The Dissertation Committee for Sayantan Bhowmik Certifies that this is the approved version of the following dissertation:

PARTICLE TRACKING PROXIES FOR PREDICTION OF CO_2 PLUME MIGRATION WITHIN A MODEL SELECTION FRAMEWORK

Committee:

Sanjay Srinivasan, Supervisor

Steven L. Bryant, Co-Supervisor

Larry W. Lake

Matthew Balhoff

Seyyed A. Hosseini

PARTICLE TRACKING PROXIES FOR PREDICTION OF CO_2 PLUME MIGRATION WITHIN A MODEL SELECTION FRAMEWORK

by

Sayantan Bhowmik, B.Tech.; M.S.E.

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

May 2014

Acknowledgements

I would like to express my sincerest gratitude to Dr. Sanjay Srinivasan for supervising my work these past 6 years. He has been a wonderful mentor, guide and friend, and instilled in me the desire to be an excellent researcher and a sincere person. His advice and patience in introducing me to the world of geostatistics has been invaluable, and has helped me work through complicated problems very easily. The discussions I have had with him have always encouraged me to look at problems from different points of view and oftentimes from a completely unexpected direction. Without his constant support and encouragement, this journey would have been an especially arduous one.

I would also like to express my thanks to my co-supervisor, Dr. Steven Bryant, whose constant reminder to look at the big picture encouraged me to often take a step back and examine how every aspect of my work aligned with the larger problem I was trying to address. The discussions with Dr. Srinivasan and Dr. Bryant were some of the most stimulating exchange of ideas I have had at the university. I would also like to express my gratitude to my committee members, Dr. Larry Lake, Dr. Matthew Balhoff and Dr. Seyyed Hosseini, whose support and encouragement allowed me to make rapid progress on my dissertation.

Special thanks are also due to Jin Lee for all her help over the duration of this work, and to Roger Terzian and Joanna Castillo, for their help in matters related to all the software that have been used in my research.

Finally, I would like to thank my research colleagues and friends, who have made graduate life so much easier with their friendship and support: Ankesh, Arti, Azor,

Brandon, Cesar, Deji, DJ, Donovon, Harpreet, Henry, Hoonyoung, James, John, Juliana, Krupa, Kwangjin, Mayuri, Morteza, Namdi, Nitish, Selin, Travis and Youngkim.

I am grateful to the sponsors of Geologic CO_2 Storage Industrial Associates Project at The University of Texas at Austin: BP, Chevron, ConocoPhillips, ExxonMobil, Foundation CMG, Halliburton/Landmark Graphics, Luminant, Shell, Statoil and USGS, for making this work possible

This material is partially based on work supported by the Department of Energy under Award Number DE-DE-FE0004962.

PARTICLE TRACKING PROXIES FOR PREDICTION OF CO_2 PLUME MIGRATION WITHIN A MODEL SELECTION FRAMEWORK

Sayantan Bhowmik, Ph.D.

The University of Texas at Austin, 2014

Supervisor: Sanjay Srinivasan

Co-supervisor: Steven Bryant

Geologic sequestration of CO_2 in deep saline aquifers has been studied extensively over the past two decades as a viable method of reducing anthropological carbon emissions. The monitoring and prediction of the movement of injected CO_2 is important for assessing containment of the gas within the storage volume, and taking corrective measures if required. Given the uncertainty in geologic architecture of the storage aquifers, it is reasonable to depict our prior knowledge of the project area using a vast suite of aquifer models. Simulating such a large number of models using traditional numerical flow simulators to evaluate uncertainty is computationally expensive. A novel stochastic workflow for characterizing the plume migration, based on a model selection algorithm developed by Mantilla in 2011, has been implemented. The approach includes four main steps: (1) assessing the connectivity/dynamic characteristics of a large prior ensemble of models using proxies; (2) model clustering using the principle component analysis or multidimensional scaling coupled with the k-mean clustering approach; (3) model selection using the Bayes' rule on the reduced model space, and (4) model expansion using an ensemble pattern-based matching scheme.

In this dissertation, two proxies have been developed based on particle tracking in order to assess the flow connectivity of models in the initial set. The proxies serve as fast approximations of finite-difference flow simulation models, and are meant to provide rapid estimations of connectivity of the aquifer models. Modifications have also been implemented within the model selection workflow to accommodate the particular problem of application to a carbon sequestration project.

The applicability of the proxies is tested both on synthetic models and real field case studies. It is demonstrated that the first proxy captures areal migration to a reasonable extent, while failing to adequately capture vertical buoyancy-driven flow of CO_2. This limitation of the proxy is addressed in the second proxy, and its applicability is demonstrated not only in capturing horizontal migration but also in buoyancy-driven flow. Both proxies are tested both as standalone approximations of numerical simulation and within the larger model selection framework.

Table of Contents

List of Tables

List of Figures

Chapter 1 : Introduction

Anthropogenic climate change caused by emission of greenhouse gases has been of increasing concern over the past few decades. The capture and removal of CO_2 (the most abundant greenhouse gas after water vapor) at large-scale primarily from coal-fired power plants is one of the many methods being considered and tested to mitigate the climate-change effects of greenhouse gas emissions. The process consists of three major steps (http://www.epa.gov/climatechange/ccs/index.html): capture from industrial sources, transport to disposal sites, and injection into subsurface formations primarily saline aquifers. The injected CO_2 is retained in the subsurface due to a combination of physical processes: trapping under cap rock or other structural traps, dissolution in formation brine, capillary trapping and mineralization (Kumar et.al 2005, IPCC special report 2005). Over the course of a sequestration project, it is necessary to be able to monitor and predict the movement of the injected CO_2 plume, so as to ensure the containment of the CO_2 within the storage volume. Currently, this is primarily achieved using time-lapse seismic monitoring and/or satellite measurements of surface deflection. Both these processes, however, only give us a snapshot of the current position of the plume and hence need to be integrated with a prediction scheme to enable proper monitoring of the plume.

1.1. Problem Description

To make the process of monitoring of subsurface CO_2 plume migration robust, there is need for integration of data from remote sensing methods to accurately find the current location of the plume and a history-matching / prediction framework which can be sequentially updated using the remote sensing data. This would reduce the frequency

of expensive monitoring techniques and make the process of monitoring more efficient, especially during closure and post-closure stages of the project.

The objective of history-matching is to create a static model of the reservoir which, when evaluated using a forward model, will yield response similar to observed data. However, the data which form part of reservoir models (reservoir structure, petrophysical properties, geologic description, fluid properties etc.) are subject to a lot of uncertainty, and hence the particular static model which satisfies the history-matching objective is not unique. There is also no linear relation between the static properties (like permeability distribution) and the dynamic response of the reservoir, making the problem of calibrating static properties to dynamic data non-trivial. This is thus a non-linear, inversion problem, with multiple non-unique solutions, which also implies that predictions made using these models will also have an associated uncertainty. Further complications during this inversion process arise due to the existence of static geologic properties at multiple scales, with the dynamic response being a combination of the effect at each scale of heterogeneity. One of the methods to model this uncertainty taking into consideration, the multiscale nature of reservoir heterogeneity, is through the use of multiple, multiscale models. History matching then becomes a process of calibrating static reservoir properties, in multiple models and at multiple scales, to dynamic information. This process becomes more challenging in the case of CO_2 sequestration due to the scarcity of data (dynamic data is only available at injection wells and there are no producers) and due to the much larger time scale of sequestration projects compared to oil and gas field operations.

Uncertainty assessment is possible using a large suite of reservoir models, however, the use of a large suite of models introduces an additional level of complexity to the problem: the need for an efficient method of assessment of the models without a

high computational overhead. Hence, there is need to develop alternative forward models that can provide quick assessment of reservoir connectivity at a fraction of the computation cost of numerical simulators.

1.2. RESEARCH OBJECTIVES

The primary objective of this dissertation is to implement a fast proxy within the model-selection framework introduced by Mantilla (2011), in order to assess the uncertainty in predicting the plume migration path during CO_2 sequestration.. This requires the adaptation of the model selection workflow to meet the requirements of a carbon sequestration project. *It is hypothesized that efficient forward models can be developed for simulating the flow of CO_2 in an aquifer at a fraction of the computational cost of a numerical simulator, and can be implemented within the model selection framework to predict future plume migration.* The larger objective is divided into the following parts:

- **Development of fast-transfer functions**: The dominant physical processes at play during CO_2 migration need to be incorporated into a fast transfer function to approximate the migration of CO_2 in the aquifer. In this dissertation, it is hypothesized that a fast transfer function based on random walker particle tracking processes can be developed to accurately represent the physics of plume displacement. Such proxies are currently used in solute transport problems (Tompson and Gelhar 1990, Quinodoz and Valocchi, 1993). This would allow rapid screening of a large number of reservoir models that might be representative of the prior uncertainty.

- **Validation of fast-transfer functions within model selection framework:** The developed fast transfer functions will need to be integrated within a model selection algorithm and tested on synthetic and real field cases to ensure the validity of the development.
- **Development of a software suite to implement the algorithm:** There are a large number of sub-processes that constitute the model selection algorithm. These will need to be implemented in a robust software module such that it can be used by a lay user. This will make the current work more accessible and enable easy implementation to real field cases.

1.3. DISSERTATION OUTLINE

The rest of this dissertation will describe the model selection algorithm and the development of two particle-tracking proxies for rapid evaluation of reservoir connectivity. Chapter 2 will discuss the existing body of work pertaining to monitoring of geologic carbon sequestration processes, as well as established methods of history matching pertaining to multiple models and the use of fast-transfer proxies for numerical simulation prevalent in the energy industry. Chapter 3 will briefly describe the model selection algorithm as applied to the case of carbon sequestration. Chapter 4 will describe the development of the first of two particle-tracking proxies for use within the model selection algorithm. Test cases, both synthetic and real field examples, will also be presented. In Chapter 5, we will layout the development of the second particle tracking proxy, together with test cases. It will also show the application of the new proxy to overcome some challenges faced by the old proxy. Chapter 6 will outline the development of the software module for easy implementation of the entire work, with

additional information available in the appendix. Chapter 7 will describe some additional applications of the model selection algorithm to some specific problems encountered during sequestration. The final chapter will discuss the primary findings of this work and make recommendations for future work.

Chapter 2 : Literature Review

In the context of geologic carbon sequestration, it is of primary importance to both operators and regulators to be able to ensure containment of the injected CO_2 within the storage volume. Achieving this purpose requires a two-step process: monitoring the current location of the CO_2 plume and predicting the future migration of the plume. In this chapter, we will explore some of the current monitoring techniques available to operators to track the migration of the injected CO_2. Since we have implemented a method for making probabilistic predictions of plume migration using a set of reservoir models reflecting the observed injection data, we will also discuss come existing methods of history-matching. Finally, given that the major focus of our work was the development of proxies to estimate reservoir connectivity, we will discuss some other proxies for rapid estimation of fluid migration.

2.1. MONITORING CO_2 PLUME MIGRATION

The process of monitoring the migration of injected CO_2 generally involves a combination of different detection and prediction mechanisms. This combination of technologies would allow operators to take remedial steps if they detect the possibility of anomalous migratory behavior of the plume. Benson et.al. (2004) described such an approach to monitoring by dividing the implementation into four phases: pre-injection, injection, post-injection and closure. The pre-operational stage is composed of characterization and assessment of the storage volume and the development of the site by deploying necessary infrastructure and facilities, and drilling of injection wells (Implementation of Directive 2009/31/EC on the Geological Storage of Carbon Dioxide: Guidance Document 1, European Commission). Operations phase is when the operator

starts injection of CO_2 into the ground. The closure phase starts when the operator has either met the pre-determined storage requirements for the site, or has decided to stop injection. Post-closure phase starts after the operator has ensured all required closure monitoring, plugging and abandonment of wells and handing over of responsibility to a competent authority. At each of these stages, there can be multiple monitoring requirements based on the objective of the operator and the needs for the site.. The details of this approach are described below and shown in Table 2.1.

1. Pre-injection Monitoring: These operations could include exploratory subsurface measurements like 3D seismic and gravity surveys, well level surveys like log data acquisition, pressure tests in existing wells, analyzing injection data from previous wells, as well as further drilling and injection tests if feasible. These data can then be used for detailed characterization of the storage volume, and also making decisions about future feasibility and requirements for remote monitoring techniques.
2. Injection Monitoring: during this phase, the subsurface measurements like seismic and gravity would be continued, and there would be new rate and pressure data from injection and monitoring wells. There is also possibility of remote satellite measurements of ground deformation (Goldstein et.al. 1993, Onuma and Ohkawa 2009) to possibly estimate the movement of the CO_2 plume. These data can be used to test the models from the pre-operational phase and update them appropriately. These updated models can then be used to make predictions of future migration and compared to additional field data over time. This combination of monitoring and modeling approaches would gradually reduce the monitoring requirement over time as the aquifer models become more and more robust.
3. Closure and post-closure monitoring: the activities during both these phases would be similar, with seismic/gravity surveys at reduced intervals. If there are monitoring

wells present, it would also be possible to get pressure data to match to model predictions. Unless there is a drastic offset between monitoring results and model predictions, the monitoring surveys would be only be required at very infrequent intervals during these stages.

	Monitoring methods	Aquifer modeling
Pre-operational monitoring	Well logs; Wellhead pressure/rate; Seismic survey; Formation pressures; electromagnetic/gravity survey	Creation of models for the aquifer using prior information, if available, and current surveys
Operational monitoring	Wellhead pressure/rate; Microseismicity; Seismic surveys; Gravity survey	Aquifer models created in the previous stage simulated and compared with the data recorded at this stage, to validate and/or update models
Closure monitoring	Seismic survey; Gravity survey; Electromagnetic survey	Simulated seismic response based on predictions compared to field seismic surveys to ensure confinement; simulations can also be used for risk assessment

Table 2.1. Monitoring stages of a sequestration project (adapted from Benson et.al. 2004)

2.2. HISTORY MATCHING: AN OVERVIEW

In order for the aquifer modeling during the various phases of operation to be continuously updated using the available data, it is necessary to implement efficient schemes for data assimilation and uncertainty quantification. Because dynamic injection and monitoring data is likely to be readily available, integration of such data within a

history-matching framework is necessary. The history matched process can be followed by prediction of future migration of CO_2 and comparison to monitoring surveys. In this section, we take a brief look at different history-matching workflows that have been proposed in the literature and can be used for this purpose.

History matching, in its most basic form, can be described as the process of creating representations (models) of the subsurface region under study such that forward modeling results of these representations are a close match to observed data. The underlying assumption in this process is that if the modeled results match the observed data, the model is deemed a close approximation to the actual subsurface reservoir and can thus be used to reliably predict future performance [see for example, Review of Inverse methods (Zhou et.al. 2013), Manual history matching (Wallen et.al 1968, Mann and Johnson 1970, Coats et.al. 1970), Gradient-based methods (Lee et al 1986, Zhang et al 2003), Simulated annealing (Deutsch and Journel, 1994), Gradual deformation (Hu 2000, Le Ravalec 2002)]. The problem with this assumption, however, is that there could be multiple models that meet this criterion, thus making the result of history-matching non-unique. Further, given this non-uniqueness of the solution, it would be erroneous to make predictions about the future performance using *any one* of the models. One possible solution to this process is to adopt methods for multi-model history matching whereby multiple 'matched' models are created, thus honoring the non-uniqueness of the entire process. At the same time, predictions made by multiple models can be used as probability estimates of future performance. There has been extensive work on this problem, with various methods suggested for generating an ensemble of history-matched models conditioned to static and dynamic data. In this section, we will look at some of these methods that have become popular in recent years.

2.2.1. Ensemble Kalman Filter

The ensemble Kalman filter (EnKF), developed by Evensen in 1994, is a variation of the Kalman filter (Kalman, 1960). It is a closed-loop update process for the state variables of a system, by progressive integration of data. In EnKF, the step variables of the system could be both dynamic data like pressure, saturation, gas-oil ratio etc., and static data like permeability and porosity. EnKF sequentially runs a forecast step using a forward model that solves the subsurface flow equations, and then an update step where the state variables at the end of the forecast are updated guided by the mismatch between the predicted and observed well responses. The steps involved in EnKF can be summarized as follows (Nævdal, 2003):

1. Create an initial ensemble of models
2. Run forward full-physics simulation to the first update step when observations are available.
3. Create a correlation matrix that relates the state variables to the corresponding simulated responses.
4. Compute the mismatch between the simulated responses and the actual well observations.
5. Update the state vector guided by this mismatch.
6. Run the models forward using the updated state vector, to the next time step when observations are available
7. Repeat steps 3-5

The updated reservoir parameters such as porosity and permeability as well as dynamic parameters such as pressures, saturations and well responses obtained by

running the flow simulation at the end of each forecast step make up the state vector $s_{k,i}^{f}$, where k is the time step and *i* is the model number, the superscript f indicates that this value has been calculated forward in time. The simulated well responses that have to be compared to the observations are extracted from the state vector by:

$$\mathrm{d_k} = \mathbf{H}\mathrm{s_k} \tag{2.1}$$

Here H is a matrix with identity values only corresponding to the well response values. It has been demonstrated that the observed variables need to be considered as a random variable, otherwise subsequent updates while preserving the mean will underestimate the variance (Burgers et.al., 1998). Hence, a random noise is added to the observation vector:

$$\mathrm{d_{k,i}} = \mathrm{d_{k,i}} + e_{k,i}^{0} \tag{2.2}$$

where $e_{k,i}^{0}$ is a random noise drawn from a multinormal distribution with zero mean and covariance $\mathrm{R_k}$. The forecast state vector $s_{k,i}^{f}$ is then updated using the equation:

$$s_{k,i}^{u} = s_{k,i}^{f} + K_k\left(d_{k,i} - \mathbf{H}s_{k,i}^{f}\right) \tag{2.3}$$

The term K_k is known as the Kalman gain and is given by the relation:

$$K_k = \mathbf{P}_k^f\mathbf{H}^T\left(\mathbf{H}\mathbf{P}_k^f\mathbf{H}^T + \mathbf{R}_k\right)^{-1} \tag{2.4}$$

Here, $\mathbf{P}_k^f$ is the model error covariance matrix given by the following equations:

$$\mathbf{P}_k^f = \mathbf{L}_k^f\left(\mathbf{L}_k^f\right)^T \tag{2.5}$$

$$\mathbf{L}_k^f = \frac{1}{\sqrt{N-1}}\left[\left(s_{k,1}^f - \overline{s_k^f}\right)\ldots\left(s_{k,N}^f - \overline{s_k^f}\right)\right] \tag{2.6}$$

$$\overline{s_k^f} = \frac{1}{N}\sum_{i=1}^{N} s_{k,i} \tag{2.7}$$

and $\overline{s_k^f}$ is the ensemble mean. The operation $\mathbf{P}_k^f\mathbf{H}^T$ extracts the covariance between the state values permeability, porosity, saturations etc. and the well responses. In other words, the Kalman gain functions like the Hessian within a reservoir simulator.

The sequential update of EnKF makes it an attractive option for model updating as new data becomes available. The implementation also makes it possible to generate probability distribution of model predictions because an ensemble of updated values is available at the end of the procedure. However, EnKF is predicated on the relationship between the state variables being adequately represented using a covariance function. This only holds true as long as the multivariate distribution describing the state values is strongly Gaussian in nature and the relationship between state variables and observed data (given by **H** in equation (2.4)) is linear. Zafari (2007) showed that EnKF is unable to represent the posterior probability distribution in cases where the data was bi-model. Even in cases when the state vector is not Gaussian, Emerick (2012) showed that the use of efficient sampling techniques like MCMC (Markov Chain Monte Carlo) can be combined with EnKF to enable proper sampling of the posterior distribution of reservoir parameters. However, the iterative nature of MCMC makes the model updating procedure computationally expensive. Zhou et.al. (2011) showed that using a transformation of non-Gaussian state variables to a Gaussian distribution and subsequently performing EnKF and finally back transforming the variables to the non-Gaussian space preserves the non-Gaussian nature of the initial state throughout the update process. There has also been a lot of work done using methods like truncated-Gaussian and Gaussian mixture models to circumvent the inability of EnKF to model non-Gaussian fields like facies distributions, channels etc. (Lantuéjoul 2002, Liu and Oliver 2005).

A further problem with EnKF lies in the fact that the update step only considers the mismatch between observed and predicted responses forward in time i.e. getting a good match to observed data at any stage has no guarantee that the matches for previous time steps will be preserved. This can cause a loss of geologic information over multiple

updates, and the final model set might be geologically inconsistent with the initial data (Mantilla 2007).

Finally, with the evolution of EnKF, additional reservoir parameters such as fluid contacts, fault transmissibilities and NTG ratio have also been added to the state vector. This can cause reduction in the updating of some parameters. Indeed, Chen (2010) and Wang (2010) have both pointed out that if there are variables like fluid contacts in the state vectors, it might be better to restart the simulation from the initial time after occasional updates, since updated contacts are not reflected by phase saturation updates.

2.2.2. Neighborhood Algorithm

The Neighborhood Algorithm (NA) was developed by Sambridge in 1999 for solving inverse problems in geophysics. It is a stochastic sampling algorithm that finds an ensemble of models by non-linear interpolation in the parameter space, and guided sampling of well-fitting regions of the parameter space. Non-linear interpolation is achieved by dividing the entire parameter space into Voronoi cells (Voronoi 1908, details in Okabe et.al., 2000). The division of the parameter space using this method is always unique, space filling and inversely proportional to the number of points (Sambridge 1999).

The algorithm quantifies uncertainty using two basic phases: search and appraisal. In the search phase, the initial model set is run forward using a simulator, and a misfit function with observed data is created for each member of the ensemble. The n_r models with the lowest misfit are chosen out of the entire n_s models, and a new set of n_s models are created using a Gibbs sampler in the n_r Voronoi cells. Selective sampling of good regions is achieved by using information about *all* previous models, thus overcoming the

convergence problem in stochastic sampling. The process is repeated until it reaches a pre-defined number of iterations. The ratio of n_s/n_r defines the trade-off between exploring the parameter space and finding better matched models. For n_s/n_r value of 1, the algorithm explores the entirety of the parameter space; as the value of this ratio increases, there is increased emphasis on finding better matched models at the cost of exploration.

The NA algorithm by itself does not provide an estimate of prediction probability, hence a separate calculation is conducted to find the posterior probabilities in the appraisal phase, using Bayes' rule (the NA-Bayes algorithm):

$$p(m|m_{ref}) = \frac{[p(m_{ref}|m)p(m)]}{\int p(m_{ref}|m)p(m)dm} \tag{2.8}$$

Here, m_{ref} is the history data, and m is the model response. The calculation uses the volume of the Voronoi cells and the value of the model misfit (assumed constant for each Voronoi cell) to define the likelihood function, $p(m_{ref}|m)$. According to Subbey (2003), this approach offers an advantage over traditional methods of uncertainty analysis by allowing the use of non-Gaussian distributions. The denominator (called the normalizing term), however, is not easy to calculate. A solution lies in the use of Markov Chain Monte Carlo (MCMC) formulation to sample from the posterior distribution without the need to exactly calculate the Bayes term (Christie 2002).

2.2.3. Particle Swarm Optimization

Particle swarm optimization (PSO) is an example of using swarm intelligence for exploring the parameter space of a problem. It is inspired by the interaction of natural animal populations, where the individual actions of the swarm members are integrated

with the sharing of information between members, leading to powerful problem-solving techniques using 'collective intelligence'. This nature is seen in the behavior of ants in an anthill (itself the basis of another similar algorithm called Ant Colony Optimization [Dorigo 1992]) and in flocks of birds.

PSO was developed by Kennedy and Ebarhart in 1995 as a population-based stochastic optimization algorithm, and has been widely used to solve a variety of problems (Kennedy and Eberhart 1995; Eberhart and Shi 2001). In this algorithm, multiple particles explore the parameter space, with the movement of each particle guided by a combination of the memory of good locations that it sampled and the swarm memory of good locations. The position of a particular particle in parameter space represents a possible solution to the optimization problem.

The steps of the algorithm can be summarized as follows (Mohamed et.al. 2010):

1. Define a random selection of particles in parameter space, together with a random initialization of particle velocities. Run each particle through a simulator.
2. Calculate the fitness function for each particle. This will dictate the quality of the location in the parameter space.
3. Update the position of p_{best}, the best location for the particle with the current value. This gets updated every time the particle encounters a location with a better fit to observed data.
4. Calculate the global best position using the p_{best} information across the entire swarm. This is designated by g_{best}, and updated similarly to p_{best}.
5. Update the velocity of the particle using the equation:

$$v_I^{k+1} = \omega v_I^k + c_1 rand_1\left(p_{best,i} - x_i^k\right) + c_2 rand_2\left(g_{best}^k - x_i^k\right) \quad (2.9)$$

Here, the velocity term is composed of three components. The first term is called the inertia term and it reflects the tendency of the particle to keep moving in the same direction as before. The second term is called the memory term and defines the tendency of the particle to move towards its own best location. The third social term, defines the tendency of the particle to move towards the best position of the entire swarm. This is shown in the vector diagram in Figure 2.1. The velocity of the particle is updated as the vector sum of the current velocity (inertia term), the vector representing the tendency of the particle to move towards its own best location (memory term) and a vector representing the tendency of the particle to move towards the best location found by the entire swarm (social term).

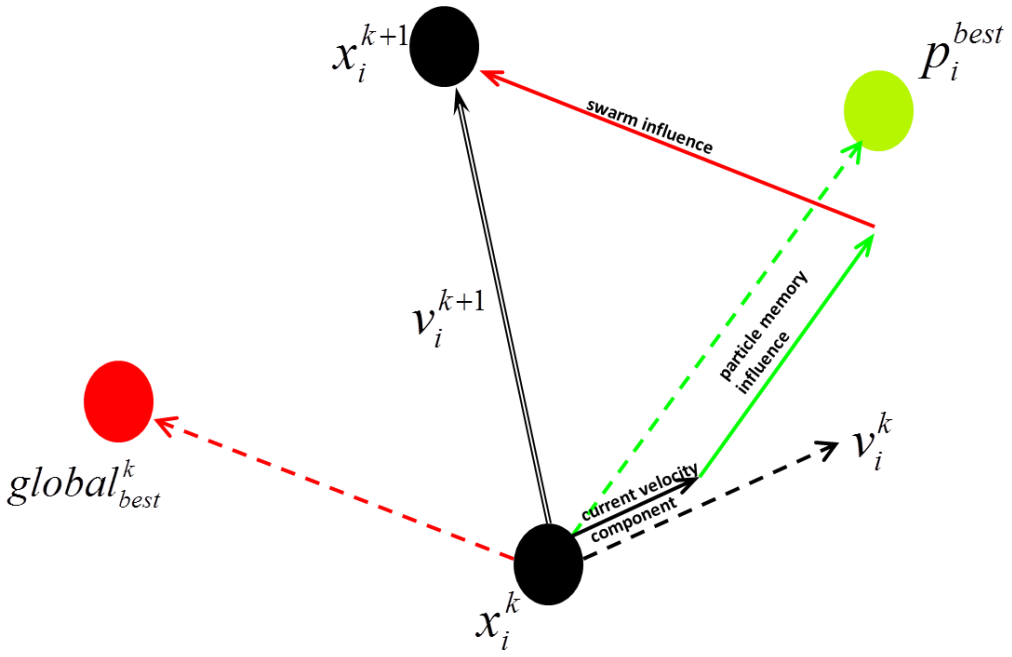


Figure 2.1. Schematic of Particle Swarm Optimization (adapted from Christie et.al., 2010. SPE 135264)

6. Update the position of the particle using the equation:

$$x_i^{k+1} = x_i^k + v_i^{k+1} \tag{2.10}$$

7. Repeat steps 2 to 6 until the required number of iterations is reached.

Similar to the NA algorithm, the PSO does not have an explicit posterior probability calculation, and hence needs additional computation to quantify the probability envelope for predictions.

2.2.4. Genetic Algorithm

Genetic algorithm (GA) belongs to a general class of computational methods that are derived from natural selection and genetics. It operates on the principles of population

genetics and natural rules for propagation of the fittest members of the species. It starts with a population of models, with a given gene structure (the parameter set for each model) and selects the 'fittest' members of the ensemble; these fittest members are then subject to modification of their genetic material (evolution) in order to generate new members of the species.

The process is initiated by coding the parameters of the model into an array called the chromosome, composed of parameters (genes) to define each model. The parameters are static properties like permeability and porosity, recorded at certain pilot points. For example, if the models are defined by permeability at 100 pilot points, the chromosome for each model will be composed of 100 values of permeability, and the individual permeability values will be the genes. In early applications of GAs, the genes were always coded in binary and later mapped to some real values (Sen et.al. 1995); however, modern applications have been designed to directly code real values into the gene structure (Oliveira 1997, Romero 2000).

The initial population then evolves through the processes of reproduction, crossover and mutation. Reproduction is the process of copying the genetic material into the next generation, the probability of which is defined by its goodness-of-fit to the observed data. In its most basic form, the probability of selection for reproduction is given by (Goldberg 1989):

$$P_S(m) = f(m) / \sum_{models} f(m) \tag{2.11}$$

where f(m) is the goodness-of-fit measure for model 'm'

The crossover operation, also known as the 'recombination operator', combines different genetic material by choosing the well-performing genes from different chromosomes.

Using the permeability example from above, the crossover operation will chose the particular permeabilities at pilot points that showed a high degree of fit to observed data. Thus, the process tries to create a new member that retains the best properties of the previous generation. The process of mutation introduces new genetic material into the population, driven by a mutation probability. Using the permeability example once again, the process will *change* the permeability values at the pilot points to drive models towards a better fit. Various kinds of mutation operators, like jump and creep, have been implemented. The process again can be of two types: uniform and non-uniform. Uniform mutations alter genetic material randomly within a range of possible values, which could cause the mutations to move away from good solutions. Wardlaw (1999) implemented a modified method of uniform mutations which mutated a specific gene only by a specific amount, positive or negative. In non-uniform mutations, the degree of mutations are reduced as the run progresses, thus allowing small adjustments and fine-tuning in later generations.

The GA algorithm produces a population of best-fit models to the observed data. The process, however, does not have internal measures for measuring posterior probabilities, and thus need post-operational analysis for estimating such probabilities.

2.2.5. Summary

In this section, we discussed four existing methods for multi-model history matching and uncertainty estimation: Ensemble Kalman Filter (EnKF), Neighborhood Algorithm (NA), Particle Swarm Optimization (PSO) and Genetic Algorithms (GAs).

EnKF provides a method for sequential integration of data in model ensembles, but suffers from high computation costs when modified to apply to non-Gaussian distributions. In its basic form, it is limited to Bayesian linear models, though a lot of

additional research has looked at non-Bayesian applications with highly non-linear relations between state vectors and observations.

Both NA and PSO are based on exploration of the parameter space to find multiple minima. The drawback of both methods is the need for computation of the forward model for a large number of parameter selections, which makes it expensive. However, this problem can be addressed by implementing in parallel computer architectures. Despite that, it can be expensive to generate multiple models using these methods in order to assess the residual uncertainty.

The initial formulation of PSO also suffered from problems like velocities being too high which caused particles to venture outside the parameter space (fixed by velocity clamping [Eberhart and Shi 2001]), inadequate tradeoff between exploration and exploitation (fixed by using appropriate inertia weights [Birge 2003, Trele 2003]), and domain boundary problems.

The primary drawback of GAs, or any of the population based approaches, is their slow rate of convergence. This can be alleviated to an extent by reduction in the dimensionality of each chromosome by lumping together of parameters (Schulze-Riegert et.al. 2003), or be accelerating using a gradient-based search method.

2.3. PROXY FORMULATION AS AN ALTERNATIVE TO FLOW SIMULATORS

The algorithms outlined in the previous section almost always suffer from one common drawback: the significant computational cost of running flow simulations on large sets of models. Aside from the application of parallelization schemes, it is possible to address this problem by replacing the flow simulator with an inexpensive approximation or proxy that allows us to rapidly evaluate flow responses. In this section,

we will look at some existing methods of proxy formulations that have been implemented in the industry.

2.3.1. Streamline Simulation

The process of modeling fluid flow using streamtubes can be traced back to the work of Muskat (1933), when he described the theoretical analysis of water-flooding networks. The earlier applications were mostly limited to the use of streamtubes. Streamtube simulation does not suffer from numerical dispersion as do finite difference simulations. It is readily applicable to cases with slowly changing velocity fields (like waterfloods). However, initially this approach was oapplied only in two-dimensions, and extension to three dimensions was non-trivial. Lake et.al (1981) were able to devise a modification of the streamtubes method by combining areal streamtubes with cross-sectional finite difference based methods, that enabled application of streamtubes to a number of cases - waterflood prediction (Emmanuel et.al. 1997) and miscible flood predictions (Mathews et.al. 1989). Steamline simulations are an adaptation of these early streamtube methods, without the need for explicit calculation of tube geometries. The key concept here is to reformulate the transport equation in time-of-flight coordinates, which decouples the flow from the transport and enables solving one-dimensional transport equations along streamlines, as detailed below.

The key steps in a streamline simulation can be stated as follows (Dutta-Gupta and King 1995, Crane and Blunt 1999, Kharghoria 2004):

1. Compute the pressure field and hence the velocity field using standard finite difference formulations of flow equations in a reservoir.

2. Using the computed velocity field, trace streamlines. The components of the velocity field can be used to find the entry and exit points of each streamline moving across a grid block. An important underlying assumption here is that the velocity is *linearly* varying from the entry to the exit face.
3. Once all the streamlines have been traced, compute the travel time along a streamline or the time-of-flight. This is given by the equation:

$$\tau(\mathbf{x}) = \int_{\psi} \frac{ds}{|\mathbf{v}(\mathbf{x})|} \tag{2.12}$$

   Where τ is the travel time along the streamline ψ and v(x) is the interstitial velocity.
4. The transport equations (saturations, concentrations) are converted to time-of-flight coordinates using the operator identity (Datta-Gupta and King 1995):

$$\mathbf{v}(\mathbf{x}) \cdot \boldsymbol{\nabla} = \frac{\partial}{\partial \tau} \tag{2.13}$$

   For example, consider the Buckley-Leverett equation:

$$\Phi \frac{\partial S_w}{\partial t} + \vec{v} \cdot \nabla F(S_w) = 0 \tag{2.14}$$

   Then, using the relation in equation (2.13), we can rewrite equation (2.14) as:

$$\frac{\partial S_w}{\partial t} + \frac{\partial F(S_w)}{\partial \tau} = 0 \tag{2.15}$$

   The transport quantities (S_w in the above case) are calculated and propagated along each streamline, and then finally mapped onto the underlying grid.
5. Occasionally, the pressure and velocity solutions need to be recomputed to account for changes in the fields over time.

Streamline simulations have proved to be a powerful tool for evaluating fluid migration in reservoirs and have been extensively used for the purpose of uncertainty quantification (Alpak et.al. 2009, Park et.al. 2012). Even though the initial formulations were restricted to velocity-dominated incompressible flow, the process has been extended to account for compressibility (Datta-Gupta et.al. 2001), dissolution (Thiele et.al. 1997), dispersion (Obi and Blunt 2006), relative permeability hysteresis (Qi et.al. 2007) and other additional physical processes.

However, there is still considerable difficulty in solving for the pressure and velocity fields using IMPES methods, which limits the accuracy of the process for smaller time steps. Further, the inherent assumption that the flow is along a streamline is violated in cases like transverse diffusion, well rate alterations etc. In such cases, it becomes necessary to use methods like multiple timesteps and timestep operator splitting, which makes the process numerically expensive. There have also not been many field scale applications of streamline simulators for compositional cases.

2.3.2. Artificial Neural Networks

Artificial neural networks (ANNs) are based on the concept of natural neural networks as seen in the central nervous system of living creatures. In the oil and gas industry, ANNs have been used to solve a wide variety of problems, like fluid property analysis (Hegeman et.al. 2009), relative permeability analysis (Guler et.al. 2003), prediction of PVT properties (Gharbi and Elsharkawy, 1999) and prediction of well responses (Boomer 1995). It is this last application which is of primary importance in history-matching and uncertainty analysis.

There are three main components of an ANN: the input, the hidden nodes and the output nodes. The input node consists of the vector of prior information. The hidden nodes are the intermediate nodes which are connected to the input nodes by connection and weighting applied to the connections. The outputs from the hidden nodes are mapped to the output nodes and there is flexibility to configure that mapping and weight functions also. A basic single-layer ANN is shown in Figure 2.2.

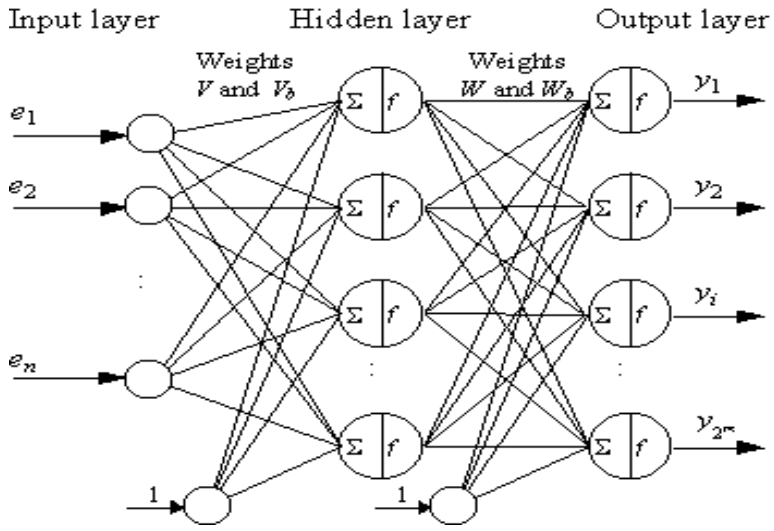


Figure 2.2. Layout of an artificial neural network, showing the input nodes, hidden nodes and output nodes (from Laboratoire d'Automatique et d' Informatique Industrielle de Lille, http://sic.ici.ro/sic1999_2/art03.html)

The primary steps involved in an ANN simulation are: designing, training and prediction. The designing stage is when the structure of the neural network is defined. The training stage is the critical step in ANNs, which is when the learning algorithm for the process is defined. These can be of two kinds: supervised and unsupervised. Unsupervised learning is used primarily for pattern recognition, where it learns the pattern of the input data and learns to reflect that pattern in the output data. Supervised learning, on the other hand, compares the values at the output nodes to actual observed

data, and modifies the internal weighting (the connections between input and hidden nodes) such that the mismatch between output and prediction is minimized.

Though neural networks can capture the highly non-linear relationship between static variables and dynamic output from a reservoir, it still has some significant drawbacks. The process of training the neural net, in itself, is a laborious task that requires many executions of the full-physics flow simulator on a large set of training models. Furthermore, integration of new data into the model would require retraining of the neural net, at an additional computation cost. The biggest drawback, however, is that the mapping between the input and the output obtained by a neural network is purely statistical with limited understanding of the physics linking the input and output parameters. In spite of these drawbacks, ANNs are powerful and are becoming increasingly popular in the petroleum industry.

2.3.3. Particle Tracking

The modeling of fluid transport in porous media has evolved into a robust numerical framework, accounting for all relevant physics. The biggest drawback of such a system, however, lies in the computational cost of solving the finite difference equations for pressure, saturations and concentrations. Additionally, there is numerical dispersion and instability in the form of artificial oscillations associated with these solvers, necessitating smaller time steps and finer grid resolutions, further adding to computation times. Particle-tracking based methods have been tested for a variety of such cases as a useful alternative to full physics numerical solvers.

There are two primary approaches for using particle tracking to simulate fluid flow in porous media: continuous time random walker (CTRW) and random walker

particle tracking (RWPT). Both approaches rely on representing the physics of fluid flow using random particles that move through the grid using certain rules. RWPT moves the random particles using the velocity field obtained by solving the flow equations, and then adds an uncertainty term to capture dispersion. On the other hand, CTRW (introduced by Montroll and Weiss, 1964) combines both advection and dispersion into a single master equation, as shown below (Berkowitz et.al. 2000):

$$R(\mathbf{s}, t) = \sum \int_0^t \psi(\mathbf{s} - \mathbf{s}', t - t') R(\mathbf{s}', t') dt' \qquad (2.16)$$

Here, R(**s**,*t*) is the probability for a random walker particle to arrive at location **s** in time t, and $\psi(\mathbf{s}, t)$ is the probability of transition of a particle between two locations separated by **s-s'** over a time interval *t-t'*. Equation (2.16) combines advective, dispersive and diffusive effects into a single equation.

In our work in later chapters, we will use the RWPT formulation to guide our modeling of CO_2 migration through an aquifer. The basis of using RWPT to model fluid transport in porous media is based on the analogy between the random walker equation and the Fokker-Planck equation (Kinzelbach 1987). In the following section ,we look at the mathematical formulation of the scheme.

Mathematical formulation of RWPT

The basic advection-diffusion transport equation can be stated as:

$$\frac{\partial c}{\partial t} + \nabla \cdot (\mathbf{u}c) = \nabla \cdot (\mathbf{D}\nabla c) \qquad (2.17)$$

where **D** is the dispersion tensor, **u** is the velocity term and *c* is the concentration term. This is a second-order PDE, which can be solved using standard finite-difference or finite element methods. To overcome the problem of numerical instability and dispersion, the Peclet and Courant number have to be sufficiently small (Huyakorn et.al. 1983).

RWPT simulates solute transport by partitioning the solute mass into a large number of representative particles. The evolution in time of a particle is driven by a drift term that relates to the advective movement and a superposed Brownian motion responsible for dispersion. The displacement of a particle is written in its traditional form given by the following integration scheme (Gardiner, 1990):

$$X_p(t + \Delta t) = X_p(t) + A\big(X_p, t\big)\Delta t + B\big(X_p, t\big) \cdot \xi(t)\sqrt{\Delta t} \qquad (2.18)$$

where Δt is time step, $X_p(t)$ is the position of a particle at time t, A is a drift vector, B is the displacement matrix, and $\xi(t)$ is a vector of independent, normally distributed random variables with mean 0 and variance 1.

To establish a parallel between equation 2.18 and equation 2.17, equivalence is established between the displacement scheme in a random walk and the Fokker-Planck equation and then between the Fokker-Planck and transport equations. The Fokker-Planck equation describes the evolution of the probability density function of a particle under the influence of a stochastic process (like diffusion). It has been demonstrated (Ito 1951) that the probability of finding a particle within a given interval at a given time t [$f(X_p, t)$], obtained from Equation (2.18) satisfies the Fokker–Planck equation for large particle numbers and a very small step (Kinzelbach 1987). This equation describes the motion of the particle density distribution *f* and is given by:

$$\frac{\partial f}{\partial t} + \nabla \cdot (uf) = \nabla\nabla\colon (Df) \qquad (2.19)$$

Where:

$$\nabla\nabla\colon (\mathbf{D}f) \equiv \sum_{i=1}^{n}\sum_{j=1}^{n} \frac{\partial^2 D_{ij}}{\partial x_i \partial x_j} f \qquad (2.20)$$

and n is the number of dimensions.

Both advection-dispersion and Fokker-Planck equations are similar to each other in that they contain an advection and a diffusion term. To establish a better parallel between the two equations, equation (2.17) can be modified as follows:

$$\frac{\partial c}{\partial t} + \nabla \cdot (\mathbf{u}c) + \nabla \cdot (c\nabla \cdot \mathbf{D}) = \nabla\nabla\colon (\mathbf{D}c) \tag{2.21}$$

Using $\mathbf{u}^* = \mathbf{u} + \nabla \cdot \mathbf{D}$, the equation can be transformed into an equivalent Fokker-Planck equation:

$$\frac{\partial c}{\partial t} + \nabla \cdot (\mathbf{u}^* c) = \nabla\nabla\colon (\mathbf{D}c) \tag{2.22}$$

Substituting the drift vector A in equation (2.17) with the modified velocity vector, the RWPT scheme is obtained:

$$X_p(t + \Delta t) = X_p(t) + \left[\mathbf{u}\left(X_p, t\right) + \nabla \cdot \mathbf{D}\left(X_p, t\right)\right]\Delta t + \mathbf{B}\left(X_p, t\right) \cdot \xi(t)\sqrt{\Delta t} \tag{2.23}$$

Where the displacement matrix **B** is related to the dispersion tensor as: $2\mathbf{D} = \mathbf{B}\cdot\mathbf{B}^{\mathbf{T}}$

The particle tracking system conserves mass exactly (since the random particles are not lost or destroyed). Further, it can be applied to a gridless system if needed, since the basic mathematical formulation does not contain any grid-based formulation. There is also no numerical dispersion associated with the walkers.

RWPT approaches to simulate fluid flow in the subsurface have been especially popular in hydrology and environmental engineering (Kinzelbach 1988, van Dop 1985, Li et.al. 2011), though there has been limited application in the oil and gas industry [for example, John et.al. (2010) used a particle tracking approach to study dispersive mixing in field scale miscible displacements]. Later in this dissertation, we will outline the use of particle tracking algorithms for use in geologic carbon sequestration, which can be extended to flow in oil and gas reservoirs.

2.3.4. Summary

In this section, we looked at a few proxy models that can be used in place of a full-physics simulator during multi-model history matching. Streamline simulators are the most popular among these approaches and have been widely tested on a large number of synthetic and field cases. Though their implementation initially was limited to incompressible, single phase, two-dimensional flow problems, continued evolution has seen additional physics like multiphase, multicomponent flow with compressibility being accounted for by these simulators. Artificial neural networks have recently seen an increased interest, mostly due to the "black-box" approach of the implementation; however, that has also been one of their biggest drawbacks. They suffer from not being able to capture the actual physics linking the non-linear relationship between static data and dynamic responses. The application of particle tracking methods to model solute transport has been particularly popular in hydrodynamic studies due to their computational advantage over full-physics simulators; however, their use has been quite limited in the oil and gas industry. In subsequent chapters, we will show implementation of the random walker particle tracking to model fluid flow during CO_2 sequestration and demonstrate its use in a model selection process for assimilating dynamic data.

Chapter 3 : Model Selection Algorithm

The uncertainty in reservoir architecture, geology and distribution of rock types and their associated petrophysical properties make it necessary to not just develop a single model but rather multiple models, each conditioned to all the available data. Performing grid-node updates to reservoir parameters in order to minimize the mismatch between the observed and predicted response and subsequently repeating that process in order to develop multiple reservoir models, can be extremely time consuming. In lieu of this iterative updating process, it is useful to interpret the process of history-matching as an effort to find the most suitable candidates for the reservoir under study based on the observed responses, that is history-matching becomes an exercise in *selecting* best-fit models for the reservoir. At the end of the model selection process the objective is not to get the single best-fit model, but a cluster of models that share the reservoir characteristics important to match the history and that permit assessment of the residual uncertainty after the data assimilation process. In this chapter, we outline a model selection process based on this idea of multi-model history matching, conditioned to data from multiple field sources.

3.1. Model Selection Algorithm: An Overview

Traditionally, geostatistical approaches have sought to calculate the conditional probability P(A|B) where A is a simulation event at the grid-block level (e.g. permeability, porosity etc.) and B is static geologic data. In the event dynamic data (such as production or injection history information) is available to model the reservoir, the goal is to construct the conditional distribution P(A|B,C) and subsequently sample several models from that distribution. In contrast to this grid-based approach, in our model-

selection approach, the event A is an entire model. Thus, P(A|B) is the conditional probability of a model given prior geological information. The objective of the model selection process is to estimate the posterior probability P(A|B,C) where C is the given field data (like bottom-hole pressure / rate at wells), i.e. the probability of a model given both static geologic data and dynamic well data. This posterior probability is represented by all the models in the final set of models derived at the end of the model selection process that share a common characteristic.

The first step is to represent the prior uncertainty about reservoir geology and architecture using an initial set of models. In order to adequately capture the prior uncertainty, it is important to make this initial model set as wide as possible, considering all possible interpretations for the reservoir. The next step in the algorithm is to assess the flow connectivity of the models. Differences between the models are computed in terms of the connectivity metrics. The difference/distance between pairs of models are then subject to multivariate analysis techniques such as principal component analysis or Multi-Dimensional Scaling (MDS). These analysis techniques are used to project the models on an n-dimensional space as a cloud of points, with each point representing one model. The n-dimensions refers to the minimum dimensionality required to capture most of the variability (variance) exhibited by the flow characteristics of the prior models. This cloud is then divided into distinct groups or clusters, such that models grouped together show connectivity characteristics that are similar to each other and different from that of models in other groups. Once the models have been clustered, representative models are picked from each cluster and run through a full-physics numerical simulator. The responses from the simulator are compared to the response observed in the field in order to find the model cluster closest to the observed field data. This process of projection, clustering and simulation to find the best cluster is then repeated on the best-fit cluster of

models from the previous step. The process is terminated when the model clustering does not improve the posterior probability of the clusters or the clusters become equiprobable. The entire process is shown in Figure 3.1and described in detail in the following sections.

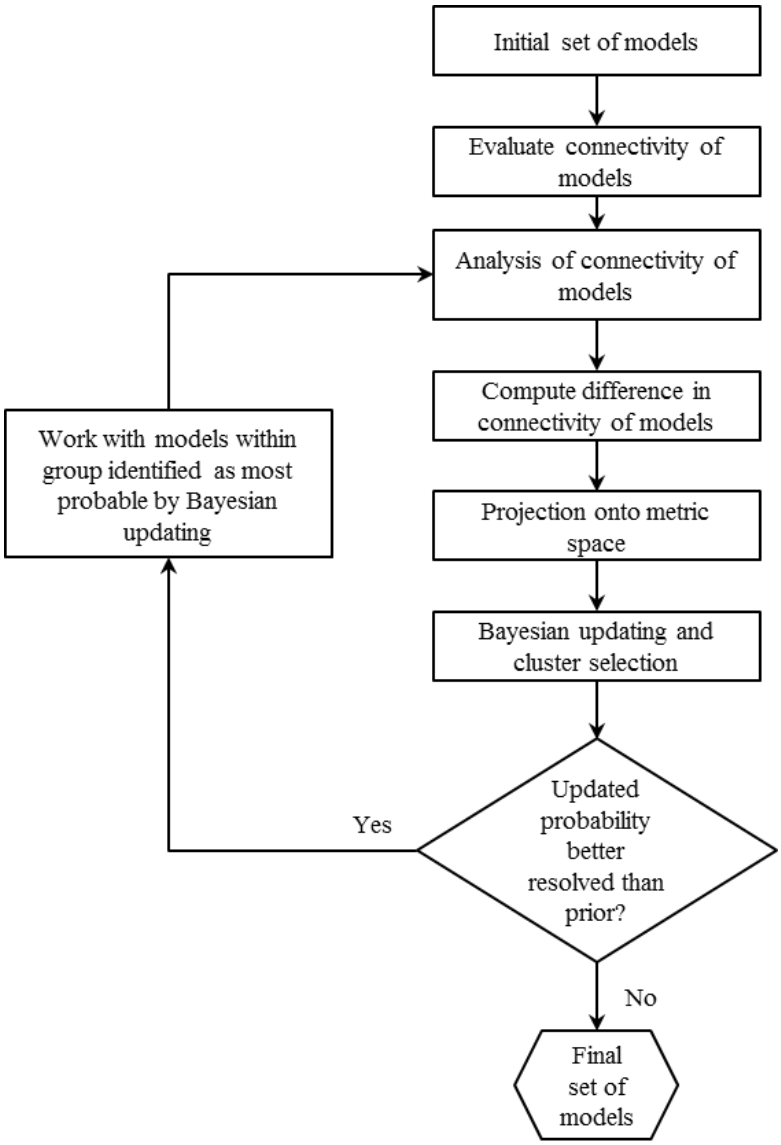


Figure 3.1. Schematic of the model selection algorithm (Bhowmik 2010)

3.2. INITIAL SET OF MODELS

The initial set of models is intended to represent the uncertainty in reservoir architecture, geology and rock type distribution of the subsurface entity under study. To illustrate this concept of prior uncertainty, consider a fluvial channel system as shown in Figure 3.2. The larger channel system consists of two primary channel types:

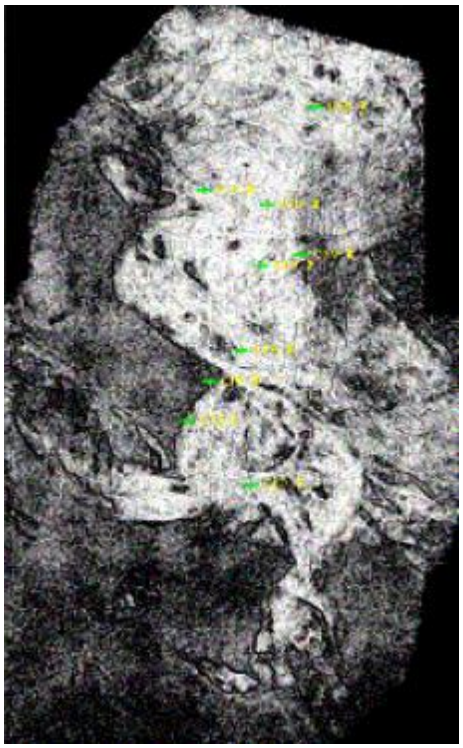


Figure 3.2. Seismic section at the top of a carboniferous reservoir in central Africa, showing the outline of a channel system (Wright 2007)

distributaries and estuaries (Allen and Chambers 1998). These channels are stacked within the system, the type of stacking dependent on the sediment load at depositional times. There is uncertainty associated with this stacking and distribution of the individual channels. Further, there is a lot of uncertainty of the distribution of sand and mud within the individual distributaries or estuaries. Thus, to represent these uncertainties, we would

need to account for various possible layouts of the individual channels within the system, and also various distributions of sand and muds within the channels. Thus, our initial model set will have to be large enough to account for the combined uncertainty at different scales of the reservoir. The initial set of models have to reflect our judgment about what scales of heterogeneity effect the specific fluid flow processes that we are considering.

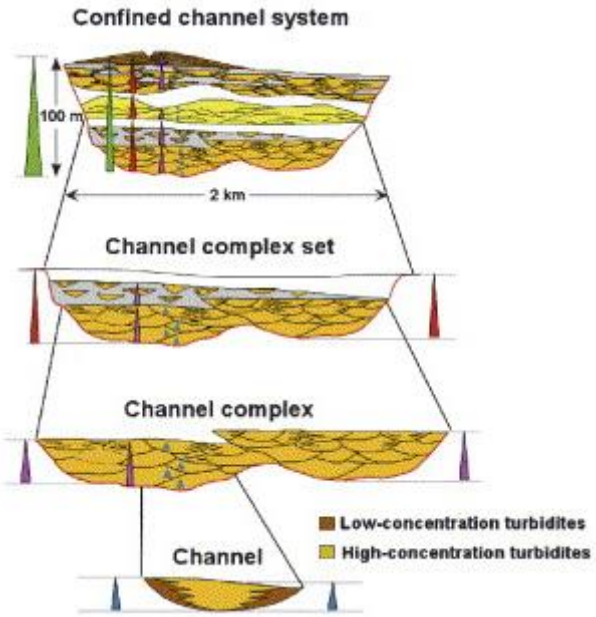


Figure 3.3. Hierarchy of channel deposits, showing individual channels distributed within the system and the facies distributions within each channel (adapted from Abreu et.al., 2003)

3.3. EVALUATION OF CONNECTIVITY OF RESERVOIR MODELS

Once the required reservoir models have been created, they have to be analyzed in order to assess their flow connectivity. This can be achieved by using a numerical simulator or by using a fast-transfer function that approximates the flow characteristics captured by a numerical simulator. Given that the initial suite of models may be large, the

use of fast techniques to assess the flow connectivity of the models is preferable. For our work, we developed two particle-tracking proxies, which we will describe in greater details in later chapters. It has to be emphasized that our goal in this step is to rapidly assess the connectivity of initial suite of models so that we can begin dividing the models into groups that exhibit similar characteristics.

The results from the simulator or the proxy are recorded at certain locations within the models and the response at these locations is used to characterize the flow connectivity of the models. The choice of proxy/flow response monitoring locations becomes crucial and techniques for deciding the locations are discussed later in Chapter 7 of this dissertation.

3.4. Analysis of Connectivity Measurements for Model Clustering

The monitoring locations can be used to record responses from the numerical simulator or fast-transfer function. The set of responses for each model defines a set of metrics that describe the connectivity of that model. To illustrate this point, consider two models as shown in Figure 3.4.

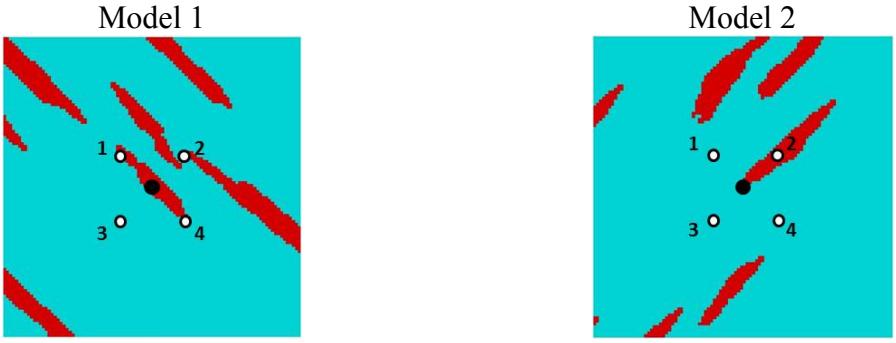


Figure 3.4. Two models with permeability features in opposite directions. An injection well is shown in black at the center of the grid. There are 4 measurement locations (numbered 1 through 4) in white around the injector.

The two grids have high-permeability features running in different directions, with an injection well at or close to one such feature. The fluid injected into the well will move predominantly in a NW-SE direction in the model on the left and in a NE-SW direction in the model on the west. In this case, if we were to record saturations at the measurement locations at certain intervals of time, we would see higher saturations in locations 1 and 4 in model 1, and location 2 in model 2. Denoting a high saturation value as 1 and a low one as 0, we can construct a corresponding binary array for the responses. This is shown below:

Locations	Model 1	Model 2		Model 1	Model 2
1	High	Low		1	0
2	Low	High	→	0	1
3	High	Low		1	0
4	Low	Low		0	0

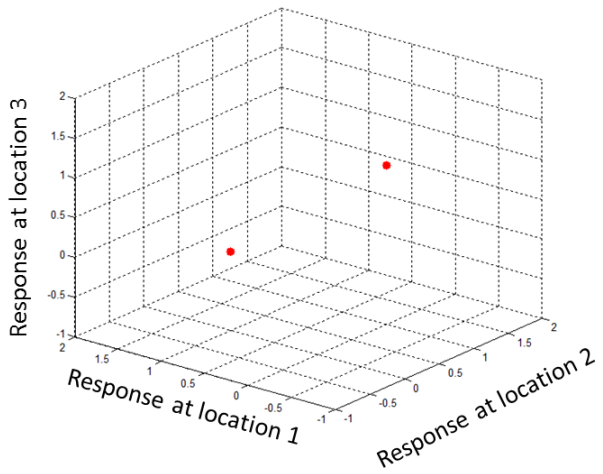


Figure 3.5. The models above represented on a three-dimensional space, using responses from locations 1, 2 and 3

The combination of lows and highs (restated as 0s for lows and 1s for highs) are representative of the particular model. If this process is executed for all models in a model set for 'm' locations at 't' different time intervals, we will then have $m \times t$ number of metrics to describe each model. Using these metrics, we can then represent every model in a multi-dimensional space. To continue the example above, we can decide to represent the two models on a three-dimensional space, whose axes are given by the response (low/high) at locations 1, 2 and 3. This is shown in Figure 3.5.

This example also demonstrates another important idea: In spite of having 4 measurement locations, the difference between the models can be assessed using the response at using the only three of the monitoring locations, because there is no difference in the responses at location 4 between the two models. Thus, even if we have $m \times t$ metrics for each model, we can represent the models by a much smaller set of metrics, making further computations on the dataset less expensive.

Another assumption that goes into creating the scatter plot in Figure 3.5 is that the variables are orthogonal to each other. In order for this assumption to be true, the responses at locations 1, 2 and 3 have to be independent of each other. However, the saturations at those three locations are most likely *not* independent of each other; in model 1, for example, the saturation at locations 1 and 3 will definitely be higher than at location 2 but the actual amount of saturation recorded at 1 and 2 are related to each other due to the same underlying permeability feature being responsible for carrying fluid to these locations from the injector. Thus, before any representation of the model on the basis of the connectivity metrics, we need to create a truly orthogonal basis on which we can project the models.

3.4.1. Projection of Models on an Orthogonal Set of Axes

We used principal component analysis of the connectivity metrics in order to find the leading principal component directions on which to project the models. This approach allows us to not only infer an orthogonal basis for our model set, but also highlights the directions along which maximum variability among the models is observed. Because PCA is a dimensionality reduction technique, it enables the differences between the models to be projected to a lower dimensional space. This is best demonstrated using an example. Suppose we have 1000 models, each represented by 3 metrics, for example saturation at 3 snapshots in time; the metrics are *not* orthogonal to each other since the saturation at a given time is not independent of saturations in the earlier times. Principal component analysis allows us to represent the models on an orthogonal space, as shown

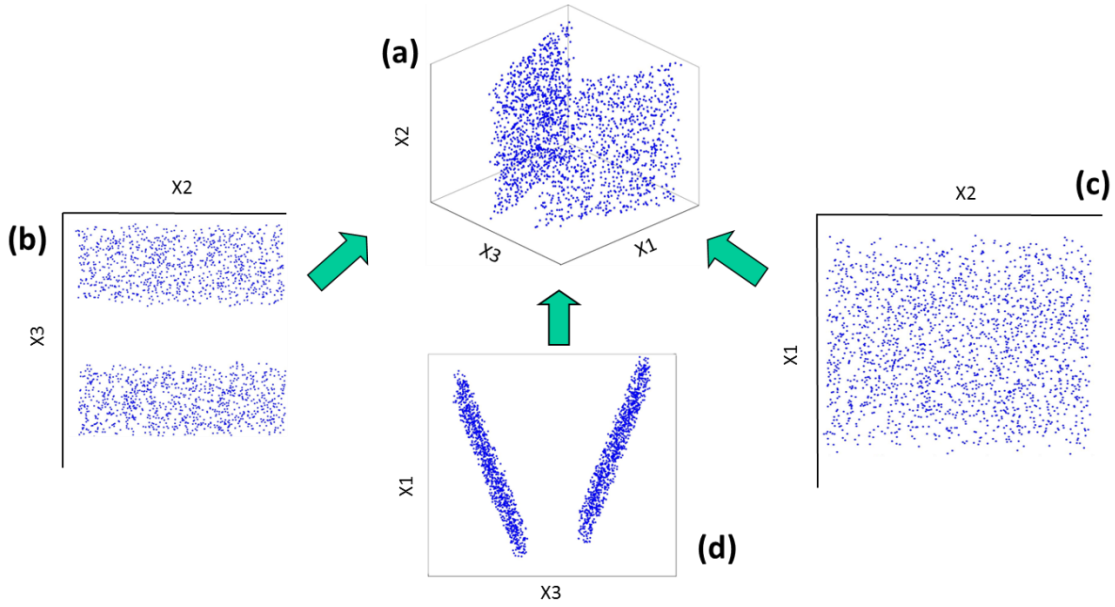


Figure 3.6. (a) Representation of models onto an orthogonal three-dimensional space, identified by principal component analysis, (b) Projection of models on X2-X3 plane, (c) Projection of models on the X1-X2 plane, (d) Projection of models on X1-X3 plane.

in Figure 3.6. Continuing the example, suppose the saturations at the three times is represented by X1, X2 and X3. If we project the models onto the three two-dimensional planes defined by the X1-X2, X2-X3 and X3-X1 axes, we can see different patterns in the projections. Projection onto the X1-X2 plane (Figure 3.6 c) reveals no information about the models, the X2-X3 projection (Figure 3.6 b) shows a degree of grouping among all the models while the X1-X3 projection (Figure 3.6 d) clearly shows the clustering characteristics of the models. Principal component analysis allows us to identify this set of axes which enables us to best resolve differences between the models. In this case, it is identifying the saturation at the first and last time snapshots.

For the purpose of our analysis, suppose we have *m* metrics defined for each of the *N* models available. These can be defined by the matrix M given below:

$$M = \begin{bmatrix} x_{1,1} & \cdots & x_{1,N} \\ \vdots & \ddots & \vdots \\ x_{m,1} & \cdots & x_{m,N} \end{bmatrix} \tag{3.1}$$

Each column of the matrix defines the *m* responses recorded for a given model. Thus, matrix M has *m* rows and *N* columns. In order to find the principal component axes, we first convert each row of M to a zero-mean vector by subtracting from each value along a vector the mean of that row. This creates the matrix M_0 given as:

$$M_0 = \begin{bmatrix} x0_{1,1} & \cdots & x0_{1,N} \\ \vdots & \ddots & \vdots \\ x0_{m,1} & \cdots & x0_{m,N} \end{bmatrix} \tag{3.2}$$

where $x0_{i,j} = x_{i,j} - \frac{1}{N}\sum_{j=1}^{N} x_{i,j}$

Now, the covariance matrix is created for the recorded metrics as an $m \times m$ matrix, showing the covariance between all possible pairs of the metrics. This is given by the matrix C below:

$$C = \begin{bmatrix} c_{1,1} & \cdots & c_{1,m} \\ \vdots & \ddots & \vdots \\ c_{m,1} & \cdots & c_{m,m} \end{bmatrix} = \frac{M_0 M_0^T}{N-1} \tag{3.3}$$

In order to find the principal component directions, we compute the eigenvalues and corresponding eigenvectors for the matrix C. The eigenvectors corresponding to the leading eigenvalues define the principal component directions for the given matrix M (from eq. 3.1). We can decide on how many principal components we want to use to adequately represent all models by computing the variance contribution of each principal component as a fraction of the eigenvalue for that component divided by the sum of all eigenvalues. The original matrix M can then be projected down to this reduced orthogonal set of axes to represent the models.

3.4.2. Clustering of Projected Models

After the models have been projected to an orthogonal set of axes that highlight the difference between the models based on the measurement locations, we can group the models using a clustering algorithm. We chose the K-means cluster analysis algorithm due to its ease of implementation.

K-means is a centroid-based clustering method, where the clusters are defined by a central node that might not be a part of the original dataset. The k-cluster centers are determined such that the sum of squares of distances between data points and its centroid

is minimized. This automatically results in the maximum distance between the clusters. The algorithm can be stated as follows:

1. Assign *k* random cluster centroids
2. Based on the distance of each point from the *k* centroids, assign each point to one of the *k*-clusters.
3. Re-compute the cluster centroids depending on the cluster assignments.
4. Repeat steps 2 and 3.
5. Terminate process when the centroids do not move upon computation of new centroids in step 3.

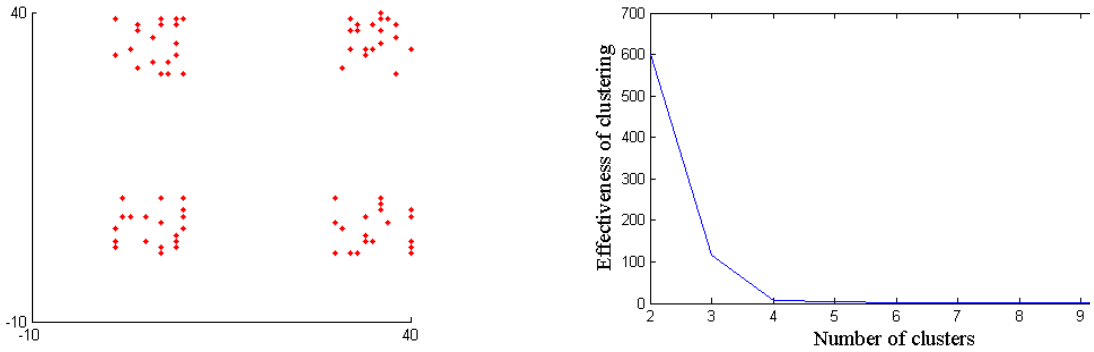
The process suffers from some drawbacks: the number of clusters has to be pre-defined and the optimization algorithm for determining the cluster centroids may converge to a local minimum. In order to address the problem of convergence to a local minimum, the process needs to be repeated a number of times with different starting centroids in order to find the optimum clustering. The problem of defining the number of clusters is a more complicated question to handle.

We implemented a method of associating the results of clustering with a measure of the effectiveness of clustering, given as the ratio of sum of square distances of each data point from its cluster centroid to the sum of square distances between cluster centroids:

$$\eta_k = \sum_{i=1}^{k} \sum_{\substack{j=1, \\ j\ in\ i}}^{m} d_{ij}^2 \Big/ \sum_{i=1}^{k} \sum_{\substack{j=1, \\ j \neq i}}^{k} d_{ij}^2 \qquad (3.4)$$

η_k: effectiveness of clustering with *k* clusters, *m*: models in a particular cluster

Since the objective of the clustering is to maximize the distance between clusters while reducing the spread within each cluster, we can say that lower values of η_k indicate better clustering. This measure can then be used to plot η_k against *k*. The value of η_k will decrease with increasing number of clusters, and in the limit when number of clusters equals the number of data points, it will reach zero. The objective of the plot is to find the point in the curve after which there is a marked change in slope. That point gives an indication of the optimum number of clusters. To illustrate, 4 distinct clusters of points



(a) (b)

Figure 3.7. (a) Actual data points used for the demonstration. There are clearly 4 clusters of points in this case. (b) Plot of effectiveness of clustering vs Number of clusters clearly shows a kink at 4, which is the correct number of clusters.

were divided into 2 to 10 clusters, and their corresponding effectiveness of clustering plotted against the number of clusters (Figure 3.7). The figure clearly shows that the curve almost flattens out after point 4, indicating that the ideal number of clusters is 4 in this case.

3.5. BAYESIAN UPDATING AND CLUSTER SELECTION

Once the models have been divided into clusters, they need to be evaluated using a full-physics numerical simulator so that their responses can be compared to actual field data. For this purpose, we first need to find one representative model for each cluster, and evaluate that model using a full-physics reservoir flow simulator. Since the basis of clustering was the assumption that there were certain common connectivity characteristics among all models within a cluster, the representative model needs to be chosen such that it can reflect the common feature(s).

3.5.1. Representative Model for Cluster

The representative model for each cluster of reservoir models has to encompass the different geological features of all the models within the cluster. A weighted averaging process is used for this purpose, where the weight given to a model is inversely proportional to the distance of the model from the cluster centroid. This serves to highlight the features that are common to most of the models while averaging out the features which are present only in some models but not common to the cluster. However, this average model will not reflect key statistics such as the histogram of heterogeneity features for the reservoir under study. So, a histogram transformation of the average model to the target histogram for the region (same as the histogram of any of the starting models) is subsequently carried out, which creates the representative model.

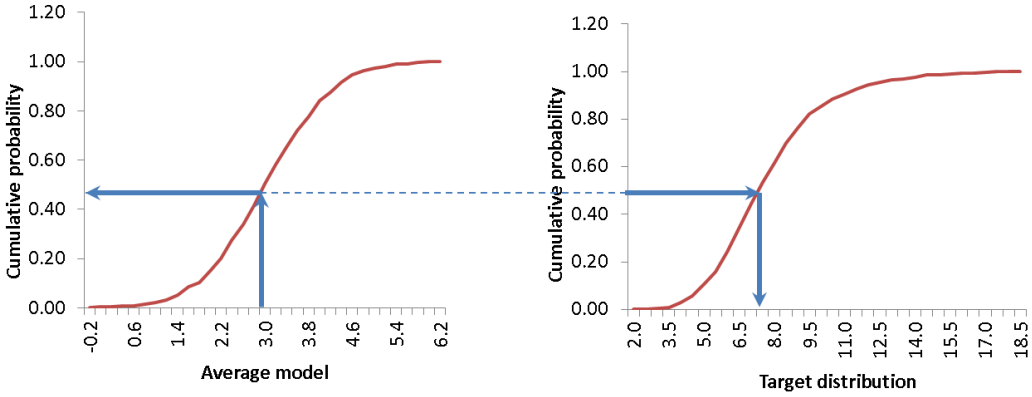


Figure 3.8. Example of histogram transformation. The figure on the left is the distribution of the average model, and the figure on the right is the target distribution

In order to transform the histogram, we created the CDF of the average model and the CDF of our target histogram. For transforming any value in the average model, we computed the quantile of that value from the original CDF, and then found the value corresponding to that same quantile in the target CDF. This was the histogram-transformed value of the average model. This process is shown in Figure 3.8. These representative models (one for each cluster) are then run through a full flow simulator in order to compare them to the field history.

3.5.2. Bayesian Update of Cluster Probability

The simulated responses can be compared to field data in order to estimate how close a particular cluster of models is to the 'real' reservoir. The comparison is done at a well level, in terms of rates / bottom-hole pressures at wells (injectors / producers). Though the comparison may be done qualitatively by comparing the simulated and field responses visually, it is more robust to compute the comparison in quantitative terms. For

this purpose, a Bayesian calculation of posterior probability is implemented adapted from the work of Mantilla (2010).

To demonstrate the process, assume we have *N* models divided into *k* clusters. In the absence of any other sources of information before the model selection process is implemented, all *N* models can be considered equiprobable. Hence, the prior probability of cluster *m*, $P\left(z^m(\mathbf{u})\right)$ can be stated as follows:

$$P\left(z^m(\mathbf{u})\right) = \frac{Number\ of\ models\ in\ cluster\ m}{Total\ number\ of\ models} \tag{3.5}$$

Using Bayes' rule, the posterior probability of each cluster, conditioned to well response can be computed knowing the likelihood function $\mathrm{P}\left(\mathrm{RF_{ref}}\middle|\mathrm{z^m(u)}\right)$:

$$P\left(z^m(\mathbf{u})\middle|RF_{ref}\right) = \frac{P\left(RF_{ref}\middle|z^m(\mathbf{u})\right)}{P\left(RF_{ref}\right)} \times P\left(z^m(\mathbf{u})\right) \tag{3.6}$$

The likelihood function can be calculated from full-physics flow simulations on the representative model for each cluster. If the representative model with simulated response farthest from the observed data be *m*, and the simulated response of the variable of interest be represented by RF^m, then given an observed response RF_{ref}, the deviation of the simulated response from the observed response can be given as

$$\sigma_m^2 = \left|RF_{ref} - RF^m\right|^2 \tag{3.7}$$

or if the response is a series in time, the deviation can be computed as:

$$\sigma_m^2 = [\mathbf{RF_{ref}} - \mathbf{RF^m}]^T[\mathbf{RF_{ref}} - \mathbf{RF^m}] \tag{3.8}$$

Assuming a Gaussian distribution for the mismatch between simulated and observed values, with the observed data $\mathbf{RF_{ref}}$ as mean and σ_m^2 as variance, we can compute probability envelopes around the observed response. Then, the likelihood $P\left(RF_{ref}\middle|z^m(\mathbf{u})\right)$ can be calculated according to the position of the simulated response within the probability envelope. Because the simulated response may not follow any one of the calculated probability contours, we assign the contour corresponding to the

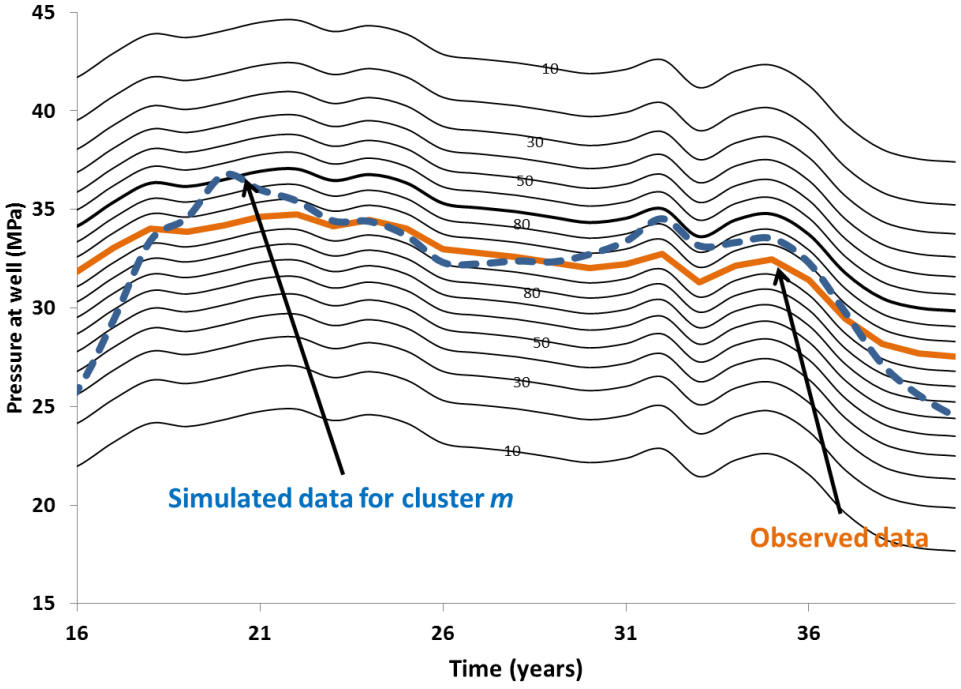


Figure 3.9. Demonstration of probability envelopes around observed response. The probability of a particular cluster is inferred from the envelope it lies in.

maximum deviation of the simulated response as the likelihood value. This is demonstrated in Figure 3.9. In this example, the probability of the dashed blue line is 0.65.

The denominator in eq. 3.6 $\mathrm{P(RF_{ref})}$ is the prior probability of the response RF_{ref} and can be calculated from the law of total probability as:

$$P\left(RF_{ref}\right) = \sum_{m=1}^{k} P\left(RF_{ref}|z^m(\mathbf{u})\right).P\left(z^m(\mathbf{u})\right) \tag{3.9}$$

3.6. STOPPING CRITERION

The model selection algorithm is an iterative process, where the clustering and Bayesian updating is repeated using the best-fit models from the previous iteration. Each iteration process thus successively refines the process of model cluster selection, narrowing it to smaller sets of models. The process thus requires a criterion for stopping the iterations. This can be based on the number of models remaining in the cluster or on the updated probability of clusters.

Since the process iteratively narrows in on a smaller set of models, there arises the possibility that the number of models might be reduced to an extent that it would not be possible to represent the residual uncertainty to any reasonable degree. Thus, a lower limit on the number of models available during any given iteration can serve as a stopping criterion for the process. Alternatively, when successive iterations reach a stage where the Bayesian updating either yields equiprobable clusters or does not yield any improvement on the prior probability, it would be reasonable to say that the process can be terminated and we have reached the maximum refinement in models possible for the given conditioning data. If additional data were to become available at a later stage, either for longer periods of time or from a different source, the model selection process can be continued using this refined set of models.

3.7. CONCLUSIONS

The model selection process provides an efficient way of addressing the uncertainties that are an integral part of any modeling / simulation process. It acknowledges that the reservoir model has large prior uncertainty associated with it that

might pertain to the reservoir structure, rock properties, distributions of various rock types, facies distribution within different rock types and even fluid properties. Given this significant prior uncertainty, the process of history matching amounts to implementing an efficient method to iteratively arrive at a smaller set of best-fit models that honor the characteristics observed in the dynamic field data. The residual uncertainty in the reservoir model can be assessed using the final set of models obtained by applying the model selection process.

One of the most important steps in the model selection process is the connectivity analysis of the initial model suite. This can be accomplished using conventional numerical simulators; however, given that the initial set of model might number several hundreds or even thousands of models, numerical simulation of such a large model set can become extremely computationally expensive. The computation cost can be significantly reduced if we are able to design a fast-transfer function that captures the essential physics of the process of fluid flow in the reservoir while being computationally efficient. This is especially feasible because the process of connectivity analysis does not need to be an exact replication of a numerical simulator; rather, it needs to be a quick assessment of how the geologic description of the model influences fluid flow and how models differ from each other in that respect. In the rest of this dissertation, we will discuss the development of two such fast-transfer functions, which enable us to capture the essential characteristics of fluid flow at a fraction of the computational cost of a numerical simulator.

Chapter 4 : Particle tracking proxy – I

As mentioned in the previous chapter, the efficient execution of the model selection process depends primarily on the method used for analyzing the connectivity characteristics of the initial model set. Full physics flow and transport simulators are computationally time consuming, and dramatically increase the implementation time of the entire model selection algorithm. In this chapter, we discuss the first of two fast-transfer functions we developed to approximately estimate the differences between prior reservoir models in terms of the movement of fluid within the reservoir.

4.1. RANDOM WALKER PROXY: AN INTRODUCTION

The random walker proxy described here is based on the Random Walker Particle Tracking described in chapter 2. The fluid flow through a reservoir is represented as an assemblage of random particles moving through a simulation grid. Multiple particles are introduced at an injection location and their movements are tracked through the grid over time. The movement of the particles from any grid location to any of the neighboring locations is dependent only on the current location of the particle. The actual movement of the particle is driven by a transition probability distribution, which depicts the inclination of a particle to make a transition to a neighboring grid block. This transition probability is loosely based on the RWPT equation defined in chapter 2:

$$X_p(t+\Delta t) = X_p(t) + \boldsymbol{u}(X_P,t) + \nabla \cdot D(X_P,t)\Delta t + B\big(X_p,t\big)\cdot \xi(t)\sqrt{\Delta t} \qquad (4.1)$$

where Δt is time step, $X_p(t)$ is the position of a particle at time t, $\boldsymbol{u}$ is the velocity of the particle, D is the dispersion tensor, B is the displacement matrix given as $^1/_2\,(BB^T)$, and $\xi(t)$ is a vector of independent, normally distributed random variables with mean 0 and

variance 1. Similar to this equation, our formulation of the transition probability also consists of a dispersion term and an advection term, in the following form:

$$Prob\big(A(t) \rightarrow B(t+\Delta t)\big) = \Delta N_{A(t)\rightarrow B(t)} + K_{avg,A\rightarrow B} \times \exp(\Delta P_{A\rightarrow B}) \quad (4.2)$$

where A is the current location of the particle, B is the target location of the particle, $N_{A\rightarrow B}$ is the difference in particle count between locations A and B, $K_{avg,A\rightarrow B}$ is the average permeability between A and B and $\Delta P_{A\rightarrow B}$ is the static pressure differences between A and B. The $\Delta N_{A\rightarrow B}$ term mirrors the dispersion term in equation 5.1, and the $K_{avg,A\rightarrow B} \times \exp(\Delta P_{A\rightarrow B})$ term is equivalent to the velocity term. These terms are described in more detail below:

a. *Difference in particle count:* Since the particles in this case are meant to represent the physical fluid flowing through the reservoir, the particle count is analogous to a concentration term. The higher the difference in particle count (or in terms of the actual fluid, the concentration gradient) between the current and target grid blocks, higher the probability of transitioning to that grid block.
b. *Permeability term:* The average permeability is taken as the harmonic average between the current and target grid blocks. Higher values of the average permeability translate to higher transmissibility between the grids, and thus a higher transition probability.
c. *Pressure differences:* The static pressure reflects the structure of the grid, and contributes to the movement of the particle between grid blocks that are not at the same vertical depth. The pressure is calculated from an initialization step of a numerical simulator.

Apart from the terms in the transition probability function, we implemented some additional constraints on the particle movements.

- Since the particles in this case are meant to be representative of the actual fluid, there is a volumetric limitation on the maximum accommodation space of every grid block. This limitation, related to its porosity, has been implemented as follows:

  If the injection rate be q m3/day, and represented by an injection rate of N particles/day, then the volume representation ratio (number of particles used to represent a unit volume of fluid at reservoir conditions) is given as N/q. The maximum accommodation $\mathrm{N}_{\mathrm{max}}$ is then calculated as:

$$\mathrm{N}_{\mathrm{max}} = \frac{\mathrm{V}_b \times \phi}{q/\mathrm{N}} \tag{4.3}$$

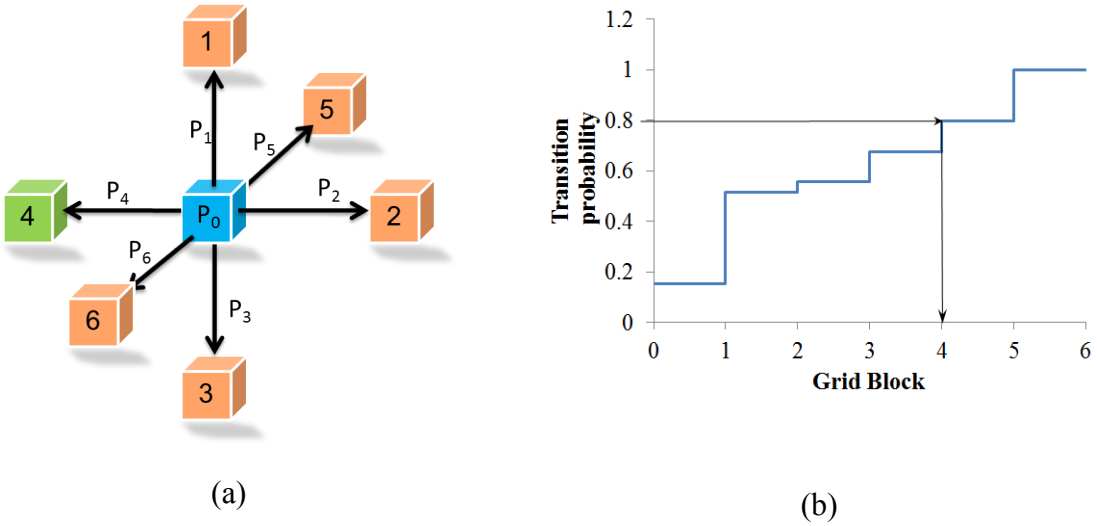
  Hence the particle movements are limited by a maximum particle count for every grid block. If the target grid block has already reached its maximum particle count, there can be no transition to the target grid block and the transition probability is zero.
- To account for the compressibility of the fluid, together with the probability of transitions to the neighboring grid blocks, we have also defined a probability of non-transition out of the current grid block. This has been defined as a function of the difference between number of particles in the current grid block and the maximum accommodation space of that grid block. Particle count closer to N_{max} accounts for higher values of probability of non-transition, and this value decreases as the particle count becomes smaller.
- There is also a lower limit on the number of particles that a particular grid block can accommodate, related to the critical saturation as follows:

$$N_{min} = \frac{V_b \times \phi}{q/N} \times S_{crit} \tag{4.4}$$

When the number of particles in a given grid block is less than or equal to N_{min}, all transition probabilities out of that grid block are equal to zero, and the probability of non-transition out of the grid block is 1.

It can be seen that we do not have a $\sqrt{\Delta t}$ term as in equation 5.1; however, since our formulation introduces uncertainty at both the dispersion level and the advection level, we do not need the additional of uncertainty term as the classical formulation.

This function is evaluated for every neighboring grid block (4 in the case of a two-dimensional grid, or 6 in the case of a three-dimensional grid). The movement of particles driven by transition probability is demonstrated in Figure 4.1. Suppose the current location of a particle is the central grid block, from where it can move to any of



(a) (b)

Figure 4.1. (a) Neighboring grid blocks for current position of particle, given by the blue block. (b) Transition probability distribution calculated from P0, P1 … P6 and sampled using Monte Carlo sampling. P0 is the probability of the particle not moving out if the block. In this case, sampling yields 0.8 which corresponds to grid block 3. Hence, the particle moves to grid block 3, shown in (a).

the six neighboring grid blocks. The transition probability is calculated for movement to each of the six grid blocks, together with a calculation of the probability of the particle

remaining in the same block. These are plotted as a cumulative probability distribution for transition probabilities, as shown in Figure 4.1(b). This distribution is then sampled using Monte-Carlo sampling. In the example presented below, the sampled value was 0.8, which corresponds to the particle moving to grid block 3.

For every particle introduced into the grid, it is followed through the grid until it reaches a location where either the current particle count is less than N_{min} or the transition probability sampling chosen does not move the particle from the current location. The movement can be considered a cascade, where the injected particle moves into a neighboring grid block and displaces a particle from there in a cascading effect, which is stopped only when a particle moving into a grid block that does not displace a particle out of that block. This process is repeated for all particles injected at that time step. At the end of the step, the particles are counted in every grid block which serves as an analogue for saturation or concentration. Then, an analogue for pressure is calculated from the particle distribution in the grid. This analogue is a function of distance from occupied grid locations and the particle count at these locations. It is represented by the following equation:

$$P_{analog}\big|_i = \sum_j \left(N_j d_{ij}^{-2}\right) \; \forall \, j \in \; locations \; with \; N_j > 0 \tag{4.5}$$

where N_j: number of particles at location j, d_{ij}: distance between locations i and j.

The inverse square distance term in the above equation comes from the solution of the diffusivity equation, given as:

$$\Delta P = \frac{qB\mu}{4\pi kh} \ln\left(\frac{4kt}{\phi\mu c_t r^2}\right) \tag{4.6}$$

The N_j term implies that higher the particle count (and hence fluid content) of a particular grid block, larger is the contribution of that location to the pressure analog.

After the initial injection step, there are two different implementations possible for moving particles in subsequent steps. The first is to inject another set of particles at the injection location again and track its movement through the grid, and add the new particle count to the existing particle count. However, such an approach would not reflect the fluid flow in the reservoir, because it is not just the injected fluid that moves in the reservoir but rather all available mobile fluid. An alternate strategy for moving particles in subsequent steps had to be implemented.

We consider all locations which have non-zero particle count. Because we are considering movement of *mobile* fluid only, we only consider locations that have particle counts greater than N_{min}. We first move particles from these locations, and then to mimic the continued injection into the grid, we also move new particles from the injection location. These new particle counts are added to existing particle counts and the process is repeated for subsequent fluid injection steps.

The random walker proxy described can thus create maps of particle count and pressure analog for each time step. In order to reduce computation and storage cost, we only record these maps at certain specific time steps. The particle count and pressure analog recorded at certain locations in the grid at these time steps can be used to represent each model for further operations, as described in section 3.4. The steps of the proxy are summarized in Figure 4.2, and the code is included in Appendix A.

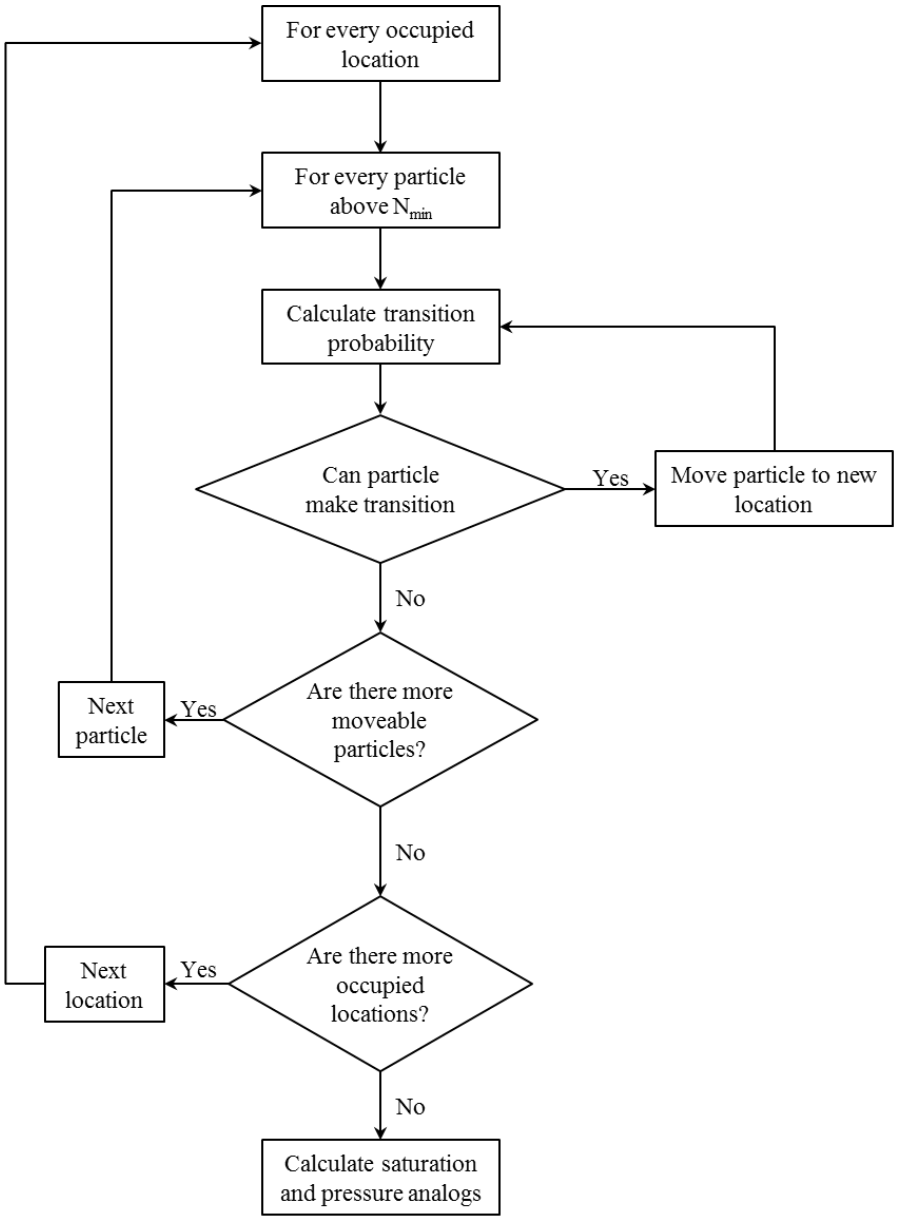


Figure 4.2. Flowchart of the connectivity proxy. This is the implementation for a single time step and the whole process will be repeated for multiple time steps.

4.2. COMPARISON OF PROXY RESPONSE TO NUMERICAL SIMULATIONS

The objective of the proxy is to analyze the connectivity of the reservoir model and its impact on the migration of CO_2 by correctly modeling the movement of fluid

within the reservoir. Hence, in order to test the validity of the proxy model, it was compared to the numerical simulation of some synthetic and field models. The comparison was performed on the basis of CO_2 saturations in the geologic models.

4.2.1. Comparison to synthetic models

The simplest model for comparison with numerical simulation was a horizontal layer-cake model consisting of some high permeability channels in a low permeability matrix. The high permeability contrast was used to ensure that the fluid flow was along crisp paths, which would make the visual comparison easier. The model used for this purpose was a 201 x 201 x 10 model, with injection at layer 5. The high permeability pathways had a permeability of 300 mD while the low permeability matrix was 0.1 mD. The model is shown in Figure 4.3.

Figure 4.3. Synthetic model used to compare areal migration of CO_2 in the numerical simulation model and using a proxy. Model has dimensions 201 x 201 x 10. The red channels are 300 mD and the white matrix is 0.1 mD. Fluid injection is at the center of the grid, in layer 5.

The simulated and proxy results are shown in Figure 4.4. The numerical simulation shows migration of the fluid primarily along the high permeability pathways, a behavior that was mirrored in the proxy implementation. However, there was also some movement of particles outside the clearly defined channel system, as seen from the blue regions on Figure 4.4 (b). The movement of fluid outside the channel (not seen in the numerical simulation) can be attributed to the $\Delta N_{A\rightarrow B}$ term representing the compressibility effect, which seems to be getting an inordinately high weight compared to the viscous terms ($K_{avg,A\rightarrow B}, \Delta P_{A\rightarrow B}$)

(a) (b)

Figure 4.4. (a) Simulation result on model from Figure 4.3, showing the saturation of CO_2 after 3 years of injection. (b) Result of proxy run for the same model.

The next synthetic model used for comparison of proxy result to numerical simulation sought to test the interplay of two competing forces during fluid movement: viscous forces and gravity. For this purpose, the model used had some well-defined channels while the entire reservoir slopes in one direction. The channels run in the NW-SE direction, while the reservoir structure slopes up towards the east. The model is shown in Figure 4.5.

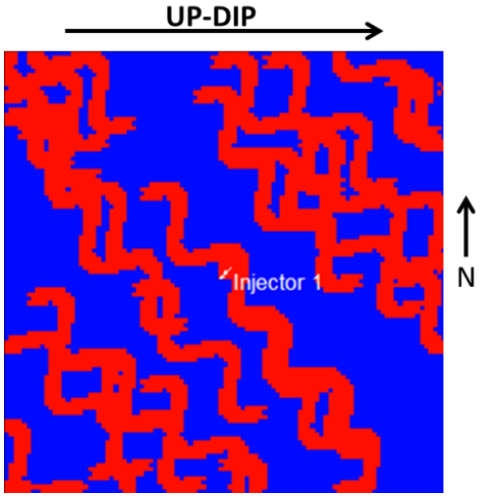
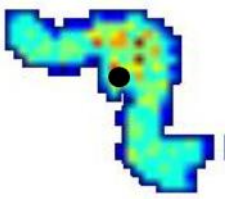
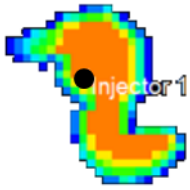


Figure 4.5. Model used for testing proxy behavior in the presence of competing forces: viscous and gravity. The red channels are high permeability (1000 mD), while the blue regions are low permeability (1 mD).

The simulated results and proxy results are similar in that they are both able to pick up the preferential movement of the injected fluid along the channel. However, the proxy result seems to be much less influenced by the gravity term than the numerical simulation, as evidenced by the preferential spread of the CO_2 in the up-dip direction.



CO_2 saturation based on proxy



CO_2 saturation based on flow simulation

Figure 4.6. Results of running numerical simulation and the proxy on the model in Figure 4.5. The black dot represents the injection location.

This indicates that, for a given case, it might be necessary to do some sensitivity studies on the proxy in order calibrate the proxy to capture the relative effects of the viscous and gravity forces adequately. The results are shown in Figure 4.6.

The results shown above demonstrate that the proxy seems adequate to capture the predominant movement of injected fluid along preferential flow paths. However, we still need to compare results for a case that represents a real field model.

4.2.2. Comparison to real field model

The In Salah project in central Algeria is a complex of gas fields that have been supplying natural gas to markets in southern Europe for almost a decade. The Krechba field is part of this project. This is a carboniferous formation at a depth of 1800 m (5905 ft.) below the surface. The formation contains a gas cap overlying a water leg. The natural gas in the gas cap contains about 10% CO_2, and cannot be sold without reducing the CO_2 concentration to less than 0.3% (Wright 2007). Rather than venting the CO_2 stripped out of the produced gas, it is being re-injected into the water leg of the formation using 3 horizontal wells. This is the production and injection scenario that we simulated using a compositional numerical simulator (CMG-GEM©), and then compared that response to what we get using the random walker proxy.

The reservoir model is a non-orthogonal corner point grid, created using the surface contour map for the top surface of the carboniferous interval (Figure 4.7 (a)). This model was populated with porosity values obtained by sampling from a map of reservoir quality (Figure 4.7 (b)). A porosity-permeability relationship ($k = 10^{-A+B\times\phi}$) was inferred from core data (BP internal communication) and was used to convert the porosity model to a permeability model.

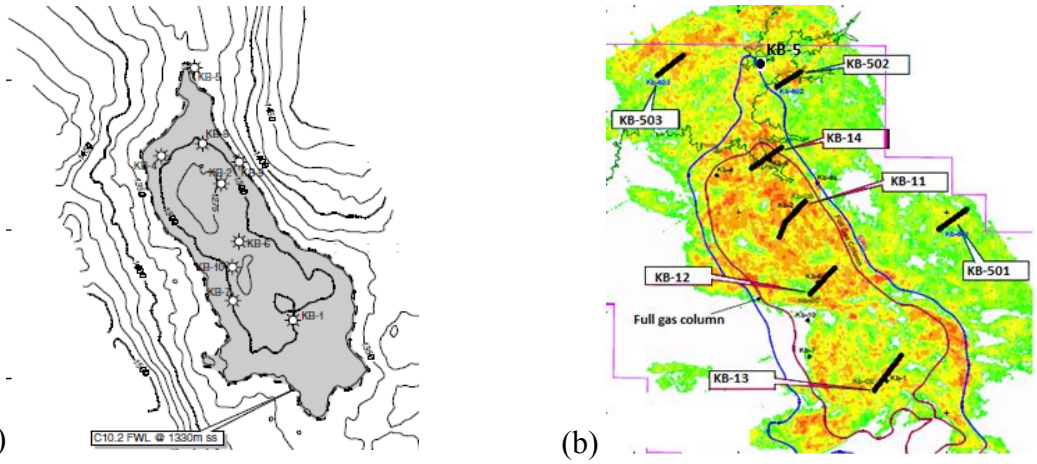


Figure 4.7. (a) Surface contour map at the top of the carboniferous formation, used to create the structure of the simulation grid (Davis et.al. 2001), (b) Reservoir quality map (Wright 2007), used to create maps of porosity. This porosity map was converted to a permeability map using a porosity-permeability relation

For the purpose of comparison, an injection location was defined coinciding with a real injection well in the Krechba formation. CO_2 was injected into the well for 5 years, and the simulated saturation at the top layer of the formation at the end of 5 years was compared to the proxy response. The results are shown in Figure 4.8. Once again, the proxy was able to capture the predominant migration behavior of the injected fluid.

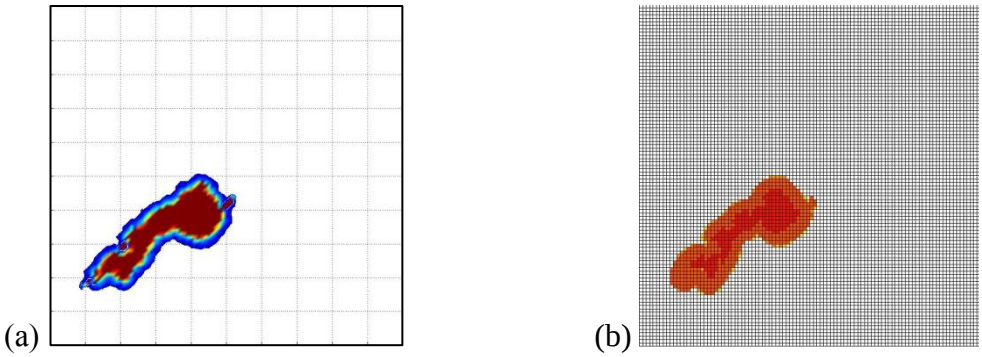


Figure 4.8. Comparison of CO_2 plume as inferred from (a) random-walk based proxy, and (b) numerical flow simulation.

While the comparison of the proxy to numerical simulation results is important for testing its validity, the real test for the proxy lies in its larger goal: that of being able to characterize the connectivity of models within the framework of the model selection process described in Chapter 3. As such, it is instructive to discuss the implementation of the entire model selection workflow using the proxy described in section 4.2. In the next section, we will demonstrate the use to the proxy to refine a set of models for the In Salah field.

4.3. IMPLEMENTATION OF PROXY WITHIN MODEL-SELECTION FRAMEWORK

As we described previously, the produced gas at the In Salah field contains about 10% CO_2, which is stripped out of the produced gas and re-injected into the aquifer underlying the gas-cap using three horizontal wells. An abandoned well location close to one the injectors (KB-502) showed traces of CO_2 at the wellhead, attributed to the injected CO_2. Tracer tests run on the three injectors showed that the gas was in fact coming from well KB-502, along a much faster migration path than had been previously anticipated (Ringrose et.al. 2009). This was hypothesized to be caused by a high permeability pathway close to the injector, causing rapid migration away from the natural up-dip direction. By applying the model selection process with the particle tracking proxy, we show that the injection well pressures, recorded before the CO_2 broke through at the abandoned well, still contained enough information to have enabled the inference of this high-permeability channel.

4.3.1. Initial set of models

The initial model set consisted of 400 models for the In Salah field that capture the initial uncertainty about the reservoir, they were constructed based only on well information available at the start of the project. Thus, the models were created using the background permeability as described in section 4.2.2, and then overlain by high permeability streaks. These streaks represented the high permeability pathway attributed to the rapid CO_2 migration, and were appropriate to use in the initial model set since there was evidence in drilling records of the presence of a NW-SE trending fracture network in the formation (Iding and Ringrose 2009). However, in the absence of direct information about the presence of a high permeability pathway close to any of the injectors, the streaks were created completely unconditioned to any hard data. A sample of models from the initial set is shown in Figure 4.9.

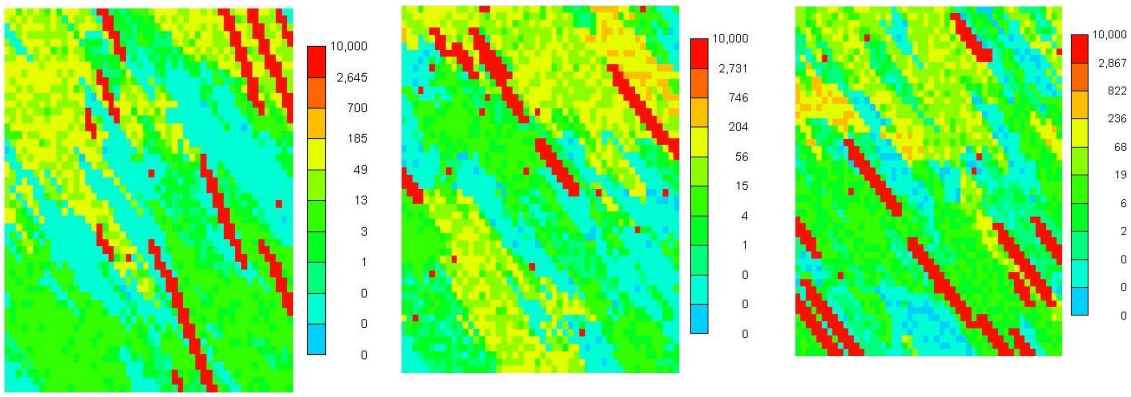


Figure 4.9. Sample reservoir models showing different high permeability streak direction

4.3.2. Connectivity analysis of models using proxy

All members of the initial model set were analyzed using the particle tracking proxy. For the purpose of discriminating between models, various statistics were recorded at four locations close to the injection well KB-502. These locations do not represent any real well locations within the grid; rather, they were meant to adequately capture flow behavior that would help distinguish models from one another. Since we knew that we were looking for features close to KB-502, the locations were chosen to be around that particular well (Figure 4.10). In a real field case, we might not have this information about where prominent features might be, and it would be necessary to have

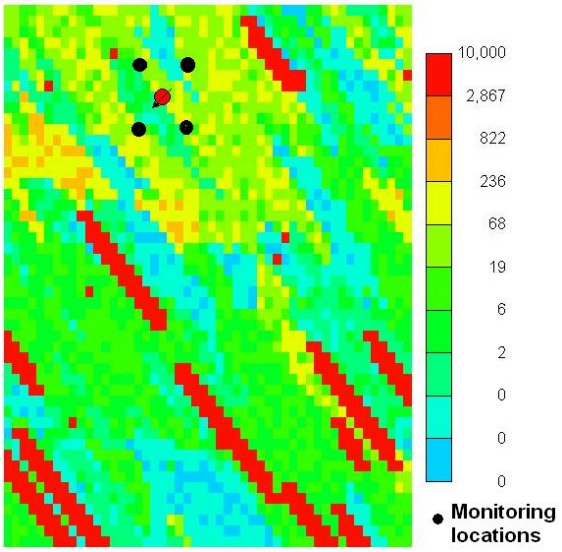


Figure 4.10. Locations chosen for recording proxy statistics. These locations are around the injection well of interest in this particular problem

a more generalized method to pick such locations for measuring proxy response. We will describe such a method later in chapter 7. For this application, we will demonstrate the method based on the 4 locations described above.

The statistics recorded at the monitoring locations were meant to capture the flow behavior of the models, driven by the predominant permeability features. Specifically,

they need to capture how rapidly the injected fluid migrated in a particular direction. The particular statistics recorded at each monitoring location were as follows:

- The particle count at the location, recorded at 4 different times during the injection period. This would reflect how rapidly the fluid is migrating in a particular direction
- An average value for the pressure analog. This reflects the average fluid distribution around the location.
- Time when maximum value of the pressure is reached at a monitoring location. This quantity is again related to the rapidity of fluid movement in a particular direction.

These statistics were recorded for each of the 4 locations, and then used to discriminate between the models as discussed below.

4.3.3. Model clustering

Eigen decomposition of the covariance matrix of the model statistics was performed, and the models were projected onto the orthogonal set of axes described by the leading eigenvectors. Efficiency of clustering η_k (details in section 3.4.2) was computed for different number of clusters for the projected models. The analysis revealed (Figure 4.11) that the ideal number of clusters in this case was 7. Hence, the models were divided into 7 clusters and representative models were found for each cluster. The clustering and the associated representative models are shown in Figure 4.12. The scatter plot shows how the representative models from different clusters of models highlight high permeability streaks in different parts of the reservoir.

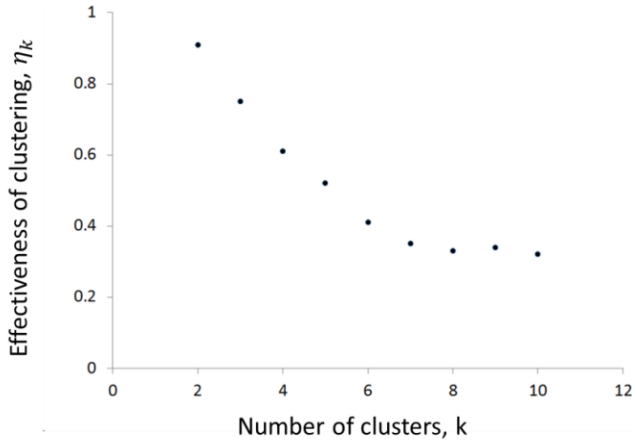


Figure 4.11. Plot of η_k versus number of clusters for In Salah.

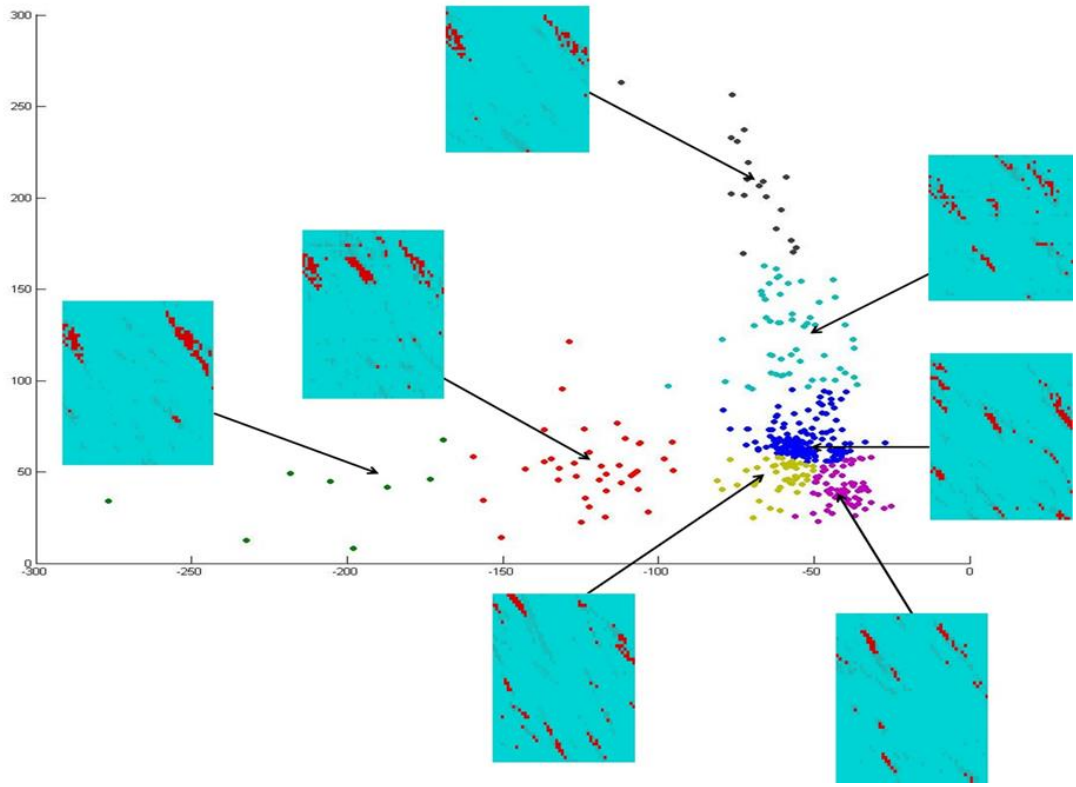


Figure 4.12. Clustering of models for In Salah, with the representative model shown for each cluster.

The representative models were run through a numerical simulator (CMG-GEM) and the pressure response at well KB-502 was used to compute the likelihood function for the Bayesian calculation discussed in Section 3.5.2. Since the objective of the entire process is to test if the data available before the breakthrough of CO_2 in the abandoned well contains information about the presence of high permeability channels close to KB-502, the injection data used for the model selection process is terminated at close to 600 days (approximate time of breakthrough). The rest of the available data is used to test the validity of the final set of models. The process of clustering and model selection was repeated for two iterations, to get a final set of 10 best-fit models. The results are discussed in the next section.

4.3.4. Results of the model selection process for the In Salah case

To test the validity of the entire workflow, the models in the final best matched cluster were each run through the flow simulator, and the results compared to the field data. The simulated results showed a good match not only to the conditioning part of the data (0 – 600 days) but also matched the prediction part (600 – 1600 days) of the injection data to a reasonable extent. The results are shown in Figure 4.13.

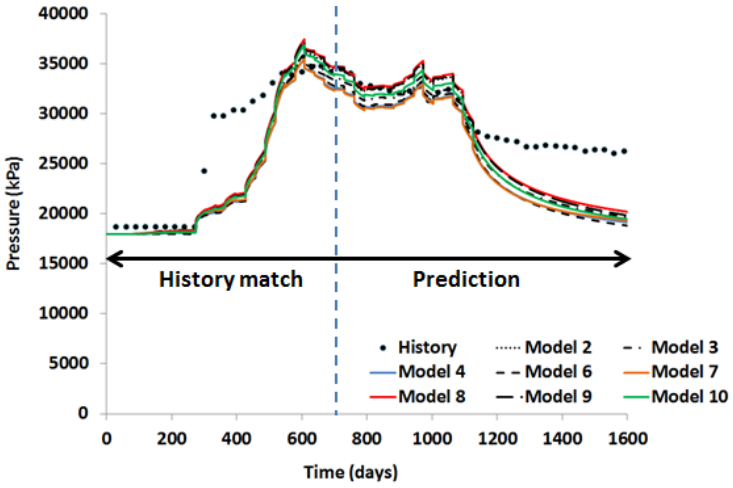


Figure 4.13. Comparison of bottom-hole pressure for KB-502 to field data. There is a reasonable match both to the history that was used for conditioning the model selection and the part of the injection data that was left out.

It would be further instructive to look at the individual models in the final cluster to examine their common characteristics. Some of these models are shown in Figure 4.14.

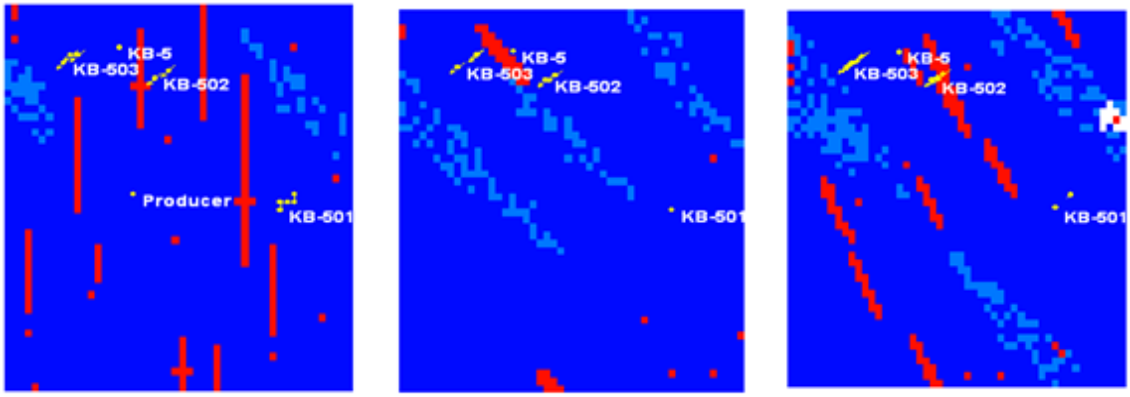


Figure 4.14. Sample of models from the final best-matched cluster of models. High permeability features are clearly visible in the vicinity of KB-502 in all models.

Visual examination of all these models shows that they all exhibit a high-permeability feature between well KB-502 and the abandoned well location (KB-5 on the maps). The

common characteristic of these models can be highlighted by calculating the ensemble average of the models in the selected cluster. Following the computing of this average, the models were transformed so that they all reflect the histogram of permeability inferred from the well data. This average transformed model is shown in Figure 4.15. The

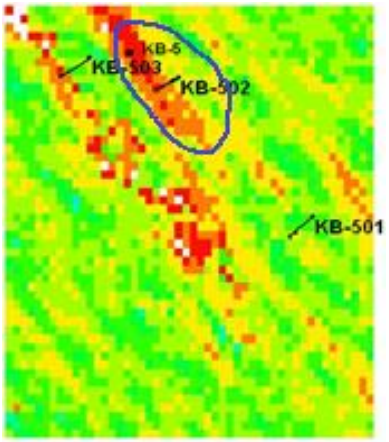


Figure 4.15. Ensemble average of all models in the final cluster. High permeability feature near KB-502 is clearly highlighted (circled).

models in the final set predominantly contain a high permeability feature close to KB-502 leading towards the abandoned well location. All other features in other parts of the reservoir are highly variable from one model to the next and so the ensemble average map does not highlight them. This indicates that while the available injection data can inform the process of finding prominent features close to the injector, it is not adequate to highlight such features in more remote parts of the model.

The model-selection process shows that the current version of proxy is adequate for representing the predominant migration patterns in reservoirs and thus capturing the difference in connectivity between models within the model-selection framework. However, the small thickness of the formation in this case (20 m) compared to the areal extent of the aquifer meant that the primary migration was areal, driven largely by

permeability. Given the buoyancy of CO_2, it can be expected that vertical migration of injected fluid would be a dominant mechanism in thicker formations. In the following section, we explore the ability of the proxy in cases where vertical buoyancy-driven flow is dominant.

4.4. REPRESENTING GRAVITY-DRIVEN FLOW USING THE PROXY

As mentioned earlier, the particle tracking proxy captures the structure of the reservoir using the initial pressure distribution in the model. This initialization pressure drives the movement of the injected CO_2 in the vertical direction (thus becoming a surrogate for buoyancy effects). We tested whether this current formulation for the proxy was adequate for representing gravity effects.

4.4.1. Application of proxy to synthetic models with strong buoyancy effects

The first synthetic model used for testing the validity of gravity driven flow was the same model as used in section 4.2.1 and in Figure 4.3. The model in this case consisted of 10 layers, and the CO_2 was injected in layer 5. So, there was some vertical movement of the injected CO_2. The result is shown in Figure 4.16, and compared to CO_2 saturation results from numerical simulation.

Fluid saturation from proxy Fluid saturation from numerical simulation

Figure 4.16. Cross-section through the reservoir at the injection location, showing the fluid saturation and the effect of buoyancy.

The results show that the proxy, while capturing the essential vertical trend in fluid flow, shows a lot more spread in the horizontal plane along that path. This seems to indicate that proxy needs to be calibrated so that the trade-off between the gravity and viscous effects is adequately captured.

To test this further, we simulated fluid flow in a two-dimensional grid with fluid movement driven only by gravity in the vertical direction. The model is a 101 x 1 x 100 model, with injection in the bottom-most layer. The model has uniform permeability, except for some very low permeability layers (Figure 4.17).

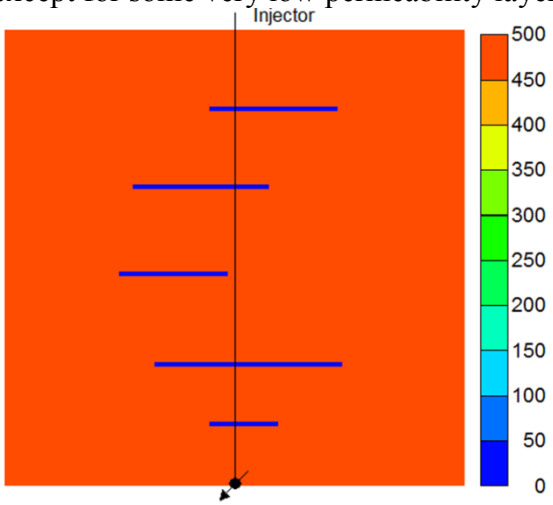


Figure 4.17. Permeability distribution in the second synthetic model used to test the proxy in case of gravity-dominated flow. The blue layers are shale baffles (0.1 mD) and the background is sand (500 mD).

The injected fluid is expected to move primarily in the vertical direction until it reaches one of the permeability baffles, and then it should move horizontally along the baffle until it is able to move vertically again. This behavior is clearly seen in Figure 4.18, which shows the saturation distribution from numerical simulation.

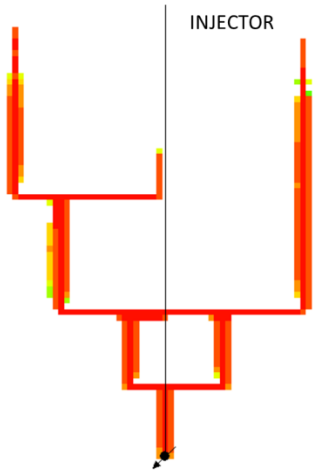
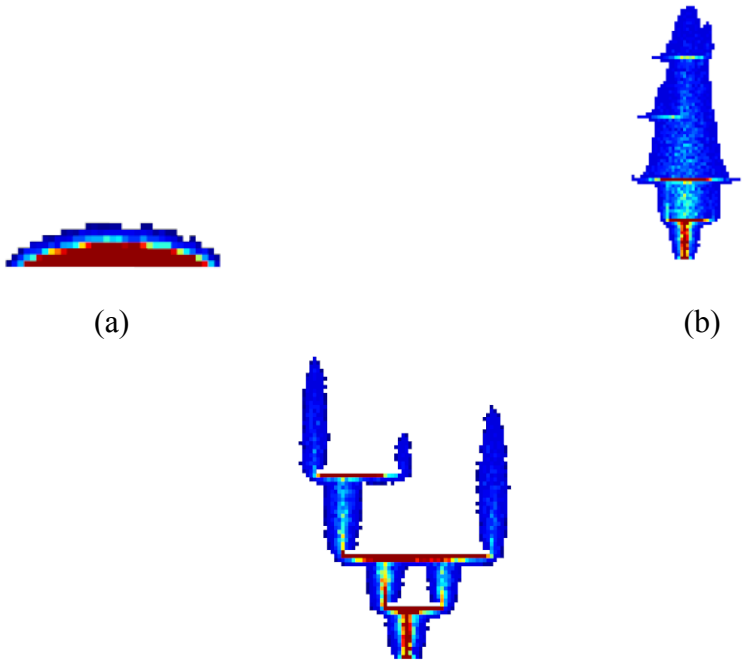


Figure 4.18. CO_2 saturation from numerical simulation on the grid in Figure 4.17



(a) (b)

(c)

Figure 4.19. Proxy results for model in Figure 4.17, showing the particle count (analogous to saturation) distribution in the model. (a) Original proxy, (b) Proxy with gravity term given extra weight, (c) Proxy with weighted gravity term and low permeability baffles that are assigned to be impermeable.

The proxy was run on the same permeability model. However, in this case, the proxy showed primarily areal migration with a little vertical movement (Figure 4.19(a)). When the transition probability in the vertical direction was arbitrarily weighted more than the transition probabilities in the horizontal direction, the vertical movement was captured a little better (Figure 4.19(b)); however, there was still significant migration of CO_2 into the low permeability baffles and so, the combination of horizontal movement below the shale baffles and subsequent vertical movement is poorly represented. Indeed, the only way to get close to the simulated response was to make the low permeability baffles impermeable (i.e. permeability equal to zero md), and simultaneously use a high weighting factor for the vertical movement (Figure 4.19(c)).

The results indicate that in order to use the proxy for such cases, calibration would have to be performed for each case before the proxy is implemented. The decision as to what cutoff value would make the low permeability layers impermeable would also need to be investigated.

4.4.2. Field case for testing the proxy when gravity effects are significant

While the synthetic models clearly showed that the proxy lacked the ability to adequately capture gravity-driven flow, we still needed to find out how it would perform in a real field case when there is both significant vertical migration and areal spreading of the CO_2 plume. For this purpose, we used the Utsira formation from the Sleipner field in the North Sea (Figure 4.20).

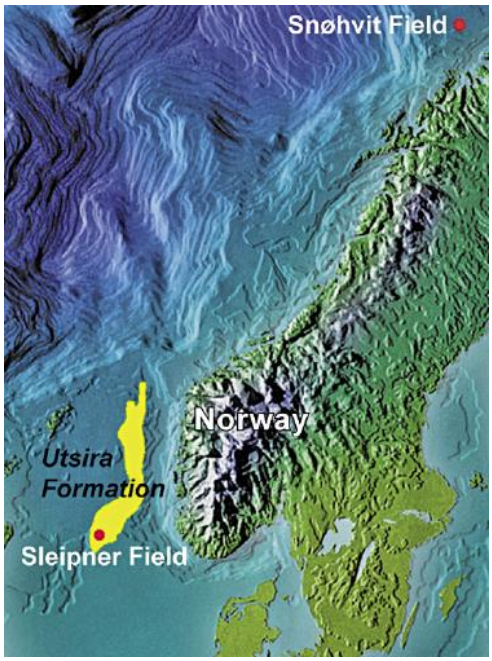


Figure 4.20. Location of the Utsira formation, off the coast of Norway (Arts et.al. 2008)

Description of the Utsira sand

Utsira is a late-Miocene / early-Pliocene formation, overlain by clay-rich sediments of the Nordland group and underlain by shaly sediments of the Hordaland formation. Utsira itself consists of a shaly top package, and a predominantly sandy bottom part. It is this lower part that is termed the Utsira sand. The sand is extremely permeable (permeability lies between 1 to 3 D), made up of overlapping fan-lobes and separated by thin mudstone drapes (Arts et.al. 2008). These mudstone layers are typically 1 to 1.5 m in thickness (Zweigel et.al. 2004) and extensive (Figure 4.21); however, they are not completely sealing due to the presence of 'holes' of erosive or deformational origin.

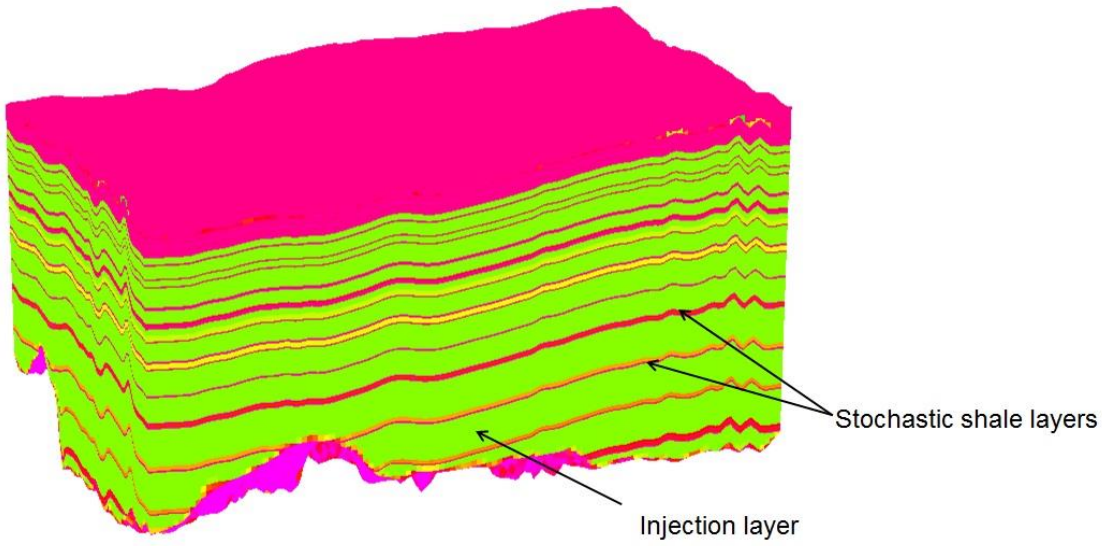


Figure 4.21. Simulation model for Utsira, showing the shale layers

CO_2 injected into the Utsira sand is sourced from the natural gas produced at Sleipner West field. The project was initialed in 1996, with injection rates of about 1 MT every year, using a single horizontal well injecting 200 m below the reservoir top (Chadwick et.al. 2012). The CO_2 rises almost vertically until it reaches one of the shale layers, then it migrates along the bottom of these layers till it reaches one of the sand 'holes' in the shale and then migrates upwards again. This has caused stratified CO_2 layers below the shales. Time-lapse seismic data shows the accumulation of the CO_2 plume below these shale layers as 'bright, sub-horizontal reflections', growing with time. This is shown in Figure 4.22.

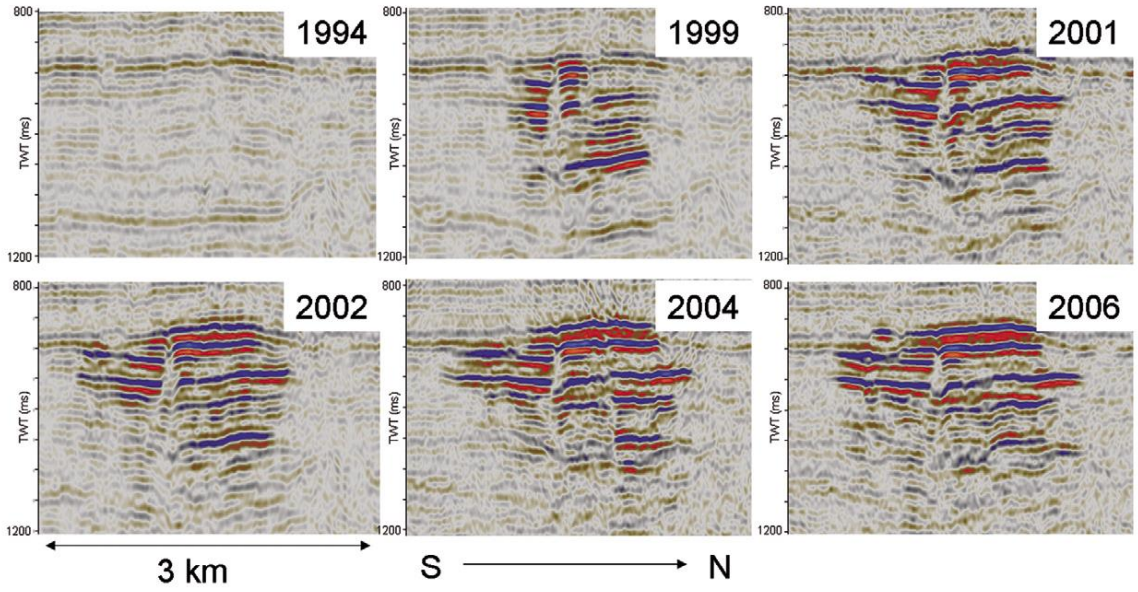


Figure 4.22. Time-lapse seismic data showing the evolution of the CO_2 plume over time and the accumulations below shale baffles (from Arts et.al. 2008).

It is clear that the distribution of the sand holes plays a major role in the movement of the injected CO_2. The locations of these holes are uncertain, so the objective of the work was to calculate probabilistic estimates of the location of these holes based on the results of the model selection approach conditioned to available injection data.

Unlike the injection data at In Salah, the bottom hole pressure at the injector was fairly constant in this case, since the sand is fairly uniform and of high permeability. The shale baffles only influence the vertical migration of the CO_2 plume and as such have very little effect on the injection pressure. Instead, the extent of the CO_2 plume as inferred from the time-lapse seismic data, at certain horizons in the reservoir interval was used for the model selection procedure. We will describe the application of the model selection procedure for this case in greater detail in a later chapter. At this stage, we will present some results for CO_2 distribution in the reservoir using the proxy in order to

investigate whether it is adequate for capturing the vertical migration of the injected CO_2 properly.

Results for CO_2 plume migration using the proxy and comparison to numerical simulation

Numerical simulation of the Utsira sand was implemented using a compositional simulator (ECLIPSE-E300©). The model was represented by 64 x 118 x 241 grid blocks, with injection in layer 182. The grid was a non-orthogonal corner-point grid. Injection was initiated in 1996, and continued through 2011 (when the last data was available from Statoil), after which the injected CO_2 was just allowed to migrate through the grid. The results of the simulation are shown in Figure 4.23.

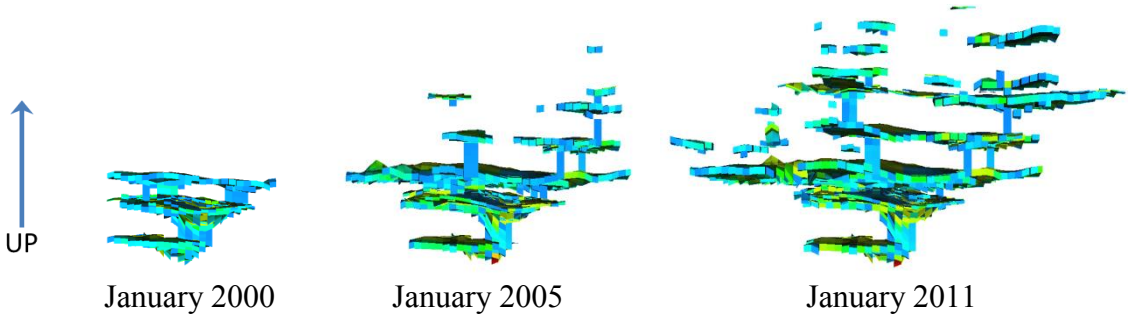


Figure 4.23. Simulated result of the migration of injected CO_2 in Utsira, showing CO_2 saturation at different snapshots in time, using E300 simulator. The z-axis has been exaggerated.

The simulation closely captures the characteristics of CO_2 migration seen in the time-lapse seismic (Figure 4.22). There is clear stratification of the CO_2 below the shale baffles. Our aim is to capture this behavior of the CO_2 in the proxy. The results are shown below.

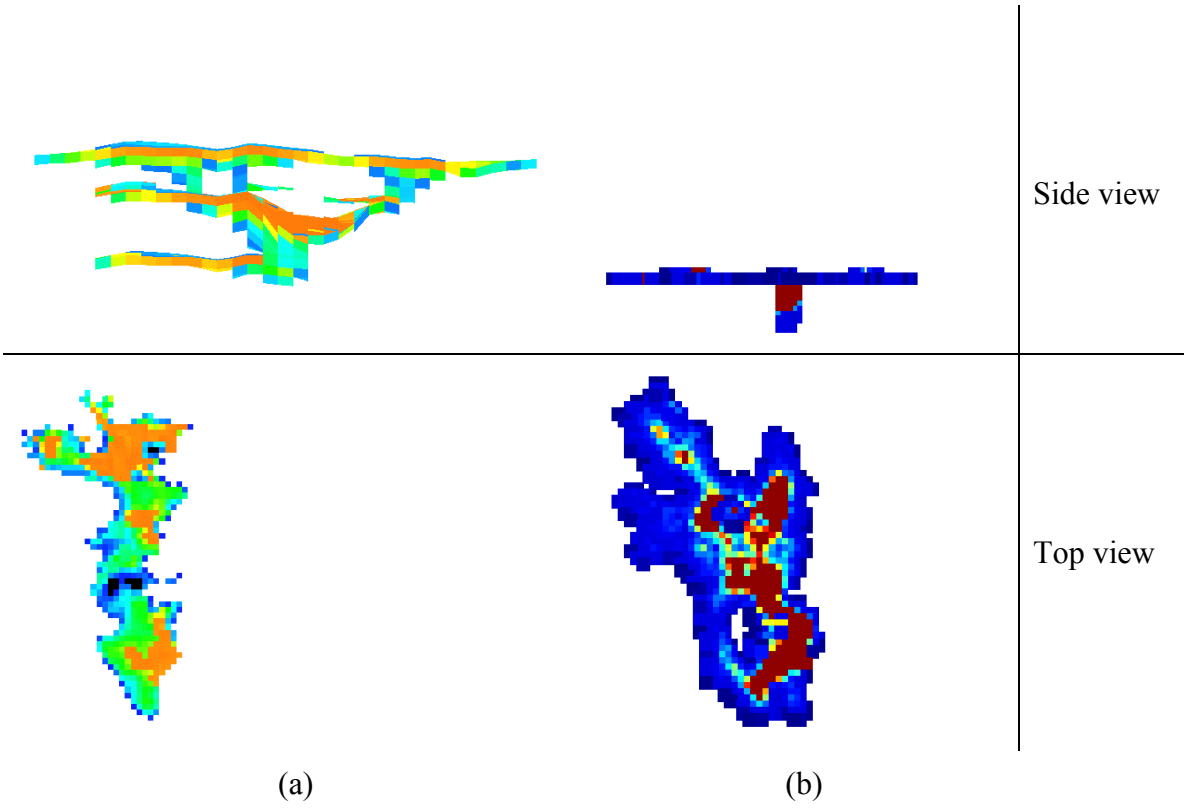


Figure 4.24. Comparison of proxy results (showing particle counts) with flow simulation results (showing CO_2 saturation) for Utsira. (a) Results obtained by flow simulation, (b) Proxy result.

From the discussion in the previous part (Section 4.4.1) it was quite clear that the proxy would not work effectively without assigning additional importance to the vertical migration term in the transition probabilities. Using this weighting approach, the vertical migration of CO_2 and subsequent spread under the shale was captured to a limited extent; however, as shown in Figure 4.24, even though there was some vertical movement of the CO_2 until the first shale baffle was encountered, the subsequent migration along a stratified flow paths is not adequately captured. Instead the plume is more diffused in the horizontal direction (darker blue in Figure 4.24 (b) Top view). This clearly shows that in

the presence of competing forces (gravity, buoyancy and viscous forces) the proxy becomes incapable of accurately representing the migration. It might be possible to overcome this drawback by adequately weighting the contribution of the various forces properly in the transition probability calculation (section 4.4.1 on page 69), but that would require a calibration exercise for *every* case that is evaluated by the proxy.

4.5. CONCLUSIONS

The random-walker based proxy provides a good approximation of fluid flow in the aquifer. The transition probability driving the flow of the fluid, primarily dependent on the heterogeneity and the initialization pressure, is adequate to capture large-scale viscosity-driven migration patterns. The use of this random walker within the model selection process gives it the necessary ability to differentiate between models based on their flow characteristics, at an extremely low computation cost compared to a full-physics numerical simulator.

However, the proxy suffers from some drawbacks when we implement it for several cases with varying driving forces such as gravity for CO_2 migration. The transition probability described in equation 4.2 reflects the local advection term, since it only looks at properties in the immediate neighborhood of the current location of the particle. As a result, though the areal migration is captured reasonably well, it still shows significant spreading of CO_2 away from the expected migration path, as seen in Figure 4.4. When strong buoyancy effects are prevalent, the proxy in its base form cannot adequately capture the gravity effects (Figure 4.19(a)). Adjustment to the transition probability by weighting the initial hydrostatic pressure more, and by artificially lowering the permeability of the shale baffles alleviates the problem to an extent. However, even

then, the proxy fails to capture the fluid migration adequately, as seen in the case for Sleipner. It should also be noted that multiphase flow effects described by relative permeability, variations in injection pressures and fluid density is not taken into account at all in this present formulation of the proxy.

Given the inability of the proxy to capture buoyancy-driven flow adequately, and the other factors described above, we felt it necessary at this stage to develop a new formulation for the proxy that would address these problems, and allow us to explore the model selection algorithm for the Sleipner case. This new proxy will be discussed in the next chapter.

Chapter 5 : Development of a New Particle Tracking Proxy

The particle tracking proxy described in the previous chapter worked quite well for capturing the areal migration of the injected CO_2, as demonstrated by the application to the In Salah project. The regional distribution of CO_2 modeled by the proxy compares well against that obtained using a full-physics flow simulation and the proxy responses are adequate for measuring dissimilarities between reservoir models within the model selection framework. However, it failed to adequately capture migration in cases where the CO_2 flow was primarily buoyancy-driven, as evidenced by its implementation to the Utsira case. Further, it does not capture all the physics associated with CO_2 migration. In this chapter, we will discuss the development and application of a new tracer-like particle tracking proxy that addresses the drawbacks of the previous formulation of the proxy. We will also show the applicability of the new proxy formulation to both real field cases discussed in the previous chapter - In Salah and Utsira.

5.1. New Formulation of Particle Tracking Proxy

The new proxy we developed is based on the approximation of tracer movement within the aquifer. The proxy still consists of moving particles through the aquifer, subject to certain rules dictated by static petrophysical properties and dynamic fluid properties. However, unlike the proxy outlined in the previous chapter, the particles are no longer representative of the fluid itself; rather, the particles are assumed to be mass-less, non-reactive tracers moving with the injected CO_2, subject to the same physics as the fluid itself. The entire migration period is divided into smaller intervals, and the pressures and saturations are assumed constant within each of these intervals (Figure 5.1). Particles are moved within these intervals under the influence of this pressure field,

conditioned to the static permeability distribution. The movements of the particles are kept independent of each other, and driven by a transition probability described in a later section. We keep track of all locations visited by each particle, and at the end of each interval, this gives us an estimate of the probability that a particular location within the domain will be visited by the injected CO_2, given a particular permeability distribution. The process of moving the particles is detailed below.

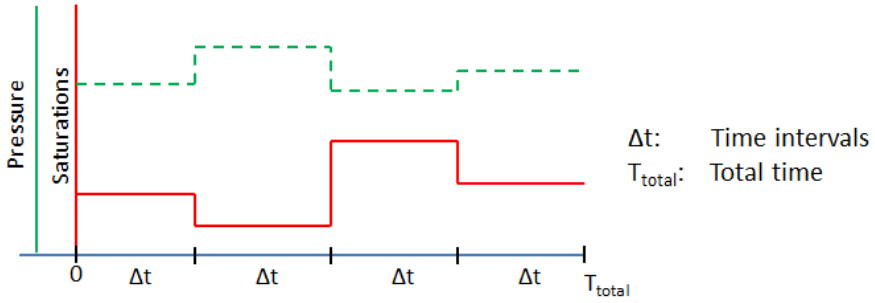


Figure 5.1. Schematic of pressure and saturation updates with time in the proxy

5.1.1. Revised particle tracking algorithm

The new formulation of the particle tracking is best described using an example. Consider a 5x4 grid (Figure 5.2), where 4 particles are injected at the top-left corner of the grid and allowed to migrate through the grid. The path followed by each particle is shown by the lines of different colors, and is the path followed by each particle is considered to be independent of the other paths. Each particle is allowed to undergo a predetermined number of (seven for this demonstration) displacement steps of equal length and we keep track of every grid block visited by each particle.

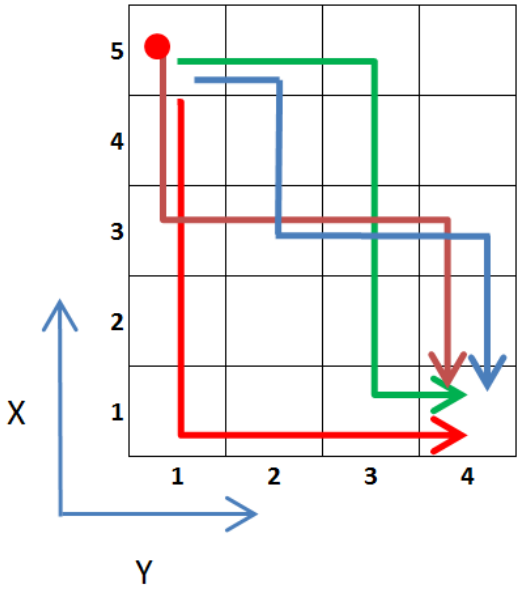


Figure 5.2. Movement of 4 particles through a 5 x 4 grid, as a demonstration of the new proxy formulation

After all the particles have been moved for a particular time interval, the number of particles that visited each grid block is counted, which can then be represented as a probability that a particular grid block will encounter the injected fluid (Figure 5.3 (a)). For example, grid block [X=3, Y=2] is crossed by the blue and brown paths; thus 2 of the 4 moving particles visit this location, and hence the probability of this grid block encountering the injected fluid is 0.5. This method is thus used to create a probability map at the end of each time interval, as is shown in Figure 5.3 (b).

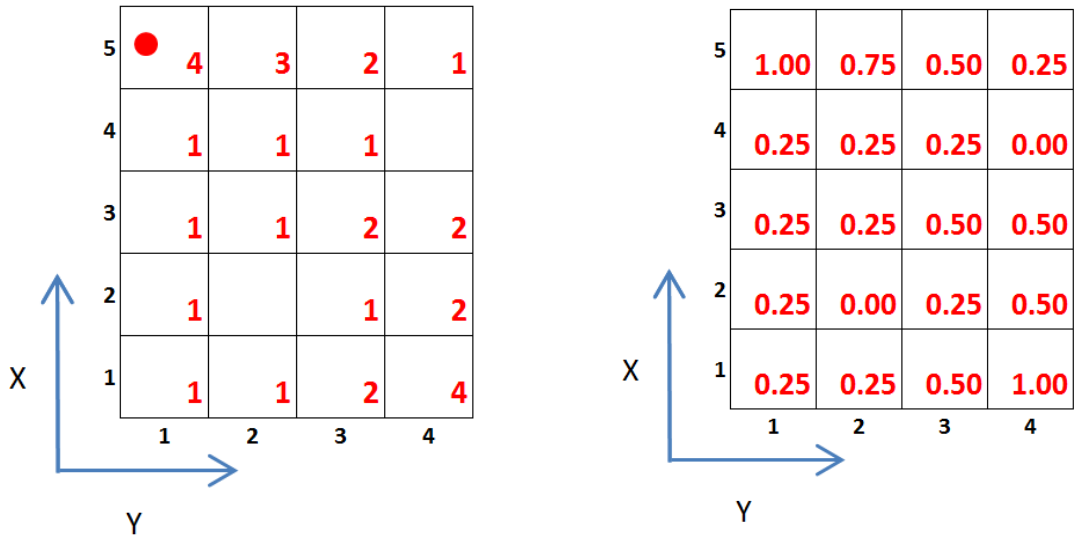


Figure 5.3. (a) Counting the number of particles crossing particular grid blocks, (b) Converting the particle counts to measures of probability.

Suppose a net volume V_{inj} has been injected into the grid. The fractional flow curve is used to get an estimate of the average saturation behind the CO_2-water front. We assume that the average saturation in a grid cell with probability exceeding a threshold is the same as this value from the fractional flow curve. Suppose this saturation is given by S_{avg}. Then, the number of occupied grid blocks can be given as:

$$N_{occupied} = \frac{V_{inj}}{V_b \phi S_{avg}} \tag{5.1}$$

where V_b is the bulk volume of a grid block, ϕ is its porosity and S_{avg} is the average saturation described previously. Now, starting from the highest value in the probability map (which will be 1), we define a cutoff value of probability, then count the number of grid blocks with probability value greater than or equal to the cutoff (given by N_{cutoff}. The cutoff is progressively decreased until N_{cutoff} is equal to (or nearly equal to)

$N_{occupied}$. All grid blocks with probability less than or equal to this cutoff is then assigned the saturation S_{avg}.

5.1.2. Redefined transition probability

The previous formulation of the RWPT relies on a transition probability – defined as the probability of a random particle to move from its current location to any of the neighboring grid locations. The transition probability controls the movement of the particles through the grid. In the previous implementation of the proxy, the transition probability is a function of the difference in particle count, average permeability and initialization pressure (or depth) gradient between the current location of the particle and the candidate location, and was formulated as:

$$P_{transition} = \Delta N_{co_2} + K_{avg} * \mathrm{e}^{\Delta P_i} \tag{5.2}$$

Here, $P_{transition}$ is the transition probability, K_{avg} is the average permeability and ΔP_i is the initial hydrostatic pressure difference between the grid locations. This formulation has been successfully demonstrated to enable model discrimination in the case of the Krechba formation in the In Salah gas field in central Algeria.

However, the above formulation shows a high degree of particle movement out of the primary migration pathway, due to the local nature of the transition probability function. Further, application of this formulation to the Sleipner CO_2 project was initially not as successful. In this case, it became evident that the effect of gravity was not properly captured by the specified transition probability (Equation (5.2)). We initially attempted to fix this problem by calibrating the proxy definition to numerical simulations and attaching more weight to the ΔP_i term. However, the calibration procedure defeats the objective of having a fast transfer-function model that can quickly screen the suite of

reservoir models and identify the similarity and dissimilarity between the models. It was therefore necessary to develop a more robust formulation of the random walker that would capture not just the effects of gravity and permeability, but all the physics relevant to the migration and trapping of CO_2 in the aquifer.

In order to better represent all the processes that contribute to the migration and trapping of CO_2 in the aquifer, we revisited the classical formulation of the random walker, as defined in Section 2.3.3 and mentioned in the previous chapter:

$$X_P(t + \Delta t) = X_P(t) + [\boldsymbol{u}(X_P, t) + \nabla \cdot D(X_P, t)]\Delta t + B(X_P, t).\xi(t)\sqrt{\Delta t} \qquad (5.3)$$

The important thing to note in this equation is that the randomness in the motion of the particle is imparted by the ξ term associated with B, which itself is associated with the dispersion tensor. Thus, the motion of particles using this equation largely depends on the solution of the velocity field and the randomness in motion is a function of the dispersion around this velocity field.

The important thing we learn from this classical formulation is that the transition probability has two parts associated with it: a diffusion term and an advection term. However, as opposed to the classical formulation of the random walker that needed the computation of the entire velocity field for every time step, we decided to calculate the velocity at every step as a random variable associated with the particular particle. Thus, in our formulation of the random walk, there is randomness in particle motion not just at the scale of dispersion but also due to uncertain velocities..

The new formulation of the transition probability we now started working with is calculated from the velocity (equivalent to a combination of the $\boldsymbol{u}(X_P, t)$ term and $\nabla \cdot D(X_P, t)$ term in equation 5.3) for the particular transition and normalized to get the probability, as follows:

$$P_{transition_{i\to j}} = \frac{v_{i\to j}}{\sum_{\substack{i:current\ location \\ k:neighboring\ locations}} (v_{i\to k})} \tag{5.4}$$

This equation shows that the transition probability is defined by the normalized velocity. The velocity is estimated using a calculation of the Buckley-Leverett velocity obtained from the fractional flow equation, and a macroscopic velocity term, as described in the subsequent sub-sections. It can be stated that this formulation is more similar to the master equation formulation for CTRW (Continuous time random walk) mentioned in chapter 2 than the RWPT formulation.

The use of deterministic velocity values to define a transition probability is made possible by invoking the condition of stationarity. The transition probability from a current block to the neighboring grid blocks could rigorously calculated if the flow simulation results of a large ensemble of models was available. Alternatively, we can invoke the assumption of stationary velocity in one single model, and calculate the transition probability by looking at all transitions between two nodes with the same separation and orientation as nodes i and j. However, even that would be expensive and so the calculation above assumes that the velocities required in Equation 5.4 have sampled all such transitions during their history and hence can be used to calculate the transition probability.

Fractional Flow Equation

Consider the displacement of a non-wetting phase by a wetting phase in a formation dipping at an angle α to the horizontal. The fraction of the wetting phase in the total mobile fluid is given by the following equation:

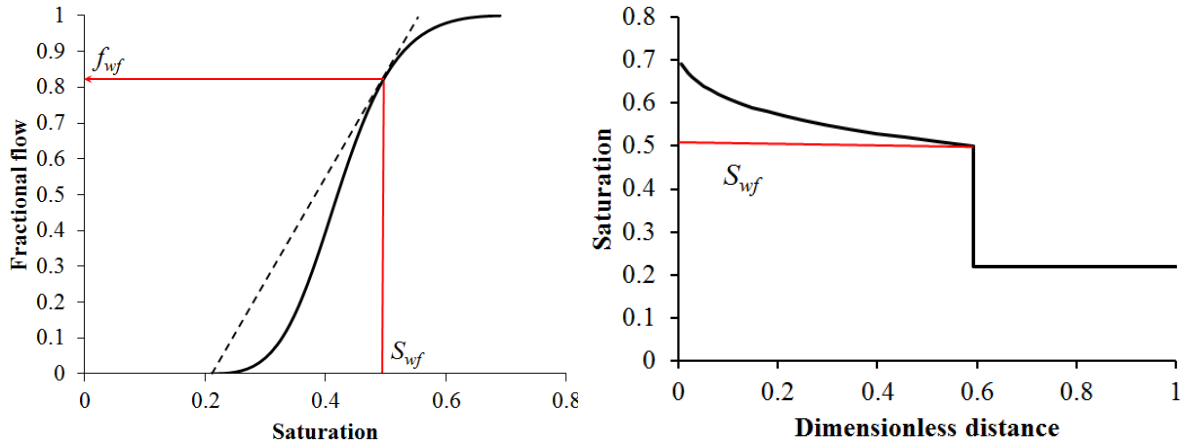
$$f_w = \frac{1 + (\frac{k k_{rnw} A}{q \mu_{nw}} (\frac{\partial P_c}{\partial x} - \Delta \rho g sin\alpha)}{1 + \frac{k_{rnw}}{\mu_{nw}} \frac{\mu_w}{k_{rw}}} \tag{5.5}$$

Here, k_r is the relative permeability, μ is the viscosity, P_c is the capillary pressure and ρ is the density. This equation can be used to create a plot of saturation against fractional flow. In the absence of capillary pressure, a closed form solution of the mass conservation equation for one-dimensional, two-phase immiscible displacement yields the specific velocity of a front of constant saturation. Using this solution, a tangent drawn from the point of initial saturation to this curve defines the saturation of the displacing front, and the slope of this tangent is the velocity of the displacing front. The slopes of tangents drawn on the curve at saturations higher than the front saturation define the velocity of those saturations (Buckley and Leverett 1942):

$$v_{BL} = \frac{\partial f_W}{\partial S_W} \tag{5.6}$$

This is shown in Figure 5.4(a). The distribution of saturation with distance from the injection location is shown in Figure 5.4(b). This definition of the Buckley-Leverett velocity is incorporated in our new formulation of the transition probability (Eqn (5.4) and Eqn (5.8)).

Fractional flow for CO_2 injection in aquifers



(a) (b)

Figure 5.4. (a) Front velocity from fractional flow curve, (b) Saturation profile with distance.

The fractional flow theory provides a method for computing the velocity term in the transition probability while incorporating the effects of gravity, capillary pressures and relative permeability. However, using the complete formulation of the fractional flow equation makes the random walker computationally expensive and hence a simplified formulation of the fractional flow equation was used, which ignores the capillary pressure and gravity terms:

$$f_w = \frac{1}{1 + \frac{k_{rnw}}{\mu_{nw}} \frac{\mu_w}{k_{rw}}} \tag{5.7}$$

The methods outlined thus far have accounted for diffusion and relative permeability. However, we still have not accounted for absolute permeability, buoyancy and compressibility. Furthermore, during the implementation of this improved version of the transition probability it was observed that the new proxy was computationally expensive. One reason for the increased computational expense is that in this new formulation, the walker does not have an estimate for the general direction of movement,

and so quite a few transitions of the walker would be towards the injector instead of moving away from the injection location. This motivates the addition of a macroscopic velocity that would define the general direction of movement – away from injection locations, along higher permeability pathways – which could then be combined with the Buckley-Leverett velocity derived from the fractional flow curve.

$$v_{i\to j} = v_{BL} \times v_{macro} \tag{5.8}$$

Next, we recognized that the particle also needed move out further from the injection location over successive time intervals. This is reasonable because the particle would be first moving through near well regions, where the CO_2 saturation is higher, which implies higher velocity through this region. Once it reached the outer edge of the fully saturated zone and moves into the two-phase brine-CO_2 region, its velocity would drop off. Once it reaches the uninvaded zone, it should have a low velocity (but not zero).

To mimic this spatial variation in velocity, we used the fractional flow term f_W described above. The fractional flow value f_W could serve as a direct indicator of the 'degree' to which we needed to speed up a particle moving through the two-phase region. We define the local particle velocity as:

$$v_{local_{i\to j}} = v_{i\to j} \times (1 + f_W) \tag{5.9}$$

For example, if the f_W value is 1.0 in a grid-block 100% saturated with CO_2, a particle moving through this grid block would be twice-as-fast as when moving through a grid-block not previously invaded by CO_2 (and thus with a saturation of 0.0).

To further speed up execution, we decide not just to move particles out of the injection location but rather out of all locations that had some cutoff value of saturation. This was reasonable since fluid movement does not stop after fluid has moved out of the

injector; indeed, fluid keeps moving over time as long as it has a potential gradient driving it.

Estimation of macroscopic velocity

The estimation of macroscopic velocity could be made using the pressure distribution in the aquifer, and then solving for Darcy velocity across each face of the grid block. The pressure distribution in the aquifer can be easily computed using a single-phase pressure solution using the following system of equations:

$$T \cdot P^{n+1} = B \tag{5.10}$$

Here, T is the transmissibility matrix, P^{n+1} is the pressure at a new time step, and B is the forcing function defined by well rates and pressure at the previous time step.

This pressure solution depends on the absolute permeability in the reservoir and the well rates, and will define a general direction of flow away from injection locations, towards producing locations (if any) and along higher permeability paths (if present). The Darcy velocity can also account for the gravity term.

$$v = \frac{kk_r}{\mu}\left(\frac{\partial P}{\partial x} - \rho g \Delta z\right) \tag{5.11}$$

or

$$v = \frac{[\Delta \mathrm{P} + \Delta \rho \mathrm{g} \Delta \mathrm{z}]\mathrm{k}_{\mathrm{avg}}}{\mu_{CO_2} L} \tag{5.12}$$

This formulation of the transition probability explicitly accounts for density difference, absolute permeability, fluid viscosity and grid block depths (and thus the gross features of the storage structure). The continuous point source in a three-dimensional domain (Raghavan 1993) was used to compute the pressure drop ΔP was:

$$\Delta p(x_D, y_D, z_D, t_D) = \frac{k^{1/2} qB\mu}{4\pi\sqrt{k_x k_y k_z} L r_D} erfc\left(\frac{r_D}{2\sqrt{t_D}}\right) \tag{5.13}$$

where

$$r_D = \frac{1}{L}\left[\frac{(x-x')^2}{k_x/k} + \frac{(y-y')^2}{k_y/k} + \frac{(z-z')^2}{k_z/k}\right]^{\frac{1}{2}} \tag{5.14}$$

$$t_D = \frac{kt}{\phi\mu c_t L^2} \tag{5.15}$$

For an isotropic formation, this can be simplified to:

$$\Delta p = \frac{qB\mu}{4\pi kr} erfc\left(\sqrt{\frac{\phi\mu c_t r^2}{4kt}}\right) \tag{5.16}$$

where

$$r = \sqrt{(x-x')^2 + (y-y')^2 + (z-z')^2} \tag{5.17}$$

This expression for ΔP incorporated into the transition probability allows us to account for injection rates, fluid viscosities, total compressibility and time. We initially defined *r* as a radial distance from the injector; however, this was modified later to account for the length of the path followed by the particle from the injector. Suppose a particle were to follow a highly tortuous path from the injector to its current location. Then, even if the final distance of the particle from the injector is not very large, it can be expected that the pressure drop that the particle experienced moving along the highly tortuous path would be much larger than the radial pressure drop from the injector. Hence, it made sense to consider the total path length in this calculation of the ΔP term. Using the same idea, the average permeability was also considered along the path taken by the particle to reach the current location. This approach also addresses one of the

problems noted about the previous proxy formulation: that of taking only local values into consideration for the calculation of transition probabilities.

Thus, summarizing the steps outlined above, the transition probability is defined by the following equations:

$$P_{transition_{i\to j}} = \frac{v_{local_{i\to j}}}{\sum_{\substack{i:current\ location \\ k:neighboring\ locations}} v_{local_{i\to k}}} \tag{5.18}$$

where

$$v_{local} = v_{i\to j} \times (1 + f_W) \tag{5.19}$$

$$v_{i\to j} = v_{BL} \times v_{macro} \tag{5.20}$$

$$v_{BL} = \frac{\partial f_W}{\partial S_W} \tag{5.21}$$

$$v_{macro} = \frac{[\Delta P + \Delta\rho g \Delta z]k_{avg}}{\mu_{CO_2} L} \tag{5.22}$$

$$\Delta P = \frac{qB\mu}{4\pi k_{avg} r} erfc\left(\sqrt{\frac{\phi\mu c_t r^2}{4k_{avg} t}}\right) \tag{5.23}$$

It should be pointed out that this formulation contains all the physics of particle movement within a single equation, which makes it more similar to the master equation formulation of the Continuous Time Random Walk. At the same time, the movement of particles on a gridded system makes the implementation closer to the Random Walker Particle Tracking. The major point of departure from the classical random walker formulations is that it introduces uncertainty both at the advection and diffusion level, while the RWPT relies on a deterministic solution of the advection velocity with uncertainty reflected only at the diffusion scale.

Particle migration and step size

This modified implementation of the proxy was tested on a single synthetic model to compare it to flow simulations. The model is shown in Figure 5.5 (a). The random particles moved a fixed number of steps during each time interval. While this proxy was able to capture the influence of heterogeneity on migration quite efficiently, and without spreading out of the flow path as seen with the proxy formulation in the previous chapter, the size of the swept volume as a function of *time* is still a bit inaccurate when compared to the flow simulator response. As seen in Figure 5.5 (b) and Figure 5.5 (c), even though the proxy tracked the channel quite effectively, it underestimated the spread of the fluid along the channel towards the north-west and overestimated the fluid movement close to the injector.

It is important for the proxy to not just capture the movement of fluid along permeability pathways, but also how this migration occurs in time. The fluid would have higher velocities and thus move farther in higher permeability zones than in lower permeabilities, and thus, if the step sizes (distance the particle moves when going from one grid block to the next) of the particles were equal, over a fixed time interval, particles

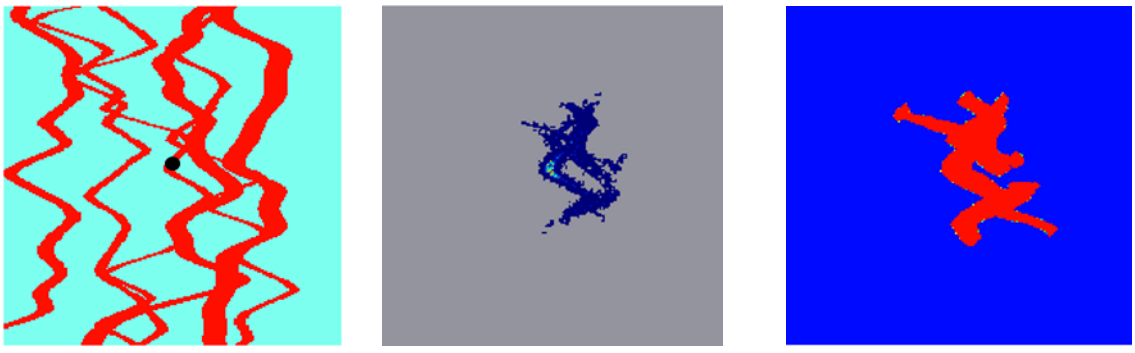


Figure 5.5. Comparison of proxy response with simulator, initial formulation with FIXED NUMBER of steps. (a) Permeability distribution, with injection location in black, (b) Proxy Response, and (c) Simulator response.

should travel variable number of steps depending on the velocity of the particle.

To make the number of steps variable, the velocity in Equation (5.19) was used to find the 'transition time' from one grid block to the next. Using the time intervals shown in Figure 5.1, the particle was allowed to move as long as:

$$Cumulative\ transition\ time, t_{trans} = \sum \frac{d_{i\to j}}{v_{local_{i\to j}}} \leq \Delta t \tag{5.24}$$

where $d_{i\to j}$ is the distance between grid block *i* and *j* and Δt is the time interval

We recognized that this calculation of cumulative transition time would not be exact, so a window of 10% of the interval length was used such that a particle was allowed to make transitions up to ±10% of the total interval length:

$$0.9 \times \Delta t \leq t_{trans} \leq 1.1 \times \Delta t \tag{5.25}$$

5.1.3. Summary of New Formulation of Proxy

The steps described above for moving the particles and for calculating the new transition probability can be summarized by the flowchart given below:

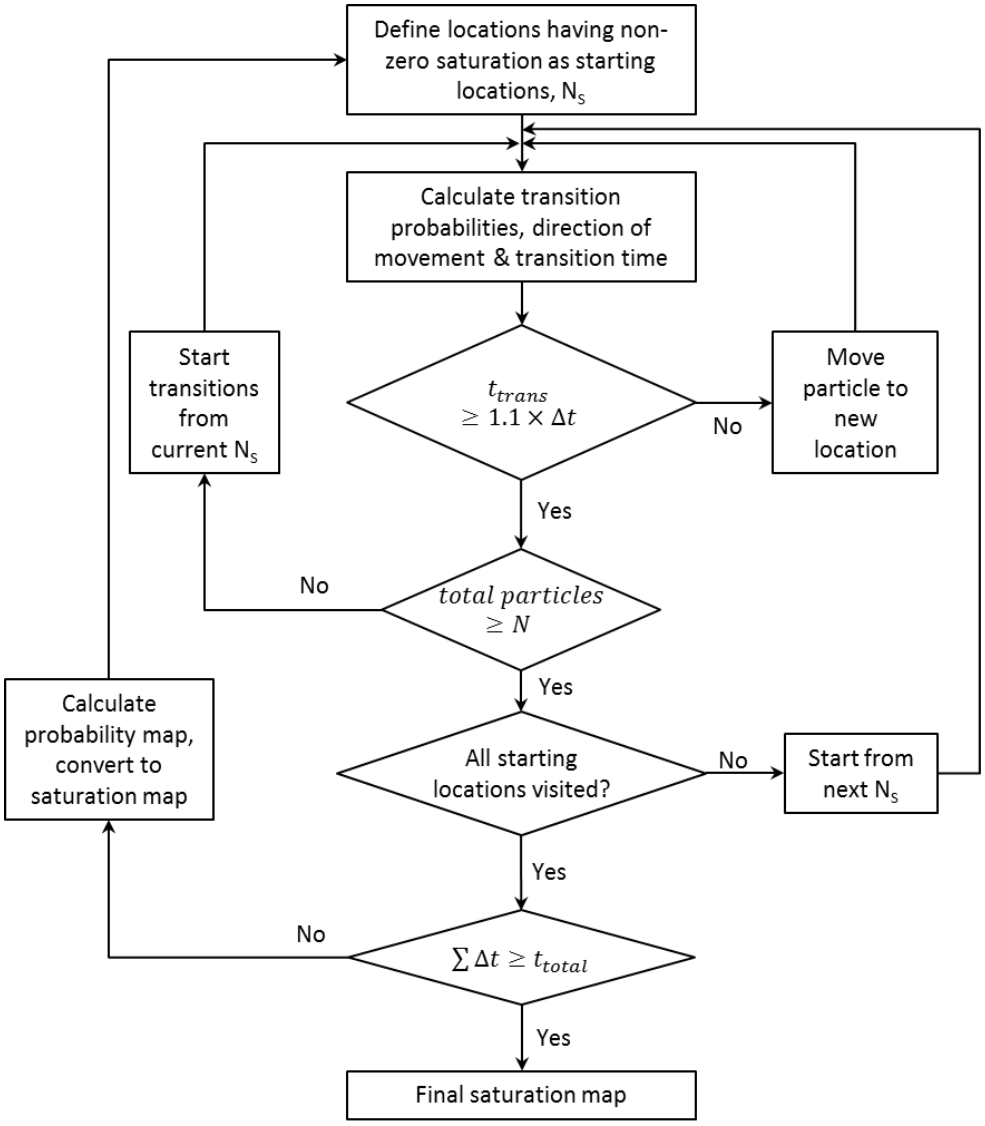


Figure 5.6. Flowchart of new formulation for proxy

5.2. VALIDATION OF NEW PROXY FOR CAPTURING AREAL MIGRATION

This new implementation of the proxy was first validated for cases where the migration of the injected CO_2 was primarily areal with only a little vertical migration at the beginning of injection. Validation was performed for two different models: the first a

synthetic model, similar to the one used in Section 4.2.1, and the second a model for In Salah, discussed in section 4.2.2. These validation cases will be discussed in this section.

5.2.1. Synthetic model for validation of proxy

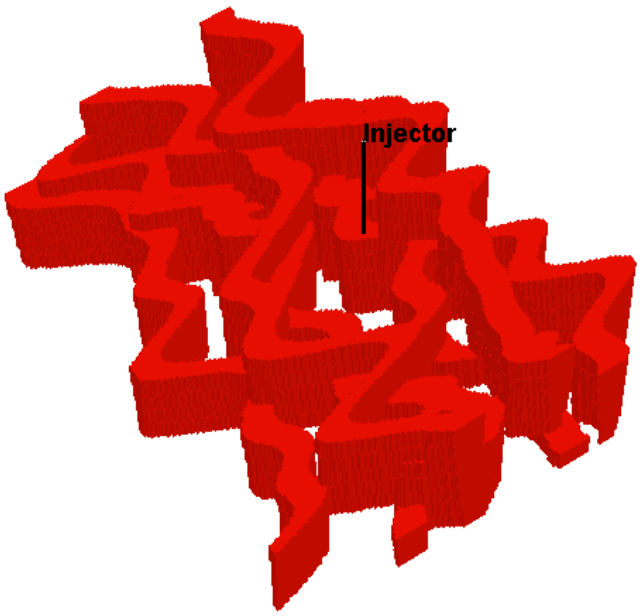


Figure 5.7. Permeability model used for validation of new proxy formulation. This is the same model as was used for validation in section 4.2.1 for the old proxy.

The model consists of 201 x 201 x 10 grid blocks, with injection in the fifth layer. Even though there is some scope for vertical migration of injected CO_2, most of the fluid movement is along the top of the reservoir (below the impermeable cap rock) where it is largely driven by the high-permeability channels seen in Figure 5.7. The fluid properties and other relevant data for the simulations are the same as presented in Section 4.2.1.

(a) (b) (c)

Figure 5.8. Migration of injected fluid captured by (a) New proxy formulation, (b) Numerical simulation, (c) Old proxy formulation

The areal migration was captured quite effectively by the tracer-based proxy as seen in Figure 5.8. The migration pattern obtained using the particle tracking proxy resemble the results of the numerical flow simulation using CMG-GEM© quite closely. There is little spreading of the injected fluid out of the high permeability pathways. This is a clear improvement over the earlier proxy that indicated significant migration of fluid away from the high permeability pathways (as seen in the comparison shown in Figure 5.8). Further, the temporal aspect of the migration, as discussed in section 5.1.2 is also successfully captured by the tracer-based RWPT. This is clearly seen in Figure 5.9, where the areal migration pattern is compared at three different snapshots in time.

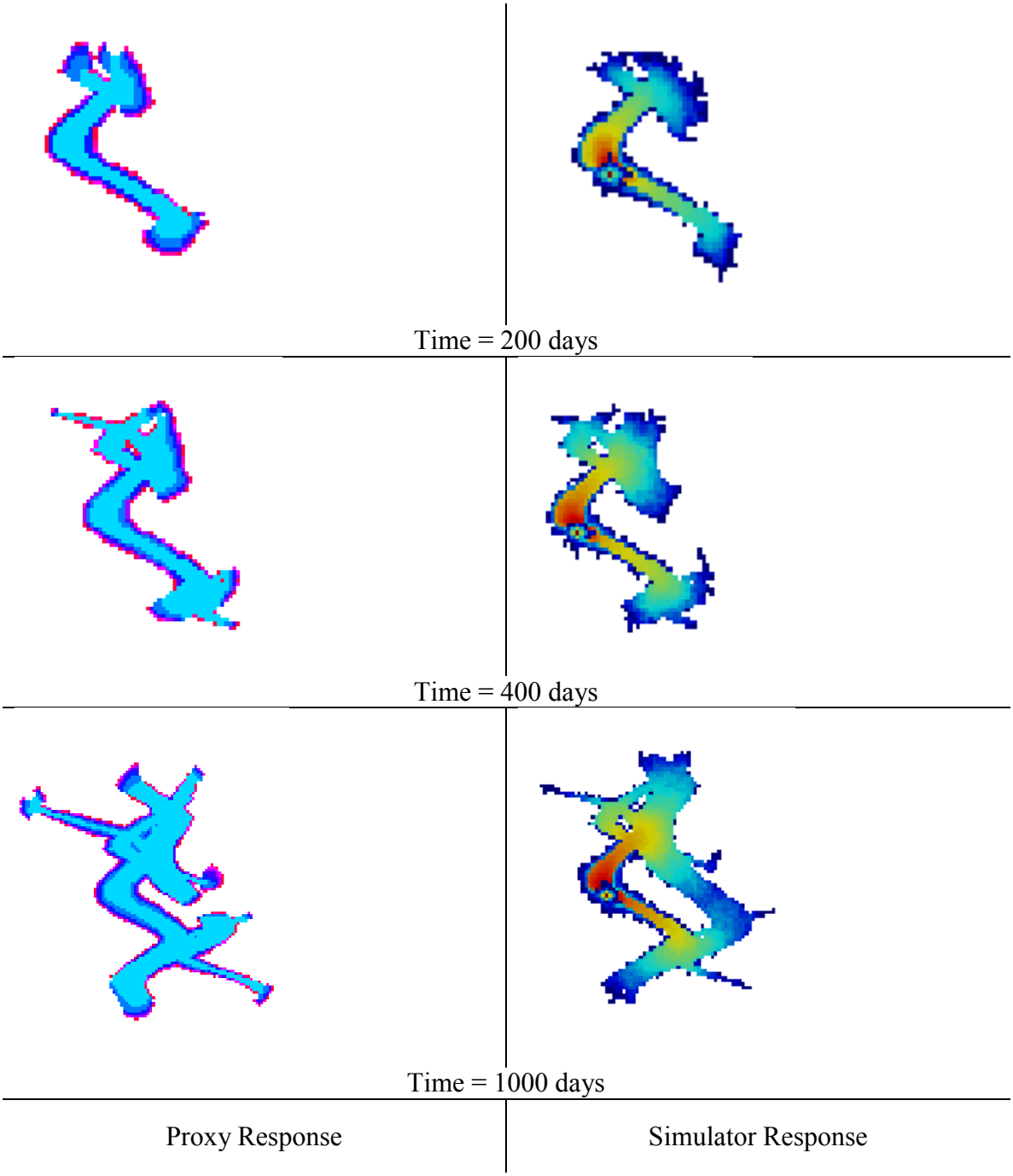


Figure 5.9. Comparison of proxy and simulator response. The migration of CO_2 over time is captured accurately by the new proxy.

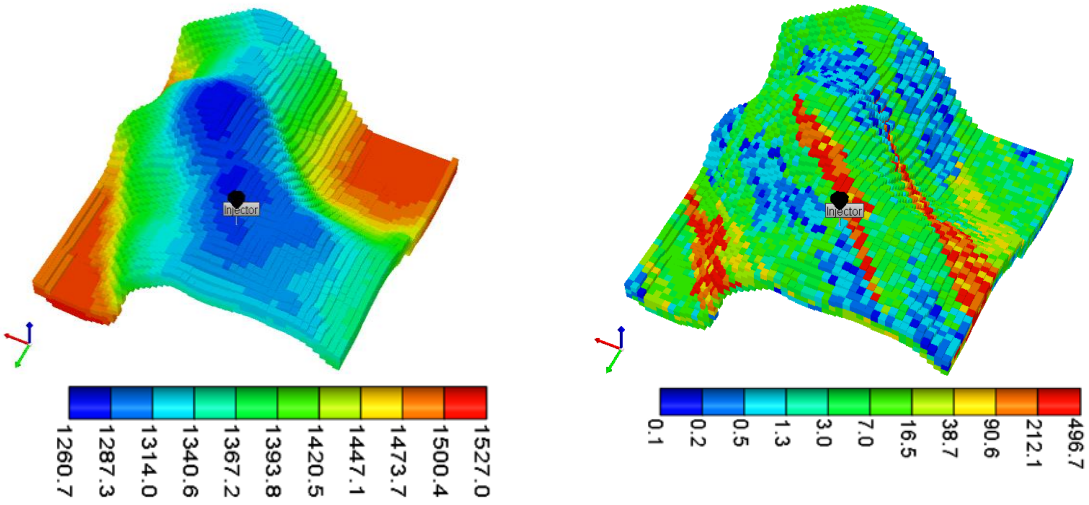
The last part of the comparison looks at the vertical distribution of CO_2 close to the injection location, small as it is. The new formulation for the proxy was able to capture the buoyant flow much more effectively than the old proxy. This is clearly seen in Figure 5.10, where cross-sections through the injection location for both cases (new and old proxy formulations) are compared to numerical simulation results. The new formulation is clearly able to capture the vertical movement much more efficiently, without the need for any arbitrary weights. The diffuse nature of the vertical movement in the case of the fluid-based RWPT once again highlights the inability of the old proxy to properly weigh the viscous and gravity forces against each other.

New Proxy Response | Simulator Result | Old Proxy Response

Figure 5.10. Cross-section of fluid saturation at the injection location for buoyancy-driven flow for the new and old formulation of the proxy, compared to numerical simulation results.

5.2.2. Field model for validation of tracer-based proxy: In Salah

The In Salah model described previously was tested using the tracer-based proxy and compared to results from numerical simulation. The model chosen for the purpose was based on the In Salah structure, but without the initial fluid distribution seen in section 4.2.2; rather, the model only had brine in-place with CO_2 injection through a well coincidental with KB-502. The purpose of the exercise was to demonstrate that the new proxy formulation is able to capture the major areal migration pathways, in the presence of competing forces due to permeability features and the reservoir structure. The model is displayed in Figure 5.11.



Depth from surface to top of grid (m) Permeability (mD)

Figure 5.11. Model used for testing the tracer-based proxy on a real field case (In Salah).

Using this input model and the same injection location, the new proxy formulation was tested. The result is shown in Figure 5.12. It is clear that the proxy is able to capture migration along the prominent heterogeneity feature responsible for the movement of the injected CO_2. Further, there is preliminary indication that it is able to adequately capture the interplay of gravity (reflected by the structure of the reservoir in this case) and permeability, so that fluid movement is along the high permeability *and* towards the top of the structure. There is some spurious particle incursion outside the region indicated by the numerical flow simulation. This is because of the stochastic nature of the particle tracking algorithm. There is however, enough similarity between the two results to make the proxy a viable fast alternative to the numerical simulator.

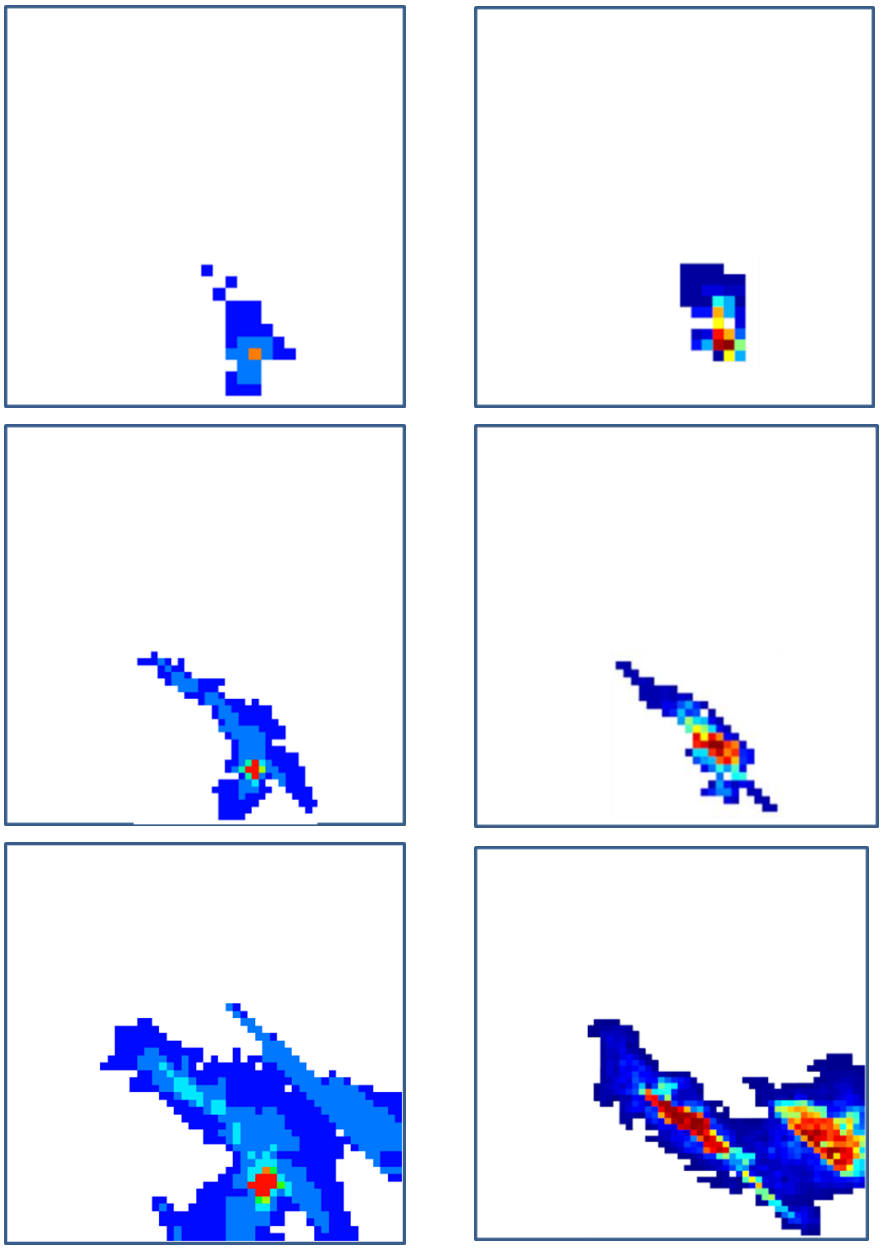


Figure 5.12. Comparison of results from numerical simulation and proxy, showing the saturation in the grid at certain snapshots in time.

5.2.3. Model Selection applied to In Salah

As a final test of the effectiveness of the model selection process, we implemented the model selection process in its entirety using the new proxy on a set of models for In Salah. The initial model set consisted of only a hundred models, which were conditioned to the injection pressure at KB-502. Similar to the implementation in the previous chapter, the initial model set consisted of a low background permeability (between 10 mD and 300 mD, as reported at In Salah) overlain by high permeability streaks to represent the fracture network, as shown in Figure 5.13. To make the demonstration robust, the streaks were aligned in different directions. It should also be noted that the streaks were not conditioned to data and were not forced to occur in the vicinity of the well under study.

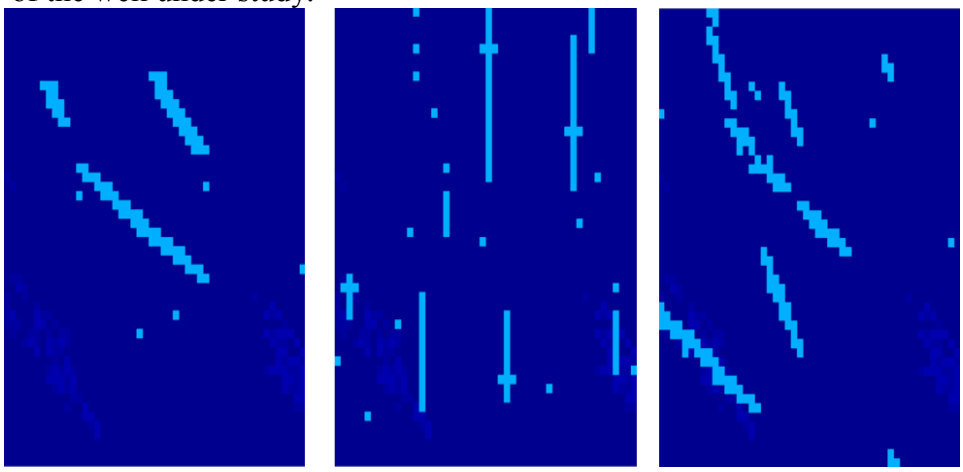
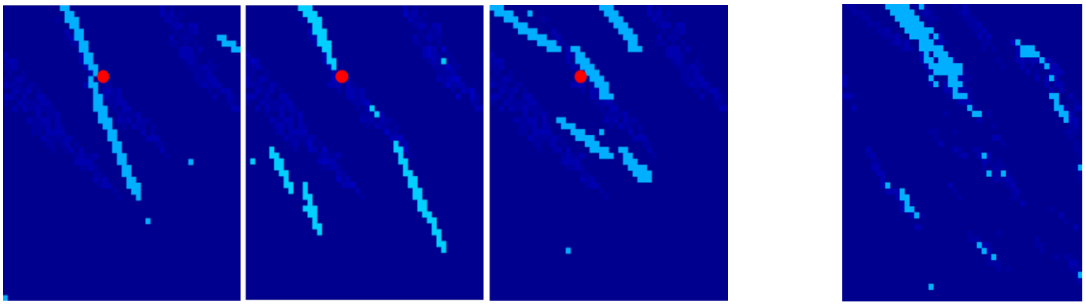


Figure 5.13. Sample of models from the initial set for In Salah. The high permeability streaks are shown in light blue.

The models were taken through a single iteration of the model selection process and the best-fit models at the end were analyzed. The models consistently showed a high permeability feature near the problem well, KB-502 (Figure 5.14 (a)). This effect across all models in the best-fit cluster becomes more apparent when we find the average model

for that cluster. As can be seen in Figure 5.14 (b), the average model shows a high permeability pathway between well KB-5-2 and the abandoned well location.



(a) (b)

Figure 5.14. Final best-fit models for In Salah. (a) Sample of models from best-fit cluster of models, (b) Average of all models in final cluster

5.3. VALIDATION OF NEW PROXY FOR CAPTURING VERTICAL MIGRATION OF CO_2

In the previous section, we found that the new proxy formulation captures the areal migration of CO_2 adequately and is also able to capture the gravity-driven flow of CO_2, for example in the In Salah (Figure 5.12) case. However, we still needed to properly test the ability of the proxy to accurately represent flow in the vertical direction due to gravity effects. For this purpose, we used synthetic and field models, similar to the ones used for testing the old proxy in section 4.4.2.

5.3.1. Validation of tracer-based proxy using synthetic model

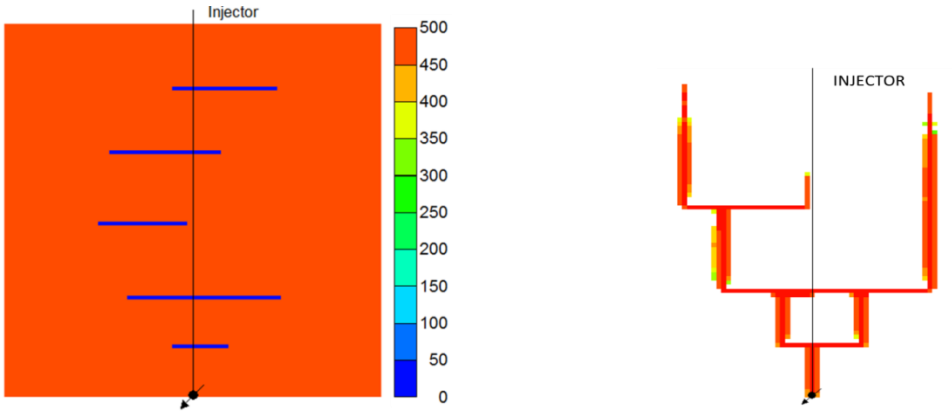


Figure 5.15. (a) Simulation model used for testing vertical migration of CO_2, (b) Saturation of CO_2 obtained by flow simulation showing the expected movement of the injected CO_2

The synthetic model used to test the validity of the tracer-based proxy for capturing gravity-driven flow is similar to the model used in Section 4.4.1, shown in Figure 5.15. This same model was analyzed using the new proxy formulation and the results compared to numerical simulation, as shown in Figure 5.16.

It can clearly be seen that the new proxy performs better than the previous version of the proxy presented in earlier Chapters. It is able to capture the migration of the CO_2 both vertically due to buoyancy and along the bottom of the baffles, mirroring the migration seen in the numerical simulator. It should further be noted that this response did not require any artificial weighting of terms or alteration of low permeabilities to 0 mD, as in the previous case (refer to section 4.4.1). Furthermore, the proxy is able to represent the timing of the migration process accurately.

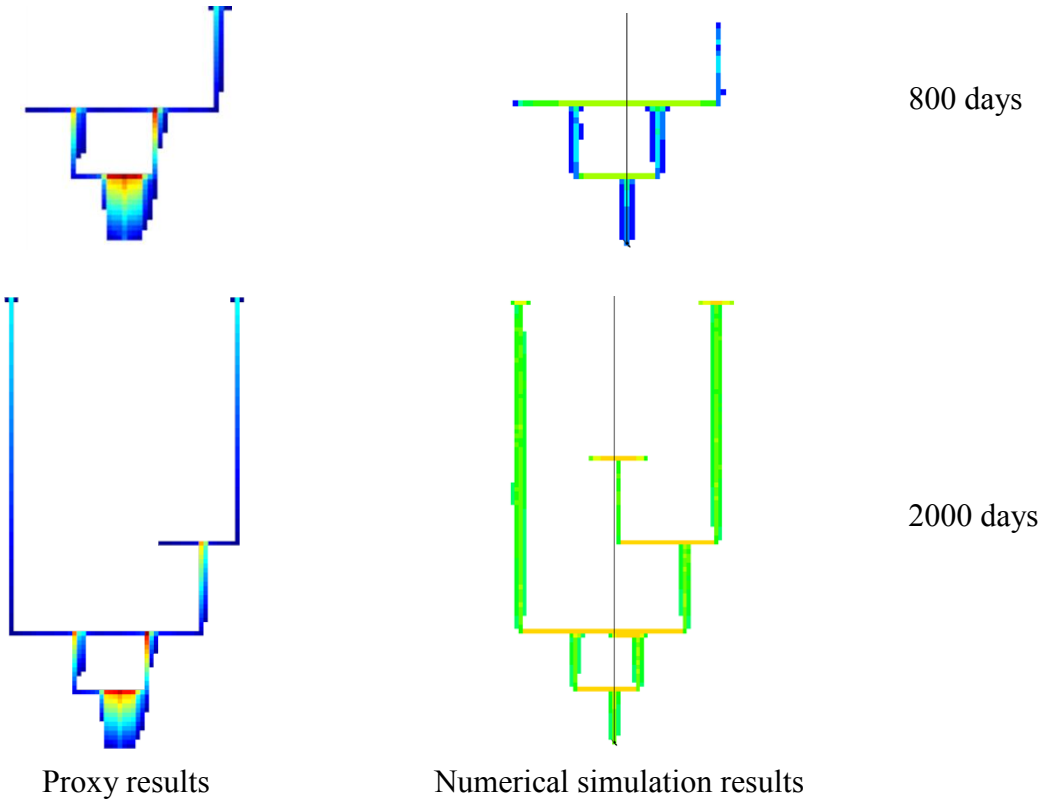


Figure 5.16. Comparison of tracer-based proxy to flow simulation, showing the ability of the proxy to capture vertical buoyancy-dominated CO_2 migration.

Given that the primary motivation behind development of the proxy was the rapid evaluation of reservoir connectivity, it is instructive to look at the computation time of the proxy. The computation time is a little over 2 seconds, which is comparable to the 146 seconds of computation time of the simulator. At this stage, it is important to test the performance of the proxy on a full field model that had a combination of all the effects studied in isolation in the preceding sections. Hence, we used the Sleipner model to test our new proxy formulation.

5.3.2. Validation of the new proxy formulation using the Sleipner model

The Utsira formation in the Sleipner field has been described in details in chapter 4. In this section, we analyze the model for the Utsira formation and test it against numerical simulation results to validate the applicability of the new proxy for full field models with multiple fluid flow mechanisms. The model mimics the Utsira reservoir, and is composed of 100 x 100 x 50 grid blocks. It uses the same structure as the top of the Utsira formation; however, the subsequent 49 layers of the model were generated using a uniform thickness for each layer. Similar to the stochastic shale layers in the original Utsira case, the synthetic model contained shale baffles in four layers with stochastically distributed sand 'holes' representing the erosional surfaces (Figure 5.17). The CO_2 was injected into the bottom-most layer and allowed to migrate upwards under the influence

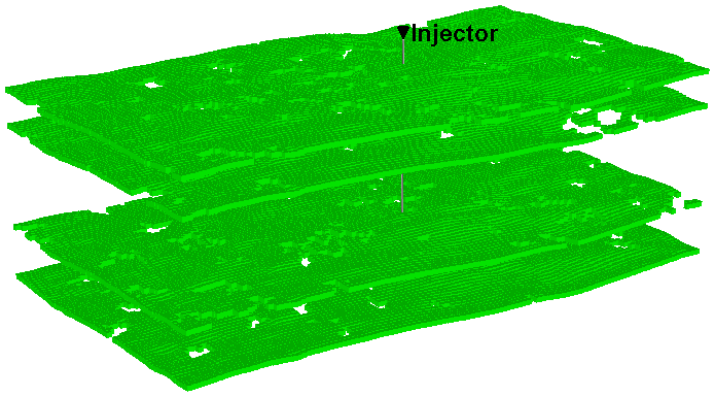


Figure 5.17. Model used for testing an Utsira-like synthetic case. Shown here are 4 shale layers (permeability 0.1 mD) containing stochastic sand 'holes'. The rest of the model (total 50 layers) has a permeability of 2 D.

of gravity. The upwards movement would be occasionally interrupted by the shale baffles, whereby the CO_2 would follow the structure of the layer and spread laterally until it reaches a sand 'hole', at which point the upwards migration would resume. This behavior has been observed in Figure 4.21. The same model was analyzed using the new

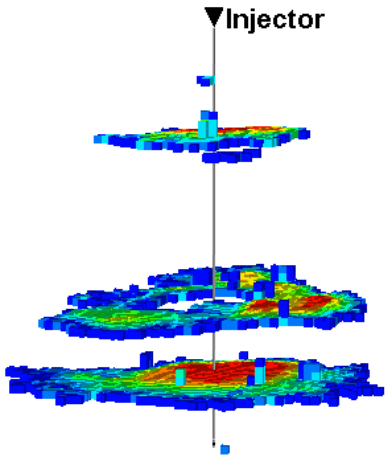
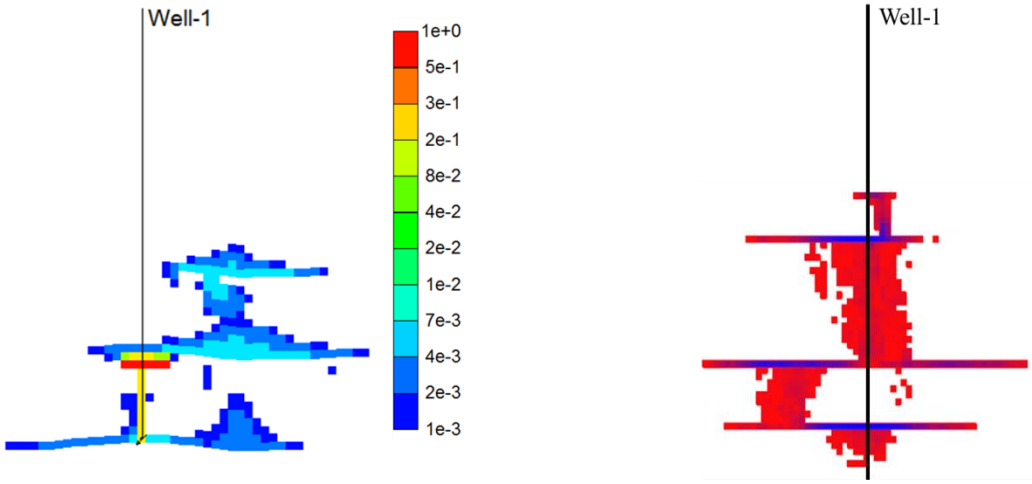


Figure 5.18. Proxy result showing migration of CO_2 captured both vertically and areally under shale baffles

proxy (results in Figure 5.18), and the results from both cases are compared in Figure 5.19.



Numerical simulation New proxy

Figure 5.19. Comparison of numerical simulation with the new proxy, for case with strong influence of gravity.

The result from the proxy shows that the fluid movement in the vertical direction is being captured, as is the interruption to flow by the shale baffles and subsequent movement laterally. The comparison with the simulation clearly shows that the proxy is

able to capture the vertical migration of the CO_2 plume similar to the simulation results. It should be further noted that this response captured by the proxy was generated in less than 3 minutes of *CPU* time, as opposed to the simulation which required a runtime of 16.5 hours, running on 6 computer cores in parallel. This processing time is proportionally faster than the cases shown in Section 5.2.2; the increase in efficiency compared to the numerical simulation is expected in this case since the fraction of the reservoir volume sampled by the proxy is smaller in this case. The proxy thus clearly has a computational advantage over the simulator, especially when our objective is to use the proxy response to discriminate between models.

We next verify the applicability of the proxy to the full model selection process for the Sleipner model.

5.3.3. Model selection using tracer-based proxy for Sleipner

The model selection process was implemented on an initial set of 150 models. The base case model shown in the previous section was used as a representation of the 'real' field model. This model was simulated for a period of 3 years of injection, and the model selection process was conditioned on the bottom-hole pressure data at intervals of 100 days. All the models in the initial set contained shale baffles with sand holes in the same four layers as the 'real' field model. The only uncertainty was in the distribution of the sand holes in the shale layers. The depth of the shale layers are quite well defined using well logs and correlated across the Sleipner field. The initial set of models therefore considers the uncertainty in the spatial extent of the shale layers.

The sand holes were generated using sequential indicator simulation as short scale features. The generation of the sand-hole model was not only independent from one model to another, but even between the four shale layers in each model. This is justified by the fact that the sand holes are erosional surfaces laid down by different geologic occurrences and bear little correlation to each other. A sample of these initial models is shown in Figure 5.20. These initial models were analyzed using the new particle-tracking proxy. The final model set and its simulated response was studied to test the entire model selection process and by extension, the validity of the proxy.

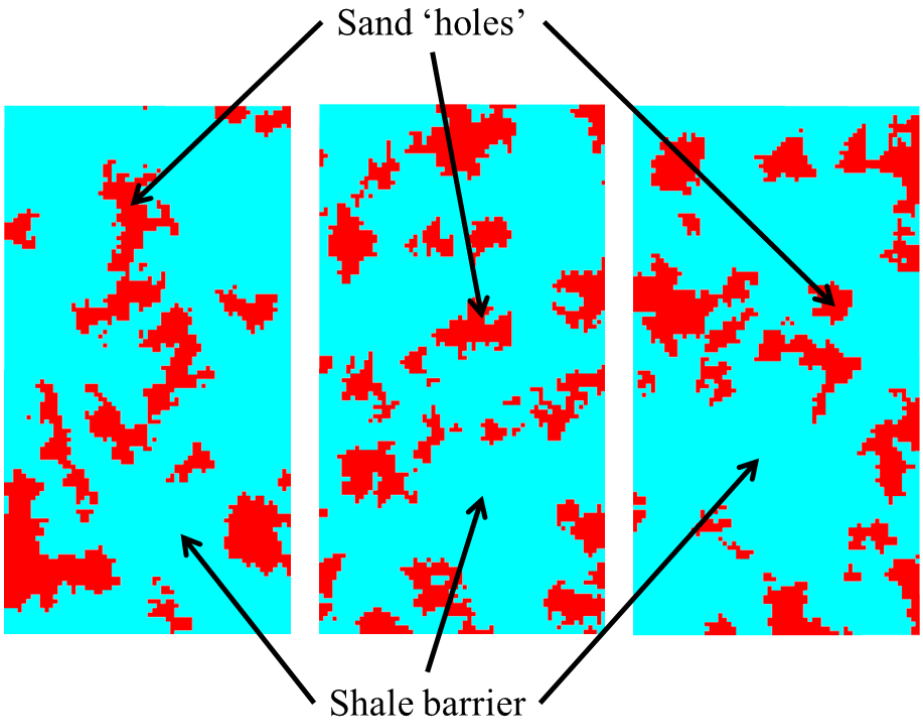
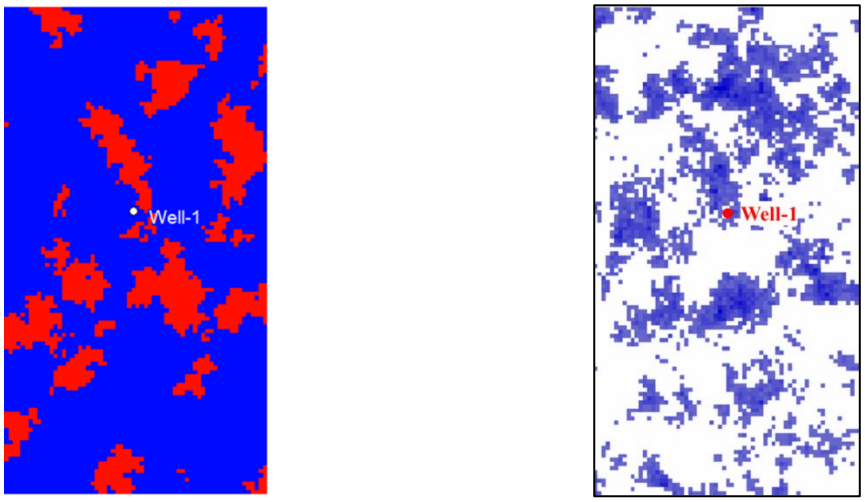


Figure 5.20. Sample of models in the initial model set for Utsira. The sand is shown in red and the shale in cyan.



Real model, base case Average model from best-fit models

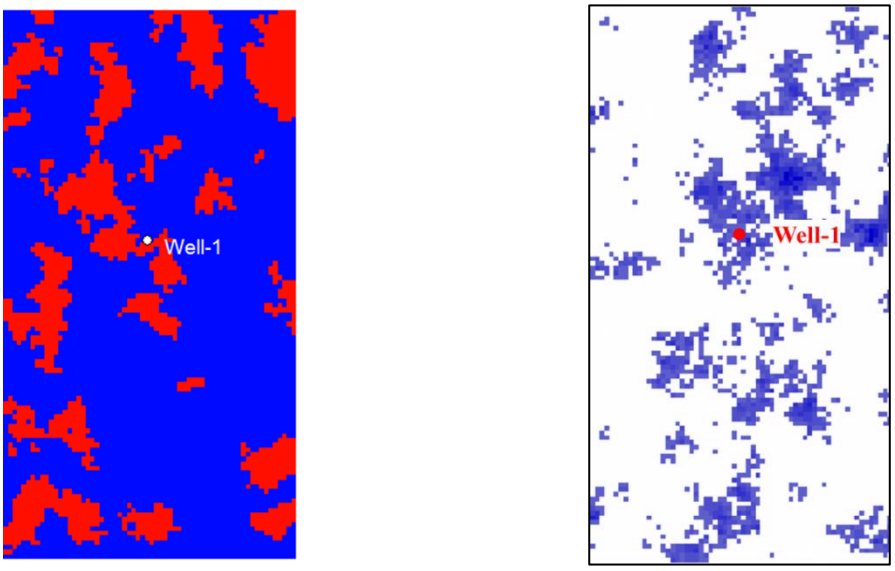
Figure 5.21. Comparison of real sand holes distribution and sand holes in average of best-fit models, at bottom-most shale layer.

In order to understand the major features highlighted by the final model set, we computed an average permeability model for the best-fit cluster of models and compared it to the original base model. Since the only uncertainty was in the distribution of the sand holes within the shale layers, our comparison was limited to these layers. Further, given that the time of injection was limited to 3 years, by which time the CO_2 had only just reached the second shale layer, we limited our analysis to the bottom two shale layers.

When we compare the distribution of sand holes in the bottom-most shale layer for the average of the best-fit models, we notice that it is able to capture the distribution of shales just above the well location (Figure 5.21). This is a reasonable outcome of the process, since the migration of the fluid and its effect on bottom-hole pressure is largely limited to the zone right above the perforation. The comparison shows that the average model reflects the presence of a sand hole located just above the perforation, oriented in

the SE-NW direction. It also captures the presence of sand holes to the south and west of the injection location.

When we repeat this comparison for the next higher shale layer, we notice that the ability of the model selection process to accurately represent the location of sand holes has decreased (Figure 5.22). The best-fit models captured the presence and orientation of sand holes around the injection location, but create a number of spurious features. This is once again expected, since the injected CO_2 had not reached the second shale layer within the time period specified, and so the ability of the injection well pressure to be influenced by the second shale layer is rather limited.



Real model, base case

Average model from best-fit models

Figure 5.22. Comparison of real sand holes distribution and sand holes in average of best-fit models, for the second shale layer from the bottom.

5.4. CONCLUSIONS

In this chapter, we detailed the development of a new particle-tracking proxy that mimics the flow of non-reactive tracers moving with the reservoir fluid. This proxy moves thousands of particles within user-defined time intervals driven by potential differences between neighboring grid blocks. It then calculates probability maps of how likely each grid block is of being contacted by the migrating fluid within the given interval of time. At the end of each time interval, the pressure and saturation fields are updated and the process is repeated. The proxy is quite effective for representing viscosity driven flow, gravity, fluctuations in injection rates, reservoir structure and two-phase flow.

We tested the proxy using different models to show its validity for different scenarios. The proxy response for synthetic models was compared to full-physics numerical simulations, and was shown to be effective in capturing both areal migration and vertical gravity-driven migration in reservoirs. It was also tested on real field models (In Salah and Sleipner) where there is a combination of areal and vertical migration taking place, and was shown to yield responses close to numerical simulations. Additionally, the proxy run times were almost 40 times less than for comparable numerical simulations, making it an ideal computational tool for quickly discriminating differences between various reservoir models.

As a final test of the effectiveness of the proxy, we demonstrated the ability of the proxy to be used as a part of the model selection process in order to enable the evaluation of model connectivity. The model selection process was run in its entirety using the tracer-based proxy for Sleipner, and was shown to be efficient for accurately modeling the location of sand holes in the shale layers in that field. It was also tested on a set of

initial models for the In Salah field, and is demonstrated to be effective for representing the spatial location of high permeability pathway between wells quite effectively.

Chapter 6 : Development of a software module for Model Selection

In the previous chapters, we have outlined a model selection process and detailed the development and validation of two particle-tracking proxies. The various parts of the model selection algorithm have, until now, been implemented as separate pieces of code. This requires a lot of work on the part of the user to move the results from one step of the workflow to the next step. It is therefore necessary to integrate all the parts of the algorithm into integrated software, which can be used by an end-user without explicit intervention at every step. For this purpose, we decided to implement the particle-tracking workflow as a plugin within the existing geostatistical software, SGeMS. In this chapter, we detail the development of this plugin, and outline steps required to use it.

6.1. SGeMS: Stanford Geostatistical Modeling Software

The Stanford Geostatistical Modeling Software (SGeMS) is an open source software module for geostatistical applications, developed at Stanford University. It allows the use of various established estimation algorithms (kriging, co-kriging, indicator kriging) and simulation algorithms (sequential Gaussian / indicator simulation etc.), as well as various utilities to create, manipulate and post-process reservoir models. It also has an accompanying visualization module that allows users to look at models in two- or three-dimensions, and extract information along horizontal and vertical slices. Since the code is open-source, it is free to manipulate existing modules or even create new modules in order to implement additional workflows not available in the current distribution. The ability to implement standard modules to create geologic consistent models, the existence of the visualization module, and the flexibility to write additional modules within the

existing code, made this the ideal software to implement our workflow as a plugin within SGeMS. In this section, we will briefly describe the user-interface for the software.

The main window for SGeMS is shown in Figure 6.1. It consists of three main parts: the algorithm panel, the data panel and the visualization panel. The algorithms

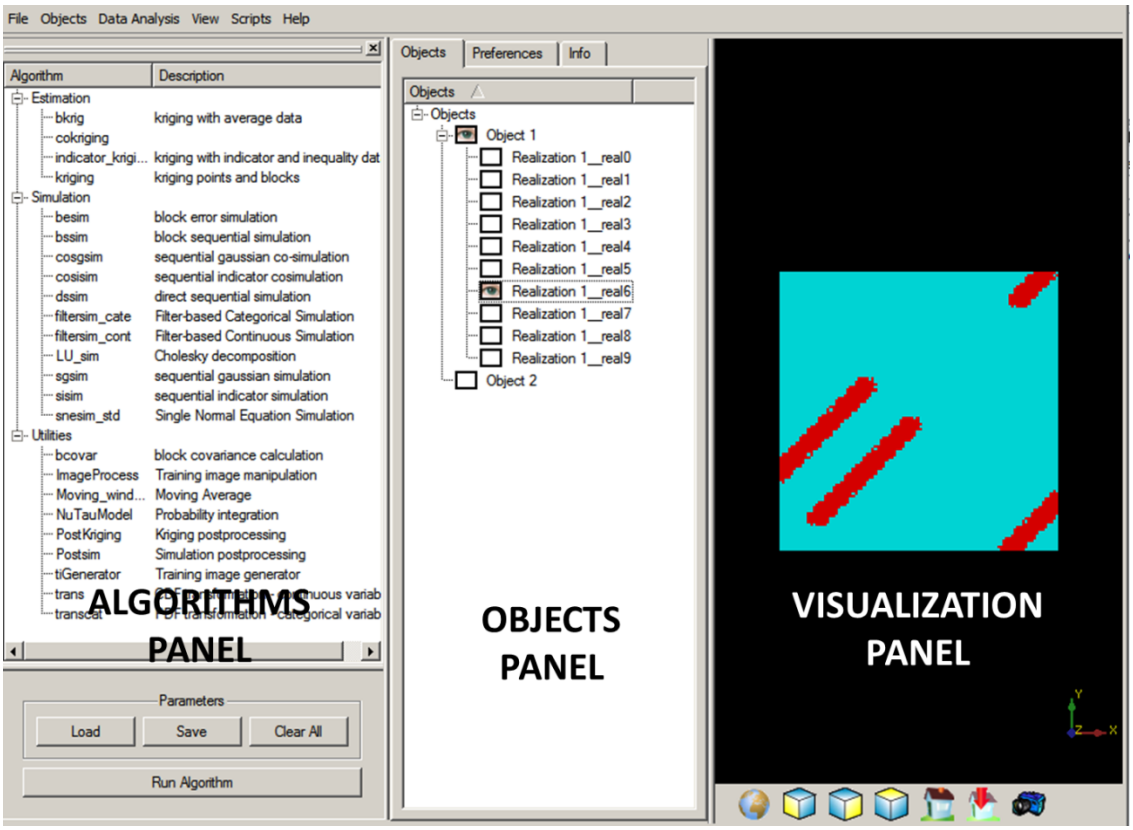


Figure 6.1. SGeMS user interface, showing the three main panels: algorithms, objects and visualization

panel contains all the geostatistical software under two main headings: estimation and simulation. There is a third heading called Utilities, which contains various smaller programs for post-processing and manipulation of models within the program. The 'objects' panel shows a list of all data that have either been loaded into the program or

output by the implementation of algorithms within the program. Two types of data can be read by the program: Cartesian grids which contain data for every grid block in a model, or point sets which contain data at specific coordinates. Multiple realizations of a particular variable can exist within a single object as shown in Figure 6.1 where Object 1 has 10 realizations. The visualization panel displays any selected realization within an object, and also allows actions like panning, zooming etc. of the image.

The model selection program was implemented as a separate heading in the 'Algorithms' panel.

6.2. MODEL SELECTION PLUGIN: AN OVERVIEW

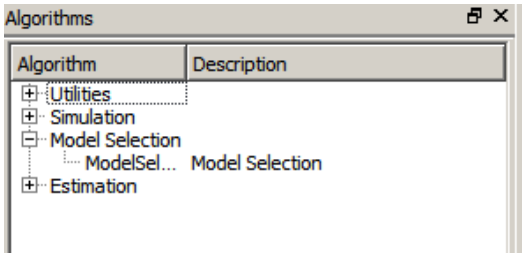


Figure 6.2. Algorithms panel, showing the model selection algorithm tab

The model selection algorithm was implemented as a separate heading in the algorithms panel (Figure 6.2). Once this option is selected, it brings up the input window for the model selection process, where the user needs to provide all the parameters and data used to the process. The main tab in the model selection input panel contains drop-down menus to select the particular permeability and porosity objects to be used, together with details about the connectivity proxy and reservoir flow simulator to be used. The various input options are listed below, and referenced in Figure 6.3:

[1]. Permeability object: contains all the models in the initial model set

[2]. Porosity object: similar to permeability object

[3]. Proxy selection: drop-down menu to select the particular proxy to be used

[4]. Working folder: this is where all the results are stored

[5]. History file: this is the dynamic conditioning data used by the process

[6]. Simulator location: the physical address to the executable of the reservoir flow simulator to be used. Our current implementation uses CMG-GEM simulator.

[7]. Simulator file name: name of the simulation data deck. It is assumed to reside in the working folder.

[8]. RESULTS (CMG) location: location of the RESULTS GRAPH© program in CMG that reads the output file for the simulation in order to compare to the conditioning data.

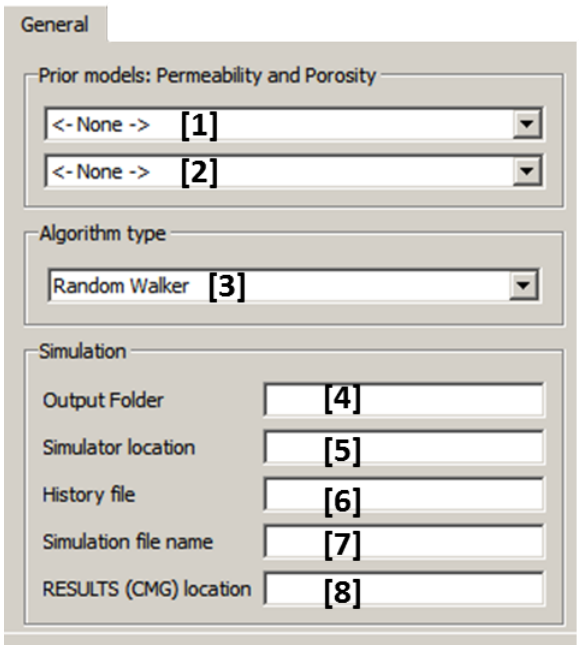


Figure 6.3. Main input panel for the Model Selection algorithm

Once the random walker proxy is selected, the user has to enter information pertaining to the proxy in the 'Random Walker' tab. This window is shown below (Figure 6.4), and is described in the following list:

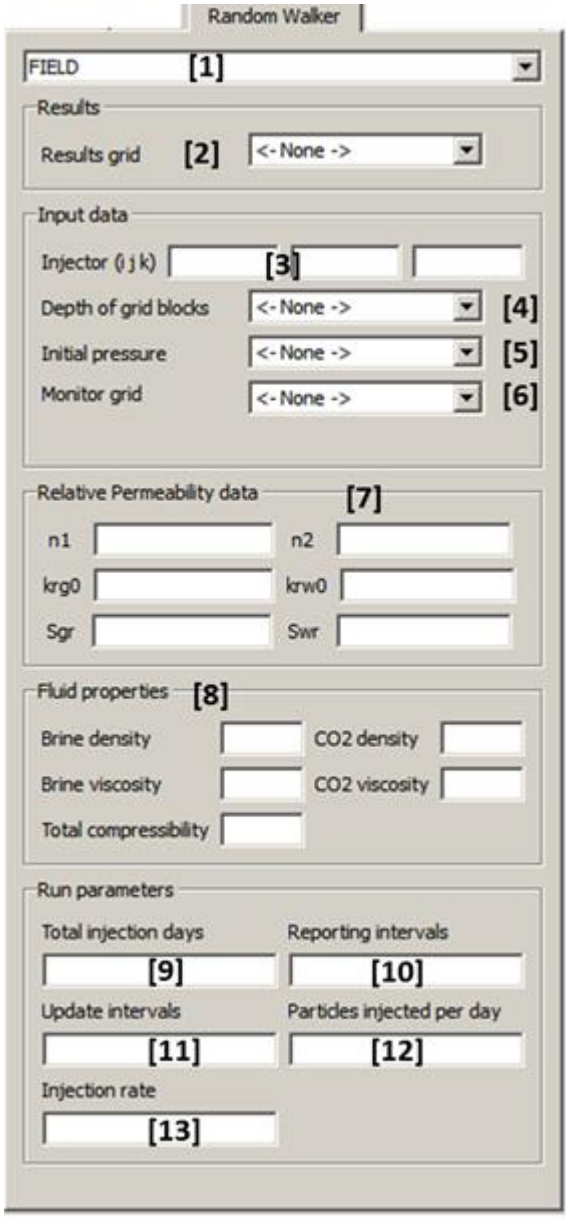


Figure 6.4. Random Walker tab, where the user needs to input data needed for the proxy

[1]. Unit system to be used for data: the code allows for using either field units or SI units.

[2]. Results grid: the object where the final best-fit models are saved

[3]. Injector location: the x, y and z coordinates of the injection location

[4]. Depth of grid blocks: an object containing a single realization of depths to each grid block for the uppermost layer of the models. It is assumed that the structure of the grid is same for all initial models

[5]. Initial pressure: object containing the initial static pressure for the entire grid. This can be generated from the initialization step of a numerical simulator. This is also assumed to be the same across all models, so the object contains a single realization.

[6]. Monitor grid: a point set containing the locations where the proxy measurements are recorded for further analysis and grouping

[7]. Relative permeability data: Corey-type relative permeability model is implemented currently. Thus, this panel needs values for the Corey exponents, the end-point relative permeabilities and the residual saturations.

[8]. Fluid properties: values for fluid viscosities, densities and total compressibility. Care should be taken to keep the units consistent with the unit system defined at the top.

[9] – [13]. Here, the user defines the step sizing and particle count parameters for the run. The proxy will be run for 'Total injection days' [9], and pressure and saturation updates will occur after every 'update interval' [11], while the proxy measurements will be noted every 'reporting interval' [10]. The injection rate [13] is specified either in m^3/s or ft^3/day.

Once all the data has been specified, the user runs the program using the 'Run Algorithm' button at the bottom of the algorithms panel. The results of the proxy at specified time intervals are stored in the object called 'RW_results', and the best-fit models are saved within the user-specified 'Results' object (item [2] in Figure 6.4). The user can then perform further post-processing of the best-fit models using existing algorithms within SGeMS, or export the models to text files.

6.3. DETAILS OF PLUGIN DEVELOPMENT

SGeMS is implemented in C++ as an object-oriented code, with all geostatistical, data analysis and visualization operations encoded as classes. In order to create a plugin for SGeMS, we need two elements: a shared library (.dll file) that contains the algorithm and a graphical interface through which parameters are read into the program. The graphical interface is created as an *.ui file, using either a text editor or Qt Designer. The details of creating the interface are beyond the scope of this work. In this section, we will describe the creation of the shared library and its basic components.

6.3.1. Creating the shared library

In order to create the SGeMS shared library, we first need to define a class derived from the Geostat_algo class, and within it create three virtual functions *initialize, execute* and *name*. This is shown for the model selection algorithm in Figure 6.5.

```
81  class GEOSTAT_DECL ModelSelection : public Geostat_algo {
82  public:
83      ModelSelection();
84      ~ModelSelection();
85
86      virtual bool initialize( const Parameters_handler* parameters,
87                               Error_messages_handler* errors );
88      virtual int execute( GsTL_project* proj=0 );
89      virtual std::string name() const { return "ModelSelection"; }
90
```

Figure 6.5. Declaration of the ModelSelection class, derived from GeoStat_algo, and the virtual functions

The *initialize* function initializes the various input variables needed for the particular algorithm by reading it in from the user interface. The *execute* function contains the code for execution of the algorithm, and the *name* function returns the name of the algorithm. The final step is to define a static function called *create_new_interface* within the new class. This creates a new instance of the plugin object. In our case, this function was defined as shown below:

```
Named_interface* ModelSelection::create_new_interface( std::string& ) {

  return new ModelSelection;

}
```

This code is then compiled to create the library file.

6.3.2. The *execute* function for the Model Selection algorithm

The *execute* function contains the details of implementation of the model selection algorithm. As such, it contains the following basic building blocks: the data entry module, the particle-tracking proxy, the module for principal components analysis and model clustering, and a module to run simulations on the best model of each cluster,

compare the simulated result with field data and calculate posterior probabilities. Figure 6.6 shows how the different modules fit into the model selection workflow.

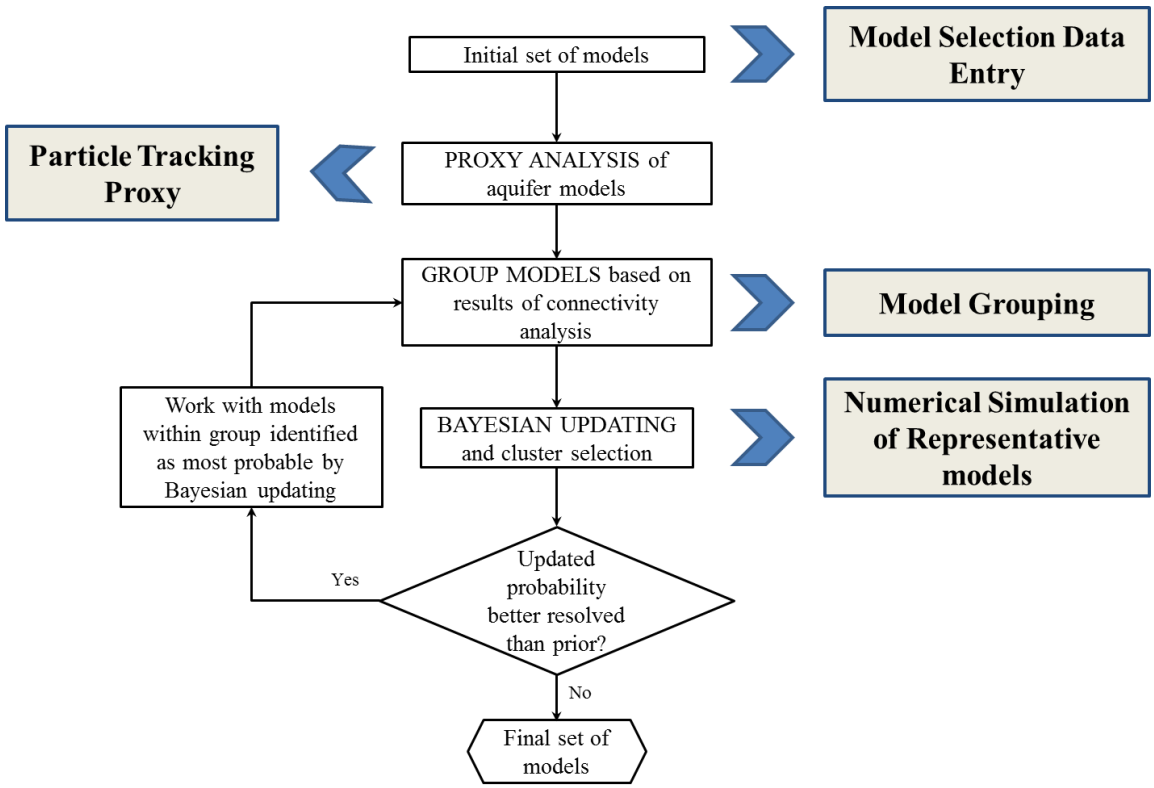


Figure 6.6. Modules (grey boxes) required for the implementation of the model selection algorithm within the SGeMS framework

These modules are all implemented as independent functions in a separate file and included into the SGeMS code as an external library. The details of the *execute* function are provided in Appendix B.

6.4. CONCLUSIONS

In this chapter, we have outlined the development of the model selection algorithm as a plugin to the geostatistical software suite, SGeMS. The plugin is implemented using a derived class called 'ModelSelection', which includes three main functions: *initialize, execute* and *name*. The various modules needed for the model selection process are implemented as functions and then included in the main code as external libraries. The plugin seeks to harness the existing capabilities of the software in order to generate the initial set of models. Alternatively, a set of models generated externally by the user can also be imported and then the plugin can be used to implement the model selection process on these models. This approach also enables the user to visualize data very efficiently using internal SGeMS routines, and also post-processes the final best-fit models to quality check and to generate posterior uncertainty maps and statistics. This chapter, in conjunction with Appendix A, should provide the information necessary to make future changes to this code.

Chapter 7 : Some Applications of the Model Selection Algorithm

In this chapter, we will explore some possible applications of the model selection process, applied to problems encountered during geologic carbon sequestration. In previous chapters, we have already demonstrated the use of the model selection workflow incorporating the particle-tracking proxy for the delineation of best-fit aquifer models conditioned to observed field response. The cases demonstrated in this chapter incorporate further modifications to the model selection algorithm. We will look at ways to generalize the location of proxy measurement locations to better inform the model selection process. We will demonstrate the use of the model selection process to investigate the presence of leaks in the aquifer, to assess the impact of mineralization on the injection well response. We also investigate issues such as the impact of location of injection and montoring wells and their role in determining the limits of applicability of the model selection process.

7.1 Optimizing the configuration of proxy measurement sites

In previous examples of the model selection process, we have predefined proxy measurement sites at the corners of a square around the injector (as in the case of In Salah in Chapter 4) or at fixed locations in regions of interest (e.g. just below the shale layers at Utsira in chapter 6). Since the statistics of random walkers recorded at proxy measurement sites are a representation of the connectivity characteristics of the models, it might be hypothesized that the location of these sites is critical to the entire model selection process. For example, if we situated our measurement sites in In Salah far from the region of interest, as shown in Figure 7.1, the connectivity in the region between KB-502 and the abandoned well would have very little bearing on proxy response at those

sites. Under these circumstances, the models represented by these proxy measurements would show little or no grouping to enable distinction on the basis of connectivity. The user might then be misled to think that the data available to him/her is not adequate for the model selection process, where in fact the problem lies with the choice of proxy measurement sites. As such, it might be better to have a carefully considered algorithm for choosing these sites rather than leaving it completely to the discretion of the user.

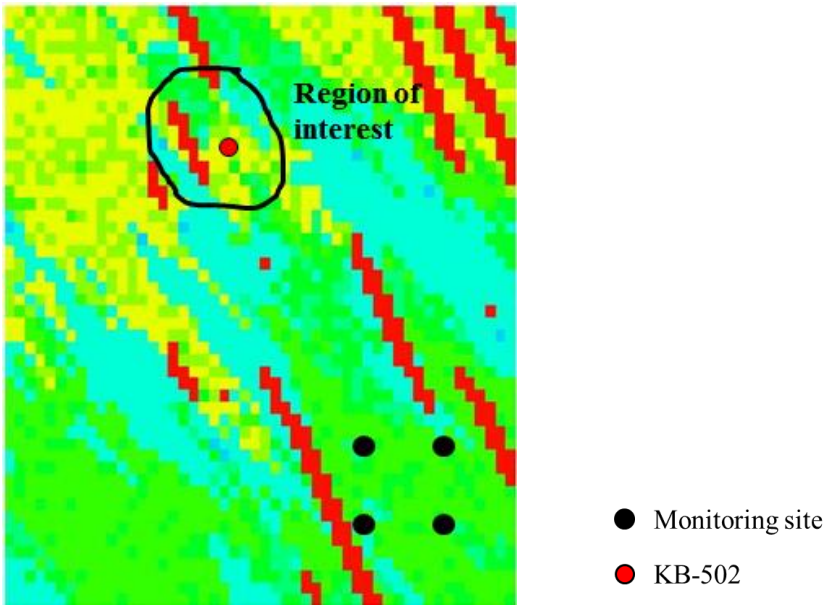


Figure 7.1. Location of proxy measurement sites far from the region of interest will not inform the model selection process.

In the next section, we demonstrate a method to determine the optimum monitoring locations.

7.1.1. PCA to find optimum measurement sites

The connectivity analysis yields various statistics (e.g. particle count, pressure analog) at every grid node within a model. These are akin to the grid pressure, saturation

and specified concentration values obtained using a full-physics simulation. Locations along a particular similar feature are expected to show similar particle statistics as compared to locations outside the feature or locations on other connected features within the reservoir. Because our objective is to detect differences between reservoir models, it is preferable to locate the monitoring locations sufficiently far apart so that they pick up differences over a larger extent of the reservoir. For this purpose, it is necessary that the monitoring locations be located such that they are spaced across several features (and not all clustered within a single feature). Thus, a strategy for locating the measurement location could be to seek locations that show maximum variability in response across all the available models, indicating limits to the size and shape of connected features. For this purpose, a single response (e.g. particle count at a particular time or average particle count over the entire run) is measured at all grid locations across all models. The covariance between the proxy responses at any two grid locations was calculated across all models, and a principal component analysis was then performed on the covariance matrix. The first principal component direction was identified, which consisted of a weighted linear combination of the response at all grid locations (the eigenvector values corresponding to the first eigenvalue are the weights or loadings). The locations are sorted according to their weights, and the locations within a fixed cutoff of the maximum weight are retained. The locations, which have significant weight in the first principal axis, represent the locations that exhibit maximum variability across all models with respect to measured responses; they would be optimal for capturing the variability off spatial connectivity observed within the suite of models. This is shown in Figure 7.2.

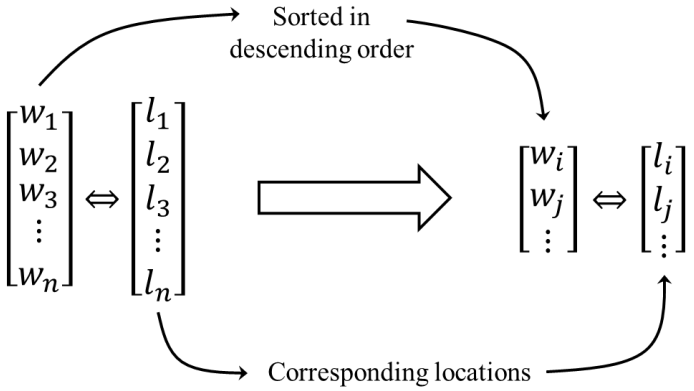
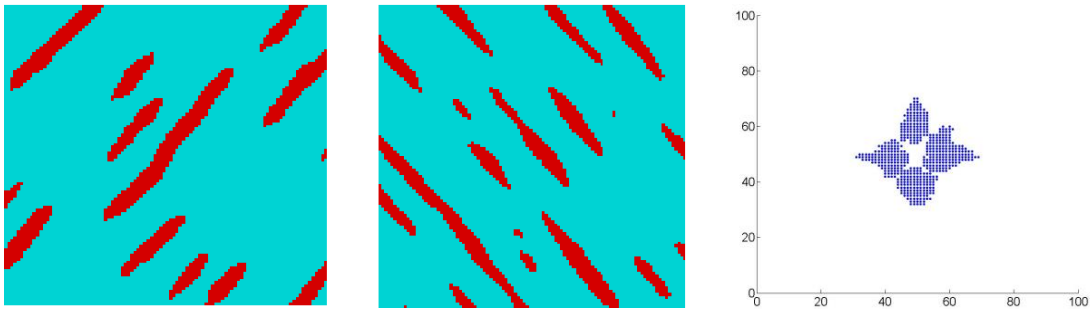


Figure 7.2. Demonstration of process for finding optimum locations. Only the top N% locations on the right are retained. In the demonstration case below, this percentage is 10%

As a demonstration of the process, this method of finding measurement locations was performed on a set of 100 models with features in orientations ranging from 0^O to 90^O azimuth. The initial model set, and the final measurement locations inferred for this set of models are shown in Figure 7.3.



(a) (b)

Figure 7.3. (a) Two different kinds of prior models. 50 realizations of each type of models, were used for the initial model set, (b) Regions identified by performing Principal component Analysis using statistics of particles recorded within each grid block. These locations for the proxy monitoring location will help identify regions exhibiting maximum variability across all 100 models.

Once the locations with maximum variability have been determined, the next step was to find a method for choosing sites amongst these locations where the proxy-measurement sites could be located, since there is a lot of redundancy between information provided by points that are located close to each other. Two methods have been devised for this purpose:

1. *Choosing arbitrary points within the location clusters:* The first method chooses certain points within the clusters of locations given by Figure 7.3 (b) as the proxy monitoring sites. The plausible proxy monitoring locations can be seen as being divisible into 4 different clusters out of which one location was chosen at the centroid of the clusters and two other locations were selected within each cluster that are maximally separated from each other and the centroid. These locations approximately reflected the directions of major features in the model set. Sites chosen in this manner have been shown in Figure 7.4. One problem with this method, as seen in the two points circled in red, is that there might be redundancy due to the chosen points being too close to each other.

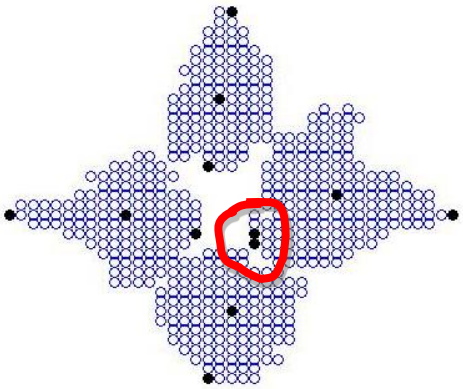


Figure 7.4. Arbitrary choice of points as proxy monitoring locations. Three points (shown as black circles) were chosen within each cluster.

2. *Choosing locations using repeated PCA:* This second approach extends the method of finding locations using principal components analysis (PCA) to multiple iterations. For subsequent iterations, only grid locations selected from the previous step were used. The process was terminated when the number of locations does not change with further

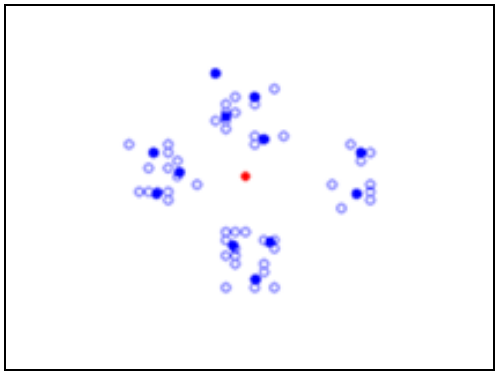


Figure 7.5. Locations identified for proxy monitoring after repeated PCA.

iterations. The grid locations for the same set of 100 models after the final PCA iteration are shown in Figure 7.5.

The points shown as blue filled circles were *chosen* as proxy-measurement sites following this process. These points are chosen to reduce redundancy between points (because the principal components are orthogonal to each other) as can be seen by comparing the location in Figure 7.5 with those in Figure 7.4 that contained some points very close to each other.

To demonstrate the applicability of this process of identifying proxy monitoring locations, we revisited the In Salah case. In our original approach (Chapters 4 and 5), we had placed 4 measurement sites at the corners of a square centered on the injector KB-502, since we knew that we were looking for a feature of interest around that injector.

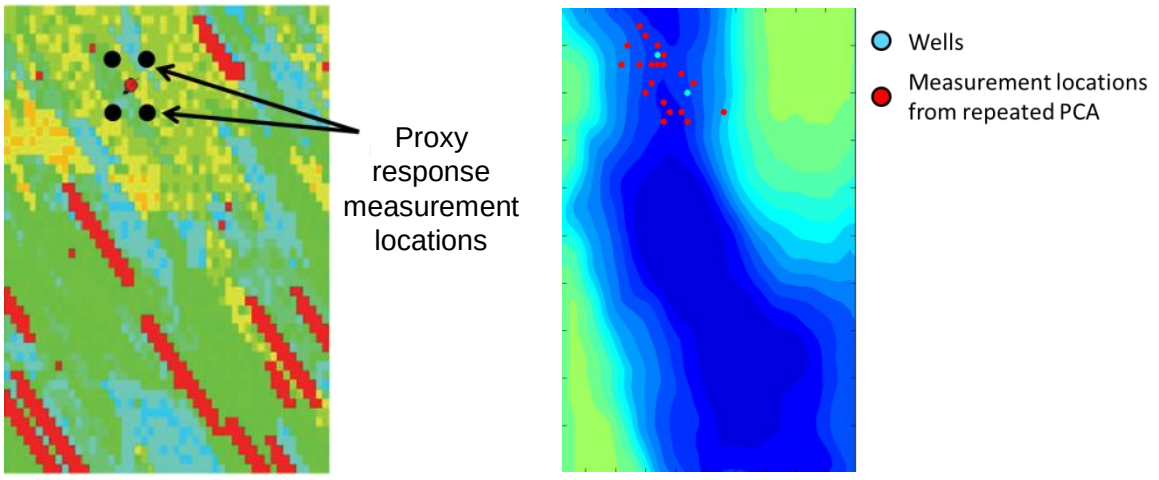


Figure 7.6. (a) Proxy response measurement locations in previous work (Chapter 4 and Chapter 5). (b) Layout of proxy measurement locations using the new PCA-based method, displayed on a structure map.

Here, we recalculated the location of the measurement sites using the repeated PCA approach shown above. These new locations are shown in Figure 7.6.

Using these new measurement locations, the model selection algorithm was applied and carried forward for two iterations, with seven clusters of models chosen after each iteration. The results are shown in Figure 7.7 (a), (b). Only data from the first 700 days are used in the model selection. When these results are compared to the results from our previous paper (Figure 7.7 (c), (d)) – when the proxy measurement locations were located at the corners of a square template around KB-502 – we can clearly see that the new method is comparable to the previous method; the advantage of the method lies in its applicability to more general cases, when we do not have any prior information about where dominant features are located. Figure 7.8 shows the cluster average of the best-matched models obtained using both approaches for locating proxy-measurement

locations. A high permeability feature between the wells KB-502 and KB-5 can be seen on the average across all models, from both methods.

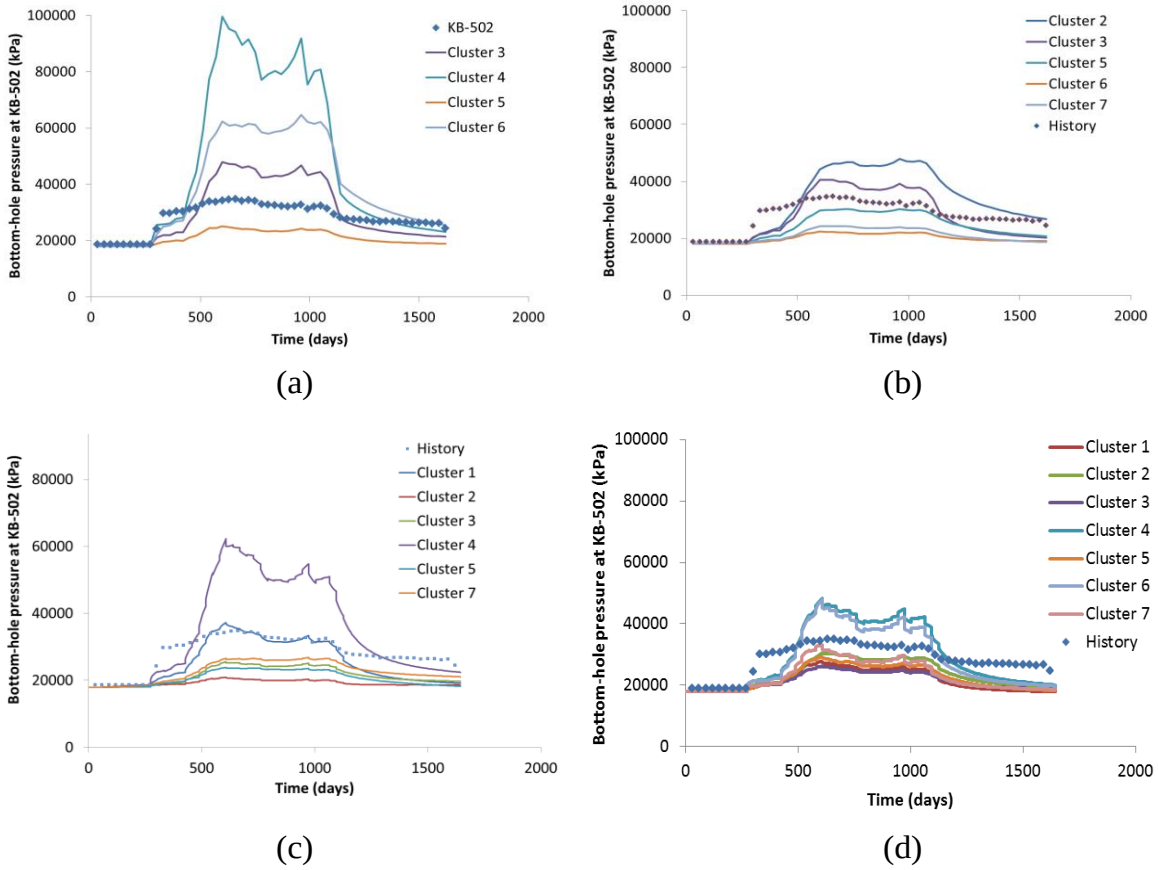
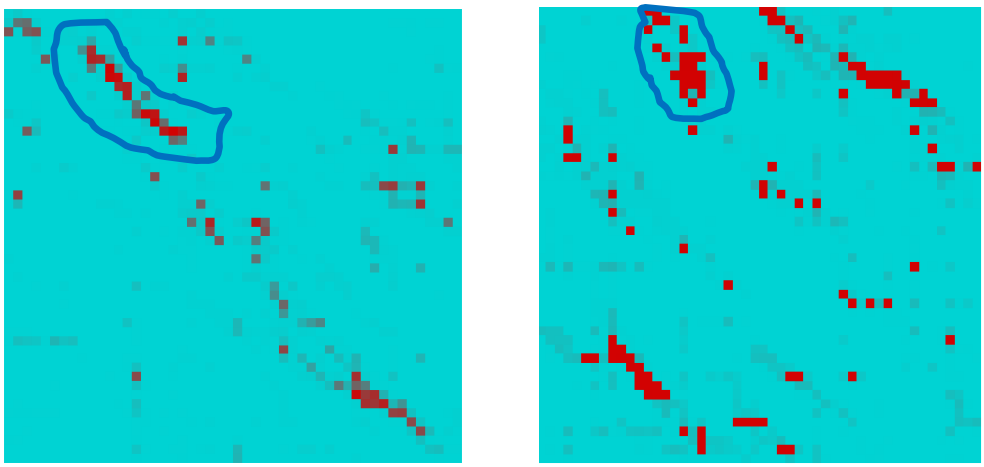


Figure 7.7. (a), (b) Cluster response at the end of the first two iterations of the model selection process, respectively, with proxy monitoring locations chosen by the repeated PCA method. (c), (d) Cluster response after the first two iterations of the model selection process, with proxy monitoring locations arranged in the form of a square around KB-502. The results are comparable, highlighting the ability of the current method to find proxy measurement locations regardless of prior knowledge about the heterogeneous characteristics of the target reservoir.



(a) (b)

Figure 7.8 Average permeability of all models in best-match cluster from (a) measurement locations in a square, and (b) Measurement locations from PCA. Both show a high-permeability feature (circled) connecting KB-502 and KB-5.

7.2 MINERALIZATION DURING CO_2 MIGRATION IN THE AQUIFER

During the operation of a sequestration project, there are 4 dominant mechanisms that contribute to the capture of the injected CO_2. These mechanisms are: structural trapping of supercritical or gaseous CO_2, dissolution of CO_2 in brine, residual phase trapping due to relative permeability hysteresis, and mineralization. In this work, we will simulate the mineralization process as a bulk permeability modifier and explore the feasibility of using model selection process to detect location of permeability alterations due to mineralization during a sequestration project. The basic assumption in the cases described below is that the mineral composition of the rock is invariant, and the mineralization process is operational wherever the rock comes in contact with the injected CO_2.

7.2.1. Background

Mineralization is the process of reaction and precipitation when subsurface fluids interact with minerals in the rock. In the case of CO_2 sequestration in chemically active rocks, the CO_2 dissolved in brine forms a weak acid that interacts with the clay minerals in the rock leading to the formation of carbonates. The injected CO_2 is thus permanently stored in the form of mineral carbonates. Johnson et.al. (2004) identified four distinct mechanisms of mineralization that exist in saline aquifers: cementation of Dawsonite [$NaAlCO_3(OH)_2$] that occurs throughout the extent of the CO_2 plume, calcite-based carbonate precipitation along the lateral and upper margins of the plume, and mechanisms that take place within interbedded shales (like in Utsira) or the cap rock. When the injected CO_2 comes in contact with the formation brine, it forms a hydrated oxide, which then reacts with water to form a weak acid. This is shown by the reaction:

$$CO_2 + H_2O \Leftrightarrow CO_2(aq) \Leftrightarrow HCO_3^- + H^+$$

This reaction takes place within the entire volume of the plume. The weak carbonic acid reduces the pH and promotes the dissolution and precipitation of Dawsonite and calcite-group carbonates by the following reactions:

$$KALSi_3O_8 + Na^+ + HCO_3^- + H^+ \Leftrightarrow NaAlCO_3(OH)_2 + 3SiO_2 + K^+$$

$$M^{+2} + CO_2(aq) + H_2O \Leftrightarrow MCO_3 + 2H^+ \text{ where M is Fe, Mg and Ca}$$

Both these processes cause precipitation of minerals in pore bodies and throats and reduce porosity and permeability. Increasing pressure increases the dissolution of CO_2 and reduces the pH of the solution (Park et.al. 2003), which in turn enhances the dissolution of the minerals. The efficiency of this trapping mechanism is determined by the mineral composition of the formation rocks. Increased trapping occurs with increased concentration of carbonate forming elements like Fe, Mg, Ca, Na and Al (Johnson et.al. 2004).

As pointed out above, the combination of dissolution and precipitation causes local reduction in porosity and permeability. In this work, we modeled mineralization as a bulk permeability reduction factor effective only in regions where the CO_2 concentration is high enough to cause significant reactions.

7.2.2. Model setup for mineralization

In order to assess the impact of permeability alteration due to mineralization on the migration of the CO_2 plume, we implemented a model in CMG-GEM, shown in Figure 7.9.

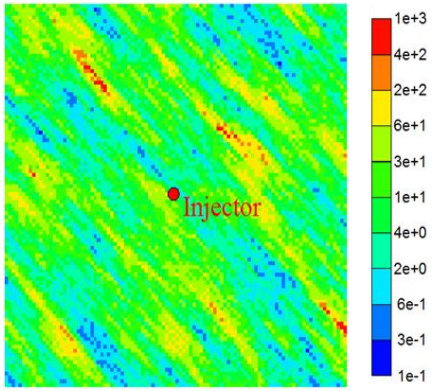


Figure 7.9. Base case model for demonstrating the effect of mineralization on CO_2 plume migration, showing the permeability distribution of the model.

The total injection period for this model, which was 20 years (2001 to 2021), was divided into 4 parts (2001 – 2006, 2006 – 2011, 2011 – 2016, and 2016 – 2021). At the end of each injection period, the saturation of CO_2 throughout the entire model was recorded, and the permeability was reduced by a factor of 2 for all grid blocks which had CO_2 above a cutoff saturation. For example, Figure 7.11(a) shows the saturation map of CO_2 at the end of the first 5 years (2001 – 2006) for the base model, and Figure 7.11 (b)

shows the difference between the permeability used for the first 5 years and the permeability used to run the simulation for the next 5 years. It shows the permeability change introduced into the model due to mineralization, assumed to occur uniformly over the entire CO_2 saturated zone.

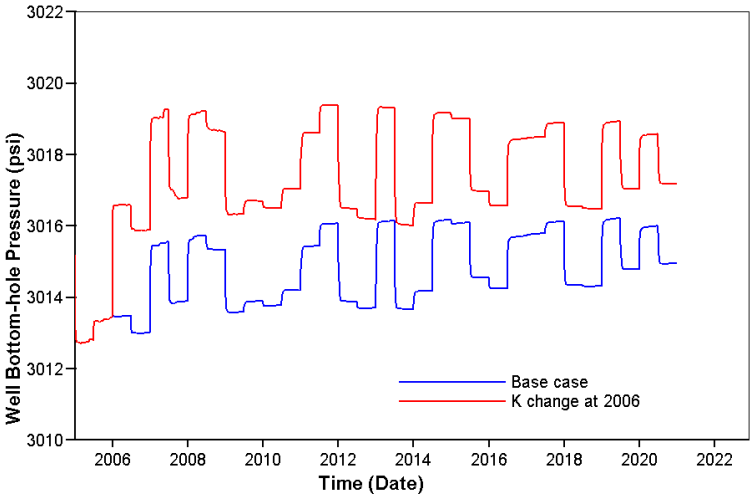
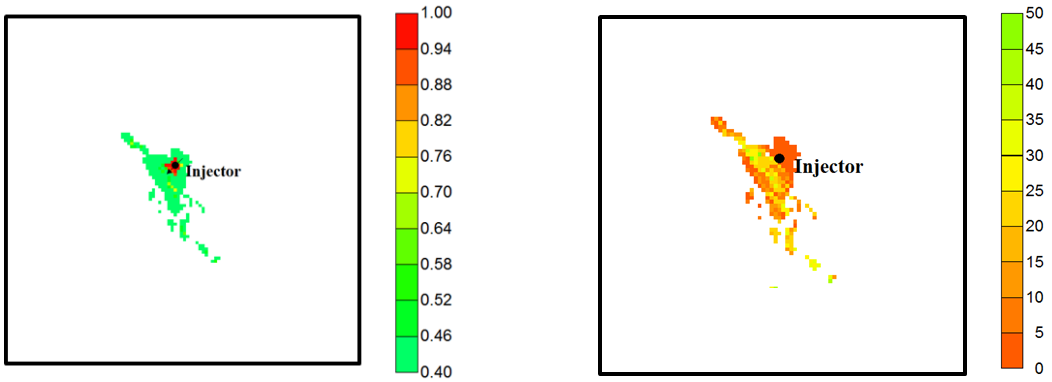


Figure 7.10. Plot of injection pressures for the mineralization case. The base case is run with no permeability changes (blue line). The red line shows the injection pressure when permeability change is made in 2006.



(a) (b)

Figure 7.11. (a) Map of CO_2 saturation after 5 years of injection, showing the regions above a cutoff of 40% saturation, (b) Map of difference between permeability of base model (Figure 7.9) and model used to simulate the next 5 years.

This process was repeated for each time interval, and the simulation for the next time interval was then restarted from the end time of the previous interval. As a demonstration, the injection pressure from the base case and a case with permeability change only in 2006 is shown in Figure 7.10. The plot shows that accounting for mineralization causes a drop in permeability, which reduces the injectivity and at constant injection rates, increases the injection pressure. This indicates that with multiple changes in permeability, we should expect injection pressure increase at the end of every interval. It would be better to implement a method whereby continuous permeability updates are made such that the change in injection pressure is not so drastic; however, for the purpose of demonstration, the current implementation with 4 permeability changes was assumed.

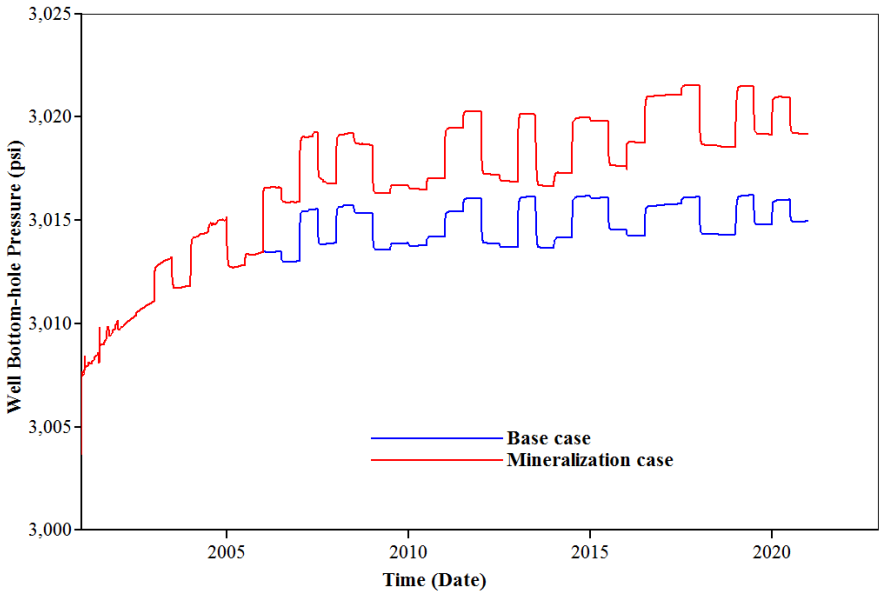
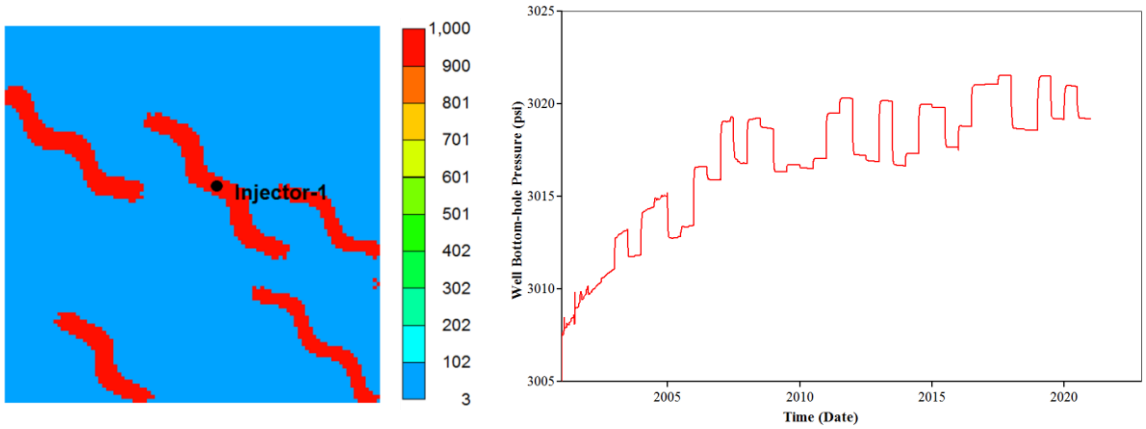


Figure 7.12. Bottom-hole pressure at the injection well, for the base case with no permeability alteration (blue line) and the mineralization case when permeability is altered in 2006, 2011 and 2016.

Using the method shown above, simulations were run for the entire time period of 20 years, with permeability alterations and simulation restarts at 2006, 2011 and 2016. The injection well was constrained by a constant injection rate and the injection pressures were noted. The results are shown in Figure 7.12. As previously stated, the injection pressure increases at every permeability reduction; the amount of pressure increase gradually decreases as the degree of injectivity reduction in previously altered zones reduces with each successive alteration.

7.2.3. Model selection with mineralization

The objective is to demonstrate how the model selection algorithm yields a different model set if mineralization is accounted for, as opposed to a case where there is no permeability alteration due to mineralization. For this purpose, we created a synthetic model containing some high permeability sinusoidal channels and a single injector completed inside one such channel. The injection period was divided into 3 intervals, similar to those in the test case above. The well was assumed to be rate constrained and a fluctuating injection rate schedule was used. The bottom-hole pressure of the injector was designated to be the dynamic data to be used for the model selection algorithm. The synthetic model and the injection pressure for the reference model are shown in Figure 7.13. This injection pressure profile takes into account the permeability reduction after each of the 3 time intervals.



(a) (b)

Figure 7.13. (a) The initial permeability map of the reference model, (b) Reference injection pressure profile that was used within the model selection process.

To demonstrate the applicability of the model selection process to this case, the algorithm was run on a suite of 100 models. The models all contain high permeability

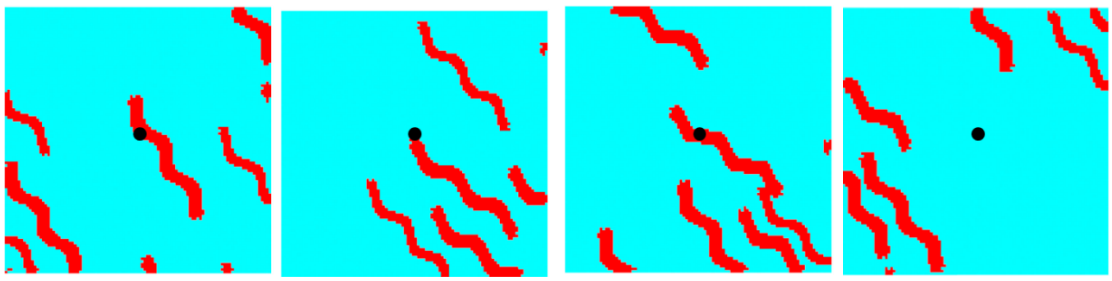


Figure 7.14. Some sample models from the initial model set. The red channels are 1000 mD while the cyan background is normally distributed around 10 mD. The black circle represents the position of the injector.

sinusoidal channels in a low permeability matrix; however, the positions of the channels are not constrained by any data and hence exhibit large variability from one model to the next. Some of these starting models are shown in Figure 7.14. Note that some models have injection location close to channel boundaries, while some are located far away from channels in the low permeability zone. The model set was run for two separate

cases: one in which permeability is altered at the end of each injection interval similar to the test case discussed before (section 7.3.2), and another in which there was no permeability alteration.

7.2.4. Results

There is considerable difference in the spread of injected CO_2 with and without permeability alteration, as can be seen in Figure 7.15. The reduction in permeability blocks off high-permeability regions early in the injection process, and causes the plume to seek out alternative migration pathways.

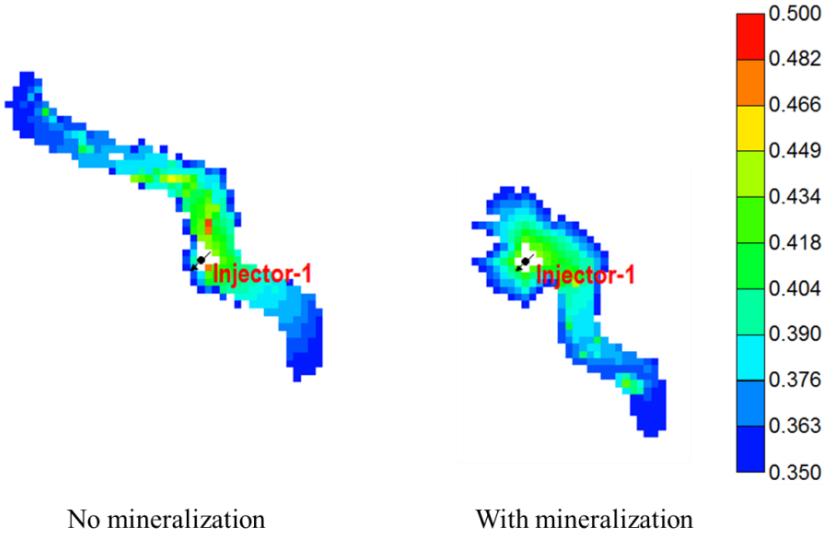


Figure 7.15. Difference in CO_2 saturation due to mineralization. High permeability pathways are blocked off and the plume seeks out alternative migration pathways

When there is no permeability alteration, the clusters after using dynamic data for different time durations are fairly close to one another, as shown in Figure 7.16. The proximity of the clusters to each other is expected because the underlying permeability

model does not change and the additional duration of the injection data only informs more details within the existing permeability models.

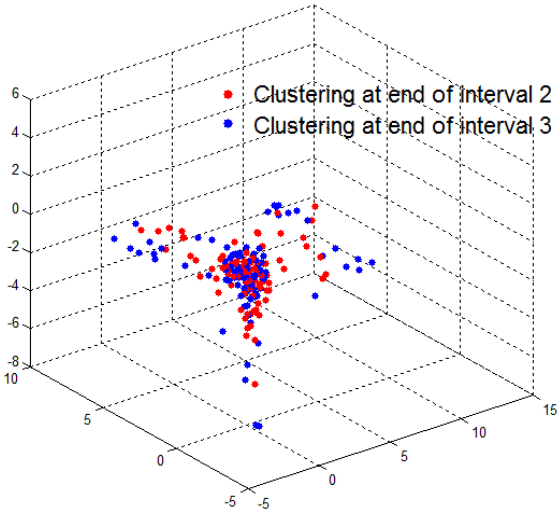
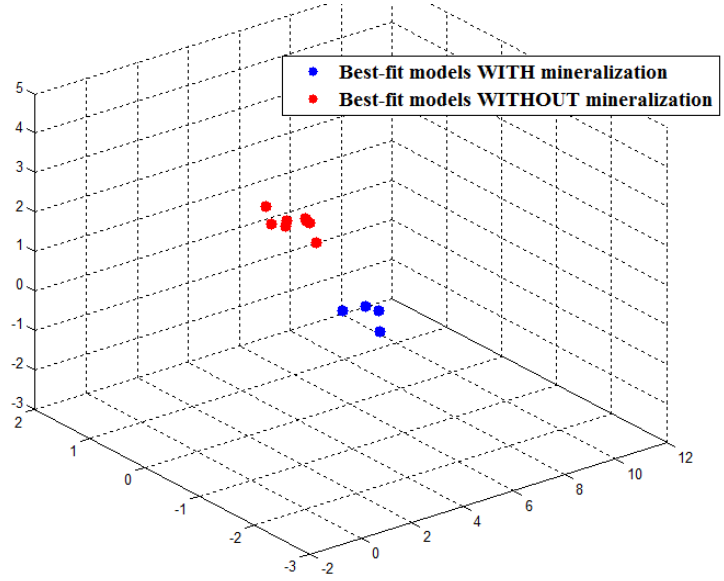


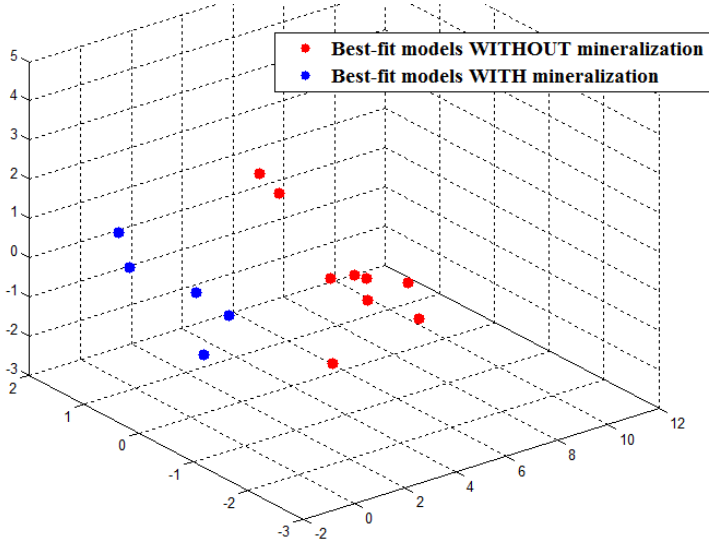
Figure 7.16. Clustering at the end of second and third time interval, in the case where no permeability alterations were assumed within the model selection process. The reference injection data used to select the models is affected by the mineralization process.

For the case **with permeability alteration**, the models were first run to the end of the first time interval, then based on the saturation at the end of this time interval, the permeabilities were altered and then run to the end of the second interval, and so on. The process thus followed the same procedure as was used to generate the reference history for the model selection process, described previously. At the end of each time interval, best-fit models were selected conditioned to the partial injection history up to that point in time. The best-fit models were then projected onto the principal component space, and compared with projections of the best-fit models selected from the case without permeability alteration. For this the covariance between each pair of models in the final set of models was computed and then subject to PCA. This is shown in Figure 7.17. There are different numbers of points of each color at different instants in time because

the final selected cluster at the end of model selection may have different numbers of models.



(a) End of second time interval



(b) End of third time interval

Figure 7.17. Best-fit models with / without mineralization projected onto an orthogonal space. (a) At the end of the 2nd interval, and (b) At the end of the 3rd interval.

Figure 7.17 clearly shows the gradual divergence of the two cases with time. In the case which accounts for mineralization, the continued injection of CO_2 causes the region of altered permeability to spread farther away from the injector, and thus exhibit marked difference in the characteristics of the selected model as we proceed in time. In Figure 7.17(a), the two clusters are separated but are still relatively close to each other; each cluster is also much tighter than those in Figure 7.17(b), when the model clusters have moved further apart from each other as permeability over large regions of the reservoir change due to mineralization. We can thus conclude that if mineralization is not taken into account within the model selection process, in the case that it is suspected that mineralization has a significant influence in the injection response, the models chosen by the model selection process will slowly diverge away from the 'real' model. In fact, if the permeability alteration were done much more frequently (smaller intervals of time), we would expect that this divergence would be more dramatic and take place much more quickly.

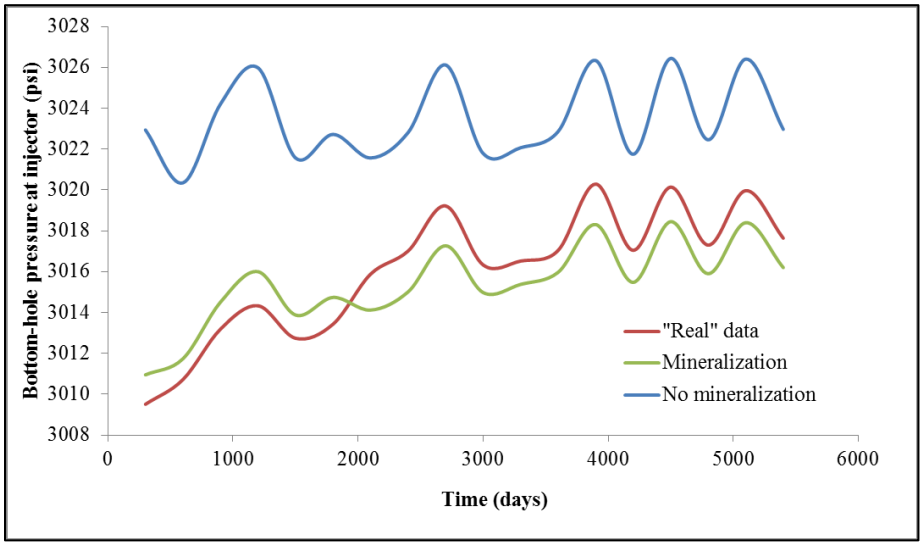


Figure 7.18. Injection well pressures compared to the "real" data for the synthetic model.

Figure 7.18 shows the match to the observed injection pressure data using the final selection of models for both cases. The model selection process does successfully find a group of best-fit models in both cases (with and without mineralization). But the "best-fit" is a rather poor match to the data in the case where the effect of mineralization has not been taken into consideration while performing model selection. In contrast, the group of models found in the case that includes mineralization (green curve) tracks the actual response well. Thus, comparing simulated response to field response during the model selection process would serve as a useful indicator of the possibility that there might be some phenomenon (in this case, mineralization) that is not being accounted for in our models.

7.3 LEAK INDICATION BY USING THE MODEL SELECTION ALGORITHM

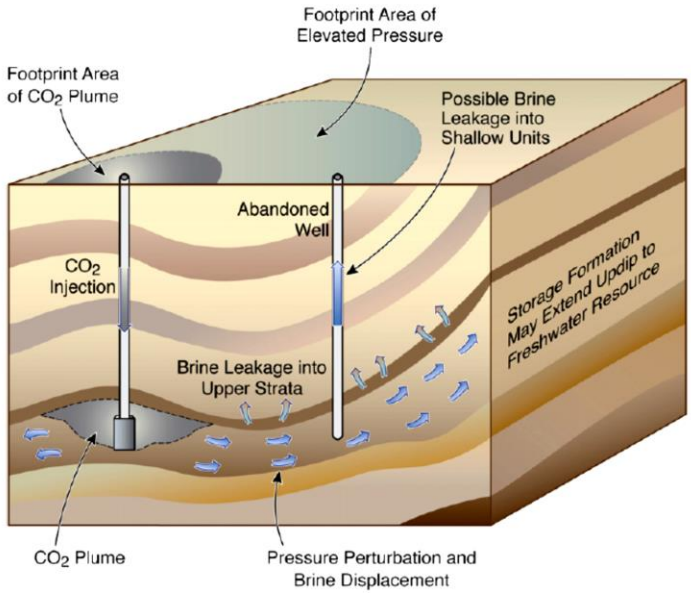


Figure 7.19. Leakage pathways for formation brine and CO_2 (from Birkholzer et. al., 2009)

The presence of a leak in the caprock would allow the formation brines and/or the injected CO_2 to leave the storage volume and migrate upwards to shallower formations (Figure 7.19). The leakage of formation brines could lead to contamination of shallower fresh water formations, and greater discharge into rivers and lakes. The problem is further worsened by the leakage of CO_2, which could be due to a an existing flow conduit (like an existing wellbore or through a transmitting fault / fracture), by reactivation of a closed fracture in the caprock (for example, through a transmitting fault or a fracture) or due to capillary leakage (when the phase pressure in the CO_2 exceeds the capillary entry pressure of the caprock). The effectiveness of long-term sequestration of CO_2 can be compromised by such leakage.

In this section, we first study CO_2 leakage through pre-existing open pathways using a commercial simulator (CMG-GEM©), and then provide a demonstration and discussion of the effect of leaks on the model selection process.

7.3.1. Simulation of CO_2 leakage and its effects on injection well pressure

The simulation model used for this study was based on the structure of the In Salah field in Algeria. The area was discretized using a 50x50x3 grid, with a leak simulated using a producing well (Well-P) at the crest of the structure and an injection well (Well-1) downdip from this location (Figure 7.20). The injection well introduced close to 1.3 million metric tons of CO_2 every year into the formation. The injection lasted 200 years, and then the CO_2 was allowed to migrate for another 300 years. The type of leakage, existing or reopened high permeability pathway, was implemented using different constraints at the production well.

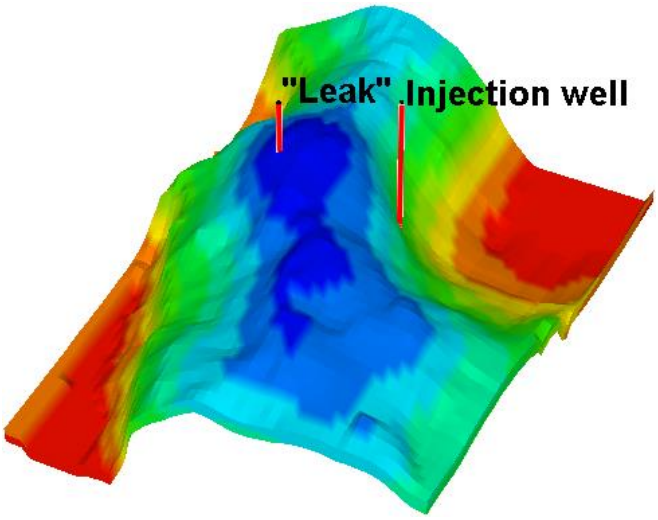


Figure 7.20. Layout of model used for leak simulation

For the case of leakage through a high permeability pathway, the only constraint placed on the production well was a maximum fluid rate constraint. This imposes the condition that the amount of fluid (brine and CO_2 combined) is limited by the flow capacity of the leakage pathway. As the CO_2 injection is started, it displaces the in-place brine, which is moved away from the injection well. The pressure signature of the injection is, however, felt over a much larger area than the area covered by the CO_2 plume, and this causes an initial leakage of formation brine through the leakage pathway. Once the CO_2 plume reaches the leakage site, the efflux of brine drops off sharply and the primary fluid leaving the storage volume is CO_2.

The concept of the brine leaking out of the reservoir due to the pressure increase induced by the injected CO_2 has been discussed in other works. Birkholzer *et.al.* (2009) discuss this issue during a study of the effects of the pressure footprint of injected CO_2. They stated that pressure increase and brine displacement could cause both lateral and vertical migration of formation brine out of the storage volume into shallower aquifers or

surface water accumulations. Further, they state that upward brine migration and pressure communication could occur through high permeability pathways like faults or abandoned boreholes. Zhou *et.al.* (2008) showed that storage reservoirs with imperfect seals may allow for enough displaced brine leaking out of the formation, while still having sufficient sealing capacity to trap supercritical CO_2.

The leakage of formation brine before any CO_2 even reaches the leakage site exhibits a distinct signature at the injection well. This is shown in Figure 7.21, where the difference in bottom-hole pressure at the injection well is plotted over time, between two cases without and without a leak. There is a distinct difference in response between the cases, which decreases significantly when the CO_2 finally reaches the leak site.

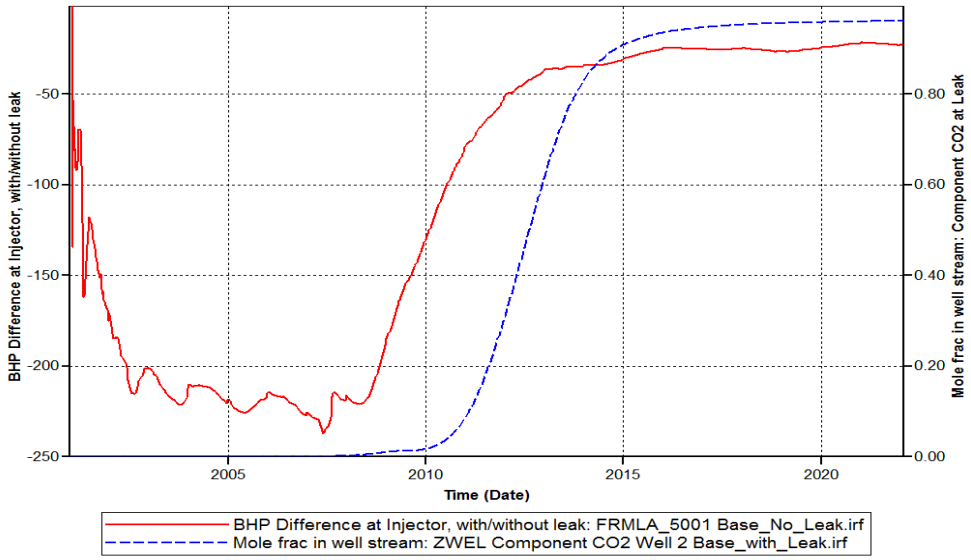


Figure 7.21. Difference between cases with/without leak. Plot shows BHP difference (in psi) at the injector in red, and mole fraction of CO_2 in the leaked fluid in blue

Based on the results above, we have come to the conclusion that the leak would have an observable effect on the bottom-hole pressure at the injector if there is substantial efflux of brine from the storage volume that is manifested in the form of substantial

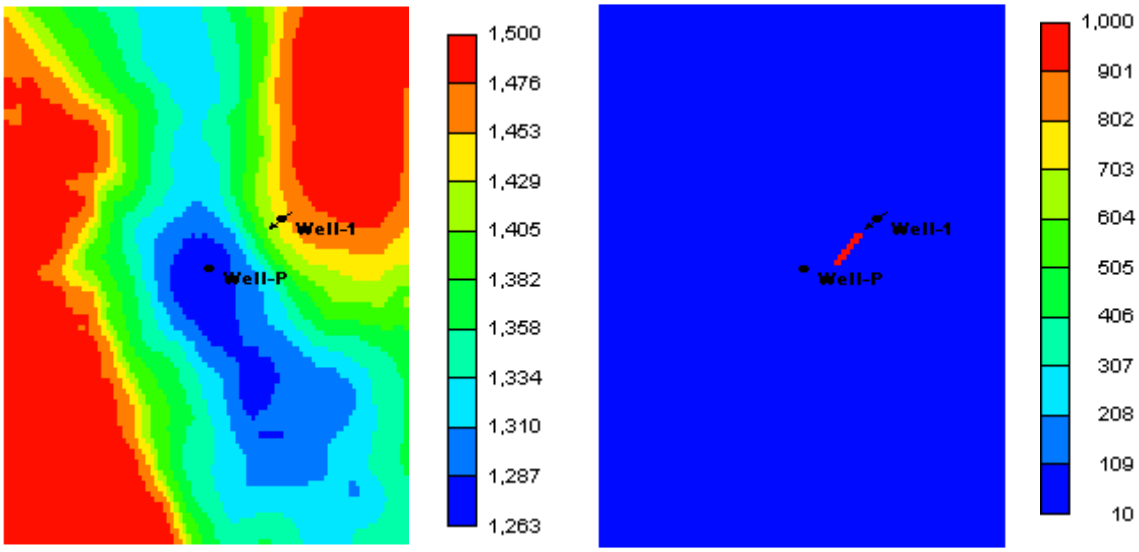
pressure perturbation caused by CO_2 injection. Once the CO_2 reaches the leak site, the effect of pressure signature at the injection well will diminish, and therefore not affect the model selection process as much.

7.3.2. Demonstration of influence of leak on model selection

In this part of our work, we seek to demonstrate that during model selection, failure to account for the presence of a leak could yield a completely different model set than would be created if the leak is accounted for, assuming the field data comes from a storage aquifer which does contain a leak. We will further show that the model set created by not taking the leak into account shows an injection profile that is clearly indicative of the fact that there are additional constraints we need to account for in our initial model set to get a better match to the field data. This would provide a way of getting some indication of the possibility of a leak using model selection conditioned only to injection/monitoring well pressure histories.

Synthetic Model to represent Real Field Data

For the purposes of this demonstration, we created an artificial aquifer model, in order to derive a 'real' dataset to condition our model selection process. The synthetic model consists of an anticlinal structure, with a CO_2 injection well in the flank of the anticline and a leak (represented by a producing well) at the crest of the structure, similar in structure to the model in the previous section. A high permeability pathway extends from near the injector to close to the leakage location (Figure 7.22).



(a) (b)

Figure 7.22. (a) Depth to top of layer (in m), showing the structure of the synthetic model, (b) Permeability of synthetic model (in mD)

Model Selection Process Implementation

The initial model set is made up of 700 models, 100 each created using high permeability features in 6 different directions, and 100 models with no high-permeability feature at all. This initial model set was run using two different types of aquifer models: model set A accounting for the leak and, model set B without accounting for the leak. The framework of this process is outlined in Figure 7.23. The model selection was run for two successive iterations for both sets, and the final model set for both cases were examined.

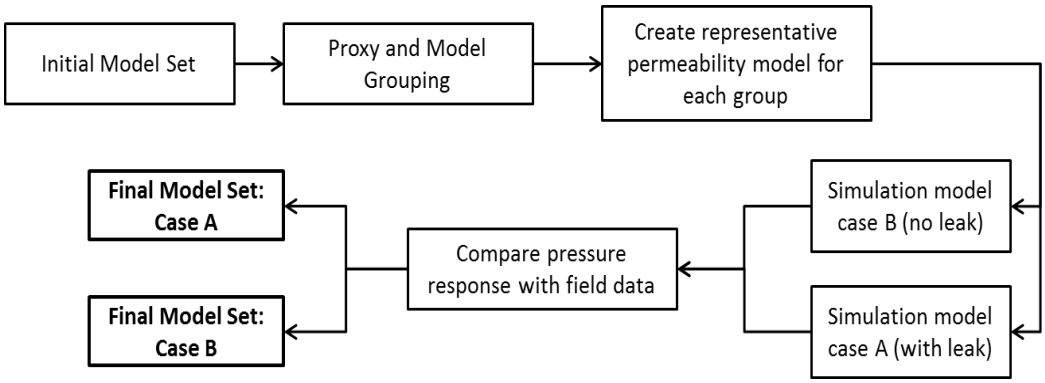


Figure 7.23. Workflow for model selection showing how the two cases with and without leak is incorporated

7.3.3. Results and Discussions

To show the effect of the leak on model selection, we compared the injection pressure profiles from the best-matched cluster to the 'real' field data for both cases, with and without the leak. The first indication of the presence of a leak is the difference in pressure of the final model set compared with the 'field data', as shown in Figure 7.24. The pressure profile for the representative model of the best-matched cluster, when the leak is not accounted for, differs from the 'real' data by a greater amount than when the leak is accounted for. This clearly indicates that in order for the model selection process to reflect accurate results, it is necessary that the procedure and the proxy incorporate the effect of the presence of a leak.

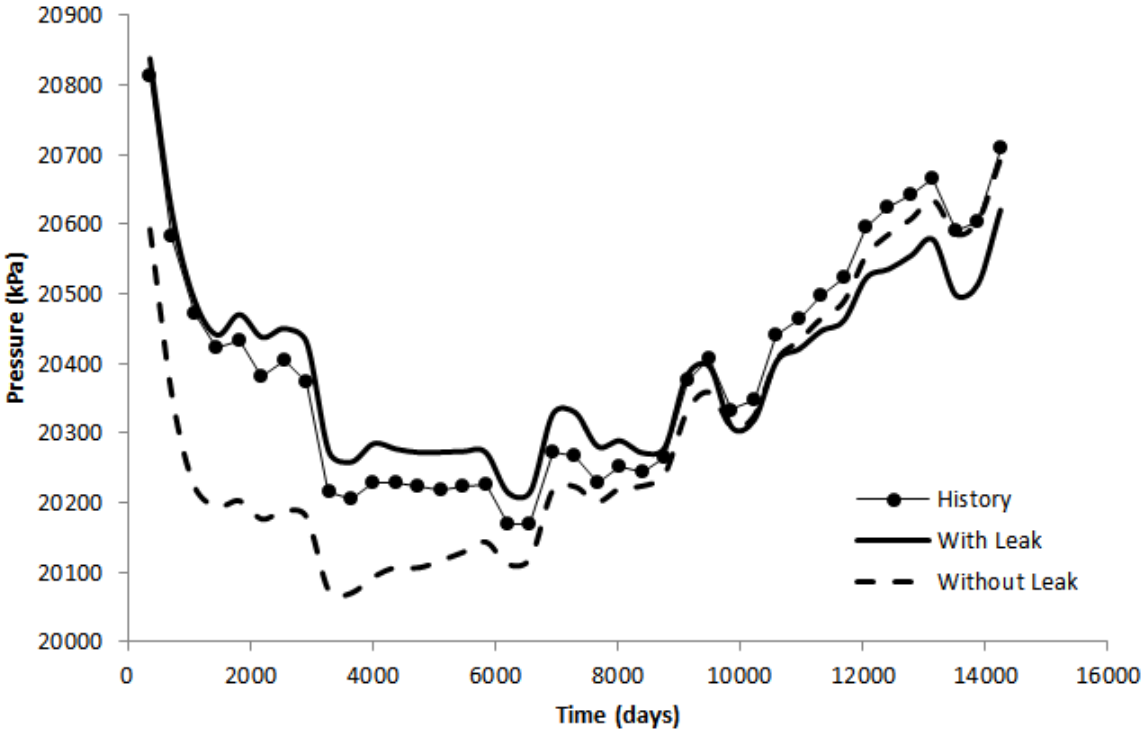
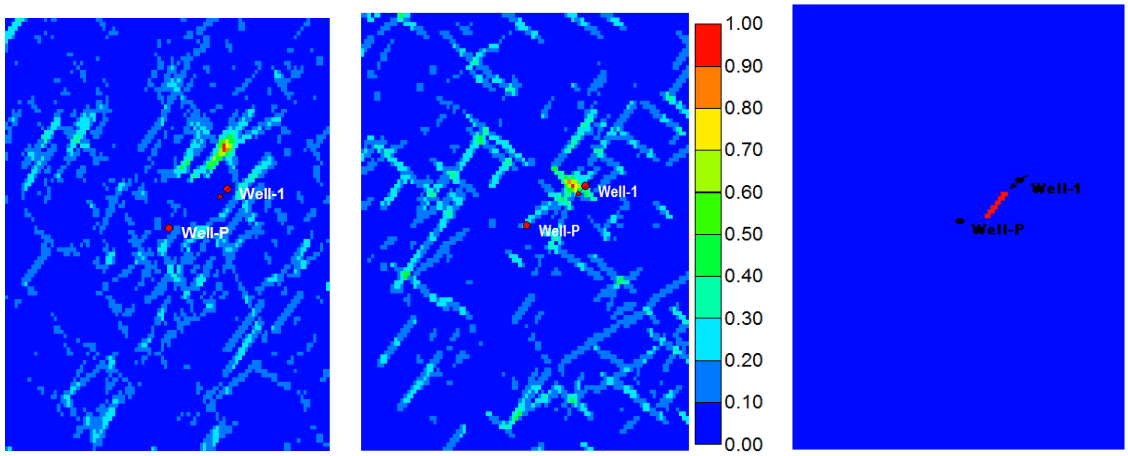


Figure 7.24. Injection pressure response comparison for best matched cluster, before and after accounting for the leak

It is even more instructive to look at the models that make up the final clusters for both cases. Since the initial set is made up of models that exhibit permeability pathways (streaks) in 6 different directions, it is reasonable that the final model set would also contain models that exhibit permeability pathways in different directions. However, in both cases, model selection is able to pick models that show streaks starting near the injector and reject all models that do not have streaks close to the injector. Indeed, in spite of having 100 models with no streaks at all, the model selection was able to pick only those models that did have streaks close to the injector.

To understand the effect of including the leak in the model selection process, we constructed a probability map of the existence of high permeability pathways at different

locations in the final model sets. This is shown in Figure 7.25. The 'reference' field model consisted of a streak starting close to the injector, moving updip towards the leak location. When the leak is not accounted for during model selection (case B), the probability map shows a high probability of the existence of streaks close to the injector; however, this streaks is in a different direction from that in the 'reference' model, moving along the strike of the anticline. If the leak is accounted for in the model selection process (case A in Figure 7.23), the probability map again shows a high-permeability streak close to the injector; but unlike the previous case, this streak is in the updip direction towards the injector, thus reflecting better the geologic setting of the 'reference' model.



(a) (b) (c)

Figure 7.25. Probability map for high permeability streaks, derived from final model set (a) Leak accounted for during model selection, (b) Leak NOT accounted for during model selection. Compare with the 'reference' field permeability (c)

The difference in final model sets is also assessed through the projections of the final models on the principal component axes of the proxy-derived statistics. The statistics derived from the proxy run on all the initial models was used to project the models onto an orthogonal set of axes determined by principal component analysis.

These projections are shown in Figure 7.26. It is clear that the model sets created with and without accounting for the leak are completely different with very little overlap.

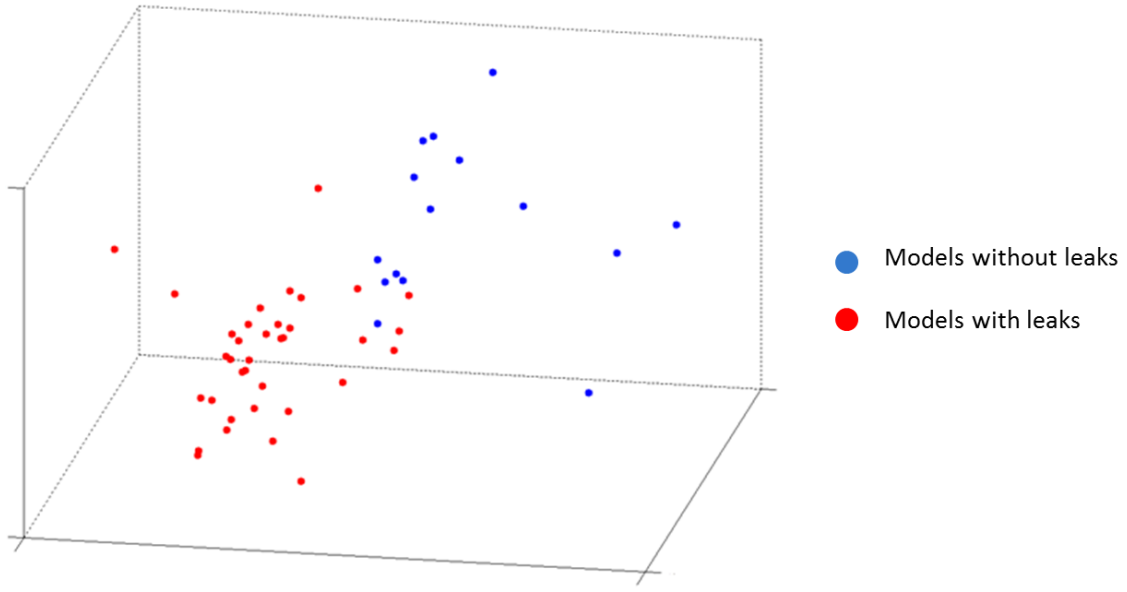


Figure 7.26. A projection of final model sets, with and without leaks, clearly shows the separation of the two cases.

The discussion above clearly demonstrates that the presence of a leak can be reflected in the injection well pressures, and hence can affect the final set of models obtained by the model selection process.

7.4 INFLUENCE OF LOCATION OF CONDITIONING WELLS ON MODEL SELECTION

During the course of the model selection process, we have always implicitly assumed that the injection data used to condition the entire process was adequate to inform the process of delineating prominent features within the final model set. However, it can be stated that not all injection data used for the conditioning process has the same influence on the model selection process; the proximity of the injection location with

respect to prominent features driving the fluid migration should have a direct bearing on the ability of the model selection process to highlight that particular feature. In this section, we investigate this hypothesis in greater detail.

7.4.1. Synthetic model to study the effect of injector location on the model selection process

To investigate the effect of proximity of injector location to prominent reservoir features on the performance of the model selection process, we used a synthetic model

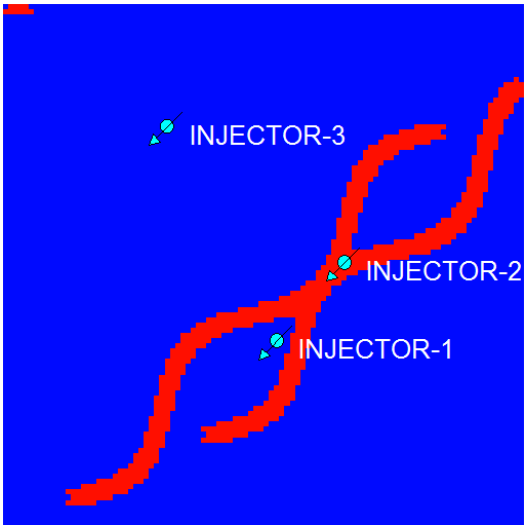


Figure 7.27. Schematic of base case model used to study the effect of proximity of injector to prominent features on the model selection process

containing a prominent sinusoidal high-permeability channel feature embedded in a low permeability over-bank deposit, and three injectors located at different proximities to the channel location. The model is shown in Figure 7.27. The distinctly different locations of the injectors with respect to the channel allowed us to use the bottom-hole pressure data from each of these three wells independently and test the aforementioned hypothesis.

The given model was run forward for 50 years with the wells constrained by fluctuating injection rates, and the injection pressure from all three wells was noted and used later to condition the model selection process. The injection pressure histories for the three wells are shown in Figure 7.28.

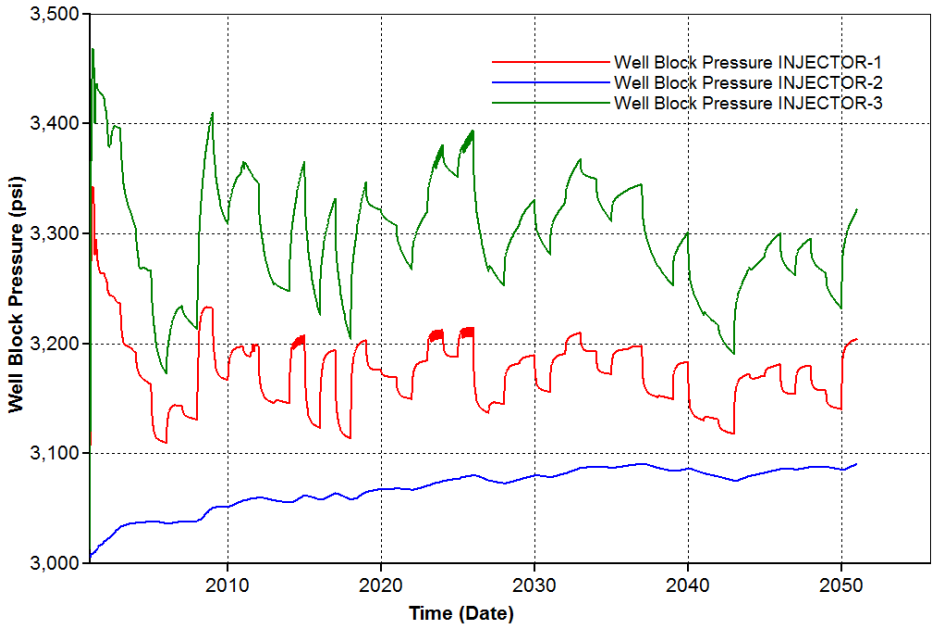


Figure 7.28. Injection pressure history for all three injectors. These histories were used individually in the model selection process.

7.4.2. Model selection process conditioned to each individual pressure history

The initial model set created for this model selection process was composed of high permeability sinusoidal pathways embedded in a low permeability matrix, similar to the base case. However, the location of these high permeability pathways was created unconditioned to any data, so the actual locations of the sinusoidal features are not restricted to any particular location or region within the grid. Some of these models are shown in Figure 7.29, which highlights the completely random nature of the location of the high permeability pathways.

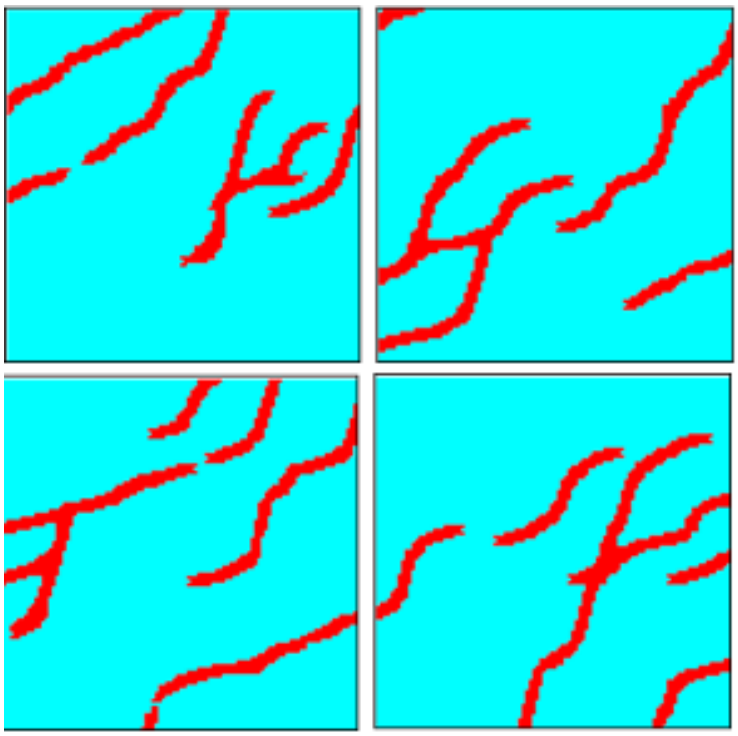
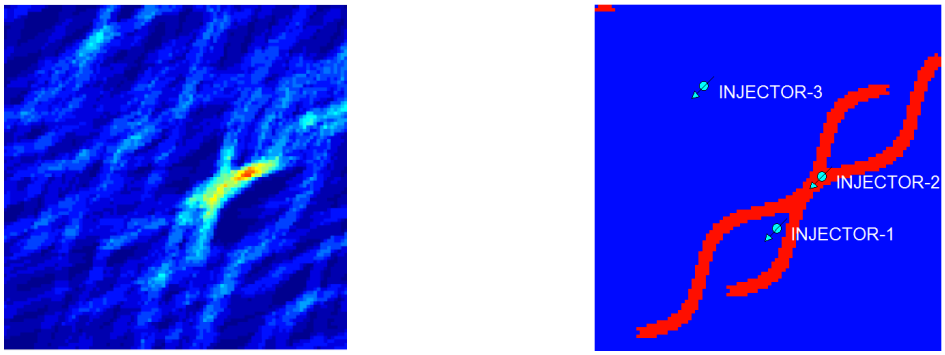


Figure 7.29. Sample of models from the initial model set (out of a total of 120 models), showing the random distribution of the high-permeability channels (in red).

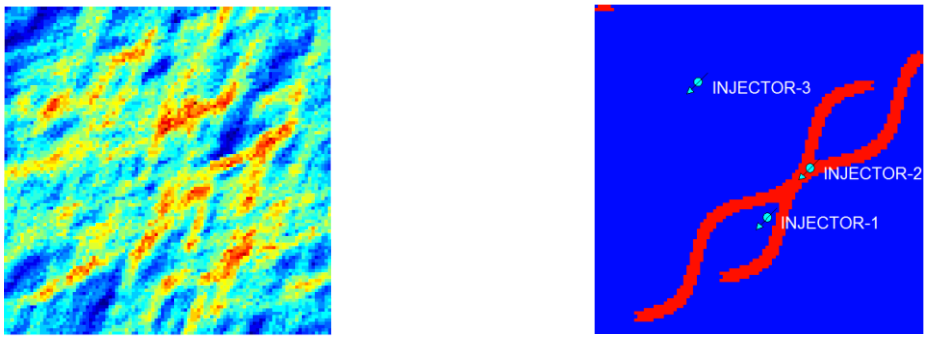
Three distinct cases were run with the particle-tracking proxy, with the same model set but different injection location corresponding to each of the injectors in the base case. The conditioning data for each case was the bottom-hole pressure for that particular injector. So, for example, if the location of the injector for the proxy corresponded to the well 'INJECTOR-1', then the conditioning data would be the bottom-hole pressure for 'INJECTOR-1'. This yielded three different final best-fit model sets, each of which were studied to see what prominent features, if any, were present across all (or a majority) of the models.

When the injection location used was 'INJECTOR-2', which was located inside the channel, the model selection process was able to pick up a dominant high permeability feature close to the actual base case. The dominant feature was inferred by finding the mean of the models in the final model set, as seen in Figure 7.30**.**



Average model from best-fit models **'Real' model**

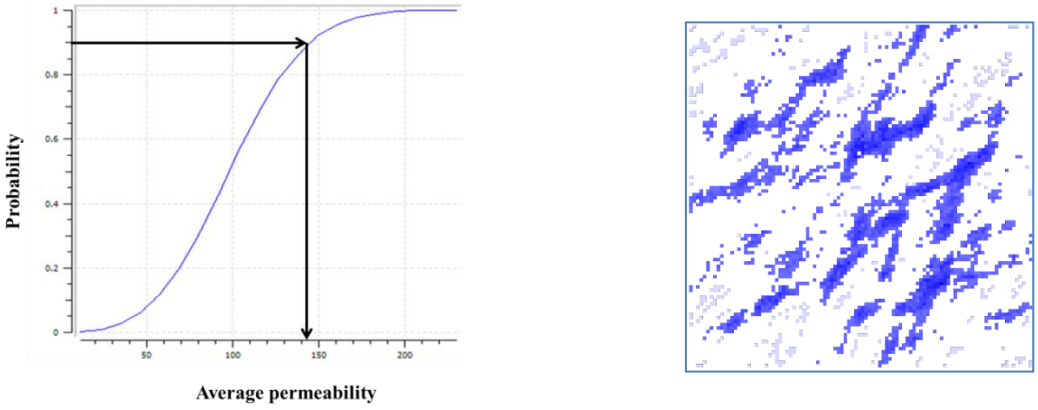
Figure 7.30. Comparison of average model from best-fit cluster, conditioned to INJECTOR-2, shows that it can delineate the feature of interest to a reasonable degree



Average model from best-fit models **'Real' model**

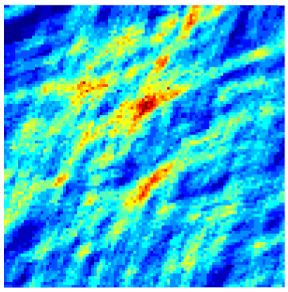
Figure 7.31. Comparison of the average of best-fit models, conditioned to INJECTOR-1, shows that it does not delineate the feature as sharply as in previous case (Figure 7.30)

When, however, the conditioning well was 'INJECTOR-1' which lay just outside the channel, the degree of detail visible in the ensemble average was greatly reduced (Figure 7.32). In fact, the average of the best-fit models showed that there were some prominent features indicated in several regions of the reservoir; however, these features were not as crisp as in the previous case, and there were a number of spurious features that were not present in the original model. Another way to look at the dominant feature is to look at values in the average model that lie in the top 90-percentile of the distribution of that model. This model representation is shown in Figure 7.32(b), and highlights the prominent features outlined by the best-fit models more clearly. The average model contains values between 10 mD and 230 mD, but in this case the 90-percentile cutoff means that only locations with values greater than 140 mD (from the distribution of values in the average model, Figure 7.32(a)) are shown. As can be clearly seen, it delineates the feature close to INJECTOR-1 and INJECTOR-2 to a degree, but the result also contains a lot of 'spurious' features in more distant parts of the reservoir.

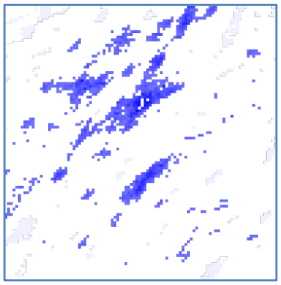


(a) (b)

Figure 7.32. (a) Distribution of values in average model, (b) Prominent features in average model highlighted by taking out all values below the 90th percentile



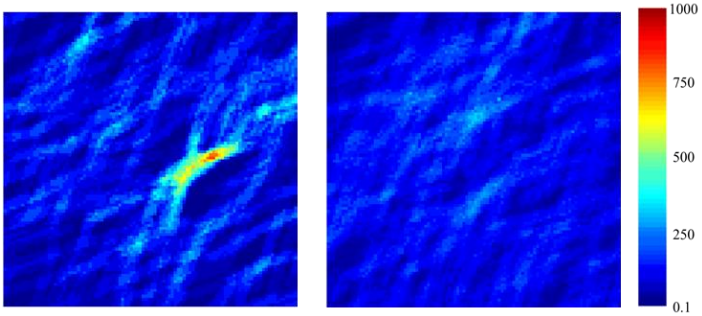
Average model for best matched cluster



Prominent features highlighted by taking out all values less than the 90th percentile

Figure 7.33. Average model of best-matched cluster, when the conditioning data is from INJECTOR-3, far away from the prominent sinusoidal feature.

This effect becomes even more prominent when the conditioning well used is 'INJECTOR-3', which lies far away from the feature of interest. The features highlighted are not even in the vicinity of the actual sinusoidal feature (Figure 7.33). In fact, the common features in the final selected cluster are not that prominent when the selection is performed using the data for 'INJECTOR-3'. This is demonstrated by showing the average models from cases with 'INJECTOR-3' and 'INJECTOR-2' on the same scale, as seen in Figure 7.34.



Average model, conditioning data INJECTOR-2

Average model, conditioning data INJECTOR-3

Figure 7.34. Comparison of average model, with conditioning well inside and outside the channel feature.

It might be argued that the bottom-hole pressure response of a given injector is not only a function of the permeability features in its proximity, but also the injection behavior of other wells in the vicinity. This hold true for the effect of INJECTOR-1 on INJECTOR-3. However, as seen in Figure 7.35, the well INJECTOR-2 *does not* have a

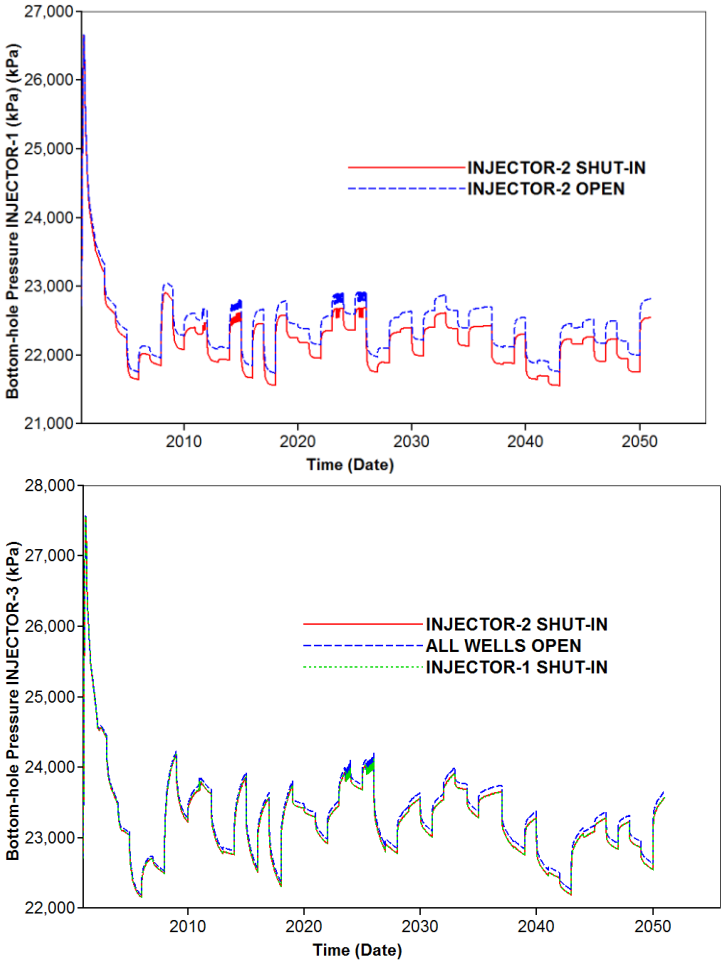


Figure 7.35. Inter-well pressure effects of the three injectors

very prominent effect on INJECTOR-1, and INJECTOR-3 is almost unaffected by the other two injectors. As such, the results from INJECTOR-1 and INJECTOR-3

conditioned model selection hold good even if the inter-well effects are taken into account.

It is also instructive to compare the response of the representative models of the identified clusters to the conditioning well response, as seen in Figure 7.36. In this case, it

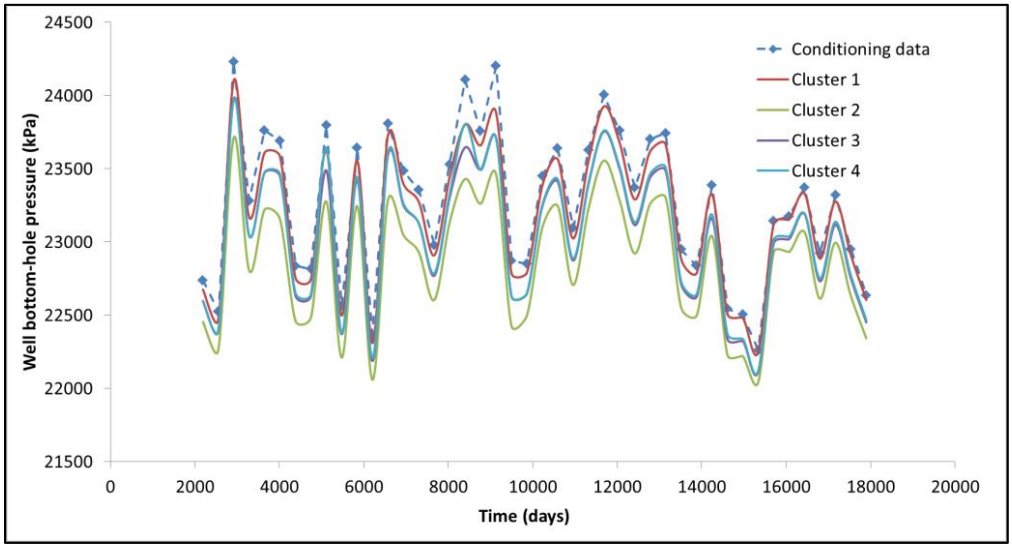


Figure 7.36. Comparison of response of representative model of individual clusters to reference data, when model selection is conditioned to INJECTOR-3

is clear that the models themselves have very few features that differentiate the pressure responses of each cluster from one another. Furthermore, the posterior probability of the clusters (using the Bayesian updating equation 3.5) was calculated to be 20.9%, 23.2%, 27.9% and 27.9%, which clearly indicates that the data is inadequate to inform the model selection process as the contrast in the posterior probabilities is not that significant. A similar calculation performed for the case when the injector was inside the channel (Figure 7.37), showed the updated probability of the two clusters in that case to be 27.2% and 72.7%, which clearly points to cluster 2 as the more probable cluster of models in that case.

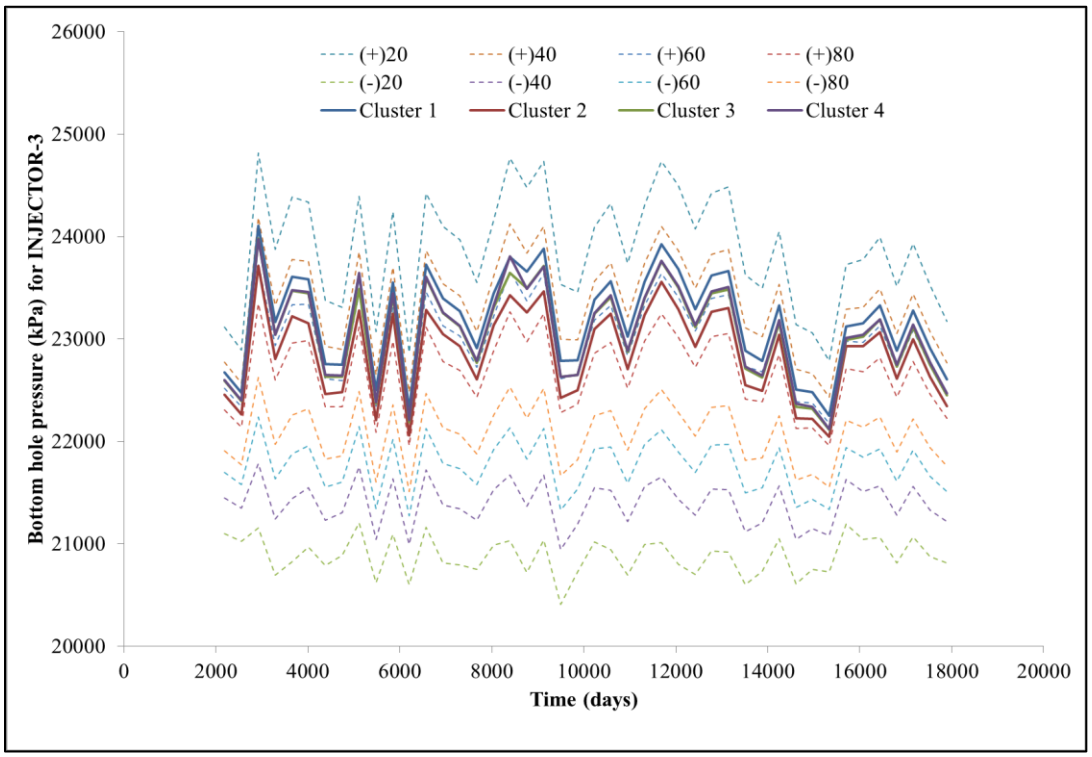


Figure 7.37. Probability cluster for calculating the posterior updated probability of the model clusters, when the conditioning well is far from the channel.

This calculation of posterior probabilities can thus clearly show when the conditioning data is inadequate to inform the model selection process, and thus serves as an efficient stopping mechanism for the algorithm.

We showed how the model selection process is not an indiscriminate workflow that can be used in *all* cases. We demonstrated that the process will fail to yield any robust result in cases when the conditioning data is not available close to any prominent features. However, at the same time, the calculation of posterior probabilities for each cluster can clearly show when the conditioning data is inadequate, and hence can provide an effective criterion for terminating the process.

7.5 CONCLUSIONS

In this chapter, we have explored the application of the model selection algorithm and the proxy, and demonstrated its use in a wide variety of applications. First, we showed an efficient technique to locate the proxy monitoring locations such that the differences between the models in a flow connectivity context can be emphasized. Next, we showed how the model selection process can be used as an indicator of active mineralization within the aquifer during CO_2 sequestration, leading to a continuous process of permeability and porosity alteration. Third, we showed the application of model selection as an indicator for a leak within the storage volume. While the algorithm cannot, at this stage, yield estimates of location of the leak, it can very well point to the presence of one. Finally, we showed how the model selection process is not an indiscriminate workflow that can be used in *all* cases. We demonstrated that the process will fail to yield any robust result in cases when the conditioning data is not available close to any prominent features. However, at the same time, the calculation of posterior probabilities for each cluster can clearly show when the conditioning data is inadequate, and hence can provide an effective criterion for terminating the process.

Chapter 8 : Conclusions and Recommendations for Future Work

8.1. CONCLUSIONS

Probabilistic assessment of plume migration during geologic carbon sequestration requires the use of multiple models to reflect the initial uncertainty in aquifer geologic parameters. The assessment of these models needs efficient forward models that can overcome the computational cost of numerical simulators. In Chapter 1, it was hypothesized that *efficient forward models can be developed for simulating the flow of CO_2 in an aquifer at a fraction of the computational cost of a numerical simulator, and can be implemented within the model selection framework to predict future plume migration*, and the following objectives were envisioned for addressing the problem:

1. Development of fast-transfer functions
2. Validation of fast-transfer functions within model selection framework
3. Development of a software suite to implement the algorithm

Before these objectives could be addressed, the model selection workflow needed to be modified for the carbon sequestration case, and this was shown in Chapter 3. While initial implementations of the workflow relied on user-defined monitoring locations to record proxy measurements for model grouping, a method was also developed for using the variability across models to pick optimum locations for recording the measurements. A way of calculating the optimum number of model groups was also developed, together with a scheme for computing a representative aquifer model for each model cluster. Finally, computation of posterior probabilities of model clusters using Bayes' rule was facilitated by the use of probability envelopes around the observed data.

The first and second objectives were addressed in Chapters 4 and 5. Chapter 4 detailed the development of a proxy based on random walker particle tracking. The proxy

was shown to be capable of capturing viscosity-driven CO_2 migration in the aquifer by comparing it with numerical flow simulations, both for synthetic cases and a real field case (In Salah field in Algeria). It was also tested within the model selection framework, and shown to be effective in delineating best-fit models for the aquifer conditioned to injection well pressures. However, the first proxy was not adequate for capturing migration in cases when buoyancy-driven flow was dominant, as demonstrated for a synthetic two-dimensional vertical model. and a real field case (Sleipner field in the North Sea). The problem can be alleviated to an extent by weighting the vertical migration component in the transition probabilities higher; however, this is a calibration exercise rather than an actual physical reflection of fluid transport. Even with a higher weight assigned to the vertical migration component, the proxy fails to adequately capture the migration, as shown in the case of the Sleipner field. Further, multiphase flow effects described by relative permeability, variations in injection pressures and fluid density are not taken into account at all in this present formulation of the proxy.

Chapter 5 detailed the development of a new proxy to overcome the limitations of the first proxy. The proxy mimicked the flow of non-reactive tracers with injected fluid. While it retained the effect of permeability and reservoir structure like in the previous proxy, it additionally incorporated the effects of gravity, relative permeability and fluctuations in injection rates. It was shown to be still efficient in capturing viscosity-driven migration, while also being able to capture migration in buoyancy-dominated cases. The new proxy was also tested within the model selection framework for both the Sleipner and In Salah cases. It was able to capture the location of a high-permeability pathway between problem wells in In Salah, and was shown to be efficient in capturing the location of sand holes in shale layers in Sleipner.

The final objective of developing a software suite for implementing the various parts of the model selection algorithm was detailed in chapter 6. This was coded in C++ and implemented within the geostatistical software SGeMS. The details of the code are available in Chapter 6 and Appendix A. The software module was tested on both the field cases in Chapter 5. Further, it was implemented on some additional applications of the model selection process in Chapter 7, like the effects of cap-rock leak on the model selection process, the limitations of injection data for delineating best-fit models depending on the location of the conditioning wells, and the possibility of using the algorithm to indicate the presence of mineralization in the aquifer.

8.2. RECOMMENDATIONS FOR FUTURE WORK

Based on the work implemented in this dissertation, the following recommendations are made to improve and extend the model selection process:

- **Regeneration of models:** The current implementation of the model selection process terminates the process either when the posterior probabilities of model clusters become uniform, or when the number of models is deemed improbable. The second termination rule, however, is a limitation to the model selection workflow rather than a concrete statement. If a method can be devised for regenerating new models at the end of each iteration (Figure 8.1), the termination condition will then be limited only to equiprobable posterior probabilities. Further, if the model regeneration is based on characteristic features in the best-fit cluster, the new models will be able to emphasize the dominant features while also adding new features to the models. This would make the model selection an iterative process of incorporating heterogeneity

features within the model set, and might progressively lead to tighter clusters with reduced spread in prediction.

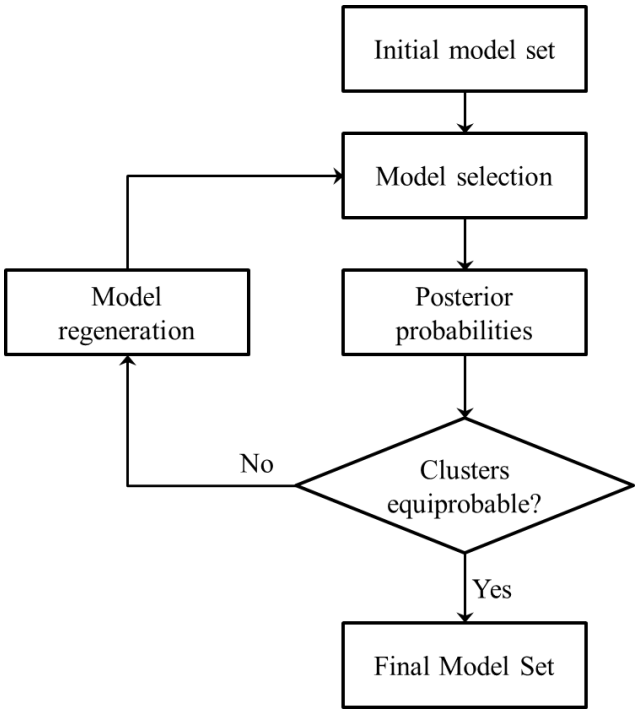


Figure 8.1. Modified model selection algorithm with model regeneration

- **Incorporating additional physics into the proxy:** The proxy formulation has been updated to incorporate various physics into the transition probability calculation; however, it is still missing a capillary pressure term and a CO_2-brine dissolution term. Both of these effects might be incorporated into the proxy using the fractional-flow theory. Capillary pressure is included in the fractional flow formulation as shown in Equation 5.5:

$$f_w = \frac{1 + (\frac{kk_{rnw}A}{q\mu_{nw}}(\frac{\partial P_c}{\partial x} - \Delta\rho g sin\alpha)}{1 + \frac{k_{rnw}}{\mu_{nw}}\frac{\mu_w}{k_{rw}}} \tag{5.5}$$

This expression for the fractional flow value can be used in the $(1 + f_W)$ term in equation 5.9. However, since a closed form solution with capillary pressure does not exist for the Buckley-Leverett equation, this should not be used in the specific velocity term in equation 5.9.

This can be combined with the fractional flow method developed by Noh et.al. (2007) for modeling CO_2 injection in aquifers, which also addresses the effects of dissolution of CO_2 in aquifer brine. The authors described two displacing fronts created within the aquifer: a pure CO_2 front displacing two-phase CO_2-brine mixture (the drying front), and the two-phase front displacing pure in-place aquifer brine (the saturating front). This is shown in Figure 8.2.

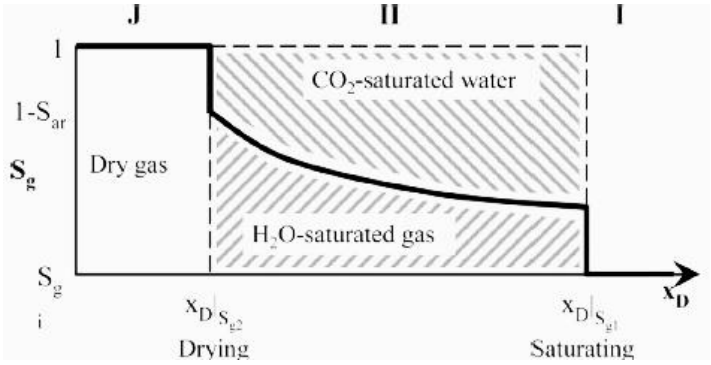


Figure 8.2. CO_2-phase saturation profile in an aquifer (from Noh et.al. 2007).

The velocities of the two-fronts are given as slopes of tangents to the fractional flow curve from points expressed as functions of CO_2

dissolution in brine, as is shown in Figure 8.3. The coordinates of points I and J are functions of the partitioning of the CO_2 component between the aqueous and gas phases.

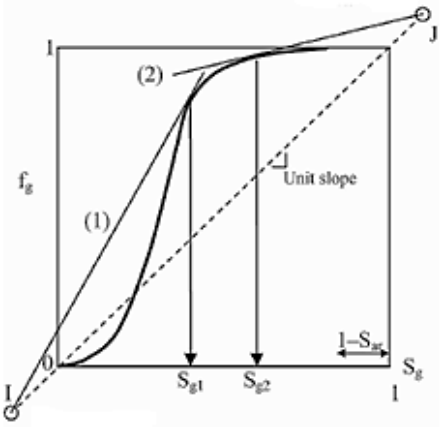


Figure 8.3. Front velocities from fractional-flow curve (from Noh et.al. 2007).

- **Improvements to posterior probability calculation:** The posterior probability calculation is based on creating probability envelopes around the observed data. Further, the distribution of the envelopes is assumed to be Gaussian. To make the process more general, there is need to develop a process of stepping away from the Gaussian assumption. Also, the conditioning data used for the model selection process is based on a single variable. There is need to incorporate data from multiple sources such that the process can be conditioned to a wider set of data and probably yield a richer posterior model set.

Appendix A: Code for old random walker

```
#include <iostream.h>
#include <iomanip.h>
#include <fstream.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <time.h>
#include <conio.h>
#include <string.h>
#include <windows.h>

char Parfile[80],Prfile[250],Sfile[250],cs_pr1[250],cs_pr2[250],sat_time[250];
double Kmax[200],Pmax,pr_max[100],sat_max[100],sat_min[100];
double perm[200][200][200][2],Press[200][200][2],Pr[200],sat[200][200][2],mon_pr[10000][200],mon_sat[10000][200];
int
NTotal,Tsteps,dx,dy,dz,NX,NY,NZ,Nsource,sx,sy,sz,mx,my,mz,NInitial,models,Pcount[200][200][2],Preport[200][200][200][2],Tco
unt[150][150][5],flag4,flag5,flag6,flag7,flag8;
void ReadParameter(void);
double pr_dist(int,int,int,int);
const int MIN=5,MAX=50;

void main(void)
{
	int i,j,k,l,p,rndm,report=10,total=0,t=0;
	int *rep = new int[report];
	long start,stop;
	double cdf,pdf,pr_val;
	int tt=0;

	ReadParameter();
	cout<<"\nNumber of particles injected every time step:  ";
	cin>>NTotal;
	cout<<"\nNumber of steps to be taken:  ";
	cin>>Tsteps;
	start=time(0);
	ofstream monitor_pr1(cs_pr1);
	ofstream monitor_pr2(cs_pr2);
	ofstream monitor_sat(sat_time);
	ofstream satn(Sfile);
	ofstream pres(Prfile);
	ofstream chk("check.dat");
	remove("Echeck.xls");
	const int a=1,b=1;
	double add=1.0,subt=0.5,c=10;

	// ****************** creating the permeability field ******************
	for ( i=1;i<=NX;i++)
	{
		for ( j=1;j<=NY;j++)
		{
			for ( k=1;k<=NZ;k++)
			{
				for (l=1;l<=models;l++)
				{
					perm[l][i][j][k]=perm[l][i][j][k]/Kmax[l];
				}
			}
		}
	}
```

```
	// ***************** starting the walk *********************
	for (l=1;l<=models;l++)
	{
		int Npart=0;
//		pres<<"\nModel "<<l<<"\n";


		for (i=1;i<=NZ;i++)
			{
			for (j=1;j<=NY;j++)
				{
				for (k=1;k<=NX;k++)
					{
					Pcount[k][j][i]=0;
					}
				}
			}


	for (int t1=1;t1<=Tsteps;t1++)
	{
		int count=0;
		//cout<<"\n**********************\n"<<t1<<"\n**********************\n";

		for (int np1=1;np1<=NTotal;np1++)
		{
			int zzz=1;
			Npart++;
				i=sx; j=sy; k=sz;
				Pcount[i][j][k]+=1;

				do
				{
				for (p=1; p<=5; p++)
					{Pr[p]=0;}



					if (i>1 && i<NX && j>1 && j<NY && Pcount[i][j][k]>=MIN)
						{
						Pr[2]=(a*((Pcount[i][j][k]-Pcount[i-1][j][k])/Npart)+b*pow(perm[l][i-
1][j][k]*perm[l][i][j][k],0.5))*exp(-Pcount[i-1][j][k]/MAX);
						Pr[3]=(a*((Pcount[i][j][k]-
Pcount[i+1][j][k])/Npart)+b*pow(perm[l][i+1][j][k]*perm[l][i][j][k],0.5))*exp(-Pcount[i+1][j][k]/MAX);
						Pr[4]=(a*((Pcount[i][j][k]-Pcount[i][j-1][k])/Npart)+b*pow(perm[l][i][j-
1][k]*perm[l][i][j][k],0.5))*exp(-Pcount[i][j-1][k]/MAX);
					   Pr[5]=(a*((Pcount[i][j][k]-
Pcount[i][j+1][k])/Npart)+b*pow(perm[l][i][j+1][k]*perm[l][i][j][k],0.5))*exp(-Pcount[i][j+1][k]/MAX);
						Pr[1]=(a*(Pcount[i][j][k]/Npart)+b*perm[l][i][j][k])*exp(-Pcount[i][j][k]/MAX);
					}
					else
					{
						Pr[5]=0; Pr[2]=0; Pr[3]=0; Pr[4]=0; Pr[1]=1;
					}


					if (i>1 && i<NX && j>1 && j<NY && Pcount[i][j][k]>=MAX)
					{
						Pr[2]=(a*((Pcount[i][j][k]-Pcount[i-1][j][k])/Npart)+b*pow(perm[l][i-1][j][k]*perm[l][i][j][k],0.5));
						Pr[3]=(a*((Pcount[i][j][k]-
Pcount[i+1][j][k])/Npart)+b*pow(perm[l][i+1][j][k]*perm[l][i][j][k],0.5));
						Pr[4]=(a*((Pcount[i][j][k]-Pcount[i][j-1][k])/Npart)+b*pow(perm[l][i][j-1][k]*perm[l][i][j][k],0.5));
					   Pr[5]=(a*((Pcount[i][j][k]-Pcount[i][j+1][k])/Npart)+b*pow(perm[l][i][j+1][k]*perm[l][i][j][k],0.5));
						Pr[1]=0;
```

```
                }

                double sum=0;

                for (p=1; p<=5; p++)
                        {sum+=Pr[p];}
                if (sum>0)
                {
                        //if (t1%50==0) {cout<<t1<<"              "<<Pr[1]/sum<<"       "<<Pr[2]/sum<<"       "<<Pr[3]/sum<<"
"<<Pr[4]/sum<<"       "<<Pr[5]/sum<<"\n";}
                        rndm=rand()%100;
                        cdf=0;
                        //cout<<rndm<<"\n";
                        for (p=1; p<=5; p++)
                        {
                                pdf=Pr[p]/sum;
                                cdf=cdf+pdf;
                                //cout<<cdf<<"          ";
                                if (rndm<cdf*100) {break;}
                        }
                        //cout<<"\n";


                        switch (p)
                        {
                                case 2:
                                        dx=i-1; dy=j; dz=k;
                                        break;
                                case 3:
                                        dx=i+1; dy=j; dz=k;
                                        break;
                                case 4:
                                        dx=i; dy=j-1; dz=k;
                                        break;
                                case 5:
                                        dx=i; dy=j+1; dz=k;
                                        break;
                                case 1:
                                        dx=i; dy=j; dz=k;
                                        break;
                        }
                }

                Pcount[dx][dy][dz]+=1;
                Pcount[i][j][k]-=1;
                if (dx==i && dy==j && dz==k)
                {
                        zzz=0;
                }
                else
                {
                        i=dx; j=dy; k=dz; zzz=1;
                }
                }while(zzz==1);// ********** end of while loop ************


        }// ******* end of 1 timestep ************

        for (i=1;i<=NZ;i++)
                {
                  for (j=1;j<=NY;j++)
                        {
                                for (k=1;k<=NX;k++)
                                        {
```

```
                                Preport[l][k][j][i]=Pcount[k][j][i];
                            }
                        }
                }

        mon_sat[t1][l]=Preport[l][mx][my][mz];
        if (mon_sat[t1][l]>mon_sat[t1-1][l] && t1>1) sat_max[l]=mon_sat[t1][l];
        if (mon_sat[t1][l]<mon_sat[t1-1][l] && t1>1) sat_min[l]=mon_sat[t1][l];
        mon_pr[t1][l]=pr_dist(mz,my,mx,l);
        if (mon_pr[t1][l]>mon_pr[t1-1][l] && t1>1) pr_max[l]=mon_pr[t1][l];

} // end of all time steps //


    }// ********************* end of all models ************************

    pres<<"Pressure Profile\n"<<models+3<<"\nX\nY\nZ\n";
    satn<<"Saturation\n"<<models+3<<"\nX\nY\nZ\n";
    for (int fff=0;fff<models;fff++)
    {
        satn<<"model "<<fff+1<<"\n";
        pres<<"model "<<fff+1<<"\n";
    }

                for (i=1;i<=NZ;i++)
                    {
                    for (j=1;j<=NY;j++)
                        {
                        for (k=1;k<=NX;k++)
                        {
                            satn<<k<<"              "<<j<<"    "<<i<<"    ";
                            pres<<k<<"              "<<j<<"    "<<i<<"    ";
                            for (int l=1;l<=models;l++)
                            {
                                    satn<<Preport[l][k][j][i]<<"        ";
                                    pr_val=pr_dist(i,j,k,l);
                                    pres<<pr_val<<"        ";
                            }
                            satn<<"\n";
                            pres<<"\n";
                        }
                    }

                    }



                    for (j=1;j<=models;j++)
                    {
                        chk<<"Model "<<j<<"\n";
                        flag4=0, flag5=0; flag6=0; flag7=0; flag8=0;
                        monitor_sat<<"Model "<<j<<"            ";
                        for (i=1;i<=Tsteps;i++)
                        {
                            if (mon_sat[i][j]>=5 && flag6==0)
                                    {
                                    flag6=1;
                                    monitor_sat<<i<<"\n";
                                    }
                            if (mon_sat[i][j]>=1 && flag7==0)
                            {
                                    flag7=1;
                                    monitor_sat<<i<<"     ";
                            }
                            if (mon_sat[i][j]>=10 && flag8==0)
```

```
                                {
                                        flag8=1;
                                        monitor_sat<<i<<"     ";
                                }
                                if (mon_pr[i][j]/pr_max[j]>0.4 && flag4==0)
                                {
                                        monitor_pr1<<i<<"    ";
                                        flag4=1;
                                }

                                if (mon_pr[i][j]/pr_max[j]>0.8 && flag5==0)
                                {
                                        monitor_pr2<<i<<"    ";
                                        flag5=1;
                                }
                                chk<<i<<" "<<mon_sat[i][j]<<"\n";
                        }
                        monitor_pr1<<"\n";
                        monitor_pr2<<"\n";
                }


   monitor_pr1.close();
        monitor_pr2.close();
        monitor_sat.close();
        satn.close();
        pres.close();
        monitor_sat.close();
        chk.close();

        stop=time(0);
        cout<<"\nTime taken:  "<<stop-start<<" secs\n";
        delete []rep;
        Beep (2750,500);


} // ********************** END OF MAIN **********************

void ReadParameter(void)
{
        int i,j,k;
        char Pfile[80];
        cout<<"Name of parameter file:  ";
        cin.getline(Parfile,80);
        ifstream data(Parfile);
        data>>NX;
        data>>NY;
        data>>NZ;
        //data>>Nsource;
        cout<<"Size:  "<<NX<<"    "<<NY<<"                "<<NZ<<"\n";
                data>>sx;
                data>>sy;
                data>>sz;
                cout<<"\nSource:  "<<sx<<"       "<<sy<<"  "<<sz<<"\n";

                data>>mx;
                data>>my;
                data>>mz;
                cout<<"\nMonitor:  "<<mx<<"    "<<my<<" "<<mz<<"\n";

        cout<<"\nPermeability file name:  ";
        data.getline(Pfile,25);
        cout<<Pfile<<"\n";
        ifstream permblty(Pfile);
        permblty>>models;
```

```
	for (i=1;i<=NZ;i++)
	{
		for ( j=1;j<=NY;j++)
		{
			for ( k=1;k<=NX;k++)
			{
				for (int l=1;l<=models;l++)
				{
				permblty>>perm[l][k][j][i];
				if (Kmax[l]<perm[l][k][j][i])
					Kmax[l]=perm[l][k][j][i];
				}
			}
		}
	}

	cout<<"\n\nData Reading completed.";
	cout<<"\nPressure file:  ";
	data.getline(Prfile,125);
	cout<<Prfile<<"\n";
	cout<<"\nSaturation file:  ";
	data.getline(Sfile,125);
	cout<<Sfile<<"\n";
	cout<<"\nPressure Monitoring files:\n";
	cout<<"Low:  ";
	data.getline(cs_pr1,125);
	cout<<cs_pr1<<"\n";
	data.getline(cs_pr2,125);
	cout<<"High:  "<<cs_pr2<<"\n";
	cout<<"\nFile for satn data:  ";
	data.getline(sat_time,125);
	cout<<sat_time<<"\n\n";
	permblty.close();
	data.close();

}

double pr_dist(int z, int y, int x, int mod)
{
	double press=0,d;
	int c=0;
	const double a=1,b=10,l=1;
	for (int i=1;i<=NX;i++)
	{
		for (int j=1;j<=NY;j++)
		{
			for (int k=1;k<=NZ;k++)
			{
				d=pow((x-i)*(x-i)+(y-j)*(y-j)+(z-k)*(z-k),0.5)/l;
				if (d!=0 && Preport[mod][i][j][k]!=0)
				{
					press+=a*Preport[mod][i][j][k]/MAX*(exp(b/d));
					c++;
				}
			}
		}
	}
	press=press/c;
	return(press);

}
```

Appendix B: SGeMS Plugin

B.1. BUILDING THE PLUGIN

We will now detail the development of the library file for the SGeMS plugin, which is integrated with the larger SGeMS code. The entire code can be retrieved from the GitHub repository at https://github.com/ar2tech/ar2tech-SGeMS-public. The codes that were developed for the plugin will need to be compiled with the original SGeMS code in order to use the plugin.

B.1.1. The *initialize* function

```
80  bool ModelSelection::initialize( const Parameters_handler* parameters,
81              Error_messages_handler* errors ) {
82
83      #pragma region Get parameters
84      std::string perm_grid_name = parameters->value( "perm_grid.value" );
85      errors->report( perm_grid_name.empty(), "perm_grid", "No grid selected" );
86      if( perm_grid_name.empty() ) return false;
87
```

Figure B.1. Example of the *initialize* function

An example of part of the initialize function is given in Figure B.1. The function is passed two arguments: list of all parameters input by the user, and an object to handle errors. The parameter names read by the initialize function depend on their particular

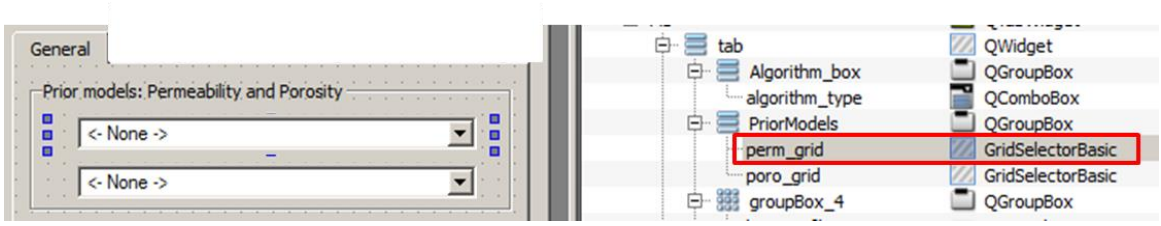


Figure B.2. Qt Designer layout, showing the name assigned to the permeability widget

description in the .ui file. For example, as seen in Figure B.2, the widget for selecting the

permeability models is called ‘perm_grid’, so variable *perm_grid_name* is assigned the value in that specific box using the statement:

```
std::string perm_grid_name = parameters->value( "perm_grid.value" );
```

A similar approach will be taken for populating all variables in the *initialize* function.

B.1.2. The *execute* function

As mentioned in chapter 6, the *execute* function controls the workflow for implementing the model selection algorithm, invoking the different modules needed for the process and enabling proper data transfer from one part of the workflow to the next. The modules are defined as functions defined in a separate script file called *RW.cpp.* In this section, we detail the development of the various modules within the algorithm.

Particle-tracking proxy

The particle tracking proxy is the same proxy as was implemented in chapter 5. Thus it needs variables like model size, grid dimensions, fluid and rock properties, and depth and initial pressure values of the reservoir. The input tab for these properties has been described in section 6.2 and Figure 6.4.

In order to run the proxy over all models, the code needs the ability to cycle through the list of permeabilities and porosities that were specified in the input tab. For this purpose, it needs a count for the total number of models, which is achieved using the following snippet of code within the *initialize* function:

```
  std::string perm_grid_name = parameters->value("
perm_grid.value" );

  perm_grid_ = dynamic_cast<RGrid*>( Root::instance()->
interface (gridModels_manager + "/" + perm_grid_name)
.raw_ptr() );

  perm_name_list = perm_grid_->property_list();

  num_of_en = perm_name_list.size();
```

The variable *perm_grid_* is a pointer to the permeability object and represents the grid the algorithm will be applied to, defined in the class *ModelSelection*. This pointer is used to find the size of the object, and hence the number of models in the object as shown above. It is also used to iterate through the list of models in the 'permeability' object, as shown below:

```
list<string>::iterator perm_name_iter = perm_name_list.begin();
Grid_continuous_property *tmp_perm_prop;
for (int model=0; model<num_of_en; model++)
{
  tmp_perm_prop = perm_grid_->select_property(*perm_name_iter);
  // operation with tmp_perm_prop //
  perm_name_iter++;
}
```

In the above example, a list iterator is being created for the list of names in the permeability object called *perm_name_iter*. This iterator is then used to populate a pointer called *tmp_perm_prop,* which points to the variable listed as a specific name for the permeability object. All operations with a particular model of permeability will be done on the pointer *tmp_perm_prop*. Then, the program can move on to the next model in the permeability object by incrementing the list iterator for the permeability names, *perm_name_iter*.

The next step is to know how to use the pointer *tmp_perm_prop* to extract values of the property at specific grid locations for the model. For this purpose, we would use a code as follows:

```
NX = perm_grid_->nx();
NY = perm_grid_->ny();
NZ = perm_grid_->nz();

int counter = 0;

for (int i=0; i<NZ; i++)
{
  for (int j=0; j<NY; j++)
  {
    for (int k=0; k<NX; k++)
    {
      perm[k][j][i] = tmp_perm_prop->get_value(counter);
      counter++;
    }
  }
}
```

In the above example, we extract the dimensions of the grid from the *perm_grid_* pointer itself, and then use that to iterate through *tmp_perm_prop* to get individual values of the variable using the function *get_value(int)*.

The last piece of information we need is to know how to write values calculated by our program back into the GUI, under a newly created object. For this purpose, we can use a snippet of code as follows:

```
1  string opt_grid_name = "Random_walk_results";
2  opt_grid_ = dynamic_cast<RGrid*>( Root::instance()->interface(
   gridModels_manager + "/" + opt_grid_name).raw_ptr() );
3
4  Grid_continuous_property *tmp_rw_prop;
5  tmp_rw_prop = opt_grid_->add_property("RW_result_Time_" +
   static_cast<ostringstream*>( &(ostringstream() << sum_del_T) )->
   str() + "_" + tmp_perm_prop->name());
6
7  int counter = 0;
8  for (int i=0; i<NZ; i++)
9  {
10   for (int j=0; j<NY; j++)
11   {
12     for (int k=0; k<NX; k++)
13     {
14       tmp_rw_prop->set_value(saturations[k][j][i], counter);
15       counter++;
16     }
17   }
18 }
```

In this code, we first create a new object called “Random_walk_results” in lines 1-2. Then, we define a *Grid_contiuous_property* pointer called *tmp_rw_prop* to write values to the object in lines 4-5. Finally, we write the values of the output (in this case called saturations) using the function *set_value()* on line 14.

The information presented above is adequate to enable us to get values from the user interface and execute the proxy.

PCA and Model Grouping

The results of the proxy noted at user-specified locations are used for principal components analysis and subsequent clustering, as detailed in chapter 3. Here, we show parts of the code where that workflow is implemented.

The computation of the principal component axes and projection of the models onto those axes are accomplished using a function *projections()*:

```
projected = Allocate2D<double>(num_of_en,3);
projections(original, projected, num_of_en, parms);
```

This function takes as input the statistics derived from the proxy (*original*), a pre-allocated array to store the projections of the models onto principal component space (*projected*), the total number of models (*num_of_en*) and the total number of parameters for each model (*parms*). The code can be seen in later in this Appendix.

The projections of the models are then divided into clusters and the model closest to each cluster centroid is found.

```
1 clusterid = new int[num_of_en];
2 num_clus = cluster(num_of_en, projected, clusterid);
3 centroids = Allocate2D<double>(num_clus,3);
4 find_centroids(num_of_en, num_clus, centroids, clusterid,
  projected);
5 rep_models = new int[num_clus];
6 find_best_models(num_of_en,num_clus, clusterid, centroids,
  projected, rep_models);
```

The function *cluster()* is used for this purpose: it finds the ideal number of clusters for the current case and saves that value in *num_clus* and it assigns cluster identifier numbers to each model in the array *clustered*. Finally, the function *find_centroids()* finds the cluster centroids for the allocated clusters and saves them in the array *centroids* (line 4), which is then used in the function *find_best_models()* to find the model in each cluster closest to the centroid and save its identifying number in the variable *rep_models* (line 6). These functions are also given in full in the next section of this Appendix.

Numerical simulation and Bayesian updating

The final step in a model selection iteration is to evaluate the representative model for each cluster using a full physics flow simulation, comparing the simulation results to observed data and computing posterior probabilities using Bayes' rule. For this purpose, our code needs to be able to do the following: run simulations for each representative model, read the simulated result for the simulation run, and once all models have been run, compute the posterior probabilities.

In order to run the numerical simulation, in our current implementation we have used CMG-GEM; however, it is possible to use any other simulator by making simple changes to the code. The variables that need changing for each run are the permeability and porosity distributions, and this is made possible by incorporating them in the simulation data deck as external files using the INCLUDE command in CMG. Thus, the code was required to write out permeability and porosity files before initiating a particular run. The simulator was run using the command prompt version of CMG, using the following code:

```
1  FILE *fp_perm, *fp_por;
2  fp_perm = fopen(fname_perm, "w");
3  fp_por = fopen(fname_por, "w");
4  for (int j=0; j<NX*NY*NZ; j++)
5  {
6    fprintf(fp_perm,"%.2f\n", tmp_perm_prop->get_value(j));
7    fprintf(fp_por,"%.2f\n",tmp_poro_prop->get_value(j));
8  }
9  fclose(fp_perm); fclose(fp_por);
10
11 // Run CMG
12
13 string simulation_command = "\"" + simulator_location;
14 simulation_command = simulation_command + "\\gm201110.exe\" -parasol 6 -f " +
   simulation_folder + "\\" + sim_file_name;
15 simulation_command = simulation_command + " -wd " + simulation_folder;
16 system(simulation_command.c_str());
17
18 string results_command = "\"" + results_location;
19 results_command = results_command + "\\report.exe\" /f ";
20 results_command = results_command + simulation_folder + "\\Report.rwd /o ";
21 results_command = results_command + simulation_folder + "\\Report.rwo";
22 char aa[10];
23 system(results_command.c_str());
```

In this code snippet, the permeability and porosity files are being written to the same folder where the simulation data deck lies (the working folder) using lines 1 – 9. The simulator is then run using line 16, which invokes the windows command processor to execute the following simulation command:

```
<sim_loc>\gm201110.exe -parasol 6 -f <working_folder\sim_file> -wd <working_folder>
```

Here, `sim_loc` refers to the location of the simulator on the particular computer (specified previously by the user in the GUI), and `sim_file` refers to the simulation data deck, which should exist within the working folder and is also specified by the user in the GUI.

Once the particular representative model has been run, the REPORT.exe program in CMG extracts the necessary simulated data for the conditioning wells from the simulation output, using line 23. This invokes the command processor to execute the following:

```
<sim_loc>\report.exe -f <working_folder>\Report.rwd -wd Report.rwo
```

Here, the REPORT program uses the file *Report.rwd* in the working folder to evaluate the output of the simulation and save the results in *Report.rwo*. The structure of the *Report.rwd* file is as follows:

```
*FILE 'BASE_CASE.irf'
*TIMES-FOR 100 200 300 400 500
*TABLE-FOR
  *COLUMN-FOR *PARAMETERS 'Well Bottom-hole Pressure' *WELLS 'INJECTOR-3'
*TABLE-END
```

Here, the program reads the output from the BASE_CASE file, the particular output being the bottom-hole pressure for the well called INJECTOR-3 at times 100, 200, 300, 400 and 500 days. The results are read into an array called `simulation_results` using a `read_report` function.

```
read_report(i, simulated_results, nrows_hist,simulation_folder + "\\Report.rwo");
```

Once all the simulations have been run, the final step is the calculation of the posterior probabilities and finding the cluster with the highest updated probability. This is achieved using the following code:

```
1  vector<double> class_probabilities(num_clus);
2  class_probabilities = calc_probability(simulated_results, history_data,
   nrows_hist, num_clus);
3
4  vector < vector<double> > link_probs_clusterid (num_clus);
5  vector<double>::iterator it;
6  int count_clus = 0;
7  for (it = class_probabilities.begin(); it!= class_probabilities.end(); it++)
8  {
9    link_probs_clusterid[count_clus].push_back(count_clus);
10   link_probs_clusterid[count_clus].push_back(*it);
11   count_clus++;
12 }
13 sort(link_probs_clusterid.begin(), link_probs_clusterid.end(), [](const std::
   vector< double >& a, const std::vector< double >& b){ return a[1] > b[1]; } );
14
15 int best_cluster_id = link_probs_clusterid[0][0];
```

Here, the function `calc_probability` (details later in this Appendix) calculates the updated probability for all the clusters, using the `simulated_results` and the `history` data. Then, lines 4 – 13 are used to create a vector of vectors to store the values

of the updated probabilities and sort them in the order of decreasing probability. Finally, in line 15, the index of the cluster with the highest probability is stored as the variable `best_cluster_id`.

Saving the results

Once the model selection process is terminated, either due to equiprobable final clusters or a limit on the number of iterations, the best-fit models need to be written out to the GUI. For this purpose, the user creates an empty object within the GUI before the start of the program, and this object is then populated using the models in the best-fit cluster, using the following code:

```
1  for (int i=0; i<num_of_en; i++)
2  {
3    if (clusterid[i]==best_cluster_id)
4    {
5      string cmd("CopyProperty");
6      string cmd_parameters;
7      Error_messages_handler copy_error_msg;
8      std::list<string>::iterator perm_name_iter = perm_name_list.begin();
9
10     for (int j=0; j<i; j++)
11     {
12       perm_name_iter++;
13     }
14
15     cmd_parameters = perm_grid_->name() + "::" + *perm_name_iter + "::" +
RW_grid_->name() + "::" + *perm_name_iter + "::0::0";
16     bool copy_ok = proj->execute(cmd, cmd_parameters, &copy_error_msg);
17
18   }
19 }
```

The process uses internal commands available in SGeMS for copying parameters from one object into another. The code on line 16 achieves this by running the following SGeMS script:

```
CopyProperty  A::A_1::B::B_1::0::0
```

Where ‘A’ is the object from which the property ‘A_1’ is being copied into the object ‘B’, and the new property is being called ‘B_1’.

B.2. CODES FOR PLUGIN

In this section, we layout the codes that are needed for compiling SGeMS with the model selection plugin.

ModelSelection.cpp : Primary code for defining the *initialize* and *execute* functions

```
/* ----------------------------------------------------------------------------
** Copyright (c) 2012 Advanced Resources and Risk Technology, LLC
** All rights reserved.
**
** This file is part of Advanced Resources and Risk Technology, LLC (AR2TECH)
** version of the open source software sgems.  It is a derivative work by
** AR2TECH (THE LICENSOR) based on the x-free license granted in the original
** version of the software (see notice below) and now sublicensed such that it
** cannot be distributed or modified without the explicit and written permission
** of AR2TECH.
**
** Only AR2TECH can modify, alter or revoke the licensing terms for this
** file/software.
**
** This file cannot be modified or distributed without the explicit and written
** consent of AR2TECH.
**
** Contact Dr. Alex Boucher (aboucher@ar2tech.com) for any questions regarding
** the licensing of this file/software
**
** The open-source version of sgems can be downloaded at
** sourceforge.net/projects/sgems.
** ----------------------------------------------------------------------------*/



/*****************************************************************************
** Author: Nicolas Remy
** Copyright (C) 2002-2004 The Board of Trustees of the Leland Stanford Junior
**   University
** All rights reserved.
**
** This file is part of the "geostat" module of the Geostatistical Earth
** Modeling Software (GEMS)
**
** This file may be distributed and/or modified under the terms of the
** license defined by the Stanford Center for Reservoir Forecasting and
** appearing in the file LICENSE.XFREE included in the packaging of this file.
**
** This file may be distributed and/or modified under the terms of the
** GNU General Public License version 2 as published by the Free Software
** Foundation and appearing in the file LICENSE.GPL included in the
** packaging of this file.
**
** This file is provided AS IS with NO WARRANTY OF ANY KIND, INCLUDING THE
** WARRANTY OF DESIGN, MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
**
** See http://www.gnu.org/copyleft/gpl.html for GPL licensing information.
**
** Contact the Stanford Center for Reservoir Forecasting, Stanford University
** if any conditions of this licensing are not clear to you.
**
*****************************************************************************/
```

```
#include <grid/rgrid.h>
#include <utils/gstl_plugins.h>
#include <utils/string_manipulation.h>
#include <grid/grid_property.h>
#include <utils/gstl_messages.h>
#include <utils/manager.h>
#include "ModelSelection.h"
#include "matrix_def.h"

int ModelSelection::execute( GsTL_project* proj )  {
	switch(selected_algorithm)
	{
//		case SCA:
//			run_SCA(proj);
//			break;
		case RANDOM_WALKER:
			run_RW(proj);
			break;
		case MODEL_EXPANSION:
			break;
	}
	return 0;
}

bool ModelSelection::initialize( const Parameters_handler* parameters,
			Error_messages_handler* errors ) {

	#pragma region Get parameters
	std::string perm_grid_name = parameters->value( "perm_grid.value" );
	errors->report( perm_grid_name.empty(), "perm_grid", "No grid selected" );
	if( perm_grid_name.empty() ) return false;

	std::string poro_grid_name = parameters->value( "poro_grid.value" );
	errors->report( poro_grid_name.empty(), "poro_grid", "No grid selected" );
	if( poro_grid_name.empty() ) return false;


	std::string algorithm_name = parameters->value( "algorithm_type.value" );
	errors->report( algorithm_name.empty(), "algorithm_type", "No grid selected" );
	if( algorithm_name.empty() ) return false;

	perm_grid_ = dynamic_cast<RGrid*>( Root::instance()->interface(
						gridModels_manager + "/" + perm_grid_name).raw_ptr() );
	poro_grid_ = dynamic_cast<RGrid*>( Root::instance()->interface(
						gridModels_manager + "/" + poro_grid_name).raw_ptr() );

	perm_name_list = perm_grid_->property_list();
	poro_name_list = poro_grid_->property_list();

	num_of_en = perm_name_list.size();

	if(String_Op::contains( algorithm_name, "Scaled Connectivity Analysis", false ))
		selected_algorithm = SCA;
	else if(String_Op::contains( algorithm_name, "Random Walker", false ))
		selected_algorithm = RANDOM_WALKER;
	else if(String_Op::contains( algorithm_name, "Model Expansion", false ))
		selected_algorithm = MODEL_EXPANSION;
	else
		return false;

	//simulation file
	sim_file_name = parameters->value( "sim_file_name.value" );
	errors->report( sim_file_name.empty(), "sim_file_name", "Missing simulation file name" );
	if( sim_file_name.empty() ) return false;
```

```
	//simulator location
	simulator_location = parameters->value( "simulator_location.value" );
	errors->report( simulator_location.empty(), "simulator_location", "Missing simulator location" );
	if( simulator_location.empty() ) return false;

	//CMG RESULTS location
	results_location = parameters->value( "results_location.value" );
	errors->report( results_location.empty(), "results_location", "Missing results location" );
	if( results_location.empty() ) return false;

	//history file
	std::string history_file_name = parameters->value( "history_file_name.value" );
	errors->report( history_file_name.empty(), "history_file_name", "Missing history file name" );
	if( history_file_name.empty() ) return false;

	fstream fp_hist;
	fp_hist.open(history_file_name.c_str(), ios::in);
	fp_hist>>nrows_hist>>ncols_hist;
	history_data =  Allocate2D<double>(nrows_hist, ncols_hist);
	for (int i=0; i<nrows_hist; i++)
	{
		for (int j=0; j<ncols_hist; j++)
			fp_hist>>history_data[i][j];
	}
	fp_hist.close();

	// working folder for simulations
	simulation_folder = parameters->value( "simulation_folder.value" );
	errors->report( simulation_folder.empty(), "simulation_folder", "No folder for simulation selected" );
	if( simulation_folder.empty() ) return false;

	switch(selected_algorithm)
	{
//		case SCA:
//			if(!Initialize_SCA(parameters, errors))
//				return false;
		case RANDOM_WALKER:
			if(!Initialize_RW(parameters, errors))
				return false;
		case MODEL_EXPANSION:
			if(!Initialize_ME(parameters, errors))
				return false;
	}
	#pragma endregion

	if( !errors->empty() ) {
		return false;
	}

	this->extract_parameters(parameters);
	return true;
}
```

RW.cpp : Classes and associated /lfunctions / variables are decleared in ***RW.h***

```
// Header files <>
#include "matrix_def.h"
#include <time.h>
#include <grid/point_set.h>
#include "RW.h"
#include "ModelSelection.h"
#include "cluster_to_simulation.h"

template <typename T>
double percentile(Grid_continuous_property *tmp_perm_prop, int NX, int NY, int NZ, double
target_p, T cutoff)
{
    // finds the target_p-th percentile among all data in the 3D array dArray which is above the
cutoff //
    vector<double> a;
    int counter = 0;
    for (int i=0; i<NX; i++)
    {
        for (int j=0; j<NY; j++)
        {
        for (int k=0; k<NZ; k++)
        {
            if (tmp_perm_prop->get_value(counter)>0)
            {
            a.push_back(tmp_perm_prop->get_value(counter)*9.869e-16);
            }
            counter++;
        }
        }
    }
    fstream fp;
    fp.open("in_percentile.txt",ios::out);
    fp<<"In percentile\n";
    sort(a.begin(),a.end());
    int length = a.size();
    fp<<length<<endl;
    int b = length*(target_p/100.0);
    fp<<b<<" "<<a[b-1]<<endl;
    fp.close();

    double return_val;
    if (b>0)
        return_val = a[b-1];
    else
        return_val = a[0];
    a.clear(); a.shrink_to_fit();

    return return_val;
}


bool ModelSelection::Initialize_RW( const Parameters_handler* parameters, Error_messages_handler*
errors )
{
    fstream fp;
    fp.open("check_read.txt",ios::out | ios::app);

    num_injectors = 1;
    inj_locs = Allocate2D<int>(num_injectors,3);
    inj_locs[0][0] = String_Op::to_number<int>( parameters->value( "NX.value" ) );
    errors->report( inj_locs[0][0] <= 0, "NX", "Invalid injector location" );
```

```
    if( inj_locs[0][0] <= 0 ) return false;

    inj_locs[0][1] = String_Op::to_number<int>( parameters->value( "NY.value" ) );
    errors->report( inj_locs[0][1] <= 0, "NY", "Invalid injector location" );
    if( inj_locs[0][1] <= 0 ) return false;

    inj_locs[0][2] = String_Op::to_number<int>( parameters->value( "NZ.value" ) );
    errors->report( inj_locs[0][2] <= 0, "NZ", "Invalid injector location" );
    if( inj_locs[0][2] <= 0 ) return false;


    fp<<"Injectors: "<<inj_locs[0][0]<<" "<<inj_locs[0][1]<<" "<<inj_locs[0][2]<<endl;
    fp.close();

    std::string depth_grid_name = parameters->value( "depth_grid.value" );
    errors->report( depth_grid_name.empty(), "depth_grid", "No grid selected" );
    if( depth_grid_name.empty() ) return false;

    depth_grid_ = dynamic_cast<RGrid*>( Root::instance()->interface(gridModels_manager + "/" +
depth_grid_name).raw_ptr() );

    fp.open("check_read.txt",ios::app);
    fp<<"Depth grid name: "<<depth_grid_name<<endl;
    fp.close();

    std::string ini_pr_grid_name = parameters->value( "ini_pr_grid.value" );
    errors->report( ini_pr_grid_name.empty(), "ini_pr_grid", "No grid selected" );
    if( ini_pr_grid_name.empty() ) return false;

    fp.open("check_read.txt",ios::app);
    fp<<"Pressure grid name: "<<ini_pr_grid_name<<endl;
    fp.close();

    ini_pr_ = dynamic_cast<RGrid*>( Root::instance()->interface(gridModels_manager + "/" +
ini_pr_grid_name).raw_ptr() );


    depth_grid_list = depth_grid_->property_list();
    ini_pr_list = ini_pr_->property_list();

    rel_perm_table = Allocate2D<double>(1000,3);
    calc_rel_perms(rel_perm_table,parameters,errors);

    read_fluid_properties(parameters,errors, brine_den_rw, co2_den_rw, brine_visc_rw, co2_visc_rw,
ct_rw);

    fw_table = Allocate2D<double>(1000,2);
    calc_fw_table( rel_perm_table, fw_table, co2_visc_rw, brine_visc_rw );

    read_run_time_data(parameters,errors, total_days, inj_rate, delta_T, reporting_interval,
particles_per_time);

    std::string unit_system = parameters->value( "unit_system.value" );
    errors->report( unit_system.empty(), "unit_system", "No UNIT SYSTEM selected" );
    if( unit_system.empty() ) return false;

    if(String_Op::contains( unit_system, "FIELD", false ))
        selected_units = FIELD;
    else if(String_Op::contains( unit_system, "SI", false ))
        selected_units = SI;
    else
        return false;

    std::string RW_grid_name = parameters->value( "RW_grid.value" );
```

```
    errors->report( RW_grid_name.empty(), "RW_grid", "No grid selected" );
    if( RW_grid_name.empty() ) return false;

    RW_grid_ = dynamic_cast<RGrid*>( Root::instance()->interface(gridModels_manager + "/" +
RW_grid_name).raw_ptr() );

/*  simulation_folder = parameters->value( "simulation_folder.value" );
    errors->report( simulation_folder.empty(), "simulation_folder", "No folder for simulation
selected" );
    if( simulation_folder.empty() ) return false;
*/
    // read in the monitoring locations
    std::string Monitor_file_name = parameters->value( "monitoring_locs_2.value" );
    errors->report( Monitor_file_name.empty(), "monitoring_locs_2", "Invalid monitoring locations"
);
    if( Monitor_file_name.empty() ) return false;

    Point_set* grid_mon_ = dynamic_cast<Point_set*>(  Root::instance()-
>interface(gridModels_manager + "/" + Monitor_file_name).raw_ptr() );

    const std::vector<Point_set::location_type>& locs = grid_mon_->point_locations();
    std::vector<Point_set::location_type>::const_iterator vec_it = locs.begin();
    num_monitors = locs.size();
    monitor_locs = Allocate2D<int>(num_monitors,3);
    fp<<"Number of monitors: "<<num_monitors<<endl;
    for(int i=0; i<num_monitors; i++)
    {
        monitor_locs[i][0] = (int) vec_it->x();
        monitor_locs[i][1] = (int) vec_it->y();
        monitor_locs[i][2] = (int) vec_it->z();
        vec_it++;
    }
/*
    //history file
    std::string history_file_name = parameters->value( "history_file_name.value" );
    errors->report( history_file_name.empty(), "history_file_name", "Missing history file name" );
    if( history_file_name.empty() ) return false;

    fstream fp_hist;
    fp_hist.open(history_file_name.c_str(), ios::in);
    fp_hist>>nrows_hist>>ncols_hist;
    history_data =  Allocate2D<double>(nrows_hist, ncols_hist);
    for (int i=0; i<nrows_hist; i++)
    {
        for (int j=0; j<ncols_hist; j++)
        fp_hist>>history_data[i][j];
    }
    fp_hist.close();

    //simulation file
    sim_file_name = parameters->value( "sim_file_name.value" );
    errors->report( sim_file_name.empty(), "sim_file_name", "Missing simulation file name" );
    if( sim_file_name.empty() ) return false;

    //simulator location
    simulator_location = parameters->value( "simulator_location.value" );
    errors->report( simulator_location.empty(), "simulator_location", "Missing simulator location"
);
    if( simulator_location.empty() ) return false;

    //CMG RESULTS location
    results_location = parameters->value( "results_location.value" );
    errors->report( results_location.empty(), "results_location", "Missing results location" );
    if( results_location.empty() ) return false;
```

```
*/
    fp.open("check_read.txt",ios::app);
    fp<<"Done reading all locations etc.\n";
    fp.close();

    return true;
}

int ModelSelection::run_RW(GsTL_project* proj)
{
    model_data static_data;

    NX = perm_grid_->nx();
    NY = perm_grid_->ny();
    NZ = perm_grid_->nz();

    stats = Allocate2D<double>(num_of_en,(int) num_monitors*total_days/reporting_interval);


    dx = perm_grid_->geometry()->cell_dims().x();
    dy = perm_grid_->geometry()->cell_dims().y();
    dz = perm_grid_->geometry()->cell_dims().z();

    list<string>::iterator perm_name_iter = perm_name_list.begin();
    list<string>::iterator poro_name_iter = poro_name_list.begin();
    list<string>::iterator depth_name_iter = depth_grid_list.begin();
    list<string>::iterator ini_pr_name_iter = ini_pr_list.begin();

    Grid_continuous_property *tmp_perm_prop;
    Grid_continuous_property *tmp_poro_prop;
    Grid_continuous_property *tmp_depth_prop;
    Grid_continuous_property *tmp_ini_pr_prop;

    #pragma region Allocate data to struct
    static_data.NX = NX;
    static_data.NY = NY;
    static_data.NZ = NZ;
    static_data.dx = dx;
    static_data.dy = dy;
    static_data.dz = dz;
    static_data.simulation_folder = simulation_folder;

    static_data.brine_den_rw  = brine_den_rw;
    static_data.brine_visc_rw = brine_visc_rw;
    static_data.co2_den_rw = co2_den_rw;
    static_data.co2_visc_rw = co2_visc_rw;
    static_data.ct_rw = ct_rw;

    static_data.num_injectors = num_injectors;
    static_data.num_monitors = num_monitors;
    static_data.nrows_hist = nrows_hist;
    static_data.ncols_hist = ncols_hist;
    static_data.inj_locs = inj_locs;
    static_data.monitor_locs = monitor_locs;
    static_data.particles_per_time = particles_per_time;

    static_data.fw_table = fw_table;
    static_data.history_data = history_data;

    static_data.total_days = total_days;
    static_data.inj_rate = inj_rate;
    static_data.delta_T = delta_T;
    static_data.reporting_interval = reporting_interval;
    switch(selected_units)
```

```
    {
        case SI:
            static_data.selected_units = 1;
            break;
        case FIELD:
            static_data.selected_units = 2;
            break;
    }

    #pragma endregion

    string cmd("NewCartesianGrid");
    string cmd_parameters;
    Error_messages_handler copy_error_msg;
    cmd_parameters = "Random_walk_results::"  + static_cast<ostringstream*>( &(ostringstream() <<
NX) )->str() + "::"
        + static_cast<ostringstream*>( &(ostringstream() << NY) )->str() + "::"
        + static_cast<ostringstream*>( &(ostringstream() << NZ) )->str() + "::"
        + static_cast<ostringstream*>( &(ostringstream() << dx) )->str() + "::"
        + static_cast<ostringstream*>( &(ostringstream() << dy) )->str() + "::"
        + static_cast<ostringstream*>( &(ostringstream() << dz) )->str() + "::0::0::0::0.00";

    bool copy_ok = proj->execute(cmd, cmd_parameters, &copy_error_msg);

    // Create grid to write current RW results //
    string opt_grid_name = "Random_walk_results";
    opt_grid_ = dynamic_cast<RGrid*>( Root::instance()->interface(gridModels_manager + "/" +
opt_grid_name).raw_ptr() );


    SmartPtr<Progress_notifier> progress_notifier = utils::create_notifier( "Running Random
Walker", num_of_en, 1);
    for (int model=0; model<num_of_en; model++)
    {
        tmp_perm_prop = perm_grid_->select_property(*perm_name_iter);
        tmp_poro_prop = poro_grid_->select_property(*poro_name_iter);
        tmp_depth_prop = depth_grid_->select_property(*depth_name_iter);
        tmp_ini_pr_prop = ini_pr_->select_property(*ini_pr_name_iter);
        progress_notifier->message() << "Working on realization " << model+1 << " of "<< num_of_en
<< gstlIO::end;
        if( !progress_notifier->notify() ) return 1;
        if (restart_flag==0) then run random walker else dont. Similarly for creating object
Random_walk_results above.
        random_walker(model, tmp_perm_prop, tmp_poro_prop, tmp_depth_prop, tmp_ini_pr_prop,
static_data, opt_grid_, stats);
        if ((model+1)%60==0)
        {
            cmd = "SaveProject";
            cmd_parameters = simulation_folder + "//temp_results";
            bool copy_ok = proj->execute(cmd, cmd_parameters, &copy_error_msg);
        }

        perm_name_iter++; poro_name_iter++;
    }

    fstream fp2;
    char stats_file[1000];
    strcpy(stats_file, simulation_folder.c_str());
    strcat(stats_file, "\\stats_clean.txt");
    fp2.open(stats_file,ios::out);
    for (int i=0; i<total_days/reporting_interval*num_monitors; i++)
    {
        for (int j=0; j<num_of_en; j++)
            fp2<<stats[j][i]<<" ";
```

```
        fp2<<endl;
    }
    fp2.close();


    #pragma region clustering to simulation

    char resultRW[100], history[100], Pfile[100];
    int num_grids, num_models, parms, Nx, Ny, Nz, num_iter;
    int *clusterid, *rep_models;
    double **original, **cov, **perms, **observed, **zmean, **evecs, **projected, **average,
**rep, errs[4], **centroids, **simulated_results;
    int check=0, num_clus, most_probable,run_num,i,j;
    double prob_value, dif;
    time_t start,end;
//  Reading the stats file. This contains statistics for each model at the monitoring locations

    fstream fp_check;
    fp_check.open(stats_file,ios::in);
    int count_parms = 0;
    double temp_fp_check;
    while (!fp_check.eof())
    {
        for (int j=0; j<num_of_en; j++)
            fp_check>>temp_fp_check;
        count_parms++;
    }
    fp_check.close();
    parms = count_parms;
    original = Allocate2D<double>(num_of_en,parms);

    fstream fp;
    fp.open(stats_file,ios::in);
    for (int i=0; i<parms; i++)
    {
        for (int j=0; j<num_of_en; j++)
            fp>>original[j][i];
    }
    fp.close();

//  Projection and all
    projected = Allocate2D<double>(num_of_en,3);
    projections(original, projected, num_of_en, parms);

//  Clustering
    clusterid = new int[num_of_en];
    int get_clus;
    cin>>get_clus;
    num_clus = cluster(num_of_en, projected, clusterid);
    centroids = Allocate2D<double>(num_clus,3);
    find_centroids(num_of_en, num_clus, centroids, clusterid, projected);
    rep_models = new int[num_clus];
    find_best_models(num_of_en,num_clus, clusterid, centroids, projected, rep_models);

// Write clustering data to output files
    fstream fp_cluster_data;
    char cluster_op[1000];
    strcpy(cluster_op, simulation_folder.c_str());
    strcat(cluster_op, "\\cluster_data.txt");
    fp_cluster_data.open(cluster_op, ios::out);
    fp_cluster_data<<"Projections: \n";
    for (i=0; i<num_of_en; i++)
    {
        for (int j=0; j<3; j++)
```

```
            fp_cluster_data<<projected[i][j]<<"\t";
        fp_cluster_data<<endl;
    }

    fp_cluster_data<<"\nCluster IDs: \n";
    for (i=0; i<num_of_en; i++)
        fp_cluster_data<<clusterid[i]<<endl;

    fp_cluster_data<<"\nCentroids: \n";
    for (i=0; i<num_clus; i++)
    {
        for (int j=0; j<3; j++)
            fp_cluster_data<<centroids[i][j]<<"\t";
        fp_cluster_data<<endl;
    }
    fp_cluster_data.close();



    simulated_results = Allocate2D<double>(static_data.nrows_hist, num_clus);
//  Run CMG
    int frequency = 1;
    for (int i=0; i<num_clus; i++)
    {
        std::list<string>::iterator perm_name_iter = perm_name_list.begin();
        std::list<string>::iterator poro_name_iter = poro_name_list.begin();
        Grid_continuous_property *tmp_perm_prop;
        Grid_continuous_property *tmp_poro_prop;
        for (int j=0; j<rep_models[i]; j++)
        {
            perm_name_iter++;
            poro_name_iter++;
        }
        tmp_perm_prop = perm_grid_->select_property(*perm_name_iter);
        tmp_poro_prop = poro_grid_->select_property(*poro_name_iter);

        // write to simulation folder
        char fname[1000], fname_perm[1000], fname_por[1000];
        strcpy(fname,simulation_folder.c_str());
        strcpy(fname_perm, fname);
        strcpy(fname_por, fname);
        strcat(fname_perm, "/perm.txt");
        strcat(fname_por, "/por.txt");

        FILE *fp_perm, *fp_por;
        fp_perm = fopen(fname_perm, "w");
        fp_por = fopen(fname_por, "w");
        for (int j=0; j<NX*NY*NZ; j++)
        {
            fprintf(fp_perm,"%.2f\n", tmp_perm_prop->get_value(j));
            fprintf(fp_por,"%.2f\n",tmp_poro_prop->get_value(j));
        }
        fclose(fp_perm); fclose(fp_por);

        // Run CMG

        string simulation_command = "\"" + simulator_location;
        simulation_command = simulation_command + "\\gm201110.exe\" -parasol 6 -f " +
simulation_folder + "\\" + sim_file_name;
        simulation_command = simulation_command + " -wd " + simulation_folder;
        system(simulation_command.c_str());

        string results_command = "\"" + results_location;
        results_command = results_command + "\\report.exe\" /f ";
```

```
        results_command = results_command + simulation_folder + "\\Report.rwd /o ";
        results_command = results_command + simulation_folder + "\\Report.rwo";
        char aa[10];
        system(results_command.c_str());
        cout<<results_command<<endl;
        cin>>aa;


        read_report(i, simulated_results, nrows_hist,simulation_folder + "\\Report.rwo");


    }

//  Write simulated results to file
    fstream fp_sim;
    fp_sim.open(simulation_folder + "\\simulated_results.txt", ios::out);
    for (int i=0; i<nrows_hist; i++)
    {
        for (int j=0; j<num_clus; j++)
        {
            fp_sim<<simulated_results[i][j]<<" ";
        }
        fp_sim<<endl;
    }
    fp_sim.close();

//  Find highest probability cluster
    vector<double> class_probabilities(num_clus);
    class_probabilities = calc_probability(simulated_results, history_data, nrows_hist, num_clus);

    vector < vector<double> > link_probs_clusterid (num_clus);
    vector<double>::iterator it;
    int count_clus = 0;
    for (it = class_probabilities.begin(); it!= class_probabilities.end(); it++)
    {
        link_probs_clusterid[count_clus].push_back(count_clus);
        link_probs_clusterid[count_clus].push_back(*it);
        count_clus++;
    }
    sort(link_probs_clusterid.begin(), link_probs_clusterid.end(), [](const std::vector< double >&
a, const std::vector< double >& b){ return a[1] > b[1]; } );

    int best_cluster_id = link_probs_clusterid[0][0];

//  Write results to an OBJECT
    for (int i=0; i<num_of_en; i++)
    {
        if (clusterid[i]==best_cluster_id)
        {
            string cmd("CopyProperty");
            string cmd_parameters;
            Error_messages_handler copy_error_msg;
            std::list<string>::iterator perm_name_iter = perm_name_list.begin();

            for (int j=0; j<i; j++)
            {
                perm_name_iter++;
            }

            cmd_parameters = perm_grid_->name() + "::" + *perm_name_iter + "::" + RW_grid_-
>name() + "::" + *perm_name_iter + "::0::0";
            bool copy_ok = proj->execute(cmd, cmd_parameters, &copy_error_msg);

        }
```

```
    }

    free(clusterid);
    Free2D<double>(projected);



#pragma endregion

    return 0;
}

bool read_run_time_data(const Parameters_handler* parameters, Error_messages_handler* errors,
double &total_days, double &inj_rate, double &delta_T, double &reporting_interval, int
&particles_per_time)
{
    total_days = String_Op::to_number<double>( parameters->value( "total_days.value" ) );
    errors->report( total_days <= 0, "total_days", "Invalid total run time" );
    if( total_days <= 0 ) return false;

    inj_rate = String_Op::to_number<double>( parameters->value( "inj_rate.value" ) );
    errors->report( inj_rate <= 0, "inj_rate", "Invalid injection rate" );
    if( inj_rate <= 0 ) return false;

    delta_T = String_Op::to_number<double>( parameters->value( "delta_T.value" ) );
    errors->report( delta_T <= 0, "delta_T", "Invalid update interval" );
    if( delta_T <= 0 ) return false;

    reporting_interval = String_Op::to_number<double>( parameters->value(
"reporting_interval.value" ) );
    errors->report( reporting_interval <= 0, "reporting_interval", "Invalid reporting interval" );
    if( reporting_interval <= 0 ) return false;

    particles_per_time = String_Op::to_number<double>( parameters->value(
"particles_per_time.value" ) );
    errors->report( particles_per_time <= 0, "particles_per_time", "Invalid number of particles
per day" );
    if( particles_per_time <= 0 ) return false;

    return true;

}

void calc_fw_table( double **rel_perm_table, double **fw_table, double co2_visc_rw, double
brine_visc_rw )
{
    double krg, krw, sat;
    for (int s=0; s<1000; s++)
    {
        sat = s/1000.0;
        krg = rel_perm_table[s][0];
        krw = rel_perm_table[s][1];
        fw_table[s][0] = sat;
        fw_table[s][1] = 1/(1+krw*co2_visc_rw/krg/brine_visc_rw);
    }
}

bool calc_rel_perms(double **rel_perm_table, const Parameters_handler* parameters,
Error_messages_handler* errors)
{
    double end_pts[2], expn[2], Sr[2];
    double sat, kr;
    end_pts[0] = String_Op::to_number<double>( parameters->value( "krg0.value" ) );
    errors->report( end_pts[0] <= 0, "krg0", "Invalid end point rel-perm for gas" );
```

```
    if( end_pts[0] <= 0 ) return false;

    end_pts[1] = String_Op::to_number<double>( parameters->value( "krw0.value" ) );
    errors->report( end_pts[1] <= 0, "krw0", "Invalid end point rel-perm for brine" );
    if( end_pts[1] <= 0 ) return false;

    expn[0] = String_Op::to_number<double>( parameters->value( "n1.value" ) );
    errors->report( expn[0] <= 0, "n1", "Invalid exponent for gas" );
    if( expn[0] <= 0 ) return false;

    expn[1] = String_Op::to_number<double>( parameters->value( "n2.value" ) );
    errors->report( expn[1] <= 0, "n2", "Invalid exponent for brine" );
    if( expn[1] <= 0 ) return false;

    Sr[0] = String_Op::to_number<double>( parameters->value( "Sgr.value" ) );
    errors->report( Sr[0] <= 0, "Sgr", "Invalid residual gas saturation" );
    if( Sr[0] <= 0 ) return false;

    Sr[1] = String_Op::to_number<double>( parameters->value( "Swr.value" ) );
    errors->report( Sr[1] <= 0, "Swr", "Invalid residual gas saturation" );
    if( Sr[1] <= 0 ) return false;

    for (int s=0;s<1000;s++)
    {
        sat = s/1000.0;
        rel_perm_table[s][0] = sat;
        if (sat<Sr[1])
            kr=0;
        else
        {
            if (sat>1-Sr[0])
                kr = end_pts[1];
            else
                kr = end_pts[1] * pow((sat-Sr[1])/(1-Sr[1]-Sr[0]),expn[1]);
        }
        rel_perm_table[s][1] = kr;
    }

    for (int s=0;s<1000;s++)
    {
        sat = 1 - s/1000.0;
        if (sat<Sr[0])
            kr=0;
        else
        {
            if (sat>1-Sr[1])
                kr = end_pts[0];
            else
                kr = end_pts[0] * pow((sat-Sr[0])/(1-Sr[1]-Sr[0]),expn[0]);
        }
        rel_perm_table[s][2] = kr;
    }

    return true;
}

bool read_fluid_properties(const Parameters_handler* parameters, Error_messages_handler* errors,
double &brine_den_rw, double &co2_den_rw, double &brine_visc_rw, double &co2_visc_rw, double
&ct_rw )
{
    brine_den_rw = String_Op::to_number<double>( parameters->value( "brine_den_rw.value" ) );
    errors->report( brine_den_rw <= 0, "brine_den_rw", "Invalid brine density" );
    if( brine_den_rw <= 0 ) return false;
```

```
    co2_den_rw = String_Op::to_number<double>( parameters->value( "co2_den_rw.value" ) );
    errors->report( co2_den_rw <= 0, "co2_den_rw", "Invalid CO2 density" );
    if( co2_den_rw <= 0 ) return false;

    brine_visc_rw = String_Op::to_number<double>( parameters->value( "brine_visc_rw.value" ) );
    errors->report( brine_visc_rw <= 0, "brine_visc_rw", "Invalid brine viscosity" );
    if( brine_visc_rw <= 0 ) return false;

    co2_visc_rw = String_Op::to_number<double>( parameters->value( "co2_visc_rw.value" ) );
    errors->report( co2_visc_rw <= 0, "co2_visc_rw", "Invalid CO2 viscosity" );
    if( co2_visc_rw <= 0 ) return false;

    ct_rw = String_Op::to_number<double>( parameters->value( "ct_rw.value" ) );
    errors->report( ct_rw <= 0, "ct_rw", "Invalid total compressibility" );
    if( ct_rw <= 0 ) return false;

    return true;

}


void random_walker(int model, Grid_continuous_property *tmp_perm_prop, Grid_continuous_property*
tmp_poro_prop, Grid_continuous_property *tmp_depth_prop, Grid_continuous_property
*tmp_ini_pr_prop, model_data &static_data, RGrid* opt_grid_, double **stats)
{
    // Get static data //
    ofstream fp;
    char check_file[1000];
    strcpy(check_file, static_data.simulation_folder.c_str());
    strcat(check_file, "\\running_check.txt");
    fstream fp2;
    fp2.open("file_name.txt",ios::out);
    fp2<<check_file<<" "<<static_data.simulation_folder<<endl;
    fp2.close();
    fp.open(check_file, ios::out|ios::trunc);
    double ***perm, ***por, ***depth, ***pr_0;
    perm = Allocate3D<double>(static_data.NX,static_data.NY,static_data.NZ);
    por = Allocate3D<double>(static_data.NX,static_data.NY,static_data.NZ);
    depth = Allocate3D<double>(static_data.NX,static_data.NY,static_data.NZ);
    pr_0 = Allocate3D<double>(static_data.NX,static_data.NY,static_data.NZ);
    fp<<"Allocated static arrays\n";
    int counter = 0;
    for (int i=0; i<static_data.NZ; i++)
    {
        for (int j=0; j<static_data.NY; j++)
        {
            for (int k=0; k<static_data.NX; k++)
            {
                perm[k][j][i] = tmp_perm_prop->get_value(counter);
                por[k][j][i] = tmp_poro_prop->get_value(counter);
                pr_0[k][j][i] = tmp_ini_pr_prop->get_value(counter);
                counter++;
            }
        }
    }
    fp<<"Done reading perm, por and pr_0\n";

    counter = 0;
    for (int i=0; i<static_data.NY; i++)
    {
        for (int j=0; j<static_data.NX; j++)
        {
            depth[j][i][0] = tmp_depth_prop->get_value(counter);
            for (int k=1; k<static_data.NZ; k++)
```

```
                depth[j][i][k] = depth[j][i][0] + k*static_data.dz;
            counter++;
        }
    }
    fp<<"Depth: "<<depth[static_data.NX-1][static_data.NY-1][0]<<endl;
    fp<<"Done reading static values\n";
    // CHECK UNITS AND CONVERT TO SI, IF NECESSARY
    if (static_data.selected_units==2)
    {
        if (model==0)
        {
            static_data.dx /= 3.28084; static_data.dy /= 3.28084;
            static_data.brine_den_rw *= 16.018;
            static_data.co2_den_rw *= 16.018;
            static_data.brine_visc_rw *= 0.001;
            static_data.co2_visc_rw *= 0.001;
            static_data.ct_rw /= 6894.7573;
            static_data.inj_rate *= 3.2774e-7;
        }
        matrix_mult<double>(perm, static_data.NX, static_data.NY, static_data.NZ, 9.869e-16);
        matrix_mult<double>(depth, static_data.NX, static_data.NY, static_data.NZ, 1/3.28084);
        matrix_mult<double>(pr_0, static_data.NX, static_data.NY, static_data.NZ, 6894.75);

    }
    fp<<perm[static_data.NX-1][static_data.NY-1][0]<<" "<<depth[static_data.NX-1][static_data.NY-
1][0]<<" "<<pr_0[static_data.NX-1][static_data.NY-1][0]<<endl;
    fp<<"Done converting to SI\n";

    // ********** Initialize random walker arrays **********
    const double pi = 3.14159265359;
    double sum_del_T = 0, ***saturations, ***probability_map;
    double perm_along_path, curr_time, dist_from_injector, avg_perm, avg_satn, potential_diff,
v_BL, incr_time;
    int moveable_particles, num_transitions, MC_sample;

    saturations = Allocate3D<double>(static_data.NX, static_data.NY, static_data.NZ);
    initialize3D<double>(saturations,static_data.NX, static_data.NY, static_data.NZ, 0.0);
    probability_map = Allocate3D<double>(static_data.NX, static_data.NY, static_data.NZ);

    clock_t start;

    int ***carbon_count, ***temp_array, ***check_passed;
    carbon_count = Allocate3D<int>(static_data.NX, static_data.NY, static_data.NZ);
    temp_array = Allocate3D<int>(static_data.NX, static_data.NY, static_data.NZ);
    check_passed = Allocate3D<int>(static_data.NX, static_data.NY, static_data.NZ);

    vector<double> v_macro (6,0.0);
    vector<double> Tr (6,0.0);

    int reporting_counter = 0;

    const int num_jumps = 20;
    double cutoff_percentile = 0.5;
    double time_factor = scouts(model, perm, saturations, pr_0, depth, num_jumps, static_data,
cutoff_percentile);
    fp<<"Time factor: "<<time_factor<<endl;
    while (sum_del_T<static_data.total_days)
    {

        initialize3D<int>(carbon_count,static_data.NX, static_data.NY, static_data.NZ,0);
        vector< vector<int> > occupied_locs;

        if (sum_del_T!=0)
        {
```

```
            find_cutoff(saturations,static_data.NX, static_data.NY,
static_data.NZ,occupied_locs,0.0);
            if (occupied_locs.size()<=1)
            {
                cutoff_percentile += 0.1;
                time_factor = scouts(model, perm, saturations, pr_0, depth, num_jumps,
static_data, cutoff_percentile);
                sum_del_T = 0;
                fp<<"\nTime factor rewrite: "<<time_factor<<endl;
                initialize3D<int>(carbon_count,static_data.NX, static_data.NY, static_data.NZ,0);
                occupied_locs.clear(); occupied_locs.shrink_to_fit();
                vector< vector<int> > occupied_locs;
            }
        }

        for (int location = 0; location<static_data.num_injectors; location++)
        {
            vector<int> temp_locs(3);
            temp_locs[0] = static_data.inj_locs[location][0]-1;
            temp_locs[1] = static_data.inj_locs[location][1]-1;
            temp_locs[2] = static_data.inj_locs[location][2]-1;
            occupied_locs.push_back(temp_locs);
        }
        int total_moving_particles = 0;
        fp<<"Total locations: "<<occupied_locs.size()<<endl;
        for (int location = 0; location<int(occupied_locs.size()); location++)
        {

            start = clock();
            int curr_x = occupied_locs[location][0], curr_y = occupied_locs[location][1], curr_z
= occupied_locs[location][2];
            if (sum_del_T==0)
                moveable_particles = static_data.particles_per_time;
            else
                moveable_particles = static_data.particles_per_time/10;
            total_moving_particles += moveable_particles;

            for (int particle = 0; particle<moveable_particles; particle++)
            {
                fp<<"Location "<<location<<" particle "<<particle<<endl;
                initialize3D<int>(temp_array, static_data.NX, static_data.NY, static_data.NZ, 0);
                initialize3D<int>(check_passed, static_data.NX, static_data.NY, static_data.NZ,
0);

                num_transitions = 1;
                perm_along_path = 0;
                curr_time = 0;

                vector<double> perm_list;
                int x,y,z;

                while (curr_time < static_data.delta_T)
                {
                    if (curr_time==0)
                    {
                        x = curr_x; y = curr_y; z = curr_z;
                    }
                    fp<<x<<","<<y<<","<<z<<" -> ";
                    temp_array[x][y][z] = 1;
                    check_passed[x][y][z] = 1;
                    perm_list.push_back(perm[x][y][z]);
                    dist_from_injector = perm_list.size()*static_data.dx;
                    if (perm_list.size()==1)
```

```
                        perm_along_path =
percentile(tmp_perm_prop,static_data.NX,static_data.NY,static_data.NZ,100,0);
                    else
                    {
                        if (check_passed[x][y][z]!=1)
                                perm_along_path = num_transitions/((num_transitions-
1)/perm_along_path + 1/perm[x][y][z]);
                    }

                    #pragma region *************** TRANSITION PROBABILITIES ****************//*
                    double del_P;

                    for (int i=0; i<6; i++)
                    {
                        v_macro[i] = 0.0;
                        Tr[i] = 0.0;
                    }
                    // (+)X

                    if (x!=static_data.NX-1 && static_data.NX!=1 && check_passed[x+1][y][z]!=1)
                    {
                        del_P = abs(pr_0[x][y][z]-pr_0[x+1][y][z]) + static_data.inj_rate *
static_data.brine_visc_rw/(4*pi*perm_along_path)*(1/dist_from_injector-1/(dist_from_injector +
static_data.dx));
                        avg_perm =
2*perm[x][y][z]*perm[x+1][y][z]/(perm[x][y][z]+perm[x+1][y][z]);
                        avg_satn = (saturations[x][y][z]+saturations[x+1][y][z])/2;
                        potential_diff = del_P + (static_data.brine_den_rw -
static_data.co2_den_rw) *9.8* (depth[x][y][z]-depth[x+1][y][z]);
                        v_macro[0] =
(potential_diff*avg_perm/static_data.co2_visc_rw/static_data.dx);
                        if (avg_satn<=0)
                                v_BL = 0;
                        else
                        {
                                int index = ((floor(avg_satn*1000)-1)<0)?0:(floor(avg_satn*1000)-
1);
                                v_BL = static_data.fw_table[index][1];
                        }
                        Tr[0] = v_macro[0];
                        v_macro[0] = v_macro[0]*(1+v_BL);
                        if
(static_data.dx/v_macro[0]/24/3600/time_factor>(static_data.delta_T*1.2 - curr_time))
                                Tr[0] = 0;
                    }

                    // (-)X

                    if (x!=0 && static_data.NX!=1 && check_passed[x-1][y][z]!=1)
                    {
                        del_P = abs(pr_0[x][y][z]-pr_0[x-1][y][z]) +
static_data.inj_rate*static_data.brine_visc_rw/(4*pi*perm_along_path)*(1/dist_from_injector-
1/(dist_from_injector+static_data.dx));
                        avg_perm = 2*perm[x][y][z]*perm[x-1][y][z]/(perm[x][y][z]+perm[x-
1][y][z]);
                        avg_satn = (saturations[x][y][z]+saturations[x-1][y][z])/2;
                        potential_diff = del_P + (static_data.brine_den_rw -
static_data.co2_den_rw) * 9.8 * (depth[x][y][z]-depth[x-1][y][z]);
                        v_macro[1] =
(potential_diff*avg_perm/static_data.co2_visc_rw/static_data.dx);
                        if (avg_satn<=0)
                                v_BL = 0;
                        else
                        {
```

```
                                int index = ((floor(avg_satn*1000)-1)<0)?0:(floor(avg_satn*1000)-
1);
                                v_BL = static_data.fw_table[index][1];
                        }
                        Tr[1] = v_macro[1];
                        v_macro[1] = v_macro[1]*(1+v_BL);
                        if
(static_data.dx/v_macro[1]/24/3600/time_factor>(static_data.delta_T*1.2 - curr_time))
                                Tr[1] = 0;
                    }

                    // (+)Y

                    if (y!=static_data.NY-1 && static_data.NY!=1 && check_passed[x][y+1][z]!=1)
                    {
                        del_P = abs(pr_0[x][y][z]-pr_0[x][y+1][z]) + static_data.inj_rate *
static_data.brine_visc_rw/(4*pi*perm_along_path)*(1/dist_from_injector-
1/(dist_from_injector+static_data.dy));
                        avg_perm =
2*perm[x][y][z]*perm[x][y+1][z]/(perm[x][y][z]+perm[x][y+1][z]);
                        avg_satn = (saturations[x][y][z]+saturations[x][y+1][z])/2;
                        potential_diff = del_P + (static_data.brine_den_rw -
static_data.co2_den_rw) * 9.8 * (depth[x][y][z]-depth[x][y+1][z]);
                        v_macro[2] =
(potential_diff*avg_perm/static_data.co2_visc_rw/static_data.dy);
                        if (avg_satn<=0)
                                v_BL = 0;
                        else
                        {
                                int index = ((floor(avg_satn*1000)-1)<0)?0:(floor(avg_satn*1000)-
1);
                                v_BL = static_data.fw_table[index][1];
                        }
                        Tr[2] = v_macro[2];
                        v_macro[2] = v_macro[2]*(1+v_BL);
                        if
(static_data.dy/v_macro[2]/24/3600/time_factor>(static_data.delta_T*1.2 - curr_time))
                                Tr[2] = 0;
                    }

                    // (-)Y

                    if (y!=0 && static_data.NY!=1  && check_passed[x][y-1][z]!=1)
                    {
                        del_P = abs(pr_0[x][y][z]-pr_0[x][y-1][z]) +
static_data.inj_rate*static_data.brine_visc_rw/(4*pi*perm_along_path)*(1/dist_from_injector-
1/(dist_from_injector+static_data.dy));
                        avg_perm = 2*perm[x][y][z]*perm[x][y-1][z]/(perm[x][y][z]+perm[x][y-
1][z]);
                        avg_satn = (saturations[x][y][z]+saturations[x][y-1][z])/2;
                        potential_diff = del_P + (static_data.brine_den_rw-
static_data.co2_den_rw)*9.8*(depth[x][y][z]-depth[x][y-1][z]);
                        v_macro[3] =
(potential_diff*avg_perm/static_data.co2_visc_rw/static_data.dy);
                        if (avg_satn<=0)
                                v_BL = 0;
                        else
                        {
                                int index = ((floor(avg_satn*1000)-1)<0)?0:(floor(avg_satn*1000)-
1);
                                v_BL = static_data.fw_table[index][1];
                        }
                        Tr[3] = v_macro[3];
                        v_macro[3] = v_macro[3]*(1+v_BL);
```

```
                        if
(static_data.dy/v_macro[3]/24/3600/time_factor>(static_data.delta_T*1.2 - curr_time))
                                Tr[3] = 0;
                    }

                    // (+)Z

                    if (z!=static_data.NZ-1 && static_data.NZ!=1 && check_passed[x][y][z+1]!=1)
                    {
                        del_P = abs(pr_0[x][y][z]-pr_0[x][y][z+1]) +
static_data.inj_rate*static_data.brine_visc_rw/(4*pi*perm_along_path)*(1/dist_from_injector-
1/(dist_from_injector+abs(depth[x][y][z]-depth[x][y][z+1])));
                        avg_perm =
2*perm[x][y][z]*perm[x][y][z+1]/(perm[x][y][z]+perm[x][y][z+1]);
                        avg_satn = (saturations[x][y][z]+saturations[x][y][z+1])/2;
                        potential_diff = del_P + (static_data.brine_den_rw-
static_data.co2_den_rw)*9.8*abs(depth[x][y][z]-depth[x][y][z+1]);
                        v_macro[4] =
(potential_diff*avg_perm/static_data.co2_visc_rw/(depth[x][y][z]-depth[x][y][z+1]));
                        if (avg_satn<=0)
                                v_BL = 0;
                        else
                        {
                                int index = ((floor(avg_satn*1000)-1)<0)?0:(floor(avg_satn*1000)-
1);
                                v_BL = static_data.fw_table[index][1];
                        }
                        Tr[4] = v_macro[4];
                        v_macro[4] = v_macro[4]*(1+v_BL);
                        if
(static_data.dz/v_macro[4]/24/3600/time_factor>(static_data.delta_T*1.2 - curr_time))
                                Tr[4] = 0;
                    }

                    // (-)Z

                    if (z!=0 && static_data.NZ!=1&& check_passed[x][y][z-1]!=1)
                    {
                        del_P = abs(pr_0[x][y][z]-pr_0[x][y][z-1]) +
static_data.inj_rate*static_data.brine_visc_rw/(4*pi*perm_along_path)*(1/dist_from_injector-
1/(dist_from_injector+abs(depth[x][y][z]-depth[x][y][z-1])));
                        avg_perm = 2*perm[x][y][z]*perm[x][y][z-1]/(perm[x][y][z]+perm[x][y][z-
1]);
                        avg_satn = (saturations[x][y][z]+saturations[x][y][z-1])/2;
                        potential_diff = del_P + (static_data.brine_den_rw-
static_data.co2_den_rw)*9.8*abs(depth[x][y][z]-depth[x][y][z-1]);
                        v_macro[5] =
(potential_diff*avg_perm/static_data.co2_visc_rw/(depth[x][y][z]-depth[x][y][z-1]));
                        if (avg_satn<=0)
                                v_BL = 0;
                        else
                        {
                                int index = ((floor(avg_satn*1000)-1)<0)?0:(floor(avg_satn*1000)-
1);
                                v_BL = static_data.fw_table[index][1];
                        }
                        Tr[5] = v_macro[5];
                        v_macro[5] = v_macro[5]*(1+v_BL);
                        if
(static_data.dz/v_macro[5]/24/3600/time_factor>(static_data.delta_T*1.2 - curr_time))
                                Tr[5] = 0;
                    }
                    #pragma endregion
```

```
    double sum_Tr = 0;

    for (int i = 0; i<6; i++)
    {

        if (Tr[i]<0)
                Tr[i] = 0.0;
        sum_Tr += Tr[i];
        if (v_macro[i]<0)
                v_macro[i] = 0.0;
    }

    if (sum_Tr==0)
        break;
    else // Transition probability distribution sampling
    {
        for (int i=0; i<6; i++)
        {
                Tr[i] /= sum_Tr;
        }
        MC_sample = monte_carlo_sampling(Tr);

        switch(MC_sample)
        {
        case 1:
                x=x+1;
                incr_time = static_data.dx/v_macro[0];
                break;
        case 2:
                x=x-1;
                incr_time = static_data.dx/v_macro[1];
                break;
        case 3:
                y=y+1;
                incr_time = static_data.dy/v_macro[2];
                break;
        case 4:
                y=y-1;
                incr_time = static_data.dy/v_macro[3];
                break;
        case 5:
                z=z+1;
                incr_time = abs(depth[x][y][z-1]-depth[x][y][z])/v_macro[4];
                break;
        case 6:
                z=z-1;
                incr_time = abs(depth[x][y][z+1]-depth[x][y][z])/v_macro[5];
                break;
        }
    }


    if (check_passed[x][y][z]!=1)
    {
        num_transitions += 1;
        curr_time += incr_time/(24*3600)/time_factor;
    }

}// end of while (curr_time<delta_T)
fp<<endl;
perm_list.clear(); perm_list.shrink_to_fit();
for (int i=0; i<static_data.NX; i++)
{
    for (int j=0; j<static_data.NY; j++)
```

```
                    {
                        for (int k=0; k<static_data.NZ; k++)
                        {
                                carbon_count[i][j][k] += temp_array[i][j][k];
                        }
                    }
                }


            }// end of -> for (int particle = 0; particle<moveable_particles; particle++)


        }// end of -> for (int location = 0; location<occupied_locs.size(); location++)
        fp<<"\nEnd of movements! Total moving particles: "<<total_moving_particles<<endl;
        occupied_locs.clear(); occupied_locs.shrink_to_fit();

        initialize3D<double>(probability_map,static_data.NX, static_data.NY, static_data.NZ, 0.0);
        for (int i=0; i<static_data.NX; i++)
        {
            for (int j=0; j<static_data.NY; j++)
            {
                for (int k=0; k<static_data.NZ; k++)
                {
                    probability_map[i][j][k] =
carbon_count[i][j][k]/double(total_moving_particles);
                }
            }
        }

        fp<<"\nGoing into probability_to_saturations... ";
        probability_to_satn(saturations, probability_map, 0.5, static_data, time_factor);
        fp<<"and back!\n";
        double sum_PV = 0;
        int non_zero_counter = 0;
        for (int i=0; i<static_data.NX; i++)
        {
            for (int j=0; j<static_data.NY; j++)
            {
                for (int k=0; k<static_data.NZ; k++)
                {
                    if (saturations[i][j][k]>0)
                    {
                        non_zero_counter++;
                        sum_PV += saturations[i][j][k] * static_data.dx * static_data.dy *
static_data.dz * por[i][j][k];
                    }
                }
            }
        }

        //reporting_counter += (sum_PV/static_data.inj_rate)/(24*3600);
        //sum_del_T += static_data.delta_T;
        //sum_del_T += sum_PV/static_data.inj_rate/24/3600;



        /*if ( reporting_counter > static_data.delta_T)
        {*/
        sum_del_T += static_data.delta_T;  reporting_counter = 0;
        if (int(sum_del_T) % int(static_data.reporting_interval) == 0)
        {
            Grid_continuous_property *tmp_rw_prop;
            if (non_zero_counter>1)
            {
```

```
                tmp_rw_prop = opt_grid_->add_property("RW_result_Time_" +
static_cast<ostringstream*>( &(ostringstream() << sum_del_T) )->str() + "_" + tmp_perm_prop-
>name());
                int counter = 0;
                int row_num = sum_del_T/static_data.reporting_interval - 1;
                for (int i=0; i<static_data.NZ; i++)
                {
                    for (int j=0; j<static_data.NY; j++)
                    {
                        for (int k=0; k<static_data.NX; k++)
                        {
                                tmp_rw_prop->set_value(saturations[k][j][i], counter);
                                counter++;
                        }
                    }
                }
                fp<<"Writing stats...\n";
                for (int counter2 = 0; counter2<static_data.num_monitors; counter2++)
                {
                    int x_ = static_data.monitor_locs[counter2][0]; int y_ =
static_data.monitor_locs[counter2][1]; int z_ = static_data.monitor_locs[counter2][2];
                    stats[model][row_num * static_data.num_monitors + counter2] =
probability_map[x_-1][y_-1][z_-1];// + (rand()%100)/100000.0;

                }
            }
        }
//      }

    }// end of while (sum_del_T<=total_days)
    fp.close();
    Tr.clear(); Tr.shrink_to_fit();
    v_macro.clear(); v_macro.shrink_to_fit();
    Free3D<int>(carbon_count);
    Free3D<int>(temp_array);
    Free3D<int>(check_passed);
    Free3D<double>(saturations);
    Free3D<double>(por);
    Free3D<double>(perm);
    Free3D<double>(depth);
    Free3D<double>(probability_map);
    Free3D<double>(pr_0);
}

int get_index_rw(int i, int j, int k, int NX, int NY)
{
    if (k>=0)
        return ( i + NX*j + NX*NY*k );
    else
        return ( i + NX*j );
}


double scouts(int model, double ***perm, double ***saturations, double ***pr_0, double ***depth,
const int MAX_JUMPS, model_data &static_data, double cutoff_percentile)
{
//  Function to find the median ‘time’ out of MAX_JUMPS transition made by the walker.
}


int monte_carlo_sampling(vector<double>a)
{
//  Function for Monte-Carlo sampling
}
```

```
void probability_to_satn(double ***satn, double ***prob, double S, model_data static_data, double
time_factor)
{
//  Function to convert from probability map to saturation map
}
```

***Cluster_to_simulation.cpp*:** This contains the codes for various operations during the model selection, like projections to principal component axes, clustering, and calculation of posterior probabilities.

```
// Header files <>
#include "matrix_def.h"
#include "cluster_to_simulation.h"

void zeromean(int count, int parms, double **original, double **zmean)
{
    int i,j;
    cout<<"\nInside zeromean\n";
    for (i=0;i<parms;i++)
    {
        double sum=0;
        for (j=0;j<count;j++)
        {
            sum+=original[j][i];
        }
        sum/=count;
        for (j=0;j<count;j++)
            zmean[j][i]=original[j][i]-sum;
    }

}

void covmat(int count, double **cov, int parms, double **zmean)
{
    int i,j,k;

    for (i=0;i<parms;i++)
    {
        for (j=0;j<parms;j++)
        {
            cov[i][j]=0;
            for (k=0;k<count;k++)
                cov[i][j]+=zmean[k][i]*zmean[k][j];
            cov[i][j]/=count;
        }
    }
}

void svd(int nmods, int parms, double **cov, double **evecs)
{
    cout<<"\nInside SVD\n";
    int k,l,m,n,count=0;
    m=parms; n=parms;
    Vec_DP w(n);
    Mat_DP a(m,n),u(m,n),v(n,n);
    for (k=0;k<m;k++)
    {
        for (l=0;l<n;l++)
        {
            a[k][l]=cov[k][l];
            u[k][l]=a[k][l];
        }
    }
    // Done reading inside SVD
    NR::svdcmp(u,w,v);
    for (k=0;k<m;k++)
```

```
    {
        for (l=0;l<3;l++)
            evecs[k][l]=u[k][l];
    }
    //Done with SVD
}


void NR::svdcmp(Mat_IO_DP &a, Vec_O_DP &w, Mat_O_DP &v)
{
    bool flag;
    int i,its,j,jj,k,l,nm;
    DP anorm,c,f,g,h,s,scale,x,y,z;

    int m=a.nrows();
    int n=a.ncols();
    Vec_DP rv1(n);
    g=scale=anorm=0.0;
    for (i=0;i<n;i++)
    {
        l=i+2;
        rv1[i]=scale*g;
        g=s=scale=0.0;
        if (i < m)
        {
            for (k=i;k<m;k++) scale += fabs(a[k][i]);
            if (scale != 0.0) {
                for (k=i;k<m;k++) {
                    a[k][i] /= scale;
                    s += a[k][i]*a[k][i];
                }
                f=a[i][i];
                g = -SIGN(sqrt(s),f);
                h=f*g-s;
                a[i][i]=f-g;
                for (j=l-1;j<n;j++) {
                    for (s=0.0,k=i;k<m;k++) s += a[k][i]*a[k][j];
                    f=s/h;
                    for (k=i;k<m;k++) a[k][j] += f*a[k][i];
                }
                for (k=i;k<m;k++) a[k][i] *= scale;
            }
        }
        w[i]=scale *g;
        g=s=scale=0.0;
        if (i+1 <= m && i+1 != n) {
            for (k=l-1;k<n;k++) scale += fabs(a[i][k]);
            if (scale != 0.0) {
                for (k=l-1;k<n;k++) {
                    a[i][k] /= scale;
                    s += a[i][k]*a[i][k];
                }
                f=a[i][l-1];
                g = -SIGN(sqrt(s),f);
                h=f*g-s;
                a[i][l-1]=f-g;
                for (k=l-1;k<n;k++) rv1[k]=a[i][k]/h;
                for (j=l-1;j<m;j++) {
                    for (s=0.0,k=l-1;k<n;k++) s += a[j][k]*a[i][k];
                    for (k=l-1;k<n;k++) a[j][k] += s*rv1[k];
                }
                for (k=l-1;k<n;k++) a[i][k] *= scale;
            }
        }
```

```
    anorm=MAX(anorm,(fabs(w[i])+fabs(rv1[i])));
}
for (i=n-1;i>=0;i--) {
    if (i < n-1) {
        if (g != 0.0) {
            for (j=l;j<n;j++)
                v[j][i]=(a[i][j]/a[i][l])/g;
            for (j=l;j<n;j++) {
                for (s=0.0,k=l;k<n;k++) s += a[i][k]*v[k][j];
                for (k=l;k<n;k++) v[k][j] += s*v[k][i];
            }
        }
        for (j=l;j<n;j++) v[i][j]=v[j][i]=0.0;
    }
    v[i][i]=1.0;
    g=rv1[i];
    l=i;
}
for (i=MIN(m,n)-1;i>=0;i--) {
    l=i+1;
    g=w[i];
    for (j=l;j<n;j++) a[i][j]=0.0;
    if (g != 0.0) {
        g=1.0/g;
        for (j=l;j<n;j++) {
            for (s=0.0,k=l;k<m;k++) s += a[k][i]*a[k][j];
            f=(s/a[i][i])*g;
            for (k=i;k<m;k++) a[k][j] += f*a[k][i];
        }
        for (j=i;j<m;j++) a[j][i] *= g;
    } else for (j=i;j<m;j++) a[j][i]=0.0;
    ++a[i][i];
}
for (k=n-1;k>=0;k--) {
    for (its=0;its<30;its++) {
        flag=true;
        for (l=k;l>=0;l--) {
            nm=l-1;
            if (fabs(rv1[l])+anorm == anorm) {
                flag=false;
                break;
            }
            if (fabs(w[nm])+anorm == anorm) break;
        }
        if (flag) {
            c=0.0;
            s=1.0;
            for (i=l;i<k+1;i++) {
                f=s*rv1[i];
                rv1[i]=c*rv1[i];
                if (fabs(f)+anorm == anorm) break;
                g=w[i];
                h=pythag(f,g);
                w[i]=h;
                h=1.0/h;
                c=g*h;
                s = -f*h;
                for (j=0;j<m;j++) {
                    y=a[j][nm];
                    z=a[j][i];
                    a[j][nm]=y*c+z*s;
                    a[j][i]=z*c-y*s;
                }
            }
```

```
            }
            z=w[k];
            if (l == k) {
                if (z < 0.0) {
                    w[k] = -z;
                    for (j=0;j<n;j++) v[j][k] = -v[j][k];
                }
                break;
            }
            if (its == 29) nrerror("no convergence in 30 svdcmp iterations");
            x=w[l];
            nm=k-1;
            y=w[nm];
            g=rv1[nm];
            h=rv1[k];
            f=((y-z)*(y+z)+(g-h)*(g+h))/(2.0*h*y);
            g=pythag(f,1.0);
            f=((x-z)*(x+z)+h*((y/(f+SIGN(g,f)))-h))/x;
            c=s=1.0;
            for (j=l;j<=nm;j++) {
                i=j+1;
                g=rv1[i];
                y=w[i];
                h=s*g;
                g=c*g;
                z=pythag(f,h);
                rv1[j]=z;
                c=f/z;
                s=h/z;
                f=x*c+g*s;
                g=g*c-x*s;
                h=y*s;
                y *= c;
                for (jj=0;jj<n;jj++) {
                    x=v[jj][j];
                    z=v[jj][i];
                    v[jj][j]=x*c+z*s;
                    v[jj][i]=z*c-x*s;
                }
                z=pythag(f,h);
                w[j]=z;
                if (z) {
                    z=1.0/z;
                    c=f*z;
                    s=h*z;
                }
                f=c*g+s*y;
                x=c*y-s*g;
                for (jj=0;jj<m;jj++) {
                    y=a[jj][j];
                    z=a[jj][i];
                    a[jj][j]=y*c+z*s;
                    a[jj][i]=z*c-y*s;
                }
            }
            rv1[l]=0.0;
            rv1[k]=f;
            w[k]=x;
        }
    }
}

DP NR::pythag(const DP a, const DP b)
{
```

```
    DP absa,absb;

    absa=fabs(a);
    absb=fabs(b);
    if (absa > absb) return absa*sqrt(1.0+SQR(absb/absa));
    else return (absb == 0.0 ? 0.0 : absb*sqrt(1.0+SQR(absa/absb)));
}

void proj(int count, double **projected, int parms, double **original, double **evecs)
{
    int i,j,k;
    cout<<"\nCreated projection memory allocation\n";
    for (i=0;i<count;i++)
    {
        for (j=0;j<3;j++)
        {
            projected[i][j]=0;
            for (k=0;k<parms;k++)
                projected[i][j]+=original[i][k]*evecs[k][j];
        }
    }

    cout<<"\nDone projecting\n";
}

int cluster(int count, double **projected,int *clusterid)
{
    int i,j,k;
    int num_clus, **mask, ifound, *dtemp;
    int nrows = count, ncols = 3;
    double *wt, **d, error, errs[3];
    wt = (double*) malloc(count*sizeof(double));
    d = Allocate2D<double>(count,3);
    mask  = Allocate2D<int>(count,3);

    for (j=0;j<count;j++)
    {
        for (i=0;i<3;i++)
        {
            d[j][i]=projected[j][i];
            mask[j][i]=1;
        }
        wt[j]=1.0;
    }

    num_clus=4;
    kcluster(num_clus,nrows,ncols,d,mask,wt,0,20,'a','e',clusterid, &error, &ifound);

    // Done clustering
    free(mask);
    free(wt);
    free(d);
    return(num_clus);
}

void find_centroids(int count, int num_clus, double **cdata, int *clusterid, double **projected)
{
    int i,j,k;
    int **mask, ifound;
    int nrows = count, ncols = 3;
    double *wt, **d, error, errs[3];
    wt = (double*) malloc(count*sizeof(double));
    d = Allocate2D<double>(count,3);
    mask  = Allocate2D<int>(count,3);
```

```
    for (j=0;j<count;j++)
    {
        for (i=0;i<3;i++)
        {
            d[j][i]=projected[j][i];
            mask[j][i]=1;
        }
        wt[j]=1.0;
    }
    int **cmask;
    cmask = Allocate2D<int>(num_clus,3);
    int success = getclustercentroids(num_clus, nrows, ncols, d, mask, clusterid, cdata, cmask, 0,
'a');
    free(mask);
    free(wt);
    free(d);
    Free2D<int>(cmask);
}
void read_report(int clus_num, double **rep, int t_steps, string report_file)
{
    int i;
    fstream opt1;
    char temp[10000], fname[1000];
    strcpy(fname,report_file.c_str());
    double temp2;
    opt1.open(fname,ios::in);
    if (opt1.is_open())
    {
        for (i=0;i<10;i++)
            opt1.getline(temp,10000);

        for (i=0;i<t_steps;i++)
        {
            opt1>>temp2;
            opt1>>rep[i][clus_num];
        }
        opt1.close();
    }
}

double erf(double val)
{
    double x;
    if (val<=.15)
        x = 1.12838*val;
    if (val>.15 && val<=1.5)
        x = -.0198+val*(1.2911-.4262*val);
    if (val>1.5 && val<=2)
        x = .8814+.0584*val;
    if (val>2)
        x = 1;
    return(x);
}

void find_best_models(int count, int num_clus, int *clusterid, double **centroids, double
**projected, int *rep_models)
{
    int i,j;

    fstream fp;
    fp.open("cluster_bests.txt",ios::out);

    // FIND MODEL CLOSEST TO CLUSTER CENTROID
```

```
    for (i=0;i<num_clus;i++)
    {
        fp<<"Cluster "<<i+1<<endl;
        vector< vector<int> > distances;
        for (j=0; j<count; j++)
        {
            if (clusterid[j]==i)
            {
                vector <int> c(2);
                c[0] = j;
                c[1] = std::abs(centroids[i][0]-projected[j][0]) + std::abs(centroids[i][1]-
projected[j][1]) + std::abs(centroids[i][2]-projected[j][2]);
                distances.push_back(c);
            }
        }
        sort(distances.begin(), distances.end(), [](const std::vector< int >& a, const
std::vector< int >& b){ return a[1] < b[1]; } );

        fp<<distances[0][0]<<" "<<distances[1][0]<<" "<<distances[2][0]<<endl;
        rep_models[i] = distances[0][0];

    }
}

vector <double> calc_probability(double **rep, double **observed, int t_steps, int num_clus)
{
    int i, j, k;
    double **prob, **dev, *stdev;
    // CALCULATION OF CLUSTER PROBABILITY
    dev = Allocate2D<double>(t_steps, num_clus);
    prob = Allocate2D<double>(t_steps, num_clus);

    stdev = (double*) malloc(t_steps*sizeof(double));

    for (j=0;j<t_steps;j++)
    {
        for (k=0;k<num_clus;k++)
            dev[j][k]=fabs(rep[j][k]-observed[j][1]);
        double  max = dev[j][0];
        for (k=1;k<num_clus;k++)
        {
            if (dev[j][k]>max)
                max=dev[j][k];
        }
        stdev[j] = max;
    }

    double x;

    for (j=0;j<t_steps;j++)
    {
        for (k=0;k<num_clus;k++)
        {
            x=((1-dev[j][k]/stdev[j])-0.5)/(sqrt(2*.25));
            prob[j][k]=0.5*(1+erf(x));
        }
    }

    vector<double> min_(num_clus);

    for (j=0; j<num_clus; j++)
    {
        min_[j] = prob[0][j];
        for (k=1; k<t_steps; k++)
```

```
        {
            if (prob[k][j]<min_[j])
                min_[j] = prob[k][j];
        }
    }

    Free2D<double>(dev); free(stdev); Free2D<double>(prob);

    return min_;
}

void projections(double **stats, double **projections, int nrows, int ncolumns)
{
    double mean, stdev, **v, *w;

    for (int i=0; i<ncolumns; i++)
    {
        mean = 0; stdev = 0;
        for (int j=0; j<nrows; j++)
            mean+= stats[j][i];
        mean = mean/nrows;
        for (int j=0; j<nrows; j++)
        {
            stats[j][i] = stats[j][i]-mean;
            stdev+= pow(stats[j][i],2);
        }
        stdev = std::pow(stdev/nrows,0.5);
        for (int j=0; j<nrows; j++)
            stats[j][i] = stats[j][i]/stdev;
    }
    v = Allocate2D<double>(ncolumns,ncolumns);
    w = new double[ncolumns];
    int result = pca(nrows, ncolumns, stats, v, w);
    for (int i=0; i<3; i++)
    {
        for (int j=0; j<nrows; j++)
            projections[j][i] = stats[j][i];
    }
}
```

Codes for PCA and Clustering

Aside from the above codes, the model selection plugin used codes from external developers for clustering, principal component analysis and related operations. These were developed as open-source codes at the Laboratory of DNA Information Analysis, at the University of Tokyo and are available for download at:

http://bonsai.hgc.jp/~mdehoon/software/cluster/software.htm

Nomenclature

s_k	State vector in EnKF
d_k	Observation vector in EnKF
$e_{k,i}^0$	Random noise drawn from a multinormal distribution with zero mean and covariance R_k in EnKF
$s_{k,i}^f$	Forecast vector in EnKF
K_k	Kalman gain in EnKF
P_k^f	Model error covariance matrix in EnKF
v_i^{k+1}	Velocity of particle *i* at time *k+1* in PSO
x_i^k	Position of particle *i* at time *k* in PSO
$p_{best,i}$	Best position of particle *i* in PSO
g_{best}^k	Best position of entire swarm in PSO
$\tau(\mathbf{x})$	Travel time along streamline
D	Dispersion tensor
B	Displacement matrix in RWPT, related to the dispersion tensor $\mathbf{D} = \frac{\mathbf{1}}{\mathbf{2}}\mathbf{B} \cdot \mathbf{B^T}$
k_{avg}	Harmonic average permeability
$P\left(z^m(\mathbf{u})\|RF_{ref}\right)$	Probability of cluster 'm' given conditioning data RF_{ref}
RF^m	Simulated response of cluster 'm'
RF_{ref}	Observed response
u	Velocity
$\mathbf{X_p}(t)$	Position of random particle at time 't'
ΔN	Difference in particle count

η_k	Efficiency of clustering
σ_m^2	Deviation of simulated response from observed response
P_{analog}	Pressure analog from random walker proxy
ΔP	Difference in pressure
V_b	Bulk volume
ϕ	Porosity
S_{crit}	Critical saturation
V_{inj}	Injected volume of fluid
S_{avg}	Average saturation
$\xi(t)$	Normal distribution of mean 0 and variance 1
P_c	Capillary pressure
k_{rnw}	Relative permeability of non-wetting phase
k_{rw}	Relative permeability of wetting phase
μ_w	Viscosity of wetting phase
μ_{nw}	Viscosity of non-wetting phase
$\Delta\rho$	Density difference
α	Angle with horizontal
v_{BL}	Buckley-Leverett velocity
v_{macro}	Darcy velocity
f_W	Fractional flow value

References

Abdollahzadeh, A., Reynolds, A., Christie, M., Corne, D., Williams, G., & Davies, B. (2013). Estimation of Distribution Algorithms Applied to History Matching, (May 2012), 21–23.

Abreu, V., Sullivan, M., Pirmez, C., & Mohrig, D. (2003). Lateral accretion packages (LAPs): an important reservoir element in deep water sinuous channels. *Marine and Petroleum Geology*. doi:10.1016/j.marpetgeo.2003.08.003

Allen, G., & Chambers, J. (1998). Sedimentation in the modern and Miocene Mahakam Delta. Retrieved from http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Sedimentation+in+the+modern+and+Miocene+Mahakam+Delta#0

Alpak, F. O., Barton, M. D., & Caers, J. (2009). A flow-based pattern recognition algorithm for rapid quantification of geologic uncertainty. *Computational Geosciences*, *14*(4), 603–621. doi:10.1007/s10596-009-9175-5

Arnold, D., Demyanov, V., Tatum, D., Christie, M., Rojas, T., Geiger, S., & Corbett, P. (2013). Hierarchical benchmark case study for history matching, uncertainty quantification and reservoir characterization. *Computers & Geosciences*, *50*, 4–15. doi:10.1016/j.cageo.2012.09.011

Arts, R., Chadwick, A., Eiken, O., Thibeau, S., & Nooner, S. (2008). Ten years' experience of monitoring CO_2 injection in the Utsira Sand at Sleipner, offshore Norway. *First Break*, *26*, 65–72.

Birkholzer, J., Zhou, Q., & Tsang, C. (2009). Large-scale impact of CO2 storage in deep saline aquifers: A sensitivity study on pressure response in stratified systems. *International Journal of Greenhouse Gas Control*, *3*(2), 181–194. doi:10.1016/j.ijggc.2008.08.002

Boomer, R. J. (1996). Predicting production using a neural network (artificial intelligence beats human intelligence). In *Permian Basin oil & gas recovery conference* (pp. 65–74).

Buckley, S. E., & Leverett, M. C. (1942). Mechanism of fluid displacements in sand. *Transactions AIME*, *146*, 107–116.

Chadwick, R. A., Holloway, S., Kirby, G. A., Gregersen, U., & Johannessen, P. N. (2001). The Utsira Sand, central North Sea -- an assessment of its potential for

regional CO_2 storage. In *Fifth International Conference on Greenhouse Gas Control Technologies, August 13-16 2000, Cairns, Australia* (pp. 349–354).

Chadwick, R. A., Williams, G. A., Williams, J. D. O., & Noy, D. J. (2012). Measuring pressure performance of a large saline aquifer during industrial-scale CO2 injection: The Utsira Sand, Norwegian North Sea. *International Journal of Greenhouse Gas Control*, *10*, 374–388. doi:10.1016/j.ijggc.2012.06.022

Chen, Y., & Oliver, D. (2010). Ensemble-Based Closed-Loop Optimization Applied to Brugge Field. *SPE Reservoir Evaluation & Engineering*, *13*(1). doi:10.2118/118926-PA

Coats, K. H., Dempsey, J. R., & Henderson, J. H. (1970). A New Technique for Determining Reservoir Description from Field Performance Data. *Society of Petroleum Engineers Journal*, *10*(1). doi:10.2118/2344-PA

Datta-Gupta, A., & Kulkarni, K. (2001). Streamlines, ray tracing and production tomography: Generalization to compressible flow. *Petroleum Geoscience*, *7*, 75–86.

Davis, N., Riddiford, F., Bishop, C., Taylor, B., & Froukhi, R. (2001). The In Salah Gas Project , Central Algeria : Bringing an Eight Field Gas Development to Sanction. In *SPE Middle East Oil Show* (Vol. 2). doi:10.2118/68180-MS

Efendiev, Y., Hou, T., & Luo, W. (2006). Preconditioning Markov Chain Monte Carlo Simulations Using Coarse-Scale Models. *SIAM Journal on Scientific Computing*, *28*(2), 776–803. doi:10.1137/050628568

Emerick, A., & Reynolds, A. (2012). Combining the Ensemble Kalman Filter With Markov-Chain Monte Carlo for Improved History Matching and Uncertainty Characterization. *SPE Journal*, *17*(2). doi:10.2118/141336-PA

Gharbi, R., & Elsharkawy, A. (1999). Neural Network Model for Estimating the PVT Properties of Middle East Crude Oils. *SPE Reservoir Evaluation & Engineering*, *2*(3). doi:10.2118/56850-PA

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning* (1st ed.). Boston, MA: Addison-Wesley Longman Publishing Co., Inc.

Goldstein, R. M., Engelhardt, H., Kamb, B., & Frolich, R. M. (1993). Satellite radar interferometry for monitoring ice sheet motion: application to an Antarctic ice stream. *Science (New York, N.Y.)*, *262*(5139), 1525–30. doi:10.1126/science.262.5139.1525

Guler, B., Ertekin, T., & Grader, A. (2003). An Artificial Neural Network Based Relative Permeability Predictor. *Journal of Canadian Petroleum Technology*, *42*(4). doi:10.2118/03-04-02

Hegeman, P., Dong, C., Varotsis, N., & Gaganis, V. (2009). Application of Artificial Neural Networks to Downhole Fluid Analysis. *SPE Reservoir Evaluation & Engineering*, *12*(1). doi:10.2118/123423-PA

Iding, M., & Ringrose, P. (2009). Evaluating the impact of fractures on the long-term performance of the In Salah CO2 storage site. *Energy Procedia*, *1*(1), 2021–2028. doi:10.1016/j.egypro.2009.01.263

Ito, K. (1951). *On stochastic differential equations* (pp. 289–302). New York: American Mathematical Society.

John, A., Lake, L., Bryant, S., & Jennings, J. (2010). Investigation of Mixing in Field-Scale Miscible Displacements Using Particle-Tracking Simulations of Tracer Floods With Flow Reversal. *SPE Journal*, *15*(3). doi:10.2118/113429-PA

Johnson, J. W., Nitao, J. J., & Knauss, K. G. (2004). Reactive transport modelling of CO2 storage in saline aquifers to elucidate fundamental processes, trapping mechanisms and sequestration partitioning. *Geological Society, London, Special Publications*. doi:10.1144/GSL.SP.2004.233.01.08

Kinzelbach, W. (1988). The Random Walk Method in Pollutant Transport Simulation. In *Groundwater Flow and Quality Monitoring* (pp. 227–245). Springer Netherlands.

Kinzelbach, W. (1990). Simulation of pollutant transport in groundwater with the random walk method. In *Groundwater Monitoring and Management* (pp. 265–279). Dresden: International Association of Hydrological Sciences 1990.

Kumar, A., Noh, M., Ozah, R., Pope, G., Bryant, S., Sepehrnoori, K., & Lake, L. (2005). Reservoir Simulation of CO2 Storage in Deep Saline Aquifers. *SPE Journal*, *10*(3). doi:10.2118/89343-PA

Li, L., Zhou, H., & Gómez-Hernández, J. J. (2011). Transport upscaling using multi-rate mass transfer in three-dimensional highly heterogeneous porous media. *Advances in Water Resources*, *34*(4), 478–489. doi:10.1016/j.advwatres.2011.01.001

Ma, X., Al-Harbi, M., Datta-Gupta, A., & Efendiev, Y. (2008). An efficient two-stage sampling method for uncertainty quantification in history matching geological models. *SPE Journal*, (August 2007), 24–27.

Mann, L., & Johnson, G. (1970). Predicted Results of Numeric Grid Models Compared with Actual Field Performance. *Journal of Petroleum Technology*, *22*(11). doi:10.2118/2572-PA

Mantilla, C. A., Nguyen, Q. P., & Srinivasan, S. (2008). Model Updating and Optimum Control for Water Coning: an application of Ensemble Kalman Filter. In *Proceedings of International Petroleum Technology Conference*. Kuala Lumpur, Malaysia: International Petroleum Technology Conference.

Mantilla, C., & Srinivasan, S. (2011). Feedback Control of Polymer Flooding Process Considering Geologic Uncertainty. In *Proceedings of SPE Reservoir Simulation Symposium*. Society of Petroleum Engineers. doi:10.2118/141962-MS

Metz, B., Davidson, O., De Coninck, H. C., Loos, M., & Meyer, L. A. (2005). *IPCC special report on carbon dioxide capture and storage: Prepared by working group III of the intergovernmental panel on climate change*. Cambridge UK; New York USA: Cambridge University Press.

Mohamed, L., Christie, M., & Demyanov, V. (2010a). Comparison of Stochastic Sampling Algorithms for Uncertainty Quantification. *SPE Journal*, *15*(1), 31–38. doi:10.2118/119139-PA

Mohamed, L., Christie, M., & Demyanov, V. (2010b). Reservoir Model History Matching With Particle Swarms: Variants Study. In *Proceedings of SPE Oil and Gas India Conference and Exhibition*. Society of Petroleum Engineers. doi:10.2118/129152-MS

Mohamed, L., Christie, M., Demyanov, V., Robert, E., & Kachuma, D. (2010). Application of Particle Swarms for History Matching in the Brugge Reservoir. In *Proceedings of SPE Annual Technical Conference and Exhibition*. Society of Petroleum Engineers. doi:10.2118/135264-MS

Montroll, E. W., & Weiss, G. H. (1965). Random Walks on Lattices. II. *Journal of Mathematical Physics*, *6*(2), 167. doi:10.1063/1.1704269

Naevdal, G., Johnsen, L., Aanonsen, S., & Vefring, E. (2005). Reservoir Monitoring and Continuous Model Updating Using Ensemble Kalman Filter. *SPE Journal*, *10*(1). doi:10.2118/84372-PA

Noh, M., Lake, L., Bryant, S., & Araque-Martinez, A. (2007). Implications of Coupling Fractional Flow and Geochemistry for CO2 Injection in Aquifers. *SPE Reservoir Evaluation & Engineering*, *10*(4). doi:10.2118/89341-PA

Nordbotten, J. M., Celia, M. a., & Bachu, S. (2004). Analytical solutions for leakage rates through abandoned wells. *Water Resources Research*, *40*(4). doi:10.1029/2003WR002997

Obi, E.-O. I., & Blunt, M. J. (2006). Streamline-based simulation of carbon dioxide storage in a North Sea aquifer. *Water Resources Research*, *42*(3). doi:10.1029/2004WR003347

Okabe, A., Boots, B., Sugihara, K., & Shui, S. N. (2000). *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams* (2nd ed.). New York: John Wiley & Sons, Ltd.

Oliveira, R., & Loucks, D. P. (1997). Operating rules for multireservoir systems. *Water Resources Research*, *33*(4), 839–852. doi:10.1029/96WR03745

Oliver, D. S., & Chen, Y. (2010). Recent progress on reservoir history matching: a review. *Computational Geosciences*, *15*(1), 185–221. doi:10.1007/s10596-010-9194-2

Oliver, D. S., Cunha, L. B., & Reynolds, A. C. (1997). Markov chain Monte Carlo methods for conditioning a permeability field to pressure data. *Mathematical Geology*, *29*(1), 61–91. doi:10.1007/BF02769620

Onuma, T., & Ohkawa, S. (2009). Detection of surface deformation related with CO2 injection by DInSAR at In Salah, Algeria. *Energy Procedia*, *1*(1), 2177–2184. doi:10.1016/j.egypro.2009.01.283

Park, A. H. A., Jadhav, R., & Fan, L. S. (2003). CO 2 mineral sequestration: Chemically enhanced aqueous carbonation of serpentine. *Canadian Journal of Chemical Engineering*, *81*, 885–890. doi:10.1002/cjce.5450810373

Park, H., Scheidt, C., Fenwick, D., Boucher, A., & Caers, J. (2013). History matching and uncertainty quantification of facies models with multiple geological interpretations. *Computational Geosciences*, *17*(4), 609–621. doi:10.1007/s10596-013-9343-5

Quinodoz, H. A. M., & Valocchi, A. J. (1993). Stochastic analysis of the transport of kinetically sorbing solutes in aquifers with randomly heterogeneous hydraulic conductivity. *Water Resources Research*, *29*(9), 3227–3240. doi:10.1029/93WR01039

Raghavan, R. (1993). *Well Test Analysis*. (L. Lake, Ed.) (p. 558). Englewood Cliffs, N.J.: Prentice Hall.

Romero, C. E., Carter, J. N., Gringarten, A. C., & Zimmerman, R. W. (2000). A Modified Genetic Algorithm for Reservoir Characterisation. In *Proceedings of International Oil and Gas Conference and Exhibition in China*. Society of Petroleum Engineers. doi:10.2118/64765-MS

Rushton, K., & Tomlinson, L. (1979). Possible mechanisms for leakage between aquifers and rivers. *Journal of Hydrology*, *40*, 49–65. Retrieved from http://www.sciencedirect.com/science/article/pii/0022169479900878

Sambridge, M. (1999a). Geophysical inversion with a neighbourhood algorithm-I. Searching a parameter space. *Geophysical Journal International*, *138*(2), 479–494. doi:10.1046/j.1365-246X.1999.00876.x

Sambridge, M. (1999b). Geophysical inversion with a neighbourhood algorithm-II. Appraising the ensemble. *Geophysical Journal International*, *138*(3), 727–746. doi:10.1046/j.1365-246x.1999.00900.x

Sen, M. K., Datta-Gupta, A., Stoffa, P. L., Lake, L. W., & Pope, G. A. (1995). Stochastic Reservoir Modeling Using Simulated Annealing and Genetic Algorithms. *SPE Formation Evaluation*, *10*(1). doi:10.2118/24754-PA

Stephen, K., Soldo, J., Macbeth, C., & Christie, M. (2005). Multiple Model Seismic and Production History Matching: A Case Study. In *Proceedings of SPE Europec/EAGE Annual Conference* (pp. 418–430). Society of Petroleum Engineers. doi:10.2118/94173-MS

Subbey, S., Christie, M., & Sambridge, M. (2004). Prediction under uncertainty in reservoir modeling. *Journal of Petroleum Science and Engineering*, *44*(1-2), 143–153. doi:10.1016/j.petrol.2004.02.011

Thiele, M., Batycky, R., & Blunt, M. (1997). A Streamline-Based 3D Field-Scale Compositional Reservoir Simulator. In *Proceedings of SPE Annual Technical Conference and Exhibition*. Society of Petroleum Engineers. doi:10.2118/38889-MS

Tompson, A. F. B., & Gelhar, L. W. (1990). Numerical simulation of solute transport in three-dimensional, randomly heterogeneous porous media. *Water Resources Research*, *26*(10), 2541–2562. doi:10.1029/WR026i010p02541

Underschultz, J., Otto, C., & Bartlett, R. (2005). Formation fluids in faulted aquifers: examples from the foothills of Western Canada and the North West Shelf of Australia, (2), 247–260. doi:10.1306/1060768H23171

Van Dop, H., Nieuwstadt, F. T. M., & Hunt, J. C. R. (1985). Random walk models for particle displacements in inhomogeneous unsteady turbulent flows. *Physics of Fluids*, *28*(6), 1639. doi:10.1063/1.864956

Voronoi, G. (1908). Nouvelles applications des paramètres continus à la théorie des formes quadratiques. Deuxième mémoire. Recherches sur les parallélloèdres primitifs. *Journal Für Die Reine Und Angewandte Mathematik (Crelle's Journal)*, *1908*(134), 198–287. doi:10.1515/crll.1908.134.198

Wallen, D. A., Raleigh, J. T., & Distefano, C. R. (1968). Mathematical Modeling in the Coyanosa Field. In *Proceedings of SPE Automation Symposium*. Society of Petroleum Engineers. doi:10.2118/2082-MS

Wang, Y., Li, G., & Reynolds, A. (2010). Estimation of Depths of Fluid Contacts by History Matching Using Iterative Ensemble-Kalman Smoothers. *SPE Journal*, *15*(2). doi:10.2118/119056-PA

Wardlaw, R., & Sharif, M. (1999). Evaluation of Genetic Algorithms for Optimal Reservoir System Operation. *Journal of Water Resources Planning and Management*, *125*(1), 25–33. doi:10.1061/(ASCE)0733-9496(1999)125:1(25)

Witherspoon, P. A., Mueller, T. D., & Donovan, R. W. (1962). Evaluation of Underground Gas-Storage Conditions in Aquifers Through Investigations of Groundwater Hydrology. *Journal of Petroleum Technology*, *14*(5). doi:10.2118/162-PA

Wright, I. (2007). The In Salah Gas CO. In *Proceedings of International Petroleum Technology Conference* (pp. 4–6). Society of Petroleum Engineers. doi:10.2523/11326-MS

Zafari, M., & Reynolds, A. (2007). Assessing the Uncertainty in Reservoir Description and Performance Predictions With the Ensemble Kalman Filter. *SPE Journal*, *12*(3). doi:10.2118/95750-PA

Zhang, Y., & Srinivasan, S. (2005). Markov chain Monte Carlo for modelling permeability variations in reservoirs. *Journal of Canadian Petroleum Technology*, *44*(3), 40–45.

Zhou, H., Gómez-Hernández, J. J., Hendricks Franssen, H.-J., & Li, L. (2011). An approach to handling non-Gaussianity of parameters and state variables in ensemble Kalman filtering. *Advances in Water Resources*, *34*(7), 844–864. doi:10.1016/j.advwatres.2011.04.014

Zhou, H., Gómez-Hernández, J. J., & Li, L. (2014). Inverse methods in hydrogeology: Evolution and recent trends. *Advances in Water Resources*, *63*, 22–37. doi:10.1016/j.advwatres.2013.10.014

Zweigel, P., Arts, R., Lothe, A. E., & Lindeberg, E. B. G. (2004). Reservoir geology of the Utsira Formation at the first industrial-scale underground CO_2 storage site (Sleipner area, North Sea). *Geological Society, London, Special Publications*, *233*(1), 165–180. doi:10.1144/GSL.SP.2004.233.01.11