University of Alberta

Core and Field Scale Modeling of Miscible Injection Processes in Fractured Porous Media Using Random Walk and Particle Tracking Methods

by

Ekaterina Stalgorova

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Master of Science

in

Petroleum Engineering

Department of Civil and Environmental Engineering

Fall 2011
Edmonton, Alberta

Examining Committee

Dr. Tayfun Babadagli, Civil and Environmental Engineering

Dr. Juliana Leung, Civil and Environmental Engineering

Dr. Ahmed Bouferguene, Campus Saint-Jean

My work is dedicated to

My grandfather, who always wanted me to become a scientist, but still loves me though I became an engineer.

ABSTRACT

In this thesis, we introduced and applied non-classical techniques to simulate miscible flow in fractured porous media.

First, the Random Walk technique was modified to simulate miscible displacement in 2D fractured porous media at the lab-scale. The method was validated using a series of laboratory solvent injection experiments obtained from literature.

Then, this model was modified to apply it for field-scale simulations and a sensitivity analysis was performed to identify the most critical parameters of the process. To validate the model, a tracer test done in the naturally fractured Midale field was used. Subsequently, the same fracture network system, which was calibrated against the tracer test results, was used to simulate the pilot CO_2 injection applied in the same field. In this exercise, additional modifications to the algorithm were made including diffusive transfer between matrix and fracture.

In the last part of the thesis, an approach was presented to scale up the production profiles obtained for a fractured reservoir. The exponents in the scaling equation were correlated to the fracture network properties such as fracture density, box-counting fractal dimension, mass fractal dimension, and fracture volume ratio.

ACKNOWLEDGMENTS

I would like to thank my supervisor Dr. Tayfun Babadagli, whose belief in my capability made this thesis possible. I appreciate his flexible, yet explicit guidance and his support throughout the research study.

I also thank Vahapcan Er and Dimitry Bogatkov for providing detailed data about their research. Maryia Kazakevich and Denise Thornton from the UofA visualization group helped with 3D visualization of my data, which is appreciated a lot.

This research was partly funded by an NSERC Grant (No: G121210595). The funds for the equipment used were obtained from the Canadian Foundation for Innovation (CFI) (Project # 7566) and the University of Alberta. We would like to thank Schlumberger for supplying the ECLIPSE reservoir simulation suite. We are also thankful to Apache Canada Ltd for providing field and well data, and permission to use them in this research.

It is a pleasure to express my gratitude to all my family back home for their love and encouragement. In particular, to my lovely little sister, who often cheers me up in rainy days, to my mom, who is always ready to defend me even if I'm wrong and to my brother, who patiently helped me to get started with the coding part of research.

I am heartily thankful to Varun, who was always there for me; for his unconditional support and fruitful discussions.

The last word of acknowledgement is to Jesus Christ, my Lord and my Saviour for giving me life, health and intelligence, without which this research would not be possible.

Table of Contents

LIST OF TABLES

LIST OF FIGURES

NOMENCLATURE

Chapter 2

A = area of the cross section

C = solvent concentration

D = diffusivity coefficient

D_o = diffusivity coefficient used in the model for oil walkers

D_s = diffusivity coefficient used in the model for solvent walkers

k = permeability

N = number of walkers used in simulation

P = pressure

PV = pore volume of the model

q = number of walkers added to model at each time step

Q = injection rate

r = vector corresponding to walker location

t = time

Δt = length of one time step

v = velocity vector

v_w = volume of one walkers

x,y = walkers coordinates

z_x, z_y = random numbers, driven from normal distribution with mean equal to zero and standard deviation equal to 1

μ = viscosity

ρ = density

ω= mixing parameter for viscosity calculation

Chapter 3

A = crossection area for fracture

C = concentration

C_h = (spacing between off-trend fractures)/(spacing between on-trend fractures)

c_i = tracer injection concentration

D = dispersion coefficient

d = depth of the vertex

h = fracture height

k = permeability

L = edge length

L_{xy} = length of the edge between vertexes x and y

m = total mass of injected tracer

m_p = mass of one particle

N = number of particles used for simulation

P = pressure

$PermF_{ontrend}$ = permeability for main fracture set

$PermF_{offtrend}$ = permeability for secondary fracture set

$PermM$ = matrix permeability

q_i = tracer injecting rate

t = time

t_{MN} = time for M^{th} particle at N^{th} step

t_{ti} = tracer injecting time

v = velocity

v_{xy}=velocity of flow vertexes x and y

W =fracture width

$W_{ontrend}$ = width for main fracture set

$W_{offtrend}$ = width for main fracture set

x = particle location

z = random number, driven from normal distribution with mean equal to zero and standard deviation equal to 1

μ = viscosity

ρ = density

Ψ = pressure potential

Ψ_x = pressure potential at vertex x

Chapter 4

A = cross section area for fracture

A_e = effective cross section area for flow through matrix

C = concentration

C_h = (spacing between off-trend fractures)/(spacing between on-trend fractures)

D = dispersion coefficient

d = depth of the vertex

h = fracture height

k = permeability

L = edge length

L_{xy} = length of the edge between vertexes x and y

N = number of particles used for simulation

P = pressure

p_m = permeability multiplier

q_{co2} = CO_2 injecting rate

q_w = water injecting rate

Rad = maximum distance within matrix flow can happen

t = time

t_{MN} = time for M^{th} particle at N^{th} step

t_{co2} = CO_2 injection duration

t_w = water injection duration

v = velocity

v_{xy}=velocity of flow vertexes x and y

W =fracture width

$W_{ontrend}$ = width for main fracture set

$W_{offtrend}$ = width for main fracture set

x = particle location

z = random number, driven from normal distribution with mean equal to zero and standard deviation equal to 1

μ = viscosity

ρ = density

Ψ = pressure potential

Ψ = pressure potential at vertex x

Chapter 5

D_{bc}= box counting fractal dimension

D_m = mass fractal dimension

$P(t)$ = probability of the particle to reach the production well with traveling time t

$P_{mp} = P(t_{mp})$ the highest possible probability

R = distance between injecting and producing wells

sp = spacing between fractures

t_{mp} = the most probable traveling time

V_f = (volume of fractures in the system)/(total volume of the system). Volume fraction of fractures.

α = scaling parameter, exponent, relating R and t_{mp}

β = scaling parameter, exponent, relating R and P_{mp}

1. Introduction

1.1 Overview

Hydrocarbons are one of Earth's most important energy sources and demand for energy is increasing. Unfortunately, so-called ‘easy oil and gas’ have been produced for more than hundred years, and there is not much left for generations to come. For that reason, the petroleum industry now has to focus on: a) exploring unconventional resources, b) producing from complex reservoirs, and c) applying various enhanced oil recovery (EOR) techniques that allow production of residual oil from mature reservoirs.

Production from naturally fractured reservoirs (NFR) is one example of unconventional (and complex) source of hydrocarbons. This type of reservoirs contains a substantial amount of oil and gas reserves. It was estimated that more than 60% of the world’s proven oil reserves and about 40% of the world’s proven gas reserves are trapped in NFRs (Montaron 2008).

NFRs are believed to entail higher risks than conventional reservoirs because fluid behaviour is defined by the fracture network, while information regarding its geometry and properties is often incomplete. This creates a great uncertainty in the number of parameters needed for accurate prediction of the hydrocarbon production. In addition to that, even if the fracture network was properly described, simulating the flow of the fluid in such a complicated model is not an easy task. The presence of two contrasting media – matrix and fracture, as well as the irregular geometry of the fracture system, require either unreasonable computation time or significant simplifications in the media description.

Nevertheless, proper NFR characterization and oil recovery prediction are important at any stage of development, especially for risky and investment intensive EOR applications. High potential risks of such applications can and should be assessed and minimized with the help of technology.

A number of techniques to model fluid flow in fractured porous media are used in industry as well as for research purposes. However, there is still no

universal solution that can be applied to any NFR, as each technique has its own advantages, and its own drawbacks and limitations.

This research focuses on non-classical ways to simulate fluid transport in fractured porous media. This approach was adapted because non-classical techniques have capability to capture the complexity of fracture domain as opposed to classical continuum models.

Chapter 2 introduces the Random Walk algorithm and its modification to simulate miscible flow in fractured porous media. This modification is applicable for 2D lab-scale models, for the cases of horizontal or vertical flow. We validated the suggested algorithm by comparing the results of simulation with visual and production data from a series of miscible solvent injection experiments.

In Chapter 3, further modification of the algorithm, namely Random Walk Particle Tracking (RWPT), was introduced for the field-scale simulation. To be able to simulate the flow of fluid through complex fractured media, we first converted the fracture network into a graph, and then used only this graph for further simulation. This approach allows preserving information about the fracture network connectivity while significantly decreasing computational time. For validation, a series of tracer test results from the Midale field in Canada was used. A fracture network model was generated based on geological data, and then calibrated against tracer test results using RWPT. Additionally, Chapter 3 includes a sensitivity analysis to identify the importance of different parameters for the simulation results.

In Chapter 4, we further improved the RWPT algorithm to simulate CO_2 injection in the same reservoir as in Chapter 3. We used the fracture network calibrated against tracer test results as described in Chapter 3. A history match with the actual CO_2 pilot flooding results is presented, as well as a sensitivity study.

In Chapter 5, we studied how production profile curves obtained as a result of the RWPT simulation are changed when the distance between the injection and production wells is varied. Similarities of these curves obtained at different scales suggested a way to up-scale them. This chapter describes the scaling methodology

and define a scaling relationship for fractured systems. It also illustrates how scaling parameters depend on different fracture network properties such as fractal dimensions of the network and fracture density.

As this is a paper-based thesis, each chapter contains its own conclusions. The major contributions of this study are highlighted in Chapter 6. Also, the limitations of the suggested algorithms and recommendations for future works are presented in this chapter.

1.2 Literature review

The flow of fluid in naturally fractured reservoirs (NFR) is commonly studied in enhanced oil recovery, groundwater contamination, and CO_2 sequestration in oil reservoirs. Because of two contrasting media – matrix and fracture, it is difficult to predict fluid dynamics. This initially requires an accurate description of matrix and fracture characteristics within the reservoir, and then mapping its properties. After this stage, named static modeling, the next stage is to simulate how fluid will move in the described media. This is referred to as dynamic modeling. Both stages are challenging tasks and extensive work has been done in this area by many researchers.

Literature relevant to this thesis is reviewed in the subsections below.

1.2.1 Classical modeling for fractured reservoirs

One of the traditional ways to simulate flow in NFR is a single continuum approach. In this approach fracture networks are mapped based on geological and geophysical data, such as well measurements, seismic maps and outcrop studies. Then, a simulation grid is created, and fractures placed in each block are replaced by equivalent parameters (porosity and permeability). Once this is done, standard finite-difference calculations are used for further modeling.

Calculating equivalent permeability values for a given fracture network is a complex task and can be time consuming, especially in highly fractured reservoirs. Studies on this are available in the literature (Long et al. 1985; Lough et al. 1997; Jafari and Babadagli 2011). Software packages which convert fracture

networks into equivalent properties are also available in the industry (FracaFlow, Petrel, FracMan®).

Although this type of modeling is capable of representing the complexity of the fracture network to greater extent, information about separate fractures is still lost while averaging (Bogatkov and Babadagli 2010). Another limitation of this approach is that matrix-fracture interaction is not captured properly. Computational time is also an issue for single continuum modeling.

Another traditional method of simulating flow in NFRs is the dual continuum model introduced by Barenblatt and Zheltov (1960) and modified by Warren and Root (1963). In this kind of model, a second continuum is added to represent both storage and permeability characteristics of the reservoir. In dual continuum model, fluid flow occurs in the fracture network, and the matrix feeds the fractures. Interaction between matrix and fracture is described using a transfer function, and it is important that the transfer function captures all the physical aspects of the process (gravity, viscous and capillary forces as well as diffusion). Extensive research was performed on transfer function descriptions and further improvements of the dual porosity model (Kazemi and Merrill 1979; Sarma and Aziz 2004; Di Donato et al. 2007; Lu et al. 2008). Applications of the dual continuum model for different purposes have been presented in the literature over the last five decades (Ganzer 2002; Al-Khlaifat and Arastoopour 2003; Bogatkov 2008).

This type of modeling captures the physics of matrix-fracture transfer, but often fails to represent the complexity of fracture networks. This is because the model is based on an orthogonal representation of the fracture system. For example, it is not possible to model a reservoir which has a small number of large fractures dominating the flow.

A common issue for single continuum and dual continuum models is that certain information is almost always lost during averaging, the most critical information being the connectivity of the fracture network.

1.2.2 Non-classical modeling for fractured reservoirs

To incorporate all critical parameters accurately in the static models, one has to define a detailed network model. The Discrete Fracture Network (DFN) approach involves describing each fracture separately, with physical and geometrical properties (such as storage, size and orientation) assigned. A DFN model typically combines deterministic and stochastic approaches: bigger features are modeled deterministically using well and seismic information, while smaller fractures are generated stochastically, sometimes using concepts of geomechanics. To use this fracture network in the dynamic simulation, one can either up-scale it to equivalent reservoir properties (Cacas et al. 1990; Cacas et al. 2001; Bogatkov and Babadagli 2009) and convert it into permeabilities for dual continuum model (Gong et al. 2008; Dershowits et al. 2000) or use the model in all its complexity. The latter often involves unstructured gridding, converting the fracture network into a finite element mesh and applying semi-analytical or finite-element calculations. Such models have limitations in terms of applicability and capturing matrix-fracture interaction. Beyond that, this kind of modeling may require long computational times. Examples of DFN modeling are available in the literature (Doe et al. 1990).

Another class of simulation methods is called Discrete Fracture Modeling (DFM). In comparison to the DFN method, the fractures and the matrix are discretized, which eliminates the use of fracture-matrix transfer function. Instead, state unknowns (pressure and composition) are assumed to be the same in the fracture and in the adjacent matrix. The DFM allows simulating miscible and immiscible flow as well as multiphase flow. However, because of the excessive discretization, the modeling of each fracture requires a large number of finite volumes. This results in unreasonable computational time, which is practically impossible to handle, for field scale simulations (which have thousands of fractures). Descriptions of the DFM algorithms and examples are available in several recent publications (Hoteit and Firoozabadi 2004; Karimi-Fard et al. 2004).

Percolation theory is also used for simulating flow of the fluid in fractured and heterogeneous systems, including flow through highly heterogeneous media. Models based on the percolation theory proved to be successful in describing reservoir connectivity, predicting breakthrough time, and reservoir up-scaling (Stanley et al. 1999; Sahimi and Mehrabi 1999). The percolation theory describes connectivity of the reservoir as a function of its geological heterogeneities. Coefficients of the function are defined from small scale simulations and used for the bigger scale afterwards (King et al. 1999; Dokholyan et al. 1999). However, the use of the percolation theory is limited to certain cases as it is based on two assumptions. The first one is that the rock is either permeable or non-permeable, and that flow takes place only in permeable rock, which is not always the case for fractured reservoirs. In general, a considerable amount of oil is produced from the low permeable matrix and this requires the addition of a matrix-fracture transfer function into the model. The second assumption is that the pressure field and the mobility are not changed during injection. Because of this limitation, percolation-based modeling is not applicable for cases where the viscosity of the displaced fluid is much higher than the viscosity of the displacing fluid, or for the case where the injection and production rates are changing.

There are more non-classical algorithms which are used for fluid flow simulation. Invasion percolation, Diffusion Limited Aggregation, and the Lattice Boltztmann Method are some of the examples. In this literature review we are not covering all of them and will focus on one class of algorithms called Random Walk (or Random Walk Particle Tracking) techniques.

1.2.3 Review of Random Walk (Particle Tracking) methods

There is a number of techniques used in fluid flow simulation referred to as Random Walk (RW) or Random Walk Particle Tracking methods (RWPT). Although they have similar names, these methods are significantly different from each other. Various RW(PT) algorithms have one concept in common: they model fluid flow as the movement of a large number of particles. Movement of each particle involves some randomness, however, the probability of the particle's

movement to have certain lengths and direction is defined by the physics of the process.

Pearson and Blakeman (1906) presented one of the earliest Random Walk studies. They did not apply Random Walk techniques to the case of fluid flow, but investigated the probability distribution for particle locations when particles are moving randomly in space. Chandrasekhar (1943) extended the Random Walk theory to the case of reflection and adsorption, and applied the RW to study the Brownian motion. Schreidegger (1954) applied the RW concept to simulate fluid flow in isotropic homogeneous porous media and showed how the effect of dispersion can be modeled. Saffman (1959) continued development of the same concept and derived longitudal dispersion as a function of molecular diffusivity and pore-scale parameters of the media.

The Continuous Time Random Walk (CTRW) algorithm was introduced in 1973, and has been developed extensively over the last four decades. Classical RW modeling uses timesteps of some fixed duration, and tracks how particle locations change within each timestep. However, selecting the size of the timestep is a challenge since the particle may have a very high or very low velocity while moving through the media (variations in the permeability of the media is one of the reasons). It is desirable to have a small timestep when the velocity is high, but not for the low velocity situations. CTRW takes care of this problem by assigning a distribution of retention times, i.e. the probability of a particle to make a step of certain length within a certain time interval (where time is a continuous variable, not just a sequence of timesteps of fixed duration). An excellent review paper on the CTRW was provided by Berkowitz et al. (2006).

The RW algorithm can be combined with classical numerical or analytical solutions. For example, in a recent work by Roubinet et al. (2010), flow through the fracture network was modeled using the RW approach, while matrix-fracture interaction follows a known analytical solution.

The forming of viscous fingering is a physical process that happens as a result of microheterogenities and small scale perturbations. For that reason, a modeling algorithm which involves randomness is especially suitable for modeling viscous

displacement. Araktingi and Orr (1990) successfully used the RW algorithm to simulate viscous displacement.

1.2.4 Fracture network fractal properties

Naturally fractured reservoirs normally have thousands of fractures of various lengths, apertures, and orientations. The complete description of such a fracture network requires the description of each single fracture through its physical and geometrical properties. For practical reasons, it is desirable to have a way to describe the fracture network without giving all the details for each single fracture. For example, a fracture network can be described by giving a distribution functions for fracture lengths, apertures, spacing, and orientations.

It is important to ensure that the parameters used for fracture network description are sufficient to capture the critical properties of the network, such as its connectivity or equivalent permeability. Irregularity and heterogeneity should be included at any scale at the characterization stage as well. In an attempt to include all the complexities in fracture network characterization, many different approaches have been tested and presented, fractal theory is being one of the most useful one.

It was observed (Barton and Larsen 1985; La Pointe 1988) that natural fracture patterns are fractal objects, i.e., they are reminiscent of each other statistically at different scales. Due to these observations, fractal theory became popular in fracture network characterization.

One example of using fracture network fractal characteristics to estimate fracture network equivalent permeability is given by La Pointe (1988). He stated that flux through a discrete fracture network is linearly proportional to its mass fractal dimension. Jafari and Babadagli (2009) investigated the effect of various fractal characteristics on the fracture network permeability. They showed that the box-counting fractal dimension of fracture intersection points and fracture lines are the most influential parameters on fracture network permeability.

Other examples of the successful use of fractal theory for modeling naturally fractured reservoirs are available in the literature (Halvin and Ben-Avraham 1987;

Beler 1990; Chang and Yortsos 1990; Acuna and Yortsos 1991; Acuna et al. 1992; Berkowitz and Hadad 1997).

Different fractal dimensions were used to characterize different characteristic so fracture networks. In this thesis, we used box-counting method (Mandelbrot 1982), mass dimension (Bunde and Havlin 1995) and fractal dimension by construction (Sahimi 1993). Descriptions for these methods are presented in Chapter 5.

1.3 Statement of the problem

Naturally fractured reservoirs (NFRs) contain more than half of the world's proven oil reserves. Producing these reserves is a challenging but indispensable task, as conventional sources of hydrocarbons are nearly exhausted.

This type of reservoirs presents unique and specialized challenges to hydrocarbon extraction, mostly due to their high heterogeneity and complexity. However, it is vital to predict reservoir behaviour (including production rates and breakthrough times) to avoid risks due to remarkable investment for field scale applications. This is particularly important for enhanced oil recovery operations which involve high risk and cost.

Apart from petroleum engineering applications, proper modeling of fluid flow in a fractured media is also important in other engineering disciplines such as groundwater contamination, nuclear waste disposal, and CO_2 sequestration in underground reservoirs due to the risk caused by environmental and health issues.

As it is described in the 'Literature Review section', there are a number of modeling techniques for naturally fractured reservoirs. However, the classical techniques (those, which are traditionally used in industry) have certain limitations and often fail to represent the complex structure of fracture networks and thereby, to capture the actual reservoir behaviour. Non-classical techniques were proposed as an alternative, as described in the previous section, possessing certain challenges as well.

The advantageous aspect of the non-classical models is their capability to represent the complexity of fracture networks. Despite numerous studies on non-

classical methodologies for the simulation of NFRs, as of today, non-classical simulation techniques are still in the scientific development stage and not yet advanced enough to be used in industry. Additional efforts are needed to identify which non-classical simulation technique can be used for routine modeling purposes. This study is one more contribution to the research on non-classical modeling.

The purpose of the thesis is to suggest non-classical simulation techniques for three particular cases associated with fractured systems (miscible flooding at laboratory scale, tracer tests, and miscible CO_2 injection).

1.4 Solution methodology

We used previously published laboratory experiments and field cases as a starting point of our research. We scrutinized these data and identified which aspects of the physics of the process are the most essential ones to be considered in the modeling studies. Then, we proposed an algorithm to capture these aspects.

For example, in Chapter 2, the RW algorithm is applied to model a series of miscible solvent injection experiments. Injection rates were relatively small; therefore diffusion played a significant role in the displacement process. For some of the experiments, heavy oil was used as a displaced fluid and a high viscosity ratio affected the shape of the displacement patterns. Taking this into account, we suggested an algorithm which can model solvent diffusion into matrix while capturing the viscous displacement effective in the fracture.

Algorithms were implemented using the C++ programming language. Petrel and ParaView were used for 2D, 3D and 4D visualization of the results. MATLAB® was used for plotting purposes. The Genetics and Simulated Annealing Algorithms implemented in MATLAB® were used for computer-aided history matching.

Pressure field calculations on the classical simulation grid is part of the algorithms used in Chapters 3 and 4, and this was achieved by ECLIPSE simulator.

Methodology used in Chapter 5 required multiple curve fits and we used functions in MS Excel for that purpose.

Additional details regarding the algorithms used and their implementations are presented in the corresponding chapters.

1.5 Bibliography

Acuna, J.A., Ershaghi, I. and Yortsos, Y.C. 1992. Fractal Analysis of Pressure Transients in the Geysers Geothermal Field. *Proc.* Seventeenth Workshop on Geothermal Reservoir Engineering Stanford University, Stanford, CA, 87-93.

Acuna, J.A. and Yortsos, Y.C. 1991. Numerical Construction and Flow Simulation in Networks of Fractures using Fractal Geometry. Paper SPE 22703 presented at the 66th Annual Technical Conference and Exhibition of the Society of Petroleum Engineers, Dallas, Texas, 6-9 October. DOI: 10.2118/22703-MS.

Al-Khlaifat, A.L. and Arastoopour. H. 2003. Simulation of water and gas flow in fractured porous media. *Fracture and In-Situ Stress Characterization of Hydrocarbon Reservoirs* **209:**201-212.

Araktingi, U.G. and Orr, J. 1990. Viscous Fingering, Gravity Segregation, and Reservoir Heterogeneity in Miscible Displacements in Vertical Cross Sections. Paper SPE 20176 presented at the SPE/CIOE Seventh Symposium on Enhanced Oil Recovery, Tulsa, Oklahoma, 22-25 April. DOI: 10.2118/20176-MS.

Barenblatt, G. I., and Zheltov, Y. P., 1960. Fundamental Equations of Filtration of Homogenous Liquids in Fissured Rocks. *Sov. Dokl. Akad. Nauk.* Engl. Transl., 13: 545-548.

Barton, C.C., Larson, E., 1985. Fractal Geometry of Two-dimensional Fracture Networks at Yucca Mountain, South-Western Nevada. *Proceedings of International Symposium on Fundamentals of Rock Joints* Bjorkliden, Sweden, 77-84.

Beler, R.A. 1990. Pressure Transient Field Data Showing Fractal Reservoir Structure. Paper SPE 21553 presented in the international technical meeting, Calgary, Alberta, Canada, June 10-13. DOI: 10.2118/21553-MS

Berkowitz, B., Cortis, A., Dentz, M. and Scher, H. 2006. Modeling Non-Fickian Transport in Geological Formations as a Continuous Time Random Walk. *Reviews of Geophysics* **44** (RG2003).

Berkowitz, B. and Hadad, A., 1997. Fractal and Multifractal Measures of Natural and Synthetic Fracture Networks. *J. of Geophysical Research* **102**(B6): 205-218.

Bogatkov, D. 2008. Integrated Modeling of Fracture Network System of the Midale Field. MS Thesis, U. of Alberta, Edmonton, Alberta, Canada.

Bogatkov, D. and Babadagli, T. 2009. Characterization of Fracture Network System of the Midale Field, *J. Can. Petr. Tech.* **48**(7): 30-39.

Bogatkov, D. and Babadagli, T. 2010. Fracture Network Modeling Conditioned to Pressure Transient and Tracer Test Dynamic Data, *J. Petr. Sci. and Eng.* **75**: 154-167.

Bunde, A. and Havlin, S., 1995. Brief Introduction to Fractal Geometry. In: Bunde, A., Havlin, S. (Eds), *Fractals in Science*. Chapter 1.

Cacas, M. C., Ledoux, E., De Marsily, G., 1990. Modeling Fracture Flow with a Stochastic Discrete Fracture Network: Calibration and Validation. *Water Resources Research* **26(3)**: 479-500.

Cacas, M.-C., Daniel, J.M. and Letouzey, J. 2001. Nested Geological Modelling of Naturally Fractured Reservoirs. *Petroleum Geoscience* **7**(S):43-52.

Chandrasekhar, S. 1943. Stochastic Problems in Physics and Astronomy. *Rev. of Modern Physics* **15**(1):1-90.

Chang, J. and Yortsos, Y.C. 1990. Pressure-Transient Analysis of Fractal Reservoirs. *SPE Formation Evaluation* **289**:31-38. SPE- 25296-PA. DOI: 10.2118/25296-PA.

Dershowitz, B., LaPointe,P., Eiben, T. and Wei, L. 2000. Integration of Discrete Feature Network Methods with Conventional Simulator Approaches. *SPE Res Eval & Eng* **3**(2): 165-170. SPE-62498-PA. DOI: 10.2118/62498-PA.

Di Donato, G., Lu, H., Tavassoli, Z. and Blunt, M.J. 2007. Multi-Rate Transfer Dual Porosity Modeling of Gravity Drainage Imbibition. *SPE J* **12**(1):77-88. SPE-93144-PA. DOI: 10.2118/93144-PA.

Doe, T. W., Uchida, M., Kindred, J. S. and Dershowitz, W. S. 1990. Simulation of Dual-Porosity Flow in Discrete Fracture Networks. Paper PETSOS-90-120 presented at the Annual Technical Meeting, Calgary, Alberta, June 10 – 13.

Dokholyan, N.V., Buldyrev, S.V., Havlin, S., King, P.R., Lee, Y. and Stanley, H.E. 1999. Distribution of Shortest Paths in Percolation. *Physica A* **266**(1999): 55-61.

ECLIPSE Reservoir Engineering Software. 2009. Schlumberger, http://www.slb.com/services/software/reseng/eclipse2009.aspx

FracaFlow Fracture modeling application. 2008. Beicip. http://www.beicip.com/index.php/eng/software/reservoir/characterization/fracaflow__1

FracMan Discrete Feature Data Analysis software. Golder Associates Inc. http://fracman.golder.com/

Ganzer, L.J. 2002. Simulating Fractured Reservoirs Using Adaptive Dual Continuum. Paper SPE 75233 presented at the SPE/DOE Improved Oil Recovery Symposium, Tulsa, Oklahoma, 13-17 April. DOI: 10.2118/75233-MS.

Gong, B., Karimi-Fard, M. and Durlofsky, L.J. 2008. Upscaling Discrete Fracture Characterizations to Dual-Porosity, Dual-Permeability Models for Efficient Simulation of Flow With Strong Gravitational Effects. *SPE J* **12**(1):58-67. SPE-102491-PA. DOI: 10.2118/102491-PA.

Halvin, S. and Ben-Avraham, D. 1987. Diffusuion in Disordered Media. *Advances in Physics* **36**(6):695-798.

Hoteit, H. and Firoozabadi, A. 2004. Compositional Modeling of Fractured Reservoirs without Transfer Functions by the Discontinuous Galerkin and Mixed Methods. Paper SPE 90277 presented at the SPE Annual Technical Conference, Houston, Texas, September 26-29. DOI: 10.2118/90277-MS.

Jafari, A. and Babadagli, T. 2009. A Sensitivity Analysis for Effective Parameters on 2D Fracture-Network Permeability. *SPE Res Eval & Eng* **12** (3): 455-469. SPE 113618-PA. DOI: 10.2118/113618-PA.

Jafari, A. and Babadagli, T. 2011. Generating 3D Permeability Map of Fracture Networks Using Well, Outcrop, and Pressure-Transient Data. *SPR Reservoir Evaluation & Engineering* **14**(2):215-224. SPE-124077-PA. DOI: 10.2118/124077-PA.

Kazemi, H. and Merrill, L.S. 1979. Numerical Simulation of Water Imbibition in Fractured Cores, *SPE J* **19**(3):175-182. SPE- 6895-PA. DOI: 10.2118/6895-PA.

Karimi-Fard, M., Durlofsky, L.J. and Aziz, K. 2004. An Efficient Discrete-Fracture Model Applicable for General-Purpose Reservoir Simulators. *SPE J.* **9**(2):227-236. SPE 88812-PA. DOI: 10.2118/88812-PA.

King, P.R., Andrade Jr, J.S., Buldyrev, S.V., Dokholyan, N., Lee, Y. , Halvin, S. and Stanley, H.E. 1999. Predicting Oil Recovery Using Percolation. *Physica A* **266**(1999): 107-114.

La Pointe, P. R., 1988. A Method to Characterize Fracture Density and Connectivity through Fractal Geometry. *Int. J. Rock Mech. Min. Sci.* Geomech. Abstr., **25**(6): 421-429.

Long, J.C.S., Gilmour, P. and Witherspoon, P.A. 1985. A Model for Steady Fluid Flow in Random Three-Dimensional Networks of Disc-Shaped Fractures. *Water Resour. Res.* **21**(8): 1105-1115.

Lough, M.F., Lee, S.H., and Kamath, J. 1997. A New Method to Calculate Effective Permeability of Grid Blocks Used in the Simulation of Naturally Fractured Reservoirs. *SPE Res Eng* **12**(3):219-224. SPE-36730-PA. DOI: 10.2118/36730-PA.

Lu,H., Di Donato, G. and Blunt, M.J. 2008. General Transfer Functions for Multiphase Flow in Fractured Reservoirs. *SPE J.* **13**(3):289-297. SPE-102542-PA, DOI: 10.2118/102542-PA.

Mandelbrot, B. B., 1982. The Fractal Geometry of Nature. W. H. Freeman and Co., San Francisco, 460.

MATLAB The Language Of Technical Computing. MathWorks. http://www.mathworks.com/products/matlab/index.html

Montaron, B. 2008. Carbonate Evaluation. *Oil&Gas Middle East* **2008** (8): 26-32.

ParaView Scientific Visualization Software. http://www.paraview.org/

Pearson, K. and Blakeman, J. 1906. *A mathematical theory of random migration*. London, Dulau and co.

Petrel Seismic to Simulation Software. 2010. Schlumberger. http://www.slb.com/services/software/geo/petrel.aspx

Roubinet, D., Liu, H.-H. and de Dreuzy, J.-R. 2010. A New Particle-Tracking Approach to Simulating Transport in Heterogeneous Fractured Porous Media. *Water Resources Research* **46** (W11507).

Saffman, P.G. 1959. A Theory of Dispersion in a Porous Medium. *Fluid Mech.* **6:**321-349.

Sahimi, M. 1993. Flow Phenomena in Rocks: from Continuum Models to Fractals, Percolation, Cellular Automata, and Simulated Annealing. *Reviews of Modern Physics* **65**(1993):1393-1537.

Sahimi, M. and Mehrabi, A.R. 1999. Percolation and Flow in Geological Formations: Upscaling from Microscopic to Megascopic Scales. *Physica A.* **266** (1999): 136-152.

Sarma, P. and Aziz, K. 2004. New Transfer Functions for Simulation of Naturally Fractured Reservoirs with Dual Porosity Models. Paper SPE 90231 presented at the SPE Annual Technical Conference and Exhibition, Houston, Texas, 26-29 September. DOI: 10.2118/90231-MS.

Scheidegger, A.E. 1954. Statistical Hydrodynamics in Porous Media. *Journal of Applied Physics* **25**(8): 994-1001.

Stanley, E.H., Andrade Jr., J.S., Halvin, S., Makse, H. and Suki B. 1999. Percolation Phenomena: a Broad-brush Introduction with Some Recent Applications to Porous Media, Liquid Water, and City Growth. *Physica A.* 266(1999):5-16.

Warren, J.E., and Root, P.J. 1963, The Behavior of Naturally Fractured Reservoirs. *SPE J.* **3**(3): 245-255. SPE 426-PA. DOI: 10.2118/426-PA.

2. Random Walk algorithm applied for 2D lab-scale simulations

2.1 Overview

The objective of this chapter is to introduce an adaptation of a non-classical simulation method (random walk, RW) for simulation of fully miscible displacement in fractured porous media, and to validate this method using production and visual data obtained from an experimental work.

The RW technique deals with particles (walkers), each of which moves randomly, but the probability of the movement is defined considering the physics of the process. By tracing a large number of particles, one can model the process and have an idea about the transport of injected and displaced fluid in complex systems. The RW technique allows capturing micro heterogeneities, the random nature of the diffusion process and viscous fingering. It also requires less computational time compared to classical simulation methods.

The RW model introduced was validated using experimental – visual and production - data for different oil types, displacement directions (horizontal and vertical), and injection rates. Experiments used for validation were performed by Er (2009). The history and image matching processes were presented and critical parameters used in the matching processes were critically evaluated.

2.2 Algorithm description

The fluid flow process during fully miscible injection in fractured porous media is governed by Darcy's Law and the Advection Dispersion Equation (ADE):

$$\vec{v} = -\frac{k}{\mu}(\nabla P + \rho\vec{g}) \tag{2-1}$$

$$\frac{\partial C(x,t)}{\partial t} = -\frac{\partial}{\partial x}\left(v(x,t)C(x,t)\right) + \frac{\partial}{\partial x}\left(D(x,t)\frac{\partial C(x,t)}{\partial x}\right) \tag{2-2}$$

A version of this chapter was presented and also submitted for publication. Stalgorova, E. and Babadagli, T. 2010. "Modeling Miscible Injection in Fractured Porous Media using Non-classical Simulation Approaches". SPE 135903. SPE Russian Oil and Gas Conference and Exhibition, Moscow, Russia.

For the case of constant diffusivity coefficient *D*, the flow described by ADE can be simulated by a large number of particles, moving according to the following rule:

$$\vec{r}(t+\Delta t) = \vec{r}(t) + \vec{v}\Delta t + \bar{z}\sqrt{2D\Delta t} \tag{2-3}$$

where $\vec{r} = (x, y)$ is a particle location, $\vec{v}$ is a mean velocity vector, $\bar{z}$ is a vector with random components, obtained from normal distribution with a mean of 0 and a standard deviation of 1, and *D* is a diffusivity coefficient (the derivation is given by Delay et al. 2005).

In other words, fluid flow is represented by the number of particles (walkers); at each time step, each walker moves, and its movement consists of convective component, defined by the velocity field (which is given by the solution of Darcy's Law), and a random component as a dependant on the diffusivity coefficient. Similar techniques were used before (Araktingi 1988; Araktingi and Orr 1990).

A detailed description of this algorithm is described step by step below:

(1) The model is represented by a 2D grid system. A permeability value is assigned to each grid cell, and the fracture is represented by a row of highly permeable cells. Two selected cells represent injection ('in') and production ('out') ports (**Figure 2-1**).

(2) Initially, a large number of oil walkers are distributed uniformly within the grid. Each walker represents a certain constant volume in such a way that the total volume of all walkers is equal to the pore volume of the model.

(3) The injection is represented by adding *q* solvent walkers on each time step, and the length of time step (Δt) is taken in a way that added volume per unit time the is same as the desired injection rate.

(For example, let us assume the pore volume of the model is *PV*, and initially grid is populated by *N* walkers; then each walker represents

A version of this chapter was presented and also submitted for publication. Stalgorova, E. and Babadagli, T. 2010. "Modeling Miscible Injection in Fractured Porous Media using Non-classical Simulation Approaches". SPE 135903. SPE Russian Oil and Gas Conference and Exhibition, Moscow, Russia.

volume $v_w=PV/N$. To simulate the injection with a constant rate Q, we require $Q=(q\cdot v_w)/\Delta t$; therefore Δt can be calculated as: $\Delta t=(q\cdot v_w)/Q$).

(4) Then at each time step:

a. Injection: q walkers are added in 'in' cell.

b. As flow is incompressible, equation (2-1) can be re-written in a Laplace form: $\nabla\cdot\left(\frac{k}{\mu}\left(\nabla P+\rho\bar{g}\right)\right)=0.$ The finite difference approximation of this equation for a 2D grid system is composed (total flow through each cell is zero, except for 'in' and 'out' cells) as follows:

$$\begin{cases} \frac{k_{[i,j][i,j-1]}A_{[i,j][i,j-1]}}{\mu_{[i,j][i,j-1]}\Delta y_{[i,j][i,j-1]}}(P_{[i,j]}-P_{[i,j-1]}+\rho g\Delta y_{[i,j][i,j-1]})+\frac{k_{[i,j][i,j+1]}A_{[i,j][i,j+1]}}{\mu_{[i,j][i,j+1]}\Delta y_{[i,j][i,j+1]}}(P_{[i,j]}-P_{[i,j+1]}-\rho g\Delta y_{[i,j][i,j+1]})+ \\ +\frac{k_{[i,j][i-1,j]}A_{[i,j][i-1,j]}}{\mu_{[i,j][i-1,j]}\Delta x_{[i,j][i-1,j]}}(P_{[i,j]}-P_{[i-1,j]})+\frac{k_{[i,j][i+1,j]}A_{[i,j][i+1,j]}}{\mu_{[i,j][i+1,j]}\Delta x_{[i,j][i+1,j]}}(P_{[i,j]}-P_{[i+1,j]})=0 \\ Q_{in}=Q \\ P_{out}=0 \end{cases} \quad (2\text{-}4)$$

Δx and Δy – are distances between corresponding cell centers (Figure 2-1), A – is the area of the crossection between cells. $k_{[i,j][i',j']}$ is a permeability between adjacent cells $[i,j]$ and $[i',j']$. It is equal to zero on the boundaries, and otherwise can be calculated as a harmonic mean:

$$k_{[i,j][i',j']}=\frac{2}{1/k_{[i,j]}+1/k_{[i',j']}} \quad (2\text{-}5)$$

$\mu_{[i,j][i',j']}$ is a viscosity of the mixture between the cells $[i,j]$ and $[i',j']$. This value depends on solvent saturation and changes with time. To calculate it for any particular time step, oil and solvent walkers are counted between cells $[i,j]$ and $[i',j']$ (indicated by the yellow area in Figure 2-1). Then solvent concentration is calculated as follows:

A version of this chapter was presented and also submitted for publication. Stalgorova, E. and Babadagli, T. 2010. "Modeling Miscible Injection in Fractured Porous Media using Non-classical Simulation Approaches". SPE 135903. SPE Russian Oil and Gas Conference and Exhibition, Moscow, Russia.

$$C_S = \frac{\text{(number of solvent walkers)}}{\text{(number of oil walkers + number of solvent walkers)}} \qquad (2\text{-}6)$$

and then viscosity is calculated as:

$$\mu_{[i,j][i',j']} = ((1-C_s)\mu_o^{\omega} + C_s\mu_s^{\omega})^{1/\omega} \qquad (2\text{-}7)$$

where μ_o and μ_s are oil and solvent viscosities respectively, and ω – is a mixing parameter. Traditionally a value ω =0.25 is used (Koval 1963; Araktingi 1988); alternatively *w* can be used as a matching parameter.

Note: For the case of horizontal flow, the system (2-4) will not have gravity terms.

c. A composed system of linear equations is solved for pressures in each cell ($P_{[i,j]}$). **Note:** The coefficients of the system include viscosity, which depends on solvent concentration and therefore, changes with time. It means that system (2-4) has to be re-calculated at each time step.

d. Velocities are calculated from pressure values using the Darcy's Law, and then interpolated. Initially the velocities are calculated in cell centers, and then for each walker velocities are interpolated from four nearest cell centers.

e. Each walker moves according to the following rule:

$$x(t+\Delta t) = x(t) + \vec{v_x}\Delta t + z_x\sqrt{2D\Delta t} \qquad (2\text{-}8)$$

$$y(t+\Delta t) = y(t) + \vec{v_y}\Delta t + z_y\sqrt{2D\Delta t} \qquad (2\text{-}9)$$

where $(x(t), y(t))$ is a current walker location**,** v_x and v_y are components of walkers velocity vector (calculated at Step (4)-d), z_x and z_y are random numbers obtained from a normal distribution with a mean of 0 and a standard deviation of 1, Δt is length of the time step, and *D* is a diffusivity coefficient. The diffusivity coefficients may be different for oil and solvent walkers.

A version of this chapter was presented and also submitted for publication. Stalgorova, E. and Babadagli, T. 2010. "Modeling Miscible Injection in Fractured Porous Media using Non-classical Simulation Approaches". SPE 135903. SPE Russian Oil and Gas Conference and Exhibition, Moscow, Russia.

If the walker is close to the boundary, then it gets reflected from the boundary.

f. Production: all walkers from the 'out' cell are removed from the system. The number of produced oil and solvent walkers is recorded at each time step. Knowing that each walker represents a certain volume, it is possible to calculate production data (volume of oil and solvent produced at each time step).

g. Go to next time step.

(5) Repeat this until the process reaches the state when no more oil walkers are produced (recovery curve flattens out).

It is obvious that a higher number of walkers give a more accurate solution but as it will be shown in the next section, a relatively small number (16 walkers per grid block) is sufficient to capture the physics of the process.

C++ code used to implement the algorithm is provided in Appendix A.

2.3 Validation

To validate the model, experiments reported by Er (2009) were used. Experiments were conducted on a transparent 2-D glass bead model of 10×15×0.17cm, with a 0.1 cm width fracture in the middle (**Figure 2-2**). Matrix was represented by tightly packed small (0.3-0.6 mm) glass beads, and fracture was represented by a channel, bounded with bigger (2.0-2.3 mm) glass beads. Initially, the models were saturated with oil, and then coloured pentane was continuously injected from the injection port located at one end of the fracture. The oil-solvent mixture was collected from the production port on the other end of the fracture. During the experiments, displacement patterns were captured periodically and the production data was obtained continuously. The details of experiments are described in detail in the relevant reference, as well as in Er and Babadagli (2010).

A version of this chapter was presented and also submitted for publication. Stalgorova, E. and Babadagli, T. 2010. "Modeling Miscible Injection in Fractured Porous Media using Non-classical Simulation Approaches". SPE 135903. SPE Russian Oil and Gas Conference and Exhibition, Moscow, Russia.

Ten experiments were selected to match the model results. These experiments considered different types of oil as a displaced fluid, different injection rates, and different positions of the model (horizontal and vertical) as listed in **Table 2-1**. The properties of displacing and displaced fluids are given in **Table 2-2.**

Er and Babadagli (2010) reported that a classical continuum simulation approach, using commercial software, was successful to some extent. A critical issue reported as a limitation of the continuum models was that as many as six parameters (solvent and oil diffusivity coefficients, fracture and matrix, longitudinal and transverse dispersivities) were used to obtain a match. This significantly affects the uniqueness of the process. Due to limited experimental work, it was also difficult to correlate the six matching parameters to system characteristics such as oil viscosity, displacement direction, solvent properties, and injection rate. It was also reported by Er and Babadagli (2010) that the simulation runs were not able to capture the shape of displacement patterns for some of the vertical cases, especially for heavy oil. Also note that the matching exercise reported in Er and Babadagli (2010) focused only on matching the displacement images, not paying attention to the production data. In the present study the matching was done not only on the visual but also on the production data.

By using the adapted RW algorithm described above, we simulated the same experiments and obtained reasonable matches for both production data and displacement images. The following parameters were used in the simulations:

(1) The model is represented by a grid of 15×40×1 cells; sizes of the whole model are the same as the size of the glass bead model - 10×15×0.17cm (Figure 2-2).

(2) A permeability value is assigned to each grid cell. Matrix cells have a permeability of 150 D (as reported as a measured value in Er and Babadagli, 2010), and the fracture is represented by a 0.1 cm width row of high permeable cells (15000 D).

A version of this chapter was presented and also submitted for publication. Stalgorova, E. and Babadagli, T. 2010. "Modeling Miscible Injection in Fractured Porous Media using Non-classical Simulation Approaches". SPE 135903. SPE Russian Oil and Gas Conference and Exhibition, Moscow, Russia.

(3) Porosity is 40% for the matrix cells (as reported by the author of the experiment) and 1.0 for the fracture cells. The first and last cells of the high permeable row represent as injection ('in') and production ('out') ports (Figure 2-1).

A sensitivity analysis for grid cells sizes was performed to obtain an optimal grid size and to reduce the computational time. We compared the results of simulations using different sizes of grid cells. A comparison of the model with 15×40×1 cells and the results obtained using a 31×80×1 grid system is given in **Figure 2-3**). The shapes of the displacement patterns look quite similar for both cases, as well as the production curves. The reason of this similarity is can be attributed to the fact that, when we calculate the velocity for each particle, we interpolate the value from four surrounding grid cells instead of just taking the value from the grid cell centre. A grid system of 15×40×1 cells was observed to be optimal and was used in all runs.

In order to define the optimal number of walkers per cell for accurate modeling, we compared results obtained with 16 walkers per cell and 36 walkers per cell (**Figure 2-4**). Results for the displacement patterns looked less noisy for the larger number of walkers; however, the shape of the pattern remained similar. For the production data, the result did not show any significant difference when the number of walkers increased from 16 to 36 walkers per cell. Hence, the value of 16 walkers per grid cell was found to be optimal to obtain an accurate result and minimize the computational time. (Although optimizing number of walkers may not be critical for the size of model considered in this study, it is critical for applications of the model on larger scale.)

As shown above (see example in the Algorithm Description, step 3), there is a relation between the number of walkers in the grid (N), the number of walkers we add every timestep (q) and the duration of the timestep (Δt): $\Delta t=(q \cdot PV)/(Q \cdot N)$. Therefore, if one wants to increase the accuracy of calculation by using smaller

A version of this chapter was presented and also submitted for publication. Stalgorova, E. and Babadagli, T. 2010. "Modeling Miscible Injection in Fractured Porous Media using Non-classical Simulation Approaches". SPE 135903. SPE Russian Oil and Gas Conference and Exhibition, Moscow, Russia.

timesteps, one can either decrease q or increase N. Figures 2-5 and 2-6 illustrate that the optimal number of walkers to be used is N=16×(number of grid cells). In the similar manner, we checked the optimal value of q, and estimated it to be q=4. Further simulations were performed using 16 walkers per grid cell and q=4.

A comparison of displacement patterns with experimental ones was performed for validation purpose. In the matching exercise, the mixing parameter ω was altered, but a final value of 0.25 (as suggested by Koval (1963)) was selected as it yielded the best result. The diffusivity coefficient was used as the main controlling (matching) parameter. Initially, modeling was done with the same diffusivity coefficient used for oil and solvent walkers, but we were not able obtain a good match with experimental data (especially for vertical cases). Therefore, we modified the algorithm to use different diffusivity coefficients for oil and solvent walkers as also commonly used in this type of modeling studies (Er and Babadagli 2010). The oil and solvent diffusivity coefficients (D_o and D_s), which gave a match with experimental results are summarized in **Tables 2-3** and **2-4** for the horizontal and vertical cases, respectively.

2.4 Results and discussion

The results of simulations and their comparison with the experimental data are given in **Figures 2-3** through **2-13.** As seen in all cases, reasonable matches were obtained for both production curves and displacement patterns. Figures 2-11 and 2-12 compare the results of the RW simulation with those obtained from the continuum modeling previously reported by Er (2009). One can observe that the random walk simulation is more accurate in capturing the irregularity of the displacement pattern. Because a random component of fluid flow is honoured in the model, the displacement patterns look more irregular compared to the simulation results obtained from a continuum model (continuum simulation results can be seen in Er and Babadagli (2010)). The matches obtained for the horizontal cases are better than those of the vertical displacement cases.

A version of this chapter was presented and also submitted for publication. Stalgorova, E. and Babadagli, T. 2010. "Modeling Miscible Injection in Fractured Porous Media using Non-classical Simulation Approaches". SPE 135903. SPE Russian Oil and Gas Conference and Exhibition, Moscow, Russia.

In this study, the diffusivity coefficients were used as the matching parameter and it was desired to find correlations for diffusivity coefficients, as they are not practically measurable. One may expect that the diffusivity coefficient for the solvent is higher than that of oil, and also that these coefficients will not change with the injection rate, but will decrease with increasing viscosity of the displaced fluid. Similar correlations for diffusivity coefficients were described in the literature and a number of studies reported a linear dependency between the diffusivity coefficient and viscosity (Sho-Wei Lo et al. 2000; Wen et al. 2005).

Diffusivity coefficients used for modeling of the horizontal cases in this study support these hypotheses (Table 2-3 and **Figure 2-14).** Hence, based on existing experimental data (six experiments for horizontal displacement), the diffusion coefficients can be calculated as a linear function of displaced fluid viscosity:

$$D_o = -0.00000064\mu_{oil} + 0.00402155 \quad (2\text{-}10)$$

$$D_s = -0.00000129\mu_{oil} + 0.00254308 \quad (2\text{-}11)$$

It was not possible to obtain any consistent trend for the vertical cases, and thereby a correlation (Table 2-4). The diffusivity coefficients were changed both with viscosity and rate, and the number of experiments were not enough to observe any trends.

In this study, we used uniform matrix permeability for the pressure and velocity field calculations, i.e, it is a single value for all grids. The matrix permeability field significantly affects the displacement process as it controls the velocity field. However, if there are any heterogeneities in the matrix the proposed model can be used just by altering the permeability for a given grid.

One of the advantages of the RW algorithm over classical modeling is a shorter computational time. In this study, we also compared the computational times for the RW technique and a commercial simulator (classical continuum models). As Er (2009) does not report computational times for his simulations, we performed classical modeling using a commercial simulator. Computational times for the commercial simulator were 5-7 times longer, than those of the RW model. The reason for that is that, in the RW model, only equation for pressure is solved

A version of this chapter was presented and also submitted for publication. Stalgorova, E. and Babadagli, T. 2010. "Modeling Miscible Injection in Fractured Porous Media using Non-classical Simulation Approaches". SPE 135903. SPE Russian Oil and Gas Conference and Exhibition, Moscow, Russia.

directly, while the equations for concentration/composition are not. Instead, the solution of the Advection-Dispersion Equation is obtained by moving oil and solvent walkers, which takes less time than the finite-difference modeling.

2.5 Conclusions

(1) An RW algorithm was adapted to simulate miscible flow in fractured media and validated by comparison with a series of experiments. Simulation results showed good agreement with experimental data, especially for the cases of horizontal displacement. The matches were obtained not only for different direction of flow but also for different oil viscosities and injection rates.

(2) In the algorithm introduced here, there are only two unknown matching parameters (diffusivity coefficients of oil and solvent), comparing with six parameters used in classical modeling (Er 2009). That makes the algorithm easier to use, and this also reduces the uncertainty in the performance prediction and history matching exercises.

(3) A linear dependency of oil and solvent diffusivity coefficients on viscosity was derived for the horizontal cases.

(4) Suggested algorithm allows using non-uniform matrix and fracture permeabilities. Hence, it can be used for the uncertainty analysis in cases when permeability data are insufficient or if the permeability distribution is not uniform.

(5) The RW algorithm introduced was validated for a simple case (a lab scale single matrix-fracture system). This algorithm, which requires less computational time comparing to finite-difference modeling, can be potentially extended to a larger scale, 3D cases with more complex fracture geometry.

A version of this chapter was presented and also submitted for publication. Stalgorova, E. and Babadagli, T. 2010. "Modeling Miscible Injection in Fractured Porous Media using Non-classical Simulation Approaches". SPE 135903. SPE Russian Oil and Gas Conference and Exhibition, Moscow, Russia.

2.6 Tables

Table 2-1. List of cases [experiments taken from Er (2008)] used in the matching process.

Exp. #	Displaced fluid	Viscosity of the displaced fluid(cp)	Injection rate (ml/hr)	Model orientation
1	Kerosene	2.9	15	horizontal
2	Mineral oil	33.5	15	horizontal
3	Mineral oil	33.5	25	horizontal
4	Mineral oil	33.5	45	horizontal
5	Mineral oil	33.5	15	vertical
6	Mineral oil	33.5	45	vertical
7	Mineral oil	500	15	horizontal
8	Mineral oil	500	45	horizontal
9	Mineral oil	500	15	vertical
10	Mineral oil	500	45	vertical

Table 2-2. Properties of the fluids used in the experiments and modeling study.

Fluid	Density (g/cc)	Viscosity (cp)
Pentane	0.63	0.38
Kerosene	0.79	2.9
Mineral oil	0.81	33.5
Mineral oil	0.89	500

Table 2-3. Diffusivity coefficients used for simulation (horizontal flow).

Displaced fluid	Viscosity (cp)	Rate (ml/hr)	D_o (cm^2/s)	D_s (cm^2/s)
Kerosene	2.9	15	0.00254	0.00402
Mineral oil	33.5	15	0.0025	0.004
Mineral oil	33.5	25	0.0025	0.004
Mineral oil	33.5	45	0.0025	0.004
Mineral oil	500	15	0.0019	0.0037
Mineral oil	500	45	0.0019	0.0037

A version of this chapter was presented and also submitted for publication. Stalgorova, E. and Babadagli, T. 2010. "Modeling Miscible Injection in Fractured Porous Media using Non-classical Simulation Approaches". SPE 135903. SPE Russian Oil and Gas Conference and Exhibition, Moscow, Russia.

Table 2-4. Diffusivity coefficients used for simulation (vertical flow).

Displaced fluid	Viscosity (cp)	Rate (ml/hr)	D_o (cm^2/s)	D_s (cm^2/s)
Mineral oil	33.5	15	0.001105	0.004
Mineral oil	33.5	45	0.001105	0.004
Mineral oil	500	15	0.0014	0.0037
Mineral oil	500	15	0.002	0.0037

2.7 Figures

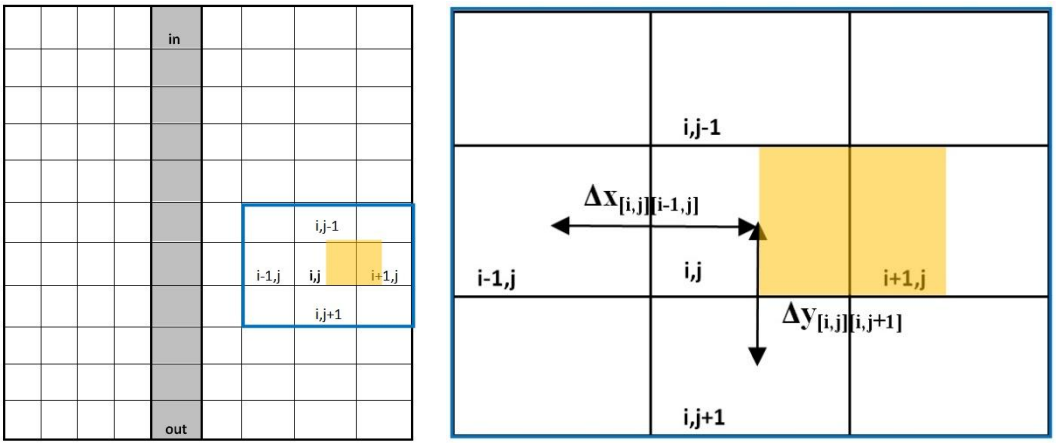


Figure 2-1 Schematic of the Random Walk simulation model.
Oil and solvent walkers in shaded (yellow) area are counted to calculate viscosity between cells [i,j] and [i+1,j].

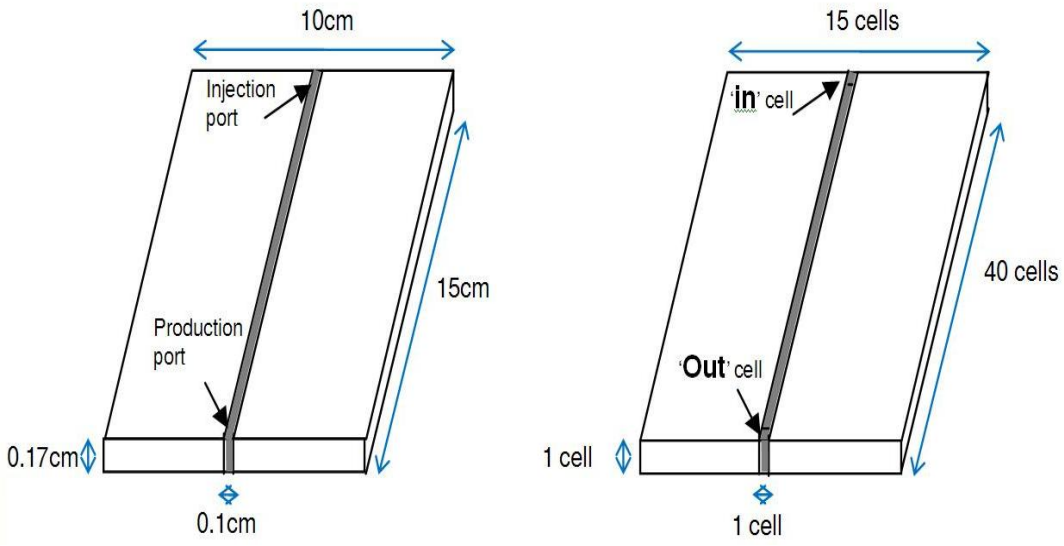


Figure 2-2. Sketches of the experimental (left) and simulation (right) models.

A version of this chapter was presented and also submitted for publication. Stalgorova, E. and Babadagli, T. 2010. "Modeling Miscible Injection in Fractured Porous Media using Non-classical Simulation Approaches". SPE 135903. SPE Russian Oil and Gas Conference and Exhibition, Moscow, Russia.

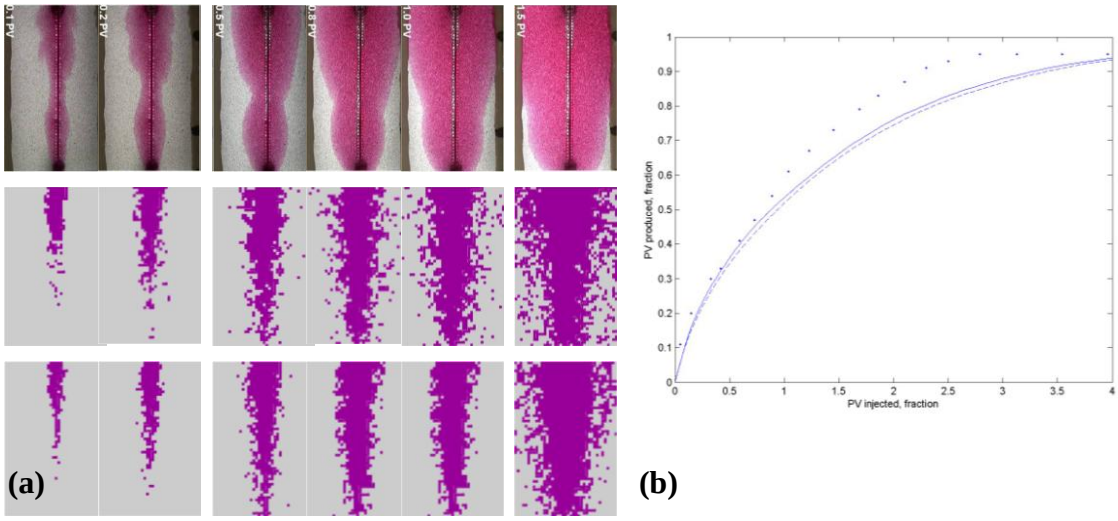


Figure 2-3. Comparison of experimental data given by Er (2009) with results of simulation using 15*40*1 grid and 31*80*1 grid.
Horizontal light oil displacement at 15 ml/h.
(a): upper row – experimental results; middle row – results of simulation using 15*40*1 grid; lower row - results of simulation using 31*80*1 grid.
(b): dots – experimental data; solid line - results of simulation using 15*40*1 grid; dashed line - results of simulation using 31*80*1 grid.

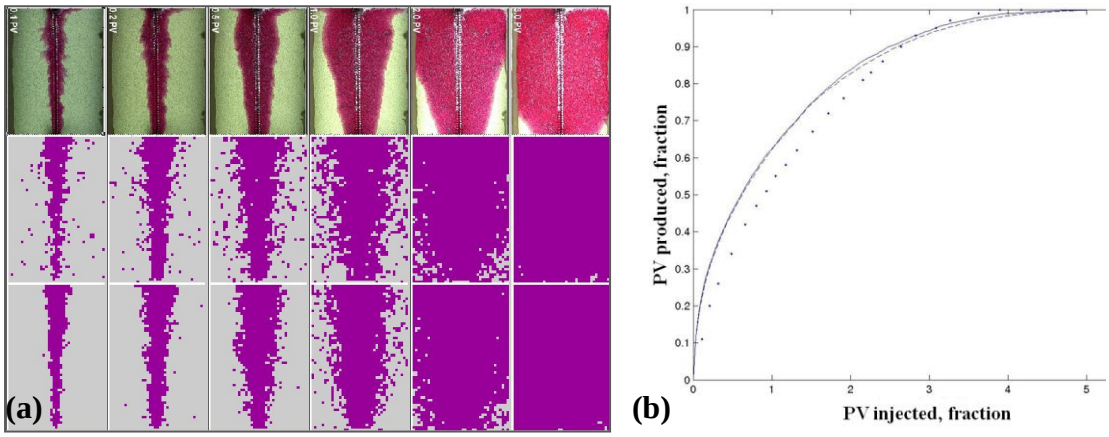


Figure 2-4. Comparison of experimental data given by Er (2009) with results of simulation using 16 and 36 walkers per grid cell.
Horizontal heavy oil displacement at 15 ml/h.
(a): upper row – experimental results; middle row – results of simulation using 16 walkers per grid cell; lower row - results of simulation using 36 walkers per grid cell.
(b): dots – experimental data; solid line - results of simulation using 16 walkers per grid cell; dashed line - results of simulation using 36 walkers per grid cell.

A version of this chapter was presented and also submitted for publication. Stalgorova, E. and Babadagli, T. 2010. "Modeling Miscible Injection in Fractured Porous Media using Non-classical Simulation Approaches". SPE 135903. SPE Russian Oil and Gas Conference and Exhibition, Moscow, Russia.

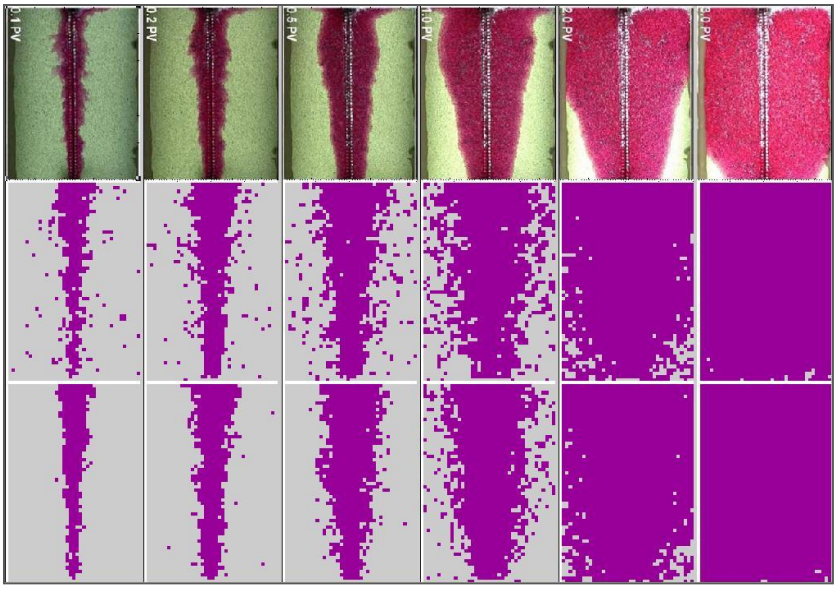


Figure 2-5. Comparison of experimental visual data with results of simulation using 16 and 36 walkers per grid cell.

Experimental images (upper row) are given by Er(2009). Images in the middle row are obtained using 16 walkers per grid cell; images in the lower row are obtained using 36 walkers per grid cell. Horizontal heavy oil displacement at 15 ml/h. Simulation results: darker (purple) areas show a saturation of oil less than 40%.

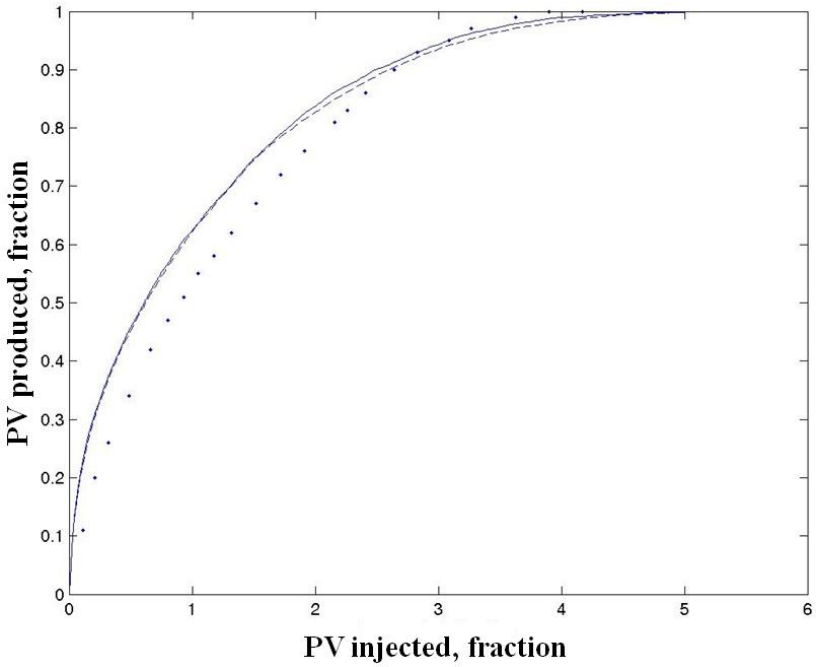


Figure 2-6. Comparison of experimental production data with results of simulation using 16 and 36 walkers per grid cell.

Volume of oil produced vs. volume of solvent injected: dots - experimental data; solid line - results of simulation using 16 walkers per grid cell; dashed line – results of simulation using 36 walkers per grid cell. Horizontal heavy oil displacement at 15 ml/hr.

A version of this chapter was presented and also submitted for publication. Stalgorova, E. and Babadagli, T. 2010. "Modeling Miscible Injection in Fractured Porous Media using Non-classical Simulation Approaches". SPE 135903. SPE Russian Oil and Gas Conference and Exhibition, Moscow, Russia.

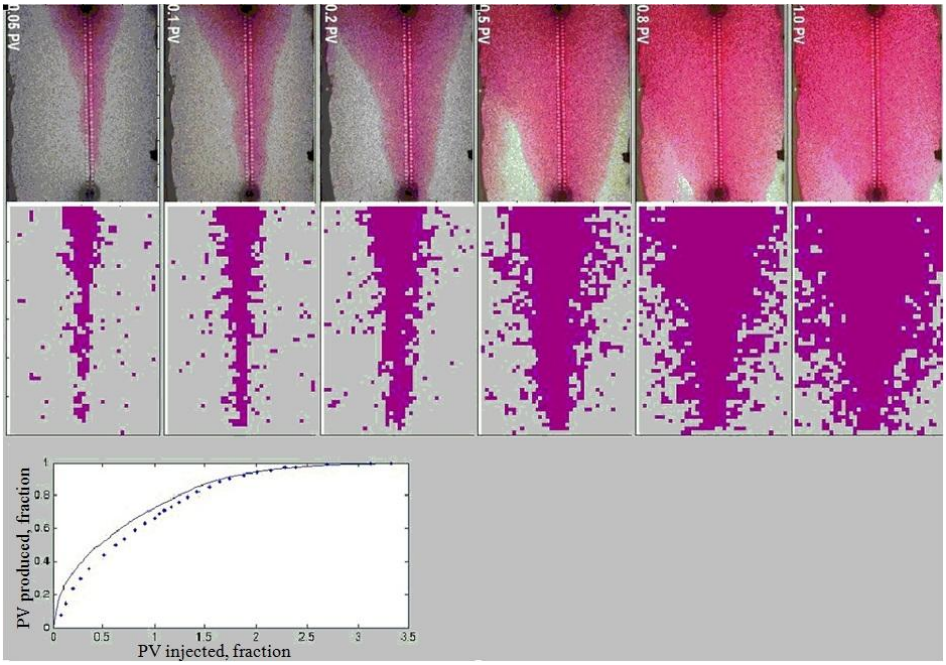


Figure 2-7. Experimental and simulation results for horizontal kerosene displacement at 15 ml/hr.

Upper images: experimental displacement patterns (Er, 2009). Lower images: simulation results, oil saturation less that 40% is shown in purple. Plot: experimental data shown by dots, simulation results - by solid line.

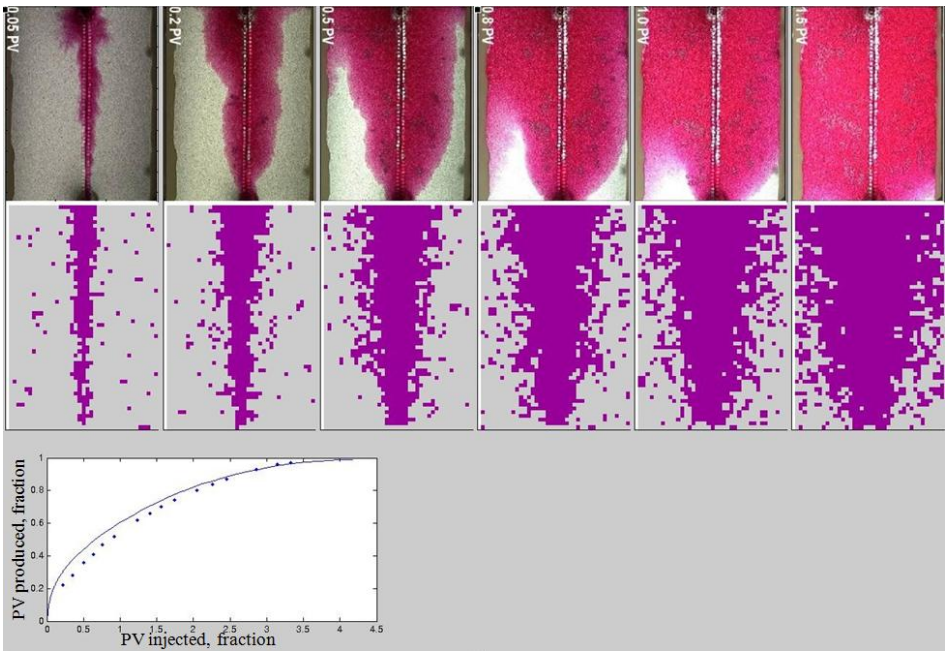


Figure 2-8. Experimental and simulation results for horizontal light oil displacement at 25 ml/hr.

Upper images: experimental displacement patterns (Er, 2009). Lower images: simulation results, oil saturation less that 40% is shown in purple. Plot: experimental data shown by dots, simulation results - by solid line.

A version of this chapter was presented and also submitted for publication. Stalgorova, E. and Babadagli, T. 2010. "Modeling Miscible Injection in Fractured Porous Media using Non-classical Simulation Approaches". SPE 135903. SPE Russian Oil and Gas Conference and Exhibition, Moscow, Russia.

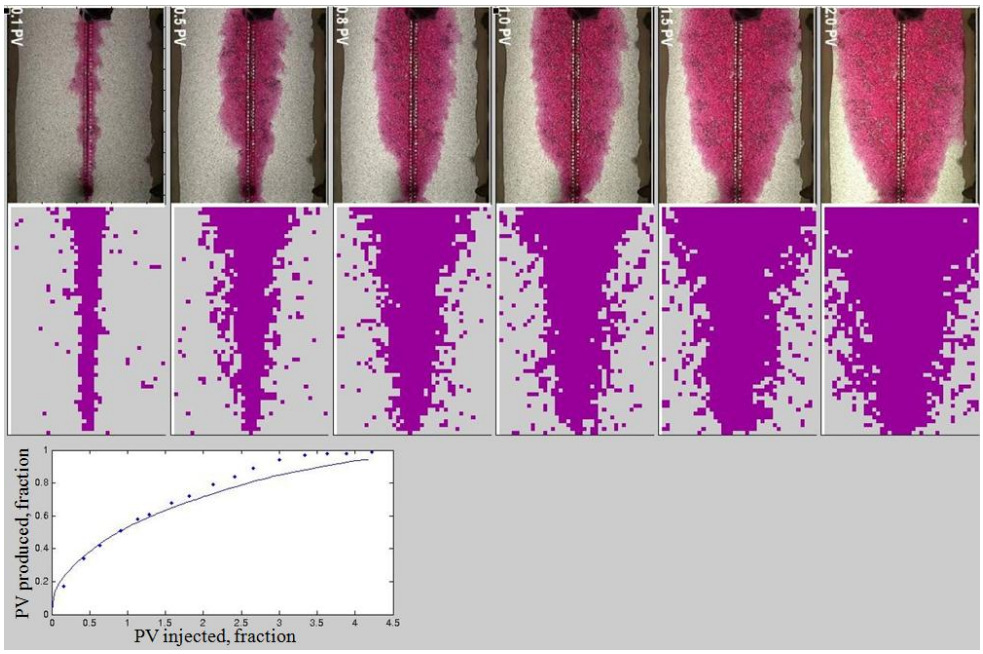


Figure 2-9. Experimental and simulation results for horizontal light oil displacement at 45 ml/hr.

Upper images: experimental displacement patterns (Er, 2009). Lower images: simulation results, oil saturation less that 40% is shown in purple. Plot: experimental data shown by dots, simulation results - by solid line.

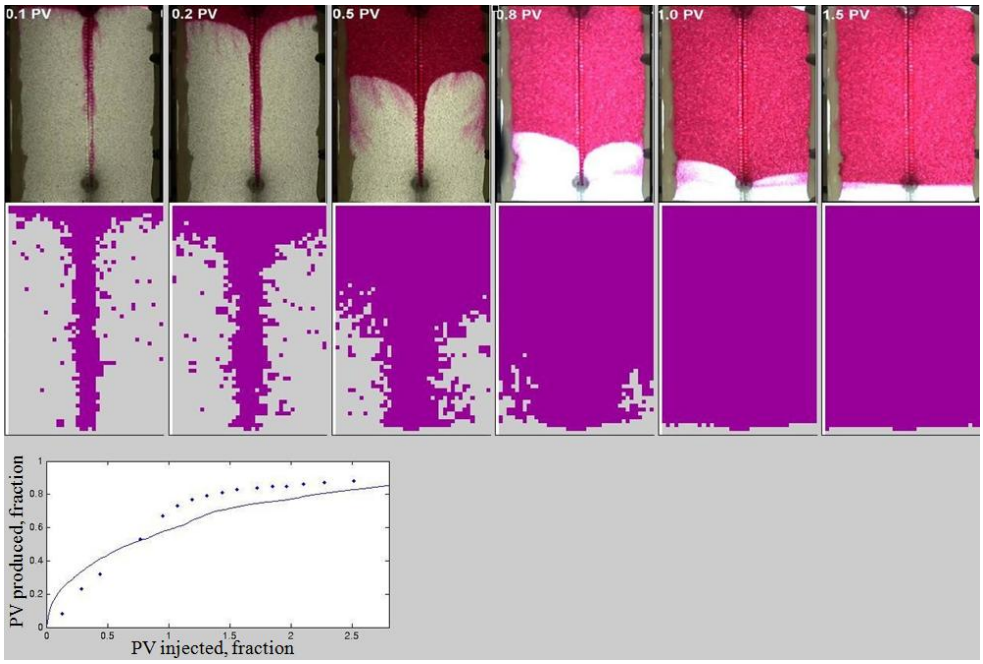


Figure 2-10. Experimental and simulation results for vertical light oil displacement at 15 ml/hr.

Upper images: experimental displacement patterns (Er, 2009). Lower images: simulation results, oil saturation less that 40% is shown in purple. Plot: experimental data shown by dots, simulation results - by solid line.

A version of this chapter was presented and also submitted for publication. Stalgorova, E. and Babadagli, T. 2010. "Modeling Miscible Injection in Fractured Porous Media using Non-classical Simulation Approaches". SPE 135903. SPE Russian Oil and Gas Conference and Exhibition, Moscow, Russia.

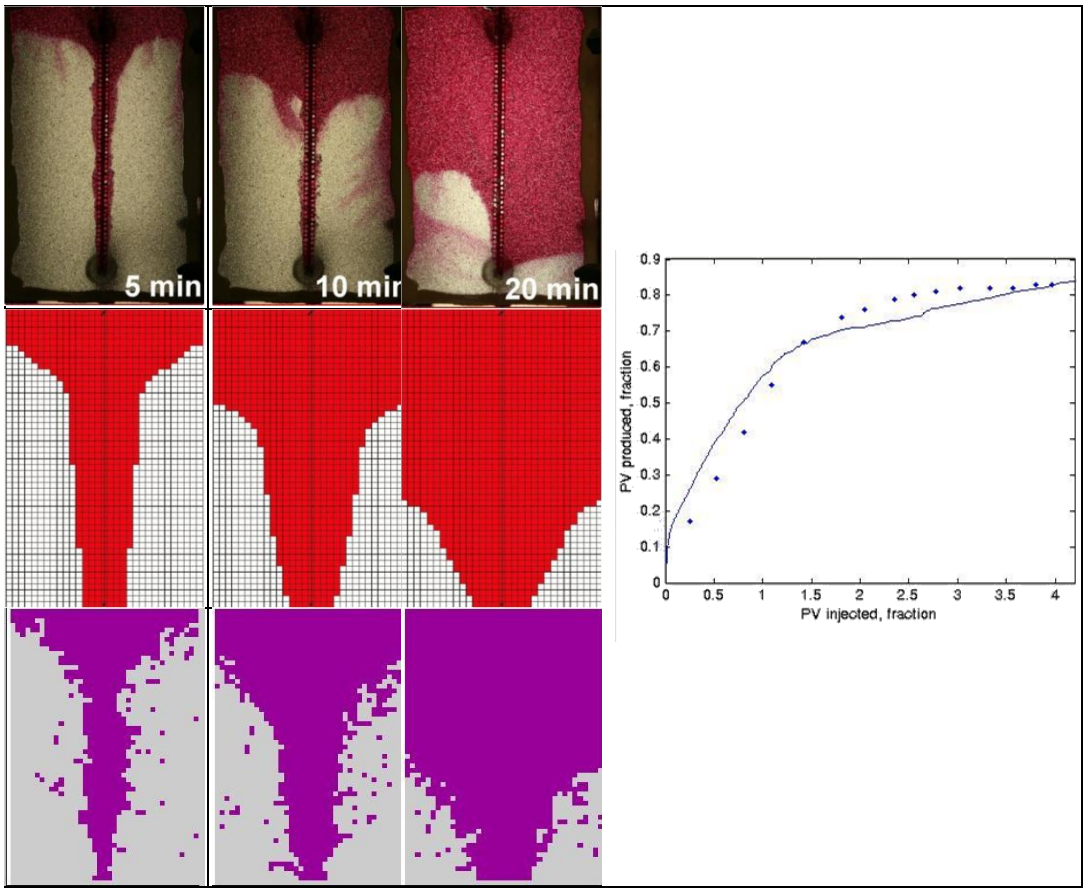


Figure 2-11. Vertical light oil displacement at 45 ml/hr.
Upper images: experimental displacement patterns (Er, 2009). Middle images: results of single continuum modeling (Er, 2009). Lower images: RW simulation results, oil saturation less that 40% is shown in purple. Plot: experimental data shown by dots, RW simulation results - by solid line.

A version of this chapter was presented and also submitted for publication. Stalgorova, E. and Babadagli, T. 2010. "Modeling Miscible Injection in Fractured Porous Media using Non-classical Simulation Approaches". SPE 135903. SPE Russian Oil and Gas Conference and Exhibition, Moscow, Russia.

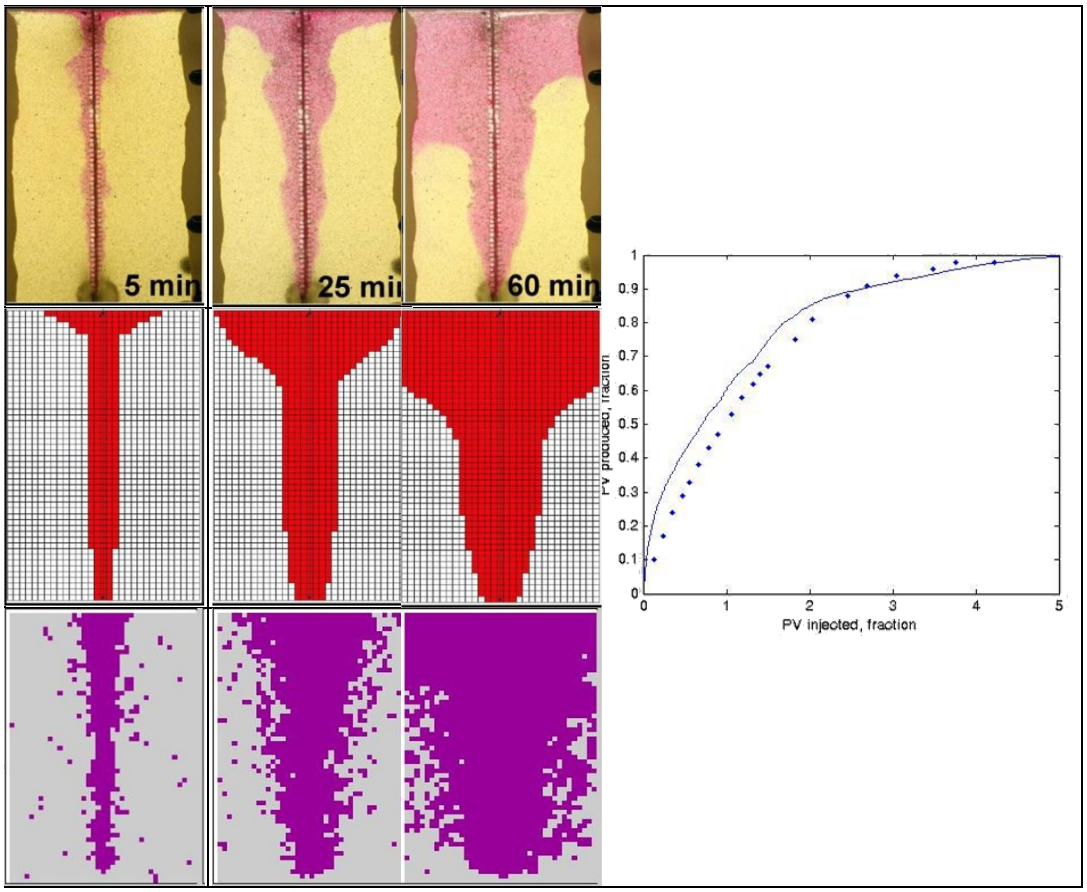


Figure 2-12. Vertical heavy oil displacement at 15 ml/hr.
Upper images: experimental displacement patterns (Er 2009). Middle images: results of single continuum modeling (Er 2009). Lower images: RW simulation results, oil saturation less that 40% is shown in purple. Plot: experimental data shown by dots, RW simulation results - by solid line.

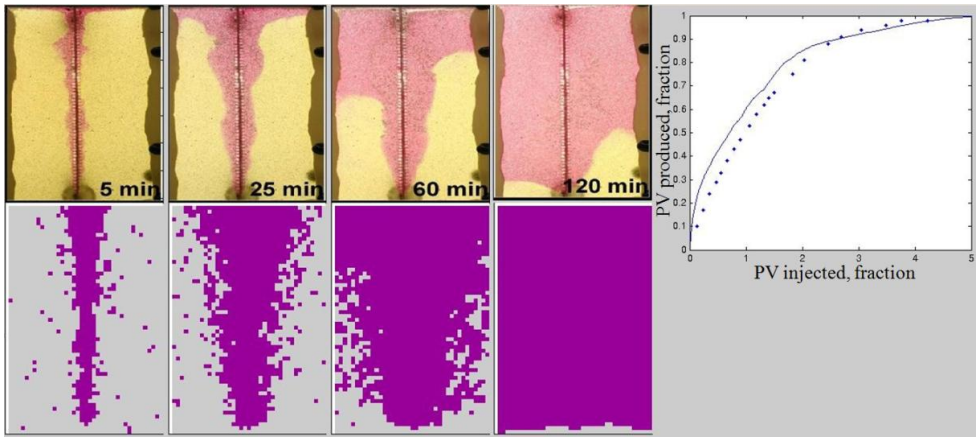


Figure 2-13. Experimental and simulation results for vertical heavy oil displacement at 15 ml/hr.
Upper images: experimental displacement patterns (Er 2009). Lower images: simulation results, oil saturation less that 40% is shown in purple. Plot: experimental data shown by dots, simulation results - by solid line.

A version of this chapter was presented and also submitted for publication. Stalgorova, E. and Babadagli, T. 2010. "Modeling Miscible Injection in Fractured Porous Media using Non-classical Simulation Approaches". SPE 135903. SPE Russian Oil and Gas Conference and Exhibition, Moscow, Russia.

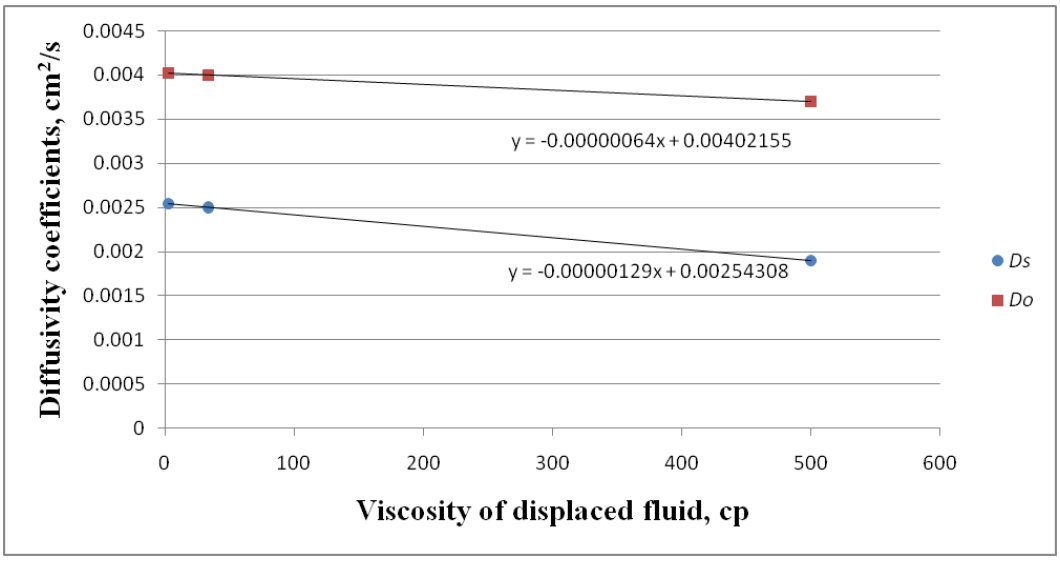


Figure 2-14. Correlation between oil and solvent diffusivity coefficients and viscosity of displaced fluid for horizontal flow.

2.8 Bibliography

Araktingi, U. 1988. Viscous Fingering in Heterogeneous Porous Media. PhD dissertation, Stanford U., Stanford, California.

Araktingi, U.G. and Orr, J. 1990. Viscous Fingering, Gravity Segregation, and Reservoir Heterogeneity in Miscible Displacements in Vertical Cross Sections. Paper SPE 20176 presented at the SPE/CIOE Seventh Symposium on Enhanced Oil Recovery, Tulsa, Oklahoma, 22-25 April. DOI: 10.2118/20176-MS.

Delay, F., Ackerer, P. and Danquigny, C. 2005. Simulating Solute Transport in Porous or Fractured Formations Using Random Walk Particle Tracking: A Review. *Vadose Zone Journal* **4**(2): 360–379.

Er, V. 2009. 2-D Pore and Core Scale Visualization and Modeling of Immiscible and Miscible CO_2 Injection in Fractured Systems. MS Thesis, U. of Alberta, Edmonton, Alberta, Canada.

A version of this chapter was presented and also submitted for publication. Stalgorova, E. and Babadagli, T. 2010. "Modeling Miscible Injection in Fractured Porous Media using Non-classical Simulation Approaches". SPE 135903. SPE Russian Oil and Gas Conference and Exhibition, Moscow, Russia.

Er, V. and Babadagli, T., 2010. Miscible Interaction between Matrix and Fracture – A Visualization and Simulation Study. *SPE Res Eval & Eng* **13**(1): 109-117. SPE-117579-PA. DOI:10.2118/117579-PA.

Koval, E.J. 1963. A Method for Predicting the Performance of Unstable Miscible Displacement in Heterogeneous Media. *SPE J.* **3**(2):145-154. SPE- 450-PA. DOI: 10.2118/450-PA.

Sho-Wei Lo, S.-W., Hirasaki, G.J., House, V.W. and Kobayashi, R. 2000. Correlations of NMR Relaxation Time with Viscosity, Diffusivity, and Gas/Oil Ratio of Methane/Hydrocarbon Mixtures. Paper SPE 63217 presented at the SPE Annual Technical Conference and Exhibition, Dallas, Texas, 1-4 October. DOI: 10.2118/63217-MS.

Wen, Y., Bryan, J., Kantzas, A. 2005. Estimation of Diffusion Coefficients in Bitumen Solvent Mixtures as Derived From Low Field NMR Spectra. *Journal of Canadian Petroleum Technology* **44(**4): 29-35.

A version of this chapter was presented and also submitted for publication. Stalgorova, E. and Babadagli, T. 2010. "Modeling Miscible Injection in Fractured Porous Media using Non-classical Simulation Approaches". SPE 135903. SPE Russian Oil and Gas Conference and Exhibition, Moscow, Russia.

3. Field scale tracer test modeled with Random Walk Particle Tracking

3.1 Overview

Modeling complex transport processes in naturally fractured reservoirs (NFRs) using classical models has certain limitations. Single continuum and dual continuum models require averaging of fracture network properties, therefore information about reservoir connectivity and heterogeneity is not captured properly. In addition, finite-difference calculations for highly heterogeneous models often cause convergence problems.

In this chapter we present modifications to the Random Walk technique for the field-scale applications. For validation, a series of tracer test results from the Midale field in Canada was used. A fracture network model was constructed based on geological data. Then, the Random Walk Particle Tracking (RWPT) model was used to calibrate the fracture network against tracer test results. The results were compared to the ones obtained using continuum (dual-porosity) models and it was observed that the connectivity and breakthrough times can be captured better with the RWPT model.

We performed a sensitivity analysis to identify the importance of different parameters of the simulation results. The new model and observations can be used to validate and calibrate stochastically generated fracture network models and to estimate the EOR performances of NFRs.

3.2 Problem statement and objectives

Construction of fracture network models is usually done using some kind of available data (usually image logs and cores) in practice. The validation of the model through field performance data is, however, not a simple exercise. Often times, the model is fine-tuned by changing different network properties globally or locally until a good match to data like pressure transient tests, tracer tests, or even injection-production data, is obtained. Bogatkov and Babadagli (2009a-b,

A version of this chapter was submitted for publication. Stalgorova, E. and Babadagli, T. 2011. "Field scale modeling of tracer injection in naturally fractured reservoirs using the Random Walk Particle Tracking."

2010) published a series of papers on this subject and showed the use of well test and tracer test data in this exercise. Two problems were critically identified from their studies: (1) the representation of complex (and irregular) fracture network in continuum (single or dual porosity models) models, and (2) the limitations of dual porosity models in capturing the real physics of tracer transport (mainly matrix fracture interaction) if the fracture networks do not show an orthogonal structure. To overcome these problems, one should use a simulation approach, which allows for a more realistic representation of the complex structure of fracture networks.

The purpose of that study is to develop an algorithm that reflects a realistic representation of the fracture network and of the physics of the transport process.

To test the suggested model, we simulated the results of a tracer test performed in the Midale field. The Midale field is a highly heterogeneous naturally fractured reservoir. Tracer test simulation of the Midale field using a single continuum model as well as a dual permeability model is described by Bogatkov (2008) and Bogatkov and Babadagli (2010). They stated that tracer test simulation results are very sensitive to fracture network geometry. Using single continuum or dual continuum models involves averaging fracture network properties. As a result of this simplification, the model does not capture all of the complexity of the fracture network and this hinders a reasonable history match. In this study, we introduced a modified Random Walk Particle Tracking algorithm, in which each fracture is described separately; each fracture has its own geometrical and physical properties (such as length, width, orientation, and permeability), and those properties are used directly, without averaging.

3.3 Algorithm description

The algorithm described below models the flow of fluid through fractured media in a simplified way. To justify these simplifications, we have to make the following assumptions:

(1) We are dealing with one-phase flow (water only).

(2) Fluid flow is dominated by fractures.

(3) Each fracture can be represented by a sub vertical rectangle.

A version of this chapter was submitted for publication. Stalgorova, E. and Babadagli, T. 2011. "Field scale modeling of tracer injection in naturally fractured reservoirs using the Random Walk Particle Tracking."

(4) Production and injection rates are constant.

In the case of the Midale field tracer tests: (1) tracer tests were performed after many years of waterflooding, so most of the oil had washed out of the reservoir, and the remaining oil could be neglected, (2) early tracer breakthrough times indicated fracture-dominated flow (Lavoie 1987), (3) if the fracture has more complicated geometry, we can always represent it by several rectangles joined together, and (4) tracer tests on the Midale field were conducted at nearly constant production and injection rates.

Modeling tracer test with Random Walk Particle Tracking (RWPT) consists of five stages:

(1) Generate the fracture network based on geological information,
(2) Calculate the pressure field in the model,
(3) Convert the fracture network to a graph,
(4) Use this graph to simulate the tracer test,
(5) Compare the simulated and observed results, and edit the fracture network accordingly.

We describe each stage in detail below.

Generate the fracture network based on geological information

Characterization of naturally fractured reservoirs (NFR) starts from the recognition of fractures at different scales and by defining their properties. Available information (such as DST, cores, conventional and image logs, outcrops, well tests and seismic data) should be analyzed to define fracture network properties. In a typical case, it is possible to obtain deterministic information describing several bigger fractures (or major faults), and stochastic parameters, describing the whole fracture network (such as the existence and orientation of a trend, fracture density distribution, fracture lengths distribution, fracture permeability distribution, and fracture widths distribution). A discrete fracture network can be generated based on those parameters, by using any programming language or commercial software. Each fracture is defined as

rectangular and has certain widths and permeability. In order to simulate injection and production of a tracer, one has to add fractures passing through wells.

For this study, we generated a fracture network model based on stochastic parameters given in the papers on the Midale field (Bogatkov and Babadagli 2009a-b, 2010). A more detailed description is given in section 3.5.

Calculate pressure field in the model

Once the fracture network model was described, the pressure at each point was calculated by solving the Darcy's equation numerically. The easiest way to do this is to use a commercial simulator (ECLIPSE was used in this particular study). To achieve this, we needed to create a grid, in which fractures were represented by thin blocks of high permeable cells. The width and permeability of each fracture should be taken from the fracture network generated before.

The next step was to introduce the wells and to run the simulation. Obviously, if the DFN contains many fractures, the simulation model will have a large number of cells which results in enormous computational time. However, because the model had only one phase and we had to simulate only one time step, in practice, the computational times were not remarkably high. The fracture network incorporated in a simulation grid is shown in **Figure 3-1**.

Convert the fracture network to a graph

First, we created a set of vertexes (the end of each fracture and all fracture intersections are represented by vertexes). To create the edges of a graph, we added an edge between those and only those vertexes, which were connected by a fracture. **Figures 3-2** illustrates how the fracture network is represented by a graph.

Next, we have to assign certain properties to the vertexes and edges:

1) For each vertex indicate if it belongs to any well,
2) Pressure value (P) for each vertex: result of the classical simulation at previous stage,

A version of this chapter was submitted for publication. Stalgorova, E. and Babadagli, T. 2011. "Field scale modeling of tracer injection in naturally fractured reservoirs using the Random Walk Particle Tracking."

3) Pressure potential for each vertex: $\Psi = P - \rho g d$ where P is pressure, ρ is density of water, and d is depth,
4) Length for each edge (L): distance between corresponding vertexes,
5) Width and height for each edge (W and h): inherited from the corresponding fracture,
6) Crossection area for each edge: $A=Wh$,
7) Permeability for each edge (k): inherited from the corresponding fracture,
8) Pressure gradient for each edge: if the edge connects vertex x with potential Ψ_x and vertex y with potential Ψ_y and the distance between vertexes is L, then the pressure gradient will be $\nabla\Psi_{xy} = (\Psi_x - \Psi_y)/L$,
9) Velocity for each edge: Calculated from Darcy's Law $v_{xy} = \nabla\Psi_{xy}\frac{k}{\mu}$ where k is permeability of the edge, and μ is the viscosity of the water. **Note:** if Ψ_{xy} is negative, there is no flow from x to y, thus v_{xy} is undefined, but flow from y to x will occur, and v_{yx} can be calculated using the formula above.

Use this graph to simulate tracer test

The flow of the dissolved tracer is represented by the movement of a large number of particles. Each particle represents a certain mass of tracer. Hence, if we know the tracer injection rate and the concentration of the tracer in the injected water, we can calculate how many particles are injected every second.

As the tracer is dissolved in water, its concentration is governed by the Advection Dispersion Equation (ADE):

$$\frac{\partial C(x,t)}{\partial t} = -\frac{\partial}{\partial x}\left(v(x,t)C(x,t)\right) + \frac{\partial}{\partial x}\left(D(x,t)\frac{\partial C(x,t)}{\partial x}\right) \qquad (3\text{-}1)$$

Flow described by the ADE can be simulated by a large number of particles, moving according to the following rule:

$$x(t+\Delta t) = x(t) + v\Delta t + z\sqrt{2D\Delta t} \tag{3-2}$$

where x is a particle location, v is a velocity of the flow, z is a random number obtained from normal distribution with a mean of 0 and a standard deviation of 1, and D is a dispersion coefficient (see Delay et al. 2005, Salamon et al. 2006 for more details).

Note that we are tracking movement of the particle along the edges of the graph, so at each step, we deal with one-dimensional flow. x in the above equation is a 1-D coordinate.

If at time t the particle was at vertex x, and then moved to vertex y, the equation (3-2) becomes:

$$L_{xy} = v_{xy}\Delta t + z\sqrt{2D\Delta t} \tag{3-3}$$

L_{xy} and v_{xy} are known properties of the edge connecting x and y, so we can solve (3-3) to find how much time the particle takes to get from vertex x to vertex y:

$$\Delta t = \left(-\frac{z\sqrt{D}}{v_{xy}\sqrt{2}} + \sqrt{\frac{z^2 D}{2v_{xy}^2} + \frac{L_{xy}}{v_{xy}}} \right)^2 \tag{3-4}$$

To simulate the tracer test we performed the following steps:

1) Assign a certain mass to each particle (m_p). Based on the tracer injection period (t_{ti}), injection rate (q_i), and tracer concentration (c_i), calculate how many particles are needed to represent injection:

   Total mass of injected tracer: m=(volume)(concentration)=(q_i t_{ti})(c_i)

   Number of particles: $N=m/m_p=(q_i\ t_{ti})(c_i)/m_p$

   To each particle j assign a starting time t_{j0} in a way that the starting times of all the N particles are distributed uniformly within the tracer injection period:

   $t_{00}=0$, $t_{10}=(t_{ti}/N)$, $t_{20}=2(t_{ti}/N)$, $t_{30}=3(t_{ti}/N)$,… $t_{N0}=N(t_{ti}/N)$.

2) For each particle, we released it at the injection well, and tracked its way through the reservoir:

   a. Initially particle j had a time t_{j0} and a vertex corresponding to the injection well.

b. In each step, the particle moves from the current vertex to the next one, following one of the edges. The probability of the particle moving along the edge is equal to zero if the pressure potential of the edge is negative, and is proportional to this rate if it's positive: $Prob_{xy} \sim rate = (velocity)(crossection\ area) = v_{xy}A$, where v_{xy} and A are the properties of the edge which were calculated at the previous stage. This is illustrated in **Figure 3-3**.

c. The particle moved to the next vertex, as described in (*b*) so now its time was $t_{j1}=t_{j0}+\Delta t$, where Δt is found from equation (3-4).

d. Continue steps (*b*) and (*c*) until the particle reaches the production well or 'dead end' (vertexes, which do not have any edges with a positive pressure gradient). If the particle reached the production well, record which well and at what time.

3) When all particles are released and tracked, we can convert the recorded data (# of produced particles vs. time) to a concentration profile (concentration vs. time) to analyse the results and compare them with observed data.

Compare simulation and observed results, edit the fracture network accordingly

The results of simulation are very sensitive to fracture network geometry and it is highly likely that initial simulation results will not match the observed data. In the next section, we discuss how different parameters affect the results of simulation and give an idea of how and which parameters need to be modified to obtain the desired match.

C++ code used to implement the algorithm and corresponding Eclipse data file are provided in Appendix B and Appendix C.

A version of this chapter was submitted for publication. Stalgorova, E. and Babadagli, T. 2011. "Field scale modeling of tracer injection in naturally fractured reservoirs using the Random Walk Particle Tracking."

3.4 Sensitivity analysis

To analyze the sensitivity of the algorithm to different parameters, we used the model of the Midale CO_2 pilot area (**Figure 3-4**) and tracer test data reported by Bogatkov and Babadagli (2010). The fracture network of the Midale area is characterized by the existence of two distinct trends: (1) On-trend (main) fractures (SW-NE) and (2) off-trend (secondary) fractures (SE-NW). The on-trend fractures dominate the system; their lengths and density are higher than those of the off-trend fractures. The off-trend fractures are perpendicular to the trend direction. The parameters tested and the responses of the simulations are presented below.

3.4.1 Effect of randomness

As mentioned earlier, the DFN is described by a number of stochastic parameters such as length distribution for on-trend and off-trend fractures, average spacing between fractures (for main and secondary set), as well as distributions of fracture widths and permeabilities. Based on those stochastic parameters, we created a fracture network. Because the process involves randomness, we generated many different fracture network realizations described by the same set of stochastic parameters. This makes classical sensitivity analysis difficult, i.e., to check the sensitivity of one certain parameter by varying it, we have to fix all of the other factors. In other words, if all parameters are fixed and only one varied, one cannot ensure whether the change has occurred because of the variation in that parameter or because of the changes in the nature of the network caused by different random realizations.

To clarify the effect of randomness and check if it diminishes as the number of fractures is increased, we presented the following sensitivity analysis exercise. We ran a set of simulations, in which the set of parameters was fixed, and ten different fracture networks described were generated using this set of parameters. Then, simulations were performed for each network model. Afterwards, we took the same set of parameters and changed (increased) only the number of fractures. With this new set of parameters, we created ten fracture networks realizations and

A version of this chapter was submitted for publication. Stalgorova, E. and Babadagli, T. 2011. "Field scale modeling of tracer injection in naturally fractured reservoirs using the Random Walk Particle Tracking."

ran the model for each of them. The results are presented in **Figures 3-5a** and **3-5b**.

To generate the data seen in these figures, we compared each of ten fracture network realizations with a 'base case' (realization obtained with seed=0) for each number of fractures. To compare the results of a certain realization with the base case, we created cumulative tracer recovery curves for both of them and estimated the difference between the two curves using the following formula:

$$\sum_i \left| Curve_1(x_i) - Curve_2(x_i) \right| \qquad (3\text{-}5)$$

This summation gives values in the y-axis of Figures 3-5a and 3-5b and indicates how far apart those two curves (cumulative tracer production for base case realization and for any other realization) are from each other. In other words, the value in the y-axis indicates the "*difference between cumulative tracer production curves for different stochastic realizations."*

Different points in the same column in Figures 3-5a and 3-5b represent different stochastic realizations (i.e., different fracture networks represented by the same set of stochastic parameters but different random realizations). As expected, the fluctuations for the same number of fracture cases decreases as the number of fractures increases. However, this change is still significant and cannot be ignored even for the highest number of fractures. Therefore, while performing sensitivity analysis, one always has to keep in mind that the effect of randomness cannot be neglected and that the observed results are affected by the randomness to a degree seen in Figures 3-5a and 3-5b.

3.4.2 Experimental design for sensitivity analysis

We tested the relative effect of the following parameters:

1) Average spacing between on-trend fractures,
2) C_h=(spacing between off-trend fractures)/(spacing between on-trend fractures),
3) Fracture permeability for on-trend fractures,
4) Fracture permeability for off-trend fractures,

5) Matrix permeability,
6) Average length for on-trend fractures,
7) Average length for off-trend fractures,
8) Width for on-trend fractures,
9) Width for off-trend fractures,
10) Dispersion coefficient,
11) Trend orientation.

To check the sensitivity to each of these parameters, we fixed all parameters and changed the value for one parameter only. As mentioned above, we cannot neglect randomness involved in the process. Therefore, each parametric analysis (fixing all parameters and varying only one) was performed for five different random realizations. As an example, the effect of on-trend fracture width is illustrated in **Figures 3-6a** and **3-6b.** As seen, the effect of on-trend fracture widths changes from realization to realization. One can also observe that relative changes are more significant for the pair of wells I1-S1, compared to the pair I2-S1. This is expected as the pair I1-S1 is more aligned with the trend (see Figure 3-4), and thereby, the flow between these two wells is more affected by on-trend fracture properties.

To obtain a solid idea about the effect of each parameter on the behavior of the whole network system and to quantify this relationship, we defined the possible range of maximum and minimum values of parameters. In other words, we characterized the effect of each parameter not by a single number but by a range. The range values and the relative effects of each parameter on the behavior of the whole network are summarized in **Table 3-1**.

3.4.3 Sensitivity analysis summary

Results summarized in Table 3-1 are represented graphically as a Pareto chart in **Figures 3-7** and **3-8**. As seen, the relative effect of each parameter is strongly affected by the randomness involved in the process and this effect differs from

A version of this chapter was submitted for publication. Stalgorova, E. and Babadagli, T. 2011. "Field scale modeling of tracer injection in naturally fractured reservoirs using the Random Walk Particle Tracking."

case to case (that is the reason why the light coloured portions of each bar are so wide). Also, these effects are different for different well pairs taken here as an example (compare Figures 3-7 and 3-8). However, some general observations can still be made. The most critical parameter that affects the result of the simulation is the widths of the on-trend fractures, and the second most important parameter is C_h, which was defined earlier as

C_h=(spacing between off-trend fractures)/(spacing between on-trend fractures).

This ratio characterizes the connectivity of the network.

One should emphasize as a final note that the parametric effect obviously depends on how much the value of this parameter has been changed. Here, we compare parameters of a different nature. For example, what is the proper way to compare the effect of change in permeability by 1 Darcy and the effect of change in trend orientation by 1 degree? In this study, we changed each parameter within some physically meaningful range. Defining this range is based on common sense judgement and cannot be formalized. Hence, obtained results are more of a qualitative description rather than quantitative estimate to give an idea of the importance of the parameters to be considered in generating fracture networks and to validate them using dynamic data like tracer tests.

3.5 Application for the Midale field tracer test

To validate the described RWPT algorithm, we modeled a series of tracer tests conducted in the Midale field CO_2 flood pilot area (data obtained from Bogatkov and Babadagli (2010)). This area was chosen because it as an example of a highly fractured reservoir, and for the quality of data and the available information (especially the field tracer data). A detailed description of this data on the Midale field and the Midale CO_2 flood pilot area is available in the literature (Lavoie 1987; Mundry 1989; Fischer 1994; Bunge 2000; Bogatkov 2008; Bogatkov and Babadagli 2010).

The work of Beliveau et al. (1993) contains a detailed analysis of the Midale CO_2 flood pilot area fracture network, which was the focus area in the present

A version of this chapter was submitted for publication. Stalgorova, E. and Babadagli, T. 2011. "Field scale modeling of tracer injection in naturally fractured reservoirs using the Random Walk Particle Tracking."

study. We used the following characteristics of fracture network properties taken from this reference:

1) Most of the fractures (main fracture set) are parallel to each other and are nearly vertical
2) Fractures maintain their orientation and occurrence over large areas
3) The main fracture set is oriented Northeast/Southwest
4) Average fracture spacing is 0.61-0.91m
5) There is a secondary fracture set, oriented perpendicular to the main fracture set. Spacing of the fractures in the secondary set is much greater than the spacing of the main set.

A series of tracer tests was performed in the Midale CO_2 flood pilot area. In this application, different salts were injected through four injecting wells and the productions of those salts from different wells were recorded at four production wells. Well locations and the direction of the main trend are shown in Figure 3-4.

3.5.1 Computer-aided history matching

To simulate the tracer test, we need to generate a fracture network using the following data obtained from several earlier reports (Beliveau et al. 1993; Fischer 1994; Bogatkov 2008; Bogatkov and Babadagli 2010):

- The main (on-trend) fracture set consists of fractures, which lengths vary from 100 to 300 m
- The orientation of the main fracture set is N48^{o}E
- The secondary (off-trend) fracture set is orthogonal to the main set and has 5-10 times less fracture spacing
- Lengths of secondary fractures vary from 30 to 50 m
- Heights of all fractures are distributed as $N(7,1)$ (meters)
- Widths of all fractures are 6 cm
- The fracture permeability is 150 D
- The matrix permeability is 0.15 D

A version of this chapter was submitted for publication. Stalgorova, E. and Babadagli, T. 2011. "Field scale modeling of tracer injection in naturally fractured reservoirs using the Random Walk Particle Tracking."

In order to match the results of simulation with the observed results of tracer tests, parameters of the model had to be tuned. As observed above, the model is particularly sensitive to the change in the properties of the main fracture set (on-set fractures). However, describing the whole fracture set by the same parameters (i.e., assuming that all on-trend fractures have the same width, same permeability, and that spacing does not change all over the area) is not justified from geological point of view and does not give enough flexibility to achieve the history match. On the other hand, if we take properties of each individual fracture (location, permeability, width) and use all of them as history matching parameters, the number of simulation runs becomes enormously high. In this exercise, we selected an area just in the middle of the Midale CO_2 flood pilot area and created eleven rectangular blocks as shown in **Figure 3-9**. Blocks were aligned along the trend. While generating the fracture network, we populated each block with on-trend fractures, using a separate set of parameters to describe the fractures in each block. Off-trend fractures were added to increase connectivity. This way, we gained some flexibility but the number of history matching parameters was still at practically executable level. To decrease the number of parameters even more, we used the results of the sensitivity analysis given above and chose only those parameters, which proved to affect the results of simulation significantly. For example, if we used only on-trend fracture widths for each block and C_h, we would have just twelve parameters to vary.

One of the advantages of the RWPT algorithm is that computational time for each run is relatively small (about 1 minute for the fracture network containing 400 fractures). This makes the algorithm a suitable candidate for computer-aided history matching. Several optimization algorithms can be used to vary parameters automatically. In this study, we used the Genetic and Simulated annealing Algorithms inbuilt in the MATLAB environment. Widths, permeabilities, and spacing for on-trend fractures within each block were used as history matching parameters. Results are shown in **Figures 3-10** through **3-13.**

A version of this chapter was submitted for publication. Stalgorova, E. and Babadagli, T. 2011. "Field scale modeling of tracer injection in naturally fractured reservoirs using the Random Walk Particle Tracking."

3.5.2 History matching results

As seen in Figures 3-10 through 3-13**,** the simulation results matched reasonably well with the observed data for many of the well pairs though some exceptions exist. As we discussed earlier, the results of simulation are extremely sensitive to the fracture network geometry and therefore, the exact match would require a very fine tuning (playing with location and property for every single fracture). Achieving the perfect history match is not the purpose of the study; therefore we confined ourselves to the quick computer-aided history match, which was able to describe connectivity of the reservoir on the qualitative level (i.e. fractures which are connected according to the tracer test results are connected in the model). One may observe that even the results obtained by the quick computer-aided history matching technique described the connectivity of the reservoir (or the fracture network) reasonably well. To illustrate this more clearly, we compared the connectivity of the reservoir based on the real tracer test breakthrough times (data given by Bogatkov and Babadagli 2010) with the connectivity based on the results of RWPT (the present study) and dual porosity (DP) simulations (as given in Bogatkov 2008 and Bogatkov and Babadagli 2010). The results are shown in **Figure 3-14**. The connecting lines between the wells indicate the degree of connectivity, i.e., thicker lines correspond to shorter breakthrough times. The main observation out of this analysis is that the breakthrough times are similar for all directions in the DP model (Figure 3-14c), and it accounts for many other connections indicated by no breakthrough (S3-I3, S1-I4, S2-I1). The RWPT results showed variable breakthrough times in different directions (Figure 3-14b) represented by different thicknesses of the lines as similar to the field observations (Figure 3-14a). This implies that the degree of connectivity in all directions was captured even if the breakthrough times were not estimated precisely.

To further analyze the breakthrough times between each injector-producer pair, the breakthrough times obtained from the field test, DP and RWPT modeling are illustrated in **Figures 3-15** through **3-18.** Only one case of on-trend well pair

A version of this chapter was submitted for publication. Stalgorova, E. and Babadagli, T. 2011. "Field scale modeling of tracer injection in naturally fractured reservoirs using the Random Walk Particle Tracking."

field data showed good agreement with both the DP and RWPT models: I1-S3. In other on-trend cases, the RWPT models showed much better agreement with the field data. The RWPT approach was more successful in estimating the breakthrough times for the off-trend fractures (I1-S1, I2-S1, I3-S1, I4-S1). Out of sixteen well pair comparisons given in Figure 11, only two cases presented better results with the DP model: I2-S2, I4-S3. Both are well pairs in the off-trend direction (NW-SE). These observations verify that the RWPT model is more successful in determining the breakthrough time compared to the DP model in addition to the connectivity of the network.

3.6 Results and discussion

An RWPT algorithm to simulate tracer transport (single phase flow) in highly fractured media was introduced, implemented and tested on a field case. A few observations were critical and need to be highlighted:

1) The algorithm allows for the modeling of each fracture separately, without averaging fracture properties. Hence, we do not lose the details of the fracture network geometry and properties. However, in most cases, information about the location and properties of each and every fracture is not available. Then, we have to use stochastically generated fractures, which bring additional uncertainty.
2) The suggested model has a large number of history matching parameters (properties and location of each fracture). This gives an opportunity to fine-tune the model even though manual history matching may take unreasonable time. On the other hand, the comparatively short computational time of each simulation run makes the model suitable for computer-aided history matching.
3) One of the limitations of the described algorithm is that it requires constant injection and production rates. However, the algorithm can be modified for variable rates. The idea of modification is based on the fact that resolution of production and injection data is normally much lower than tracer concentration data resolution (months vs. hours). In

A version of this chapter was submitted for publication. Stalgorova, E. and Babadagli, T. 2011. "Field scale modeling of tracer injection in naturally fractured reservoirs using the Random Walk Particle Tracking."

the modified model, pressures can be calculated for all existing combinations of production and injection rates (assuming that the number of combinations is not too high), and several versions of graph properties can be created accordingly. Now, while tracking the particle at each particular step, we can check the injection and production rates and use a corresponding set of graph properties for calculations at this step. Example of using RWPT algorithm for the case of non-constant rates is given in Chapter 4.

4) One can observe that RWPT computational times are small for smaller number of fractures, but increases exponentially when the number of fractures is increased (**Table 3-2**, **Figure 3-19**). The Eclipse run takes the major part of the total calculation (Table 3-2). One of the reasons is that the Eclipse calculation does not only solve the equation for the pressure field, but also calculates the fluid flux, which can cause convergence problem. We expect that replacing pressure field calculation in Eclipse by a piece of C++ code may significantly decrease the total computational time. Additional reason to eliminate using Eclipse is its grid size limitations.

3.7 Conclusions

1) The Random Walk Particle Tracking (RWPT) algorithm was applied for the case of flow in naturally fractured reservoirs and tested on a field case through a history match exercise. Then, using the same data, a sensitivity analysis was performed.
2) It was observed that the model is very sensitive to the geometry of the fracture network. Out of 11 different fracture networks and reservoir properties, the widths of on-trend fractures and C_h (the factor that represents fracture network connectivity) were found to be the most influencing ones.
3) The algorithm has a number of limitations such as excessive history matching parameters and the requirement of constant rates. However, overcoming these limitations is a possibility and suggestions were made for this.

A version of this chapter was submitted for publication. Stalgorova, E. and Babadagli, T. 2011. "Field scale modeling of tracer injection in naturally fractured reservoirs using the Random Walk Particle Tracking."

4) A computer-aided history matching technique was used to tune the model. The resulting model represents the connectivity of the reservoir better than that of the DP model, especially for the direction in the off-trend fractures.
5) The approach and algorithm introduced in this paper showed that fracture networks generated can be calibrated through RWPT modeling more reliably than continuum models if tracer data are available.

3.8 Tables

Table 3-1. Relative effect of change in parameters on the results for well pairs I1-S1 and I2-S1.

parameter name	units	minimum value for parameter	maximum value for parameter	minimum change in result (for wells I1-S1)	maximum change in result (for wells I1-S1)	minimum change in result (for wells I2-S1)	maximum change in result (for wells I2-S1)
Spacing between on-trend fractures	m	0.3	0.4	103027	283033	22737	116718
C_h		5	30	173168	596976	74729	215402
Fracture permeability for on-trend fractures	mD	80000	230000	30030	137268	14161	81433
Fracture permeability for off-trend fractures	mD	80000	230000	139735	424273	43853	129002
Matrix permeability	mD	40	150	116442	315276	41886	76479
Average length for on-trend fractures	m	180	230	74218	199933	22304	74306
Average length for off-trend fractures	m	40	90	117234	262717	38796	165157
Width for on-trend fractures	m	0.002	0.05	517737	739326	115573	265505
Width for off-trend fractures	m	0.002	0.05	96107	359702	33679	168681
Dispersion coefficient	m^2/sec	0	0.0001	177406	296512	40888	113750
Trend orientation	rad	-0.02	0.08	53864	196584	68593	177338

Table 3-2. Computational times for different fracture networks.

Number of fractures	Number of grid blocks	Total computational time, seconds	Computational time for pressure calculations, seconds
90	130*173*13=292370	19	15
120	196*209*13=532532	36	30
160	222*295*13=851370	58	48
260	436*398*13=2255864	189	159
300	506*452*13=2973256	263	218
340	558*499*13=3619746	507	451
396	668*592*13=5140928	1193	1118

A version of this chapter was submitted for publication. Stalgorova, E. and Babadagli, T. 2011. "Field scale modeling of tracer injection in naturally fractured reservoirs using the Random Walk Particle Tracking."

3.9 Figures

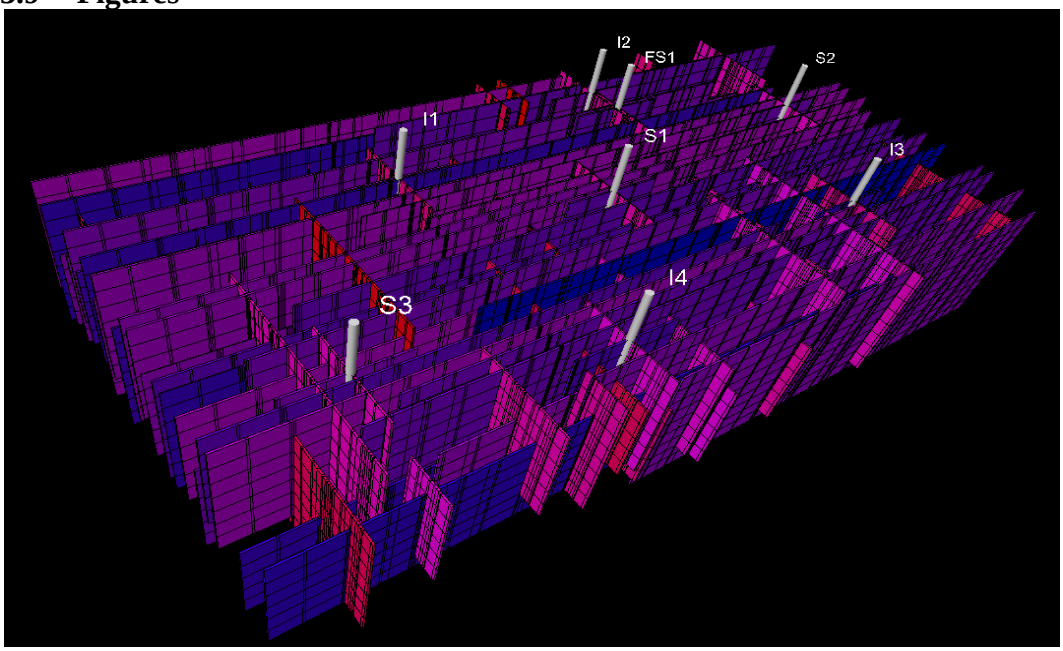


Figure 3-1. Fracture network represented by classical simulation grid.
Only high permeable (fracture) cells are shown. Color shows permeability of each fracture.

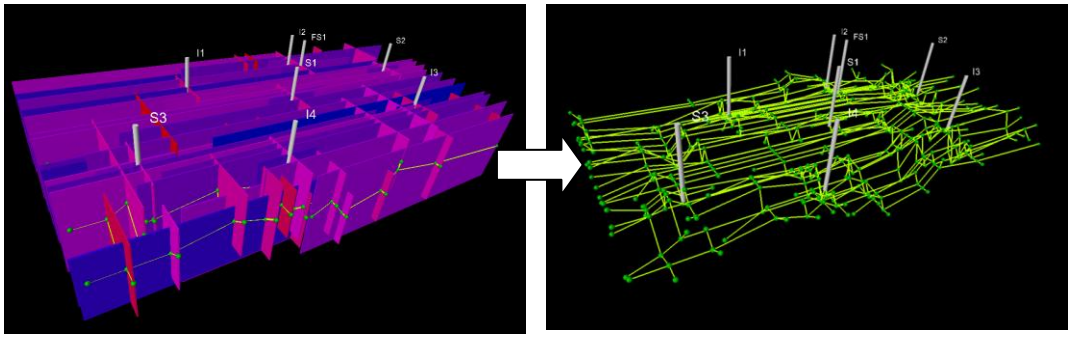


Figure 3-2. Graph created based on fracture network.
Vertexes are added at fracture ends and fracture intersections (green spheres). Edges are added between connected vertexes (yellow lines). Only this graph is used for further simulation.

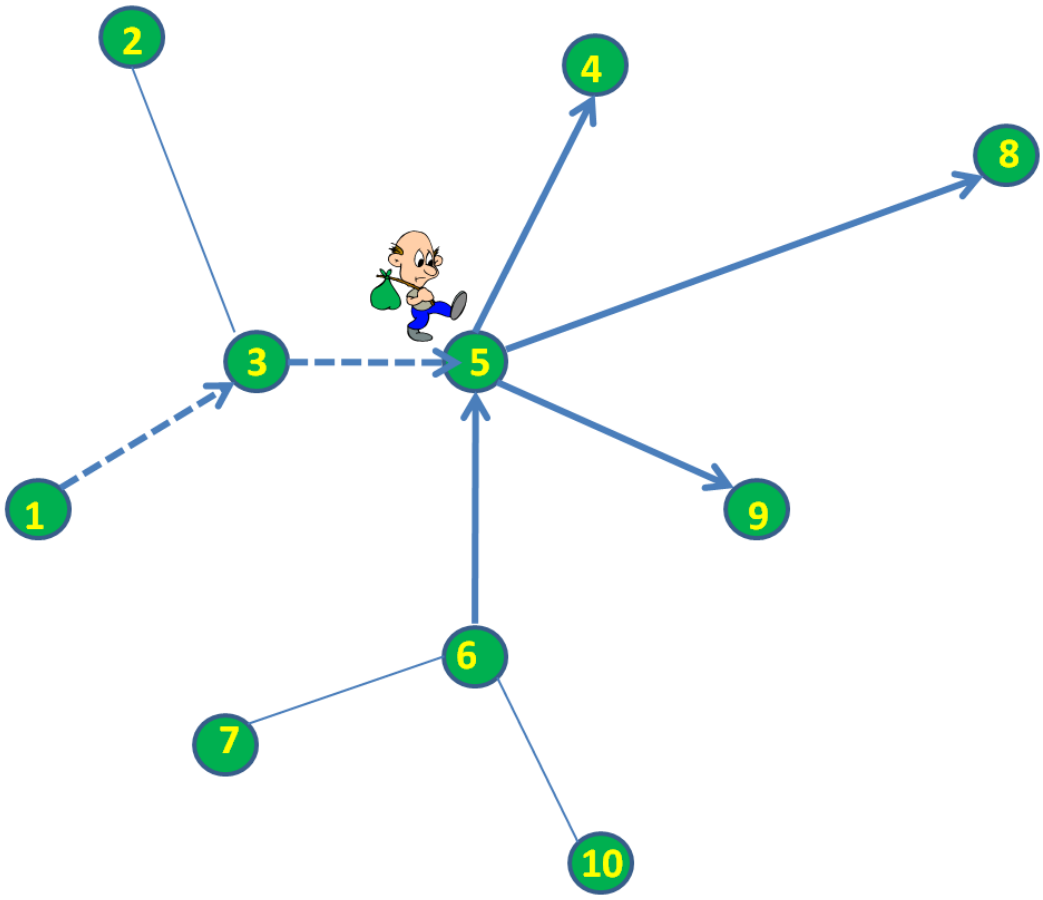


Figure 3-3. Movement of the Walker through the graph.
Walker (particle) started at vertex 1 (injection well), then went to vertex 3 and then to vertex 5. Now he 'decides' which way to go. He will not go to vertex 3 or 6, because the pressure gradient is negative (indicated by arrow direction), and his probability to go to vertexes 4, 8 or 9 is proportional to the flow rate long those edges.

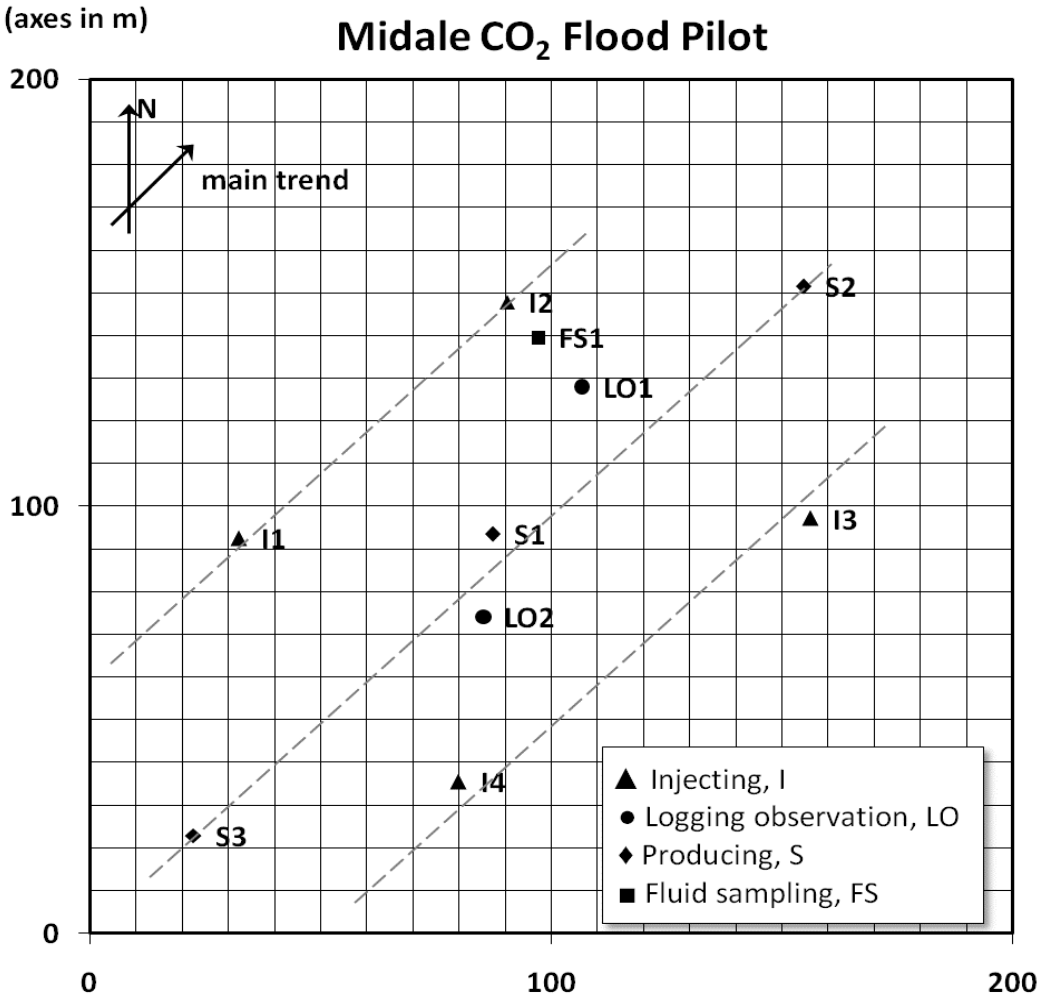


Figure 3-4. Midale CO_2 Flood Pilot configuration (After Bogatkov, 2008).

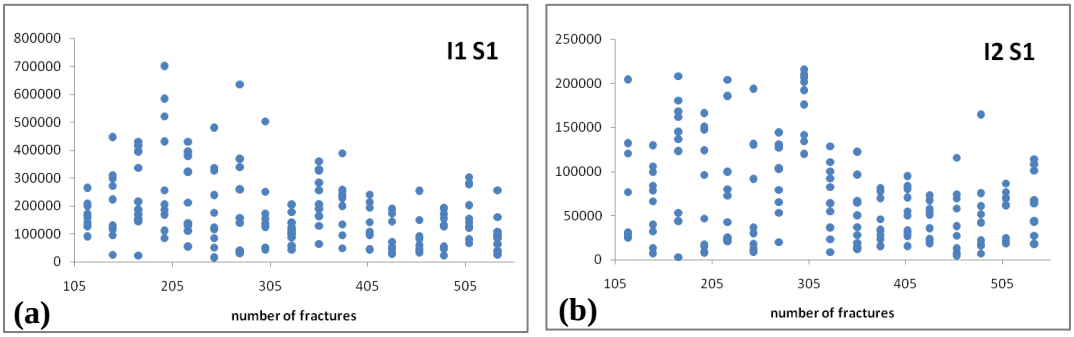


Figure 3-5. Simulation result vs. number of fractures.
The value in the y-axis indicates the difference between cumulative tracer production curves for different stochastic realizations. Figure (a) shows result for injector I1 and producer S1; Figure (b) shows result for injector I2 and producer S1.

A version of this chapter was submitted for publication. Stalgorova, E. and Babadagli, T. 2011. "Field scale modeling of tracer injection in naturally fractured reservoirs using the Random Walk Particle Tracking."

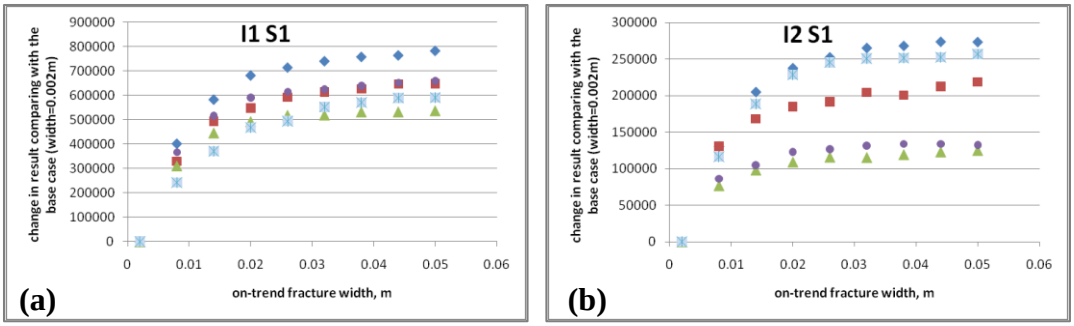


Figure 3-6. Effect of on-trend fracture width.
Each color represents changing fracture width for one particular fracture network. All fracture networks are described by same set of stochastic parameters. Figure (a) shows result for injector I1 and producer S1; Figure (b) shows result for injector I2 and producer S1.

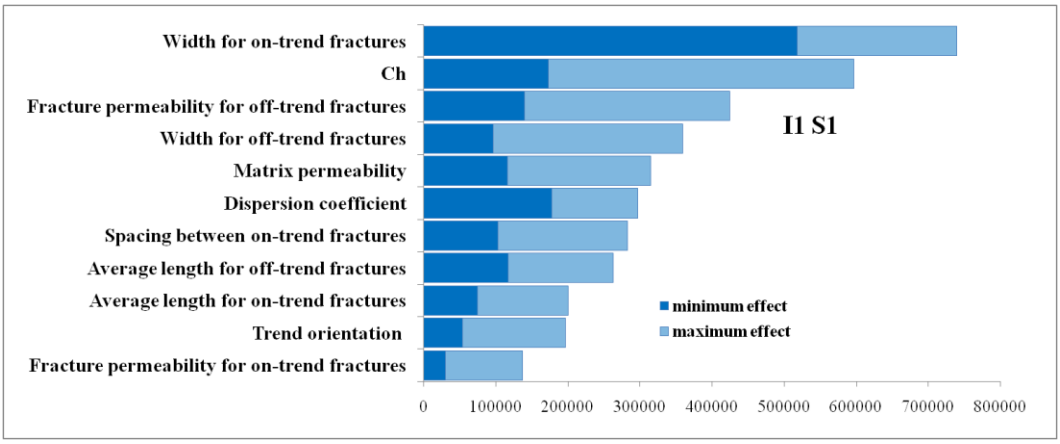


Figure 3-7. Effect of parameters for injector I1 and producer S1.

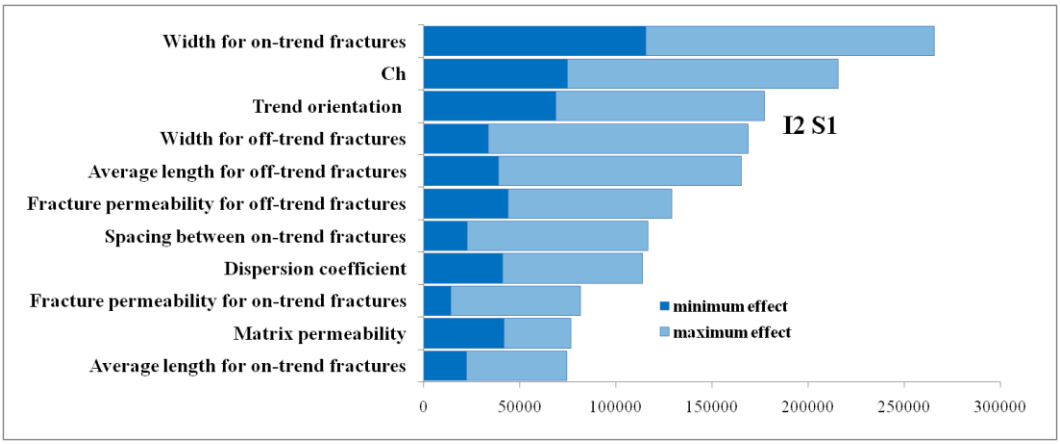


Figure 3-8. Effect of parameters for injector I2 and producer S1.

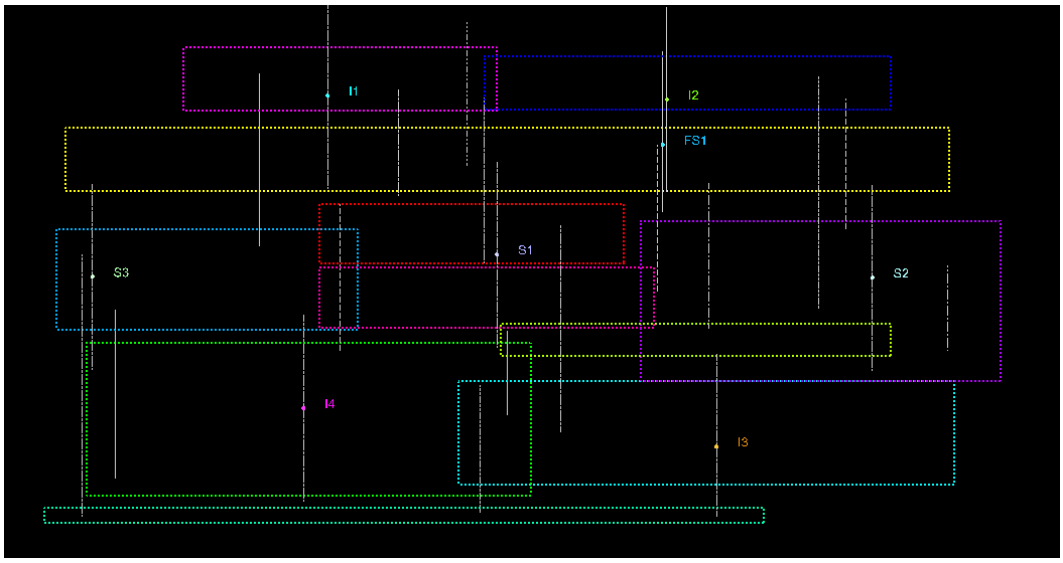


Figure 3-9. Midale CO_2 pilot area divided into blocks for modeling.
Rectangular blocks are to be populated by on-trend fractures. Off-trend fractures are shown by grey vertical lines.

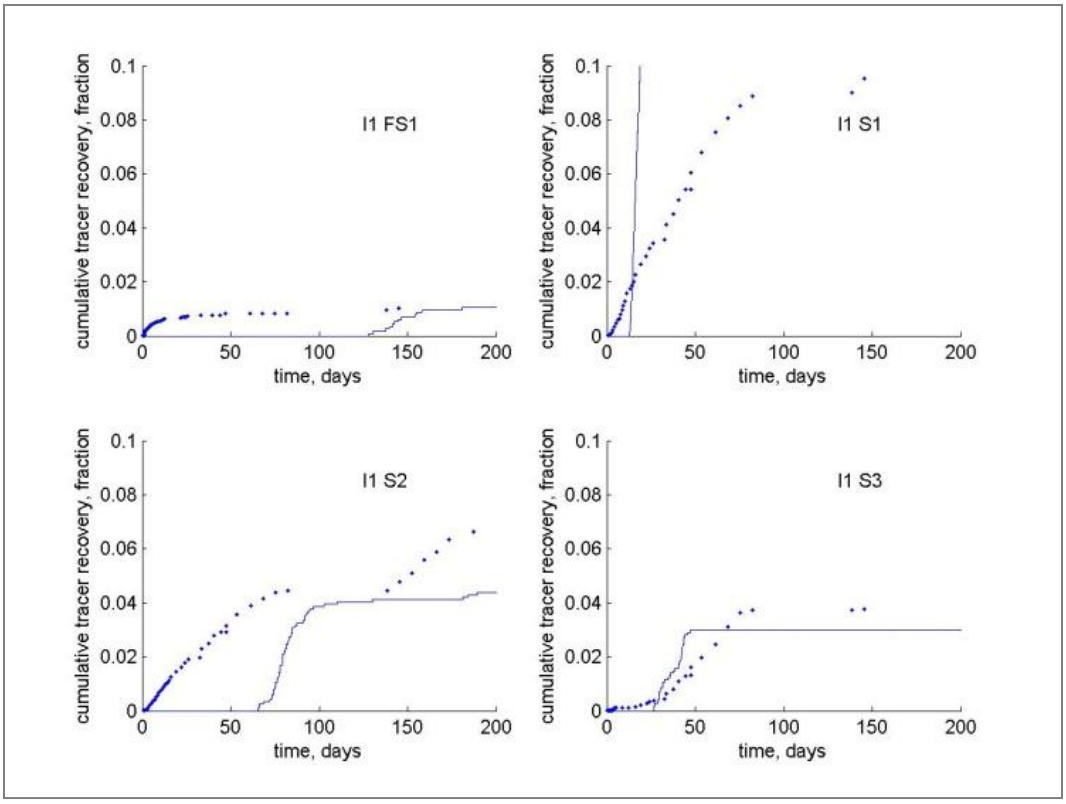


Figure 3-10. History match of tracer tests for injector I1.
Dots: observed tracer recovery; line: simulated tracer recovery.

A version of this chapter was submitted for publication. Stalgorova, E. and Babadagli, T. 2011. "Field scale modeling of tracer injection in naturally fractured reservoirs using the Random Walk Particle Tracking."

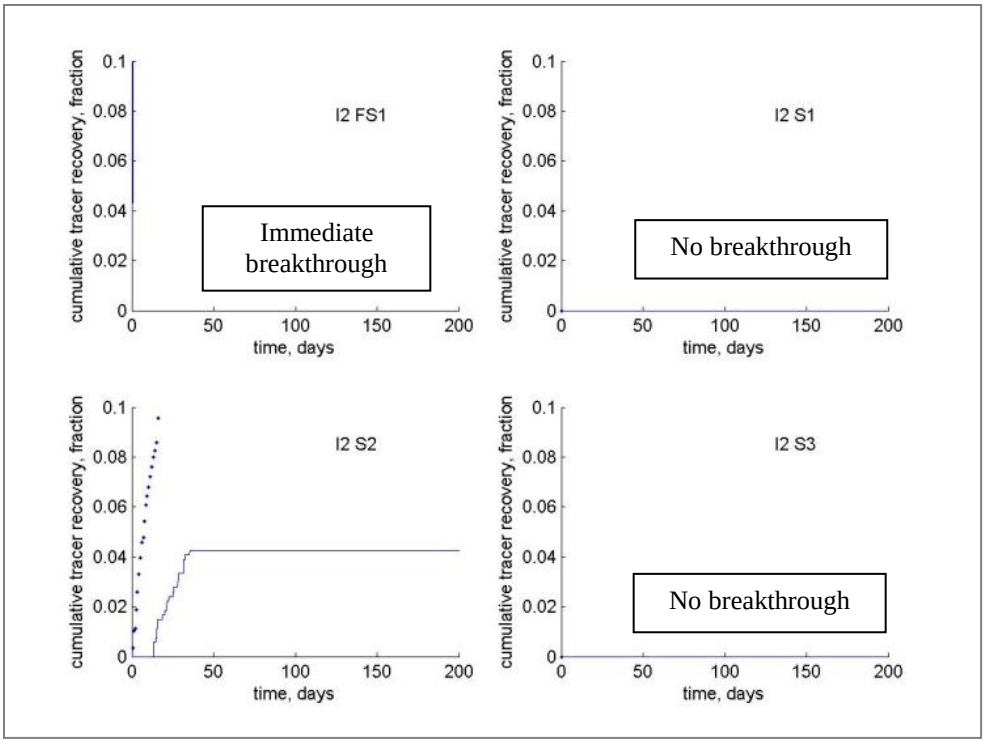


Figure 3-11. History match of tracer tests for injector I2.
Dots: observed tracer recovery; line: simulated tracer recovery.

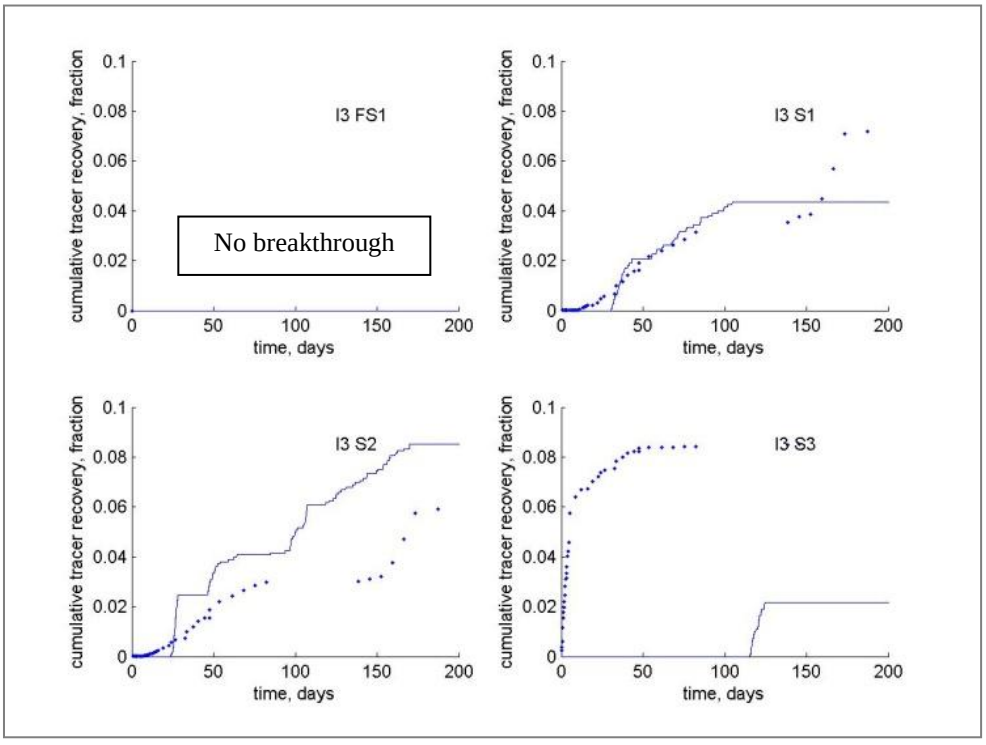


Figure 3-12. History match of tracer tests for injector I3.
Dots: observed tracer recovery; line: simulated tracer recovery.

A version of this chapter was submitted for publication. Stalgorova, E. and Babadagli, T. 2011. "Field scale modeling of tracer injection in naturally fractured reservoirs using the Random Walk Particle Tracking."

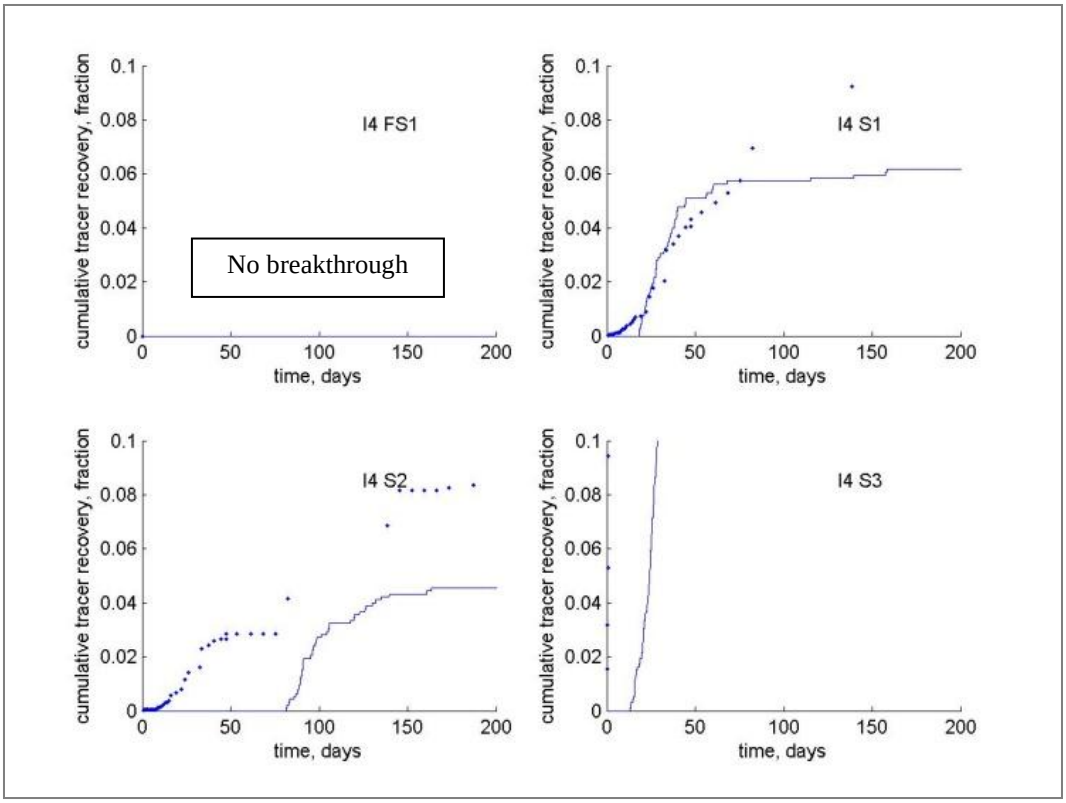


Figure 3-13. History match of tracer tests for injector I4.
Dots: observed tracer recovery; line: simulated tracer recovery.

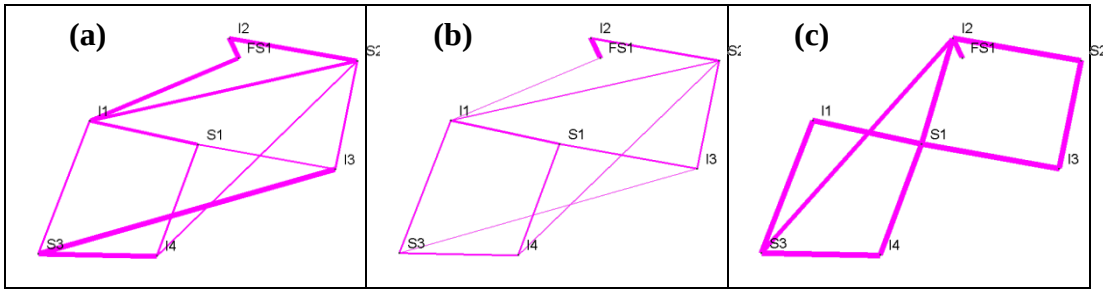


Figure 3-14. Well connectivity analysis based on breakthrough time.
(a) Observed data (Bogatkov, 2008), (b) RWPT simulation (middle),
(c) DP model(Bogatkov, 2008). I: Injector, S: Producer, FS: Observation well.

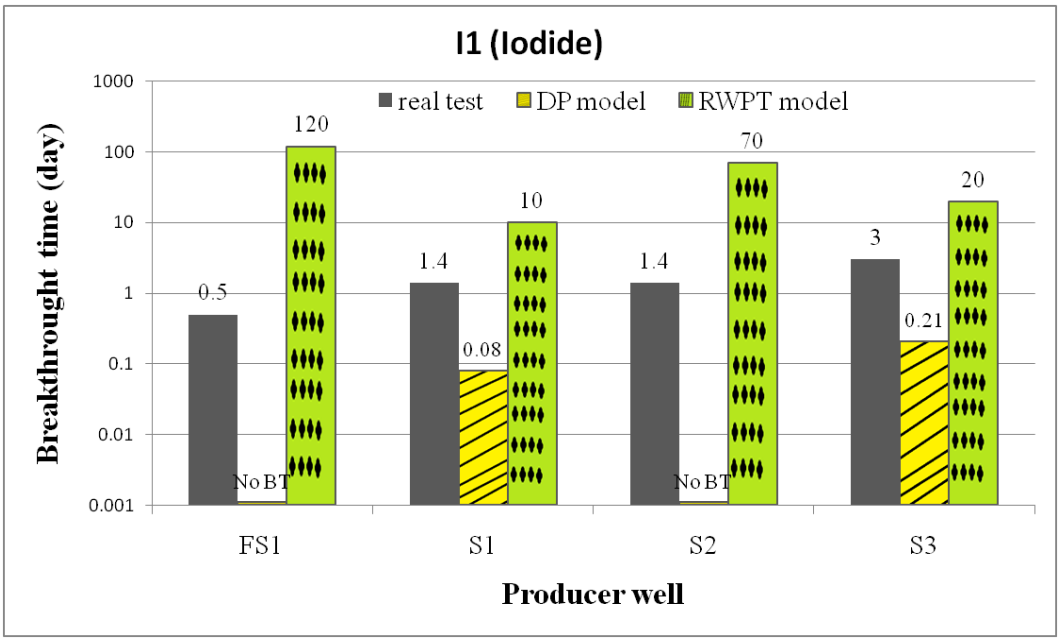


Figure 3-15. Comparison of breakthrough times obtained through field test and two different modeling approaches for injector I1.
Field data and results of DP modeling are is given by Bogatkov (2008).

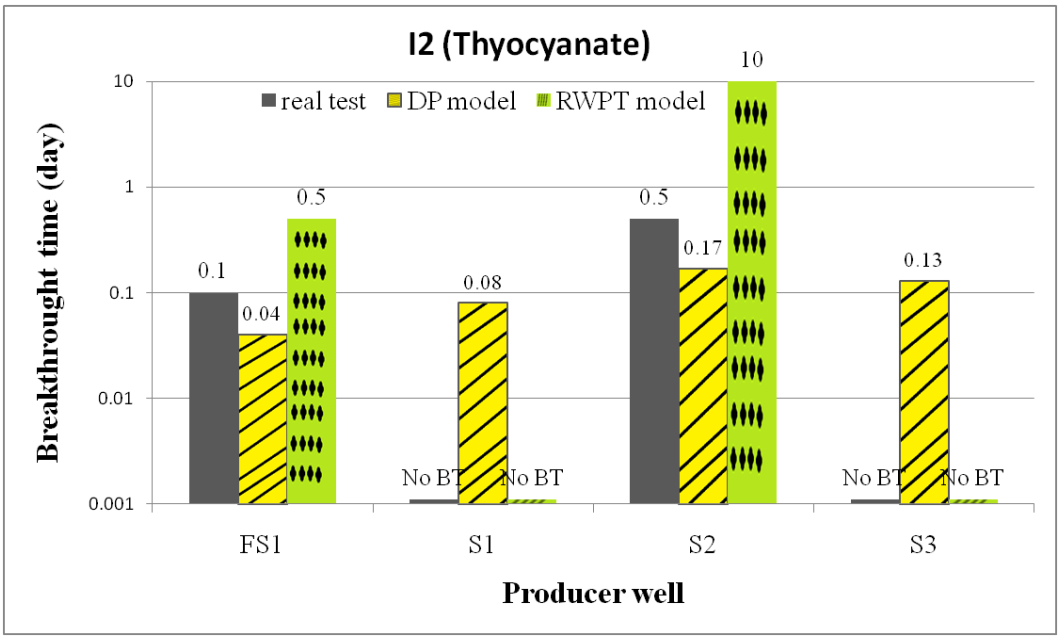


Figure 3-16. Comparison of breakthrough times obtained through field test and two different modeling approaches for injector I2.
Field data and results of DP modeling are is given by Bogatkov (2008).

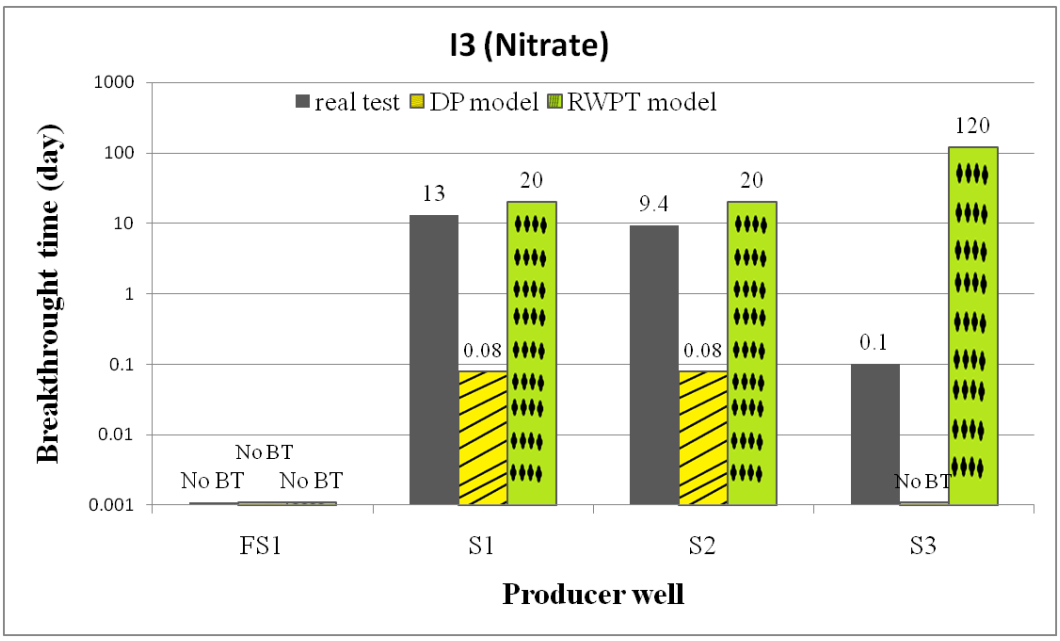


Figure 3-17. Comparison of breakthrough times obtained through field test and two different modeling approaches for injector I3.
Field data and results of DP modeling are is given by Bogatkov (2008).

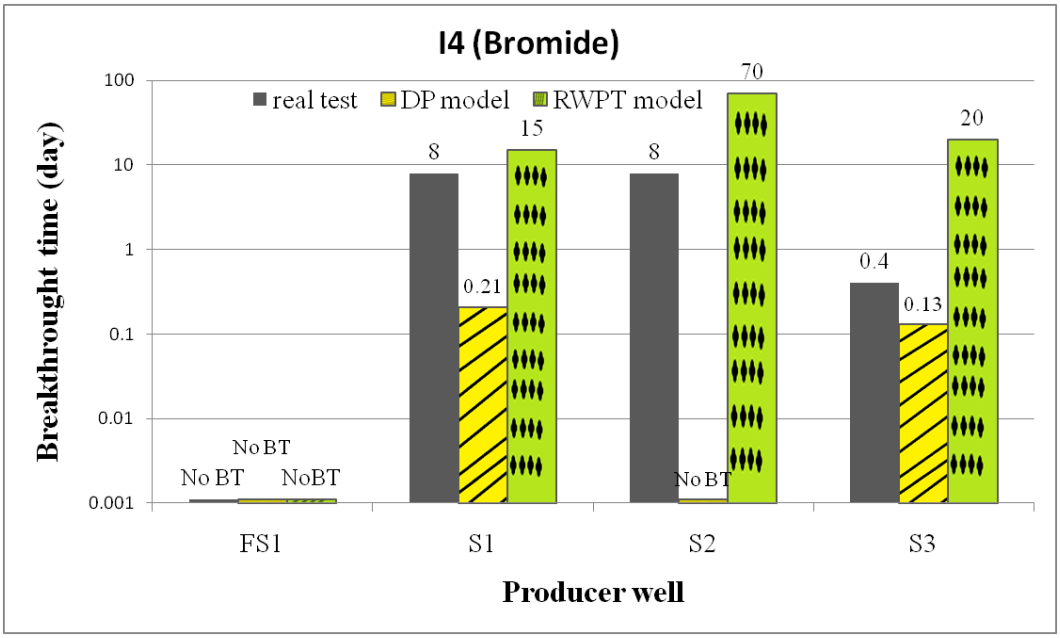


Figure 3-18. Comparison of breakthrough times obtained through field test and two different modeling approaches for injector I4.
Field data and results of DP modeling are is given by Bogatkov (2008).

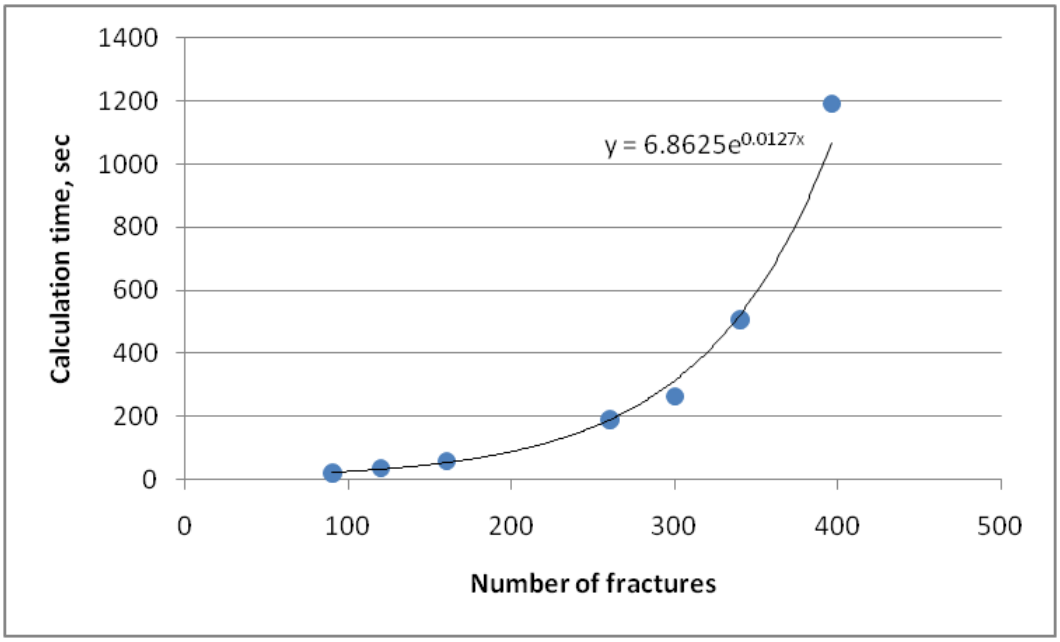


Figure 3-19. Calculation time for different fracture networks.

3.10 Bibliography

Beliveau, D., Payne, D.A. and Mundry, M. 1993. Waterflood and CO2 Flood of the Fractured Midale Field, *JPT* **45** (9): 881–817.

Bogatkov, D. 2008. Integrated Modeling of Fracture Network System of the Midale Field. MS Thesis, U. of Alberta, Edmonton, Alberta, Canada.

Bogatkov, D. and Babadagli, T. 2009a. Characterization of Fracture Network System of the Midale Field, *J. Can. Petr. Tech.* **48**(7): 30-39.

Bogatkov, D. and Babadagli, T. 2009b. Integrated Modeling of the Fractured Carbonate Midale Field and Sensitivity Analysis through Experimental Design, *SPE Res. Eval. and Eng*. **12**(6): 951-962.

Bogatkov, D. and Babadagli, T. 2010. Fracture Network Modeling Conditioned to Pressure Transient and Tracer Test Dynamic Data, *J. Petr. Sci. and Eng.* **75**(1-2): 154-167.

Bunge, R.J. 2000. Midale Reservoir Structure Characterization Using Integrated Well and Seismic Data, Weyburn Field, Saskatchewan. MS Thesis, Colorado School of Mines, Golden, Colorado.

Delay, F., Ackerer, P. and Danquigny, C. 2005. Simulating Solute Transport in Porous or Fractured Formations Using Random Walk Particle Tracking: a Review. *Vadose Zone Journal* **4**(2):360–379.

Fischer, B.F. 1994. Fracture Analysis: Midale Field, South-eastern Saskatchewan. Calgary Research Centre, *Shell Canada Limited*.

A version of this chapter was submitted for publication. Stalgorova, E. and Babadagli, T. 2011. "Field scale modeling of tracer injection in naturally fractured reservoirs using the Random Walk Particle Tracking."

Lavoie, R.G.J. 1987. Evaluation of Inter-Well Tracer Tests Relating to CO2 Miscible Flooding in Midale. Calgary Research Centre, *Shell Canada Limited*.

Mundry, M. U. 1989. A Petrophysical and Geological Engineering Study, Midale Unit, Saskatchewan. Implications for Unit Development. Vol. 1, CAOR.89.055. Calgary Research Centre, *Shell Canada Limited*.

Salamon, P., Fernàndez-Garcia, D. and Jaime Gómez-Hernández, J.J. 2006. A Review and Numerical Assessment of the Random Walk Particle Tracking Method. *Journal of Contaminant Hydrology* **87**: 277–305.

A version of this chapter was submitted for publication. Stalgorova, E. and Babadagli, T. 2011. "Field scale modeling of tracer injection in naturally fractured reservoirs using the Random Walk Particle Tracking."

4. RWPT simulation of the Midale pilot area CO_2 flooding

4.1 Overview

In the last few decades, CO_2 emissions to the atmosphere became a major environmental concern. There are a number of approaches used to reduce their negative environmental impact, and one of them is to sequester CO_2 into underground reservoirs.

The other case where CO_2 is injected into underground reservoirs is enhanced oil recovery (EOR). Pressure in many oil reservoirs is high enough for CO_2 to become miscible with the reservoir oil. This results in the reduction of oil viscosity and water-oil interfacial tension; thereby, oil left after primary production can be recovered (Klins 1984; Ravagnani et al. 2009).

Although CO_2 injections for EOR purposes have been practiced for more than 40 years, they have shown their potential as a means of sequestration very recently (Huo and Gong 2010; Plasynski and Damiani 2008). One of the technical advantages of using oil and gas reservoirs for sequestration purposes is their sealing properties. If hydrocarbons remained trapped in the reservoir for thousands of years, we can expect not to have CO_2 migration through the cap rock to the surface, as long as rock integrity is not damaged while sequestering.

Naturally fractured reservoirs (NFR) are ideal candidates for sequestration due to the high injectivity and storage capability of the matrix (Trivedi and Babadagli 2008; 2009). On the other hand, oil recovery in a great portion of this type of reservoirs is very low compared to conventional oil reservoirs because a significant amount of oil in the matrix is bypassed. Hence, the NFRs are good candidates for CO_2 flooding from an oil production point of view as well and interestingly, it turned out to be one of the most commonly applied EOR method in NFRs (Schechter 2005). However, a highly conductive fracture network involves additional risks, such as early CO_2 breakthrough or leakage through the

A version of this chapter was submitted for publication. Stalgorova, E. and Babadagli, T. 2011. "Modified Random Walk - Particle Tracking model for CO_2 flooding/sequestration in naturally fractured oil reservoirs."

wellbore. Therefore, it is critical to accurately model miscible CO_2 flow in fractured media before any application of CO_2 flooding or sequestration.

In this chapter, we present modification of the Random Walk Particle Tracking (RWPT) technique to simulate fully miscible CO_2 injection, including matrix-fracture interaction, in naturally fractured oil reservoirs.

The fracture network obtained and calibrated against tracer test data (Chapter 3) was used for modeling. We introduced additional parameters required to adapt the model for CO_2 flooding case and investigated the effect of these parameters on the performance.

In addition, we performed a history match study to reproduce the observed CO_2 production rate and discussed the possibility of using the same technique to simulate CO_2 sequestration in naturally fractured reservoirs while injecting it for oil recovery.

The simplifications used in the modeling and the limitations of the suggested technique are described in this chapter as well.

4.2 Statement of the problem and solution methodology

Field scale modeling of miscible CO_2 flow in a naturally fractured reservoir is a challenging task because of the fact that representing the real physics (including matrix-fracture interaction, diffusion and dispersion, interaction between phases, change in fluid properties, etc.) mathematically on complex fracture network structures (a large number of fractures with varying geometrical and physical properties and a highly heterogeneous matrix) is rather difficult.

Conventional ways for that kind of simulation involve significant simplification in the media description, while the physics of the process is captured rather elaborately. For example, using dual continuum compositional modeling in commercial simulation allows describing various physical aspects of the process such as change in fluid composition, fluid diffusion into the matrix, etc. However, the fracture network in this case is represented by a uniform orthogonal grid resulting in poor representation of reservoir heterogeneity and, more importantly, connectivity.

A version of this chapter was submitted for publication. Stalgorova, E. and Babadagli, T. 2011. "Modified Random Walk - Particle Tracking model for CO_2 flooding/sequestration in naturally fractured oil reservoirs."

In this study, we approached the problem in a different way and used a relatively simplified description of the fluid flow itself, while the description of the media (naturally fractured reservoir) remained very detailed. Each fracture was modeled separately, having its own geometrical and physical properties, so that fracture network connectivity information is preserved. We considered a fully miscible case in which oil in the matrix and fracture mixes with CO_2 flowing in fractures at first contact, as the given pressure and temperature in the reservoir is suitable for this.

In Chapter 3 we introduced a non-classical technique for field scale modeling of naturally fractured reservoirs. The technique known as Random Walk Particle Tracking (RWPT) was applied to model one phase fluid flow in fractured porous media. The main advantage of the suggested algorithm is that each fracture can be modeled individually, which prevents the loss of information about the fracture network connectivity. At the same time, because the physics of the flow is described in a simplified way, simulation requires reasonable computational time.

In previous chapter we generated a discrete fracture network for the Midale CO_2 pilot area based on known geological data and calibrated it against the tracer test data using RWPT. In this chapter we modify the technique to simulate the miscible CO_2 flooding on this network, adding the matrix effect. Results of simulation were compared with results from actual CO_2 flooding and history matching exercise was performed.

Additionally, we investigated the effect of different parameters on the result of simulation.

4.3 Algorithm description

The algorithm description summarized below is similar to the algorithm given in Chapter 3, but includes some modifications to capture certain CO_2 flooding features as well as the matrix-fracture interaction.

To be able to simulate CO_2 injection in a highly fractured reservoir within a reasonable computational time, we modeled the fluid flow in a simplified way. To justify these simplifications, we have to make the following assumptions:

A version of this chapter was submitted for publication. Stalgorova, E. and Babadagli, T. 2011. "Modified Random Walk - Particle Tracking model for CO_2 flooding/sequestration in naturally fractured oil reservoirs."

1) Interaction between phases can be neglected.
2) Each fracture can be represented by a sub vertical rectangle.
3) Flooding consists of several periods, and production and injection rates are constant within each period.

The example selected for simulation is the CO_2 pilot test area in the Midale oil field in Canada. The following characteristics of this project show that the above assumptions are valid:

1) CO_2 flooding was performed at fully miscible conditions as is in the field application (Beliveau et al. 1993, Malik et al. 2006). Hence, it can be treated as a one-phase flow. Note that at a later stage of the project, CO_2 flooding was followed by water injection. For this stage, we used a constant permeability multiplier to model decreasing effective permeability due to interaction between the phases.
2) If a fracture has a complicated geometry, we can always represent it by several rectangles joined together.
3) CO_2 flooding in the Midale field consisted of 6 periods, and injection and production rates were nearly constant within each period (**Table 4-1**).

Modeling CO_2 flooding with Random Walk - Particle Tracking (RW-PT) consists of five steps:

1) Fracture network generation,
2) Calculation of pressure field for each constant injection rate period,
3) Conversion of fracture network to a graph,
4) Simulation of CO_2 flooding using created graph,
5) Comparison of the simulated and observed results, and change of uncertain parameters and/or the fracture network accordingly for the history match.

We describe each step in detail below.

STEP 1: Fracture network generation

A discrete fracture network can be generated using the available geological information, which includes deterministic information (the size and location of major faults and fractures) as well as stochastic information (the existence and direction of the trend, average spacing between fractures, average length and width of fracture, etc.). For the case of RWPT modeling, each fracture is defined as a rectangle and has certain width and permeability. In order to simulate the injection and production, fractures passing through wells should be added.

In this study we used stochastically generated fracture networks for sensitivity analysis, and for the history matching exercise, we used the fracture network from a portion of the Midale field, which was obtained and calibrated against tracer test results in Chapter 3.

STEP 2: Calculation of pressure field for each constant injection rate period

After we introduce the fracture network, the next step is to calculate the pressure at each point by solving Darcy's equation numerically. First, we created a grid in which each fracture from the earlier generated network was represented by thin blocks of high permeable cells; the width and permeability of each fracture inherited from the DFN (**Figure 4-1**).

The next step was to introduce the wells and to run the simulation (ECLIPSE was used in this particular study). This simulation is only used to find the pressure field. Therefore, we can use just one phase (water) and keep the total reservoir volumes of the produced and injected fluid same, as was done in the field. For example, in the case of the Midale CO_2 flooding during the first period (Table 5-1), they injected 22900 m^3 of CO_2 (reservoir conditions), and produced 1150 m^3 of oil, 4150 m^3 of water, 15220 m^3 of CO_2 and 1025 m^3 of gas (all in reservoir conditions). Thus, the total produced volume during this period was 21545 m^3. To find the pressure field for that period, we simply modeled water injection and production with constant rates so that at reservoir conditions, the total injected volume is 22900 m^3 and the total produced volume is 21545 m^3.

Obviously, if the DFN contains many fractures, the simulation model will have a large number of cells which results in enormous computational time. However, because the model had only one phase and we only had to simulate several time steps (one for each constant rate period; in the case of Midale CO_2 flooding there will be six time steps), in practice, the computational times were not remarkably high.

STEP 3: Conversion of fracture network to a graph

The end of each fracture and all fracture intersections should be represented by vertexes and we started with creating a set of vertexes. To create the edges of a graph, we added an edge between those vertexes, which were connected by a fracture (fracture edges). To introduce flow through the matrix, we also took a circle of a certain radius around each vertex and added edges between the central vertex and each other vertex within the circle (matrix edges). The radius of the circle (*Rad*) is the uncertain parameter. **Figure 4-2** shows the way the fracture network is represented as a graph. **Figure 4-3** illustrates the vertexes connected by fracture and matrix edges.

The next step is to assign certain properties to the vertexes and edges. The procedure is similar to the one described in Chapter 3, but certain modifications for CO_2 flooding was introduced:

1) Each vertex: Indicate if it belongs to any well,
2) Pressure value (P) for each vertex at each simulation period: use the result of the classical simulation from STEP 2,
3) Obtain the pressure potential for each vertex at each simulation period using $\Psi = P - \rho g d$ where P is pressure, ρ is density of water, and d is depth of the vertex,
4) Define the length for each edge (L) (it is the distance between two corresponding vertexes),

5) Define the width and height for each fracture edge (W and h): we defined width and height for each fracture in STEP 1; fracture edges inherit these properties from the corresponding fractures.
6) Calculate the crossection area for each fracture edge using: $A=Wh$,
7) Effective crossection area for each matrix edge (A_e): there is no way to define the effective crossection area; therefore, it is used as the uncertain parameter. Initially, we assign it any feasible value and then change it during the history matching process.
8) Assign the permeability for each edge (k): the permeability of each fracture was defined in STEP 1; fracture edges inherit permeability from the corresponding fractures.
9) Calculate the pressure gradient for each edge: if the edge connects vertex x with potential Ψ_x and vertex y with potential Ψ_y and the distance between vertexes is L, then the pressure gradient will be $\nabla\Psi_{xy} = (\Psi_x - \Psi_y)/L$,
10) Calculate the velocity for each edge: v_{xy} is undefined if $\nabla\Psi_{xy}$ is negative (there is no flow from x to y), and in the case of positive $\nabla\Psi_{xy}$ velocity is calculated from Darcy's Law: $v_{xy} = \nabla\Psi_{xy}\frac{k}{\mu}\cdot p_m$

where k is the permeability of the edge, μ is the viscosity of the water, and p_m is a multiplier for permeability, which takes into account that flow velocity can decrease because of the interaction between phases or friction against the fracture walls. To increase model flexibility, the values of p_m can be different for on-trend and off-trend fracture edges. Additionally, we can change the p_m depending on the period of flooding. During CO_2 flooding, the p_m is close to one because the flooding is conducted at miscible conditions. However, once water post-flood is started it makes sense to decrease the p_m to represent phase interaction.

STEP 4: Simulation of CO_2 flooding using created graph

The flow of injected CO_2 is represented by the movement of a large number of particles along the graph edges. Each particle represents a certain volume of CO_2 (in reservoir conditions). Hence, if we know the CO_2 injection rate, we can calculate how many particles (or how much CO_2) are injected every second.

As CO_2 is injected at fully miscible conditions, its concentration is governed by the Advection Dispersion Equation (ADE):

$$\frac{\partial C(x,t)}{\partial t} = -\frac{\partial}{\partial x}\left(v(x,t)C(x,t)\right) + \frac{\partial}{\partial x}\left(D(x,t)\frac{\partial C(x,t)}{\partial x}\right) \qquad (4\text{-}1)$$

The flow described by Eq. (4-1) can be simulated by a large number of particles, moving according to the following rule:

$$x(t+\Delta t) = x(t) + v\Delta t + z\sqrt{2D\Delta t} \qquad (4\text{-}2)$$

Here, x is a particle location, v is a velocity of the flow, z is a random number obtained from normal distribution with a mean of 0 and a standard deviation of 1, and D is a dispersion coefficient. Delay et al. (2005) and Salamon et al. (2006) provided details for this step of the calculations.

As we are tracking movement of the particle along the edges of the graph, we deal with one-dimensional flow at each step (x in the above equation is a 1-D coordinate).

If at time t the particle was at vertex x, and then moved to vertex y, Eq. (2) becomes:

$$L_{xy} = v_{xy}\Delta t + z\sqrt{2D\Delta t} \qquad (4\text{-}3)$$

L_{xy} and v_{xy} are the known properties of the edge connecting x and y. Hence, we can solve Eq. (4-3) to find how much time the particle takes to go from vertex x to vertex y:

$$\Delta t = \left(-\frac{z\sqrt{D}}{v_{xy}\sqrt{2}} + \sqrt{\frac{z^2 D}{2v_{xy}^2} + \frac{L_{xy}}{v_{xy}}} \right)^2 \qquad (4\text{-}4)$$

To simulate the CO_2 injection, we performed the following steps adapted from the tracer test simulation algorithm given in Chapter 3:

1) Assign a certain volume to each particle. Based on the injection period (t_{ti}) and injection rate, calculate how many particles are needed to represent injection.

   Assign a starting time t_{j0} to each particle j, in such a way that the starting times of all the N particles are distributed uniformly within the injection period: $t_{00}=0$, $t_{10}=(t_{ti}/N)$, $t_{20}=2(t_{ti}/N)$, $t_{30}=3(t_{ti}/N)$,… $t_{N0}=N(t_{ti}/N)$.

2) Release each particle at the injection well, and track its way through the reservoir:

   a. Initially particle j had a time t_{j0} and a vertex corresponding to the injection well.

   b. In each step, the particle moves from the current vertex to the next one, following one of the edges. The probability of the particle moving along the edge is equal to zero if the pressure potential of the edge is negative, and is proportional to the rate if it is positive:

   $Prob_{xy} \sim rate = (velocity) \times (crossection\ area) = v_{xy} A$

   where v_{xy} and A are the properties of the edge which were calculated at the STEP 3.

   c. The particle moves to the next vertex as described in (*b*). Then, its time is defined as $t_{j1}=t_{j0}+\Delta t$. Δt is calculated using Eq. (4-4).

   d. Continue steps (*b*) and (*c*) until the particle reaches the production well or ‘dead end’. The dead ends are the vertexes, which do not have any edges with a positive pressure gradient. If the particle reached the production well, record which well and at what time.

3) When all particles are released and tracked, we can convert the recorded data (the number of produced particles against time) to the production rate profile (CO_2 production rate vs. time) to analyse the results and compare them with the observed data.

STEP 5: Comparison of the simulated and observed results, and change uncertain parameters and/or fracture network accordingly for history match.

It is expected that the first simulation results will not match the observed data. Therefore, the sensitivity of each parameter on the results needs to be analyzed first. This will lead to a decision about how these parameters can be tuned for a desirable history match as described in the next section.

4.4 Effect of parameters

To analyze the sensitivity of the algorithm to different parameters, we used the fracture network model of the Midale CO_2 pilot area defined earlier and the CO_2 flooding production and injection data reported by Baxter (1990). The fracture network of the Midale area is characterized by the existence of two distinct sets of fractures: (1) On-trend (main) fractures (SW-NE) and (2) off-trend (secondary) fractures (SE-NW). The on-trend fractures dominate the system; their lengths and density are higher than those of the off-trend fractures. The off-trend fractures are perpendicular to the trend direction. The parameters tested and the responses of the simulations are presented in this section.

As mentioned earlier, the DFN is described by a number of stochastic parameters such as length distribution for on-trend and off-trend fractures, average spacing between fractures (for the main and secondary set), as well as distributions of fracture widths and permeabilities. Based on those stochastic parameters, we created a fracture network. Because the process involves randomness, the same set of parameters can result in many different fracture network realizations.

To perform a classical sensitivity analysis of one certain parameter, we have to fix all of the other parameters and vary the selected parameter. Then, the

relative effect of each parameter on the transport process is clarified. To ensure that the observed effects do not only take place for one stochastic realization, we ran sensitivity analyses for five different stochastic realizations. **Figures 4-4** through **4-21** illustrate the effects of different parameters. For each parameter, top and middle images illustrate its effect on CO_2 recovery and CO_2 breakthrough time, respectively. The bottom image compares CO_2 production rate curves for two values of the selected parameter.

An analysis of the relative effect of each parameter on CO_2 recovery and the breakthrough time is given below.

4.4.1 Effect of spacing between on-trend fractures

Figure 4-4 illustrates the effect of average spacing between on-trend fractures. When spacing between fractures increases, the number of fractures decreases, which results in a decreasing recovery factor (top image); also decreasing the number of fractures results in an earlier and sharper breakthrough (middle and bottom images).

4.4.2 Effect of C_h

The parameter C_h represents fracture network connectivity and is given by the following relationship:

C_h = (spacing between off-trend fractures)/(spacing between on-trend fractures)

Increasing C_h means increasing the spacing between the off-trend fractures that yield a decreasing number of off-trend fractures. This results in a slightly shorter breakthrough time (Figure 4-5, middle image). However, the effect of C_h on the CO_2 recovery factor is not very significant and may vary from case to case as seen in the top image of Figure 4-5.

4.4.3 Effect of fracture lengths

Figures 4-6 and 4-7 illustrate the effect of on-trend and off-trend fracture lengths, respectively. The lengths of on-trend fractures do not affect recovery or breakthrough time remarkably (Figure 4-6). The reason is that if the number of on-trend fractures is high, even if each of them is small, the fracture network

connectivity remains good. However, the lengths of off-trend fractures affect the CO_2 recovery significantly (Figure 4-7). Longer off-trend fractures connect more on-trend fractures with each other, yielding an increase in overall reservoir connectivity. This in turn results in higher recovery.

4.4.4 Effect of matrix permeability

As shown in Figure 4-8, higher matrix permeability will cause the larger part of the flow to go through the matrix, and as flow through the matrix is slower than flow through the fractures, the recovery factor decreases (Figure 4-8, top image). This is very crucial in terms of CO_2 sequestration. Increasing matrix permeability results in more CO_2 storage (due to strengthened CO_2 transfer by diffusion into the matrix) in this medium. As the breakthrough is related to the fracture network characteristics, the breakthrough time shows almost no change with changing matrix permeability (middle and bottom image in Figure 4-8).

4.4.5 Effect of fracture permeabilities

Effect of fracture permeabilities is illustrated in Figures 4-9 and 4-10 for on- and off-trend fractures, respectively. Increasing fracture permeability results in an increase in recovery for both on-trend and off-trend fractures. However, off-trend permeability plays a more significant role (compare bottom images of Figures 4-9 and 4-10) in this process. As can also be observed in Figure 3-4, the flow between the injection and production wells is not aligned with trend direction. Thus, the properties of the off-trend fractures become more critical on recovery. No remarkable change in breakthrough times was observed with changing fracture permeability for both on- and off-trend fractures (middle images of figures 4-9 and 4-10).

4.4.6 Effect of fracture widths

Effect of fracture widths is shown in Figures 4-11 and 4-12 for on- and off-trend fractures, respectively. An increase of on-trend fracture widths causes a decrease in flow velocity, which delays the breakthrough and decreases the recovery at early stages. The off-trend fracture widths do not affect the simulation

results in the same fashion. A possible reason is that the total length of the off-trend fractures (or density) is much smaller than that of the on-trend fractures.

4.4.7 Effect of matrix effective crossection area (A_e) and *Rad*

As mentioned in the algorithm description section, the parameter *Rad* controls how far a particle can move through the matrix (Figure 4-3). Increasing A_e, as well as increasing *Rad*, results in a larger portion of the flow going through the matrix. This causes a decrease in CO_2 recovery, which can be seen in Figures 4-13 and 4-14. More CO_2 storage in the matrix also has a positive effect on CO_2 sequestration.

4.4.8 Effect of fracture permeability multipliers

Permeability multipliers were used to represent the decrease in permeability due to the interaction between phases or friction against the fracture wall (see algorithm description). As one can expect, increasing the fracture permeability multiplier (for on-trend fractures and for off-trend fractures as well) results in a CO_2 recovery increase (Figures 4-15 and 4-16).

4.4.9 Effect of dispersion coefficient

The matrix dispersion coefficient almost does not affect the results of simulations (figures are not included). An increase in the fracture dispersion coefficient results in a smaller CO_2 recovery and a smoother rate profile (Figure 4-17).

4.4.10 Effect of CO_2 and water injection rates and durations

In field applications, certain parameters can be controlled by people while others naturally exist and are unchangeable. Engineers make decisions on development strategies based on these unchangeable field characteristics, and play with controllable parameters such as the rate and type of fluid to be injected and the duration of the process. To investigate the effect of these parameters on the simulation process, we considered the following injection scheme. First, we injected CO_2 for t_{co2} days with injection rate q_{co2}. The CO_2 injection was followed

by a period of water injection with injection rate q_w and duration of t_w days. The effect of q_{co2}, t_{co2}, q_w and t_w is shown in Figures 4-18 through 4-21.

Increasing the CO_2 injection rate resulted in an earlier breakthrough; however, the CO_2 recovery factor did not change significantly (Figure 4-18). The almost constant CO_2 recovery in the top image of Figure 4-18 was caused by the algorithm limitation. Note that fracture and matrix storage capacity was used while calculating pressure distribution but this property was not taken into account in further calculations. Each particle travels through the network until it reaches either a production well or a 'dead end'. Particles which reach production wells contribute to the simulated CO_2 production, while particles which end up in 'dead ends' represent the sequestrated CO_2. However, in the model, each 'dead end' can contain as many particles as needed, while in reality its capacity is limited by the pore volume of the surrounding fractures and the matrix. For that reason, the RWPT model is applicable only in the cases when injected volumes are small, compared to the pore volume of the reservoir. In other words, the RWPT model works well for predicting breakthrough times and CO_2 rate at the early stage of CO_2 injection, but it needs further developments for the late stages of CO_2 injection. Based on these observations, one may conclude that the RWPT method is useful for early stage development and reservoir characterization purposes (using any injection data, i.e., tracers, water, CO_2). Overcoming this limitation is a subject of future work.

Increasing the CO_2 injection duration slightly increases the CO_2 recovery factor (Figure 4-19). As discussed above, an increase in recovery is underestimated. In reality, after a certain point of injection, the reservoir cannot take any more CO_2, and almost all injected CO_2 is produced, and the CO_2 recovery factor increases significantly.

A higher water injection rate results in faster CO_2 recovery (Figure 4-20, bottom image). However, it does not affect the recovery factor itself (top image of Figure 4-20). Also, water injection parameters do not effect CO_2 breakthrough times because CO_2 breaks through before the water flood was started. Increasing

water injection duration results in a higher CO_2 recovery factor (Figure 4-21). A longer water post flood is more efficient in terms of oil recovery, but less effective in terms of CO_2 sequestration.

4.4.11 Analysis of the results

Finally, the data from Figures 4-4 through 4-21 are summarized in **Table 4-2** and a comparative analysis was performed. For each parameter, we reported the range within which this parameter was changed for the sensitivity analysis. For each particular fracture network realization (the total number is five), we calculated the spectrum, i.e., the maximum and minimum values, of CO_2 recovery factor and breakthrough time. The spectrum was different for different fracture network realizations as seen in Figures 4-4 through 4-21. To have a general idea, we report the minimum change in the CO_2 recovery factor and the breakthrough time (among five fracture network realizations) in one column and the maximum change in values in the other column. The 'relative importance' column indicates how each parameter affects the CO_2 recovery factor and breakthrough time. The most critical parameters affecting the CO_2 recovery are the on-trend fracture widths, the on-trend and off-trend permeability multipliers and the dispersion coefficient for fractures. Parameters affecting breakthrough time the most are the on-trend fracture widths, and the on-trend and off-trend permeability multipliers. Considering the relative importance of the weight of these parameters on the process, one should pay attention to them in a reservoir characterization study to be used in building the fracture network model.

4.5 History matching of the Midale CO_2 flood pilot

To validate the described RWPT algorithm, we modeled CO_2 flooding in the Midale CO_2 flood pilot area (the data is given in Baxter, 1990). As it is described in Chapter 3 we performed a tracer test simulation for the same area; as a result, we obtained a fracture network calibrated against the tracer test results. The pilot area has four injecting and four producing wells (Figure 3-4), which gives sixteen 'injector-producer' well pairs. The tracer test modeling study included simulating

results for all sixteen well pairs. Therefore, the obtained fracture network describes reservoir connectivity reasonably well. This validated network was used to model CO_2 flooding. Due to the unavailability of well-by-well production and injection data for CO_2 flooding, the history matching study was achieved using the combined CO_2 production rates of all producing wells.

The locations of the fractures remained the same as in the fracture network calibrated against tracer test results, but the other parameters were altered to achieve a history match. The following parameters were used as history matching parameters:

1) Matrix permeability
2) Fracture permeability: theoretically each fracture has its own permeability, but to have realistic number of parameters, we used just one permeability value for all on-trend fractures and one permeability value for all off-trend fractures
3) Fracture width: defined separately for on-trend and off-trend fractures.
4) A_e – effective crossection area for flow through the matrix
5) *Rad* – (this parameter was defined in the algorithm description section)
6) Permeability multipliers (different for on-trend and off-trend fractures)
7) Dispersion coefficients for fracture and for matrix

From a practical point of view, we desired to reduce the number of uncertain parameters to make the history matching faster. As we discussed in the previous section, an increase in matrix permeability, A_e or *Rad* affects the results of simulation in a similar way (essentially, the larger part of the flow goes through the matrix, which decreases CO_2 recovery). Hence, for history matching purposes, we can fix two out of these three parameters, and change only the remaining one. Additionally, we can eliminate the matrix dispersion coefficient as it almost does not affect the simulation results based on the observation in the previous section.

One of the advantages of the RWPT algorithm is that computational time for each run is relatively small (about 1 minute for the fracture network containing 400 fractures). This makes the algorithm a suitable candidate for computer-aided

history matching. The number of optimization algorithms can be used to vary the parameters automatically. In this study, we used Genetic and Simulated annealing Algorithms inbuilt in the MATLAB environment. Matrix and fracture permeabilities, widths of the fractures, A_e, fracture permeability multipliers and the dispersion coefficient were used as history matching parameters. The best match is shown in **Figure 4-22**.

As seen, the simulated results do not reproduce the observed data well even though the trend was captured. One may assume that it is because the optimization algorithm did not give the best possible combination of the values for uncertain parameters. However, from the sensitivity analysis exercise (Figures 4-4 through 4-17), we see that any change of uncertain parameters gives a plateau type of profile during the first 500 days, while the observed data has two major peaks (Figure 4-22). Thus, just the suggested tuning parameters are not enough to reproduce the observed data.

We realized that the reason for the peaks in the observed CO_2 production was due to variable injection rates during CO_2 flooding. For our modeling, we approximated CO_2 injection by a constant rate injection (in a way that the total injected volume remains the same as in the actual case) and because of this simplification, the simulation was not able to capture the peaks in CO_2 production. The actual CO_2 injection rate profile is shown in **Figure 4-23.** As seen, the injection was not performed at a constant rate and was even interrupted three times. Hence, substituting this injection profile by a constant injection rate was too much of a simplification.

On the basis of this observation, we divided the CO_2 injection period into eight smaller sub-periods (shown as vertical lines in Figure 4-23) to perform more detailed modeling. The algorithm was adjusted accordingly and we had to calculate the pressure field separately for each sub-period. After we ran the optimization for the new model (varying the same set of uncertain parameters as before: fracture permeabilities and widths, matrix permeability, A_e, fracture permeability multipliers and the dispersion coefficient), we were able to

reproduce the peaks in the CO_2 production profile. The best history match with the modified model is shown in **Figure 4-24**.

It is highly likely that we would be able to obtain an even better match if we could divide the CO_2 injection periods into smaller sub-periods. However, increasing the number of sub-periods significantly increases the computational time resulting in missing the main advantage of the RWPT algorithm. While performing RWPT modeling, it is important to find a balance between calculation accuracy and computational time. This modeling approach is more applicable to the cases where injection and production history consists of just a few periods, each of which has almost constant production and injection rates.

4.6 Summary and concluding remarks

The Random Walk Particle Tracking (RWPT) algorithm was modified for the case of miscible CO_2 injection in highly fractured reservoirs and tested on a field case through a history match exercise. A sensitivity analysis was performed and the effects of fifteen different parameters on CO_2 recovery factor, breakthrough time and the production rate profile were described. As opposed to conventional modeling, where fluid flow is described in detail and the fracture network is simplified, the RWPT algorithm used a simplified description of the flow, while detailed fracture network characteristics were preserved. The fracture network, obtained and calibrated against tracer tests (see Chapter 3), was used to model the pilot CO_2 flooding test in the Midale field. A computer-aided history matching technique was used to tune the model. The resulting model reproduced the observed data reasonably well.

A few observations in this study were thought to be critical and need to be highlighted as follows:

1) The algorithm allows for the modeling of each fracture separately, without averaging fracture properties. Hence, we do not lose the details of the fracture network geometry and properties.
2) Physics of the miscible flooding is not captured in all of its complexity. A number of phenomena, such as matrix-fracture

interaction, diffusion of CO_2 into oil stored in the matrix, density difference between CO_2 and oil and interaction between different phases are not modeled directly. Instead, scaling parameters are used to mimic those effects.

3) The suggested model has a large number of history matching parameters. This gives an opportunity to fine-tune the model even though manual history matching may take unreasonable time. On the other hand, a comparatively short computational time of each simulation run makes the model suitable for computer-aided history matching.
4) A sensitivity analysis for the RWPT model was performed. It was shown that on-trend fracture widths, on-trend and off-trend permeability multipliers and the dispersion coefficient for fractures affected the CO_2 recovery factor the most. The most influential parameters affecting the breakthrough time were on-trend fracture widths, and on-trend and off-trend permeability multipliers.
5) The RWPT algorithm is applicable for cases where injection and production history can be represented by several periods of a sub-constant injection and production rate. A larger number of periods results in greater computational times.
6) The suggested simulation approach is applicable when injected volumes are small compared to the pore volume of the reservoir. In other words, the RWPT model works well for predicting breakthrough times and CO_2 rate at the early stage of CO_2 injection, but is not capable of describing the flow behavior at the late times. Therefore, one has to be careful in using it for the analysis of very late stages of CO_2 sequestration applications. Based on these observations, one may conclude that the RWPT method proposed in this paper is useful for early stage development and reservoir characterization purposes (using

any injection data, i.e., tracers, water, CO_2). Overcoming this limitation is a subject of future work.

4.7 Tables

Table 4-1. Stages of the CO_2 flooding in the Midale pilot (Baxter 1990).

name	start	finish	duration	description	oil produced	water produced	CO2 produced	gas produced
	date	date	days		rm^3	rm^3	rm^3	rm^3
CO_2 flood	06/07/1986	05/08/1987	395	total CO_2 injected - 22900 rm^3	1150	4150	15220	1025
blowdown-1	05/08/1987	18/11/1987	105	production; no injection	570	1930	1980	260
shut-in period	18/11/1987	13/01/1988	56	all wells are shut				
blowdown-2	13/01/1988	23/03/1988	70	production; no injection	320	1720	190	90
brine post-flood-1	23/03/1988	23/04/1988	31	water injection; rates are similar to CO_2 injection rates; no production				
brine post-flood-2	23/04/1988	30/06/1988	68	water injection and production	280	7520	190	40
				Total:	**2320**	**15320**	**17580**	**1415**

Table 4-2. Relative effects of each parameter on CO_2 recovery factor and the breakthrough time.

parameter name	units	minimum value	maximum value	minimum change in RF	maximum change in RF	relative importance	minimum change in BT time, days	maximum change in BT time, days	relative importance
Spacing between on-trend fractures	m	0.9	2.1	0.049226	0.103947	medium	24	45	medium
C_h		3.8	5.4	0.017874	0.044465	low	14	20	medium
Average length for on-trend fractures	m	180	260	0.018094	0.030913	low	6	16	low
Average length for off-trend fractures	m	40	80	0.055013	0.102042	medium	8	22	low
Matrix permeability	mD	35	75	0.05157	0.071202	medium	9	23	low
On-trend fracture permeability	mD	150000	350000	0.009669	0.031865	low	9	16	low
Off-trend fracture permeability	mD	150000	350000	0.072155	0.082191	medium	4	24	low
On-trend fracture width	m	0.01	0.05	0.20555	0.23866	**high**	44	76	**high**
Off-trend fracture width	m	0.01	0.05	0.066881	0.107683	medium	13	28	medium
A_e	m^2	0.3	8.3	0.021464	0.05267	low	8	19	low
Rad	m	5	9	0.013991	0.02498	low	5	22	low
On-trend p_m		0.2	1	0.39784	0.433954	**high**	74	122	**high**
Off-trend p_m		0.2	1	0.369272	0.426629	**high**	65	95	**high**
Dispersion coefficient (fracture)	m^2/sec	0.00001	0.0001	0.172805	0.216025	**high**	16	40	medium
Dispersion coefficient (matrix)	m^2/sec	0.00001	0.0001	0.004175	0.207601	low	5	22	low
q_{co2}	rm^3/day	70	190	0.007374	0.026321	low	17	23	medium
t_{co2}	days	10	310	0.011797	0.046559	low	4	12	low
q_w	rm^3/day	70	190	0.002613	0.008307	low	3	9	low
t_w	days	10	310	0.05423	0.075418	medium	2	5	low

4.8 Figures

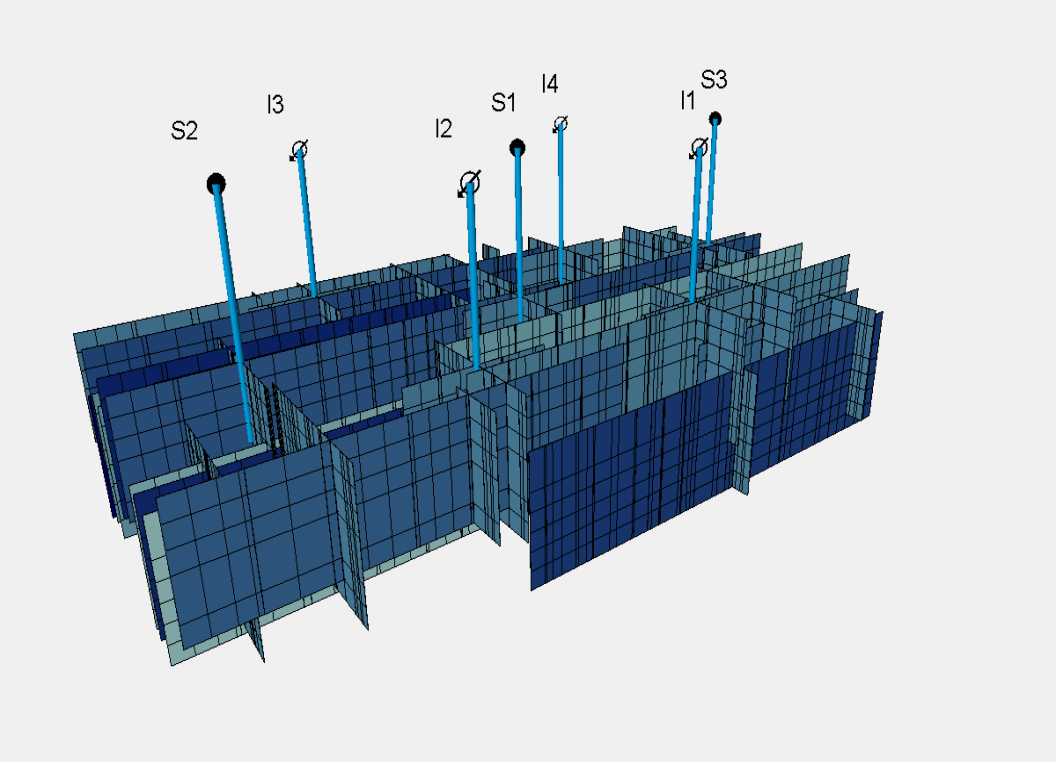


Figure 4-1. Fracture network represented by a classical simulation grid.
Only high permeable (fracture) cells are shown. Color shows permeability of each fracture.

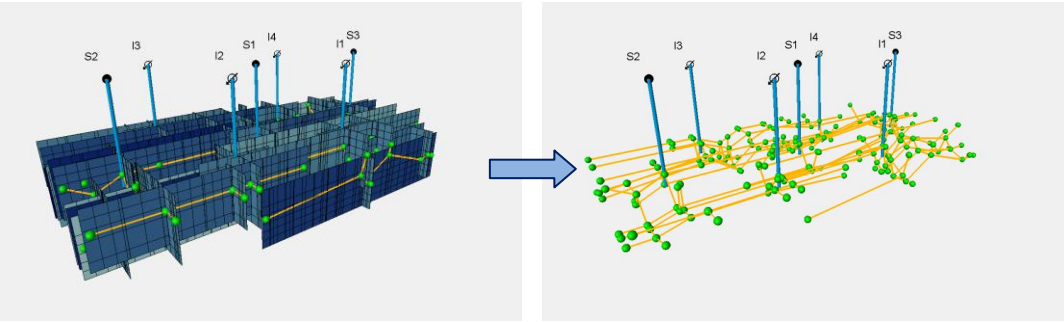


Figure 4-2. Graph created based on fracture network.
Vertexes are added at fracture ends and fracture intersections (green spheres). Edges are added between connected vertexes (yellow lines). Only this graph is used for further simulation.

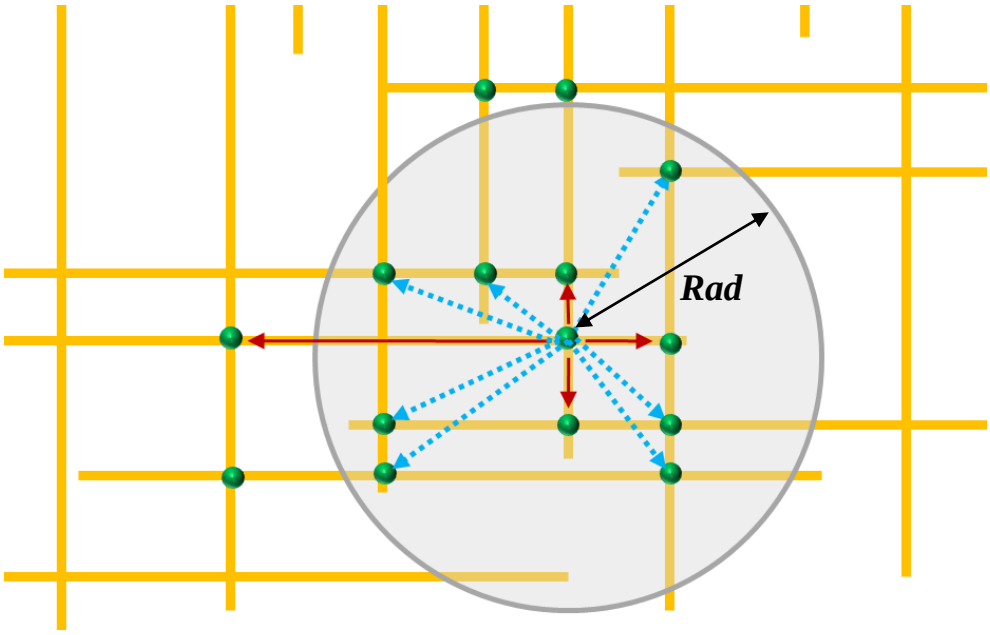
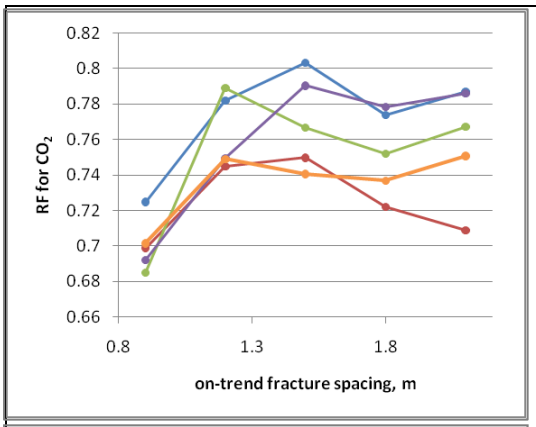
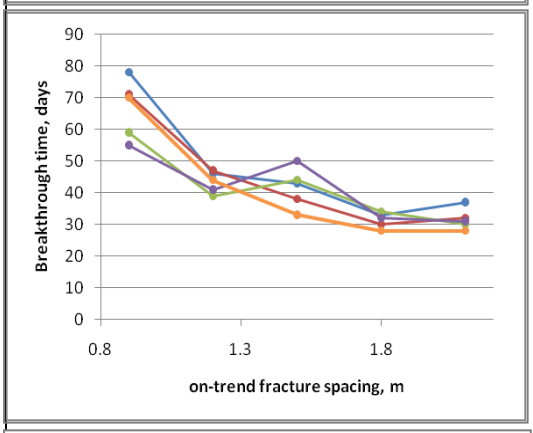


Figure 4-3. Fracture and matrix edges in the graph.
Yellow lines represent fractures; green spheres - vertexes of the graph. Vertexes may be connected by fracture edges (red solid arrows) or by matrix edges (blue dashed arrows).





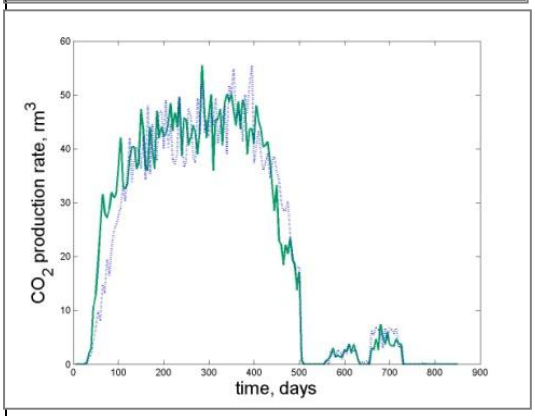
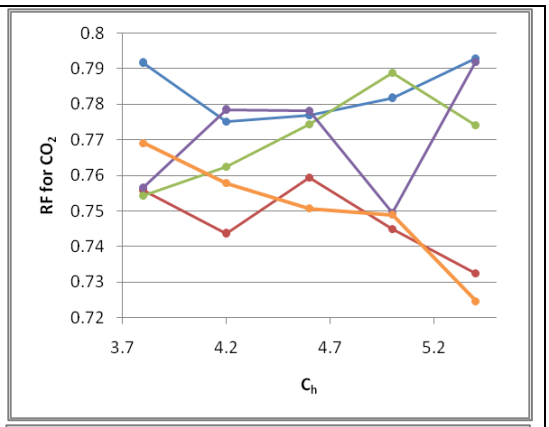
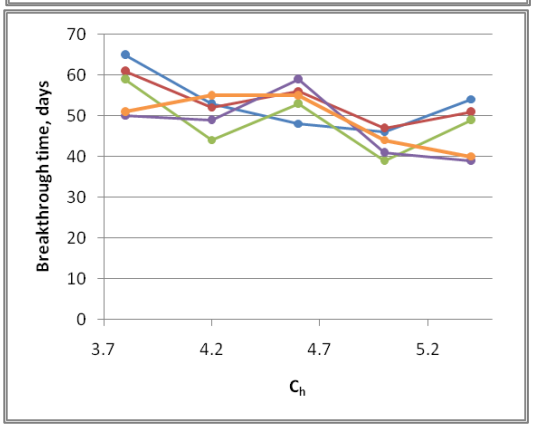


Figure 4-4. Effect of on-trend fracture spacing.

Top image: effect on CO_2 recovery factor. Middle image: effect on breakthrough time. (Each color represents changing fracture spacing for one fracture network realization.) Bottom image: effect on CO_2 production rate. Green solid line – spacing between on-trend fractures is 1.6 m; blue dotted line - spacing between on-trend fractures is 1.2 m. Other parameters do not change.





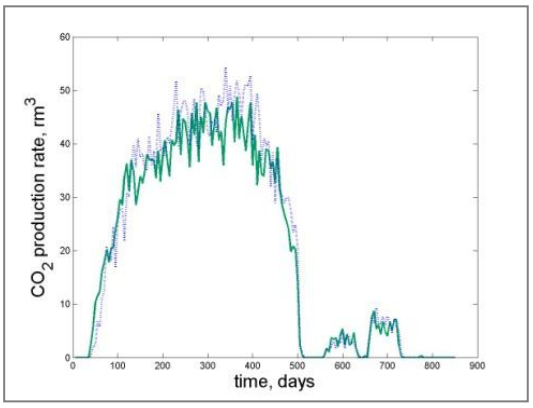
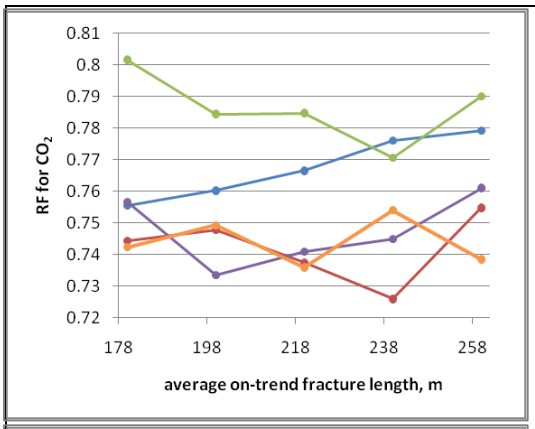
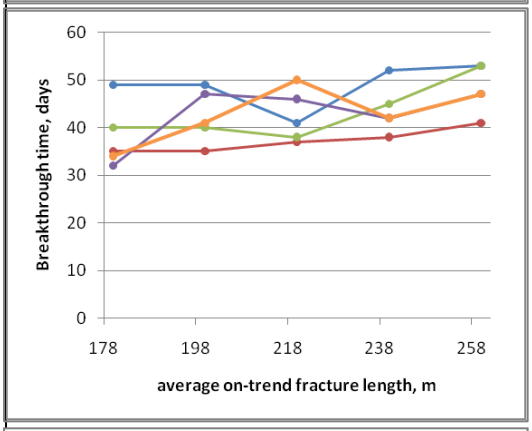


Figure 4-5. Effect of C_h.

Top image: effect on CO_2 recovery factor. Middle image: effect on breakthrough time. (Each color represents changing fracture spacing for one fracture network realization.) Bottom image: effect on CO_2 production rate. Green solid line – C_h = 5.4; blue dotted line - C_h =3.8. Other parameters do not change.





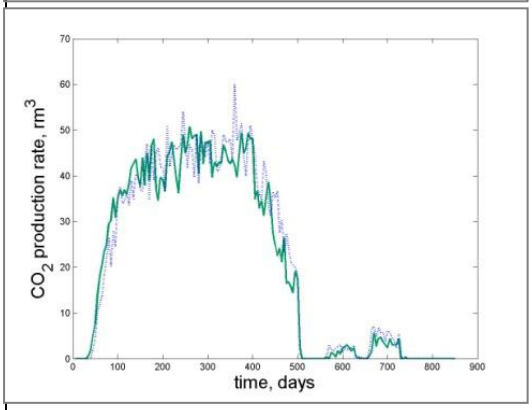
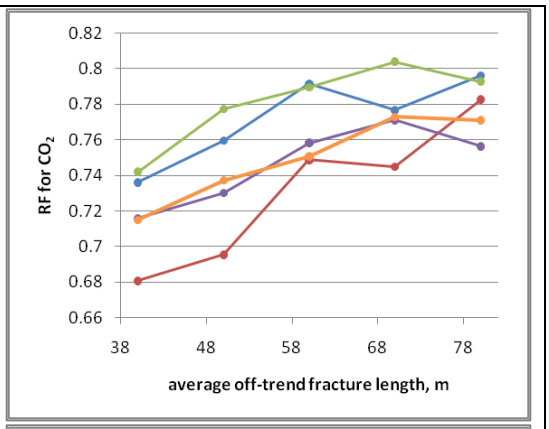
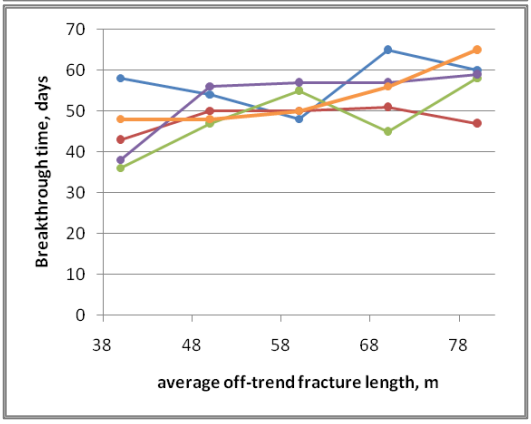


Figure 4-6. Effect of on-trend fracture length.

Top image: effect on CO_2 recovery factor. Middle image: effect on breakthrough time. (Each color represents changing fracture spacing for one fracture network realization.) Bottom image: effect on CO_2 production rate. Green solid line – average length is 180 m; blue dotted line – average length is 260 m. Other parameters do not change.





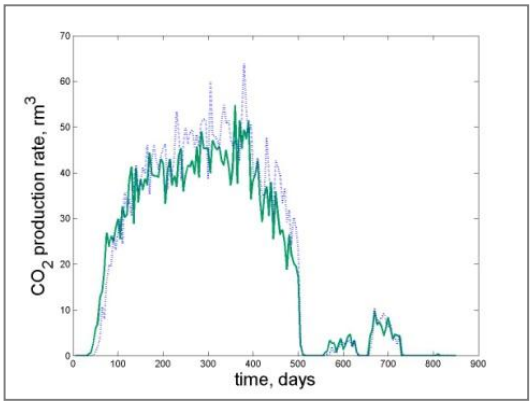
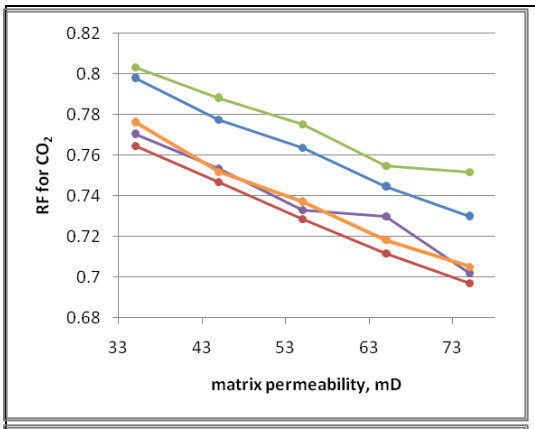
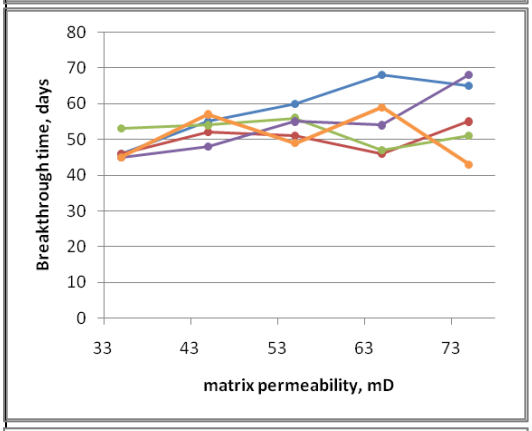


Figure 4-7. Effect of off-trend fracture length.

Top image: effect on CO_2 recovery factor. Middle image: effect on breakthrough time. (Each color represents changing fracture spacing for one fracture network realization.) Bottom image: effect on CO_2 production rate. Green solid line – average length is 40 m; blue dotted line – average length is 80 m. Other parameters do not change.





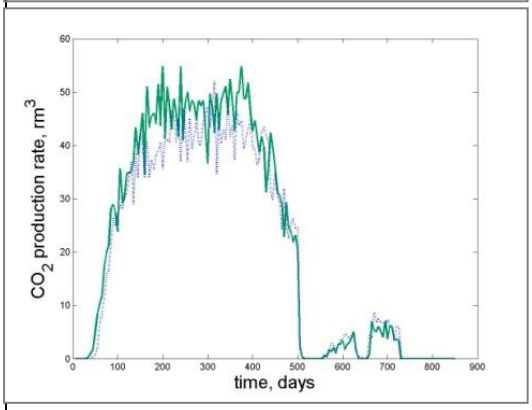
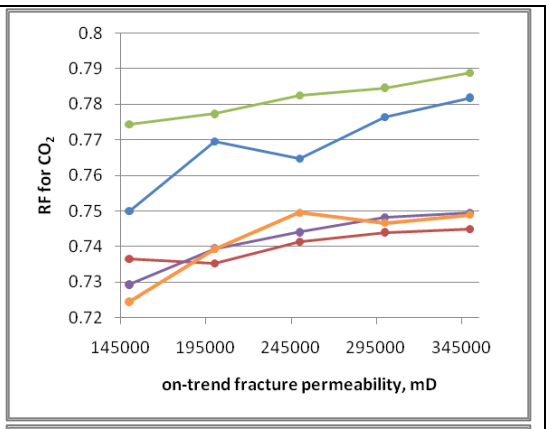
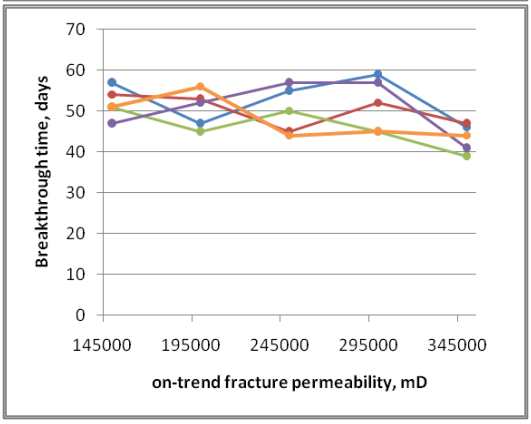


Figure 4-8. Effect of matrix permeability.

Top image: effect on CO_2 recovery factor. Middle image: effect on breakthrough time. (Each color represents changing fracture spacing for one fracture network realization.) Bottom image: effect on CO_2 production rate. Green solid line – matrix permeability is 35 mD; blue dotted line – matrix permeability is 75 mD. Other parameters do not change.





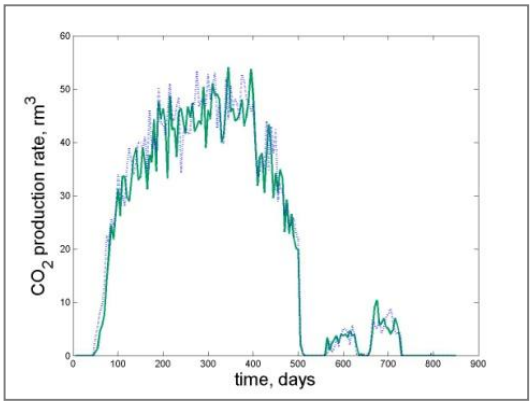
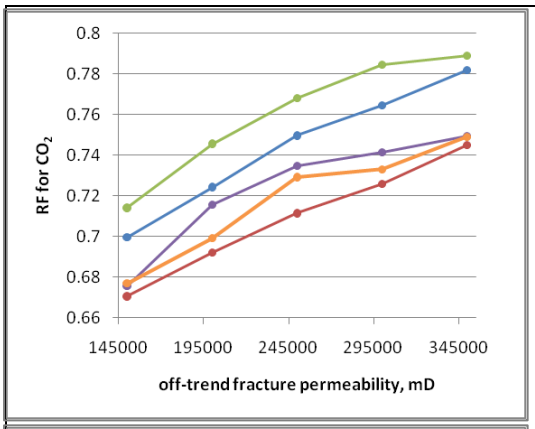
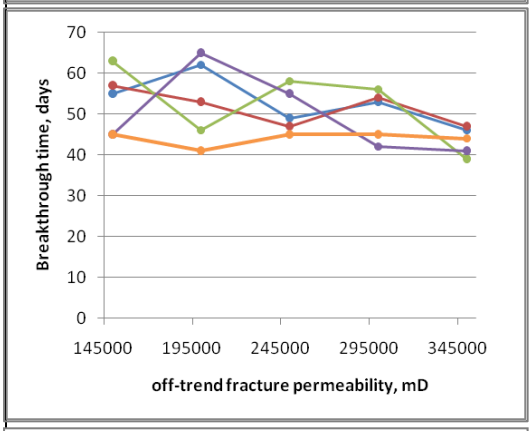


Figure 4-9. Effect of on-trend fracture permeability.

Top image: effect on CO_2 recovery factor. Middle image: effect on breakthrough time. (Each color represents changing fracture spacing for one fracture network realization.) Bottom image: effect on CO_2 production rate. Green solid line –permeability is 150000 mD; blue dotted line – permeability is 350000 mD. Other parameters do not change.





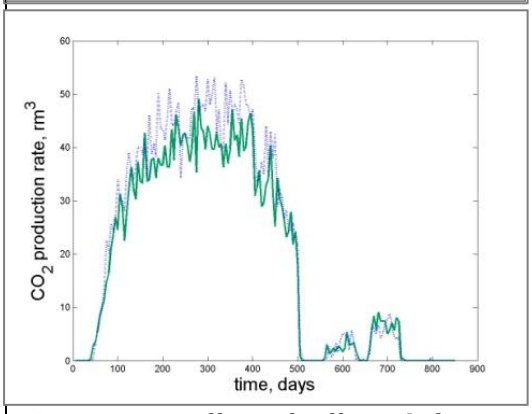
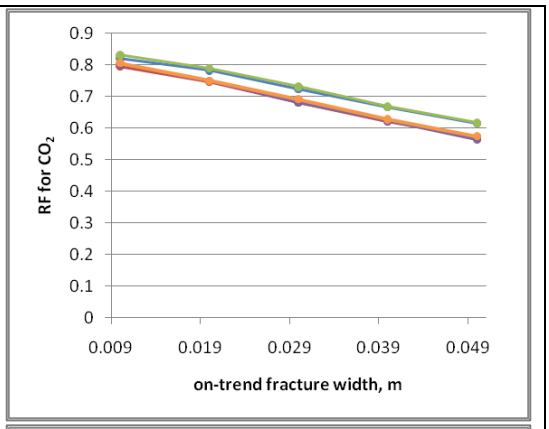
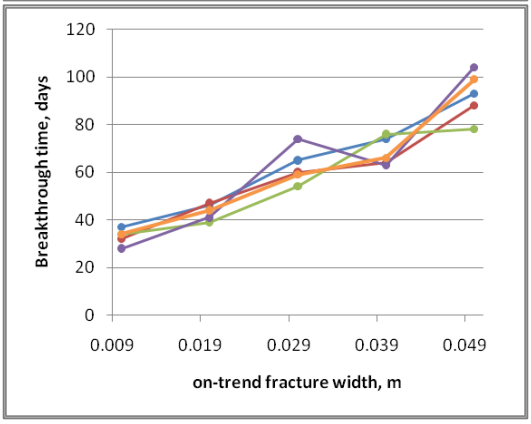


Figure 4-10. Effect of off-trend fracture permeability.

Top image: effect on CO_2 recovery factor. Middle image: effect on breakthrough time. (Each color represents changing fracture spacing for one fracture network realization.) Bottom image: effect on CO_2 production rate. Green solid line –permeability is 150000 mD; blue dotted line – permeability is 350000 mD. Other parameters do not change.





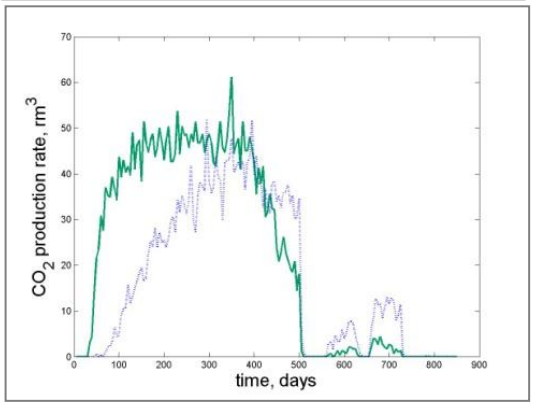
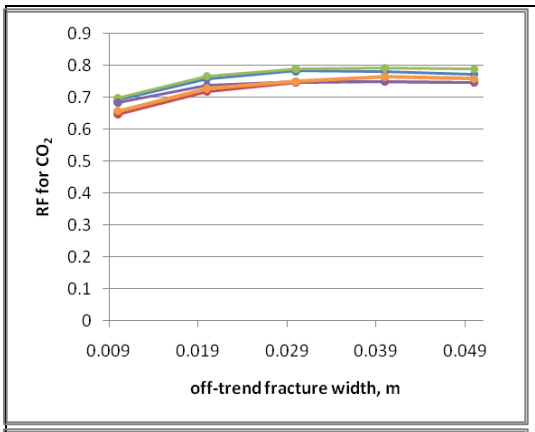
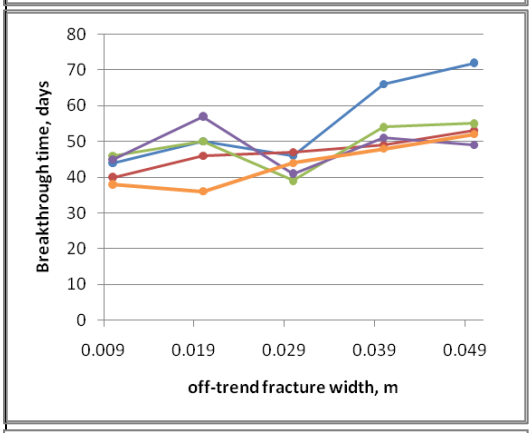


Figure 4-11. Effect of on-trend fracture width.

Top image: effect on CO_2 recovery factor. Middle image: effect on breakthrough time. (Each color represents changing fracture spacing for one fracture network realization.) Bottom image: effect on CO_2 production rate. Green solid line – width is 1 cm; blue dotted line – widths is 5 cm. Other parameters do not change.





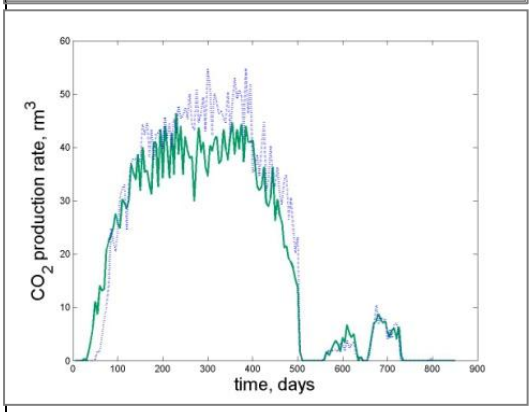
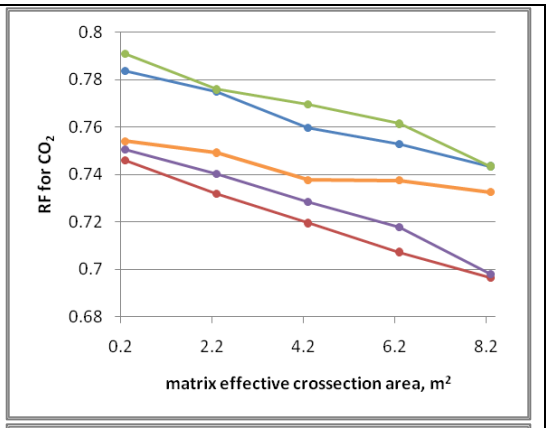
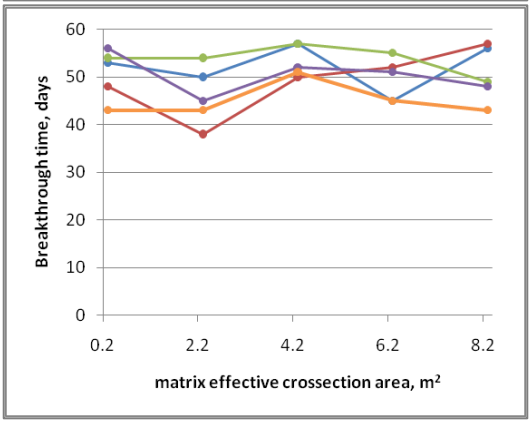


Figure 4-12. Effect of off-trend fracture width.

Top image: effect on CO_2 recovery factor. Middle image: effect on breakthrough time. (Each color represents changing fracture spacing for one fracture network realization.) Bottom image: effect on CO_2 production rate. Green solid line – width is 1 cm; blue dotted line – widths is 5 cm. Other parameters do not change.





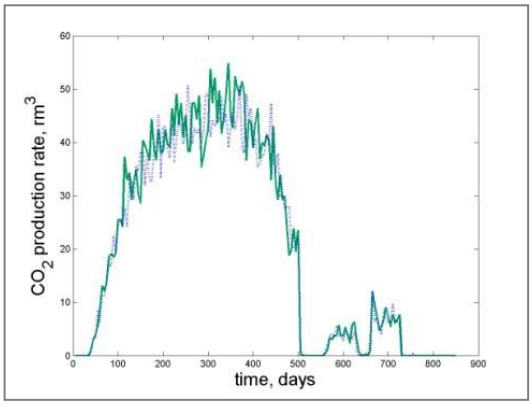


Figure 4-13. Effect of matrix effective crossection area

Top image: effect on CO_2 recovery factor. Middle image: effect on breakthrough time. (Each color represents changing fracture spacing for one fracture network realization.) Bottom image: effect on CO_2 production rate. Green solid line – A_e=0.3 m^2; blue dotted line – A_e=8.3 m^2. Other parameters do not change.

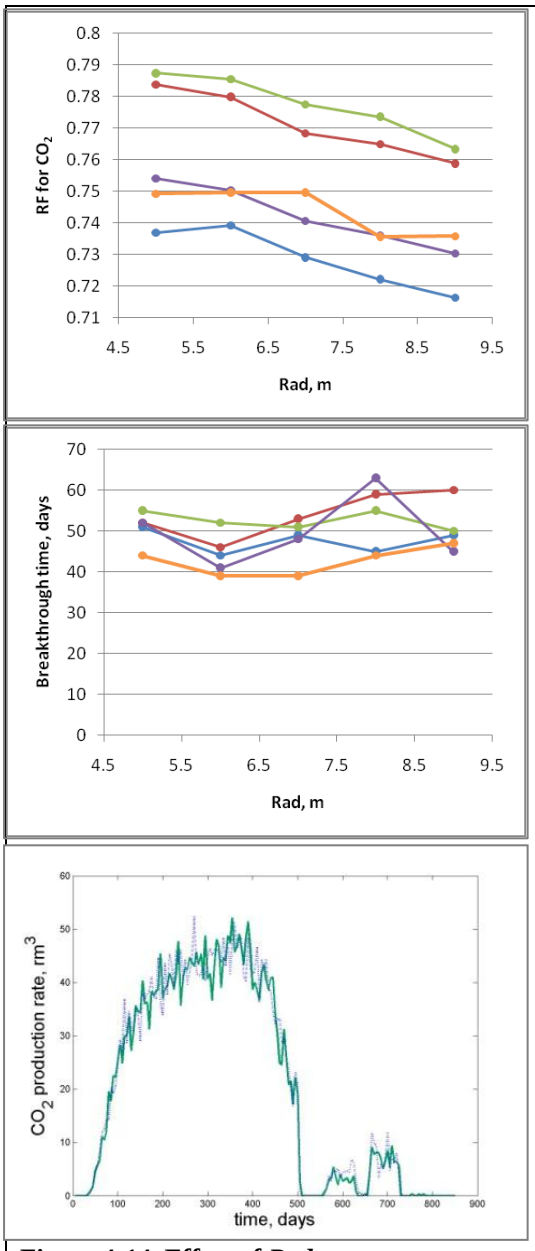


Figure 4-14. Effect of *Rad*.

Top image: effect on CO_2 recovery factor. Middle image: effect on breakthrough time. (Each color represents changing fracture spacing for one fracture network realization.) Bottom image: effect on CO_2 production rate. Green solid line – Rad=5 m; blue dotted line – Rad=9 m. Other parameters do not change.

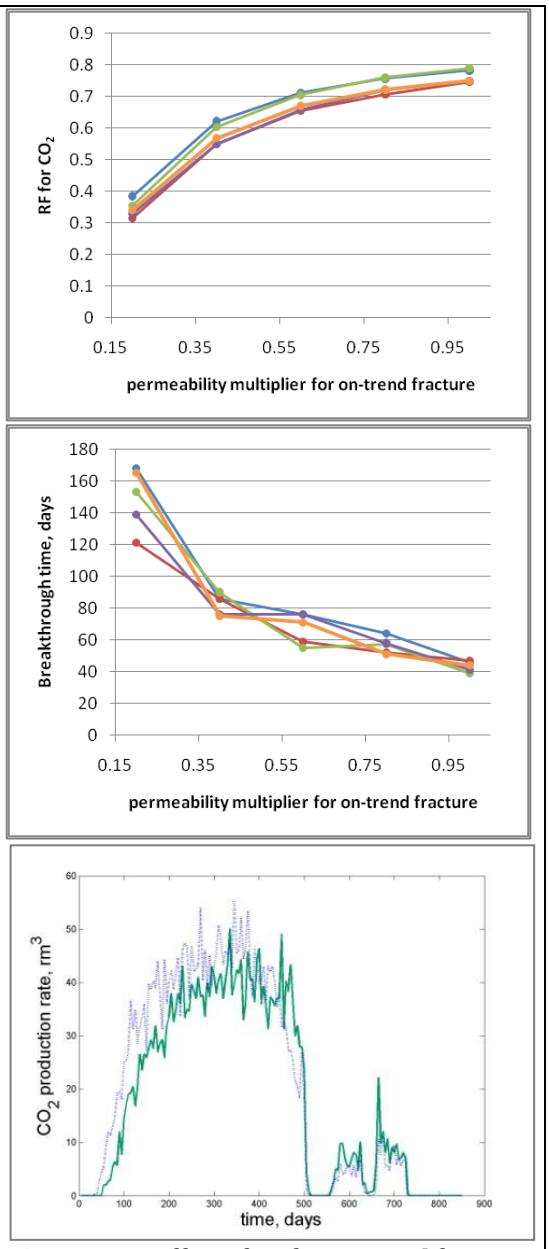
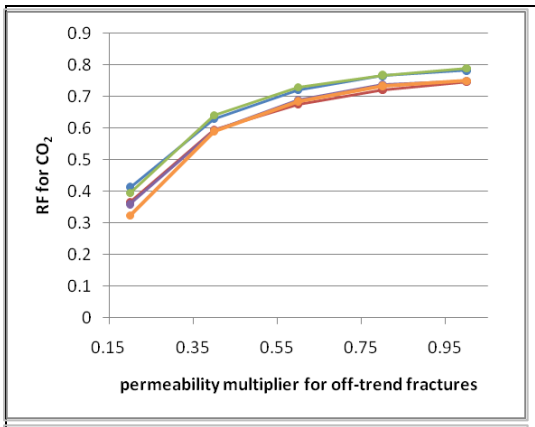
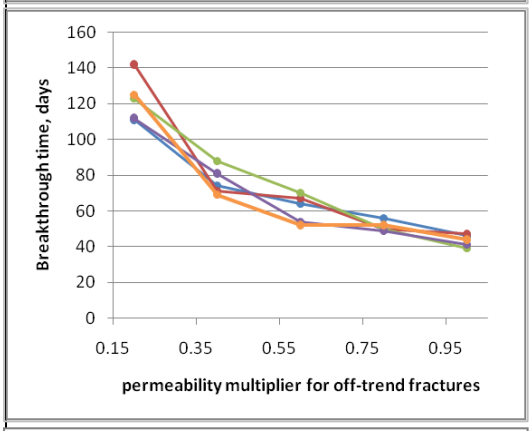


Figure 4-15. Effect of p_m for on-trend fracture

Top image: effect on CO_2 recovery factor. Middle image: effect on breakthrough time. (Each color represents changing fracture spacing for one fracture network realization.) Bottom image: effect on CO_2 production rate. Green solid line – p_m=0.6; blue dotted line – p_m=1. Other parameters do not change.





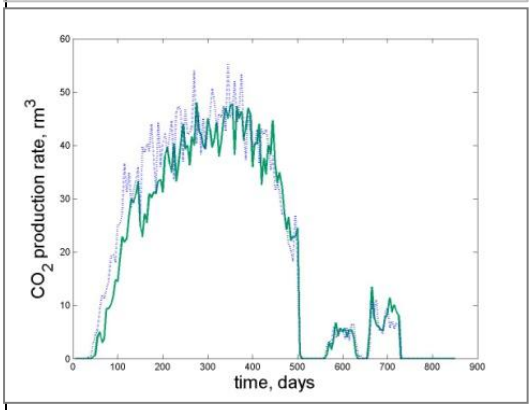
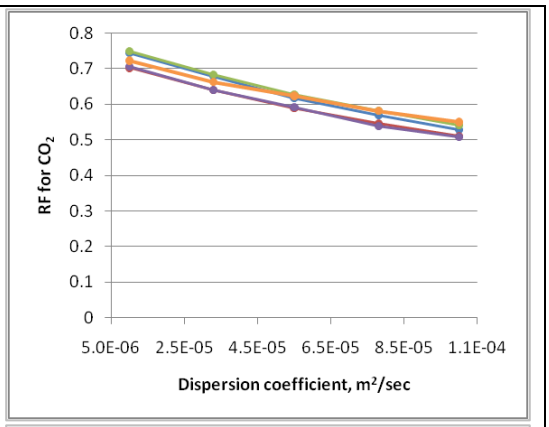
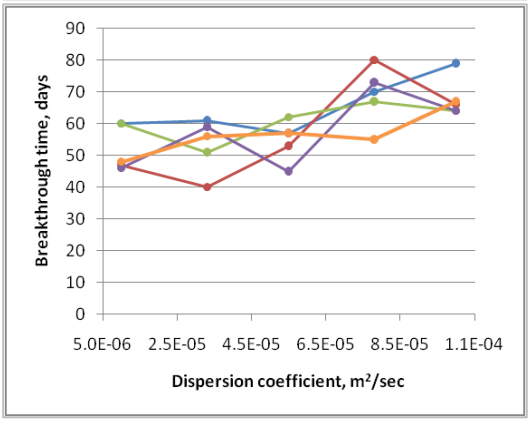


Figure 4-16. Effect of p_m for off-trend fractures.

Top image: effect on CO_2 recovery factor.
Middle image: effect on breakthrough time.
(Each color represents changing fracture spacing for one fracture network realization.)
Bottom image: effect on CO_2 production rate. Green solid line – p_m=0.6; blue dotted line – p_m=1. Other parameters do not change.





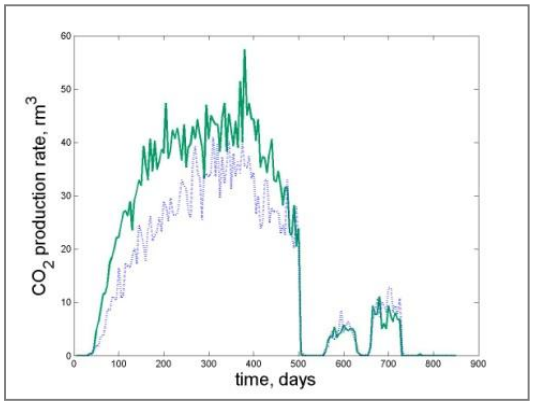
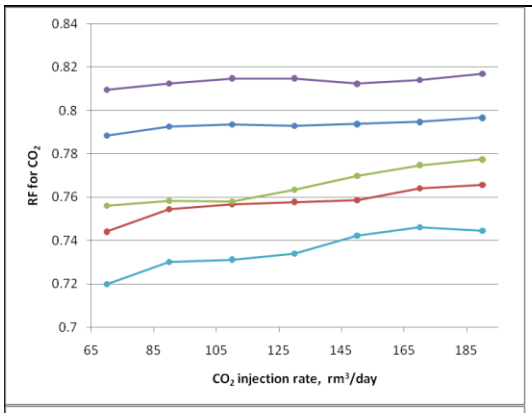
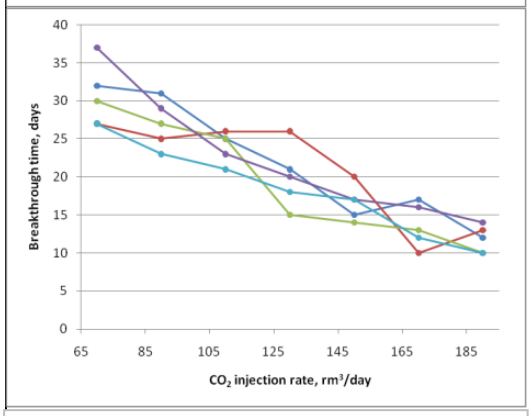


Figure 4-17. Effect of fracture dispersion coefficient.

Top image: effect on CO_2 recovery factor.
Middle image: effect on breakthrough time.
(Each color represents changing fracture spacing for one fracture network realization.)
Bottom image: effect on CO_2 production rate. Green solid line - $D=10^{-5}$ m^2/sec; blue dotted line – $D=10^{-4}$ m^2/sec. Other parameters do not change.





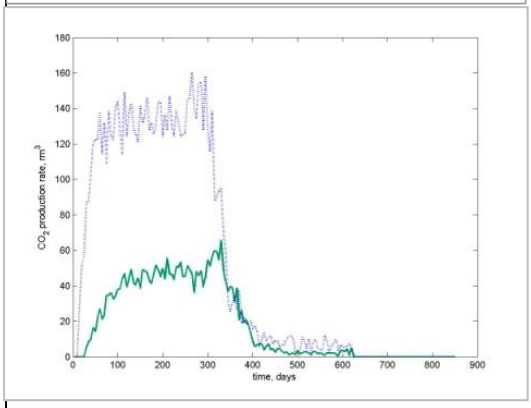
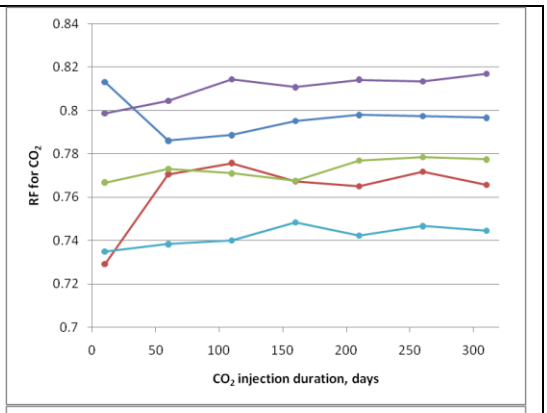
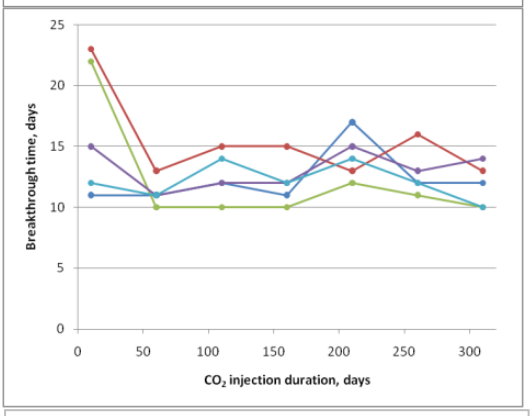


Figure 4-18. Effect of CO_2 injection rate.

Top image: effect on CO_2 recovery factor. Middle image: effect on breakthrough time. (Each color represents changing fracture spacing for one fracture network realization.) Bottom image: effect on CO_2 production rate. Green solid line – q_{co2}=70 m^3/day; blue dotted line – q_{co2}=190 m^3/day. Other parameters do not change.





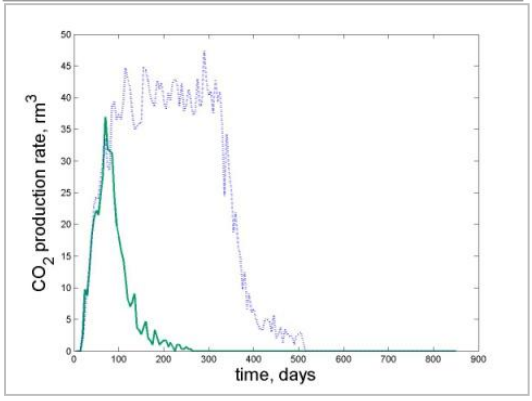
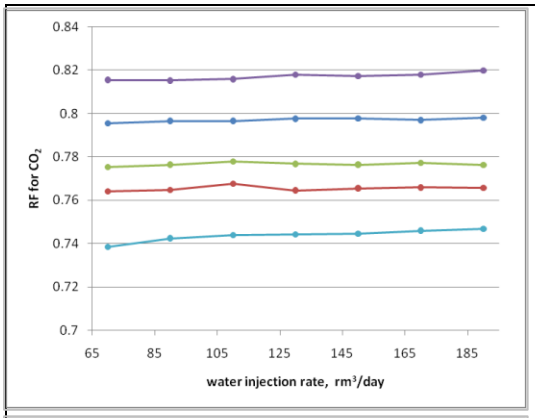
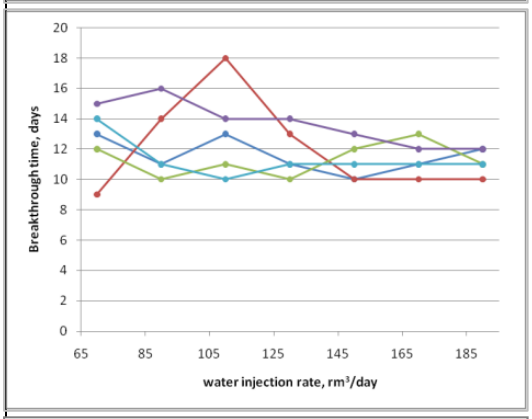


Figure 4-19. Effect of CO_2 injection duration.

Top image: effect on CO_2 recovery factor. Middle image: effect on breakthrough time. (Each color represents changing fracture spacing for one fracture network realization.) Bottom image: effect on CO_2 production rate. Green solid line – t_{co2}= 60 days; blue dotted line – t_{co2}= 310 days. Other parameters do not change.





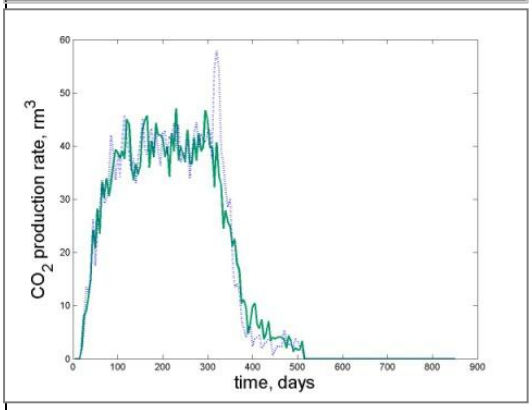
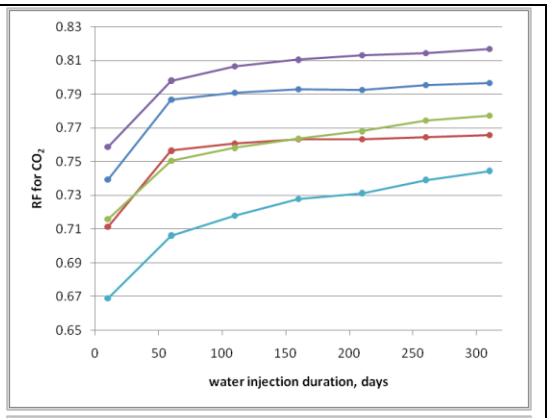
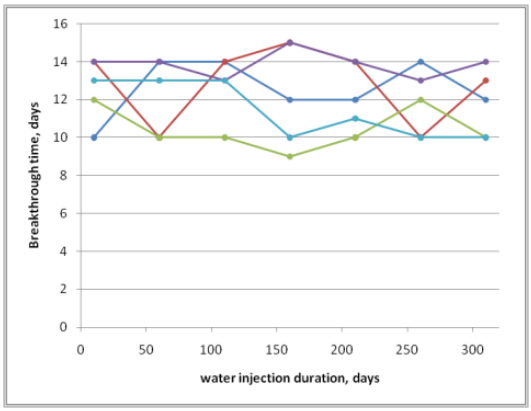


Figure 4-20. Effect of water injection rate.

Top image: effect on CO_2 recovery factor. Middle image: effect on breakthrough time. (Each color represents changing fracture spacing for one fracture network realization.) Bottom image: effect on CO_2 production rate. Green solid line – q_w=70 m^3/day; blue dotted line – q_w=190 m^3/day. Other parameters do not change.





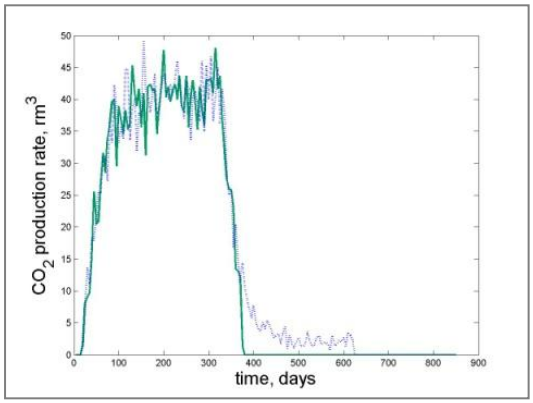


Figure 4-21. Effect of water injection duration.

Top image: effect on CO_2 recovery factor. Middle image: effect on breakthrough time. (Each color represents changing fracture spacing for one fracture network realization.) Bottom image: effect on CO_2 production rate. Green solid line – t_w= 60 days; blue dotted line – t_w= 310 days. Other parameters do not change.

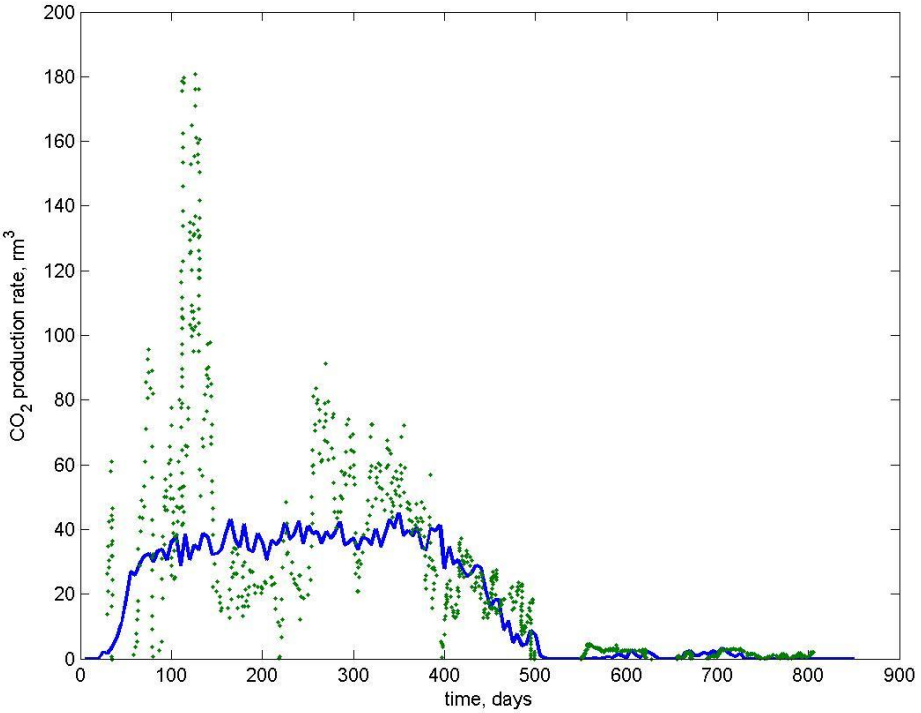


Figure 4-22. Best history match for the model, where CO_2 injection is represented by one period of injection at constant rate.
Green dots – observed data; blue solid line – simulated data.

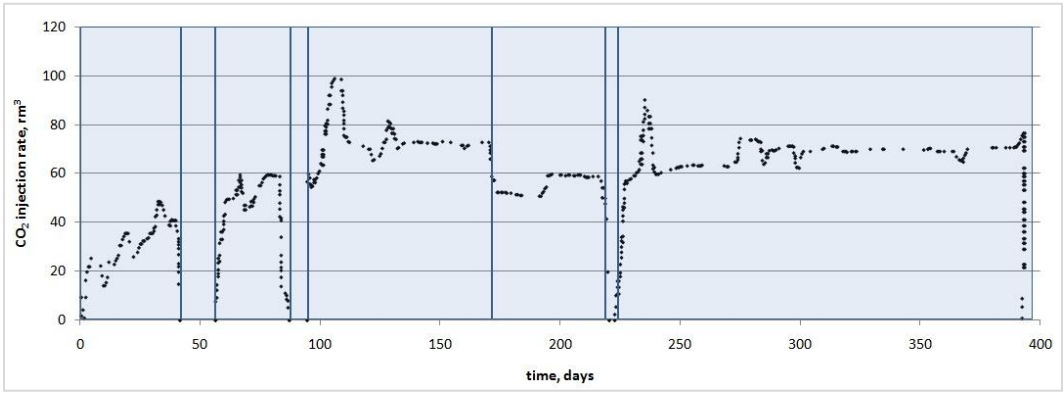


Figure 4-23. CO_2 injection rate (re-produced from Baxter, 1990).
CO_2 injection period is divided into 8 smaller sub-periods.

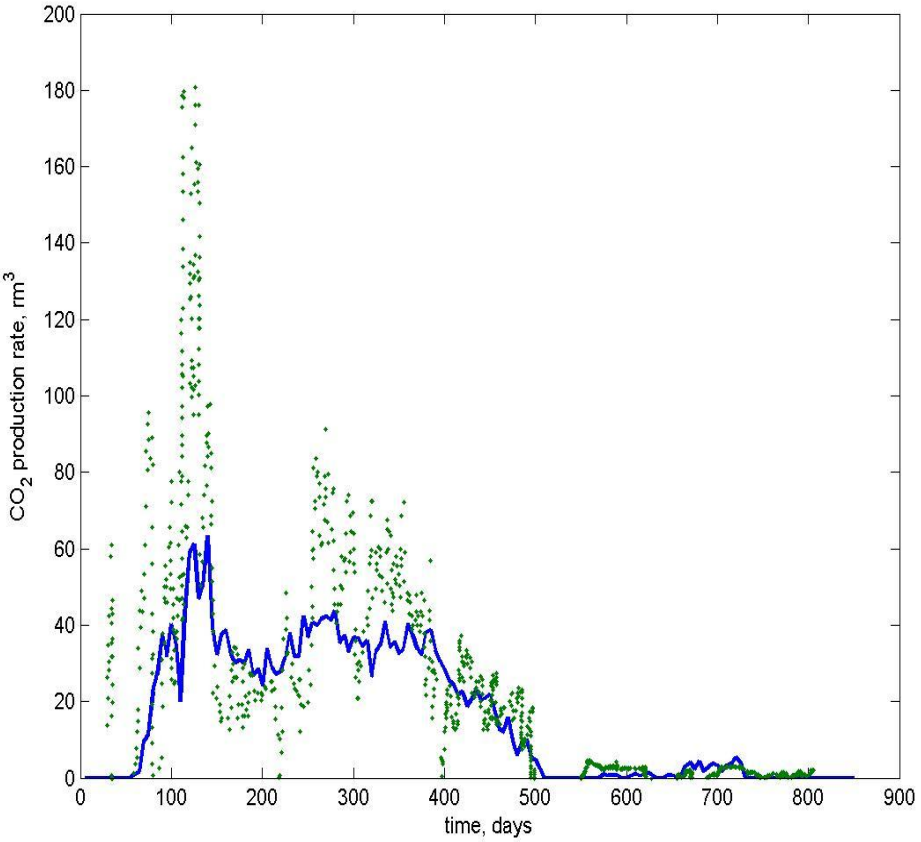


Figure 4-24. Best history match for the model, where CO_2 injection is represented by eight periods of injection at constant rate.
Dots: observed data; solid line: simulated data.

4.9 Bibliography

Baxter, W.A. 1990. Midale CO_2 Flood Pilot History and Results. Oil Development Division Enhanced Oil recovery, *Shell Canada Limited*.

Beliveau, D., Payne, D.A., Mundry, M. 1993. Waterflood and CO_2 Flood of the Fractured Midale Field. *JPT* **45** (9): 881–817.

Delay, F., Ackerer, P. and Danquigny, C. 2005. Simulating Solute Transport in Porous or Fractured Formations Using Random Walk Particle Tracking: a Review. *Vadose Zone Journal* **4**:360–379.

Huo, D. and Gong, B. 2010. Discrete Modeling and Simulation on Potential Leakage through Fractures in CO_2 Sequestration. Paper SPE 135507 presented at the Annual Technical Conference and Exhibition, Florence, Italy, September 19-22. DOI: 10.2118/135507-MS.

Klins, M.A., 1984. Carbon Dioxide Flooding. Basic Mechanisms and Project Design. International Human Resources Development Corporation, Boston, M.A.

Malik, S., Chugh, S. and McKishnie, R.A. 2006. Field-Scale Compositional Simulation of a CO_2 Flood in the Fractured Midale Field. *Journal of Canadian Petroleum Technology* **45**(2):41-50.

Plasynski, S.I. and Damiani, D., 2008. Carbon Sequestration through Enhanced Oil Recovery. Program Facts, U. S. Department of Energy, Office of Fossil Energy, National Energy Technology Laboratory. http://www.netl.doe.gov/publications/factsheets/program/Prog053.pdf.

Ravagnani, G., Ligero, E.L. and Suslick, S.B. 2009. CO_2 Sequestration through Enhanced Oil Recovery in a Mature Oil Field. *Journal of Petroleum Science and Engineering* **65**: 129-138.

Salamon, P., Fernàndez-Garcia, D. and Jaime Gómez-Hernández, J.J. 2006. A Review and Numerical Assessment of the Random Walk Particle Tracking Method. *Journal of Contaminant Hydrology* **87**: 277–305.

Schechter, D.S. 2005. Investigation of Efficiency Improvements during CO_2 Injection in Hydraulically and Naturally Fractured Reservoirs. Semi-Annual Technical Progress Report. Contract No. DE-FC26-01BC15361, US DOE,

http://www.netl.doe.gov/KMD/cds/disk22/G-CO2%20&%20Gas%20Injection/BC15361_11.pdf

Trivedi, J. and Babadagli, T. 2008. Efficiency Analysis of Greenhouse Gas Sequestration during Miscible CO_2 Injection in Fractured Oil Reservoirs. *Env. Sci. and Tech.* **42**(15): 5473-5479.

Trivedi, J. and Babadagli, T. 2009. Oil Recovery and Sequestration Potential of Naturally Fractured Reservoirs during CO_2 Injection. *Energy and Fuels* **23**(8):4025-4036.

5. Scaling for the production curves simulated by RWPT

5.1 Overview

We modeled flow tracer injection through highly fractured media by Random Walk Particle Tracking (RWPT) simulations, where the distance between injection and production wells is equal to *R*. After analyzing the production data obtained, we observed that production curves for different *R* values are similar to each other and can be scaled. Hence, if we define *P(t)* as a probability of a particle to reach the production well within traveling time *t*, the unscaled production profiles *P(t)* vs. *t* will be significantly different for the different values of *R*. However, if we plot $P(t)R^{\beta}$ vs. t/R^{α}, production curves for different values of *R* will overlay. The exponents (α and β) are different for different fracture networks.

In this chapter, we investigated and quantified the dependencies of α and β parameters on fracture network parameters, such as the spacing between fractures (fracture density) and the fractal characteristics of fracture networks.

5.2 Background and problem description

Numerical simulation of fluid flow in porous media is commonly practiced in many different engineering disciplines including oil and gas recovery, groundwater contamination, and waste disposal. These applications involve a detailed description of the underground reservoir and running the simulation on the created detailed model. This can be challenging and requires high computational time.

For practical purposes, it is sometimes important to obtain quick (though not completely accurate) results. In these cases, instead of numerical simulation other techniques are used. Analog fields (Meehan 2011) or analytical modeling (Gontijo and Aziz 1984) - are some of the examples.

Another practical solution for quick (but not necessarily precise) results is to apply scaling rules. The idea behind this approach is that the behaviour of flow parameters (such as recovery, breakthrough time, productivity index, etc.)

A version of this chapter was submitted for publication. Stalgorova, E. and Babadagli, T. 2011. "Scaling of production data obtained from Random Walk Particle Tracking simulations in highly fractured porous media."

depends on the characteristic length of the media (for example, size of the reservoir or the distance between wells). Once this is dependency described, it can be used for predictions.

In Chapter 3 we suggested a non-classical modeling technique to simulate miscible flow in fractured porous media – Random Walk Particle Tracking. Detailed description of the algorithm is provided in Chapter 3 and here we briefly remind the main steps of the algorithm:

(1) Convert discrete fracture network to a classical simulation grid, where fractures are represented by a set of thin highly permeable cells.
(2) Calculate the pressure field using standard simulation tools.
(3) Convert the fracture network to a directed graph; assign the graph edges a set of properties, such as permeability, pressure gradient, flow velocity, etc.
(4) Model the flow of the fluid as movement of a large number of particles (walkers). Particles are released at the production well and move along the graph according to the pressure potential between the nodes. The probability of moving along a particular edge of the graph is proportional to the flow rate along that edge.

In this chapter we simulate the flow of a tracer for one pair of wells, where the distance between the injection and production well is defined as R. The injection flow rate is the same as the production flow rate, and we use the same rate for all simulation cases. We release 10,000 particles at the injection point at $t=0$**,** track each particle until it reaches the production well and record the traveling time. $P(t)$ is the probability of the particle to reach the production well in time t.

Figure 5-1 illustrates the result of five different simulations. All simulations were performed on the same fracture network but distance between the injection and production wells is different for different cases. As seen, the shape (or trend) of all cases are similar even though the size and location are different. One can expect to find a way to scale the traveling time distribution curves. In other words,

A version of this chapter was submitted for publication. Stalgorova, E. and Babadagli, T. 2011. "Scaling of production data obtained from Random Walk Particle Tracking simulations in highly fractured porous media."

we would like to define a simple transformation for each curve so that transformed (scaled) curves for different values of R will overlay.

Once a scaling is described, it can be used to reduce simulation efforts and one can perform simulation on the smaller area and then transform the results to the bigger scale in practice.

The purpose of this chapter is to describe the scaling process, define a scaling relationship for complex fractured systems, and finally to correlate the scaling parameters to the fracture network properties to universalize the scaling relationships proposed.

5.3 Solution methodology

We adapted a methodology that used percolation theory to generate scaling relationships for non-fractured systems (King et al., 1999; Lee et al., 1999).

Let us define t_{mp} as the most probable traveling time, i.e., the time, where the time distribution curve has peaked. Let us also define $P_{mp}=P(t_{mp})$ as the probability of the particle to have traveling time t_{mp}, i.e., the value of the time distribution curve peak.

To find a transformation that overlays all of the time distribution curves, we should first find a transformation that overlays the t_{mp} values for all curves. To do so, we plot t_{mp} vs. R for all curves (**Figure 5-2**). One can see that a logarithmic relationship exists: $t_{mp}\sim R^{\alpha}$, where α=1.8138. This means that if we use t/R^{α} instead of t on the horizontal axes for each curve, the peak locations for the resulting curves will overlay (**Figure 5-3**).

Similarly, plotting P_{mp} vs. R yields another logarithmic relationship: $P_{mp}\sim R^{-\beta}$ where β=1.658 (**Figure 5-4**). Hence, plotting $P(t)R^{\beta}$ instead of $P(t)$ in the y-axis will overlay the peak values for all curves (**Figure 5-5**).

The exercise described above shows that plotting $P(t)R^{\beta}$ vs. t/R^{α} overlays the traveling time distribution curves for different distances between the injecting and production wells. For the fracture network used in this example, α=1.8138 and β=1.658. However, α and β are not unique values and would vary for different fracture networks. In the following section, we investigate how these two scaling

exponents are related to fracture network properties. In this exercise, we will refer to the exponent relating R and t_{mp} as α and to the exponent relating R and P_{mp} as β. This will eventually lead us to define a more universalized scaling relationship for flow in fractured media.

5.4 Fracture network properties and their relation to scaling parameters

The RWPT algorithm summarized above allows simulations for the 3D fracture networks. However, all fracture networks used in the simulations for this chapter, had fractures with the same height. Hence, they can be treated as a 2D object. In other words, a cross section of the fracture network model in the x-y plane can be used as a characteristic 2D network representing the 3D system. This facilitates the quantification of fracture network properties such as density and fractal dimensions.

For the initial part of the study we used a set of fracture networks in which all fractures had equal lengths. Also, the fracture network consisted of two sets of fractures; fractures in one set were oriented NS and fractures in the other set were oriented in the EW direction. Average spacing between the fractures (*sp*) was constant in each particular network, and we changed *sp* to observe how it is correlated the scaling parameters α and β. The angle between the EW direction and the line between the wells is referred to as θ. A representative fracture network and well configuration are shown in **Figure 5-6**.

For each value of *sp*, we created ten different fracture network realizations and for each realization, we ran simulations with ten different values of R to find α and β values (the same way as it is described in the previous section). We repeated the same procedure for two different values of θ. The results are presented in **Figures 5-7a** and 5-**7b**. As seen, the values of the scaling parameters are different for different stochastic realizations; the value of α is scattered around the same value for all values of *sp*, while the value of β tends to increase with increasing *sp*. For practical purposes, we can average α and β over all ten stochastic realizations and use the obtained trends (**Figures 5-8a** and 5-**8b**). As

seen, the exponent α shows no correlation with *sp* but β has a very distinct relationship with similar slope values for different θ values.

Spacing between the fractures is just one of the fracture network properties affecting scaling parameters. Fractal dimensions are the other properties of the fracture network which showed to be a powerful tool for not only fracture network characterization but also for the formulation of hydraulic characteristics of the networks. There are different types of fractal dimensions described in the literature for different characteristics of the fracture networks (La Pointe, 1988; Babadagli, 2001; Jafari and Babadagli, 2009). In the next section, we define three different fractal dimensions to be correlated to the α and β exponents.

In addition, we investigate how scaling parameters depend on the volume fraction of fractures in the system. Volume fraction is referred to as V_f :

V_f = (volume of fractures in the system)/(total volume of the system). (5-1)

5.5 Generation of fractal fracture networks and estimation of fractal dimensions

Using fracture networks, which consists of the fractures of the same lengths is acceptable when there is lack of information, however such kind of networks do not always represent actual fracture network correctly. In reality, naturally fractured reservoir contains fractures on a different length scales - this is a result of fracturing process and the interaction of the stress in the rock with its fluid content (Sahimi and Mehrabi, 1999). There is strong evidence based on field investigation that the actual fracture networks are very irregular and in many cases the fracture network is a fractal object. Here, by calling fracture network a fractal object we mean that the number of fractures of the length *l* is given by:

$$N_l = k \cdot l^{-D_f} \tag{5-2}$$

Here *k* is a constant of proportionality (any constant can be used, as long as we keep it the same for all calculations; we fixed *k*=4000000) and D_f is a fractal dimension of the network and $2.3 \leq D_f \leq 2.7$ (Sahimi, 1993).

We can now express l from equation (5-2):

$$l = \sqrt[D_f]{\frac{k}{N_l}} \tag{5-3}$$

Because we wanted to get a wide distribution of fracture lengths, but didn't want too many fractures, we took N_l=1, 5, 9, ... 41 and for each N_l calculated corresponding l using equation (5-3) and added N_l fractures of the length l to the network. The centers of added fractures were distributed randomly within the model area. We varied D_f in the range $2.3 \leq D_f \leq 2.7$, and for each value of D_f we created ten fracture network realizations. Next, we measured two different fractal dimensions of the networks generated:

(1) Fractal mass dimension (sandbox method)

Fractal mass dimension, D_m was measured by plotting the cumulative length of the fractures contained inside a square box with a side length of x and a fixed center. The log-log plot of the cumulative length vs. x yields a straight line and the fracture network is a fractal object and its mass dimension D_m is given by the slope of the line (Acuna et al., 1992).

(2) Box-counting fractal dimension

In this method, we cover the fracture network by a regular 2D square grid with a given cell size and count how many cells are filled with a fracture network (i.e. which have at least some part of a fracture inside) (Barton and Larson, 1985; Sammis et al., 1987; Acuna and Yortsos, 1991; Barton, 1995; Babadagli, 2001). If plotting the number of filled cells vs. cell size on a log-log scale gives a straight line, the slope of this line is a box-counting fractal dimension (D_{bc}).

Once calculated, α and β were plotted against D_m, D_{bc} and V_f in **Figures 5-9** through **5-11**, respectively. As seen, there is a strong correlation between α and D_m (Figure 5-9a), whereas exponent β does not vary with D_m (Figure 5-9b). On the other hand, D_{bc} (Figure 5-10b) and V_f (Figure 5-11b) showed a correlation with

the exponent β. This procedure can be used to obtain the scaling exponents if one is able to obtain the representative fractal characteristics or density (sp or V_f) of the fracture network.

5.6 Validation exercise

As shown in Figures 5-9 through 5-11, although a clear trend exists between the exponent and the fracture network property (such as Figures 5-9a, 5-10b, and 5-11b), the data is scattered due to the random nature of fracture networks. An approximation of this relationship can help for practical analyses. This can be illustrated by an example: we generated a fracture network, which consists of fractures of the same length, θ = 0.79 and with a spacing value, sp, (between fractures) of 2. The values of the scaling parameters (α and β) for this particular fracture network can be calculated using the relationship given in Figures 5-8a and 5-8b.

α=-0.01sp+1.81=1.79

β=0.19sp+1.17=1.55

The results of plotting traveling time distribution curves with the above values for α and β are presented in **Figure 5-12**. The four curves overlay reasonably well. This indicates that the approach proposed here for fractured systems is useful and simulating just one case at a small scale (with a small distance between the wells) can be scaled up to the reservoir scale for the traveling time distribution.

5.7 Conclusions

We proposed an approach for scaling production profiles (traveling time distribution curves) using the RWPT technique for miscible transport in naturally fractured reservoirs. It was shown that the following scaling rules exist: $P(t)R^{\beta}$ vs. t/R^{α}. To universalize this scaling rule the exponents (α and β) were correlated to four characteristic fracture network properties such as the spacing between fractures (sp), mass (D_m) and box-counting (D_{bc}) fractal dimensions and void fraction (V_f). The relationships between these parameters and the exponents were

tested. The results showed that α was strongly correlated to D_m, and β has strongly dependent on D_{bc}, sp and V_f. In a validation example, we tested the approximated relationship between the sp and the scaling exponents (α and β). The outcome was promising for the practical use of the scaling equation proposed in this chapter.

5.8 Figures

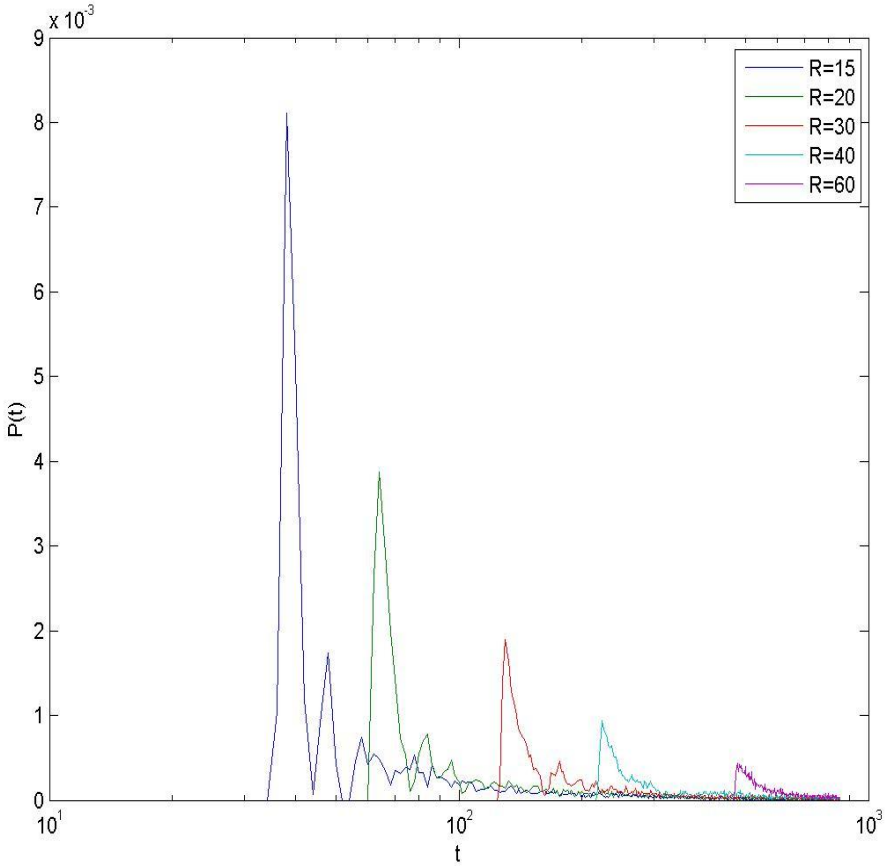


Figure 5-1. Semi-log plot of traveling time distribution *P(t)* for *R*=15, 20, 30, 40 and 60.

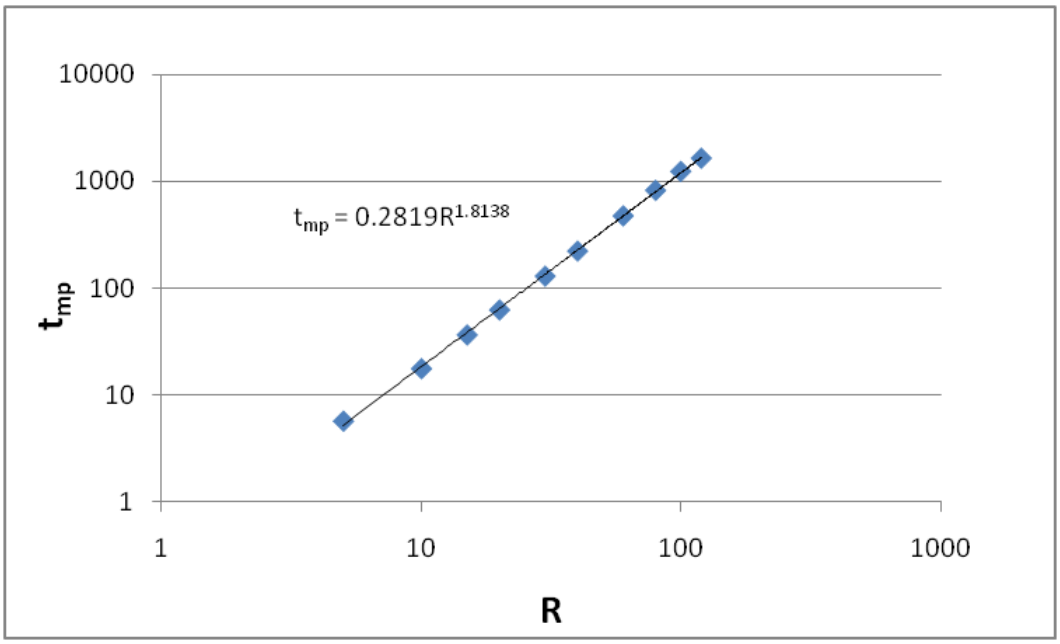


Figure 5-2. Log-log plot of the most probable traveling time vs. *R*.
Slope of the straight line fit is: α=1.8138.

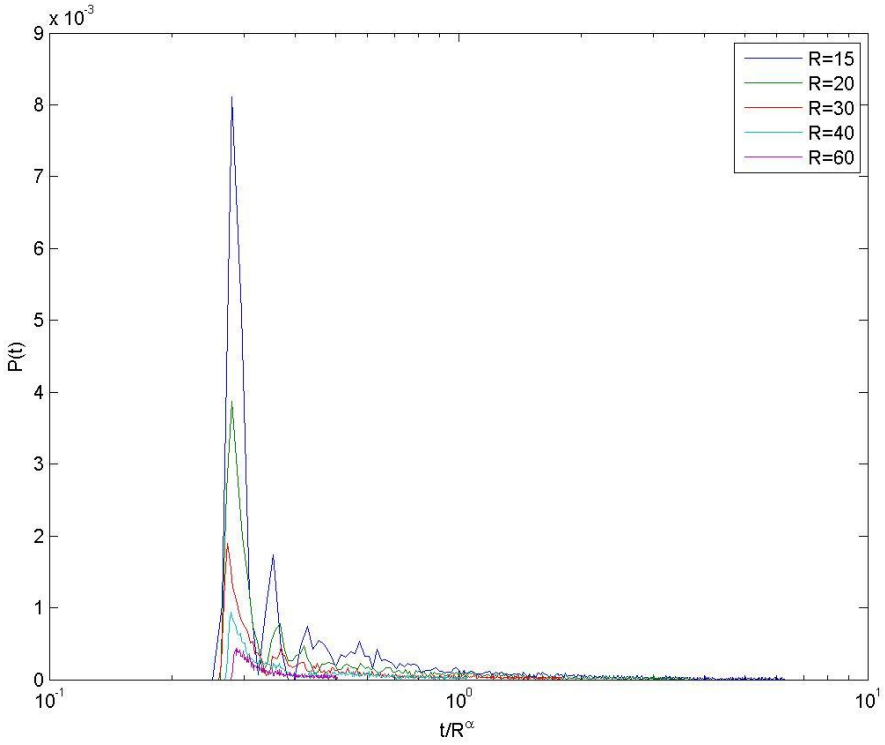


Figure 5-3. Using t/R^{α} on the horizontal axes makes all curves overlay in the horizontal direction.

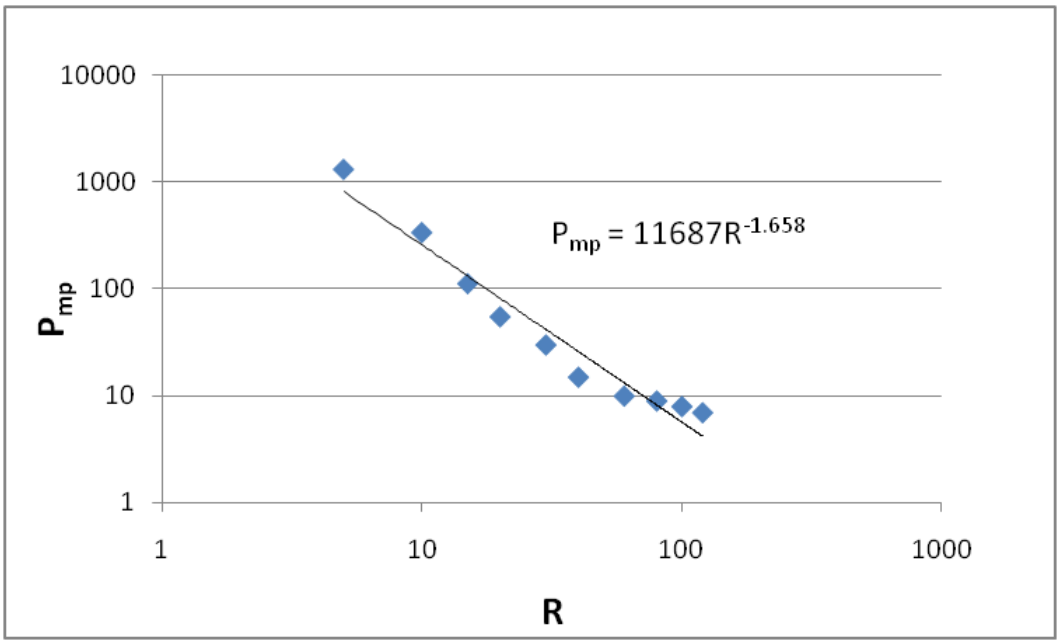


Figure 5-4. Log-log plot of the highest possible probability $P_{mp}=P(t_{mp})$ vs. R.

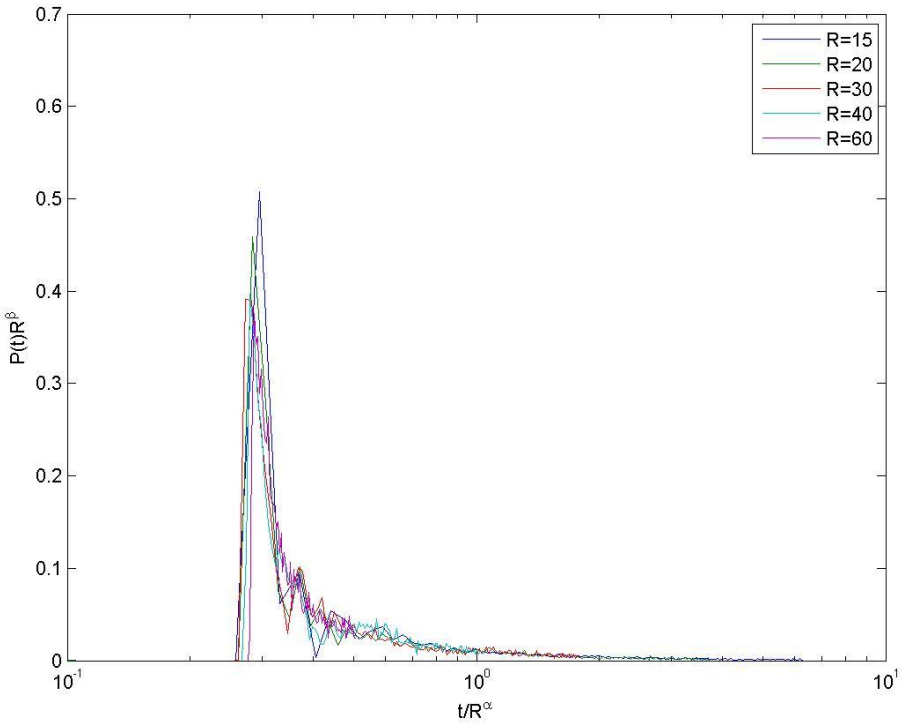


Figure 5-5. Plotting $P(t)R^{\beta}$ vs. t/R^{α} overlays all curves.
α=1.8138; β=1.658 for the fracture network used in this particular example.

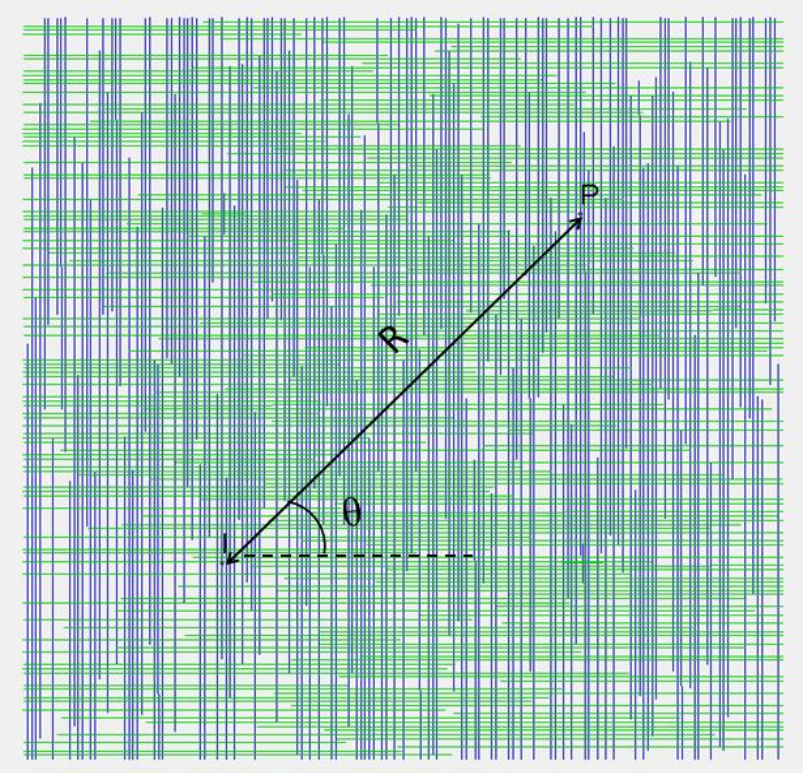


Figure 5-6. Fracture network consisting of two mutually perpendicular fracture sets. Lengths of all fractures are nearly equal.

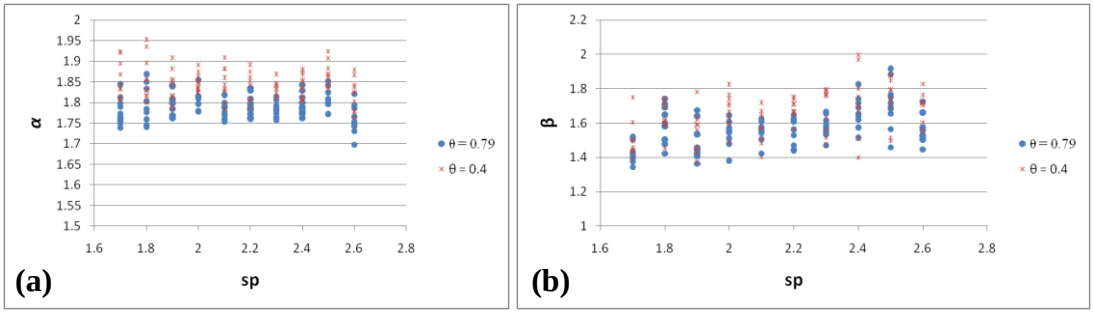


Figure 5-7. *α* and *β* vs. spacing between fractures.
Different points at the same value of *sp* represent different stochastic fracture network realizations. Different colours represent different directions between injection and production wells.

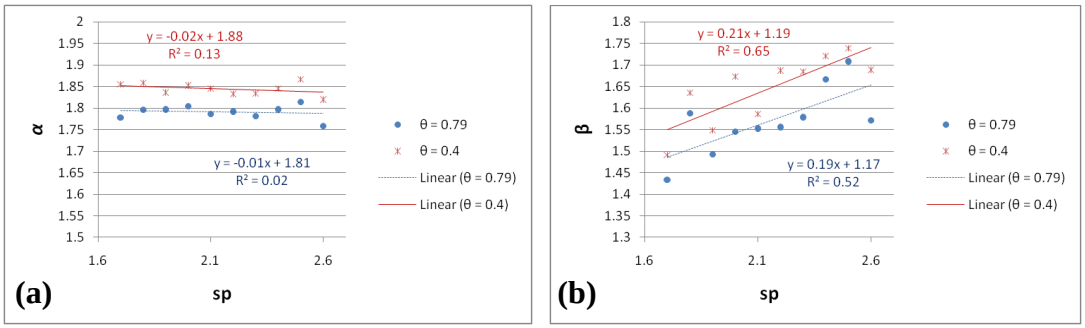


Figure 5-8. Average values for α and β vs. spacing between fractures.
α and β are averaged over ten stochastic realizations for each value of *sp*. Different colours represent different directions between injection and production wells.

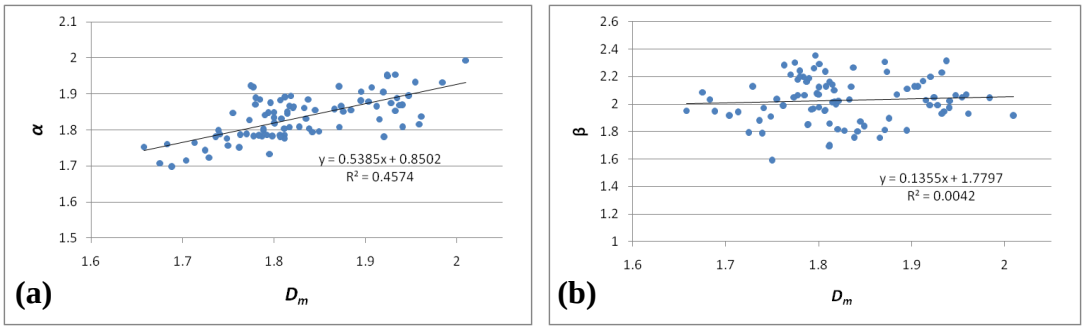


Figure 5-9. α and β vs. fracture network fractal mass dimension.

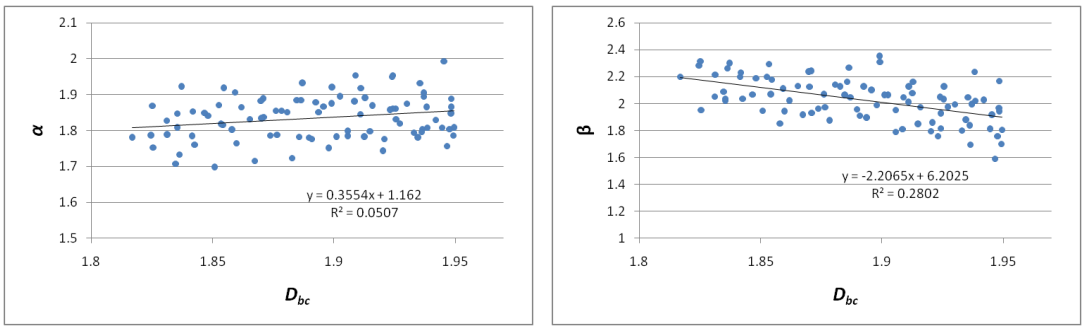


Figure 5-10. α and β vs. fracture network box-counting fractal dimension.

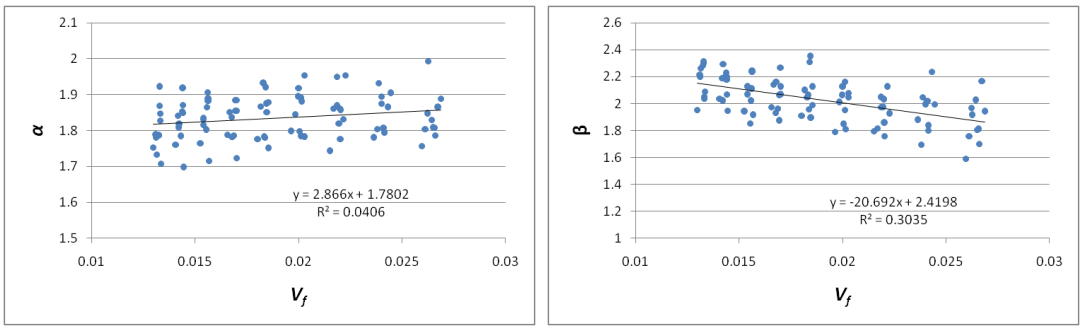


Figure 5-11. α and β vs. volume fraction of fractures in the network.

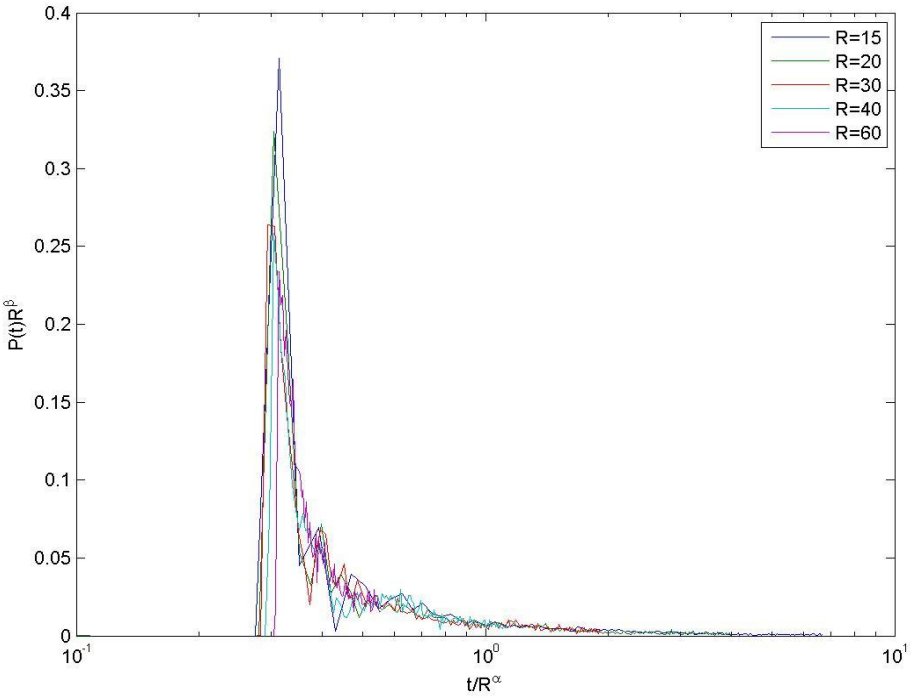


Figure 5-12. Plotting $P(t)R^{\beta}$ vs. t/R^{α} overlays all curves reasonably well even when we use approximate values for scaling parameters. Values of scaling parameters obtained using trends from Figures 8a and 8b are: α=1.79; β=1.55.

5.9 Bibliography

Acuna, J.A., Ershaghi, I. and Yortsos, Y.C. 1992. Fractal Analysis of Pressure Transients in the Geysers Geothermal Field. *Proc.*, Seventeenth Workshop on Geothermal Reservoir Engineering Stanford University, Stanford, CA, 87-93.

Acuna, J.A. and Yortsos, Y.C. 1991. Numerical Construction and Flow Simulation in Networks of Fractures using Fractal Geometry. Paper SPE 22703 presented at the 66th Annual Technical Conference and Exhibition of the Society of Petroleum Engineers, Dallas, Texas, 6-9 October. DOI: 10.2118/22703-MS.

Babadagli, T. 2001. Fractal Analysis of 2-D Fracture Networks of Geothermal Reservoirs in South-Western Turkey. *J. of Volcanology and Geothermal Research* **112**(1-4): 83-103.

Barton, C. C. 1995. Fractal Analysis of Scaling and Spatial Clustering of Fractures. In: Barton, C. C, La Pointe, P. R. (Eds.), *Fractals in the earth sciences*. Plenum Press, New York, pp. 168.

Barton, C.C. and Larson, E. 1985. Fractal Geometry of Two-dimensional Fracture Networks at Yucca Mountain, South-Western Nevada. Proceedings of International Symposium on Fundamentals of Rock Joints, Bjorkliden, Sweden. 77-84.

Beler, R.A. 1990. Pressure Transient Field Data Showing Fractal Reservoir Structure. Paper SPE 21553 presented at the Int. Tech. Meet. Jointly hosted by Petroleum Society of CIM and SPE, Calgary, Alberta, Canada, June 10-13. DOI: 10.2118/21553-MS.

Berkowitz, B., 1995. Analysis of Fracture Network Connectivity Using Percolation Theory. *Mathematical Geology* **27**(4): 467-483.

Berkowitz, B. and Ewing, R.P. 1998. Percolation Theory and Network Modeling Applications in Soil Physics. *Surveys in Geophysics* **19:** 23-72.

Bogdanov, I.I., Mourzenkov, V.V., Thovert, J.-F. and Adler, P.M. 2003. Effective Permeability of Fractured Porous Media in Steady State Flow. *Water Res. Res*. **39**(1), 1023.

Chang, J. and Yortsos, Y.C. 1990. Pressure-Transient Analysis of Fractal Reservoirs. *SPE Formation Evaluation*, **5**(1):31-38. SPE-18170-PA. DOI: 10.2118/18170-PA.

Gontijo, J.E. and Aziz, K. 1984. A Simple Analytical Model for Simulating Heavy Oil Recovery by Cyclic Steam in Pressure-Depleted Reservoirs. Paper SPE 13037 presented at the 59th Annual Conference and Exhibition, Houston, Texas, September 16-19. DOI: 10.2118/13037-MS.

Halvin, S. and Ben-Avraham, D. 1987. Diffusion in Disordered Media. *Advances in Physics* **36**(6):695-798.

Jafari, A. and Babadagli, T. 2009. A Sensitivity Analysis for Effective Parameters on 2D Fracture-Network Permeability. *SPE Res Eval & Eng* **12** (3): 455-469. SPE-113618-PA. DOI: 10.2118/113618-PA.

King, P.R., Andrade, J.S., Buldyrev, V., Dokholyan, Lee. Y., Havlin, S. and Stanley, H.E. 1999. Predicting Oil Recovery Using Percolation. *Physica A* **266**(1999):107-114.

La Pointe, P. R. 1988. A Method to Characterize Fracture Density and Connectivity through Fractal Geometry. *Int. J. Rock Mech. Min. Sci. Geomech. Abstr.* **25**(6)**:** 421-429.

Lee, Y., Andrade, J.S., Buldyrev, V., Dokholyan, N.V., Havlin, S., King, P.R., Paul, G. and Stanley, H.E. 1999. Traveling Time and Traveling Length in Critical Percolation Clusters. *Physical Review E* **60**(3):3425-3428.

Masihi, M, King, P.R. and Nurafza, P. 2007. Fast Estimation of Connectivity in Fractured Reservoirs using Percolation Theory. *SPE J* **12**(2):167-178. SPE 94186-PA. DOI: 10.2118/94186-PA.

Meehan, D.N. 2011. Using Analog Reservoir Performance to Understand Type I Fractured Reservoir Behaviour with Strong Water Drives. Paper SPE 144177 presented at the SPE Enhanced Oil Recovery Conference, Kuala Lumpur, Malaysia, 19-21 July. DOI: 10.2118/144177-MS.

Mourzenkov, V.V., Thovert, J.F. and Adler, P.M. 2005. Percolation and Permeability of Three Dimensional Fracture Networks with a Power Law Size Distribution. *Fractals in Engineering*, Springer London.

Sahimi, M. 1993. Flow Phenomena in Rocks: from Continuum Models to Fractals, Percolation, Cellular Automata, and Simulated Annealing. *Reviews of Modern Physics* **65**(1993):1393-1537.

Sahimi, M. and Mehrabi, A.R. 1999. Percolation and Flow in Geological Formations: Upscaling From Microscopic to Megascopic Scales. *Physica A* **266**(1999):136-152.

Sammis, C., King, G. and Biegel., R. 1987. The Kinematics of Gouge Deformation. *PAGEOPH* **125**(5):777-812.

6. Contributions and recommendations

As this is a paper-based thesis, conclusions were provided at the end of each chapter. Here, we present the major contributions and recommendations for future work.

6.1 Major contributions

(1) We adapted a non-classical algorithm to simulate miscible solvent injection in fractured media. The suggested algorithm (Random Walk, RW) is capable of simulating miscible flow for 2D lab scale models, including horizontal and vertical flow direction. The RW algorithm requires less computational time than classical finite-difference modeling. Additionally, it can capture randomness involved in the process, which is critical in miscible displacement modeling in fractured systems.

(2) The results of simulation reproduced experimental results obtained from the literature reasonably well, especially for the cases of horizontal flow. A relationship between the viscosity of the displaced fluid and the diffusivity coefficient was proposed for the horizontal case. The accuracy of the model was improved compared to the earlier attempts on classical modeling and computational time was significantly reduced compared to fine grid continuum modeling.

(3) The algorithm introduced has only two unknown matching parameters (diffusivity coefficients of oil and solvent), which results in easier history matching process and reduces the uncertainty in the performance prediction. This number was six for single porosity models.

(4) In continuation of the lab scale modeling, another non-classical simulation algorithm (Random Walk Particle Tracking) was suggested for the field scale simulations of the flow in fractured media. This

A version of this chapter was submitted for publication. Stalgorova, E. and Babadagli, T. 2011. "Modified Random Walk - Particle Tracking model for CO_2 flooding/sequestration in naturally fractured oil reservoirs."

algorithm uses discrete description of the fracture network, without averaging its property. This allows for capturing reservoir irregularity and connectivity.

(5) The Random Walk Particle Tracking (RWPT) algorithm was tested by modeling a series of tracer tests. Although an exact history match was not achieved, it was observed that the created model represents the reservoir connectivity reasonably well. We also performed a sensitivity analysis of the RWPT model to various fracture network characteristics.

(6) We attempted to modify RWPT algorithm to model CO_2 flooding and sequestration. Modeling of the flow through matrix and the decrease in mobility due to phase interaction were incorporated in the model. History matching and sensitivity analysis exercises were performed to test the suggested model.

(7) We suggested a new scaling rule for the production curves obtained by the RWPT simulations and described how scaling parameters depend on the fracture network characteristics. The two exponents in the scaling relationship were correlated to the fracture network properties such as fracture density, volume fraction of fracture, mass and box-counting fractal dimensions.

6.2 Recommendations for future work

(1) The RW algorithm can be extended for a 3D case as there are no principal limitations to do so. However, dealing with a large number of grid cells and large number of walkers may require optimization in data handling and calculations.

(2) The RW algorithm showed better results for the cases of horizontal flow than for the vertical flow, which may be due to the incomplete description of the gravity in the model. This is an interesting research subject to look into.

A version of this chapter was submitted for publication. Stalgorova, E. and Babadagli, T. 2011. "Modified Random Walk - Particle Tracking model for CO_2 flooding/sequestration in naturally fractured oil reservoirs."

(3) In this research, we used uniform matrix and fracture permeability. However, the RW algorithm allows using non-uniform permeability. Testing the effect of non-uniform permeability cases (i.e. modeling microheterogeneities which are presented in any glass bead sample) may be an interesting study.

(4) The RWPT modeling is currently limited to modeling only about 500 fractures. We believe that this limitation can be overcome by using proper ways of data treatment (i.e. using pointers instead of arrays etc) and solving the finite-difference equation for pressure in the C++ code instead of using the Eclipse software.

(5) The RWPT model for CO_2 flooding does not capture several important physical aspects of the CO_2 flooding process, such as the proper description of the diffusion into matrix, the difference between oil and CO_2 densities, and the change in oil composition and limited storage capacity of the media. A description of CO_2 flooding in the RWPT model requires further improvements to capture the physics of the process in all its complexity.

(6) The scaling rule, suggested in Chapter 5 uses scaling parameters, which strongly depend on the fracture network characteristics. We investigated the effect of a few of them. However, there are more characteristics, which can be used to describe a fracture network (such as fracture lengths and widths distributions, apertures, permeabilities). An investigation of the effects of other fracture network characteristics and, possibly, deriving a more universal equation for scaling parameters would be an elaborate work.

Appendix A (C++ code for the RW algorithm)

Header file (main.h)

```
#define M 15 // size of the grid in i direction
#define N  40 // size of the grid in j direction
#define Mid  8// Mid = [M/2]+1 - fracture location
#define m 4 // number of walkers along each grid cell in i direction
#define n 4 // number of walkers along each grid cell in j direction
#define g 0 // cm/s2 - gravity  (0 - for horizontal flow; 981.2 - for vertical)

int NumTimeSteps; // number of Time Steps
int q;  // number of walkers we add at each timestep
int freq; //we add walkers once in freq steps (time_s=1, freq+1, 2freq+1, ...)
double Q; //injection rate, cm3/s - GIVEN
double PV; //pore volume, cm3 - GIVEN
double Perm_f;  // Perm fracture, m^2
double Perm_m;  //Perm matrix, m^2 - GIVEN
double mu1; // viscosity of oleic phase (kerosene), Pa*s - GIVEN
double mu2; //viscosity of solvent (pentane), Pa*s - GIVEN
double rho1; // density of oleic phase (kerosene), kg/cm3 - GIVEN
double rho2; //viscosity of solvent (pentane), kg/cm3 - GIVEN
double dt; // length of timestep, seconds
double dx; // x-size of the model, cm
double dy; // y-size of the model, cm
double dz; // z-size of the model, cm
double Pin; // pressure at injection point, Pa - approximated through Darcys Law with effective
permeability 1000 D
double w; // parameter for blending rule
double D, Do, Ds; //diffusivity coefficient

class Walker{
public:
        int i,j,I,J,fluid;  // I=[x*M/dx]+1; J=[y*N/dy]+1; I=[i/m]+1; J=[j/n]+1;
        double x,y, Vx, Vy;
        double temp, Vy12f, Vy34f; //for test
        double t1,t2,t3,t4;
        void ConvectiveStep();
        void RandomStep();
        void CalculateV();
        void VforCell(int, int, int, int, int, int, int, int);
        Walker(){
                x=0;y=0;I=1;J=1;i=1;j=1;
                temp=0; Vy12f=0;Vy34f=0;  //for test
                t1=0;t2=0;t3=0;t4=0; //for test
                fluid=1;
                }
        };

// coarse arrays - all these arrays have boundary cells, which are 'dummy', when i=0 or i=M+1 or
j=0 or j=N+1
```

```
double mU[M+1+1][N+1+1],mD[M+1+1][N+1+1],mL[M+1+1][N+1+1],mR[M+1+1][N+1+1];
//mobilities
double DX[M+2], DY[M+2]; //size of coarse cell, cm
double TLCx[M+1+1][N+1+1],TLCy[M+1+1][N+1+1]; // coordinates of Top Left corner of (I,J)
coarse cell
double rhoL[M+1+1][N+1+1],rhoR[M+1+1][N+1+1]; //densities
double perm[M+1+1][N+1+1],p[M+1+1][N+1+1];
// matrixes for equations system solution
double A[M*N][M*N], A_I[M*N][M*N], T[M*N][M*N+1];
double b[M*N], x[M*N]; // summ[M*N];
// fine arrays
int NumWalkersO[M*m][N*n],NumWalkersS[M*m][N*n], NumOilC[M+1+1][N+1+1],
NumSolvC[M+1+1][N+1+1];
double NumWalkersSd[M*m][N*n]; // for test
double ConcC[M+1+1][N+1+1],ViscMixC[M+1+1][N+1+1],DensMixC[M+1+1][N+1+1];
int ToProduce[100]; // 3*q should be enough
int OilProducedTotal;
double TS[7];
double ProdZone;

// walkers Array
Walker walkers[10*M*m*N*n];
int RealSize;
int time_s;

// for system solution
void SetT();
void Solve();
void T_div(int, double); // line number, divider
void T_min(int, int, double); // (line int2):=(line int2) - (line int1)*double

// for walking
void SetTimeSteps();
void SetParameters();
void SetGrid();
void SetPerms(); //set permeabilities for the grid
void PopulateWalkers();
void Inject();
void Produce();
void ProduceAll();
void UpdateP();
void UpdateMs();
void Setb();
void MakeStep();
void UpdateA();
void UpdateNumWalkers();
void CalcPropCoarse();
void Run();
double Norm(double, double); //gives normally distributed random variable

//for output
void WalkersToFile();
void WalkersToFileM();
void ToFile(double *, int, int, char);
```

```
void ParametersToFile();
```

Source file (main.cpp)

```
#include "main.h"
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <fstream>
#include <iostream>
#include <string>
#include <cmath>
#include <cstdio>
#include <time.h>
#include<conio.h>
#include <sstream>
using namespace std;

int main()
{SetParameters();
SetGrid();
Run();
ParametersToFile();
}
void SetParameters()
{q=8; // number of walkers we add at each timestep
freq=1; // we add walkers once in freq steps (time_s=1, freq+1, 2freq+1, ...)
Q=(double)15/3600; // injection rate, cm3/s - GIVEN
Perm_f=15000; //Perm fracture, D
Perm_m=150;  //Perm matrix, D - GIVEN
mu1=33.5; // viscosity of oleic phase (kerosene), cP - GIVEN (2.9|33.5|500)
mu2=0.38; //viscosity of solvent (pentane), cP - GIVEN
rho1=0.00081;// density of oleic phase (kerosene), kg/cm3 - GIVEN (0.00079|0.00081|0.00089)
rho2=0.00063; //density of solvent (pentane), kg/cm3 - GIVEN
dx=10; // x-size of the model, cm
dy=15;// y-size of the model, cm
dz=0.17; // z-size of the model, cm
w=0.25; // parameter for blending rule
PV = 12; //Pore volume in cm3 - more or less GIVEN
dt=(double)(q*PV)/(M*N*m*n*Q*freq); //in seconds! if Q cm3/h is represented by q/dt walkers
ProdZone=(double)q/(m*n)*(dy/N); // defines area in the 'out' cell from where walkers are taken
out for production
time_s=0; //current time
RealSize=0; //current number of walkers in the system
D=-999;// diffusivity coefficient, cm2/s
Do=0.0025;
Ds=0.004;
NumTimeSteps=18000;//how many timesteps to calculate
}
void SetGrid()  //fills DX, DY, TLCx and TLCy arrays
{
        for (int I=0;I<M+2; I++)
                DX[I]=(double)dx/M;
        for (int J=0;J<N+2; J++)
        {DY[J]=(double)dy/N;}
```

```
	for (int I=1;I<=M;I++)
	{		for (int J=1;J<=N;J++)
	{		 TLCx[I][J]=0;
	TLCy[I][J]=0;
	for (int Is=1;Is<I;Is++)
		TLCx[I][J]=TLCx[I][J]+DX[Is];
	for (int Js=1;Js<J;Js++)
		TLCy[I][J]=TLCy[I][J]+DY[Js];
	}}  //now (TLCx[I][J], TLCy[I][J]) - coordinates of Top Left corner of (I,J) cell.
}
void Run(){
	fopen ("production.txt", "w");
	OilProducedTotal=0;
	time_s=0;
	SetTimeSteps();
	Setb();
	SetPerms();
	//ToFile(&perm[0][0], M+2, N+2, 'K'); //any array can be exported
	PopulateWalkers();
	for (int counter=0; counter<=NumTimeSteps;counter++)
	{
		MakeStep();
		if (time_s==TS[1] || time_s==TS[2] || time_s==TS[3] || time_s==TS[4] ||
time_s==TS[5] || time_s==TS[6])
		{
			WalkersToFileM();  //output walkers positions only for the times for
which we have images
		}
	}
}
void SetTimeSteps() //sets at how many timesteps we need to output walkers for comparing with
experimental images
{
	if(g==0)
	{
		if(mu1==2.9 && Q==(double)15/3600)  // Kerosene 15
		{TS[1]=0.05; TS[2]=0.1; TS[3]=0.2; TS[4]=0.5; TS[5]=0.8; TS[6]=1.0; }
		if(mu1==33.5 && Q==(double)15/3600)  // LMO 15
		{TS[1]=0.1; TS[2]=0.2; TS[3]=0.5; TS[4]=0.8; TS[5]=1.0; TS[6]=1.5; }
		if(mu1==33.5 && Q==(double)25/3600)  // LMO 25
		{TS[1]=0.05; TS[2]=0.2; TS[3]=0.5; TS[4]=0.8; TS[5]=1.0; TS[6]=1.5; }
		if(mu1==33.5 && Q==(double)45/3600)  // LMO 45
		{TS[1]=0.1; TS[2]=0.5; TS[3]=0.8; TS[4]=1.0; TS[5]=1.5; TS[6]=2.0; }
		if(mu1==500 && Q==(double)15/3600)  // HMO 15
		{TS[1]=0.1; TS[2]=0.2; TS[3]=0.5; TS[4]=1.0; TS[5]=2.0; TS[6]=3.0; }
		if(mu1==500 && Q==(double)45/3600)  // HMO 45
		{TS[1]=0.5; TS[2]=1.0; TS[3]=2.0; TS[4]=3.0; TS[5]=4.0; TS[6]=5.0; }
	}
	if(g>0)
	{
		if(mu1==33.5 && Q==(double)15/3600)  // LMO 15
		{TS[1]=0.1;		TS[2]=0.2; TS[3]=0.5; TS[4]=0.8; TS[5]=1.0; TS[6]=1.5; }
		if(mu1==33.5 && Q==(double)45/3600)  // LMO 45
```

```
                    {TS[1]=0.31;      TS[2]=0.63; TS[3]=1.25; TS[4]=2.50; TS[5]=2.7; TS[6]=2.9;
                    if(mu1==500 && Q==(double)15/3600)  // HMO 15
                    {TS[1]=0.1;       TS[2]=0.52; TS[3]=1.25; TS[4]=2.5; TS[5]=2.7; TS[6]=2.9; }
                    if(mu1==500 && Q==(double)45/3600)  // HMO 45
                    {TS[1]=0.31;      TS[2]=1.88; TS[3]=3.75; TS[4]=4.69; TS[5]=4.9; TS[6]=5.1;
}
          }
          for (int i=1; i<=6;i++)
          {TS[i]=(int)(TS[i]*M*N*m*n*freq/(q));}
          string strT="timesteps.txt";
          ofstream output(strT.c_str());
          output.flush();
          output<<TS[1]<<' '<<TS[2]<<' '<<TS[3]<<' '<<TS[4]<<' '<<TS[5]<<' '<<TS[6];
          output.close();
}
void Setb(){ //set right part for the system Ax=b
          // right part for cell [ij] - b[(i-1)+(j-1)*M]
          for (int i=1;i<=M;i++){
                    for(int j=1;j<=N;j++)
                    {
                    b[(i-1)+(j-1)*M]=(double)100*(rhoR[i][j]*mR[i][j]-
rhoL[i][j]*mL[i][j])*g*DX[i];
                    }
          }
          b[Mid-1+(1-1)*M]=b[Mid-1+(1-1)*M]+(double)100000*Q/(dz*0.4);
          b[Mid-1+(N-1)*M]=0; // p[Mid][N]=Pout=0
}
void SetPerms(){
          for (int I=0;I<=M+1;I++)
                    for(int J=0;J<=N+1; J++)
                              perm[I][J]=Perm_m;
          for (int J=1;J<=N; J++)
                    perm[Mid][J]=Perm_f;
}
void PopulateWalkers() //uniformly distribute m*n walkers in each grid cell
{int s=1;
for (int I=1;I<=M;I++)
{
          for (int J=1;J<=N;J++)
          {         for (int is=0;is<m;is++)
          {for (int js=0;js<n;js++)
          {
                    walkers[s].i=m*(I-1)+is;
                    walkers[s].j=n*(J-1)+js;
                    walkers[s].I=I;
                    walkers[s].J=J;
                    walkers[s].x=TLCx[I][J]+DX[I]/(m*2)+is*DX[I]/m;
                    walkers[s].y=TLCy[I][J]+DY[J]/(n*2)+js*DY[J]/n;
                    walkers[s].fluid=1;
                    s++;
          }
          }
          }
}
```

```
RealSize=s-1;
}
void MakeStep()
{ time_s++;
Inject();
UpdateNumWalkers();
UpdateMs();
Setb();
UpdateA(); //updates matrix for Darcy's eq-n system
SetT(); //T=A|b
Solve();
UpdateP();
CalcPropCoarse();
for (int s=1; s<=RealSize;s++)
{if (walkers[s].fluid>0)  // Convective Step for all non-removed walkers
{walkers[s].CalculateV();
walkers[s].ConvectiveStep();}}
CalcPropCoarse();
for (int s=1; s<=RealSize;s++)
{if (walkers[s].fluid>0)  // Random step for all non-removed walkers
walkers[s].RandomStep();
}
ProduceAll(); // this also includes printing production to file
printf(" %5.5d", time_s);
}
void Inject(){ //locate q walkkers in the 'in' cell
        for (int s=1;s<=q;s++)
        {       double r=(double)rand()/RAND_MAX; // r in [0...1)
        RealSize++;
        walkers[RealSize].x=TLCx[Mid][1]+DX[Mid]*r;
        walkers[RealSize].y=DY[1]/(2*n);
        walkers[RealSize].I=Mid;
        walkers[RealSize].J=1;
        walkers[RealSize].i=(Mid-1)*m+(int)(m*r);
        walkers[RealSize].j=0;
        walkers[RealSize].fluid=2;
        }}
void UpdateNumWalkers(){
        for (int i=0;i<M*m;i++)
        {for(int j=0;j<N*n;j++)
        {NumWalkersO[i][j]=0;
        NumWalkersS[i][j]=0;
        }}
        for (int s=1;s<=RealSize;s++)
        {if (walkers[s].fluid==1) {NumWalkersO[walkers[s].i][walkers[s].j]++;}
        if (walkers[s].fluid==2) {NumWalkersS[walkers[s].i][walkers[s].j]++;}
        }
}
void UpdateMs(){ //calculates mobilities in grid cells
        double xo, xs;  //concentrations
        for (int I=1;I<=M;I++){
                for (int J=1;J<=N;J++)
                { //mU
                        if(I==1) mU[I][J]=0;
```

```
		else{
		int LocNumO=0;
		int LocNumS=0;
		for (int i= (I-1)*m-int(m/2);i<I*m-int(m/2);i++){
			for (int j= (J-1)*n;j<J*n;j++)  //area 'up' from center of I,J cell
			{LocNumO=LocNumO+NumWalkersO[i][j];
			LocNumS=LocNumS+NumWalkersS[i][j];}
		}  // now LocNum - are total number of walkers in the 'up' area
if ((LocNumO+LocNumS)==0 || (LocNumS<5 && LocNumO==0)) xo=0.8;
else xo=(double)LocNumO/(LocNumO+LocNumS); // to avoid dividing by zero
			xs=1-xo;
mU[I][J]=pow(xo*pow(mu1,w)+xs*pow(mu2,w),(1/w)); // viscosity_av
mU[I][J]=(2/(1/perm[I][J]+1/perm[I-1][J]))/mU[I][J];
		}
		//mD
		if(I==M) mD[I][J]=0;
		else{
			int LocNumO=0;
			int LocNumS=0;
			for (int i= (I-1)*m+int(m/2);i<I*m+int(m/2);i++){
			for (int j= (J-1)*n;j<J*n;j++)  //area 'down' from I,J cell center
				{LocNumO=LocNumO+NumWalkersO[i][j];
				LocNumS=LocNumS+NumWalkersS[i][j];}
			}// LocNum - are total number of walkers in the 'down' area
if ((LocNumO+LocNumS)==0 || (LocNumS<5 && LocNumO==0)) xo=0.8;
else xo=(double)LocNumO/(LocNumO+LocNumS); // to avoid dividing by zero
		xs=1-xo;
mD[I][J]=pow(xo*pow(mu1,w)+xs*pow(mu2,w),(1/w)); // viscosity_av
mD[I][J]=(2/(1/perm[I][J]+1/perm[I+1][J]))/mD[I][J]; // perm_av/viscosity_av
		}
		//mL
		if(J==1)
		{mL[I][J]=0;
		rhoL[I][J]=0;}
		else{
			int LocNumO=0;
			int LocNumS=0;
			for (int i= (I-1)*m;i<I*m;i++){
		for (int j= (J-1)*n-int(n/2);j<J*n-int(n/2);j++)  //area 'left' from I,J cell
				{LocNumO=LocNumO+NumWalkersO[i][j];
			LocNumS=LocNumS+NumWalkersS[i][j];}
		}  // now LocNum - are total number of walkers in the 'left' area
if ((LocNumO+LocNumS)==0 || (LocNumS<5 && LocNumO==0)) xo=0.8;
else xo=(double)LocNumO/(LocNumO+LocNumS); // to avoid dividing by zero
xs=1-xo;
mL[I][J]=pow(xo*pow(mu1,w)+xs*pow(mu2,w),(1/w)); // viscosity_av
mL[I][J]=(2/(1/perm[I][J]+1/perm[I][J-1]))/mL[I][J]; //perm_av/viscosity_av
			rhoL[I][J]=xo*rho1+xs*rho2;
		}
		//mR
		if(J==N)
		{mR[I][J]=0;
		rhoR[I][J]=0;}
		else{
```

```
                                int LocNumO=0;
                                int LocNumS=0;
                                for (int i= (I-1)*m;i<I*m;i++){
                        for (int j= (J-1)*n+int(n/2);j<J*n+int(n/2);j++)  //area 'right'
                                        {LocNumO=LocNumO+NumWalkersO[i][j];
                                        LocNumS=LocNumS+NumWalkersS[i][j];}
                                }  // now LocNum - are total number of walkers in the 'right' area
                        if ((LocNumO+LocNumS)==0 || (LocNumS<5 && LocNumO==0)) xo=0.8;
                        else xo=(double)LocNumO/(LocNumO+LocNumS); // to avoid dividing by zero
                                xs=1-xo;
                        mR[I][J]=pow(xo*pow(mu1,w)+xs*pow(mu2,w),(1/w)); // viscosity_av
                        mR[I][J]=(2/(1/perm[I][J]+1/perm[I][J+1]))/mR[I][J]; //perm_av/viscosity_av
                                rhoR[I][J]=xo*rho1+xs*rho2;
                                }
                        } } //ind of ij loops
}
void UpdateA(){
        // eqn for cell [ij] - A[(i-1)+(j-1)*M][*]
        // coeficient at p_ij - A[*][(i-1)+(j-1)*M]
        for (int i=1;i<=M;i++){
                for(int j=1;j<=N;j++)
                {
                        A[(i-1)+(j-1)*M][(i-1)+(j-1)*M]=mU[i][j]*2*DY[j]/(DX[i]+DX[i-
1])+mD[i][j]*2*DY[j]/(DX[i]+DX[i+1])+mR[i][j]*2*DX[i]/(DY[j]+DY[j-
1])+mL[i][j]*2*DX[i]/(DY[j]+DY[j-1]);
                        if (j<N) A[(i-1)+(j-1)*M][(i-1)+((j+1)-1)*M]=(-
1)*mR[i][j]*2*DX[i]/(DY[j]+DY[j+1]);
                        if (j>1) A[(i-1)+(j-1)*M][(i-1)+((j-1)-1)*M]=(-
1)*mL[i][j]*2*DX[i]/(DY[j]+DY[j-1]);
                        if (i<M) A[(i-1)+(j-1)*M][(i+1-1)+(j-1)*M]=(-
1)*mD[i][j]*2*DY[j]/(DX[i]+DX[i+1]);
                        if (i>1) A[(i-1)+(j-1)*M][(i-1-1)+(j-1)*M]=(-
1)*mU[i][j]*2*DY[j]/(DX[i]+DX[i-1]); //if - because we wont to avoid calling A[-1][*] - even if
we are going to multiply by 0.
                }} //end of ij loop
        for (int k=0; k<M*N;k++)
        {A[Mid-1+(N-1)*M][k]=0;}
        A[Mid-1+(N-1)*M][Mid-1+(N-1)*M]=1;  // p[Mid][N]=Pout=0
}
void UpdateP(){  //take pressure values from system solution and locate them in p[][]
        for (int I = 0; I<=M+1;I++)
                for (int J = 0; J<=N+1;J++) p[I][J]=-999;

        for (int I = 1; I<=M;I++)
                for (int J = 1; J<=N;J++)
                {
                        p[I][J]=0;
                        for (int k=0; k<M*N; k++)
                                p[I][J]=x[(I-1)+(J-1)*M];
                }
}
void CalcPropCoarse(){
        for (int I=0;I<=M+1;I++)
        {for(int J=0;J<=N+1;J++)
```

```
        {NumSolvC[I][J]=0;
        NumOilC[I][J]=0;
        }}
        for (int s=1;s<=RealSize;s++)
        {if (walkers[s].fluid==1) {NumOilC[walkers[s].I][walkers[s].J]++;}
        if (walkers[s].fluid==2) {NumSolvC[walkers[s].I][walkers[s].J]++;}
        }
        for (int I=0;I<=M+1;I++)
        {for(int J=0;J<=N+1;J++)
        {
                if (NumOilC[I][J]==0 && NumSolvC[I][J]==0)
                        ConcC[I][J]=0;
                else

        ConcC[I][J]=(double)NumSolvC[I][J]/(NumSolvC[I][J]+NumOilC[I][J]);
                ViscMixC[I][J]=pow((1-
ConcC[I][J])*pow(mu1,w)+ConcC[I][J]*pow(mu2,w),(1/w));
                DensMixC[I][J]=(1-ConcC[I][J])*rho1+ConcC[I][J]*rho2;
        }
        }}
void Walker::ConvectiveStep(){
        x=x+Vx*dt;
        y=y+Vy*dt;

        if (x<0) x=(-1)*x;
        if (y<0) y=(-1)*y;
        if( dx-x<0) x=2*dx-x; // particle reflects from boundary;
        if( dy-y<0)
        {if (I=Mid)
        {y=dy;}
        else
        {y=2*dy-y;}
        }   // near to outlet particle 'sticks' to the exit, not reflected
        //
        for (int IS=1;IS<M;IS++)
        {if (x>=TLCx[IS][1] && x < TLCx[IS+1][1])
        I=IS;}
        if (x>=TLCx[M][1] && x<=dx)
                I=M;  //defined I from x
        for (int JS=1;JS<N;JS++)
        {if (y>=TLCy[1][JS] && y < TLCy[1][JS+1])
        J=JS;}
        if (y>=TLCy[1][N] && y<=dy)
                J=N; //defined J from y
        i=(I-1)*m+int(m*(x-TLCx[I][J])/DX[I]); //defined i from I,x
        j=(J-1)*n+int(n*(y-TLCy[I][J])/DY[J]); //defined j from J,y
}
void Walker::CalculateV(){
        int I1,I2,I3,I4,J1,J2,J3,J4; //(I1;J1), (I2;J2), (I3;J3) and (I4;J4) - surrounding cells;
velocity will be interpolated using values in their centers
        // left half of the cell
        if (j%n<n/2)
        {if(j<(n/2)) // at the left edge
        {J1=1;
```

```
J2=2;
} // J3=J1;J4=J2
else
{J2=J;
J1=J2-1;
}
}
// right half of the cell
if (j%n>=(n/2))
{if(j>=(N*n-n/2)) // at the right edge
{J1=N-1;
J2=N;
}
else
{J1=J;
J2=J1+1;
}
}
// top half of the cell
if (i%m<m/2)
{if(i<m/2) // at the top edge
{I1=1;
I3=2;
}
else
{I3=I;
I1=I3-1;
}
}
// bottom half of the cell
if (i%m>=m/2)
{if(i>=(M*m-m/2)) // at the bottom edge
{
        I1=M-1;
        I3=M;
}
else
{I1=I;
I3=I1+1;
}
}
I2=I1;
I4=I3;
J3=J1;
J4=J2;
VforCell(I1,J1,I2,J2,I3,J3,I4,J4);
}
void Walker::VforCell(int I1, int J1, int I2, int J2, int I3, int J3, int I4, int J4)
{ // interpolates velosities Vx and Vy for the walker from cells [I1,J1],[I2,J2],[I3,J3],[I4,J4]
double Vx13=(p[I1][J1]-p[I3][J3])*mD[I1][J1]*2/((DX[I1]+DX[I3])*100000);
double Vx24=(p[I2][J2]-p[I4][J4])*mD[I2][J2]*2/((DX[I2]+DX[I4])*100000);
double Vy12=(p[I1][J1]-p[I2][J2])*mR[I1][J1]*2/((DY[J1]+DY[J2])*100000);
double Vy34=(p[I3][J3]-p[I4][J4])*mR[I3][J3]*2/((DY[I3]+DY[J4])*100000);
double x1=TLCx[I1][J1]+DX[I1]/2;
```

```
        double y1=TLCy[I1][J1]+DY[J1]/2;
        double x2=TLCx[I2][J2]+DX[I2]/2;
        double y2=TLCy[I2][J2]+DY[J2]/2;
        double x3=TLCx[I3][J3]+DX[I3]/2;
        double y3=TLCy[I3][J3]+DY[J3]/2;
        double x4=TLCx[I4][J4]+DX[I4]/2;
        double y4=TLCy[I4][J4]+DY[J4]/2;
        Vx=((y-y1)/(y2-y1))*Vx24+((y2-y)/(y2-y1))*Vx13;
        Vy=((x-x1)/(x3-x1))*Vy34+((x3-x)/(x3-x1))*Vy12;
        t1=Vx13;
        t2=Vx24;
        t3=Vy12;
        t4=Vy34;
        if (fluid==1)
                Vy=Vy+(double)perm[I][J]*rho1*g/(10000000*ViscMixC[I][J]); //Vy - cm/s
10^7 - conversion
        if (fluid==2)
                Vy=Vy+(double)perm[I][J]*rho2*g/(10000000*ViscMixC[I][J]); //Vy - cm/s
10^7 - conversion
}
void Walker::RandomStep(){
        if (fluid==1)
                D=Do;
        else if (fluid==2)
                D=Ds;
        double rx=Norm(0,1);
        double ry=Norm(0,1);
        if (rx<-4)
        {rx=-4;}
        if (rx>4)
        {rx=4;}
        if (ry<-4)
        {ry=-4;}
        if (ry>4)
        {ry=4;}
        x=x+sqrt(2*D*dt)*rx;
        y=y+sqrt(2*D*dt)*ry;
        if (x<0) x=(-1)*x;
        if (y<0) y=(-1)*y;
        if( dx-x<0) x=2*dx-x; // particle reflects from boundary;
        if( dy-y<0)
        {if (I==Mid)
        {y=dy;}
        else
        {y=2*dy-y;}
        }   // near to outlet particle 'sticks' to the exit, not reflected

        for (int IS=1;IS<M;IS++)
        {if (x>=TLCx[IS][1] && x < TLCx[IS+1][1])
        I=IS;}
        if (x>=TLCx[M][1] && x<=dx)
                I=M;  //defined I from x
        for (int JS=1;JS<N;JS++)
        {if (y>=TLCy[1][JS] && y < TLCy[1][JS+1])
```

```
		J=JS;}
		if (y>=TLCy[1][N] && y<=dy)
			J=N; //defined J from y
		i=(I-1)*m+int(m*(x-TLCx[I][J])/DX[I]); //defined i from I,x
		j=(J-1)*n+int(n*(y-TLCy[I][J])/DY[J]); //defined j from J,y
}
void ProduceAll()
{int PrCounter=0;
int OProduced=0;
int SProduced=0;
string str = "production.txt";
ofstream output(str.c_str(), std::ios::app);
output.flush();
for (int s=1; s<=RealSize; s++)
{if (walkers[s].I==Mid && walkers[s].y>(dy-ProdZone)&&walkers[s].fluid>0)
{PrCounter++;
ToProduce[PrCounter]=s;
}
}  // there are PrCounter walkers in ProdZone; numbers of those walkers are stored in ToProduce
array.
{for (int t=1; t<=PrCounter; t++)
{int s=ToProduce[t];
if (walkers[s].fluid==1) OProduced=OProduced+1;
if (walkers[s].fluid==2) SProduced=SProduced+1;
walkers[s].fluid=-999;
walkers[s].x=-999;
walkers[s].y=-999;
walkers[s].i=-999;
walkers[s].j=-999;
walkers[s].I=-999;
walkers[s].J=-999; // this walker is not any more in the system
}
OilProducedTotal=OilProducedTotal+OProduced;
output<<time_s<<' '<<OProduced<<' '<<SProduced<<' '<<OilProducedTotal<<endl;
}
output.close();
}
void WalkersToFile(){  //output in petrel welltops format
		char text[256]="";
		// for oil
		itoa(time_s,text,10);
		string str = "Walkers";
		string strH = "Horizon";
		str+=text;
		str+=".txt";
		ofstream output(str.c_str());
		output.flush();
		output<<'*'<<'W';
		output<<endl;
		for (int s=1;s<=RealSize;s++){
			if(walkers[s].fluid>0) //don't want to output 'produced' walkers
			{
output<<walkers[s].x<<' '<<walkers[s].y<<' '<<'0'<<' '<<s<<' '<<walkers[s].fluid<<' '<<strH<<'
'<<time_s;
```

```
                output<<endl;
            }
        }
        output.close();
}
void WalkersToFileM(){ //output in matlab format
        char text[256]="";
        string str = "MW";
        for (int i=1;i<=6;i++)
        {if (time_s==TS[i])
        itoa(i,text,10);
        }
        str+=text;
        str+=".txt";
        ofstream output(str.c_str());
        output.flush();
        for (int s=1;s<=RealSize;s++){
            if(walkers[s].fluid>0) //don't want to output 'produced' walkers
            {
                output<<walkers[s].x<<' '<<walkers[s].y<<' '<<walkers[s].fluid;
                output<<endl;
            }
        }
        output.close();
}
void ParametersToFile(){ //export parameters used for this run
        string str = "parameters.txt";
        ofstream output(str.c_str());
        output.flush();
        output<<mu1<<' '<<(Q*3600)<<' '<<Do<<' '<<Ds<<' '<<Perm_f<<' '<<w;
        output.close();
}
double Norm(double mean, double d) //returns randon value distributed as N(mean,d)
{double r1=(double)rand()/RAND_MAX; // r1, r2 in [0...1)
double r2=(double)rand()/RAND_MAX;
double Z=sqrt(-2*log(r1)/log(2.718281828))*cos(2*3.14159265358979*r2); //Z - N(0,1)
return(mean+d*Z);
}
void ToFile(double *Matrix, int NumLin, int NumCol, char FileName){ //exports *Matrix array to
file FileName
        char text[256]="";
        itoa(time_s,text,10);
        string str;
        stringstream ss;
        ss << FileName;
        ss >> str;
        str+=text;
        str+=".txt";
        ofstream output(str.c_str());
        output.flush();
        output<<'*'<<FileName;
        output<<endl;
        for (int k=0;k<NumLin;k++){
            for (int r=0;r<NumCol;r++)
```

```
                {output<<*(Matrix+(NumCol*k+r))<<' ';}
                output<<endl;
        }
        output.close();
}
// for system solving
void SetT() //T=A|b
{
        for (int i=0;i<M*N; i++)
        {for (int j=0; j<M*N; j++)
        {T[i][j]=A[i][j];}}
        for (int i=0; i<M*N; i++)
        {T[i][M*N]=b[i];}
}
void T_div(int k, double d)
{
        for (int j=0; j<=M*N; j++)
                T[k][j]=T[k][j]/d;
}
void T_min(int i1, int i2, double mult)
{
        for (int j=0; j<=M*N;j++)
                T[i2][j]=T[i2][j]-T[i1][j]*mult;
}
void Solve()  // solving AX=B, T is {A|B}, solution will be stored in X.
{
        for (int s=0;s<=M*N-1;s++)
        {
                T_div(s,T[s][s]);
                for (int k=s+1;k<=M*N-1;k++)
                        if (T[k][s]!=0) {T_min(s,k,T[k][s]);}
        }
        x[M*N-1]=T[M*N-1][M*N];
        for (int s=M*N-2;s>=0;s--)
        {double sum=0;
        for(int k=(s+1);k<=M*N-1;k++)
        {sum=sum+(T[s][k]*x[k]);}
        x[s]=T[s][M*N]-sum;
        }
    }
```

Appendix B (C++ code for the RWPT algorithm)

Header file (main.h)

```
int nXFr;  // (current) number of fractions in X direction
int nYFr;  // (current) number of fractions in Y direction - should be much bigger than nXFr
#define nXFrM  1500 //max number of X-fractures
#define nYFrM  1500 //max number of Y-fractures
#define nM 30 // maximum number of matrix edges going from each vertex
#define ProdTime 250 //max production time, days
double fwx, fwy; // width of a fracture
```

```
double MaxX; // maximum x-size of a cell
double XFr[nXFrM][14]; // array with x-fractures
double YFr[nYFrM][14]; // array with y-fractures
double Xj[nXFrM*2+nYFrM*2+200]; // all x-ticks, 2 ends for esch x-fracture and for each y-
fracture
double Yj[nXFrM*2+nYFrM*2]; // all y-ticks, 2 ends for esch x-fracture and for each y-fracture
double Zj[nXFrM*2+nYFrM*2]; // all z-ticks, 2 ends for esch x-fracture and for each y-fracture
double WellCoord[16]; //well coords data: I1x, I1y, I2x, I2y, ....
int CompDat[16]; // data for wcompdat keyword: I1i, I1j, I2i, I2j, ...
int InjProdInd[8]; //vertexes corresponding to wells I1, I2,I3,I4,FS1,S1,S2,S3,S4
int Production[4][ProdTime*10]; //how many walkers of each type (=from each prod well) were
produced at each 0.1 day
double PermF, PoroF, PoroM, PermM,PermXfr,PermYfr;
double g; //m/s2
double rho; //kg/m3
double C; //correction coefficient
double q;// rate, m3/day
double q_i1, q_i2, q_i3, q_i4, q_fs1, q_s1, q_s2, q_s3; // rate, m3/day
int Nx, Ny, Nz; //number of cells in x, y, z directions
double MarH, MarV; //margins: model has margins of these sizes, margins do not have fractures
double XShift, YShift; //to shift wells in a way that S1 will go to the center of the grid
int NumVert;
double mu; //viscosity, cp.
double dt;//how often we send particles, sec.
double Dt1, Dt2, Dt3, Dt4;//tracer injection duration for each well, sec.
double D; //dispersion coeff-t
double Seed;//
int N; //number of particles
int ParticlesProduced;
//-----DFN parameters-----
double theta; //angle between main fracture set geol and simulated; -0.22643 rad - will put S3 and
I3 on the same line
double Xspm, XspM,Yspm,YspM; // spasing for x-fractures and y fractures
double Lxm,Lxd,Lym,Lyd,Lzm,Lzd;//lengths of fractures are distributed ~N(Lxm,Lxd)
double Xmin,XMax,Ymin,YMax,Zmin,ZMax,Zcmin,ZcMax; //limits for x,y,z
double WellFL; //half of the well fracture lengths;

//-----Matrix flow parameters-----
double Rad;
double MF1, MF2; //coefficient responsible for Kr, m-f interaction etc - MF1 - for calc velosity;
MF2 - for calc probability

class Pairs{
public:
        int vid,xory;
        double dob;
};
class Cell{
public:
        double P;
};
class grid {
public:
        Cell* cells;
```

```
        void ToFile(int,int,int,char);
        void FillPress();
        grid(){
   cells=new Cell[(Nx+1)*(Ny+1)*(Nz+1)];
        }
        ~grid(){
   delete [] cells;
        }
        Cell * G(int,int, int);
};
class vertice {
public:
        int edges[4]; //edges going from that vertexes
        double dP[4]; //(Potential at this vertex) - (Potential at the other end)
        int OtherEnd[4]; //number of vertex on the other end of the fracture edge
        int FrNum[4];//to which fracture this Edge belongs. If FrNum is [0 .. nXFr-1] - its x-
fracture (XFr[FrNum]). If it is [nXFr .. nXFr+nYFr-1] - then its y-fracture (YFr[FrNum-nXFr]).
        double L[4]; //length of the edge - horizintal component only
        double V[4];// velocity for fracture edge, m/s
        double Prob[4];//probability~q~V*A=V*H*fw
        int OtherEndM[nM];  //number of vertex on the other end of the matrix edge
        double dPM[nM]; //pressure potential for matrix edge
        double LM[nM]; //length for matrix edge
        double VM[nM]; //velosity for matrix edge
        double ProbM[nM]; //probability for matrix edge
        int NEdg,i,j,k,NM;
        double P;
        double pot;
};
class GraphV{
public:
        vertice* verts;
        void ToFile(char);
        void ToFileWT(char);
        void ToFilePoly(char);
         GraphV(){
         verts=new vertice[Nx*Ny*Nz];
         }
         ~GraphV(){
                delete [] verts;
         }
};
class particle{
public:
        int vid;//current vertex
        int tr;//particle index
        int WN; //well number
        double t;
        void Run(GraphV&);
};
class particles{
public:
        particle* SetOfP;
        void OneRun(GraphV&, int, int, double, int );
```

```
        particles(){
                SetOfP=new particle[100];
        }
                ~particles(){
                        delete [] SetOfP;
                }
};

bool MyCompare(Pairs, Pairs);
double MyRand(double , double ); // generates random value between min and max
double Norm(double, double); //generates normally distributed random number, with mean m and
deviation d
double MyRound(double , int); //rounding to certain decimal
void GenerateXFr();
void GenerateYFr(); //to generate based on DFN parameters
void CheckXFr();
void CheckYFr();
void Initiate();
void EditFromFile();
void CreateInput();
void MergeX();
void MergeY();
void MergeZ();
void Dimens();
void DX();
void DY();
void DZ();
void TOPS();
void MAPAXES();
void WELSPECS();
void COMPDAT();
void WCONINJ();
void WCONPROD();
void WELPI();
void CompleteXFr();
void CompleteYFr();
void EQUALS();
void FillCompDat();
void FillWellCoord(); //well S1 will be at (0,0), and they all will be rotated by theta
void FillGraphGeom(grid&, GraphV&);
void FillGraphProps(grid&, GraphV&);
void GraphAddMatr(grid&, GraphV&);
void FillInjProdInd(GraphV&);
void CreateGraph(GraphV& , grid& );
void SendChunk(GraphV& , particles& ,int);
void EmptyProduction();

//output
void ParametersToFile();
void ToFile(int, int, char);
void XFracturesAsPolygons(int , char ); //exports x-fractures as polygons - in petrel format
void YFracturesAsPolygons(int , char ); //exports y-fractures as polygons - in petrel format
```

Source file (main.cpp)

```
#include "main.h"
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <fstream>
#include <iostream>
#include <string>
#include <cmath>
#include <cstdio>
#include <time.h>
#include<conio.h>
#include <sstream>
#include<algorithm>
#include<windows.h>
using namespace std;
int main()
{Initiate();
//EditFromFile(); //usefull to run set of simulations from external application, e.g. Matlab
 CreateInput(); // generates fracture network based on DFN parameters and creates include files for
Eclipse run based on the fracture network
  grid MyGrid;
  GraphV MyGraph;
  particles MyParticles;
            system("run.bat");   //runs Eclipse file; if eclipse file in other folder - it should be
indicated in the bat file
    system("run2.bat");  // copies eclipse output file (*.F000*) back to the main folder
  CreateGraph(MyGraph, MyGrid);
  SendChunk(MyGraph, MyParticles,InjProdInd[0]);
}
void Initiate()
{ParticlesProduced=0;
nXFr=0;
nYFr=0;
fwx=0.005; // width of x-fractures
fwy=0.005; // width of y-fractures
 MarH=18;
 MarV=1;
 PermXfr=160000;
 PermYfr=140000;
 PoroF=1;
 PermM=50;
 PoroM=0.05;
 MaxX=10;
 rho=1020.3;
 g=9.80665;
 C=0.984294491895537; // C=1;
 q=20;
 q_s1=20;
 q_s2=20;
 q_s3=20;
 q_fs1=3.5;
 q_i1=20;
```

```
 q_i2=20;
 q_i3=20;
 q_i4=20;
 mu=1;
 D=0.000001;
 dt=10;
 Dt1=11400; //for I1 - tracer injection duration, sec.
 Dt2=5400;  // for I2   -"-
 Dt3=11040; //for I3 -"-
 Dt4=9240;  // for I4 -"-
  //---------DFN parameters-----------
 theta=0.04;//-angle between main fracture set geol and simulated; -0.22643 rad - will put S3 and
I3 on the same line
 Yspm=3.5; //
 YspM=4.5;//spasing between y-fractures is betweem Yspm and YspM
 Xspm=Yspm*4; //
 XspM=YspM*4; // spasing between x-fractures is betweem Xspm and XspM
 Lym=200; Lyd=10; //lengths of y-fractures are distributed ~N(Lym,Lyd)
 Lxm=50; Lxd=5; //lengths of x-fractures are distributed ~N(Lxm,Lxd)
 Lzm=7; Lzd=1; //heights of fractures are distributed ~N(Lzm,Lzd)
 Xmin=-140; XMax=140;
 Ymin=-80;YMax=80;
 Zmin=-1412;ZMax=-1392;Zcmin=-1400; ZcMax=-1395;
 WellFL=22;
 //-----Matrix flow parameters-----
Rad=2; //
MF1=20;  // MF1 - for calc velosity;
MF2=0; // MF2 - for calc probability
 }
void EditFromFile()
{string str;
            {ifstream indata; // indata is like cin
  indata.open("Edit.txt"); // opens the file
   if(!indata) { // file couldn't be opened
      cerr << "Error: file could not be opened" << endl;
   }
   indata >> str; char *a=new char[str.size()+1]; memcpy(a,str.c_str(),str.size());
   Yspm=atof(a);
   YspM=Yspm;
   indata >> str;  a=new char[str.size()+1]; memcpy(a,str.c_str(),str.size());
   Xspm=atof(a)*Yspm;
   XspM=Xspm;
   delete [] a;
   indata.close();
   srand(Seed);
 }
}
void CreateInput()
{FillWellCoord();
 GenerateXFr();
 if(nXFr>nXFrM)printf("nXFr>nXFrM");
 GenerateYFr();
 if(nYFr>nYFrM)printf("nYFr>nYFrM");
 CheckXFr();
```

```
 CheckYFr();
 MergeX();
 MergeY();
 MergeZ();
 FillCompDat();
 Dimens();
 DX();
 DY();
 DZ();
 TOPS();
 MAPAXES();
 CompleteXFr();
 CompleteYFr();
 EQUALS();
 WELSPECS();
 COMPDAT();
 WCONINJ();
 WCONPROD();
system("xcopy DIMENS \EclFiles /d /y");
system("xcopy DX \EclFiles /d /y");
system("xcopy DY \EclFiles /d /y");
system("xcopy DZ \EclFiles /d /y");
system("xcopy TOPS \EclFiles /d /y");
system("xcopy MAPAXES \EclFiles /d /y");
system("xcopy EQUALS \EclFiles /d /y");
system("xcopy WELSPECS \EclFiles /d /y");
system("xcopy COMPDAT \EclFiles /d /y");
system("xcopy WCONINJ \EclFiles /d /y");
system("xcopy WCONPROD \EclFiles /d /y");
system("xcopy WELPI \EclFiles /d /y");
}
void CreateGraph(GraphV& MyGraph, grid& MyGrid)
{ MyGrid.FillPress();
  FillGraphGeom(MyGrid, MyGraph);
  FillGraphProps(MyGrid, MyGraph);
  GraphAddMatr(MyGrid, MyGraph);
  FillInjProdInd(MyGraph);
}
void SendChunk(GraphV& MyGraph, particles& MyParticles, int InjWellNo)
{           int N1, N2, N3, N4, Nt;
            //------I1------//
            EmptyProduction();
    N1=(int)Dt1/dt;
for (int i=0;i<N1;i++)
  { MyParticles.SetOfP[0].tr=i;//particle index;
 MyParticles.SetOfP[0].WN=1;//InjWellNumber;
 MyParticles.SetOfP[0].vid=InjProdInd[0];
 MyParticles.SetOfP[0].t=i*dt;
 MyParticles.SetOfP[0].Run(MyGraph);
 }
ToFile(ProdTime*10, 4, '1');
//------I2------//
EmptyProduction();
 N2=(int)Dt2/dt;
```

```
 EmptyProduction();
 for (int i=0;i<N2;i++)
  { MyParticles.SetOfP[0].tr=i+N1;//particle index;
 MyParticles.SetOfP[0].WN=2;//InjWellNumber;
 MyParticles.SetOfP[0].vid=InjProdInd[1];
 MyParticles.SetOfP[0].t=i*dt;
 MyParticles.SetOfP[0].Run(MyGraph);
 }
ToFile(ProdTime*10, 4, '2');
//------I3------//
EmptyProduction();
 N3=(int)Dt3/dt;
 for (int i=0;i<N3;i++)
  { MyParticles.SetOfP[0].tr=i+N1+N2;//particle index;
 MyParticles.SetOfP[0].WN=3;//InjWellNumber;
 MyParticles.SetOfP[0].vid=InjProdInd[2];
 MyParticles.SetOfP[0].t=i*dt;
 MyParticles.SetOfP[0].Run(MyGraph);
 }
ToFile(ProdTime*10, 4, '3');
//------I4------//
EmptyProduction();
 N4=(int)Dt4/dt; //(!ch)
 for (int i=0;i<N4;i++)
 { MyParticles.SetOfP[0].tr=i+N1+N2+N3;//particle index;
 MyParticles.SetOfP[0].WN=4;//InjWellNumber;
 MyParticles.SetOfP[0].vid=InjProdInd[3];
 MyParticles.SetOfP[0].t=i*dt;
 MyParticles.SetOfP[0].Run(MyGraph); //(!ch)
 }
ToFile( ProdTime*10, 4, '4');
}
void FillWellCoord()
{ifstream myfile("WellCoord0.txt");
//I1x,I1y,I2x,I2y,I3x,I3y,I4x,I4y,FS1x,FS1y,S1x,S1y,S2x,S2y,S3x,S3y - rotated, shifted in a way
that S1 is at (0,0)
  if(!myfile)
  {
    cout<<"Could not open file"<<std::endl;
  }
 int lin=0;
 for (lin=0;lin< 16;lin++)
 {
                        myfile>> WellCoord[lin];
 }
 // now we'll have to rotate by theta
 for (int i=0;i<8;i++)
 {double newx=WellCoord[i*2]*cos(theta)+WellCoord[i*2+1]*sin(theta);
  double newy=-WellCoord[i*2]*sin(theta)+WellCoord[i*2+1]*cos(theta);
  WellCoord[i*2]=MyRound(newx,1);
  WellCoord[i*2+1]=MyRound(newy,1);
 }
}
void GenerateXFr()
```

```
{double L,c;
            //nXFr=0 initially; keep increasing while adding fractures
// add x-fractures because of wells
 for (int i=0;i<8;i++)
 {XFr[nXFr][0]=WellCoord[i*2];
  XFr[nXFr][3]=WellCoord[i*2];
  XFr[nXFr][1]=max(WellCoord[i*2+1]-WellFL,Ymin+MarH);
  XFr[nXFr][4]=min(WellCoord[i*2+1]+WellFL, YMax-MarH);
  XFr[nXFr][5]=ZMax-MarV;
  XFr[nXFr][2]=XFr[nXFr][5]-7;
  XFr[nXFr][12]=PermXfr;
  XFr[nXFr][13]=fwx;
  nXFr++;
 }
 int InB=1;
 //first x-fracture (with smallest x-value)
 XFr[nXFr][0]=Xmin+MarH+MyRand(Xspm,XspM);
 XFr[nXFr][3]=XFr[nXFr][0];
 L=Norm(Lxm,Lxd); //length of fraction
 c=MyRand(Ymin,YMax);//y-coord of center
 XFr[nXFr][1]=max(c-(double)L/2,Ymin+MarH);
 XFr[nXFr][4]=min(c+(double)L/2, YMax-MarH);
 L=Norm(Lzm,Lzd); //height of fraction
 c=MyRand(Zcmin,ZcMax);//z-coord of center
 XFr[nXFr][2]=max(c-(double)L/2,Zmin+MarV);
 XFr[nXFr][5]=min(c+(double)L/2, ZMax-MarV);
 XFr[nXFr][12]=PermXfr;
 XFr[nXFr][13]=fwx;
 //---
 while(InB==1) //now all remaining x-fractures
 {nXFr++;
 XFr[nXFr][0]=XFr[nXFr-1][0]+MyRand(Xspm,XspM);
 XFr[nXFr][3]=XFr[nXFr][0];
 L=Norm(Lxm,Lxd); //length of fraction
 c=MyRand(Ymin,YMax);//y-coord of center
 XFr[nXFr][1]=max(c-(double)L/2,Ymin+MarH);
 XFr[nXFr][4]=min(c+(double)L/2, YMax-MarH);
 L=Norm(Lzm,Lzd); //height of fraction
 c=MyRand(Zcmin,ZcMax);//z-coord of center
 XFr[nXFr][2]=max(c-(double)L/2,Zmin+MarV);
 XFr[nXFr][5]=min(c+(double)L/2, ZMax-MarV);
 XFr[nXFr][12]=PermXfr;
 XFr[nXFr][13]=fwx;
 if (XFr[nXFr][0]>XMax-MarH) //this fracture should not be considered, and process should be
stopped
 {nXFr--;
  InB=0;
 }
 }
 //now lets round everything
 for (int lin=0;lin<nXFr;lin++)
 {
            for (int col=0;col<5;col++)
            {XFr[lin][col]=MyRound(XFr[lin][col],1);}
```

```
            XFr[lin][2]=MyRound(XFr[lin][2],0);
            XFr[lin][5]=MyRound(XFr[lin][5],0);
 }
}
void GenerateYFr()
{double L,c;
          //nYFr=0 initially; keep increasing while adding fractures
          // add y-fractures because of wells
 for (int i=0;i<8;i++)
 {YFr[nYFr][1]=WellCoord[i*2+1];
  YFr[nYFr][4]=WellCoord[i*2+1];
  YFr[nYFr][0]=WellCoord[i*2]-WellFL;
  YFr[nYFr][3]=WellCoord[i*2]+WellFL;
  YFr[nYFr][5]=ZMax-MarV;
  YFr[nYFr][2]=YFr[nYFr][5]-7;
  YFr[nYFr][12]=PermYfr;
  YFr[nYFr][13]=fwy;
  nYFr++;
 }
 int InB=1;
 //---first y-fracture (with smallest y-value)
 YFr[nYFr][1]=Ymin+MarH+MyRand(Yspm,YspM);
 YFr[nYFr][4]=YFr[nYFr][1];
 L=Norm(Lym,Lyd); //length of fraction
 c=MyRand(Xmin,XMax);//y-coord of center
 YFr[nYFr][0]=max(c-(double)L/2,Xmin+MarH);
 YFr[nYFr][3]=min(c+(double)L/2, XMax-MarH);
 L=Norm(Lzm,Lzd); //height of fraction
 c=MyRand(Zcmin,ZcMax);//z-coord of center
 YFr[nYFr][2]= max(c-(double)L/2,Zmin+MarV);
 YFr[nYFr][5]=min(c+(double)L/2, ZMax-MarV);
 YFr[nYFr][12]=PermYfr;
 YFr[nYFr][13]=fwy;
 //---
 while(InB==1) //now all remaining y-fractures
 {nYFr++;
 YFr[nYFr][1]=YFr[nYFr-1][1]+MyRand(Yspm,YspM);
 YFr[nYFr][4]=YFr[nYFr][1];
 L=Norm(Lym,Lyd); //length of fraction
 c=MyRand(Xmin,XMax);//y-coord of center
 YFr[nYFr][0]=max(c-(double)L/2,Xmin+MarH);
 YFr[nYFr][3]=min(c+(double)L/2, XMax-MarH);
 L=Norm(Lzm,Lzd); //height of fraction
 c=MyRand(Zcmin,ZcMax);//z-coord of center
 YFr[nYFr][2]=max(c-(double)L/2,Zmin+MarV);
 YFr[nYFr][5]=min(c+(double)L/2, ZMax-MarV);
 YFr[nYFr][12]=PermYfr;
 YFr[nYFr][13]=fwy;
 if (YFr[nYFr][1]>YMax-MarH) //this fracture should not be considered, and process should be
stopped
 {nYFr--;
  InB=0;
 }
 }
```

```
 //now lets round everything
 for (int lin=0;lin<nYFr;lin++)
 {
            for (int col=0;col<5;col++)
            {
                        YFr[lin][col]=MyRound(YFr[lin][col],1);
            }
            YFr[lin][2]=MyRound(YFr[lin][2],0);
            YFr[lin][5]=MyRound(YFr[lin][5],0);
 }
}
void CheckXFr()
{int s=0;
 while (s<nXFr)
 {int f=s+1;
  while (f<nXFr) //checking for fracture s if it overlaps with any of the following fractures; remove
any which will overlap
  {if(XFr[s][0]==XFr[f][0]&&(XFr[s][4]-XFr[f][1])*(XFr[f][4]-XFr[s][1])>0 && (XFr[s][5]-
XFr[f][2])*(XFr[f][5]-XFr[s][2])>0) //if they overlap
  {for(int j=0;j<12;j++)XFr[f][j]=XFr[nXFr-1][j]; //replaced f-th element my last element of the
array
   nXFr=nXFr-1;
  }
  f++;
  }
  s++;
 }
}
void CheckYFr()
{int s=0;
 while (s<nYFr)
 {int f=s+1;
  while (f<nYFr) //checking for fracture s if it overlaps with any of the following fractures; remove
any which will overlap
  {if(YFr[s][1]==YFr[f][1]&&(YFr[s][3]-YFr[f][0])*(YFr[f][3]-YFr[s][0])>0 && (YFr[s][5]-
YFr[f][2])*(YFr[f][5]-YFr[s][2])>0) //if they overlap
  {for(int j=0;j<12;j++)YFr[f][j]=YFr[nYFr-1][j]; //replaced f-th element my last element of the
array
   nYFr=nYFr-1;
  }
  f++;
  }
  s++;
 }
}
void MergeX() //put all 'ticks' because of x and y fractures to the array Xj
{   int i,j,k;
            for(i=0;i<nXFr;i++){
            Xj[2*i]=XFr[i][0];
    Xj[2*i+1]=XFr[i][0]+XFr[i][13]; //ticks because of x-fractures ; XFr[i][13] is fw
    }
            for( i=0;i<nYFr;i++){
            Xj[2*nXFr+2*i]=YFr[i][0];
            Xj[2*nXFr+2*i+1]=YFr[i][3]; //ticks because of y-fractures
```

```
   }
   sort(Xj, Xj+(nXFr+nYFr)*2);
// now we'll remove repeating values
 j=1;
for ( i=1;i<(nXFr+nYFr)*2;i++)
{if (Xj[i]==Xj[i-1])
{} //just move to next i
else{
            Xj[j]=Xj[i];
            j++;
}
}
Nx=j+1; // there are only j different values => j+1 cells.
//  now we'll add additional nodes in a way that each cell has DX not more that MaxX;
 i=0;
while(i<Nx-2)
{if((Xj[i+1]-Xj[i])>MaxX)
{int N=(int)((Xj[i+1]-Xj[i])/MaxX); //add N nodes to the interval (Xj(i);Xj(i+1)).
 double Size=(Xj[i+1]-Xj[i])/(N+1); //size of each cell;
 for( k=Nx-2;k>=i+1;k--)
 {Xj[k+N]=Xj[k];} //shift all elements starting from i+1 N positons right (to create empty space
for N nodes)
 for( k=1;k<=N;k++)
 {Xj[i+k]=Xj[i]+k*Size;} //adding nodes
 Nx=Nx+N;//because we added N nodes
 i=i+N;
}
else i++;
}
}
void MergeY()
{           int i,j,k;
   for( i=0;i<nYFr;i++){
            Yj[2*i]=YFr[i][1];
    Yj[2*i+1]=YFr[i][1]+YFr[i][13]; //ticks because of y-fractures ; YFr[i][13] if fw
    }
   for( i=0;i<nXFr;i++){
            Yj[2*nYFr+2*i]=XFr[i][1];
            Yj[2*nYFr+2*i+1]=XFr[i][4]; //ticks because of x-fractures
   }
sort(Yj, Yj+(nXFr+nYFr)*2);
// now we'll remove repeating values
 j=1;
for ( i=1;i<(nXFr+nYFr)*2;i++)
{if (Yj[i]==Yj[i-1])
{} //just move to next i
else{
            Yj[j]=Yj[i];
            j++;
}
}
Ny=j+1; // there are only j different values => j+1 cells.
}
void MergeZ()
```

```
{   for(int i=0;i<nXFr;i++){
            Zj[2*i]=XFr[i][2];
    Zj[2*i+1]=XFr[i][5]; //ticks because of x-fractures
    }
   for(int i=0;i<nYFr;i++){
            Zj[2*nXFr+2*i]=YFr[i][2];
            Zj[2*nXFr+2*i+1]=YFr[i][5]; //ticks because of y-fractures
   }
sort(Zj, Zj+(nXFr+nYFr)*2);
// now we'll remove repeating values
int j=1;
for (int i=1;i<(nXFr+nYFr)*2;i++)
{if (Zj[i]==Zj[i-1])
{} //just move to next i
else{
            Zj[j]=Zj[i];
            j++;
}
}
Nz=j+1; // there are only j different values => j+1 cells.
}
void FillCompDat()
{for (int i=0;i<8;i++)
            {int j=0;
             while (j<=Nx-2)
             {if(WellCoord[i*2]==Xj[j])
             {CompDat[i*2]=j+2; //i coordinate of i-th well
              j=Nx; }// to finish cycle
             else {j++;}
             }
             j=0;
             while (j<=Ny-2)
             {if(WellCoord[i*2+1]==Yj[j])
             {CompDat[i*2+1]=j+2; //j coordinate of i-th well
              j=Ny; }// to finish cycle
             else {j++;}// to finish cycle
             }
}}
void Dimens()
{string str = "DIMENS";
ofstream output(str.c_str());
output.flush();
output<<str<<endl<<Nx<<' '<<Ny<<' '<<Nz<<' '<<'/';
output.close();
}
void DX()
{string str = "DX";
ofstream output(str.c_str());
output.flush();
string str2 = "BOX";
string str3 = "ENDBOX";
output<<str2<<endl<<'1'<<' '<<Nx<<' '<<'1'<<' '<<Ny<<' '<<'1'<<' '<<'1'<<'/'<<endl;
output<<str<<endl;
for(int j=1;j<=Ny;j++)
```

```
{
	output<<Xj[0]-Xmin<<' ';
	for(int i=0;i<=Nx-3;i++)
	//{output<<Xj[i+1]-Xj[i]<<' ';}
	{output<<Xj[i+1]-Xj[i]<<endl;}
	output<<XMax-Xj[Nx-2]<<endl;
}
output<<'/'<<endl<<str3;
output.close();
}
void DY()
{string str = "DY";
ofstream output(str.c_str());
output.flush();
string str2 = "BOX";
string str3 = "ENDBOX";
output<<str2<<endl<<'1'<<' '<<Nx<<' '<<'1'<<' '<<Ny<<' '<<'1'<<' '<<'1'<<'/'<<endl;
output<<str<<endl;
	output<<Nx<<'*'<<Yj[0]-Ymin<<endl;
	for(int i=0;i<=Ny-3;i++)
	{output<<Nx<<'*'<<Yj[i+1]-Yj[i]<<endl;}
	output<<Nx<<'*'<<YMax-Yj[Ny-2]<<endl;
output<<'/'<<endl<<str3;
output.close();
}
void DZ()
{string str = "DZ";
ofstream output(str.c_str());
output.flush();
output<<str<<endl;
	output<<Nx*Ny<<'*'<<MarV<<endl;
	for(int i=Nz-2;i>=1;i--)
	{output<<Nx*Ny<<'*'<<Zj[i]-Zj[i-1]<<endl;}
	output<<Nx*Ny<<'*'<<MarV<<endl;
output<<'/';
output.close();
}
void MAPAXES()
	{string str = "MAPAXES";
ofstream output(str.c_str());
output.flush();
output<<str<<endl;
output<<Xmin<<' '<<YMax<<' '<<Xmin<<' '<<Ymin<<' '<<XMax<<' '<<Ymin<<'/'<<endl;
output.close();
}
void TOPS()
{string str = "TOPS";
ofstream output(str.c_str());
output.flush();
string str2 = "BOX";
string str3 = "ENDBOX";
output<<str2<<endl<<'1'<<' '<<Nx<<' '<<'1'<<' '<<Ny<<' '<<'1'<<' '<<'1'<<'/'<<endl;
output<<str<<endl;
	output<<Nx*Ny<<'*'<<(-1)*ZMax<<endl;
```

```
output<<'/'<<endl<<str3;
output.close();
}
void CompleteXFr() //add i,j,k limits for each fracture
{
            for (int i=0;i<nXFr;i++)
            {int j=0;
             while (j<=Nx-2)
             {if(XFr[i][0]==Xj[j])
             {XFr[i][6]=j+2;
              XFr[i][7]=j+2; //i cells fracture occupies
              j=Nx; }// to finish cycle
             else {j++;}
             }
             j=0;
             while (j<=Ny-2)
             {if(XFr[i][1]==Yj[j])
             {XFr[i][8]=j+2; //first j-cell fracture occupies
             j=Ny;} // to finish cycle
             else {j++;}
             }
             j=0;
             while (j<=Ny-2)
             {if(XFr[i][4]==Yj[j])
             {XFr[i][9]=j+1; //last j-cell fracture occupies
             j=Ny;} // to finish cycle
             else {j++;}
             }
             j=0;
             while (j<=Nz-2)
             {if(XFr[i][2]==Zj[j])
             {XFr[i][11]=Nz-j-1; //first k-cell fracture occupies
             j=Nz;} // to finish cycle
             else {j++;}
             }
             j=0;
             while (j<=Nz-2)
             {if(XFr[i][5]==Zj[j])
             {XFr[i][10]=Nz-j; //last k-cell fracture occupies
             j=Nz;} // to finish cycle
             else {j++;}
             }
            }
}
void CompleteYFr()
{int j;
            for (int i=0;i<nYFr;i++)
            {
             j=0;
             while (j<=Ny-2)
             {if(YFr[i][1]==Yj[j])
             {YFr[i][8]=j+2;
              YFr[i][9]=j+2; //j cells fracture occupies
              j=Ny; }// to finish cycle
```

```
            else {j++;}
            }
            j=0;
            while (j<=Nx-2)
            {if(YFr[i][0]==Xj[j])
            {YFr[i][6]=j+2; //first i-cell fracture occupies
            j=Nx;} // to finish cycle
            else {j++;}
            }
            j=0;
            while (j<=Nx-2)
            {if(YFr[i][3]==Xj[j])
            {YFr[i][7]=j+1; //last i-cell fracture occupies
            j=Nx;} // to finish cycle
            else {j++;}
            }
            j=0;
            while (j<=Nz-2)
            {if(YFr[i][2]==Zj[j])
            {YFr[i][11]=Nz-j-1;//{YFr[i][10]=j+2; //first k-cell fracture occupies
            j=Nz;    } // to finish cycle
            else {j++;}
            }
            j=0;
            while (j<=Nz-2)
            {if(YFr[i][5]==Zj[j])
            {YFr[i][10]=Nz-j;//{YFr[i][11]=j+1; //last k-cell fracture occupies
            // printf ("k for y fr %d", j);
            j=Nz; } // to finish cycle
            else {j++;}
            }
           }
}
void EQUALS()
{string str = "EQUALS";
ofstream output(str.c_str());
output.flush();
string str1 = "PERMX";
string str2 = "PORO";
output<<str<<endl;
output<<str1<<' '<<PermM<<'/'<<endl;
output<<str2<<' '<<PoroM<<'/'<<endl;
for(int i=0;i<nXFr;i++)
{output<<str1<<' '<<XFr[i][12]<<' '<<XFr[i][6]<<' '<<XFr[i][7]<<' '<<XFr[i][8]<<'
'<<XFr[i][9]<<' '<<XFr[i][10]<<' '<<XFr[i][11]<<'/'<<endl; //box for PermXFr
output<<str2<<' '<<PoroF<<' '<<XFr[i][6]<<' '<<XFr[i][7]<<' '<<XFr[i][8]<<' '<<XFr[i][9]<<'
'<<XFr[i][10]<<' '<<XFr[i][11]<<'/'<<endl;} //box for PoroF
for(int i=0;i<nYFr;i++)
{output<<str1<<' '<<YFr[i][12]<<' '<<YFr[i][6]<<' '<<YFr[i][7]<<' '<<YFr[i][8]<<'
'<<YFr[i][9]<<' '<<YFr[i][10]<<' '<<YFr[i][11]<<'/'<<endl; //box for PermYFr
output<<str2<<' '<<PoroF<<' '<<YFr[i][6]<<' '<<YFr[i][7]<<' '<<YFr[i][8]<<' '<<YFr[i][9]<<'
'<<YFr[i][10]<<' '<<YFr[i][11]<<'/'<<endl;} //box for PoroF
output<<'/';
output.close();
```

```
}
void WELSPECS()
{string str = "WELSPECS";
ofstream output(str.c_str());
output.flush();
string str0 = "I1";
string str1 = "I2";
string str2 = "I3";
string str3 = "I4";
string str4 = "FS1";
string str5 = "S1";
string str6 = "S2";
string str7 = "S3";
string str8 = "1*";
string str9 = "WATER";
output<<str<<endl;
output<<str0<<' '<<str8<<' '<<CompDat[0*2]<<' '<<CompDat[0*2+1]<<' '<<str8<<'
'<<str9<<'/'<<endl;
output<<str1<<' '<<str8<<' '<<CompDat[1*2]<<' '<<CompDat[1*2+1]<<' '<<str8<<'
'<<str9<<'/'<<endl;
output<<str2<<' '<<str8<<' '<<CompDat[2*2]<<' '<<CompDat[2*2+1]<<' '<<str8<<'
'<<str9<<'/'<<endl;
output<<str3<<' '<<str8<<' '<<CompDat[3*2]<<' '<<CompDat[3*2+1]<<' '<<str8<<'
'<<str9<<'/'<<endl;
output<<str4<<' '<<str8<<' '<<CompDat[4*2]<<' '<<CompDat[4*2+1]<<' '<<str8<<'
'<<str9<<'/'<<endl;
output<<str5<<' '<<str8<<' '<<CompDat[5*2]<<' '<<CompDat[5*2+1]<<' '<<str8<<'
'<<str9<<'/'<<endl;
output<<str6<<' '<<str8<<' '<<CompDat[6*2]<<' '<<CompDat[6*2+1]<<' '<<str8<<'
'<<str9<<'/'<<endl;
output<<str7<<' '<<str8<<' '<<CompDat[7*2]<<' '<<CompDat[7*2+1]<<' '<<str8<<'
'<<str9<<'/'<<endl;
output<<'/'<<endl;
output.close();
}
void COMPDAT()
{string str = "COMPDAT";
ofstream output(str.c_str());
output.flush();
string str0 = "I1";
string str1 = "I2";
string str2 = "I3";
string str3 = "I4";
string str4 = "FS1";
string str5 = "S1";
string str6 = "S2";
string str7 = "S3";
string str8 = "1*";
string str9 = "OPEN";
int k1=2; // change if needed
int k2=Nz-1;//   change if needed
output<<str<<endl;
output<<str0<<' '<<CompDat[0*2]<<' '<<CompDat[0*2+1]<<' '<<k1<<' '<<k2<<' '<<str9<<'
'<<str8<<' '<<'1'<<'/'<<endl;
```

```
output<<str1<<' '<<CompDat[1*2]<<' '<<CompDat[1*2+1]<<' '<<k1<<' '<<k2<<' '<<str9<<'
'<<str8<<' '<<'1'<<'/'<<endl;
output<<str2<<' '<<CompDat[2*2]<<' '<<CompDat[2*2+1]<<' '<<k1<<' '<<k2<<' '<<str9<<'
'<<str8<<' '<<'1'<<'/'<<endl;
output<<str3<<' '<<CompDat[3*2]<<' '<<CompDat[3*2+1]<<' '<<k1<<' '<<k2<<' '<<str9<<'
'<<str8<<' '<<'1'<<'/'<<endl;
output<<str4<<' '<<CompDat[4*2]<<' '<<CompDat[4*2+1]<<' '<<k1<<' '<<k2<<' '<<str9<<'
'<<str8<<' '<<'1'<<'/'<<endl;
output<<str5<<' '<<CompDat[5*2]<<' '<<CompDat[5*2+1]<<' '<<k1<<' '<<k2<<' '<<str9<<'
'<<str8<<' '<<'1'<<'/'<<endl;
output<<str6<<' '<<CompDat[6*2]<<' '<<CompDat[6*2+1]<<' '<<k1<<' '<<k2<<' '<<str9<<'
'<<str8<<' '<<'1'<<'/'<<endl;
output<<str7<<' '<<CompDat[7*2]<<' '<<CompDat[7*2+1]<<' '<<k1<<' '<<k2<<' '<<str9<<'
'<<str8<<' '<<'1'<<'/'<<endl;
output<<'/'<<endl;
output.close();
}
void WCONINJ()
{string str = "WCONINJ";
ofstream output(str.c_str());
output.flush();
string str0 = "I1";
string str1 = "I2";
string str2 = "I3";
string str3 = "I4";
string str4 = "WATER OPEN RATE ";
string str5 = " 3* 300 /";
output<<str<<endl;
output<<str0<<' '<<str4<<q_i1<<str5<<endl;
output<<str1<<' '<<str4<<q_i2<<str5<<endl;
output<<str2<<' '<<str4<<q_i3<<str5<<endl;
output<<str3<<' '<<str4<<q_i4<<str5<<endl;
output<<'/'<<endl;
output.close();
}
void WCONPROD()
{string str = "WCONPROD";
ofstream output(str.c_str());
output.flush();
string str0 = "S1";
string str1 = "S2";
string str2 = "S3";
string str3 = "FS1";
string str4 = "OPEN LRAT 3* ";
string str5 = " 1* 50 /";
output<<str<<endl;
output<<str0<<' '<<str4<<q_s1<<str5<<endl;
output<<str1<<' '<<str4<<q_s2<<str5<<endl;
output<<str2<<' '<<str4<<q_s3<<str5<<endl;
output<<str3<<' '<<str4<<q_fs1<<str5<<endl;
output<<'/'<<endl;
output.close();
}
void grid::FillPress()  //reads pressure values from V4.F0005 file and locates them in G(I,J,K)->P
```

```
{ifstream indata; // indata is like cin
 int num; // variable for input value
  string str1="'PRESSURE'";
 string str;
  indata.open("V4.F0005"); // opens the file
   if(!indata) { // file couldn't be opened
      cerr << "Error: file could not be opened" << endl;
   }
  indata >> str;
     while (!indata.eof()) { // keep reading until end-of-file
                if (str==str1)
                {indata >> str;
                indata >> str;
                indata >> str;
                for (int K=1;K<=Nz;K++)
                {for (int J=1;J<=Ny;J++)
                {for (int I=1;I<=Nx;I++)
                {char *a=new char[str.size()+1];
                          a[str.size()]=0;
                          memcpy(a,str.c_str(),str.size());
                          G(I,J,K)->P=atof(a);
                           indata>>str;
                                    }
                }
                }}
                indata>>str;
               }
                indata.close();
}
void FillGraphGeom(grid& MyGrid, GraphV& MyGraph) //fills i, j, k, NEdg, other End; This one
creates vertexes on fracture ends, not only on intersections
{          int Ntemp; int NEdg;
           int v=0; //element of Graph which we are going to fill
           int s; //counter
           int i;
           int n;
for (s=0;s<nXFr;s++) //for each X-fracture
{
           int xi=XFr[s][6];
           int xj1=XFr[s][8];
           int xj2=XFr[s][9];
           int xk1=XFr[s][10];
           int xk2=XFr[s][11];
           Pairs temp[nYFrM*2]; //temporary array; we will put all y-intersections of that fracture
and fracture ends in this array
           Ntemp=0; //number of vertexes on this x-fracture
           MyGraph.verts[v].i=xi;
           MyGraph.verts[v].j=xj1;
           MyGraph.verts[v].k=(int)(xk1+xk2)/2;
           MyGraph.verts[v].NEdg=0;
           temp[Ntemp].vid=v;
           temp[Ntemp].xory=xj1; //added one end of the x-fracture
           Ntemp++;
           v++;
```

```
            MyGraph.verts[v].i=xi;
            MyGraph.verts[v].j=xj2;
            MyGraph.verts[v].k=(int)(xk1+xk2)/2;
            MyGraph.verts[v].NEdg=0;
            temp[Ntemp].vid=v;
            temp[Ntemp].xory=xj2;
            Ntemp++;
            v++;//added other end of the x-fracture
            for (n=0;n<nYFr;n++) //for each Y-fracture
            {int yi1=YFr[n][6];
             int yi2=YFr[n][7];
             int yj=YFr[n][8];
             int yk1=YFr[n][10];
             int yk2=YFr[n][11];
             if((yi1<xi)&&(xi<yi2)&&(xj1<yj)&&(yj<xj2)&&((xk2-yk1)*(yk2-xk1)>=0)) //those 2
fractures are intersecting (but not at the end fracture, because we are adding ends separately)
             {          MyGraph.verts[v].i=xi;
                        MyGraph.verts[v].j=yj;
                        MyGraph.verts[v].k=int((max(xk1,yk1)+min(xk2,yk2))/2);
                        MyGraph.verts[v].NEdg=0;
                        temp[Ntemp].vid=v;
                        temp[Ntemp].xory=yj; //put all y-intersections of that fracture in the temporary
array)
                        Ntemp=Ntemp+1;
                        v++;
             } //end of if
            } //end of cycle -> go to next y-fracture
            //sort temp array and create edges
            if(Ntemp>1)
            {
            sort(temp,temp+Ntemp,MyCompare);
                        //first vertex for this x-fracture
            NEdg=MyGraph.verts[temp[0].vid].NEdg;
            MyGraph.verts[temp[0].vid].OtherEnd[NEdg]=temp[1].vid;
            MyGraph.verts[temp[0].vid].FrNum[NEdg]=s; //this edge is on x-fracture #s
            MyGraph.verts[temp[0].vid].NEdg=MyGraph.verts[temp[0].vid].NEdg+1;
            for(i=1;i<Ntemp-1;i++) //this will run only if Ntemp>=3
            {NEdg=MyGraph.verts[temp[i].vid].NEdg;
             MyGraph.verts[temp[i].vid].OtherEnd[NEdg]=temp[i-1].vid;
             MyGraph.verts[temp[i].vid].FrNum[NEdg]=s; //this edge is on x-fracture #s
             MyGraph.verts[temp[i].vid].NEdg=MyGraph.verts[temp[i].vid].NEdg+1; //added edge
which goes towards 'left' vertex
             NEdg=MyGraph.verts[temp[i].vid].NEdg;
             MyGraph.verts[temp[i].vid].OtherEnd[NEdg]=temp[i+1].vid;
             MyGraph.verts[temp[i].vid].FrNum[NEdg]=s; //this edge is on x-fracture #s
             MyGraph.verts[temp[i].vid].NEdg=MyGraph.verts[temp[i].vid].NEdg+1; //added edge
which goes towards 'right' vertex
            } //for all 'middle/ points on that x-fracture
            //last -Ntemp-1- vertex for this x-fracture
            NEdg=MyGraph.verts[temp[Ntemp-1].vid].NEdg;
            MyGraph.verts[temp[Ntemp-1].vid].OtherEnd[NEdg]=temp[Ntemp-2].vid;
            MyGraph.verts[temp[Ntemp-1].vid].FrNum[NEdg]=s; //this edge is on x-fracture #s
            MyGraph.verts[temp[Ntemp-1].vid].NEdg=NEdg+1;
            } //end of if Ntemp>1 cycle
```

```
} //end of cycle -> go to next x-fracture
//now add ends of y-fractures
for (s=0;s<nYFr;s++) //for each Y-fracture
{	int yi1=YFr[s][6];
	int yi2=YFr[s][7];
	int yj=YFr[s][8];
	int yk1=YFr[s][10];
	int yk2=YFr[s][11];
	MyGraph.verts[v].i=yi1;
	MyGraph.verts[v].j=yj;
	MyGraph.verts[v].k=(int)(yk1+yk2)/2;
	MyGraph.verts[v].NEdg=0;
	v++; //added one end of the y-fracture
	MyGraph.verts[v].i=yi2;
	MyGraph.verts[v].j=yj;
	MyGraph.verts[v].k=(int)(yk1+yk2)/2;
	MyGraph.verts[v].NEdg=0;
	v++; //added other end of the y-fracture
} //done for all y-fractures
NumVert=v;
//--vertexes are created; edges along x-fractures - also; now add edges along all y-fractures ---------
for (s=0;s<nYFr;s++) //for each Y-fracture
{	int yj=YFr[s][8];
	Pairs temp[nXFrM*2]; //temporary array; we will put all vertexes laying on this y-
fracture to this array
	Ntemp=0; //number vertexes on this y-fracture
	for (v=0;v<NumVert;v++) //for each vertex
	{
	if((MyGraph.verts[v].j==yj)&&(MyGraph.verts[v].i>=YFr[s][6])&&(MyGraph.verts[v].i
<=YFr[s][7])&&(MyGraph.verts[v].k>=YFr[s][10])&&(MyGraph.verts[v].k<=YFr[s][11]))
		//vertex belongs to that y-fracture
	{	temp[Ntemp].vid=v;
		temp[Ntemp].xory=MyGraph.verts[v].i; //put all vertexes on that y-fracture in
the temporary array
		Ntemp=Ntemp+1;
	 } //end of if
	} //end of v-cycle
	//sort temp array and create edges
	if(Ntemp>1)
	{
	sort(temp,temp+Ntemp,MyCompare);
			//first vertex for this y-fracture
	NEdg=MyGraph.verts[temp[0].vid].NEdg;
	MyGraph.verts[temp[0].vid].OtherEnd[NEdg]=temp[1].vid;
	MyGraph.verts[temp[0].vid].FrNum[NEdg]=nXFr+s; // this edge belongs to y-fracture #s
	MyGraph.verts[temp[0].vid].NEdg=MyGraph.verts[temp[0].vid].NEdg+1;
	for(i=1;i<Ntemp-1;i++) //this will run only if Ntemp>=3
	{NEdg=MyGraph.verts[temp[i].vid].NEdg;
	 MyGraph.verts[temp[i].vid].OtherEnd[NEdg]=temp[i-1].vid;
	 MyGraph.verts[temp[i].vid].FrNum[NEdg]=nXFr+s; // this edge belongs to y-fracture #s
	 MyGraph.verts[temp[i].vid].NEdg=MyGraph.verts[temp[i].vid].NEdg+1; //added edge
which goes towards 'left' vertex
	 NEdg=MyGraph.verts[temp[i].vid].NEdg;
	 MyGraph.verts[temp[i].vid].OtherEnd[NEdg]=temp[i+1].vid;
```

```
            MyGraph.verts[temp[i].vid].FrNum[NEdg]=nXFr+s; // this edge belongs to y-fracture #s
            MyGraph.verts[temp[i].vid].NEdg=MyGraph.verts[temp[i].vid].NEdg+1; //added edge
which goes towards 'right' vertex
            } //for all 'middle/ points on that x-fracture
           //last -Ntemp-1- vertex for this y-fracture
           NEdg=MyGraph.verts[temp[Ntemp-1].vid].NEdg;
           MyGraph.verts[temp[Ntemp-1].vid].OtherEnd[NEdg]=temp[Ntemp-2].vid;
           MyGraph.verts[temp[Ntemp-1].vid].FrNum[NEdg]=nXFr+s;// this edge belongs to y-
fracture #s
           MyGraph.verts[temp[Ntemp-1].vid].NEdg=MyGraph.verts[temp[Ntemp-
1].vid].NEdg+1;
           } //end of if Ntemp>1 cycle
} //end of cycle -> go to next y-fracture
}
void FillGraphProps(grid& MyGrid, GraphV& MyGraph) //fills P, pot, dP, L
{int i,j,k;
 double x1,y1,z1,x2,y2,z2,fw,h;
           for (int v=0;v<NumVert;v++)
           {k=MyGraph.verts[v].k;
            z1= (Zj[Nz-k]+Zj[Nz-k-1])/2; //z coord of v

MyGraph.verts[v].P=MyGrid.G(MyGraph.verts[v].i,MyGraph.verts[v].j,MyGraph.verts[v].k)->P;
//set pressure
            MyGraph.verts[v].pot=MyGraph.verts[v].P+0.00001*rho*g*z1*C; //set potential
           }
           printf("NumVert %d/n", NumVert);
           for (int v=0; v<NumVert;v++)
           {i=MyGraph.verts[v].i;
            j=MyGraph.verts[v].j;
            k=MyGraph.verts[v].k;
            x1=(Xj[i-1]+Xj[i-2])/2;
            y1=(Yj[j-1]+Yj[j-2])/2;
            z1=(Zj[Nz-k]+Zj[Nz-k-1])/2; //coords of v
            for (int s=0;s<MyGraph.verts[v].NEdg;s++)
           {MyGraph.verts[v].dP[s]=MyGraph.verts[v].pot-
MyGraph.verts[MyGraph.verts[v].OtherEnd[s]].pot;
            i=MyGraph.verts[MyGraph.verts[v].OtherEnd[s]].i;
            j=MyGraph.verts[MyGraph.verts[v].OtherEnd[s]].j;
            k=MyGraph.verts[MyGraph.verts[v].OtherEnd[s]].k;
            x2=(Xj[i-1]+Xj[i-2])/2;
            y2=(Yj[j-1]+Yj[j-2])/2;
            z2=(Zj[Nz-k]+Zj[Nz-k-1])/2; //coords of MyGraph.verts[v].OtherEnd[s]
           MyGraph.verts[v].L[s]=sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2));//sqrt((x1-x2)*(x1-
x2)+(y1-y2)*(y1-y2)+(z1-z2)*(z1-z2));
           //MyGraph.verts[v].H[s]=sqrt((z1-z2)*(z1-z2));
           //MyGraph.verts[v].V[s]=0.0000001*PermF*MyGraph.verts[v].dP[s]/(mu*MyGraph.ver
ts[v].L[s]);
           if (MyGraph.verts[v].FrNum[s]<nXFr) //it's an x-fracture #FrNum[s]
           {PermF=XFr[MyGraph.verts[v].FrNum[s]][12];
            fw=XFr[MyGraph.verts[v].FrNum[s]][13];
            h=XFr[MyGraph.verts[v].FrNum[s]][5]-XFr[MyGraph.verts[v].FrNum[s]][2]; //// height
of the fracture
           }
```

```
        if (nXFr<=MyGraph.verts[v].FrNum[s] && MyGraph.verts[v].FrNum[s]<nXFr+nYFr)
//it's an y-fracture #FrNum[s]-nXFr
        {PermF=YFr[MyGraph.verts[v].FrNum[s]-nXFr][12];
         fw=YFr[MyGraph.verts[v].FrNum[s]-nXFr][13];
         h=YFr[MyGraph.verts[v].FrNum[s]-nXFr][5]-YFr[MyGraph.verts[v].FrNum[s]-
nXFr][2]; //// h
        }
        MyGraph.verts[v].V[s]=0.0000001*PermF*MyGraph.verts[v].dP[s]/(mu*MyGraph.verts
[v].L[s]);
        MyGraph.verts[v].Prob[s]=MyGraph.verts[v].V[s]*fw*h;
        if (MyGraph.verts[v].j!=MyGraph.verts[MyGraph.verts[v].OtherEnd[s]].j &&
MyGraph.verts[v].i!=MyGraph.verts[MyGraph.verts[v].OtherEnd[s]].i)
        {printf("no i no j equal, v,s %d  %d ",v,s);}
         }
        }
}
void GraphAddMatr(grid& MyGrid, GraphV& MyGraph) //NM, OtherEndM,
LM,dpM,VM,ProbM
{int i,j,k;
 double x1,y1,z1,x2,y2,z2,fw;
        for (int v=0;v<NumVert;v++)
        {MyGraph.verts[v].NM=0; //number of matrix connections
         i=MyGraph.verts[v].i;
         j=MyGraph.verts[v].j;
         x1=(Xj[i-1]+Xj[i-2])/2;
         y1=(Yj[j-1]+Yj[j-2])/2;
                for (int s=0;s<NumVert;s++)
                {if (MyGraph.verts[v].NM>=nM){s=NumVert; printf ("too many.. decrease
Rad");}
                else{
                i=MyGraph.verts[s].i;
         j=MyGraph.verts[s].j;
         x2=(Xj[i-1]+Xj[i-2])/2;
         y2=(Yj[j-1]+Yj[j-2])/2;
           if(((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2))<Rad*Rad) //s is within Rad distance from v
                {//check if s and v don't belong to the same fracture
                        int belong=0;
                        for (int se=0;se<MyGraph.verts[v].NEdg; se++) //for all edges of v
                        {for (int ve=0;ve<MyGraph.verts[s].NEdg; ve++) //for all edges of s
                        {if (MyGraph.verts[v].FrNum[ve]==MyGraph.verts[s].FrNum[se])
belong=1;
                        }
                        }
                if(belong==0)      //s does no belong to the same fracture as v
                {MyGraph.verts[v].OtherEndM[MyGraph.verts[v].NM]=s;
                MyGraph.verts[v].LM[MyGraph.verts[v].NM]=sqrt((x1-x2)*(x1-x2)+(y1-
y2)*(y1-y2));
                MyGraph.verts[v].dPM[MyGraph.verts[v].NM]=MyGraph.verts[v].pot-
MyGraph.verts[s].pot;

        MyGraph.verts[v].VM[MyGraph.verts[v].NM]=MF1*0.0000001*PermM*MyGraph.vert
s[v].dPM[MyGraph.verts[v].NM]/(mu*MyGraph.verts[v].LM[MyGraph.verts[v].NM]); //velosity
of flow through fracture
```

```
            MyGraph.verts[v].ProbM[MyGraph.verts[v].NM]=MF2*MyGraph.verts[v].VM[MyGrap
h.verts[v].NM]; //MF2 should reflect area in a way
                        MyGraph.verts[v].NM++;
                        }}
                        }}
            }
}
void FillInjProdInd(GraphV& MyGraph) //InjProdInd will have number of vertexes
correspondung to wells I1, I2, ...
{for (int v=0;v<NumVert;v++)
{for (int j=0;j<8;j++)
{
if(MyGraph.verts[v].i==CompDat[j*2] && MyGraph.verts[v].j==CompDat[j*2+1] )
InjProdInd[j]=v;
}
 }
}
void EmptyProduction()
{
            for (int i=0;i<4;i++)
             for(int j=0;j<(ProdTime*10);j++)
                        Production[i][j]=0;
}
Cell* grid::G(int I, int J, int K)
{return (&cells[(Nx)*(Ny)*K+(Nx)*J+I]);
}
void particle::Run(GraphV& MyGraph) //this one includes flow through matrix
{//string str = "production.txt";
            string str;
            double t_d; //time in days, rounded to 0.1 days
            int flow =-1; //0 - fracture, 1 - matrix
            int NPosEdg;
ofstream output(str.c_str(), std::ios::app);
output.flush();
Pairs temp[105];
int Ntemp;
double r,r1;
int ExitCode=-1;
while(ExitCode<0)
{Ntemp=0;
NPosEdg=0;
 temp[Ntemp].dob=0;
 for (int s=0;s<MyGraph.verts[vid].NEdg;s++)
 { if (MyGraph.verts[vid].Prob[s]>0)
 {           Ntemp++;
  temp[Ntemp].vid=s; //vertex, where we will move if random value selects this choise will be -
MyGraph.verts[vid].OtherEnd[temp[s+1].vid]
  temp[Ntemp].dob=temp[Ntemp-1].dob+MyGraph.verts[vid].Prob[s];
 }}//end if , end for
 NPosEdg=Ntemp; //number of fracture edges which were added to temp
 for (int s=0;s<MyGraph.verts[vid].NM;s++)
 { if (MyGraph.verts[vid].ProbM[s]>0)
 {           Ntemp++;
```

```
  temp[Ntemp].vid=s; //vertex, where we will move if random value selects this choise will be -
MyGraph.verts[vid].OtherEndM[temp[s+1].vid]
  temp[Ntemp].dob=temp[Ntemp-1].dob+MyGraph.verts[vid].ProbM[s];
   }
 } //filled temp array
 if (Ntemp==0){ExitCode=vid; } //particle is at the 'dead end' - wont go anywhere from there
 else{
  r1=(double)rand()/RAND_MAX;
 r=r1*temp[Ntemp].dob; //random value to define in which direction to go now;
 //printf("r1= %f",r1);
 for(int s=0;s<Ntemp;s++)
 {if (r>=temp[s].dob && r<=temp[s+1].dob) //means probability 'shows' edge#s, so new vertex is
OtherEnd[temp[s+1].vid]
 { // printf("s= %d", s);
            if (s<NPosEdg) flow=0;
            else flow=1;
            double Z=Norm(0,1);
            if (flow==0) //fracture flow
            {           double L=MyGraph.verts[vid].L[temp[s+1].vid];
                        double V=MyGraph.verts[vid].V[temp[s+1].vid];
                        t=t+(-(double)Z*sqrt(D/2)/V+sqrt((double)Z*Z*D/(2*V*V)+(double)L/V))*(-
(double)Z*sqrt(D/2)/V+sqrt((double)Z*Z*D/(2*V*V)+(double)L/V));
                        vid=MyGraph.verts[vid].OtherEnd[temp[s+1].vid];}
            if (flow==1)//matrix flow
            {
t=t+(double)MyGraph.verts[vid].LM[temp[s+1].vid]/MyGraph.verts[vid].VM[temp[s+1].vid];
//no diffusion here
             vid=MyGraph.verts[vid].OtherEndM[temp[s+1].vid];}
    //----now lets check if it reached any well
  for (int j=0;j<8;j++)
  {if (InjProdInd[j]==vid)
  {ExitCode=vid;
  t_d=(double)t/(3600*24);
  t_d=floor(t_d*10+0.5)/10; //same as rounding to 1 decimal: round(t_d,1)
   if(t_d<ProdTime)
  {Production[j-4][int(t_d*10)]++;
   ParticlesProduced++;} //column #j-4 is for (j-4)th production well; add one more particle
produced in this time interval by this well;
   }
  } //end of checking if it reached any well
 } //end of if cycle
 } //end of for cycle
 }
} //end of while cycle
output.close();
}
//math functions
double MyRand(double min, double max)
{double r=(double)rand()/RAND_MAX; //random between 0 and 1
if(max<min) printf("max<min");
else
 return(min+r*(max-min));
}
double Norm(double m, double d)
```

```
{double r1=(double)rand()/RAND_MAX; // r1, r2 in [0...1)
 double r2=(double)rand()/RAND_MAX;
            double Z=sqrt(-2*log(r1)/log(2.718281828))*cos(2*3.14159265358979*r2); //Z - N(0,1)
            Z=min(Z,4);
            Z=max(Z,-4);
            return(m+d*Z);
}
double MyRound(double value, int dec)
{//int poften=1;
double poften=1;
 for(int i=0;i<dec;i++)poften=poften*10;
            return(floor(value*poften+0.5)/poften);
// int tm=(int)floor(value*poften+0.5);
 //return((double)tm/poften);
 // return (double)tm/poften;
}
bool MyCompare(Pairs i, Pairs j)
{return(i.xory<j.xory);}
//output
void ToFile(int NumLin, int NumCol, char FileName){
            string str;
            stringstream ss;
            ss << FileName;
            ss >> str;
            str+=".txt";
            ofstream output(str.c_str());
            output.flush();
            for (int lin=0;lin<NumLin;lin++){   // export
            for (int col=0;col<NumCol;col++)    // without edges
            {output<<Production[col][lin]<<' ';}
            //output<<Xj[lin]<<' ';
            output<<endl;
            }
            output.close();
}
void XFracturesAsPolygons(int NumFract, char FileName){
            string str;
            stringstream ss;
            ss << FileName;
            ss >> str;
            str+=".txt";
            ofstream output(str.c_str());
            output.flush();
            for (int lin=0;lin<NumFract;lin++){
   output<<XFr[lin][0]<<' '<<XFr[lin][1]<<' '<<lin<<' '<<XFr[lin][2]<<endl;
            output<<XFr[lin][0]<<' '<<XFr[lin][1]<<' '<<lin<<' '<<XFr[lin][5]<<endl;
            output<<XFr[lin][0]<<' '<<XFr[lin][4]<<' '<<lin<<' '<<XFr[lin][5]<<endl;
            output<<XFr[lin][0]<<' '<<XFr[lin][4]<<' '<<lin<<' '<<XFr[lin][2]<<endl;
            output<<XFr[lin][0]<<' '<<XFr[lin][1]<<' '<<lin<<' '<<XFr[lin][2]<<endl;
            }
            output.close();
}
void YFracturesAsPolygons(int NumFract, char FileName){
            string str;
```

```
            stringstream ss;
            ss << FileName;
            ss >> str;
            str+=".txt";
            ofstream output(str.c_str());
            output.flush();
            for (int lin=0;lin<NumFract;lin++){
   output<<YFr[lin][0]<<' '<<YFr[lin][1]<<' '<<lin<<' '<<YFr[lin][2]<<endl;
            output<<YFr[lin][0]<<' '<<YFr[lin][1]<<' '<<lin<<' '<<YFr[lin][5]<<endl;
            output<<YFr[lin][3]<<' '<<YFr[lin][1]<<' '<<lin<<' '<<YFr[lin][5]<<endl;
            output<<YFr[lin][3]<<' '<<YFr[lin][1]<<' '<<lin<<' '<<YFr[lin][2]<<endl;
            output<<YFr[lin][0]<<' '<<YFr[lin][1]<<' '<<lin<<' '<<YFr[lin][2]<<endl;
            }
            output.close();
}
void ParametersToFile()
{string str = "Parameters.txt";
ofstream output(str.c_str());
output.flush();
output<<(int)Dt1/dt<<' '<<(int)Dt2/dt<<' '<<(int)Dt3/dt<<' '<<(int)Dt4/dt<<' '<<nXFr+nYFr<<' ';
output<<theta<<' '<<(double)(Yspm+YspM)/2<<' '<<(double)(Xspm+XspM)/2<<' '<<D<<'
'<<PermF<<' '<<PermM<<' '<<PoroM<<' '<<ParticlesProduced<<' ';
output<<fwx<<' '<<fwy<<' '<<PermXfr<<' '<<PermYfr<<' '<<nXFr<<' '<<nYFr;
output.close();
}
```

Appendix C (ECLIPSE file used for the RWPT modeling)

```
RUNSPEC  ======================
INCLUDE
'DIMENS' /
WATER
WELLDIMS
8 15 1 8/
START
1 'JAN' 1986 /
FMTOUT
METRIC
GRID   =========================
INCLUDE
'DX' /
INCLUDE
'DY' /
INCLUDE
'DZ' /
INCLUDE
'TOPS' /
INCLUDE
'MAPAXES' /
INCLUDE
'EQUALS' /
COPY
PERMX PERMY/
```

```
PERMX PERMZ/
/
MINPV
0.000000001/
GRIDFILE
0 1 /
RPTGRID
'DX' 'DY' 'DZ' 'PERMX' 'PORO' 'TOPS' 'PORV'/
INIT
EDIT    ==================
PROPS   ==================
ROCKOPTS
1* 1* ROCKNUM /
ROCK
400  5.787E-005 /
PVTW
215 1.0132 3.9795E-005 0.39851 0 /
DENSITY
1020.3 1020.3 0.81172 /
RPTPROPS
/
REGIONS  ==================
SOLUTION ==================
EQUIL
1300 128.54 1*  0 1*  0 0 0 0 /
RPTRST
BASIC=3 FLOWS /
RPTSOL
RESTART=2 FIP /
SUMMARY  ==================
SCHEDULE ==================
TSTEP
1/
RPTSCHED
'PRES' /
INCLUDE
'WELSPECS' /
INCLUDE
'COMPDAT' /
INCLUDE
'WCONINJ' /
INCLUDE
'WCONPROD' /
TSTEP
1/
END
```