STOCHASTIC DATA ASSIMILATION WITH APPLICATION TO MULTI-PHASE FLOW AND HEALTH MONITORING PROBLEMS

by

George A. Saad

A Dissertation Presented to the
FACULTY OF THE GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(CIVIL ENGINEERING)

December 2007

Dedication

To My Beloved Mother

Acknowledgments

I would like to express my sincere gratitude and appreciation to my advisor, Professor Roger Ghanem, for his insightful suggestions, technical guidance, support and encouragement throughout the course of this study. I would also like to thank my thesis committee members, Professor Sami Masri, Professor Erik Johnson, and Professor Iraj Ershaghi, for their review, discussion, and valuable comments on the thesis.

Furthermore, I would like to extend special thanks to my family for their support and encouragement in completing this work. Without their support, this work would not be possible.

The studies reported in this thesis were supported in part by grants from the National Science Foundation (NSF), and the Center for Interactive Smart Oilfield Technologies (CiSoft).

Table of Contents

List of Tables

List of Figures

Abstract

Model-based predictions are critically dependent on assumptions and hypotheses that are not based on first principles and that cannot necessarily be justified based on known prevalent physics.Constitutive models, for instance, fall under this category. While these predictive tools are typically calibrated using observational data, little is usually done with the scatter in the thus-calibrated model parameters. In this study, this scatter is used to characterize the parameters as stochastic processes and a procedure is developed to carry out model validation for ascertaining the confidence in the predictions from the model.

Most parameters in model-based predictive tools are heterogeneous in nature and have a large range of variability. Thus the study aims at improving these predictive tools by using the Polynomial Chaos methodology to capture this heterogeneity and provide a more realistic description of the system's behavior. Consequently, a data assimilation technique based on forecasting the error statistics using the Polynomial Chaos methodology is developed. The proposed method allows the propagation of a stochastic representation of the unknown variables using Polynomial Chaos instead of propagating an ensemble of model states forward in time as is suggested within the framework of the Ensemble Kalman Filter (EnKF) . This overcomes some of the drawbacks of the EnKF. Using the proposed method, the update preserves all the statistics of the posterior unlike the EnKF which maintains the first two moments only. At any instant in time,

the probability density function of the model state or parameters can be easily obtained by simulating the Polynomial Chaos basis. Furthermore it allows representation of non-Gaussian measurement and parameter uncertainties in a simpler, less taxing way without the necessity of managing a large ensemble. The proposed method is used for realistic nonlinear models, and its efficiency is first demonstrated for reservoir characterization using automatic history matching and then for tracking the fluid front dynamics to maximize the waterflooding sweeping efficiency by controlling the injection rates. The developed methodology is also used for system identification of civil structures with strong nonlinear behavior.

History matching, the act of calibrating a reservoir model to match the observed reservoir behavior, has been extensively studied in recent years. Standard methods for reservoir characterization required adjoint or gradient based methods to compute the gradient of the objective function and consequently minimize it. The computational cost of such methods increases exponentially as the number of model parameters or observational data increase. Recently, the EnKF was introduced for automatic history matching . The Ensemble Kalman Filter uses a Monte Carlo scheme for achieving satisfactory history matching results at a relatively low computational cost. In this study, the developed data assimilation methodology is used for improving the prediction of reservoir behavior. To enhance the forecasting ability of a reservoir model, first a better description of the reservoir's geological and petrophysical features using a stochastic approach is adopted, and then the new data assimilation method based on forecasting the error statistics using the Polynomial Chaos (PC) methodology, is employed. The reservoir model developed in this study is that of multiphase immiscible flow in a randomly heterogeneous porous media. The model uncertainty is quantified by modeling the intrinsic permeability and porosity of the porous medium as stochastic processes via their PC expansions. The Spectral Stochastic Finite Element Method (SSFEM) is

used to solve the multiphase flow equations. SSFEM is integrated within SUNDANCE 2.0, a software developed in Sandia National Laboratories for solving partial differential equations using finite element methods. Thus, SUNDANCE is used for the analysis or prediction step of the reservoir characterization, and an algorithm using the newmat C++ library is developed for updating the model via the new data assimilation methodology.

Using the same underlying physics, the proposed method is coupled with a control loop for the purpose of optimizing the fluid front dynamics in flow in porous media problem through rate control methods. The rate control is carried out using the developed data assimilation technique; the water injection rates are included as part of the state vector, and are continuously updated so as to minimize the mismatch between the predicted front and a specified target front.

The second application of the proposed method aims at presenting a robust system identification technique for strongly nonlinear dynamics by combining the filtering methodology with a non-parametric representation of the system's nonlinearity. First a non-parametric representation of the system nonlinearity is adopted. The Polynomial Chaos expansion is employed to characterize the uncertainty within the model, and then the proposed data assimilation technique is used to characterize the robust stochastic polynomial representation of the system's non-linearity. This enables monitoring the system's reaction and identifying any obscure changes within its behavior. The presented methodology is applied for the structural health monitoring of a four story shear building subject to a ground excitation. The corresponding results prove that the proposed system identification techniques accurately detects changes in the system behavior in spite of measurement and modeling noises.

The results obtained from these two applications depict the efficiency of the proposed method for parameter estimation problems. The combination of the Ensemble Kalman Filter with the Polynomial Chaos methodology thus proves to be an efficient

sequential data assimilation technique that surpasses standard Kalman Filtering techniques while maintaining a relatively low computational cost.

Chapter 1

Introduction

Data assimilation aims at predicting and estimating true state unknowns by combining the observed information with the underlying dynamical model. It is an interdisciplinary field involving engineering, mathematics, and physical sciences. Most data assimilation methods can be classified as either control or estimation theory methods. Control theory methods are mainly the gradient based methods. The computational cost of these methods increases exponentially as the number of unknown parameters increases. This study focuses on estimation theory methods with Kalman Filter at their heart. It aims at developing a new data assimilation methodology based on combining the Kalman Filtering techniques with the Polynomial Chaos methodology.

The Kalman Filter is a sequential data assimilation methodology, that integrates the model forward in time, and whenever measurements are available, they are used to reinitialize the model before the integration continues [Kal60]. It is an optimal state estimation process applied to dynamical systems involving random perturbations. The Kalman Filter provides a linear, unbiased, minimum variance algorithm to optimally estimate the state of the system from noisy measurements. If the model turns out to be nonlinear or a parameter estimation is required, a linearization procedure is usually performed in deriving the filtering equations [CC91]. The Kalman Filter thus obtained is known as the extended Kalman Filter (EKF). The ensemble Kalman Filter was lately introduced [Eve94] to overcome some of the drawbacks of the EKF. The EnKF considers the effects of the higher order statistics that were neglected by the EKF due to linearization, but preserves the first two moments only when propagating the error statistics. It also reduces

the computational cost of the EKF by presenting simpler techniques for propagating the model errors.

This study presents a data assimilation method based on forecasting the error statistics using the Polynomial Chaos methodology. Instead of propagating an ensemble of model states forward in time as is suggested within the framework of the EnKF, the proposed method allows the propagation of a stochastic representation of the unknown variables using Polynomial Chaos. This overcomes some of the drawbacks of the EnKF. Using the proposed method, the update preserves all the statistics of the posterior unlike the EnKF which maintains the first two moments only. Furthermore, at any instant in time, the probability density function of the model state or parameters can be easily obtained by simulating the Polynomial Chaos basis. It also allows the representation of non-Gaussian measurement and parameter uncertainties in a simpler, less taxing way without the necessity of managing a large ensemble. The computational load for relevant accuracy is comparable to that of the EnKF and the storage size is limited to the number of terms in the PC expansion of the model states. The proposed method can be used for realistic nonlinear models, and its efficiency is demonstrated on a reservoir characterization using automatic history matching, for front tracking of the flow in porous media, and for system identification of structures with strong nonlinearities.

1.1 Reservoir Characterization

Reservoir simulation is a powerful tool for reservoir characterization and management. It enhances the production forecasting process. The efficiency of a reservoir model relies on its ability to characterize the geological and petrophysical features of the actual field. One of the most commonly used methods for reservoir characterization is the automatic history matching methodology. History matching aims at estimating reservoir

parameters such as porosities and permeabilities so as to minimize the square of the mismatch between observations and computed values.

In short, history matching is a parameter estimation problem aiming at finding the probability density function of the parameters and associated model states based on measurements related to these states and possibly the parameters themselves [Eve05]. Well established history matching techniques rely on the gradient type methods for minimization of cost functions [Ca74, CDL75, Ma93]. The gradient-based approach was extended for multiscale estimation [GMNU03, Aan05]. Further the gradual deformation method introduced by *Roggero* and *Hu* [RH98] has gained some interest. These traditional history matching techniques suffer from some drawbacks [NBJ06]:

- They are usually only performed after period of years on a campaign basis.
- Ad hoc matching techniques are applied, and the process usually involves manual adjustment of model parameters instead of systematic updating.
- Measurement and modeling uncertainties in the state vector are usually not explicitly taken into account.
- Often, the resulting history matched model violates essential geological constraints.
- The most critical issue is that although the updated model may reproduce the production data perfectly, it has little or no predictive capacity because it may have been over fitted by adjusting a large number of unknown parameters using a much smaller number of measurements.

Various techniques for automated history matching have been developed over the past decade to address these issues. The Ensemble Kalman Filter (EnKF) introduced by Evensen [Eve94, BLE98, Eve03] was recently used for online estimation of reservoir parameters and state variables given the production history data [NMV02, NJAV03, GO05, GR05, LO05, WC05]. Although, the EnKF yields satisfactory history matching results it has some drawbacks. The EnKF copes badly with non-Gaussian probability density functions since it requires a large ensemble size to accurately characterize the statistics of the corresponding functions [Kiv03]. Thus, this study presents a variation of the EnKF that allows the propagation of a stochastic representation of the variables using the Polynomial Chaos expansion. Consequently all the statistics of the random variables are preserved. First, it is required to accurately represent the geological features of the reservoir and the fluid flowing within, and consequently uncertainty must be quantified. Major sources of uncertainty in the process of history matching and production forecasting include [AML05]:

- Measurement errors
- Uncertain description of geological and fluid parameters
- Model errors and imperfections

The model uncertainty is quantified by modeling the intrinsic permeability and porosity of the porous medium as stochastic processes via their Polynomial Chaos expansions. This yields a system of coupled nonlinear stochastic partial differential equations. The Spectral Stochastic Finite Element Method (SSFEM) proposed by Ghanem and Spanos [GS03] is employed to solve this system at discrete time steps. SSFEM is an extension of the deterministic finite element method (FEM) to stochastic boundary value problems. In the context of SSFEM, the randomness in the problem is regarded as an additional dimension with an associated Hilbert space of L_2 functions in

which a set of basis functions is identified. This set is known as Polynomial Chaos and is used to discretize the random dimension [GS03, Wie38]. The SSFEM is integrated within SUNDANCE 2.0 [Lon04], a finite element partial differential equation solver, for maximal computational efficiency. SUNDANCE is a toolkit that allows construction of an entire parallel simulator and its derivatives using a high-level symbolic language. It is sufficient to specify the weak formulation of the partial differential equation and its discretization along with a finite element mesh to obtain a solution. The proposed method is then used to filter the stochastic representations obtained by the SSFEM. The proposed history matching technique is used for the characterization of both one dimensional and two dimensional heterogeneous reservoirs. Different ways for representing the medium uncertainties are discussed, and the obtained results reveal the efficiency of the proposed scheme.

1.2 Front Tracking

Another important aspect in reservoir simulations is the maximization of some measure of the displacement efficiency. In cases where the injected fluid composition is fixed, the only given way to control the front dynamics is through controlling the allocation of the injected fluid to the injection wells. Therefore, to maximize the sweeping efficiency of the water flooding process, an algorithm to update the injection flow rate so as to maintain a specified front is devised. The algorithm uses the proposed filtering scheme and it aims at minimizing the mismatch between the predicted water saturation and a discretization of the objective front. This is coupled with the history matching technique described earlier rendering a flow control problem which takes into consideration the effects of parametric, modeling, and measurement uncertainties. This has been an active research area in the past couple of decades [NBJ06, SY00, FR86, FR87, Ash88].

Using the same underlying physics described earlier, the proposed method is coupled with a control loop for the purpose of optimizing the fluid front dynamics in flow in porous media problem through rate control methods. The rate control is carried out using the developed data assimilation technique; the water injection rates are included as part of the state vector, and are continuously updated so as to minimize the mismatch between the predicted front and a specified target front. The efficiency of this control approach is demonstrated on a two dimensional reservoir model with two injectors and multiple producers. The aim is to maintain a uniform flow throughout the simulation.

1.3 Structural Health Monitoring

Numerous structural engineering problems exhibit non-linear dynamical behavior with uncertain and complex governing laws. With the recent technological advances, sensors and other monitoring devices became more accurate and abundant. Therefore, their use for health monitoring of civil structures gained popularity. The major problem that remained is to devise the proper mathematical models that can cope with and analyze the humongous measurements data flow. This has been a very active research area over the past decade [GS95, LBL02, ZFYM02, FBL05, FIIN05, GF06].

System identification and damage detection in most real life structures must be adapted to uncertainties and noise sources that can not be modeled as Gaussian processes. These uncertainties are typically associated with modeling, parametric, and measurement errors. In cases where these uncertainties are significant, standard identification and damage detection techniques are either unsuitable or inefficient. In this study, a system identification procedure based on coupling robust non-parametric non-linear models with the Polynomial Chaos methodology in the context of the Kalman Filtering techniques is presented. First a non-parametric representation of the system

nonlinearity is adopted. The Polynomial Chaos expansion is employed to characterize the uncertainty within the model, and then the proposed data assimilation technique is used to characterize the robust stochastic polynomial representation of the system's nonlinearity which enables monitoring the system's reaction and identifying any obscure changes within its behavior.

The presented methodology is applied for the structural health monitoring of a four story shear building subject to a ground excitation. The corresponding results prove that the proposed system identification techniques accurately detects changes in the system behavior in spite of measurement and modeling noises.

1.4 Summary of Original Contributions

This section provides a summary of the original contributions presented in this study:

- A stochastic sequential data assimilation technique based on forecasting the error statistics using the Polynomial Chaos methodology is presented. This enables the calibration of predictive models by characterizing the parameters as stochastic processes.

- The application of the proposed technique for reservoir characterization is an innovative approach that allows approximating the higher order statistics of the production forecast. The use of PC to represent the uncertainty in the reservoir model and the measurement help convey the actual medium in a more realistic way.

- A novel control approach using the Polynomial Chaos Kalman Filter is presented for optimizing the fluid front dynamics in porous media using rate control.

- The Polynomial Chaos methodology is used to characterize the uncertainty in robust non-parametric representations of structural systems nonlinearities. The latter is coupled with the developed data assimilation technique to render a robust structural health monitoring methodology.

The details of all the aforementioned approaches are presented in the following chapters.

1.5 Outline

Chapter two gives a review of the various methods used in this study to represent uncertainties. It presents the details of the Polynomial Chaos expansion, and shows the implementation details of the Spectral Stochastic Finite Element Method. Furthermore, chapter two present a model reduction technique to improve the computational efficiency of the Spectral Stochastic Finite Element Method for solving high dimensional stochastic systems.

In chapter three, sequential data assimilation techniques with a focus on the Kalman Filter and its various nonlinear extensions are discussed. The implementation details of a novel data assimilation technique based on coupling the Ensemble Kalman Filter with the Polynomial Chaos methodology are given.

Chapter four presents the basics of transport and flow in porous media. The governing flow equations are derived, and the various geological and petrophysical features of the reservoir and fluid parameters are discussed. The efficiency of automatic History matching using the proposed filter is demonstrated. Furthermore, a rate control technique for optimizing the sweep efficiency in the water flooding process is devised.

In chapter five, the proposed filter is applied for the system identification of highly nonlinear structures. Non-parametric representations of the structural nonlinearity are

discussed, and the setup is applied to the structural health monitoring of a four story shear building subject to a ground excitation.

Chapter six gives a description of SUNDANCE's capabilities. It presents the implementation details of the spectral library within SUNDANCE, and gives a detailed example as a guide for users to follow.

Finally, some concluding remarks and future research direction are mentioned.

Chapter 2

Uncertainty Representation

Uncertainty in physical data and phenomena could be attributed to two sources. The first one is the inherently irregular phenomena which could not be described deterministically, and the other is the phenomena that are not uncertain in nature but to which uncertainties could be attributed due to lack of available data [GS03]. Therefore, in order for any numerical model of a physical system to be useful, these uncertainties have to be accurately represented in the model and proper numerical schemes should be used for the analysis. To have a complete understanding of the approach used in this study, it is important to introduce some mathematical concepts.

Let H be the Hilbert space of functions [Ode79] defined over a domain D with values on the real line R. Denote by (Ω, Ψ, P) a probability space where Ω represents the domain, Ψ a measurable subset of the domain, and P is a measure on Ψ with $P(\Omega) = 1$. A random variable is defined as a mapping $\Omega \rightarrow P$. Let x be an element of D and θ an element of Ω. Then, Θ denotes the space of functions mapping Ω to the real line. The inner products over H and Θ are defined using the Lebesgue measure and the probability measure respectively. For any two elements $h_i(x)$ and $h_j(x)$ in H, the inner product is defined as,

$$(h_i(x), h_j(x)) = \int_D h_i(x) h_j(x) dx. \tag{2.1}$$

Similarly, for any two elements $\alpha(\theta)$ and $\beta(\theta)$ in Ω, the inner product is defined as,

$$(\alpha(\theta), \beta(\theta)) = \int_\Omega \alpha(\theta)\beta(\theta) dP, \tag{2.2}$$

where dP is a probability measure. Elements in the Hilbert spaces described above are said to orthogonal if their inner product is zero. A random process is then defined as a function in the product space $D \times \Omega$.

In what follows, some of the available techniques for addressing problems with stochasticity are reviewed. First different methods for representing the uncertainty are described, and then the details of the Spectral Stochastic Finite Element Method (SSFEM) as a tool for analyzing stochastic models is presented. The computational cost of the SSFEM depends on the mesh size, the dimension of the polynomial chaos expansion of the random variables, and the order of the polynomial chaos (PC) expansion of the solution. Specifically, the size of the linear system resulting from the SSFEM increases rapidly as the number of terms in the PC expansion grows. Therefore, a stochastic model reduction technique is presented to help cope with this problem.

2.1 Karhunen-Loeve Expansion

In cases where uncertainties possess a well defined covariance function, the Monte Carlo simulation is the most widely used method for representing the randomness. It consists of sampling the functions in a random, collocation like, scheme. Therefore, it requires a large ensemble of samples to accurately represent the process statistics. The Karhunen-Loeve expansion[Loe77] is a more theoretically appealing way to represent these functions. It is a Fourier-type series representation of random processes. It is based on the spectral decomposition of the covariance function of the stochastic process being represented. The expansion takes the following form:

$$Y(x,\theta) = \langle Y(x) \rangle + \sum_{i=1}^{\infty} \xi_i(\theta) \sqrt{\lambda_i} \phi_i(x) \tag{2.3}$$

where,

$$\langle \xi_i \rangle = 0, \quad \langle \xi_i \xi_j \rangle = \delta_{ij}, \quad i, j = 1, ..., \mu \quad \xi_0 \equiv 1. \tag{2.4}$$

In the above equation, θ is a random index spanning a domain in the space of random events, and the product $\xi_i(\theta)\sqrt{\lambda_i}$ represents the random amplitude associated with the deterministic shape functions $\phi_i(x)$, and $\langle . \rangle$ denotes the mathematical expectation operator. The sets $\{\phi_i\}_{i=1}^{\infty}$ and $\{\lambda_i\}_{i=1}^{\infty}$ are associated with the solution to the covariance kernel integral eigenvalue problem,

$$\int_D C(x_1, x_2)\phi_i(x_2)dx_2 = \lambda_i \phi_i(x_1) \tag{2.5}$$

where D denotes the spatial extent of the stochastic process, and $C(x_1, x_2)$ is the covariance function.

In practice, the Karhunen-Loeve expansion is usually truncated to a finite number of terms,

$$Y(x, \theta) = \langle Y(x) \rangle + \sum_{i=1}^{\mu} \xi_i(\theta)\sqrt{\lambda_i}\phi_i(x). \tag{2.6}$$

The number of terms is dependent on the distribution of the random pattern of the stochastic processes being modeled. The closer the process is to white noise, the more terms are required, while if a random variable is to be represented, a single term in the expansion is sufficient. This is directly correlated with eigenvalues resulting from solving the covariance kernel problem. The series is truncated after the first μ largest generalized eigenvalues of 2.5 such that $\sum_{i=1}^{\mu} \lambda_i / \sum_i \lambda_i$ is sufficiently close to one. The monotonic decay of the eigenvalues is guaranteed by the symmetry of the covariance function and the rate of the decay is related to the correlation length of the process being expanded. Figure 2.1 shows the monotonic decay of the eigenvalues λ_i.

The Karhunen-Loeve expansion is mean-squared convergent and optimal provided that the process being expanded has a finite variance. It is optimal in the sense that the

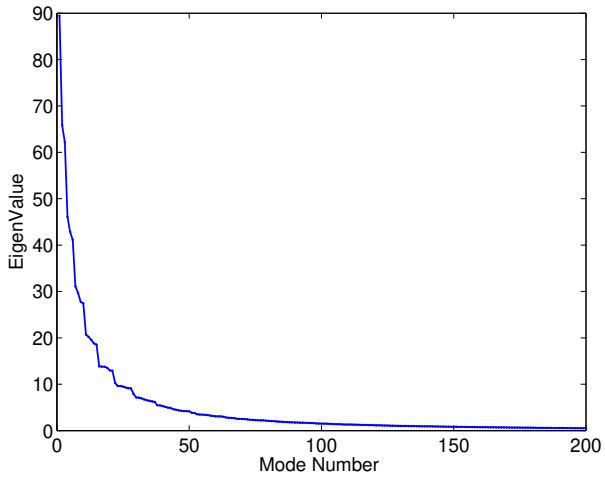


Figure 2.1: Contribution of successive scales of fluctuations in the KL expansion

mean square error resulting from a finite representation of the process $Y(x, \theta)$ is minimized. The disadvantage of the Karhunen Loeve expansion is that its use is limited to the prior knowledge of the covariance function. Therefore more general representation are necessary for representing different processes.

2.2 Polynomial Chaos Expansion

The Polynomial Chaos expansion is more general than the Karhunen Loeve expansion in the sense that it does not require prior knowledge of the covariance function. It is used to represent the solution of a stochastic partial differential equation which is usually a function of the problem parameters. The expansion involves an infinite basis set that completely spans the L_2 space of random variables, and whose elements are orthogonal polynomials. It can be shown [CM47] that, provided the solution has a finite variance,

this functional dependence could be expressed in terms of polynomials in Gaussian random variables, in the form,

$$u(x;\theta) = a_0(x)\Gamma_0 + \sum_{i_1=1}^{\infty} a_{i_1}(x)\Gamma_1(\xi_{i_1}(\theta)) + \sum_{i_1=1}^{\infty}\sum_{i_2=1}^{\infty} a_{i_1 i_2}(x)\Gamma_2(\xi_{i_1}(\theta)\xi_{i_2}(\theta)) + \dots \tag{2.7}$$

In this equation, $\Gamma_n(\xi_{i_1}, \dots, \xi_{i_n})$ denotes the n^{th} order polynomial chaos in the variables $(\xi_{i_1}, \dots, \xi_{i_n})$. These are generalizations of the multidimensional Hermite polynomials, and $a_{i_1,\dots,i_N}$ are deterministic coefficients in the expansion [Wie38]. Upon introducing a one to one mapping to a set of ordered indices $\{\psi_i\}$ and truncating the polynomial chaos expansion after the P^{th} term, the above equation can be written as

$$u(x;\theta) = \sum_{j=0}^{P} u_j(x)\psi_j(\theta). \tag{2.8}$$

These polynomials are orthogonal with respect to the joint probability measure of $(\xi_{i_1}, \dots, \xi_{i_n})$; i.e. their inner product, $\langle \psi_j \psi_k \rangle$, is equal to zero for $j \neq k$. The j^{th} order polynomial, $\psi_j(\xi)$ can be explicitly evaluated as,

$$\psi_o(\xi) = 1 \tag{2.9}$$

$$\psi_1(\xi_{i_1}) = \xi_{i_1} \tag{2.10}$$

$$\psi_2(\xi_{i_1}, \xi_{i_2}) = \xi_{i_1}\xi_{i_2} - \delta_{i_1 i_2} \tag{2.11}$$

$$\psi_3(\xi_{i_1}, \xi_{i_2}, \xi_{i_3}) = \xi_{i_1}\xi_{i_2}\xi_{i_3} - \xi_{i_1}\delta_{i_1 i_2} - \xi_{i_2}\delta i_1 i_3 - \xi_{i_3}\delta_{i_1 i_2} \tag{2.12}$$

where δ_{ij} is the Kronecker delta. In general, the Polynomial Chaos of order n can be obtained as,

$$\psi_n(\xi_{i_1}, \ldots, \xi_{i_n}) = (-1)^n e^{\frac{1}{2}\xi^T\xi}(\frac{\partial^n e^{\frac{-1}{2}\xi^T\xi}}{\partial\xi_{i_1}\ldots\xi_{i_n}}). \tag{2.13}$$

Once the deterministic coefficients $\{u_i\}$ are calculated, a complete probabilistic characterization of the stochastic process is achieved.

A truncated Polynomial Chaos can be refined along the random dimension by either adding more random variables to the set $\{\xi_i(\theta)\}$ or by increasing the order of the polynomials in the PC expansion. Note that the total number of terms $P+1$ in a PC expansion with order less than or equal to p in M random dimensions is given by

$$P + 1 = \frac{(p+M)!}{M!p!}. \tag{2.14}$$

2.3 Polynomial Chaos Representation of Some Functions of Stochastic Processes

In many stochastic systems, one is often faced with the obstacle of efficiently representing functions of stochastic processes be they dependent or independent, Gaussian or Non-Gaussian. These function could be ratios, products, or even polynomials of stochastic processes. As long as these functions belong to L_2, they admit their own Polynomial Chaos representations. In this study algorithms are devised to perform these functions on the PC expansion of the random processes.

2.3.1 The Lognormal Process

Consider the process $l(x,z)$ obtained from the Gaussian field $Y(x,z)$ via exponentiation. This process is a stochastic field having a lognormal marginal probability distribution. It can be represented in terms of multidimensional Hermite polynomials orthogonal with respect to the Gaussian measure [Gha99],

$$l(x,z;\theta) = \sum_{i=0}^{P} \psi_i(\theta) l_i(x,z). \tag{2.15}$$

The zeroth order term in the polynomial chaos expansion refers to the mean of the process and is given by,

$$l_0(x,z) = \exp[\mu_Y + \frac{\sigma_Y^2}{2}] \tag{2.16}$$

where μ_Y and σ_Y denote the mean and standard deviation of the Gaussian random field Y respectively. The higher order terms are

$$l_i(x,z) = \frac{\langle l(x,z)\psi_i \rangle}{\langle \psi_i^2 \rangle} = \frac{\left\langle e^{Y(x,z)}\psi_i \right\rangle}{\langle \psi_i^2 \rangle}. \tag{2.17}$$

The denominator can be easily evaluated and tabulated, and the numerator could be expressed as [Gha99]

$$\langle l(x,z)\psi_i \rangle = \exp[Y_0(x,z) + \frac{1}{2}\sum_{j=1}^{N} Y_j^2(x,z)] \left\langle \psi_i(\eta) \right\rangle, \tag{2.18}$$

where $\langle \psi_i(\eta) \rangle$ represents the average of the polynomial chaos centered around Y_i. Consequently, the lognormal process can be represented as,

$$l(x,z;\theta) = l_o(x,z)(1 + \sum_{i=1}^{N} \xi_i(\theta) Y_i(x,z) +$$

$$\sum_{i=1}^{N}\sum_{j=1}^{N}\frac{(\xi_i(\theta)\xi_j(\theta)-\delta ij)}{\langle(\xi_i\xi_j-\delta ij)^2\rangle}Y_i(x,z)Y_j(x,z)+...). \tag{2.19}$$

2.3.2 Product of Two or More Stochastic Processes

Consider two random processes, $A(x,\theta)$ and $B(x,\theta)$, and their respective Polynomial Chaos approximations:

$$\hat{A}(x,\theta)=\sum_{i=0}^{P}A_i(x)\psi_i(\xi), \tag{2.20}$$

$$\hat{B}(x,\theta)=\sum_{j=0}^{P}B_j(x)\psi_j(\xi). \tag{2.21}$$

We need to determine the PC expansion of the process, C, obtained from the product of A and B,

$$\hat{C}(x,\theta)=\sum_{k=0}^{P}C_k(x)\psi_k(\xi)=\sum_{i=0}^{P}A_i(x)\psi_i(\xi)\sum_{j=0}^{P}B_j(x)\psi_j(\xi). \tag{2.22}$$

The C_k coefficients are obtained by projection on a higher order chaos.

$$C_k(x)=\sum_{i=0}^{P}\sum_{j=0}^{P}A_i(x)B_j(x)C_{ijk} \qquad \forall k\in 0,\ldots,P \tag{2.23}$$

where

$$C_{ijk}=\frac{\langle\psi_i\psi_j\psi_k\rangle}{\langle\psi_k^2\rangle}. \tag{2.24}$$

This can be easily extended to the product of three processes,

$$D_l(x)=\sum_{i=0}^{P}\sum_{j=0}^{P}\sum_{k=0}^{P}A_i(x)B_j(x)C_k(x)D_{ijkl} \qquad \forall l\in 0,\ldots,P \tag{2.25}$$

where

$$D_{ijkl} = \frac{\langle \psi_i \psi_j \psi_k \psi_l \rangle}{\langle \psi_l^2 \rangle}. \tag{2.26}$$

For computing the PC expansion of a polynomial, each product is computed separately using the above machinery, and then addition and subtraction are carried on. Addition and Subtraction are performed by adding/subtracting the corresponding PC coefficients of the variables being added/subtracted.

2.3.3 Ratio of Two Stochastic Processes

Consider the stochastic process given by a ratio of two different stochastic processes

$$C(x;\theta) = \frac{A(x;\theta)}{B(x;\theta)} \tag{2.27}$$

using the polynomial chaos expansion to represent the latter equation yields,

$$\sum_{k=0}^{P} C_k(x)\psi_k(\xi) = \frac{\sum_{i=0}^{P} A_i(x)\psi_i(\xi)}{\sum_{j=0}^{P} B_j(x)\psi_j(\xi)} \tag{2.28}$$

cross-multiplying, and requiring the difference to be orthogonal to the approximating space spanned by the Polynomial Chaos $\{\psi_l\}_{l=0}^{P}$ yields,

$$\sum_{k=0}^{P}\sum_{j=0}^{P} C_k(x)B_j(x)\frac{\langle \psi_j \psi_k \psi_l \rangle}{\langle \psi_l^2 \rangle} = A_l(x) \qquad \forall l \in 0, \ldots, P. \tag{2.29}$$

Expanding the equation for all values of l results in a system of linear algebraic equations which when solved gives the deterministic chaos coordinates of C. The size of the resulting system is equal to the number of terms used in the chaos representation of the approximation.

2.4 Uncertainty Propagation: The Spectral Stochastic Finite Element Method

The SSFEM proposed by Ghanem and Spanos [GS03] is an extension of the deterministic finite element method FEM to stochastic boundary value problems. In the context of SSFEM, the randomness in the problem is regarded as an additional dimension with an associated Hilbert space of L_2 functions in which a set of basis functions is identified. This set is known as Polynomial Chaos and is used to discretize the random dimension [GS03, Wie38].

In the deterministic finite element method, the spatial domain is replaced by a set of nodes that represent the finite element mesh and the resulting solution is expressed in terms of nodal degrees of freedom $u_i, i = 1, \ldots, N$ augmented into a vector U. The strain energy V^e stored in each element A^e is expressed as,

$$V^e = \frac{1}{2} \int_{A^e} \sigma^T(x, z)\epsilon(x, z) dA^e \tag{2.30}$$

where dA^e is a differential element in A^e, and $\sigma(x, z)$ and $\epsilon(x, z)$ are the stress and strain vector respectively. For simplicity, assume that the model specified for the analysis is a plane strain linearly elastic one, the stress maybe expressed in terms of the strain as,

$$\sigma = D^e \epsilon \tag{2.31}$$

where D^e is the stochastic plane strain matrix of constitutive relations. The two dimensional displacement vector $u(\theta)$ representing the longitudinal and transverse displacements within each element can be expressed in terms of the nodal displacements of the element in the form

$$u(\theta) = N^e(r, s)U^e(\theta) \tag{2.32}$$

where $N^e(r,s)$ is the local interpolation matrix, $U^e(\theta)$ is the random nodal response vector, and r and s are local coordinates over the element. The total strain energy V is obtained by summing the contributions from all the elements. This procedure gives

$$V = \frac{1}{2}\sum_{e=1}^{N_e} U^{eT} \int_0^1 \int_0^1 B^{eT} D^e(x,z;\theta) B^e |J^e| dr ds U^e \tag{2.33}$$

where $|J^e|$ denotes the determinant of the Jacobian of the transformation that maps an arbitrary element (e) onto the three-nodded triangle with sides equal to one, and B^e is the matrix that describes the dependence of strains on displacements.

The polynomial chaos expansion of the matrix of constitutive properties may be substituted in the above equation to transform the latter into,

$$V = \frac{1}{2} U^T \sum_{k=0}^{P} K_k \psi_k(\theta) U. \tag{2.34}$$

Minimizing the total potential energy with respect to U, and expanding the response vector U using Polynomial Chaos, lead to the following equation

$$\sum_{i=0}^{P_1} K_i \psi_i \sum_{j=0}^{P_2} U_j \psi_j - F = 0 \tag{2.35}$$

where F is the vector of nodal forces, and U_j is the deterministic coordinate vector of the response and is evaluated as the solution of the following system of algebraic equations

$$\sum_{j=0}^{P_2} K_{jk} U_j = F_k, \qquad k = 0, 1, 2, ..., P_2 \tag{2.36}$$

where,

$$K_{jk} = \sum_{i=0}^{P_1} C_{ijk} K_i \tag{2.37}$$

$$F_k = \langle \psi_k F \rangle \tag{2.38}$$

$$C_{ijk} = \langle \psi_i \psi_j \psi_k \rangle \, . \tag{2.39}$$

The size of the resulting linear system could rapidly increase with the growing number of terms in the polynomial chaos expansion, and with a fine mesh, one could foresee significant computational challenges. This emphasizes the need for establishing a reduction method that computationally simplifies the problem while preserving the accuracy and veracity of the SSFEM.

2.5 Stochastic Model Reductions

The stochastic model reduction for chaos representation method [DGRH] aims at obtaining an alternative to the polynomial chaos basis for representing the response. In order to do so, the method involves solving the problem using the complete order polynomial chaos basis on a coarse mesh. From the resulting solution, the covariance kernel of the response is estimated, and used to approximate the dominant Karhunen-Loeve representation of the response,

$$u(x, z; \theta) = \langle u(x, z) \rangle + \sum_{i=1}^{\mu} \sqrt{\nu_i} \eta_i(\theta) \phi_i(x, z). \tag{2.40}$$

Moreover, noting the differentiability properties of the response u, it has been suggested [DGRH] that optimality in a more adapted functional space can be achieved by using a modified version of the Karhune-Loeve expansion known as the Hilbert-Karhunen-Loeve representation. Accordingly the sets $\{\nu_i\}$ and $\{\phi_i\}$ are associated with the solution of the following eigenvalue problem

$$(R(., x_2, z_2), \phi_i(x_2, z_2))_{E(D)} = \nu_i \phi_i(x_1, z_1) \quad \forall i. \tag{2.41}$$

where $R(x_1, z_1, x_2, z_2) = \langle (u(x_1, z_1) - \langle u(x_1, z_1) \rangle)(u(x_2, z_2) - \langle u(x_2, z_2) \rangle) \rangle$, and $(.,.)_{E(D)}$ denotes the inner product induced by the energy norm associated with the mean of the process $D(x, z)$. The random variables $\{\eta_i\}$ characterized on the coarse mesh are assumed to retain their joint probabilistic measure as the solution is approximated on the fine mesh. Furthermore, each of the η_i's is expressed in a Polynomial Chaos representation that is assumed to be well approximated on the coarse mesh. This method was applied for a Benchmark study that was presented in ICOSSAR'05 and whose details will be presented in what follows [GSD07].

2.6 Application to the Embankment Dam Problem

The embankment dam problem of the benchmark study is treated using the newly developed Stochastic Model Reduction for Polynomial Chaos Representations method. The elastic and shear moduli of the material, in the present problem, are modeled as two stochastic processes that are explicit functions of the same process possessing a relatively low correlation length. The state of the system can thus be viewed as a function defined on a high-dimensional space, associated with the fluctuations of the underlying process. In such a setting, the spectral stochastic finite element method (SSFEM) for the specified spatial discretization is computationally prohibitive. The approach adopted in this paper enables the stochastic characterization of a fine mesh problem based on the high dimensional polynomial chaos solution of a coarse mesh analysis. The problem to be examined is that of an embankment dam with trapezoidal cross-section made of earth or rock fill subject to compressive deterministic loading conditions as is shown in

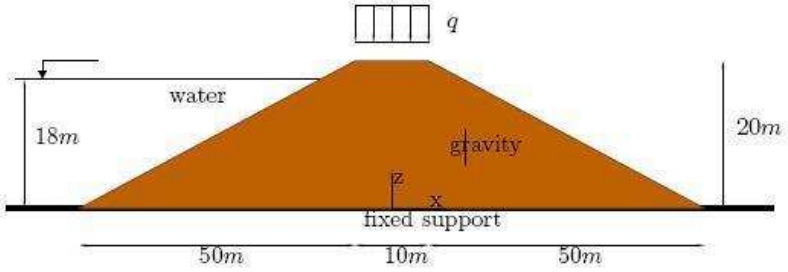


Figure 2.2: Schematic of the Embankment Dam

Figure 2.2. Due to the nature of the material, it is proposed to model the low strain elastic and shear moduli as non-homogeneous lognormal random fields $E(x,z)$ and $G(x,z)$ respectively,

$$E(x,z) = m_E(z) + \sigma_E(z)\frac{e^{Y(x,z)} - m_y}{\sigma_y} \tag{2.42}$$

$$G(x,z) = m_G(z) + \sigma_G(z)\frac{e^{Y(x,z)} - m_y}{\sigma_y}, \tag{2.43}$$

where $m_E(z)$ and $m_G(z)$ are the means, $\sigma_E(z)$ and $\sigma_G(z)$ are the standard deviations, $Y(x,z)$ is a homogeneous, zero mean, unit variance Gaussian random field with given autocorrelation function $R_Y(\Delta x, \Delta z)$ given by eq. 2.44, $m_Y = E[e^Y] = e^{0.5}$ and $\sigma_Y^2 = Var[e^Y] = e^2 - e$ [ICO04],

$$R_Y(\Delta X, \Delta Z) = \exp(-\frac{|\Delta X|}{10} - \frac{|\Delta Z|}{3}). \tag{2.44}$$

Table 2 of [ICO04] presents the properties of the low strain elastic parameters. The elastic moduli are represented as dependent stochastic processes possessing a relatively low correlation length, and consequently the problem can be viewed as a function defined on a very high dimensional space. The spatial domain is discretized into triangular, 3-node, plane strain finite elements. Figure 2.3 shows the proposed mesh of 2160 elements.

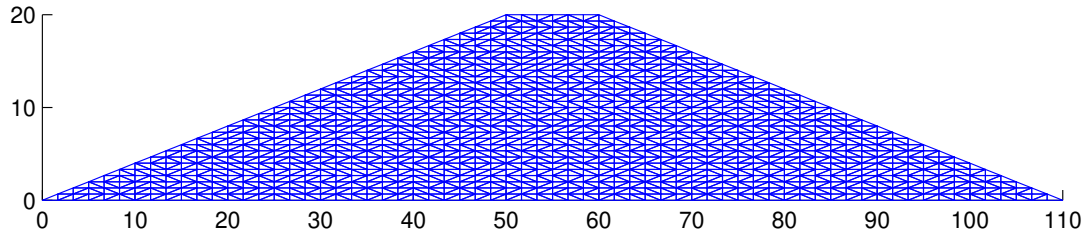


Figure 2.3: Proposed Mesh of 2160 elements

2.6.1 Implementation Details

First, the Karhunen-Loeve expansion is employed to help reduce the dimensionality of the problem. A Galerkin type procedure is applied to transform the Fredholm equation to a generalized eigenvalue problem which is solved for the eigenvalues λ_i's and eigenvectors ϕ_i's of the covariance kernel. Figure 2.4 shows the computed eigenvalues of the given covariance kernel. It shows that the decay of the eigenvalues is rather slow because of the relatively low correlation length provided; this indicates the necessity of using a big number of terms in the KL expansion.

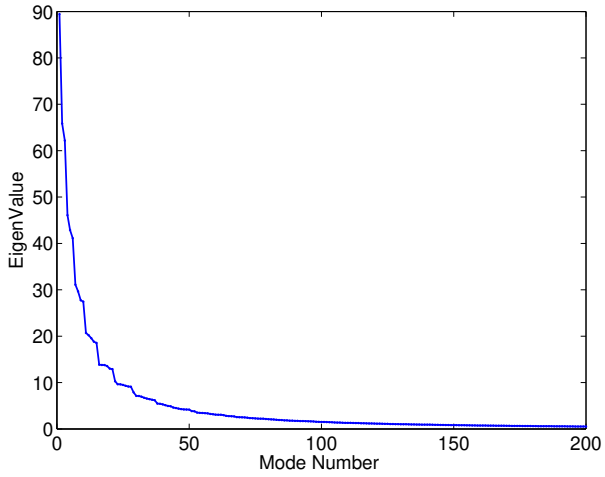


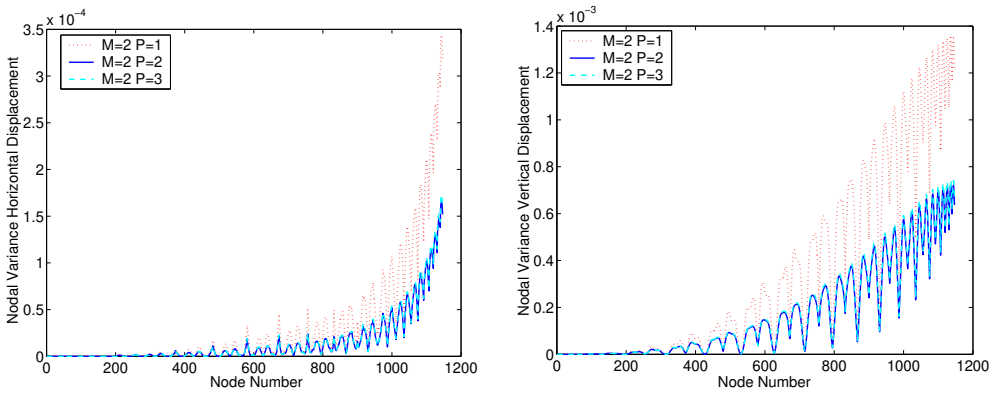
Figure 2.4: Eigenvalues λ_n of the Exponential Covariance Kernel

Knowing the significance of each of the random dimensions in the problem, the total number of terms in the polynomial chaos representation of the response must be

decided. Many terms are often used in the Polynomial Chaos expansion. For stability issues, the minimum order of chaos expansion of the process $D(x, z)$ should be at least twice that of the response u [MK04]. This way the computational cost of the solution is greatly reduced without sacrificing the method's accuracy. The problem at hand is solved for various combinations of the number of terms in the the KL and increasing orders of PC expansions to test the convergence of the response.

Due to computational difficulties in experimenting with high dimensional solutions, the problem was first solved, based on the proposed fine mesh, using a relatively low number of dimensions in the KL expansion, and the solution was investigated for second, third, and fourth order PC expansions. It turns out, as is shown in figure 2.5 that the significance of adding the third order terms is negligible. Figures 2.5 a and b show the nodal variances convergence of the the horizontal and vertical displacements for 2 terms in the KL expansion of the solution $(M = 2)$ and as the order of the PC expansion (p) is increased. Similar results were obtained for 3 and 4 terms in the KL expansion, and results obtained using the stochastic model reductions for 15 and 20 terms in the KL expansion confirmed that the second order is sufficient to portray the uncertainty in the response. Although the SSFEM converged for a second order polynomial chaos expansion, results were still vastly remote from the solution obtained through a Monte Carlo simulation of the problem. This implies that more terms in the KL expansion are needed.

As noted earlier, as the number of terms in the polynomial chaos expansion increases, the computational cost of SSFEM rises significantly. The size of the resulting algebraic system that is to be solved for the deterministic coefficients of the response is $N(P + 1) \times N(P + 1)$ where N is the number of nodal degrees of freedom.



(a) (b)

Figure 2.5: The Significance of third order chaos expansion on the response.

For the embankment dam problem, the proposed mesh has 2294 degrees of freedom. In order to attain convergence, it was determined through computational experimentation using the stochastic model reduction method that about 60 terms in the KL expansion are needed. The corresponding total number of terms needed in the polynomial chaos second order expansion is 1891 which leaves us with a linear algebraic system in 4337954 unknowns. Solving such a system requires a super computer with very high memory capacity.

To solve this problem, the stochastic model reduction method is used. A coarse mesh of 60 elements shown in figure 2.6 is used to carry out the full SSFEM analysis. The solution obtained from solving the problem on the coarse scale is expressed as,

$$U(\theta) = \sum_{i=0}^{P} U_i \psi_i(\theta) \tag{2.45}$$

where, U is an $N \times 1$ vector containing the horizontal and vertical nodal displacements. Having the above approximation, the covariance kernel of the response is estimated as

$$R_{UU} = \langle (\sum_{i=1}^{P} U_i \psi_i)(\sum_{j=1}^{P} U_j \psi_j)^T \rangle = \sum_{i=1}^{P} U_i U_i^T \left\langle \psi_i^2 \right\rangle \tag{2.46}$$

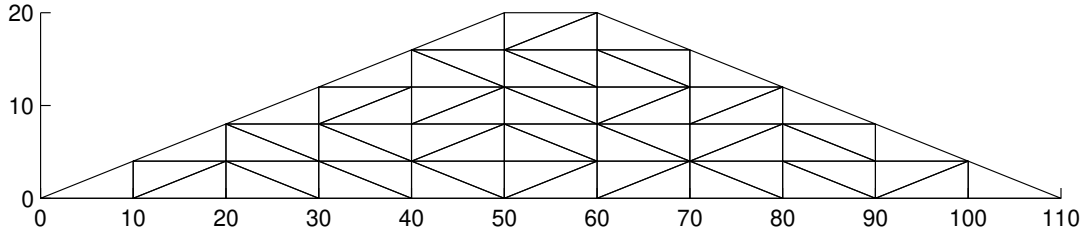


Figure 2.6: Coarse Mesh of 60 elements

Now, the Hilbert-Karhunen-Loeve expansion can be obtained by solving equation 2.41 using a Galerkin projection scheme on the coarse mesh, which will lead to a discrete generalized eigenvalue problem whose size is determined by the number of degrees of freedom of the coarse discretization. In discrete form, the eigenvalue problem becomes,

$$K_0^T R K_0 f = \nu K_0 f \tag{2.47}$$

where K_0 is the stiffness matrix associated with the mean of the process $D(x, z)$ and obtained from the SSFEM solution on the coarse mesh. It should be noted that K_0 is readily available from the current analysis on the coarse mesh. The matrix R is the covariance of the solution, ν is a diagonal matrix whose elements are the eigenvalues of the generalized eigenvalue problem, and f a matrix containing the corresponding eigenvectors. The Hilbert-Karhunen-Loeve representation is given by

$$u(x, z; \theta) = \langle u(x, z) \rangle + \sum_i \sqrt{\nu_i} f_i(x, z) \eta_i(\theta) \tag{2.48}$$

The above expansion can be truncated after the first μ largest generalized eigenvalues of (2.47) such that $\sum_{i=1}^{\mu} \nu_i / \sum_i \nu_i$ is sufficiently close to one. Now, the set of random

variables $\{\eta_i\}_{i=1}^{\mu}$ are recasted by a linear transformation of the set of the polynomial chaos basis $\{\psi_i\}_{i=1}^{P}$ [DGRH].

$$\eta_i(\theta) = \sum_{j=1}^{P} \alpha_{ij}\psi_j(\theta) \qquad \forall i \tag{2.49}$$

where

$$\alpha_{ij} = \frac{f_i^T K_0 U_j}{\sqrt{\nu_i}}; \qquad \forall i, j. \tag{2.50}$$

The fine scale problem involves the finite dimensional space of piecewise continuous polynomials corresponding to the spatial discretization shown in figure 2 and the space of random variables spanned by the basis $\{\eta_i\}_{i=1}^{\mu}$ [DGRH]. The application of the SSFEM becomes

$$\sum_{i=0}^{P} K_i\psi_i \sum_{j=0}^{\mu} U_j\eta_j - F = 0 \tag{2.51}$$

$$\sum_{j=0}^{\mu} K_{jk}U_j = F_k, \qquad k = 0, 1, 2, \ldots, \mu \tag{2.52}$$

where,

$$K_{jk} = \sum_{i=0}^{P} d_{ijk}K_i \tag{2.53}$$

$$F_k = \langle \eta_k F \rangle \tag{2.54}$$

$$d_{ijk} = \langle \psi_i\eta_j\eta_k \rangle \tag{2.55}$$

$$\begin{aligned} d_{ijk} &= \left\langle \psi_i \left(\sum_{m=1}^{P} \alpha_{jm}\psi_m \right) \left(\sum_{n=1}^{P} \alpha_{kn}\psi_n \right) \right\rangle = \sum_{m=1}^{P}\sum_{n=1}^{P} \alpha_{jm}\alpha_{kn} \langle \psi_i\psi_m\psi_n \rangle \\ &= \sum_{m=1}^{P}\sum_{n=1}^{P} \alpha_{jm}\alpha_{kn}c_{imn} \qquad \forall\ j, k. \end{aligned} \tag{2.56}$$

Where the coefficients $c_{imn} := \langle \psi_i\psi_m\psi_n \rangle$ are already tabulated [GS03].

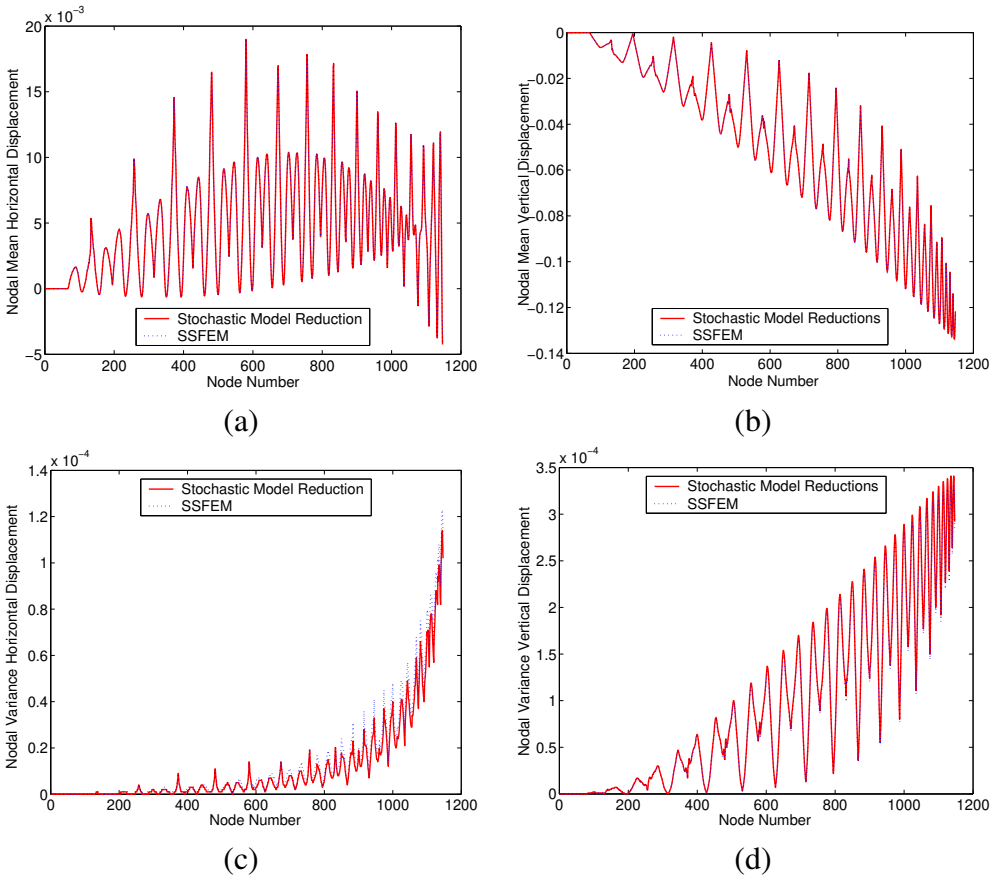


Figure 2.7: The response representation using a 15 dimensional second order expansion obtained via SSFEM and SSFEM with Stochastic Model Reductions

The stochastic model reduction is first verified on a reduced version of the problem. The second order solution in 15 stochastic dimensions is obtained using only SSFEM and the combination of SSFEM and stochastic model reductions. The results show great compatibility and are presented in figure 2.7.

The problem is solved using the stochastic model reduction for chaos expansions for a range of terms in the KL expansion starting with 15 dimensions, for which comparison is made with SSFEM, and ending with 60 dimensions which gives close results to those obtained from a Monte Carlo simulation. Using a sixty dimensional polynomial expanded up to second order, 1891 bases are required to capture the uncertainty of the response using SSFEM analysis on the coarse scale problem. Using the coarse mesh

solution to obtain the covariance matrix of the response, the eigenvalue problem (2.47) is solved to obtain the number of significant basis for the fine mesh analysis. Figure 2.8 gives the decay of the eigenvalues of (2.47). Note that 29 new bases are sufficient to capture the uncertainty in the response for the fine scale problem. The response obtained using these 29 basis in a SSFEM analysis on the fine scale is presented in the analysis section. Clearly the computation costs associated with 29 bases in random dimensions are dramatically less than those of the full order SSFEM with 1891 bases.

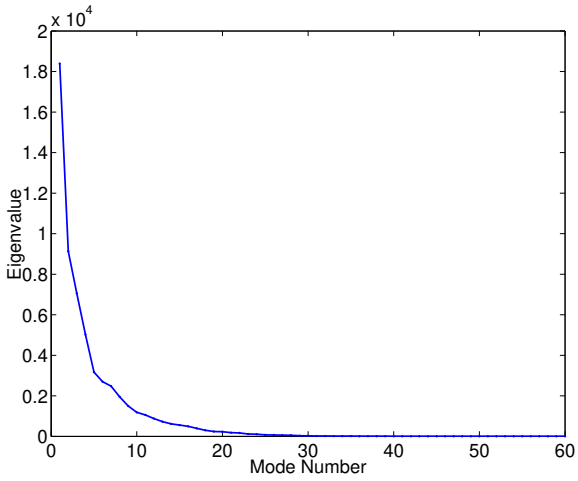


Figure 2.8: Eigenvalues of equation 2.47

2.6.2 Analysis of Results

To assess the validity of the results obtained by the SSFEM and test its effectiveness and convergence, the same problem is also treated using a Monte Carlo Simulation. The Monte Carlo simulation logic involves solving the whole system once for every realization in a statistical sample associated with the random system, and consequently synthesizing a corresponding sample of the random solution [Sch01b]. The elastic and shear moduli are simulated by sampling an n-dimensional Gaussian random vector and

simulating the field $Y(x,z)$ based on the solution of the integral eigenvalue problem of the covariance kernel given by 2.44.

$$Y(x,z;\theta) = Y_0(x,z) + \sum_{i=1}^{N} \sqrt{\lambda_i}\phi_i(x,z)\xi_i(\theta) \tag{2.57}$$

For each simulation, the response is obtained by solving the deterministic finite element problem. After 150,000 simulations, the solution seems to have converged in both mean and variance. The results obtained from the 150,000 Monte Carlo simulations are plotted together with those from SSFEM with 60 random dimensions expanded up to second order in figure 2.9. The responses obtained from the two methods seem to be very coherent, although while using the same machine for the computations, the Monte Carlo simulations take about 24 hours while the SSFEM with stochastic model reductions take less than 8 hours.

The goal behind solving the problem is estimating the probability of failure of the embankment dam for different combinations of the cohesive strength c, the friction angle ϕ, the soil mass density γ, and the top load q. Failure is defined when the Mohr-Coulomb criterion is satisfied in at least one point in the analysis domain,

$$\tau \geq c + \sigma_n tan\phi \tag{2.58}$$

where τ is the shear stress, and σ_n is the normal stress.

For plane strain conditions, element stresses in the analysis plane are calculated as:

$$\sigma = DBU, \tag{2.59}$$

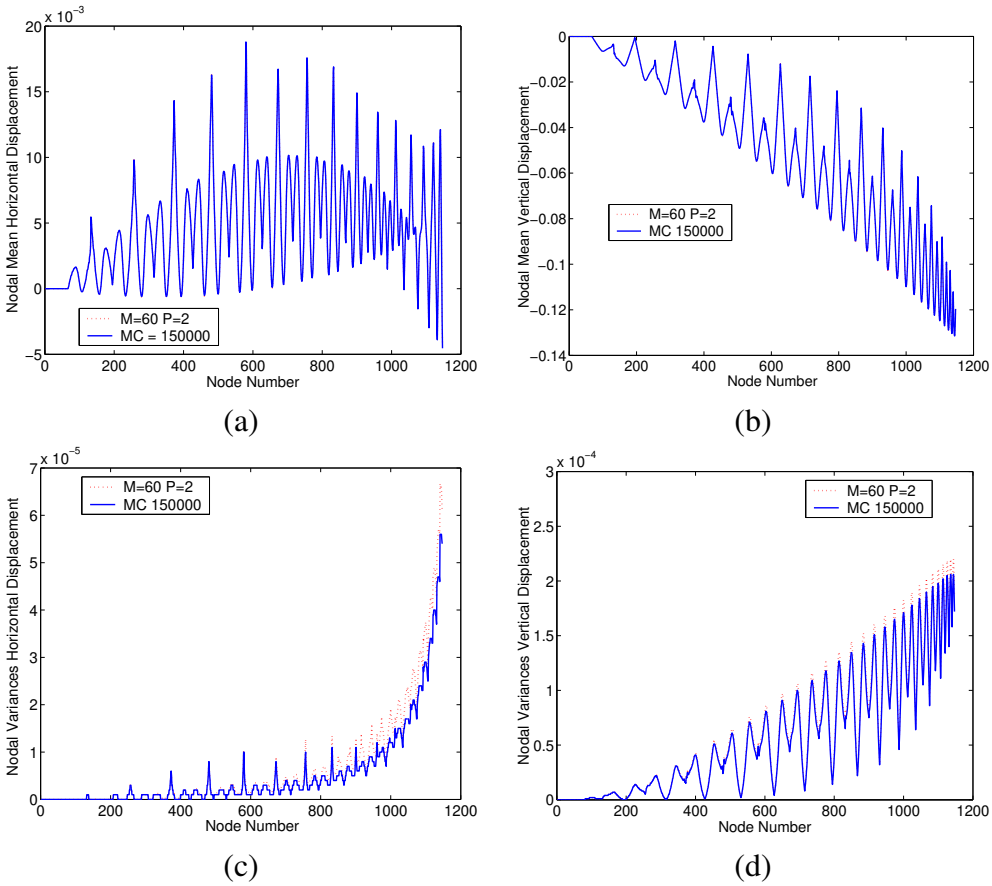


Figure 2.9: The response's representation obtained from 150000 Monte Carlo Simulations as Compared to those resulting from a 60 dimensional second order Chaos expansion

where σ is a 3-dimensional vector whose components are σ_x, σ_y, and τ_{xy} respectively. Representing these terms by their Polynomial Chaos decompositions yields,

$$\sum_{i=0}^{P_2} \sigma_i \eta_i = \sum_{i=0}^{P_2} \sum_{j=0}^{P_1} D_j B U_i \eta_i \psi_j. \tag{2.60}$$

Accordingly, the deterministic shapes of the stress vector are calculated from those of the constitutive relations matrix and response vector via the above relation by projecting on a higher order chaos. Consequently, the stress vector corresponding to each element in the mesh, is represented by a polynomial function of sixty Gaussian random variables.

Table 2.1: Probability of failure for different load and material strength combinations

	1	2	3
Cohesive Strength c	125 KPa	225 KPa	150 KPa
Friction Angle ϕ	30^o	22^o	40^o
Mass Density γ	$1800Kg/m^3$	$1800Kg/m^3$	$1800Kg/m^3$
Top Load q	30 KPa	30 KPa	30 KPa
$P_f\ (MonteCarlo)$	1.70 E -03	2.20 E-05	3.40E-04
$P_f\ (M = 60\ P = 2)$	2.0E-03	2.0E-05	6.5E-04

This vector is simulated by sampling the ξ's and the Mohr-Coulomb failure criteria is checked for each element.Clearly, the computational cost of sampling the basis is negligible as compared to solving the problem via a Monte Carlo Simulation. Failure is defined when the Mohr-Coulomb failure criterion is satisfied in at least one element in the domain. 100,000 samples of the stress vector are simulated and the failures obtained are presented in table 1.

Chapter 3

Sequential Data Assimilation for Stochastic Models

3.1 Overview of Data Assimilation

Data assimilation is a novel and versatile methodology for estimating unknown state variables and parameters. It relies on a set of observational data and the underlying dynamical principles governing the system under observation. General schemes for data assimilation often relate to either estimation theory or control theory, but some approaches like direct minimization, stochastic and hybrid methods can be used in both frameworks. Most of these schemes are based on the least-squares criteria which have had a great success. The optimality of the least squares may degenerate in the cases of very noisy and sparse data, or in the presence of multi-modal probability density functions. In such situations melding criteria, such as the maximum likelihood, minimax criterion or associated variations might be more appropriate. [RL00].

3.1.1 Control Theory

Control theory or variational assimilation approaches such as the generalized inverse and adjoint methods perform a global time-space adjustment of the model solution to all observations and thus solve a smoothing problem. The aim of these methods is to minimize a cost function reducing the time space misfit between the simulated data and the observations, with the constraints of the model equations and their parameters. The

dynamical model could be either considered as a strong constraint, i.e. it is to be exactly fulfilled, or a weak one.

In the case where the dynamical system, $F(\theta) = 0$, describing the temporal evolution of the models state variable θ is a strong constraint, the variational approach is known as the adjoint method. Within the assimilation period the initial and boundary conditions and the parameter values control the evolution of the model variables. Variables describing these conditions are commonly denoted as control variables, u. The approach consists of minimizing a cost function J resulting from the summation of two components. The first weights the uncertainties in the initial conditions, boundary conditions and parameters with their respective a priori error covariances. The other is the sum over time of all data-model misfits at observation locations, weighted by measurement error covariances [RL00]. This optimization procedure is called the adjoint method, because the adjoint model equations offer a sophisticated but relative cheap way to calculate the gradient of the cost function. This is done by introducing the Lagrange function,

$$\mathcal{L} = J + \int_D \int_T \lambda(x,t) F(\theta, x, t) dt dx, \tag{3.1}$$

where D and T are the spatial and temporal domains respectively. Partial differentiation with respect to the Lagrange multipliers , also denoted as adjoint variables, returns the model equations:

$$\frac{\partial \mathcal{L}}{\partial \lambda} = F(\theta) = 0, \tag{3.2}$$

while the differentiation with respect to θ yields the adjoint model equations, which describe the temporal evolution of the Lagrange multipliers:

$$\frac{\partial \mathcal{L}}{\partial \theta} = Adj(\lambda) + \frac{\partial J}{\partial \theta} = 0. \tag{3.3}$$

The third condition entails differentiating with respect to the control variables u:

$$\frac{\partial \mathcal{L}}{\partial u} = 0. \tag{3.4}$$

The latter insures that the optimal choice of the control variables u have been selected. Under the above condition the gradient $\nabla_u J$ can be easily calculated as:

$$\begin{aligned} \nabla_u J = \nabla_u \mathcal{L} &= \frac{\partial \mathcal{L}}{\partial u} && (3.5) \\ &= \frac{\partial \mathcal{L}}{\partial u} + \frac{\partial \mathcal{L}}{\partial \lambda}\frac{\partial \lambda}{\partial u} + \frac{\partial \mathcal{L}}{\partial \theta}\frac{\partial \theta}{\partial u}. && (3.6) \end{aligned}$$

The second and third term in the above equation vanish since $\frac{\partial \mathcal{L}}{\partial \theta} = 0$ and $\frac{\partial \mathcal{L}}{\partial \lambda} = 0$, and thus the calculation of the gradient $\nabla_u J$ simplifies to,

$$\nabla_u J = \frac{\partial J}{\partial u} + \int_D \int_T \lambda(x,t) \frac{\partial F(\theta, x, t)}{\partial u} dt dx. \tag{3.7}$$

The generalized inverse method is used when the inverse problem is expanded to weakly fit the constraints of both the data and the dynamics. Using this approach, the estimate is still defined in a least square sense; the cost function to be minimized is the same as that in the adjoint method, but a third term consisting of the dynamical model uncertainties weighted by a priori model error covariances is now added. The dynamical model uncertainties thus couple the state evolution with the adjoint evolution. This coupling renders the iterative solution of the backward and forward equations difficult [RL00].

3.1.2 Direct Minimization Methods

These methods aim at minimizing cost functions similar to those defined for the generalized inverse problem but without utilizing the Euler-Lagrange equations. Iterative methods are generally used to determine the direction of the descending cost function. At each iteration, a line search minimization is performed to determine the lowest descending direction. The method of steepest descend is the simplest of these approaches, but it is the slowest since it converges linearly. The conjugate-gradient method on the other hand calculates the search direction orthogonal to the local Hessian Matrix. It has a good convergence rate and a low storage requirements. Other descent methods are the Newton and quasi-Newton methods, but the problem with all these approaches is that they are very sensitive to the choice of initial conditions. This is because they are basically local minimization schemes.

When the cost functions become sufficiently non-linear, non-local methods become more attractive. Of these methods, it is important to note the methods of simulated annealing and Genetic algorithms. In the simulated annealing method, each point s of the search space is compared to a state of some physical system, and the function to be minimized is interpreted as the internal energy of the system in that state. Therefore the goal is to bring the system, from an arbitrary initial state, to a state with the minimum possible energy. The advantages are the origin in theoretical physics, which leads to convergence criteria, and the relative independence on the specifics of the cost function and initial guess, which allows nonlocal searches. The main disadvantages are the large computer requirements and, in practice, uncertain convergence to the global minimum [KGV83].

Genetic algorithms are based upon searches generated in analogy to the genetic evolution of natural organisms. At each iteration of the search, the genetic scheme keeps a population of approximate solutions. The population is evolved by manipulations of past

populations that mimic genetic transformations such that the likelihood of producing better data-fitted generations increases for new populations. Genetic algorithms allow nonlocal minimum searches, but convergence to the global minimum is not assured due to the limited theoretical base [Sch01a].

3.1.3 Estimation Theory

In estimation theory, statistical approaches are used to estimate the state of a dynamical system by combining all available knowledge pertaining to the system including the measurements and the modeling theories. Of significant importance in the estimation process is the a priori hypotheses and melding criterion since they determine the influence of dynamics and data onto the state estimate.

One of the most widely used tools in estimation theory is the Kalman Filter, which gives a sequential, unbiased, minimum error variance estimate based upon a linear combination of all past measurements and dynamics. Many extensions of the Kalman Filter have been developed over time to tackle the different challenges associated with the problem of sequential data assimilation. The Kalman Filter and its various extensions are the focus of this chapter.

In what follows, a review of the most commonly used Kalman Filtering techniques is presented, leading us to the description of proposed data assimilation methodology which combines the Kalman Filtering techniques with the Polynomial Chaos machinery.

3.2 The Kalman Filter

The Kalman Filter (KF) also known as the Kalman-Bucy Filter was developed in 1960 by R.E. Kalman [Kal60]. It is an optimal sequential data assimilation method for linear dynamics and measurement processes with Gaussian error statistics. It provides a linear,

unbiased, minimum variance algorithm to optimally estimate the state of the system from noisy measurements. With the recent technological advancements, the KF became more useful for very complex real-time applications such as, video and laser tracking systems, satellite navigation, radars, ballistic missile trajectories estimation, Fire control ....

This section is devoted for developing the Kalman filtering "prediction-correction" algorithm based on the optimality criterion of least-squares unbiased estimation. Consider a linear system with the following state-space description,

$$x_{k+1} = A_k x_k + \Gamma_k \xi_k, \tag{3.8}$$

$$z_k = H_k x_k + \eta_k \tag{3.9}$$

where A_k, Γ_k, and H_k are constant known matrices, and $\{\xi_k\}$ and $\{\eta_k\}$ are respectively system and observation noise sequences with known statistical information. For the KF, $\{\xi_k\}$ and $\{\eta_k\}$ are assumed to be sequences of zero-mean Gaussian white noise such that $Var(\xi_k) = Q_k$ and $Var(\eta_k) = R_k$ are positive definite matrices and $E(\xi_k \eta_l) = 0$ for all k and l. The initial state x_0 is also assumed to be independent of ξ_k and η_k for all k.

Let $\hat{x}_{k/k-1}$ be the to be the a priori state estimate at step k given knowledge of the process prior to step k, and $\hat{x}_{k/k}$ be the a posteriori state estimate at step k given the measurement z_k. The a priori and a posteriori estimate of the errors are then defined as,

$$e_{k/k-1} = x_k - \hat{x}_{k/k-1} \tag{3.10}$$

$$e_{k/k} = x_k - \hat{x}_{k/k}, \tag{3.11}$$

and their respective error covariance as,

$$P_{k/k-1} = E[e_{k/k-1}e_{k/k-1}^T] \tag{3.12}$$

$$P_{k/k} = E[e_{k/k}e_{k/k}^T]. \tag{3.13}$$

The a posteriori estimate $\hat{x}_{k/k}$ is obtained as a combination of the a priori estimate $\hat{x}_{k/k-1}$, the measurement z_k, and the measurement prediction $H_k\hat{x}_{k/k-1}$,

$$\hat{x}_{k/k} = \hat{x}_{k/k-1} + K_G(z_k - H_k\hat{x}_{k/k-1}). \tag{3.14}$$

The difference $(z_k - H_k\hat{x}_{k/k-1})$ is known as the innovation sequence or residual. The matrix K_G is chosen to be the gain or blending factor that minimizes the a posteriori error covariance 3.13. The following sections presents the details for achieving this matrix.

3.2.1 Derivation of the Kalman Gain

The error covariance matrix associated with the a posteriori estimate is defined as:

$$P_k = E[e_{k/k}e_{k/k}^T] = E[(x_k - \hat{x}_{k/k})(x_k - \hat{x}_{k/k})^T]. \tag{3.15}$$

If we replace equation 3.9 in 3.14 and substitute the resultant in 3.15, the expression for the a posteriori covariance matrix becomes:

$$\begin{aligned} P_k = E[[(x_k - \hat{x}_{k/k-1}) - K_G(H_k x_k + \eta_k - H_k\hat{x}_{k/k-1})] \\ [(x_k - \hat{x}_{k/k-1}) - K_G(H_k x_k + \eta_k - H_k\hat{x}_{k/k-1})]^T]. \end{aligned} \tag{3.16}$$

Evaluating the expectations and noting that $(x_k - \hat{x}_{k/k-1})$ is the a priori estimation error gives,

$$P_k = (I - K_G H_k) P_{k/k-1} (I - K_G H_k)^T + K_G R_k K_G^T. \tag{3.17}$$

The latter is a general expression, and it is valid for all K_G, optimal or otherwise. The goal is to find a specific K_G that minimizes the individual terms along the major diagonal of P_k. This optimization can be done in several ways, but the documentation mostly followed in the literature is that which follows the completing the square approach [Bro83]. The subscripts are dropped in the remaining of this derivation to avoid unnecessary clutter in the expressions. In what follows the a priori estimate is denoted by a $-$ superscript. Upon expanding and regrouping the terms in the P_k, we have:

$$P = P^- - (KHP^- - P^- H^T K^T) + K(HP^- H^T + R)K^T, \tag{3.18}$$

where the second term in the above equation is Linear in K, and the third is a quadratic function of K. It is assumed the $(HP^- H^T + R)$ is symmetric, and thus can be written in factored form as SS^T i.e.,

$$SS^T = HP^- H^T + R. \tag{3.19}$$

Therefore, the expression of P may now be rewritten as,

$$P = P^- - (KHP^- - P^- H^T K^T) + KSS^T K^T. \tag{3.20}$$

In order to complete the square, P is expressed in the form,

$$P = P^- + (KS - A)(KS - A)^T - AA^T, \tag{3.21}$$

where A is independent from K. If 3.21 is expanded and compared term by term with 3.20, the following equality must hold:

$$KSA^T + AS^TK^T = KHP^- + P^-H^TK^T. \tag{3.22}$$

Hence, it is easily noticed that 3.22 is only satisfied if A is expressed as,

$$A = P^-H^T(S^T)^{-1}. \tag{3.23}$$

Only the second term in 3.21 involves K; it is the product of a matrix with its transpose, which ensures that all terms along the major diagonal will be nonnegative. The aim is to minimize the diagonal terms K; therefore, the best way to do that is to adjust K so that the middle term in 3.21 is zero. Hence, K is chosen so that

$$KS = A. \tag{3.24}$$

The Kalman Gain matrix is thus given as,

$$\begin{aligned} K &= AS^{-1} \\ &= P^-H^T(S^T)^{-1}S^{-1} \\ &= P^-H^T(SS^T)^{-1}. \end{aligned} \tag{3.25}$$

But, SS^T is the factored form of $(HP^-H^T + R)$, and therefore the final expression of the optimum K is,

$$K = P^-H^T(HP^-H^T + R)^{-1} \tag{3.26}$$

Combining all the results above, we arrive at the Kalman filter algorithm [CC91]:

$$
\begin{aligned}
P_{0,0} &= Var(x_0) \\
P_{k,k-1} &= A_{k-1} P_{k-1,k-1} A_{k-1}^T + \Gamma_{k-1} Q_{k-1} \Gamma_{k-1}^T \\
K_G &= P_{k,k-1} H_k^T (H_k P_{k,k-1} H_k^T + R_k)^{-1} \\
P_{k,k} &= (I - K_G H_k) P_{k,k-1} \\
\hat{x}_{0/0} &= E(x_0) \\
\hat{x}_{k/k-1} &= A_{k-1} \hat{x}_{k-1,k-1} \\
\hat{x}_{k/k} &= \hat{x}_{k/k-1} + K_G (z_k - H_k \hat{x}_{k/k-1}) \\
k &= 1, 2, \ldots
\end{aligned}
\tag{3.27}
$$

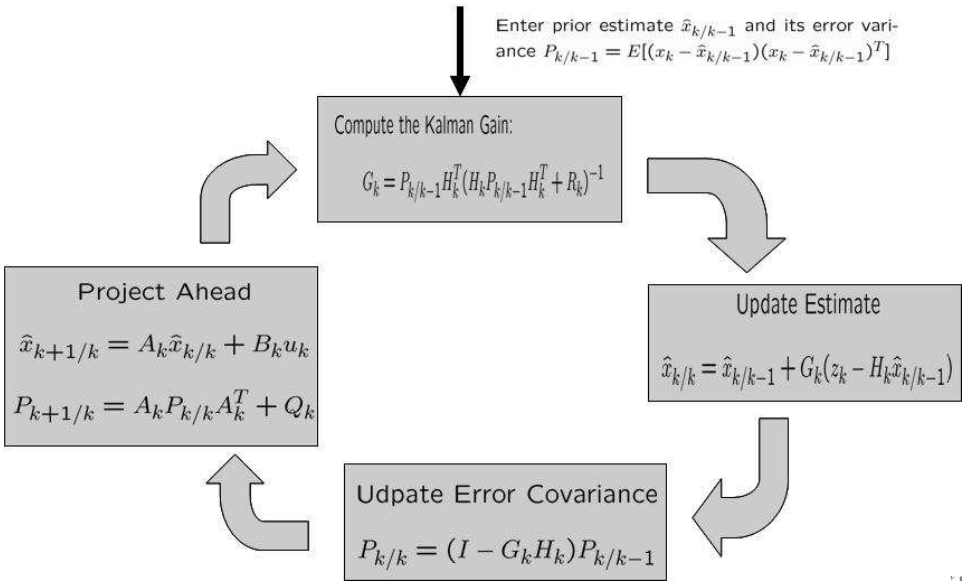


Figure 3.1: The Kalman Filter Loop.

3.2.2 Relationship to Recursive Bayesian Estimation

The true state of the dynamical system is considered as an unknown Markov process. Similarly the observational data are basically the hidden states of a Markov model. The Markov assumption implies that the true state is independent of all earlier states given the immediately previous one,

$$p(x_k/x_0, x_1, \ldots, x_{k-1}) = p(x_k/x_{k-1}). \tag{3.28}$$

Furthermore, the measurement at the k^{th} time step is only dependent on the current state and conditionally independent of all other states given the current one,

$$p(z_k/x_0, x_1, \ldots, x_k) = p(z_k/x_k). \tag{3.29}$$

Under these assumptions the probability density function of all the states in the hidden Markov Model can be expressed as,

$$p(x_0, \ldots, x_k, z_1, \ldots, z_k) = p(x_0)\Pi_{i=1}^{k} p(z_i/x_i)p(x_i/x_{i-1}). \tag{3.30}$$

The Kalman Filter is used to estimate the state x; the associated probability density is that corresponding to the current state conditioned on all the measurements available at the current time,

$$p(x_k/z_0, z_1, \ldots, z_{k-1}) = \int p(x_k/x_{k-1})p(x_{k-1}/z_0, z_1, \ldots, z_{k-1})dx_{k-1}. \tag{3.31}$$

After the k^{th} measurement we have,

$$p(x_k/z_0, z_1, \ldots, z_k) = \frac{p(z_k/x_k)p(x_k/z_0, \ldots, z_{k-1})}{p(z_k/z_0, \ldots, z_{k-1})}. \tag{3.32}$$

The denominator is a normalizing term,

$$p(z_k/z_0, z_1, \ldots, z_{k-1}) = \int p(z_k/x_k)p(x_k/z_0, z_1, \ldots, z_{k-1})dx_k, \tag{3.33}$$

and the remaining probability distributions are,

$$p(x_k/x_{k-1}) = N(Ax_{k-1}, Q_k) \tag{3.34}$$

$$p(z_k/x_k) = N(Hx_k, R_k) \tag{3.35}$$

$$p(x_{k-1}/z_0, \ldots, z_{k-1}) = N(\hat{x}_k, P_{k-1}), \tag{3.36}$$

where $N(m, \sigma)$ is a the standard normal distribution with mean m and a σ standard deviation.

3.3 The Extended Kalman Filter

Once the model dynamics turn out to be nonlinear, a linearization technique is implemented in deriving the filter equations. A real time linear Taylor approximation approach through which the nonlinear functions are recursively linearized around most recent estimate is considered. The resulting filter is known as the extended Kalman Filter (EKF). Note that the model nonlinearity also arises when solving parameter estimation problems even when the underlying dynamics is that of a simple linear model. Therefore, a nonlinear model arises once there exists a nonlinear dynamical model of the form,

$$x_{k+1} = f_k(x_k) + B_k(x_k)\xi_k \tag{3.37}$$

$$z_k = g_k(x_k) + \eta_k, \tag{3.38}$$

or when a system (even a linear one) is augmented with an equation describing the dynamics of the unknown parameters. The EKF yields statistically acceptable results for most nonlinear dynamical systems but there is a high chance that EKF updates are poorer than the nominal ones especially in the events where the initial uncertainty and measurement error are large [Bro83].

After linearizing the nonlinear function, the system is treated in the same way as in the KF. Consequently the EKF algorithm is [CC91]:

$$
\begin{aligned}
& P_{0,0} = Var(x_0), \qquad \hat{x}_0 = E(x_0) \\
& For \quad k = 1, 2, \ldots, \\
& P_{k,k-1} = \left[\frac{\partial f_{k-1}}{\partial x_{k-1}}(\hat{x}_{k-1})\right] P_{k-1,k-1} \left[\frac{\partial f_{k-1}}{\partial x_{k-1}}(\hat{x}_{k-1})\right]^T \\
& \qquad + B_{k-1}(\hat{x}_{k-1}) Q_{k-1} B_{k-1}^T(\hat{x}_{k-1}) \\
& \hat{x}_{k/k-1} = f_{k-1}(\hat{x}_{k-1}) \\
& K_G = P_{k,k-1} \left[\frac{\partial g_k}{\partial x_k}(\hat{x}_{k/k-1})\right]^T \\
& \qquad \cdot \left[\left[\frac{\partial g_k}{\partial x_k}(\hat{x}_{k/k-1})\right] P_{k,k-1} \left[\frac{\partial g_k}{\partial x_k}(\hat{x}_{k/k-1})\right]^T + R_k\right]^{-1} \\
& P_{k,k} = \left[I - K_G \left[\frac{\partial g_k}{\partial x_k}(\hat{x}_{k/k-1})\right]\right] P_{k,k-1} \\
& \hat{x}_{k/k} = \hat{x}_{k/k-1} + K_G \left(z_k - g_k(\hat{x}_{k/k-1})\right)
\end{aligned}
\tag{3.39}
$$

3.4 The Unscented Kalman Filter

The unscented Kalman filter [JU97] (UKF) is built on the idea that a set of discretely sampled points can be used to characterize the mean and covariance of the unknown model state. It uses a deterministic sampling technique known as the unscented transform to pick a minimal set of sample points (called sigma points) around the mean.

These sigma points are then propagated through the non-linear functions and the covariance of the estimate is then recovered. Using the unscented transformation approach, an n-dimensional random variable x with mean $\bar{x}$ and covariance P_{xx} is approximated by $2n+1$ weighted points given by

$$\begin{aligned} \mathcal{X}_0 &= \bar{x} \qquad\qquad\qquad W_0 = k/(n+k) \\ \mathcal{X}_i &= \bar{x} + (\sqrt{(n+k)P_{xx}})_i \quad W_i = 1/2(n+k) \\ \mathcal{X}_{i+n} &= \bar{x} - (\sqrt{(n+k)P_{xx}})_i \quad W_{i+n} = 1/2(n+k), \end{aligned} \tag{3.40}$$

where $k \in R$, $(\sqrt{(n+k)P_{xx}})_i$ is the i^{th} row or column of the matrix square root of $(n+k)P_{xx}$ and W_i is the weight associated with the i^{th} point. The algorithm of the UKF can thus be summarized in the following algorithm: [JU97]

1. Create the set of sigma points by applying the set of equations 3.41 on the system's initial conditions.

2. Each point in the set is propagated through the process model,

$$\mathcal{X}_i(k+1/k) = f[\mathcal{X}_i(k/k)]. \tag{3.41}$$

3. The predicted mean is computed as

$$\hat{x}(k+1/k) = \sum_{i=0}^{2n} W_i \mathcal{X}_i(k+1/k). \tag{3.42}$$

4. The corresponding covariance is computed as,

$$P(k+1/K) = \sum_{i=0}^{2n} W_i \mathcal{X}_i(k+1/k) - \hat{x}(k+1/k)\mathcal{X}_i(k+1/k) - \hat{x}(k+1/k)^T. \tag{3.43}$$

5. Each of the prediction points is instantiated through the observation model,

$$\mathcal{Z}_i(k+1/k) = h[\mathcal{X}_i(k+1/k), k] \tag{3.44}$$

6. The predicted observation is calculated by

$$\hat{z}(k+1/k) = \sum_{i=0}^{2n} W_i \mathcal{Z}_i(k+1/k). \tag{3.45}$$

7. Given that the observation noise is additive and independent, the innovation covariance is,

$$P_{zz}(k+1/k) = R(k+1) + \tag{3.46}$$
$$\sum_{i=0}^{2n} W_i[\mathcal{Z}_i(k+1/k) - \hat{z}(k+1/k)][\mathcal{Z}_i(k+1/k) - \hat{z}(k+1/k)]^T$$

8. The cross correlation matrix is determined by

$$P_{xz}(k+1/k) = \sum_{i=0}^{2n} W_i[\mathcal{X}_i(k+1/k) - \hat{x}(k+1/k)][\mathcal{X}_i(k+1/k) - \hat{z}(k+1/k)]^T \tag{3.47}$$

3.5 The Ensemble Kalman Filter

The Ensemble Kalman Filter (EnKF) proposed by Evensen [Eve94] and clarified by Burgers et al. [BLE98] aims at resolving some of the drawbacks of the EKF. The EnKF is based on forecasting the error statistics using Monte Carlo methods which turns out to be a better alternative to solving the traditional and computationally expensive approximate error covariance equation used in the EKF. It was designed to resolve two major problems related to the use of the EKF with nonlinear dynamics in large state spaces.

The first is that the EKF adopts an approximate closure scheme, and the second is the huge computational requirements associated with the storage and forward integration of the error covariance matrix [Eve03]. Since its development, the EnKF has been extensively used in various research areas such as ocean engineering, weather forecast, petroleum engineering, hydrology, system identification ...

3.5.1 Practical Implementation of the EnKF

As mentioned earlier, the EnKF propagates an ensemble of state vectors forward in time. The initial ensemble is chosen so that it properly represents the error statistics of the initial guess of the model states. Therefore, the initial ensemble is usually created by adding some kind of perturbations to a best-guess estimate, and then the ensemble is integrated over a time interval covering a few characteristic time scales of the dynamical system.

Let $\mathbf{A}$ be the matrix holding the ensemble members $x_i \in \Re^n$,

$$\mathbf{A} = (x_1, x_2, \ldots, x_N) \in \Re^{n \times N}, \tag{3.48}$$

where N is the number of ensemble members, and n is the size of the model state vector. The ensemble mean is then given by,

$$\bar{\mathbf{A}} = \mathbf{A}\mathbf{1}_N, \tag{3.49}$$

where $1_N \in \Re^{N \times N}$ is a matrix with all its elements equal to $1/N$. The ensemble perturbation matrix is defined as,

$$\mathbf{A}' = \mathbf{A} - \bar{\mathbf{A}} = \mathbf{A}\left(\mathbf{I} - \mathbf{1}_N\right), \tag{3.50}$$

and the ensemble covariance matrix $\mathbf{P}_e \in \Re^{n \times n}$ as,

$$\mathbf{P}_e = \frac{\mathbf{A}'(\mathbf{A}')^T}{N-1}. \tag{3.51}$$

A new ensemble of observations is generated at each correction step, by adding perturbations drawn from a distribution with zero mean and covariance equal to the measurement error covariance matrix [BLE98]. Let $\mathbf{d} \in \Re^m$, where m is the number of measurements, be a vector of measurements. The ensemble of observations,

$$\mathbf{D} = (\mathbf{d}_1, \mathbf{d}_2, \ldots, \mathbf{d}_N) \in \Re^{m \times N}, \tag{3.52}$$

is obtained by perturbing the measurement vector $\mathbf{d}$ as follows,

$$\mathbf{d}_j = \mathbf{d} + \epsilon_j, \qquad j = 1, \ldots, N. \tag{3.53}$$

The corresponding measurement error covariance matrix is given by

$$\mathbf{R}_e = \frac{\Upsilon\Upsilon^T}{N-1}, \tag{3.54}$$

where Υ is the ensemble of perturbations,

$$\Upsilon = (\epsilon_1, \epsilon_2, \ldots, \epsilon_N) \in \Re^{m \times N}. \tag{3.55}$$

The analysis or updating step can be performed on either each of the model state ensemble members,

$$x_j^{k/k} = x_j^{k/k-1} + \mathbf{G}_{k_e}(d_j - Hx_j^{k/k-1}), \tag{3.56}$$

or on the ensemble itself

$$\mathbf{A}^{k/k} = \mathbf{A}^{k/k-1} + \mathbf{G}_{k_e}(\mathbf{D} - \mathbf{H}\mathbf{A}^{k/k-1}), \tag{3.57}$$

where $\mathbf{G}_{k_e}$ is the KF gain matrix

$$\mathbf{G}_{k_e} = \mathbf{P}_e\mathbf{H}^T(\mathbf{H}\mathbf{P}_e\mathbf{H}^T + \mathbf{R}_e)^{-1}. \tag{3.58}$$

3.6 The Particle Filter

The particle filters, also known as the Sequential Monte Carlo methods (SMC), are a set of flexible simulation-based methods for sampling from a sequence of probability distributions; each distribution being only known up to a normalising constant. These methods are similar to the importance sampling methods and they often serve as replacements for the Ensemble Kalman Filter and the Unscented Kalman Filter with the advantage that they approach the Bayesian optimal estimate if a sufficient number of samples is used. These filters are a sequential analogue of the Markov Chain Monte Carlo (MCMC) methods. While the MCMC is used to model the full posterior distribution $p(x_0, x_1, \ldots, x_k/y_0, y_1, \ldots, y_k)$ of a hidden state or parameter x given a set of observation $y_0, y_1, \ldots, y_k$, the particle filter aims at estimating x_k from the a posteriori distribution $p(x_k/y_0, y_1, \ldots, y_k)$ [DdFG01].

One of the most commonly used particle filtering algorithms, is the Sampling Importance Resampling (SIR) algorithm. It is a sequential verion of the importance sampling method that aims at approximating a distribution using a weighted set of particles. Particle filters are beyond the scope of this study, a brief overview of these filters is mentioned for the sake of completeness.

3.7 Coupling of Polynomial Chaos with the EnKF

The filter used in this study allows the propagation of a stochastic representation of the unknown variables using Polynomial Chaos. This overcomes some of the drawbacks of the EnKF. Using the proposed method, At any instant in time, the probability density function of the model state or parameters can be easily obtained by simulating the Polynomial Chaos basis. Furthermore, this method allows representation of non-Gaussian measurement and parameter uncertainties in a simpler, less taxing way without the necessity of managing a large ensemble.

3.7.1 Representation of Error Statistics

The Kalman Filter defines the error covariance matrices of the forecast and analyzed estimate in terms of the true state as,

$$\mathbf{P}^f = \left\langle (x^f - x^t)(x^f - x^t)^T \right\rangle, \tag{3.59}$$

$$\mathbf{P}^a = \left\langle (x^a - x^t)(x^a - x^t)^T \right\rangle, \tag{3.60}$$

where $\langle\rangle$ denotes the mathematical expectation, x is the model state vector at a particular time, and the superscripts f, a, and t represent the forecast, analysis, and true state, respectively. However, in the Polynomial Chaos based Kalman Filter, the true state is not known, and therefore the error covariance matrices are defined using the Polynomial Chaos representations of the model state. In the PCKF, the model state is given by,

$$x = \sum_{i=0}^{P} x_i \psi_i(\xi), \tag{3.61}$$

where $P+1$ is the number of terms in the Polynomial Chaos expansion of the state vector, and $\{\psi_i\}$ is the set of Hermite polynomials. Consequently, the covariance matrices are defined around the mean, the zeroth order term, of the stochastic representation,

$$\begin{aligned}
\mathbf{P}^f &\approx \left\langle (\sum_{i=0}^{P} x_i^f \psi_i - x_0^f)(\sum_{i=0}^{P} x_i^f \psi_i - x_0^f)^T \right\rangle \\
&\approx \left\langle (\sum_{i=1}^{P} x_i^f \psi_i)(\sum_{i=1}^{P} x_i^f \psi_i)^T \right\rangle \\
&\approx \sum_{i=1}^{P} x_i^f x_i^{f^T} \left\langle \psi_i^2 \right\rangle ,
\end{aligned} \tag{3.62}$$

and similarly,

$$\mathbf{P}^a \approx \sum_{i=1}^{P} x_i^a x_i^{a^T} \left\langle \psi_i^2 \right\rangle . \tag{3.63}$$

The Polynomial Chaos representation depicts all the information available through the complete probability density function, and therefore allows the propagation of all the statistical moments of the unknown parameters and variables.

The observations are also treated as random variables represented via a Polynomial Chaos expansion with a mean equal to the first-guess observations. Since the model and measurement errors are assumed to be independent, the latter is represented as a Markov process.

3.7.2 Analysis Scheme

For computational efficiency, the dimensionality and order of the Polynomial Chaos expansion are homogenized through out the solution. These parameters are initially defined based on the uncertainty within the problem at hand and are assumed to be constant thereafter. Since the model state and measurement vectors are assumed independent, the Polynomial Chaos representation of these variables have a sparse structure.

Let **A** be the matrix holding the chaos coefficients of the state vector $y \in R^n$,

$$\mathbf{A} = (x_0, x_1, \ldots, x_P) \in R^{n \times P+1}, \tag{3.64}$$

where $P+1$ is the total number of terms in the Polynomial Chaos representation of x and n is the size of the model state vector. The mean of x is stored in the first column of **A** and is denoted by x_0. The state perturbations are given by the higher order terms stored in the remaining columns. Consequently, the state error covariance matrix $\mathbf{P} \in R^{n \times n}$ is defined as

$$\mathbf{P} = \sum_{i=1}^{P} x_i x_i^T \left\langle \psi_i^2 \right\rangle . \tag{3.65}$$

Given a vector of measurements $d \in R^m$, with m being the number of measurements, a Polynomial chaos representation of the measurements is defined as

$$D = \sum_{j=0}^{P} d_j \psi_j(\xi), \tag{3.66}$$

where the mean d_0 is given by the actual measurement vector, and the higher order terms represent the measurement uncertainties. The representation D can be stored in the matrix

$$\mathbf{B} = (d_0, d_1, \ldots, d_P) \in R^{m \times P+1}. \tag{3.67}$$

From Eq. 3.66, we can construct the measurement error covariance matrix

$$\mathbf{R} = \sum_{i=1}^{P} d_i d_i^T \left\langle \psi_i^2 \right\rangle \in R^{m \times m}. \tag{3.68}$$

The Kalman Filter forecast step is carried out by employing a stochastic Galerkin scheme, and the assimilation step simply consists of the traditional Kalman Filter correction step applied on the Polynomial Chaos expansion of the model state vector,

$$\sum_{i=0}^{P} x_i^a \psi_i = \sum_{i=0}^{P} x_i^f \psi_i + \mathbf{PH}^T(\mathbf{HPH}^T + \mathbf{R})^{-1}(\sum_{i=0}^{P} d_i \psi_i - \mathbf{H}\sum_{i=0}^{P} x_i^f \psi_i). \tag{3.69}$$

Projecting on an approximating space spanned by the Polynomial Chaos $\{\psi_i\}_{i=0}^{P}$ yields,

$$x_i^a = x_i^f + \mathbf{PH}^T(\mathbf{HPH}^T + \mathbf{R})^{-1}(d_i - \mathbf{H}x_i^f) \quad for\ i = 0, 1, \ldots, P. \tag{3.70}$$

In matrix form, the assimilation step is expressed as,

$$\mathbf{A}^a = \mathbf{A}^f + \mathbf{PH}^T(\mathbf{HPH}^T + \mathbf{R})^{-1}(\mathbf{B} - \mathbf{HA}^f). \tag{3.71}$$

Chapter 4

Multiphase Flow in Porous Media

Flow through porous media is a problem encountered in many realms of science and engineering. Applications include ground water hydrology, reservoir engineering, soil mechanics, chemical and Biomedical engineering .... In this study the focus is on reservoir engineering, but the proposed methods could be easily extended to other disciplines of multiphase flow in porous media. Fluid motions in porous media are governed by the same fundamental laws that govern their motion in the atmosphere, pipelines, rivers .... These are the mass, momentum, and energy conservation laws. All the laws considered in this study are valid at the macroscopic level, i.e. for a volume of porous medium which is infinitely large with respect to the size of fluid particles and of the pores, but can be infinitely small with respect to the size of the field itself [CJ86].

4.1 Flow Equations

In order to model a multiphase flow system, conservation of mass for each existing phase is required. The general form of these equations can be expressed as

$$\frac{\partial}{\partial t}\left(\phi \rho_i S_i\right) = -\nabla . \left(\rho_i \vec{V_i}\right) + q_i \qquad\qquad i = 1, 2, \ldots \tag{4.1}$$

where i denotes the fluid phase, ϕ is the porosity of the medium, ρ_i is the density of phase i, S_i is the phase saturation, $\vec{V_i}$ is the phase velocity, q_i is a source/sink term, and t is time.

Darcy's laws is commonly used for calculating phase flow velocities. It was developed by Darcy in mid 19th century for one phase flow only. Muskat [Mus49] showed experimentally that Darcy's law remains valid for each fluid separately when two or more immiscible fluids share the pore space. Darcy's law for phase velocities is given by

$$\vec{V_i} = -\frac{\vec{K} K_{ri}}{\mu_i} \left(\nabla P_i - \rho_i g \nabla z(x)\right) \tag{4.2}$$

where $\vec{K}$ is the intrinsic permeability, K_{ri} is the relative permeability of phase i, μ_i is the viscosity of phase i, P_i is the pressure of phase i, g is the gravitational acceleration, and z is the depth of the fluid. Substituting Eqs. 4.2 into Eqs. 4.1 yields the general form of the flow equations for all phases,

$$\frac{\partial}{\partial t}\left(\phi \rho_i S_i\right) = \nabla . \left[\frac{\rho_i \vec{K} K_{ri}}{\mu_i} \left(\nabla P_i - \rho_i g \nabla z(x)\right)\right] + q_i \qquad i = 1, 2, \ldots \tag{4.3}$$

The latter set of equations signify that the flow in porous media is driven by gravity, pressure gradients, and viscous forces. It also incorporates the effects of porous matrix compressibility, fluid compressibility, capillary pressure, and spatial variability of permeability and porosity. The nonlinearity arises from the interdependence of the phase relative permeabilities and capillary pressure on the phase saturations.

4.2 Features of the Reservoir and the Fluids

In this section, the details of the geological features of the reservoir and the fluids are presented. In the following, the physical properties and characteristics of all the unknown parameters in eqs. 4.3 are described.

4.2.1 Density and Viscosity

The density and the viscosity are intrinsic fluid properties. Density is defined as the mass per unit volume, it varies mainly with temperature. In this study, fluid density variation is neglected. The state equation for an ideal liquid with constant compressibility is given by

$$\rho = \rho_o \exp\left[c(p - p_o)\right], \tag{4.4}$$

and that for an ideal gas is

$$\rho = \frac{W}{RT}p \tag{4.5}$$

where ρ_o is the fluid density at some reference pressure, p_o, c is the fluid's compressibility, W is the molecular weight, R is the gas constant, and T is the absolute temperature.

Viscosity is the measure of the resistance to flow caused by the internal friction within the fluid. Viscosity is also a function of temperature; it decreases as temperature increases. The differences in the viscosities of two interacting immiscibe fluids cause instabilities resulting in a phenomena known as viscous fingering. Fingering manifests itself as unexpected macroscopic deformations in the inter facial boundary separating the two fluids.

4.2.2 Porosity

Porosity of a medium is the percentage of void space existing between soil particles. It is a function of the size-sorting and shape of the soil grains. Methods for computing the porosity of a medium are well studied and documented [Bea72].

Consider at a point in the porous medium a volume ΔU_i much larger than a single pore or grain. For this volume the following ratio is determined,

$$\phi_i = \left(\Delta U_v\right)_i / \Delta U_i \tag{4.6}$$

where $(\Delta U_v)_i$ is the volume of void space within ΔU_i. The medium's volumetric porosity, ϕ, at that point is defined as:

$$\phi = \lim_{\Delta U_i \to \Delta U_o} \frac{(\Delta U_v)_i}{\Delta U_i} \tag{4.7}$$

where ΔU_o is the representative elementary volume (REV) of the porous medium at the specified point.

4.2.3 Saturation and Residual Saturation

Saturation of a fluid is the ratio of the volume of the void filled with fluid to the total void volume. The values of the saturations vary between zero and one, and the sum of all coexisting fluid phases add up to one leading to the following constraint

$$\sum_{i}^{n} s_i = 1, \tag{4.8}$$

where n is the number of existing phases in the system. Two critical values of the phase saturations are identified, (1) S_{wc} the saturation at which the injected fluid starts to flow, and (2) S_{nc} the saturation at which the displaced fluid ceases to flow. S_{wc} and S_{nc} are the known as residual saturations.

4.2.4 Intrinsic and Relative Permeability

The Intrinsic permeability $\vec{K}$ is a tensorial quantity depending on the pore size opening available for fluid flow. Moreover, $\vec{K}$ depends on the nature of the fluid saturating the porous medium.

Relative permeability signifies the fraction of the pore space available for the phase to flow. Recent studies [BBM87, MG87] have suggested that the relative permeability

varies with the saturation in a tensorial behavior. For two phase flow, Brooks and Corey [BC64] derived empirically-based expressions for the relative permeabilities of the wetting and non wetting phases. Figure 4.1 presents typical two-phase relative permeability-saturation relations. These permeability curves are expressed as

$$K_{rw} = S_e^{(2+3\lambda)/\lambda} \tag{4.9}$$

and,

$$K_{rn} = (1 - S_e)^2(1 - S_e^{(2+\lambda)/\lambda}) \tag{4.10}$$

where K_{rw} and K_{rn} are the relative permeability of the wetting non-wetting phases respectively, λ is a model fitting parameter related to the pore size distribution of the soil material, and S_e is the reduced saturation given by

$$S_e = \frac{S_w - S_{wc}}{S_m - S_{wc}}, \tag{4.11}$$

where S_m is the maximum wetting phase saturation.

In the case of three-phase flow systems, theoretical models have been developed for three-phase relative permeability curves [Sto70, DP89, FM84]. Two common beliefs coexist for three-phase relative permeabilities. The first argues that the heterogeneity's effect on the relative permeability leads to the hysteretic behavior demonstrated by the curves [Dul91, LP87, KP87]. The second argument suggests that the fluid channel network available for the flow is mainly dependent on the fluid phase contents and not on the saturation history of the medium which leads to the conclusion that relative permeability curves are non-hysteretic [LGN89].

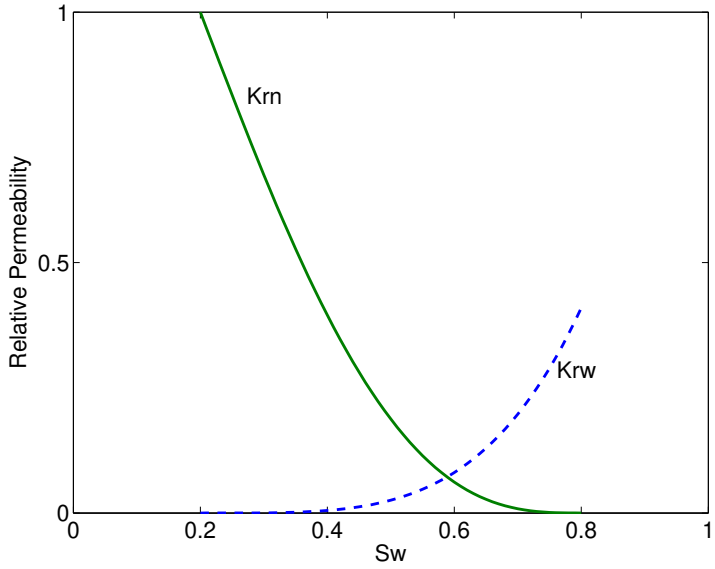


Figure 4.1: Typical Relative Permeability Curves.

4.2.5 Capillary Pressure

The capillary pressure is the pressure differences that occur across the fluid-fluid interfaces of the coexisting phases in the subsurface. It is defined as

$$p_c = p_{nw} - p_w, \tag{4.12}$$

where p_c is the capillary pressure, p_{nw} is the pressure of the non-wetting phase, and p_w is the pressure of the wetting phase. In order to show which of the fluids is the wetting one, one has to look at the meniscus separating the two fluids in a capillary tube: the concavity of the meniscus is oriented toward the non-wetting fluid [CJ86]. Figure 4.2 demonstrates this phenomena. The capillary pressure increases with a decrease in contact angle between the phases in the pore size. The smaller the pore radius, the higher is the resistance to non-wetting phase to occupy a pore that is preoccupied with a wetting phase. Therefore, the non-wetting phase usually occupies the larger pore spaces

where there exist lower capillary resistance, and the wetting phase occupies the smaller ones.

The capillary pressure can be expressed as a function of the wetting phase saturation only. Empirical relationships describing this functional dependence are known as the capillary pressure-saturation curves. Figure 4.3 shows a typical capillary pressure-saturation curve. Brooks and Corey [BC64] developed a close form expression that represents these curves for two phase flow. Brooks and Corey's equation is,

$$p_c = p_d S_e^{-1/\lambda} \qquad\qquad p_c \geq p_d, \tag{4.13}$$

where p_d is the displacement or threshold pressure which first gives rise to the oil phase permeability.

Later, Parker et.al [PL87] extended Brooks and Corey's model to three-phase flow and came up with an expression for the capillary pressure that can be reduced to two-phase flow as well. This model is given by

$$p_c = P_d \left(S_e^{-1/m} - 1\right)^{1-m} \qquad\qquad pc \geq 0, \tag{4.14}$$

where m is a model fitting parameter.

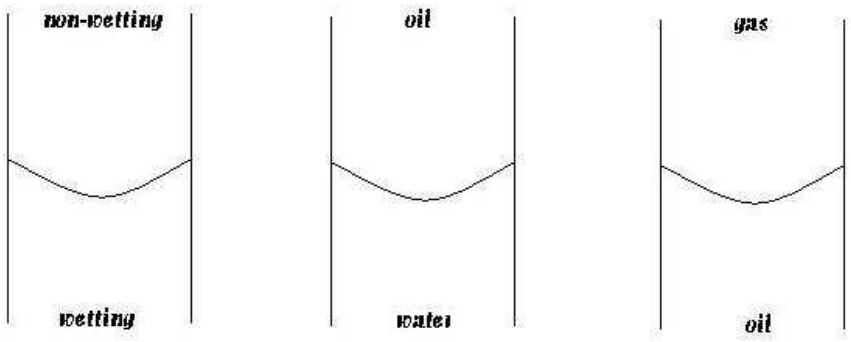


Figure 4.2: Determination of the wetting phase.

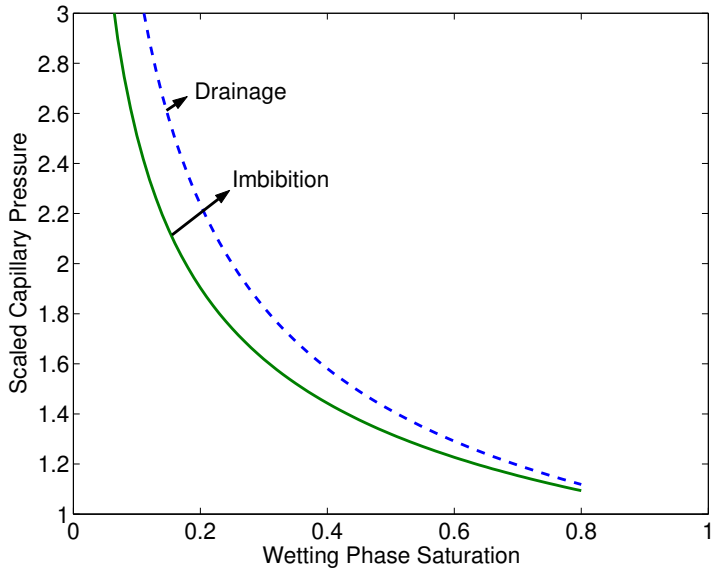


Figure 4.3: Typical Capillary Pressure-Saturation Curve.

4.3 Characterization of Reservoir Simulation Models

Reservoir simulation is a powerful tool for reservoir characterization and management. It enhances the production forecasting process. The efficiency of a reservoir model relies on its ability to characterize the geological and petrophysical features of the actual field.

One of the most commonly used methods for reservoir characterization is automatic history matching. History matching aims at estimating reservoir parameters such as porosities and permeabilities so as to minimize the square of the mismatch between observations and computed values. The heterogeneities of the geological formations and the uncertainties associated with the medium properties render history matching a very complex and challenging phenomena. In this study a novel history matching methodology based on the Polynomial Chaos Kalman Filter is presented.

This study deals with the two-phases immiscible flow water flooding reservoir engineering problem. Initially the porous medium is assumed to be fully saturated with oil, and water is pumped through one well to push the oil out through other wells in the field. The governing flow equations consist of the water continuity equation,

$$\phi\left(\frac{\partial S_w}{\partial t}\right) = \nabla.\left[\frac{\vec{K}K_{rw}}{\mu_w}\left(\nabla P_w - \rho_w g \nabla z(x)\right)\right] + q_w, \tag{4.15}$$

and the oil continuity equation,

$$\phi\left(\frac{\partial S_o}{\partial t}\right) = \nabla.\left[\frac{\vec{K}K_{ro}}{\mu_o}\left(\nabla P_o - \rho_o g \nabla z(x)\right)\right] + q_o. \tag{4.16}$$

These equations are subject to the following constraints:

$$S_w + S_o = 1 \tag{4.17}$$

$$p_c(S_w) = p_o - p_w. \tag{4.18}$$

4.3.1 Uncertainty Quantification

The heterogeneity in the porous medium is dealt with in the probabilistic sense, i.e. by modeling the intrinsic permeability and porosity of the medium as stochastic processes

via their Polynomial Chaos expansion. In this study, two scenarios are used for representing the uncertain medium properties. In the first, both the intrinsic permeability and porosity are represented as stochastic processes using the Polynomial Chaos expansion:

$$K(x,\theta) \approx \hat{K}(x,\theta) = \sum_{i=0}^{M} K_i(x)\psi_i(\xi(\theta)), \tag{4.19}$$

$$\phi(x,\theta) \approx \hat{\phi}(x,\theta) = \sum_{j=0}^{M} \phi_j(x)\psi_j(\xi(\theta)), \tag{4.20}$$

where $\{K_i(x)\}$ and $\{\phi_i(x)\}$ are sets of deterministic functions to be estimated using the proposed sequential data assimilation technique.

The second scenario suggests representing the intrinsic permeability of the porous medium as a stochastic process while modeling the porosity as a random variable. Therefore the ratio, $\alpha(x,\theta)$, of the intrinsic permeability and porosity of the medium is represented as,

$$\alpha(x,\theta) = \frac{K(x,\theta)}{\phi(\theta)} \approx \exp(\sum_{i=0}^{P} \alpha_i(x)\psi_i(\xi(\theta))), \tag{4.21}$$

where $\{\alpha_i(x)\}$ is a set of deterministic functions to be estimated using the proposed sequential data assimilation technique. Although both scenarios work well, the first requires more monitoring to guarantee that the estimated numbers remain physically valid.

The solution of the resulting system of stochastic partial differential equations is also represented via Polynomial Chaos,

$$S_w(x,\theta) \approx \hat{S}_w(x,\theta) = \sum_{k=0}^{P} S_{w_k}(x)\psi_k(\xi), \tag{4.22}$$

and

$$P_w(x,\theta) \approx \hat{P}_w(x,\theta) = \sum_{k=0}^{P} P_{w_k}(x)\psi_k(\xi), \tag{4.23}$$

where $\{P_{w_k}\}$ and $\{S_{w_k}\}$ are the deterministic nodal vectors to be solved for, P denotes the number of terms in the Polynomial Chaos expansion, and $\{\psi_k(\xi)\}$ is a basis set consisting of orthogonal polynomial chaoses of consecutive orders.

The relative permeabilities and the capillary pressure are functions of water saturation, and therefore, they are represented using their Polynomial Chaos expansion as well. Fourth order polynomial approximations of the Brooks-Corey Model are adopted and the relative permeabilities and capillary pressure are approximated as,

$$K_{rw} \approx \hat{K}_{rw} = \sum_{i=0}^{4}(\sum_{j=0}^{P} S_{e_j}\psi_j)^i c_i + \eta(\xi), \tag{4.24}$$

$$K_{ro} \approx \hat{K}_{ro} = \sum_{i=0}^{4}(\sum_{j=0}^{P} S_{e_j}\psi_j)^i c_i + \eta(\xi), \tag{4.25}$$

and

$$P_c \approx \hat{P}_c = \sum_{i=0}^{4}(\sum_{j=0}^{P} S_{e_j}\psi_j)^i c_i + \eta(\xi), \tag{4.26}$$

where η represents the modeling error, and the coefficients $c_i's$ are numerically calculated based on the value of the Brooks-Corey model fitting parameter, λ.

4.3.2 Model Formulation

The error, ϵ , resulting from truncating the Polynomial Chaos expansions at a finite number of terms is minimized by forcing it to be orthogonal to the solution space spanned

by the basis set of orthogonal stochastic processes appearing in the Polynomial Chaos expansion [GS03]. Mathematically, this is expressed as

$$\langle \epsilon, \psi_m \rangle \qquad\qquad m = 1, 2, \ldots, \tag{4.27}$$

where $\langle . \rangle$ denotes mathematical expectation. This will yield a system of $N \times P + 1$ nonlinear coupled algebraic equations to be solved for the deterministic coefficients of the water saturation and pressure, where N is the number of nodes in the spatially discretized mesh, and $P + 1$ is the total number of terms in the Polynomial Chaos expansion [Hat98]. If the first scenario is adopted, the system is represented as,

$$\sum_{i=0}^{P}\sum_{j=0}^{P} \phi_i \left(\frac{\partial S_{w_j}}{\partial t}\right) \langle \psi_i \psi_j \psi_m \rangle = \tag{4.28}$$

$$\nabla . \left[\sum_{i=0}^{P}\sum_{j=0}^{P}\sum_{k=0}^{P} \frac{\vec{K}_i K_{rw_j}}{\mu_w} \left(\nabla P_{w_k}\right) \langle \psi_i \psi_j \psi_k \psi_m \rangle\right] + q_w \langle \psi_m \rangle \quad \forall m$$

$$\sum_{i=0}^{P}\sum_{j=0}^{P} \phi_i \left(\frac{-\partial S_{w_j}}{\partial t}\right) \langle \psi_i \psi_j \psi_m \rangle = \tag{4.29}$$

$$\nabla . \left[\sum_{i=0}^{P}\sum_{j=0}^{P}\sum_{k=0}^{P} \frac{\vec{K}_i K_{rw_j}}{\mu_w} \nabla \left(P_{w_k} + P_{c_k}\right) \langle \psi_i \psi_j \psi_k \psi_m \rangle\right] + q_o \langle \psi_m \rangle \quad \forall m,$$

where the expectations $\langle \psi_i \psi_j \psi_m \rangle$ and $\langle \psi_i \psi_j \psi_k \psi_m \rangle$ are easily calculated and tabulated in the literature [GS03]. On the other hand, when the second scenario is employed, the resulting system is expressed as,

$$\sum_{j=0}^{P}\left(\frac{\partial S_{w_j}}{\partial t}\right)\langle\psi_j\psi_m\rangle = \tag{4.30}$$
$$\nabla.\left[\sum_{j=0}^{P}\sum_{k=0}^{P}\frac{K_{rw_j}}{\mu_w}\left(\nabla P_{w_k}\right)\left\langle\exp(\sum_{i=0}^{P}\alpha_i\psi_i)\psi_j\psi_k\psi_m\right\rangle\right] + q_w\langle\psi_m\rangle \quad \forall m$$
$$\sum_{j=0}^{P}\left(\frac{-\partial S_{w_j}}{\partial t}\right)\langle\psi_j\psi_m\rangle = \tag{4.31}$$
$$\nabla.\left[\sum_{j=0}^{P}\sum_{k=0}^{P}\frac{K_{rw_j}}{\mu_w}\nabla\left(P_{w_k}+P_{c_k}\right)\left\langle\exp(\sum_{i=0}^{P}\alpha_i\psi_i)\psi_j\psi_k\psi_m\right\rangle\right] + q_o\langle\psi_m\rangle \quad \forall m.$$

Evaluating the above expectations is not a trivial task; an algorithm is developed to evaluate them as a function of $\langle\psi_i\psi_j\psi_m\rangle$ by the using the method of change of variables followed by employing the following Hermite polynomials identity,

$$2^{n/2}\psi_n(\frac{x+y}{\sqrt{2}}) = \sum_{k=0}^{n}\binom{n}{k}\psi_k(x)\psi_{n-k}(y). \tag{4.32}$$

The spectral stochastic finite element method capabilities are developed within SUNDANCE [Lon04], and the package is used to numerically solve the above stochastic systems. Although the implementation of the second scenario is computationally more expensive, it guarantees the positiveness of the estimated medium parameters.

4.4 Numerical Applications

Two synthetic sets of problems are selected to assess the efficiency of the proposed history matching technique. The first one explores the one dimensional two-phase water flooding, while the second solves the two dimensional two-phase model. In both sets, the model state vector consists of all the reservoir variables that are uncertain and need to be specified. These include the phase saturations and pressures as well as the suggested

Table 4.1: Uncertainty Representation Using Polynomial Chaos

Source of Uncertainty	Representation
Parametric (1)	$\xi_1,\ \xi_1^2 - 1$
Parametric (2)	$\exp(\xi_1)$
Modeling	ξ_2
Measurement	ξ_3

representation of the medium properties. The objective is to statistically estimate the medium properties through measurements of the water saturation at specific locations.

The modeling and measurement errors are assumed to be independent Gaussian noises, and therefore, they are represented using one dimensional, first order Polynomial Chaos expansions. Unlike the EnkF where the model error is represented using an additive noise, in PCKF the model error is incorporated in the Brooks-Corey model. In order to accommodate the fact that the medium properties may deviate from Gaussianity, the unknown porosity and permeability of the medium are either modeled as dependent one dimensional, second order Polynomial Chaos expansions according to scenario one proposed earlier, or their ratio is expressed as an exponential function of a one dimensional first order Polynomial Chaos Expansion as is explained in the second scenario. Table 4.1 details the uncertainty representation in the numerical model.

4.4.1 One Dimensional Buckley-Leverett Problem

The first test problem is that of the incompressible water flood Buckley-Leverett example. The relative permeabilities within the model are given by the Brooks-Corey model. The reservoir is horizontal with a length of $1000ft$, cross-sectional area of $10000ft^2$, and constant initial water saturation $S_{w_i} = 0.16$. Oil is produced at $x = 1000ft$ at a rate of $426.5ft^3/day$, and water is injected at $x = 0$ at the same rate. Three case studies are

developed on this problem. In all three cases, scenario one is adopted for representing the parametric uncertainties.

4.4.1.1 Case 1: Homogeneous Medium

To Test the validity of the proposed approach, the same Brooks-Corey model parameter, λ, is assumed for both the forward and inverse analysis. The field is assumed to be homogeneous with an intrinsic permeability of $270md$ and a porosity of 0.275, and measurements of the water saturation and pressure are available each $50ft$ every 10 time steps. Figure 4.4 gives the statistical properties of the estimated parameters; it represents the variation of these parameters with time. It is noticed that as more measurements are available, the mean estimate converges toward the true model parameters, and the Polynomial Chaos coefficients decay exponentially indicating a deterministic estimate. This is expected since the model used to estimate the reservoir state is identical to the model used for generating the measurements.

4.4.1.2 Case 2: Continuous Intrinsic Permeability and Porosity Profiles

In this case continuous profiles for the porosity and intrinsic permeability are assumed. The measurements of the water saturation and pressure are also assumed available each $50ft$ every 10 time steps. However, the Brooks-Corey Model used to generate the measurements has a fitting parameter $\lambda = 2.2$ while the the model used in the filtering scheme has $\lambda = 2.0$. This will result in uncertainties within the estimate associated with modeling errors. Figure 4.5 represents a Polynomial Chaos estimate of the medium properties obtained after 2000 updates. Figure 4.6 shows the probability density functions (pdf's) of the estimated porosity and intrinsic permeability at quarter span. It

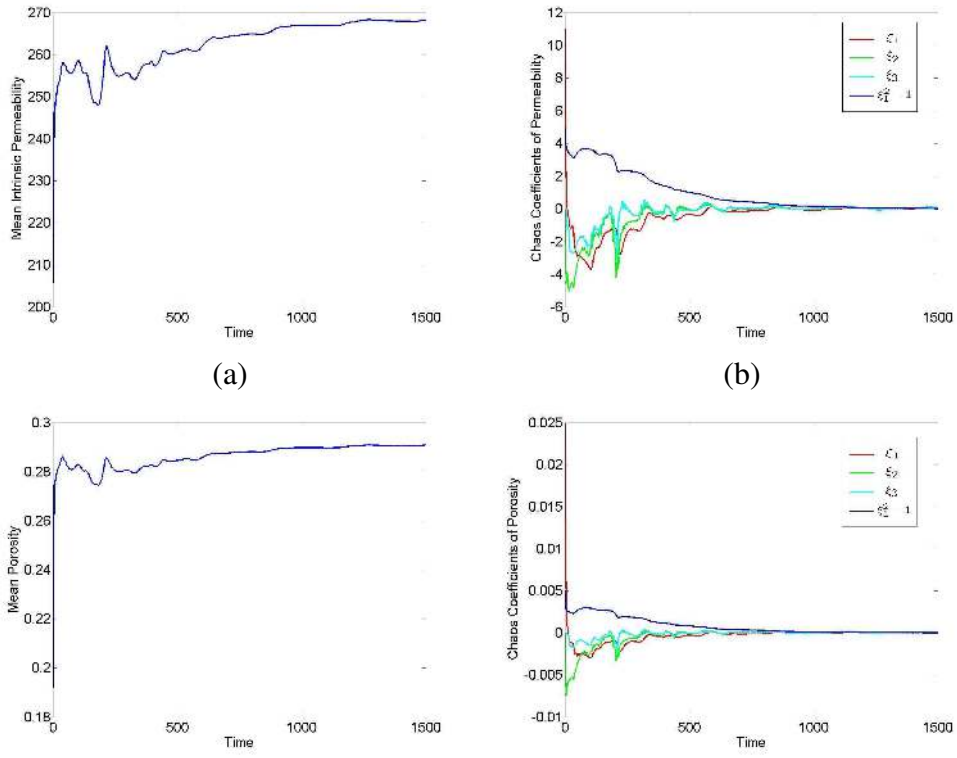


Figure 4.4: Case 1: Estimate of medium properties: (a) Mean intrinsic permeability, (b) Polynomial Chaos coefficients of intrinsic permeability, (c) mean porosity, and (d) Polynomial Chaos coefficients of porosity

is clear from the obtained pdf's that the porosity and intrinsic permeability have non-Gaussian properties. In order to validate the estimated parameters, the estimated residual water saturation is plotted against the true model state in figure 4.7.

4.4.1.3 Case 3: Discontinuous Porosity and Intrinsic Permeability Profiles

The only difference between cases 2 and 3 is that in the latter the porosity and intrinsic permeability have discontinuous profiles. Figure 4.8 presents the estimated medium properties. In Figure 4.9, the estimated residual water saturation profile is plotted along with the true model state at different times during the simulation.

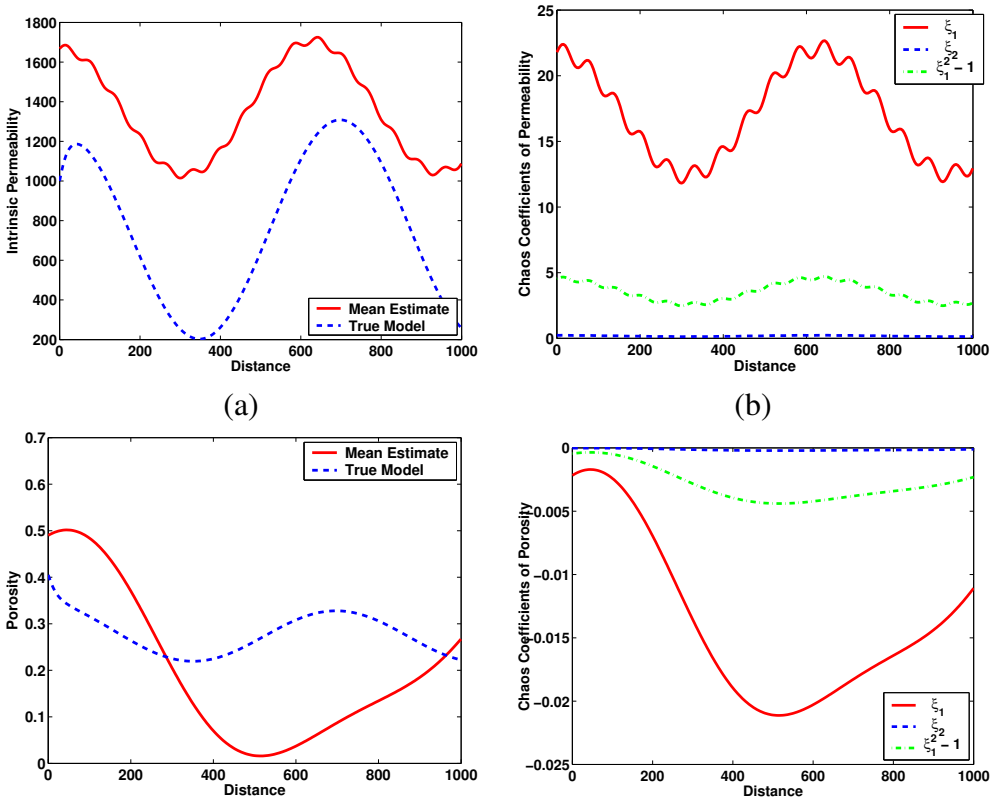


Figure 4.5: Case 2: Estimate of medium properties: (a) Mean intrinsic permeability, (b) Polynomial Chaos coefficients of intrinsic permeability, (c) mean porosity, and (d) Polynomial Chaos coefficients of porosity

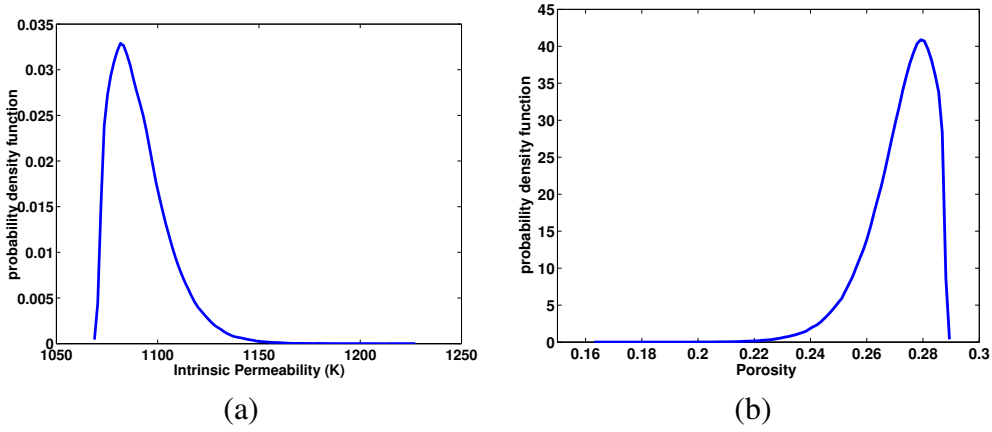
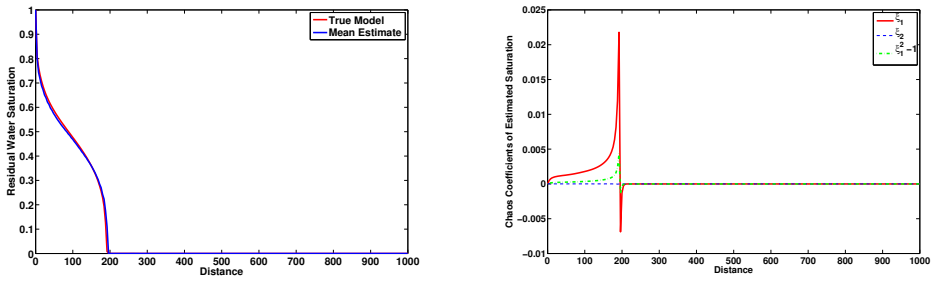
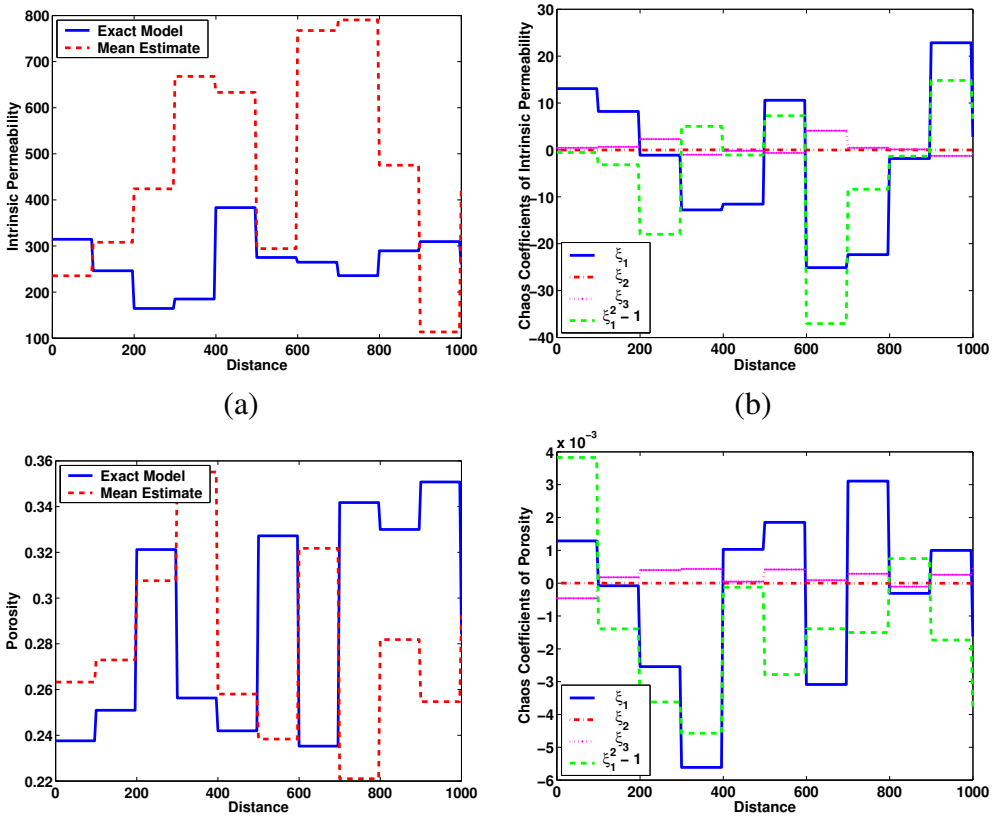


Figure 4.6: Case 2. The probability density function of the estimated medium properties: (a) intrinsic permeability, (b) porosity



(a) (b)

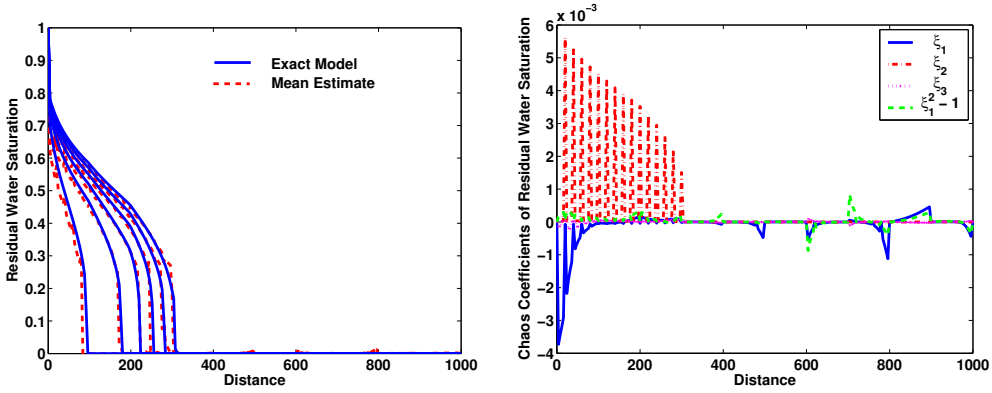
Figure 4.7: Case 2: (a) Mean residual water saturation, (b) Polynomial Chaos coefficients of the estimated residual water saturation



(a) (b)

(c) (d)

Figure 4.8: Case 3. Estimate of medium properties: (a) Mean intrinsic permeability, (b) Polynomial Chaos coefficients of intrinsic permeability, (c) mean porosity, and (d) Polynomial Chaos coefficients of porosity



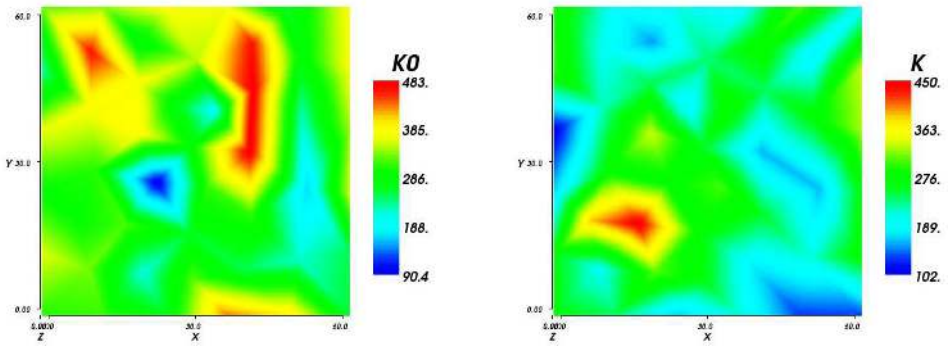
(a) (b)

Figure 4.9: Case 3: (a) Mean residual water saturation, (b) Polynomial Chaos coefficients of the last estimated residual water saturation

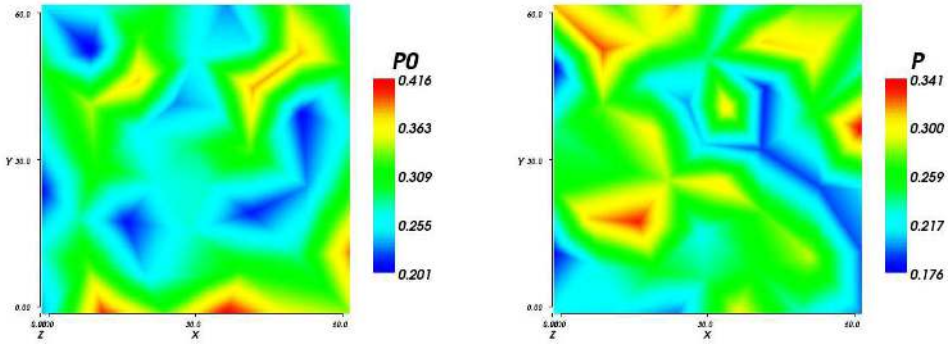
4.4.2 Two-Dimensional Water Flood Problem - Scenario 1

Two example problems are studies to explore the two dimensional water flood example. In these problems, the relative permeabilities are also represented by a polynomial approximation of the Brooks-Corey model. While the first example adopts scenario one to represent the parametric uncertainties, scenario two is used in the second.

The first example consists of a rectangular reservoir with a length of $60ft$, width of $60ft$, cross-sectional area of $1ft^2$, and constant initial water saturation $S_{w_i} = 0.20$. Oil is produced at $x = 60ft$ at a rate of $0.0323ft^3/day$, and water is injected at $x = 0$ at the same rate. Measurements are available at 20 equidistant points within the domain at a frequency of 10 time steps. Figure 4.10 gives the means of the estimated medium properties. Although the estimated and true parameters are slightly different, it can be noticed from figure 4.13 that the water front is captured accurately. This can be explained by either the uncertainty prevalent in the estimated parameters and shown in figures 4.11 and 4.12 or the non-uniqueness of the solution



(a) (b)

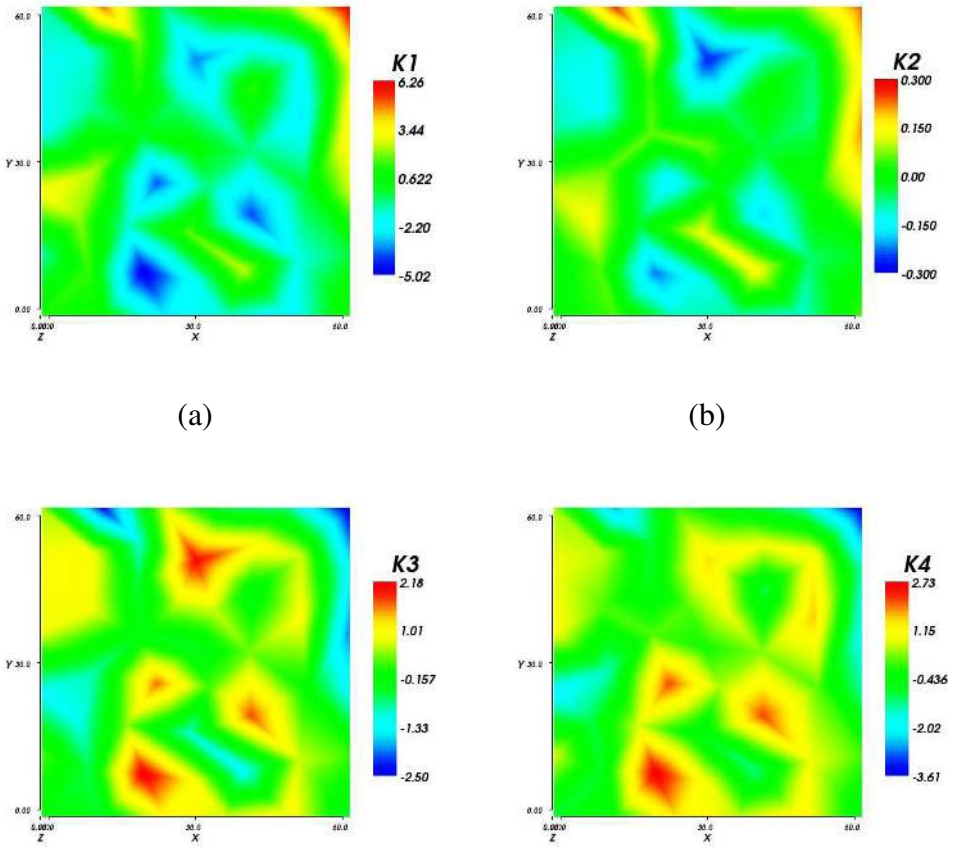


(c) (d)

Figure 4.10: (a) mean of estimated intrinsic permeability, (b) true intrinsic permeability, (c) mean of estimated porosity, and (d) true porosity

4.4.3 Two-Dimensional Water Flood Problem - Scenario 2

The second scenario is used to represent the parametric uncertainty in the second group of problems. This problem consists of a rectangular domain of a length of $100ft$, width of $60ft$, cross-sectional area of $1ft^2$, and constant initial water saturation $S_{w_i} = 0.20$. Oil is produced from one well at a rate of $7.94ft^3/day$, and water is injected from two different point sources at rates of 3.83 and $4.11ft^3/day$, respectively. Figure 4.14 shows location of these wells along with the measurement locations. In this example,



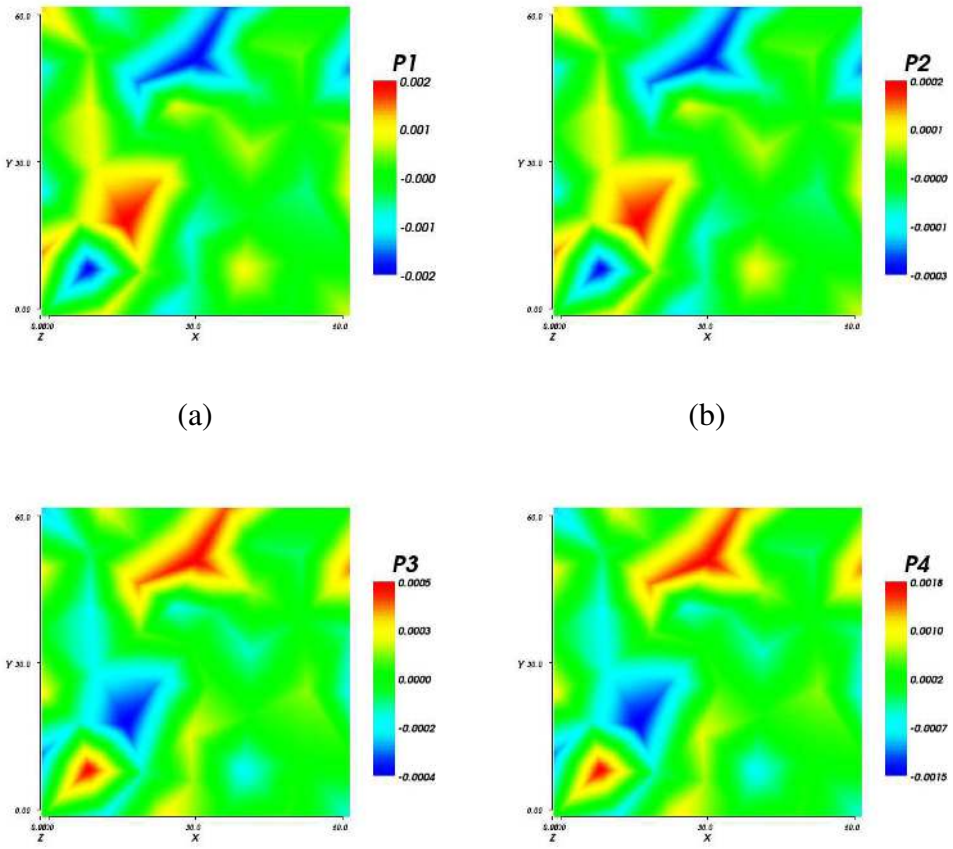
(a) (b)

(c) (d)

Figure 4.11: Polynomial Chaos coefficients of the estimated intrinsic permeability, (a) ξ_1 (b) ξ_2 , (c) ξ_3 , and (d) $\xi_1^2 - 1$

measurements are also available at a frequency of 10 time steps. Figure 4.15 presents the the medium properties of the forward problem used to generate the measurements.

In order, to assess the importance of the different terms in the Polynomial Chaos expansion used to represent the unknown model parameters, three different approximations for the parametric and modeling uncertainties are assumed.



(a) (b)

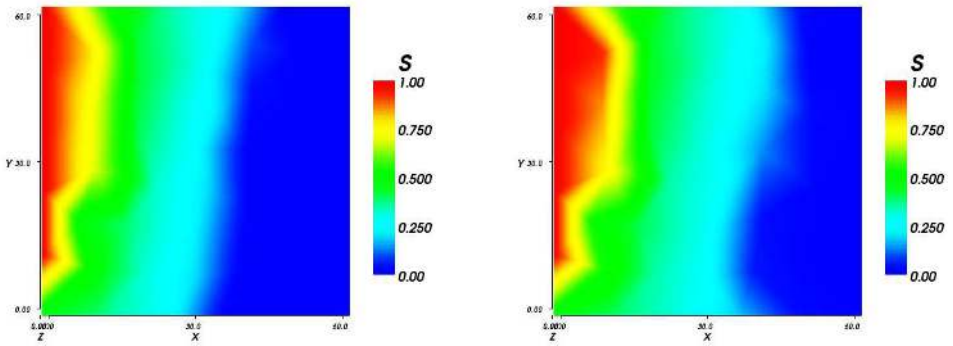
(c) (d)

Figure 4.12: Polynomial Chaos coefficients of the estimated intrinsic porosity, (a) ξ_1 (b) ξ_2 , (c) ξ_3 , and (d) $\xi_1^2 - 1$

4.4.3.1 Case 1: Three Dimensions First Order Approximation

For this part, it is assumed that the unknown model parameters are represented as:

$$\alpha = \exp(\alpha_0 + \alpha_1 \xi_1), \tag{4.33}$$



(a) (b)

Figure 4.13: (a) Mean of the estimated residual water saturation, (b) true residual water saturation

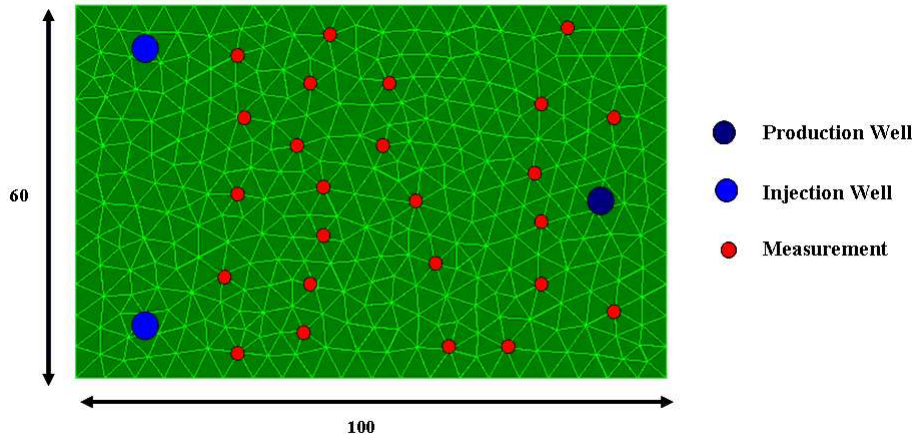


Figure 4.14: Example 2 Scenario 2: Problem Description

where α_0 and $\alpha 1$ are the unknown coefficients to be estimated using the filtering scheme. In this sub-example, the modeling and measurement error are considered Gaussian represented by ξ_2 and ξ_3 respectively. It is important to note that via this representation, the parametric uncertainty is independent from other noise sources.

Figure 4.16 presents the agreement between the mean estimated and the actual water saturation profiles. Figure 4.17 represents the coefficients of the exponential chaos

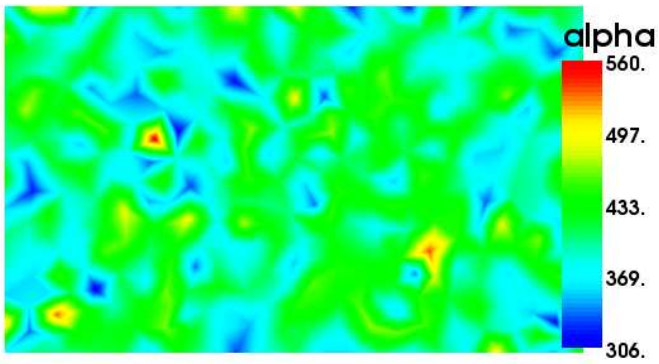


Figure 4.15: Example 2 Scenario 2: The Medium properties of the forward problem

expansion used to represent the ratio of the permeability and porosity, and figure 4.18 shows the mean and variance of the estimated ratio.

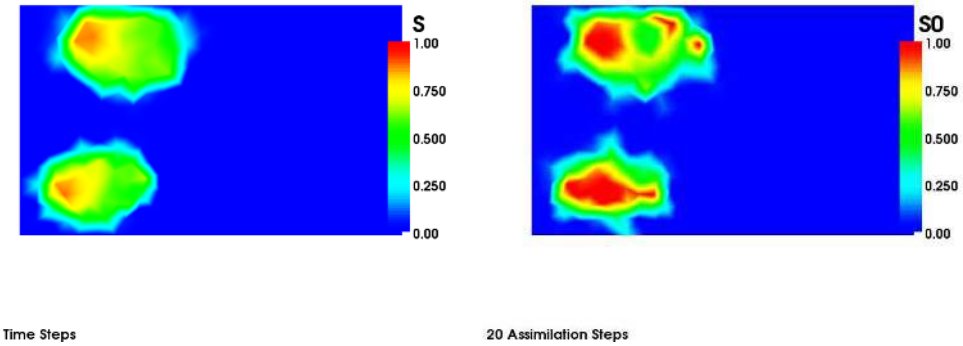
4.4.3.2 Case 2: Coupled Three Dimensions Second Order Approximation

For this part, it is assumed that the unknown model parameters are represented as:

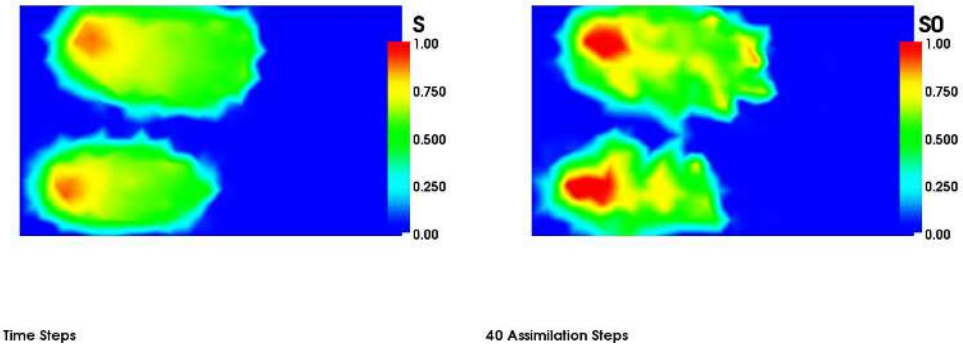
$$\alpha = \exp(\alpha_0 + \alpha_1\xi_1 + \alpha_2\xi_2 + \alpha_3\xi_3), \tag{4.34}$$

where α_0, $\alpha 1$, α_2 and α_3 are the unknown coefficients to be estimated using the filtering scheme. In this sub-example, the modeling error is considered Gaussian represented by ξ_3, while the modeling uncertainties are modeled as a second order one dimensional expansion in terms of ξ_1 and $\xi_1^2 - 1$.

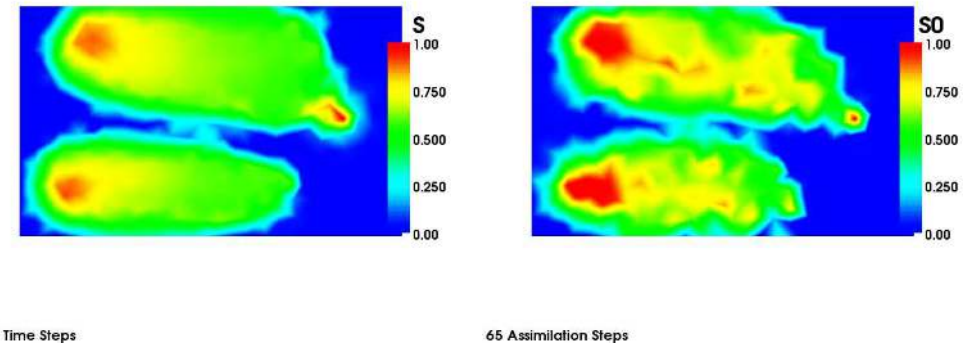
Figure 4.19 shows that using the latter approximation, a better agreement between the mean estimated and the actual water saturation profiles is achieved. Figure 4.20



(a) (b)

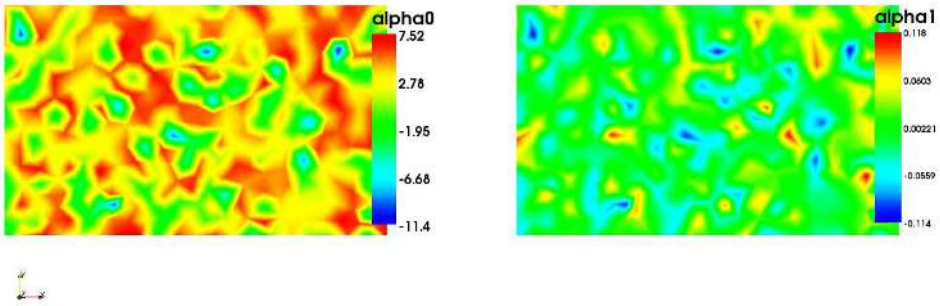


(c) (d)



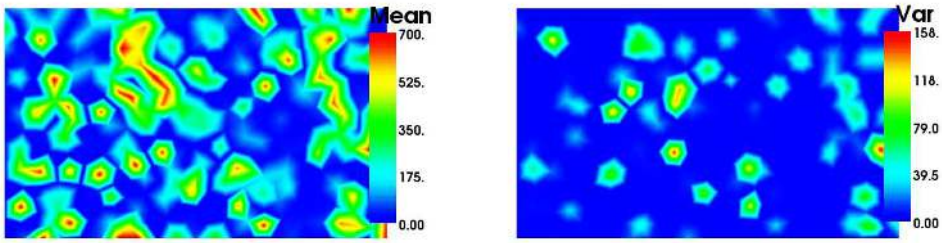
(e) (f)

Figure 4.16: Case 1: True (left) and Estimated (right) Water Saturation Profiles



(a) (b)

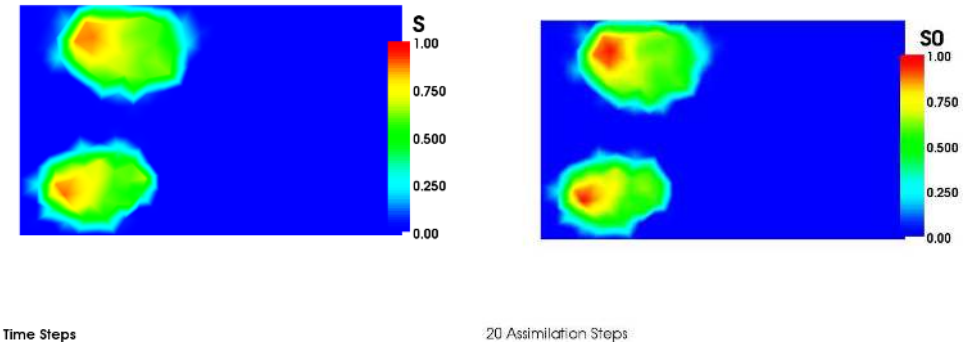
Figure 4.17: Case 1: Estimated Chaos Coefficients of the exponential representation of the Medium properties



(a) (b)

Figure 4.18: Case 1: The mean and variance of Estimated Medium property $\alpha = \frac{K}{\phi}$

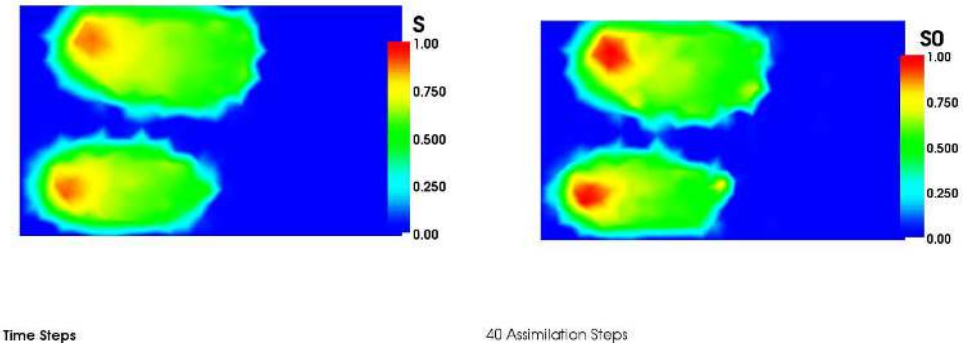
represents the coefficients of the exponential chaos expansion used to represent the ratio of the permeability and porosity, and figure 4.21 shows the mean and variance of the estimated ratio.



200 Time Steps

20 Assimilation Steps

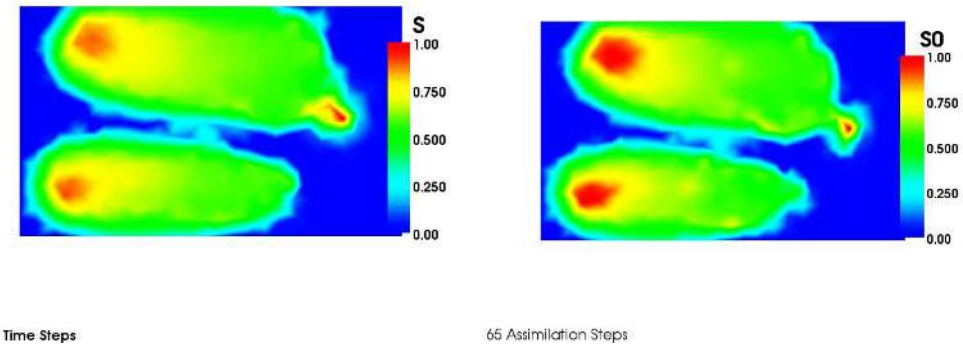
(a) (b)



400 Time Steps

40 Assimilation Steps

(c) (d)



650 Time Steps

65 Assimilation Steps

(e) (f)

Figure 4.19: Case 2: True (left) and Estimated (right) Water Saturation Profiles

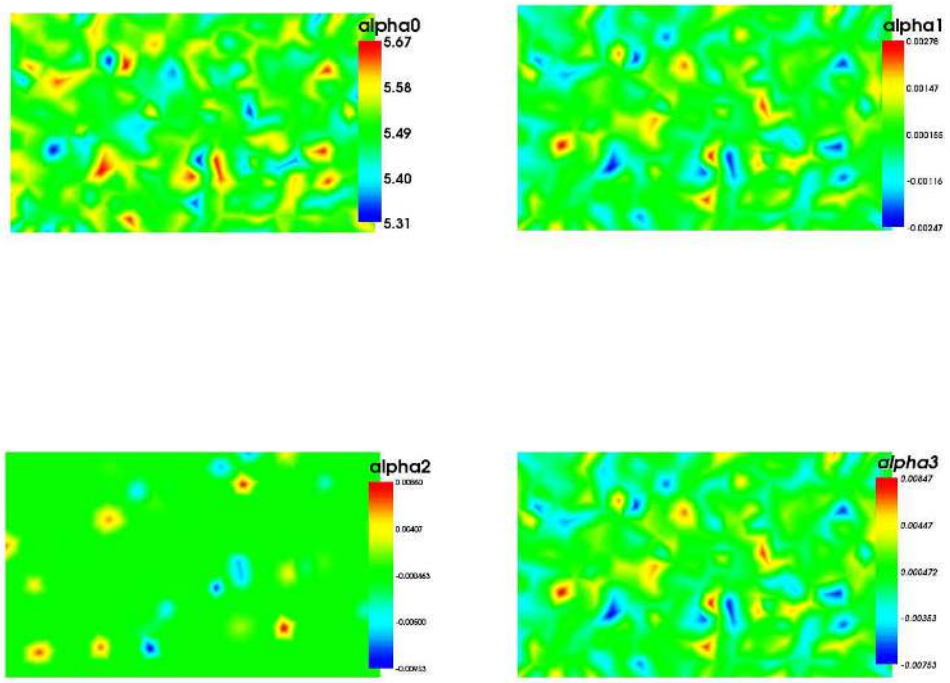


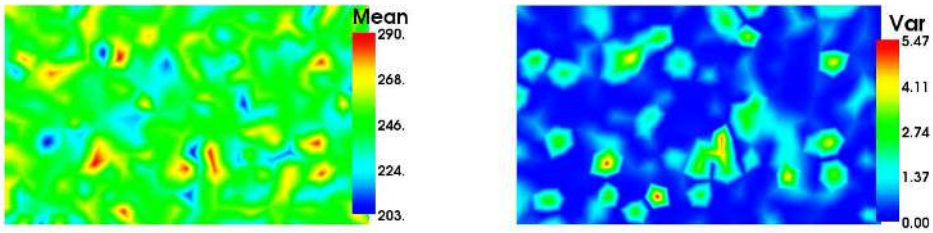
Figure 4.20: Case 2: Estimated Chaos Coefficients of the exponential representation of the Medium properties

4.4.3.3 Case 3: Coupled Ten Dimensions Second Order Approximation

In case three, it is assumed that the unknown model parameters are represented as:

$$\alpha = \exp(\alpha_0 + \sum_{i=1}^{10} \alpha_i \xi_i), \tag{4.35}$$

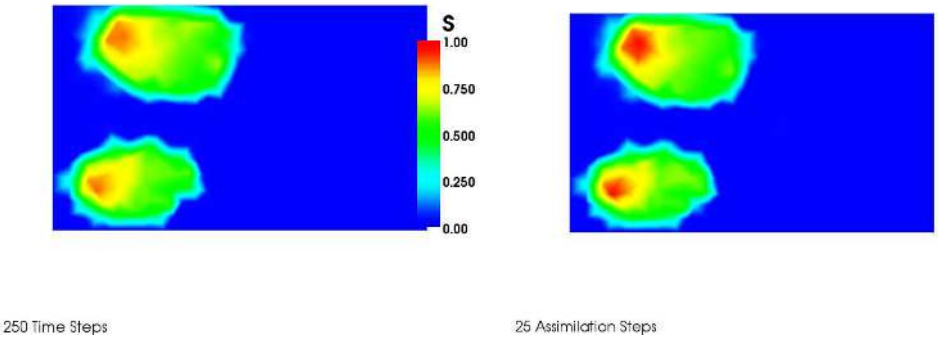
where $\{\alpha_i\}_{i=1}^{10}$ are the unknown coefficients to be estimated using the filtering scheme. In this sub-example, the modeling error is considered Gaussian represented by $\xi_1 0$, while the modeling uncertainties are modeled as a second order one dimensional expansion in terms of ξ_1 and $\xi_1^2 - 1$.



(a) (b)

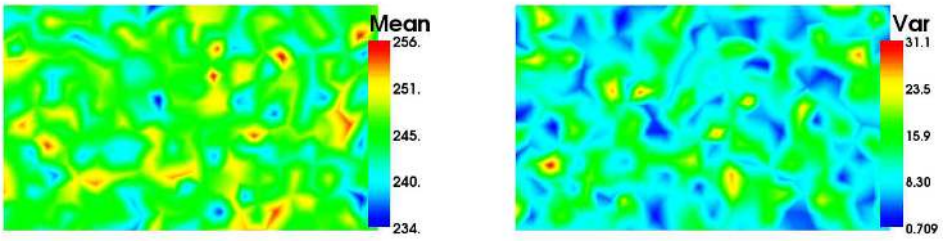
Figure 4.21: Case 2: The mean and variance of Estimated Medium property $\alpha = \frac{K}{\phi}$

Figure 4.22 shows the compatibility between the mean estimated and the actual water saturation profiles, and figure 4.23 shows the mean and variance of the estimated ratio.



(a) (b)

Figure 4.22: Case3: True (left) and Estimated (right) Water Saturation Profiles



(a) (b)

Figure 4.23: Case 3: The mean and variance of Estimated Medium property $\alpha = \frac{K}{\phi}$

4.5 Control of Fluid Front Dynamics

Having characterized the reservoir simulation model, maximization of the displacement efficiency becomes the new target. Such optimization is possible by controlling the injection rate for a fixed arrangement of injection and production points. In this study a fundamental approach which relies on the Polynomial Chaos Kalman Filter to control the injection rates is provided. The benefits of such an approach are best realized when "smart" wells having both measurement and control equipment are used. The approach consists of two filtering loops as is portrayed in figure 4.24. The measurements are used to update the model parameters, and based on the most recent updates another control loop is used to control the injection rates. The objective is to minimize the mismatch between the predicted front and a pre-specified target. The methodology is demonstrated in a simple water flooding example using two injectors and multiple producers, each equipped with several individually controllable inflow control valves. The model parameters (permeabilities) and dynamic states (pressures and saturations) are updated using measurements of the water saturation at various boreholes within the medium. The setup of the problem is shown in figure 4.25. The problem objective is to maintain

a uniform flow by adjusting the inflow rates. Two variations of the problem are solved. The first assumes that the medium properties are known a priori, and thus the problem at hand simplifies to a control problem only. The second problem aims at estimating both the medium properties and controlling the injection rate to maintain a uniform front.

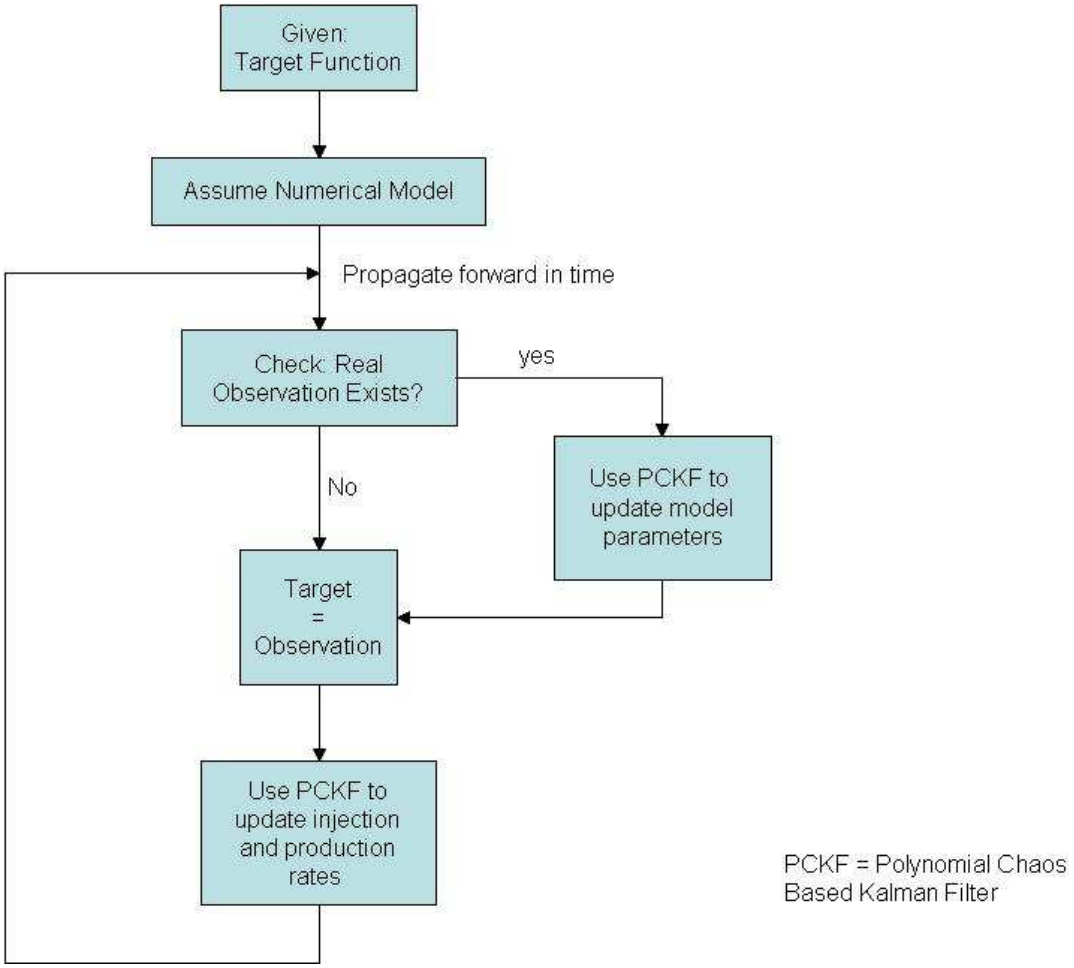


Figure 4.24: Front control flow chart

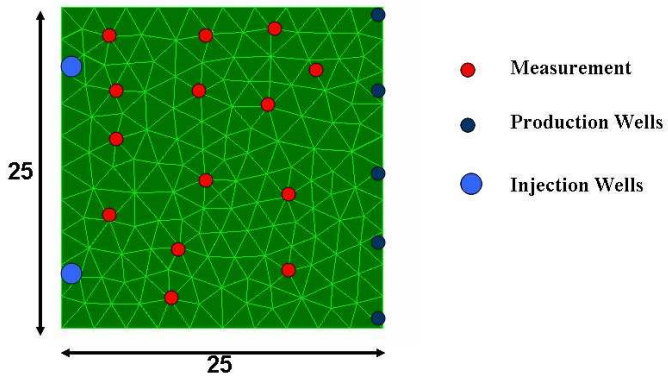


Figure 4.25: Flow Control Example: Problem Setup

4.5.1 Example 1: Known Medium Properties

In the example 1, it is assumed that the medium properties are given. The ratio K/P is represented as $\exp(\alpha)$ where α is shown in figure 4.26. Therefore, the problem simplifies to a control problem only where the injection rates are adjusted every 10 time steps so as to minimize the difference between the predicted front and a pre-specified target function. Two sources of uncertainty are represented in this example. The first is related to the uncertainty in the injection rate. Although the injection rate is a deterministic concept in practice, the uncertainty associated with it is modeled as a one dimensional second order polynomial chaos expansion. This can be explained by associating the scatter within the estimated rate with that of the predicted front. The aim is to determine a combination of rates that will render the desired front. The second source of uncertainty is associated with the exactness of the target function. If it have a very small tolerance, the latter uncertainty vanishes. In this problem it is modeled as a Gaussian noise with a relatively small variance. Figure 4.27 shows the evolution of the estimated injection rate with time. Figure 4.28 gives a comparison between the predicted and estimated water saturation profiles at different times during the simulation.

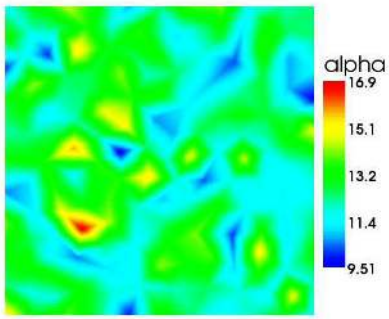
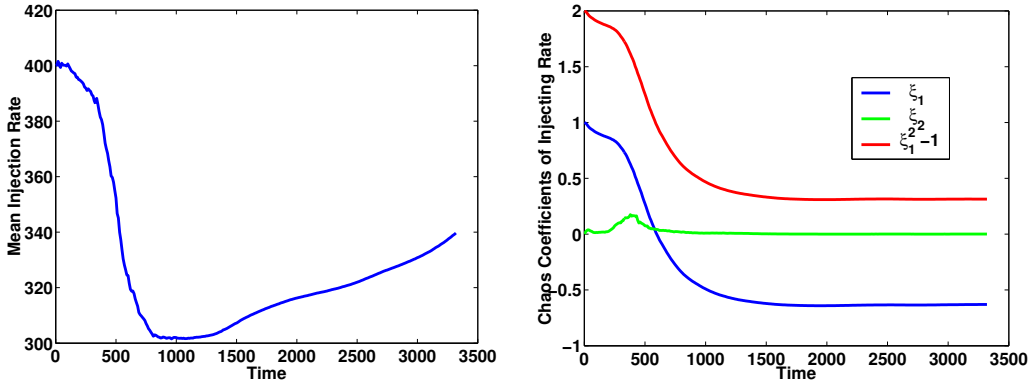


Figure 4.26: The coefficient α representing the Medium Properties



(a) (b)

Figure 4.27: Example 1: The estimated injection rate (a) Mean (b) Higher Order Coefficients

4.5.2 Example 2: Stochastic Medium Properties

The second example is a combination of the reservoir characterization problem described in the previous section and that of the control problem presented in example 1. Here it is assumed that the medium properties are random and not known a priori. Thus the ratio K/P is modeled as:

$$\alpha = \exp(\alpha_0 + \alpha_1\xi_1 + \alpha_2\xi_2 + \alpha_3\xi_3), \tag{4.36}$$

where α_0, $\alpha 1$, α_2 and α_3 are the unknown coefficients to be estimated using the filtering scheme. Furthermore, in this example it is assumed that the characteristic model is not known exactly, and therefore modeling errors are introduced. These are represented as one dimensional PC expansions, ξ_2. As in the history matching problems described earlier, the measurement errors are also represented as one dimensional PC expansions, ξ_3. Finally, the uncertainty associated the water injection rates is represented as a one dimensional second order PC expansion, ξ_1.

Figure 4.29 shows the evolution of the estimated injection rate with time. Figure 4.30 gives a comparison between the predicted and estimated water saturation profiles at different times during the simulation.

Furthermore, figure 4.31 gives the estimated coefficients α_i of the medium properties.

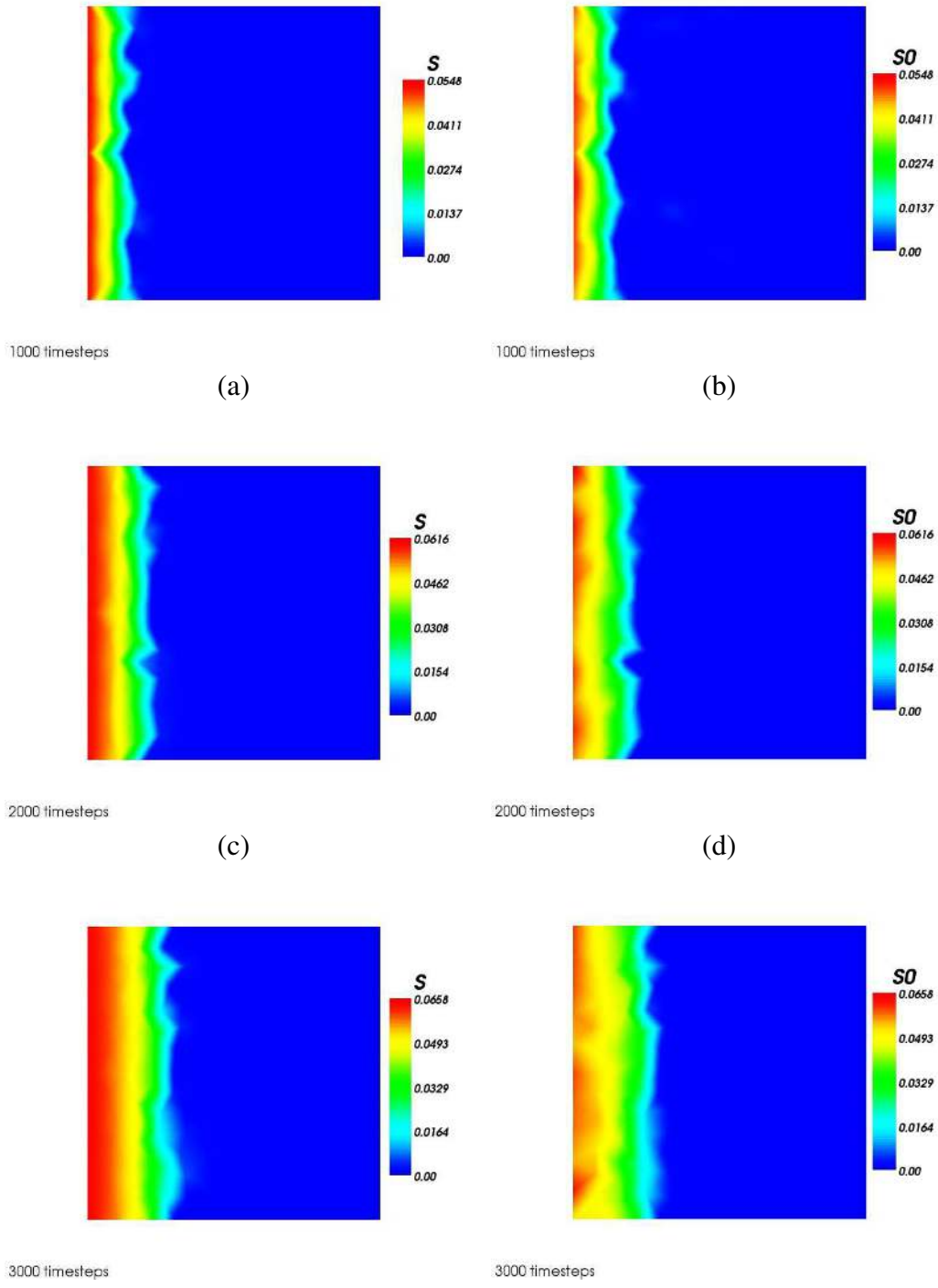
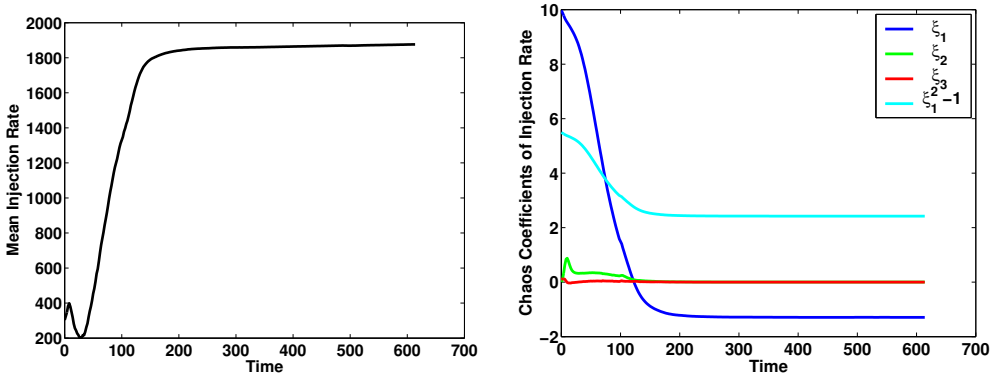


Figure 4.28: Example 1: Target (left) and Mean Estimated (right) Water Saturation Profiles



(a) (b)

Figure 4.29: Example 2: The estimated injection rate (a) Mean (b) Higher Order Coefficients

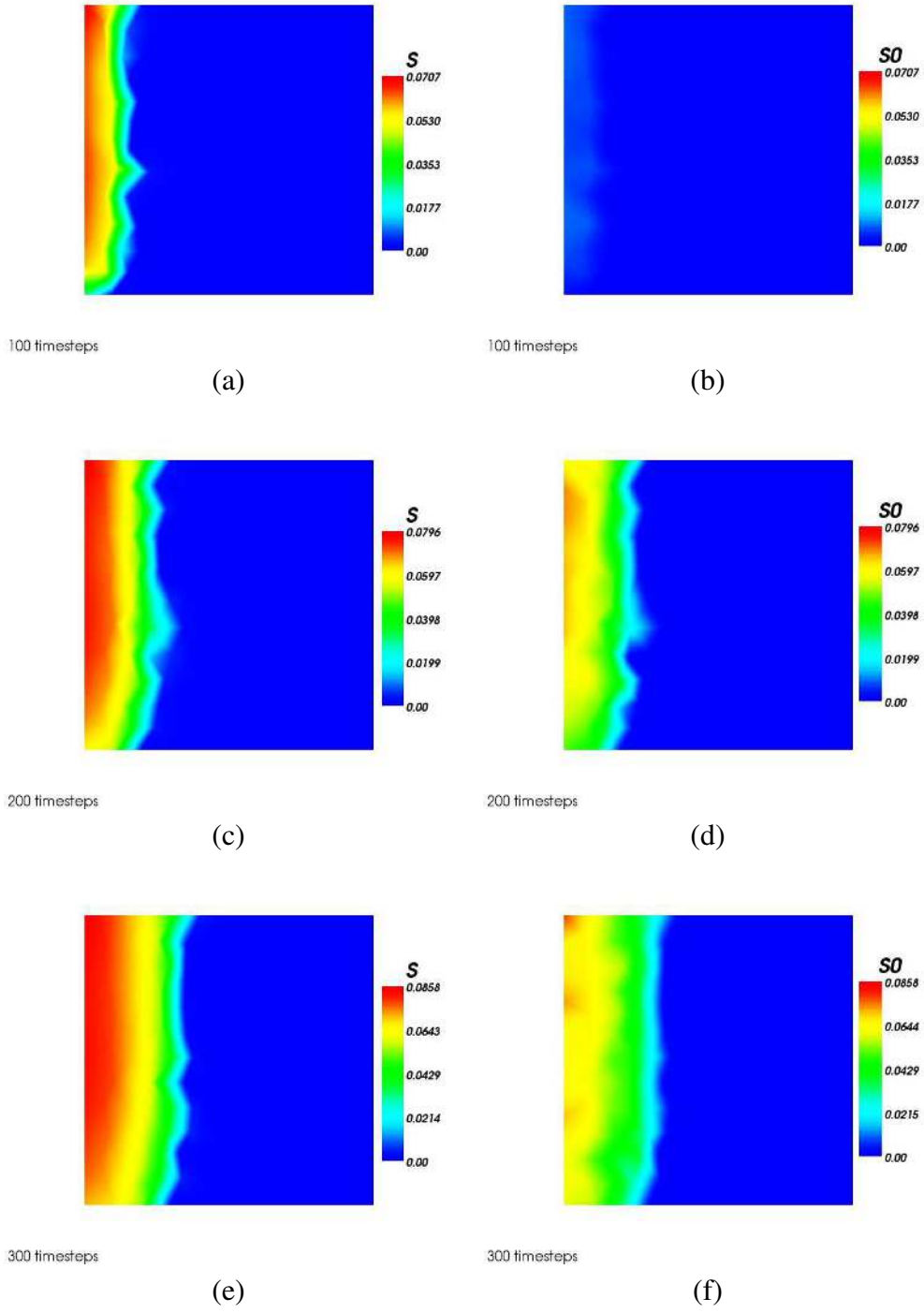


Figure 4.30: Example 2: Target (left) and Mean Estimated (right) Water Saturation Profiles

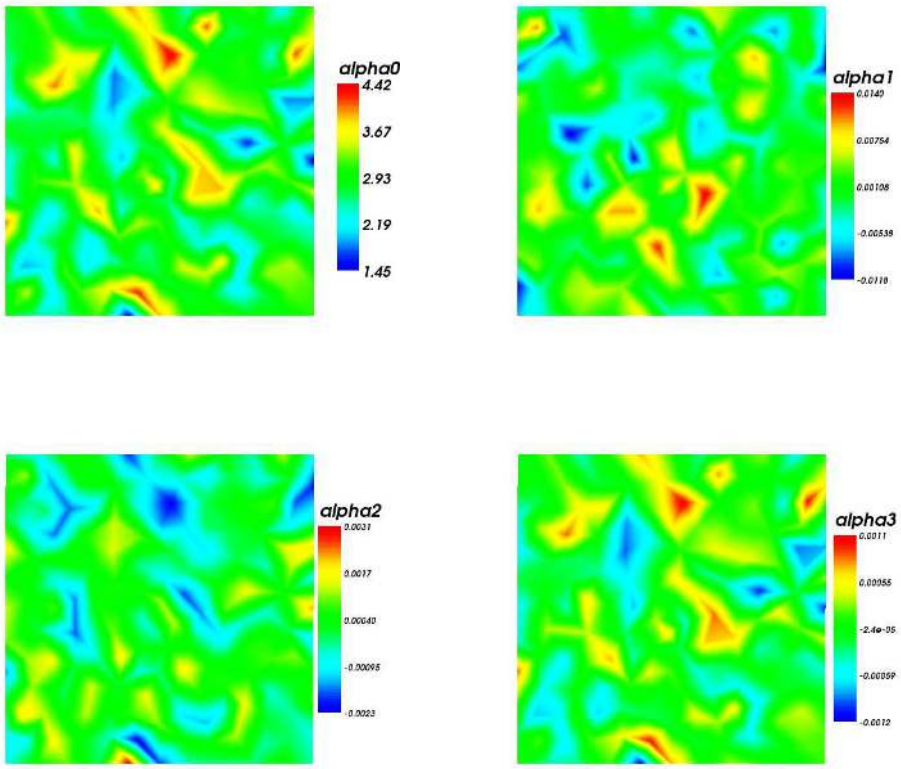


Figure 4.31: Example 2: estimated coefficients of the medium properties

Chapter 5

Structural Health Monitoring of Highly Nonlinear Systems

5.1 Introduction

With the recent developments in monitoring technologies such as high performance sensors, optical or wireless networks, and the global position system, health monitoring of civil structures became a more accurate, faster, and cost efficient process. However, significant challenges associated with modeling the physical complexity of systems comprising these structures remain. This is mainly due to the fact that these systems exhibit non-linear dynamical behavior with uncertain and complex governing laws. These uncertainties render standard system identification techniques either unsuitable or inefficient. Therefore, the need rises for robust system identification algorithms that can tackle the aforementioned challenges. This has been a very active research area over the past decade [GS95, LBL02, ZFYM02, FBL05, FIIN05, GF06].

Sequential data assimilation has been widely used for structural health monitoring and system identification problems. Many extensions of the Kalman Filter were developed as adaptations to important classes of these problems. While the Extended Kalman Filter may fail in the presence of high non-linearities, Monte Carlo based Kalman Filters usually give satisfactory results. The Ensemble Kalman Filter (EnKF) [Eve94] was recently used for damage detection in strongly nonlinear systems [GF06], where it is

combined with non-parametric modeling techniques to tackle structural health monitoring for non-linear systems. The EnKF uses a Monte Carlo Simulation scheme for characterizing the noise in the system, and therefore allows representing non-Gaussian perturbations. Although this combination gives good results, it requires a relatively accurate representation of the non-linear system dynamics. It also requires a large ensemble size to quantify the non-Gaussian uncertainties in such systems and consequently imposes a high computational cost.

The objective of this study is to propose a new system identification approach based on coupling robust non-parametric non-linear models with the Polynomial Chaos methodology [GS03] in the context of the Kalman Filtering techniques. The proposed approach uses a Polynomial Chaos expansion of the nonparametric representation of the system's non-linearity to statistically characterize the system's behavior. A filtering technique that allows the propagation of a stochastic representation of the unknown variables using Polynomial Chaos is used to identify the chaos coefficients of the unknown parameters in the model. The introduced filter is a modification of the EnKF that uses the Polynomial Chaos methodology to represent uncertainties in the system. This allows the representation of non-Gaussian uncertainties in a simpler, less taxing way without the necessity of managing a large ensemble. It also allows obtaining the probability density function of the model state or parameters at any instant in time by simply simulating the Polynomial Chaos basis.

5.2 Numerical Application

The efficiency of the proposed method is assessed by applying it to the structural health monitoring of a the four story shear building shown in figure 5.1. This model has a constant stiffness on each floor and a 5% damping ratio in all modes. All structural

elements of this frame are assumed to involve hysteretic behavior, and it is supposed that a change in the hysteretic loop of the first floor element occurs at some point. It is of utmost importance to localize that point in time and track the state of the system throughout and subsequent to that point.

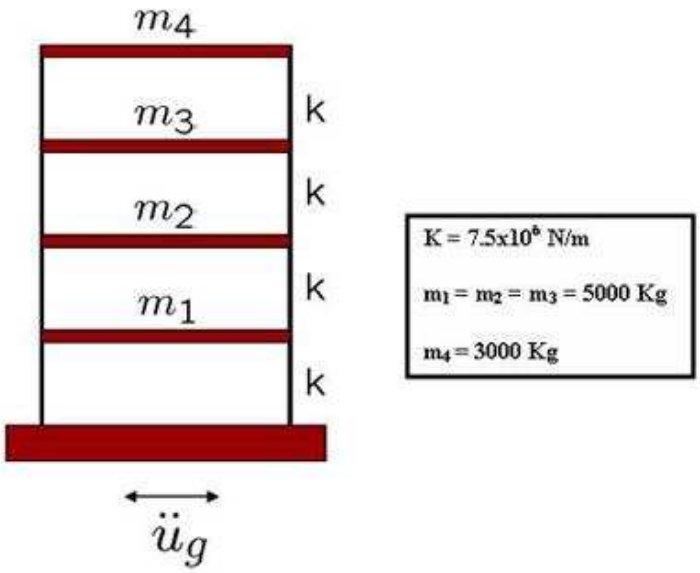


Figure 5.1: Shear Building under analysis.

A synthetically generated dataset representing measurements of the displacements and velocities at each floor is obtained by representing the hysteretic restoring force by the Bouc-Wen model, which is therefore considered as the exact hysteretic behavior of the system. Thus, the equation of motion of the system is given by,

$$\mathbf{M}\ddot{u}(t) + \mathbf{C}\dot{u}(t) + \alpha\mathbf{K}_{el}u(t) + (1-\alpha)\mathbf{K}_{in}z(x,t) = -\mathbf{M}\tau\ddot{u}_g(t), \tag{5.1}$$

where $\mathbf{M}$, $\mathbf{C}$, $\mathbf{K}_{el}$, and $\mathbf{K}_{in}$ are the mass, damping, elastic and inelastic stiffness matrices respectively; α is the ratio of the post yielding stiffness to the elastic stiffness, τ is the

Table 5.1: Bouc-Wen Model Coefficients

BW Coef.	pre-change	post-change
α	0.15	0.15
β	0.1	10
γ	0.1	10
A	1	1
n	1	1

influence vector, u is the diplacement vector, x is the interstorey drift vector, and z is n-dimensional evolutionary hysteretic vector whose i^{th} component is give by the Bouc-Wen model as,

$$\dot{z}_i = A_i \dot{x}_i - \beta_i \left|\dot{x}_i\right| \left|z_i\right|^{n_i - 1} - \gamma_i \dot{x}_i \left|z_i\right|^{n_i}, i = 1, \ldots, n. \tag{5.2}$$

Table 5.1 presents the Bouc-Wen model parameters adopted in this application. The structure is subject to a base motion specified by a time series consistent with the El-Centro earthquake shown in figure 5.2, and a change of the first floor hysteric behavior is assumed to take place five seconds after the excitation.

Two monitoring scenarios are considered. In both scenarios, observations of the floor displacements and velocities are available at all floors. However, the frequency of measurements is varied to explore the impact of hardware limitations on the performance of the filter. In the first scenario, it is assumed that measurements are available every 5 time steps, while they are given every 20 time steps during the second scenario. A nonparametric representation of the system nonlinearity is adopted, and the filtering technique is used to characterize the latter representation in order to capture any ambiguous behavior of the structure examined.

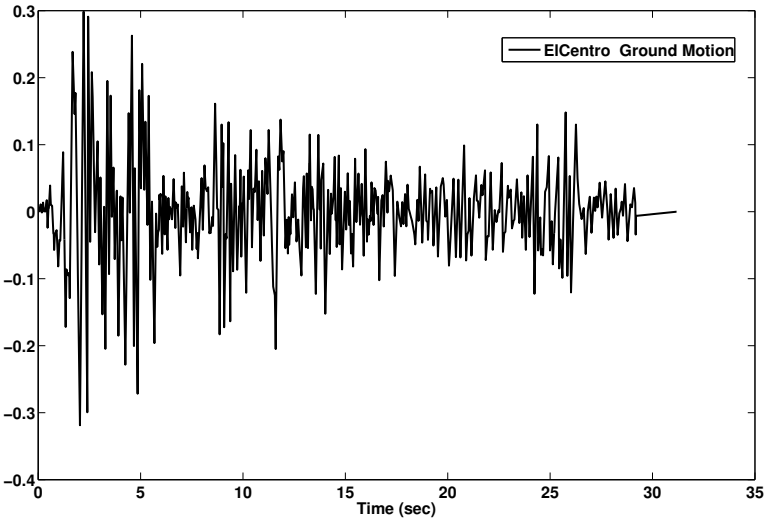


Figure 5.2: The Elcentro Excitation Applied to the Structure.

5.2.1 Non-parametric Representation of the Non-Linearity

The proposed filtering methodology is combined with a non-parametric modeling technique to tackle structural health monitoring of non-linear systems in a fashion similar to that used earlier by Ghanem [GF06], but instead of adopting a deterministic non-parametric representation of the non-linearity, a stochastic representation via Polynomial Chaos is used. The basic idea behind the non-parametric identification technique used is to determine an approximating analytical function $\hat{F}$, that approximates the actual system non-linearities, with the form of $\hat{F}$ including suitable basis functions that are adapted to the problem at hand [MCC$^+$04]. For general non-linear systems, a suitable choice of basis would be the list of terms in the power series expansion in the doubly indexed series

$$S = \sum_{i=0}^{i_{max}} \sum_{j=0}^{j_{max}} u^i \dot{u}^j, \tag{5.3}$$

where u and $\dot{u}$ are used to represent the system's displacement and velocity respectively. Therefore, if $i_{max} = 3$ and $j_{max} = 3$, the basis functions become

$$basis = \{1, \dot{u}, \dot{u}^2, \dot{u}^3, u, u\dot{u}, u\dot{u}^2, u\dot{u}^3, u^2, u^2\dot{u}, u^2\dot{u}^2, u^2\dot{u}^3, u^3, u^3\dot{u}, u^3\dot{u}^2, u^3\dot{u}^3\}. \quad (5.4)$$

This notation readily generalizes from the SDOF case presented in Eqs. 5.3 and 5.4 as shown later. In the proposed method the displacements and velocities are stochastic processes represented by their Polynomial Chaos expansion. Thus, the approximating function is also expressed as a stochastic processes via a Polynomial Chaos representation.

The model adopted within the Kalman Filter is hence given by

$$\mathbf{M}\ddot{u}(t) + \mathbf{F}(u, \dot{u}) = -\mathbf{M}\tau\ddot{u}_g(t), \quad (5.5)$$

where F is the non-parametric representation of the non-linearity whose i^{th} floor component is given by

$$\begin{aligned} F^i \approx \sum_j F_j^i(u,\dot{u})\psi_j &= \sum_j a_j^i\psi_j(\sum_k(u_k^i - u_k^{i-1})\psi_k) + \sum_j a_j^{i+1}\psi_j(\sum_k(u_k^i - u_k^{i+1})\psi_k) \\ &+ \sum_j b_j^i\psi_j(\sum_k(u_k^i - u_k^{i+1})\psi_k)^2 + \sum_j b_j^{i+1}\psi_j(\sum_k(u_k^i - u_k^{i+1})\psi_k)^2 \\ &+ \sum_j c_j^i\psi_j(\sum_k(\dot{u}_k^i - \dot{u}_k^{i-1})\psi_k) + \sum_j c_j^{i+1}\psi_j(\sum_k(\dot{u}_k^i - \dot{u}_k^{i+1})\psi_k) \\ &+ \sum_j d_j^i\psi_j(\sum_k(u_k^i - u_k^{i-1})\psi_k)(\sum_l(\dot{u}_l^i - \dot{u}_l^{i-1})\psi_l) \\ &+ \sum_j d_j^{i+1}\psi_j(\sum_k(u_k^i - u_k^{i+1})\psi_k)(\sum_l(\dot{u}_l^i - \dot{u}_l^{i+1})\psi_l). \end{aligned} \quad (5.6)$$

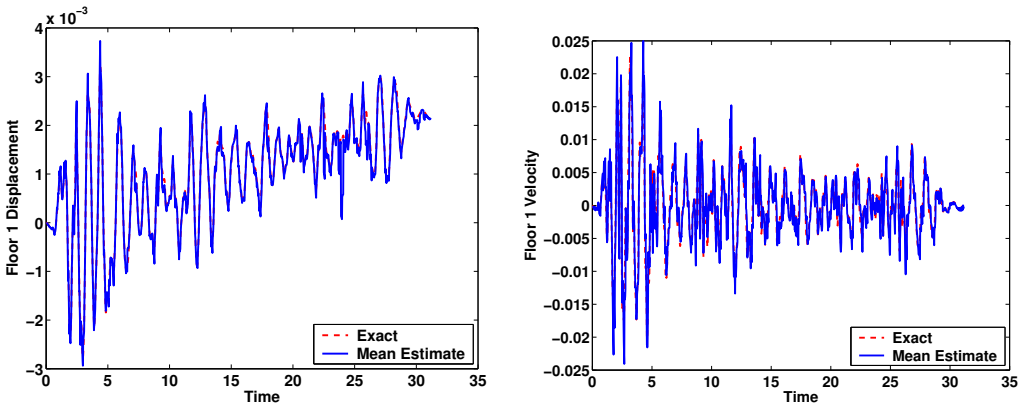
In Eq. 5.6 $\{a_j\}, \{b_j\}, \{c_j\}$, and $\{d_j\}$ represent the Chaos coefficients of the unknown parameters to be identified. The fourth order Runge-Kutta method is used for the time stepping, and a stochastic Galerkin scheme is employed to solved the system at each time step.

5.2.2 Results

In this application, it is assumed that observations of displacements and velocities from all floors are available. The noise signals perturbing both the model and measurements are modeled as first order, one dimensional, independent, Polynomial Chaos expansions having zero-mean and an RMS of 0.05 and 0.001 respectively. The parameters' uncertainties on the other hand, are modeled as second order, one dimensional, Polynomial Chaos expansions whose coefficients are to be determined in accordance with the available observations. This is done to incorporate the possibility that the unknown parameters may deviate from Gaussianity. Furthermore, it is assumed that the first floor undergoes a change in its hysteretic behavior 5 seconds after the ground excitation. The purpose of the application is to detect this behavioral change.

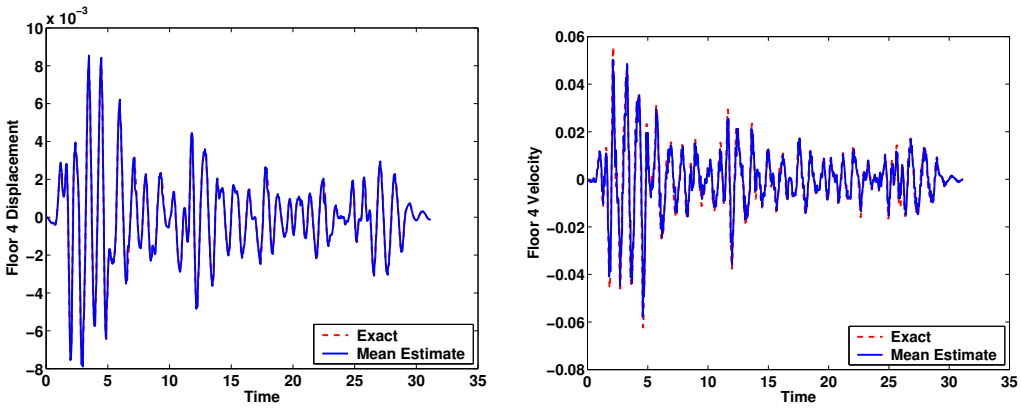
5.2.2.1 Example 1: $\Delta t = 5$ Time Steps

In the first example, it is assumed that the measurements of the floors' displacements and velocities are available every 5 time steps. Figures 5.3 and 5.4 describe the tracking of the displacement and velocity for the first and fourth floor respectively. Excellent match between the results estimated using the Polynomial Chaos based Kalman Filter and the true states is observed. Figure 5.5 presents the evolution of the mean of the unknown parameters identified by the proposed filtering technique. Error bars representing the scatter in the estimated parameters are also present in figure 5.5. The different jumps within the parameters are associated with the perks in the corresponding excitation.



(a) (b)

Figure 5.3: (a) Estimate of the first floor displacement ($\Delta t = 5$ time steps), (b) Estimate of the first floor velocity ($\Delta t = 5$ time steps).



(a) (b)

Figure 5.4: (a) Estimate of the fourth floor displacement ($\Delta t = 5$ time steps), (b) Estimate of the fourth floor velocity ($\Delta t = 5$ time steps).

Further investigation of the parameters indicate that the main changes take place in the first floor following the $5sec$ time interval. Note that the parameters a and c in floors 1 and 2 undergo the greatest jumps since they are associated with inter-story drift and velocity, respectively. One of the main advantages of using the Polynomial Chaos Kalman filter is that is provides a scatter around the estimated parameters. This is represented by the probability density functions corresponding to each of the estimated parameters. Figure 5.6 presents the pdf's of the estimated floor 1 parameters.

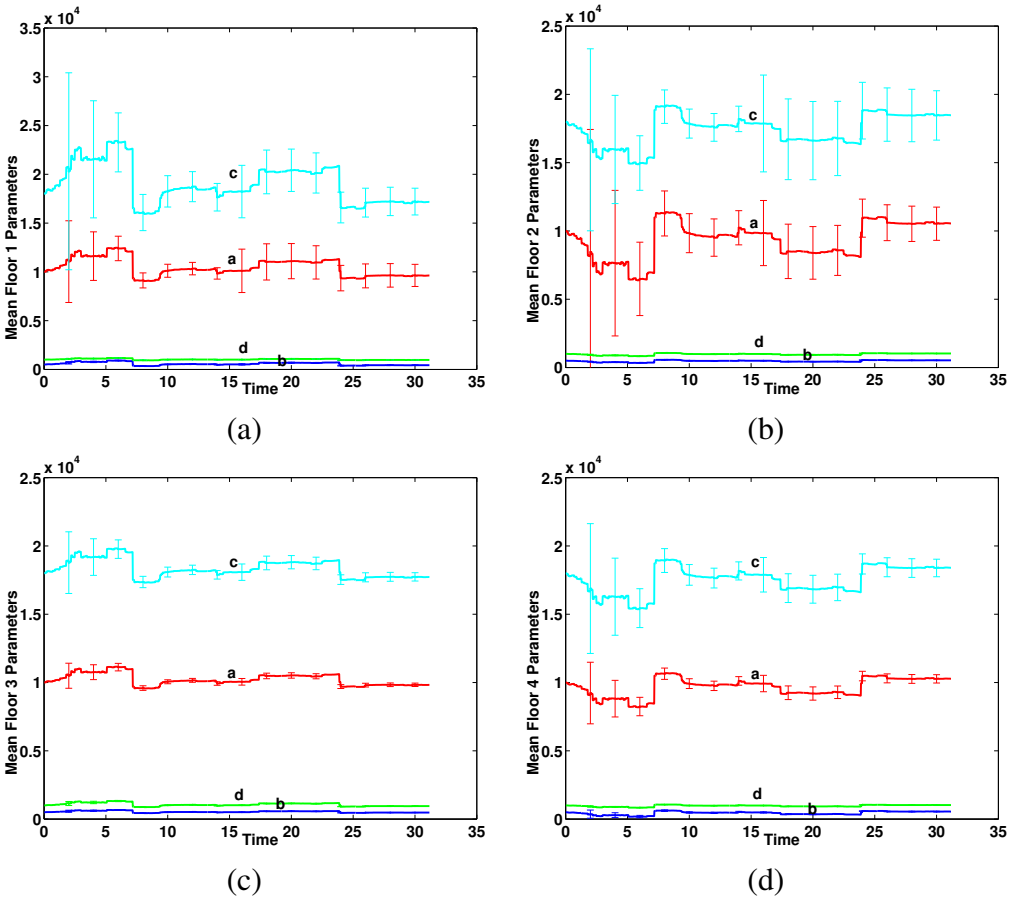


Figure 5.5: Estimate of the Mean floor parameters ($\Delta t = 5$ time steps): (a) first floor, (b) second floor, (c) third floor, and (d) fourth floor

5.2.2.2 Example 2: $\Delta t = 20$ Time Steps

In the second run, to test the limitation of the monitoring hardware and software, it is assumed that the measurements are available every 20 time steps. Figures 5.7 and 5.8 describe the tracking of the displacement and velocity for the first and fourth floor respectively. While the match between the results estimated using the Polynomial Chaos based Kalman Filter and the true states is observed is not as good as the previous example, the estimate still gives a good representation of the true state. Figure 5.9 presents the evolution of the mean of the unknown parameters identified by the proposed filtering technique. It is again readily noted that the change in hysteretic behavior is concentrated

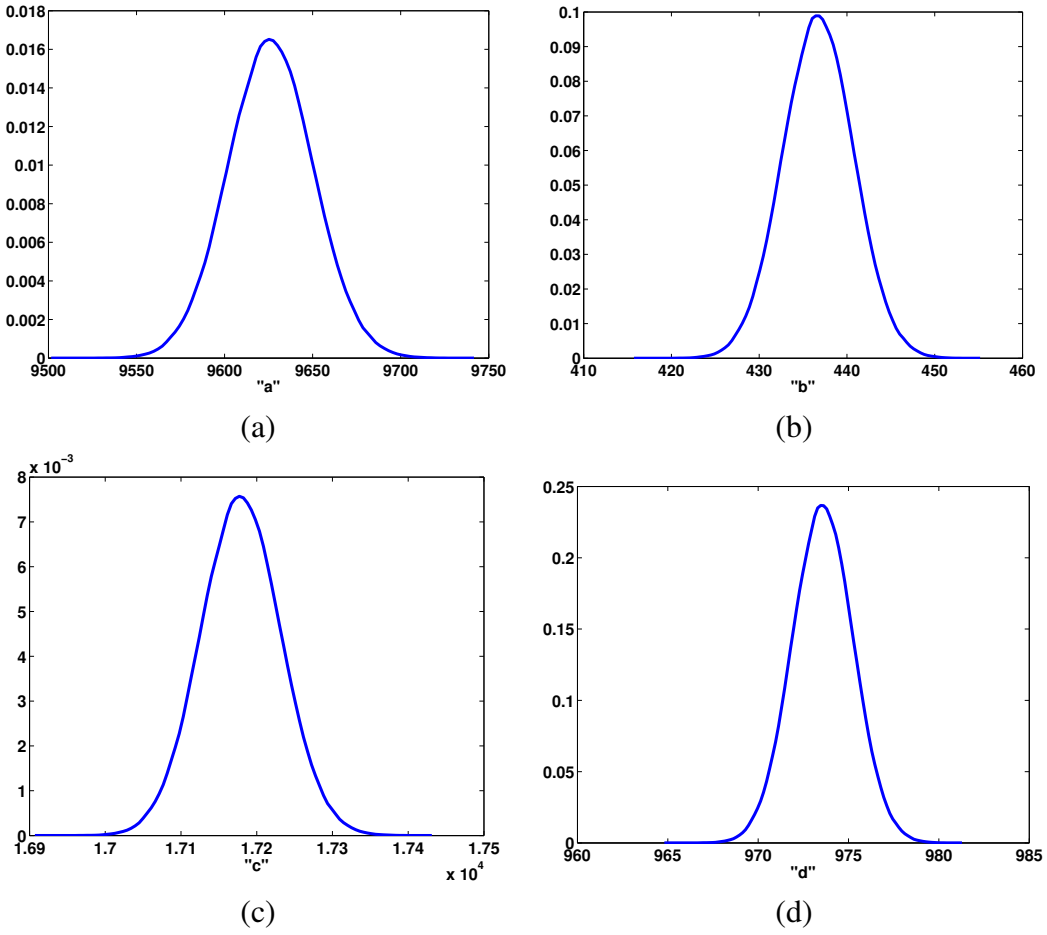
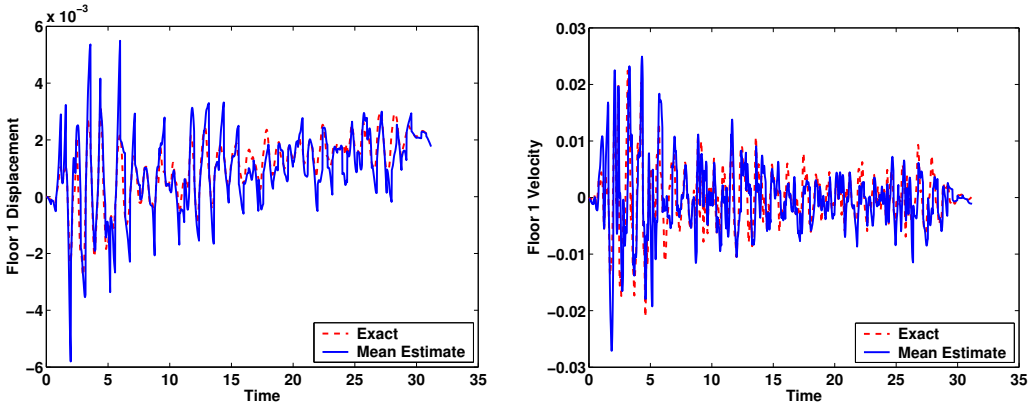


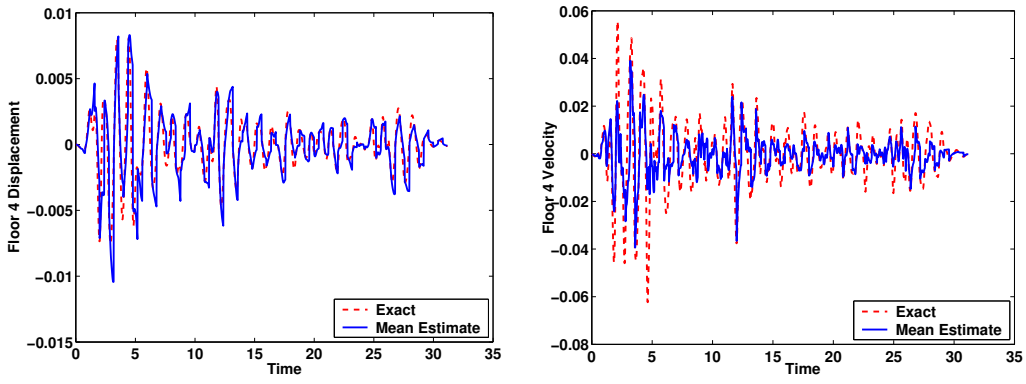
Figure 5.6: Probability density function of estimated floor 1 parameters ($\Delta t = 5$ time steps): (a) “a”, (b) “b”, (c) “c”, and (d) “d”

at the first floor at a time after $5sec$. Figure 5.10 presents the probability density functions associated with the estimated first floor parameters. Although the mean of the identified parameters is closely comparable to that in example 1, it is obvious that the associated scatter has increased, and thus indicating a higher coefficient of variation.



(a) (b)

Figure 5.7: (a) Estimate of the first floor displacement ($\Delta t = 20$ time steps), (b) Estimate of the first floor velocity ($\Delta t = 20$ time steps).



(a) (b)

Figure 5.8: (a) Estimate of the fourth floor displacement ($\Delta t = 20$ time steps), (b) Estimate of the fourth floor velocity ($\Delta t = 20$ time steps).

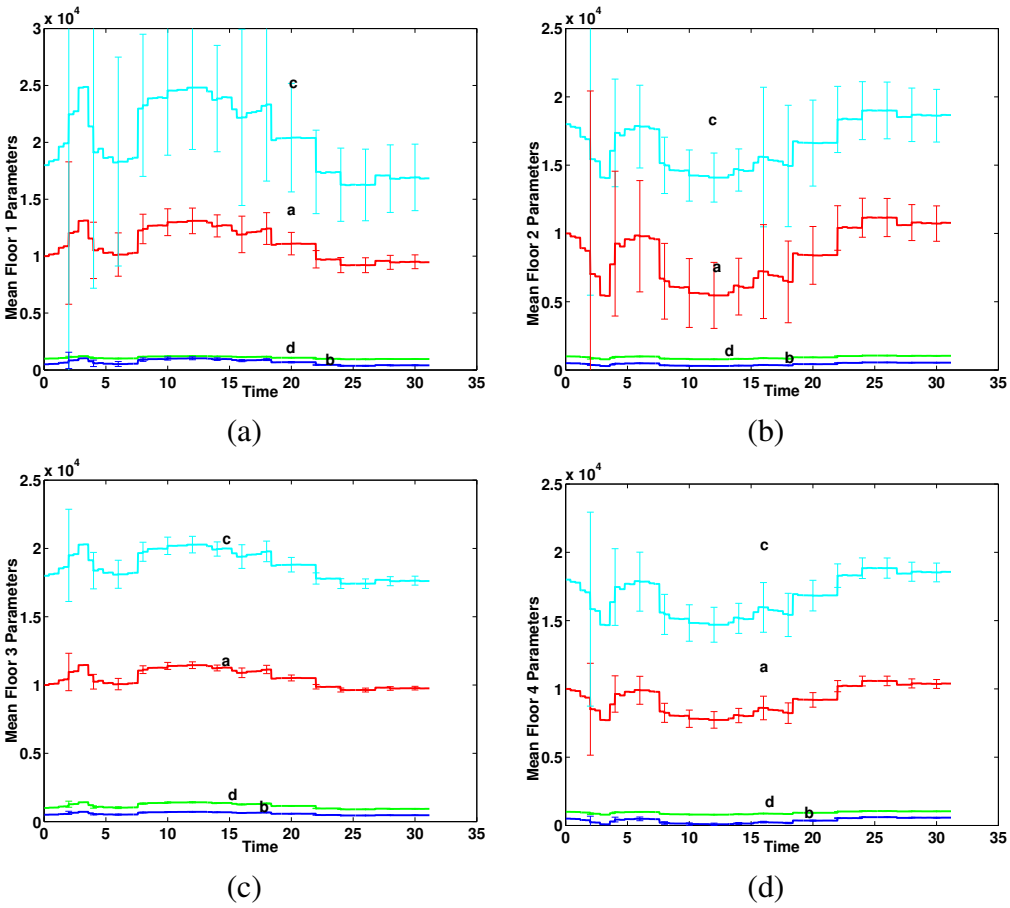


Figure 5.9: Estimate of the Mean floor parameters ($\Delta t = 20$ time steps): (a) first floor, (b) second floor, (c) third floor, and (d) fourth floor

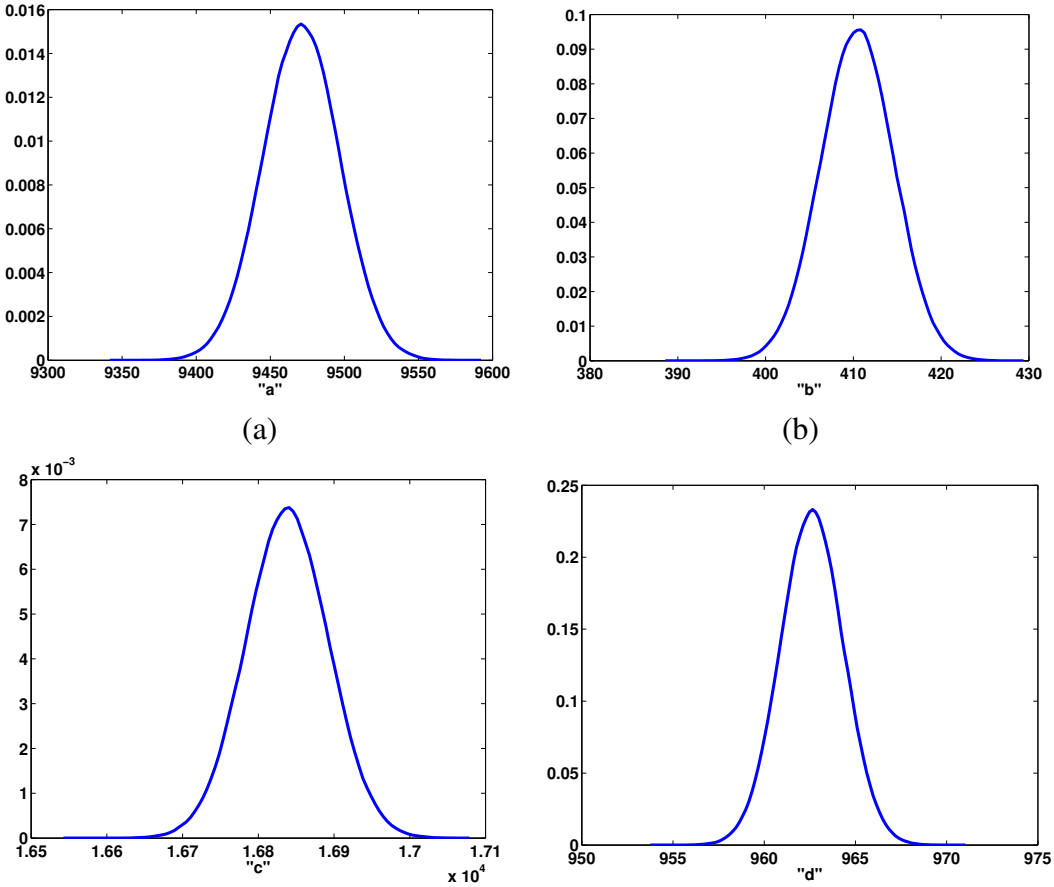


Figure 5.10: Probability density function of estimated floor 1 parameters ($\Delta t = 20$ time steps): (a) “a”, (b) “b”, (c) “c”, and (d) “d”

Chapter 6

Sundance

Sundance is a system for rapid development of high-performance parallel finite-element solutions of partial differential equations [Lon04]. This toolkit software is developed mainly by Kevin Long in SANDIA National Laboratories. The high level nature of Sundance relieves from worrying about the tedious and error-prone bookkeeping details. It also allows a high degree of flexibility in the formulation, discretization, and the solution of the problem. Moreover, Sundance can both assemble and solve problems in parallel. One of its design goal, is to make parallel solutions as simple as serial ones.

Sundance is written in C++ programming language with optional python wrappers. Therefore, it is necessary to know how to write and compile C++ codes to be able to use it. Only a fraction of the objects and methods that make up Sundance are ever needed in user code; most are used internally by Sundance.

Solution of partial differential equations is a complicated endeavor with many subtle difficulties, and there is no standard approach to tackle all PDE's. Sundance is a set of high-level objects that will allow the user to build his own simulation code with a minimal effort. Although, these objects shield the user from the rather tedious bookkeeping details required in writing a finite-element code, they still require him to understand how to do a proper formulation and discretization of a given problem.

6.1 Guidelines for Using Sundance

- Finite Element methods for solving PDE's are based on the PDE's weak form. Therefore, Sundance requires describing PDE's by employing a high-level symbolic notation for writing weak forms.

- The spatial discretization or mesh of problem can be created internally for simple one dimensional or two dimensional problems, but for more complex problems it is required to use a third-party mesher and import that mesh into Sundance.

- The FEM approximates PDE solution by a system of linear or nonlinear algebraic equations. Many factors influence this approximation, namely, choice of weak form, method of imposing Boundary Conditions, basis functions for the unknowns in the problem, and more.

- Sundance has a built in Library linking to the famous TRILINOS solvers. Therefore it suffice to identify the solver required for the problem and specify its parameters such as the tolerance level, the maximum number of iterations, linearization or iteration method for nonlinear problems , and more.

- Solution of a time-dependent problem can be reduced to solving a sequence of linear (or possibly nonlinear) problems, by timestepping or marching. Again, there are many possible marching algorithms [Lon04].

6.2 SSFEM Using Sundance

In order to make use of Sundance in solving stochastic partial differential equations via the SSFEM, certain modifications to the code are done. A spectral library is developed

within Sundance to allow the high level implementation of the SSFEM. It entails defining a spectral basis whose type, dimensionality and order are to be specified along with methods for computing the expectations over the space approximated by the specified basis.

6.2.1 Example: One-dimensional Bar with a Random Stiffness

Consider a bar element of Length L, clamped at both ends and subject to a deterministic static uniform body load P. It is assumed that the modulus of elasticity E associated with the bar is a realization of a lognormal random process indexed over the spatial domain occupied by the bar. It is also assumed that the bar has a uniform cross-sectional area, A.

The governing differential equation for this problem is,

$$\frac{d}{dx}(AE\frac{du}{dx}) = P. \tag{6.1}$$

Although, this is a simple example, many of the complex Sundance syntax can be demonstrated through its solution scheme. The boundary conditions, BCs, for the above problem are,

$$u = 0 \ at \ x = 0 \ and \ x = L.$$

With the equation and BCs in hand, we can write the problem in weak form. Multiplying by a test function v and integrating by parts, we have

$$\int_{interior} \frac{dv}{dx} AE\frac{du}{dx} - \int_{interior} vP \tag{6.2}$$

6.2.2 Step by Step Explanation

In this section a walk through the code for solving the problem is detailed. At the end, the complete code is listed for reference.

6.2.2.1 Boilerplate

A dull but essential first step is to show the boilerplate C++ common to nearly every Sundance code:

```
#include "Sundnace.hpp"

int main(int argc, char** argv)
{
  try
    {
      Sundance::init(&argc, &argv);
      /*
       * Code Goes Here
       */
    }
  catch(exception& e)
    {
      Sundance::handleException(e);
    }
  Sundance::finalize();
}
```

These lines control initialization and result gathering for the problem, initializing and finalizing MPI if MPI is being used, and other administrative tasks.

6.2.2.2 Getting the Mesh

A mesh object is used by Sundance to represent a tessellation of the problem domain. There are many ways of getting a mesh, all abstracted with the MeshSource interface. Sundance is designed to work with different mesh underlying implementations, the choice of which is done by specifying a MeshType object. In this example, we use the BasicSimplicialMeshType which is a lightweight parallel simplicial mesh. For the simple geometry associated with this problem, the discretization is done internally by Sundance as follows,

```
MeshType meshType = new BasicSimplicialMeshType();
MeshSource mesher = new
           PartitionedLineMesher(0.0, 1.0, 10, meshType);
Mesh mesh = mesher.getMesh();
```

In this example, it is assumed that the total bar length is 1.0 discretized into 10 elements.

6.2.2.3 Defining the Domains of Integration

Having defined the mesh, it is time to define the domains on which the mesh equations and boundary conditions are to be applied. CellFilter objects are used to represent domains and subdomains within a mesh. For this problem, we only need to define the interior and the edges where the boundary conditions are to be applied. For the latter purpose, the CellPredicate object is used. This object is a binary command used to test whether a cell is in the boundary domain or not.

```
CELL_PREDICATE(LeftPointTest,{return fabs(x[0])< 1e-8;});
CELL_PREDICATE(RightPointTest,{return fabs(x[0]-1.0)< 1e-8;});
```

This identifies the extremities of the domain located at $x = 0$ and $x = 1.0$. Now, the CellFilter object is used to define the separate domains:

```
CellFilter interior = new MaximalCellFilter();
CellFilter points = new DimensionalCellFilter(0);
CellFilter leftPoint = points.subset(new LeftPointTest());
CellFilter rightPoint = points.subset(new RightPointTest());
```

6.2.2.4 Defining the Spectral Basis and the Spatial Interpolation Basis

The spectral basis is defined by specifying its type, the number of dimensions and the order of the expansion used to generate it. For the purpose of implementing the SSFEM, the Hermite polynomials basis is used and it is generated as follows,

```
int ndim = 1;
int order = 6;
SpectralBasis sbasis = new HermiteSpectralBasis(ndim, order);
```

Similarly, a spatial interpolation basis is defined by specifying its type and order. Given these bases, we can now construct both the physical dicsretespace object and the random one.

```
DiscreteSpace discSpace(mesh, new Lagrange(1), vecType);
```

The latter specifies that the problem utilizes a first order lagrangian interpolation on the mesh.

6.2.2.5 Defining the Model Parameters and Excitation

The modulus of elasticity is defined as a lognormal process. Using the standard PCE expansion for a lognormal variable, the different coefficients of the one-dimensional sixth order expansion are obtained and stored in data files. Now, we have to read these files into Sundance vectors and define the associated Spectral Expressions. Sundance has built in function to read data files into Discrete Functions. This is done as follows,

```
Expr E0 = new DiscreteFunction(discSpace, 0.0, "E0");
Vector<double> vec = DiscreteFunction::
                     discFunc(E0)->getVector();
const RefCountPtr<DOFMapBase>& dofMap =
                 DiscreteFunction::discFunc(E0)->map();
ifstream is("E0.dat");
int nNodes;
is >> nNodes;
TEST_FOR_EXCEPTION(mesh.numCells(0) != nNodes, RuntimeError
"number of nodes in data file fieldData.dat is "
<< nNodes << " but number of nodes in mesh is "
<< mesh.numCells(0));
Array<int> dofs(1);
double fVal;
for (int i=0; i<nNodes; i++)
{
  dofMap->getDOFsForCell(0, i, 0, dofs);
  int dof = dofs[0];
  is >> fVal;
  vec.setElement(dof, fVal);
 }
DiscreteFunction::discFunc(E0)->setVector(vec);
```

In a similar fashion, the remaining PCE coefficients of E are read into Sundance. The excitation force is deterministic and hence its spectral representation is represented by its value as the mean of the expansion and zero higher order terms.

```
/* array of coefficients for the spectra expression */
Array<Expr> Coeff(sbasis.nterms());
```

```
Coeff[0] = 1.0;
Coeff[1] = 0.0;
Coeff[2] = 0.0;
Coeff[3] = 0.0;
Coeff[4] = 0.0;
Coeff[5] = 0.0;
Coeff[6] = 0.0;

Expr F = new SpectralExpr(sbasis, Coeff);
Expr  A = 1.0;

Expr E = new SpectralExpr(sbasis,
                  List(E0, E1, E2, E3, E4, E5, E6));
```

6.2.2.6 Defining the Unknown and Test Functions

Expressions representing the test and unknown functions are defined easily:

```
Expr u = new UnknownFunction(new Lagrange(1), sbasis, "u");
Expr v = new TestFunction(new Lagrange(1), sbasis, "v");
```

6.2.2.7 Writing the Weak form

To write the weak form, we need to define the Quadrature rule for computing the integration first. We'll use second-order Gaussian quadrature. The weak form with a quadrature specification is written in Sundance as:

```
QuadratureFamily quad2 = new GaussianQuadrature(2);
Expr eqn = Integral(interior, (dx*v)*(E*(dx*u)), quad2)
+ Integral(interior, v*F, quad2) ;
```

6.2.2.8 Writing the Essential Boundary Conditions

The weak form above contains the physics in the body of the domain plus the Neumann BCs on the edges. We still need to apply the Dirichlet boundary condition on the inlet, which we do with an EssentialBC object

```
Expr bc = EssentialBC(rightPoint, v*u , quad2)
+ EssentialBC(leftPoint, v*u, quad2);
```

6.2.2.9 Creating the Linear Problem Object

A LinearProblem object contains everything that is needed to assemble a discrete approximation to our PDE: a mesh, a weak form, boundary conditions, specification of test and unknown functions, and a specification of the low-level matrix and vector representation to be used. We will use Epetra as our linear algebra representation, which is specified by selecting the corresponding VectorType subtype,

```
VectorType<double> vecType = new EpetraVectorType();
```

Now the Linear problem object is defined as,

```
LinearProblem prob(mesh, eqn, bc, v, u, vecType);
```

6.2.2.10 Specifying the Solver and Solving the Problem

The BICGSTAB method with ILU preconditioning is used for solving the problem. Level one preconditioning alond with a tolerance of 10^{-14} within 1000 iterations is used. The whole setup is configured using the Parameterlist TRILINOS object to read the specs from an xml file.

```
ParameterXMLFileReader reader("bicgstab.xml");
ParameterList solverParams = reader.getParameters();
```

```
LinearSolver<double> linSolver
   = LinearSolverBuilder::createSolver(solverParams);
```

Then the problem is simply solved by using the following command,

```
Expr soln = prob.solve(linSolver);
```

6.2.3 Complete Code for the Bar Problem

```
#include "Sundance.hpp"

CELL_PREDICATE(LeftPointTest, {return fabs(x[0]) < 1.0e-10;});
CELL_PREDICATE(RightPointTest, {return fabs(x[0]-1.0) < 1.0e-10;});

int main(int argc, char** argv)
{
  try
    {
      Sundance::init(&argc, &argv);

      VectorType<double> vecType = new EpetraVectorType();

      /* Create a mesh. It will be of type BasisSimplicialMesh, and will
       * be built using a PartitionedLinedMesher. */
      MeshType meshType = new BasicSimplicialMeshType();
      MeshSource mesher = new PartitionedLineMesher(0.0, 1.0, 10, meshType);
      Mesh mesh = mesher.getMesh();

      /* Create a cell filter that will identify the maximal cells
       * in the interior of the domain */
      CellFilter interior = new MaximalCellFilter();
      CellFilter points = new DimensionalCellFilter(0);
      CellFilter leftPoint = points.subset(new LeftPointTest());
      CellFilter rightPoint = points.subset(new RightPointTest());



      /* Create the Spectral Basis */
      int ndim = 1;
```

```
 int order = 6;

 SpectralBasis sbasis = new HermiteSpectralBasis(ndim, order);


/* create an empty (zero-valued) discrete function */
 DiscreteSpace discSpace(mesh, new Lagrange(1), vecType);


 Expr E0 = new DiscreteFunction(discSpace, 0.0, "E0");
 Vector<double> vec = DiscreteFunction::discFunc(E0)->getVector();
 const RefCountPtr<DOFMapBase>& dofMap =
  DiscreteFunction::discFunc(E0)->map();
 ifstream is("E0.dat");
 int nNodes;
 is >> nNodes;
 TEST_FOR_EXCEPTION(mesh.numCells(0) != nNodes, RuntimeError,
    "number of nodes in data file fieldData.dat is "
    << nNodes << " but number of nodes in mesh is "
    << mesh.numCells(0));

 Array<int> dofs(1);
 double fVal;
 for (int i=0; i<nNodes; i++)
 {
  dofMap->getDOFsForCell(0, i, 0, dofs);
  int dof = dofs[0];
  is >> fVal;
  vec.setElement(dof, fVal);
 }
 DiscreteFunction::discFunc(E0)->setVector(vec);

 /* create an empty (zero-valued) discrete function */
 Expr E1 = new DiscreteFunction(discSpace, 0.0, "E1");

 Vector<double> vec1 = DiscreteFunction::discFunc(E1)->getVector();
 const RefCountPtr<DOFMapBase>& dofMap1 =
  DiscreteFunction::discFunc(E1)->map();

 ifstream is1("E1.dat");
```

```
is1 >> nNodes;

TEST_FOR_EXCEPTION(mesh.numCells(0) != nNodes, RuntimeError,
   "number of nodes in data file fieldData.dat is "
   << nNodes << " but number of nodes in mesh is "
   << mesh.numCells(0));


for (int i=0; i<nNodes; i++)
{
 dofMap1->getDOFsForCell(0, i, 0, dofs);
 int dof = dofs[0];
 is1 >> fVal;
 vec1.setElement(dof, fVal);
}
DiscreteFunction::discFunc(E1)->setVector(vec1);

/* create an empty (zero-valued) discrete function */
Expr E2 = new DiscreteFunction(discSpace, 0.0, "E2");

Vector<double> vec2 = DiscreteFunction::discFunc(E2)->getVector();
const RefCountPtr<DOFMapBase>& dofMap2 =
 DiscreteFunction::discFunc(E2)->map();

ifstream is2("E2.dat");
is2 >> nNodes;

TEST_FOR_EXCEPTION(mesh.numCells(0) != nNodes, RuntimeError,
    "number of nodes in data file fieldData.dat is "
    << nNodes << " but number of nodes in mesh is "
   << mesh.numCells(0));


for (int i=0; i<nNodes; i++)
{
 dofMap2->getDOFsForCell(0, i, 0, dofs);
 int dof = dofs[0];
 is2 >> fVal;
 vec2.setElement(dof, fVal);
}
```

```
DiscreteFunction::discFunc(E2)->setVector(vec2);


/* create an empty (zero-valued) discrete function */
Expr E3 = new DiscreteFunction(discSpace, 0.0, "E3");

Vector<double> vec3 = DiscreteFunction::discFunc(E3)->getVector();
const RefCountPtr<DOFMapBase>& dofMap3 =
 DiscreteFunction::discFunc(E3)->map();

ifstream is3("E3.dat");
is3 >> nNodes;

TEST_FOR_EXCEPTION(mesh.numCells(0) != nNodes, RuntimeError,
   "number of nodes in data file fieldData.dat is "
   << nNodes << " but number of nodes in mesh is "
   << mesh.numCells(0));


for (int i=0; i<nNodes; i++)
 {
 dofMap3->getDOFsForCell(0, i, 0, dofs);
   int dof = dofs[0];
 is3 >> fVal;
 vec3.setElement(dof, fVal);
}
DiscreteFunction::discFunc(E3)->setVector(vec3);

/* create an empty (zero-valued) discrete function */
Expr E4 = new DiscreteFunction(discSpace, 0.0, "E4");

Vector<double> vec4 = DiscreteFunction::discFunc(E4)->getVector();
const RefCountPtr<DOFMapBase>& dofMap4 =
 DiscreteFunction::discFunc(E4)->map();

ifstream is4("E4.dat");
is4 >> nNodes;

TEST_FOR_EXCEPTION(mesh.numCells(0) != nNodes, RuntimeError,
   "number of nodes in data file fieldData.dat is "
```

```
   << nNodes << " but number of nodes in mesh is "
   << mesh.numCells(0));

for (int i=0; i<nNodes; i++)
{
 dofMap4->getDOFsForCell(0, i, 0, dofs);
 int dof = dofs[0];
 is4 >> fVal;
 vec4.setElement(dof, fVal);
}
DiscreteFunction::discFunc(E4)->setVector(vec4);


/* create an empty (zero-valued) discrete function */
Expr E5 = new DiscreteFunction(discSpace, 0.0, "E5");

Vector<double> vec5 = DiscreteFunction::discFunc(E5)->getVector();
const RefCountPtr<DOFMapBase>& dofMap5 =
 DiscreteFunction::discFunc(E5)->map();

ifstream is5("E5.dat");
is5 >> nNodes;

TEST_FOR_EXCEPTION(mesh.numCells(0) != nNodes, RuntimeError,
    "number of nodes in data file fieldData.dat is "
   << nNodes << " but number of nodes in mesh is "
   << mesh.numCells(0));


for (int i=0; i<nNodes; i++)
{
 dofMap5->getDOFsForCell(0, i, 0, dofs);
 int dof = dofs[0];
 is5 >> fVal;
 vec5.setElement(dof, fVal);
}
DiscreteFunction::discFunc(E5)->setVector(vec5);


/* create an empty (zero-valued) discrete function */
```

```
Expr E6 = new DiscreteFunction(discSpace, 0.0, "E6");
Vector<double> vec6 = DiscreteFunction::discFunc(E6)->getVector();
const RefCountPtr<DOFMapBase>& dofMap6 =
 DiscreteFunction::discFunc(E6)->map();

ifstream is6("E6.dat");
is6 >> nNodes;

TEST_FOR_EXCEPTION(mesh.numCells(0) != nNodes, RuntimeError,
   "number of nodes in data file fieldData.dat is "
   << nNodes << " but number of nodes in mesh is "
   << mesh.numCells(0));

for (int i=0; i<nNodes; i++)
{
 dofMap6 ->getDOFsForCell(0, i, 0, dofs);
 int dof = dofs[0];
 is6 >> fVal;
 vec6.setElement(dof, fVal);
}
DiscreteFunction::discFunc(E6)->setVector(vec6);

/* Create Spectral Unknown And test Functions */
Expr u = new UnknownFunction(new Lagrange(1), sbasis, "u");
Expr v = new TestFunction(new Lagrange(1), sbasis, "v");

/* Create differential operator and coordinate functions */
Expr x = new CoordExpr(0);
Expr dx = new Derivative(0);

/* We need a quadrature rule for doing the integrations */
QuadratureFamily quad2 = new GaussianQuadrature(2);

/* array of coefficients for the spectra expression */
Array<Expr> Coeff(sbasis.nterms());
Coeff[0] = 1.0;
Coeff[1] = 0.0;
Coeff[2] = 0.0;
Coeff[3] = 0.0;
Coeff[4] = 0.0;
```

```
Coeff[5] = 0.0;
Coeff[6] = 0.0;

Expr F = new SpectralExpr(sbasis, Coeff);
Expr  A = 1.0;

Expr E = new SpectralExpr(sbasis, List(E0, E1, E2, E3, E4, E5, E6));

/* Define the weak form */

Expr eqn = Integral(interior, (dx*v)*(E*(dx*u)), quad2) +
  Integral(interior, v*F, quad2) ;

/* Define the Dirichlet BC */
Expr bc = EssentialBC(rightPoint, v*u , quad2)+
          EssentialBC(leftPoint, v*u, quad2);

/* We can now set up the linear problem! */
LinearProblem prob(mesh, eqn, bc, v, u, vecType);

/* Read the parameters for the linear solver from an XML file */

ParameterXMLFileReader reader("bicgstab.xml");
ParameterList solverParams = reader.getParameters();
LinearSolver<double> linSolver
  = LinearSolverBuilder::createSolver(solverParams);

/* solve the problem */

Expr soln = prob.solve(linSolver);

FieldWriter wr = new MatlabWriter("Spectralbar");
wr.addMesh(mesh);
wr.addField("u0", new ExprFieldWrapper(soln[0]));
wr.addField("u1", new ExprFieldWrapper(soln[1]));
wr.addField("u2", new ExprFieldWrapper(soln[2]));
wr.addField("u3", new ExprFieldWrapper(soln[3]));
wr.addField("u4", new ExprFieldWrapper(soln[4]));
wr.addField("u5", new ExprFieldWrapper(soln[6]));
wr.write();
```

```
    }
  catch(exception& e)
    {
      Sundance::handleException(e);
    }
  Sundance::finalize();
}
```

Chapter 7

Conclusion

The combination of Polynomial Chaos with the Ensemble Kalman Filter renders an efficient data assimilation methodology that surpasses standard Kalman Filtering techniques while maintaining a relatively low computational cost. Although the proposed method employs traditional Kalman Filter updating schemes, it preserves all the error statistics, and hence allows the computation of the probability density function of the uncertain parameters and variables at all time steps. This is done by simply simulating the PC representations of these parameters. This simulation has a negligible computational cost.

Applying the proposed method for calibrating reservoir simulation models is an innovative approach that allows approximating the higher order statistics of the production forecast. The use of PC to represent the uncertainty in the reservoir model and the measurement help convey the actual medium in a more realistic way. The proposed method is tested on the one-dimensional Buckley-Leverett and the two-dimensional Five Spot two-phase immiscible flow problems using synthesized measurement data. A novel approach for representing the reservoir medium properties as a rational function is presented, and the obtained results depict the validity and effectiveness of the proposed method.

The efficiency of the method is also conveyed through applying it for optimizing the fluid front dynamics in porous media by using injection rate control. The obtained results show that representing the controls as part of the Kalman Filter state vector is an effective approach for controlling the flow and maintaining a desired target.

The data assimilation technique is also applied for health monitoring of highly nonlinear structures. Together with the non-parametric representation of the nonlinearities, the approach constitutes an effective system identification technique that accurately detects any changes in the systems behavior. The Polynomial Chaos representation of the non-parametric model for the nonlinearities is a robust innovative approach that permits damage identification and tracking the dynamical state beyond that point. Using Polynomial Chaos, the uncertainty associated with the assumed non-parametric model is inherently present and thus represents the actual nonlinearity in a more accurate way.

References

[Aan05] S.I. Aanonsen. Efficient history matching using a multiscale technique. *SPE reservoir simulation symposium*, (SPE 92758), 2005.

[AML05] M. Alvarado, D. McVay, and W. Lee. Quantification of uncertainty by combining forecasting with history matching. *Petroleum Science and Technology*, 23:445–462, 2005.

[Ash88] H. Asheim. Maximization of water sweep efficiency by controlling injection and production rates. *SPE*, (18365), 1988.

[BBM87] J. Bear, C. Brestar, and P.C. Menier. Effective and relative permeabilities of anistropic porous media. *Transport in porous media*, 2:301–316, 1987.

[BC64] R.H. Brooks and T. Corey. Hydraulic properties of porous media. Technical Report 3, Civ. Eng. Dept, Colorado State Univ., Fort Collins, Hydrol. paper 1964.

[Bea72] J. Bear. *Dynamics of Fluids in Porous Media*. Dover Publications, New York, 1972.

[BLE98] G. Burgers, P.J. Van Leeuwen, and G. Evensen. Analysis scheme in the ensemble kalman filter. *Monthly Weather Review*, 126(6):1719–1724, 1998.

[Bro83] R.G. Brown. *Introduction to Random Signal Analysis and Kalman Filtering*. John Wiley and Sons Inc., USA, 1983.

[Ca74] W.H. Chen and al. A new algorithm for automatic history matching. *SPE Journal*, 14(6):593–608, 1974.

[CC91] C. Chui and G. Chen. *Kalman Filtering with real time applications*. Springer-Verlag, USA, 2nd edition, 1991.

[CDL75] G. Chavent, M. Dupuy, and P. Lemonnier. History matching by use of optimal control theory. *SPE Journal*, 15(1):74–86, 1975.

[CJ86] G. Chavent and J. Jaffre. *Mathematical Models and Finite Elements for Reservoir Simulation*. Studies in mathematics and its applications. Elsevier Science Publishers B.V., 1986.

[CM47] R. Cameron and W. Martin. The orthogonal development of nonlinear functionals in series of fourier-hermite functionals. *Ann. MAth*, 48:385–392, 1947.

[DdFG01] A Doucet, N de Freitas, and N Gordon. *Sequential Monte Carlo Methods in Practice,*. Springer, 2001.

[DGRH] A. Doostan, R. Ghanem, and J. Red-Horse. Stochastic model reductions for chaos represenations. *CMAME*. submitted.

[DP89] M. Delshad and G.A. Pope. Comparison of the three-phase oil relative permeability models. *Transport in Porous Media*, 4(1):59–83, 1989.

[Dul91] F.A.L. Dullien. *Porous media: Fluid Transport and Pore Structure*. Academic Press, San Diego, California, 1991.

[Eve94] G. Evensen. Sequential data assimilation with a nonlinear quasi-geostrophic model using monte carlo methods to forecast error statistics. *Journal of Geophysical Research*, 99:10143–10162, 1994.

[Eve03] G. Evensen. The ensemble kalman filter: Theoretical formulation and practical implementation. *Ocean Dynamics*, 53:343–367, 2003.

[Eve05] G. Evensen. The combined parameter and state estimation problem. *Computational Geosciences*, 2005. submitted.

[FBL05] G. Franco, R. Betti, and S. Lus. Identification of structural systems using an evolutionary strategy. *Journal of Engineering Mechanics*, 130:1125–1139, 2005.

[FIIN05] T. Furukawa, M. Ito, K. Izawa, and M. Noori. System identification of base isolated building using seismic response data. *Journal of Engineering Mechanics*, 131:268–275, 2005.

[FM84] F.J. Fayers and J.P. Mathews. Evaluation of normalized stone's methods for estimating three phase relative permeabilities. *SPE Journal*, 24:224–232, 1984.

[FR86] Z. Fathi and W.F. Ramirez. Use of optimal control theory for computing optimal injection policies for enhanced oil recovery. *Automatica*, 22(1):333, 1986.

[FR87] Z. Fathi and W.F. Ramirez. Optimization of an enhanced oil recovery process with boundary controls: a large scale nonlinear maximization. *Automatica*, 23(3):301, 1987.

[GF06] R. Ghanem and G. Ferro. Health monitoring for strongly non-linear systems using the ensemble kalman filter. *Structural Control and Health Monitoring*, 13:245–259, 2006.

[Gha99] R. Ghanem. The nonlinear gaussian spectrum of log-normal stochastic processes and variables. *Journal of Appl Mech-T ASME*, 66(4):964–973, 1999.

[GMNU03] A. Grimstad, T. Mannseth, G. Naevdal, and H. Urkeda. Adaptive multiscale permeabilty estimmation. *Computational Geosciences*, 7:1–25, 2003.

[GO05] Y. Gu and D.S. Oliver. History matching of the punq-s3 reservoir model using the ensemble kalman filter. *SPE Journal*, 10(2):217–224, 2005.

[GR05] G. Gao and A. Reynolds. Quantifying the uncertainty for the punq-s3 problem in a bayesian setting with rml and enkf. *SPE reservoir simulation symposium*, (SPE 93324), 2005.

[GS95] R. Ghanem and M. Shinozuka. Structural systems identification i: Theory. *Journal of Engineering Mechanics*, 121:255–264, 1995.

[GS03] R. Ghanem and P. Spanos. *Stochastic Finite Elements: A Spectral Approach*. Dover Publications, Inc., revised ed. edition, 2003.

[GSD07] Roger Ghanem, George Saad, and Alireza Doostan. Efficient solution of stochastic systems: application to the embankment dam problem. *Structural Safety*, 29:238–251, 2007.

[Hat98] S. Hatoum. *A Computational Model for the Analysis of Multiphase flow in Random Porous Media*. PhD thesis, University of New York at Buffalo, 1998.

[ICO04] ICOSSAR05. Benchmark study on reliability estimation in higher dimensions of structural systems. 2004. Communication 1.

[JU97] Simon J. Julier and Jeffrey K. Uhlmann. A new extension of the kalman filter to nonlinear systems. In *AeroSense: The 11th International Symposium on Aerospace/Defense Sensing, Simulation and Controls, Multi Sensor Fusion, Tracking and Resource Management II*. SPIE, 1997.

[Kal60] R.E. Kalman. A new approach to linear filtering and perdiction problems. *ASME Journal of Basic Engrg.*, 82D:35–45, 1960.

[KGV83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[Kiv03] G.A. Kivman. Sequential parameter estimation for stochastic systems. *Nonlinear Processes in Geophysics*, 10:253–259, 2003.

[KP87] J.B. Kool and J.C. Parker. Development and evaluation of closed-form expressions for hysteretic soil hydraulic properties. *Water Resource Research*, 23(1):105–114, 1987.

[LBL02] H. Lus, R. Betti, and R. Longman. Obtaining refined-first order predictive models of linear structural systems. *Earthquake Engineering and Structural Dynamics*, 31:1413–1440, 2002.

[LGN89] L. Luckner, M.T. Van Genuchten, and D.R. Nielsen. A consistent set of parametric models for the two phase flow of the immiscible fluids in the subsurface. *Water Resources Research*, 25(10):2187–2193, 1989.

[LO05] N. Liu and D.S. Oliver. Critical evaluation of the ensemble kalman filter on history matching of geologic facies. *SPE Reservoir Evaluation and Engineering*, 8(6):470–477, 2005.

[Loe77] M. Loeve. *Probability Theory*. Springer, New York, 4th edition, 1977.

[Lon04] K. Long. Sundance 2.0 tutorial. Technical report, Sandia National Lab, 2004.

[LP87] R.J. Lenhard and J.C. Parker. A model for hysteretic constitutive relations governing multiphase flow .2. permeability-saturation relations. *Water Resources Research*, 23(12):2197–2206, 1987.

[Ma93] E.M. Makhlouf and al. A general history matching algorithm for three phase, three dimensional petroleum reservoir. *SPE Advanced Technology Series*, 1(2):83–91, 1993.

[MCC+04] S.F. Masri, J.P. Caffrey, T.K. Caughey, A.W. Smyth, and A.G. Chassiakos. Identification of the state equation in complex non-linear systems. *International Journal of Non-Linear Mechanics*, 39:1111–1127, 2004.

[MG87] A. Montoglou and L.W. Gelhar. Effective hydraulic conductivities of transient unsaturated flow in stratified soils. *Water Resources Research*, 23:57–67, 1987.

[MK04] H. Matthies and A. Keese. Galerkin methods for linear and nonlinear elliptic stochastic partial differential equations. *Comput. Methods Appl. Mech. Engng.*, 2004. In Press.

[Mus49] M. Muskat. *Physical Principles of Oil Production*. Mc Graw Hill, New York, 1949.

[NBJ06] Geir Naevdal, D. Roald Brouwer, and Jan-Dirk Jansen. Waterflooding using closed loop control. *Computational Geosciences*, 10:37–60, 2006.

[NJAV03] G. Naevdal, L.M. Johnsen, S.I. Aanonsen, and E. Vefring. Reservoir monitoring and continuous model updating using the ensemble kalman filter. *SPE Annual Technical Conference and Exhibition*, (SPE 84372), 2003.

[NMV02] G. Naevdal, T. Mannseth, and E. Vefring. Near well reservoir monitoring through ensemble kalman filter. *Proceeding of SPE/DOE Improved oil recovery Symposium*, (SPE 84372), 2002.

[Ode79] J.T. Oden. *Applied Functional Analysis*. Prentice Hall, New Jersey, 1979.

[PL87] J.C. Parker and R.J. Lenhard. A model for hysteretic constitutive relations governing multiphase flow. 1. saturation-pressure relations. *Water Resources Research*, 23(12):2187–2196, 1987.

[RH98] F. Roggero and L. Hu. Gradual deformation of continuous geostatical models for history matching. *SPE Annual Technical Conference and Exhibition*, (SPE 49004), 1998.

[RL00] A. Robinson and P. Lermusiaux. Overview of data assimilation. Harvard reports in physical/interdisciplinary ocean science, 2000.

[Sch01a] Lothar M Schmitt. *Theory of Genetic Algorithms*, 259:1–61, 2001.

[Sch01b] G. Schueller. Computational stochastic mechanics - recent advances. *Computers and Structures*, 79(22-25):2225–2234, 2001.

[Sto70] H.L. Stone. Probability model for estimationg three phase relative permeability. *Journal of Petroleum Technology*, 20:214–218, 1970.

[SY00] Bagus Sudaryanto and Yannis Yortsos. Optimization of fluid front dynamics in porous media using rate control. 1. equal mobility fluids. *Physics of Fluids*, 12(7):1656–1670, 2000.

[WC05] X. Wen and W. Chen. Real time reservoir model updationg using the ensemble kalman filter. *SPE reservoir simulation symposium*, (SPE 92991), 2005.

[Wie38] N. Wiener. The homogeneous chaos. *Am. J. Math.*, 60:897–936, 1938.

[ZFYM02] H. Zhang, G. Foliente, Y. Yang, and F. Ma. Parameter identification of elastic structures under dynamic loads. *Earthquake Engineering and Structural Dynamics*, 31:1113–1130, 2002.

Appendices

Appendix A

Complete SUNDANCE Reservoir Characterization Codes

A.1 Two-Dimensional Water Flooding Problem - Scenario 1

```
#include "Sundance.hpp"

int main(int argc, char** argv)
{
  try
    {
      MPISession::init(&argc, &argv);

      /* We will do our linear algebra using Epetra */
      VectorType<double> vecType = new EpetraVectorType();

      MeshType meshType = new BasicSimplicialMeshType();
      MeshSource mesher = new ExodusNetCDFMeshReader("square_60x60.ncdf", meshType);
      Mesh mesh = mesher.getMesh();

      /* Create a cell filter that will identify the maximal cells
        in the interior of the domain */

      CellFilter interior = new MaximalCellFilter();
      CellFilter edges = new DimensionalCellFilter(1);
      CellFilter Source = edges.labeledSubset(1);
      CellFilter Sink = edges.labeledSubset(2);

      /* Create the Spectral Basis */
      int ndim = 3;
      int order = 2;
      int nterm = 5;

      /* Hermite Basis truncated after 5 terms */
      SpectralBasis sbasis = new HermiteSpectralBasis(ndim, order, nterm);


      /* Create unknown and test functions, discretized using first-order
```

```
 * Lagrange interpolants */

BasisFamily L1 = new Lagrange(2);
Expr Se = new UnknownFunction(L1, sbasis, "Se");
Expr Ns = new TestFunction(L1, sbasis, "Ns");
Expr Pw = new UnknownFunction(L1, sbasis, "Pw");
Expr Nn = new TestFunction(L1, sbasis, "Nn");

/* Create differential operator and coordinate functions */
Expr x = new CoordExpr(0);
Expr y = new CoordExpr(1);
Expr dx = new Derivative(0);
Expr dy = new Derivative(1);
Expr grad = List(dx, dy);

/* We need a quadrature rule for doing the integrations */
QuadratureFamily quad2 = new GaussianQuadrature(4);

DiscreteSpace d1(mesh, L1, vecType);
DiscreteSpace discSpace(mesh, List(L1, L1) , sbasis, vecType);

/* Initial Guess for the dynamic states */
Expr u0 = new DiscreteFunction(discSpace, 0.0, "u0");

/* Initial Condition for the Saturation */
Expr Se00 = new DiscreteFunction(d1, 0.0, "Se00");
Expr Se01 = new DiscreteFunction(d1, 0.0, "Se01");
Expr Se02 = new DiscreteFunction(d1, 0.0, "Se02");
Expr Se03 = new DiscreteFunction(d1, 0.0, "Se03");
Expr Se04 = new DiscreteFunction(d1, 0.0, "Se04");

Expr Se0 = new SpectralExpr(sbasis, List(Se00, Se01, Se02, Se03, Se04));

/* Medium Porosity */

Expr P0 = new DiscreteFunction(d1, 0.20, "P0");
Expr P1 = new DiscreteFunction(d1, 0.0, "P1");
Expr P2 = new DiscreteFunction(d1, 0.0, "P2");
Expr P3 = new DiscreteFunction(d1, 0.0, "P3");
Expr P4 = new DiscreteFunction(d1, 0.0, "P4");

/* Medium Permeability */

Expr K0 = new DiscreteFunction(d1, 170.0, "K0");
Expr K1 = new DiscreteFunction(d1, 0.0, "K1");
Expr K2 = new DiscreteFunction(d1, 0.0, "K2");
Expr K3 = new DiscreteFunction(d1, 0.0, "K3");
Expr K4 = new DiscreteFunction(d1, 0.0, "K4");

Vector<double> K1Vec  = DiscreteFunction::discFunc(K1)->getVector();
Vector<double> P1Vec  = DiscreteFunction::discFunc(P1)->getVector();
Vector<double> K4Vec  = DiscreteFunction::discFunc(K4)->getVector();
Vector<double> P4Vec  = DiscreteFunction::discFunc(P4)->getVector();
```

```
int nNodes1 = mesh.numCells(0); /*number of nodes in mesh */
int nElems = mesh.numCells(1); /* number of elements */
int vecSize = nElems + nNodes1;

/* crossectional Area */
double As = 1.0;

/* initial guess for the scatter of the unknown medium properties */
double x1, x2, x3, x4;
int seed = 1000;

for(int MC=0; MC<vecSize; MC++)
  {
    seed += 1;
    StochasticLib1 sto(seed);
    if ((MC%1) == 0)
      {
        x1 = sto.Normal(0,10);
        x2 = sto.Normal(0,0.01);
      x3 = sto.Normal(0,5);
      x4 = sto.Normal(0,0.005);

        K1Vec.setElement(MC, (x1));
        P1Vec.setElement(MC, (x2));
        K4Vec.setElement(MC, (x3));
        P4Vec.setElement(MC, (x4));
      }
  }

DiscreteFunction::discFunc(K1)->setVector(K1Vec);
DiscreteFunction::discFunc(P1)->setVector(P1Vec);
DiscreteFunction::discFunc(K4)->setVector(K4Vec);
DiscreteFunction::discFunc(P4)->setVector(P4Vec);

Expr P = new SpectralExpr(sbasis, List(P0, P1, P2, P3, P4));
Expr K = new SpectralExpr(sbasis, List(K0, K1, K2, K3, K4));


double MuO = 1.735e06; /* Oil Viscosity */
double MuW = 1.735e05; /* Water Viscosity */
double Pe = 10000.0;
double INJ =  0.0323; /* Water Injection rate */

/* Setup the time stepping with timestep = 0.1 */
double deltaT = 0.10;

double Sm = 0.35; /*residual saturation */

/* The coefficients of the different powers of the chaos coefficients */

Expr Squad = pow(Se,4);
Expr Scube = pow(Se,3);
Expr Ssquare = pow(Se,2);
```

```
  /* Computing the relative pemeabilities and capillary pressure */
  Expr Krw = 0.864*Squad + 0.201*Scube - 0.065*Ssquare ;
  Krw.setcoeff(3, 0.05*Krw.getcoeff[1]); /* modeling error */
  Expr Kro = 1.595*Squad - 4.784*Scube  + 5.994*Ssquare - 3.805*Se + 1.0;
  Kro.setcoeff(3, 0.05*Kro.getcoeff[1]); /* modeling error */
  Expr Pc = Pe*(-4.286*Scube + 8.974*Ssquare - 6.993*Se + 3.152);
  Pc.setcoeff(3, 0.05*Pc.getcoeff[1]); /* modeling error */


  /* Weak Form of coupled PDE */
  Expr Weqn = Integral(interior, Ns*(As*(1.0-Sm)*(P*(Se - Se0))), quad)
  + Integral(interior, (dx*Ns)*(As/MuW*(Krw*K*(dx*Pw))*deltaT), quad)
  - Integral(Source, Ns*INJ*deltaT,quad);

  Expr Oeqn = Integral(interior,-Nn*(As*(1.0-Sm)*(P*(Se - Se0))), quad)
+ Integral(interior, (dx*Nn)*(As/MuO*(Kro*K*(dx*(Pw + Pc)))*deltaT), quad);


  Expr eqn =  Weqn + Oeqn;

  /* Boundary Conditions */
  Expr bc = EssentialBC(leftPoint, Nn*(Pw-10000),quad);

  /*setting up the nonlinear problem */
  NonlinearOperator<double> F =
   new NonlinearProblem(mesh, eqn, bc,List(Ns,Nn), List(Se,Pw), u0, vecType);

  ParameterXMLFileReader reader("nox.xml");
  ParameterList noxParams = reader.getParameters();

  NOXSolver solver(noxParams, F);

  int Nn = nNodes1;
  /* array of the global ID's of the measurement locations */
  int GID[22] = {1, 4, 5, 8, 11, 12, 14, 15, 18,
          19, 22, 25, 27, 30, 31, 32, 34, 37, 40, 41, 42, 47};
  int LID[Nn];

  // get local ID of node given GID
  const RefCountPtr<DOFMapBase>& dofMap = DiscreteFunction::discFunc(Se00)->map();
  for (int i=0; i< Nn; i++)
    {
      int ID = mesh.mapGIDToLID(0, i);
      Array<int> dofIndices;
      dofMap -> getDOFsForCell(0,ID,0, dofIndices);

      LID[i] = dofIndices[0];
    }


  /* Open the data file */
  FILE * mFile;
  mFile = fopen("../true/SMeasurement.dat", "r");
```

```
    int nSteps = 20001;
    for(int ns=0; ns < nSteps; ns++)
     {
      solver.solve();

       /* Reading the solution into vectors so that it could be transfered
         to the Newmat Library for the assimilation step */

  Expr Pww0 = new DiscreteFunction(d1, 0.0, "Pww0");
   Expr Pww1 = new DiscreteFunction(d1, 0.0, "Pww1");
   Expr Pww2 = new DiscreteFunction(d1, 0.0, "Pww2");
   Expr Pww3 = new DiscreteFunction(d1, 0.0, "Pww3");
   Expr Pww4 = new DiscreteFunction(d1, 0.0, "Pww4");

   Vector<double> SolnVec  = DiscreteFunction::discFunc(u0)->getVector();

   Vector<double> Se00V = DiscreteFunction::discFunc(Se00)->getVector();
   Vector<double> Se01V = DiscreteFunction::discFunc(Se01)->getVector();
   Vector<double> Se02V = DiscreteFunction::discFunc(Se02)->getVector();
   Vector<double> Se03V = DiscreteFunction::discFunc(Se03)->getVector();
   Vector<double> Se04V = DiscreteFunction::discFunc(Se04)->getVector();

   Vector<double> Pww0V = DiscreteFunction::discFunc(Pww0)->getVector();
   Vector<double> Pww1V = DiscreteFunction::discFunc(Pww1)->getVector();
   Vector<double> Pww2V = DiscreteFunction::discFunc(Pww2)->getVector();
   Vector<double> Pww3V = DiscreteFunction::discFunc(Pww3)->getVector();
   Vector<double> Pww4V = DiscreteFunction::discFunc(Pww4)->getVector();

   Vector<double> K0V = DiscreteFunction::discFunc(K0)->getVector();
   Vector<double> K1V = DiscreteFunction::discFunc(K1)->getVector();
   Vector<double> K2V = DiscreteFunction::discFunc(K2)->getVector();
   Vector<double> K3V = DiscreteFunction::discFunc(K3)->getVector();
   Vector<double> K4V = DiscreteFunction::discFunc(K4)->getVector();

   Vector<double> P0V = DiscreteFunction::discFunc(P0)->getVector();
   Vector<double> P1V = DiscreteFunction::discFunc(P1)->getVector();
   Vector<double> P2V = DiscreteFunction::discFunc(P2)->getVector();
   Vector<double> P3V = DiscreteFunction::discFunc(P3)->getVector();
   Vector<double> P4V = DiscreteFunction::discFunc(P4)->getVector();

          for (int n=0; n<vecSize; n++)
            {
   Se00V.setElement(n, SolnVec.getElement(10*n));
   Se01V.setElement(n, SolnVec.getElement(10*n+1));
   Se02V.setElement(n, SolnVec.getElement(10*n+2));
   Se03V.setElement(n, SolnVec.getElement(10*n+3));
   Se04V.setElement(n, SolnVec.getElement(10*n+4));
   Pww0V.setElement(n, SolnVec.getElement(10*n+5));
   Pww1V.setElement(n, SolnVec.getElement(10*n+6));
   Pww2V.setElement(n, SolnVec.getElement(10*n+7));
   Pww3V.setElement(n, SolnVec.getElement(10*n+8));
   Pww4V.setElement(n, SolnVec.getElement(10*n+9));
}
```

```
  /* Measurements exist every 10 timesteps */

   if ( ((ns% 10) == 0) && (ns > 0))
     {
      /* the coefficients of the state vector containing the dynamic states
 , Se and Pw, and the model parameters, K and P. */
       ColumnVector A0(4*Nn);
       ColumnVector A1(4*Nn);
       ColumnVector A2(4*Nn);
       ColumnVector A3(4*Nn);
       ColumnVector A4(4*Nn);

      /* populating the state vectors with the Sundance solution */

      for(int i=1; i<=4*Nn; i++)
{
  if(i<=Nn)
    {
       A0(i) = 0.64*Se00V.getElement(LID[i-1]) + 0.16;
       A1(i) = 0.64*Se01V.getElement(LID[i-1]);
       A2 (i) = 0.64*Se02V.getElement(LID[i-1]);
                  A3(i) = 0.64*Se03V.getElement(LID[i-1]);
       A4(i) = 0.64*Se04V.getElement(LID[i-1]);
    }
   else if ((i > Nn) && (i<=2*Nn))
     {
      A0(i) = Pww0V.getElement(LID[i-(Nn+1)]);
       A1(i) = Pww1V.getElement(LID[i-(Nn+1)]);
      A2(i) = Pww2V.getElement(LID[i-(Nn+1)]);
      A3(i) = Pww3V.getElement(LID[i-(Nn+1)]);
      A4(i) = Pww4V.getElement(LID[i-(Nn+1)]);
     }
  else if ((i > 2*Nn) && (i<=3*Nn))
     {
       A0(i) = K0V.getElement(LID[i-(2*Nn+1)]);
       A1(i) = K1V.getElement(LID[i-(2*Nn+1)]);
       A2(i) = K2V.getElement(LID[i-(2*Nn+1)]);
       A3(i) = K3V.getElement(LID[i-(2*Nn+1)]);
       A4(i) = K4V.getElement(LID[i-(2*Nn+1)]);
     }
  else if ( (i > 3*Nn) && (i <= 4*Nn))
     {
       A0(i) = P0V.getElement(LID[i-(3*Nn+1)]);
       A1(i) = P1V.getElement(LID[i-(3*Nn+1)]);
       A2(i) = P2V.getElement(LID[i-(3*Nn+1)]);
                  A3(i) = P3V.getElement(LID[i-(3*Nn+1)]);
       A4(i) = P4V.getElement(LID[i-(3*Nn+1)]);
     }
}

      // State Error Covariance Matrix
      Matrix P1m(4*Nn,4*Nn);
      P1m = 0;
```

```
      P1m = A1*A1.t()*Exp.expectation(0,1,1) + A2*A2.t()*Exp.expectation(0,2,2) +
            A3*A3.t()*Exp.expectation(0,3,3) + A4*A4.t()*Exp.expectation(0,4,4);

      // Measurement Matrix
      const int mp = 22;
      Matrix Z(mp,nterm);
      Z = 0.0;
      ColumnVector mt(mp);

        for(int i=1; i<=mp; i++)
         {
    fscanf(mFile, "%le", &mt(i));
}

      Z.column(1) << mt;

      ColumnVector er1(mp);

      /* Measurement Error */
       for(int i=1; i<=mp; i++)
{
   if (mt(i) > 0.20)
     {
      er1(i) = 0.001*(mt(i)-0.20);
     }
  else
     {
       er1(i) = 0.0;
     }
}


      Z.column(3) << er1;


      // Measurement Error Covariance Matrix
      Matrix Pzz(mp,mp);
      Pzz = 0;

      Pzz =  er1*er1.t();



      /* Defining the observation matrix */

      Matrix H(mp,4*Nn);
        H = 0;
        int bct = 1;

        for(int i=0; i<mp; i++)
          {
            H(bct,GID[i]) = 1.0;
            bct ++;
           }
```

```
        //Gain Matrix
        Matrix KG(4*Nn,mp);
        Matrix KG1 = H*P1m*H.t();
        Matrix KG2 = Pzz + KG1;
        Matrix KG3 = KG2.i();
        KG = (P1m*H.t())*KG3;

      //Update

      A0 = A0 + KG*(Z.column(1) - H*A0);
      A1 = A1 - KG*H*A1;
      A2 = KG*Z.column(3);
      A3 = A3 - KG*H*A3;
      A4 = A4 - KG*H*A4;

      // Give back to Sundance

      for (int i=1; i<=Nn; i++)
{
   Se00V.setElement(LID[i-1], (A0(i)-0.20)/0.65);
   Se01V.setElement(LID[i-1], A1(i)/0.65);
   Se02V.setElement(LID[i-1], A2(i)/0.65);
   Se03V.setElement(LID[i-1], A3(i)/0.65);
   Se04V.setElement(LID[i-1], A4(i)/0.65);

            Pww0V.setElement(LID[i-1], A0(i+Nn));
            Pww1V.setElement(LID[i-1], A1(i+Nn));
            Pww2V.setElement(LID[i-1], A2(i+Nn));
            Pww3V.setElement(LID[i-1], A3(i+Nn));
   Pww4V.setElement(LID[i-1], A4(i+Nn));

  K0V.setElement(LID[i-1], A0(i+2*Nn));
   K1V.setElement(LID[i-1], A1(i+2*Nn));
   K2V.setElement(LID[i-1], A2(i+2*Nn));
   K3V.setElement(LID[i-1], A3(i+2*Nn));
   K4V.setElement(LID[i-1], A4(i+2*Nn));

  P0V.setElement(LID[i-1], A0(i+3*Nn));
   P1V.setElement(LID[i-1], A1(i+3*Nn));
   P2V.setElement(LID[i-1], A2(i+3*Nn));
   P3V.setElement(LID[i-1], A3(i+3*Nn));
   P4V.setElement(LID[i-1], A4(i+3*Nn));
}


      DiscreteFunction::discFunc(Se00)->setVector(Se00V);
      DiscreteFunction::discFunc(Se01)->setVector(Se01V);
      DiscreteFunction::discFunc(Se02)->setVector(Se02V);
      DiscreteFunction::discFunc(Se03)->setVector(Se03V);
      DiscreteFunction::discFunc(Se04)->setVector(Se04V);

      DiscreteFunction::discFunc(Pww0)->setVector(Pww0V);
      DiscreteFunction::discFunc(Pww1)->setVector(Pww1V);
```

```
    DiscreteFunction::discFunc(Pww2)->setVector(Pww2V);
    DiscreteFunction::discFunc(Pww3)->setVector(Pww3V);
    DiscreteFunction::discFunc(Pww4)->setVector(Pww4V);

    DiscreteFunction::discFunc(K0)->setVector(K0V);
    DiscreteFunction::discFunc(K1)->setVector(K1V);
    DiscreteFunction::discFunc(K2)->setVector(K2V);
    DiscreteFunction::discFunc(K3)->setVector(K3V);
    DiscreteFunction::discFunc(K4)->setVector(K4V);

    DiscreteFunction::discFunc(P0)->setVector(P0V);
    DiscreteFunction::discFunc(P1)->setVector(P1V);
    DiscreteFunction::discFunc(P2)->setVector(P2V);
    DiscreteFunction::discFunc(P3)->setVector(P3V);
    DiscreteFunction::discFunc(P4)->setVector(P4V);

    Expr Se0 = new SpectralExpr(sbasis, List(Se00, Se01, Se02, Se03, Se04));
    Expr P = new SpectralExpr(sbasis, List(P0, P1, P2, P3, P4));
    Expr K = new SpectralExpr(sbasis, List(K0, K1, K2, K3, K4));

    /* Write out the updated state to VTK files */

    FieldWriter wr3 = new VTKWriter("K" + Teuchos::toString(ns/10));
    wr3.addMesh(mesh);
    wr3.addField("K0", new ExprFieldWrapper(K0));
    wr3.addField("K1", new ExprFieldWrapper(K1));
    wr3.addField("K2", new ExprFieldWrapper(K2));
    wr3.addField("K3", new ExprFieldWrapper(K3));
    wr3.addField("K4", new ExprFieldWrapper(K4));
    wr3.write();

    FieldWriter wr4 = new VTKWriter("P" + Teuchos::toString(ns/10));
    wr4.addMesh(mesh);
    wr4.addField("P0", new ExprFieldWrapper(P0));
    wr4.addField("P1", new ExprFieldWrapper(P1));
    wr4.addField("P2", new ExprFieldWrapper(P2));
    wr4.addField("P3", new ExprFieldWrapper(P3));
    wr4.addField("P4", new ExprFieldWrapper(P4));
    wr4.write();

  }
else
  {
    DiscreteFunction::discFunc(Se00)->setVector(Se00V);
    DiscreteFunction::discFunc(Se01)->setVector(Se01V);
    DiscreteFunction::discFunc(Se02)->setVector(Se02V);
    DiscreteFunction::discFunc(Se03)->setVector(Se03V);
    DiscreteFunction::discFunc(Se04)->setVector(Se04V);

    Expr Se0 = new SpectralExpr(sbasis, List(Se00, Se01, Se02, Se03, Se04));

  }
```

```
      fclose(mFile);
    }

  catch(exception& e)
    {
      Sundance::handleException(e);
    }
  Sundance::finalize();
}
```

A.2 Two-Dimensional Flow Control Problem

In this example, the unknown medium properties are modeled as an exponential function of a Polynomial Chaos expansion. Therefore the standard spectral library implemented in Sundance can not be used to solve the problem. In what follows a work around for the problem is presented, it entails expanding the equations for each coefficient in the chaos expansion independently and then assemble and solve a larger system. This also gives an insight on the implementation details of the spectral library within Sundance.

```
#include "Sundance.hpp"

/*Function to create a list of unknown functions of arbitrary size */
Expr makeBigUnknownFunctionList(int numFuncs, const BasisFamily& basis) {
  Array<Expr> big(numFuncs);
  for (unsigned int i=0; i<big.size(); i++)
    {
      big[i] = new UnknownFunction(basis);
    }
  return new ListExpr(big);
}



/* Function to create a list of test functions of arbitrary size */
Expr makeBigTestFunctionList(int numFuncs, const BasisFamily& basis) {
  Array<Expr> big(numFuncs);
  for (unsigned int i=0; i<big.size(); i++)
    {
      big[i] = new TestFunction(basis);
    }
  return new ListExpr(big);
}
```

```
/* Function to create a discrete space with an arbitrary number of functions */

DiscreteSpace makeBigDiscreteSpace(const Mesh& mesh,
                                   int numFuncs,
                                   const BasisFamily& basis,
                                   const VectorType<double>& vecType) {
  BasisArray big(numFuncs);
  for (unsigned int i=0; i<big.size(); i++)
    {
      big[i] = basis;
    }
  return DiscreteSpace(mesh, big, vecType); }

/* Function to evaluate the average of three Hermite Polynomials of Known
   dimension and order  where one is expressed as an exponential function.
   The function takes as input DiscreteFunction Expr and returns Expr as well */

Expr CijkList(Expr alpha1, Expr alpha2, Expr alpha3, DiscreteSpace d1,
      int VecSize, cijk Exp, int i, int j,int k)
{
  Vector<double> alpha1V = DiscreteFunction::discFunc(alpha1)->getVector();
  Vector<double> alpha2V = DiscreteFunction::discFunc(alpha2)->getVector();
  Vector<double> alpha3V = DiscreteFunction::discFunc(alpha3)->getVector();

  Expr rtn = new DiscreteFunction(d1, 0.0, "rtn");
  Vector<double> rtnV = DiscreteFunction::discFunc(rtn)->getVector();

  VectorSpace<double> sS = alpha1V.space();
  int lowc = sS.lowestLocallyOwnedIndex();
  int highc = lowc +sS.numLocalElements();

  for (int ix=lowc; ix<highc; ix++)
    {
      rtnV.setElement(ix,Exp.sumexpectation3(alpha1V.getElement(ix),
 alpha2V.getElement(ix), alpha3V.getElement(ix), i, j, k));
    }

  DiscreteFunction::discFunc(rtn)->setVector(rtnV);

  return rtn;
}


int main(int argc, char** argv)
{
  try
    {
      Sundance::init(&argc, &argv);

      /* We will do our linear algebra using Epetra */
      VectorType<double> vecType = new EpetraVectorType();

      MeshType meshType = new BasicSimplicialMeshType();
      MeshSource mesher = new ExodusNetCDFMeshReader("doubleloop1.ncdf", meshType);
```

```
Mesh mesh = mesher.getMesh();

/* Create a cell filter that will identify the maximal cells
 * in the interior of the domain */
CellFilter interior = new MaximalCellFilter();
CellFilter edges = new DimensionalCellFilter(1);
CellFilter Source1 = edges.labeledSubset(1);
CellFilter Source2 = edges.labeledSubset(2);
CellFilter Sink = edges.labeledSubset(3);

const int nf = 10 /*Number of unknown functions */;

/* Create unknown and test functions, discretized using first-order
 * Lagrange interpolants */

BasisFamily basis = new Lagrange(2);

Expr Unk = makeBigUnknownFunctionList(nf, basis);

Expr Se0 = Unk[0];    Expr Se1 = Unk[1];    Expr Se2 = Unk[2];
Expr Se3 = Unk[3];    Expr Se4 = Unk[4];

Expr Pw0 = Unk[5];    Expr Pw1 = Unk[6];    Expr Pw2 = Unk[7];
Expr Pw3 = Unk[8];    Expr Pw4 = Unk[9];

Expr Test = makeBigTestFunctionList(nf, basis);

Expr Ns0 = Test[0];   Expr Ns1 = Test[1];   Expr Ns2 = Test[2];
Expr Ns3 = Test[3];   Expr Ns4 = Test[4];

Expr Nn0 = Test[5];   Expr Nn1 = Test[6];   Expr Nn2 = Test[7];
Expr Nn3 = Test[8];   Expr Nn4 = Test[9];

/* Create differential operator and coordinate functions */
Expr x = new CoordExpr(0);
Expr y = new CoordExpr(1);
Expr dx = new Derivative(0);
Expr dy = new Derivative(1);
Expr grad = List(dx, dy);


/* We need a quadrature rule for doing the integrations */
QuadratureFamily quad2 = new GaussianQuadrature(4);

DiscreteSpace d1(mesh, basis, vecType);
DiscreteSpace discSpace = makeBigDiscreteSpace(mesh, nf, basis, vecType);

/* Initial Guess */
Expr u0 = new DiscreteFunction(discSpace, 0.0, "u0");

/*Initial Condition */
Expr Seo0 = new DiscreteFunction(d1, 0.0, "Seo0");
Expr Seo1 = new DiscreteFunction(d1, 0.0, "Seo1");
Expr Seo2 = new DiscreteFunction(d1, 0.0, "Seo2");
```

```
   Expr Seo3 = new DiscreteFunction(d1, 0.0, "Seo3");
   Expr Seo4 = new DiscreteFunction(d1, 0.0, "Seo4");

   /* Cross-sectional Area */
   double As = 10000;

   /* The intrinsic permeability of the medium K, and its porosity P*/

   int nNodes1 = mesh.numCells(0);
   int nElems = mesh.numCells(1);
   int vecSize = nElems + nNodes1;

   /* representation of alpha = K/P */
   Expr alpha0 = new DiscreteFunction(d1, 0.0, "alpha0");
   Expr alpha1 = new DiscreteFunction(d1, 0.0, "alpha1");
   Expr alpha2 = new DiscreteFunction(d1, 0.0, "alpha1");
   Expr alpha3 = new DiscreteFunction(d1, 0.0, "alpha1");

   /*Dimensions and order of spectral basis */
   int ndim = 3;
   int order = 2;
   const int nterm = 5;

   /*Randomly Generated Initial Guess for alpha */
   Vector<double> alpha0V  = DiscreteFunction::discFunc(alpha0)->getVector();
   Vector<double> alpha1V  = DiscreteFunction::discFunc(alpha1)->getVector();
   Vector<double> alpha2V  = DiscreteFunction::discFunc(alpha2)->getVector();
   Vector<double> alpha3V  = DiscreteFunction::discFunc(alpha3)->getVector();

   int32 seed = 19850;               // random seed
   StochasticLib1 sto(seed);

   for (int i=0; i<vecSize; i++)
     {
alpha0V.setElement(i, 2.93 + 0.1*sto.Normal(0,1));
       seed++;
       alpha1V.setElement(i, 0.005*sto.Normal(0,1));
       seed++;
       alpha2V.setElement(i, 0.005*sto.Normal(0,1));
       seed++;
       alpha3V.setElement(i, 0.005*sto.Normal(0,1));
     }

   DiscreteFunction::discFunc(alpha1)->setVector(alpha1V);
   DiscreteFunction::discFunc(alpha2)->setVector(alpha2V);
   DiscreteFunction::discFunc(alpha3)->setVector(alpha3V);

   double MuO = 1.735; /*Oil Viscosity */
   double MuW = 1.735; /*Water Viscosity */
   double Pe = 10000.0;

   /*Initial Guess for the Water Injection Rate to be Controlled */
   /*Source 1*/
   double Qa0 = 300.00;
```

```
 double Qa1 = 10.00;
 double Qa2 = 0.0;
 double Qa3 = 0.0;
 double Qa4 = 5.50;
 /*Source 2 */
 double Qb0 = 300.00;
 double Qb1 = 10.00;
 double Qb2 = 0.0;
 double Qb3 = 0.0;
 double Qb4 = 5.50;

 /* Setup the time stepping with timestep = 0.005 */

 double deltaT = 0.005;

 double Sm = 0.35;

 cijk Exp(ndim, order);

 /* Evaluating the coefficients of the various orders of the unknown saturation */

 Expr Squad0 = (1*1*Se0*Se0*Se0*Se0 + 6*1*Se0*Se0*Se1*Se1 + 6*1*Se0*Se0*Se2*Se2 +
6*1*Se0*Se0*Se3*Se3 + 6*2*Se0*Se0*Se4*Se4 + 12*2*Se0*Se1*Se1*Se4 +
4*8*Se0*Se4*Se4*Se4 + 1*3*Se1*Se1*Se1*Se1 + 6*1*Se1*Se1*Se2*Se2 +
6*1*Se1*Se1*Se3*Se3 + 6*10*Se1*Se1*Se4*Se4 + 1*3*Se2*Se2*Se2*Se2 +
6*1*Se2*Se2*Se3*Se3 + 6*2*Se2*Se2*Se4*Se4 + 1*3*Se3*Se3*Se3*Se3
+ 6*2*Se3*Se3*Se4*Se4 + 1*60*Se4*Se4*Se4*Se4  )/ 1;
 Expr Squad1 = (4*1*Se0*Se0*Se0*Se1 + 12*2*Se0*Se0*Se1*Se4 + 4*3*Se0*Se1*Se1*Se1 +
12*1*Se0*Se1*Se2*Se2 + 12*1*Se0*Se1*Se3*Se3 + 12*10*Se0*Se1*Se4*Se4
+ 4*12*Se1*Se1*Se1*Se4 + 12*2*Se1*Se2*Se2*Se4 + 12*2*Se1*Se3*Se3*Se4
+ 4*68*Se1*Se4*Se4*Se4  )/ 1;
 Expr Squad2 = (4*1*Se0*Se0*Se0*Se2 + 12*1*Se0*Se1*Se1*Se2 + 4*3*Se0*Se2*Se2*Se2 +
12*1*Se0*Se2*Se3*Se3 + 12*2*Se0*Se2*Se4*Se4 + 12*2*Se1*Se1*Se2*Se4
+ 4*8*Se2*Se4*Se4*Se4  )/ 1;
 Expr Squad3 = (4*1*Se0*Se0*Se0*Se3 + 12*1*Se0*Se1*Se1*Se3 + 12*1*Se0*Se2*Se2*Se3 +
4*3*Se0*Se3*Se3*Se3 + 12*2*Se0*Se3*Se4*Se4 + 12*2*Se1*Se1*Se3*Se4 +
4*8*Se3*Se4*Se4*Se4  )/ 1;
 Expr Squad4 = (4*2*Se0*Se0*Se0*Se4 + 6*2*Se0*Se0*Se1*Se1 + 6*8*Se0*Se0*Se4*Se4 +
12*10*Se0*Se1*Se1*Se4 + 12*2*Se0*Se2*Se2*Se4 + 12*2*Se0*Se3*Se3*Se4
+ 4*60*Se0*Se4*Se4*Se4 + 1*12*Se1*Se1*Se1*Se1 + 6*2*Se1*Se1*Se2*Se2
+ 6*2*Se1*Se1*Se3*Se3 + 6*68*Se1*Se1*Se4*Se4 + 6*8*Se2*Se2*Se4*Se4
+ 6*8*Se3*Se3*Se4*Se4 + 1*544*Se4*Se4*Se4*Se4  )/ 2;
 Expr Scube0 = (1*1*Se0*Se0*Se0 + 3*1*Se0*Se1*Se1 + 3*1*Se0*Se2*Se2 + 3*1*Se0*Se3*Se3
+ 3*2*Se0*Se4*Se4 + 3*2*Se1*Se1*Se4 + 1*8*Se4*Se4*Se4  )/ 1;
 Expr Scube1 = (3*1*Se0*Se0*Se1 + 6*2*Se0*Se1*Se4 + 1*3*Se1*Se1*Se1 + 3*1*Se1*Se2*Se2
+ 3*1*Se1*Se3*Se3 + 3*10*Se1*Se4*Se4  )/ 1;
 Expr Scube2 = (3*1*Se0*Se0*Se2 + 3*1*Se1*Se1*Se2 + 1*3*Se2*Se2*Se2 + 3*1*Se2*Se3*Se3
+ 3*2*Se2*Se4*Se4  )/ 1;
 Expr Scube3 = (3*1*Se0*Se0*Se3 + 3*1*Se1*Se1*Se3 + 3*1*Se2*Se2*Se3 + 1*3*Se3*Se3*Se3
+ 3*2*Se3*Se4*Se4  )/ 1;
 Expr Scube4 = (3*2*Se0*Se0*Se4 + 3*2*Se0*Se1*Se1 + 3*8*Se0*Se4*Se4 + 3*10*Se1*Se1*Se4
+ 3*2*Se2*Se2*Se4 + 3*2*Se3*Se3*Se4 + 1*60*Se4*Se4*Se4  )/ 2;
 Expr Ssquare0 = (1*1*Se0*Se0 + 1*1*Se1*Se1 + 1*1*Se2*Se2 + 1*1*Se3*Se3 + 1*2*Se4*Se4)/ 1;
 Expr Ssquare1 = (2*1*Se0*Se1 + 2*2*Se1*Se4  )/ 1;
```

```
  Expr Ssquare2 = (2*1*Se0*Se2  )/ 1;
  Expr Ssquare3 = (2*1*Se0*Se3  )/ 1;
  Expr Ssquare4 = (2*2*Se0*Se4 + 1*2*Se1*Se1 + 1*8*Se4*Se4  )/ 2;

  /*Coefficients of water relative permeability */
  Expr Krw0 = 0.8723*Squad0 + 0.1632*Scube0 - 0.0392*Ssquare0 + 0.0037*Se0;
  Expr Krw1 = 0.8723*Squad1 + 0.1632*Scube1 - 0.0392*Ssquare1 + 0.0037*Se1;
  Expr Krw2 = 0.8723*Squad2 + 0.1632*Scube2 - 0.0392*Ssquare2 + 0.0037*Se2 + 0.1*Krw0;
  Expr Krw3 = 0.8723*Squad3 + 0.1632*Scube3 - 0.0392*Ssquare3 + 0.0037*Se3;
  Expr Krw4 = 0.8723*Squad4 + 0.1632*Scube4 - 0.0392*Ssquare4 + 0.0037*Se4;

  /*Coefficients of oil relative permeability */
  Expr Kro0 = -2.7777*Squad0 + 4.9888*Scube0  - 1.3543*Ssquare0 - 1.8568*Se0 + 1.0;
  Expr Kro1 = -2.7777*Squad1 + 4.9888*Scube1  - 1.3543*Ssquare1 - 1.8568*Se1;
  Expr Kro2 = -2.7777*Squad2 + 4.9888*Scube2  - 1.3543*Ssquare2 - 1.8568*Se2 + 0.1*Kro0;
  Expr Kro3 = -2.7777*Squad3 + 4.9888*Scube3  - 1.3543*Ssquare3 - 1.8568*Se3;
  Expr Kro4 = -2.7777*Squad4 + 4.9888*Scube4  - 1.3543*Ssquare4 - 1.8568*Se4;

  /* Coefficients for the capillary pressure */
  Expr Pc0 = Pe*(3.9051*Squad0 - 12.0155*Scube0 + 14.2754*Ssquare0 - 8.4337*Se0 + 3.2688);
  Expr Pc1 = Pe*(3.9051*Squad1 - 12.0155*Scube1 + 14.2754*Ssquare1 - 8.4337*Se1);
  Expr Pc2 = Pe*(3.9051*Squad2 - 12.0155*Scube2 + 14.2754*Ssquare2 - 8.4337*Se2) + 0.1*Pc0;
  Expr Pc3 = Pe*(3.9051*Squad3 - 12.0155*Scube3 + 14.2754*Ssquare3 - 8.4337*Se3);
  Expr Pc4 = Pe*(3.9051*Squad4 - 12.0155*Scube4 + 14.2754*Ssquare4 - 8.4337*Se4);


  /* Weak forms of the PDE's corresponding to the different terms in the chaos expansion */

  Expr Weqn0 = Integral(interior, Ns0*As*(1.0-Sm)*(Se0 - Seo0), quad2) + Integral(interior,(grad*Ns0)
*As/MuW*exp(alpha0 + 0.5*alpha1*alpha1 + 0.5*alpha2*alpha2 + 0.5*alpha3*alpha3)*(CijkList(alpha1
,alpha2,alpha3, d1, vecSize, Exp, 0, 0, 0)*Krw0*(grad*Pw0) + CijkList(alpha1,alpha2,alpha3, d1,
vecSize, Exp, 1, 1, 0)*Krw1*(grad*Pw1) + CijkList(alpha1,alpha2,alpha3, d1, vecSize, Exp, 2, 2, 0)
*Krw2*(grad*Pw2) + CijkList(alpha1,alpha2,alpha3, d1,vecSize, Exp, 3, 3, 0)*Krw3*(grad*Pw3) +
CijkList(alpha1,alpha2,alpha3, d1, vecSize, Exp, 4,4, 0)*Krw4*(grad*Pw4) )*deltaT, quad2)
- Integral(Source1, Ns0*Qa0*deltaT, quad2) - Integral(Source2, Ns0*Qb0*deltaT, quad2);

  Expr Weqn1 = Integral(interior, Ns1*As*(1.0-Sm)*(Se1 - Seo1), quad2) + Integral(interior, (grad*Ns1)
*As/MuW*exp(alpha0 + 0.5*alpha1*alpha1 + 0.5*alpha2*alpha2 + 0.5*alpha3*alpha3)*(CijkList(alpha1,
alpha2,alpha3, d1, vecSize, Exp, 0, 1, 1)*Krw0*(grad*Pw1) + CijkList(alpha1,alpha2,alpha3, d1,
vecSize, Exp, 1, 0, 1)*Krw1*(grad*Pw0) + CijkList(alpha1,alpha2,alpha3, d1, vecSize, Exp, 1, 4, 1)
*Krw1*(grad*Pw4) + CijkList(alpha1,alpha2,alpha3, d1, vecSize, Exp, 4, 1, 1)*Krw4*(grad*Pw1) )
*deltaT, quad2)- Integral(Source1, Ns1*Qa1*deltaT,quad2) - Integral(Source2, Ns1*Qb1*deltaT,quad2);

  Expr Weqn2 = Integral(interior, Ns2*As*(1.0-Sm)*(Se2 - Seo2), quad2) + Integral(interior, (grad*Ns2)
*As/MuW*exp(alpha0 + 0.5*alpha1*alpha1 + 0.5*alpha2*alpha2 + 0.5*alpha3*alpha3)*(CijkList(alpha1,
alpha2,alpha3, d1, vecSize, Exp, 0, 2, 2)*Krw0*(grad*Pw2) + CijkList(alpha1,alpha2,alpha3, d1,
vecSize, Exp, 2, 0, 2)*Krw2*(grad*Pw0) )*deltaT, quad2)  - Integral(Source1, Ns2*Qa2*deltaT,quad2)
- Integral(Source2, Ns2*Qb2*deltaT,quad2);

  Expr Weqn3 = Integral(interior, Ns3*As*(1.0-Sm)*(Se3 - Seo3), quad2) + Integral(interior, (grad*Ns3)
*As/MuW*exp(alpha0 + 0.5*alpha1*alpha1 + 0.5*alpha2*alpha2 + 0.5*alpha3*alpha3)*(CijkList(alpha1,
alpha2,alpha3, d1, vecSize, Exp, 0, 3, 3)*Krw0*(grad*Pw3) + CijkList(alpha1,alpha2,alpha3, d1,
vecSize, Exp, 3, 0, 3)*Krw3*(grad*Pw0) )*deltaT, quad2)  - Integral(Source1, Ns3*Qa3*deltaT,quad2)
- Integral(Source2, Ns3*Qb3*deltaT,quad2);
```

```
  Expr Weqn4 = Integral(interior, Ns4*As*(1.0-Sm)*2*(Se4 - Seo4), quad2) + Integral(interior, (grad*Ns4)
  *As/MuW*exp(alpha0 + 0.5*alpha1*alpha1 + 0.5*alpha2*alpha2 + 0.5*alpha3*alpha3)*(CijkList(alpha1,
  alpha2,alpha3, d1, vecSize, Exp, 0, 4, 4)*Krw0*(grad*Pw4) + CijkList(alpha1,alpha2,alpha3, d1,
  vecSize, Exp, 1, 1, 4)*Krw1*(grad*Pw1) + CijkList(alpha1,alpha2,alpha3, d1, vecSize, Exp, 4, 0, 4)
  *Krw4*(grad*Pw0) + CijkList(alpha1,alpha2,alpha3, d1, vecSize, Exp, 4, 4, 4)*Krw4*(grad*Pw4) )*
  deltaT, quad2)- Integral(Source1, Ns4*Qa4*deltaT,quad2) - Integral(Source2, Ns4*Qb4*deltaT,quad2);

  Expr Oeqn0 = Integral(interior, -Nn0*As*(1.0-Sm)*(Se0 - Seo0), quad2) + Integral(interior, (grad*Nn0)
  *As/MuO*exp(alpha0 + 0.5*alpha1*alpha1 + 0.5*alpha2*alpha2 + 0.5*alpha3*alpha3)*(CijkList(alpha1,
  alpha2,alpha3, d1, vecSize, Exp, 0, 0, 0)*Kro0*(grad*(Pw0+Pc0)) + CijkList(alpha1,alpha2,alpha3,
  d1, vecSize, Exp, 1, 1, 0)*Kro1*(grad*(Pw1+Pc1)) + CijkList(alpha1,alpha2,alpha3, d1, vecSize,
  Exp, 2, 2, 0)*Kro2*(grad*(Pw2+Pc2)) + CijkList(alpha1,alpha2,alpha3, d1, vecSize, Exp, 3, 3, 0)*
  Kro3*(grad*(Pw3+Pc3)) + CijkList(alpha1,alpha2,alpha3, d1, vecSize, Exp, 4, 4, 0)*Kro4*
  (grad*(Pw4+Pc4)) )*deltaT, quad2) + Integral(Sink, Nn0*0.4*(Qa0+Qb0)*deltaT,quad2);

  Expr Oeqn1 = Integral(interior, -Nn1*As*(1.0-Sm)*(Se1 - Seo1), quad2) + Integral(interior, (grad*Nn1)
  *As/MuO*exp(alpha0 + 0.5*alpha1*alpha1 + 0.5*alpha2*alpha2 + 0.5*alpha3*alpha3)*(CijkList(alpha1,
  alpha2,alpha3, d1, vecSize, Exp, 0, 1, 1)*Kro0*(grad*(Pw1+Pc1)) + CijkList(alpha1,alpha2,alpha3,
  d1, vecSize, Exp, 1, 0, 1)*Kro1*(grad*(Pw0+Pc0)) + CijkList(alpha1,alpha2,alpha3, d1, vecSize, Exp,
  1, 4, 1)*Kro1*(grad*(Pw4+Pc4)) + CijkList(alpha1,alpha2,alpha3, d1, vecSize, Exp, 4, 1, 1)*Kro4*
  (grad*(Pw1+Pc1)) )*deltaT, quad2) + Integral(Sink, Nn1*0.4*(Qa1+Qb1)*deltaT,quad2);

  Expr Oeqn2 = Integral(interior, -Nn2*As*(1.0-Sm)*(Se2 - Seo2), quad2) + Integral(interior, (grad*Nn2)
  *As/MuO*exp(alpha0 + 0.5*alpha1*alpha1 + 0.5*alpha2*alpha2 + 0.5*alpha3*alpha3)*(CijkList(alpha1,
  alpha2,alpha3, d1, vecSize, Exp, 0, 2, 2)*Kro0*(grad*(Pw2+Pc2)) + CijkList(alpha1,alpha2,alpha3,
  d1, vecSize, Exp, 2, 0, 2)*Kro2*(grad*(Pw0+Pc0)) )*deltaT, quad2)
  + Integral(Sink, Nn2*0.4*(Qa2+Qb2)*deltaT,quad2);

  Expr Oeqn3 = Integral(interior, -Nn3*As*(1.0-Sm)*(Se3 - Seo3), quad2) + Integral(interior, (grad*Nn3)
  *As/MuO*exp(alpha0 + 0.5*alpha1*alpha1 + 0.5*alpha2*alpha2 + 0.5*alpha3*alpha3)*(CijkList(alpha1
  ,alpha2,alpha3, d1, vecSize, Exp, 0, 3, 3)*Kro0*(grad*(Pw3+Pc3)) + CijkList(alpha1,alpha2,alpha3,
  d1, vecSize, Exp, 3, 0, 3)*Kro3*(grad*(Pw0+Pc0)) )*deltaT, quad2)
  + Integral(Sink, Nn3*0.4*(Qa3+Qb3)*deltaT,quad2);

  Expr Oeqn4 = Integral(interior, -Nn4*As*(1.0-Sm)*(2*(Se4 - Seo4)), quad2) + Integral(interior, (grad*Nn4)
  *As/MuO*exp(alpha0 + 0.5*alpha1*alpha1 + 0.5*alpha2*alpha2 + 0.5*alpha3*alpha3)*(CijkList(alpha1,
  alpha2,alpha3, d1, vecSize, Exp, 0, 4, 4)*Kro0*(grad*(Pw4+Pc4)) + CijkList(alpha1,alpha2,alpha3, d1,
  vecSize, Exp, 1, 1, 4)*Kro1*(grad*(Pw1+Pc1)) + CijkList(alpha1,alpha2,alpha3, d1, vecSize, Exp, 4, 0,
  4)*Kro4*(grad*(Pw0+Pc0)) + CijkList(alpha1,alpha2,alpha3, d1, vecSize, Exp, 4, 4, 4)*Kro4*(grad*
  (Pw4+Pc4)))*deltaT, quad2) + Integral(Sink, Nn4*0.4*(Qa4+Qb4)*deltaT,quad2);


  /* The above set of equations is automatically generated using a C++ algorithm that takes the number of
dimensions and order as input */

  Expr eqn = Weqn0 + Oeqn0 + Weqn1 + Oeqn1 + Weqn2 + Oeqn2 + Weqn3 + Oeqn3 + Weqn4 + Oeqn4;

  Expr bc = EssentialBC(Sink, Nn0*(Pw0-10000.0),quad2)+ EssentialBC(Sink, Nn1*(Pw1-0.0),quad2)
+ EssentialBC(Sink, Nn2*(Pw2-0.0),quad2) + EssentialBC(Sink, Nn3*(Pw3-0.0),quad2)
+  EssentialBC(Sink,Nn4*(Pw4- 0.0),quad2);


  NonlinearOperator<double> F = new NonlinearProblem(mesh, eqn, bc,Test,Unk, u0, vecType);
```

```
      ParameterXMLFileReader reader("nox.xml");
      ParameterList noxParams = reader.getParameters();

      NOXSolver solver(noxParams, F);

      printf("Solving System \n");

      const int Nn = nNodes1;

      /*Measurement Locations */
      int GID1[13] = {5, 11, 19, 23, 33, 61, 69, 86, 87, 111, 117, 126, 131};

      int GID[Nn]; /* Target function is treated as a dicretized function on the mesh
       and thus considered as measurements available at all nodal locations */
      int LID[Nn];

      for (int i=1; i<=Nn; i++)
        {
          GID[i-1] = i;
        }

      // get local ID of node given GID

      const RefCountPtr<DOFMapBase>& dofMap = DiscreteFunction::discFunc(Seo0)->map();
      for (int i=0; i< Nn; i++)
        {
          int ID = mesh.mapGIDToLID(0, i);
          Array<int> dofIndices;
          dofMap -> getDOFsForCell(0,ID,0, dofIndices);

          LID[i] = dofIndices[0];

        }


      FILE * o1File; /*Output file to record the updated flow at Source 1 */
      o1File = fopen("Inflow1.dat","w");

      FILE * o2File; /*Output file to record the updated flow at Source 2 */
      o2File = fopen("Inflow2.dat","w");

      FILE * mFile;  /* File Containing the Discretized Target Function */
      mFile = fopen("../truemodel/STarget.dat", "r");

      FILE * m1File;  /*File Containing the observation Data */
      m1File = fopen("../truemodel/SMM.dat","r");

      int nSteps = 2001;
      for(int ns=0; ns < nSteps; ns++)
{
   solver.solve();

   Expr Pww0 = new DiscreteFunction(d1, 0.0, "Pww0");
```

```
Expr Pww1 = new DiscreteFunction(d1, 0.0, "Pww1");
Expr Pww2 = new DiscreteFunction(d1, 0.0, "Pww2");
Expr Pww3 = new DiscreteFunction(d1, 0.0, "Pww3");
Expr Pww4 = new DiscreteFunction(d1, 0.0, "Pww4");

Vector<double> SolnVec  = DiscreteFunction::discFunc(u0)->getVector();

Vector<double> Seo0V = DiscreteFunction::discFunc(Seo0)->getVector();
Vector<double> Seo1V = DiscreteFunction::discFunc(Seo1)->getVector();
Vector<double> Seo2V = DiscreteFunction::discFunc(Seo2)->getVector();
Vector<double> Seo3V = DiscreteFunction::discFunc(Seo3)->getVector();
Vector<double> Seo4V = DiscreteFunction::discFunc(Seo4)->getVector();

Vector<double> Pww0V = DiscreteFunction::discFunc(Pww0)->getVector();
Vector<double> Pww1V = DiscreteFunction::discFunc(Pww1)->getVector();
Vector<double> Pww2V = DiscreteFunction::discFunc(Pww2)->getVector();
Vector<double> Pww3V = DiscreteFunction::discFunc(Pww3)->getVector();
Vector<double> Pww4V = DiscreteFunction::discFunc(Pww4)->getVector();


       for (int n=0; n<vecSize; n++)
         {
    Seo0V.setElement(n, SolnVec.getElement(10*n));
    Seo1V.setElement(n, SolnVec.getElement(10*n+1));
    Seo2V.setElement(n, SolnVec.getElement(10*n+2));
    Seo3V.setElement(n, SolnVec.getElement(10*n+3));
    Seo4V.setElement(n, SolnVec.getElement(10*n+4));
    Pww0V.setElement(n, SolnVec.getElement(10*n+5));
    Pww1V.setElement(n, SolnVec.getElement(10*n+6));
    Pww2V.setElement(n, SolnVec.getElement(10*n+7));
    Pww3V.setElement(n, SolnVec.getElement(10*n+8));
    Pww4V.setElement(n, SolnVec.getElement(10*n+9));
  }

/* Updating the System parameters and states whenever a measurement is available */

      if ( ((ns% 100) == 0) && (ns > 0))
         {
           ColumnVector A0(3*Nn);
           ColumnVector A1(3*Nn);
           ColumnVector A2(3*Nn);
           ColumnVector A3(3*Nn);
           ColumnVector A4(3*Nn);

           for(int i=1; i<=3*Nn; i++)
             {
               if(i<=Nn)
                 {
                   A0(i) = 0.65*Seo0V.getElement(LID[i-1]) + 0.20;
                   A1(i) = 0.65*Seo1V.getElement(LID[i-1]);
                   A2(i) = 0.65*Seo2V.getElement(LID[i-1]);
                   A3(i) = 0.65*Seo3V.getElement(LID[i-1]);
                   A4(i) = 0.65*Seo4V.getElement(LID[i-1]);
                 }
```

```
    else if ((i > Nn) && (i<=2*Nn))
      {
        A0(i) = Pww0V.getElement(LID[i-(Nn+1)]);
        A1(i) = Pww1V.getElement(LID[i-(Nn+1)]);
        A2(i) = Pww2V.getElement(LID[i-(Nn+1)]);
        A3(i) = Pww3V.getElement(LID[i-(Nn+1)]);
        A4(i) = Pww4V.getElement(LID[i-(Nn+1)]);
      }
    else if ((i > 2*Nn) && (i<=3*Nn))
      {
        A0(i) = alpha0V.getElement(LID[i-(2*Nn+1)]);
        A1(i) = alpha1V.getElement(LID[i-(2*Nn+1)]);
        A2(i) = alpha2V.getElement(LID[i-(2*Nn+1)]);
        A3(i) = alpha3V.getElement(LID[i-(2*Nn+1)]);
        A4(i) = 0.0;
      }
  }


// State Error Covariance Matrix
Matrix P1m(3*Nn,3*Nn);
P1m = 0;

P1m = A1*A1.t()*Exp.expectation(0,1,1) + A2*A2.t()*Exp.expectation(0,2,2) +
               A3*A3.t()*Exp.expectation(0,3,3) + A4*A4.t()*Exp.expectation(0,4,4);


// Measurement Matrix

const int mp = 13;

Matrix Z(mp,nterm);
Z = 0.0;
ColumnVector mt(mp);

for(int i=1; i<=mp; i++)
  {
    fscanf(m1File, "%le", &mt(i));
  }

Z.column(1) << mt;

ColumnVector er1(mp);

for(int i=1; i<=mp; i++)
  {
    if (mt(i) > 0.20)
      {
        er1(i) = 0.001*mt(i);
      }
    else
      {
        er1(i) = 0.0;
      }
```

```
  }


Z.column(4) << er1;


// Measurement Error Covariance Matrix
Matrix Pzz(mp,mp);
Pzz = 0;

Pzz =  er1*er1.t();

/* Defining the observation matrix */

Matrix H(mp,3*Nn);
H = 0;

int bct = 1;

for(int i=0; i<mp; i++)
  {
    H(bct,GID1[i]) = 1.0;
    bct ++;
  }

cout << "computing the gain" <<endl;

//Gain Matrix
Matrix KG(3*Nn,mp);
Matrix KG1 = H*P1m*H.t();
Matrix KG2 = Pzz + KG1;
Matrix KG3 = KG2.i();
KG = (P1m*H.t())*KG3;


A0 = A0 + KG*(Z.column(1) - H*A0);
A1 = A1 - KG*H*A1;
A2 = A2 - KG*H*A2;
A3 = KG*Z.column(4);
A4 = A4 - KG*H*A4;


// Give back to Sundance


for (int i=1; i<=Nn; i++)
  {
    Seo0V.setElement(LID[i-1], (A0(i)-0.20)/0.65);
  Seo1V.setElement(LID[i-1], A1(i)/0.65);
  Seo2V.setElement(LID[i-1], A2(i)/0.65);
    Seo3V.setElement(LID[i-1], A3(i)/0.65);
Seo4V.setElement(LID[i-1], A4(i)/0.65);

      Pww0V.setElement(LID[i-1], A0(i+Nn));
```

```
                Pww1V.setElement(LID[i-1], A1(i+Nn));
                Pww2V.setElement(LID[i-1], A2(i+Nn));
                Pww3V.setElement(LID[i-1], A3(i+Nn));
                Pww4V.setElement(LID[i-1], A4(i+Nn));

  alpha0V.setElement(LID[i-1], A0(i+2*Nn));
  alpha1V.setElement(LID[i-1], A1(i+2*Nn));
  alpha2V.setElement(LID[i-1], A2(i+2*Nn));
  alpha3V.setElement(LID[i-1], A3(i+2*Nn));
}

            DiscreteFunction::discFunc(Seo0)->setVector(Seo0V);
            DiscreteFunction::discFunc(Seo1)->setVector(Seo1V);
            DiscreteFunction::discFunc(Seo2)->setVector(Seo2V);
            DiscreteFunction::discFunc(Seo3)->setVector(Seo3V);
            DiscreteFunction::discFunc(Seo4)->setVector(Seo4V);


            DiscreteFunction::discFunc(Pww0)->setVector(Pww0V);
            DiscreteFunction::discFunc(Pww1)->setVector(Pww1V);
            DiscreteFunction::discFunc(Pww2)->setVector(Pww2V);
            DiscreteFunction::discFunc(Pww3)->setVector(Pww3V);
            DiscreteFunction::discFunc(Pww4)->setVector(Pww4V);


            DiscreteFunction::discFunc(alpha0)->setVector(alpha0V);
            DiscreteFunction::discFunc(alpha1)->setVector(alpha1V);
            DiscreteFunction::discFunc(alpha2)->setVector(alpha2V);
            DiscreteFunction::discFunc(alpha3)->setVector(alpha3V);


            FieldWriter wr2 = new VTKWriter("alpha" + Teuchos::toString(ns/100));
            wr2.addMesh(mesh);
            wr2.addField("alpha0", new ExprFieldWrapper(alpha0));
            wr2.addField("alpha1", new ExprFieldWrapper(alpha1));
            wr2.addField("alpha2", new ExprFieldWrapper(alpha2));
            wr2.addField("alpha3", new ExprFieldWrapper(alpha3));
            wr2.write();
          }


 /*Update the system states and control every other timestep to minimize the mismatch
   between the predicted state and the target function */

 else if ( ((ns% 2) == 0) && (ns >= 0))
   {
     ColumnVector AC0(2*Nn + 2);
     ColumnVector AC1(2*Nn + 2);
     ColumnVector AC2(2*Nn + 2);
     ColumnVector AC3(2*Nn + 2);
     ColumnVector AC4(2*Nn + 2);

     for(int i=1; i<=2*Nn; i++)
       {
```

```
 if(i<=Nn)
   {
     AC0(i) = 0.65*Seo0V.getElement(LID[i-1]) + 0.20;
     AC1(i) = 0.65*Seo1V.getElement(LID[i-1]);
     AC2(i) = 0.65*Seo2V.getElement(LID[i-1]);
     AC3(i) = 0.65*Seo3V.getElement(LID[i-1]);
     AC4(i) = 0.65*Seo4V.getElement(LID[i-1]);
   }
 else if ((i > Nn) && (i<=2*Nn))
   {
     AC0(i) = Pww0V.getElement(LID[i-(Nn+1)]);
     AC1(i) = Pww1V.getElement(LID[i-(Nn+1)]);
     AC2(i) = Pww2V.getElement(LID[i-(Nn+1)]);
     AC3(i) = Pww3V.getElement(LID[i-(Nn+1)]);
     AC4(i) = Pww4V.getElement(LID[i-(Nn+1)]);
    }
}

      AC0(2*Nn+1) = Qa0;
      AC1(2*Nn+1) = Qa1;
      AC2(2*Nn+1) = Qa2;
      AC3(2*Nn+1) = Qa3;
      AC4(2*Nn+1) = Qa4;

      AC0(2*Nn+2) = Qb0;
      AC1(2*Nn+2) = Qb1;
      AC2(2*Nn+2) = Qb2;
      AC3(2*Nn+2) = Qb3;
      AC4(2*Nn+2) = Qb4;


      // State Error Covariance Matrix
      Matrix PC1m((2*Nn+2),(2*Nn+2));
      PC1m = 0;

      PC1m = AC1*AC1.t()*Exp.expectation(0,1,1) + AC2*AC2.t()*Exp.expectation(0,2,2) +
     AC3*AC3.t()*Exp.expectation(0,3,3) + AC4*AC4.t()*Exp.expectation(0,4,4);

      // Measurement Matrix

      const int Cmp = 160;

      Matrix ZC(Cmp,nterm);
      ZC = 0.0;
      ColumnVector mtC(Cmp);

      for(int i=1; i<=Cmp; i++)
{
   fscanf(mFile, "%le", &mtC(i));
}

      ZC.column(1) << mtC;

      ColumnVector Cer1(Cmp);
```

```
        for(int i=1; i<=Cmp; i++)
{
   if (mtC(i) > 0.20)
     {
       Cer1(i) = 0.001*mtC(i);
     }
   else
     {
       Cer1(i) = 0.0;
     }
}


      ZC.column(4) << Cer1;


      // Measurement Error Covariance Matrix
      Matrix PCzz(Cmp,Cmp);
      PCzz = 0;

      PCzz =  Cer1*Cer1.t();


      /* Defining the observation matrix */

      Matrix HC(Cmp,2*Nn+2);
        HC = 0;

        int bct = 1;

        for(int i=0; i<Cmp; i++)
          {
            HC(bct,GID[i]) = 1.0;
            bct ++;
           }

      cout << "computing the gain" <<endl;

        //Gain Matrix
        Matrix KGC(2*Nn+2,Cmp);
        Matrix KG1C = HC*PC1m*HC.t();
        Matrix KG2C = PCzz + KG1C;
        Matrix KG3C = KG2C.i();
        KGC = (PC1m*HC.t())*KG3C;


      //Update

      AC0 = AC0 + KGC*(ZC.column(1) - HC*AC0);
        AC1 = AC1 - KGC*HC*AC1;
        AC2 = AC2 - KGC*HC*AC2;
        AC3 = KGC*ZC.column(4);
        AC4 = AC4 - KGC*HC*AC4;
```

```
      // Give back to Sundance


        for (int i=1; i<=Nn; i++)
          {
   Seo0V.setElement(LID[i-1], (AC0(i)-0.20)/0.65);
   Seo1V.setElement(LID[i-1], AC1(i)/0.65);
   Seo2V.setElement(LID[i-1], AC2(i)/0.65);
   Seo3V.setElement(LID[i-1], AC3(i)/0.65);
   Seo4V.setElement(LID[i-1], AC4(i)/0.65);

            Pww0V.setElement(LID[i-1], AC0(i+Nn));
            Pww1V.setElement(LID[i-1], AC1(i+Nn));
            Pww2V.setElement(LID[i-1], AC2(i+Nn));
            Pww3V.setElement(LID[i-1], AC3(i+Nn));
            Pww4V.setElement(LID[i-1], AC4(i+Nn));
}

      Qa0 = AC0(2*Nn+1);
      Qa1 = AC1(2*Nn+1);
      Qa2 = AC2(2*Nn+1);
      Qa3 = AC3(2*Nn+1);
      Qa4 = AC4(2*Nn+1);

      Qb0 = AC0(2*Nn+2);
      Qb1 = AC1(2*Nn+2);
      Qb2 = AC2(2*Nn+2);
      Qb3 = AC3(2*Nn+2);
      Qb4 = AC4(2*Nn+2);

      DiscreteFunction::discFunc(Seo0)->setVector(Seo0V);
      DiscreteFunction::discFunc(Seo1)->setVector(Seo1V);
      DiscreteFunction::discFunc(Seo2)->setVector(Seo2V);
      DiscreteFunction::discFunc(Seo3)->setVector(Seo3V);
      DiscreteFunction::discFunc(Seo4)->setVector(Seo4V);

      DiscreteFunction::discFunc(Pww0)->setVector(Pww0V);
      DiscreteFunction::discFunc(Pww1)->setVector(Pww1V);
      DiscreteFunction::discFunc(Pww2)->setVector(Pww2V);
      DiscreteFunction::discFunc(Pww3)->setVector(Pww3V);
      DiscreteFunction::discFunc(Pww4)->setVector(Pww4V);

      fprintf(o1File, "%d   %le   %le   %le   %le   %le\n",  ns, Qa0, Qa1, Qa2, Qa3, Qa4);
      fprintf(o2File, "%d   %le   %le   %le   %le   %le\n",  ns, Qb0, Qb1, Qb2, Qb3, Qb4);

      fflush(o1File);
      fflush(o2File);

    }

  else
    {
      DiscreteFunction::discFunc(Seo0)->setVector(Seo0V);
```

```
      DiscreteFunction::discFunc(Seo1)->setVector(Seo1V);
      DiscreteFunction::discFunc(Seo2)->setVector(Seo2V);
      DiscreteFunction::discFunc(Seo3)->setVector(Seo3V);
      DiscreteFunction::discFunc(Seo4)->setVector(Seo4V);
    }


  if ( ((ns % 1) == 0)  && (ns >=  0) )
    {
        FieldWriter wr1 = new VTKWriter("S" + Teuchos::toString(ns/1));
        wr1.addMesh(mesh);
        wr1.addField("S0", new ExprFieldWrapper(Seo0));
        wr1.addField("S1", new ExprFieldWrapper(Seo1));
        wr1.addField("S2", new ExprFieldWrapper(Seo2));
        wr1.addField("S3", new ExprFieldWrapper(Seo3));
        wr1.addField("S4", new ExprFieldWrapper(Seo4));
        wr1.write();
    }
}

     fclose(mFile);
     fclose(o1File);
     fclose(o2File);
   }

  catch(exception& e)
    {
      Sundance::handleException(e);
    }
  Sundance::finalize();
}
```