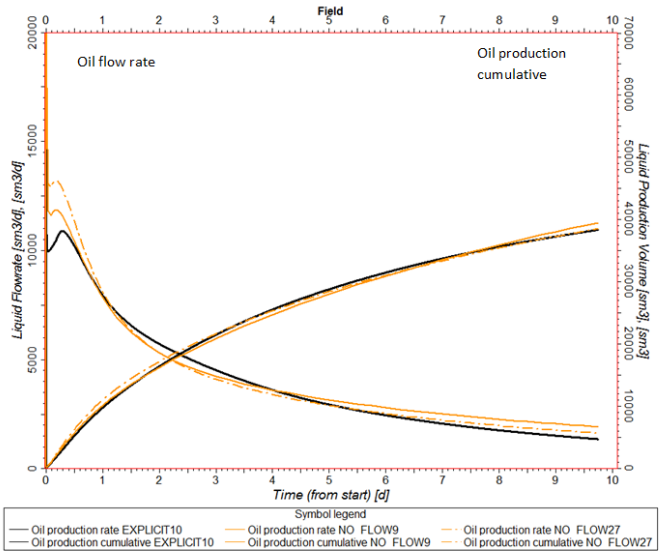

Figure D.37: Oil flow rate and oil production cumulative with the Explicit simulation (black line) and simulations using the **No Flow** numerical method on a grid with 9x9 cells (solid orange line) and 27x27 cells (dashed orange line)

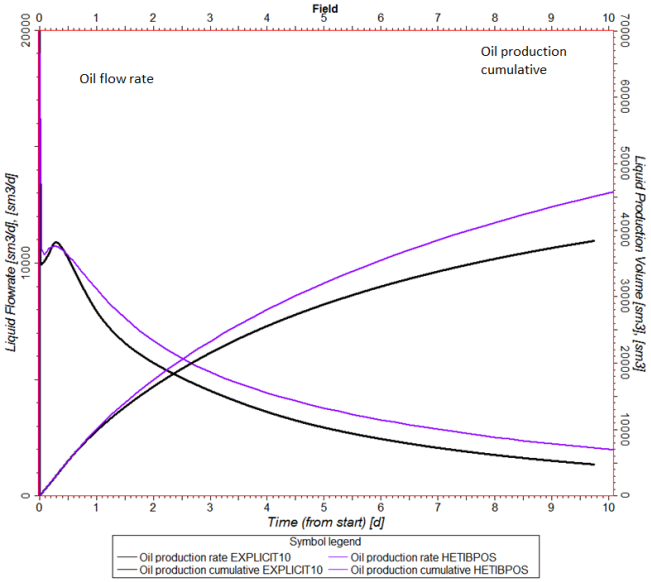

Figure D.38: Oil flow rate and oil production cumulative with the Explicit simulation (black line) and simulations using the **IBPOS** numerical method on a grid with 9x9 cells (solid purple line)

3.2. Oil and water production rate

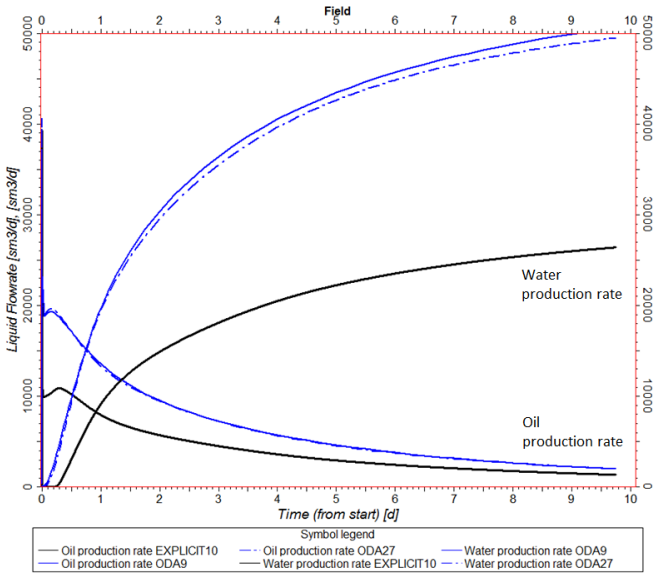

Figure D.39: Oil and water production rate with the Explicit simulation (black line) and simulations using **Oda**'s method on a grid with 9x9 cells (solid blue line) and 27x27 cells (dashed blue line)

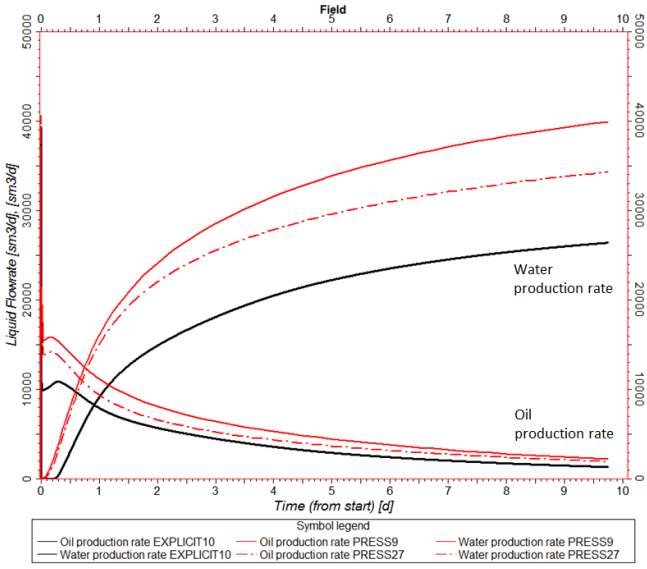

Figure D.40: Oil and water production rate with the Explicit simulation (black line) and simulations using the **Linear Pressure** numerical method on a grid with 9x9 cells (solid red line) and 27x27 cells (dashed red line)

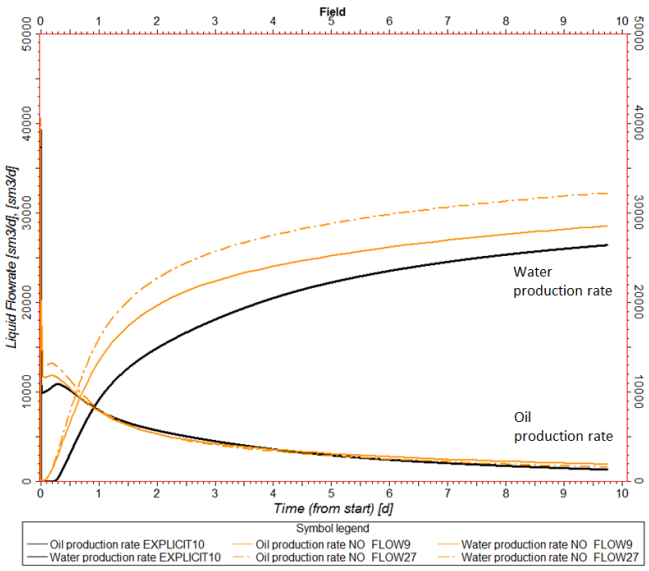

Figure D.41: Oil and water production rate with the Explicit simulation (black line) and simulations using the **No Flow** numerical method on a grid with 9x9 cells (solid orange line) and 27x27 cells (dashed orange line)

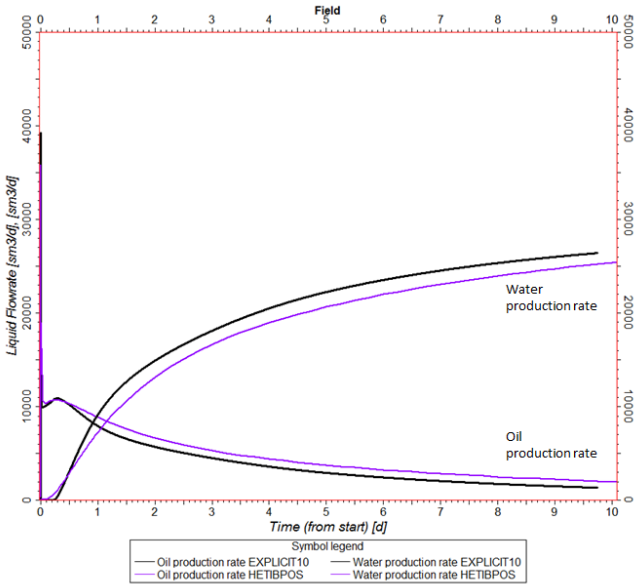

Figure D.42: Oil and water production rate with the Explicit simulation (black line) and simulations using the **IBPOS** numerical method on a grid with 9x9 cells (solid purple line)

3.3. Buckley-Leverett analysis

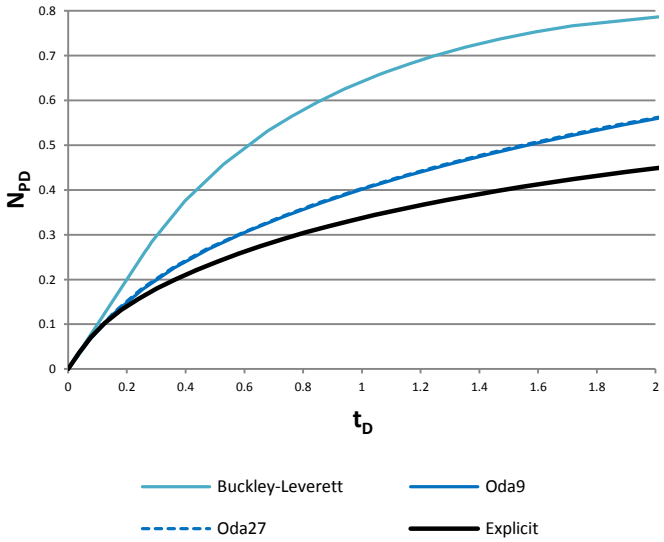

Figure D.43: Dimensionless oil production cumulative (N_{PD}) against dimensionless water injection cumulative (t_D) with the Buckley-Leverett theory (green line), the Explicit simulation (black line) and simulations using **Oda**'s method on a grid with 9x9 cells (solid blue line) and 27x27 cells (dashed blue line)

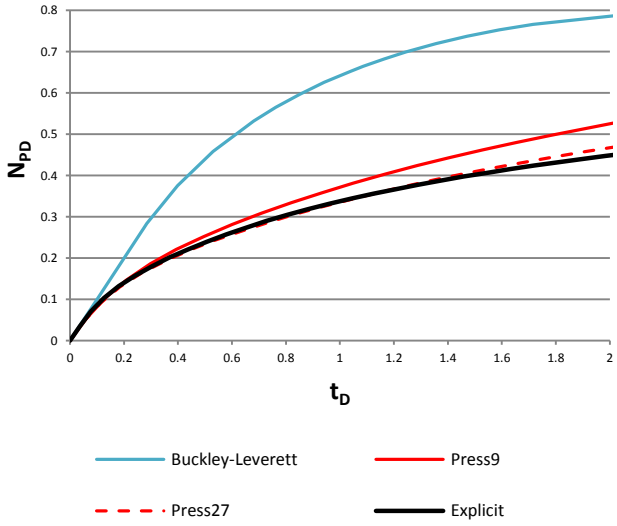

Figure D.44: Dimensionless oil production cumulative (N_{PD}) against dimensionless water injection cumulative (t_D) with the Buckley-Leverett theory (green line), the Explicit simulation (black line) and simulations using the **Linear Pressure** numerical method on a grid with 9x9 cells (solid red line) and 27x27 cells (dashed red line)

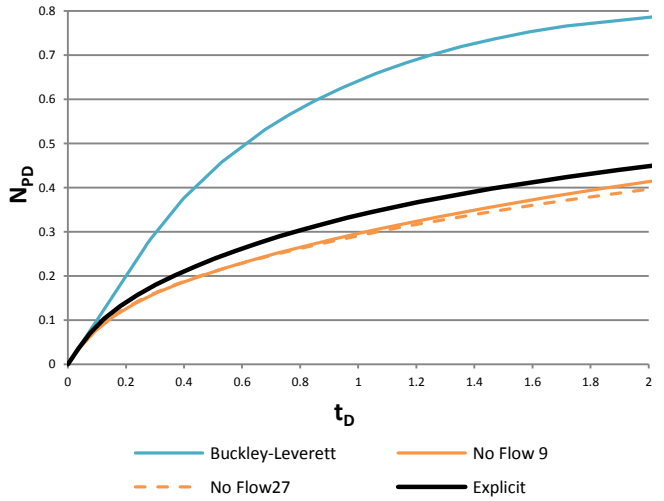

Figure D.45: Dimensionless oil production cumulative (N_{PD}) against dimensionless water injection cumulative (t_D) with the Buckley-Leverett theory (green line), the Explicit simulation (black line) and simulations using the **No Flow** numerical method on a grid with 9x9 cells (solid orange line) and 27x27 cells (dashed orange line)

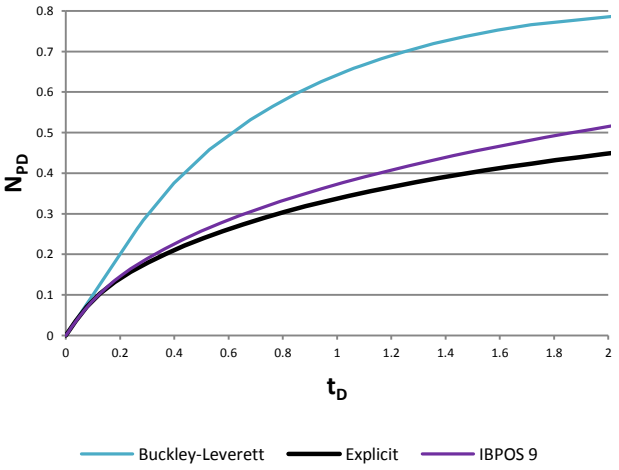

Figure D.46: Dimensionless oil production cumulative (N_{PD}) against dimensionless water injection cumulative (t_D) with the Buckley-Leverett theory (green line), the Explicit simulation (black line) and simulations using the **IBPOS** numerical method on a grid with 9x9 cells (solid purple line)

APPENDIX E: FRACS2000 properties

1. Model properties

Model properties	
Pressure gradient	9.809 10^{-2} bar m^{-1}
Section area	200 10^{3} m^{2}
Viscosity	1.6 cP
Fracture permeability	8.33 10^{7} mD
Fracture aperture	1 mm
Matrix permeability	1.01 10^{-2} mD
Matrix porosity	0.25

2. Grid with 5x5x4 cells

	Mean	Std. Deviation
Effective porosity	3.066E-5	0.72E-5
Sigma	0.0383	0.0589

	Mean	Std. Deviation
Effective perm. X using **Oda**'s method	1,366 mD	401 mD
Effective perm. X using **Linear Pressure** numerical method	695 mD	414 mD
Effective perm. X using **No Flow** numerical method	40 mD	115 mD

	Mean	Std. Deviation
Effective perm. Y using **Oda**'s method	1,241 mD	377 mD
Effective perm. Y using **Linear Pressure** numerical method	541 mD	395 mD
Effective perm. Y using **No Flow** numerical method	71 mD	168 mD

	Mean	Std. Deviation
Effective perm. Z using **Oda**'s method	2,503 mD	590 mD
Effective perm. Z using **Linear Pressure** numerical method	1406 mD	550 mD
Effective perm. Z using **No Flow** numerical method	1,319 mD	534 mD

3. Grid with 10x10x4 cells

	Mean	Std. Deviation
Effective porosity	3.066E-5	1.21E-5
Sigma	0.0286	0.0601

	Mean	Std. Deviation
Effective perm. X using **Oda**'s method	1,366 mD	655 mD
Effective perm. X using **Linear Pressure** numerical method	573 mD	505 mD
Effective perm. X using **No Flow** numerical method	177 mD	359 mD

	Mean	Std. Deviation
Effective perm. Y using **Oda**'s method	1,241 mD	671 mD
Effective perm. Y using **Linear Pressure** numerical method	555 mD	539 mD
Effective perm. Y using **No Flow** numerical method	185 mD	367 mD

	Mean	Std. Deviation
Effective perm. Z using **Oda**'s method	2,503 mD	989 mD
Effective perm. Z using **Linear Pressure** numerical method	1,443 mD	878 mD
Effective perm. Z using **No Flow** numerical method	1,300 mD	871 mD

4. Grid with 20x20x4 cells

	Mean	Std. Deviation
Effective porosity	3.078E-5	1.88E-5
Sigma	0.0299	0.0813

	Mean	Std. Deviation
Effective perm. X using **Oda**'s method	1,371 mD	1,097 mD
Effective perm. X using **Linear Pressure** numerical method	680 mD	853 mD
Effective perm. X using **No Flow** numerical method	532 mD	850 mD

	Mean	Std. Deviation
Effective perm. Y using **Oda**'s method	1,246 mD	1,027 mD
Effective perm. Y using **Linear Pressure** numerical method	563 mD	760 mD
Effective perm. Y using **No Flow** numerical method	422 mD	746 mD

	Mean	Std. Deviation
Effective perm. Z using **Oda**'s method	2,512 mD	1,535 mD
Effective perm. Z using **Linear Pressure** numerical method	1,414 mD	1,298 mD
Effective perm. Z using **No Flow** numerical method	1,223 mD	1,328 mD

5. CSMP++ simulation

Effective perm. X	239 mD
Effective perm. Y	287 mD
Effective perm. Z	904 mD

APPENDIX F: Program generating a DFN

This program (written in the Python language) is used to generate a 2D DFN from a facies map assuming two fracture sets (one horizontal and one vertical) and a constant fracture density in each facies. The fracture sets are defined with a constant length, aperture and permeability.

```
# ***************************************************************
# Generator of a 2D DFN
# Author: Benoit Decroux
# MSc Petroleum Engineering
# Imperial College London
# 2012
# ***************************************************************


#*************************************************************************************************************
#*************************************  PHYSICAL PROPERTIES  *************************************
#*************************************************************************************************************

LENGTH_HOR = 15 #(cells)
LENGTH_VER = 15 #(cells)

SIZEX = 180 #(cells)
SIZEY = SIZEX #(cells)

DX_ELEMENT = 20 #(cells)
DY_ELEMENT = DX_ELEMENT #(cells)

TITLE = "NetworkHomogene"
TITLE_FACIES = "Facies10"

NB_FRAC_HOR_PER_ELEM_0 = 10                 #number of fractures per element
NB_FRAC_VER_PER_ELEM_1 = NB_FRAC_HOR_PER_ELEM_0 #number of fractures per element

NB_FRAC_HOR_PER_ELEM_1 = 10                 #number of fractures per element
NB_FRAC_VER_PER_ELEM_0 = NB_FRAC_HOR_PER_ELEM_1 #number of fractures per element

# *************************************************************************************************************
# *************************************************************************************************************
# *************************************************************************************************************

SIZEX += DX_ELEMENT
SIZEY += DY_ELEMENT

NB_ELEMENT_X = int(SIZEX / DX_ELEMENT)
NB_ELEMENT_Y = int(SIZEY / DY_ELEMENT)
assert NB_ELEMENT_X * DX_ELEMENT == SIZEX
assert NB_ELEMENT_Y * DY_ELEMENT == SIZEY

NB_CELL_IN_ELEM = DX_ELEMENT * DY_ELEMENT

#Reading facies file

import os
mFile = open(TITLE_FACIES + ".txt", "r")
string = mFile.read().split()
mFile.close()
iterator = 0

NB_FRAC_HOR_PER_ELEM = []
NB_FRAC_VER_PER_ELEM = []
for j in range(NB_ELEMENT_X):
    tempHor = []
    tempVer = []
    for i in range(NB_ELEMENT_Y):
        if string[iterator] == "0":
            tempHor.append(NB_FRAC_HOR_PER_ELEM_0)
            tempVer.append(NB_FRAC_VER_PER_ELEM_0)
        else:
            assert(string[iterator] == "1")
            tempHor.append(NB_FRAC_HOR_PER_ELEM_1)
            tempVer.append(NB_FRAC_VER_PER_ELEM_1)

        iterator += 1
```

```
    NB_FRAC_HOR_PER_ELEM.append(tempHor)
    NB_FRAC_VER_PER_ELEM.append(tempVer)

FRAC = 0
EMPTY = 1
FORBIDDEN = 2

import random
random.seed()

#*****************************************************************************************************************
#************************************  HORIZONTAL FRACTURES  *****************************************
#*****************************************************************************************************************

#Initialisation

IS_FRACTURE_HOR = []
for i in range(SIZEX):
    temp = []
    for j in range(SIZEY):
        temp.append(EMPTY)
    IS_FRACTURE_HOR.append(temp)

NB_CELL_AVAILABLE_HOR = []
for i in range(NB_ELEMENT_X):
    temp = []
    for j in range(NB_ELEMENT_Y):
        temp.append(NB_CELL_IN_ELEM)
    NB_CELL_AVAILABLE_HOR.append(temp)

#Calculation

for i in range(NB_ELEMENT_X):
    for j in range(NB_ELEMENT_Y):

        for k in range(NB_FRAC_HOR_PER_ELEM[i][j]):

            #Find the location of the fracture k
            mRand = random.randint(0, NB_CELL_AVAILABLE_HOR[i][j] - 1)
            #print mRand, NB_CELL_AVAILABLE_HOR[i][j]
            l = -1
            mx = i * DX_ELEMENT
            my = j * DY_ELEMENT -1
            while l < mRand:
                l +=1
                my += 1
                if my == (j + 1) * DY_ELEMENT:
                    my = j * DY_ELEMENT
                    mx += 1
                    assert(mx < (i+1) * DX_ELEMENT)

                while IS_FRACTURE_HOR[mx][my] != EMPTY:
                    my += 1
                    if my == (j + 1) * DY_ELEMENT:
                        my = j * DY_ELEMENT
                        mx += 1
                        assert(mx < (i+1) * DX_ELEMENT)

            #Write the fracture
            l = 0
            mXt = mx
            while l < LENGTH_HOR and mXt < SIZEX:

                if IS_FRACTURE_HOR[mXt][my] == EMPTY:
                    NB_CELL_AVAILABLE_HOR[int(mXt/DX_ELEMENT)][int(my/DY_ELEMENT)] -= 1

                assert IS_FRACTURE_HOR[mXt][my] != FRAC
                IS_FRACTURE_HOR[mXt][my] = FRAC
                l += 1
                mXt += 1

            l = 0
            mXt = mx - 1
            while l < LENGTH_HOR - 1 and mXt >= 0:
                if IS_FRACTURE_HOR[mXt][my] == EMPTY:
                    IS_FRACTURE_HOR[mXt][my] = FORBIDDEN
                    NB_CELL_AVAILABLE_HOR[int(mXt/DX_ELEMENT)][int(my/DY_ELEMENT)] -= 1
                l += 1
```

```
            mXt -= 1

#Write the String

stringHor = []
count = 0
for x in range(DX_ELEMENT, SIZEX):
    for y in range(DY_ELEMENT, SIZEY):
        if IS_FRACTURE_HOR[x][y] == FRAC:
            stringHor.append("1")
        else:
            stringHor.append("0")
        count += 1
        if count == 30:
            stringHor.append("\n")
            count = 0

#*****************************************************************************************************
#*************************************  VERTICAL FRACTURES  ******************************************
#*****************************************************************************************************

#Initialisation

IS_FRACTURE_VER = []
for i in range(SIZEX):
    temp = []
    for j in range(SIZEY):
        temp.append(EMPTY)
    IS_FRACTURE_VER.append(temp)

NB_CELL_AVAILABLE_VER = []
for i in range(NB_ELEMENT_X):
    temp = []
    for j in range(NB_ELEMENT_Y):
        temp.append(NB_CELL_IN_ELEM)
    NB_CELL_AVAILABLE_VER.append(temp)

#Calculation

for i in range(NB_ELEMENT_X):
    for j in range(NB_ELEMENT_Y):

        for k in range(NB_FRAC_VER_PER_ELEM[i][j]):

            #Find the location of the fracture k
            mRand = random.randint(0, NB_CELL_AVAILABLE_VER[i][j] - 1)
            l = -1
            mx = i * DX_ELEMENT
            my = j * DY_ELEMENT -1
            while l < mRand:
                l +=1
                my += 1
                if my == (j + 1) * DY_ELEMENT:
                    my = j * DY_ELEMENT
                    mx += 1
                    assert(mx < (i+1) * DX_ELEMENT)

                while IS_FRACTURE_VER[mx][my] != EMPTY:
                    my += 1
                    if my == (j + 1) * DY_ELEMENT:
                        my = j * DY_ELEMENT
                        mx += 1
                        assert(mx < (i+1) * DX_ELEMENT)

            #Write the fracture
            l = 0
            mYt = my
            while l < LENGTH_VER and mYt < SIZEY:

                if IS_FRACTURE_VER[mx][mYt] == EMPTY:
                    NB_CELL_AVAILABLE_VER[int(mx/DX_ELEMENT)][int(mYt/DY_ELEMENT)] -= 1

                assert IS_FRACTURE_VER[mx][mYt] != FRAC
                IS_FRACTURE_VER[mx][mYt] = FRAC
                l += 1
                mYt += 1

            l = 0
            mYt = my - 1
```

```
            while l < LENGTH_VER - 1 and mYt >= 0:
                if IS_FRACTURE_VER[mx][mYt] == EMPTY:
                    IS_FRACTURE_VER[mx][mYt] = FORBIDDEN
                    NB_CELL_AVAILABLE_VER[int(mx/DX_ELEMENT)][int(mYt/DY_ELEMENT)] -= 1
                l += 1
                mYt -= 1
#Write the String

stringVer = []
count = 0
for x in range(DX_ELEMENT, SIZEX):
    for y in range(DY_ELEMENT, SIZEY):
        if IS_FRACTURE_VER[x][y] == FRAC:
            stringVer.append("1")
        else:
            stringVer.append("0")
        count += 1
        if count == 30:
            stringVer.append("\n")
            count = 0

#Final

final = " ".join(stringHor) + "\n\n" + " ".join(stringVer)

import os
os.chdir("C:\Users\BCD111\Desktop\Explicit")
mFile = open(TITLE + ".txt", "w")
mFile.write(final)
mFile.close()

print TITLE + ".txt"

print "Done"
```

APPENDIX G: Program converting a DFN to Petrel format

This program (written in the Python language) converts a DFN written in my in-house format to a format readable by Petrel to calculate the fracture network effective properties.

```
# ************************************************************
# Fracture network converter from my in house format to Petrel format
# Author: Benoit Decroux
# MSc Petroleum Engineering
# Imperial College London
# 2012
# ************************************************************

#*************************************************************************************************************
# ***************************************  PHYSICAL PROPERTIES  ***************************************
#*************************************************************************************************************

SIZEX = 180 #(cells)

DEPTH = 150.0 #(m)
DZ = 10.0 #(m)
PERM_F_HOR = 200000.0 #(mD)
PERM_F_VER = 200000.0 #(mD)
SIZEY = SIZEX #(cells)
DX = 4.0 #(m)
EDGE_DX = 1.0 #(m)
APERTURE = 0.1 #(m)
TITLE = "NetworkHomogene"
TITLE_NETWORK = TITLE
TITLE_OUT = "Petr_" + TITLE_NETWORK
COMPRESSIBILITY = 3.40282E38

# *************************************************************************************************************
# *************************************************************************************************************
# *************************************************************************************************************


class Point:
    def __init__(self, mX, mY):
        self.x = mX
        self.y = mY

class Fracture:
    def __init__(self, mPerm, mCompr, mApert, mx1, my1, mx2=0, my2=0):
        self.point1 = Point(mx1, my1)
        self.point2 = Point(mx2, my2)
        self.perm = mPerm
        self.compr = mCompr
        self.apert = mApert

    def prop(self):
        mReturn = []
        mReturn.append("    4    1    " + str(self.perm) + "    " + str(self.compr) + "    " + str(self.apert))
        mReturn.append("    1    " + str(self.point1.x) + "    " + str(-self.point1.y) + "    {depth2}")
        mReturn.append("    2    " + str(self.point2.x) + "    " + str(-self.point2.y) + "    {depth2}")
        mReturn.append("    3    " + str(self.point1.x) + "    " + str(-self.point1.y) + "    {depth}")
        mReturn.append("    4    " + str(self.point2.x) + "    " + str(-self.point2.y) + "    {depth}")
        mReturn.append("    0    0.1    0.99    0.1")
        return "\n".join(mReturn)



HALF = DX / 2.0

#Reading network file

import os
mFile = open(TITLE_NETWORK + ".txt", "r")
network = mFile.read().split()
mFile.close()
networkIterator = 0

IS_FRACTURE_HOR = []

for x in range(SIZEX):
    temp = []
    for y in range(SIZEY):
```

```
        temp.append(network[networkIterator] == "1")
        networkIterator += 1
    IS_FRACTURE_HOR.append(temp)

IS_FRACTURE_VER = []

for x in range(SIZEX):
    temp = []
    for y in range(SIZEY):
        temp.append(network[networkIterator] == "1")
        networkIterator += 1
    IS_FRACTURE_VER.append(temp)

#Coordinates

MAP_X = []
realX = EDGE_DX

for x in range(SIZEX + 1):
    MAP_X.append(realX)
    realX += DX


MAP_Y = []
realY = 0.0

for y in range(SIZEY + 1):
    MAP_Y.append(realY)
    realY += DX


FRACTURES = []

#Vertical fracture generation

#FRACTURES.append(Fracture(PERM_F, COMPRESSIBILITY, APERTURE, MAP_X[0] - EDGE_DX/2.0, MAP_Y[0], MAP_X[0] - EDGE_DX/2.0, MAP_Y[SIZEY]))

#FRACTURES.append(Fracture(PERM_F, COMPRESSIBILITY, APERTURE, MAP_X[SIZEX] + EDGE_DX/2.0, MAP_Y[0], MAP_X[SIZEX] + EDGE_DX/2.0, MAP_Y[SIZEY]))

for x in range(SIZEX):
    #frac = Fracture(PERM_F, COMPRESSIBILITY, APERTURE, MAP_X[x] + HALF, MAP_Y[0] - EDGE_DX)
    #FRACTURES.append(frac)
    #isPoint1 = False
    isPoint1 = True

    for y in range(SIZEY):
        if IS_FRACTURE_VER[x][y] and isPoint1:
            frac = Fracture(PERM_F_VER, COMPRESSIBILITY, APERTURE, MAP_X[x] + HALF, MAP_Y[y])
            FRACTURES.append(frac)
            isPoint1 = False
        elif (not IS_FRACTURE_VER[x][y]) and (not isPoint1):
            frac.point2 = Point(MAP_X[x] + HALF, MAP_Y[y])
            isPoint1 = True

    #if isPoint1:
    #    frac = Fracture(PERM_F, COMPRESSIBILITY, APERTURE, MAP_X[x] + HALF, MAP_Y[SIZEY])
    #    FRACTURES.append(frac)
    #    isPoint1 = False

    if not isPoint1:
        #frac.point2 = Point(MAP_X[x] + HALF, MAP_Y[SIZEY] + EDGE_DX)
        frac.point2 = Point(MAP_X[x] + HALF, MAP_Y[SIZEY])
        isPoint1 = True

#Horizontal fracture generation

#FRACTURES.append(Fracture(PERM_F, COMPRESSIBILITY, APERTURE, MAP_X[0], MAP_Y[0] - EDGE_DX/2.0, MAP_X[SIZEX], MAP_Y[0] - EDGE_DX/2.0))

#FRACTURES.append(Fracture(PERM_F, COMPRESSIBILITY, APERTURE, MAP_X[0], MAP_Y[SIZEY] + EDGE_DX/2.0, MAP_X[SIZEX], MAP_Y[SIZEY] + EDGE_DX/2.0))


for y in range(SIZEY):
    #frac = Fracture(PERM_F, COMPRESSIBILITY, APERTURE, MAP_X[0] - EDGE_DX, MAP_Y[y] + HALF)
    #FRACTURES.append(frac)
    #isPoint1 = False
```

```
    isPoint1 = True

    for x in range(SIZEX):
        if IS_FRACTURE_HOR[x][y] and isPoint1:
            frac = Fracture(PERM_F_HOR, COMPRESSIBILITY, APERTURE, MAP_X[x], MAP_Y[y] + HALF)
            FRACTURES.append(frac)
            isPoint1 = False
        elif (not IS_FRACTURE_HOR[x][y]) and (not isPoint1):
            frac.point2 = Point(MAP_X[x], MAP_Y[y] + HALF)
            isPoint1 = True

    #if isPoint1:
    #    frac = Fracture(PERM_F, COMPRESSIBILITY, APERTURE, MAP_X[SIZEX], MAP_Y[y] + HALF)
    #    FRACTURES.append(frac)
    #    isPoint1 = False

    if not isPoint1:
        #frac.point2 = Point(MAP_X[SIZEX] + EDGE_DX, MAP_Y[y] + HALF)
        frac.point2 = Point(MAP_X[SIZEX], MAP_Y[y] + HALF)
        isPoint1 = True

NB_FRACTURES = len(FRACTURES)

mstring = []

mstring.append("""BEGIN FORMAT
    Format = Ascii
    Length_Unit = M
    XAxis = East
    Scale = 100.0
    No_Fractures = {nbFractures}
    No_TessFractures = 0
    No_Nodes = {nbFractures*4}
    No_RockBlocks = 0
    No_NodesRockBlock = 0
    No_Properties = 3
END FORMAT

BEGIN PROPERTIES
    Prop1    =    (Real*4)"Permeability"
    Prop2    =    (Real*4)"Compressibility"
    Prop3    =    (Real*4)"Aperture"
END PROPERTIES

BEGIN SETS
    Set1 = "Discrete fractures"
END SETS
""")

mstring.append("\nBEGIN FRACTURE\n")
for i in range(NB_FRACTURES):
    mstring.append("    " + str(i+1) + FRACTURES[i].prop() + "\n")
mstring.append("END FRACTURE\n")

mstring.append("""
BEGIN TESSFRACTURE
END TESSFRACTURE

BEGIN ROCKBLOCK
END ROCKBLOCK

""")

final = "".join(mstring)

final = final.replace("{depth}", str(-DEPTH))
final = final.replace("{depth2}", str(-DEPTH - DZ))
final = final.replace("{nbFractures}", str(NB_FRACTURES))
final = final.replace("{nbFractures*4}", str(NB_FRACTURES*4))

mFile = open(TITLE_OUT + ".txt", "w")
mFile.write(final)
mFile.close()

print TITLE_OUT + ".txt"

print "Done"
```

APPENDIX H: Program generating the explicit simulation

This program (written in the Python language) generates an Eclipse simulation file (.data file) for an explicit simulation by reading a DFN written in my in-house format.

```
# ************************************************************
# Explicit simulation of a 2D fractured reservoir
# Author: Benoit Decroux
# MSc Petroleum Engineering
# Imperial College London
# 2012
# ************************************************************

#*************************************************************************************************************
# ************************************  PHYSICAL PROPERTIES  ************************************
#*************************************************************************************************************

LENGTH_X = 720.0 #(m)
DX_FACIES = 80.0 #(m)
DY_FACIES = DX_FACIES #(m)
DEPTH = 150.0 #(m)
DZ = 10.0 #(m)
PORO_M = 0.15
PORO_F = 1.0
PERM_M = 0.0 #(mD)
PERM_F_HOR = 200000.0 #(mD)
PERM_F_VER = 200000.0 #(mD)
LENGTH_Y = LENGTH_X
DX = 4.0 #(m)
EDGE_DX = 1.0 #(m)
FRACTURE_DX = 0.1 #(m)
PRESSURE = 200.0 #(bars)
INJ_PRESSURE = PRESSURE + 50.0 #(bars)
PROD_PRESSURE = 1.0 #(bars)
TITLE = "ExplicitHX"
WELL_DIAMETER = 0.2 #(m)
TITLE_NETWORK = "NetworkHomogene"
IS_WATER = True
IS_OIL = True
WELL_PERM = 2700.0 #(mD)
WELL_PORO = 0.02
INITIAL_SW = 0.2

# *************************************************************************************************************
# *************************************************************************************************************
# *************************************************************************************************************

MONTHS = ["JAN", "FEB", "MAR", "APR", "MAY", "JUN", "JUL", "AUG", "SEP", "OCT", "NOV", "DEC"]
INJECTOR = "INJ"
PRODUCER = "PRO"
WATER = "WATER"
OIL = "OIL"
POROSITY = "PORO"
PERMEABILITY = "PERMX"
WATER_SATURATION = "SWAT"

WELLS = []

class Point:
    def __init__(self, mX, mY):
        global CELL_FACIES_X
        global CELL_FACIES_Y
        global LENGTH_TRANS
        self.x = mX
        self.y = mY
        self.realX = self.x * CELL_FACIES_X + int(CELL_FACIES_X/2)
        self.realY = self.y * CELL_FACIES_Y + int(CELL_FACIES_Y/2)
        self.finalX = self.realX * LENGTH_TRANS + 1
        self.finalY = self.realY * LENGTH_TRANS + 1

class Cell:
    def __init__(self, mdX, mdY, mPerm, mPoro, mSw):
        self.dx = mdX
        self.dy = mdY
        self.perm = mPerm
        self.poro = mPoro
        self.Sw = mSw
```

```
    def get(self, prop):
        global POROSITY
        global PERMEABILITY
        global WATER_SATURATION
        if prop == POROSITY:
            return self.poro
        if prop == PERMEABILITY:
            return self.perm
        if prop == WATER_SATURATION:
            return self.Sw

class Well:
    def __init__(self, mX, mY, mType, mPhase, mControle, mPressure, mRate, mName, mDiameter, mFlag = "OPEN"):
        self.point = Point(mX, mY)
        self.type = mType
        self.phase = mPhase
        self.controle = mControle
        self.pressure = mPressure
        self.rate = mRate
        self.name = mName
        self.diameter = mDiameter
        self.flag = mFlag

    def wellSpec(self):
        mReturn = []
        mReturn.append(self.name)
        mReturn.append(self.type)
        mReturn.append(str(self.point.finalX + 1))
        mReturn.append(str(self.point.finalY + 1))
        mReturn.append("{depth}")
        mReturn.append(self.phase)
        mReturn.append("/\n")
        return "  ".join(mReturn)

    def compDat(self):
        mReturn = []
        mReturn.append(self.name)
        mReturn.append("2*  1  1")
        mReturn.append(self.flag)
        mReturn.append("2*")
        mReturn.append(str(self.diameter))
        mReturn.append("1*  0  1*  Z  /\n")
        return "  ".join(mReturn)

    def wConProd(self):
        global PRODUCER
        if self.type != PRODUCER:
            return ""
        mReturn = []
        mReturn.append(self.name)
        mReturn.append(self.flag)
        mReturn.append(self.controle)
        mReturn.append("4*")
        mReturn.append(str(self.rate))
        mReturn.append(str(self.pressure))
        mReturn.append("/\n")
        return "  ".join(mReturn)

    def wConInje(self):
        global INJECTOR
        if self.type != INJECTOR:
            return ""
        mReturn = []
        mReturn.append(self.name)
        mReturn.append(self.phase)
        mReturn.append(self.flag)
        mReturn.append(self.controle)
        mReturn.append(str(self.rate))
        mReturn.append("1*")
        mReturn.append(str(self.pressure))
        mReturn.append("/\n")
        return "  ".join(mReturn)

def propGeneration(prop):
    global REAL_SIZE_X
    global REAL_SIZE_Y
    global MAP
```

```
    temp = []
    count = 0
    temp2 = []
    for j in range(REAL_SIZE_Y):
        for i in range(REAL_SIZE_X):
            temp.append(str(MAP[i][j].get(prop)))
            count+= 1
            if count == 6:
                if len(temp2) > 0:
                    temp2.append("\n")
                temp2.append("   ".join(temp))
                temp = []
                count = 0

    if count>0:
        if len(temp2) > 0:
            temp2.append("\n")
        temp2.append("   ".join(temp))
        temp = []
        count = 0

    return "".join(temp2)


#Initialisation

SIZEX = int(LENGTH_X / DX)
SIZEY = int(LENGTH_Y / DX)
assert (SIZEX * DX == LENGTH_X)
assert (SIZEY * DX == LENGTH_Y)

FACIES_SIZE_X = int(LENGTH_X / DX_FACIES)
FACIES_SIZE_Y = int(LENGTH_Y / DY_FACIES)
assert (FACIES_SIZE_X * DX_FACIES == LENGTH_X)
assert (FACIES_SIZE_Y * DY_FACIES == LENGTH_Y)

CELL_FACIES_X = int(DX_FACIES / DX)
CELL_FACIES_Y = int(DY_FACIES / DX)
assert (CELL_FACIES_X * DX == DX_FACIES)
assert (CELL_FACIES_Y * DX == DY_FACIES)

TRANS_DX = [(DX-FRACTURE_DX)/2.0]
TRAN_PORO = [PORO_M]
TRANS_PERM_VER = [PERM_M]
TRANS_PERM_HOR = [PERM_M]

HALF = len(TRANS_DX)
LENGTH_TRANS = 2 * HALF + 1

TRANS_DX.append(FRACTURE_DX)
TRAN_PORO.append(PORO_F)
TRANS_PERM_VER.append(PERM_F_VER)
TRANS_PERM_HOR.append(PERM_F_HOR)

for i in range(HALF - 1, -1, -1):
    TRANS_DX.append(TRANS_DX[i])
    TRAN_PORO.append(TRAN_PORO[i])
    TRANS_PERM_VER.append(TRANS_PERM_VER[i])
    TRANS_PERM_HOR.append(TRANS_PERM_HOR[i])


REAL_SIZE_X = SIZEX * LENGTH_TRANS + 2
REAL_SIZE_Y = SIZEY * LENGTH_TRANS + 2

REAL_NB_CELLS = REAL_SIZE_X * REAL_SIZE_Y


#Reading network file

import os
mFile = open(TITLE_NETWORK + ".txt", "r")
network = mFile.read().split()
mFile.close()
networkIterator = 0

IS_FRACTURE_HOR = []

for x in range(SIZEX):
    temp = []
```

```
    for y in range(SIZEY):
        temp.append(network[networkIterator] == "1")
        networkIterator += 1
    IS_FRACTURE_HOR.append(temp)

IS_FRACTURE_VER = []

for x in range(SIZEX):
    temp = []
    for y in range(SIZEY):
        temp.append(network[networkIterator] == "1")
        networkIterator += 1
    IS_FRACTURE_VER.append(temp)


#Wells creation

phaseProduction = OIL
if IS_OIL == False:
    phaseProduction = WATER

phaseInjection = WATER
if IS_WATER == False:
    phaseInjection = OIL

isShut = "OPEN"

#def __init__( mX, mY, mType, mPhase, mControle, mPressure, mRate, mName , mDiameter):

#WELLS.append(Well(0, 0, INJECTOR, phaseInjection, "BHP", INJ_PRESSURE, "1*", "INJ1", WELL_DIAMETER, isShut))

#WELLS.append(Well(FACIES_SIZE_X-1, FACIES_SIZE_Y-1, INJECTOR, phaseInjection, "BHP", INJ_PRESSURE, "1*", "INJ3", WELL_DIAMETER, isShut))

#WELLS.append(Well(0, FACIES_SIZE_Y-1, INJECTOR, phaseInjection, "BHP", INJ_PRESSURE, "1*","INJ2", WELL_DIAMETER, isShut))

#WELLS.append(Well(FACIES_SIZE_X-1, 0, INJECTOR, phaseInjection , "BHP", INJ_PRESSURE, "1*", "INJ4", WELL_DIAMETER, isShut))


WELLS.append(Well(0, int(FACIES_SIZE_Y/2), INJECTOR, phaseInjection, "BHP", INJ_PRESSURE, "1*", "INJ1", WELL_DIAMETER, isShut))

WELLS.append(Well(FACIES_SIZE_X-1, int(FACIES_SIZE_Y/2), INJECTOR, phaseInjection, "BHP", INJ_PRESSURE, "1*", "INJ3", WELL_DIAMETER, isShut))

WELLS.append(Well(int(FACIES_SIZE_X/2), FACIES_SIZE_Y-1, INJECTOR, phaseInjection , "BHP", INJ_PRESSURE, "1*", "INJ2", WELL_DIAMETER, isShut))

WELLS.append(Well(int(FACIES_SIZE_X/2), 0, INJECTOR, phaseInjection , "BHP", INJ_PRESSURE , "1*", "INJ4", WELL_DIAMETER, isShut))


WELLS.append(Well(int(FACIES_SIZE_X/2), int(FACIES_SIZE_Y/2), PRODUCER, phaseProduction, "BHP", PROD_PRESSURE, "1*", "PROD", WELL_DIAMETER))


NB_WELLS = len(WELLS)

#Map creation

Area = 0
MAP = []
temp = []
temp.append(Cell(EDGE_DX, EDGE_DX, 0.0, PORO_M, INITIAL_SW))
for y in range(SIZEY):
    for j in range(LENGTH_TRANS):
        temp.append(Cell(EDGE_DX, TRANS_DX[j], 0.0, PORO_M, INITIAL_SW))

temp.append(Cell(EDGE_DX, EDGE_DX, 0.0, PORO_M, INITIAL_SW))
MAP.append(temp)

for x in range(SIZEX):
    for i in range(LENGTH_TRANS):
        temp = []
        temp.append(Cell(TRANS_DX[i], EDGE_DX, 0.0, PORO_M, INITIAL_SW))
        for y in range(SIZEY):

            #Check if it is a well zone
            isWellZone = False
```

```
        for well in WELLS:
            if abs(x-well.point.realX) <= CELL_FACIES_X/2 and abs(y-well.point.realY) <= CELL_FACIES_Y/2:
                isWellZone = True
                break

        for j in range(LENGTH_TRANS):
            if isWellZone:
                perm = WELL_PERM
                poro = WELL_PORO

            else:
                if IS_FRACTURE_VER[x][y]:
                    perm_vert = TRANS_PERM_VER[i]
                    poro_vert = TRAN_PORO[i]
                else:
                    perm_vert = PERM_M
                    poro_vert = PORO_M

                if IS_FRACTURE_HOR[x][y]:
                    perm_hor = TRANS_PERM_HOR[j]
                    poro_hor = TRAN_PORO[j]
                else:
                    perm_hor = PERM_M
                    poro_hor = PORO_M
                perm = max(perm_vert, perm_hor)
                poro = max(poro_vert, poro_hor)
            if perm > 0:
                Area+= TRANS_DX[i]*TRANS_DX[j]*poro

            temp.append(Cell(TRANS_DX[i], TRANS_DX[j], perm, poro, INITIAL_SW))

        temp.append(Cell(TRANS_DX[i], EDGE_DX, 0.0, PORO_M, INITIAL_SW))
        MAP.append(temp)

print "Initial oil volume =" + str(Area*(1-INITIAL_SW)*DZ) + "m3"

temp = []
temp.append(Cell(EDGE_DX, EDGE_DX, 0.0, PORO_M, INITIAL_SW))
for y in range(SIZEY):
    for j in range(LENGTH_TRANS):
        temp.append(Cell(EDGE_DX, TRANS_DX[j], 0.0, PORO_M, INITIAL_SW))
temp.append(Cell(EDGE_DX, EDGE_DX, 0.0, PORO_M, INITIAL_SW))
MAP.append(temp)


#COORDS

COORDS = []

my = 0.0
for j in range(REAL_SIZE_Y + 1):
    mx = 0.0
    for i in range(REAL_SIZE_X + 1):
        temp = str(mx) + "     " + str(my)
        COORDS.append(temp  + "    {depth}    " + temp + "    {depth2}")
        if i < REAL_SIZE_X:
                mx += MAP[i][0].dx
    if j < REAL_SIZE_Y:
        my += MAP[0][j].dy

mstring = []

mstring.append("""-- ******************************************************************
-- Explicit simulation of a 2D fractured reservoir
-- Author: Benoit Decroux
-- MSc Petroleum Engineering
-- Imperial College London
-- 2012
-- ******************************************************************

----------------------------------------------------------------------
RUNSPEC
----------------------------------------------------------------------

TITLE
Explicit simulation of a 2D fractured reservoir

DIMENS
{sizeX}  {sizeY}  1  /
```

```
""")

if IS_OIL:
    mstring.append("OIL\n")
if IS_WATER:
    mstring.append("WATER\n")

mstring.append("""
METRIC

START
01 JAN 2000 00:00:00 /

UNIFIN
UNIFOUT

WELLDIMS
{nbWells}  1  2  {nbWells} /

NUPCOL
 20 /

NSTACK
 50 /

--NOSIM


----------------------------------------------------------------------
GRID
----------------------------------------------------------------------

GRIDFILE
  0 1 /

INIT

PINCH
/

GRIDUNIT
METRES /

SPECGRID
{sizeX}  {sizeY} 1  1  F /

COORDSYS
1 1/
""")

mstring.append("\nCOORD\n")
mstring.append("\n".join(COORDS))
mstring.append("\n/\n")

mstring.append("""
ZCORN
{nbCells*4}*{depth}
{nbCells*4}*{depth2}
/
""")

mstring.append("\nPERMX\n")
mstring.append(propGeneration(PERMEABILITY))
mstring.append("\n/\n")

mstring.append("""
COPY
PERMX   PERMY  /
PERMX   PERMZ  /
/

MULTIPLY
PERMZ   0.5/
/
""")

mstring.append("\nPORO\n")
mstring.append(propGeneration(POROSITY))
```

```
mstring.append("\n/\n")

mstring.append("""
----------------------------------------------------------------------
PROPS
----------------------------------------------------------------------

DENSITY
--Oil Water  Gas (kg/m3)
897  1000    1.5 /
""")

if IS_WATER:
    mstring.append("""
PVTW
--Pres(bars) Bw   Cw(bar-1)  mu(cP)  Cvis
{pressure}   1    4.0E-5     0.5     0.0 /
""")

if IS_OIL:
    mstring.append("""
PVCDO
--Pres(bars) Bo   Co(bar-1)  mu(cP)  Cvis
{pressure}   1    4.0E-5     1.4     0.0 /
""")

mstring.append("""
ROCK
-- Pres(bars)    Cr(bar-1)
{pressure}       4.934E-5 /
""")

if IS_OIL and IS_WATER:
    mstring.append("""
SWOF
--Swat  Krw    Kro    Pc(bars)
  0.20  0.0    0.9    0.0
  0.30  0.125  0.788  0.0
  0.40  0.25   0.675  0.0
  0.50  0.375  0.563  0.0
  0.60  0.5    0.45   0.0
  0.70  0.625  0.338  0.0
  0.80  0.75   0.225  0.0
  0.90  0.875  0.113  0.0
  1.00  1.0    0.0    0.0
/
""")

mstring.append("""
----------------------------------------------------------------------
SOLUTION
----------------------------------------------------------------------

RPTSOL
  'RESTART=1' 'FIP=3' /

PRESSURE
{nbCells}*{pressure} /
""")

if IS_OIL and IS_WATER:
    mstring.append("\nSWAT\n")
    mstring.append(propGeneration(WATER_SATURATION))
    mstring.append("\n/\n")


mstring.append("""
----------------------------------------------------------------------
SUMMARY
----------------------------------------------------------------------

--Field quantities

FPR

FOPR
FWPR
FGPR
FVPR
```

```
FOPT
FWPT
FGPT
FVPT

FWIR
FGIR
FOIR
FVIR

FOIT
FWIT
FGIT
FVIT

FOIP
FWIP
FGIP

FWCT

FWPIR

FGOR
FGLR


--Well quantities
WBHP
/

WOPR
/
WWPR
/
WGPR
/
WVPR
/

WOPT
/
WWPT
/
WGPT
/
WVPT
/

WWIR
/
wGIR
/
WOIR
/
WVIR
/

WWIT
/
wGIT
/
WOIT
/
WVIT
/

WWCT
/
WGOR
/
WGLR
/

--Miscellaneous and output control keywords
RUNSUM
SEPARATE
----------------------------------------------------------------------
```

```
SCHEDULE
----------------------------------------------------------------------

TUNING
 3.5E-4  3.5E-4  3.5E-4  /
   /
 2*   50 /

RPTSCHED
  'RESTART=2' 'FIP=1' 'WELLS=5' 'CPU=1' 'NEWTON=1' /
""")

mstring.append("\nWELSPECS\n")
for well in WELLS:
    mstring.append(well.wellSpec())
mstring.append("/\n")

mstring.append("\nCOMPDAT\n")
for well in WELLS:
    mstring.append(well.compDat())
mstring.append("/\n")

mstring.append("\nWCONPROD\n")
for well in WELLS:
    mstring.append(well.wConProd())
mstring.append("/\n")

mstring.append("\nWCONINJE\n")
for well in WELLS:
    mstring.append(well.wConInje())
mstring.append("/\n")

def toHour(hour):
    if hour > 9:
        return str(hour)
    else:
        return "0" + str(hour)

mstring.append("\nDATES\n")
year = 2000
month = 0
day = 1
hour = 0
minute = 0
for second in range(1,60):
    mstring.append(str(day) + "   " + MONTHS[month] + "   " + str(year) + "   " + toHour(hour) + ":" + toHour(minute) + ":" +
toHour(second) + " /\n")

minute = 0
second = 0
for hour in range(1,24):
        mstring.append(str(day) + "   " + MONTHS[month] + "   " + str(year) + "   " + toHour(hour) + ":" + toHour(minute) + ":" +
toHour(second) + " /\n")

for day in range(2,21):
    for hour in range(0,24,6):
        mstring.append(str(day) + "   " + MONTHS[month] + "   " + str(year) + "   " + toHour(hour) + ":" + toHour(minute) + ":" +
toHour(second) + " /\n")

mstring.append("/\n\nEND\n\n")

final = "".join(mstring)

final = final.replace("{depth}", str(DEPTH))
final = final.replace("{depth2}", str(DEPTH + DZ))
final = final.replace("{sizeX}", str(REAL_SIZE_X))
final = final.replace("{sizeY}", str(REAL_SIZE_Y))
final = final.replace("{nbWells}", str(NB_WELLS))
final = final.replace("{nbCells}", str(REAL_NB_CELLS))
final = final.replace("{nbCells*4}", str(REAL_NB_CELLS*4))
final = final.replace("{pressure}", str(PRESSURE))

mFile = open(TITLE + ".data", "w")
mFile.write(final)
mFile.close()

print TITLE + ".data"
print "Done"
```

APPENDIX I: Program generating upscaled simulations

This program (written in the Python language) generates an Eclipse simulation file (.data file) for a simulation based on effective properties by reading the effective properties generated by Petrel.

```
# ****************************************************************
# Petrel simulation
# Author: Benoit Decroux
# MSc Petroleum Engineering
# Imperial College London
# 2012
# ****************************************************************

#*************************************************************************************************************
# ************************************  PHYSICAL PROPERTIES  ************************************
#*************************************************************************************************************

LENGTH_X = 720.0 #(m)
#DX_PETREL is the size of the grid for the input properties
DX_PETREL = LENGTH_X / 9.0 #(m)
#DX_UPSCALE is the size of the output grid. It can only be smaller than DX_PETREL
DX_UPSCALE = DX_PETREL #(m)

INCLUDE_TITLE = "Oda9"
TITLE = INCLUDE_TITLE + "X"

DX_WELL_ZONE = LENGTH_X/9.0 #(m)
DY_UPSCALE = DX_UPSCALE #(m)
DY_WELL_ZONE = DX_WELL_ZONE   #(m)
DY_PETREL = DX_PETREL   #(m)
DEPTH = 150.0 #(m)
DZ = 10.0 #(m)
PORO = 0.15
LENGTH_Y = LENGTH_X #(m)
#EDGE_DX = 1.0 #(m)
PRESSURE = 200.0 #(bars)
INJ_PRESSURE = PRESSURE + 50.0  #(bars)
PROD_PRESSURE = 1.0             #(bars)
WELL_DIAMETER = 0.2 #(m)
IS_WATER = True
IS_OIL = True
WELL_PERM = 2700.0 #(mD)
WELL_PORO = 0.02
INITIAL_SW = 0.2

PERM_FRAC_VER = 2.0E+05 #(mD)
PERM_FRAC_HOR = 1.0E+05 #(mD)
APERTURE = 0.1 #(m)

# *************************************************************************************************************
# *************************************************************************************************************
# *************************************************************************************************************


MONTHS = ["JAN", "FEB", "MAR", "APR", "MAY", "JUN", "JUL", "AUG", "SEP", "OCT", "NOV", "DEC"]
INJECTOR = "INJ"
PRODUCER = "PRO"
WATER = "WATER"
OIL = "OIL"
POROSITY = "PORO"
PERMEABILITY_X = "PERMX"
WATER_SATURATION = "SWAT"
PERMEABILITY_Y = "PERMY"
PERMEABILITY_Z = "PERMZ"

WELLS = []

class Point:
    def __init__(self, mX, mY):
        global CELL_WELL_ZONE_X
        global CELL_WELL_ZONE_Y
        global LENGTH_TRANS
        self.x = mX
        self.y = mY
        self.realX = self.x * CELL_WELL_ZONE_X + int(CELL_WELL_ZONE_X/2)
```

```
        self.realY = self.y * CELL_WELL_ZONE_Y + int(CELL_WELL_ZONE_Y/2)
        self.finalX = self.realX# + 1
        self.finalY = self.realY# + 1

class Cell:
    def __init__(self, mdX, mdY, mPermX, mPermY, mPermZ, mPoro, mSw):
        self.dx = mdX
        self.dy = mdY
        self.permX = mPermX
        self.permY = mPermY
        self.permZ = mPermZ
        self.poro = mPoro
        self.Sw = mSw

    def get(self, prop):
        global POROSITY
        global PERMEABILITY_X
        global PERMEABILITY_Y
        global PERMEABILITY_Z
        global WATER_SATURATION
        if prop == POROSITY:
            return self.poro
        if prop == PERMEABILITY_X:
            return self.permX
        if prop == PERMEABILITY_Y:
            return self.permY
        if prop == PERMEABILITY_Z:
            return self.permZ
        if prop == WATER_SATURATION:
            return self.Sw

class Well:
    def __init__(self, mX, mY, mType, mPhase, mControle, mPressure, mRate, mName, mDiameter, mFlag = "OPEN"):
        self.point = Point(mX, mY)
        self.type = mType
        self.phase = mPhase
        self.controle = mControle
        self.pressure = mPressure
        self.rate = mRate
        self.name = mName
        self.diameter = mDiameter
        self.flag = mFlag

    def wellSpec(self):
        mReturn = []
        mReturn.append(self.name)
        mReturn.append(self.type)
        mReturn.append(str(self.point.finalX + 1))
        mReturn.append(str(self.point.finalY + 1))
        mReturn.append("{depth}")
        mReturn.append(self.phase)
        mReturn.append("/\n")
        return "  ".join(mReturn)

    def compDat(self):
        mReturn = []
        mReturn.append(self.name)
        mReturn.append("2*  1  1")
        mReturn.append(self.flag)
        mReturn.append("2*")
        mReturn.append(str(self.diameter))
        mReturn.append("1*  0  1*  Z  /\n")
        return "  ".join(mReturn)

    def wConProd(self):
        global PRODUCER
        if self.type != PRODUCER:
            return ""
        mReturn = []
        mReturn.append(self.name)
        mReturn.append(self.flag)
        mReturn.append(self.controle)
        mReturn.append("4*")
        mReturn.append(str(self.rate))
        mReturn.append(str(self.pressure))
        mReturn.append("/\n")
        return "  ".join(mReturn)

    def wConInje(self):
```

```
        global INJECTOR
        if self.type != INJECTOR:
            return ""
        mReturn = []
        mReturn.append(self.name)
        mReturn.append(self.phase)
        mReturn.append(self.flag)
        mReturn.append(self.controle)
        mReturn.append(str(self.rate))
        mReturn.append("1*")
        mReturn.append(str(self.pressure))
        mReturn.append("/\n")
        return "  ".join(mReturn)

def propGeneration(prop):
    global REAL_SIZE_X
    global REAL_SIZE_Y
    global MAP

    temp = []
    count = 0
    temp2 = []
    for j in range(REAL_SIZE_Y):
        for i in range(REAL_SIZE_X):
            temp.append(str(MAP[i][j].get(prop)))
            count+= 1
            if count == 6:
                if len(temp2) > 0:
                    temp2.append("\n")
                temp2.append("  ".join(temp))
                temp = []
                count = 0

    if count>0:
        if len(temp2) > 0:
            temp2.append("\n")
        temp2.append("  ".join(temp))
        temp = []
        count = 0

    return "".join(temp2)

def extractProperty(mProperty, mFile):
    global PETREL_SIZE_X
    global PETREL_SIZE_Y

    mReturn = []
    for i in range(PETREL_SIZE_X):
        temp = []
        for j in range(PETREL_SIZE_Y):
            temp.append(0)
        mReturn.append(temp)

    indexKeyWord = mFile.index(mProperty)
    firstLine = mFile.index("\n", indexKeyWord + 1)
    secondLine = mFile.index("\n", firstLine + 1)
    mList = mFile[secondLine + 1 :].split()
    iterator = 0
    for i in range(PETREL_SIZE_X + 2):
        assert (float(mList[iterator]) < 0.0)
        iterator+=1

    for j in range(PETREL_SIZE_Y):
        assert (float(mList[iterator]) < 0.0)
        iterator+=1

        for i in range(PETREL_SIZE_X):
            mReturn[i][j] = float(mList[iterator])
            if (mReturn[i][j] < 0.0):
                mReturn[i][j] = 0.0
            iterator+=1

        assert (float(mList[iterator]) < 0.0)
        iterator+=1

    for i in range(PETREL_SIZE_X + 2):
        assert (float(mList[iterator]) < 0.0)
        iterator+=1
```

```
    return mReturn


#Initialisation

WELL_ZONE_SIZE_X = int(round(LENGTH_X / DX_WELL_ZONE))
WELL_ZONE_SIZE_Y = int(round(LENGTH_Y / DY_WELL_ZONE))
assert (WELL_ZONE_SIZE_X * DX_WELL_ZONE == LENGTH_X)
assert (WELL_ZONE_SIZE_Y * DY_WELL_ZONE == LENGTH_Y)

CELL_WELL_ZONE_X = int(round(DX_WELL_ZONE / DX_UPSCALE))
CELL_WELL_ZONE_Y = int(round(DY_WELL_ZONE / DY_UPSCALE))
assert (CELL_WELL_ZONE_X * DX_UPSCALE == DX_WELL_ZONE)
assert (CELL_WELL_ZONE_Y * DY_UPSCALE == DY_WELL_ZONE)

PETREL_SIZE_X = int(round(LENGTH_X / DX_PETREL))
PETREL_SIZE_Y = int(round(LENGTH_Y / DY_PETREL))
assert (PETREL_SIZE_X * DX_PETREL == LENGTH_X)
assert (PETREL_SIZE_Y * DY_PETREL == LENGTH_Y)

CELL_PETREL_X = int(round(DX_PETREL / DX_UPSCALE))
CELL_PETREL_Y = int(round(DY_PETREL / DY_UPSCALE))
assert (abs(CELL_PETREL_X * DX_UPSCALE - DX_PETREL)<0.001)
assert (abs(CELL_PETREL_Y * DY_UPSCALE - DY_PETREL)<0.001)


UPSCALE_SIZE_X = int(round(LENGTH_X / DX_UPSCALE))
UPSCALE_SIZE_Y = int(round(LENGTH_Y / DY_UPSCALE))
assert (UPSCALE_SIZE_X * DX_UPSCALE == LENGTH_X)
assert (UPSCALE_SIZE_Y * DY_UPSCALE == LENGTH_Y)

REAL_SIZE_X = UPSCALE_SIZE_X# + 2
REAL_SIZE_Y = UPSCALE_SIZE_Y# + 2

REAL_NB_CELLS = REAL_SIZE_X * REAL_SIZE_Y

#Reading properties file

import os
os.chdir("C:\Users\BCD111\Desktop\Explicit")
mFile = open(INCLUDE_TITLE + ".txt", "r")
propertiesFile = mFile.read()
mFile.close()

mapPoro = extractProperty(POROSITY, propertiesFile)
mapPermX = extractProperty(PERMEABILITY_X, propertiesFile)
mapPermY = extractProperty(PERMEABILITY_Y, propertiesFile)
mapPermZ = extractProperty(PERMEABILITY_Z, propertiesFile)

#Wells creation

phaseProduction = OIL
if IS_OIL == False:
    phaseProduction = WATER

phaseInjection = WATER
if IS_WATER == False:
    phaseInjection = OIL

isShut = "OPEN"

#def __init__( mX, mY, mType, mPhase, mControle, mPressure, mRate, mName , mDiameter):

#WELLS.append(Well(0, 0, INJECTOR, phaseInjection, "BHP", INJ_PRESSURE, "1*", "INJ1", WELL_DIAMETER, isShut))

#WELLS.append(Well(FACIES_SIZE_X-1, FACIES_SIZE_Y-1, INJECTOR, phaseInjection, "BHP", INJ_PRESSURE, "1*", "INJ3", WELL_DIAMETER, isShut))

#WELLS.append(Well(0, FACIES_SIZE_Y-1, INJECTOR, phaseInjection, "BHP", INJ_PRESSURE, "1*","INJ2", WELL_DIAMETER, isShut))

#WELLS.append(Well(FACIES_SIZE_X-1, 0, INJECTOR, phaseInjection , "BHP", INJ_PRESSURE, "1*", "INJ4", WELL_DIAMETER, isShut))

WELLS.append(Well(0, int(FACIES_SIZE_Y/2), INJECTOR, phaseInjection, "BHP", INJ_PRESSURE, "1*", "INJ1", WELL_DIAMETER, isShut))

WELLS.append(Well(FACIES_SIZE_X-1, int(FACIES_SIZE_Y/2), INJECTOR, phaseInjection, "BHP", INJ_PRESSURE, "1*", "INJ3", WELL_DIAMETER, isShut))
```

```
WELLS.append(Well(int(FACIES_SIZE_X/2), FACIES_SIZE_Y-1, INJECTOR, phaseInjection , "BHP", INJ_PRESSURE, "1*", "INJ2", WELL_DIAMETER, isShut))

WELLS.append(Well(int(FACIES_SIZE_X/2), 0, INJECTOR, phaseInjection , "BHP", INJ_PRESSURE , "1*", "INJ4", WELL_DIAMETER, isShut))

WELLS.append(Well(int(FACIES_SIZE_X/2), int(FACIES_SIZE_Y/2), PRODUCER, phaseProduction, "BHP", PROD_PRESSURE, "1*", "PROD", WELL_DIAMETER))

NB_WELLS = len(WELLS)

#Upscaled map creation
Area=0
MAP = []
temp = []
#temp.append(Cell(EDGE_DX, EDGE_DX, 0.0, 0.0, 0.0, PORO, INITIAL_SW))
#for y in range(PETREL_SIZE_Y):
#    for j in range(CELL_PETREL_Y):
#        temp.append(Cell(EDGE_DX, DY_UPSCALE, mapPermX[0][y], 0.0, 0.0, PORO, INITIAL_SW))

#temp.append(Cell(EDGE_DX, EDGE_DX, 0.0, 0.0, 0.0, PORO, INITIAL_SW))
#MAP.append(temp)

for x in range(PETREL_SIZE_X):
    for i in range(CELL_PETREL_X):
        temp = []
        #temp.append(Cell(DX_UPSCALE, EDGE_DX, 0.0, mapPermY[x][0], 0.0, PORO, INITIAL_SW))
        for y in range(PETREL_SIZE_Y):
            for j in range(CELL_PETREL_Y):

                #Check if it is a well zone
                isWellZone = False
                for well in WELLS:
                    if abs(x * CELL_PETREL_X + i - well.point.realX) <= CELL_WELL_ZONE_X/2 and abs(y * CELL_PETREL_Y + j - well.point.realY) <= CELL_WELL_ZONE_Y/2:
                        isWellZone = True
                        break

                if isWellZone:
                    permX = WELL_PERM
                    permY = WELL_PERM
                    permZ = WELL_PERM * 0.5
                    poro = WELL_PORO
                else:
                    permX = mapPermX[x][y]
                    permY = mapPermY[x][y]
                    permZ = mapPermZ[x][y]
                    poro = mapPoro[x][y]

                if permX > 0 or permY > 0:
                    Area+=DX_UPSCALE*DY_UPSCALE*poro

                temp.append(Cell(DX_UPSCALE, DY_UPSCALE, permX, permY, permZ, poro, INITIAL_SW))

        #temp.append(Cell(DX_UPSCALE, EDGE_DX,0.0, mapPermY[x][-1], 0.0, PORO, INITIAL_SW))
        MAP.append(temp)

#temp = []
#temp.append(Cell(EDGE_DX, EDGE_DX, 0.0, 0.0, 0.0, PORO, INITIAL_SW))
#for y in range(PETREL_SIZE_Y):
#    for j in range(CELL_PETREL_Y):
#        temp.append(Cell(EDGE_DX, DY_UPSCALE, mapPermX[-1][y], 0.0, 0.0, PORO, INITIAL_SW))

#temp.append(Cell(EDGE_DX, EDGE_DX, 0.0, 0.0, 0.0, PORO, INITIAL_SW))
#MAP.append(temp)

print "Initial oil volume =" + str(Area*(1-INITIAL_SW)*DZ) + "m3"

#COORDS

COORDS = []

my = 0.0
for j in range(REAL_SIZE_Y + 1):
    mx = 0.0
    for i in range(REAL_SIZE_X + 1):
        temp = str(mx) + "    " + str(my)
        COORDS.append(temp  + "    {depth}    " + temp + "    {depth2}")
        if i < REAL_SIZE_X:
```

```
                mx += MAP[i][0].dx
    if j < REAL_SIZE_Y:
        my += MAP[0][j].dy

mstring = []
mstring.append("""-- ********************************************************************
-- Upscaled model simulation
-- Author: Benoit Decroux
-- MSc Petroleum Engineering
-- Imperial College London
-- 2012
-- ********************************************************************
----------------------------------------------------------------------
RUNSPEC
----------------------------------------------------------------------
TITLE
Upscaled simulation of a 2D fractured reservoir

DIMENS
{sizeX}  {sizeY}  1  /

""")

if IS_OIL:
    mstring.append("OIL\n")
if IS_WATER:
    mstring.append("WATER\n")

mstring.append("""
METRIC

START
01 JAN 2000 00:00:00 /

UNIFIN
UNIFOUT

WELLDIMS
{nbWells}  1  2  {nbWells} /

--NOSIM

----------------------------------------------------------------------
GRID
----------------------------------------------------------------------

GRIDFILE
  0 1 /

INIT

PINCH
/

GRIDUNIT
METRES /

SPECGRID
{sizeX}  {sizeY} 1  1  F /

COORDSYS
1 1/
""")

mstring.append("\nCOORD\n")
mstring.append("\n".join(COORDS))
mstring.append("\n/\n")

mstring.append("""
ZCORN
{nbCells*4}*{depth}
{nbCells*4}*{depth2}
/

""")

mstring.append("\nPERMX\n")
mstring.append(propGeneration(PERMEABILITY_X))
mstring.append("\n/\n")
```

```
mstring.append("\nPERMY\n")
mstring.append(propGeneration(PERMEABILITY_Y))
mstring.append("\n/\n")

mstring.append("\nPERMZ\n")
mstring.append(propGeneration(PERMEABILITY_Z))
mstring.append("\n/\n")

mstring.append("\nPORO\n")
mstring.append(propGeneration(POROSITY))
mstring.append("\n/\n")

mstring.append("""
----------------------------------------------------------------------
PROPS
----------------------------------------------------------------------

DENSITY
--Oil Water  Gas (kg/m3)
897  1000    1.5 /
""")

if IS_WATER:
    mstring.append("""
PVTW
--Pres(bars) Bw   Cw(bar-1)  mu(cP)  Cvis
{pressure}   1    4.0E-5     0.5     0.0 /
""")

if IS_OIL:
    mstring.append("""
PVCDO
--Pres(bars) Bo   Co(bar-1)  mu(cP)  Cvis
{pressure}   1    4.0E-5     1.4     0.0 /
""")

mstring.append("""
ROCK
-- Pres(bars)    Cr(bar-1)
{pressure}       4.934E-5 /
""")

if IS_OIL and IS_WATER:
    mstring.append("""
SWOF
--Swat  Krw    Kro    Pc(bars)
  0.20  0.0    0.9    0.0
  0.30  0.125  0.788  0.0
  0.40  0.25   0.675  0.0
  0.50  0.375  0.563  0.0
  0.60  0.5    0.45   0.0
  0.70  0.625  0.338  0.0
  0.80  0.75   0.225  0.0
  0.90  0.875  0.113  0.0
  1.00  1.0    0.0    0.0
/
""")
mstring.append("""
----------------------------------------------------------------------
SOLUTION
----------------------------------------------------------------------
RPTSOL
  'RESTART=1' 'FIP=3' /
PRESSURE
{nbCells}*{pressure} /
""")
if IS_OIL and IS_WATER:
    mstring.append("\nSWAT\n")
    mstring.append(propGeneration(WATER_SATURATION))
    mstring.append("\n/\n")

mstring.append("""
----------------------------------------------------------------------
SUMMARY
----------------------------------------------------------------------
--Field quantities

FPR
```

```
FOPR
FWPR
FGPR
FVPR

FOPT
FWPT
FGPT
FVPT

FWIR
FGIR
FOIR
FVIR

FOIT
FWIT
FGIT
FVIT

FOIP
FWIP
FGIP

FWCT

FWPIR

FGOR
FGLR


--Well quantities
WBHP
/

WOPR
/
WWPR
/
WGPR
/
WVPR
/

WOPT
/
WWPT
/
WGPT
/
WVPT
/

WWIR
/
wGIR
/
WOIR
/
WVIR
/

WWIT
/
wGIT
/
WOIT
/
WVIT
/

WWCT
/
WGOR
/
WGLR
/
```

```
--Miscellaneous and output control keywords
RUNSUM
SEPARATE

----------------------------------------------------------------------
SCHEDULE
----------------------------------------------------------------------

RPTSCHED
  'RESTART=2' 'FIP=1' 'WELLS=5' 'CPU=1' 'NEWTON=1' /
""")

mstring.append("\nWELSPECS\n")
for well in WELLS:
    mstring.append(well.wellSpec())
mstring.append("/\n")

mstring.append("\nCOMPDAT\n")
for well in WELLS:
    mstring.append(well.compDat())
mstring.append("/\n")

mstring.append("\nWCONPROD\n")
for well in WELLS:
    mstring.append(well.wConProd())
mstring.append("/\n")

mstring.append("\nWCONINJE\n")
for well in WELLS:
    mstring.append(well.wConInje())
mstring.append("/\n")

def toHour(hour):
    if hour > 9:
        return str(hour)
    else:
        return "0" + str(hour)

mstring.append("\nDATES\n")
year = 2000
month = 0
day = 1
hour = 0
minute = 0
for second in range(1,60):
    mstring.append(str(day) + "   " + MONTHS[month] + "   " + str(year) + "   " + toHour(hour) + ":" + toHour(minute) + ":" +
toHour(second) + " /\n")

minute = 0
second = 0
for hour in range(1,24):
        mstring.append(str(day) + "   " + MONTHS[month] + "   " + str(year) + "   " + toHour(hour) + ":" + toHour(minute) + ":" +
toHour(second) + " /\n")

for day in range(2,21):
    for hour in range(0,24,6):
        mstring.append(str(day) + "   " + MONTHS[month] + "   " + str(year) + "   " + toHour(hour) + ":" + toHour(minute) + ":" +
toHour(second) + " /\n")

mstring.append("/\n\nEND\n\n")

final = "".join(mstring)
final = final.replace("{depth}", str(DEPTH))
final = final.replace("{depth2}", str(DEPTH + DZ))
final = final.replace("{sizeX}", str(REAL_SIZE_X))
final = final.replace("{sizeY}", str(REAL_SIZE_Y))
final = final.replace("{nbWells}", str(NB_WELLS))
final = final.replace("{nbCells}", str(REAL_NB_CELLS))
final = final.replace("{nbCells*4}", str(REAL_NB_CELLS*4))
final = final.replace("{pressure}", str(PRESSURE))

mFile = open(TITLE + ".data", "w")
mFile.write(final)
mFile.close()

print TITLE + ".data"
print "Done"
```