AN INVESTIGATION OF THE MULTI-SCALE MIXED FINITE ELEMENT – STREAMLINE SIMULATOR AND ITS COUPLING WITH THE ENSEMBLE KALMAN FILTER

A Thesis

by

RAHUL MUKERJEE

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

May 2008

Major Subject: Petroleum Engineering

AN INVESTIGATION OF THE MULTI-SCALE MIXED FINITE ELEMENT – STREAMLINE SIMULATOR AND ITS COUPLING WITH THE ENSEMBLE KALMAN FILTER

A Thesis

by

RAHUL MUKERJEE

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

Chair of Committee,	Akhil Datta-Gupta
Committee Members,	Yalchin R. Efendiev
	Robert A. Wattenbarger
Head of Department,	Stephen A. Holditch

May 2008

Major Subject: Petroleum Engineering

ABSTRACT

An Investigation of the Multi-scale Mixed Finite Element-Streamline Simulator and Its Coupling with the Ensemble Kalman Filter. (May 2008)

Rahul Mukerjee, B.Tech (H), Indian Institute of Technology, Kharagpur

Chair of Advisory Committee: Dr. Akhil Datta-Gupta

The multi-scale mixed finite element method (MsMFEM) discussed in this work uses a two-scale approach, where the solutions to independent local flow problems on the fine grid capture the fine-scale variations of the reservoir model, while the coarse grid equations appropriately assimilate this information in the global solution. Temporal changes in porous media flow are relatively moderate when compared to the spatial variations in the reservoir. Hence, approximate global solutions by adaptively solving these local flow problems can be obtained with significant savings in computational time. The ensemble Kalman filter, used for real-time updating of reservoir models, can thus be coupled with the MsMFEM-streamline simulator to speed up the history-matching process considerably.

To my family, thank you.

TABLE OF CONTENTS

Page

Page

LIST OF FIGURES

FIGURE Page

1. INTRODUCTION

Fluid displacement in petroleum reservoirs is governed by processes and parameters occurring on multiple scales. State-of-the-art reservoir characterization, based on stochastic models, now allows geologists to generate multiple plausible models, each consisting of several million grid cells. However, the capabilities of conventional reservoir simulators lag behind those needed for high-resolution geo-models by several orders of magnitude. A major challenge in reservoir simulation is to bridge this gap, in order to provide fast and accurate flow simulations.

Traditionally, upscaling techniques have been used for a judicious construction of coarser models, such that simulations can be achieved within a reasonable time-frame. Alternatively, the fine scale information can be incorporated into the coarse model through some kind of a numerical sub-grid technique. The latter is the key to the multi-scale mixed finite element method (MsMFEM) discussed in this work[1,2]. Based on a two-scale approach, sub-grid computations are performed to estimate the fine-scale variations, while the coarse scale equations appropriately assimilate this information. Although the motivation of this work is to enhance computational efficiency, the multi-scale framework also lends itself easily to adaptivity. This is so because the temporal changes in porous media flow are moderate as compared to spatial variations.

This thesis follows the style of *SPE Journal*.

Hence, approximate solutions can be obtained on the fine-scale with significant savings in computational time. Adaptively updating the multi-scale basis functions based on a thresholding criterion which represents the flow dynamics approximately will be looked into. Also, the sub-grid computations are defined such that they can be easily parallelized leading to a further gain in efficiency.

An efficient and accurate forward model (reservoir simulator) can speed up the history-matching process considerably. The ensemble Kalman filter (EnKF) has been reported to be very efficient in terms of real-time updating of reservoir models to match the most current production data. The EnKF was introduced as an alternative to the traditional extended Kalman filter (EKF[15]). It is a sequential data assimilation method where the error statistics are predicted by the solving the Fokker-Planck equation (for the time evolution of the probability density function) using Monte-Carlo or ensemble integrations[3]. By doing so, it is possible to calculate statistical moments like mean and error covariances whenever such information is required.

This work uses a two-phase, incompressible multi-scale simulator as a forward model and will investigate ways of leveraging its inherent nature to improve the history-matching of reservoir models. Also, owing to the highly non-linear nature of reservoir flow physics, the EnKF can experience filter divergence, wherein the model parameters and states can attain non-physical values. Some recent approaches have been suggested to alleviate the inconsistency between the model state and parameters which arises at

assimilation steps, however, at a significant computational overhead. An alternative to these schemes using the multi-scale framework will be investigated.

Also, for practical field applications, the ensemble size needs to be kept small for computational efficiency. This leads to a poor approximation of the model error statistics. A novel approach proposed earlier to overcome this limitation uses streamlines to 'localize' model parameters, effectively eliminating irrelevant model parameters from the estimation process. One such approach uses the spatial information of the streamlines for the aforementioned purpose. This will also be investigated to assess the impact on the assimilation process.

2. MATHEMATICAL MODEL

2.1 Two-Phase Flow

Differential equations modeling immiscible, incompressible two-phase flow can be derived from the continuity equation of each phase:

$$\phi \frac{\partial S_i}{\partial t} + \nabla \cdot v_i = q_i$$

Ignoring gravity effects, Darcy's law relates the phase velocities to the gradient of phase pressures as follows:

$$v_i = -k\lambda_i(\nabla p_i - \rho_i g)$$

Here ϕ denotes porosity; S_i is the saturation of phase i; q_i is a source term representing wells; k is the rock permeability tensor, assumed to diagonal; is the mobility of phase i. The relative permeability models the reduced permeability experienced by one phase due to the presence of the other, andμ_i is the phase viscosity. Finally, ρ_i is the phase density and g is the gravity acceleration vector.

The two phases, oil (o) and water (w) fill the pore space completely so that $S_o + S_w = 1$. Phase pressures are related in terms of the capillary pressure, $p_{cow} = p_o - p_w$, which according to common practice, is a known function of water saturation, $p_{cow}(S_w)$ Summing up the Darcy equations for oil, water, we obtain the fractional flow formulation for two-phase flow as:

$$v = -k\lambda_t(\nabla p_o - f_w \nabla p_{cow}) + k(\lambda_o \rho_o + \lambda_w \rho_w) g$$

Here, f_w is the fractional flow function of water given by $f_w = \frac{\lambda_w}{\lambda_t}$, where, the total fluid mobility is given by $\lambda_t = \lambda_w + \lambda_o$. Summing the continuity equations for oil, water, and using that $S_o + S_w = 1$ yield:

$$v = -k[\lambda_t \nabla p - (\lambda_o \rho_o + \lambda_w \rho_w) g], \quad \nabla \cdot v = q$$

The above equations will be solved for global pressure p and total fluid velocity $v = v_o + v_w$.

Finally, the mass-transport for the water phase is obtained as:

$$v_w = f_w[v + k\lambda_o(\rho_w - \rho_o)g + k\lambda_o \frac{\partial p_{cow}}{\partial S_w} \nabla S_w]$$

However, gravity and capillarity effects will be ignored for the sake of simplicity.

The saturation equation is solved using a streamline method. Streamlines are flow-paths traced by a neutral particle being advected by a flow field such that the velocity is tangential to the streamline at every point. The streamlines are traced using Pollock's algorithm[5], building each streamline block-by-block. Thus, for every streamline, an initial saturation profile is obtained on an irregular time-of-flight grid. Finally, these initial-value problems are solved using a front-tracking method[9]. Since the emphasis of this work is to highlight rapid reservoir performance predictions using multi-scale methods, the transport equation will not be discussed in detail hence.

2.2 A Mixed Multi-scale FEM[6]

The mixed formulation for the elliptic pressure equation is derived for a reservoir with no-flow boundaries, and wells producing at constant rate.

Let Ω denote the reservoir domain and n be the outward pointing unit normal on $\partial\Omega$.

Define the function space:

$$V = H_0^{div}(\Omega) = \{v \in (L^2(\Omega))^d : \nabla \cdot v \in L^2(\Omega) \text{ and } v \cdot n = 0 \text{ on } \partial\Omega\}$$
$$P = L^2(\Omega)$$

Then the mixed formulation with no-flow boundary conditions $v \cdot n = 0$ on $\partial\Omega$ reads:

Find $(p, v) \in L^2(\Omega) \times H_0^{div}(\Omega)$ such that:

$$\int_\Omega (k\lambda_t)^{-1} v \cdot w \, dx - \int_\Omega p \nabla \cdot w \, dx = 0,$$
$$\int_\Omega l \nabla \cdot v \, dx = \int_\Omega q l \, dx,$$

for all $w \in H_0^{div}(\Omega)$, and $l \in L^2(\Omega)$.

Here *V* and *P* are (finite-dimensional) function spaces for velocity and pressure respectively. Now, letting $\{\Psi_i\}$ and $\{\Phi_k\}$ be bases for *V* and *P* respectively, we obtain approximations $v = \sum v_i \Psi_i$ and $p = \sum p_k \Phi_k$, where the coefficients $\boldsymbol{v} = \{v_i\}$ and $p = \{p_k\}$ solve a linear system of the form

$$\begin{bmatrix} B & C \\ C^T & 0 \end{bmatrix} = \begin{bmatrix} 0 \\ q \end{bmatrix},$$

where $B = \{b_{ij}\}$, $C = \{c_{ij}\}$ and $q = \{q_k\}$ are defined by

$$b_{ij} = \int_\Omega \Psi_i \cdot (\lambda k)^{-1} \Psi_j \, dx, \quad c_{ik} = \int_\Omega \Phi_k \nabla \cdot \Psi_i \, dx, \quad q_k = \int_\Omega \Phi_k q \, dx$$

2.2.1 The MsMFEM Basis Functions[1, 2]

The domain Ω is divided into polyhedral (coarse grid) elements $\mathrm{T} = \{T\}$. In the MsMFEM, each basis function is associated with an interface $\Gamma_{ij} = \partial T_i \cap \partial T_j$ between two coarse grid blocks T_i and T_j. For each such interface, an interface flux $\psi_{ij} = -K\lambda_t \nabla \phi_{ij}$ is defined, where ϕ_{ij} is determined by solving the "pressure equations" numerically on a fine-scale sub-grid within the coarse blocks T_i and T_j:

$$(\nabla \cdot \psi_{ij})|_{T_i} = \{1/|T_i|, \text{ if } \int_{T_i} q\,dx = 0,$$

$$= \{q \Big/ \int_{T_i} q\,dx, \text{ otherwise.}$$

$$(\nabla \cdot \psi_{ij})|_{Tj} = \{-1/|T_j|, \text{ if } \int_{T_j} q\,dx = 0,$$

$$= \{-q \Big/ \int_{T_j} q\,dx, \text{ otherwise.}$$

with some compatible boundary conditions:

$\psi_{ij} \cdot n = 0$ *on* $\partial T_i \cup \partial T_j$ and $\psi_{ij} \cdot n_{ij} = v_{ij}$ *on* Γ_{ij} .

Here n is the outward unit normal on $\partial(T_i \cup \Gamma_{ij} \cup T_j)$ and n_{ij} is the unit normal pointing from T_i to T_j. The corresponding approximation space for the total Darcy velocity v is now spanned by the basis functions $V^{ms} = span\{\psi_{ij} : meas(\Gamma_{ij}) > 0\}$. These basis functions can be seen as generalizations of the lowest order Raviart-Thomas basis functions in a standard mixed method[10]. **Fig. 1** illustrates the x-velocity basis functions in two different cases. We would like to note that the mixed multi-scale basis functions can also be constructed via oscillatory boundary conditions or source terms[7, 8].

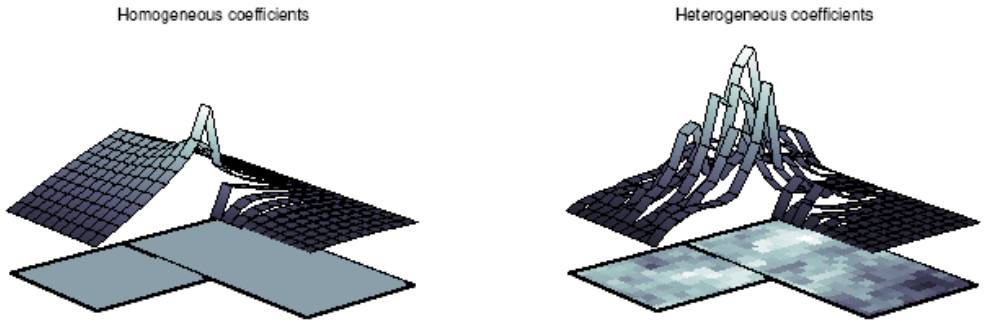


Fig.1-The x-component of the velocity basis function associated with an edge between two cells of different size for homogeneous, heterogeneous permeability fields respectively.

The local pressure solutions ϕ_{ij} do not appear explicitly in the mixed formulation, and are hence only used to generate the basis functions ψ_{ij}. Since it is important that all basis functions be mass conserving, the sub-grid problems must be solved using a mass conservative method e.g. a suitable mixed FEM or a finite volume method[6]. The particular choice of the method depends on the local grid structure. For e.g. if the sub-grid is to be discretized using a finite-volume method, then a two-point flux approximation can be used if *k* is a diagonal tensor, and the grid is orthogonal, whereas a multi-point flux approximation or mimetic finite difference methods can be used for non-orthogonal grids.

The basis functions are time-dependent since they depend on λ_t. This indicates that one has to regenerate the basis functions for every time step. However, it will be usually sufficient to regenerate a small portion of the basis functions at each time step since the total mobility varies significantly only in the vicinity of the propagation saturation front.

2.2.2 Computational Considerations

For the MsMFEM, several steps can accelerate the computation time considerably. Firstly, instead of regenerating the multi-scale basis functions at every time step, they should be updated only in regions where the total mobility has changed significantly since the previous update and has significant variations within the coarse block. Aarnes[6] observed that the accuracy obtained when updating a small fraction of the basis functions (mostly near saturation fronts) in each time step is almost the same as when updating all the basis functions. Since the calculation of the basis function dominates the computation time in MsMFEM, adaptive calculation of the basis functions can accelerate the solution procedure for the pressure equation significantly.

Another way of accelerating the MsMFEM computations is by parallel computing. Computation of the basis functions is done on a block by block basis, and has an inherent parallelism. These computations can therefore be assigned to individual processors.

2.3 Numerical Examples

To evaluate the accuracy and efficiency of the MsMFEM-streamline simulator, two synthetic test-cases were considered:

1) A 2D reservoir model:
    a. 50 x 50 x 1 grid cells, log-normally correlated permeability field.
    b. 5-spot pattern; 4 production wells, 1 injection well (**fig. 2**).

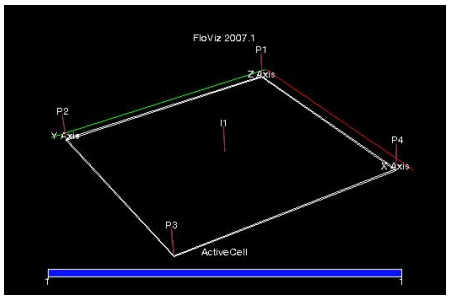


Fig.2-Well locations for the 2D model

2) A 3D reservoir model:
    a. 128 x 64 x 4 grid cells, log-normally correlated permeability field.
    b. 22 production wells, 8 injection wells (**fig. 3**).

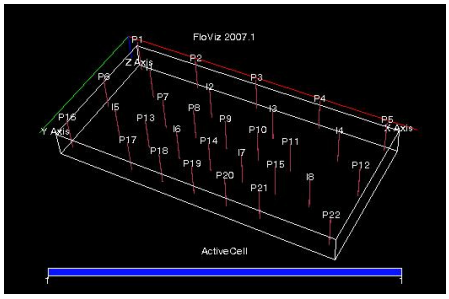


Fig.3-Well locations for the 3D model

3) All the wells in the above models are rate constrained; Total reservoir fluid rate constraint for producers, and constant water injection rates.
4) Coarse grids of varying sizes are used to assess the impact of the MsMFEM against reference solutions obtained using the Two-Point Flux (TPF) method.
    a. For 2D model, 5 x 5 x 1 and 25 x 25 x 1 sized coarse grids are used.
    b. For 3D model, 16 x 8 x 1 and 32 x 16 x 1 sized coarse grids are used.
5) Water-cuts at the producing wells, and layer-wise water saturation profiles are used to compare results.

End-point mobility ratios of 10.0, 0.80 and 0.10 are used for varying the degree of non-linearity in the fluid displacement process. Results are also compared when basis function computations are performed only during the first pressure update step, and they are used to obtain approximate solutions for the subsequent pressure update steps.

The reference solution obtained using the TPF formulation refers to the Implicit in Pressure, Explicit in Saturation (IMPES) formulation used in standard streamline-based finite difference reservoir simulators such as Frontsim. We would like to note that the results can differ when the transport equation is solved with finite volume methods because of numerical diffusion[16]. However, for our purposes, the use of streamline methods is important because they are very fast and have advantages in association with the ensemble Kalman filter.

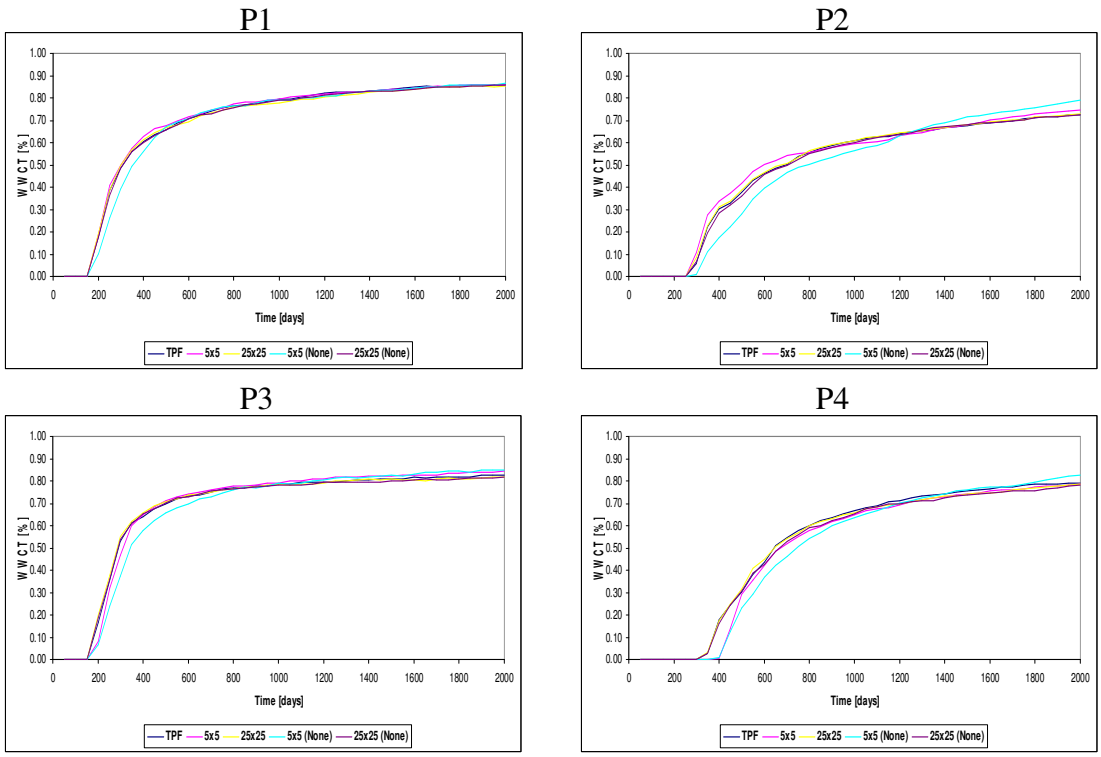


Fig.4-WWCT at different producers, (TPF, 5x5, 25x25, 5x5 (None), 25x25 (None)), 2D, M=10.0

Fig. 4 above shows the water-cuts at different producers for the 2D 5-spot example. Comparisons are made for coarse grids consisting of 5 x 5 x 1 and 25 x 25 x 1 cells. Results are also compared when computing the basis functions (BFs) only for the first pressure update step as well as when the BFs are updated at every pressure update step. Similar comparisons are made for the 3D example in **fig. 5** by using coarse grids comprising of 16 x 8 x 1 and 32 x 16 x 1 cells (plots of the corresponding oil saturation profiles are included in Appendix A). It can be seen that the production responses show only subtle variations when compared to the results obtained using the two-point flux (TPF) formulation both for the 2D as well as the 3D examples. This is true even when the velocity basis functions are computed for the very first pressure update step.

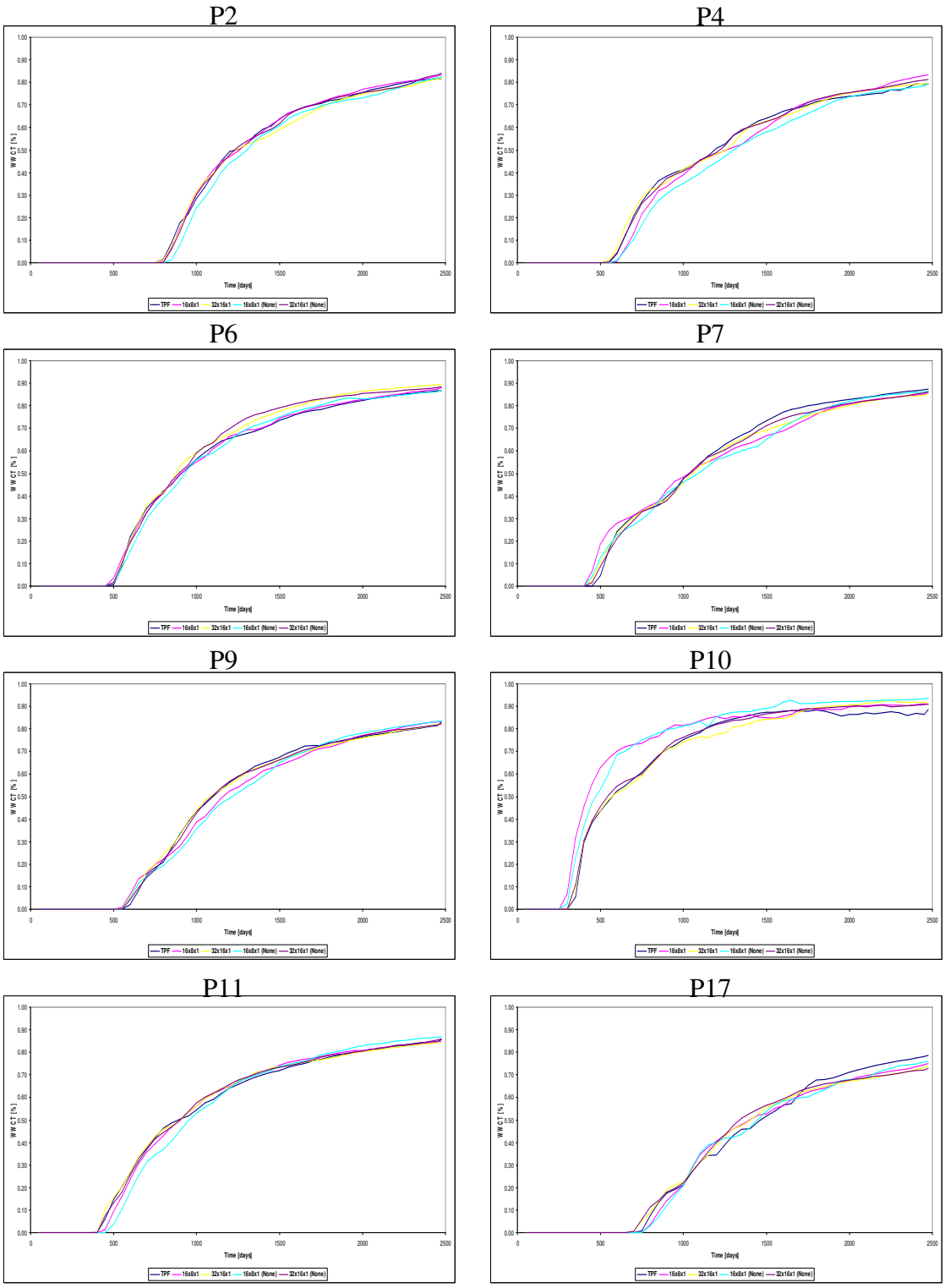


Fig.5-WWCT at different producers, (TPF, 16x8x1, 32x16x1, 16x8x1 (None), 32x16x1 (None)), 3D, M=10.0

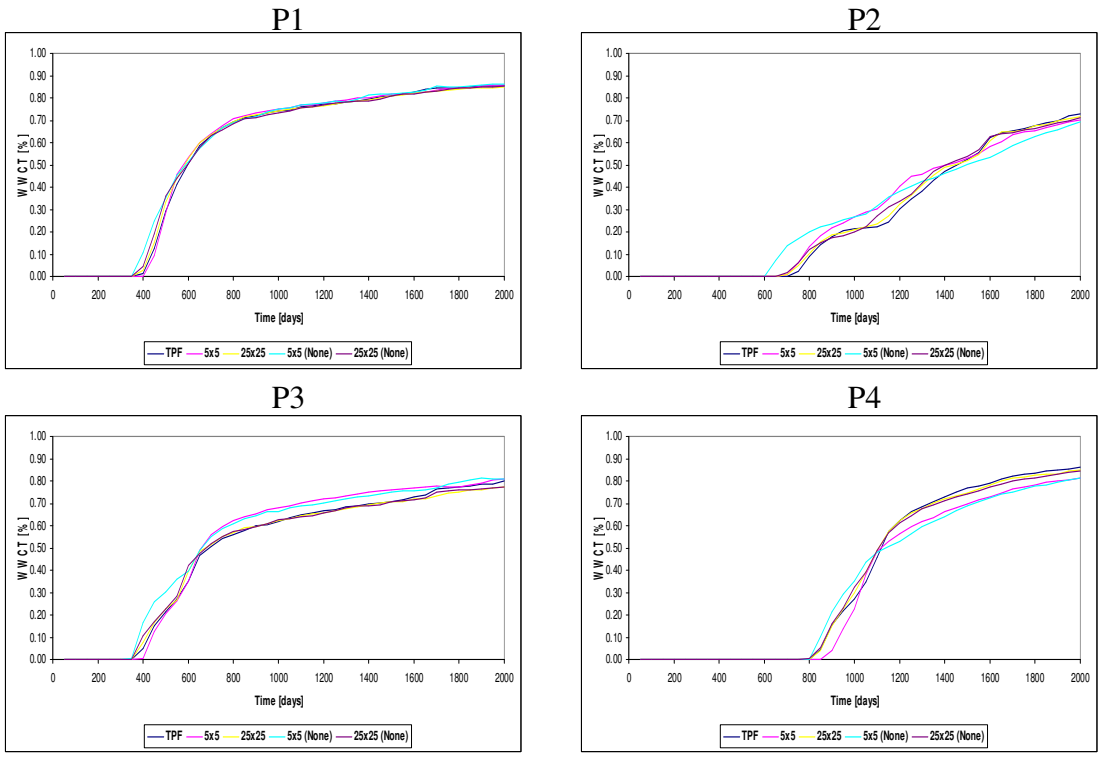


Fig.6-WWCT at different producers, (TPF, 5x5, 25x25, 5x5 (None), 25x25 (None)), 2D, M=0.80

Fig. 6 above shows similar comparisons as **fig. 4**. The results shown above correspond to a slightly favorable mobility ratio (M=0.80) unlike the previous case where a highly unfavorable mobility ratio (M=10.0) was used. Similar comparisons are shown for the 3D example in **fig. 7**.

M = 0.80

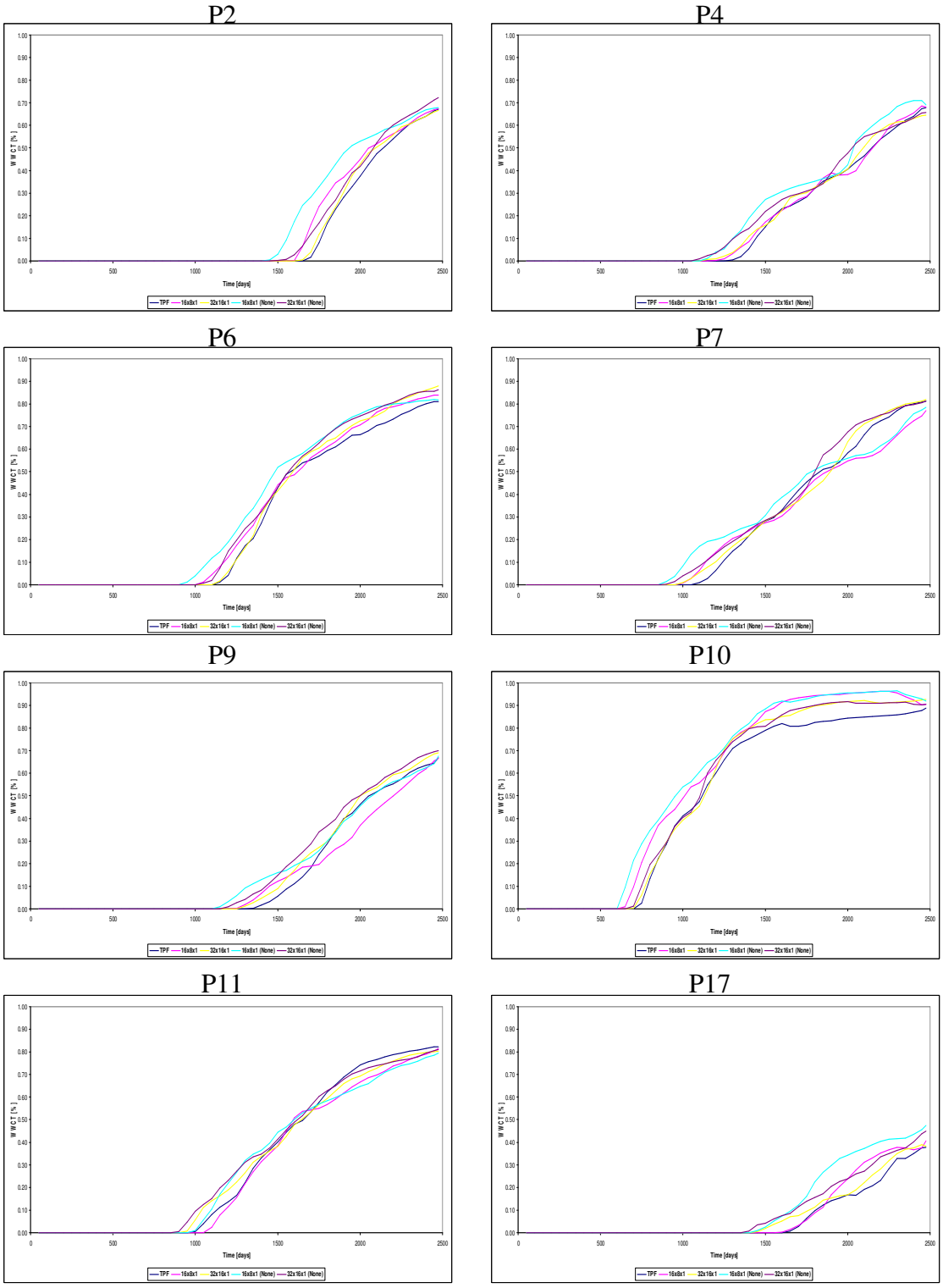


Fig.7-WWCT at different producers, (TPF, 16x8x1, 32x16x1, 16x8x1 (None), 32x16x1 (None)), 3D, M=0.80

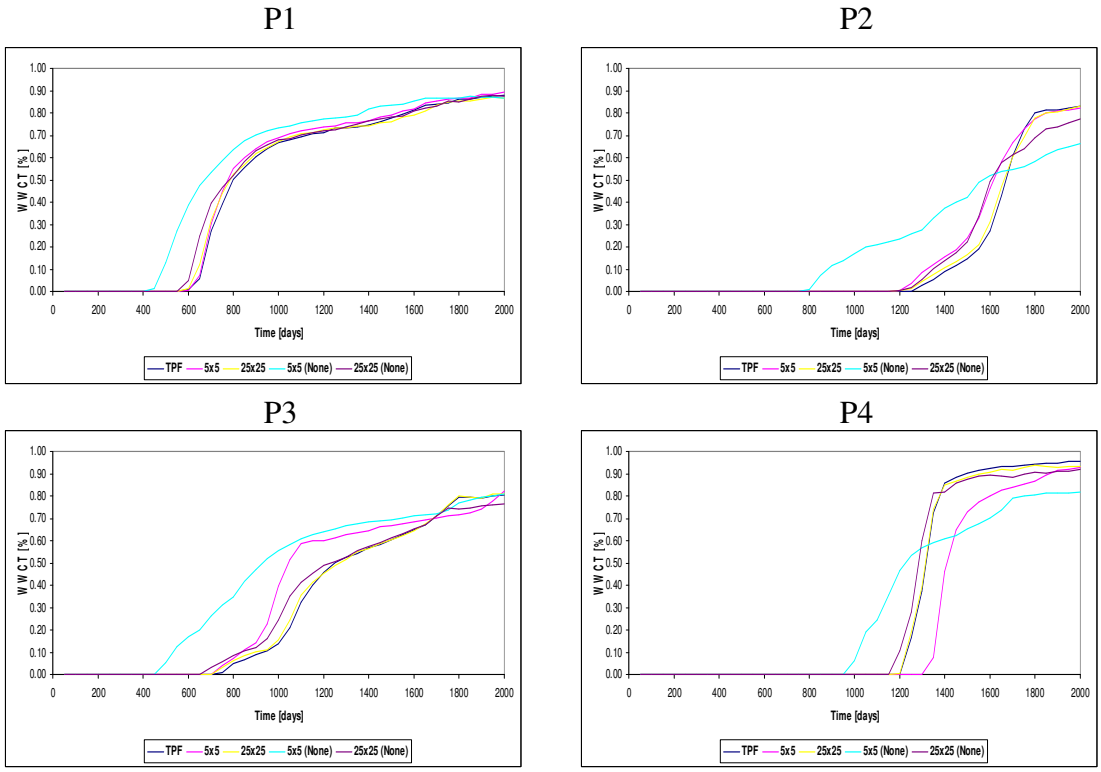


Fig.8-WWCT at different producers, (TPF, 5x5, 25x25, 5x5 (None), 25x25 (None)), 2D, M=0.10

Figs. 8 above, as well as **fig. 9** compare the production responses corresponding to a highly favorable mobility ratio (piston-like displacement). Temporal changes in the velocity basis functions as well as the saturation profiles can make a significant impact in this case. This is evident by comparing the water-cuts when the basis functions are computed only for the first pressure update step (shown with cyan color above). It is also noteworthy that similar results for the 25 x 25 x 1 coarse grid are closer to the reference solution in comparison to those for the 5 x 5 x 1 coarse grid.

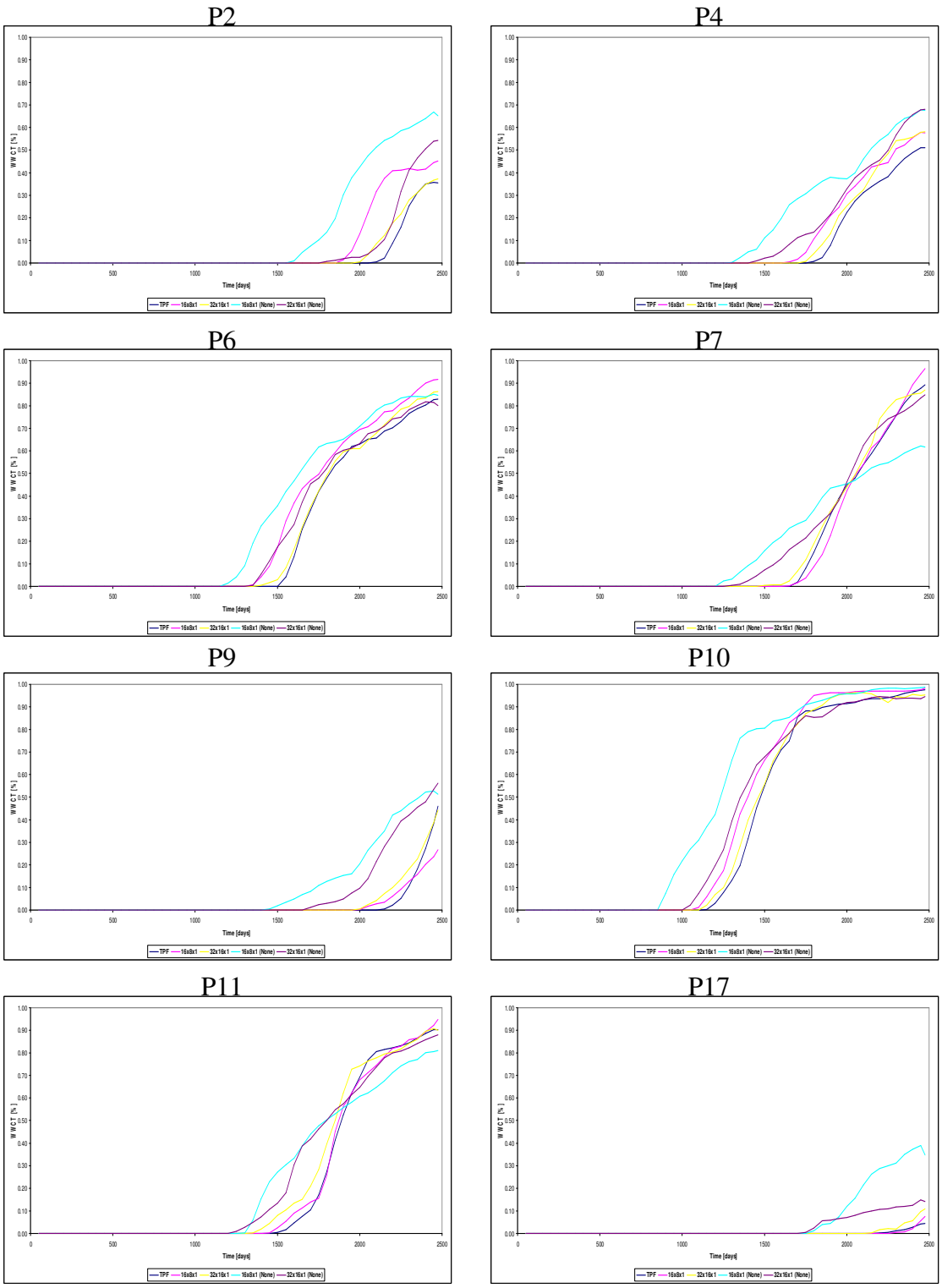


Fig.9-WWCT at different producers, (TPF, 16x8x1, 32x16x1, 16x8x1 (None), 32x16x1 (None)), 3D, M=0.10

2.4 Comments

For both the 2D as well as the 3D cases, the production responses and the saturation profiles are in close agreement in the case of unfavorable, slightly favorable fluid displacement (M=10.0, 0.80). A comparison of the saturation profiles (Appendix A) also shows close resemblance for different sized coarse grids and updating of the velocity basis functions. In these cases, viscous fingering causes inefficient fluid displacement through the reservoir. This however causes the velocity and saturations to vary smoothly.

For a favorable mobility ratio (M=0.10), the approximate solutions obtained are not very accurate due to the highly non-linear nature of the fluid displacement process. In this case, the fluid displacement is stable and a fairly sharp front separates the mobile oil and water phases. Hence, the underlying velocity and saturation profiles change significantly with time.

The simulations performed above were timed.

For the 2D / 3D cases:

Frontsim: 40s / 7m35s

TPF: 18s / 6m 03s

MsMFEM:

5 x 5 / 16 x 8 x 1:	35s / 10m12s	5 x 5 / 16 x 8 x 1 (None):	16s / 3m40s
25 x 25 / 32 x 16 x 1:	35s / 10m45s	25 x 25 / 32 x 16 x 1 (None):	17s / 3m48s

The simulation times when updating all the basis functions are of the same order of magnitude as the TPF case, or more. Depending on the dimensions of the local flow problems being solved for the basis functions, the MsMFEM computation times can vary. However, when basis functions are updated only during the first time-step, the computation time can be reduced significantly.

Hence, significant savings in computational time can be made in scenarios with unfavorable / slightly favorable mobility ratios without sacrificing the accuracy of the results significantly. Also, an adaptive approach for selective updating basis functions in the case of favorable mobility ratios can reduce the computational times noticeably.

3. PARAMETER ESTIMATION USING THE ENSEMBLE KALMAN FILTER

3.1 Background

The Kalman filter addresses the general problem of trying to estimate the state $x \in \Re^n$ of a discrete-time, controlled process that is governed by the linear stochastic difference equation: $x_k = Ax_{k-1} + w_{k-1}$

with a measurement $z \in \Re^m$ that is:

$$z_k = Hx_k + v_k$$

The random variables w_k, v_k represent the process and measurement noise (respectively). They are assumed to be independent (of each other), white, and with normal probability distributions:

$$p(w) \cong N(0, Q)$$
$$p(v) \cong N(0, R)$$

In practice, the process noise covariance *Q*, and measurement noise covariance *R* matrices might change with each time step or measurement, however here they are assumed to be constant.

The *n* x *n* matrix, *A*, in the difference equation relates the state at the previous time step, *k-1*, to the state at the current step, *k*, in the absence of either a driving function or process noise. It should be noted that in practice *A* might change with each time step. The *m* x *n* matrix, *H*, in the measurement equation relates the state to the measurement, z_k . In practice, *H* might change with each time step.

3.2 The Computational Origins of the Filter[4]

Define $\hat{x}_k^- \in \Re^n$ (note the super minus) to be the *apriori* state estimate at step *k*, given knowledge of the process prior to step *k*, and $\hat{x}_k \in \Re^n$ to be the *aposteriori* state estimate at step *k*, given measurement z_k .

The *apriori*, and *aposteriori* estimate errors can be defined as:

$$e_k^- \equiv x_k - \hat{x}_k^-, and$$
$$e_k \equiv x_k - \hat{x}_k$$

The *apriori* estimate error covariance is then:

$$P^-_k = E[e_k^-(e_k^-)^T]$$

And the *aposteriori* estimate error covariance is:

$$P_k = E[e_k(e_k)^T]$$

The goal is to find an equation that computes an *aposteriori* state estimate x_k^-, as a linear combination of the *apriori* estimate $z_k - H\hat{x}_k^-$, and a weighted difference between the actual measurement, and a measurement prediction, x_k , as:

$$\hat{x}_k = \hat{x}_k^- + K(z_k - H\hat{x}_k^-)$$

For a general linear system, the above equation can be arrived at using different approaches viz. the Minimum Mean Square Error (MMSE), Conditional Expectation (Bayes' theorem) etc.

The difference $z_k - H\hat{x}_k^-$ in the above equation is called the measurement *innovation* or the *residual*. The residual reflects the discrepancy between the predicted measurement, $H\hat{x}_k^-$, and the actual measurement, z_k. A residual of zero means that the two are in complete agreement.

The *n* x *m* matrix *K*, is the *Kalman Gain*, which is chosen such that the *aposteriori* error covariance is minimized. One form of *K* that minimizes the *aposteriori* error covariance is given by :

$$K_k = P_k^- H^T (HP_k^- H^T + R)^{-1}$$
$$= \frac{P_k^- H^T}{HP_k^- H^T + R}$$

From the above equation, it can be seen that as the measurement error covariance, *R*, approaches zero, the gain *K* weights the residual more heavily. Specifically, $\lim_{R_k \to 0} K_k = H^{-1}$ On the other hand, as the *apriori* estimate error covariance approaches zero, the gain weights the residual less heavily. Specifically, $K_k = 0$

Another way of thinking about the weighting by *K* is that as the measurement error covariance *R* approaches zero, the actual measurement z_k is "trusted" more and more, while the predicted measurement, $H\hat{x}_k^-$ is trusted less and less. On the other hand, as the *apriori* estimate error covariance, P_k^- approaches zero, the actual measurement, z_k, is trusted less and less, while the predicted measurement $H\hat{x}_k^-$ is trusted more and more.

3.3 The Ensemble Kalman Filter[3]

As mentioned above, the Kalman filter is an estimation technique for a process governed by a *linear* stochastic differential equation. But what if the process to be estimated and (or) the measurement relationship to the process is non-linear?

A Kalman filter that linearizes about the current mean and covariance is referred to as an Extended Kalman filter (EKF). This is akin to a Taylor series expansion about the current estimate by using the partial derivatives of the process and measurement functions to compute estimates even in the face of non-linear relationships. However, the linearized error covariance is only approximate (naturally). It results from a linearization of an equation which references infinitely many higher order statistical moments. This can lead to significantly erroneous results for strongly non-linear dynamics. It has been demonstrated that the EKF can result in severely under / over estimated error covariances causing filter divergence.

The Ensemble Kalman Filter (EnKF) is an alternative approach to the EKF. It is a sequential data assimilation method which predicts the time evolution of the error statistics using Monte-Carlo or Ensemble integrations. By integrating an ensemble of model states forward in time, it is possible to calculate statistical moments like mean and error covariance whenever such information is required. Thus, all the statistical information about the predicted model state is contained in the ensemble.

3.3.1 Representation of Error Statistics

The error covariance matrices for the predicted and the analyzed estimate, P_k^-, P_k , defined earlier are:

$$P_k^- = \overline{(\hat{x}_k^- - \overline{\hat{x}_k^-})(\hat{x}_k^- - \overline{\hat{x}_k^-})^T}$$

$$P_k = \overline{(\hat{x}_k - \overline{\hat{x}_k})(\hat{x}_k - \overline{\hat{x}_k})^T}$$

where the overline denotes an expectation value, $\hat{x}_k$ is the model state vector at a particular time and the superscript – sign indicates the *apriori* state. However, since the true state is not known, it is convenient to consider ensemble covariance matrices around the ensemble mean, $\overline{\hat{x}_k}$.

Since the covariances defined above are defined as ensemble averages, there exist infinitely many ensembles with the error covariances equal to P_k^-, P_k . Hence, instead of the full covariance matrix, the same error statistics are represented using an appropriate ensemble of model states. Given an ensemble of limited size, *N*, the above equations provide an approximation to the error covariance matrix. As *N* increases, the errors in the representation will decrease proportional to $1/\sqrt{N}$. In practice, an ensemble size of 100 – 500 members should represent the error covariance with reasonable accuracy.

3.3.2 Prediction of Error Statistics

For linear dynamics, and a Gaussian initial probability density, the estimated probability density will be completely characterized by its mean and covariance matrix. But for a non-linear model, the mean and covariance matrices are not, in general, sufficient to

completely characterize the process. They do, however, determine the mean path and the dispersion about the path. Thus, a correct representation of the process dynamics still enables a correct approach towards the true solution using only the first two statistical moments.

A standard practice is to calculate an initial best guess conditioned to available data and statistics. An ensemble of initial states can then be sampled with the mean as the best guess and the variance specified on the basis of knowledge of the uncertainty in the first guess initial state. The covariance or smoothness of the ensemble should reflect the true scales of the system.

3.3.3 An analysis scheme

The ensemble covariances are calculated around the ensemble mean as shown above. It is also essential that the observations are treated as random variables having a mean equal to the first guess observations (observed data), and covariance equal to *R*. Thus, we start by defining an ensemble of observations:

$$z_{k\,j} = z_k + \varepsilon_j$$

where *j* ranges from 1 → *N,* the number of ensembles. The measurement error covariance is then given by:

$$R = \overline{\varepsilon\varepsilon^T}$$

The analysis step for the EnKF consists of the following updates performed on each of the model state ensemble members :

$$\hat{x}_{k\,j} = \hat{x}_{kj}^{\,-} + K(z_{k\,j} - H\hat{x}_{k}^{\,-}{}_{j})$$

K is the Kalman gain defined as:

$$K = P_k^{-} H^T (HP_k^{-} H^T + R)^{-1}$$

The update equation implies that the relation between the analyzed and predicted ensemble mean is identical to the relation between the analyzed and predicted state in the classical Kalman filter:

$$\overline{\hat{x}_k} = \overline{\hat{x}_k^{\,-}} + K(z_k - H\overline{\hat{x}_k^{\,-}})$$

3.4 Extensions to the EnKF

The above analysis scheme, using only the parameters (permeability, porosity) is inconsistent since the Kalman filter addresses the combined parameter-state estimation problem. Ideally, the state variables (pressure, water saturation) and the parameters should be updated simultaneously. However, an ad-hoc inclusion of the state into the analysis scheme can lead to material-balance errors, and unfeasible values of the state variables (for e.g. $S_w > 1$). To improve upon the existing framework, some extensions to the algorithm were implemented.

3.4.1 Streamline-based Covariance Localization[12]

Cross-covariance calculations relate the reservoir parameters to the dynamic production response. Thus, the larger the ensemble size, the greater the accuracy with which these calculations can be made. For practical purposes, the ensemble size has to be limited due to computational considerations. This, combined with the fact that reservoir fluid displacement is a highly non-linear process, causes the reservoir parameters to most likely be under / over estimated. This is referred to, in the literature, as overshooting problems. This problem can be alleviated to an extent by eliminating spurious unknowns from the update process.

Streamlines transport fluids from injection wells to the producers in streamline-based simulators. Thus, their trajectories within the reservoir have a direct impact on the dynamic production response of the reservoir. One approach to eliminate irrelevant parameters from the update process is to consider only the cells which are traversed by streamlines during the update step and disregard the ones that are not. In order to do so, cells which are not visited by any streamlines over the entire ensemble are not considered in the update process.

Further, the multi-scale basis functions corresponding to the coarse cells visited by streamlines are re-computed. This approach can be very effective when the streamlines tend to concentrate only along streaks of high permeability in the reservoir. By

selectively re-computing the basis functions only for the updated reservoir parameters, a more rigorous history-match is expected as compared to the standard EnKF.

3.4.2 Confirming Option[11]

The Kalman update of the state vector is done as described in the analysis scheme. The current time step is then re-run using the updated model parameters (PERM), and this updated ensemble is used to resume the assimilation process. By doing so, a consistency between the model parameters (PERM) and the state variables (PRES, SWAT) is ensured. An inconsistency between the parameters and state still exists however, for the previous update step.

The confirming option effectively tries to reduce the original problem of combined parameter and state estimation to only a parameter estimation problem. However, the approach followed is an engineering solution lacking a theoretical proof. Also, the approach needs to be tested rigorously before it can be adopted as a worthy substitute to updating the state variables.

3.4.3 Confirming Option (Selectively updating MsMFEM basis functions)

The confirming option doubles the computational overhead of the update process since every update step has to be run twice. To counter this, the MsMFEM basis functions lend an efficient alternative. The basis functions are selectively updated[13] based on a simple thresholding criterion and the performance of the confirming option is then

evaluated by updating 100%, 50% and 25% of all the basis functions during the confirming step. Thus by selectively updating the basis functions, a gain in computational efficiency is expected when compared to the regular confirming step, albeit with the benefits of the former vis-à-vis the estimation of reservoir parameters.

In this case, before resuming the assimilation process, the basis functions for the model are re-computed using the updated model parameters. It is worth noting that the values of the state from the last update step are used for re-computing the basis functions. However, the results obtained using this modification are comparable or better to those obtained using the regular confirming option mentioned earlier, but at a significantly lower computational overhead.

3.5 Numerical Examples

The 2D, 3D synthetic cases described in an earlier section were used to evaluate the coupling of the EnKF with MsMFEM-Streamline simulator.

1) End-point mobility ratios of 10.0 and 0.10 are used for varying the degree of non-linearity in the fluid displacement process.
2) Ensembles of 98 members, generated using sequential Gaussian simulation (sgsim), are used for both 2D as well as 3D cases.
3) Natural logarithms of permeabilities are used for the Kalman update process.
4) Accuracy of the results is evaluated by comparing the spread of the final ensemble production response about that of the true model response.
5) Results are compared using the following options:
    a. EnKF.
    b. EnKF + Covariance localization.
    c. EnKF + Confirming option.
    d. EnKF + Confirming option (w/ thresholding of basis functions (BFs)).

Fine grid : 50 x 50 x 1, Coarse grid : 10 x 10 x 1, Mobility Ratio = 10.0

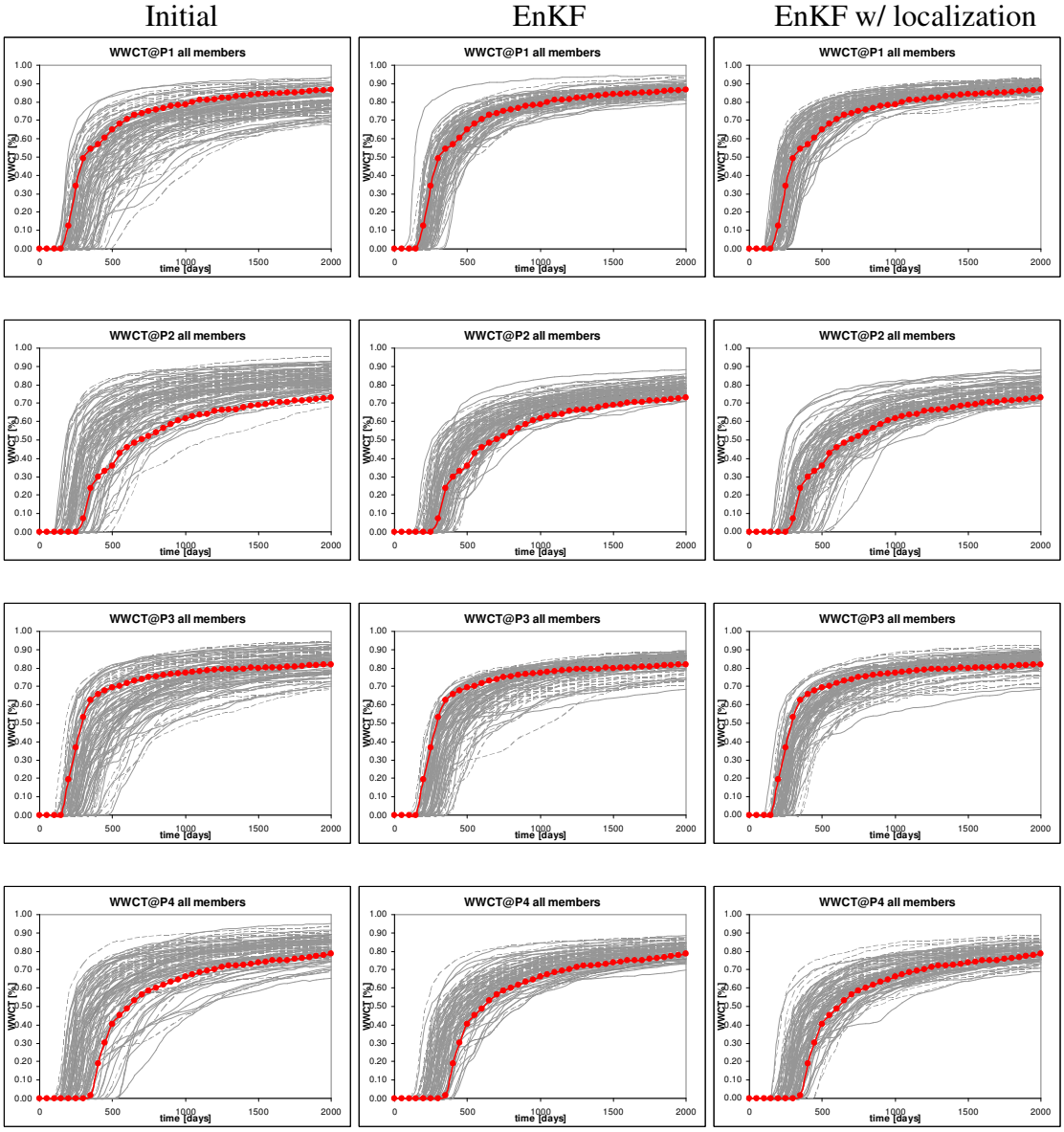


Fig.10-Ensemble spread for the initial and final models obtained with EnKF, EnKF w/ localization for different producers w.r.t. TRUE, 2D, M=10.0

Fig. 10 compares the initial spread of the ensemble water-cut to the final spread using the standard EnKF and EnKF with covariance localization using streamlines.

Fine grid : 50 x 50 x 1, Coarse grid : 10 x 10 x 1, Mobility Ratio = 10.0

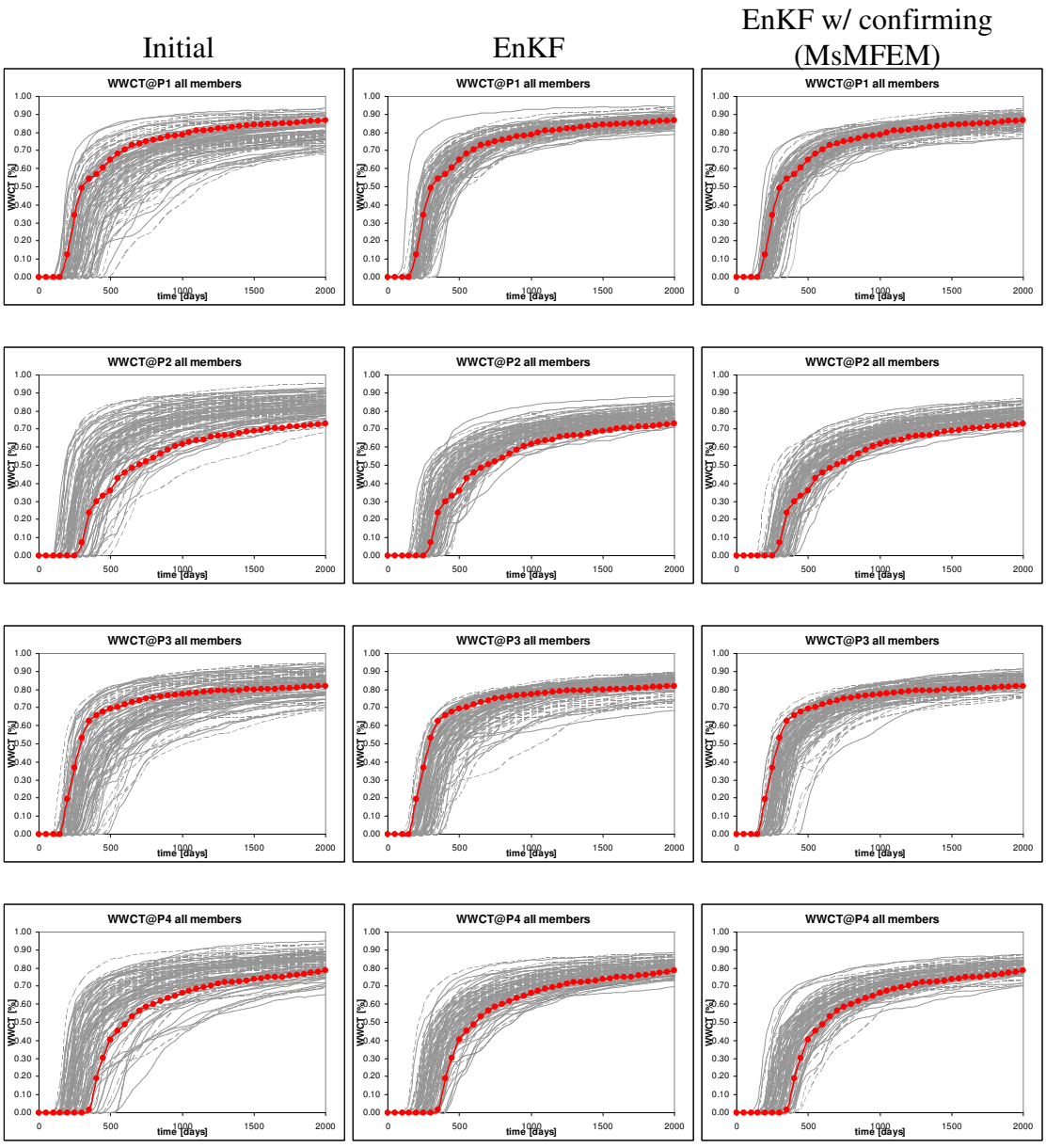


Fig.11-Ensemble spread for the final models obtained with EnKF, EnKF w/ confirming options, EnKF w/ confirming option (MsMFEM) for different producers w.r.t. TRUE, 2D, M=10.0

Fig. 11 compares the initial and final spreads in ensemble water-cuts using the standard EnKF and EnKF with confirming option while updating all the BFs.

Fine grid : 50 x 50 x 1, Coarse grid : 10 x 10 x 1, Mobility Ratio = 0.10

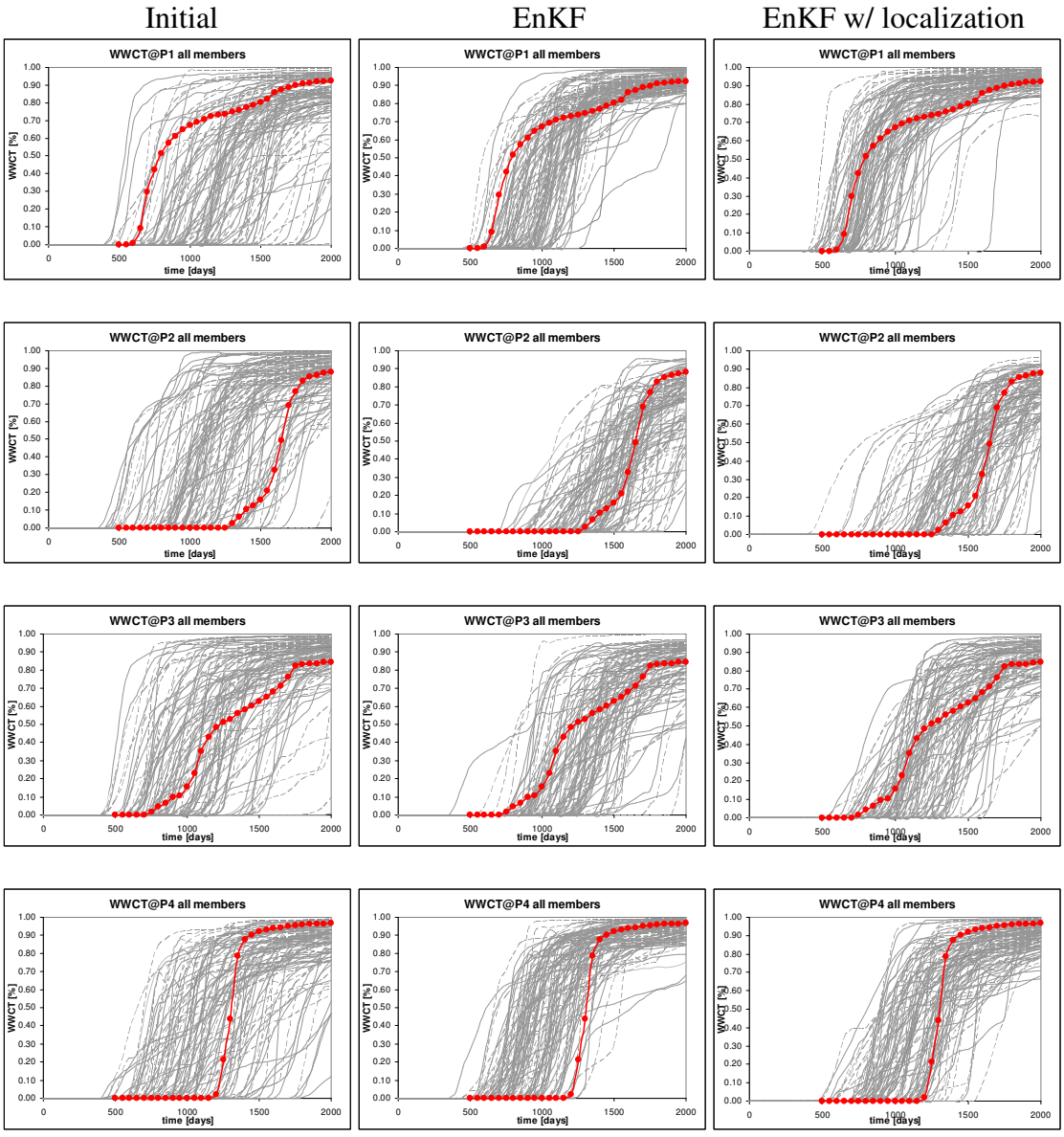


Fig.12-Ensemble spread for the initial and final models obtained with EnKF, EnKF w/ localization for different producers w.r.t. TRUE, 2D, M=0.10

Fig. 12 compares the initial spread of the ensemble water-cut to the final spread using the standard EnKF and EnKF with covariance localization using streamlines.

Fine grid : 50 x 50 x 1, Coarse grid : 10 x 10 x 1, Mobility Ratio = 0.10

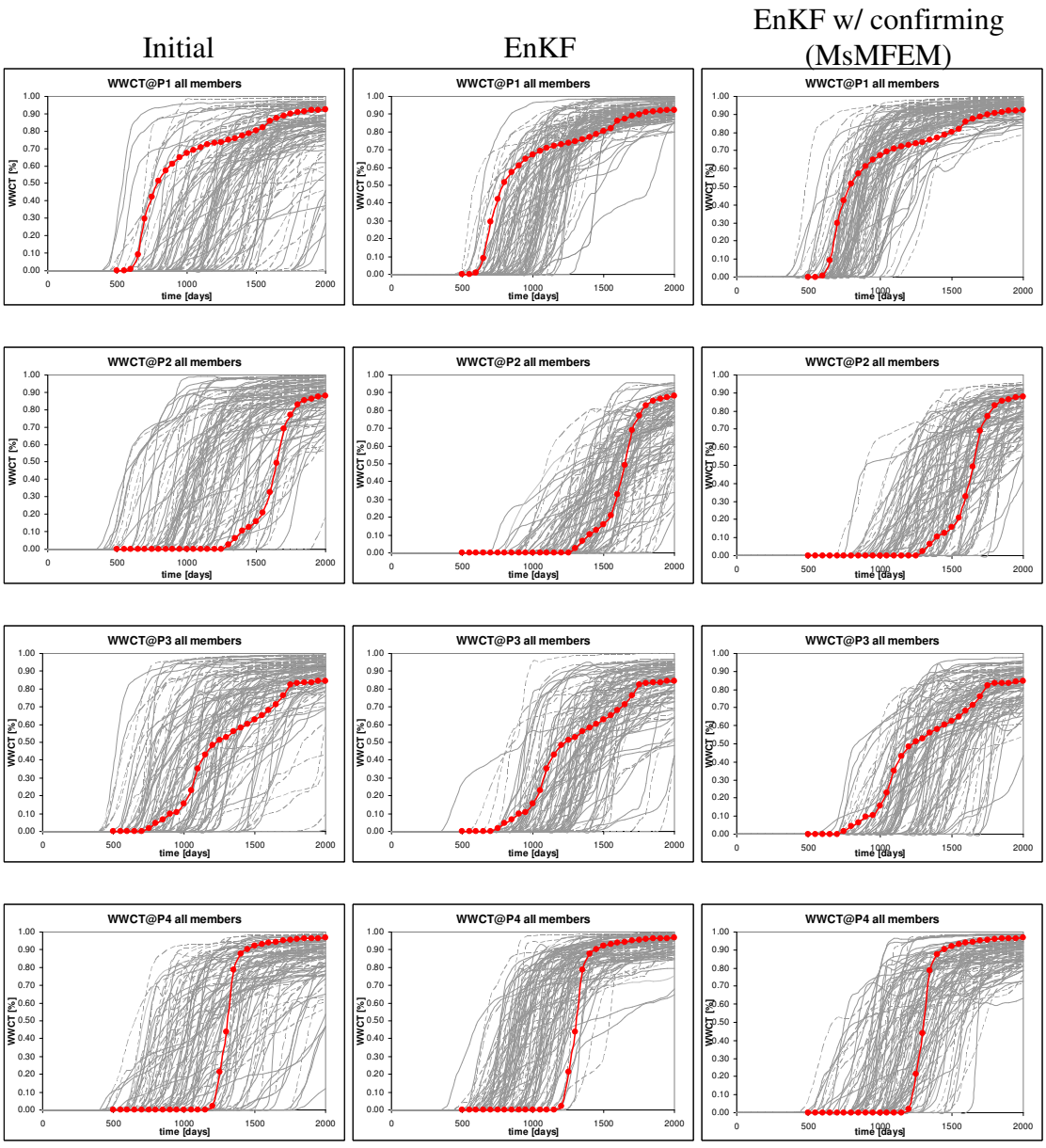


Fig.13-Ensemble spread for the final models obtained with EnKF, EnKF w/ confirming options, EnKF w/ confirming option (MsMFEM) for different producers w.r.t. TRUE, 2D, M=0.10

Fig. 13 compares the initial and final spreads in ensemble water-cuts using the standard EnKF and EnKF with confirming option while updating all the BFs.

Fine grid : 128 x 64 x 4, Coarse grid : 16 x 8 x 1, Mobility Ratio = 10.0

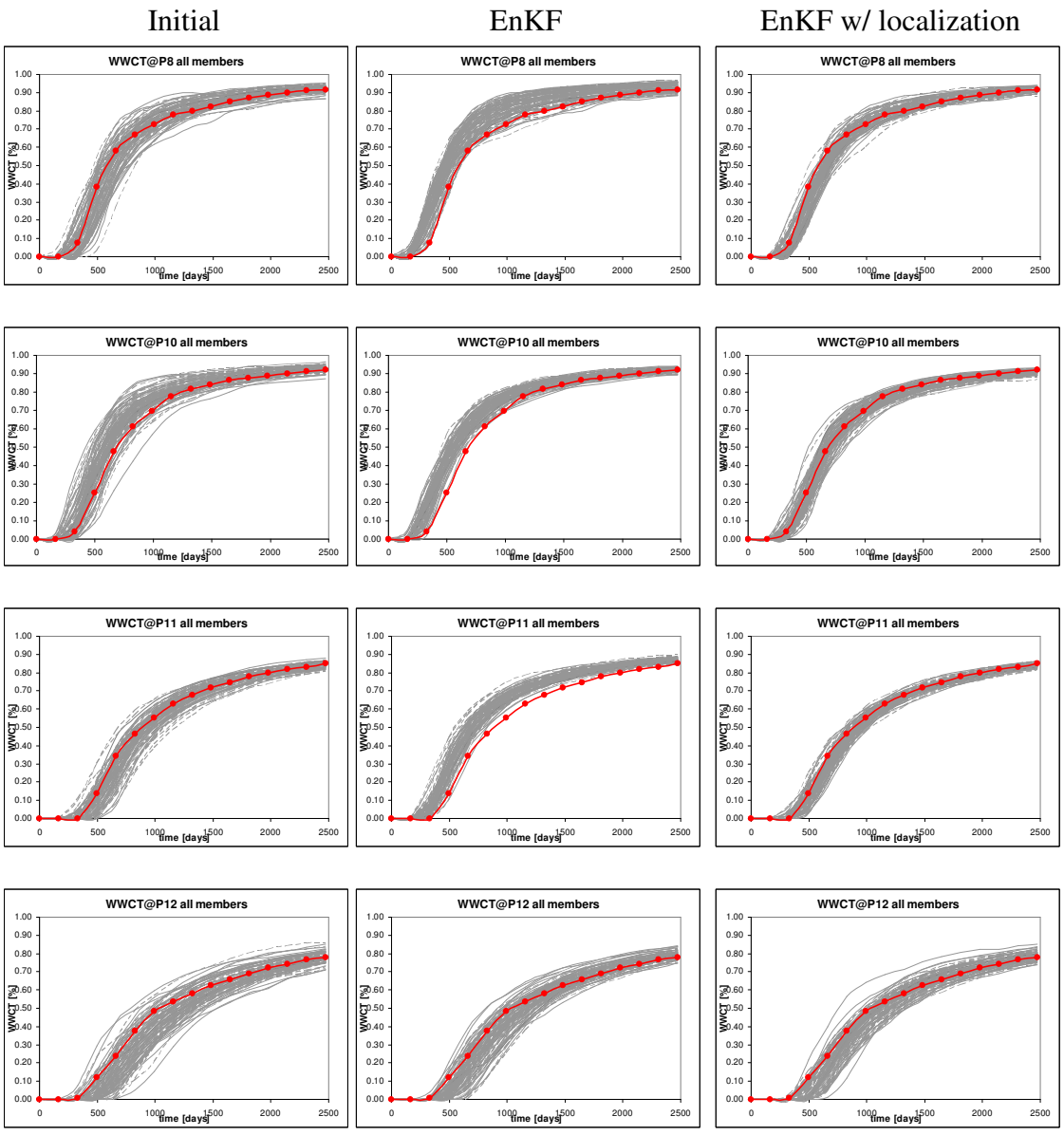


Fig.14-Ensemble spread for the initial and final models obtained with EnKF, EnKF w/ localization for different producers w.r.t. TRUE, 3D, M=10.0

Fig. 14 compares the initial spread of the ensemble water-cut to the final spread using the standard EnKF and EnKF with covariance localization using streamlines.

Fine grid : 128 x 64 x 4, Coarse grid : 16 x 8 x 1, Mobility Ratio = 10.0

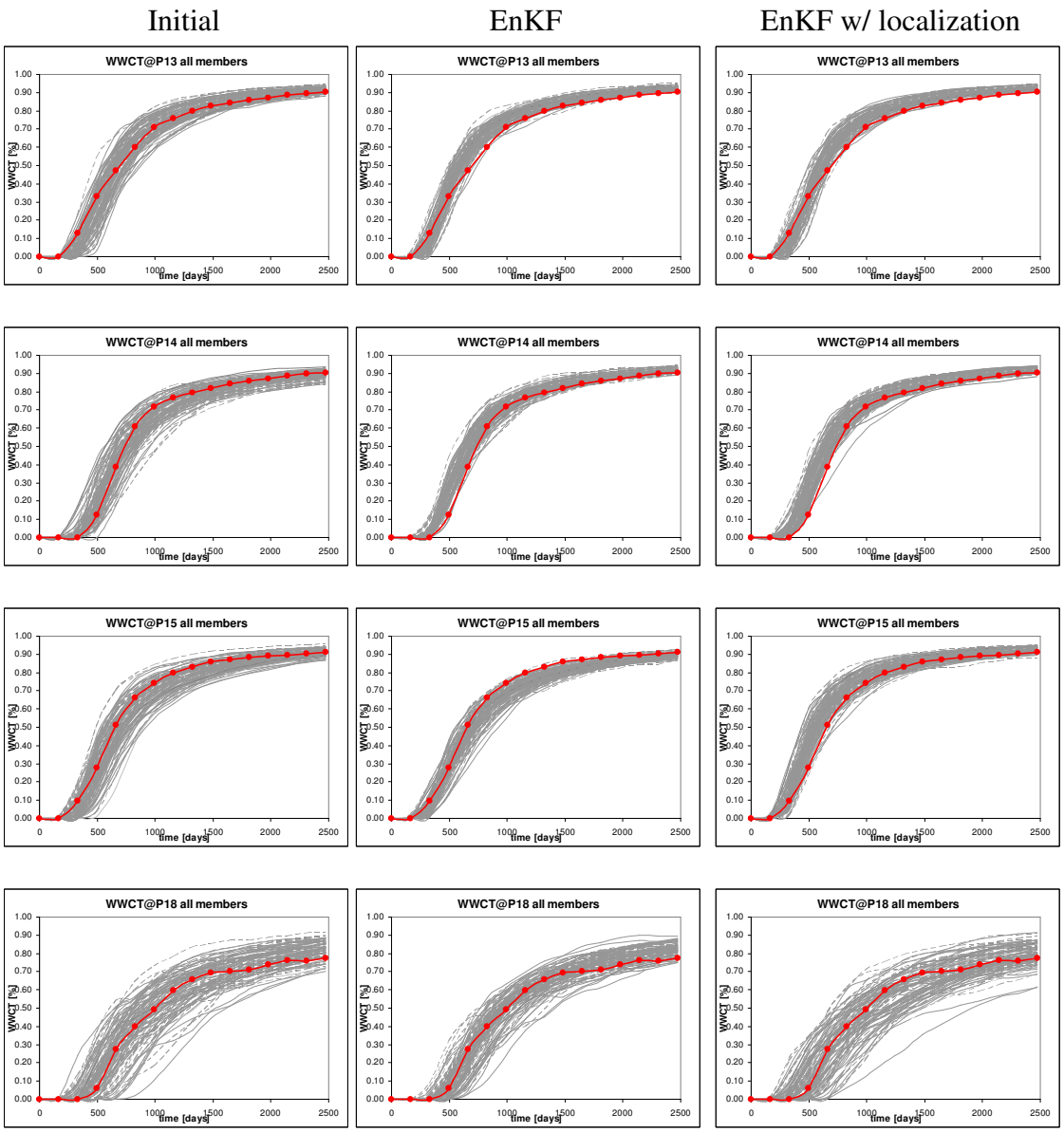


Fig.14 Continued

Fine grid : 128 x 64 x 4, Coarse grid : 16 x 8 x 1, Mobility Ratio = 10.0

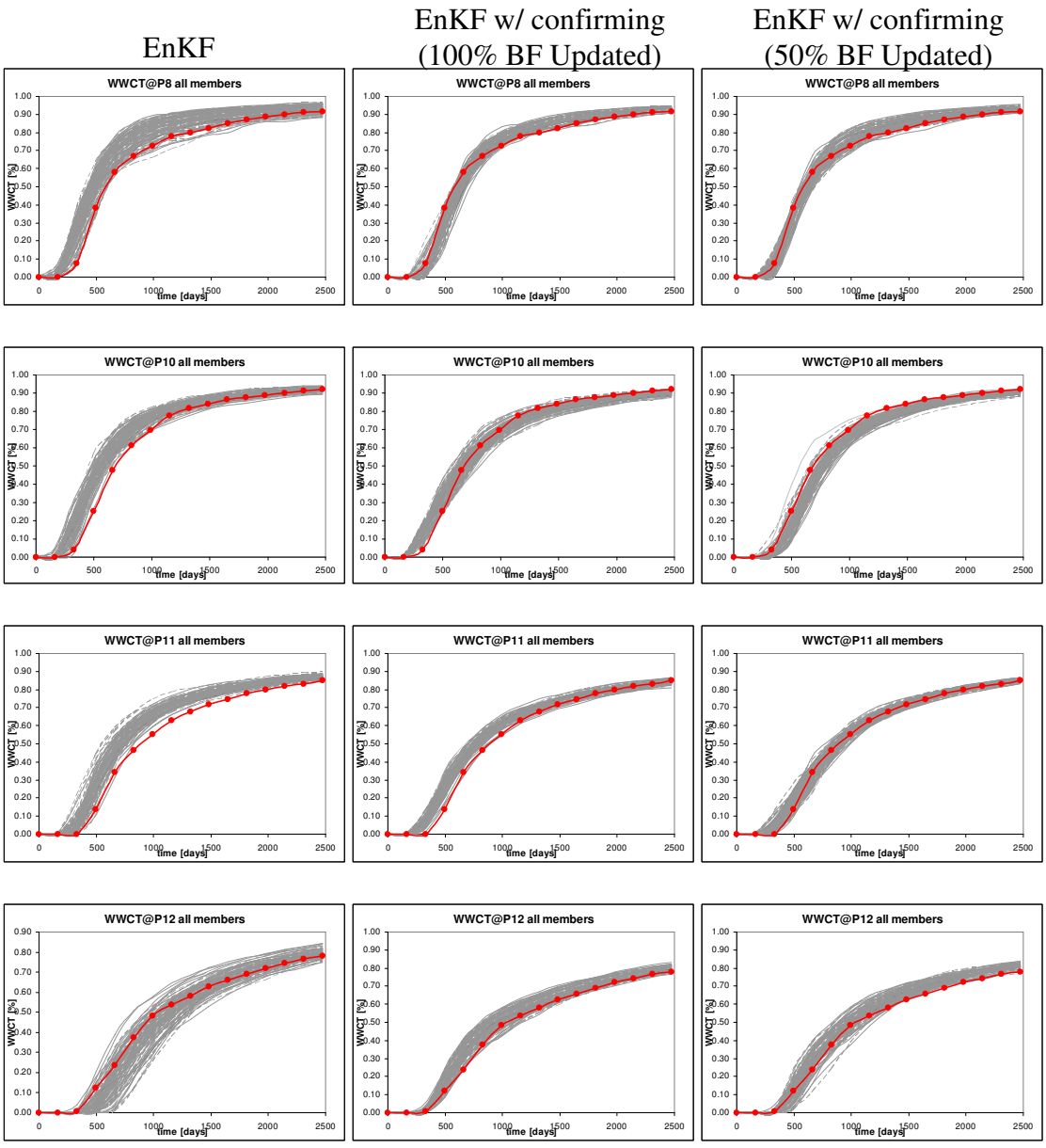


Fig.15-Ensemble spread for the final models obtained with EnKF, EnKF w/ confirming option updating 100%, 50% BFs, for different producers w.r.t. TRUE, 3D, M=10.0

Fig. 15 compares the initial and final spreads in ensemble water-cuts using the standard EnKF and EnKF with confirming option while updating 100%, 50% of the BFs.

Fine grid : 128 x 64 x 4, Coarse grid : 16 x 8 x 1, Mobility Ratio = 10.0

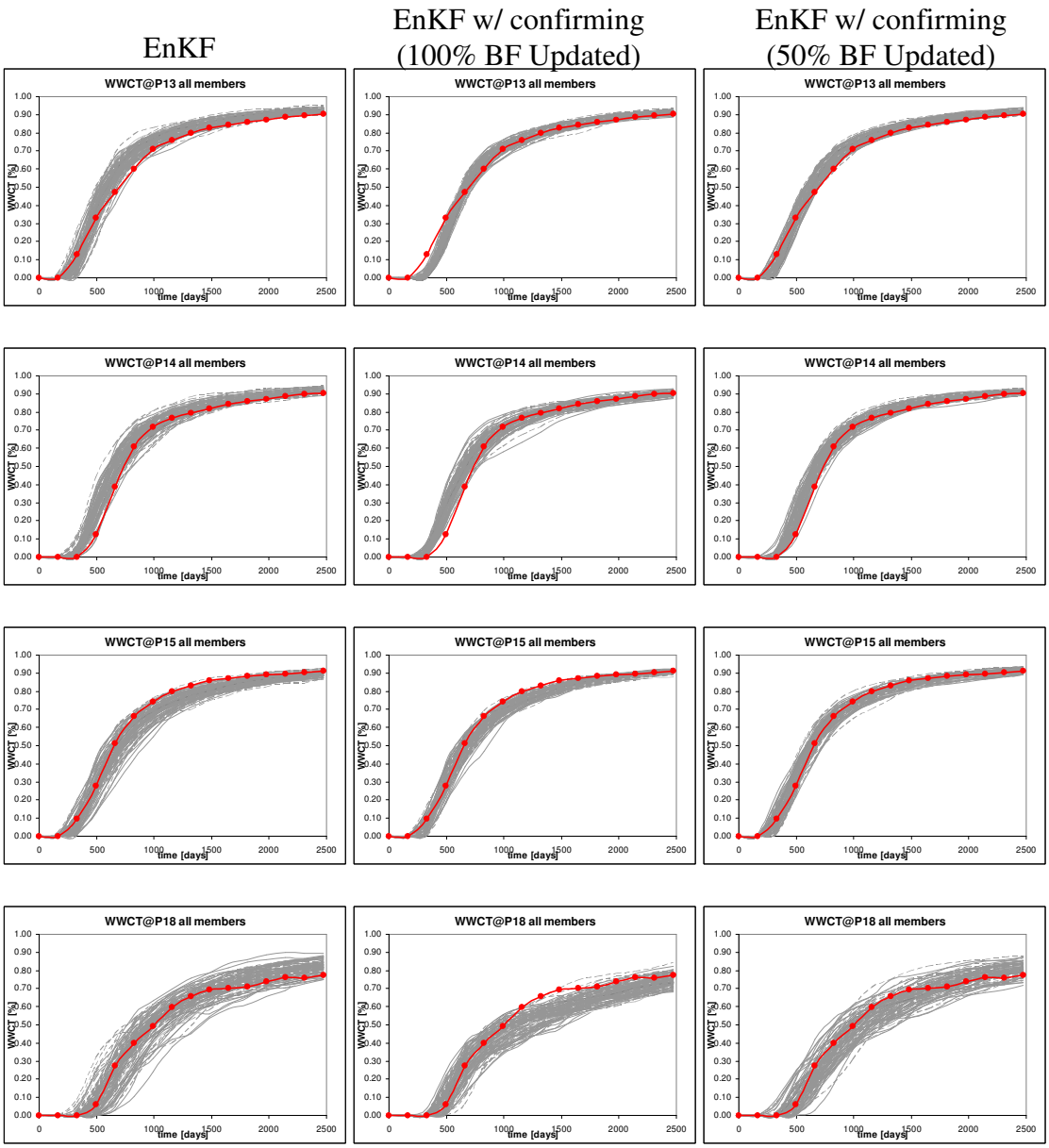


Fig.15 Continued

Fine grid : 128 x 64 x 4, Coarse grid : 16 x 8 x 1, Mobility Ratio = 0.10

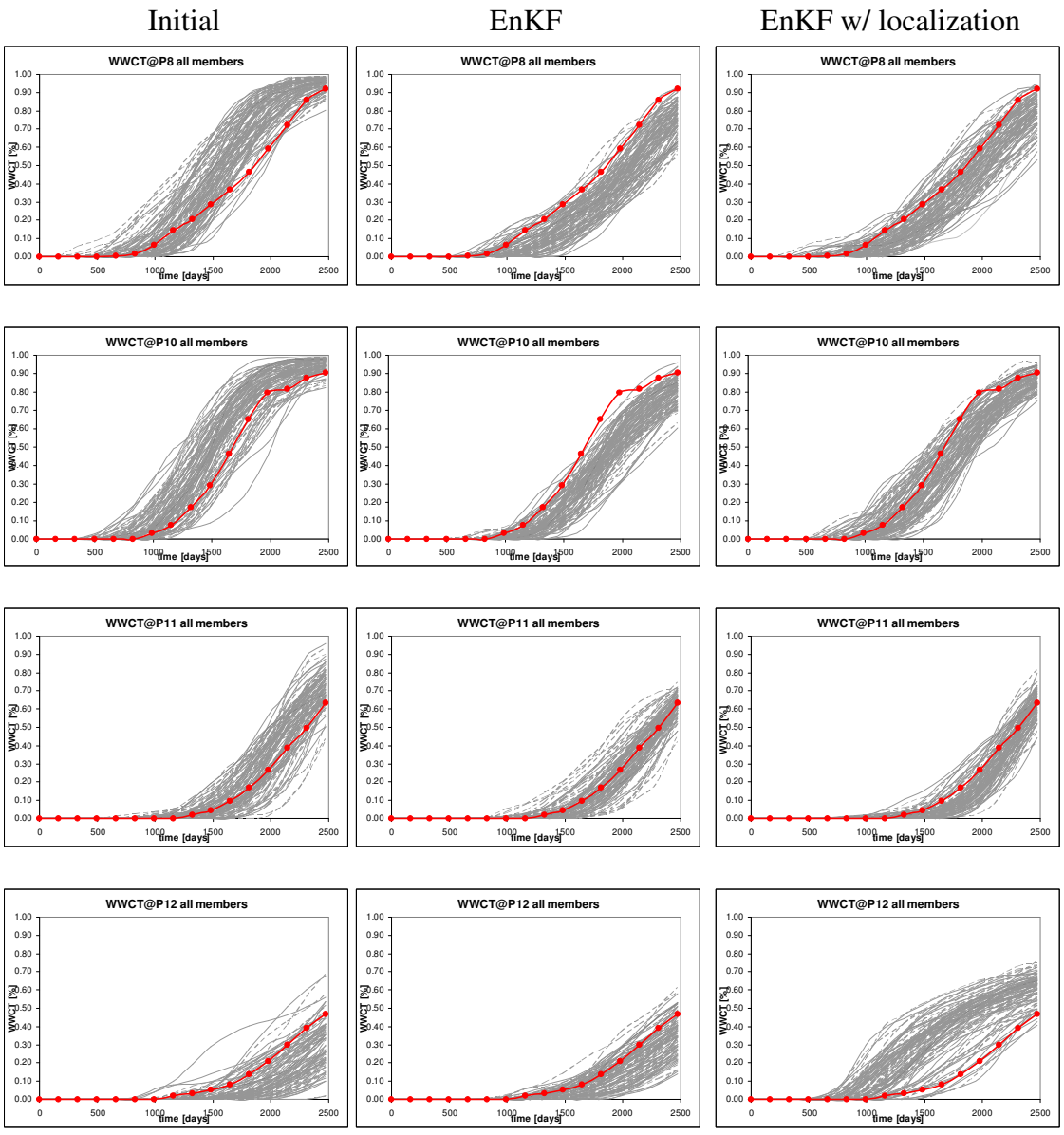


Fig.16-Ensemble spread for the initial and final models obtained with EnKF, EnKF w/ localization for different producers w.r.t. TRUE, 3D, M=0.10

Fig. 16 compares the initial spread of the ensemble water-cut to the final spread using the standard EnKF and EnKF with covariance localization using streamlines.

Fine grid : 128 x 64 x 4, Coarse grid : 16 x 8 x 1, Mobility Ratio = 0.10

Initial | EnKF | EnKF w/ localization

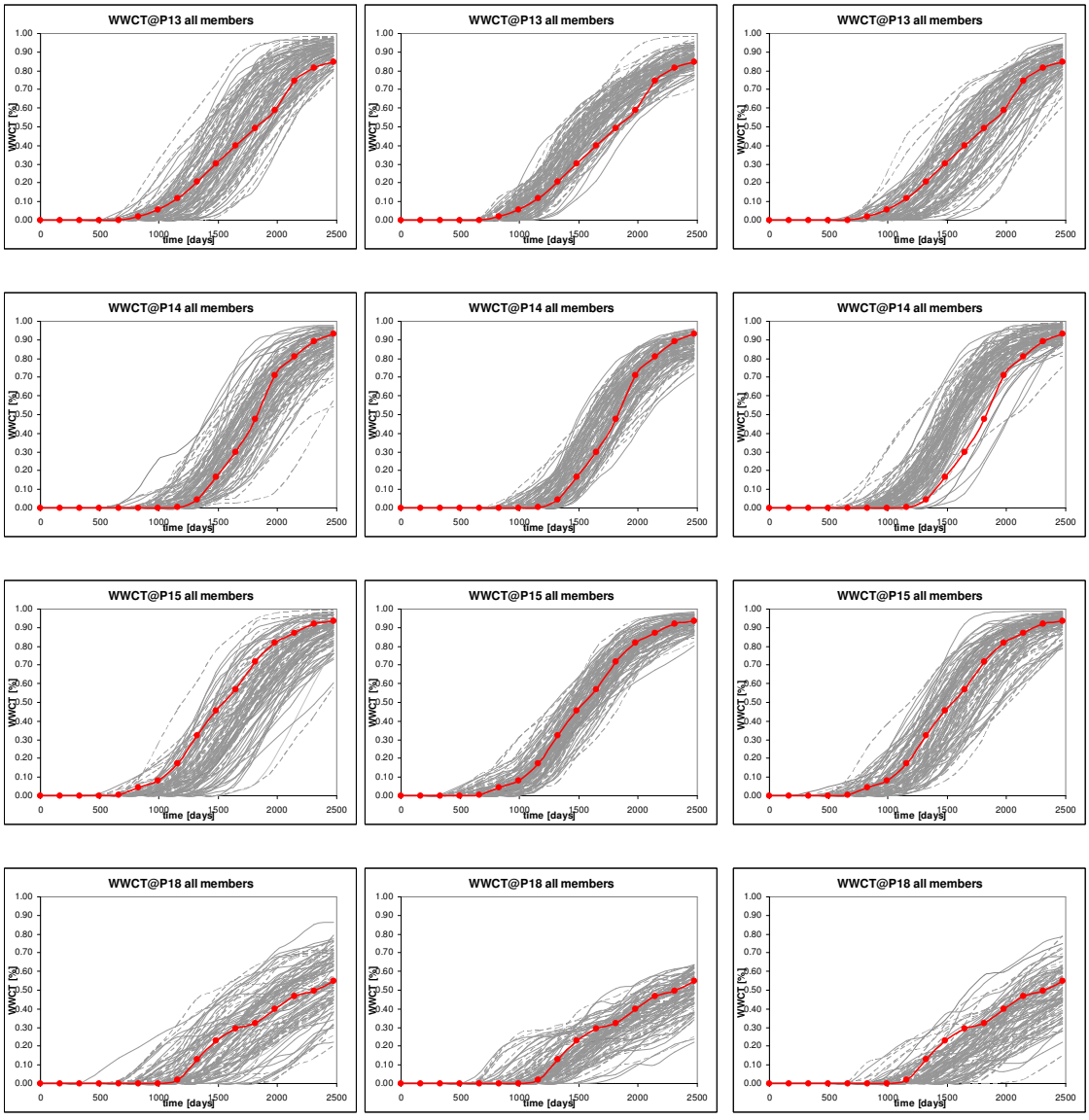


Fig.16 Continued

Fine grid : 128 x 64 x 4, Coarse grid : 16 x 8 x 1, Mobility Ratio = 0.10

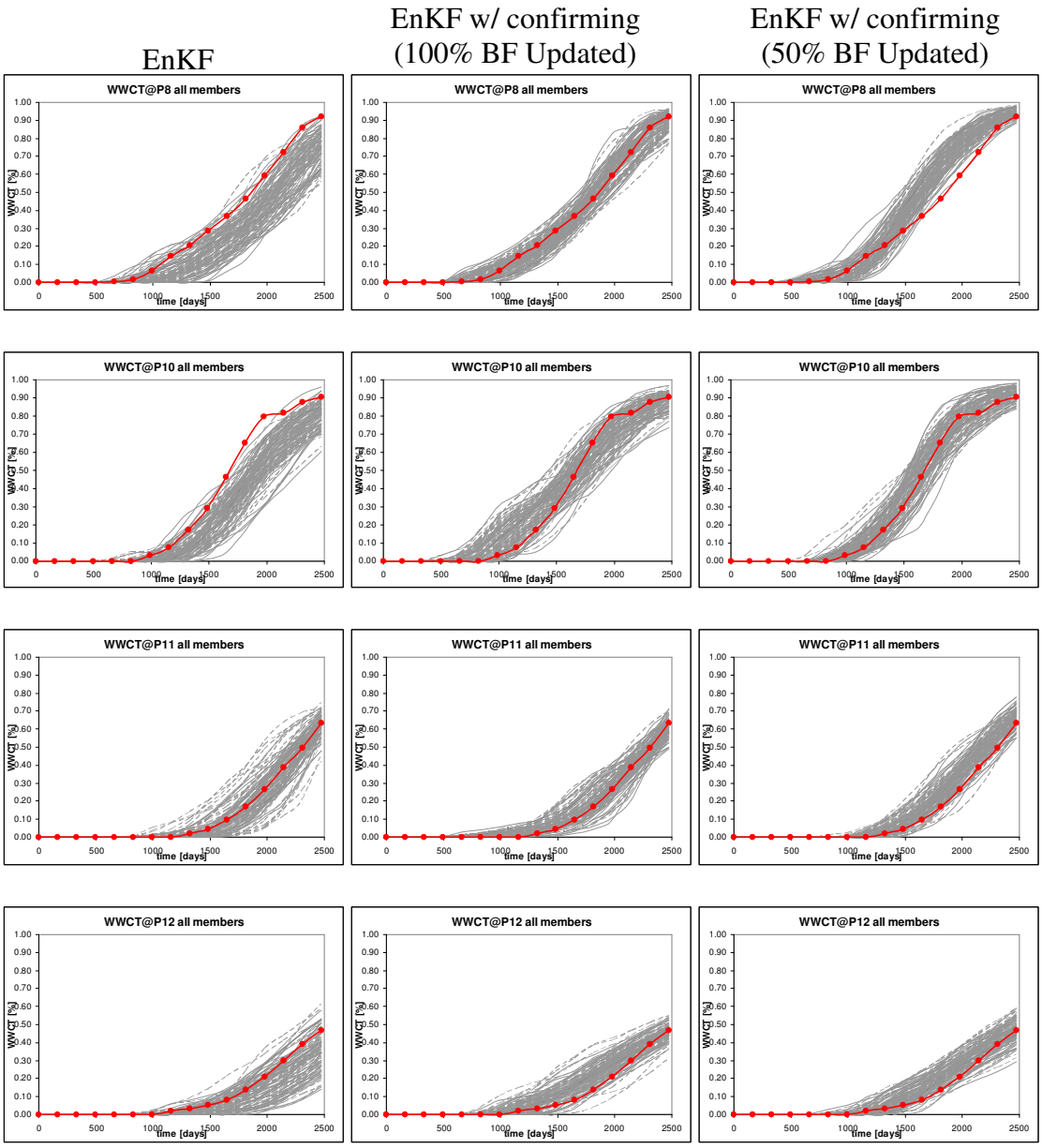


Fig.17-Ensemble spread for the final models obtained with EnKF, EnKF w/ confirming option updating 100%, 50% BFs, for different producers w.r.t. TRUE, 3D, M=0.10

Fig. 17 compares the initial and final spreads in ensemble water-cuts using the standard EnKF and EnKF with confirming option while updating 100%, 50% of the BFs.

Fine grid : 128 x 64 x 4, Coarse grid : 16 x 8 x 1, Mobility Ratio = 0.10

EnKF | EnKF w/ confirming (100% BF Updated) | EnKF w/ confirming (50% BF Updated)

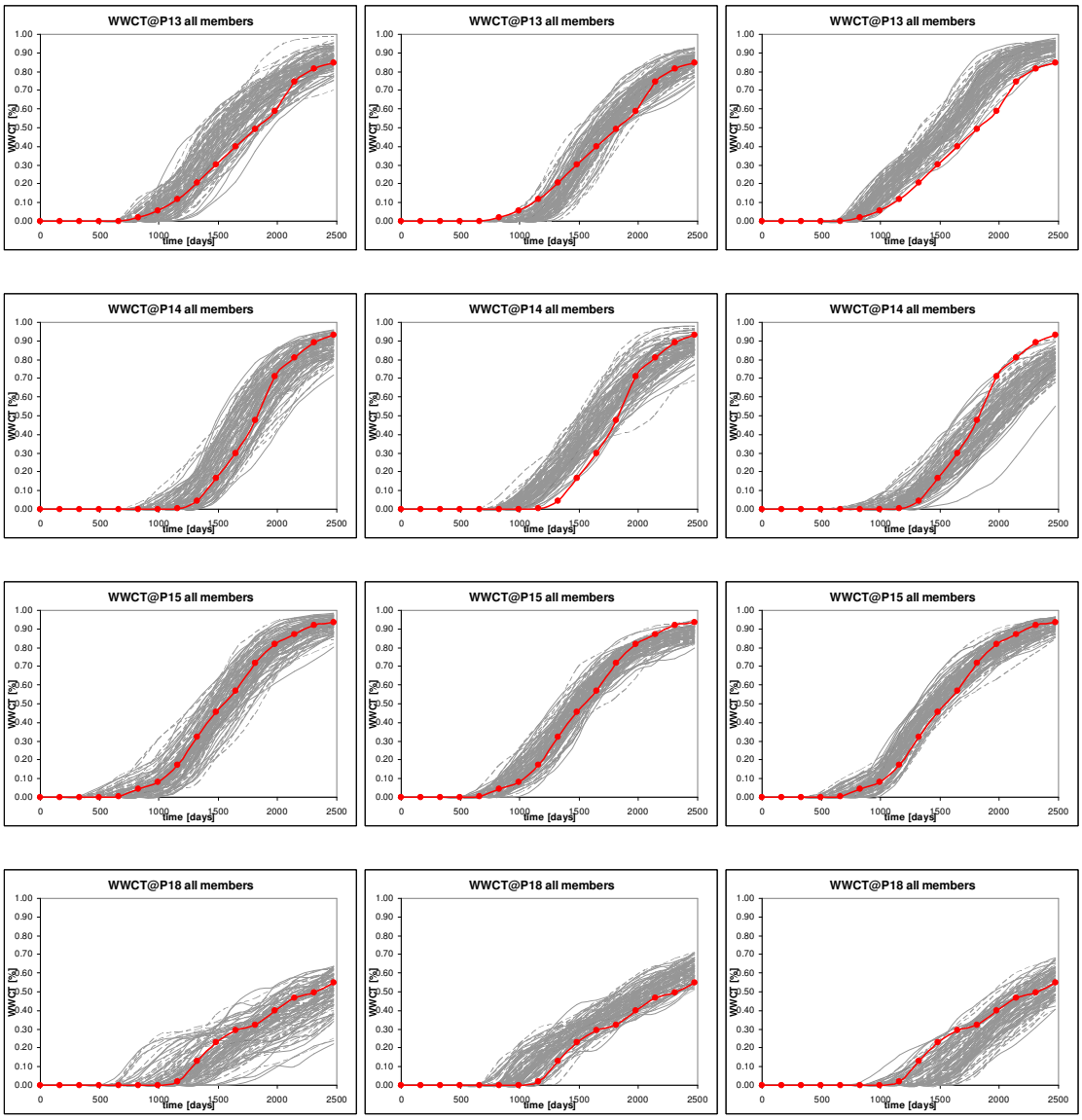


Fig.17 Continued

3.6 Comments

It is seen that the various extensions to the EnKF perform well for unfavorable as well as favorable mobility ratios when comparing the spread of the ensemble production responses with respect to the true response. High-dimensional inverse problems are known to have local maxima/minima. Hence, even a history-matched ensemble can show deviations from the true parameters (permeability) and an accurate reproduction of the model parameters is contingent on the initial guess values. A pre-conditioned initial ensemble which reflects the model error statistics correctly must be used for this purpose. Also, the true model should lie within the envelope of the initial ensemble. An easier yardstick to evaluate the method and its extensions is thus to compare the dynamic model responses of the ensemble members as opposed to the model parameters. Alternatively, the parameter estimation problem can be made tractable by simplifying the reservoir geology, thereby reducing the number of model parameters. As an example, a layered pattern with a constant value of permeability in each layer can be considered to evaluate the accuracy of the estimation process.

The covariance localization technique helps in improving the quality of the history-matched results as can be seen from **figs. 10, 12** for the 2D case, and **figs. 14, 16** for the 3D case, for M = 10.0, 0.01 respectively. For the 3D example used here, the localization option requires about **480 minutes** of simulation time (real) as opposed to **220 minutes** (real) for the standard EnKF. This however does not undermine the efficacy of this approach since in the current model every grid cell in the reservoir model is traversed by

streamlines thereby causing all the basis functions to be re-computed at after every update step. In a more realistic scenario, where a reservoir contains streaks of high permeability this approach holds a lot of promise and needs further investigation.

The EnKF w/ confirming option yields better results when compared against those obtained using EnKF only. This can be seen by comparing the spread of the final ensemble production responses in **figs. 11, 13** for the 2D case, and **figs. 15, 17** for the 3D case, for M = 10.0, 0.01 respectively. Also, in the 3D example, when the basis functions to be re-computed after every update step are determined based on a thresholding criterion, a significant reduction (13%) in the computation time is achieved without compromising the accuracy of the history-match process (**figs. 15, 17**). Similar thresholding for the 2D case does not yield noticeable gains owing to the small size of the independent flow problems being solved for, when re-computing the basis functions.

A comparison of the computation times is shown below for the 3D / 2D examples:

EnKF: 220 / 75 mins.	EnKF + Confirming (BF 100%):	830 / 200 mins.
	EnKF + Confirming (BF 50%):	720 mins (3D Only).

4. CONCLUSIONS

Based on the above results and earlier evaluations, the MsMFEM-streamline simulator is an accurate and efficient alternative for rapid reservoir simulation of two-phase incompressible flow on large geo-models and history-matching for the same, using the EnKF. The covariance localization technique tries to effectively eliminate irrelevant parameters during the update process. It promises to prove more efficient for realistic scenarios and needs further investigation. However, it does not distinguish between different cells traversed by the same streamline and thus fails to assign a higher priority to parameters which influence the production response to a larger extent. This can be improved upon by using streamline-based production response sensitivities as proposed in recent literature[14].

The confirming option leads to a 'tighter' match of the ensemble production response to the observed data. It also highlights the necessity for the proper inclusion of the state variables into the analysis scheme. Intuitively, a localization of the state variables based on streamlines will have to be investigated in this regard. This could possibly eliminate the traditional issues of material-balance errors and unrealistic values of the state when including them on an ad-hoc basis in the analysis scheme. Also, the confirming option based on the thresholding of the multi-scale basis functions yields favorable results for the history-match coupled with significant savings in computation time, as opposed to the confirming option proposed originally.

REFERENCES

1. Aarnes, J.E., Kippe, V. and Lie, K.-A.: "Mixed multiscale finite elements and streamline methods for reservoir simulation of large geomodels," *Advances in Water Resources*, (2005) **28**, (3), 257-271.

2. Aarnes, J.E., and Lie, K.-A.: "Toward Reservoir Simulation on Geological Grid Models," *Proc.*, 9th European Conference on the Mathematics of Oil Recovery, Cannes, France, 30 August - 2 September, (2004), B21.

3. Evensen, G.: "The Ensemble Kalman Filter: Theoretical Formulation and Practical Implementation," *Ocean Dynamics*, (2003) **53**, 343-367.

4. Welch, G. and Bishop, G.: "An Introduction to the Kalman Filter," Department of Computer Science, University of North Carolina, (1995) TR 95-041.

5. Pollock, D.: "Semianalytical computation of path lines for finite difference models," *Ground Water*, (1988) **26**, 743-750.

6. Aarnes, J.E.: "On the Use of a Mixed Multiscale Finite Element Method for Greater Flexibility and Increased Speed or Improved Accuracy in Reservoir Simulation," *Multiscale Modeling and Simulation*, (2004) **2**, (3), 421-439.

7. Chen, A., and Hou, T.H.: "A Mixed Multiscale Finite Element Method for Elliptic Problems with Oscillating Coefficients," *Mathematics of Computation*, (2002) **72**, 242, 541-576.

8. Aarnes, J., Efendiev, Y. and Jiang, L.: "Analysis of multiscale finite element methods using global information for two-phase simulations," Submitted to Comp. Geo.

9. Holden, H., and Risebro, N.H.: *Front Tracking for Hyperbolic Conservation Laws*, Springer-Verlag New York Inc. (2002), ISBN 3-540-43289-2.

10. Raviart, P.-A., and Thomas, J.M.: "A Mixed Finite Element Method for 2nd Order Elliptic Problems," *Mathematical Aspects of Finite Element Methods (Proc. Conf., Consiglio Naz. Delle Ricerche (C.N.R), Rome, 1975*), Lecture Notes in Mathematics, Springer, Berlin (1977) **606**, 292-315.

11. Wen, X. and Chen, W.H.: "Some Practical Issues on Real-Time Reservoir Model Updating Using EnKF," *SPE Journal*, (2007) **12**, (2), 156-166.

12. Arroyo, E., Devegowda, D., Datta-Gupta, A. and Choe, J.: "Streamline Assisted Ensemble Kalman Filter for Rapid and Continuous Reservoir Model Updating," *International Oil & Gas Conference and Exhibition in China*, DOI: 10.2118/104255-MS

13. Stenerud, V.R., Kippe V., Datta-Gupta, A. and Lie, K.-A.: "Adaptive Multiscale Streamline Simulation and Inversion of High-Resolution Geomodels," *SPE Journal*, (2008) **13**, (1), 99-111.

14. Devegowda, D., Arroyo, E., Datta-Gupta, A. and Douma, S.G.: "Efficient and Robust Reservoir Model Updating Using Ensemble Kalman Filter With Sensitivity-Based Covariance Localization," *SPE Reservoir Simulation Symposium*, DOI: 10.2118/106144-MS

15. Schmidt, S.F.: "Applications of state space methods to navigation problems," *Advanced Control Systems*, (1966) **3**, 293-340.

16. Kippe, V., Aarnes, J.E. and Lie, K.-A.: "A comparison of multiscale methods for elliptic problems in porous media flow," *Computational Geosciences, Special Issue on Multiscale Methods,* DOI: 10.1007/s10596-007-9074-6

APPENDIX A

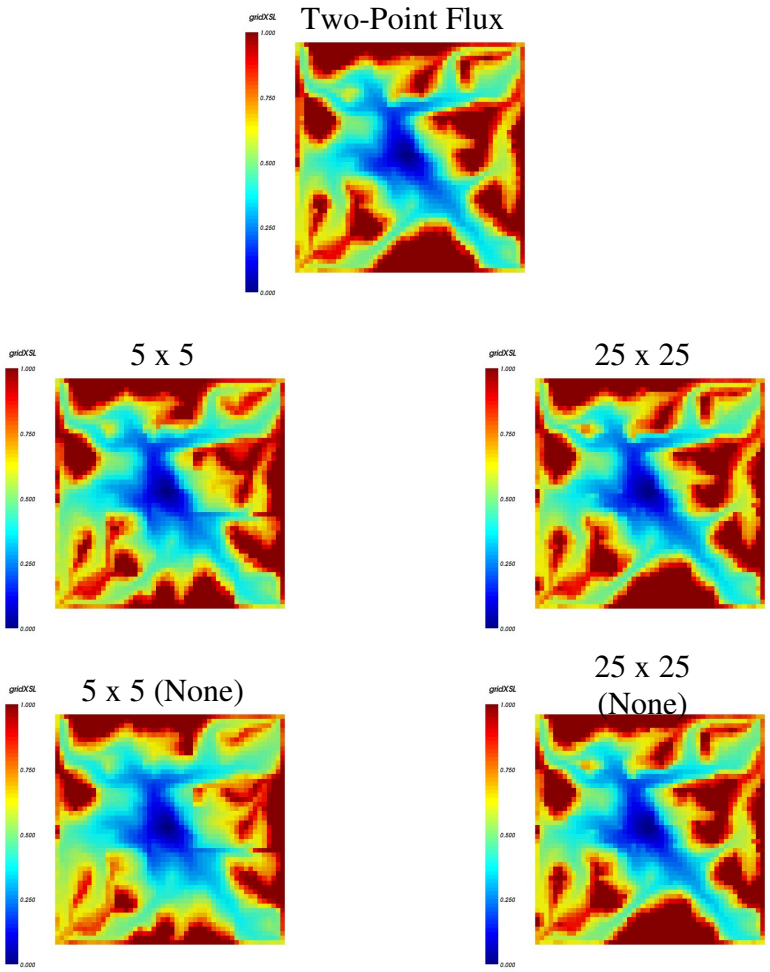


Fig.A-1 Oil saturation at t=2000 days for coarse grids, with/without updating basis functions for M=10.0

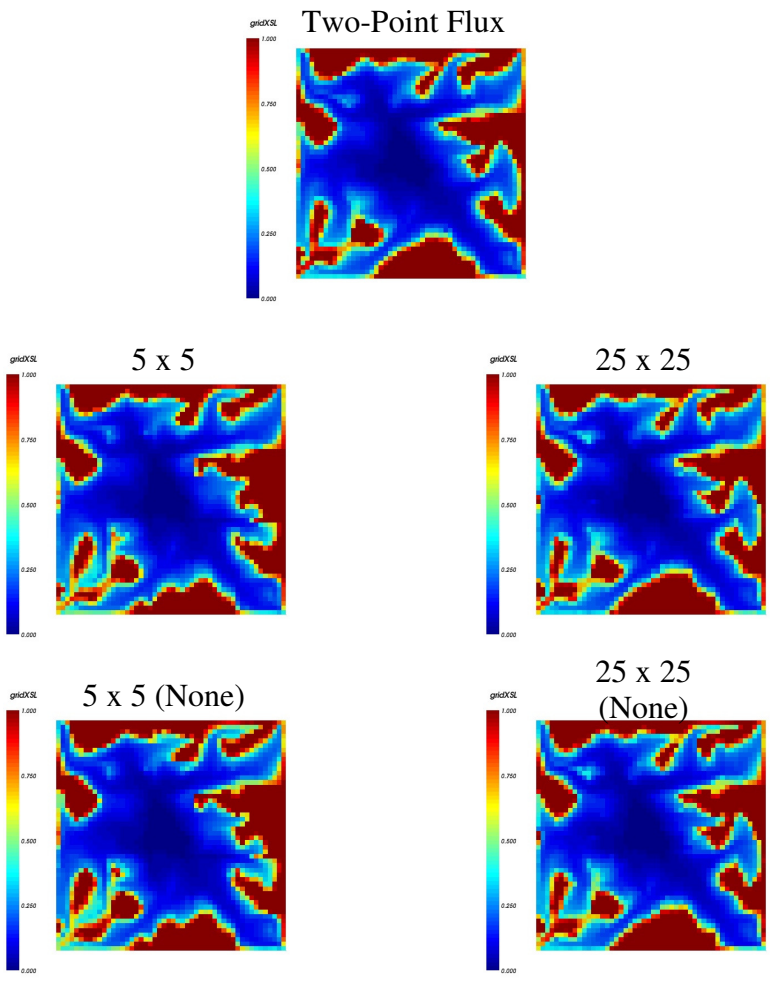


Fig.A-2 Oil saturation at t=2000 days for coarse grids, with/without updating basis functions for M=0.80

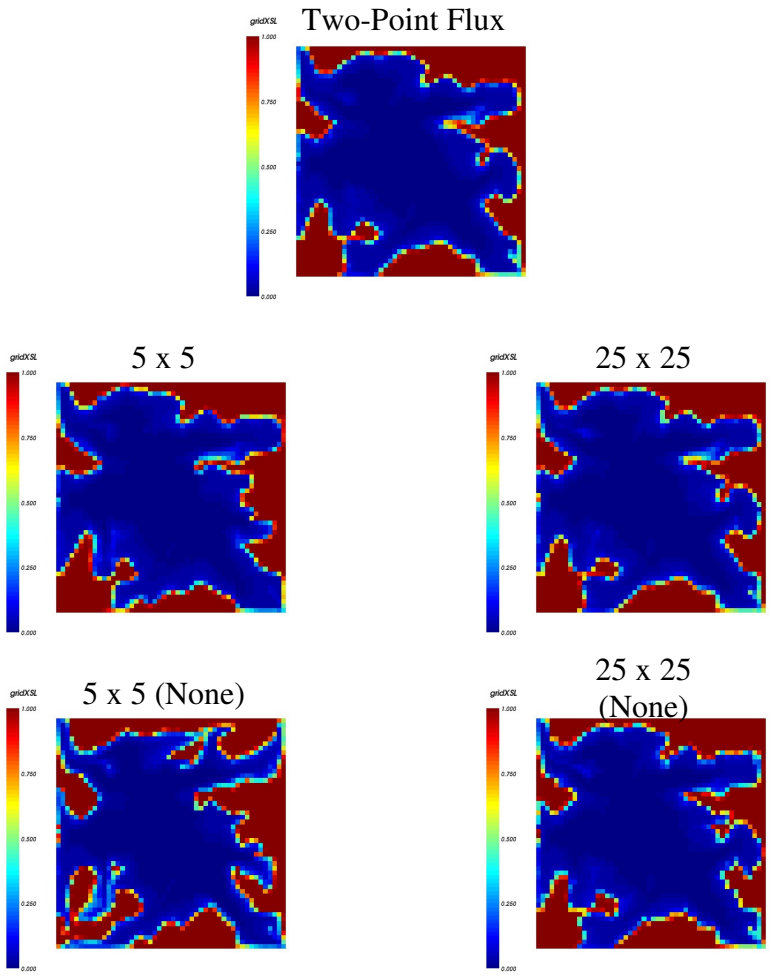


Fig.A-3 Oil saturation at t=2000 days for coarse grids, with/without updating basis functions for M=0.10

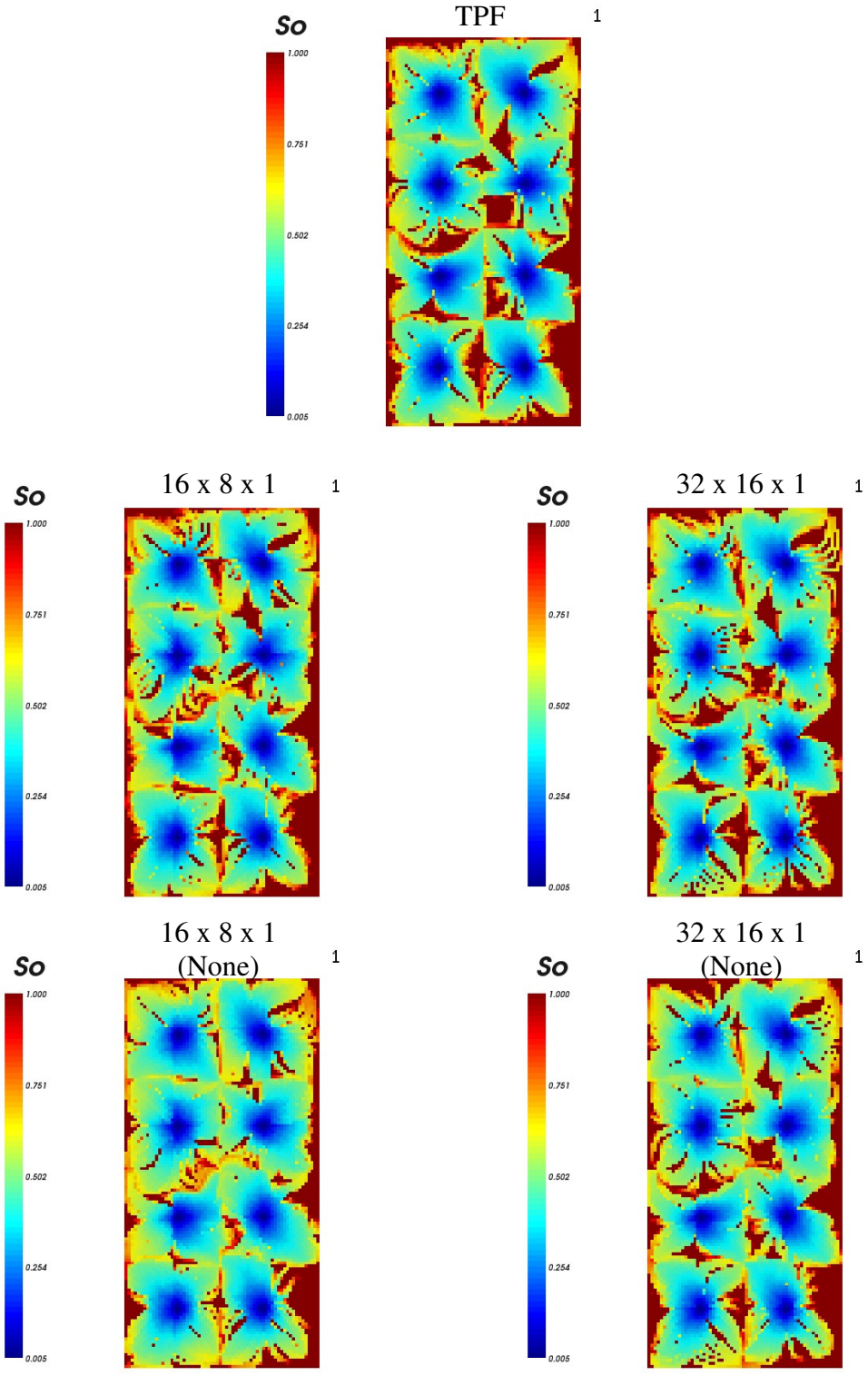


Fig.A-4 Oil saturation at t=2475 days for coarse grids, with/without updating basis functions for M=10.0

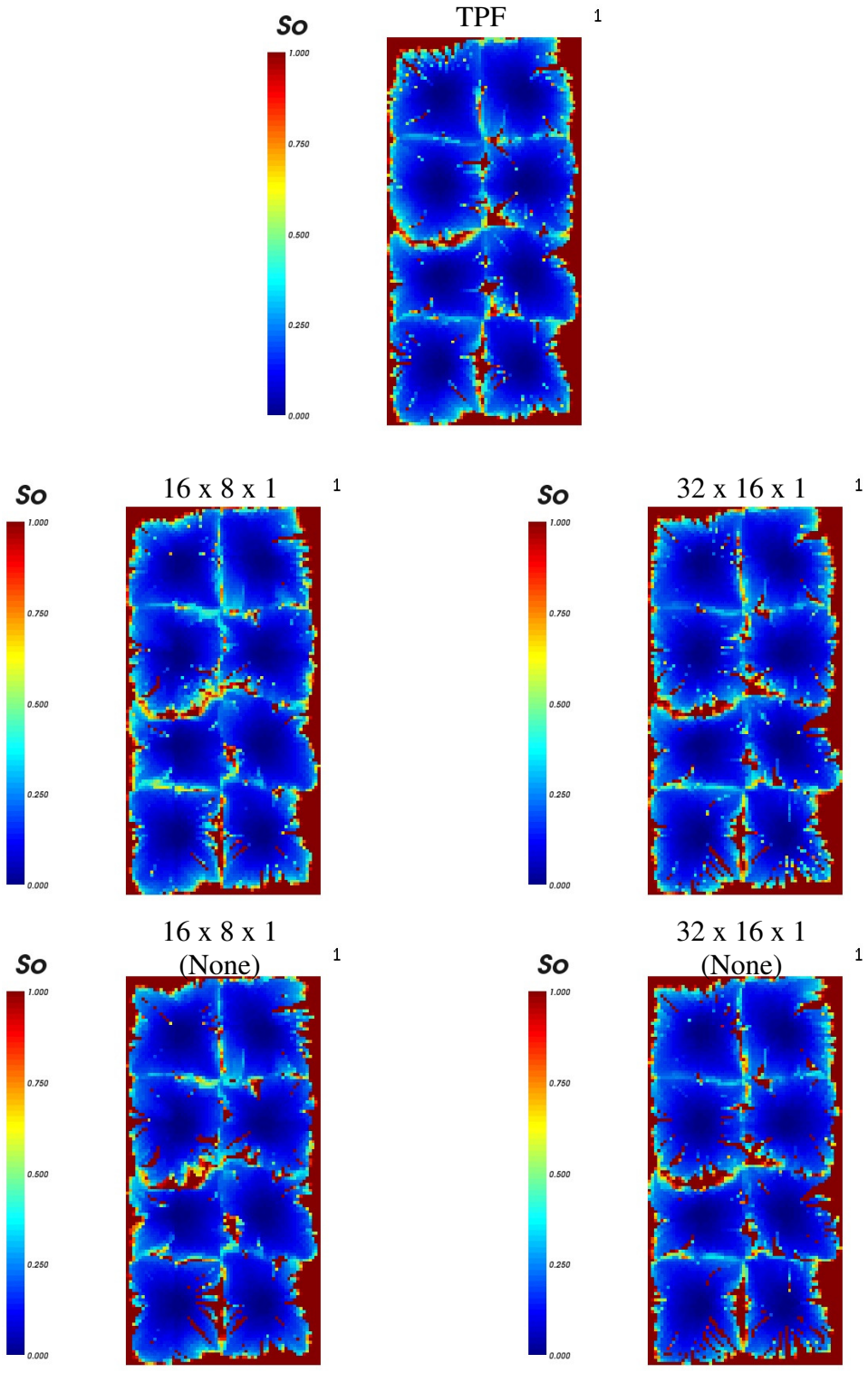


Fig.A-5 Oil saturation at t=2475 days for coarse grids, with/without updating basis functions for M=0.80

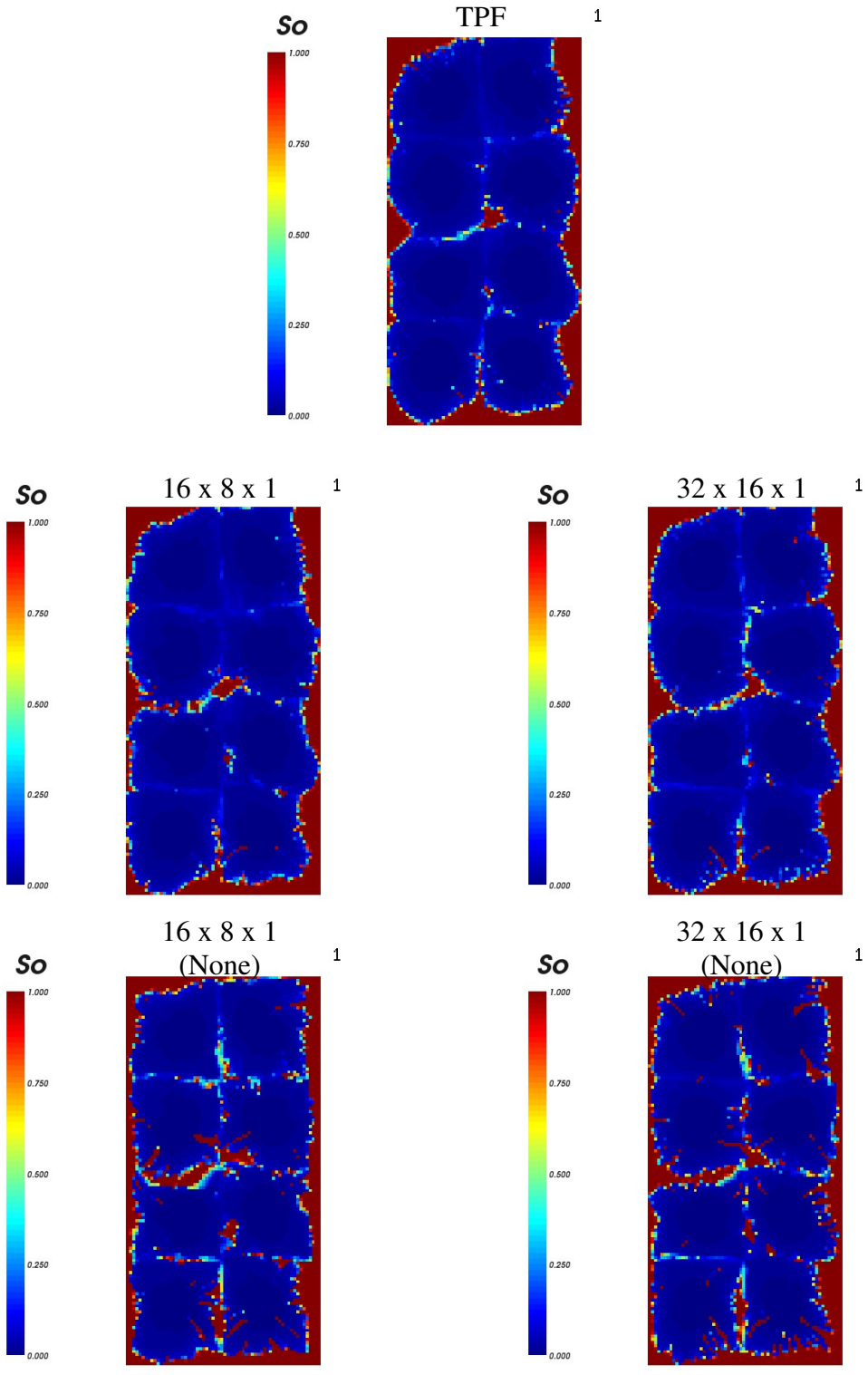


Fig.A-5 Oil saturation at t=2475 days for coarse grids, with/without updating basis functions for M=10.0

APPENDIX B

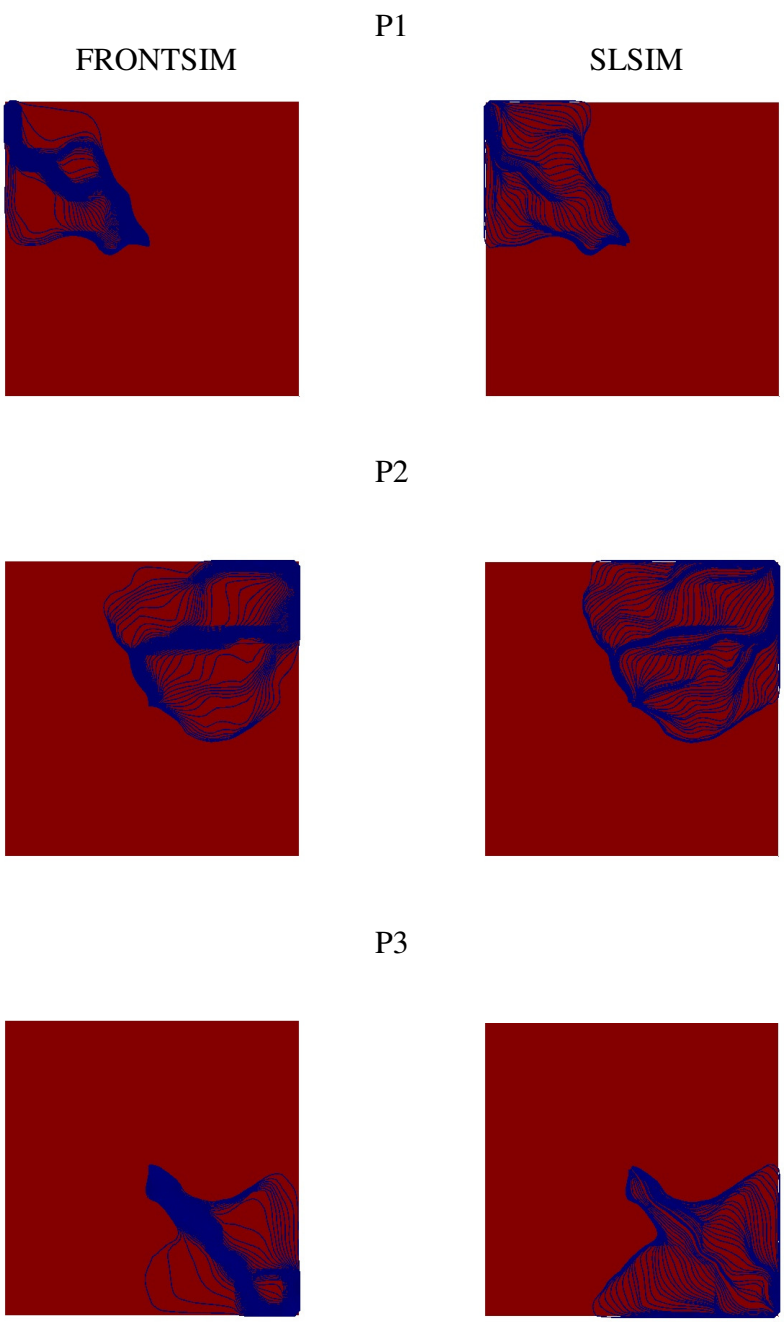


Fig.B-1 Comparison of trajectories to different producers for FRONTSIM and MsMFEM-Streamline simulator using the same number of streamlines

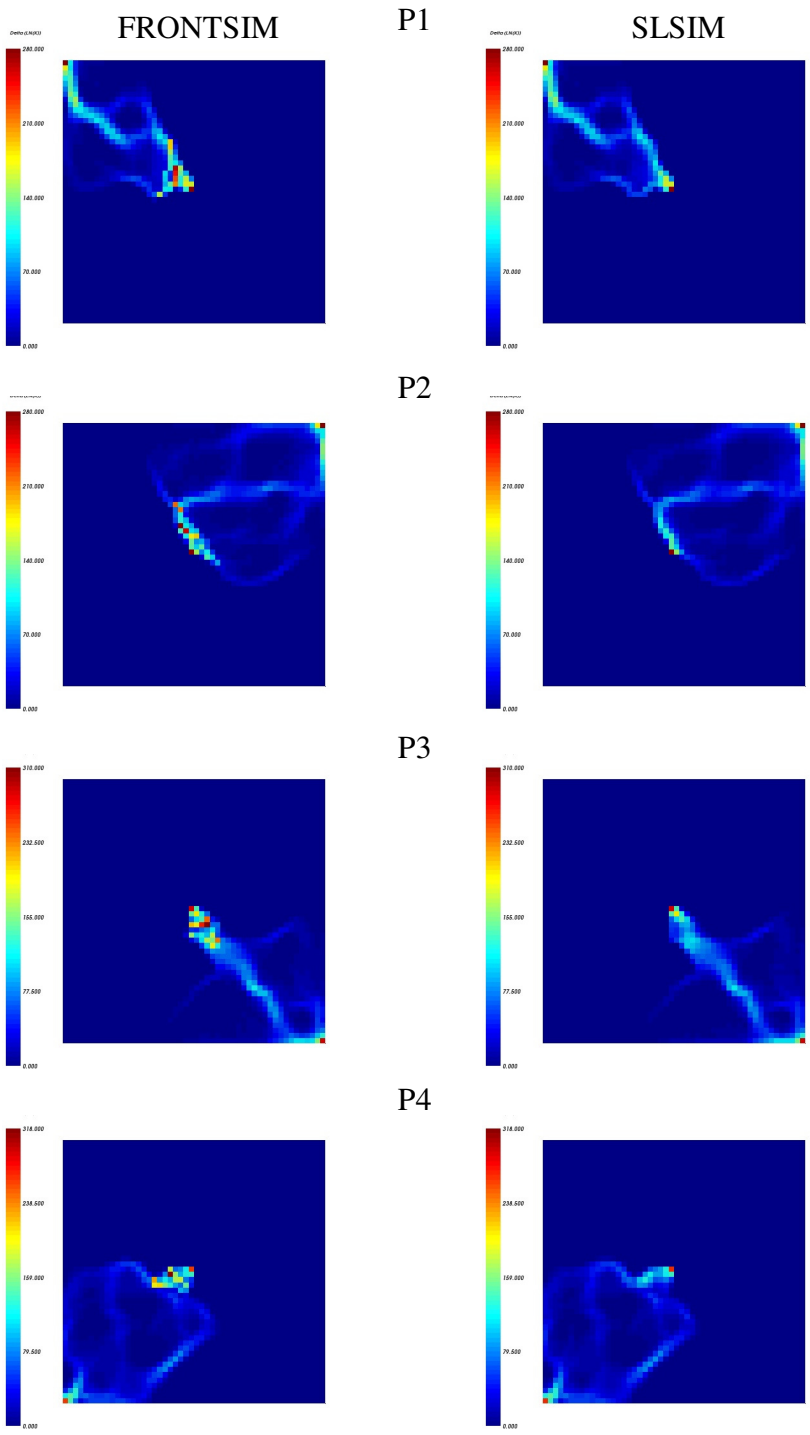


Fig.B-2 Comparison of output from filterCovInfo.dat for identical cases between FRONTSIM and MsMFEM-Streamline simulator

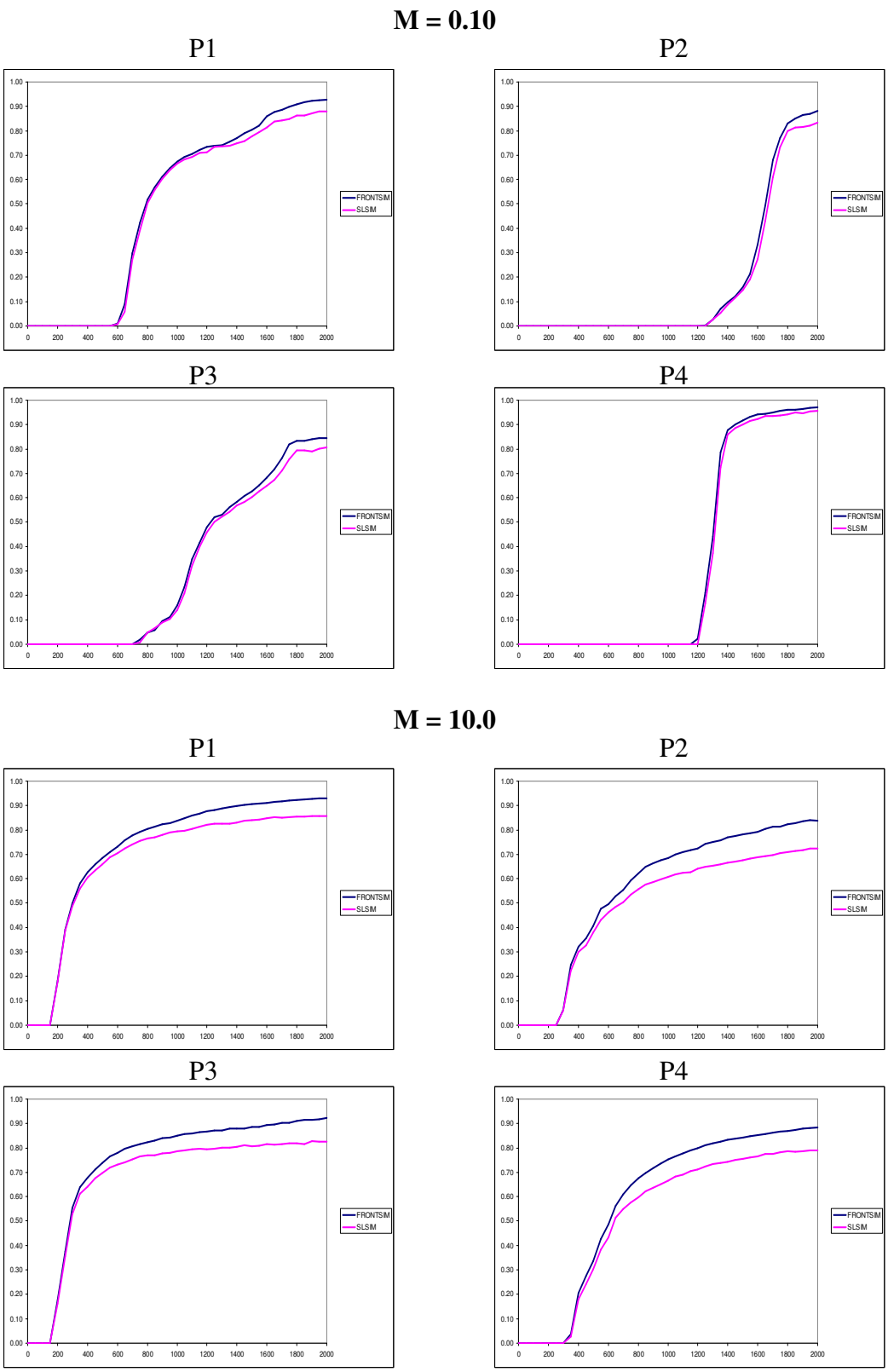


Fig.B-3 Comparison of water-cuts at different producers for identical cases with mobility ratios of 0.10, 10.0 respectively

APPENDIX C

Technical Description:

Overview:

The Multi-scale Mixed Finite Element (MsMFEM) simulator is a state-of-the-art reservoir simulator for 2-Phase incompressible flows. The code has been written in C++, and derives extensively from *Object Oriented Programming* (OOP) concepts. All class definitions are entirely *template-based* (generic). The execution of the code follows an *event-based* methodology, using *virtual functions*. These features lend the code amenable to extensions to its functionality in all possible facets. However, an undertaking of the sort requires a thorough conceptual understanding of the language. The reader is advised to do the needful beforehand. Also, a reasonable understanding of the Linux Operating System (OS) is strongly advised.

Dependencies, Compilation:

The MsMFEM uses a wide range of utility class libraries written in C, C++, and FORTRAN languages. They need to be pre-compiled (prior to compiling MsMFEM), and it must be ensured that they are placed in the OS search path (for binaries, libraries). A brief description follows:

```
------------------------------------------------------------------------
To avoid cluttering the system, install to a separate directory
structure, e.g. "Myroot", but make sure that "Myroot/bin" is in PATH.
------------------------------------------------------------------------
```

Boost C++ Libraries:

Download & extract a copy of Boost distribution from sourceforge.net.
Get bjam (bjam is the command-line tool that drives the boost build system).
Invoke bjam.

```
------------------------------------------------------------------------
      Change your current directory to the Boost root directory and
      invoke bjam (as root user), as follows:
      $>bjam --build-dir=build-directory --toolset=gcc -with-iostreams
      Copy the libraries to /usr/local/lib/
------------------------------------------------------------------------
```

Param Libraries:

A Class implementing a simple parameter map with XML save/load and command line argument parsing capabilities.

```
------------------------------------------------------------------------
      Change your current directory to the param directory:
      $>cd param
      $>./reconf
      $>./configure --prefix=path_to_Myroot --with-boost=path_to_boost
      $>make install
      Copy the binaries to /usr/local/bin/
------------------------------------------------------------------------
```

UMFPACK:

A set of C routines for solving un-symmetric sparse linear systems.

```
------------------------------------------------------------------------
      $>cd /Myroot/rsl-no-ttp-branch/linalg/umfpack/src/UMFPACKv4.3/
      Follow the UMFPACK instructions. (Note that AMD needs to be
      compiled first).
      Copy libamd.a, libumfpack.a to
      /Myroot/rsl-no-ttp-branch/linalg/umfpack/lib/
```

```
-----------------------------------------------------------------------
```

ITPACK:

A collection of FORTRAN subroutines for large sparse linear systems.

```
-----------------------------------------------------------------------
      $>cd /Myroot/rsl-no-ttp-branch/linalg/itpack/
      $>g77 -c -O6 *.f
-----------------------------------------------------------------------
```

AMG:

A set of FORTRAN routines for solving linear systems.

```
-----------------------------------------------------------------------
      $>cd /Myroot/rsl-no-ttp-branch/linalg/amg/
      $> g77 -c -O6 *.f
-----------------------------------------------------------------------
```

AtlasBLAS:

Basic Linear Algebra Subroutines (BLAS) libraries.

```
-----------------------------------------------------------------------
      $>cd /Myroot/rsl-no-ttp-branch/linalg/ATLAS/
      Follow instructions in the file INSTALL.txt
-----------------------------------------------------------------------
```

Note: For a detailed understanding on all of the above, the reader is advised to google. As an end-user however, this may not be required. The linear solvers, and BLAS libraries can be chosen from the driver program as shall be described later.

At this point, all the necessary libraries should be in place.

```
-----------------------------------------------------------------------
      $>cd /Myroot/mssl hm/
      $>make -f Makefile DEBUG=1
      Check for the binary slsim.bin, Debug version for MsMFEM.
-----------------------------------------------------------------------
```

To execute the simulator:

```
-----------------------------------------------------------------------
      $>./slsim.bin /path_to_DataDir/Main_datafilename.xml --stop_time…
-----------------------------------------------------------------------
```

Architecture of the code:

MsMFEM has been coded entirely using *template classes*. Template classes are a data abstraction mechanism to create generic class definitions. This allows the user to create a skeleton base class:

For e.g. Grid <D>, where D represents the dimension of the reservoir model i.e. 2 or 3. The parent class enforces essential *member functions* necessary to manipulate a grid data structure such as mid_point(), node_count(), edge_count(), face_count() etc. An example of a *derived* class is a StructuredGrid <D> class, which carries its specific definitions for the aforementioned functions.

The essential traits of a 2D or a 3D structured grid are the same, for e.g. we compute the mid-point given two points in space, 2D or 3D, as p_mid = 0.5 * (p_1 + p_2). However, a standard OOP methodology would require two separate definitions for the mid-point calculation for 2D and 3D respectively.

Template classes, instead, define generic code which adapts itself based on the template arguments, <D> in the above example. A generic mid-point function will accept two arguments, point #1, point #2, and return the computed mid-point. Thus, in this case, to adapt to the dimension D, the arguments passed as well as returned should be generic as well. This necessitates that the point class should be defined as a template class as well. However, this trickling-down of the generic nature varies from case to case.

The above description tries to highlight the significant benefits of template-based OOP programming, but can only be achieved with a thorough conceptual framework for the system.

The reservoir simulator is an object of the template class Simulator, as defined in Simulator2.hpp, is derived from three parent classes. These are Pressure solver, Transport solver and AbstractSimulator classes.

```
Code snippet from doxygen documentation (Simulator2.hpp):
-------------------------------------------------------------------------
00031   template<typename PSOLVER,
00032            typename TSOLVER,
00033            typename STEPPER = SimpleTimeStepper
00034            >
00035   class Simulator :
00036     public PSOLVER,
00037     public TSOLVER,
00038     public AbstractSimulator
00039   {…};
-------------------------------------------------------------------------
```

An example of the basic class hierarchy is as follows:

```
-------------------------------------------------------------------------
Simulator <PSOLVER, TSOLVER, STEPPER>
  →PressureSolverMsMHFEM <WELLS, TRAITS>
    →PressureSolver <WELLS>
      →ReservoirData_ <GRID, MOB, PERM, PORO>
      →SimpleSource_ <RESDATA>
    …
  →TransportSolverStreamline <WELLS>
    →TransportSolver <WELLS>
      →ReservoirData_ <GRID, MOB, PERM, PORO>
      →SimpleSource_ <RESDATA>
    …
  →AbstractSimulator
    …
-------------------------------------------------------------------------
```

The class ReservoirData handles reservoir input data:

```
Code snippet from doxygen documentation (ReservoirData.hpp):
-------------------------------------------------------------------------
00013   template< typename GRID,
00014             typename MOB,
00015             typename PERM,
00016             typename PORO,
00017   >
00018   class ReservoirData : public HasInit
00019   {…};
-------------------------------------------------------------------------
```

An example of the basic class hierarchy is as follows:

```
-------------------------------------------------------------------------
ReservoirData_ <GRID, MOB, PERM, PORO>
  →StructuredGrid_ <2>
    →Grid <2> …
  →PorosityVariable_
    →Porosity …
  →PermeabilityVariable_ <point2_t>
    →Permeability <point2_t> …
  →MobilityTwoPhase_ <RELPERM, PVT>
    →Mobility <RELPERM, PVT> …
  →Pressure_
    →simvector::vector <real_t> …
  →Saturation_
    →simvector::vector <sat_t> …
  →Flux_
    →simvector::vector <flux_t> …
  …
-------------------------------------------------------------------------
```

The driver program:

The following tasks are handled by the driver program, slsim.cpp:

1) Specification of the template arguments for the simulator class.
2) Instantiation and Initialization of a simulator object.

   Note: XML data file is loaded first, command line parameters are parsed next.

   Note: Initial pressures, saturation are set to zero by default.
3) Specification of initial water saturation.
   a. Constant connate water saturation can be specified.
   b. Water saturation can be read from a binary file, for RESTARTS.
4) Specification of event-based simulator listeners to generate simulation output.
5) Simulator execution.

Code snippet from doxygen documentation (slsim.cpp):

```
00106 //Specification of simulator template arguments
00107 typedef PressureSolverTPF<wells_t> psolver_t;
00109 typedef TransportSolverStreamline<wells_t> tsolver_t;
00115 typedef TimeStepperReadSteps stepper_t;
00117 typedef Simulator<
00118   psolver_t,
00119   tsolver_t,
00120   stepper_t
00121   > simulator_t;
00138 //Instantiation of the simulator
00139 simulator_t simulator;
00140 //Initialization of the simulator
00141 simulator.init (argc, argv);
00170 //Instantiation of listener to save well production data
00171 SimulatorListenerSaveWells<simulator_t> sw(simulator);
00172 //Specification to execute listener post step
00173 simulator.addListenerPostStep (&sw);
00203 //Simulator execution
00204 simulator.run ();
```

Pressure Solver:

The user can choose a Multi-Scale solver as follows:

```
-------------------------------------------------------------------
00032 #include <pressure/msmhfem/PressureSolverMsMHFEM.hpp>
00033 //#include <pressure/PressureSolverTPF.hpp>
00099 typedef PressureSolverMsMHFEM<wells_t, msmhfemtraits_t>
psolver_t;
00100 //typedef PressureSolverTPF<wells_t> psolver_t;
-------------------------------------------------------------------
```

Alternatively, the standard Two-Point Flux solver can be used, by commenting out the MsMHFEM class headers, typedefs, and including the TPF definitions as follows:

```
-------------------------------------------------------------------
00032 //#include <pressure/msmhfem/PressureSolverMsMHFEM.hpp>
00033 #include <pressure/PressureSolverTPF.hpp>
00099 //typedef PressureSolverMsMHFEM<wells_t, msmhfemtraits_t>
psolver_t;
00100 typedef PressureSolverTPF<wells_t> psolver_t;
-------------------------------------------------------------------
```

Basis Functions:

The frequency of updation of the multi-scale basis functions must also be specified. This is handled as follows:

```
Note: BASIS FUNCTIONS NEED TO BE COMPUTED FOR THE FIRST PRESSURE
STEP. FOR SUBSEQUENT PRESSURE UPDATES:
-------------------------------------------------------------------
00094   //RE-COMPUTE BASIS FUNCTIONS FOR EVERY PRESSURE UPDATE
00095   typedef UpdatePolicyUpdateAll update_policy_t;
00096   //DO NOT RE-COMPUTE BASIS FUNCTIONS
00097   //typedef UpdatePolicyUpdateNone update_policy_t;
00098   //RE-COMPUTE BASIS FUNCTIONS ENLISTED IN THE FILE,
updates.txt
00099   //typedef UpdatePolicyReadUpdates update_policy_t;
-------------------------------------------------------------------
```

Transport Solver:

For the transport solver, we have:

```
-------------------------------------------------------------------
00037 #include
<transport/streamline/TransportSolverStreamline.hpp>
```

```
00038 //#include
<transport/streamline/TransportSolverStreamlineTwoPass.hpp>
00039 //#include <transport/TransportSolverUpstream.hpp>
00102 typedef TransportSolverStreamline<wells_t> tsolver_t;
00103 //typedef TransportSolverStreamlineTwoPass<wells_t>
tsolver_t;
00104 //typedef TransportSolverUpstream<wells_t> tsolver_t;
-----------------------------------------------------------------
```

Time Stepper:

By default, a step increment and the total number of steps can be specified.

Alternatively, varying time-step sizes can be read from an input file.

```
-----------------------------------------------------------------
00009 //#include <timestep/SimpleTimeStepper.hpp>
00010 #include <timestep/TimeStepperReadSteps.hpp>
00114 //typedef SimpleTimeStepper stepper_t;
00115 typedef TimeStepperReadSteps stepper_t;
-----------------------------------------------------------------
```

Structured Cartesian Grid:

The following lines highlight a 3D implementation of the MsMFEM. When a 2D simulator is desired, the number 3 should be replaced with 2, and the simulator should be re-compiled.

```
-----------------------------------------------------------------
00072 typedef StructuredGrid<3> grid_t;
00078 typedef PermeabilityVariable<point3_t> perm_t;
00092 typedef CoarseGridCartesian<3> coarse_grid_t;
00101 typedef BasisFunctionCartesianGrid<3, LinearSolverITPACK>
basis_function_t;
-----------------------------------------------------------------
```

The grid blocks follow the numbering convention adopted by Frontsim, the x index cycling the fastest, followed by y and z indices respectively.

A 2D permeability data file should specify the total number of blocks, NX * NY, followed by Kx, Ky values of each grid block in the specified order. Similarly,

for the 3D case, the file should specify NX * NY * NZ blocks, followed by Kx, Ky, Kz permeability values for each cell according to the numbering convention.

Wells:

When using a 2D grid, the class SimpleSource must be used to simulate Injectors, Producers as:

```
-----------------------------------------------------------------
00024 #include <wells/SimpleSource.hpp>
00085 typedef SimpleSource<resdata_t> wells_t;
-----------------------------------------------------------------
```

The well-rates must be specified in a separate data file in the following format:

```
Note: C++ arrays start from the index 0. The example below
illustrates a 5-Spot pattern on a 50 x 50 x 1 grid, with DX=10.0,
DY=10.0, DZ=10.0
Note: Well rates must be specified per unit depth. Each producer
has a total-rate of 20, while the Injector has a rate of 80 in
METRIC units.
WELLNAME, IGRID, JGRID, TOTALRATE, INJECTION_SATURATION
-----------------------------------------------------------------
P1  0   0  -2.0000     1.0
P2 49   0  -2.0000     1.0
P3 49  49  -2.0000     1.0
P4  0  49  -2.0000     1.0
I1 24  24   8.0000     1.0
-----------------------------------------------------------------
```

For the 3D grid, the class SimpleSourceMultiblock must be used likewise:

```
-----------------------------------------------------------------
00025 #include <wells/SimpleSourceMultiBlock.hpp>
00086 typedef SimpleSourceMultiblock<resdata_t> wells_t;
-----------------------------------------------------------------
```

3D wells must be specified using two data files. The first file specifies the configuration as follows:

```
WELL DEFINITION FOR P1 IN WELLS_MULTIBLOCK.TXT
-----------------------------------------------------------------
BEGIN
PRODUCER P1
          5
          0           0           0
          0           0           1
```

```
          0           0           2
          0           0           3
          0           0           4
END
%
-----------------------------------------------------------------
```

The well production history is specified as follows:

```
Note:  the example below, the production rate for P1 doubles at
500 days. The entire simulated production history must be
specified.
WELL PRODUCTION HISTORY FOR P1 IN WELLHISTORY_MULTIBLOCK.TXT
-----------------------------------------------------------------
0.0       0       0       0  -5.00000
0.0       0       0       1  -5.00000
0.0       0       0       2  -5.00000
0.0       0       0       3  -5.00000
0.0       0       0       4  -5.00000
500.0     0       0       0  -10.00000
500.0     0       0       1  -10.00000
500.0     0       0       2  -10.00000
500.0     0       0       3  -10.00000
500.0     0       0       4  -10.00000
-----------------------------------------------------------------
```

Simulator Output:

The simulator output is handled by event based listeners. These listeners, as specified in the driver code, are executed after the event they are associated with occurs.

The following events are defined (in chronological order):

```
Code snippet from doxygen documentation (Simulator2.hpp):
-----------------------------------------------------------------
00170     EVENT_TYPEDEFS
00171     DEF_EVENT(PreSimulation);
00172     DEF_EVENT(PreStep);
00173     DEF_EVENT(PrePressure);
00174     DEF_EVENT(PostPressure);
00175     DEF_EVENT(PreTransport);
00175     DEF_EVENT(PostTransport);
00175     DEF_EVENT(PostStep);
00175     DEF_EVENT(PostSimulation);
-----------------------------------------------------------------
```

A variety of listeners can be chosen from. A few of them are described below:

Code snippet from doxygen documentation (slsim.cpp):

```
00152   //LISTENER TO SAVE CELL PRESSURES
00153   SimulatorListenerSavePressure<simulator_t> sp(simulator);
00154   simulator.addListenerPostPressure(&sp);
00162   //LISTENER TO SAVE CELL SATURATIONS
00163   SimulatorListenerSaveSaturation<simulator_t>
ss(simulator);
00164   simulator.addListenerPostTransport(&ss);
00170   //LISTENER TO SAVE WELL PRODUCTION DATA
00171   SimulatorListenerSaveWells<simulator_t> sw(simulator);
00172   simulator.addListenerPostStep(&sw);
00181   //LISTENER TO SAVE STREAMLINES IN FRONTSIM READABLE
FORMAT
00182   SimulatorListenerSaveStreamlineSLNFrontsim<simulator_t>
ssln(simulator);
00189   simulator.addListenerPostOneDSolver(&ssln);
```

Main input file:

Only the relevant parameter groups defined in the main XML input file have been described below. The other parameters should remain untouched, unless the user knows what he is doing.

For Restarting the simulator:

```
-----------------------------------------------------------------
<paramgroup name="slsim.cpp" description="Parameters defined at
the top level.">
      <param type="int" name="SLN_nsl" value="450">Number of
      streamlines to save in SLN files.</param>
      <param type="int" name="load_initial_sat" value="0">Should
      initial saturations be loaded from file?</param>
      <param type="string" name="initial_sat_file"
      value="initsat.bin">Name of file containing initial
      saturations.</param>
</paramgroup>
-----------------------------------------------------------------
```

Restarting the simulator requires only the binary saturation file, initsat.bin, from the last time-step when using the TPF simulator. For the MsMFEM version, the basis functions also need to be saved at the last time-step and then re-loaded for restart. This can be done by setting the parameters msmhfem_save_basis and msmhfem_load_basis to 1.

Streamlines:

```
-----------------------------------------------------------------
<paramgroup name="TransportSolverStreamline.hpp"
description="Parameters defined in class
TransportSolverStreamline.hpp.">
      <param type="int" name="nsl_to_launch" value="450">Number
      of streamlines to launch.</param>
      <param type="int" name="nministeps" value="1">Number of
      streamline ministeps.</param>
</paramgroup>
<paramgroup name="StreamlineTracer.hpp" description="Parameters
defined in class StreamlineTracer.hpp.">
      <param type="int" name="streamline_max_length"
      value="1000">Max length (in number of blocks) of each
      streamline.</param>
</paramgroup>
<paramgroup name="StreamlineDistributor.hpp"
description="Parameters defined in class
StreamlineDistributor.hpp.">
```

```
      <param type="int" name="do_backtrace" value="1">Perform
      backtrace</param>
</paramgroup>
-----------------------------------------------------------------
```

Time Stepper:

```
-----------------------------------------------------------------
<paramgroup name="TimeStepperReadSteps.hpp"
description="Parameters defined in class
TimeStepperReadSteps.hpp.">
      <param type="double" name="start_time"
      value="0.0">Simulation start time</param>
      <param type="double" name="stop_time"
      value="0.0">Simulation stop time</param>
      <param type="string" name="timestep_file"
      value="steps.dat">Name of file containing timestep
      data</param>
</paramgroup>
-----------------------------------------------------------------
```

Grid:

```
-----------------------------------------------------------------
<paramgroup name="SGridUtil.hpp" description="Parameters defined
in class SGridUtil.hpp.">
      <param type="int" name="nx" value="21">Number of fine grid
      cells in the x direction.</param>
      <param type="int" name="ny" value="21">Number of fine grid
      cells in the y direction.</param>
      <param type="int" name="nz" value="5">Number of fine grid
      cells in the z direction.</param>
      <param type="double" name="dx" value="17.5">Grid cell
      length in the x direction.</param>
      <param type="double" name="dy" value="17.5">Grid cell
      length in the y direction.</param>
      <param type="double" name="dz" value="1.0">Grid cell length
      in the z direction.</param>
</paramgroup>
<paramgroup name="CoarseGridCartesian.hpp"
description="Parameters defined in class
CoarseGridCartesian.hpp.">
      <param type="int" name="cx" value="7">Number of coarse
      grid-blocks in the x direction.</param>
      <param type="int" name="cy" value="7">Number of coarse
      grid-blocks in the y direction.</param>
      <param type="int" name="cz" value="5">Number of coarse
      grid-blocks in the z direction.</param>
</paramgroup>
-----------------------------------------------------------------
```

Reservoir Data:

```
-----------------------------------------------------------------
<paramgroup name="PVTSimple.hpp" description="Parameters defined
in class PVTSimple.hpp.">
      <param type="double" name="viscosity0"
      value="1.0">Viscosity of phase 0</param>
      <param type="double" name="viscosity1"
      value="3.0">Viscosity of phase 1</param>
      <param type="double" name="viscosity2"
      value="1.0">Viscosity of phase 2</param>
      <param type="double" name="density0" value="800.9">Density
      of phase 0</param>
      <param type="double" name="density1" value="800.9">Density
      of phase 1</param>
      <param type="double" name="density2" value="1.0">Density of
      phase 2</param>
      <param type="double" name="gravity" value="0.0">Gravity
      constant</param>
</paramgroup>
<paramgroup name="PorosityVariable.hpp" description="Parameters
defined in class PorosityVariable.hpp.">
      <param type="string" name="porosity_file"
      value="phi.dat">File name of file containing porosity
      data.</param>
</paramgroup>
<paramgroup name="PermeabilityVariable.hpp"
description="Parameters defined in class
PermeabilityVariable.hpp.">
      <param type="string" name="permeability_file"
      value="perm.dat">File containing permeability data.</param>
</paramgroup>
<paramgroup name="RelPermOilWaterTabulated.hpp"
description="Parameters defined in class
RelPermOilWaterTabulated.hpp.">
      <param type="string" name="relperm_water_file"
      value="krw_st_line.dat">File name of file containing water
      relperm data.</param>
      <param type="string" name="relperm_oil_file"
      value="kro_st_line.dat">File name of file containing oil
      relperm data.</param>
      <param type="int" name="relperm_npoints"
      value="1000">Number of points in relperm table.</param>
</paramgroup>
-----------------------------------------------------------------
```

Wells:

```
-----------------------------------------------------------------
<paramgroup name="SimpleSourceMultiBlock.hpp"
description="Parameters defined in class
SimpleSourceMultiBlock.hpp.">
      <param type="string" name="wellfile"
      value="wells_multiblock.txt">File containing well
      definitions</param>
```

```
      <param type="string" name="wellhistoryfile"
      value="wellhistory_multiblock.txt"></param>
</paramgroup>
<paramgroup name="WellConfiguration.cpp" description="Parameters
defined in class WellConfiguration.cpp.">
      <param type="string" name="well_configuration_file"
      value="wells_KATLI.txt">Well containing well
      configuration.</param>
</paramgroup>
<paramgroup name="WellHistory.cpp" description="Parameters
defined in class WellHistory.cpp.">
      <param type="string" name="well_history_file"
      value="wellhistory_KATLI.txt">Configuration file containing
      well history.</param>
</paramgroup>
-----------------------------------------------------------------
```

Pressure Solver:

```
-----------------------------------------------------------------
<paramgroup name="PressureSolverMsMHFEM.hpp"
description="Parameters defined in class
PressureSolverMsMHFEM.hpp.">
      <param type="int" name="msmhfem_load_basis"
      value="0">Should we load the msmhfem basis from
      files?</param>
      <param type="int" name="msmhfem_save_basis"
      value="0">Should we save the msmhfem basis to
      files?</param>
      <param type="int" name="msmhfem_update_loaded"
      value="0">Should we update (possibly selectively) loaded
      basis functions?</param>
      <param type="int" name="msmhfem_only_update_first"
      value="0">Should basis functions be updated only in the
      first step?</param>
      <param type="int" name="msmhfem_use_criterion"
      value="0">Should basis functions be updated selectively
      based on an updation criterion?</param>
      <param type="string" name="msmhfem_basis_dir"
      value="BASIS">Base dir for load/save of msmhfem basis
      functions.</param>
</paramgroup>
<paramgroup name="UpdatePolicyReadUpdates.hpp"
description="Parameters defined in class
UpdatePolicyReadUpdates.hpp.">
      <param type="string" name="update_file"
      value="updates.txt">File name containing indices of basis
      functions that should be updated.</param>
</paramgroup>
<paramgroup name="UpdatePolicySelectiveUpdate.hpp"
description="Parameters defined in class
UpdatePolicySelectiveUpdate.hpp.">
```

```
      <param type="double" name="update_percentage"
      value="0.00">Tolerance value for selective basis function
      updation.</param>
</paramgroup>
-----------------------------------------------------------------
```

Output Directory:

```
-----------------------------------------------------------------
<paramgroup name="Simulator2.hpp" description="Parameters defined
in class Simulator2.hpp.">
      <param type="string" name="output_dir" value="output">Base
      directory for simulation output</param>
</paramgroup>
-----------------------------------------------------------------
```

Error Handling:

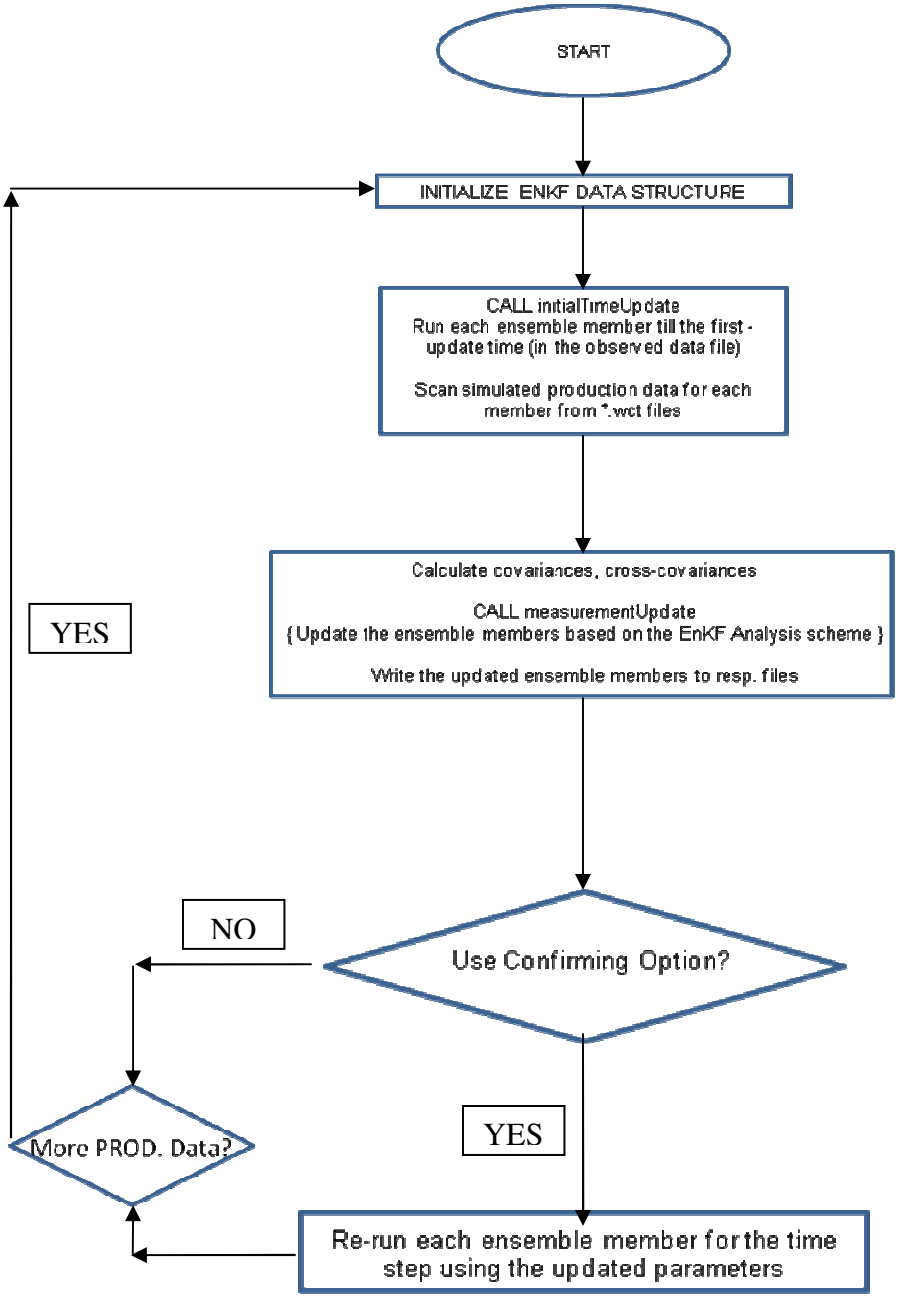
```
-----------------------------------------------------------------
<paramgroup name="Msg.hpp" description="Parameters defined in
class Msg.hpp.">
      <param type="int" name="exit_on_error" value="1">Should we
      exit on errors?</param>
</paramgroup>
-----------------------------------------------------------------
```

A few notes on the coupling between MsMFEM and EnKF codes:

1) All the filenames are case-sensitive. The MsMFEM simulator binary file must be named as ‘slsim.bin’ as the name is hard-coded in the EnKF implementation.

2) Unlike Frontsim, no schedule files need to be specified. The EnKF code handles the scheduling by reading the update step information from the observed data file, and the time-step information from the file ‘steps.dat’.

3) In the source file ‘mod_myParam.f90’, the parameter ‘dim’ needs to modified for 2D / 3D versions and ‘num_well_prod’ should be set to the number of producers being used. Also, owing to an issue that could not be resolved, the variable ‘i_nwprod’, in the source file ‘read_wct_msmfem.f’, should be set to the number of producers.


START
INITIALIZE ENKF DATA STRUCTURE
CALL initialTimeUpdate
Run each ensemble member till the first - update time (in the observed data file)
Scan simulated production data for each member from *.wct files
Calculate covariances, cross-covariances
CALL measurementUpdate
{ Update the ensemble members based on the EnKF Analysis scheme }
Write the updated ensemble members to resp. files
YES
NO
Use Confirming Option?
More PROD. Data?
YES
Re-run each ensemble member for the time step using the updated parameters

VITA

Name:	Rahul Mukerjee
Address:	PetroTel Inc.
	5240 Tennyson Parkway
	Building #1, Suite #207
	Plano, TX 75024 USA
Email Address:	rmukerjee@petrotel.com
Education:	B.TECH. (H), Aerospace Engineering,
	Indian Institute of Technology, Kharagpur, 2001
	M.S., Petroleum Engineering, Texas A&M University, 2008