Adaptive Algorithms for History Matching and Uncertainty Quantification

Submitted for the Degree of Doctor of Philosophy in

Petroleum Engineering

Asaad Abdollahzadeh

Institute of Petroleum Engineering

Heriot-Watt University

May 2014

ABSTRACT

Numerical reservoir simulation models are the basis for many decisions in regard to predicting, optimising, and improving production performance of oil and gas reservoirs. History matching is required to calibrate models to the dynamic behaviour of the reservoir, due to the existence of uncertainty in model parameters. Finally a set of history matched models are used for reservoir performance prediction and economic and risk assessment of different development scenarios.

Various algorithms are employed to search and sample parameter space in history matching and uncertainty quantification problems. The algorithm choice and implementation, as done through a number of control parameters, have a significant impact on effectiveness and efficiency of the algorithm and thus, the quality of results and the speed of the process. This thesis is concerned with investigation, development, and implementation of improved and adaptive algorithms for reservoir history matching and uncertainty quantification problems.

A set of evolutionary algorithms are considered and applied to history matching. The shared characteristic of applied algorithms is adaptation by balancing exploration and exploitation of the search space, which can lead to improved convergence and diversity. This includes the use of estimation of distribution algorithms, which implicitly adapt their search mechanism to the characteristics of the problem. Hybridising them with genetic algorithms, multiobjective sorting algorithms, and real-coded, multi-model and multivariate Gaussian-based models can help these algorithms to adapt even more and improve their performance. Finally diversity measures are used to develop an explicit, adaptive algorithm and control the algorithm's performance, based on the structure of the problem.

Uncertainty quantification in a Bayesian framework can be carried out by resampling of the search space using Markov chain Monte-Carlo sampling algorithms. Common critiques of these are low efficiency and their need for control parameter tuning. A Metropolis-Hastings sampling algorithm with an adaptive multivariate Gaussian proposal distribution and a K-nearest neighbour approximation has been developed and applied.

Keywords

History Matching, Uncertainty Quantification, Adaptive Algorithms, Evolutionary Algorithms, Estimation of Distribution Algorithms, Multiobjective Optimisation, Clustering, Markov Chain Monte Carlo, Adaptive Metropolis-Hastings.

ACKNOWLEDGMENTS

Firstly I would like to express my gratitude to my supervisors, Professor Mike Christie and Professor David Corne, who have provided me with the opportunity to work with them and study for this PhD. I am grateful for your patience, understanding, expertise, and friendly attitude. Your guidance and support throughout this study are truly appreciated.

I would also like to thank Alan Reynolds, Vasily Demyanov, and Dan Arnold, my colleagues and members of the research group, for their technical support. I am grateful for the assistance he provided throughout the research project.

I would like to thank BP for providing the real data and case studies used for my investigation. In particular, I'd like to thank Glyn Williams, Brian Davies, and Mike Elliott, for their support on data analysis and providing me with their invaluable experience.

Thanks to UK Technology Strategy Board (TSB) who sponsored the Research Project. Also, thanks to Schlumberger-GeoQuest and Halliburton-Landmark for use of their reservoir simulator software.

Thank you to all the members of the Uncertainty Project at the Institute of Petroleum Engineering, Heriot-Watt University.

I appreciate the invaluable comments from my examiners, Prof. Martin Blunt (Imperial College, London), Prof. John McCall (Robert Gordon University, Aberdeen) and Prof. Sebastian Geiger (Heriot-Watt University, Edinburgh). Their comments improved the quality of thesis significantly.

Finally, I'd like to thank my wife and my parents, the rest of the larger family and my friends in Edinburgh, UK, Kurdistan, and Iran for their support and encouragement of my endeavours.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF PUBLICATIONS

Journal publications:

- Abdollahzadeh, A., Reynolds, A., Christie, M.A., Corne, D., Williams, G., Davies, B. (2013). **Estimation of Distribution Algorithms Applied to History Matching.** SPE Number: 141161-PA. *SPE Journal, 18 (03), pp. 508-517.*
- Abdollahzadeh, A., Reynolds, A., Christie, M.A., Corne, D., Davies, B., Williams, G. (2012). **Bayesian Optimization Algorithm Applied to Uncertainty Quantification**. SPE Number: SPE-143290-PA. *SPE Journal*, 17 (03), pp. 865-873.

Conference publications:

- Abdollahzadeh, A., Christie, M.A., Corne, D., Davies, B., Elliott, M. (2013). **An Adaptive Evolutionary Algorithm for History matching**. SPE number 164824. SPE EUROPEC/EAGE Annual Conference and Exhibition, London, UK.
- Abdollahzadeh, A., Christie, M.A., Corne, D. (2012). **Gaussian-based Estimation of Distribution Algorithms for History Matching.** SPE number 161951. Abu Dhabi International Petroleum Exhibition & Conference (ADIPEC). Abu Dhabi, UAE.
- Abdollahzadeh, A., Reynolds, A., Christie, M.A., Corne, D., Williams, G., Davies, B. (2012). **Multi-objective Scheme of Estimation of Distribution Algorithm for History matching.** 13th European Conference on the Mathematics of Oil Recovery (ECMOR). Biarritz, France.
- Reynolds, A., Abdollahzadeh, A., Christie, M.A., Corne, D., Williams, G., Davies, B. (2012). **Guide Objective Assisted Particle Swarm Optimization and its Application to History Matching**. 12th International Conference on Parallel Problem Solving from Nature (PPSN). Taormina, Italy.
- Abdollahzadeh, A., Reynolds, A., Christie, M.A., Corne, D., Williams, G., Davies, B. (2012). **On Population Diversity Measures of the Evolutionary Algorithms used in History Matching**. SPE number: 154488. SPE EUROPEC/EAGE Annual Conference and Exhibition, Copenhagen, Denmark.
- Abdollahzadeh, A., Christie, M.A., Corne, D. (2012). **Reservoir Uncertainty Quantification using Clustering Methods.** SIAM Conference on Uncertainty Quantification (UQ12). Raleigh, North Carolina, USA.
- Abdollahzadeh, A., Reynolds, A., Christie, M.A., Corne, D., (2012). **A Parallel GA-EDA Hybrid Algorithm for History matching**. SPE number: 153750. SPE Oil and Gas India Conference and Exhibition (OGIC). Mumbai, India.

- Reynolds, A., Abdollahzadeh, A., Christie, M.A., Corne, D., Williams, G., Davies, B. (2011). **A Parallel BOA-PSO Hybrid Algorithm for History Matching**. 2011 IEEE Congress on Evolutionary Computation (CEC). New Orleans, U.S.A.
- Abdollahzadeh, A., Reynolds, A., Christie, M.A., Corne, D., Davies, B., Williams, G. (2011). **Bayesian Optimization Algorithm Applied to Uncertainty Quantification**. SPE Number: 143290-MS. SPE EUROPEC/EAGE Annual Conference and Exhibition. Vienna, Austria.
- Abdollahzadeh, A., Reynolds, A., Christie, M.A., Corne, D., Williams, G., Davies, B. (2011). **Estimation of Distribution Algorithms Applied to History Matching**. SPE Number: 141161-MS. SPE Reservoir Simulation Symposium (RSS). Houston, Texas, U.S.A.

CHAPTER 1:

INTRODUCTION

"Science always has its origin in the adaptation of thought to some definite field of experience."

Ernst Mach (1838-1916)

1.1 Background

Reservoir simulation models are widely used in the upstream oil and gas industry for multi-million dollar development and operational decision making, such as reservoir performance forecasting, development optimisation plans and well placement problems. A reservoir simulation model is a numerical model based on the discretisation of the reservoir in space and time. It is built from initial uncertain parameters then validated and tuned through a process known as history matching. The calibrated reservoir model is then used for predictions and uncertainty quantification in reservoir management. Figure 1.1 shows the principal workflow in reservoir simulation.

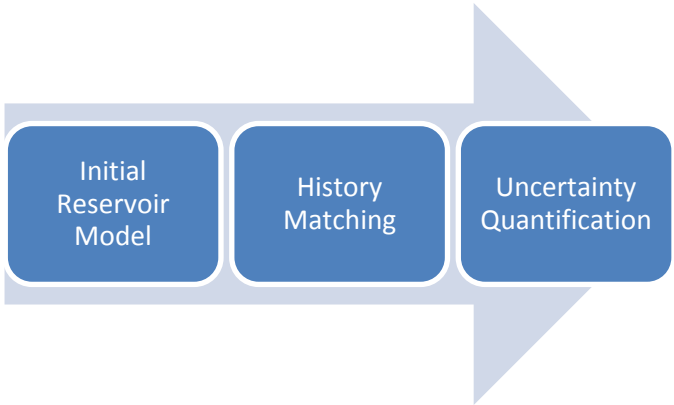


Figure 1.1: Principal workflow in reservoir simulation.

History matching is the process of calibrating the uncertain parameters of a reservoir model to available noisy observation data of the reservoir. History matching is usually a high-dimensional problem, since many model input parameters are uncertain. It is a very complex, non-linear problem, as there is strong non-linearity between production response and reservoir model parameters. It is an inverse problem, i.e. instead of using reservoir models to predict reservoir performance, observed reservoir behaviours are used to estimate reservoir model parameters. It is an ill-posed problem with non-unique solutions, as many possible reservoir models can have a similar production response that matches the observation data.

History matching, to be done manually or automatically, is a time and resource consuming process which requires experience and expertise. A structured approach is required, which involves a team approach and close collaboration between geoscientists and engineers. The process is started by examining the sensitivity of the reservoir to the model parameters, based on performing a procedure of the consequent runs. Based on the results of the sensitivity study, uncertainty parameter ranges are defined. The next step is to define match points and levels of tolerance for the difference between the historical and simulated data. Then simulation runs are scoped with a reservoir simulator in an iterative loop to obtain as many as possible informative and quality match runs at an affordable computational cost. Finally, these models conditioned to the historical data are used for decision making in reservoir management.

Reservoir uncertainty quantification is carried out using statistical and probabilistic methods on the result of a history matching. The result of assisted history matching is a set of model realisations that match observation data. These models may produce different results, in other words a range of results, in the prediction phase when we let the model predict future reservoir performance.

The uncertainty in prediction must be quantified to better and more reliably assess the risk in investment decisions. If correctly sampled, the ensemble of realisations obtained in history matching represents the posterior distribution, which provides an assessment of the uncertainty of the model parameters. Moreover, using these realisations for

predicting the future performance the reservoir, the uncertainty in prediction can be obtained. Finally, results can be shown using statistical properties such as the credible intervals (i.e. P10, P50 and P90).

The main challenges in the process of history matching and uncertainty quantification are a high number of uncertainty parameters, the presence of local minima in the parameter search space, and high computational complexity of the simulation runs. A review of the literature reveals that a variety of evolutionary algorithms are applied to history matching. In addition, there are several sampling algorithms, which are applied to uncertainty quantification.

The main critique of these algorithms in history matching and uncertainty quantification is that they have a rigid mechanism, so that, firstly, the user must carry out tuning of their control parameters and secondly, they cannot adapt to changes in condition or the circumstances of the problem.

The work carried out in this thesis has focussed on the development and application of algorithms for history matching and uncertainty quantification which adapt to the structure of the problems. An adaptive algorithm uses a set of implicit or explicit instructions and guidelines in its mechanism to adapt to the structure of the problem or changes in the circumstances or environment of the problem. Hence, adaptive algorithms are able to adjust intelligently their mechanism to achieve the best possible outcome.

1.2 Research objectives

The main theme of this thesis is to develop and apply adaptive algorithms for history matching and uncertainty quantification. The objective is not to develop and implement a single algorithm that can handle adaptation of all the major elements of evolutionary algorithms (EAs) in history matching or Markov chain Monte-Carlo (MCMC) sampling algorithms in uncertainty quantification. Instead we aim to develop different algorithms that can each address adaptation of one or more elements, resulting in more robust and efficient algorithms which are able to balance the search or sampling based on the structure of the problems.

The thesis firstly involves development and application of adaptive and flexible evolutionary search algorithms for different history matching problems to efficiently obtain a diverse set (ensemble) of good fitting-to-history models in the parameter search space. Secondly, we aim to develop and apply an ensemble-based adaptive MCMC method for efficient sampling of the posterior probability, as approximated by the ensemble of obtained history-matched models, to achieve probability distribution of the predictive parameters.

1.3 Structure of the thesis

The rest of this thesis will be structured around four parts:

- In the first part (Chapter 2), the literature is reviewed for reservoir simulation, history matching and uncertainty quantification, as well as evolutionary algorithms and clustering techniques.
- In the second part (Chapters 3-7), a diverse set of flexible and adaptive algorithms are developed and applied to optimisation problems in history matching:
    - Chapter 3 reviews estimation of distribution algorithms (EDAs), a set of implicitly adaptive evolutionary algorithms. It describes the application of three selected EDAs to history matching of a synthetic and a real North Sea reservoir model. The results are compared with results reported in the literature using other evolutionary algorithms.
    - Chapter 4 reviews adaptation of objective function in history matching by using multiobjective optimisation algorithms. It implements a multiobjective sorting mechanism in EDAs and applies it to selected optimisation and history matching problems. The results are compared to results of single objective optimisation for the same problems.
    - Chapter 5 reviews hybridisation of evolutionary algorithms as a way of adapting the search mechanism. First, it describes a simulated binary genetic algorithm for real-coded optimisation problems. Then, it presents a histogram-based EDA equipped with an incremental learning mechanism and finally develops a hybrid algorithm that combines the

advantages of the two algorithms. Finally, comparative study of these algorithms is carried out based on results obtained for history matching of selected synthetic and real examples.

  - Chapter 6 presents a set of Gaussian-based EDAs which can adapt to multivariety and multi-modality in continuous parameter spaces. It reviews Gaussian distribution as the probability model in EDAs for continuous history matching and presents the results of application on selected test functions and reservoir models.
  - Chapter 7 reviews different diversity measures and describes a diversity-based, explicitly adaptive EDA for history matching. The adaptive algorithm maintains the balance between the diversity and convergence throughout the search. The results of the algorithm have been compared with the tuned EDA.

- In the third part (chapter 8), an adaptive MCMC sampling will be developed for uncertainty quantification. The developed algorithm uses an adaptive Gaussian proposal distribution for sampling, k-nearest neighbours approximation to avoid forward simulations of numerous resampled models, and probabilistic-distance clustering to obtain the starting points of multiple walks. The new algorithm is demonstrated through several applications.
- In the fourth part (chapter 9), a summary of the chapters and general conclusions of the research are presented. The outstanding research contributions are summarised and, finally, recommendations for future work in the direction of this research are provided.

CHAPTER 2:
A LITERATURE REVIEW ON RESERVOIR HISTORY MATCHING AND UNCERTAINTY QUANTIFICATION AND ALGORITHMS USED

"Either write something worth reading, or do something worth writing."

Benjamin Franklin (1706 - 1790)

2.1 Introduction

Soft computing differs from traditional (hard) computing as it is tolerant of imprecision, uncertainty, and partial truth. It is also tractable, robust, efficient and affordable. Soft computing, nowadays, plays a vital role in modern reservoir management using techniques for intelligent reservoir characterization, engineering, and prediction. It helps to make sound reservoir decisions and improves the asset value of the oil and gas companies.

The ultimate outcome of reservoir management is a reservoir model with realistic tolerance for imprecision and uncertainty. Intelligent techniques of soft computing such as Evolutionary Algorithms (EAs) are being used for uncertainty analysis, risk assessment, optimization, and history matching of reservoir models.

History matching in reservoir simulation is calibration or conditioning a reservoir simulation model to historical production data. This technique is essential to prove an initial model created for a reservoir before using the model as a true representation of the reservoir for predicting its future performance.

An important goal in history matching is to calibrate a reservoir model for use in production forecasting and optimisation. History matching is well-known to be a time-consuming and non-unique task; for these reasons, quantifying the uncertainty in predictions is also essential, as it affects many critical decisions.

This chapter reviews reservoir simulation, history matching and uncertainty quantification and the algorithms used. The chapter is organized as follows. First a review of the reservoir modelling and simulation is provided. Then common practices for history matching are discussed, followed by a review of the methods used for uncertainty quantification. Finally, clustering techniques, which are used throughout this thesis as way of discovering patterns and regularities, are reviewed.

2.2 Reservoir modelling and simulation

In general, simulation is the replica of some real thing, state of affairs, or process. The act of simulating something entails representing certain key characteristics or behaviours of a selected physical or abstract system. Nowadays, simulation is widely used in reservoir engineering to model hydrocarbon reservoirs in order to gain insights into the reservoir's response, test and choose between alternative production and development plans, and forecast performance and future state of the reservoirs.

The reservoir simulation process starts with the collection of valid source data representing key characteristics and behaviours. Static data are time-invariant direct or indirect measurements of the reservoir properties, such as seismic data, core measurement, fluid analysis, and well logs. These data will be integrated using traditional geo-statistical algorithms, and with the use of simplification in approximating and making assumptions within reservoirs, a reservoir dynamic model will be created based on three physical concepts of conservation of mass, isothermal

fluid phase behaviour, and the Darcy law for fluid flow through porous media. This is the case in traditional finite difference simulators; the fourth element, conservation of energy is added to this list by thermal simulators, mostly used by heavy-oil reservoirs, allowing temperatures to change within the reservoir.

Dynamic data are time variant measurements and observation data of the flow response that are related to the created static model through flow equations. Integration of data into the model involves an inverse problem and requires repetitive solution of the flow equation in the model, which is a time-consuming and tedious process, referred to as 'history matching'.

Depending on the decisions to be made, following modelling types are used:

2.2.1 Analytical Modelling (Material Balance)

Classical analytical modelling of a reservoir, also known as the material balance, is one of the fundamental tools in reservoir engineering. For the first time, Schilthuis (1936) derived material balance equations based on the basic mass balance of the fluids in the reservoir. The material balance is a simple possible model we can take for analysis of reservoir behaviour, especially in the early stages of the reservoir development. A classic reference for material balance is Katz (1936).

Material balance is a valuable tool to obtain a high level of understanding of the reservoir from available data. In addition to fluid injection and production, it includes fluid and rock expansion/compression effects, but it does not consider fluid flow inside the reservoir. It assumes the reservoir as a single or multiple tanks; then pressure measurements over time are used to identify the dominant production mechanism and determine the hydrocarbon in place and the recovery factor.

This analytical modelling can be particularly useful where compartmentalisation within the reservoir is a predominant feature. In such cases, it analyses reservoir tanks, inter-tank transmissibility and connections and provides the amount of reserves, the recovery factor, the volume (geometry and structure), compartments (sectors) and the connection between them and across faults, the drive mechanism and flooding pattern, aquifer size and strength, and the original gas cap volume.

2.2.2 Well Models

Wells are treated as source sinks in reservoir simulations. Two main tasks in the reservoir simulation which require specific considerations are calculating bottom-hole flowing pressure (BHP) from the pressure of the block containing well when the well flow rate is given, and well flow rate, when BHP is specified.

Representing and handling wells in reservoir simulation entails specific considerations. The reason for this is, firstly, reservoir model blocks are large in size compared to wells, and thus the average pressure computed by the reservoir simulator for the block is not necessarily representative of well pressure. Secondly, there are usually complex interactions between the reservoir and wellbore, which must be modelled explicitly for both production and injection wells.

Peaceman (1978) introduced the first analytical well model by assuming a well at the centre of a uniform square grid and a steady-state flow between the block and its neighbouring blocks. His method defined an equivalent well-block radius, at which the steady-state flowing pressure in the reservoir is equal to the numerically calculated pressure of the block containing the well. Chappelear & Williamson (1981) and Abou-Kassem & Aziz (1985) generalised this model for random block geometry, well location within the block, and reservoir boundaries.

Since reservoir simulation was introduced in the petroleum engineering community, numerical single well models have been used to study well performance in general and well productivity specifically (e.g. Sonier & Ombret 1973). These models incorporate all relevant geological and reservoir engineering data affecting the performance of the well. Hence, in a single well model, the well is treated as an isolated system.

However, compared to the entire reservoir, saturation and pressure changes are enormous around the well, and the flow is radial. Unlike the well test model which assumes a single flow system, the single well model uses a multi-phase flow system. Numerical single well models are used for production enhancement by studying different completion strategies, water coning / gas cusping behaviour, and pseudo relative permeability functions (Ezuka et al 2004).

2.2.3 Streamline Simulation

In streamline-based flow simulation, fluids are transported over a time-step along streamlines, rather than from cell-to-cell, as in conventional finite difference methods. Thus, streamlines represent an image of the instantaneous velocity field. Therefore, anything assumed to move with the total velocity field will follow the streamlines until the velocity field is updated to account for its changing behaviour in time. The spatial distribution of the static and petrophysical properties of the reservoir yields the geometry of the streamlines and the velocity at which components travel along an individual streamline. Examples of these static parameters are permeability, porosity, and relative permeability, along with produced or injected volumes at the wells

The strengths of streamline simulation are efficiency in both computational speed and memory (Thiele et al., 2010), since fluid transport is solved along 1D streamlines and updated at user-defined time-steps only. Thus, it can be significantly faster than finite difference simulations: for example, it could be efficiently applied in simulation of fine-grid detailed geological models. Apart from computational performance, there are certain reservoir systems that could be modelled more effectively by streamline simulation: examples are slightly compressible convective-dominant systems such as water or miscible gas injection to a resident oil reservoir, which are difficult to model with conventional finite difference simulation.

The drawbacks of streamline simulation come from its two features of a dual static Eulerian - dynamic Lagrangian grid and the independent streamlines assumption, which make it not efficient for diffusion, gravity, capillarity, compressibility, and thermal dominant systems such as gas expansion and capillary pressure driving reservoirs, since there is no well-defined flow direction. However, these problems are effectively and efficiently handled by conventional finite difference simulation. Most of the real reservoir engineering cases are hybrid cases, where, in the primary phase, production starts by natural depletion that creates a gas cap, followed by the secondary phase where a repressurisation scenario, e.g. water injection, is performed to keep pressure high enough for production.

Where connectivity is a serious issue in fluid flow simulation, streamline simulation can be used. An example of this is flood surveillance by pattern analysis, to quantify the volumetric flux between the pair of producer-injector. This is done by determining the well-rate allocation factors to associate produced and injected volumes, which does not involve flow simulation. Streamline simulation could also be used for flow simulation, where fluid transport is modelled along the streamline.

For problems involving connectivity, streamline simulation can effectively simulate the model in practical runtime. History matching and uncertainty quantification, given the large uncertainties in reservoir model parameters, involves a large number of full-field simulation models, which could be computationally too expensive using full-physics finite difference simulation.

2.2.4 Full-field Simulation

As mentioned earlier, material balance can be used to predict the volume in place or recovery factors. However, it cannot address questions about why the pressures in two sectors of the reservoir are different. Streamlines can help to understand connectivity, but never replace full-physics finite difference simulation. Full-field simulation models are irreplaceable, and the only choice in many situations.

Reservoir simulation models are widely used for multi-million dollar development and making operational decisions such as reservoir performance forecasting, development optimisation plans and well placement problems in the upstream oil and gas industry.

A reservoir simulation model is a numerical model based on the discretisation of the reservoir in space and time. It is built from initial uncertain parameters, then validated and tuned through a process known as history matching. The model involves and incorporates a variety of data, such as reservoir fluids' pressure, volume, and temperature properties (PVT), reservoir properties (e.g. porosities, permeabilities) and their spatial distribution, rock and fluid/rock properties, dynamic data and production/injection phasing and controls.

The main steps to build a reservoir simulation model are:

- Data collection: to collect input data outlined above and implement quality control of the collected data.
- Build the reservoir grid: to choose reservoir grid features and build the grid from the geological and geophysical data. Upscale the model if required, for the efficiency of the simulation time.
- Define reservoir properties, PVT model and rock/fluid properties.
- Set up equilibrium data: define equilibrium regions and fluid contacts.
- Setup production schedule: set up production and injection scenarios and field and well controls such as injection rates, bottomhole pressure.
- Define outputs: choose the output variables, the frequency of and type of outputting (map, plot …).

Typical reservoir simulation outputs are average field pressure, field fluid (water, oil and gas) production and injection rates, gridblock pressure and saturations (water, oil and gas) and individual well pressures and saturations, all as a function of time.

A reservoir performance study is a central part of many reservoir management activities. The study involves three stages: creating a reservoir simulation model, history matching, and uncertainty quantification using history matched models.

2.3 History matching

History matching is the process of calibrating uncertain parameters of a reservoir model to available observation data of the reservoir. This matching process is a very complex non-linear problem as there is strong non-linearity between production response and reservoir model parameters.

History matching is also a difficult task because it is an inverse problem, i.e. instead of using reservoir models to predict reservoir performance, the observed reservoir behaviours are used to estimate reservoir model parameters. It is an ill-posed problem with non-unique solutions, since many possible reservoir models can have similar production responses that match observation data. Observation data are often noisy and

are assumed to be randomly distributed and thus have Gaussian distribution with mean zero and a standard deviation (Oliver et al., 2008). Sample and measurement bias is expected to be corrected in quality control phase and it is not considered as tolerance for in history matching.

A variety of model input parameters are uncertain (unknown or less precisely known). This uncertainty is a result of reservoir heterogeneity, scarcity and limited accuracy of measurements. The main sources of the uncertainty in reservoir models are the geometry of the reservoir (e.g. porosity and permeability), the spatial distribution of rock properties, and reservoir-fluid properties (e.g. gravity, viscosity, and wettability).

The process of history matching involves changing the uncertain reservoir model input parameters until the model predictions agree closely enough with the observed data. This task is either carried out manually by an engineer editing the reservoir model input deck and adjusting parameters, based on experience and intuition, or automatically, using either research and public domain or proprietary and commercial codes.

2.3.1 Manual history matching

Manual history matching is a trial and error process that requires user experience. It is usually done by experienced reservoir engineers who have experience and prior information from the previous studies, the reservoir history and analogue fields, which allow them to know the forecasting behaviour of reservoir model to some extent.

Nevertheless, manual history matching can be a hugely time-consuming process. For large fields, it may take many months to yield a single acceptably matched model. It often merely results in the best practical solution within the decision time, and may eventually provide a reservoir description that may be impractical and inconsistent with the geologic interpretation.

The process of manual history matching is summarised in the following sections. In addition, a practical approach for model calibration is provided, which involves modification of uncertainty parameters of reservoir model in a particular sequence of scale and reservoir response parameters.

2.3.1.1 The procedure of manual history matching

The procedure of manual history matching involves modifying and adjusting reservoir model parameters from one simulation run to the following by trial and error, to obtain simulation results that fit to observed pressure and production data. It is a way of integrating production observation data into reservoir models.

Observation data are usually measurements of pressure, flow rates (water, oil, gas, or liquid), or ratios of different flow rates, made in wells (producer or injector). Observation data are usually measured at well locations which are quite limited in number, and consequently represent an extremely small percentage of the reservoir. On the other hand, the number of measurements can be extremely high especially for a field with a large number of wells and long period of history. For a summary on observation data and measurement types used in history matching, refer to Oliver et al. (2008).

After data gathering, quality control on available observation data which is usually measured from tubing communication, reallocation, and metering is required, before initiating the manual history-match. Williams et al. (1998) proposed a procedure for manual history matching which is now the basis for many history matching studies. The procedure is summarised in Table 2.1.

2.3.1.2 Adjusting model parameters

A reservoir simulation model consists of several parameters, of which many are the source of uncertainty. The degree of variation in reservoir properties with change in location within the reservoir is one the greatest uncertainties. This involves a scale up degree for degree of variation for representing variation in reservoir properties in the reservoir description and simulation, from pores and cores to different reservoirs in a field. These degrees of variation are defined as reservoir heterogeneity.

Table 2.1: The manual-history matching procedure

1	**Gather data**	RFT and MDT measurements Bottom-Hole Pressure data Allocated Production and Injection data Well Test data Tracer data Interference Tests data 4D Seismic
2	**Quality-Control**	Errors from tubing communication Data reallocation errors Metering errors
3	**Prepare tools**	Prepare analysis tools Observation plots Observation maps
4	**Identify key wells**	Wells completed in only one flow unit. Wells have RFT for pressure match. Wells with pulsed neutron logs. Newer wells with open-hole log for water match.
5	**Interpret reservoir**	RFT and spatial pressure gradient maps for pressure match. Water front maps and water occurrence coming from vertical rise. Lateral fingering, coning for water match.
6	**Repeat matching**	until acceptable model matched to history is achieved.
	6.1 Run the model	Initially controlled by total reservoir voidage for pressure match. Thereafter controlled by oil rate for saturation match (validation). Constrained by minimum BHFP and maximum fluid and gas rates.
	6.2 Compare model	Compare model results to observed and interpreted data.
	6.3 Adjust the model	Adjust model parameters (*See following table for details*).

From the nature of the underground rock formations, it could be easily concluded that all the reservoirs are heterogeneous with a particular degree of heterogeneity. The scales of heterogeneities are defined at four levels of complexity, as shown in Table 2.2 (Kelkar, 2002).

Table 2.2: Levels of Reservoir Heterogeneities

	Scale	*Measurements*	*Effect on Performance*
Pore Level	10–100 µm	Pore Distribution	Displacement Efficiency (Trapped Oil)
		Throat Distribution	
		Throat Openings	
		Rock Lithology	
		Grain Shape and Size	
Core Level	1-100 cm	Permeability	
		Porosity	Sweep Efficiency (Bypassed Oil)
		Rock Wettability	
		Capillary Pressure	
		Fluid Saturation	
Grid Level	10-100 m	Log Properties	Sweep Efficiency (Bypassed Oil)
		Fluid Contacts	
		Pinch-outs, Discontinuity	
		Permeability Trends	
		Compartmentalization	
Reservoir Level	>1000 m	Well and Inter-Well Tests	Extraction Efficiency (Untrapped Oil)
		Depositional Description	
		Tectonic Activity	

Common sources of uncertainty are usually taken as the history-match parameters. These data come from different sources. Key uncertainties are summarised in Table 2.3 (Kelkar, 2002).

Adjustment of model parameters to match observation data is done in a sequence of the scale and observation data type. We usually start history-match with the deepest zones (bottom-up approach) for water drive reservoirs or top-down for free gas reservoirs. In addition, we match pressure before saturations. The sequence for adjusting model parameters based on Williams et al. (1998) is described in

Table 2.4.

Table 2.3: Common uncertainty parameters

Data Type	*Data*	*Source*
Pore volumes	Volume	
	Net-to-Gross (NTG)	
	Porosity	
Permeabilities or transmissibilities	Horizontal (for field areal gradient pressure match)	
	Vertical	
Relative permeabilities	Curves	Special Core Analysis, Well Test
	End-points	Special Core Analysis
Capillary Pressure		Special Core Analysis, Well Log
Contacts	Gas-Oil (GOC)	
	Water-Oil (WOC)	
Compartments	Gas-Oil (GOC)	
	Water-Oil (WOC)	
Fault	Location	
	Transmissibility	
Rock	Compressibility	
	Irreducible Water Saturation	
	Oil Residual Saturations	
Aquifer	Size (Pore Volume)	
	Strength (Permeability)	

2.3.2 Assisted history matching

Recently, with the availability of modern computer hardware and software, history matching has been done automatically, although it is not fully automated yet. There has been some resistance to the idea of a fully automated process - the engineers want to feel that they can have a role in the procedure. In this thesis, the terms automatic history matching and assisted history matching are used interchangeably.

Assisted/automatic history matching employs optimisation techniques, where the objective function is largely the discrepancy between the observed data and simulation results which is to be minimised. A standard least squares value is usually used to measure this discrepancy, which is referred to as the ‘misfit’ throughout this thesis.

Table 2.4: The sequence of adjusting model parameters and history matching

1. Match Pressure	*Where*	*How*
1.1. Global	Field	Adjust Pore volumes, Aquifer strength, permeabilities, fault transmissibility, WOC, Rock Compressibility (not adjusted if free gas available).
1.2. Regional	Flow units, layers groups, and individual layers	Adjust Lateral permeability, vertical transmissibility (Start with the deepest zones, bottom-up, in water-drive and top-down for free gas reservoirs).
1.3. Individual Wells	Well cell or surrounding cells	Change layer allocations (Well Conductivity).

2. Match Saturation	*Where*	*How*
2.1. Global	Field	**Water-cut** Only if all wells or flow-units are experiencing similar behaviour: change relative permeability, inter-sector connections, WOC, fault transmissibility, vertical transmissibility, and layer PI.
2.2. Regional	Flow units, layers groups, and individual layers	**Water-cut** (If different Water BT times seen from different zones): adjust rel-perms, layer or zone separation based on facies variation.
2.3. Individual Wells	Well cell or surrounding cells	**Water-cut** (adjust layers fluid allocations (PI) may ruin pressure match! Change inter-sector connections, WOC, relative permeability). **GOR** (Significant measurement inaccuracies and inclusion of gas-lift gas in reported gas).

Computer assisted history matching using probabilistic approaches has become part of today's Reservoir Engineering job. Nowadays, instead of a single deterministic realisation of the reservoir model, mostly referred as the base case, Reservoir Engineers look for a set of models above a certain level of fit quality to historical data, which represent reservoir's uncertain nature.

An example of a comparison between manual and assisted history matching is the comparative study of assisted and manual history matching of a large, mature reservoir carried out by Gruenwalder et al. (2007). Their results showed that assisted history matching results in as good as or better global matches for saturations and pressures

than manual history matching. Furthermore, assisted history matching gives a range of predictions which can be used to quantify reservoir uncertainty.

2.3.2.1 History matching in a Bayesian framework

As we search the parameter space, history matching is converted into an optimization problem, with the objective of minimising deviation from the historical data. History matching is an inverse problem, and just like any other inverse problem has non-unique solutions. The inverse problem is dealt with in a Bayesian framework.

The Bayesian framework is used in history matching to find models with maximum degree of likelihood to observed data. The ensemble of these models obtained in history matching allows us to quantify uncertainty in model parameters, and hence, model predictions. Figure 2.1 shows a general framework of history matching in a Bayesian framework.

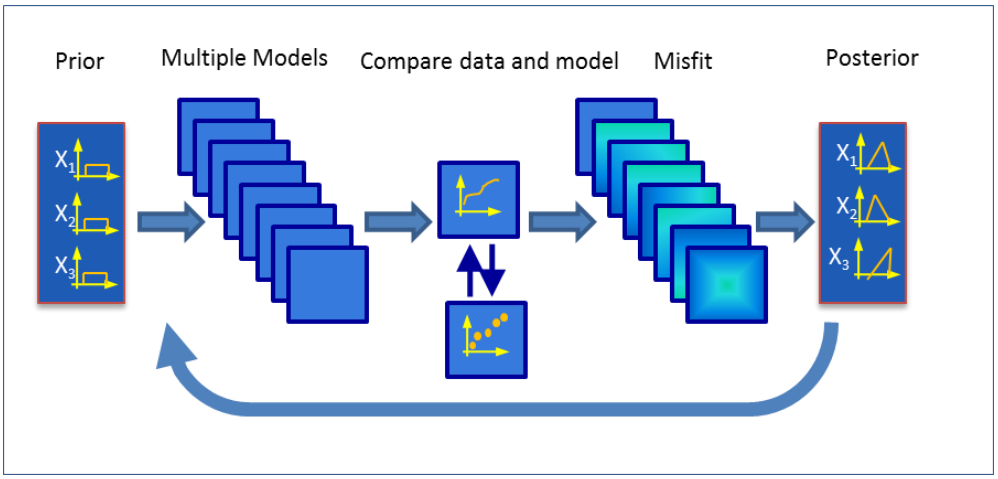


Figure 2.1: History matching in Bayesian inference.

The Bayesian solution of an inverse problem (e.g. history matching) will be the *posterior distribution* of model $\boldsymbol{M}$ conditioned on the observation data $\boldsymbol{D}$ which is $\boldsymbol{p(M|D)}$. The posterior distribution is expressed using the Bayes theorem:

$$p(M|D) = \frac{p(M).p(D|M)}{\int\big(p(M).p(D|M)\big).dM} \tag{2.1}$$

where the denominator is a normalizing factor, since a probability value $\boldsymbol{p(M|D)}$ should be between 0 and 1.

$\boldsymbol{p(M)}$ is the *prior* probability, which describes the uncertainty about the model $\boldsymbol{M}$, and it is represented with the model uncertain parameters and usually referred to as the *parameterization.*

$\boldsymbol{p(D|M)}$ is the *likelihood* of model $\boldsymbol{M}$, which tells us how likely the observation data $\boldsymbol{D}$ is to fit the model $\boldsymbol{M}$. The expression of the likelihood function depends on assumptions about modelling errors and measurement errors of the observation data. A challenge of Bayes theorem for uncertainty quantification is a proper definition of the likelihood function. A common expression for the likelihood function is equation (2.2), where underlying assumption is *normal distribution* for the errors.

$$\boldsymbol{p(D|M) = e^{-Misfit}} \tag{2.2}$$

where $\boldsymbol{Misfit}$ function is a measure of how the observation data fit the model response and is usually expressed by *Sum of Squares* or *Root Mean Square* functions.

2.3.2.2 Uncertainty Parameterization

Uncertainty is result of the reservoir heterogeneity and the scarcity and limited accuracy of measurements. The main sources of the uncertainty in the reservoir are the geometry of the reservoir, the spatial distribution of rock properties, and reservoir fluid.

One of the first studies on the use of parameterization was carried out by Coats et al. (1970), who studied the use of porosity and permeability as uncertainty parameters. They also separated the reservoir into regions of constant porosity and permeability parameters, to allow for variation around the reservoir. They observed a strong correlation between two parameters; therefore it was not necessary to estimate both parameters simultaneously.

2.3.2.3 Objective Function Definition

Coats et al. (1970) took a linear form of the objective function, which minimises the absolute difference between the observation and the simulation values, as follows:

$$Misfit = \sum_{i=1}^{N_p} |O_i - S_i| \tag{2.3}$$

where ***Misfit*** is the misfit function, N_p is the number of observations. O_i is the observation and S_i is the simulation data for point i.

Thomas et al. (1971) used a non-linear objective function definition, which improved the results for automatic history matching performed by Coates et al. (1970). Their objective function definition led to fewer costly simulation runs than the objective function used by Coats et al. (1970), for the same accuracy.

$$Misfit = \sum_{i=1}^{N_p} [W_i(O_i - S_i)]^2 \tag{2.4}$$

where W_i is the weight factor of observation point i. Weight factors are chosen to reflect the importance of each measurement in history matching. They are commonly set to unity.

The PUNQ project (Barker et al., 2001) used another form of the sum of squares, weighted for both the number of response variables and the number of observations for each response variable. They used the weighting factor to account for an unequal number of observations for response variables as follows.

$$\boldsymbol{Misfit = \frac{1}{N_V} \sum_{i=1}^{N_V} \frac{1}{N_P} \sum_{j=1}^{N_P} (W_{ij} \frac{O_{ij} - S_{ij}}{\sigma})^2} \tag{2.5}$$

where $\boldsymbol{i}$ is the subscript running over the model response variables, $\boldsymbol{j}$ is the subscript over observation data points at a reported time, $\boldsymbol{N_V}$ is the number of response variables, $\boldsymbol{N_P}$ is the number of observation data points, $\boldsymbol{O_{ij}}$ is the observed value of the response variable $\boldsymbol{i}$ at time $\boldsymbol{j}$. $\boldsymbol{S_{ij}}$ is simulated value of the response variable $\boldsymbol{i}$ at time $\boldsymbol{j}$. $\boldsymbol{\sigma_{ij}}$ is the standard deviation of errors for observed value of the response variable $\boldsymbol{i}$ at time $\boldsymbol{j}$. $\boldsymbol{W_{ij}}$ is an importance weight factor applied to the response variable $\boldsymbol{i}$ at time $\boldsymbol{j}$.

2.3.2.4 Algorithm selection

The field of optimisation has grown at an astonishing rate in recent years. This has happened with new developments in theory, algorithms, and the computational contributions of computer hardware to solve various problems in engineering and science.

A wide range of assisted history matching models come based on the abundance of optimisation techniques and tools that have recently been applied in reservoir history matching. The choice of algorithms for automatic history matching is wide.

Broadly speaking, they can be categorised as local or global algorithms. Local optimisation algorithms search and find a local optimum, but since most of the optimisation problems, as well as history matching, have more than one local optimum and local optimisation algorithms find a local optimum without a guarantee that this optimum is a global one. However, global optimisation algorithms, always reach the global optimal of the objective function if continued long enough.

If categorised based on the mechanism of the search, optimisation algorithms used in history matching fall into one of three groups: deterministic methods, stochastic methods, and data assimilation.

2.3.2.4.1 Deterministic methods

Deterministic methods take advantage of the analytical properties of the problem to solve the optimisation problem. They are also called local optimization methods, since they always reach a local optimal of the objective function. Deterministic methods are divided into two subgroups: gradient-based and sensitivity-based methods.

Gradient-based methods were the earliest deterministic optimisation methods used in automatic history matching. They are based on calculating the derivative of objective functions with respect to the model matching parameters in order to minimise the objective function, which is some measure of the discrepancy between observed and simulated points (Jahns 1966; Coats et al., 1970; Thomas et al., 1971). Gradient Algorithms are fast and efficient, but in some cases, these methods might be trapped in

local minima or may not converge at all. More disappointingly, they find only one single good solution, rather than a range of the good solutions.

Sensitivity-based methods, such as Gauss-Newton and Sparse Equations and Least Squares (LSQR), are another type of deterministic methods. In these methods, sensitivity coefficients are computed, which are simply partial derivatives that define the change in production response due to small changes in reservoir parameters. The sensitivities define the relationship between reservoir properties and production response. For a review of sensitivity-based optimisation methods used in history matching, refer to Oliver et al. (2008).

For calculation of sensitivity coefficients one of the following four methods is usually used (Oliver et al., 2008):

- The Perturbation method: This is a deterministic method in which sensitivities are estimated simply by perturbing the model parameters one at a time by a small amount and then computing the corresponding production response. This requires *N+1* forward simulations for N parameters.
- Direct Methods: In these methods, the flow and transport are differentiated to obtain expressions for the sensitivity coefficients. This involves only one equation per parameter.
- Gradient Simulator Method: This is a variation of the Direct Method that uses the discretized version of the flow equation. The coefficient matrix will not change for all parameters and will need to be decomposed only once; thus, a matrix/vector multiplication is required for each parameter to compute sensitivity. For a large number of parameters, this method could be computationally expensive.
- Adjoint-State Method: This method requires derivation and solution of adjoint equations, which for a multi-phase flow equation could be quite hard. In addition to the number of phases, adjoint solutions depend on the amount of production data.

Sensitivity-based methods are hard to perform by multi-phase flow simulators. However, streamline simulators are efficient tools for computing parameters' sensitivity coefficients by single flow simulation (Thiele et al., 2010).

2.3.2.4.2 Stochastic Methods

Stochastic methods are based on gradient-free algorithms; their convergence rate is usually slower than that of gradient methods, although their implementation may be much easier. Such algorithms incorporate a random component and, by allowing the search to move towards worse solutions occasionally, gain the ability to seek out the global optimum.

There are many stochastic algorithms that have been used for history matching in reservoir engineering. Stochastic algorithms vary in sophistication and complexity, ranging from random search to hill-climbing and population-based evolutionary algorithms.

In the last two decades, many population-based stochastic algorithms have been employed in history matching and uncertainty quantification problems. These methods provide a flexible framework, in which exploration of the search space for a diverse set of solutions, followed by local search in previously found regions (exploitation) results in an effective search. Simulated Annealing (Sultan et al., 1994), Scatter Search (April et al., 2003), Tabu Search (Yang et al,. 2007), Genetic Algorithm (Romero et al., 2000; Erbas & Christie, 2007), Neighbourhood Algorithm (Subbey et al., 2003), Evolutionary Strategy (Schulze-Riegert et al., 2001), Population-Based Incremental Learning (PBIL) (Petrovska, 2009), Particle Swarm Optimization (Mohamed et al., 2010), Differential Evolution (Hajizadeh et al., 2009) and Ant-Colony Optimization (Hajizadeh, 2010) are among the stochastic optimization algorithms that have been applied to history matching and uncertainty quantification problems.

Experimental Design with Response Surface modelling is another stochastic method used for parameter estimation and propagating the uncertainties. Examples of this method in the literature are by Damsleth et al., (1994) and Manceau et al., (2001).

2.3.2.4.3 Data assimilation methods

The final type of assisted history matching algorithm is data assimilation. In data assimilation methods, a sequential calibration of the model parameters to observation data is done in a series over the time. Ensemble Kalman Filter (EnKF) is a well-known data assimilation method applied to the history matching problem (Evensen et al., 2007). Many combinations of the gradient methods, evolutionary algorithms, and data assimilation methods have also been used in history matching (Schulze-Riegert et al., 2009; Mantica et al., 2002; Gómez et al., 2000).

EnKF has a simple formulation and can be easily implemented to history match problems. Liu and Oliver (2005) showed that EnKF is very efficient and robust compared to gradient-based methods. However it has two problems of overshooting and filter divergence, especially if initial ensemble members do not represent the true field well (Nævdal et al. 2005).

2.4 Evolutionary Algorithms

Evolutionary algorithm is an umbrella term used to describe computer-based computational algorithms which are inspired by biological process of evolution and use key evolutionary elements in their design and implementation.

Evolutionary algorithms (EAs), as defined by Michalewicz (1996), are a class of stochastic search algorithms that simultaneously navigate several regions in the search space, and explore the search space stochastically to avoid getting trapped in local minima, as many other search algorithms do. EAs do not make any assumption about the underlying fitness landscape in optimisation problems; thus, they are typically used to provide good approximate solutions to problems that cannot be solved easily using other techniques.

EAs maintain a population of structures, create an implicit distribution model base on the rule of selection, then evolve and generate new candidate solutions using created model defined by one or more variation operators, such as crossover and mutation in genetic algorithms.

EAs are implemented by control parameters which determine different components of the algorithm. To define a particular EA, a set of components must be specified. In the following sections, we briefly describe the most important components of EAs.

2.4.1 Solution representation

In order to solve an optimization problem using an EA, objects, usually in the form of vectors which form possible solutions to an optimisation problem, are referred to as *phenotypes*, while their encoding, i.e. the individual solution within the EA, is called the *genotype*. Thus, the genotype is an individual's genetic coding, which is represented in different forms, while the phenotype is the behavioural manifestation or response to that genetic information.

In history matching and uncertainty quantification, the phenotype is usually a vector of the variables and the genotypes are the bit string or a range of bins for each variable. The link between the problem and the algorithm where optimization will take place is created by a mapping from the phenotype to a set of genotypes, which is called *representation*. Traditional bit strings (binary) representation may suit some uncertainty parameters (optimization variables). However, permutation vectors and some other complex data structures may be required for other types of of uncertainty parameters. Choosing an appropriate solution representation is particularly important and sensitive issue and can have a massive impact on the performance of the algorithm (Eiben and Smith, 2003).

2.4.2 Population

The evolution process on EAs is done on an individual or a population. A population is a set of possible candidate solutions which forms the unit of evaluation in any EA by rules of changing and adapting. Population is a unit of evolution in any evolutionary algorithm and EAs involving evaluation of population called Population-based EAs (PBEAs).

Population size is a critical control parameter in any evolutionary algorithm. A large population size implies a slow convergence and vice-versa. In most evolutionary

algorithms, population size is determined heuristically and is kept constant throughout the evolution process.

The diversity of the population is a measure of the number of different solutions present; here different means dissimilarity of solutions, which for example can be defined using distance measures. A good evolutionary algorithm should keep a balance between diversity of population and convergence towards the global optimal point.

2.4.3 Population initialisation

Population initialization is an essential part of any evolutionary algorithm because it can directly affect the convergence behaviour of the algorithm and the quality of the obtained solutions. If no information about the solution is available, which is the case in many optimisation algorithms, a random initialisation is used to generate new solutions in the initial population.

2.4.4 Evaluation function

The evaluation function is used as the criteria for evaluating the candidate solutions. Thus, it is the basis for selection and a requirement for adaptation in the algorithm. The evaluation function is also called the *fitness function* in EA or the *objective function* in optimization problems.

In history matching and uncertainty quantification problems, the terms *misfit* or *mismatch* are used interchangeably for the objective function and usually expressed as the sum of the weighted square root of the difference between observation and simulation data.

2.4.5 Selection mechanism

Selection is carried out to distinguish better solutions in the population based on their quality, represented by the objective function value. The selected solutions become the parents of the next generation to undergo the variation mechanism. The selection mechanism, together with a replacement strategy, ensures quality improvement in EAs. Selection is made based on a single objective function or multiple objective functions.

Promising solutions get a higher chance, while less promising solutions still get a positive but small chance to be selected as the parents of the next generation.

2.4.6 Variation mechanism

The variation mechanism is the main characteristic of an EA. It is performed by variation operators, by which new solutions are created from the selected solutions. Usually EAs are known and distinguishable by the type of variation mechanism they use in their evaluation process. *Mutation* and *crossover* are two well-known evolutionary variation operators used in genetic algorithms, the most common class of EAs. Estimation of distribution algorithms is another class of EAs that uses a probabilistic model for variation instead of crossover and mutation operators. This enables them to adapt their operators to the structure of the problem. In the next chapters, estimation of distribution algorithms are explained more in detail.

2.4.7 Replacement strategies

To ensure convergence and maintain a certain level of diversity in the population throughout the evolution, replacement policies are used in EAs. Therefore, we replace parents, or a certain number of solutions in the entire population, by the children in each generation. Elitism and diversity are two main concepts based on which the different replacement strategies are designed.

Elitism is a concept that allows replacement of the solution in the population only with the children that are superior in terms of fitness function. In addition, duplication is not allowed and duplicate solutions are deleted from the population. There are different replacement strategies in the literature (e.g. Yu & Gen, 2010), which each use a different degree of elitism. The following are the most widely used replacement strategies:

2.4.7.1 Truncation replacement

In this type of replacement, the entire existing population is replaced by the best set among the current population and children. The size of the existing population is

maintained. In all the algorithms used in the current thesis, truncation replacement is used.

2.4.7.2 Steady-state replacement

In steady-state replacement, the children replace the least fit solutions in the existing population. This replacement is done even if those children are less fit than the solutions that they replace.

2.4.7.3 Generational replacement

Generational replacement policy replaces the entire existing population by the children. If the number of children is larger than the size of the existing population, children are truncated to the population size. A weak elitism may also be allowed, in which the best *n* solutions in the current population are allowed to survive if they are better than the worst solution in the children.

2.4.7.4 Random replacement

In random replacement, the children replace random members of the population, keeping the population size constant. A weak elitism is used, in which the best *n* solutions in the current population are maintained if they are better than the worst solution in the children.

2.4.7.5 Comma replacement

Comma replacement is another possible way of replacement, which means the entire existing population is replaced by a new population from the best solutions of the children. The number of children should be at least as large as the original population, otherwise, the population size will not be constant.

2.4.8 Stopping criteria

A stopping or termination criterion determines a stage in the evolution where the algorithm must end the search process. Commonly used criteria are a maximum allowed CPU time, a predefined number of generations or evaluation functions, a desired level

of the objective function value, a convergence status depicted by fitness improvement below a threshold, or when population diversity falls below a given threshold.

If the problem has a known global optimum, reaching this global optimum with a given precision ($\varepsilon>0$) can be defined as the stopping criterion.

2.5 Uncertainty Quantification

After building a reservoir simulation model and calibrating the model to historical data, the third stage of a reservoir performance study is uncertainty quantification using an ensemble of history-matched models.

The range of acceptably history-matched reservoir simulation models reflects the non-unique nature of the history matching process, which leads to uncertainty in production prediction, which is, in turn, a critical factor in decision making in the hydrocarbon production business. It is crucial not only to have an acceptable history-matched reservoir simulation model, but also to perform quantification of uncertainty associated with that model when it is used for predicting the performance of the reservoir.

Uncertainly in reservoir simulation comes from the lack knowledge about the reservoir. Three sources of uncertainty are the lack of data, error in measurement of input data, and complexity of subsurface system and resolution of the model, we build. Uncertainty cannot be removed but can be measured and quantified which contributes to better decision making. Uncertainty quantification is to quantify the uncertainties associated with reservoir performance simulation, where the term reservoir performance is defined as oil and gas production rates, gas-oil ratio, water-oil ratio, and cumulative oil production.

Since the knowledge and information we have on the subsurface, in the form of field data, is uncertain, decision-making in reservoir management must be performed under a state of uncertainty. According to decision theory, in such a situation, a set of primitive outcomes with probability of occurrence give an expected value that can be used for optimal decision-making. Reservoir uncertainty quantification is an essential part of

reservoir management in which, history-matched reservoir models are used for production optimization and forecasting incorporated with estimation of the uncertainty.

In order to visualize and quantitatively assess the impact of subsurface uncertainties, in reservoir history matching, uncertainty parameters need to be explored and multiple realizations in a Bayesian framework need to be created. Multiple acceptably history-matched models allow us to account for uncertainty in the model parameters, and assess the uncertainty in prediction.

2.5.1 A review of uncertainty quantification methods

A single history-matched model is insufficient to be used as the basis for key decision making in reservoir planning and management. It does not allow accounting for uncertainty and estimation of risk. Tavasolli et al. (2004) showed that a single best history-matched model is not necessarily a good predictor model for future performance of the reservoir.

The third stage of a reservoir performance study is uncertainty quantification, using an ensemble of history-matched models. If correctly sampled, the ensemble of realisations obtained in history matching represents the posterior distribution, which provides an assessment of the uncertainty of the model parameters. Moreover, using these realisations for predicting the future performance of the reservoir, the uncertainty in the prediction can be obtained, which is usually shown using statistical properties such as the cumulative distribution function (CDF) or the credible intervals (i.e. P10, P50, P90) (Figure 2.2).

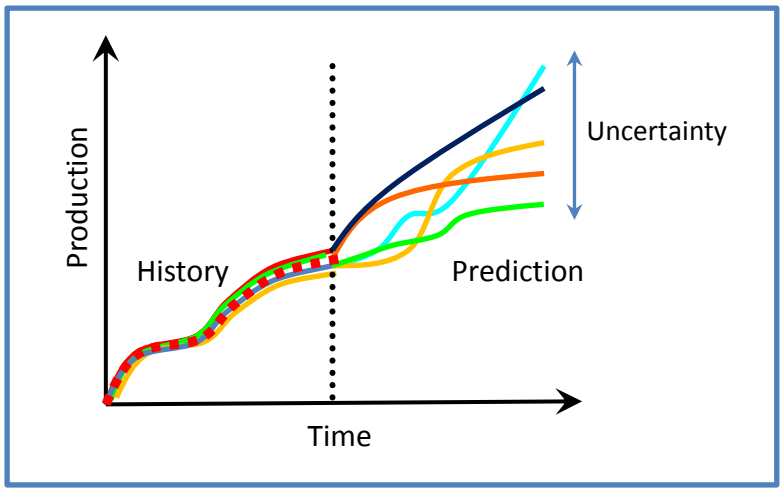


Figure 2.2: History matching and uncertainty quantification; red squares show observation data in history phase, each coloured line shows a possible fitting model. The history matched models are used for making predictions and in the prediction phase and their results show uncertainty bounds.

In last two decades, many techniques have been introduced for estimating posterior distribution, hence quantifying the uncertainty from an ensemble of models. These techniques can be classified as approximate or Monte Carlo sampling.

In the approximate techniques, we try to approximate the posterior distribution with a limited number of simulations or objective function calculations. However, the credible interval obtained by approximate techniques may be too narrow and unrealistic (Baker & Cuypers 2000).

Monte Carlo Sampling techniques, in which sampling of the parameter search space is done using Monte Carlo simulation, are particularly common. This would requires numerous forward simulation runs, which are computationally expensive. To work around this problem, proxy models instead of forward simulations can be used, but they may introduce significant modelling errors.

Design of Experiments and Response Surface Methods are also widely used for uncertainty quantification (e.g. see Damsleth et al., 1994, Manceau et al., 2001, and Montgomery, 2001). These are statistical approaches, first to identify uncertainty parameters that most affect the response variables in a minimum number of simulations and secondly, to fit a surface, usually a linear or quadratic model, to the response variables, then use it as a proxy for the simulation runs when it is used in Monte Carlo sampling.

The PUNQ project (Floris et al., 1999) proposed several approximate and sampling methods for quantifying uncertainty. The first class of these methods is the Maximum Likelihood solution plus local characterisation of the likelihood function (ML+) or Maximum *A Posteriori* solution, plus local characterisation of the objective function (MAP+) when the prior term is included (Roggero, 1997). Another class is Multi-ML and Multi-MAP, depending on the objective function used. In these methods, it is assumed that the objective function is truly multimodal and different history-matched models located at the distinct optima show the uncertainty. Multi-ML+ and Multi-MAP+ form another class and are a combination of the two previous classes, i.e. local characterisation of the objective function around distinct local optima is used. Floris et al., (1999) also present the results of Monte Carlo sampling of the full posterior distribution. The final approach in the PUNQ project is that of Oliver et al. (1996) which aims to estimate the complete posterior using an optimisation technique to sample both the prior and the observation data, and hence, to reduce the number of simulation runs needed.

Another sampling method is the Markov Chain Monte Carlo (MCMC) which can be used to estimate the posterior distribution from the ensemble of models obtained in history matching. The challenge here is that the shape of the posterior distribution is very highly dimensional, non-Gaussian and multimodal. This makes the MCMC algorithm inefficient and ineffective if not implemented correctly.

Subbey et al. (2003) used NA-Bayes (NAB) introduced by Sambridge (1999) for reservoir uncertainty quantification. NAB is a Gibbs sampler which samples full parameter search space by drawing random proposal samples from the marginal likelihood probability and accepting new samples with the probability of Bayes factor (likelihood ratio of proposed to the current sample). Thus, the sampling density is proportional to the likelihood probability multiplied by the volume of the cell approximated by the sample. In NAB, instead of running forward simulations for each sample, the likelihood function is approximated using nearest neighbour in the existing models of the ensemble.

Christie et al. (2006) proposed a method to improve the resolution of ensemble by using an Artificial Neural Network (ANN) interpolant. In their method, the ensemble is extended with new models for which the likelihood is interpolated from the models with forward simulation using ANN.

Ma et al. (2008) used a two-stage MCMC for uncertainty quantification in history matching. Their approach, in the first stage, uses a fast streamline-based approximation of the dynamic data for the MCMC candidates. In the second stage, only those proposals that pass the first stage are assessed by full physics simulation. Thus, they claimed significant improvement of the computational cost without loss of accuracy compared to MCMC using full flow simulations.

Mohamed et al. (2010a) used Hamiltonian Monte Carlo (HMC) for sampling the posterior probability over the uncertain parameters. HMC is an MCMC method that makes use of gradient information to speed up random walk in a continuous space.

Polynomial chaos expansion (PCE) has been used for uncertainty quantification (Ghanem & Spanos, 1991). It is a non-sampling based method to determine evolution of uncertainty in a model, when there is probabilistic uncertainty in the model parameters.

2.5.2 Bayesian inference

An ensemble of n models (M_j; $j=1, \ldots, n$) with their corresponding misfits are the starting point for many uncertainty quantification techniques. For a reservoir model, M, with d uncertain input parameters (X_i; $i=1, \ldots, d$), uncertainty can be expressed as a probability distribution. Bayesian inference with Monte Carlo sampling provides a general framework for the uncertainty quantification of reservoir models.

In history matching, different realisations of the reservoir model will be likely to have different fitness values. This makes the use of Bayesian integrals necessary, as models in the ensemble are not equally probable. The Bayesian solution of the inverse problem will be the posterior distribution of model, M, conditioned on the observation data, D, which is $p(M|D)$. The posterior distribution is expressed using the Bayes theorem as:

$$p(M|D) = \frac{p(M)p(D|M)}{\int p(M)p(D|M)dM} \tag{2.6}$$

where the denominator is a normalizing factor, as the probability value $\boldsymbol{p(M|D)}$ should be between 0 and 1. $\boldsymbol{p(M)}$ is the prior probability which describes the uncertainty about the model $\boldsymbol{M}$, and it is represented with the model uncertain parameters and usually referred to as the parameterization. $\boldsymbol{p(D|M)}$ is the likelihood of the model $\boldsymbol{M}$, which tells us how likely it is that the observation data $\boldsymbol{D}$ fit the model $\boldsymbol{M}$.

Any available prior information about the model parameters should be included in the prior distribution of the parameters for history matching and uncertainty quantification. It is common to update reservoir simulation models based on more current data, and the updated model is similar to a previous model; an informative prior can become the present model; hence, the posterior distribution of parameters from the previous model may be used as the prior distribution of parameters for the present model.

The likelihood function or likelihood contains the available information provided by a sample of the model when conditioned to data. The expression of the likelihood function depends on assumptions about modelling errors and measurement errors of the observation data. A challenge of history matching and uncertainty quantification in Bayesian inference is the proper definition of the likelihood function. If Gaussian errors are assumed for the data, the measure of model fitness (*Misfit*) is related to likelihood, as follows:

$$\boldsymbol{p(D|M) = e^{-Misfit}} \tag{2.7}$$

The demonstration of this equation is given in the following paragraphs. Reservoir simulation models always include an error term which is assumed to be independent of model, *M*. So the true (theoretical) data can be expressed by this error term and an approximate model asfollows:

$$\boldsymbol{d = g(M) + \gamma} \tag{2.8}$$

where $\boldsymbol{d}$ is the true data of model, $\boldsymbol{g(M)}$ is approximate model, and $\boldsymbol{\gamma}$ is the modelling error.

On the other hand, production observation data, $\boldsymbol{D}$, is a component of data, $\boldsymbol{d}$, that can be measured and usually there is an error $\boldsymbol{\epsilon}$ associated with the measurement $\boldsymbol{D}$, so that:

$$\boldsymbol{D = d + \epsilon} \tag{2.9}$$

From (2.8) and (2.9), one can obtain a general error term $\boldsymbol{\epsilon}$ for observation data of $\boldsymbol{D}$ and approximate model $\boldsymbol{g(M)}$:

$$\begin{cases} \boldsymbol{e = \gamma + \epsilon} \\ \boldsymbol{D = g(M) + e} \end{cases} \tag{2.10}$$

It is usually assumed that $\boldsymbol{\gamma}$ and $\boldsymbol{\epsilon}$ are independent and Gaussian, so that the compound error term of modelling and measurement is expressed with mean zero and covariance matrix of $\boldsymbol{\Sigma}$:

$$\boldsymbol{\Sigma = \Sigma_o + \Sigma_m} \tag{2.11}$$

where $\boldsymbol{\Sigma_o}$ is the covariance of measurement errors, $\boldsymbol{\Sigma_m}$ is the covariance of modelling error.

Thus for a given model $\boldsymbol{M}$, $\boldsymbol{D}$ is a random vector with mean vector $\boldsymbol{g(M)}$ and covariance matrix $\boldsymbol{\Sigma}$ and its conditional probability density function (PDF) can be expressed as follows (Oliver et al., 2008):

$$\mathbf{f}(\boldsymbol{D}|\boldsymbol{M}) = \frac{\mathbf{1}}{\sqrt{(\mathbf{2\pi})^{\mathbf{N}}}\sqrt{|\boldsymbol{\Sigma}|}} \mathbf{exp}[-\frac{\mathbf{1}}{\mathbf{2}}\big(\boldsymbol{D - g(M)}\big)^{\mathbf{T}}\boldsymbol{\Sigma}^{-\mathbf{1}}\big(\boldsymbol{D - g(M)}\big)] \tag{2.12}$$

The maximum likelihood estimate is defined to be model $\boldsymbol{M}$, which maximises the likelihood function in (2.12). It is usually more convenient to work with the log-likelihood rather than likelihood itself and also to work with a minimisation problem instead of maximization, so the maximum estimate of (2.12) can be converted to minimisation of the following objective function:

$$\boldsymbol{Misfit = -log(p(D|M)) = \frac{1}{2}\big(D - g(M)\big)^{T}\Sigma^{-1}(D - g(M))} \quad (2.13)$$

where $\boldsymbol{Misfit}$ function is a measure of how the observation data fit the model response. If one assumes that components of error $\boldsymbol{e}$ in the equation (2.10) are independent and follow a random Gaussian distribution, the Misfit can be expressed as:

$$\boldsymbol{Misfit_k = \frac{1}{2}\sum_{i=1}^{N_k}\left(\frac{O_{ki} - S_{ki}}{\sigma_k}\right)^2} \quad (2.14)$$

where $\boldsymbol{Misfit_k}$ is the $\boldsymbol{k}$-th element of misfit (objective function), $\boldsymbol{i}$ *is* subscript running over observations, $\boldsymbol{N_k}$ is the number of observation for each element (dimension) of $\boldsymbol{k}$, $\boldsymbol{O_{ki}}$ *is* the observed value of response element $\boldsymbol{k}$ at observation $\boldsymbol{i}$. $\boldsymbol{S_{ki}}$ *is* simulated value of response element $\boldsymbol{k}$ at observation $\boldsymbol{i}$. $\boldsymbol{\sigma_k}$ is the standard deviation of errors for observed value of response element $\boldsymbol{k}$ at observation $\boldsymbol{i}$. This equation is a form of weighted least square estimate.

2.5.3 Bayesian model averaging

Bayesian model averaging (BMA) is an approach to summarise the uncertainty in the model after observing the data. Leamer (1978) introduced the basic paradigm for BMA and pointed out that the only way that his implementation of BMA accounts for the uncertainty is through its model selection procedure. Madigan & Raftery (1995) reviewed BMA and proposed a way of introducing uncertainties into BMA.

The posterior probability can be used simply to select the best model for prediction, which is usually the one with highest posterior probability or, in case of uniform priors, the one with the highest likelihood probability (maximum-likelihood prediction).

However, if the posterior mass is not strongly concentrated on a particular model, the uncertainty in the model is considerable, and it would not be wise to leave out all other models. In such situations, BMA helps to quantify uncertainty by mixing it over models, using the posterior probabilities as weights. If a uniform set of sampled models is available, the expected value of a model's predictive quantity (Δ) can be obtained by mixing the inference from the individual models, i.e.

$$E(\Delta|D) = \sum_{j=1}^{N} p(M_j|D)\mathrm{E}(\Delta|M_j, D) \tag{2.15}$$

$E(\Delta|D)$ represents the weighted expected value of Δ across all the models uniformly sampled and supported by the observation data D, to some extent. Subscript j iterates over number of models, N. $p(M_j|D)$ is the posterior probability of model M_j when conditioned to data D, and $\mathrm{E}(\Delta|M_j, D)$ represents the expected value of Δ for model M_j.

With uniform prior probabilities in equation (2.6), equation (2.15) becomes:

$$E(\Delta|D) = \frac{1}{N\sum_{j=1}^{N} p(D|M_j)} \sum_{j=1}^{N} p(D|M_j)\mathrm{E}(\Delta|M_j, D) \tag{2.16}$$

where $p(D|M_j)$ represents the likelihood of model M_j.

2.5.4 Monte Carlo integration and uniform sampling

The model space in Bayesian inference can be extremely large, which makes the summation in equation (2.15) an exhaustive computation. Monte Carlo methods can be used to deal with very large model spaces.

A formal definition of Monte Carlo methods was given by Halton (1970). He defined a Monte Carlo method as "representing the solution of a problem as a parameter of a hypothetical population, and using a random sequence of numbers to construct a sample of the population, from which statistical estimates of the parameter can be obtained".

An example of Monte Carlo methods includes Monte Carlo integration, which is a technique for numerical integration using random numbers, particularly for higher dimensions. In uncertainty quantification, it is extensively used to generate samples from a given probability distribution and then to estimate expectations of functions under this distribution.

One can randomly choose a large number of independent and identically distributed (i.i.d.) model realisations from the prior distribution and do forward simulation of each model. In the unlikely event that all of the sampled realisations are equally probable

(i.e. all samples have the same fitness value) and a uniform sampling is done over the parameter space, one can obtain predictive distribution of the reservoir model by averaging predictions from all the sampled models (Kalos & Whitlock 1986).

$$dE(f) = \frac{1}{N}\sum_{j=1}^{N} f_j \tag{2.17}$$

where $\boldsymbol{E(f)}$ is an estimate of predicted value $\boldsymbol{f}$, $\boldsymbol{N}$ is the number of independent sampled models (a large number), $\boldsymbol{f_j}$ is the predicted value of sample $\boldsymbol{j}$.

The main drawback of the Monte Carlo methods in general was that they used to be extremely expensive to carry out as they involve lots of sampling and simulations. With the advance of digital computers, this has been changed dramatically and many physical and statistical random experiments which were difficult to perform have now become routine methods.

2.5.5 Acceptance-Rejection Monte Carlo sampling

Classical simulation techniques generate independent samples; hence, the successive observations can be said to be statistically independent. Therefore, they can be used in a Monte Carlo sampling for integration purposes. von Neumann (1951) pioneered an Acceptance-Rejection method based on the region under the density graph of a distribution.

Acceptance-Rejection Monte Carlo (ARMC) sampling is a method which generates samples from an arbitrary probability distribution function $f(x)$ by using an instrumental distribution function $g(x)$ which is easier to sample from, so that $f(x) \leq cg(x)$ where $\boldsymbol{c}$ is a known constant where $c > 1$. The algorithm for implementing ARMC sampling is as follows:

1) Generate a candidate sample from $g(x)$ and draw normal random deviate u from the standard normal distribution $\mathcal{N}(0,1)$.

2) Check if $u < f(x)/[cg(x)]$:

a. If True accept x as a realisation of $f(x)$,

b. Else reject x and repeat sampling from the step 1.

An illustration of the ARMC method is shown in Figure 2.3. The *acceptance ratio* is the proportion of proposed samples accepted. Hence, if $\boldsymbol{c}$ is low, fewer samples are rejected, and the required number of samples for the target distribution $f(x)$ is obtained more quickly and sampling becomes more efficient.

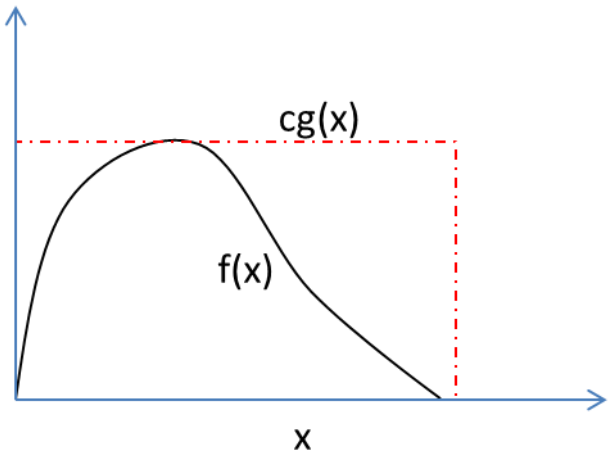


Figure 2.3: Illustration of the ARMC sampling method. Random points are chosen inside the dash-dot rectangle, *cg(x)*, and rejected if they exceed the target distribution, *f(x)*.

2.5.6 Markov Chain Monte Carlo sampling

Ordinary Monte Carlo is very powerful method for approximating the distributions and hence quantifying the uncertainty in predictions. However, it is limited to simulating independent realisations of high-dimensional model parameters. Markov chain Monte Carlo (MCMC) is a general methodology and a class of algorithms for sampling from a distribution that enables us to sample not only independent realisations but a Markov chain $\boldsymbol{X_1, X_2, \ldots}$with transition probabilities and a stationary distribution.

MCMC sampling in conjunction with the Bayesian inference enables us to create a random walk that has $\boldsymbol{p(M|D)}$ as its stationary distribution. The random walk should be long enough so that the resulting samples closely approximate samples from $\boldsymbol{p(M|D)}$. The quality of the samples improves as the number of random steps in the walk

increases. These samples can be used directly for parameter inference and prediction. The Monte Carlo integration estimate for a future prediction $\boldsymbol{f}$ becomes:

$$E(f|D) \approx \frac{1}{N}\sum_{j=1}^{N} f_j \tag{2.18}$$

The sample size, $\boldsymbol{N}$, should be large enough. Subscript j iterates over drawn samples.

There are numerous MCMC algorithms. In this work, we consider two main types of MCMC, Metropolis-Hastings and Gibbs sampling algorithms. The general feature of these algorithms is to move around the stationary distribution in random walks. Unfortunately, it can take a long time to explore a high-dimensional parameter space, especially when parameters are correlated. There is no guarantee that the walker will not search already explored regions.

Another problem of MCMC methods is that samples from the beginning of the chain, what is called the burn-in period, may not accurately represent the desired distribution. There is always some effect of the starting point of the random walk. This is why the samples taken in the burn-in period are not going to be counted towards the final set of samples and they must be thrown away. In addition, sampling using multiple walks from different starting points is a good strategy to minimise this effect.

2.5.6.1 Metropolis-Hastings algorithm

The Metropolis-Hastings algorithm (M-H) is a MCMC method to generate a sequence of random samples from a probability distribution through performing a random walk in parameter space and using a proposal density and a method for rejecting/accepting proposed moves. The M-H method was first introduced by Metropolis et al. (1953), and then generalised by Hastings (1970). For a review and introduction of the M-H method, see Chib and Greenberg (1995).

In the M-H algorithm, instead of an instrumental distribution function in the ARMC method, a Markov transition density matrix q is used to go from state x to y. To sample from a probability distribution function $f(x)$, it needs to be initialised with

some X_0 for which $f(X_0) > 0$. Then for each state $t = 0, 1, 2, \dots, T-1$ one can execute following steps:

1) At the current state X_t, generate $Y \sim q(y|X_t)$,

2) Generate a normal random deviate $U \sim \mathcal{N}(0,1)$ and then calculate:

$$X_{t+1} = \begin{cases} Y & if\ U \leq \alpha(X_t, Y) \\ X_t & otherwise \end{cases} \tag{2.19}$$

where $\alpha(x,y) = min\left\{1, \frac{f(y)q(x|y)}{f(x)q(y|x)}\right\}$ is called the acceptance probability.

Thus, one can obtain a M-H Markov chain of $X_0, X_1, X_2, \dots, X_T$ in which X_T is approximately distributed according to $f(x)$, for large T.

If the proposal function $q(y|x)$ does not depend on x, i.e. $q(y|x) = q(y)$, the acceptance probability becomes $\alpha(x,y) = min\left\{1, \frac{f(y)q(x)}{f(x)q(y)}\right\}$, and hence the algorithm here referred to as *independence sampler*. The independence sampler is similar to the ARMC method, but, differently, it generates dependent samples.

The original algorithm was introduced (Metropolis et al., 1953) for symmetric proposal functions, i.e. $q(x|y) = q(y|x)$ and $\alpha(x,y) = min\left\{1, \frac{f(y)}{f(x)}\right\}$. The algorithm with symmetric proposal functions is referred as *random walk sampler*. Hastings (1970) modified the algorithm for non-symmetric proposal functions, hence the algorithm is named the Metropolis-Hastings algorithm.

2.5.6.2 Gibbs sampling

Gibbs sampling is another common MCMC method. It is a particular case of the Metropolis-Hastings algorithm in which conditional distributions of multivariate distribution are sampled instead of joint probability distribution. This is useful when direct sampling of the joint distribution is difficult. In addition, the acceptance probability is always 1.0, so that all the proposed samples are accepted.

Gibbs sampling is a popular method because it has no tuning parameters, but it is restricted to sampling problems, for which, samples can be generated from the conditional distributions (e.g. marginal probabilities).

In the general case of a system with ***D*** parameters, a single iteration involves sampling one parameter at a time from the marginal probability distribution $P(x_1, x_2, \dots, x_D)$. The algorithm for the Gibbs sampler becomes as follows:

1) Initialise P at $t = 0$, i.e. $P(x_1^{(0)}, x_2^{(0)}, \dots, x_D^{(0)})$.
2) At current time t, generate D samples as follows:
    - draw $x_1^{(t+1)}$ from $P(x_1 | x_2^{(t)}, x_3^{(t)}, \dots, x_D^{(t)})$,
    - draw $x_2^{(t+1)}$ from $P(x_2 | x_1^{(t+1)}, x_3^{(t)}, \dots, x_D^{(t)})$,
    - draw $x_3^{(t+1)}$ from $P(x_3 | x_1^{(t+1)}, x_2^{(t+1)}, \dots, x_D^{(t)})$,
    - ...
    - draw $x_D^{(t+1)}$ from $P(x_D | x_1^{(t+1)}, x_2^{(t+1)}, \dots, x_{D-1}^{(t+1)})$
3) Set $t = t + 1$ and continue from the step 2, until the number of desired samples is achieved.

The algorithm was first introduced and used by Geman and Geman (1984) and they named it the Gibbs sampler. The first application of the algorithm for solving problems in Bayesian inference was carried out by Gelfand et al. (1990).

2.6 Clustering

Throughout this thesis, we refer to a model as a realisation of a reservoir model with a set of uncertainty parameters. Later in this thesis, we use clustering techniques to group a set of models in such a way that models in the same group are more similar.

Classification and clustering are two types of machine learning techniques. Unlike data classification, data clustering is an unsupervised learning technique, i.e. labelled data sets are not required as training data, and generally the performance of data clustering is

often poorer than data classification. A labelled data set is often difficult, expensive, and sometimes (e.g. in reservoir uncertainty quantification) impossible to obtain.

In many machine learning problems, there is little prior information available about the data. This is the case in uncertainty quantification, where one cannot easily obtain statistical models of the data. In such situations, the decision-makers make few assumptions about the data (e.g. underlying distribution of the data). Clustering is an appropriate technique for exploration of possible relationships between the data elements and their structure.

Clustering is the process of grouping a set of data elements (e.g. points, objects, etc.) into classes of similar data elements. In the application of the clustering for uncertainty quantification, data elements are reservoir models represented by a vector of their uncertain parameters. Uncertainty parameters serve as the features of the data elements; we only use quantitative continuous parameters, which are normalised between (0, 1).

Similarity of models is quantified using a metric on the parameter space. Depending on the nature of the data and the aim of clustering, different measures of similarity may be used. Two main distance measures are Euclidean and Mahalanobis distance. In the current work, we use Euclidean distance for clustering.

The Euclidean distance measure is one of the most widely used similarity measures. It works well for the compact or isolated clusters, which have spherical, Gaussian shape (Mao & Jain, 1996). Linear correlation among parameters can be problematic for this distance measure. Euclidean distance measure is expressed as follows:

$$d(x,y) = \sqrt{\sum_{i=1}^{D}(x_i - y_i)^2} = \|X - Y\| \tag{2.20}$$

where $\boldsymbol{X}$ and $\boldsymbol{Y}$ are two models represented by row vectors of total $\boldsymbol{D}$ parameters.

In the case of linear correlation among the parameters, one must use the squared Mahalanobis distance (Mahalanobis 1936). This distance assigns different weights to different parameters, based on their variances and pairwise linear correlations:

$$d(x,y) = (X-Y)\sum^{-1}(X-Y)^{T} \quad (2.21)$$

where $\sum$ is the sample covariance matrix of the parameters.

The aim in clustering is to maximize within cluster similarity while minimizing between cluster similarity, regardless of the type of similarity measure. These objectives form a combinatorial optimisation problem for which it is difficult to find the exact solution and the optimum clustering is commonly found using an iterative approach.

There are two types of clustering: hard clustering and soft clustering. In hard clustering, each point will be assigned to one and only one data cluster, i.e. partial membership is not allowed, and clusters are partitions of the data. In soft clustering (also referred to as fuzzy clustering), each data element has a probability of membership to several clusters. Thus, data elements can belong to more than one cluster, with different levels of membership. In this thesis, two hard clustering techniques, K-means and agglomerative, and one soft clustering technique, probabilistic distance, are used.

Clustering techniques are also classified into flat and hierarchical techniques. A flat technique creates a single partition of the data without resorting to the hierarchical procedure, while hierarchical clustering generates nested clusters. K-means and probabilistic distance are flat techniques while agglomerative is a hierarchical technique.

In this thesis, we use three clustering techniques as follows:

2.6.1 K-mean clustering

K-mean Clustering (KMC), first presented by MacQueen (1967), is the most widely used hard clustering algorithm. The KMC procedure follows a straightforward and easy way to classify a given data set through a certain number of clusters. KMC is carried out in the following steps:

1. Select ***k*** random models as initial cluster centres ***c***.

2. For each model ***m:***

2.1. for each cluster centre $\boldsymbol{c_k}$:

2.1.1. Compute the sum of squares of the Euclidean distance, equation (2.20), between $\boldsymbol{m}$ and $\boldsymbol{c_k}$.

2.2. Assign $\boldsymbol{m}$ to its closest cluster centre $\boldsymbol{c_k}$.

3. Move each centre $\boldsymbol{c}$ to the mean of its assigned models.

4. Go to step 2 and repeat ($\boldsymbol{i}$+=1) until the algorithm converges. The convergence time is when cluster centres do not change.

KMC is efficient in terms of computational time, with time complexity of O(***IKND***), where $\boldsymbol{D}$ is the number of parameters, $\boldsymbol{N}$ is the number of models, $\boldsymbol{K}$ is the number of clusters, and $\boldsymbol{I}$ is the number of iterations needed for convergence. In addition, KMC always converges quickly, although the global optimum is not guaranteed.

KMC is highly sensitive to initialization, as it is a stochastic algorithm and initial cluster centres are chosen randomly; thus one must perform the algorithm several times with different starting models. KMC is also sensitive to outlier models. Another problem with KMC is the assumption that clusters are spherical and may have a problem with non-convex shape; in addition, it is sensitive to variations in sizes and densities of clusters.

2.6.2 Hierarchical Agglomerative Clustering

Hierarchical Agglomerative Clustering (HAC) (Manning et al., 2008) is common non-flat hard clustering technique, outputs a hierarchy which does not result in distinct clusters and a discrimination criterion should be used to determine the number of clusters. HAC forms a dendrogram and clusters of different granularity can be created by stopping at different levels of this dendrogram.

Hierarchical clustering algorithms are either top-down or bottom-up. Bottom-up (agglomerative) algorithms take each model as a cluster, in the first place, and then successively agglomerate pairs of clusters until all clusters have been merged into a

single cluster. Top-down algorithms start from all the models in single clusters, then split the clusters recursively until each contains only a single model.

HAC uses the Euclidian distance between two clusters as the measure of similarity, based on the one of the following linkage type (Figure 2.4):

- Single link: the minimum of the distances between all pairs of models taken from two clusters. It tends to produce straggly or elongated clusters.
- Complete link: the maximum of the distances between all pairs of models taken from two clusters. It produces tightly bound or compact clusters.
- Centroids link: Two clusters with the closest centroids are merged.
- Average link: Two clusters with the average of all the pairwise distances from two clusters are merged.

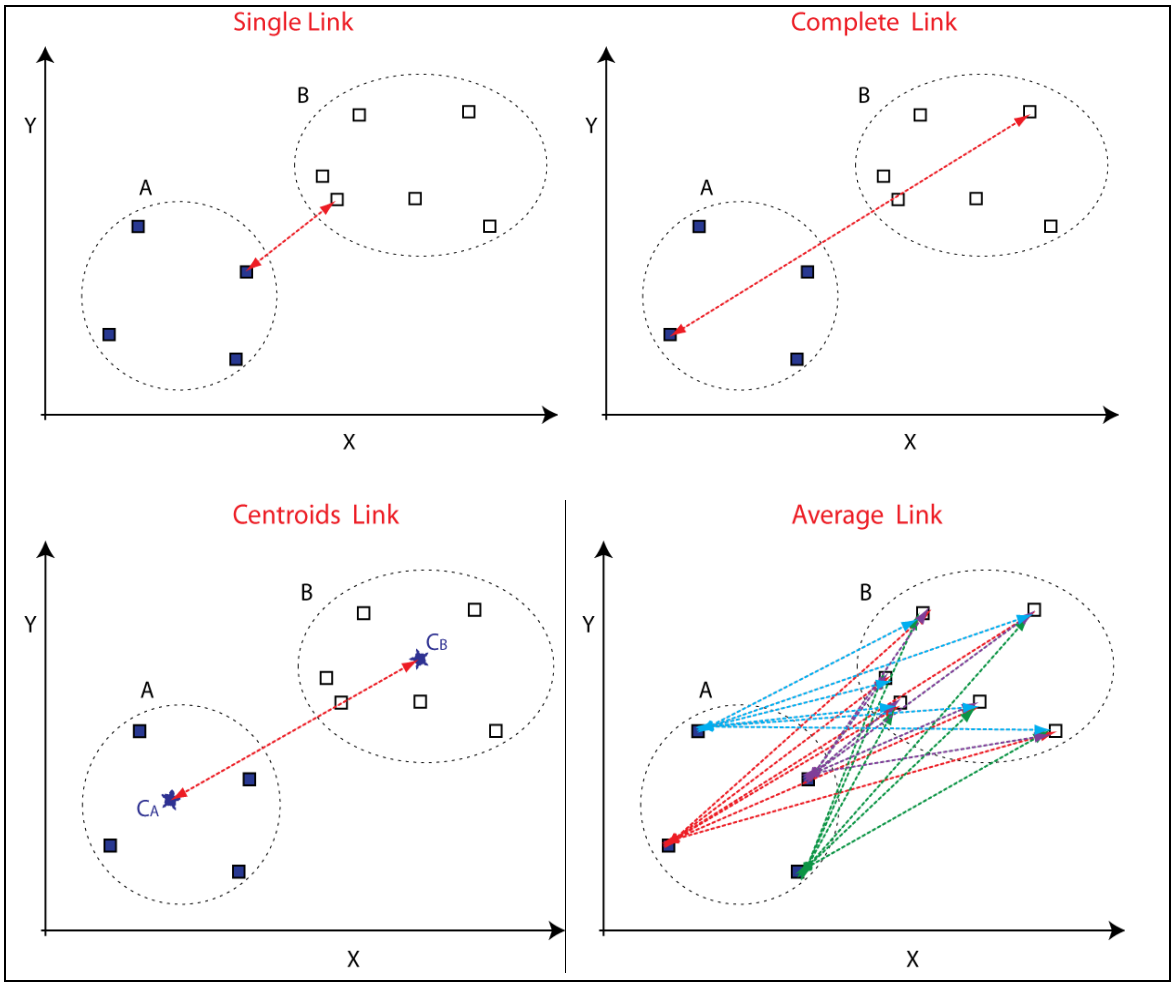


Figure 2.4: Link types for two clusters (A and B) in HAC; single link (top-left), complete link (top-right), centroids link (bottom-left), and average link (bottom-right).

A simple process for the HAC algorithm is as follows:

1. Start with each model in a separate cluster (***K***=***N***).
2. Calculate (if ***K***=***N***) or update (if ***K***≠***N***) distance between clusters.
3. Join the closest pair of clusters, based on selected link type.
4. Go to step 2 and continue until there is only one cluster ***(K***=1).

HAC could be computationally expensive as it has time complexity of O(N^3D), where ***D*** is the number of dimensions, ***N*** is the number of models.

2.6.3 Probabilistic Distance Clustering

Probabilistic Distance Clustering (PDC) (Ben-Israel. 2006, Iyigun & Ben-Israel, 2008) is a flat soft clustering technique, which like KMC, uses the sum of squares of Euclidean distance from the cluster centres as the similarity measure. In PDC, each data point is associated with a probability of membership to the cluster centres. The governing principle for PDC is that this probability is inversely proportional to the distance, i.e. for each model ***m*** and for each cluster with centre $\boldsymbol{c_k}$, following relation is true:

$$P_k(m) \times d(m, c_k) = constant, \text{depending on } m \tag{2.22}$$

Thus, the closer the data point to the cluster centre, the more probable cluster membership. The procedure to perform PDC is as follows:

1. Select ***K*** random models as initial cluster centres, $\boldsymbol{c_k = c_1, c_2 \ldots, c_K}$,
2. For each model $\boldsymbol{m_i}$ of total ***N*** models, $\boldsymbol{m_i = m_1, m_2 \ldots, m_N}$:

    2.1. For each cluster centre $\boldsymbol{c_k}$:

2.1.1. Compute the sum of the squares of Euclidean distance between model $\boldsymbol{m_i}$ and cluster $\boldsymbol{k}$ centre $\boldsymbol{c_k}$:

$$d(m_i, c_k) = \sqrt{\sum_{p=1}^{D} (m_{ip} - c_{kp})^2} \tag{2.23}$$

where $\boldsymbol{D}$ is the number of dimensions (parameters).

2.2. Assign probability of $\boldsymbol{m_i}$ belongs to cluster $\boldsymbol{k}$:

$$dP_k(m_i) = \frac{\prod_{j \neq k}^{K} d(m_i, c_j)}{\sum_{l=1}^{K} \prod_{j \neq l}^{K} d(m_i, c_j)} \tag{2.24}$$

3. Compute new cluster centres ($\boldsymbol{c_k}^*$) using computed probabilities:

$$c_k{}^* = \sum_{i=1}^{N} \left(\frac{u_k(m_i)}{\sum_{j=1}^{N} u_k(m_j)} \right) . m_i \tag{2.25}$$

$$\text{where: } u_k(m_i) = \frac{P_k(m_i)^2}{d(m_i, c_k)}$$

4. Go to step 2 and repeat until the algorithm converges. Convergence happens when the shift in cluster centres is less than a predefined cutoff ratio $\boldsymbol{\varepsilon}$:

$$\sum_{j=1}^{K} \left(\sum_{p=1}^{D} \left(c_{pj}{}^* - c_{pj} \right)^2 \right)^{1/2} < \varepsilon \tag{2.26}$$

The performance of PDC can be sensitive to probability cut-off. In addition, sometimes it struggles to converge, in which case the time complexity may get large.

Although PDC uses initial random cluster centres, unlike KMC, it is less sensitive to both initialization and outliers.

2.6.4 Clustering of the Iris dataset

The Iris data set introduced by Fisher (1936) is the standard multivariate data set for clustering techniques. It consists of 50 samples from each of three species of the Iris flower: Iris setosa, Iris virginica and Iris versicolor. For each sample, four features were measured: the length and the width of the sepals and petals. Clustering of the samples is then carried out based on the combination of these four features.

In the present research, clustering of the Iris samples was performed using each of the three algorithms for a fixed number of clusters, here equal to three. The experiments were repeated 10 times with different random seeds, due to the stochastic nature of the KMC and PDC. The results (Figure 2.5) show PDC outperforms KMC and HAC, as it better matches the truth case.

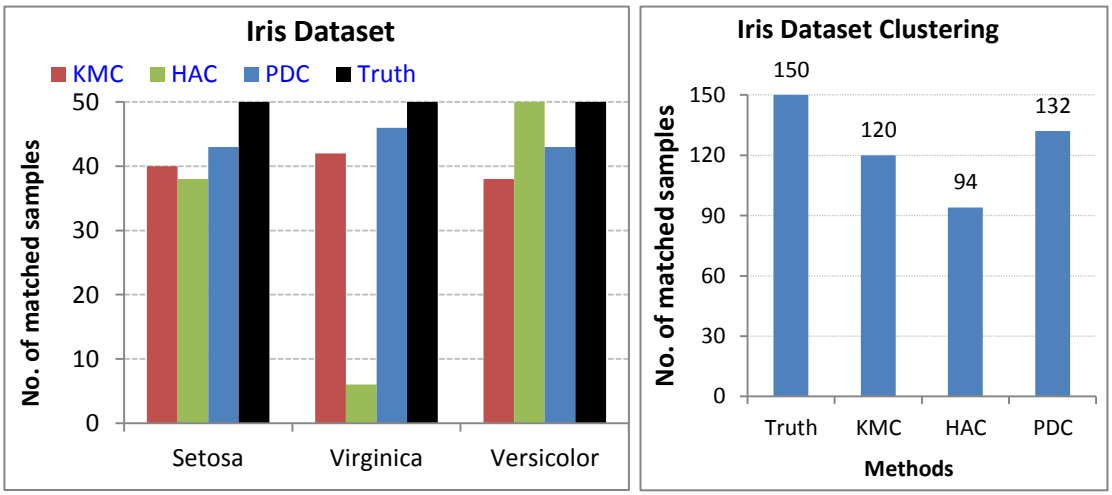


Figure 2.5: Results of Iris data set clustering as carried out in the current research; PDC results in closer clusters to truth case for all three species (left) and outperforms KMC and HAC in obtaining clusters with more samples matched with truth case (right).

2.7 Discussion

The 'No free lunch' theorem presented by Wolpert (1995) states that there is no single optimisation algorithm that is better than all the other rival algorithms on all the problems. There is a data set that an algorithm works well on and another on which it does not. Every history matching is a different optimisation problem; therefore, for each

history matching, the right algorithm should be selected. If we had plenty of computational resources, we could test multiple algorithms and control the parameters for each problem.

With limited computational resources, the main question is whether a selected algorithm uses a flexible mechanism or not, in other words, how it adapts to the structure of the problem so that it can be employed in different problems in a reliable way and with an acceptable performance. There are several important factors that affect the performance of the algorithm in general and in history matching or uncertainty quantification problems as specific. These factors are briefly discussed below.

The way that we convert the history matching to an optimisation problem has a significant impact on the results. This includes two main steps, the uncertainty parameterisation and misfit definition. When parameterising the reservoir model for history matching it is necessary to capture and consider all the key sources of uncertainty, using a sensitivity study. Pre-processing and quality control of the historical data are required for an effective misfit definition. If required, the quality of the match (misfit) could be expressed using multiple components, and then a multiobjective optimisation can be used instead of a single objective (overall misfit) optimisation.

The choice of the algorithm and the way that it adapts to the structure of the problem are other influential factors. The parameter space could be continuous or discrete. The number of search parameters could be extremely high, and they could have an unknown form of relationship with each other. The misfit landscape changes between different history matching problems, and hence the algorithm is required to perform a search which spans all the regions of the parameter space. It also should be able to give enough degree of refinement in possible regions of the minimum misfit. As mentioned earlier, such a quality search should be achieved with minimum possible algorithm testing and parameter tuning. The use of search algorithms with implicit and explicit adaptive mechanisms is a way forward for this problem.

Bayesian methodology is used in uncertainty quantification to obtain the posterior probability of prediction from an ensemble of history-matched models. The methodology involves identifying uncertainties of the model and integrating them in the

simulation representation, propagating them through the history matching framework, and finally quantifying the uncertainties in the model prediction. MCMC methods can be used to sample the posterior distribution. Simple MCMC variants are prohibitively expensive, due to slow mixing and convergence of the random walk, making it difficult to use in practical applications. Later in this thesis, we introduce a new uncertainty quantification method, based on the MCMC sampling in a Bayesian framework.

2.8 References

Abou-Kassem, J.H., Aziz, Kh. (1985). Analytical Well Models for Reservoir Simulation. *SPE Journal*, 25(4), 573-579.

Chappelear, J.E., Williamson, A.S. (1981). Representing Wells in Numerical Reservoir Simulation: Part 2 – Implementation. SPE Journal, 21 (3), 339-344.

Chib, S., Greenberg, E. (1995). Understanding the Metropolis-Hastings Algorithm. *The American Statistician*, 49 (4), 327-335.

Coats, K., Dempsey, J., & Henderson, J. (1970). A New Technique for Determining Reservoir Description from Field Performance Data. SPE Number: 2344-PA. *SPE Journal*, 10 (1), 66-74.

Damsleth, E., Hage, A., Volden, R. (1992). Maximum Information at Minimum Cost: A North Sea Field Development Study with Experimental Design. *Journal of Petroleum Technology*. 1350-1360.

Eiben, A. E. and Smith, J. E. (2003). Introduction to Evolutionary Computation, Chapter 2. Berlin: Springer.

Erbas, D., & Christie, M.A. (2007). Effect of Sampling Strategies on Prediction Uncertainty Estimation. SPE Number: 106229-MS. SPE Reservoir Simulation Symposium. Houston, U.S.A.

Evensen, G., Hove, J., Meisingset, H., Reiso, E., Seim, K., & Espelid, O. (2007). Using the EnKF for Assisted History Matching of a North Sea Reservoir Model. SPE Number: 106184-MS. SPE Reservoir Simulation Symposium. Texas, U.S.A.

Ezuka, I.O, Egbele, E.E. & Onyekonwu, M.O. (2004). Productivity Enhancement through Single Well Numerical Modeling.

Floris, F., Bush, M., Cuypers, M., Roggero, F., & Syversveen, A.-R. (2001). Methods for Quantifying the Uncertainty of Production Forecasts. *Petroleum Geoscience*, 7, 87-96.

Gelfand, A.E., Hills, S.E., Racine-Poon, A., & Smith, A.F.M. (1990). Illustration of Bayesian Inference in Normal Data Models Using Gibbs Sampling. *Journal of the American Statistical Association*, 85, 972–985.

Geman, S. & Geman, D. (1984). Stochastic Relaxation, Gibbs Distribution, and the Bayesian Restoration of Images. IEEE Transaction on Pattern Analysis and Machine Intelligence, 6, 721–741.

Ghanem, R., & Spanos, P. (1991). Stochastic Finite Elements: A Spectral Approach, Springer Verlag (reissued by Dover Publications, 2004.).

Gruenwalder, M., Poellitzer, S., & Clemens, T. (2007). Assisted and Manual History Matching of a Reservoir with 120 Wells, 58 Years Production History and Multiple Well Re-Completions. SPE number: 106039-MS. EUROPEC/EAGE Conference and Exhibition, 11-14 June 2007, London, U.K.

Gu, Y. & Oliver, D.S. (2005). History Matching of the PUNQ-S3 Reservoir Model Using the

Ensemble Kalman Filter. Soc. Petrol. Eng. J., 10(2), 217-224.

Hajizadeh, Y. (2010). Ants Can Do History Matching. SPE Number: 141137-STU. SPE Annual Technical Conference and Exhibition, 19–22 September 2010. Florence, Italy.

Hajizadeh, Y., Christie, M.A., & Demyanov, V. (2010). Comparative Study of Novel Population-Based Optimization Algorithms for History Matching and Uncertainty Quantification: PUNQ-S3 Revisited. SPE Number: 136861-MS. Abu Dhabi International Petroleum Exhibition and Conference. Abu Dhabi.

Halton, J. H. (1970) A retrospective and prospective survey of the Monte Carlo method. SIAM Review, 12, 1–63.

Hastings, W. K. (1970). Monte Carlo Sampling Methods Using Markov Chains and Their Applications. *Biometrika*, 57, 97-109.

Jahns, H. (1966). A Rapid Method for Obtaining a Two-Dimensional Reservoir Description from Well Response Data. SPE Number: 1473-PA. *SPE Journal*, 6 (4), 315-327.

Katz, D.L. (1936). Methods of Estimating Oil and Gas Reserves. Trans. AIME, Vol. 118, p.18.

Leamer, E.E. (1978). Specification Searches: Ad Hoc Inference with Nonexperimental Data. New York: Wiley.

Madigan, D.M. and Raftery, A.E. (1994). Model selection and accounting or model uncertainty in graphical models using Occam's Window. *Journal of the American Statistical Association*, 89, 1335-1346.

Manceau, E., Mezghani, M., Zabalza-Mezghani, I., Roggero, F. (2001). Combination of Experimental Design and Joint Modeling Methods for Quantifying the Risk Associated With Deterministic and Stochastic Uncertainties – An Integrated Test Study. SPE number: 71620. SPE ATCE, 30 September – 3 October. New Orleans, USA.

Manning, D., Raghavan, P., and Schütze, H. (2008). Introduction to Information Retrieval. Cambridge, UK: Cambridge University Press.

Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller. A.H. & Teller, E. (1953). Equations of State Calculations by Fast Computing Machines. *Journal of Chemical Physics*, 21, 1087-1092.

Mohamed, L., Christie, M.A., & Demyanov, V. (2010). Comparison of Stochastic Sampling Algorithms for Uncertainty Quantification. SPE Number: 119139-PA. *SPE Journal*, 15 (1), 31-38.

Nævdal, G.; Johnsen, L.M.; Aanonsen, S.I.; Vefring, E.H. (2005). Reservoir Monitoring and Continuous Model Updating Using Ensemble Kalman Filter. SPE Journal, 10 (1), pp. 66-74. SPE-84372-PA.

Oliver, D.S., Reynolds, A.C., Liu, N. (2008). Inverse Theory for Petroleum Reservoir characterization and History Matching. Cambridge University Press. Cambridge, UK.

Peaceman, D.W. (1978). Interpretation of Well Block Pressure in Numerical Reservoir Simulation. *SPE Journal*, 18 (3), 183-194.

Petrovska, I. (2009). Estimation of distribution algorithms for reservoir history matching optimisation. PHD Thesis, Imperial College. London, UK.

Romero, C.E., Carter, J.E., Gringarten, A.C., & Zimmerman, R.W. (2000). A Modified Genetic Algorithm for Reservoir Characterisation. SPE Number: 64765-MS. International Oil and Gas Conference and Exhibition. Beijing, China.

Sambridge, M. (1999). Geophysical Inversion with a Neighbourhood Algorithm -II Appraising the ensemble. *Geophysical Journal International*, 138, 727-746.

Schulze-Riegert, R., Axmann, J., Haase, O., Rian, D., & You, Y.-L. (2001). Optimization Methods for History Matching of Complex Reservoirs. SPE Number: 66393-MS. SPE Reservoir Simulation Symposium. Houston, Texas, USA.

Schulze-Riegert, R., Krosche, M., & Mustafa, H. (2009). Data Assimilation Coupled to Evolutionary Algorithms - A Case Example in History Matching. SPE Number: 125512-MS. SPE/EAGE Reservoir Characterization and Simulation Conference. Abu Dhabi, UAE.

Sonier, F. & Ombret, O. (1973). A Numerical Model of Multiphase Flow Around a Well. SPE Journal, 13 (6), 311-320.

Subbey, S., Christie, M.A., & Sambridge, M. (2003). A Strategy for Rapid Quantification of Uncertainty in Reservoir Performance Prediction. SPE Number: 79678-MS. SPE Reservoir Simulation Symposium. Houston, Texas, U.S.A.

Sultan, A., Ouenes, A., & Weiss, W. (1994). Automatic History Matching for an Integrated Reservoir Description and Improving Oil Recovery. SPE Number: 27712-MS. Permian Basin Oil and Gas Recovery Conference. Midland, Texas, U.S.A.

Thiele, M.R., Batycky, R.P. and Fenwick, D.H. (2010). Streamline Simulation for Modern Reservoir-Engineering Workflows; Journal of Petroleum Technology.

Thomas, K., Hellums, L., & Reheis, G. (1971). A Nonlinear Automatic History Matching Technique for Reservoir Simulation Models. SPE Number: 3475-PA. SPE 46th Annual Fall Meeting. New Orleans, U.S.A.

von Neumann, J. (1951). Various Techniques used in Connection With Random Digits. National Bureau of Standards Applied Mathematics Series, 12, 36–38.

Williams, M.A., Keating, J.F., & Barhouty M.F. (1998). The Stratigraphic Method: A Structured Approach to History Matching Complex Simulation Models. *SPE Reservoir Evaluation & Engineering*, 1(2), 169-176.

Wolpert, D.H. (1996). The Lack of *A Priori* Distinctions Between Learning Algorithms. *Neural Computation*, 8, 1341–1390.

Yang, C., Nghiem, L., Card, C., & Bremeier, M. (2007). Reservoir Model Uncertainty Quantification Through Computer-Assisted History Matching. SPE Number: 109825-MS. SPE Annual Technical Conference and Exhibition. Anaheim, California, U.S.A.

Yu, X. & Gen, M. (2010). Introduction to Evolutionary Algorithms. London: Springer-Verlag.

CHAPTER 3:

ESTIMATION OF DISTRIBUTION ALGORITHMS

> "The evolutionary algorithm -- of variation and selection, repeated -- searches for solutions in a world where the problems keep changing, trying all sorts of variants and doing more of what works."
>
> Tim Harford (1973 - …)

3.1 Introduction

Since the knowledge and information we have on the subsurface, in the form of field data, is uncertain, decision-making in reservoir management must be performed under uncertainty. According to decision theory, in such a situation, a set of primitive outcomes with probability of occurrence provide an expected value that can be used for optimal decision-making. Therefore, in order to get better development decisions under subsurface uncertainty, one requires this uncertainty to be quantified. Reservoir uncertainty quantification is an essential part of reservoir management, in which history-matched reservoir models are used for production optimization and forecasting, incorporated with estimation of the uncertainty.

In order to visualize and quantitatively assess the impact of subsurface uncertainties, uncertainty parameters need to be explored and multiple realizations in a Bayesian framework need to be created. As we search parameter space, history matching is converted to an optimization problem with the objective of minimizing deviation from the historical data. As pointed out above, history matching is an inverse problem, and

just like any other inverse problem has non-unique solutions. Nevertheless, multiple, acceptably history-matched models allow us to account for uncertainty in the model parameters, and estimate the uncertainty in prediction.

As discussed in Chapter Two, the exploration of parameter space and the generation of multiple model realizations in history matching are performed by three overlapping groups of algorithms: deterministic, stochastic, and data assimilation. Deterministic approaches are typically gradient-based methods, making use of the derivative of objective function with respect to reservoir parameters (Jahns, 1966; Coats et al., 1970; Thomas et al. 1971). In such approaches, uncertainty is usually estimated based on the local Hessian. However, given the reliance of these algorithms on the local gradient, there is always a significant chance of the algorithm becoming trapped in a local minimum.

Stochastic approaches incorporate probabilistic elements that allow the algorithm to escape from local minima, and hence find better minima elsewhere in the search space. Furthermore, this makes the algorithm less sensitive to modelling errors. Stochastic approaches differ in terms of maturity and sophistication, from uncomplicated random search to local search based methods such as simulated annealing and tabu search, and finally, to evolutionary population-based algorithms that aim to exploit interaction between different solutions to improve the quality of the search.

Stochastic search algorithms, such as genetic and evolutionary algorithms, exploit knowledge of the distribution of good solutions, amongst those already visited, in order to select wisely new points in the search space to evaluate. In the case of genetic algorithms, this information is stored implicitly in the current population. It is exploited through the application of variation operators, such as crossover and mutation in genetic algorithms, to the solutions in this population, which, it is hoped, lead to the generation of improved solutions.

The success of genetic algorithms is sometimes attributed to the *building block hypothesis* (Goldberg, 1989), whereby it is conjectured that the genetic algorithm efficiently identifies and recombines *building blocks*, i.e. solution components, or schemata, with above average fitness. However, in practice, genetic operators often

break such partial solutions, especially when these schemata are large, spread widely across the solution or when operators such as uniform crossover are used.

This raises the possibility of an alternative approach: one can try to understand explicitly high quality building blocks and integrate these solution components when generating new solutions. This approach would prevent the destruction of high quality schemata by the crossover operators and is the underlying concept behind *Estimation of Distribution Algorithms*, or EDAs. EDAs create an explicit model of the distribution of good solutions found so far that can, hopefully, be used to generate improved solutions in the future. Therefore, instead of manipulating solutions from some population, new solutions are generated using the model. As the search proceeds, the model is updated or adapted as new solutions are evaluated.

EDAs as population-based evolutionary stochastic algorithms are natural and attractive alternatives to genetic algorithms. Unlike genetic algorithms, EDAs do not require multiple tuning parameters; provide an underlying model that is transparent and expressive. EDAs use a probabilistic model to guide the search process, which is learnt from the promising solutions in each population and subsequently used to generate new solutions.

Many EDAs have been developed, differing in the complexity of the models generated, the degree to which interactions between variables can be modelled, the style of learning used (gradual adaptation vs. recalculation of the model) and the type of problem to which they can be applied (discrete vs. real valued). Of many EDA variants in existence, Petrovska & Carter (2006) applied the Population-Based Incremental Learning (PBIL) algorithm to the history matching problem. They showed that their selected EDA (PBIL) better suited to the subsurface uncertainty application than genetic algorithms and yielded robust results.

In this chapter, we introduce and explore the application of Estimation of Distribution Algorithms (EDAs) to uncertainty quantification. We describe three EDA Algorithms, underlining the nature and complexity of the probabilistic model of each EDA variant. Then we apply EDAs to the uncertainty quantification problem of the PUNQ-S3 synthetic model and the history matching problem of a full-field model of a North Sea

oilfield, emphasizing the EDA paradigm's potential for further research in history matching and uncertainty quantification.

3.2 Estimation of Distribution Algorithms

As explained above, the success of genetic algorithms is usually attributed to the *building block hypothesis*, wherein it is hypothesized that a genetic algorithm preserves and combines the building blocks that are found in good solutions to create still better solutions. Here, a building block is some part of the solution, for example, a sequence of bits in a bit string representation. This hypothesis might be true when considering small building blocks, such as individual bits in a bit string. If good solutions tend to have a 1 in position 4 in a bit string, then via the elimination of poor quality solutions and the selection of high quality parents, a 1 in this position will appear more frequently in the generated children. However, larger building blocks or blocks consisting of widely separated parts of the solution tend to be disrupted by the application of the mutation and crossover operations.

A natural alternative to the genetic algorithm approach is to attempt to detect the building blocks that make high quality solutions. In other words, we attempt to determine explicitly what makes a good solution. This knowledge can then be used intelligently to generate new solutions that combine building blocks with minimal disruption. This approach is the basis of Estimation of Distribution Algorithms.

A large number of Estimation of Distribution Algorithms (EDAs) are reported in the literature, differing to lesser or greater extent in the details of implementation. However, they each share a similar overall structure, involving the selection of a set of good solutions (and possibly a set of bad solutions), the fitting or adaptation of a model to the solutions selected and the generation of new solutions using the model.

3.2.1 Structure of EDAs

Like any other population-based EA, to solve an optimisation problem using an EDA, the problem is described using a number of design variables (*parameters*) thus a vector

of parameter values represents a solution for the problem. In EDAs, we create a set of *N* different parameter vectors (*solutions*) and consider it as a *population of solutions*. The quantity *N* is called the *population size*. All the solutions in the population are evaluated for their performance, and the quality of the solution is expressed in terms of a scalar valued *objective function.* Solutions with the lower or greater objective functions respectively, in a minimisation or a maximisation problem, are considered *promising*.

The algorithm proceeds to select the *P*, for which *P<N*, promising solutions out of the population to become the *parents* of the next generation according to the principle of natural *selection*, survival of the fittest. The algorithm then constructs a *probabilistic model* for each parameter out of the *P* promising solutions and generates a set of *C* new child solutions (*children*) by sampling the constructed probabilistic model. The total population is updated by adding the new child solutions and replacing some of the old ones and discarding unpromising solutions (*truncation*). The process is continued to the next generation until the *stopping criterion* is met. A stopping (or termination) rule could be a specified total number of generations or total number of function evaluations, a desired optimum value for the objective function, or a convergence situation where no improvement is made to fitness value in two successive generations.

Figure 3.1 shows the general structure of the EDAs. The shared element in all EDAs is that the true probability distribution of solutions is unknown, and one requires probabilistic models that are able to estimate the probability distribution of promising solutions effectively. New child solutions can then be generated using the estimated distribution model. Because of this model-building feature, EDAs are also called Probabilistic Model Building Genetic Algorithms (PMBGAs) (Pelikan et al., 1999).

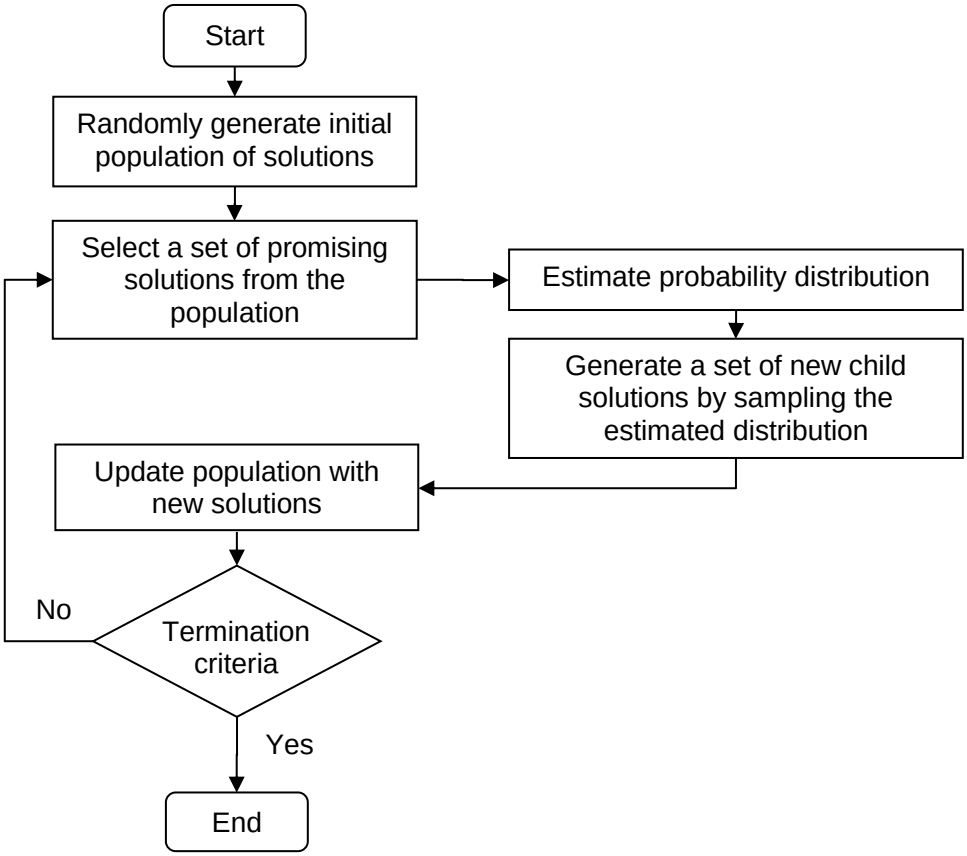


Figure 3.1: The structure of a typical EDA.

3.2.2 Classification of EDAs

A large number of EDAs are reported in the literature, differing to a lesser or greater extent in the details of implementation. Different variants of EDAs have specific characteristics which enable us choose and implement an appropriate EDA to deal with a particular optimisation problem.

Probabilistic models used by EDAs vary from discrete to real-coded problems; they may consider no interaction, pairwise interactions, or multivariate interactions between parameters; they may use basic probability models, such as histograms, or more sophisticated graphical models, such as Bayesian networks.

Here we classify EDAs according to three features: the solution representation and type of the problem to which they can be applied, the complexity of the probabilistic model used, and the extent to which the relationships between the variables are captured.

3.2.2.1 Solution representation

EDAs can be designed for different representation types of the individuals in the problem: binary, discrete or real-coded (Pelikan et al. 2012). There are many EDAs which are designed for and work with discrete representations of the solutions. One way of handling continuous domains is to discretise the range of the parameter to a limited number of bins, then use discrete EDAs. Continuous domains can also be modelled using real-coded EDAs, which generally assumes a distribution function.

In history matching, most of the variables are encoded in the continuous domain. Examples are different multipliers used to adjust the original values in the model. Discrete variables are also possible in history matching, typically representing a situation in the reservoir model such as a fault is closed or not, a rock-type or PVT model, etc.

In current chapter, we introduce and use histogram-based EDAs for history matching, which are natively working with discrete variables. In the next chapter, we describe a class of real-valued EDAs, Gaussian-based EDAs, and apply them to history matching.

3.2.2.2 Computational cost

One criterion could be the complexity of the probabilistic models used, and therefore the trade-off between the expressiveness of the probabilistic model and the computational cost of storing and learning the model. Simple models require less storage and computational resources, but are less capable of representing higher order interactions between variables. On the other hand, complex EDA models require learning algorithms that are costly in terms of storage and computation efficiency.

However, in history matching and uncertainty quantification, since the evaluation of solutions (reservoir simulations) already has a high computational cost, the additional cost of the sophisticated algorithms is unlikely to concern us.

3.2.2.3 Interaction between variables

Information about the interactions of the problem variables can increase the efficiency and the efficacy of the EDA if properly used. This information can be obtained by learning probabilistic dependencies in a set of realisations. Knowledge about the

problem may include the prior information about the interactions (e.g. see Baluja, 2006). Perturbation methods are deterministic and stochastic methods, which work by examining fitness differences by perturbations on variables, thus detecting sets of variables that are linked (e.g. see Munetomo & Goldberg, 1999). These approaches are ways of collecting information about the interaction between variables, and then choosing and employing an EDA according to the learnt information and adapting it to the dependency structure.

The extent to which the relationships between the variables are captured is another factor in designing and implementing EDAs. Therefore, EDA algorithms can be divided into different approaches depending on the degree of the interaction and dependency between variables that they take into account. In general, the following three levels of dependency can be defined:

3.2.2.3.1 No dependencies

Univariate EDAs such as Univariate Marginal Distribution Algorithm (UMDA), (Mühlenbein & Paaß, 1996), compact Genetic Algorithm (cGA) (Harik et al., 1999) and Population-Based Incremental Learning (PBIL) (Baluja, 1994) are the simplest EDA algorithms, as they do not assume interactions between the variables. They consider only the individual variables by factorizing the joint probability distribution of the selected solutions as a product of univariate marginal probabilities, as shown in equation (3.1). Only these marginal probabilities are used in the learning process: no other structural learning is needed. Thus, it is assumed that n-dimensional joint probability distribution is factorised as:

$$\mathrm{p}(x_1, x_2, .., x_n) = \prod_{1}^{n} \mathrm{p}(x_i) \tag{3.1}$$

Therefore, only marginal probabilities are used in the learning process, and no other structural learning is needed. Univariate EDAs have been shown to be effective optimizers on some problems with no or weak interactions between the variables. However, they are typically applied to problems with discrete variables. One way to deal with real parameters is to try to fit a certain probability distribution to the selected solutions. For example, Stochastic Hill Climbing with Learning by Vectors of Normal

Distributions (SHCLVND) fits normal distributions to each variable. In the next chapters, we will introduce and use a univariate Gaussian-based EDA which uses Gaussian (Normal) distribution.

Histogram-based models are able to model multi-modal, non-normal distributions. In this chapter, we introduce and apply two of the histogram based models: the Basic Histogram Model and the Equal Area Histogram Model.

3.2.2.3.2 Bivariate dependencies

There is another set of EDAs, designed to consider bivariate dependencies to solve the problems with pairwise interaction between variables. Among these EDAs are the Mutual Information Maximizing Input Clustering (MIMIC) Algorithm (De Bonet et al., 1997), Combining Optimizers with Mutual Information Tress (COMIT) (Baluja & Davies, 1997) and the Bivariate Marginal Distribution Algorithm (BMDA) (Pelikan & Mühlenbein, 1999).

3.2.2.3.3 Multivariate dependencies

In many real world optimisation problems multiple interactions occur. Multivariate EDAs, such as the Factorized Distribution Algorithm (FDA) (Mühlenbein et al., 1999), EBNA (Etxeberria & Larrañaga, 1999), the Bayesian Optimization Algorithm (BOA) (Pelikan et al., 1999) and the Estimation of Bayesian Network Algorithm (EBNA) (Larrañaga et al., 2000), take into account higher orders of the dependencies between variables. They factorize the joint probability distribution using statistics of order greater than two; therefore, the complexity of the probabilistic structure increases. As a result, the more complex learning process in these algorithms results in higher computations. However, by modelling the interactions between the variables, such algorithms can explicitly detect and preserve building blocks that cannot be handled by UMDAs or genetic algorithms. From multivariate EDAs, in current chapter, we describe and employ BOA and in Chapter 6, we will discuss and apply multivariate Gaussian-based EDA for history matching.

3.2.3 Basic Histogram Model (BH)

The Basic Histogram Model is used for strings of real values. The permitted range for each variable is split into equal sized bins, and a marginal probability value is stored for

each bin for each variable. Initially, these values are set to 1/c where c is the number of bins. After selecting good solutions, the probability for bin *b*, variable *i*, is set to n_{ib}/n, where n_{ib} is the number of solutions with a value for variable *i* that occurs in bin *b* and *n* is the total number of solutions.

New solutions are generated by selecting a bin for each variable according to the stored probability values. A value for each variable is selected at random within the range of the bin. Figure 3.2 illustrates how the basic histogram approach for a single variable models the marginal probability distribution over the selected solutions. The graph plots probability density against variable value, while above the graph are the probabilities for selecting each bin. This shows the histogram that would be generated given a variable that is permitted to take values in the range [1.4, 2.4] when, for example, 100 solutions are selected, 25 occurring in the first bin, 16 in the second and so on.

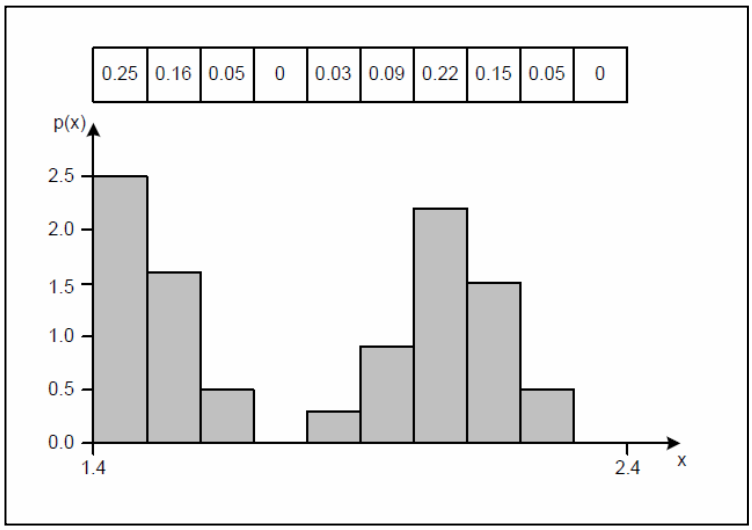


Figure 3.2: Basic Histogram Model; variable range is split into equal sized bins.

3.2.4 Equal Area Histogram Model (EAH)

The Equal Area Histogram Model is also used for real-valued strings and uses the approach of partitioning the valid variable ranges into bins. However, the probability of selecting a bin is kept fixed at *1/c*. Having selected *n* good solutions, the boundaries of bins are placed so that precisely *n/c* solutions occur in each bin. The first and the last boundaries are placed at the lower and upper bounds for the variable, while the internal boundaries are placed equidistant between the last solution before the boundary and the first one after it. This approach allows the model to focus on a small part of the

allowable range, for example, if the allowable range has been set unrealistically wide. Figure 3.3 shows the distribution of one of the variables for 20 good solutions at the top. The lines demarcate the ten bins generated, while the graph shows the resulting probability density. Note that each bar is of equal area, providing the name of this model type.

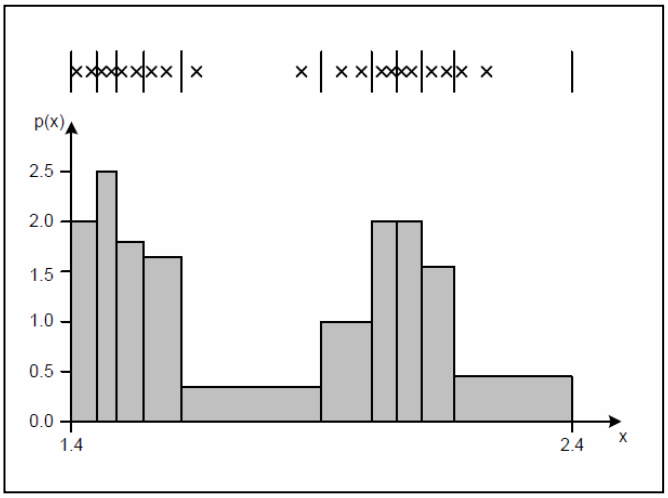


Figure 3.3: Equal Area Histogram Model; variable range is split into equal area bins.

3.2.5 Bayesian Optimization Algorithm (BOA)

BOA uses the Bayesian network for modelling. We will first describe a generation of a solution consisting of five bits (Figure 3.4). The network consists of a node for each bit in the solution representation and edges that represent relationships between the bits. For example, in Figure 3.4, the arrow from node 2 to node 3 indicates that the probability that bit 3 should be set depends on whether bit 2 is set. Associated with each node is the probability that the corresponding bit should be set, given the values of the bits associated with the parent nodes. As a whole, the network encodes the joint probability distribution for the values of all the bits in the solution representation.

Solutions are constructed from the network one bit at a time. Nodes in the network are visited in a sequence that ensures that parent nodes are visited before child nodes. On visiting a node, the associated bit is set according to the probability provided by the node and the values of the bits associated with any parent nodes.

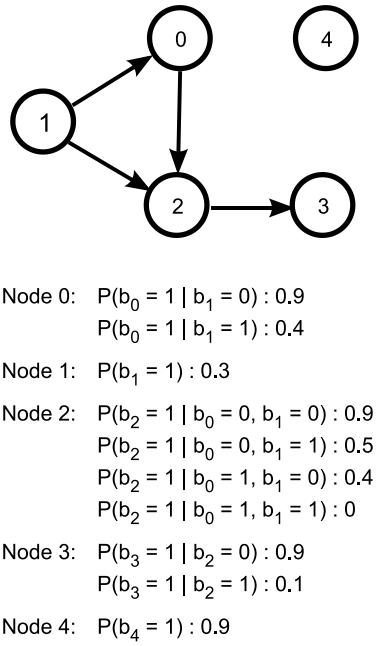


Figure 3.4: A simple Bayesian network for a five-bit problem.

For example, the nodes in Figure 3.4 may be visited in the sequence 1, 0, 2, 3, 4. Suppose we have already visited node 1 and set bit 1 to zero. Then, on visiting node 0, we look up the probability that bit 0 should be set to one, given that bit 1 has already been set to zero, giving us a probability of 0.9. The value of bit 0 is set accordingly, and we proceed to the next node in the sequence.

Note that this implies a constraint on the structure of the network: it is essential that the network be acyclic. In addition, the user specifies an upper bound on the number of parent nodes that any node in the network may have. This controls the complexity of the network and places a user-defined limit on the amount of inter-variable interaction modelled by the network.

Creating the network consists of two steps: learning the network structure and assigning the probability values. Once the network structure has been determined, probability values are assigned in much the same way as in univariate EDAs. For example, given the network structure in Figure 3.4, the probability that b_0 should be set to one, given that b_1 is set to zero, is determined by considering the subset of the good solutions that have b_1 set to zero. The frequency with which b_0 is set to one within this subset provides the new probability value for the model. Therefore, if 95% of this subset of solutions has b_1 set to one, the probability value is set to 0.95.

Determining network structure is treated as an optimization problem. Network quality can be measured using the Bayesian Dirichlet metric, which combines the prior knowledge about the problem, and the statistical data from a given data set in the form of the *posterior probability*, $p(B|D)$ with the Bayes theorem. The higher the $p(B|D)$, the more likely it is that the network B is a correct model of the data set D; *i.e.* the *posterior probability* should be maximized. The Bayes theorem is expressed as follows:

$$p(B|D) = \frac{p(B).p(D|B)}{p(D)} \tag{3.2}$$

Denominator $p(D)$ is the probability for Data D, which is neglected, as it is constant for all the networks that we compare. The prior probability for network B, $p(B)$, is set to 1.0 for all networks in this work, i.e. no network is favoured by having a larger prior. The likelihood, term $p(D|B)$, is approximated with the Dirichlet distribution:

$$p(D|B) = \prod_{i=1}^{n-1} \prod_{\pi_{X_i}} \frac{\Gamma\left(m'(\pi_{X_i})\right)}{\Gamma\left(m'(\pi_{X_i}) + m(\pi_{X_i})\right)} \cdot \prod_{x_i} \frac{\Gamma\left(m'(x_i, \pi_{X_i}) + m(x_i, \pi_{X_i})\right)}{\Gamma\left(m'(x_i, \pi_{X_i})\right)} \tag{3.3}$$

The term $m(\pi_{X_i})$ is a count of instances in the data D where the variables Π_{X_i} (the parents of X_i) are instantiated to the values π_{X_i}, while $m(x_i, \pi_{X_i})$ is the number of instances where X_i is set to x_i and variables Π_{X_i} are set to π_{X_i}. (If Π_{X_i} is empty, it is assumed to one possible value, with all N items of data in D having Π_{X_i} instantiated to this value.) The terms $m'(x_i, \pi_{X_i})$ and $m'(\pi_{X_i})$ encode prior information about the number of instances with X_i set to x_i and Π_{X_i} set to π_{X_i}. In this work, each $m'(x_i, \pi_{X_i})$ is set to 1, implying that $m'(\pi_{X_i})$ should be set to the number of possible values for X_i, i.e. 2, in the case of a bit string representation.

As mentioned earlier, the optimization of the network structure is performed subject to two constraints. First, the network must be acyclic. Second, the number of parent nodes of any node in the network is constrained to be less than a user specified parameter k. This allows a balance to be struck between the expressiveness of the resulting network and the cost of finding the optimal network.

It is known that for values of k greater than one, the problem of determining the best network structure is NP-hard (Heckerman et al., 1994). However, since the optimization of the network structure must be performed in each generation of BOA, the amount of optimization that can be performed is limited. Typically, a basic local search based method is used. Here, we perform a steepest ascent local search, starting from the empty network and only permitting edge additions (see Pelikan et al., 2000). At each step, each candidate for a new network edge is considered. If an edge results in a cycle or breaks the constraint on the number of parent nodes, then the edge is discarded. Otherwise, the addition is evaluated according to the Bayesian Dirichlet metric. The edge that maximizes this metric while satisfying the constraints is added.

A number of techniques are used to improve the performance of the optimization of the network structure (e.g. see Heckerman et al. 1994). Firstly, the factorials in the Bayesian Dirichlet metric can lead to numerical issues. Hence, the algorithm optimizes the logarithm of this metric. Secondly, it is possible to calculate the change in the metric upon the addition of a candidate edge, rather than calculate the entire Bayesian Dirichlet metric from scratch. Finally, whenever an edge is added to the network, the algorithm can update the cost changes associated with each remaining candidate edge, noting that the cost change associated with many edges will not be affected.

Determining suitable parameters for BOA is, to a degree, a matter for experimentation. However, the experimental result showed that the quality of the optimization depends more on the value of promising solutions (P) than on the new population (N) and child solutions (C). A smaller value for C results in Bayesian networks being generated more frequently, so that new high quality solutions have a more immediate effect on the search. If the evaluation of a solution is computationally expensive, a small value of C may be appropriate, although the overall search trajectory is not changed substantially. Similarly, provided N is greater than P, the only effect of a large value of N is to delay the generation of the first Bayesian network. However, P has a significant effect on the convergence rate of the algorithm. Smaller values of P result in the Bayesian network being fitted to a smaller set of data. The result tends to be over-fitting and faster convergence of the algorithm, with the resultant lack of sufficient exploration of the search space. Details of the parameter settings used follow in the results sections.

One weakness of BOA is that the optimization of the Bayesian network in each generation means that the algorithm is not particularly efficient. However, given the cost of the evaluation of a solution when performing history matching, this is not a serious concern. Another potential weakness is that the design of the algorithm emphasizes exploitation of knowledge of the location of high quality solutions at the expense of exploration of the search space. Once the algorithm converges on a small region of the search space, it will continue to generate other similar solutions in the same region. Combined with possible overfitting due to use of the Bayesian Dirichlet metric, this can result in premature convergence. Possible methods for counteracting this tendency to converge prematurely, such as the use of less elitist methods for selecting solutions, are a matter for further research.

Note that the BOA is designed to work with discrete solution representations, rather than strings of real values. Therefore, each real parameter is converted into a bit string. The first bit indicates whether the real value should be in the first half of the permitted range or the second half. The second bit further subdivides these halves, and so on. The number of bits used to represent each variable becomes a parameter of the algorithm.

3.3 Application 1: Uncertainty Quantification of the PUNQ-S3 Case

The three EDAs, i.e. EDA using basic histogram model (BHEDA), EDA using equal-area histogram model (EAHEDA), and BOA were applied to the PUNQ-S3 case study (PUNQ-S3 Test Case 2010). The objective was to compare the quantification of uncertainty obtained by EDAs with ones published for PUNQ-S3 in the literature.

3.3.1 PUNQ-S3 description

The PUNQ-S3 case is a synthetic reservoir developed for the EU-sponsored PUNQ project (Production forecasting with UNcertainty Quantification); it is based on a real field operated by Elf Exploration Production, with changes made to the petrophysical data.

The original PUNQ problem objective was to quantify uncertainty (Floris et al., 2001) in reservoir performance predictions, given some geologic information on the reservoir,

including hard data at well gridblocks and synthetic production history of the first 8 years. As the first stage, the PUNQ project partners were asked to predict cumulative oil production after 16.5 years, including uncertainty in this prediction. Porosity and permeability fields are uncertain in PUNQ-S. Each of the partners in the project used their own workflow to infer these fields. For the truth case, the cumulative oil production after 16.5 years is 3.87×10^6 Sm^3. In the second stage, 5 incremental wells were defined, and the partners were asked to forecast the incremental recovery including uncertainty. In this study, we only deal with the first stage objective.

3.3.1.1 Reservoir model specifications

The simulation model of PUNQ-S3 contains 19×28×5 (2660) grid blocks, of which 1761 blocks are active. The grid is created using corner-point geometry with dimensions of 180m × 180m. The field is structurally bounded to the east and south by a fault, while linked to a strong aquifer to the north and west. No injection is needed, since the field gets sufficient pressure support from the aquifer. The field initially has a small gas cap at the centre of the structure, and production wells are located around the gas oil contact. The top structure map and well locations are shown in Figure 3.5.

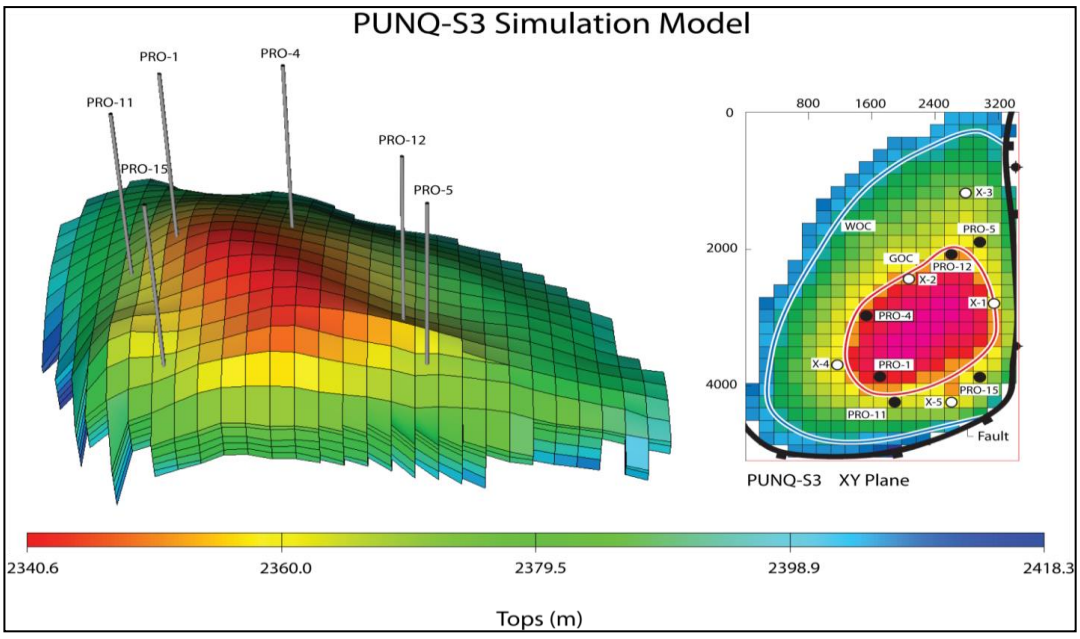


Figure 3.5: Top surface map and well locations in PUNQ-S3.

Only a brief geological description of the field, along with porosities and directional permeabilities at well locations, is given (Table 3.1). Well porosities and permeabilities include noise, to account for measurement errors. The porosity and permeability fields

for the 'truth case' are generated with a Gaussian Random Fields model and the well porosities and permeabilities are extracted from this truth case.

Table 3.1: Brief summary of PUNQ geological description including expected sedimentary facies with estimates for width and spacing of major flow units for each layer.

Layer	Facies	Width (m)	Spacing (km)
1	Channel Fill	800	2-5
2	Lagoonal Shale	—	—
3	Channel Fill	1000	2-5
4	Mouthbar	500-5000	10
5	Channel Fill	2000	4-10

The production period commences with one year extended well testing, followed by a three year shut-in period and 12.5 years production, of which 2 weeks each year is allocated to well testing. The field has a PVT, relative permeability and Carter-Tracy aquifer dataset taken from an original field data but it has no capillary function. Production is constrained by a well oil rate target of 150 sm^3/d and a BHP constraint of 120 bars.

3.3.1.2 PUNQ-S3 objective function

The objective function defined in the original study for PUNQ-S3 is to minimize the misfit function (Barker et al., 2001).

$$M = \frac{1}{N_v}\sum_{j=1}^{Nv}\frac{1}{N_p}\sum_{i=1}^{Np}\left(W_{ij}\frac{O_{ij}-S_{ij}}{\sigma_{ij}}\right)^2 \tag{3.4}$$

The observation data include BHP, WCT, and GOR for each of the 6 producer wells, resulting in N_v = 18 measurements at each of N_p points in time. Subscript i indicates the time of the measurement while subscript j indicates the particular measurement made. The other variables are as follows:

- M is the misfit function,
- O_{ij} is the observed value of measurement j at the time i.
- S_{ij} is the simulated value of measurement j at the time i.
- σ_{ij} is the standard deviation of measurement j at the time i.

- W_{ij} is the weight factor applied to measurement j at the time i.

Observation data for the synthetic PUNQ-S3 case study is generated via simulation of the reservoir, followed by the addition of Gaussian noise with zero mean, to account for measurement errors (Table 3.2).

Table 3.2: Measurement errors in observation data

Data Type	Added Noise		Comments
BHP	1 bar for the shut-in	3 bars for production	Shut-in values are more accurate than production ones.
GOR	10% before gas-breakthrough	25% after gas-breakthrough	Accounting for the source of produced gas.
WCT	2% before the water-breakthrough	5% after water-breakthrough	Accounting for the breakthrough time.

3.3.1.3 Uncertainty parameterization

As described by Floris et al., (2001), the uncertain parameters in the PUNQ-S3 model are porosity, horizontal permeability, and vertical permeability. Taking all of the 1761 active grid blocks as the uncertainty parameters is the most general approach for parameterization of the PUNQ-S3. This approach neither considers the given brief geological knowledge, nor the spatial connectivity available in reservoir models. In addition to this, it results in a larger number of uncertainty variables, which can be expected to increase the computational cost of optimization. One way to avoid this is to use homogenous regions in each layer of the reservoir to reduce the number of uncertainty variables. These regions are selected based on the given stratigraphic and geological concepts. Although the boundaries between the regions and homogeneity within each region may not be justified, the regions approach is common in manual history matching and sensitivity studies.

In this chapter, we use the parameterization of PUNQ-S3 described by Hajizadeh et al. (2010), which includes distinct porosity values for 9 homogenous regions for each of 5 layers, totalling 45 unknown values with prior range from the noise adjusted well porosities. Directional permeabilities are then computed from the following correlations extracted from least square fitting of the well porosity/permeability cross-plots (Boss, 1999).

$$\ln(k_h) = 0.77 + 9.03\emptyset$$
$$k_v = 3.124 + 0.306k_h \quad (3.5)$$

The prior ranges for the resulted porosity, horizontal permeability and vertical permeability of the each of the 5 layers are shown in Table 3.3.

Table 3.3: Initial ranges for PUNQ-S3 uncertainty parameters.

Layer	Porosity	Horizontal Permeability (md)	Vertical permeability (md)
1	0.15-0.3	133-3013	44-925
2	0.05-0.15	16-133	8-44
3	0.15-0.3	133-3013	44-925
4	0.1-0.2	47-376	17-118
5	0.15-0.3	133-3013	44-925

3.3.2 PUNQ-S3 Results

In the PUNQ-S3 experiments, we used three previously described EDAs (, BHEDA, EAHEDA and BOA) for generation of the multiple history-matched models. After optimization, the next step is to use the NAB sampler (Sambridge, 1999) to estimate the uncertainty of the production prediction of the multiple models generated by the EDA.

3.3.2.1 Algorithm set-up

Different algorithms have different parameters that need to be tuned before any performance study. Therefore, we carried out sensitivity study around the parameter set-up of each algorithm. The required parameters and the best parameter set-ups of each EDA are described in Table 3.4. We fixed the total number of simulation runs in each experiment to 3000 runs with each generation of the algorithms, after initialization, consisting of 50 runs (This is determined by the 'number of children' parameter of the EDAs). Pelikan et al. (2000) propose population size of O(n) to O($n^{1.05}$) for BOA; in the PUNQ-S3 case, with n equal to 45, 50 falls between these two numbers (45 and 54). They also propose a range of O($n^{0.5}$) to O(n) for the number of generations, although by fixing total number of simulation runs in each experiment to 3000, we took a larger number, 57.

The experimental results show that the EDAs are primarily sensitive to the number of parent solutions. This parameter influences the trade-off between the exploration and the exploitation behaviour of the algorithm in the parameter space.

Table 3.4: EDA set-up parameters.

Set-up Parameter	Algorithm(s)	Description	Best value obtained
Initial population size	BH, EAH, and BOA	Solutions in the initial population size are generated randomly from the parameter space.	150
Number of Parents	BH, EAH, and BOA	Number of 'parent' solutions selected from the best of the current population, and used to generate the probability model (e.g. Bayesian network in BOA).	100 for BOA and 50 for other EDAs
Number of Children	BH, EAH, and BOA	Number of 'child' solutions created by the probability model (e.g. Bayesian network in BOA) in each generation.	50
Maximum Parents	BOA	Maximum number of 'parent' nodes permitted within the Bayesian network.	6
Precision	BOA	The number of bits used to represent parameters.	8
Number of Bins	BH, EAH	Number of bins to represent each parameter in the model.	10

3.3.2.2 Computational resources

The computational resources used by a simulation run of PUNQ-S3 model is shown in Table 3.5. The number of active gridblocks in the model is the most important factor determining the required RAM size, which is not considerable in PUNQ-S3 model. Each RAM is shared between 8 CPUs. An iteration of the PUNQ-S3 model runs in about 19 minutes, which is still practical for performing tuning of the algorithms' control parameters.

Table 3.5: Computational resources available and used by PUNQ-S3 model.

RAM	Total RAM size	16 Gbyte
	Number of total gridblocks	2,660
	Number of active gridblocks	1,761
	Required RAM size	~ 5 Mbyte
CPU	CPU clockspeed	2.9 GHz
	Runtime for a single run using single CPU	15 seconds
	Number of runs in assisted history match	3,000
	Runtime for an iteration of EDAs using single CPU	~ 12.5 hours
	Number of CPUs available for this study	40
	Runtime for an iteration of EDAs using 40 CPUs	~ 19 minutes

3.3.2.3 Convergence results

We looked at the history matching convergence results of the algorithms. By varying two of the parameters, namely the initial population size and the number of parent solutions, we produced two set-ups for each algorithm, one more exploitative and the other more explorative. In the exploitative set-up, both parameters are set to 50, while in

explorative set-up the initial population size is set to 150 and the number of parent solutions set to 100. The rest of parameters are set as in Table 3.4.

In order to eliminate the effect of the initial population on the quality of the resulting search, each algorithm was initialized with the same initial population in both set-ups. Figure 3.6 to Figure 3.8 show performance of the EDAs in obtaining models with lower misfits. Convergence occurs for all algorithms. The histogram based algorithms (BH and EAH) are relatively effective in finding best misfit models in the search space when using the exploitative set-up while the BOA is preferred for the explorative set-up.

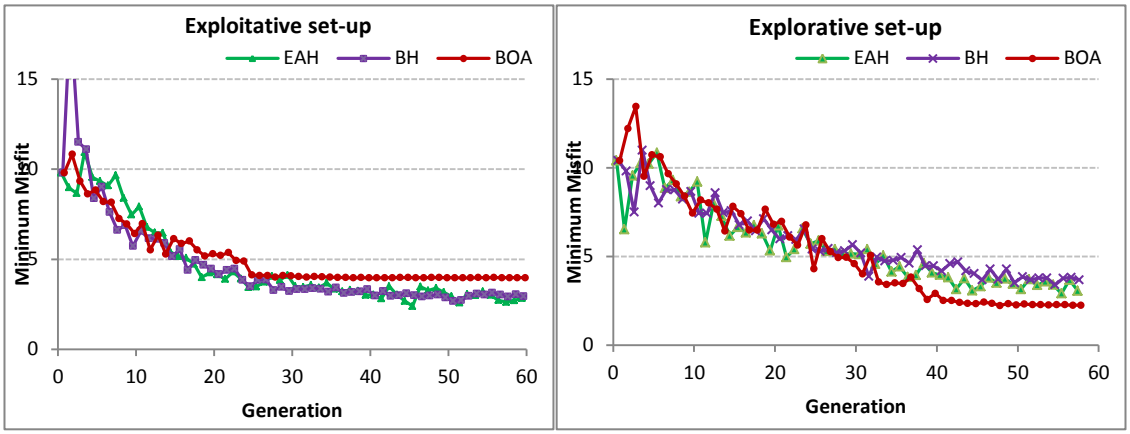


Figure 3.6: Minimum misfits over generations for 3 EDAs, for exploitative set-up (left) and explorative set-up (right); histogram-based algorithms, EAH (green) and BH (purple), show lower minimum misfit than BOA (red) in exploitative set-up, while in explorative setup BOA (red) resulted in best minimum misfit.

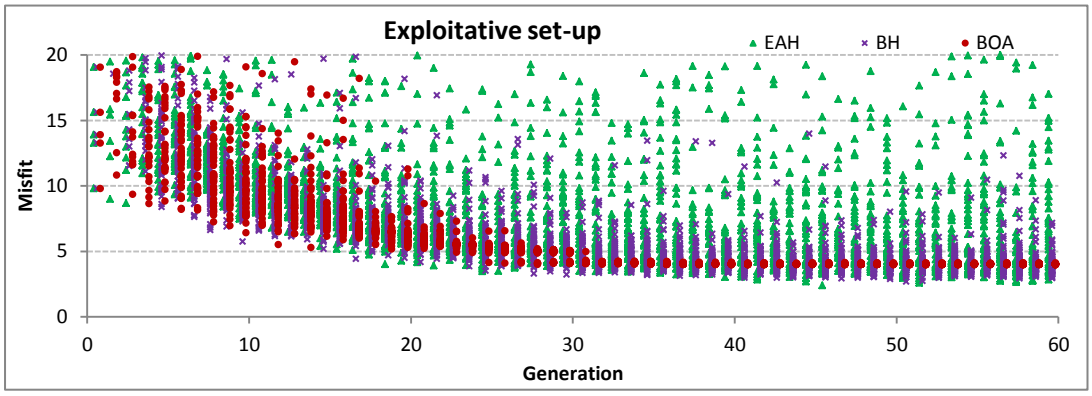


Figure 3.7: Generational misfits for 3 different EDAs, for exploitative set-up; BOA (red), perhaps because of premature convergence, seems to be trapped in a local minima; while EAH (green) and BH (purple) show better performance over the period of the search.

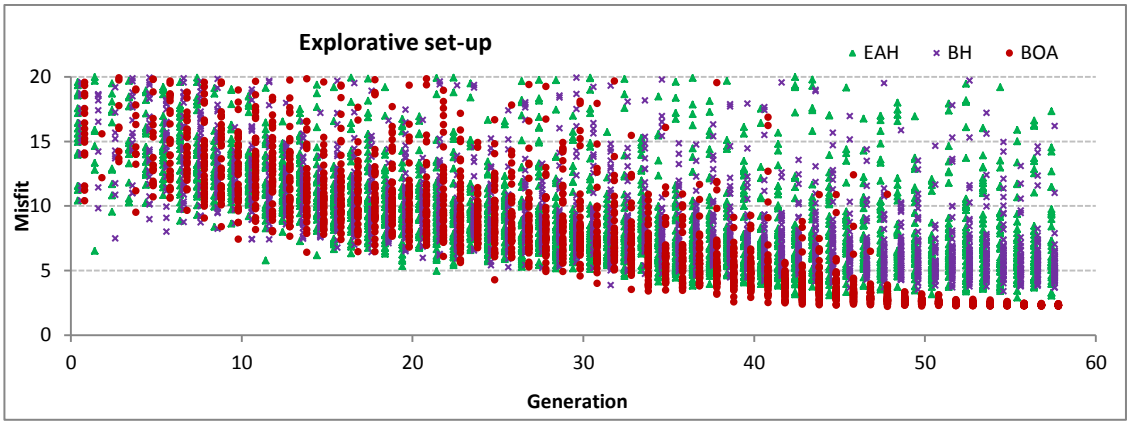


Figure 3.8: Generational misfits for 3 different EDAs, for explorative set-up; BOA (red) outperforms EAH (green) and BH (purple) in convergence speed.

3.3.2.4 Forecast uncertainty for PUNQ-S3

The multiple models generated by different EDAs in the history matching stage were used by the NAB posterior probability sampler to estimate the uncertainty around the ultimate recovery after 16.5 years of production in PUNQ-S3. NAB (acronym for neighbourhood Algorithm in Bayesian framework) is a MCMC technique for calculating Posterior Probability Distribution function using a Gibbs sampler. It constructs Voronoi cells in the search space and interpolates the likelihood of unknown points in the search space (Sambridge, 1999).

To study the redundancy and impact of the NAB sampler on the prediction results, we repeated posterior sampling and calculation of the credible interval on a fixed ensemble of models. In all the repeated cases, the starting point of the sampling process is the best misfit model. As NAB uses random walks in the sampling process, 5 NAB runs with the same random seeds and 5 runs with different seeds were carried out. The results (Figure 3.9) show that the impact of the NAB sampler is negligible; therefore, the EDA used for sampling parameter search space plays the main role in history matching and the uncertainty quantification framework.

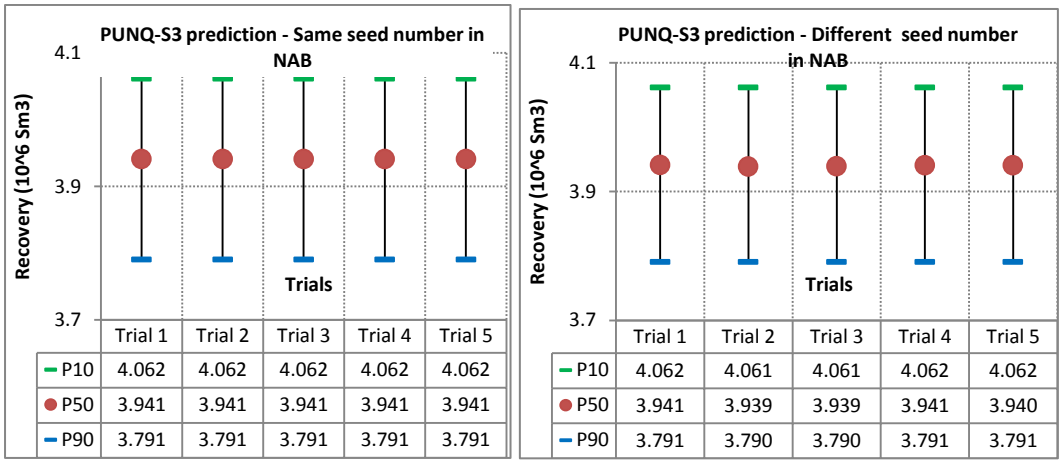


Figure 3.9: Comparison of PUNQ-S3 uncertainty intervals obtained for 5 trials of an ensemble sampled by NAB. Same seed number in NAB yields exactly same results (left), and different seed number (right) yields results with highly negligible difference.

Using NAB we sampled 500,000 solutions from original 3,000 forward simulation runs generated by each of the 3 EDAs in two different set-ups. As shown in Figure 3.10, the credible intervals obtained by EDAs a in both explorative and exploitative set-ups are very close to truth case. In general, EDAs allows a more diverse sampling of the parameter space, leading to a global optimization and a prediction remarkably close to the truth case.

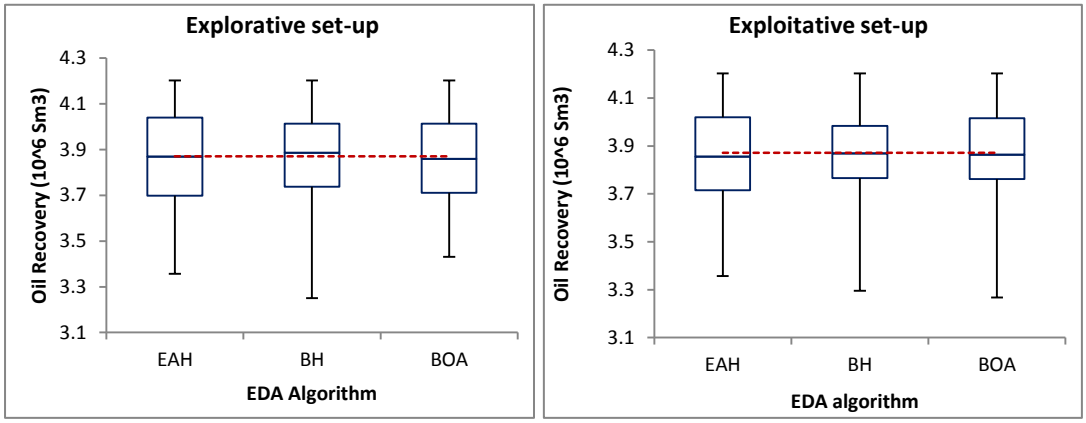
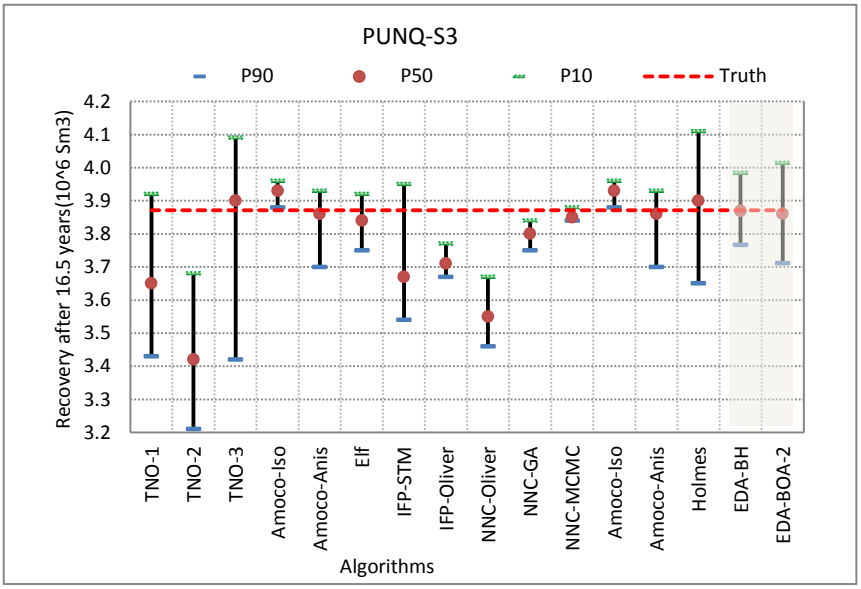


Figure 3.10: Box-plot of oil recovery obtained by 5 different EDAs, for explorative set-up (right) and exploitative set-up (left); both histogram-based algorithms (EAH and BH) and BOA predicted oil recovery with P50 (middle mark of the boxes) closer to truth case (red-dot line), lower P10 and P90 variation (top and bottom marks of the boxes), and narrower Maximum and Minimum range (top and bottom of the bars); BOA for explorative set-up and BH for exploitative set-up provides best prediction.

Predicted credible intervals (P10, P50, and P90) of the oil recovery after 16.5 years in PUNQ-S3 by BH (our best predictor model in the exploitative set-up) and BOA (our best predictor model in the explorative set-up) are compared to the original results published by the PUNQ project (Floris et al., 2001), and later results published by Demyanov et al. (2004), Mohamed et al.(2009), Hajizadeh (2010), and Hajizadeh et al. (2010) (Figure 3.11). The results show better estimation of the uncertainty than in the

original PUNQ study, and competitive results to other modern algorithms using the same uncertainty parameterisation.



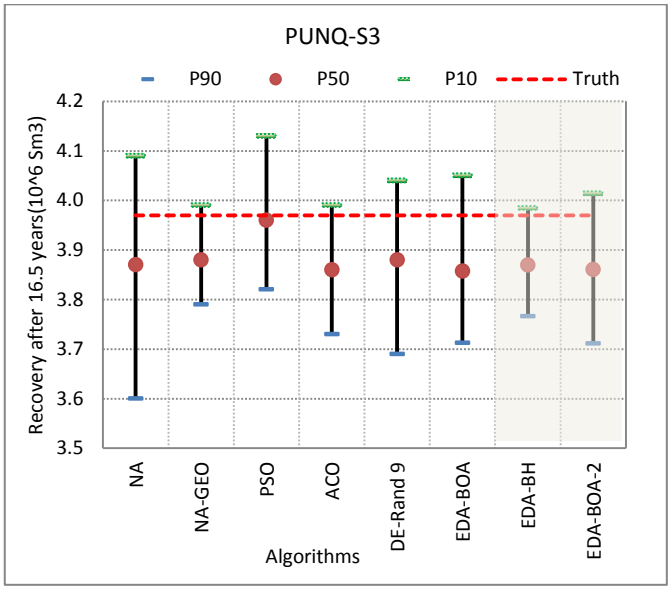


Figure 3.11: Uncertainty interval obtained for PUNQ-S3 by BH and BOA in this study, when compared with results in original PUNQ-S3 study (top), shows better estimation of uncertainty, and when compared to other algorithms in the group (bottom) shows EDAs are competitive.

3.4 Application 2: History matching of a real North Sea field (Koma)

In the second application, we apply one of the EDAs (BOA) to the history matching problem of a real, North Sea, turbidite field with multiple wells and we show improvements in performance over that of a traditional, population-based algorithm,

genetic algorithm (GA). Throughout this thesis, the field will be referred to as Koma for reasons of anonymity.

3.4.1 Field and development description

The Koma reservoir is composed of massive oil-bearing turbidite sands with 25-35% porosity, 200-1200mD permeability, in a shale background. The reservoir container is a combination of up-dip fault seal, dip closure, and pinch-out traps. Figure 3.12 shows a Net-to-Gross (NTG) map created for the entire reservoir based on the seismic net pay calibrated with well values. The field has two main reservoir compartments, defined as equilibrium regions, each associated with a PVT, WOC and GOC data.

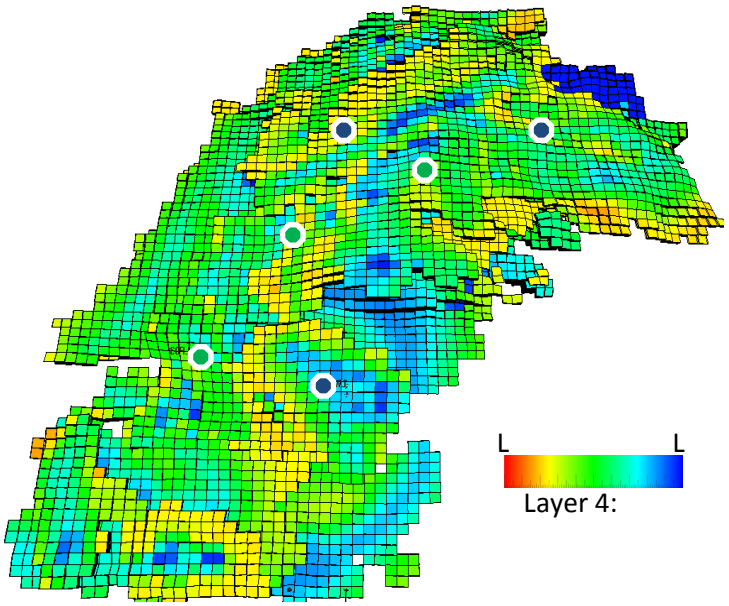


Figure 3.12: Reservoir simulation grid with NTG values, and location of the producers (green) and injectors (blue).

Koma now has three oil production wells and three water injectors. These were brought on-stream as three successive well-pairs. The present study covers the first ten years of operation; a further history period has been reserved for a future predictability study.

3.4.1.1 Reservoir model specifications

The business purpose of the original history-match study was assessing reservoir development and management opportunities of the field. This was to involve performing a history matching study yielding a base reservoir model, plus an ensemble of additional models that have acceptable match.

The model consists of some 32×103 grid blocks in 57 layers, with typical block dimensions of 50×50×3 metres. The underlying geological model is composed of sand bodies picked by seismic interpretation, with associated NTG, porosity, and permeabilities. A constant porosity-depth function is used for modelling porosity. The permeability definition comes from a porosity-permeability cross-plot yielded by rock-typing.

3.4.1.2 Key Uncertainties

Despite calibration, the difference between NTG calculated by seismic and NTG measured at wells pose a substantial uncertainty in the initial in-place volumes. Previous sensitivity studies have shown that uncertainty in permeabilities needs to be handled by using multipliers in the model. The main challenge in the Koma field is the vertical and horizontal communication between different sand bodies. Effect of these communications is seen in dynamic behaviour, reflected in the production data, RFT, and 4D-seismic data. The production is controlled by the total liquid rate (oil plus water) in the wells, with a backup BHP constraint.

3.4.2 Misfit definition and objective function

An objective function similar to that in the PUNQ-S3 case is used, as in equation (2.5). The misfit definition, including the observation datatypes and measurement error, is taken from the original genetic algorithm study. No further quality control and data analysis are done to allow like-to-like comparison of the two algorithms. Available historical data and the associated error model are shown in Table 3.6.

Table 3.6: History-match data and error model.

Parameter	Parameter abbr.	Error definition
Bottomhole pressure	BHP	50 psi
Near-well average pressure	Pave	50 psi
Oil flowrate	QOP	5% of historical value
Water flowrate	QWP	7% of historical value
Gas flowrate	QGP	30% of historical value

3.4.3 Uncertainty parameterization

From the sensitivity study and manual history matching, 54 uncertainty parameters were identified; these parameters include 16 pore volume multipliers (porosity and NTG), 8

permeability multipliers, 21 inter-region transmissibility multipliers, 8 fault transmissibility multipliers, and 1 aquifer volume multiplier. Descriptions and prior ranges of these parameters are given in Table 3.7.

Table 3.7: Koma uncertainty parameter types, with typical base value and prior ranges (downside and upside values).

Parameter type	Number	Typical base value	Typical downside	Typical upside
Pore Volume	16	0.5	0	1
Permeability multiplier	8	1.5	0	3
Inter-region transmissibility multiplier	21	0.01	0	1
Fault transmissibility multiplier	8	0	0	1
Aquifer pore volume multiplier	1	3.6	1	5
Sum	54			

3.4.4 Computational resources

The computational resources available to and used by a simulation run of Koma field are shown in Table 3.8. An iteration of Koma field assisted history matching requires 1,000 simulation runs, which take approximately 17 hours using 40 CPUs of the Heriot-Watt's HPC cluster. Thus, in real field application is not practical to perform a full control parameter tuning of the algorithms, which involves performing dozen of iterations.

Table 3.8: Computational resources available and used by Koma field.

RAM	Total RAM size	16 Gbyte
	Number of total gridblocks	187,872
	Number of active gridblocks	32,303
	Required RAM size	~ 100 Mbyte
CPU	CPU clockspeed	2.9 GHz
	Runtime for a single run using single CPU	40 minutes
	Number of runs in an EDA's iteration	1,000
	Runtime for an iteration of EDAs using single CPU	~ 667 hours
	Number of CPUs available for this study	40
	Runtime for an iteration of EDAs using 40 CPUs	~ 17 hours

3.4.5 History matching results

History matching of the Koma field was performed with the BOA algorithm to be compared with a history-match already obtained with GA by the asset team. To perform

a like-for-like comparison, the same misfit definition, uncertainty parameterisation and initial random population are used for both algorithms. The stopping criterion is defined to be 1000 simulation runs for each algorithm; the number of solutions per generation is also fixed to 50, resulting in a total of 20 generations. A cut-off criterion of 40 minutes CPU time is defined for possible simulation runs that may experience convergence problems; a large misfit value is allocated to any such runs. To avoid dependence of the results on the initial random starting population, we repeated each experiment for both algorithms 5 times and averaged the results. The GA has been tuned for optimal performance over many field studies.

We used tuned PUNQ-S3 control parameters for BOA. i.e. each history-match parameter was divided into 8 bits and the Bayesian network consists of a node for each bit in the solution (precision: 8). The best 100 solutions so far were selected as parents, resulting in 100×8 = 800 nodes in the Bayesian network (number of parents: 100); maximum 6 edges were allowed to each node (maximum number of parent nodes: 6) to generate 50 child solutions in each generation (number of children: 50).

3.4.6 Results

In this section, we report the results of BOA and GA applied to Koma history matching. Figure 3.13 shows the minimum misfit reached after 1000 simulations with identical random starting conditions for 5 separate experiments comparing BOA with GA. In each case, BOA has obtained a lower misfit than GA. In Figure 3.14, we plot the minimum generational misfit, averaged over the 5 experiments shown in Figure 3.13, against generation. The misfit found by the BOA at each generation is always lower than that found by the GA. When we compare the number of function evaluations that BOA takes to reach the minimum found by the GA after 1000 simulations, we can see that BOA is almost 1.5 times faster to achieve a given misfit than the GA. If we examine the box plots of generational misfit shown in Figure 3.14, we see that these conclusions also apply to average misfit and are robust.

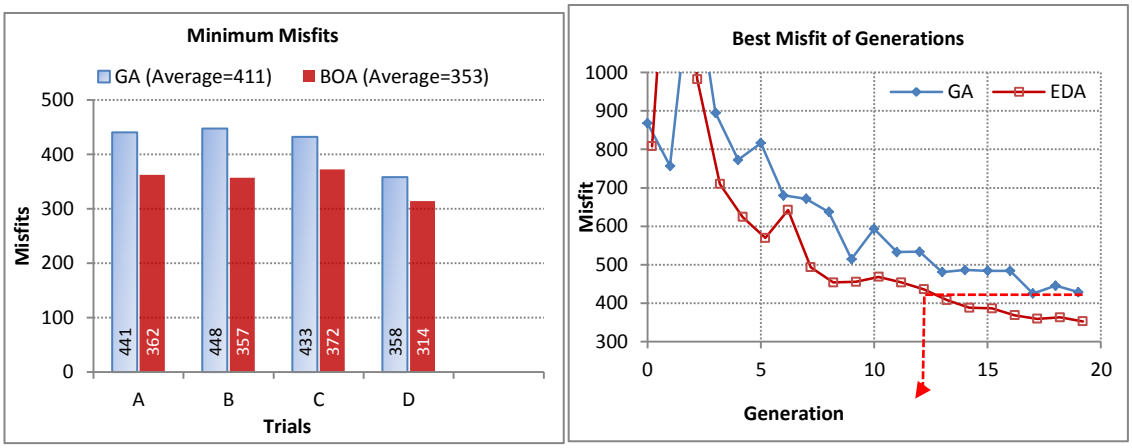


Figure 3.13: Minimum misfit of 5 experiments in GA and BOA (left) and convergence speed for the average of the best misfit of generations in 5 trials of the BOA and GA (right); BOA resulted in a lower minimum misfit in all 5 trials and about 1.5 times speed-up in convergence.

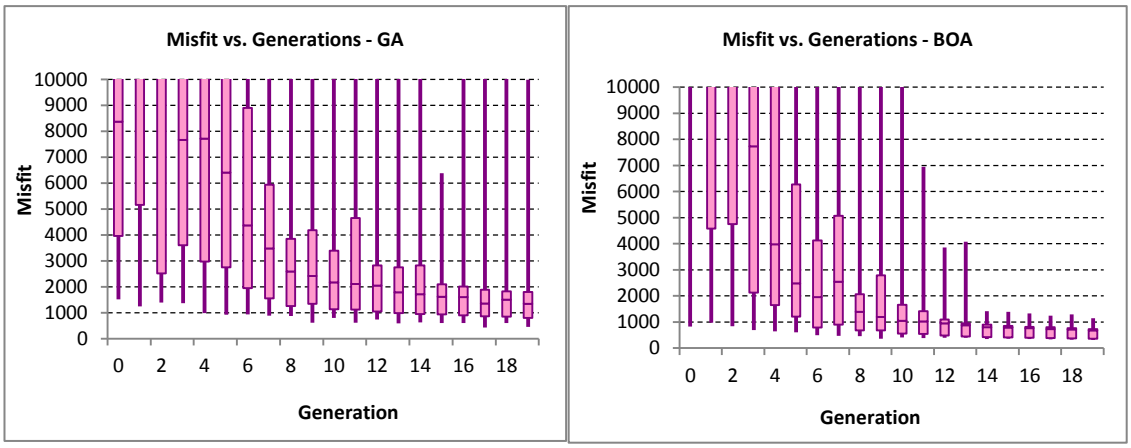
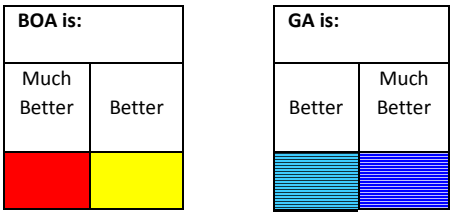


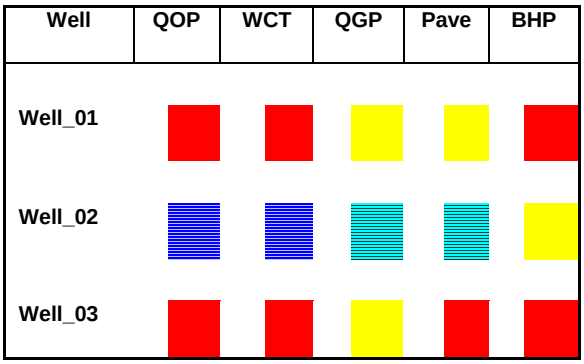
Figure 3.14: Box-plot of generational misfits from GA (left) and BOA (right); BOA results in lower average (middle mark of the boxes) and variation (top and bottom marks of the boxes), and range (top and bottom of the bars) for misfit in different generations.

The numbers shown in Figure 3.13 and Figure 3.14 are the total misfit and represent a sum of all the match quality components over wells and data types. Table 3.9 shows a comparison of the performance of BOA against GA for the 3 wells and 5 datatypes, bottom-hole pressure (BHP), average pressure (PAVE), oil production rate (QOP), gas production rate (QGP), and water-oil ratio (WCT). Out of 15 match quality components, 11 are better matched by BOA and only 4 by GA. In 7 of the quantities, the BOA match is much better, whereas GA is only much better for 2 quantities, in Well_02.

Figure 3.15 to Figure 3.19 show the matches for selected wells and match quality components, comparing GA and EDA in detail. The wells and match quality components were selected to give examples of each of the categories in Table 3.9.

Table 3.9: Summary of the GA and BOA match quality comparison.





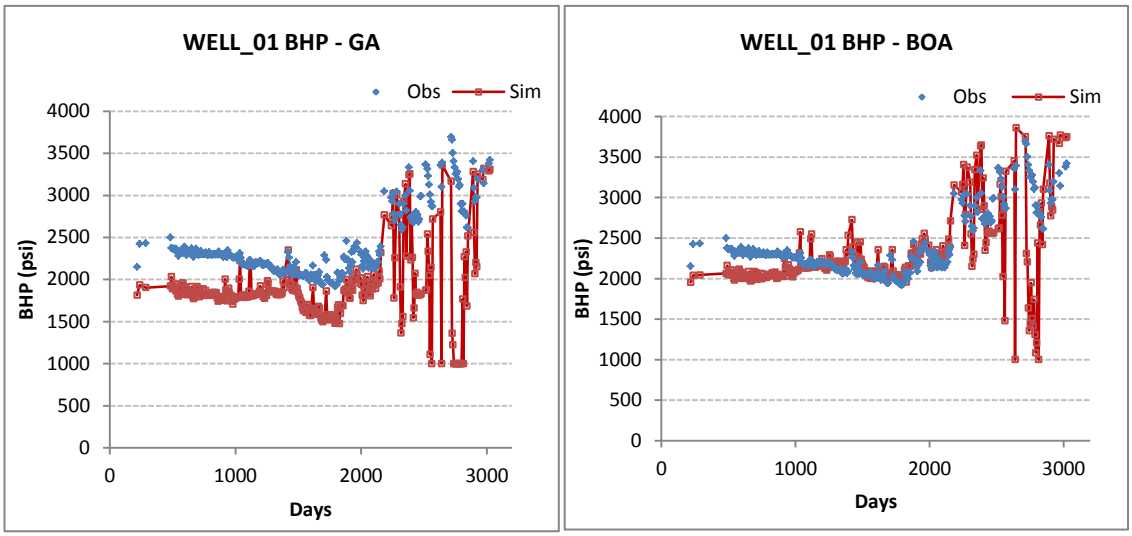


Figure 3.15: Observation and Simulation BHP of Well_01 for GA (left) BOA (right); BOA resulted in much better match.

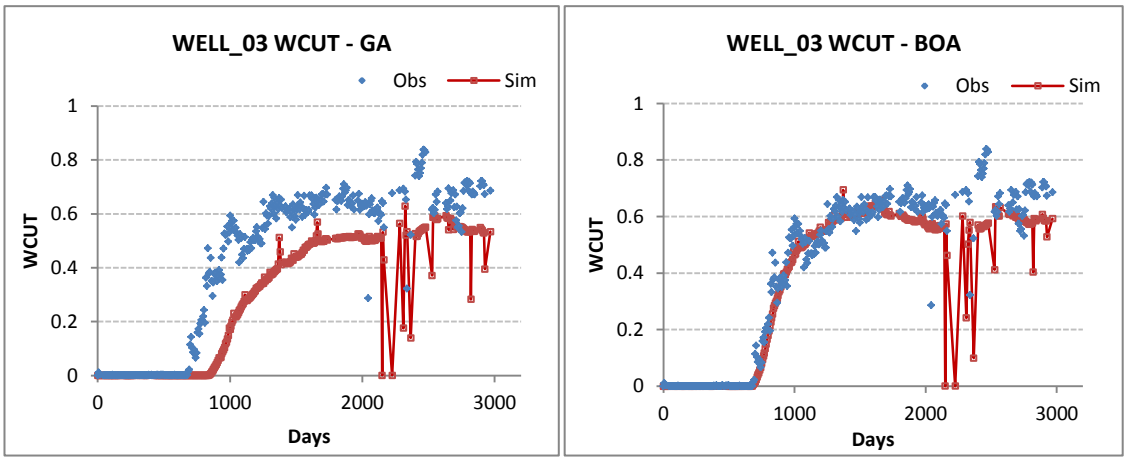


Figure 3.16: Observation and Simulation WCT of Well_03 for GA (left) BOA (right); BOA resulted in much better match.

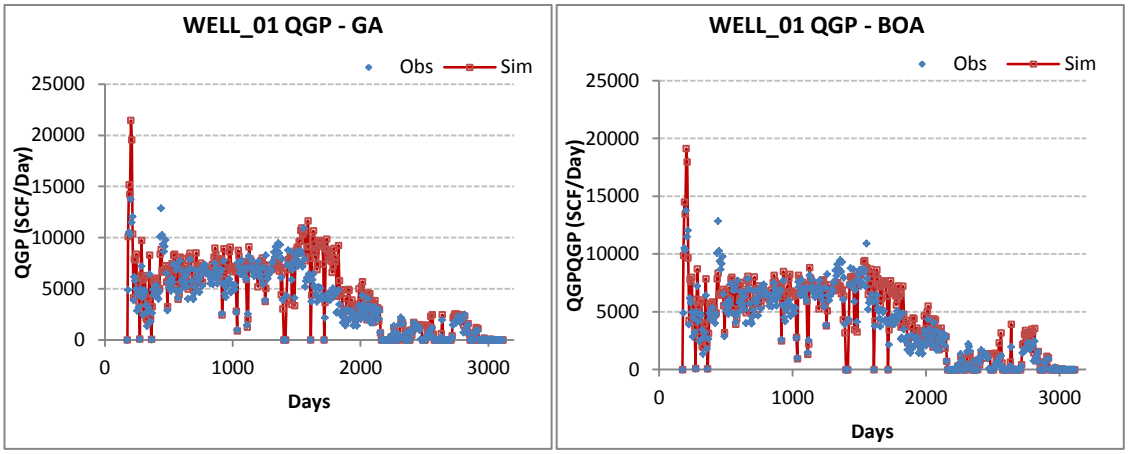


Figure 3.17: Observation and Simulation QGP of Well_01 for GA (left) BOA (right). BOA resulted in slightly better match.

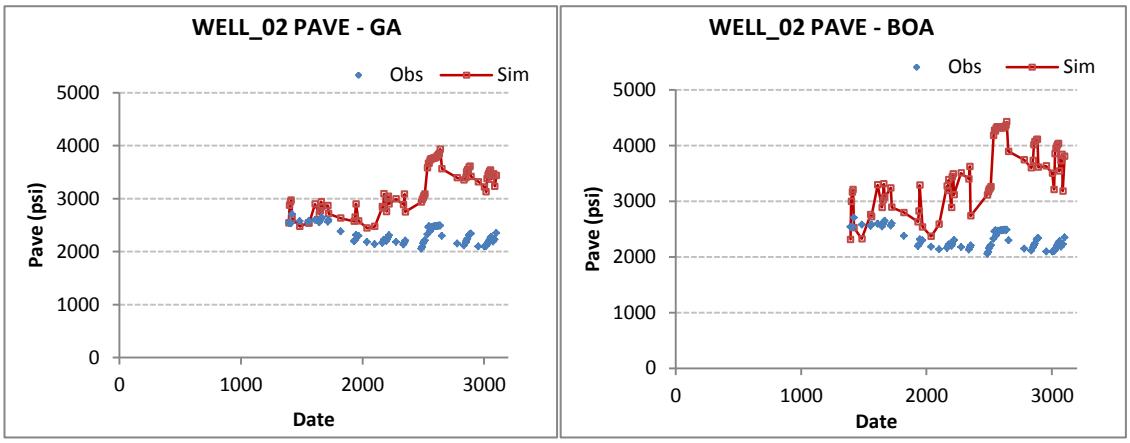


Figure 3.18: Observation and Simulation Pave of Well_02 for GA (left) BOA (right); GA resulted in slightly better match.

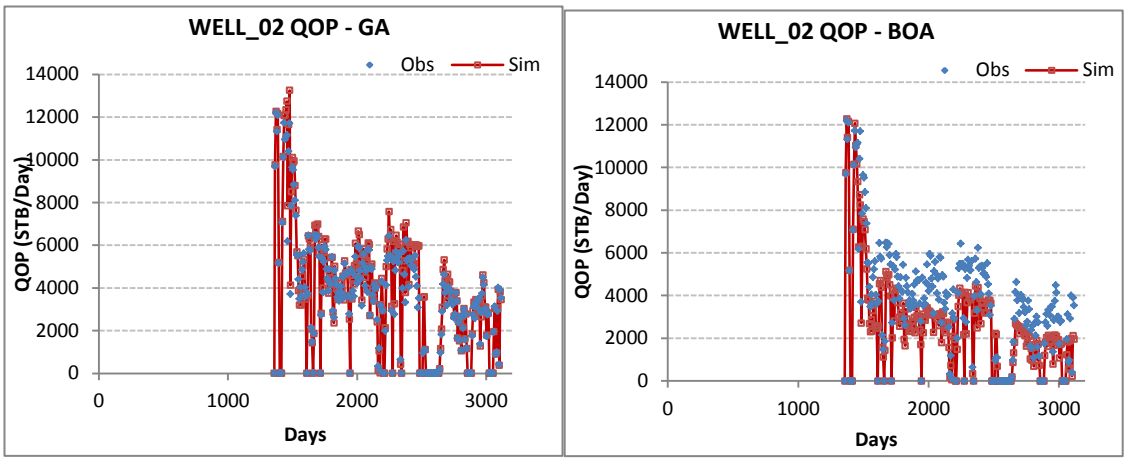


Figure 3.19: Observation and Simulation QOP of Well_02 for GA (left) BOA (right); GA resulted in much better match.

3.5 Discussion

In current chapter, we introduced and applied two univariate and one multivariate EDA to two history matching problems. The univariate EDAs use a simple bin-based

discretization for representing the continuous variables and marginal histogram models for the probabilistic modeling. Thus, they are not able to consider the interaction between the variables or the bins of variables.

However the multivariate EDA, BOA, uses bit-string representation for the continuous variables and Bayesian networks for the probabilistic modeling. In result, as in standard binary GA, BOA is able to model complex interactions between the variables in terms of the relationships between the pairs of bit-strings from two variables. Such complex relationships may not be common in history matching problems and well-understood by the reservoir engineers, but if present, BOA can discover those relationships and take the advantage of them in building the new solutions, which in turn, improves the search in terms of convergence and quality of solutions.

Through the results in the PUNQ-S3 application, we observed that univariate EDAs perform better than a BOA under smaller population size. This is because BOA exploits linkage information, and in problems with no or weak interactions among parameters, such as PUNQ-S3, it is more likely to become trapped in local minima, as it assigns part of its decision weight to relationships which might not be significant enough. One can argue that driving interactions between large numbers of parameters requires a larger number of solutions in the parent population. Thus in the exploitative set-up of algorithms, univariate models have better performance. Increasing the parent population size, however, improved performance of BOA, as observed in the explorative set-up.

To predict uncertainty bounds from the ensemble obtained in multimodal history matching problems, a diverse population of solutions is needed, where a global optimum and possibly multiple local optima are present. Sampling from a population created by an algorithm set-up that has prematurely converged will yield a bias in the predicted uncertainty bounds, since the posterior sampler samples too many solutions around the local optimum. A sufficiently explorative set-up of BOA ensures convergence toward the global optimum. However univariate EDAs are more likely to get trapped in local minima.

PUNQ-S3 parameterized using 45 distinct porosity regions is an example of separable problems where both univariate and multivariate EDAs perform well. Histogram-based

models (BH and EAH) have an intrinsic multimodality characteristic, which makes them convenient to approximate the solution distribution of complex and multimodal continuous problems. In PUNQ-S3, BH and EAH algorithms perform well, since the problem is sufficiently independent to be unaffected by correlations. In more complicated, non-separable problems with strong linkage information, misfit is based on the nonlinear combination of various parameters, so univariate EDAs fail to estimate joint probability distribution function properly, and in such cases BOA will perform better.

In the Koma history matching application, we saw that BOA outperformed a binary GA in terms of convergence speed and robustness of the history matching results. The reason is, it prevents the destruction of high quality schemata that occur when crossover is applied in GA, allowing BOA to find better quality matches in fewer simulation runs than GA. Although BOA tends to be more exploitative than explorative in nature, with the use of less elitist methods in the selection process, a better balance between exploration and exploitation can be achieved. Another possible weakness of BOA is the high computational complexity (seconds or few minutes) of the optimization of the Bayesian network in each generation. However, given the computational expense of function evaluations in history matching (usually hours and days), this is not a serious concern.

3.6 References

Baluja S. (1994). Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. Tech Rep CMU-CS-94–163, Carnegie Mellon University, Pittsburgh, PA.

Barker, J., Cuypers, M., & Holden, L. (2001). Quantifying Uncertainty in Production Forecasts: Another Look at the PUNQ-S3 Problem. SPE Number: 74707-PA. SPE Journal, 6 (4), 433-441.

Boss, C. (1999). Production forecasting with Uncertainty Quantification. Technical Report. Netherlands Institute of Applied Geoscience, TNO.

Coats, K., Dempsey, J., & Henderson, J. (1970). A New Technique for Determining Reservoir Description from Field Performance Data. SPE Number: 2344-PA. SPE Journal, 10 (1), 66-74.

Demyanov, V., Subbey, S., & Christie, M. (2004). Neighbourhood Algorithm with Geostatistical Simulations for Uncertainty Quantification Reservoir Modeling: PUNQ-S3 Case study. 9th European Conference on Mathematics in Oil Recovery ECMOR IX. Cannes, France.

Etxeberria R. & Larrañaga P. (1999). Global optimization using Bayesian networks. Proceedings of the Second Symposium on Artificial Intelligence (151–173). CIMAF-99.

Floris, F., Bush, M., Cuypers, M., Roggero, F., & Syversveen, A.-R. (2001). Methods for Quantifying the Uncertainty of Production Forecasts. Petroleum Geoscience, 7, 87-96.

Goldberg, D. E. (1989). Genetic Algorithms in Search Optimization and Machine Learning. Boston, MA, USA: Addison Wesley Longman Publishing Co., Inc.

Hajizadeh, Y. (2010). Ants Can Do History Matching. SPE Number: 141137-STU. SPE Annual Technical Conference and Exhibition, 19–22 September 2010. Florence, Italy.

Hajizadeh, Y., Christie, M.A., & Demyanov, V. (2010). Comparative Study of Novel Population-Based Optimization Algorithms for History Matching and Uncertainty Quantification: PUNQ-S3 Revisited. SPE Number: 136861-MS. Abu Dhabi International Petroleum Exhibition and Conference. Abu Dhabi.

Harik G.R., Lobo F.G., & Goldberg D.E. (1999). The compact genetic algorithm. IEEE Transactions on Evolutionary Computation, 3, 287–297.

Heckerman, D., Geiger, D. and Chickering, D. M. (1994). Learning Bayesian networks: The combination of knowledge and statistical data. Technical Report MSR-TR-94-09. Redmond, WA: Microsoft Research.

Jahns, H. (1966). A Rapid Method for Obtaining a Two-Dimensional Reservoir Description from Well Response Data. SPE Number: 1473-PA. SPE Journal, 6 (4), 315-327.

Mohamed, L., Christie, M.A., & Demyanov, V. (2010). Comparison of Stochastic Sampling Algorithms for Uncertainty Quantification. SPE Number: 119139-PA. SPE Journal, 15 (1), 31-38.

Mühlenbein H. & Paaß G. (1996). From recombination of genes to the estimation of distributions. Binary parameters. Lecture Notes in Computer Science 1411: Parallel Problem Solving from Nature, PPSN IV, 178–187.

Mühlenbein H., Mahnig T., & Ochoa A. (1999). Schemata, distributions and graphical models in evolutionary optimization. Journal of Heuristics. 5, 213–247.

Pelikan, M., Goldberg, D. E., & Cantu-Paz, E. (1999). BOA: The Bayesian Optimization Algorithm. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela. Proceedings of the Genetic and Evolutionary Computation Conference (525-532). Morgan Kaufmann.

Pelikan, M., Goldberg, D. E., & Cantu-Paz, E. (2000). Linkage Problem, Distribution Estimation and Bayesian Networks. Evolutionary Computation, 8 (3), 311-340.

Pelikan, M., Hauschild, M.W., & Lobo, F.G. (2012). Introduction to estimation of distribution algorithms. MEDAL Report No. 2012004.

Petrovska, I., & Carter, J. N. (2006). Estimation of Distribution algorithms for History matching. 10th European Conference on the Mathematics of Oil Recovery. EAGE, Amsterdam, Netherland.

PUNQ-S3 Test Case. (2010). Retrieved 7 1, 2010, from Department of Earth Science and Engineering Department Website - Imperial College: http://www3.imperial.ac.uk/earthscienceandengineering/research/perm/punq-s3model

Sambridge, M. (1999). Geophysical Inversion with a Neighbourhood Algorithm -II Appraising the ensemble. Geophysical Journal International, 138, 727-746.

Sultan, A., Ouenes, A., & Weiss, W. (1994). Automatic History Matching for an Integrated Reservoir Description and Improving Oil Recovery. SPE Number: 27712-MS. Permian Basin Oil and Gas Recovery Conference. Midland, Texas, U.S.A.

Thomas, K., Hellums, L., & Reheis, G. (1971). A Nonlinear Automatic History Matching Technique for Reservoir Simulation Models. SPE Number: 3475-PA. SPE 46th Annual Fall Meeting. New Orleans, U.S.A.

CHAPTER 4:
HISTORY MATCHING USING MULTI-OBJECTIVE OPTIMISATION

> "An idea is nothing more or less than a new combination of old elements."
>
> Vilfredo Pareto (1848–1923)

4.1 Introduction

Multiobjective evolutionary algorithm (MOEA) refers to EAs designed to find optimal solutions to problems having multiple objectives. Using multiple objectives almost invariably means that the optimal solution is not unique. Instead, a set of optimal solutions can be found, with different trade-offs between the objectives. An MOEA searches for this set of solutions, known as Pareto-optimal solutions, instead of a single optimum. Many real-world search and optimisation problems are expressed in terms of multiple objectives, for example, maximising predicted returns while minimising risk or maximising strength while minimising weight. In multiobjective optimisation, objectives are conflicting, i.e. the optimum solution according to one objective is not optimal according to another, whenever the optimal set of parameters for one objective is not as optimal for the other. Even for the problems that are not truly multiobjective, there are attempts in the literature to multiobjectivate the single objective to take the benefit of multiobjective optimisation (e.g. Knowles et al. 2001)

The first published work on multiobjective evolutionary optimisation was by Schaffner (1984). Schaffner, in a method called Vector Evaluated Genetic Algorithm (VEGA),

used proportional selection, according to each objective in turn, to create a number of sub-populations. Horn et al. (1994) proposed a new multiobjective selection scheme (NPGA) based on a Pareto domination tournament and equivalence class sharing. Subsequently Fonseca & Fleming (1995) introduced MOGA in which Pareto domination, rank-based fitness assignment and Niche-formation methods are used to sort solutions. In another novel technique, Zitzler and Thiele (1999) proposed the Strength Pareto Evolutionary Algorithm (SPEA) with the combination of elitism and the concept of non-domination. Knowles & Corne (2000) introduced the Pareto Archived Evolution Strategy (PAES). Another popular technique is Srinivas and Deb's Non-dominated Sorting Genetic Algorithm (NSGA) (1995), in which, all solutions are classified into a level of non-domination. For a complete review of the most popular multiobjective selection methods for evolutionary algorithms refer to Ghosh & Dehuri (2005).

Attempts to use these types of algorithms in multi-objective history matching are reported in the literature. For example, Schulze-Riegert et al. (2007) used the Strength Pareto Evolutionary Algorithm (SPEA). Ferraro and Verga (2009) applied the multi-objective genetic algorithm and evolution strategies for history matching and uncertainty quantification of the PUNQ-S3 synthetic case. In another application, Hajizadeh et al. (2011) used Differential Evolution for Multiobjective Optimisation based on MOGA Pareto Ranking (DEMOPR) for history matching of the PUNQ-S3 synthetic case. Mohamed et al. (2011) used Multi-Objective Particle Swarm Optimisation (MOPSO) based on NSGA2 ranking technique for history matching of the IC Fault model. However, no attempt is reported in the literature to use multiobjective EDAs for history matching.

The rest of this chapter is organised as follows: first, the methodology section discusses multiobjective optimisation, and introduces a common multiobjective sorting algorithm: the elitist Non-dominated Sorting Genetic Algorithm (NSGA2). The strategy for the comparison of the different algorithms is then described. This is followed by results of two case studies, the first study applies multiobjective Bayesian optimisation algorithm (MBOA) to the history matching problem of the PUNQ-S3 synthetic case. The second study presents and analyses the experimental results of MBOA when used for history

matching of the Koma, a real North Sea field. The generality and validity of results is discussed in the discussion section.

4.2 Methodology

As we discussed in the previous chapters, EDAs are a class of population based EAs. Like any other population-based EA, to solve an optimisation problem using an EDA, the problem is described using a vector of parameter values which represents a solution for the problem. As in other population-based algorithms, a set of N solutions are created and considered as a population. All the solutions in the population are evaluated for their performance expressed in terms of an objective function. Solutions with lower or greater objective functions respectively in a minimisation or a maximisation problem are considered promising. The algorithm proceeds to select a set of promising solutions out of the population using the sorting algorithm to become the parents of the next generation.

Evolutionary algorithms typically incorporate two forms of selection: selection of parent solutions from the population, from which child solutions are generated; selection of the solutions that survive to form the population in the next generation of the algorithm. In older algorithms, child solutions would often replace the old population in this survival phase. However, it is now commonplace for old solutions to be permitted to survive within the population for many generations, provided they are of sufficient quality. This elitism can be particularly useful in multiobjective algorithms, providing added selection pressure and encouraging the production of better solutions rather than merely a more diverse set of solutions.

In this chapter, we describe two sorting and selection mechanism for BOAs, single and multiobjective. A BOA is a multivariate EDA, discussed in Chapter 3 (see Pelikan et al. 2000). Both forms are highly elitist; i.e. solutions are first sorted in order of solution quality and then the best solutions are taken, both during parent selection and during the selection for survival. However, the two mechanisms differ in the method of sorting of the population. In single objective optimisation, the population is simply sorted according to the single objective function: fitness. In multiobjective optimisation, more sophisticated methods are required, since the dominance relation imposes ordering on

the population. The sorting algorithm used for selection in multiobjective BOA (MBOA) is that introduced in the elitist non-dominated sorting genetic algorithm (NSGA2), described in the next section.

4.2.1 Multiobjective Evolutionary Algorithms (MOEAs)

Multiobjective Evolutionary Algorithms have been employed for the past four decades; they are used to solve multiobjective optimisation problems where there is no single well-defined optimal solution and, in the presence of several objectives, several equally good solutions are available. These multiple optimal solutions are called non-dominated or Pareto optimal solutions. The general formulation of minimisation of a multiobjective problem is:

$$\begin{cases} \min\limits_{x}\{f = (f_1(x), \dots, f_m(x))\} \\ s.t.\ g(x) \leq 0 \\ h(x) = 0 \end{cases} \tag{4.1}$$

where $x = (x_1, \dots, x_n)$ is the vector of decision variables, and $f = (f_1, \dots, f_m)$ is the vector of objective functions. A multiobjective optimisation problem may have one or more constraint functions (g, h) which constrain the search space to a set of feasible solutions. s.t. stands for *subject to*.

4.2.1.1 Dominance concept

In multiobjective optimisation problems, we try to find multiple optimal solutions using *dominance* or the so-called *Pareto optimality* concept. If we wish to minimize each objective, a solution x is dominated by $\bar{x}$ if:

$$\begin{cases} f_i(x) \geq f_i(\bar{x}), & i = 1, \dots, m \\ f(x) \neq f(\bar{x}) \end{cases} \tag{4.2}$$

In simple words, solution $\bar{x}$ dominates solution x if, first, $\bar{x}$ is no worse that x in all objectives and, second, $\bar{x}$ is strictly better than x in at least one objective. If either of the two above conditions is violated, the solution $\bar{x}$ does not dominate solution x.

Given a set of solutions, any solution that is not dominated by any other in the set may be described as being *non-dominated* with respect to that set. If this set is the set of all

feasible solutions, then such solutions are *Pareto optimal*. Note that, in the absence of additional information, none of the Pareto optimal solutions can be said to be better than any other. Figure 4.1 demonstrates the concept of dominance and Pareto optimality in a bi-objective optimisation problem.

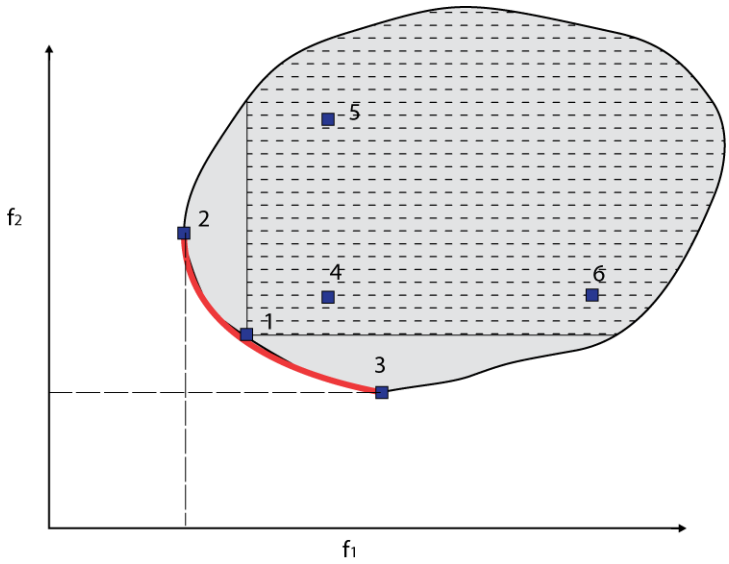


Figure 4.1: The concept of Pareto dominance for a bi-objective space (f1, f2) of a minimisation problem. The grey area shows the feasible search space. All the solutions in the dashed area (e.g. 4, 5, and 6) are dominated by solution 1; 2 does not dominate 3 and 3 does not dominate 2 since 3 is better than 2 with respect to objective f2 and 2 is better than 3 with respect to objective f1. All the solutions on the red line (e.g. 1, 2, and 3) are non-dominated Pareto optimal solutions.

The term *Pareto front* describes the image of all the Pareto optimal solutions in the objective space and is illustrated by the red line in Figure 4.1. Note that we do not expect a multiobjective EA to discover all, or indeed any of the Pareto-front, any more than we can guarantee that a single objective EA will discover the optimal solution. Rather, we hope that a search algorithm will approximate the Pareto front, both by obtaining solutions close to (or on) the front, and by obtaining a range of solutions across the front. Different multiobjective algorithms will have different performance, with regards to both of these aims.

The concepts of Pareto front and dominance are the basis of many of the sorting algorithms applied to multiobjective optimisation problems. The first attempt to use Pareto-based information in EAs was made by Goldberg (1989). More sophisticated approaches (e.g. NSGA and SPEA) sort solutions by, first, selecting solutions from the

non-dominated set and, then, if these solutions are not sufficient, selecting from the dominated solutions, using other approaches.

4.2.1.2 Elitist Fast Non-dominated Sorting Genetic Algorithm (NSGA2)

Since EAs work with a population of solutions, a simple EA can relatively easily be extended to search for a diverse set of solutions, such as the set of Pareto-optimal solutions. With an emphasis on moving toward the true Pareto-optimal region, an EA can be used to find multiple Pareto-optimal solutions.

The non-dominated sorting genetic algorithm (NSGA) was developed by Srinivas and Deb (1995) on the basis of the non-dominance fronts and sharing strategy between solutions. Criticisms of NSGA centred on high computational complexity, lack of elitism, and need for specifying the sharing parameter. To address all these problems, Deb et al. (2001) introduced an improved version of NSGA, called Elitist NSGA or NSGA2. In NSGA2, the sharing function approach was replaced by a crowding-distance comparison, which does not require any user defined parameter for maintaining the diversity. The non-dominated sorting algorithm of NSGA 2, used in MBOA, is as follows:

1. Assign a *rank* to each solution in the population equal to level of the non-dominated front, as follows:
   1.1. Initialise $j = 1$,
   1.2. Find non-dominated solutions in the population and assign them to front level j,
   1.3. Temporarily remove non-dominated solutions from the population,
   1.4. Increment j and continue from step 1.2 until all solutions have been assigned to a front.
   1.5. For each solution, i, in the current non-dominated front, compute a *crowding distance measure* in the fitness space based on the surrounding solutions, according to (see Figure 4.2):

$$C_i = \sum_{k=1}^{m} f_{after}^{k} - f_{before}^{k} \tag{4.3}$$

where C_i is the crowding distance measure and m is the number of objectives. f^k_{before} is the objective function value of objective k for the solution that comes before solution i, after the front is sorted according to object/+ive k, while f^k_{after} indicates the objective value for objective k for the solution that comes after i. Solutions with lowest and largest objective function values are assigned an infinite crowding distance. Objective function values must be normalised before these calculations.

2. Compare solutions in the entire population for the two attributes, rank and crowding distance, as follows:

    1.1. Between two solutions with different non-dominated ranks, solution with the lower (better) rank is preferred.
    1.2. Between two solutions belonging to the same front (same rank), solution with higher crowding distance (least crowded) is preferred.

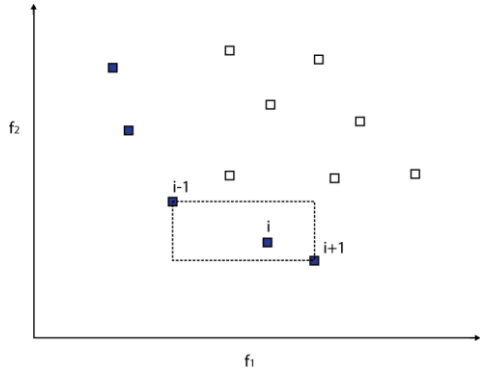


Figure 4.2: Crowding distance calculation for a bi-objective space (f1, f2) of a minimisation problem. Filled square points represent solutions of the same rank (here rank 1).

4.2.2 Strategy for Comparison Study of the Algorithms

In this chapter, we evaluate performance of the algorithms based on two factors: misfit convergence and diversity. For each of these, we use a related plot for the comparison study of different algorithms/experiments.

4.2.2.1 Misfit Convergence

As the primary goal in history matching using optimisation algorithms is to obtain solutions (reservoir simulation models) with the lowest value of misfit, we initially look

at the minimum misfit convergence plot for performance comparison of the different algorithms. For the multiobjective scheme a single value obtained for misfit which is the sum of the objective functions. We repeat the experiments for a number of trials and then average the results due to the stochastic nature of the evolutionary algorithms used here. Generational misfit, minimum misfit of a single solution obtained by the algorithm with 95% significance level t-test standard error as error bars, and boxplots (P25, P50, and P75 quartiles) of the misfit versus generation are reported in the results section.

4.2.2.2 Diversity Measure

Population diversity is a key measurement in population-based evolutionary algorithms. Diversity is an estimate of the levels and types of variety of individuals in a population. A diversity measure helps us to compare the diversity/convergence performance of the population based evolutionary algorithms in history matching. To study the diversity of solutions in the population we use an inertia based diversity measure (Morrison and Jong 2002). For this measure, first, the centroid of the population is calculated for all equally weighted parameters of the solutions in the population; then the moment of the inertia based diversity measure of these solutions about the calculated centroid is given by:

$$IDM = \frac{\sum_{j=1}^{P}\sum_{i=1}^{N}(x_{ij} - c_i)^2}{P};\ where\ c_i = \frac{1}{N}\sum_{i=1}^{N} x_{ij} \tag{4.4}$$

where:

- IDM is the inertia based diversity measure,
- i is the subscript for the variables number in solution x,
- N is the total number of variables in solution x,
- x is the variable value,
- c_i is the centroid of the variable,
- j is the subscript for the solution number in the population,
- P is the total number of solutions in the population.

It is worth pointing out that this measure shows the diversity of all the solutions in the current child population, not necessarily diversity of the good solutions with acceptable objective function value. One must use this diversity, along with the convergence plots, to infer the diversity of good solutions.

4.3 Application 1: History Matching of PUNQ-S3 Synthetic Case

We initially applied BOA and MBOA to the history matching problem of the PUNQ-S3 reservoir. For field description, uncertainty parameterisation, and misfit definition of PUNQ-S3 refer to Section 3.3 of Chapter 3. The objective here was to study the convergence performance and diversity of the single-/multi-objective scheme of BOA on a well-known synthetic case for history matching.

Multiple objectives in history matching can be defined by summing a sub-group of the misfit components instead of defining a single objective by summing all the misfit components. A misfit component can be defined as a data-type (e.g. oil rate, water-cur ratio, etc.), a data-level (e.g. well, field, region, etc.), or individual data-levels (e.g. well #1, well # 2, region #1, etc).

4.3.1 Revisited PUNQ-S3 parameterisation

In this chapter, we used a slightly different parameterization for PUNQ-S3 compared to that used in the previous chapter. The previous parameterisation (Hajizadeh et al. 2010) had not considered northwest-southeast orientation of the channels as given in the geological description. In addition, it had not considered the uncertainty in permeability-porosity correlations.

We took 38 unknown variables, using the given geological description and channel properties. We parameterized the porosity, using 8 homogenous regions for layers 1 and 2 and 6 homogenous regions for each of the layers 3 to 5, giving in total 34 porosity parameters (Figure 4.3). The number of regions is taken according to the width of the channels (see Table 3.1) and the dimensions of the reservoir. The first two top layers have a wider extension of the top structure map, so that they have one more region. Figure 4.4 shows how these regions match the porosity map of the truth case. It should

be noticed than no tuning was done for this match, and the figure is just given for confirmation of the parameterisation. As explained in the previous chapter, prior ranges for the porosities were calculated from the noise adjusted well porosities (Table 4.1).

Table 4.1: Porosity ranges for PUNQ-S3 layers.

Layer	Porosity
1	0.15 - 0.3
2	0.05 - 0.15
3	0.15 - 0.3
4	0.1 - 0.2
5	0.15 - 0.3

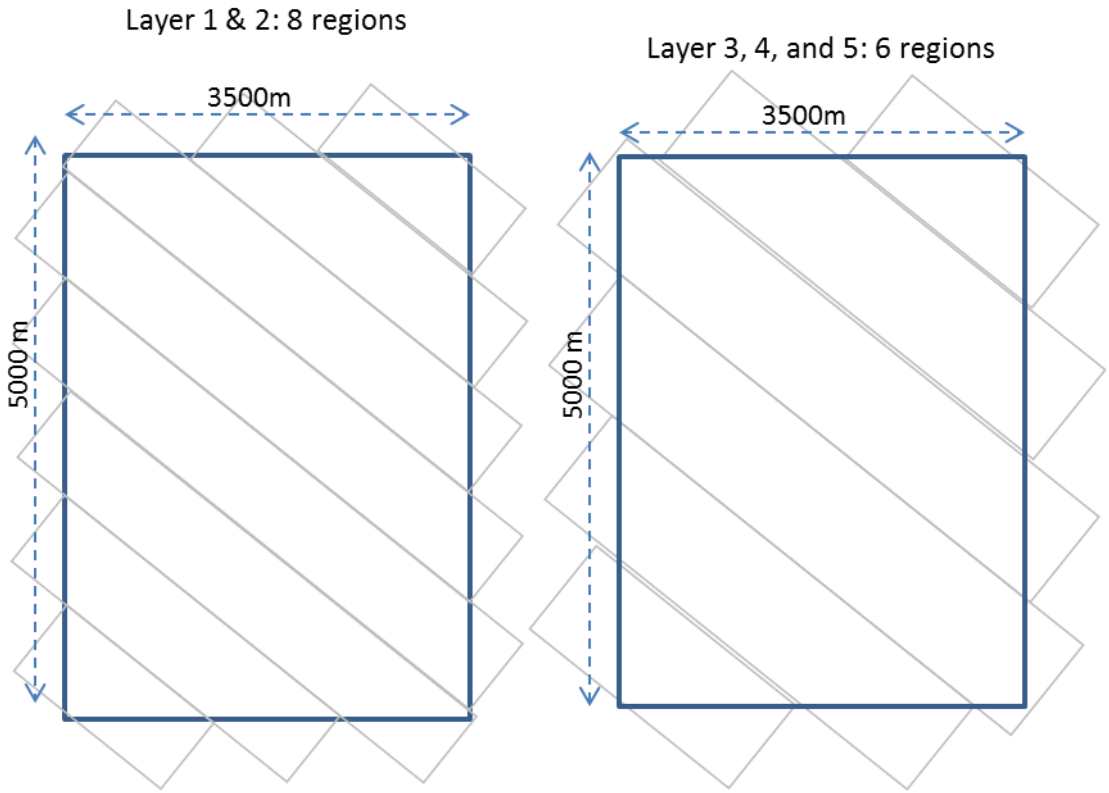


Figure 4.3: Distinct porosity regions for layers 1 and 2 (left) and layers 3, 4, and 5 (right), based on the width and spacing of the channels in revisited PUNQ-S3 parameterisation.

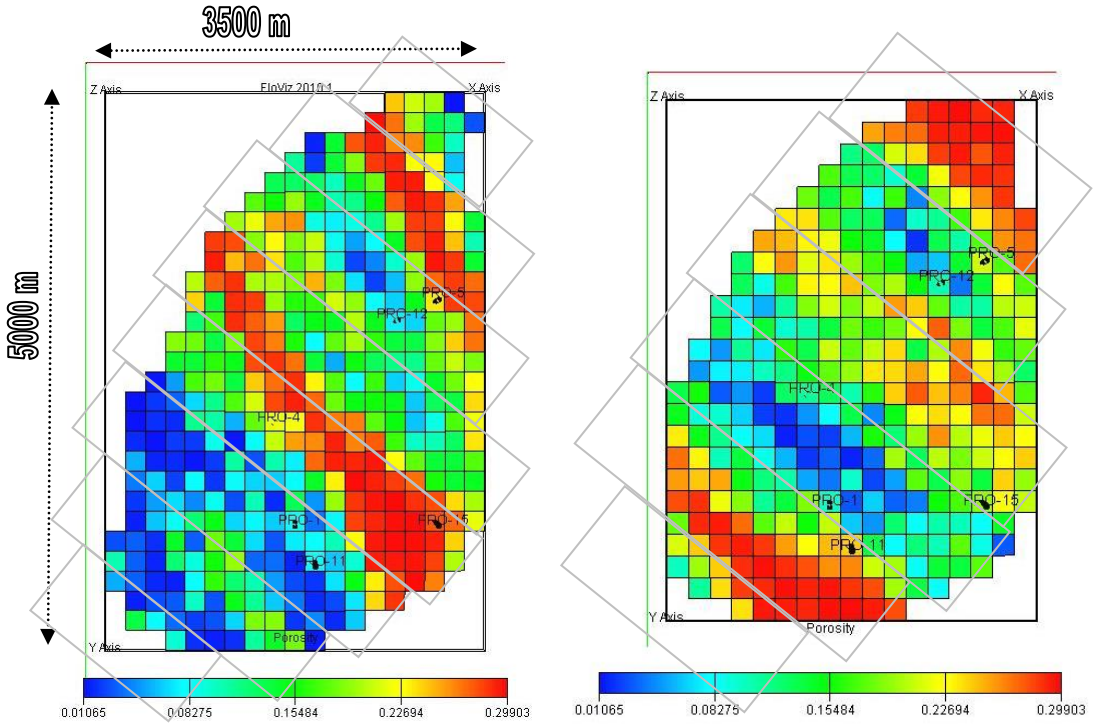


Figure 4.4: Chosen distinct porosity regions for layers 1 and 2 (left) and layers 3, 4, and 5 (right) match the truth case of PUNQ-S3.

Directional permeabilities are computed from the least square fitting of the well porosity/permeability cross-plots (Boss 1999):

$$\begin{cases} \ln(k_h) = \mathrm{b_h} + \mathrm{a_h}\emptyset \\ k_v = \mathrm{b_v} + \mathrm{a_v}k_h \end{cases} \tag{4.5}$$

Since these cross-plots are built only from the well values, the permeability fields are unknown, and hence we took two coefficients of the porosity / horizontal permeability correlation and two coefficients of the horizontal / vertical permeability correlation as uncertainty variables, with the prior ranges shown in Table 4.2.

Table 4.2: Prior ranges for the uncertainty variables related to the permeability correlations.

Parameter	Base value	Range
b_h	0.77	0.5 – 1.0
a_h	9.03	6.0 – 12.0
b_v	3.124	1.0 – 5.0
a_v	0.306	0.1 – 0.4

4.3.2 Evolution of uncertain parameters in assisted history matching

An assisted history matching study starts by creating the initial reservoir model and parameterisation of the uncertain variables of the model. In this section we summarise the parameterisation of the PUNQ-S3 model, as discussed above:

1. Take the given data in PUNQ-S3 model including the grid geometry, channels description, and porosity and directional permeabilities at the well locations.
2. Create a simulation model from the given G&G and dynamic data with porosity and directional permeabilities initialised to zero.
3. Divide the grid in each layer to North-West / South-East regions based on the channel width, length and spacing (Table 3.1 and Figure 4.3).
4. Take the porosity of each region as an uncertain variable (32 variables) and obtain the prior probability distribution (here uniform ranges) for porosity of each region based on the well values and the estimated level of measurement/interpretation errors (Table 4.1).
5. Consider the coefficients of the porosity-permeability correlations at the wells, equation (4.5), as uncertain variables (4 variables) and define their prior ranges based on the well values and the estimated level of measurement/interpretation errors (Table 4.2).

Now we take one of the uncertain variables (V1) as an example and show how the probability distribution of the selected uncertain variable (the histogram model in BHEDA) changes and evolves from the prior to the posterior distribution during the history matching process.

The initial population is a random generation of 120 solutions from the uniform prior distribution of P1. The variable range is discretised into 20 bins and each bin gets an equal probability of 1/3. Therefore the histogram model is flat (Figure 4.5).

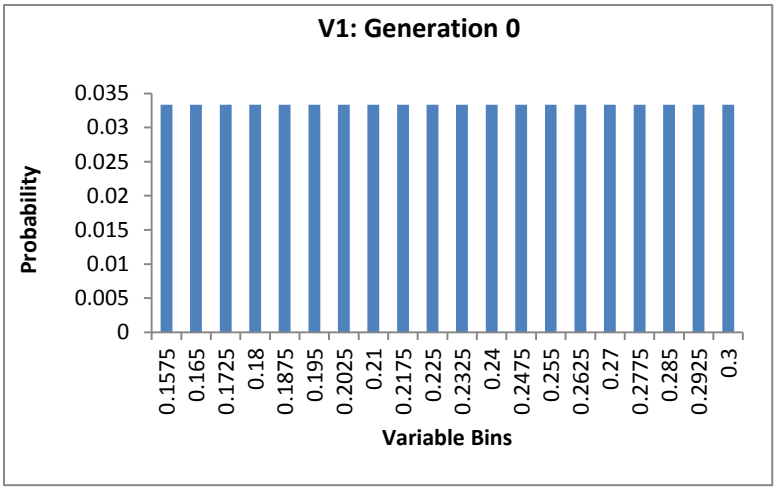


Figure 4.5: Histogram model of parameter V1 in initial generation. The population is created by random sampling of 120 values from the uniform prior range of V1.

In next generation (generation 1), top 40 solutions (in terms of lower values of match quality) out of the 120 solutions in initial generation are selected as the parents, and a histogram model is created out of them (Figure 4.6). The histogram was used to generate 40 new solutions (children).

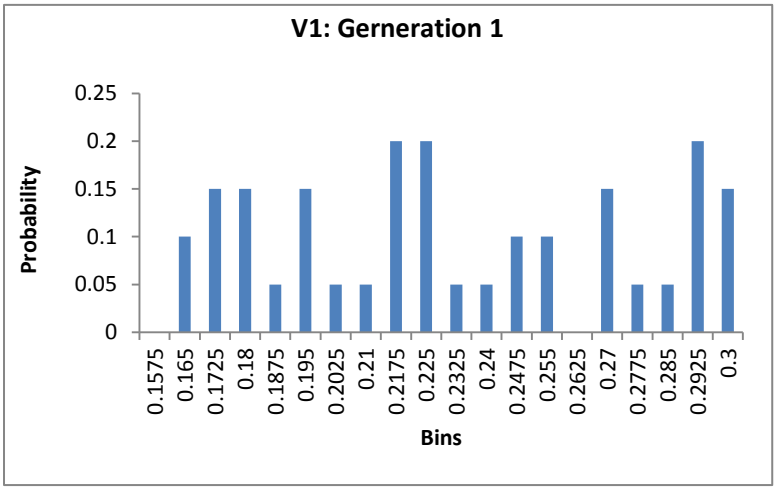


Figure 4.6: Histogram model of parents, top 40 solutions, for V1 in generation 1. The range is no longer random and some areas of the variable range have larger probabilities. This histogram model is used to sample 40 children.

This process is continued in the later generations. i.e. in each generation, top 40 solutions of the solutions created so far are selected as parents and a histogram model is fitted. Then 40 child solutions are sampled from the histogram model. Figure 4.7, Figure 4.8, Figure 4.9, and Figure 4.10 show the histogram model of the parents respectively in generations 5, 20, 45, and 72. As the figures show, the histogram model

of the parents and therefore the search is gradually converging to specific areas of the variable V1 range, which yield solutions with good match quality.

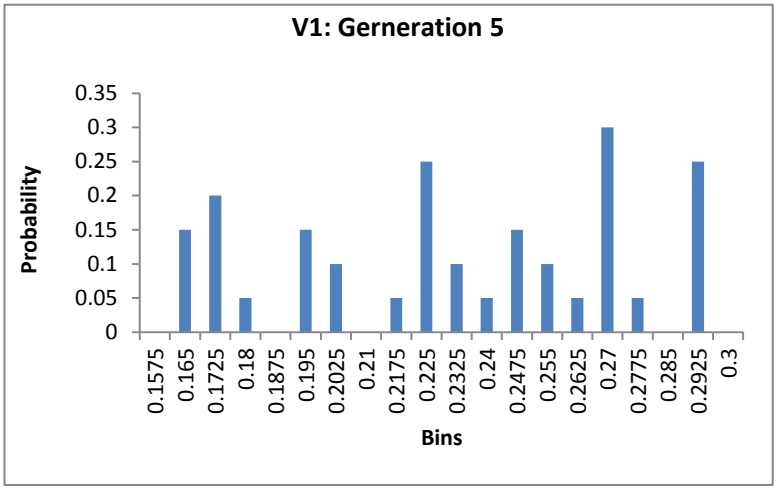


Figure 4.7: Histogram model of parents, top 40 solutions, for V1 in generation 5. A new histogram model is created in each generation.

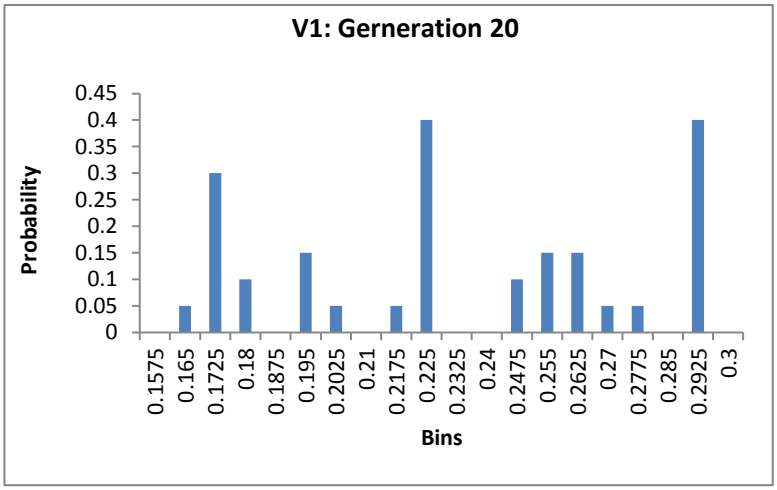


Figure 4.8: Histogram model of parents for V1 in generation 20. The search is more converged to specific areas of the search space.

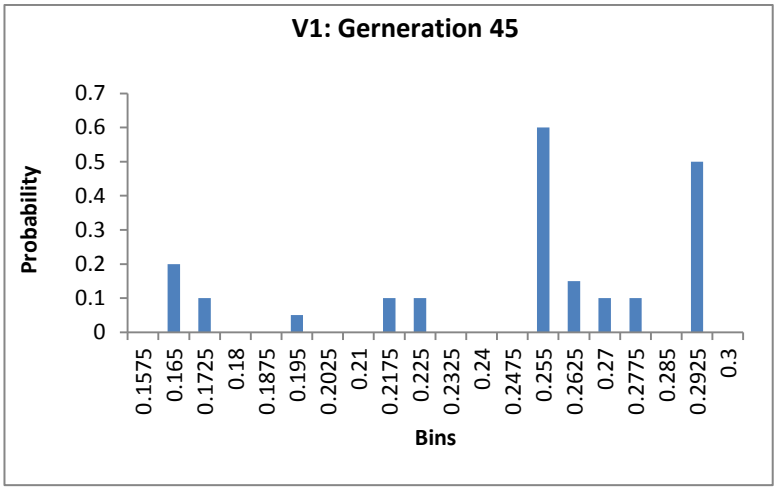


Figure 4.9: Histogram model of parents for V1 in generation 45. The search is more converged.

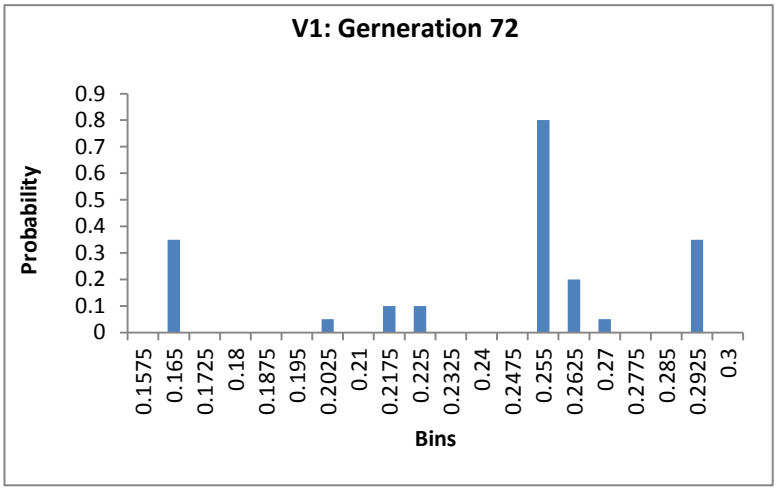


Figure 4.10: Histogram model of parents for V1 in generation 72. The search is finally converged to few regions.

4.3.3 PUNQ-S3 Results

For performance comparison of BOA and MBOA ten trials with the following control parameters were used: 60 solutions in the initial population, 40 parents and 20 children in each generation, maximum number of parents of each node in the Bayesian network equal to 6 and 10 bins, for discretising parameter values. The stopping criterion was set as the total number of 47 generations (1,000 function evaluations).

Initially we carried out a sensitivity study on different objective setups for MBOA. Four different setups for multiple objectives were considered and examined in PUNQ-S3 application as in Table 4.3.

Table 4.3: Grouping of wells for the objective function setups in PUNQ-S3.

Experiment	No. Of objectives	Wells	Comment
G1	2	Objective 1: Wells 5 & 12 Objective 2: Wells 1, 4, 11, 15	Two reservoir regions: north-west and centre-south
G2	2	Objective 1: Wells 1, 4, 5 Objective 2: Wells 11, 12, 15	No particular regions (as in Hajizadeh et al,. 2011)
G3	2	Objective 1: Wells 4, 5, 12 Objective 2: Wells 1, 11, 12	Two reservoir regions: north and south
G4	3	Objective 1: Wells 5 & 12 Objective 2: Wells 1 & 11 Objective 3: Wells 4 & 15	Three reservoir regions: north, centre, and south

Figure 4.11 shows the generational misfit convergence for the multiple objectives setups shown in the above table. Multiobjective optimisation using three objectives (G4) has not converged, since the Pareto front for three objectives in history matching becomes too complicated and two objectives is the maximum number of objectives that a multiobjective sorting algorithm can deal with in history matching. Between the two-objective setups, the grouping of wells to north and centre-south (G1), gives slightly better results than other two, perhaps because of maximum separation from other wells, based on the maximum distance between neighbouring wells. Thus G1 setup was taken for the future experiments. Figure 4.12 shows the inertia diversity of four settings. All four settings did indeed maintain diversity as they have not touched or become very close to the horizontal axis, which shows zero diversity.

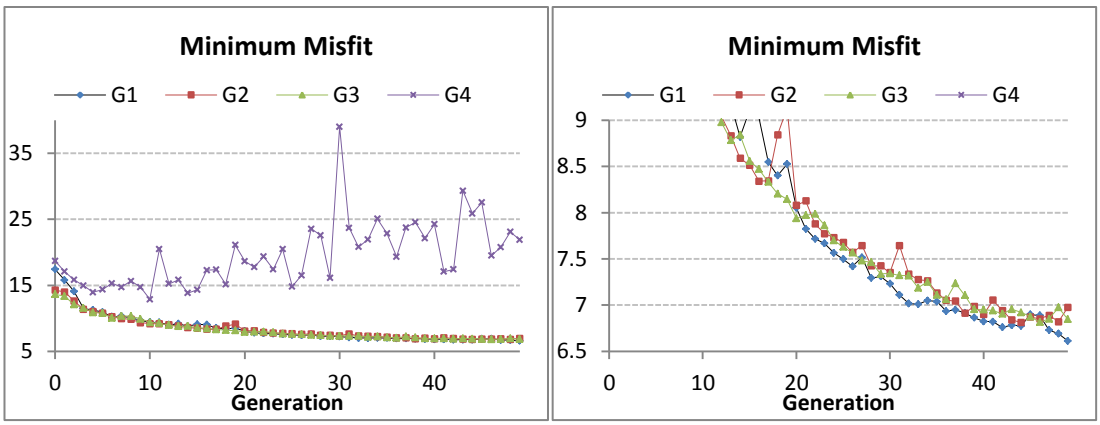


Figure 4.11: Average of minimum misfit found in each generation (left), zoomed for misfits under 10 (middle), and inertia diversity (right) for 10 trials of MBOA with four different multiple objective setups. G1 has slightly better convergence while G4 has not converged.

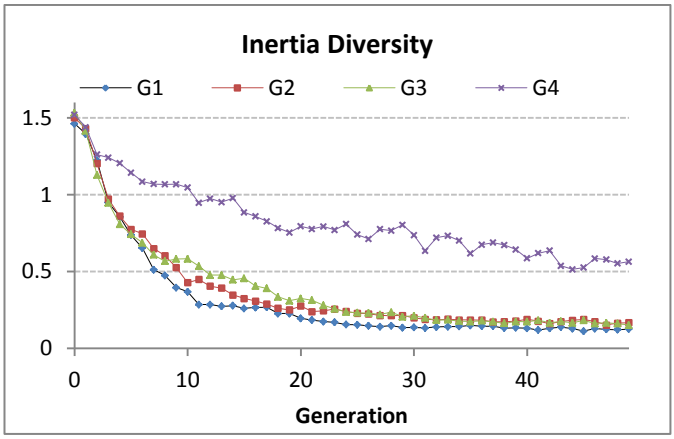


Figure 4.12: Diversity of different multiple objective setups for MBOA of PUNQ-S3. All four setups have maintained sufficient level of diversity.

Figure 4.13 shows the misfit convergence of BOA and MBOA with untuned control parameters; the figure shows that MBOA outperforms BOA for the minimum misfit. However MBOA produces too many low quality models. Figure 4.14 shows the diversity of the BOA and MBOA with tuned control parameters. As can be seen, untuned BOA suffers from an extensive loss of diversity, but MBOA better maintained diversity in the later generations.

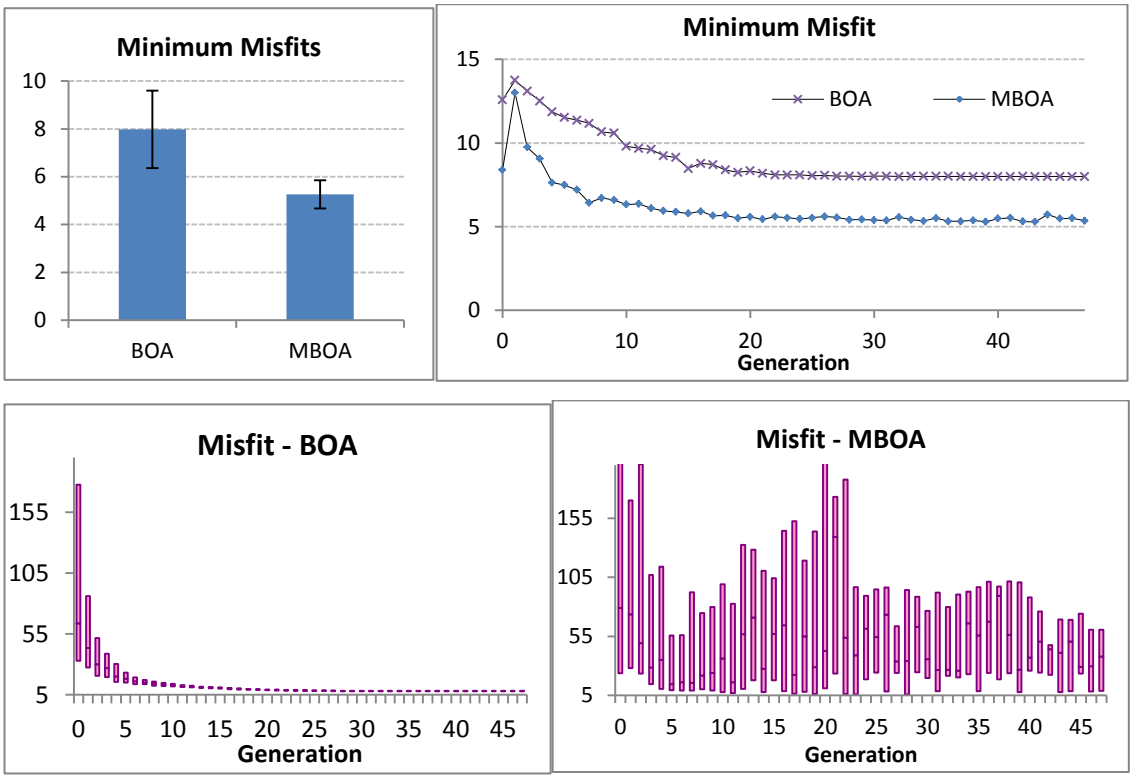


Figure 4.13: Average of minimum misfit found in 10 trials with 95% t-test standard error (top-left), average of minimum misfit found in each generation for 10 trials (top-right), boxplot of averaged misfit per generation for BOA (lower-left) and MBOA (lower-right-) with untuned control parameters for PUNQ-S3 model. MBOA outperforms BOA for the minimum misfit; MBOA produces too many low quality models as shown by the boxplot of misfit.

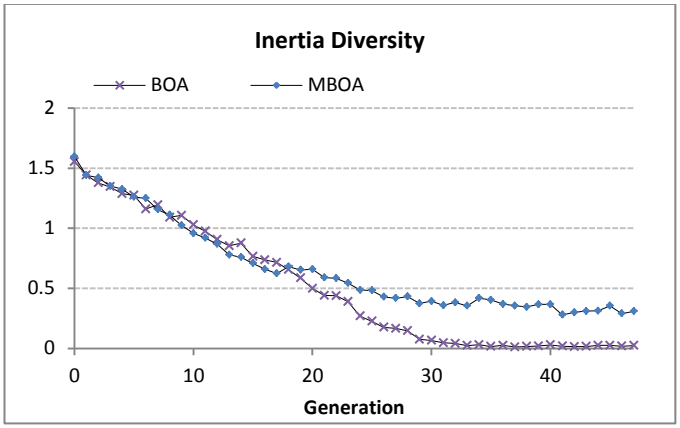


Figure 4.14: Diversity comparison of BOA and MBOA with untuned control parameters; BOA suffers from the extensive loss of diversity and MBOA better maintained the diversity from generation 18.

After tuning the BOA control parameters, we repeated the experiments for BOA and MBOA again, with 10 trials with different seed numbers. Tuned control parameters are: 150 solutions in the initial population, 100 parents and 50 children in each generation, maximum number of parents of each node in the Bayesian network equal to 6, and 20 bins for discretising parameter values. The stopping criterion was set as the total number of 47 generations, resulting in a total of 3,000 function evaluations.

Figure 4.15 shows the misfit convergence of BOA and MBOA with tuned control parameters; it shows that the misfit convergence improvement of the MBOA over BOA in this case is not significant. Figure 4.16 shows the diversity of the BOA and MBOA with tuned control parameters, where the diversity of both BOA and MBOA is improved compared to the untuned setup. Both BOA and MBOA maintained diversity throughout the evolution.

For multiobjective problems, it is common to compare non-dominated fronts obtained by the algorithm. Convergence of this front with the generation is a performance measure of these multiobjective optimisation algorithms. Thus, we looked at the non-dominated front generated by each setup of the objective function used for MBOA. As Figure 4.17 shows, the G1 setup results in the best non-dominated front, where more solutions form the front.

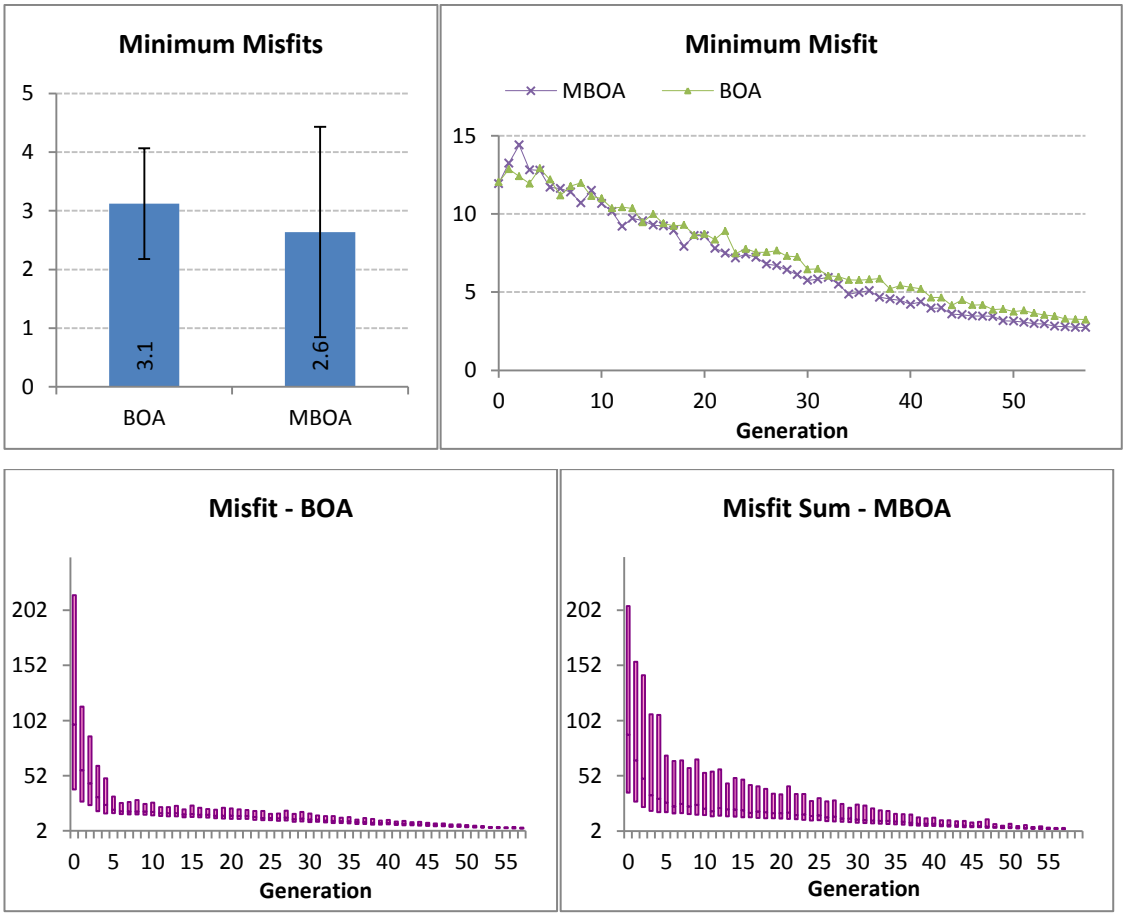


Figure 4.15: Average of minimum misfit found in 10 trials with 95% t-test standard error (top-left), average of minimum misfit found in each generation for 10 trials (top-right), boxplot of averaged misfit per generation for BOA (lower-left) and MBOA (lower-right) with tuned control parameters for the PUNQ-S3 model. The misfit convergence improvement of the MBOA over BOA is not significant; MBOA produces models with a broader range of misfits.

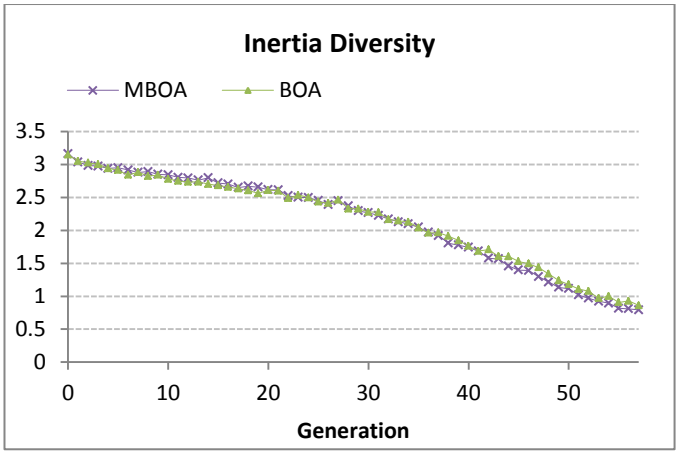


Figure 4.16: Diversity of BOA and MBOA with tuned control parameters; Diversity of BOA and MBOA is improved compared to the untuned setup. Both BOA and MBOA maintained diversity throughout the evolution.

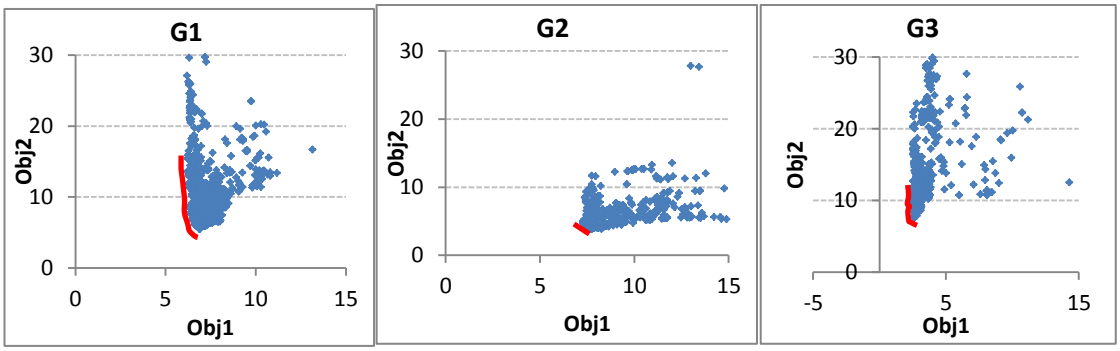


Figure 4.17: Solutions obtained by bi-objective setups of MBOA in PUNQ-S3 (blue diamonds) and finally formed non-dominated fronts (red line).
G1 shows the non-dominated front with the greatest number of solutions on the front, so best setup in terms of multiobjective criteria.

4.4 Application 2: History Matching of Koma Field

After PUNQ-S3 application, MBOA was applied to the Koma model (real North Sea full-field). In the previous chapter, we applied single objective BOA to history match the Koma field and showed that performance improvement was achieved by using BOA when compared with results achieved by a commercial GA for history matching. Unlike GA, BOA (a multivariate EDA) implicitly adapts the search mechanism to the structure of the problem by updating the search mechanism in each generation.

In current chapter, we evaluate the performance of MBOA in real world conditions and compare its results with the results obtained by the single objective BOA in terms of misfit convergence and diversity preservation. As discussed earlier, by using multiobjective sorting algorithm instead of sorting based on the overall misfit, we aim to use the benefits of multiobjectivisation and trade-offs between the misfit components. Thus we could possibly improve the diversity and convergence in the search process.

4.4.1 Uncertainty parameterisation

In the previous chapter, we used an uncertainty parameterisation for Koma which was similar to the original uncertainty study performed by the asset team. We also used a misfit definition similar to that in the PUNQ-S3 case, with the same misfit components used by the asset team. No further observation data analysis or tuning of the uncertainty variables was carried out.

In this chapter, we revisited Koma uncertainty parameterisation and misfit definition, so one could expect even better history matching in the results of applying the algorithms. In a sensitivity study and analysis of the 54 model parameters taken in the previous chapter, we reduced the number to 49 uncertainty parameters by removing 5 non-sensitive and low impact parameters. The final parameters included 14 pore volume multipliers (porosity and NTG), 6 permeability multipliers, 21 inter-region transmissibility multipliers, 7 fault transmissibility multipliers, and 1 aquifer volume multiplier.

4.4.2 Data analysis

We also performed a quality control study on the observation data to refine the misfit definition function. Koma observation data includes reliable bottom-hole flowing pressure and average gridblock pressures, as well as not particularly reliable allocated oil, gas, and water rates. Therefore, following data analysis was conducted:

4.4.2.1 Water-cut analysis

In a standard misfit function definition, usually the mismatch between the observation and simulation data for water-cut and gas-oil ratios or water, oil, and gas rates is considered, regardless of the shape of the curves. Water and gas breakthrough times play an important role in the response of the model, and hence two models with the same misfit values but different breakthrough time behave differently in terms of the prediction performance. Small amounts of water sometimes exist in the production rates, even before the breakthrough time. Yortsos et al. (1999) developed an approach to identifying the correct breakthrough time in waterflooding. Their approach involves plotting the water-cut ratio in log-log scale and finding the time at which the slope of the curve starts to increase from about 1 to a very steep slope (m>>1).

For the Koma field, a complete set of well-test data (including bottom-hole well pressure, average reservoir pressure from extended build-up pressure tests, and well rates from production log tests) was available, which are more accurate than the allocated rates. Well-test water rates are zero for times before the breakthrough. Although the frequency of the well-test data is not as high as the allocated weekly rates,

they were frequent enough to be taken into the misfit calculations. Therefore, for the Koma field, we used well-test data instead of allocated data for water and gas rates.

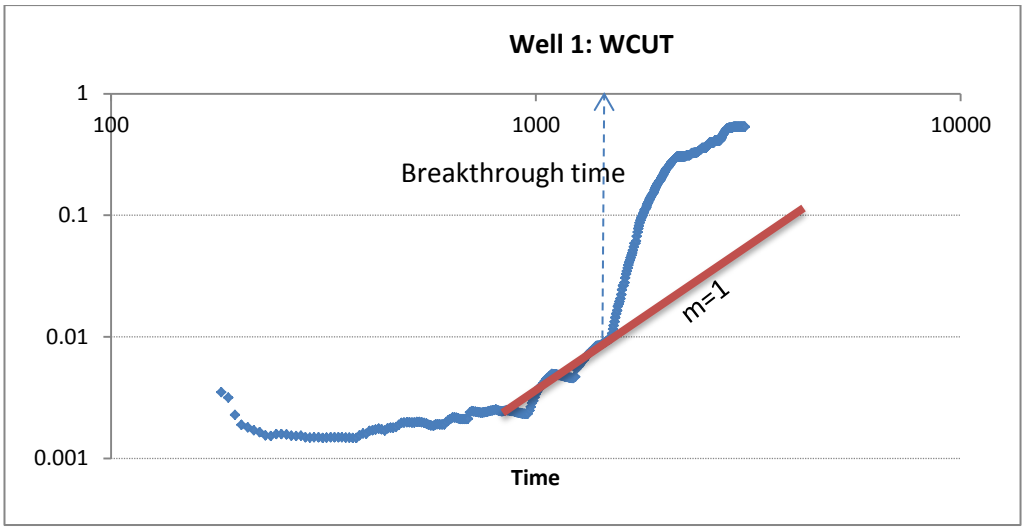


Figure 4.18: Water Breakthrough Time to be included in the Misfit definition.

4.4.2.2 Allocated water and oil rates

A common source of the error in observation data is human/machine error in the allocated water and oil rates. Oil and water rates are usually reported for a group of wells, segments, or even the entire reservoir. These reported rates are then allocated to individual wells by a production engineer, based on the well test measurements. A way of testing for this error is to look at the fractions of the cumulative oil for different wells. Basically, these fractions should stay constant during the production, although there are more complex situations, e.g. a shared platform by more than two fields in which these fractions could change. Figure 4.19 shows the fraction of allocated oil rates for three wells of the Koma field. In the current Koma application, since we decided to use well-test data instead of allocated data, this is no longer an issue.

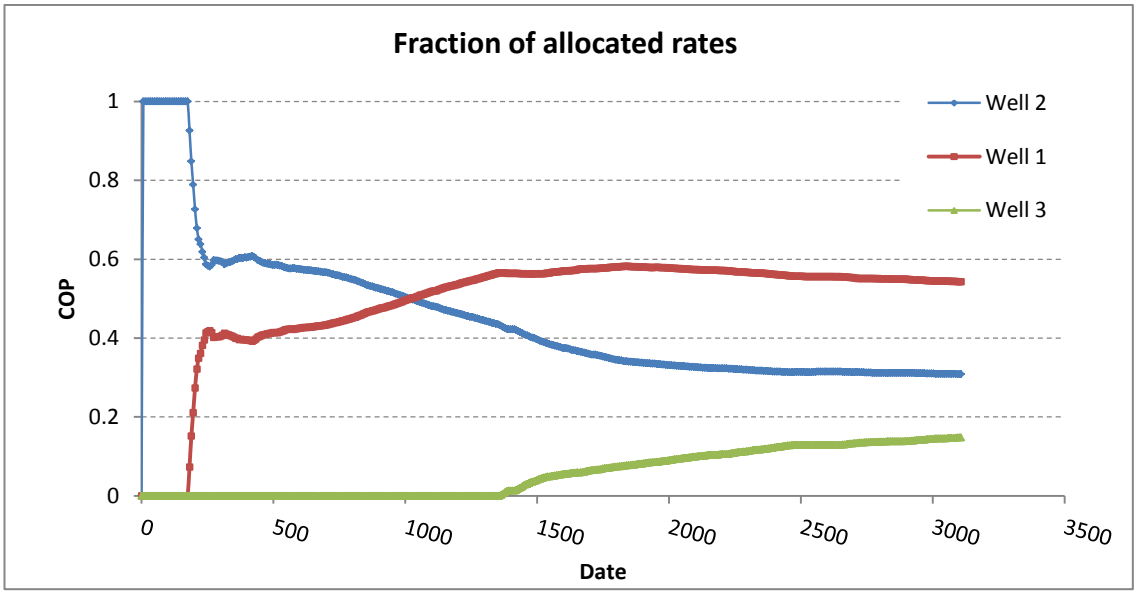


Figure 4.19: Allocation ratios of the wells versus time. These fractions should be constant, but here they vary. This could be an indication of an error in the allocated data, in particular when the platform is shared by two or more fields.

4.4.2.3 BHP constraint

In Koma field, the simulation model is initially controlled by the total liquid rate; a second constraint is set for the bottomhole pressure (BHP). Data analysis showed that when simulation models reached BHP constraint, the simulated produced/injected liquid rates did not honour the historical figures (e.g. Figure 4.20). These models required to be fixed, to honour the history. Since many of them suffer from this problem, it is not practical to fix them all manually. Instead, we used a function to penalise these models and reduce their likelihood.

The applied penalty function is in the form of two new misfit components, liquid production rate and water injection rate, with a small tolerance for difference (standard deviation). The tolerances were set by a trial and error process, so that, the penalty function does not dominate the total misfit value while it sufficiently penalises the models reaching BHP constraint.

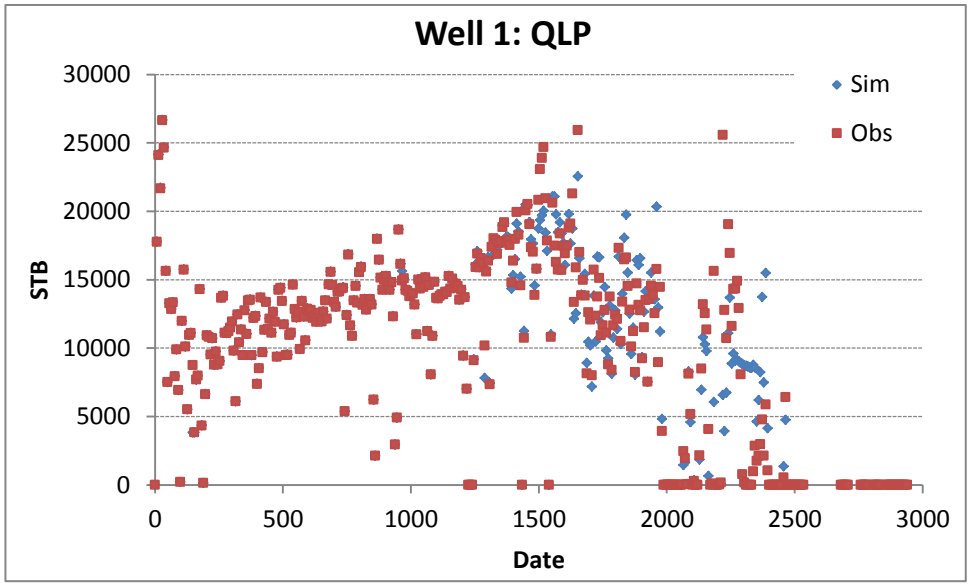


Figure 4.20: Discrepancies in liquid production rate of Well 1 in observation and simulation data. The model has reached the BHP constraint in between ~1500 to 2500 days.

4.4.3 Modified misfit function

In addition to a refined set of uncertainty parameters, the misfit definition was revisited and modified for Koma field, based on data quality control and analysis. We removed oil production rates from the objective function since the model was controlled by total liquid rate and the objective function included all three rates (oil, water and gas), which imposed a bias in the misfit definition. This is because if two of the rates are matched, the third will be matched too. For simplicity we used water-cut instead of water rates and gas-oil ratio instead of gas rates.

Data quality control showed that allocated production and injection rates are erroneous. Since Koma field had a relatively large set of well tests, we used well-test rates in the misfit function instead of allocated rates. Finally, as discussed earlier, to account for the discrepancy in simulated production and injection liquid rates in a model controlled by total liquid, we treated production liquid rate and water injection rates as new two misfit components. The final set of misfit components with their assumed Gaussian measurement error is shown in Table 4.4.

Table 4.4: History-match data and error model in revisited Koma.

Parameter	Parameter abbr.	Error definition
Bottomhole pressure	BHP	50 psi
Near-well average pressure	Pave	50 psi
Water-cut	WCT	5%
Gas-oil ratio	GOR	50 SCF/STB
Liquid production rate	QLP	2%
Water injection rate	QWI	1%

4.4.4 Results of Koma field History Matching

We used both single and multiobjective BOAs for the history matching problem of the Koma field. Based on the finding of the objectives selection study on PUNQ-S3, two reservoir compartments in the south and north of the field were taken as the basis for the two objective functions in the MBOA; i.e. the sum of the misfit components of the two wells in the north was taken as the first objective and the sum of the misfit components of the well in the south was taken as the second objective.

In both single and multiobjective optimisations, BOA parameters are used as in the previous chapter, i.e. 150 solutions in the initial population, 100 parents and 50 children in each generation, maximum number of parents to be taken for each solution in the Bayesian network equal to 6, and 8 bins for representing parameter values. The stopping criterion was set as the total number of 17 generations (1,000 function evaluations).

To minimise the effect of randomness in the results, we ran 5 trials of the BOA and MBOA with the same initial random population. Results averaged for 5 trials and the sum of the objectives in the multiobjective experiment were compared to the single misfit in the single objective experiment. Figure 4.21 shows the average of the minimum misfits found in each trial, the average of the generational minimum misfit, and a boxplot of the misfit found for BOA and MBOA respectively. As the figure shows, the minimum misfit convergence is very similar for the two but as expected, the MBOA produced broader boxplot of misfit values. This is because the MBOA converges toward a Pareto front in a two-objective space.

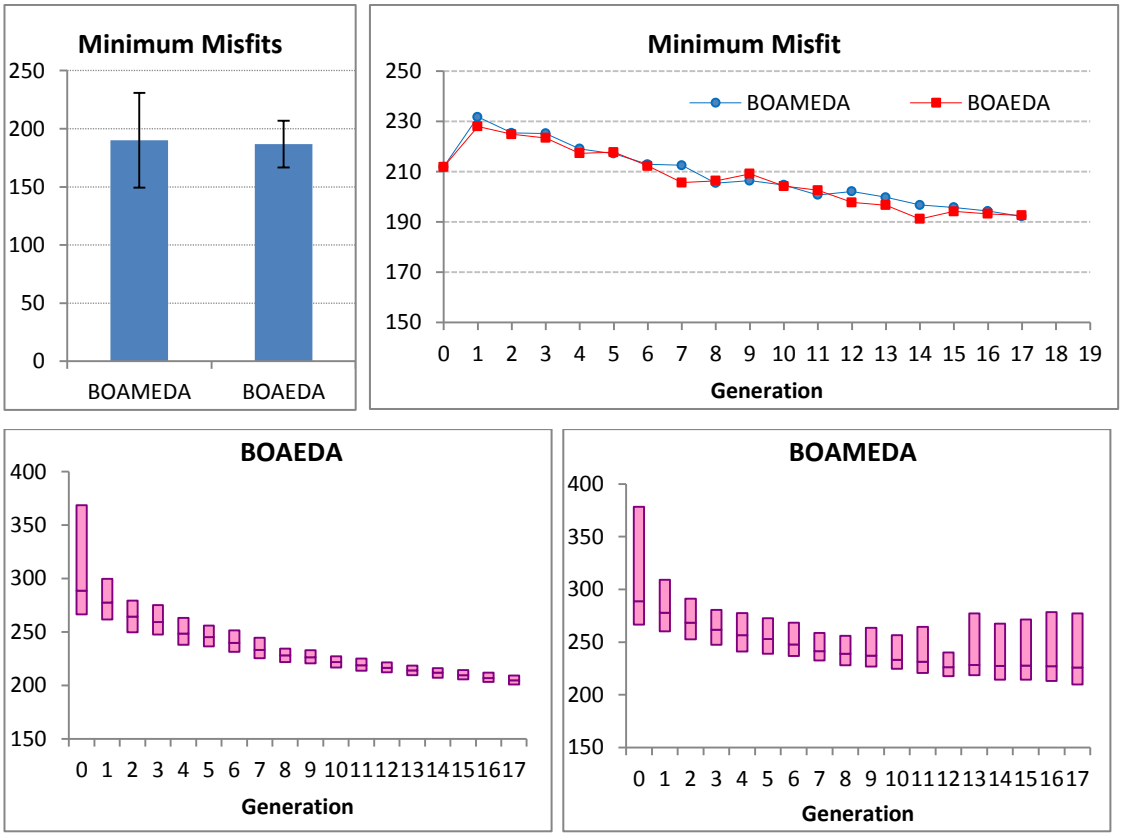


Figure 4.21: Average of minimum misfit found in 5 trials (top-left), average of minimum misfit found in each generation for 5 trials (top-right), boxplot of misfit per generation for BOA (lower-left) and MBOA (lower-right) for Koma field 1,000 simulations. BOA and MBOA show similar minimum misfit convergence, but different boxplot of misfits.

We also looked at inertia diversity per generation for BOA and MBOA. As Figure 4.22 shows, MBOA has maintained diversity among the solutions in the population better than BOA throughout the evolution.

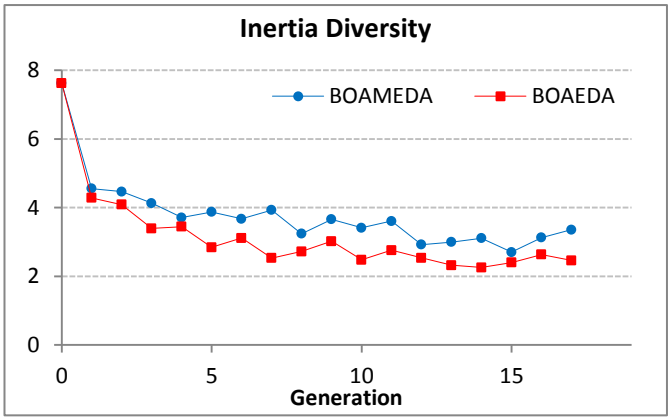


Figure 4.22: Average of inertia diversity for BOA and MBOA. MBOA experiment shows a larger diversity.

The Pareto front obtained by a selected trial of the MBOA is shown in Figure 4.23. The figure shows that Well 2 contributes more to the overall misfit, and that matching Well 2 is more difficult than Well 1 and 3 with the selected parameterisation model.

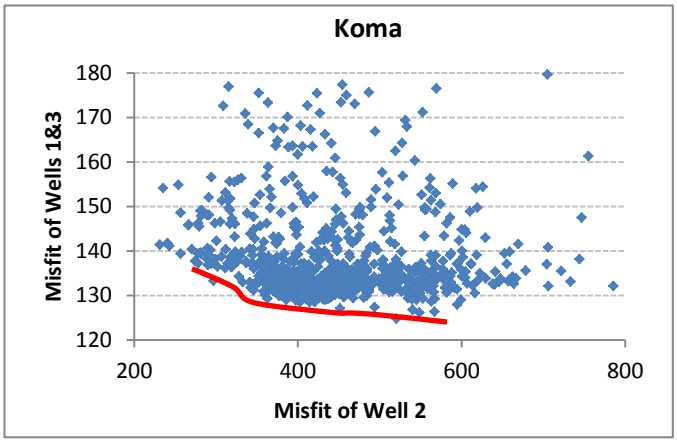


Figure 4.23: Non-dominated front (red line) obtained by MBOA.
As most of the non-dominated solutions are laid closely parallel to misfit of the Well 2 axis, Well 2 is more challenging to match with current parameterisation of the model.

4.5 Discussion

History matching uses optimisation algorithms to minimise the discrepancy (misfit) between simulation results of the reservoir model and the observation data available for the reservoir. Some attempts to take and solve history matching by splitting the total misfit function into two or more objectives are reported in the petroleum literature. If the aim in assisted/automatic history matching is to minimise the misfit, then one can argue that history matching is not a multiobjective problem by nature. Thus, one cannot expect multiobjective sorting algorithms to improve the history matching and the convergence speed of the algorithm considerably. This statement is supported by both field applications in this chapter.

Maintaining population diversity high enough is crucial to avoid premature convergence and achieve a better match quality by use of EAs. The reason behind premature convergence is that the individuals in the gene pool are too alike, thus the algorithm has not been able to extract new information that can lead the search towards the global optimum. Even though history matching is not a multiobjective problem by nature, multiobjectivisation of the overall misfit can provide EAs with more freedom to explore multiobjective space, which results inless likelihood of becoming trapped. In the PUNQ-S3 application, we showed that the untuned BOA experienced a premature convergence while the unturned MBOA exhibited its normal convergence. Thus, MBOA is less sensitive to parameter tuning. It is worth pointing out that, although

multi-objective EAs can improve the diversity of solutions in the population, other methods could be used to improve the diversity in a single objective EA, without using multiple objectives.

Multiobjective optimisation for history matching can have other added benefits wherever the user truly wishes to find a set of trade-off solutions. One example is trading off misfits for different parts of the reservoir, another situation would be to analyse how different components of the overall misfit fight for match.

If different parts of the reservoir fight for better match, multiobjective optimisation can potentially improve misfit of the selected parts in comparison with single objective history matching. In this case, the reservoir energy and as a result, global pressure match is not balanced and one can match only one or a limited part of the reservoir with the existing energy.

The trade-off between different components of the overall misfit can show how challenging it will be to match each component and one cannot match all the components simultaneously. Thus reservoir engineer may trade-off the match quality of some components. Nevertheless, better solution is to modify the model, so that, it provides enough flexibility to match all the components simultaneously.

4.6 References

Ferraro, P., & Verga, F. (2009) Use of Evolutionary Algorithms in Single and Multi-objective Optimisation Techniques for Assisted History matching. Offshore Mediterranean Conference and Exhibition RES, 04/01 (79) Ravenna, Italy.

Fonseca, C.M. & Fleming, P.J. (1995) An Overview of Evolutionary Algorithms in Multiobjective Optimisation. Evolutionary Computation, 3(1):1-16.

Ghosh, A. & Dehuri, S. (2005) Evolutionary Algorithms for Multi-Criterion Optimisation: A Survey. International Journal of Computing & Information Sciences. 2(1): 38-57.

Hajizadeh, Y., Christie, M.A., & Demyanov, V. (2011) Towards Multiobjective History Matching: Faster Convergence and Uncertainty Quantification. SPE Number: 141111-MS. SPE Reservoir Simulation Symposium, The Woodlands, Texas, USA.

Horn, J. Nafploitis, N. & Goldberg, D.E. (1994) A Niched Pareto Genetic Algorithm for Multi-objective Optimisation. Proceedings of the first IEEE Conference on Evolutionary Computation, 82-87.

Knowles, J., Corne, D. (2000) Approximating the Nondominated Front Using the Pareto Archived Evolution Strategy. Evolutionary Computation, 8(2): 149-172.

Knowles, J., Watson, R., Corne, D. (2001) Reducing Local Optima in Single-Objective Problems by Multi-objectivization. Evolutionary Multi-Criterion Optimization, 269-283.

Mohamed, L., Christie, M.A., & Demyanov, V. (2011) History Matching and Uncertainty

Quantification: Multiobjective Particle Swarm Optimisation Approach. SPE Number: 143067-MS. SPE EUROPEC/EAGE Annual Conference and Exhibition. Vienna, Austria.

Pelikan, M., Goldberg, D. E., & Lobo, F.G. (2000) A Survey of Optimisation by Building and Using Probabilistic Models. Computational Optimisation and Applications, 21: 5–20.

Schaffner, J. D. (1985) Multiple objective optimisation with vector evaluated genetic Algorithms. Proceedings of an International Conference on Genetic Algorithms and Their Applications, sponsored by Texas Instruments and the U.S. Navy Center for Applied Research in Artificial Intelligence (NCARAI), 93-100.

Schulze-Riegert, R., Krosche, M., Abul F., & Ghedan, S. (2007) Multi-objective Optimisation with Application to Model Validation and Uncertainty Quantification. SPE Number: SPE 105313-MS. SPE Middle East Oil and Gas Show and Conference. Kingdom of Bahrain.

Srinivas, N. and Deb, K. (1995) Multiobjective function optimisation using nondominated sorting genetic algorithms. Evol. Comput., 2(3): 221–248, Fall.

Yortsos, Y.C., et al. (1999). Analysis and Interpretation of Water/Oil Ratio in Waterfloods. SPEJ 413.

Zitzler, E. & Thiele, L. (1999) Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach. IEEE Transactions on Evolutionary Computation, 3(4): 257-271.

CHAPTER 5:
HYBRID EVOLUTIONARY ALGORITHMS FOR HISTORY MATCHING

> "Neighbouring species integrate or hybridise wherever they are in contact or are potentially capable of doing so."
>
> Ernst Mayr (1904 - 2005)

5.1 Introduction

Stochastic search algorithms, such as genetic and evolutionary algorithms, exploit knowledge of the distribution of good solutions amongst those already visited, to select wisely new points in the search space to evaluate. In the case of genetic algorithms, this knowledge is stored implicitly in the current population. It is exploited through the application of crossover and mutation operators (discussed below) to the solutions in this population, which, it is hoped, will lead to a generation of improved solutions.

The Genetic Algorithm (GA) is the most common Evolutionary Algorithm (Holland, 1975; Goldberg, 1989). GAs are widely used in history matching; There are several applications of GAs in the petroleum literature: Romero et al. (2000), Williams et al. (2004), Castellini et al. (2005), and Erbas & Christie (2007). Real-coded GAs have also been applied to history matching; Carter and Ballester (2004) used a real-coded GA for

history matching of a real field. Carter et al. (2006) applied the same algorithm to history matching of the IC Fault model.

As discussed in Chapter 3, the success of genetic algorithms is sometimes attributed to the *building block hypothesis* (Goldberg, 1989), whereby it is conjectured that the genetic algorithm efficiently identifies and recombines *building blocks*, i.e. solution components, or schemata, with above average fitness. In practice, genetic operators often break such partial solutions, especially when these schemata are large, spread widely across the solution or when operators such as uniform crossover are used.

As we discussed in chapters 3 and 4, EDAs create an explicit model of the location of good solutions found so far that can, hopefully, be used to generate improved solutions in the future. Therefore, instead of manipulating solutions from genes of solutions in a population, new solutions are generated using the model. As the search proceeds, the model is updated or adapted as new solutions are evaluated. This is the underlying concept behind EDAs.

In recent years, hybrid algorithms have been applied to many practical or academic history matching problems. Leitão and Schiozer (1999) used a hybrid algorithm combining a gradient method and direct methods (Polytope and Hooke & Jeeves), while Gómez et al. (1999) proposed a hybrid gradient and global optimisation method called Tunnelling for history matching and Mantica et al. (2002) used a combination of chaotic-based global and gradient-based local optimization techniques for history matching, using production and seismic data. More recently, Castellini et al. (2006) combined GA with Experimental Design and Response Surface Methodology, Schulze-Riegert et al. (2009) introduced a hybrid GA and Ensemble Kalman Filter from data assimilation techniques to history matching and finally Reynolds et al. (2011) used a hybrid EDA-PSO algorithm for history matching. However, no attempt is reported in the literature to hybridise GA and EDA algorithms for history matching.

In this chapter, we apply a simulated binary genetic algorithm (SBGA) and an incremental histogram-based estimation of distribution algorithms (iHEDA). Then we propose a new hybrid algorithm, which combines SBGA and iHEDA and apply these algorithms to optimisation of a test function and history matching of two reservoir

models, IC-Fault and Teal South. We aim to improve the quality and diversity of the search using the exploitation and exploration power of both algorithms. Experimental results indicate that SBGA/iHEDA can outperform both SBGA and iHEDA for specific problems. Furthermore, the control parameters of the proposed hybrid algorithm are also experimentally studied in this chapter.

5.2 Genetic Algorithms

As discussed in earlier chapters, in GAs, knowledge of the distribution of good solutions is stored implicitly in the current population and exploited by applying selection, crossover, and mutation operators to solutions in the population. Selection is the process of choosing the most promising solutions with regard to the fitness function value in the current generation (parents). The crossover operator generates two new solutions (children) by recombining the information from two parents; in addition, the random mutation of some gene values in a promising solution is sometimes used to generate children.

Here SBGA, a real-coded GA is used which employs polynomial simulated binary crossover and mutation operators. An outline of the SBGA employed in the present work consists of the following steps:

1. Randomly generates a population of N solutions.
2. Select a set of P promising solutions from the population.
3. Repeat until C new solutions are generated:
    3.1. Select two random parent solutions (p_1, p_2) from P
    3.2. Apply the crossover operator to the parent pair (p_1, p_2) to generate a new pair of child solutions (c_{x1} and c_{x2}).
    3.3. Apply the mutation operator to created child solutions (c_{x1} and c_{x2}) to generate the final new pair of child solutions (c_1 and c_2).
    3.4. Merge new pair (c_1 and c_2) with child solutions C.
4. Update population with C new solutions by merging the children with the previous population and discarding unpromising solutions.
5. Go to step 2 and continue until the stopping criterion is reached.

SBGA uses the simulated binary crossover (SBX) and the simulated binary mutation (SBM) for generating new child solutions from the parent solutions. Both of these operators are adaptive in the sense that the range of the possible child solutions depends on the distance between the parents. As the population converges, this distance decreases and the generated children more resemble the parents.

5.2.1 SBX operator

In a GA, the choice of crossover operator depends on the parameter representation. In a binary-coded GA, parameters are represented by discrete variables and a one or multi-point binary crossover is used; while in real-coded GAs, two types of crossover are commonly used in the literature, Simulated Binary Crossover (SBX) (Deb and Agrawal, 1995) and Parent-centric Normal crossover (PNX) (Deb et al., 2001). Ballester and Carter (2004) applied these two crossovers to different test functions and a history matching problem. In the current thesis, the SBX is used, which simulates the one-point crossover properties in a binary-coded GA. We use SBX to crossover two parent solutions, according to the following procedure:

1. First, we need to simulate the shape of the probability distribution function of the binary-coded crossover, according to:

$$\begin{cases} c(\beta) = 0.5(n+1)\beta^n, \beta \leq 1 \\ c(\beta) = 0.5(n+1)\dfrac{1}{\beta^{n+2}}, \beta > 1 \end{cases} \tag{5.1}$$

   where β is spread factor and n is a non-negative number, usually between 2 and 5, which determines the tendency to generate near-parent (higher n) or far-from-parents (lower n) solutions. Lower values of n, e.g. equal to 2, lead to performing a more exploratory search.

2. A uniform random number u between 0 and 1 is generated for each variable.
3. From the simulated probability distribution function (Figure 5.1), we get a β value that makes the area under the curve equal to u.
4. Then variable values for child solutions (c_{x1} and c_{x2}) are calculated from:

$$\begin{cases} c_{x1} = \frac{1}{2}[(1+\beta).p_1 + (1-\beta).p_2] \\ c_{x2} = \frac{1}{2}[(1-\beta).p_1 + (1+\beta).p_2] \end{cases} \quad (5.2)$$

where p_1 and p_2 are parameter values of the parent pair and β is the obtained spread factor.

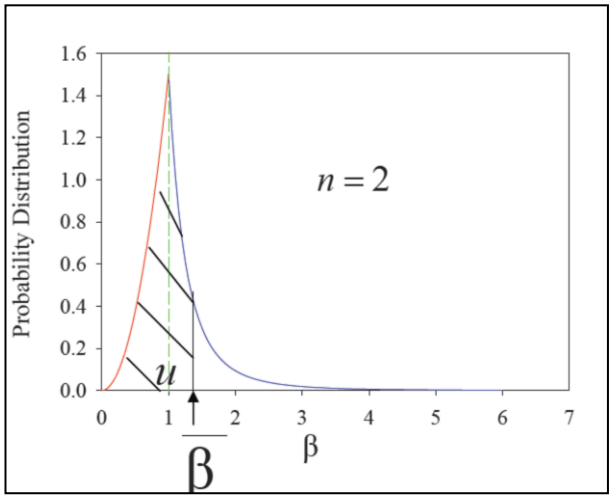


Figure 5.1: Spread factor for SBX; spread factor β corresponds to the random number *u* in a polynomial probability distribution function, which simulates one-point binary crossover.

5.2.2 SBX mutation operator

The mutation operator is applied to variables with a random probability of p_m. The polynomial mutation operator (Deb, 2001) is one of the most widely used mutation operators for real-coded GAs. The mutation operator used in the current work performs real polynomial mutation. The procedure for performing mutation on a single solution is:

1. For each variable do:
2. Generate a uniform random number u_m between (0, 1).
3. If $u_m < p_m$, then perform mutation as follows:
   3.1. Generate another uniform random number r between (0, 1).

3.2. Simulate the shape of the probability distribution function of the binary-coded mutation according to:

$$\begin{cases} \Delta = 2r^{\frac{1}{n_m+1}} - 1, r < 0.5 \\ \Delta = 1 - 2(1-r)^{\frac{1}{n_m+1}}, r \geq 0.5 \end{cases} \quad (5.3)$$

where Δ is delta factor, r is generated random number, and n_m is a non-negative number, which determines the tendency to generate near-parent (higher n_m) or far-from-parents (lower n_m) solutions. Similarly to the n factor in simulated binary crossover, a low value of n_m, e.g. 2, implies a more explorative search.

3.3. Then update the variable value for child solution (c) from:

$$p = p + \Delta(p_u - p_l) \quad (5.4)$$

where p is the variable value of the solution, p_u and p_l are upper and lower limits of the variable respectively, and Δ is the obtained delta factor.

5.3 Incremental Histogram-based EDA

In the previous chapters, we introduced Estimation of distribution algorithms (EDAs) and demonstrated the structure of EDAs in general and two histogram-based EDAs (BH and EAH) and the Bayesian optimisation algorithm (BOA) in particular.

In this chapter, we reconsider the basic histogram model (HEDA) and to include two new features: Laplace correction for initial optimisation and an incremental learning mechanism for the probabilistic model.

5.3.1 Structure of HEDA

As we discussed earlier, HEDA uses the histogram model as the probabilistic model for generating new solutions and, as a univariate EDA, it considers dependency between variables. The histogram model natively supports discrete problems in which each value

is represented by a bin. For continuous problems, the permitted range for each variable is split into equal sized bins.

Therefore a marginal probability value is stored in each bin for each variable. Initially these values are set to ***1/c*** where c is the number of bins. After selecting good solutions, for each bin, the probability is calculated as follows:

$$p_l(x_i) = \frac{\sum_{j=1}^{N} \delta_j(X_i = x_i | D_{l-1}^{Selected})}{N} \tag{5.5}$$

where ***l*** is the generation number, subscripts ***i*** iterate over the number of bins, subscript ***j*** iterates over the number of parents, $D_{l-1}^{Selected}$ is the set of ***N*** selected solutions (parents), $\delta_j(X_i = x_i | D_{l-1}^{Selected})$ =1 if the jth of the selected individual's ith bin is $\boldsymbol{x_i}$ and 0 otherwise.

New solutions are generated by selecting a bin for each variable, according to the stored probability values. A value for each variable is selected at random within the range of the bin. Figure 5.2 illustrates how the basic histogram approach models the marginal probability distribution over the selected solutions for a single variable. The graph plots probability density on the vertical axis against variable value on the horizontal axis.

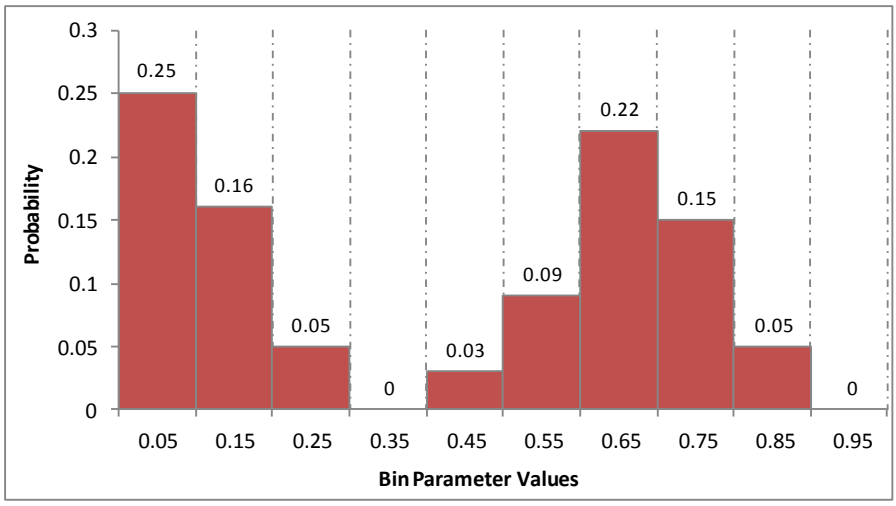


Figure 5.2: Basic histogram model in HEDA; for a continuous variable in the range [0,1), the variable range is split into equally sized bins; probability is calculated for each bin based on the frequency of the parameter values in the promising solutions of the population.

5.3.2 Laplace correction for HEDA

As in many evolutionary algorithms, in the first generation of HEDA, the solutions are created randomly. In the event that all the selected solutions in the first generation happen to have one of their bin probabilities as zero, then all the children will have it as zero too. As the joint probability in HEDA is the product of the marginals, the joint probability distribution becomes zero and HEDA sometimes may not visit a global optimum. To resolve this issue, Gonzalez et al. (2001) proposed a Laplace correction as follows:

$$p_l(x_i) = \frac{\sum_{j=1}^{N} \delta_j(X_i = x_i | D_{l-1}^{Selected}) + 1}{N + r_i} \tag{5.6}$$

where $\boldsymbol{r}_i$ is the number of distinct values that $\boldsymbol{X_i}$ may take. This correction leaves a small probability for the bins happening to have their probability zero in the initial random population.

5.3.3 Incremental learning mechanism for HEDA

In EDAs, the probabilistic model used for sampling a new solution can be either fully regenerated from the promising solutions in the current generation or updated from the previous generation by integrating a new model from the current generation. In the latter, the algorithm is incrementally learning.

HEDA with incremental learning (iHEDA) is equivalent to PBIL, introduced by Baluja (1994) which uses binary representation instead of splitting the parameter range to a number of bins. We applied following learning strategy in the building probabilistic model step of the EDA process (Mühlenbein, 1997):

a. Construct a new histogram model (N_i) from the promising solutions P in the current generation,

b. Update the histogram model (H_i) from the new model (N_i) and the histogram model in the previous generation (H_i^{k-1}) according to the following equation (5.7):

$$H_i^k(j) = \alpha H_i^{k-1}(j) + (1-\alpha) N_i^k(j) \tag{5.7}$$

where α is the learning factor which determines the effect of the old model on the new model.

5.4 Hybrid SBGA/iHEDA

Combination of different algorithms is often done with the purpose of developing new search algorithms that combine the benefits of the original algorithms and thus achieving new algorithms that have better convergence speed and more reliability. Hybridising search algorithms can be done in different ways. A loose integration of two algorithms can be achieved by using one algorithm as the pre-optimiser for the initial population of the other one. Fully hybridised strategies maintain the integration of the two algorithms in the entire run. This means that the two algorithms perform a cooperative search, either in parallel or serially, in each generation of the evolution.

As we know, the main difference between GA and EDA is how new solutions (children) are generated; thus addition of SBGA to iHEDA may benefit from both algorithms, i.e. handling complex interactions by GAs and preserving meaningful patterns and adaptive probability models in EDAs. Zhang et al. (2005) proposed a new operator, called guided mutation for the maximum clique problem by combining GA with EDA. Their guided mutation generates offspring through combination of global statistical information and the location information of solutions found so far. Santana et al. (2007) combined a GA with the univariate EDAs to build new algorithms that allow solution of problems with complex dependencies.

Here a hybrid algorithm, referred to as SBGA/iHEDA, has been developed which performs a parallel, cooperative search mechanism using a combination of the SBGA and iHEDA algorithms. In the hybrid algorithm SBGA/iHEDA, generating new child solutions is done in parallel cooperatively between the two algorithms, i.e. some of the child solutions are generated in the GA mechanism, and the rest are generated by the EDA mechanism, i.e. by sampling from a probability model created by the EDA.

An additional control parameter appears for the SBGA/iHEDA, a ratio between 0 and 1, which controls how many solutions are created by each mechanism. We call this parameter the participation ratio (p). This parameter represents the extent to which each of the GA and EDA participate in the process. A simple outline for SBGA/iHEDA is:

1. Randomly generate a population of N solutions.
2. Select a set of P promising solutions from the population.
3. Construct a probabilistic model from the promising solutions.
4. Generate a set of C new child solutions according to the participation approach:
   4.1. Generate $C1$ new child solutions using genetic operators (crossover and mutation).
   4.2. Sample $C2$ new child solutions from the constructed probabilistic model.
5. Update population with C ($=C1+C2$) new solutions by merging the children with the previous population and discarding unpromising solutions.
6. Go to step 2 and continue until a stopping criterion has been met.

A simple schema for the hybrid SBGA/iHEDA is shown in Figure 5.3. One can expect different ways of setting up this participating parameter. In this work, we employed two approaches for participating: fixed participation and adaptive participation.

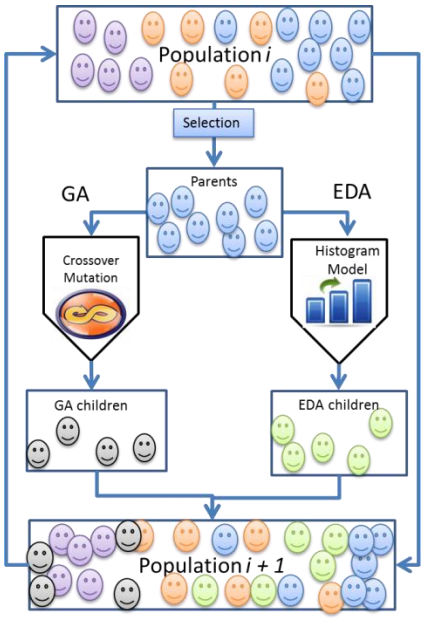


Figure 5.3: A simple schema for hybrid SBGA/iHEDA. The algorithm cycle starts from a population, and then GA and EDA both contribute to generation of new solution based on a fixed or adaptive participation ratio.

5.4.1 Fixed Participation

The straightforward strategy is a fixed participation, in which the participation ratio (p) is fixed throughout the evolution. In this approach, the number of child solutions generated by each participating algorithm is:

- SBGA: $p \times C$
- iHEDA: $(1-p) \times C$

where p is the participation ratio and C is the total number of children to be generated in each generation (both are control parameters of the hybrid algorithm SBGA/iHEDA).

5.4.2 Adaptive Participation

In the adaptive approach, the participation ratio will be increased towards the algorithm that generates more promising solutions in each generation. In this approach the number of child solutions generated by each algorithm is:

- SBGA: $X_{SBGA} \times C$
- iHEDA: $X_{iHEDA} \times C$

where X_{SBGA} is the participation ratio of SBGA, X_{iHEDA} is the participation ratio of iHEDA, and C is the total number of children to be generated in each generation. We always have equation (5.8):

$$X_{GA} + X_{iHMDA} = 1 \tag{5.8}$$

We start with a participation ratio of 0.5 for both algorithms, then in each generation, the number of best child solutions from SBGA and iHEDA are compared and the winning algorithm gets an incremental 5% of the participation ratio. The incremental 5% is scaled by the relative difference between the two algorithms; this ensures the ratio remains between two algorithms. This adaptive mechanism provides SBGA/iHEDA with a contest-based adaptive function in which algorithms are competing to get higher ratios as they generate better solutions.

Although the wining algorithm in the first generation of adaptive SBGA/iHEDA will be treated more preferentially in the coming generation by contributing to 5% more to child generation, but base on the control parameters of the wining and loosing algorithms and the shape of the misfit landscape, it is always possible for the loosing algorithm to win the search race and be treated more preferentially.

5.5 Rosenbrock function application

The three algorithms (SBGA, iHEDA, and SBGA/iHEDA) were first applied to the Rosenbrock function. The aim was to investigate the performance of the algorithms on a well-known test function and to tune the algorithms and obtain an initial guess for the control parameters in history matching applications.

5.5.1 Rosenbrock function description

The Rosenbrock function, also known as the banana function, is a well-known test function for numerical optimization problems, introduced by De Jong (1975) to test the performance of GAs. It is highly nonlinear and symmetric around quite a long, narrow and parabolic-shaped flat valley, and the variables are strongly correlated (Figure 5.4). It is a minimization problem, and often serves as a test case for premature convergence of evolutionary algorithms. The function has its global optimum, $f(x_i) = 0$, at $x_i = 1$. No algorithm can easily discover the global optimum of the Rosenbrock function.

$$f(x) = \sum_{j=2}^{n} \left(100\left(x_j - x_{j-1}^2\right)^2 + \left(1 - x_{j-1}\right)^2\right) \tag{5.9}$$

where $\boldsymbol{x} = (\boldsymbol{x_1}, \dots, \boldsymbol{x_n})$, $\boldsymbol{n} = \mathbf{5}$, and $\boldsymbol{x_j} \in [-\mathbf{5.12}, \mathbf{5.12}]$

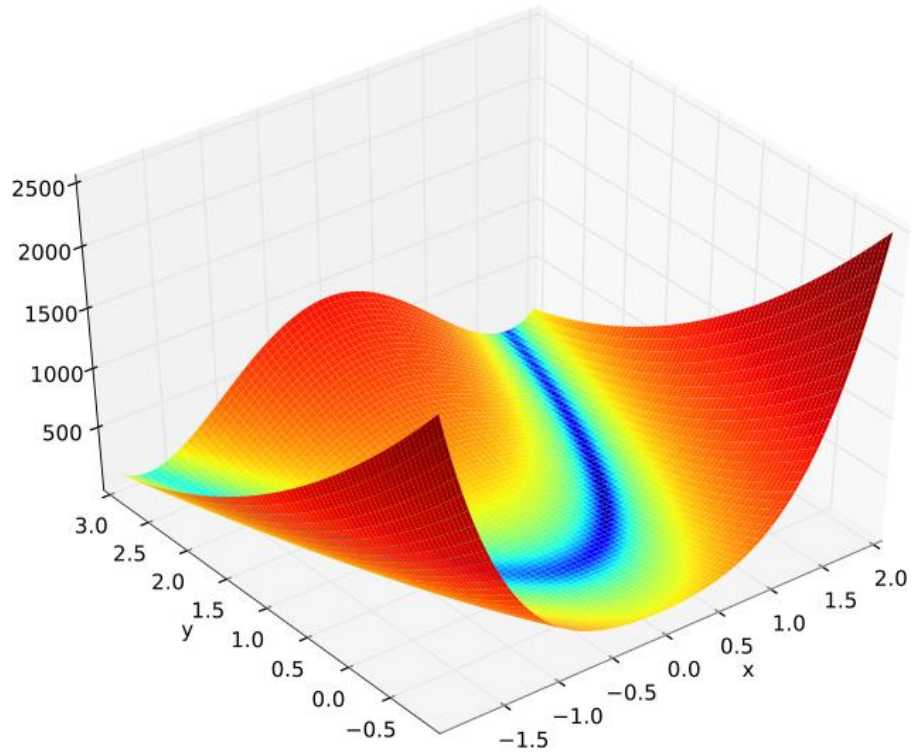


Figure 5.4: A Rosenbrock function for two variables.

5.5.2 Rosenbrock function results

We ran a sensitivity study on tuning parameters of the algorithms to find the best value for each control parameter in terms of convergence performance. Three parameters of the algorithms were fixed in the experiments, including the number of function evaluations, at 20,000, the number of random initial solutions, at 200, and the number of child solutions to be generated in each generation, at 100. To minimise the effect of the random seed required by any stochastic algorithm, we predefined the initial random population in all the experiments and repeated each experiment 10 times. A summary of the results is shown in Table 5.1.

Table 5.1: Summary of the results of sensitivity study on the algorithms' control parameters based on the Rosenbrock function.

Algorithm	Parameter	Tested values	Best value
iHEDA	Number of parents	100, 150, 200, 300	200
	Number of bins	10, 15, 20, 25, 30	20
	Learning rate	0.05, 0.1, 0.3, 0.5, 0.7, 0.9	0.5
SBGA	Number of parents	100, 150, 200, 300	100
	Mutation probability	0.05, 0.1, 0.3, 0.5, 0.7, 0.9	0.5
	Simulated binary factor (*n*)	2, 3, 4, 5	2
SBGA/iHEDA	Participation ratio	0.1, 0.3, 0.5, 0.7, 0.9	0.5

Minimum fitness in 200 generations, inertia-based diversity, and the convergence performance of the three algorithms on the Rosenbrock function were compared (Figure 5.5 and Figure 5.6). As Figure 5.5 shows, hybrid SBGA/iHEDA achieved better minimum fitness with all the participation ratios. The iHEDA and hybrid SBGA/iHEDA with adaptive participation rate (x) suffer from the loss of diversity after just 12 generations. When compared to SBGA, hybrid SBGA/iHEDA with different participation rates has converged better while it has also maintained some level of diversity. Figure 5.6 shows convergence of the different algorithms. SBGA/iHEDA with the participation ratios of 0.1, 0.3 and 0.5 have slightly better convergence compared to the other ratios.

Given the fact that in real application, we are usually limited for the computational resources and 50 is usually the maximum number of generations that one can afford, the difference shown in the following figures on the convergence performance can be significant.

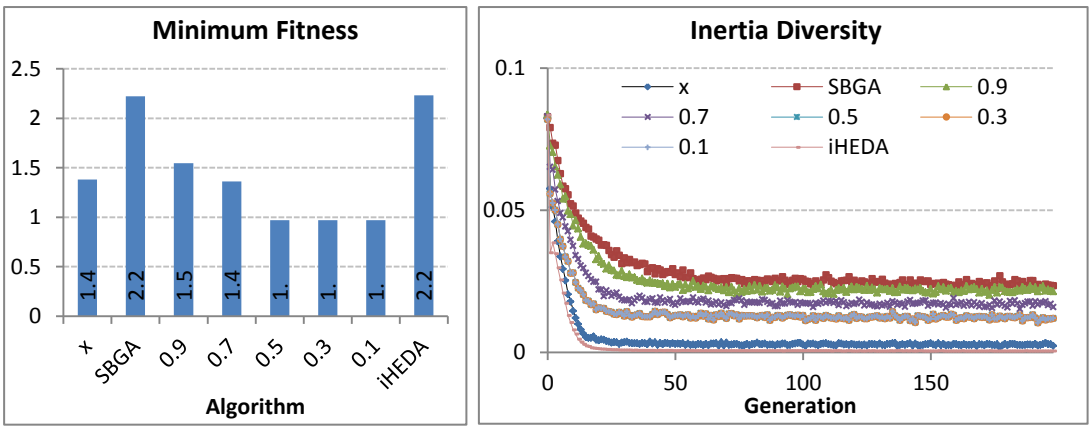


Figure 5.5: Results of Rosenbrock function application for minimum fitness (left) and inertia-based diversity (right) for SBGA, iHEDA, and different participations of hybrid SBGA/iHEDA. Diversity curves show that iHEDA and adaptive participation ratio (x) suffer from loss of diversity.

The Rosenbrock function application helped us to test all three algorithms for optimum convergence. We also tuned the individual and hybrid algorithms and found initial guesses for the control parameters in the field applications. Although these guesses are not guaranteed to work well in history-matching applications, they provide initial values to start with in parameter tuning.

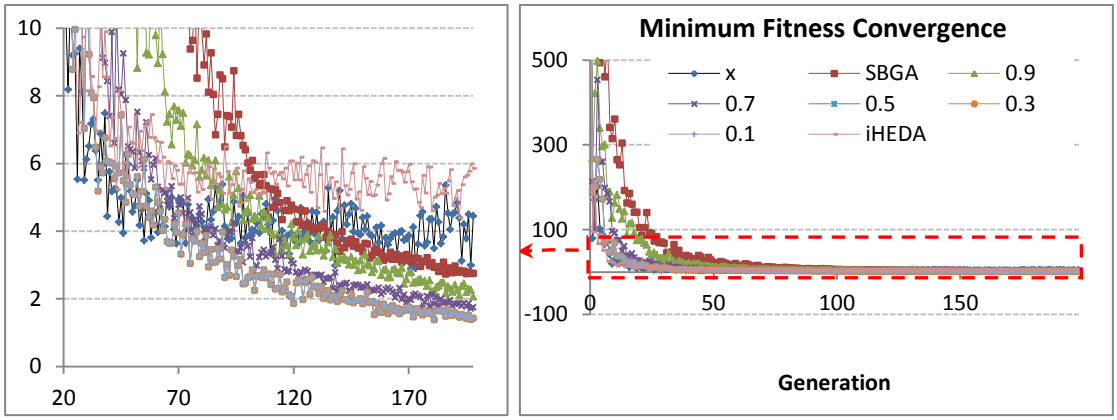


Figure 5.6: Results of Rosenbrock function application for fitness convergence (right) and as zoomed for fitness under 10.0 (left). All three algorithms converged, but Hybrid algorithms with participation ratios of 0.1, 0.3 and 0.5 (grey) show slightly better convergence than other ratios.

5.6 IC-Fault - synthetic case application

Following tuning on the Rosenbrock function, we applied SBGA, iHEDA and SBGA/iHEDA to a hard history-match problem, the IC Fault model. The aim of this study was firstly to evaluate the performance of the SBGA and iHEDA, and secondly to compare their performance to the hybrid algorithm SBGA/iHEDA.

Carter et al. (2004) showed that the best history-matched model in the IC-Fault model does not give the best forecast. This proves the necessity of using ensemble-based history matching and uncertainty quantification algorithms introduced in this work instead of a single best history-matched model.

In current chapter, we use the IC-Fault model as an example for difficult history matching problems. The difficulty is due to the presence of the steep minima resulted by interfacing the fault throw with alternating high and low permeable sand layers. This creates a complicated misfit landscape with numerous steep minima, which is very difficult to be fully discovered by optimisation and search algorithms.

5.6.1 Field description

The IC-Fault is a synthetic model, set up by Carter et al. (2004) at Imperial College, with a simple geological model consisting of six alternating high and low permeability sand layers, of which three high permeability layers have identical properties and three

low permeability layers have identical properties. The good and poor quality sands have constant porosities of 0.30 and 0.15, respectively. The thickness of the layers varies with an arithmetic progression from 12.5 feet at the top layer to 7.5 ft at the bottom layer; the total thickness is 60 ft. The model is 1000 ft in width and a simple fault at the mid-point offsets the layers. The fault throw is also considered unknown.

The model has two wells, a water injector well at the left-hand edge, and an oil producer well on the right-hand edge. Both wells are completed on all layers, and operated at fixed bottomhole pressures and drilled to a depth of 8325–8385 ft.

Each of the 6 geological layers are divided into two simulation layers with equal thicknesses, and each grid block is 10 ft wide, which results in a simulation model of 100 x 12 grid blocks. The vertical positions of the wells in the simulation grid are kept constant and equal, even though fault throw varies. Figure 5.7 shows reservoir model of the IC Fault.

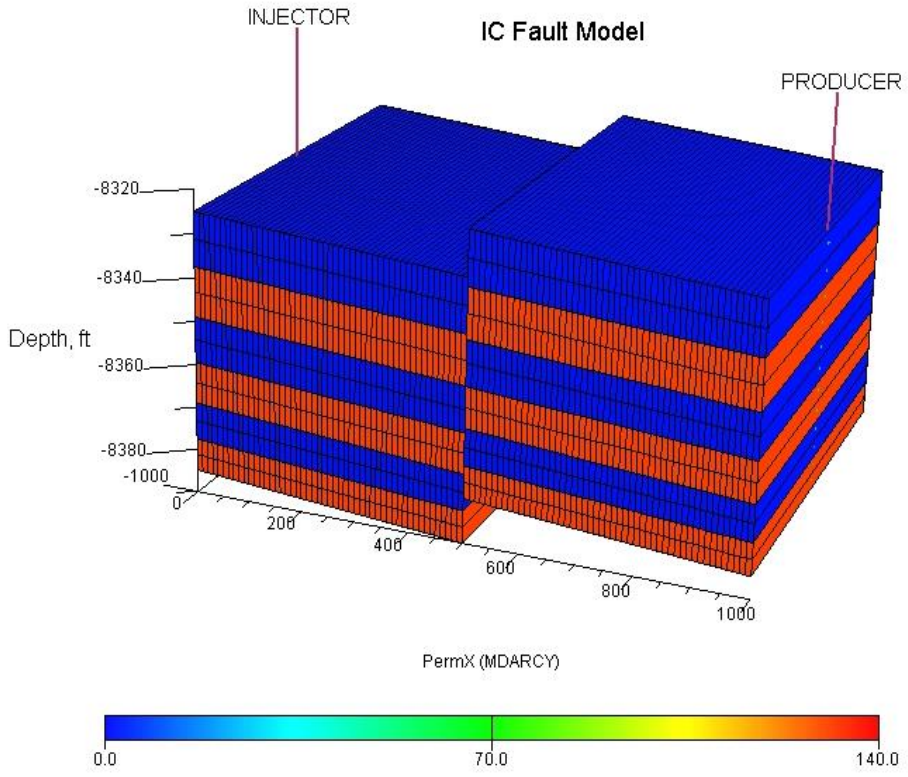


Figure 5.7: IC Fault model.

5.6.2 Uncertainty parameterization

The IC Fault model has three unknown parameters; all with uniform distribution, including high and low permeability and the fault throw (Table 5.2). Unlike the original IC Fault study, we assume homogenous good/poor quality sands in terms of porosity and permeability. The porosity and permeabilities in each grid block were randomly drawn from uniform distributions and no correlations between the two were assumed.

Table 5.2: Unknown parameters for IC Fault model.

Parameter	Description	Uniform Prior Range
K_h	Permeability of good sands	100 – 200 mD
K_l	Permeability of bad sands	1 – 2 mD
h	Fault throw	0 – 60 ft

5.6.3 Misfit definition

The misfit was defined using a standard least squares model, considering the discrepancy between simulated and observed oil and water production rates for the 3-year history matching period, as follows:

$$Misfit = \frac{1}{N_o}\sum_{i=1}^{No}\left(\frac{O_{oi} - S_{oi}}{\sigma_o}\right)^2 + \frac{1}{N_w}\sum_{i=1}^{Nw}\left(\frac{O_{wi} - S_{wi}}{\sigma_w}\right)^2 \quad (5.10)$$

where *Misfit* is the misfit function, i the subscript running over observation data points at reported time, o the subscript for oil production rate observation, w the subscript for water production rate observation, N_o is the total number of oil production rate observations, N_w is the total number of water production rate observations, O_{oi} is observed oil production rate at time point i, O_{wi} is water production rate at time point i, S_{oi} is simulated value of oil production rate at time point i, S_{wi} is simulated value of water production rate at time point i, σ_{oi} is the standard deviation of the oil rate observations, and σ_{wi} is the standard deviation of the water rate observations. Figure 5.8 shows the observation data for history matching of the IC-Fault model.

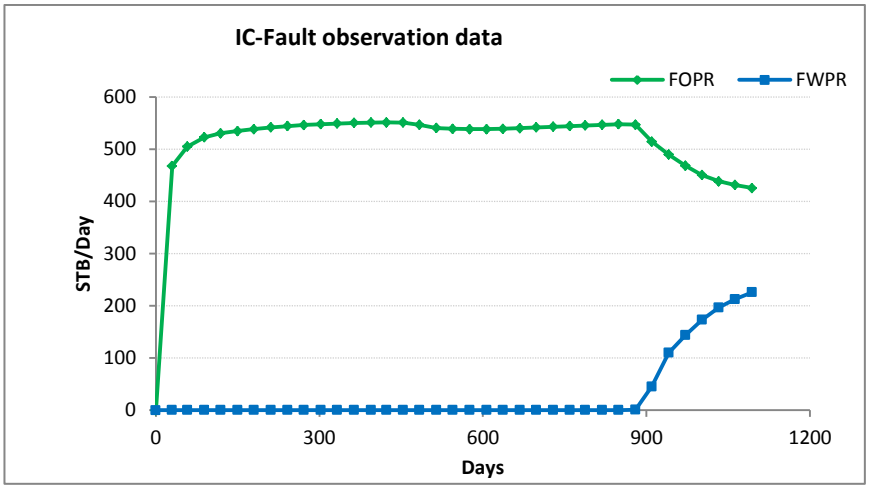


Figure 5.8: Available observation data for IC-Fault, field oil production rates (green diamonds) and field water production rate (blue squares).

The observation data are obtained from an Eclipse simulation case with heterogeneous reservoir properties, as shown in Table 5.3. A random Gaussian noise, with mean zero and standard deviation of 3% of the observed data was added to each measured point.

Table 5.3: True parameter values for IC Fault.

Parameter	Value
K_h	131.6 mD
K_l	1.3 mD
h	10.3 ft

5.6.4 Computational resources

IC-Fault is a small 2D model, which does not require huge computational resources (see Table 5.4). 40 CPU of the Heriot-Watt's HPC cluster, were available to this study. An iteration of EAs with the total 1,000 simulations runs in 3 minutes. Thus, the control parameter tuning of the algorithms can be done in a reasonable time.

Table 5.4: Computational resources available and used by Koma field.

RAM	Total RAM size	16 Gbyte
	Number of total gridblocks	187,872
	Number of active gridblocks	32,303
	Required RAM size	~ 100 Mbyte
CPU	CPU clockspeed	2.9 GHz
	Runtime for a single run using single CPU	7 seconds
	Number of runs in an EDA's iteration	1,000
	Runtime for an iteration of EDAs using single CPU	~ 2 hours
	Number of CPUs available for this study	40
	Runtime for an iteration of algorithms using 40 CPUs	~ 3 minutes

5.6.5 IC-Fault results

Initially, we ran a sensitivity study using different combinations of control parameters for each algorithm. In each trial, one control parameter was varied while all other parameters were kept constant and different values were tested for the varying parameter. This allowed achieving an initial value for each control parameter. In all three algorithms, three parameters were fixed, including the number of function evaluations at 500, the number of random initial solutions at 20, and the number of child solutions to be generated in each generation at 10. To minimise the effect of the random seed, we used the same initial random population in all the experiments. A summary of the results is shown in Table 5.5.

Table 5.5: Summary of the sensitivity study on the algorithms' control parameters, based on the IC-Fault results.

Algorithm	Parameter	Tested values	Best value found
iHEDA	Number of parents	10, 15, 20, 25, 30	20
	Number of bins	10, 15, 20, 25, 30	20
	Learning rate	0.05, 0.1, 0.3, 0.5, 0.7, 0.9, 0.95	0.7
SBGA	Number of parents	10, 15, 20, 25, 30	10
	Mutation probability	0.05, 0.1, 0.3, 0.5, 0.7, 0.9, 0.95	0.1
	Simulated binary factor (*n*)	2, 3, 4, 5	2
SBGA/iHEDA	Participation ratio	0.1, 0.3, 0.5, 0.7, 0.9	0.1

As shown in the Table 5.5, for iHEDA, we tuned the number of parent solutions in each generation, learning rate from the previous generation, and the number of bins used for variable ranges. Figure 5.9 shows the outcomes of the sensitivity study of the learning rate on minimum misfit and convergence speed. As the figure shows, a learning rate of 0.7 is preferred, due to lower minimum misfit and better convergence.

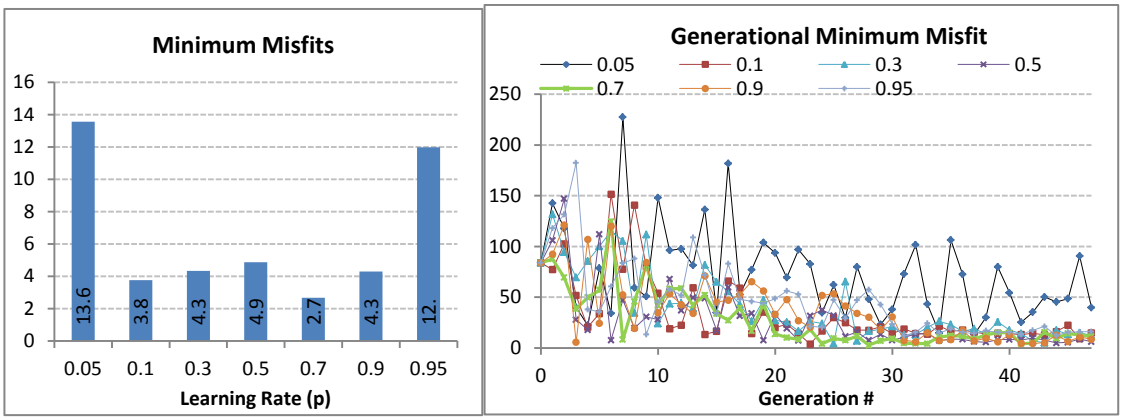


Figure 5.9: Lowest misfit found at entire evolution (left) and minimum misfit per generation (right) for different values of the learning rate. Learning rate 0.7 (green) is preferred due to lower minimum misfit and better convergence.

For the SBGA, we performed the sensitivity study around the number of parent solutions in each generation, mutation probability and simulated binary factor (*n*). Figure 5.10 shows the results of the sensitivity study on mutation probability. A mutation probability of 0.1 is preferred, due to slightly better convergence than for other probabilities.

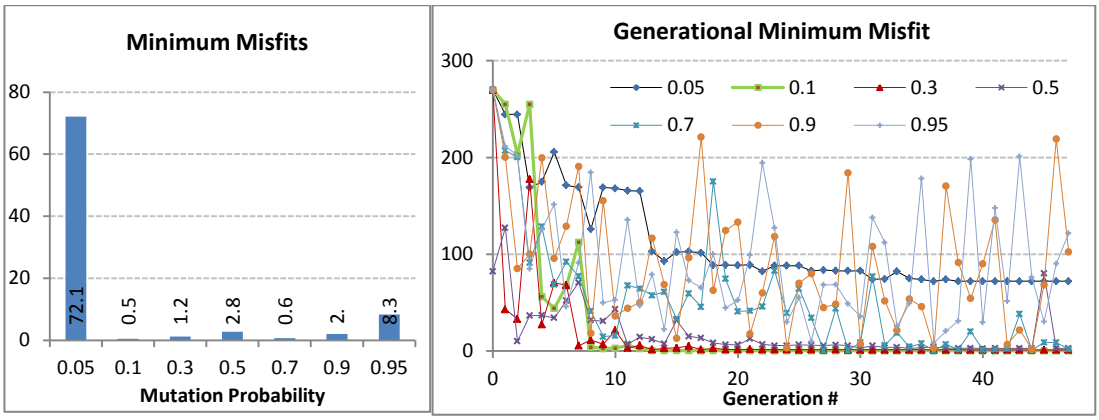


Figure 5.10: Lowest misfit found at entire evolution (left) and minimum misfit per generation (right) for different values of the mutation probability. Mutation probabilities of 0.1 and 0.5 result in better minimum misfit and convergence. Probability of 0.1 (green) is preferred due to slightly better convergence than for other probabilities.

SBGA, iHEDA, and SBGA/iHEDA are all stochastic algorithms and one cannot expect the same results in two trials of the algorithm, even with the same control parameters. To minimize the impact of randomness in the experiments, we repeated each tuned algorithm 10 times and then averaged the minimum misfit and inertia diversity (Figure 5.11) and generational minimum misfits (Figure 5.12). The hybrid algorithm with the participation ratios of 0.7, x (adaptive scheme) and 0.5 give better minimum misfit, diversity and convergence speed than both SBGA and iHEDA.

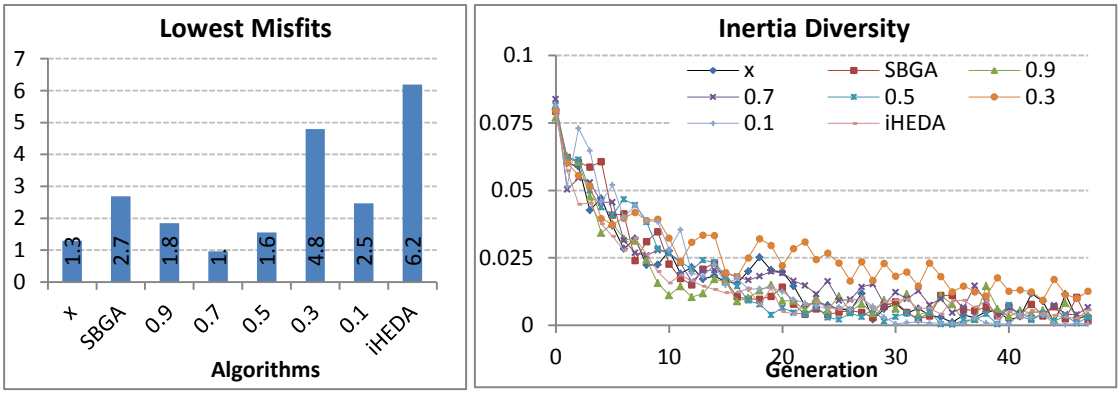


Figure 5.11: Lowest misfit found for the entire evolution (left) and inertia diversity measure per generation (right) for SBGA, iHEDA, and different participation ratios of hybrid SBGA/iHEDA. Hybrid algorithm with participation ratio of 0.7 achieved the best misfit followed by the adaptive scheme (x). Diversity measure shows that all algorithms converged and maintained some level of diversity.

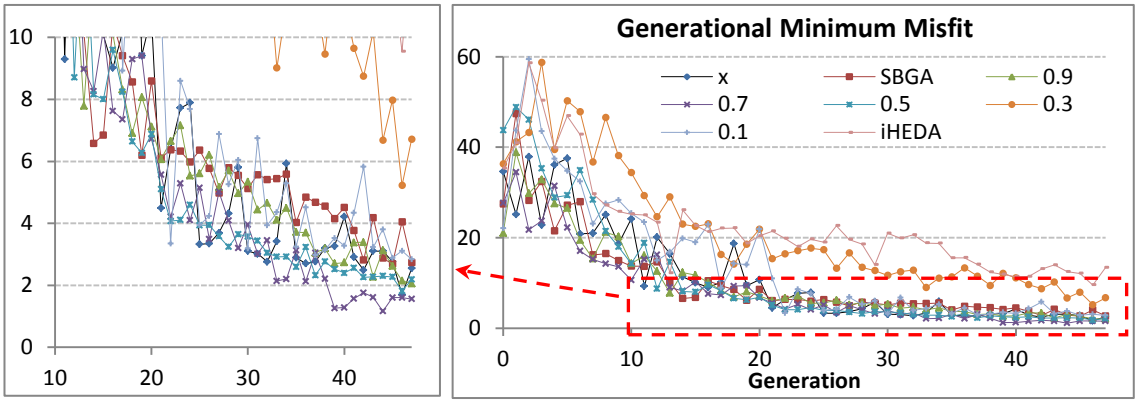


Figure 5.12: Minimum misfit per generation (left) and as zoomed for misfits under 10.0 (right) for SBGA, iHEDA and different participation ratios of hybrid SBGA/iHEDA. Hybrid SBGA/iHEDA with participation ratio of 0.7 shows the best performance.

We also analysed the history match quality of the oil and water production rates of the producer well for the best fitting model obtained by iHEDA, SBGA, and different participation ratios of SBGA/iHEDA. Figure 5.13 shows that acceptable matches are achieved for both oil and water rates using all the algorithms.

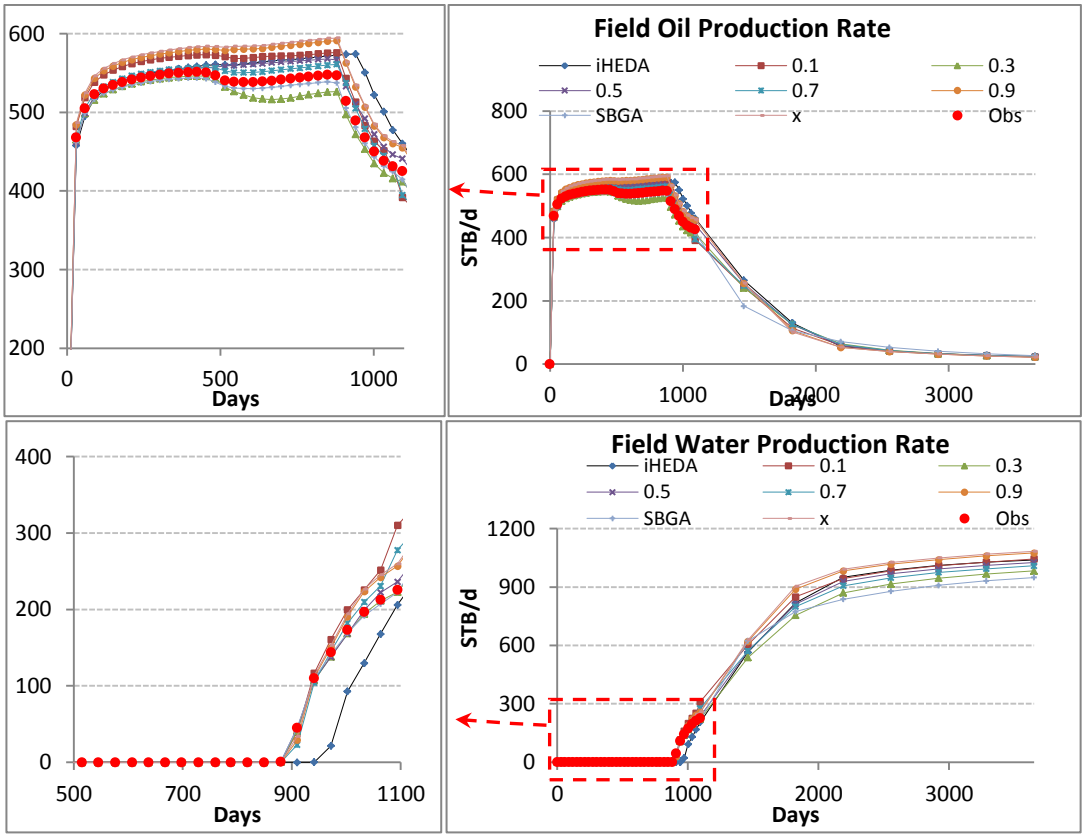


Figure 5.13: Match quality of oil production rate (top-right) and as zoomed for marked area (top-left), water production rate (bottom-right) and as zoomed for marked area (bottom -left), for best fitting models of iHEDA, SBGA, and different participation ratios of SBGA/iHEDA. Acceptable match is achieved with all the algorithms. Red dots show observations.

5.7 Teal South Reservoir

The IC Fault case study suggested that SBGA/iHEDA outperforms iHEDA and SBGA for a problem with steep local minima. In this section, these three algorithms were applied to history matching of a real field model to assess the generality of the IC Fault results. Thus, we evaluated and compared the performance of the algorithms in another history matching problem.

5.7.1 Field description

The Teal South reservoir is located near Eugene Island in the central Gulf of Mexico, southwest of New Orleans, in a water depth of 279-295 ft (Figure 5.14). This reservoir was originally developed by Mobil Oil in the mid 1980's and is currently being operated by Apache. Two exploration wells were drilled in 1994, which were followed by 17 more wells. The geological model of the reservoir is composed of the 4500 ft

sands, which are over-pressured, highly laminated channels that generally extend in the north-south direction.

We used segment 2 of the Teal South reservoir, containing only one horizontal well, which came into production in 1996. The production history of the reservoir consists of oil, gas and water rates for 1247 days. High flow rates of production for a small reservoir volume resulted in rapid depletion of the reservoir and thus a reduction in production rates after 180 days (Figure 5.15). The simulation model is constructed in 11×11×5 corner-point geometry, with uniform properties for each of the five geological layers in the model (Figure 5.14). Porosity is assumed to be fixed at 28% throughout the reservoir and permeabilities are unknown. In addition, rock compressibility in the model is unknown, and there is an aquifer with an unknown amount of pressure support.

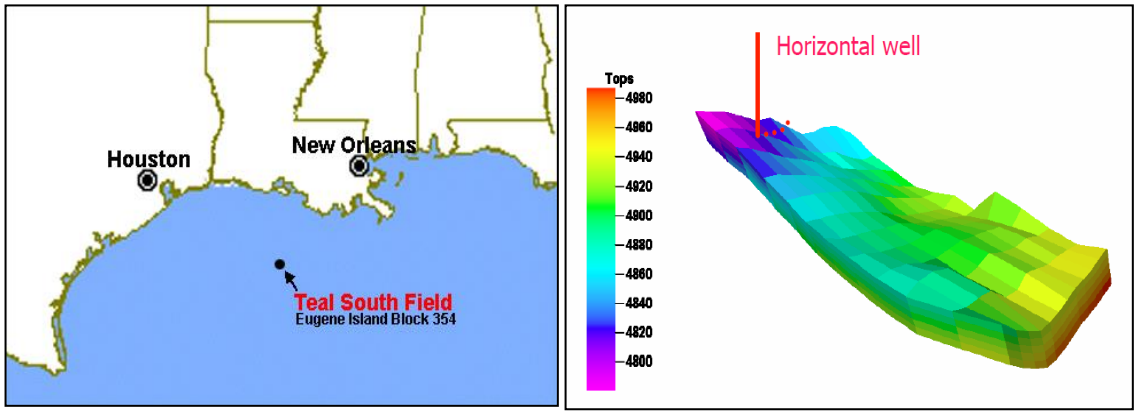


Figure 5.14: Location map (left), top structure and simulation model (right) of Teal South reservoir.

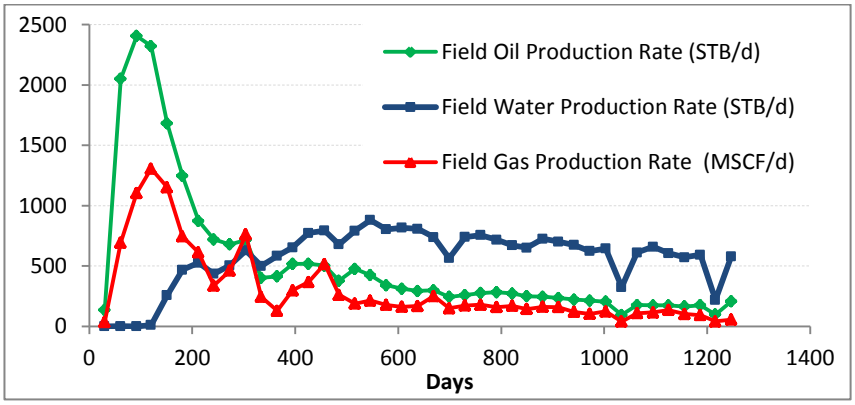


Figure 5.15: Production history of Teal South.

5.7.2 Uncertainty parameterization

We used the parameterisation of Christie et al. (2006) for history matching the Teal South field. Uncertainty parameters are horizontal permeability multipliers for each of

the five layers, a single value for each of the vertical to horizontal permeability ratio, rock compressibility, and aquifer strength respectively. A uniform prior range was taken as the parameter search space for each of these parameters (Table 5.6).

Table 5.6: Uncertainty parameters for Teal South reservoir.

Parameter	Uniform prior range
Horizontal permeability (K_h)	10 - 10^3 mD
Ratio of vertical to horizontal permeability (K_v /K_h)	10^{-4} – 10^{-1}
Rock compressibility	$5x10^{-6}$ – $1x10^{-4}$ psi^{-1}
Aquifer strength	10^7 – 10^9 MMSTB

5.7.3 Misfit definition

In the Teal South application, misfit was defined using a standard least squares model using the discrepancy between simulated and observed field oil production rates for the 1247 days of history (Figure 5.15). Random Gaussian noise was added to each measured point, with mean zero and standard deviation of 100 STB/d of the observed data. The misfit function is as follows:

$$Misfit = \frac{1}{2}\sum_{i=1}^{Np}\left(\frac{O_i - S_i}{\delta}\right)^2 \tag{5.11}$$

where:

- *Misfit* is the misfit function,
- i subscript running over observation data points at reported time,
- N_p is the number of observation points,
- O_i is observed oil rate measurement at time point i,
- S_i is simulated value of oil production rate at time point i,
- σ is the standard deviation of the oil rate measurement, equal to 100 STB/d of the oil rate.

5.7.4 Teal South results

We applied all three algorithms to Teal South model. Again, three control parameters were fixed for all three algorithms: the number of function evaluations at 1000, the

number of random initial solutions at 40, and the number of child solutions to be generated in each generation at 20. Other parameters, which are specific to the algorithms, are listed in Table 5.7. An identical initial random population was used in all of the experiments.

Table 5.7: Control parameters for Teal South case.

Algorithm	Parameter	Value
iHEDA	Number of parents	30
	Number of bins	20
	Learning rate	0.7
SBGA	Number of parents	20
	Mutation probability	0.1
	Simulated binary factor (*n*)	2
SBGA/iHEDA	Participation ratio	0.1, 0.3, 0.5, 0.7, 0.9

We looked at the minimum misfit, diversity, and convergence speed of the SBGA, iHEDA and the SBGA/iHEDA with different participation ratios. All three algorithms obtained very simillar minimum misfit (Figure 5.16 left), maintained some level of diversity (Figure 5.16 right), and converged to minimum misfit of around 9 (Figure 5.17). SBGA converges better than iHEDA and the hybrid algorithm, particularly in the early generations. After SBGA, the SBGA/iHEDA with fixed 70/30, 50/50 and adaptive participation ratios results in better performance. iHEDA has the worst convergence speed, especially at the later stage of the search.

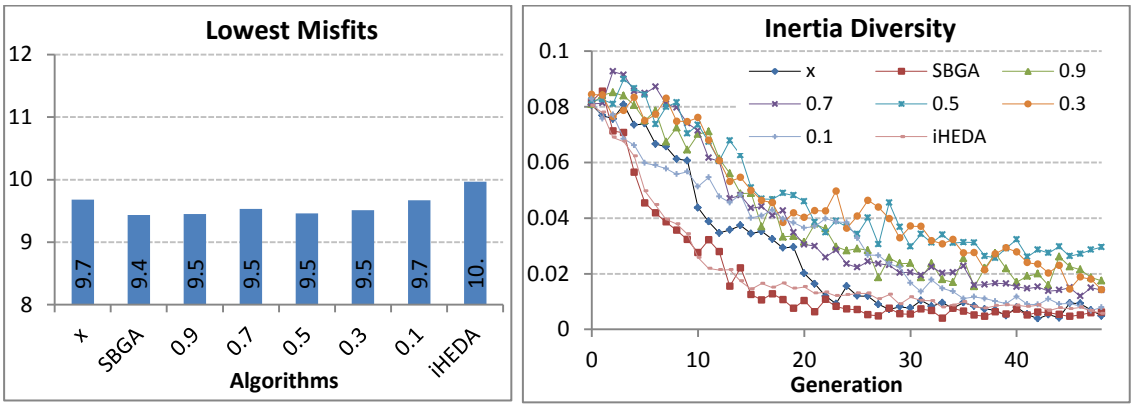


Figure 5.16: Lowest misfit found at entire evolution (left) and inertia diversity measure per generation (right) for SBGA, iHEDA, and different participation ratios of hybrid SBGA/iHEDA in the Teal South application. All algorithms obtained very similar minimum misfits. The diversity measure shows that all algorithms converged and maintained some level of diversity.

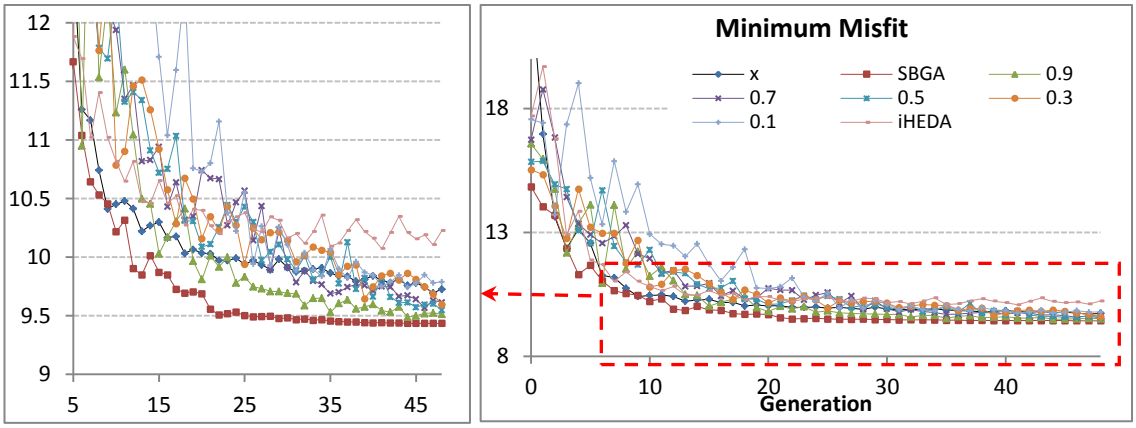


Figure 5.17: Generational misfit for SBGA, iHEDA and their hybrid algorithm with different participation ratios (right) and as zoomed for misfits under 12.0 (left). SBGA (dark red) has the best performance. Hybrid scheme improved iHEDA (purple); between the participation ratios fixed participation ratio of 0.9 (green) has the best performance.

We looked at the match for the quality of field oil production rate for the best fitting model obtained by each of iHEDA, SBGA, and different participation ratios of the SBGA/iHEDA. As Figure 5.18 shows, acceptable matches are achieved with all the algorithms.

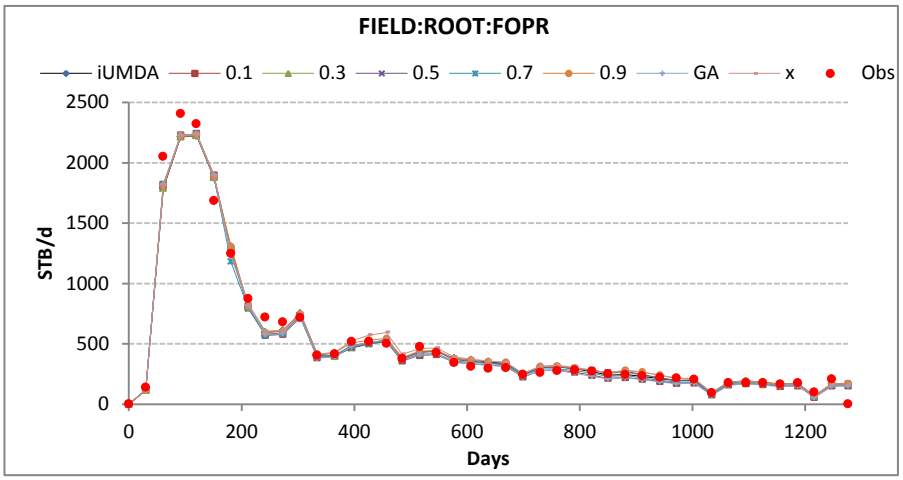


Figure 5.18: Match quality of field oil production rate for best misfit models of iHEDA, SBGA, and different participation ratios of SBGA/iHEDA. Acceptable match is achieved with all the algorithms. Red dots show the observations.

5.8 Discussion

When applied to IC Fault model, all three algorithms were able to find several minima with small enough misfit value and all converged to lower values of the misfit. When compared to iHEDA, SBGA, due to its exploratory characteristics, is able to perform a quick global search in the parameter space without being trapped in local minima. The

SBGA seems to be more appropriate than iHEDA for solving steep local minima problems, such as the IC Fault model.

The hybrid algorithm, with participation ratio of 0.7, outperformed both SBGA and iHEDA. The Rosenbrock and IC-Fault applications showed that hybridising can improve the performance of the SBGA by providing more exploitation and that of the iHEDA by providing more exploration properties of the search algorithm.

Good match quality was achieved by all three algorithms when applied to the Teal South reservoir. However, SBGA slightly outperformed iHEDA and the hybrid algorithm for the convergence speed.

In principle, combining the global search capability of GAs and early convergence advantage of EDAs can improve the performance of evolutionary algorithms for specific problems. Tests on different test functions have shown good results using a hybrid algorithm, GA/EDA (Robles et al., 2005) when compared to a pure GA and EDA. This was also shown by the three applications in this chapter.

5.9 References

Ballester, P. J, Carter, J. N. (2004). Real-Parameter Genetic Algorithms for Finding Multiple Optimal Solutions in Multi-Modal Optimization. Genetic and Evolutionary Computation Conference (Chicago, USA). Lecture Notes in Computer Science 2723, 706-717, Springer.

Baluja, S. (1994). Population based incremental learning: A method for integrating genetic search based function optimization and competitive learning. Technical Report No. CMU-CS-94-163, Carnegie Mellon University, Pittsburgh, Pennsylvania.

Carter, J.N., Ballester, P.J., Tavassoli, Z., King, P.R. (2006). Our calibrated model has poor predictive value: An example from the petroleum industry. The Fourth International Conference on Sensitivity Analysis: Reliability Engineering & System Safety, 91(10-11), 1373–1381.

Castellini, A., Yeten, B., Singh, U., Vahedi, A., Sawiris, R. (2006). History Matching and Uncertainty Quantification Assisted by Global Optimization Techniques. 10th European Conference on the Mathematics of Oil Recovery, Amsterdam, The Netherlands.

Christie, M.A., MacBeth, C., Subbey, S. (2002). Multiple history-matched models for Teal

South. The Leading Edge, March 2002, 286-289.

De Jong, K. A. (1975). An Analysis of the Behavior of a Class of Genetic Adaptive Systems, Ph.D. thesis, University of Michigan, Ann Arbor, Michigan.

Deb, K. and Agrawal, R. B. (1995). Simulated binary crossover for continuous search space. Complex Systems, 9, 115–148.

Deb, K., Joshi, D. & Anand, A. (2001). Real-coded evolutionary algorithms with parent centric recombination, Tech. Rep. KanGAL Report No. 2001003, Indian Institute of Technology, Kanpur, India.

Erbas, D., & Christie, M.A. (2007). Effect of Sampling Strategies on Prediction Uncertainty Estimation. SPE Number: 106229-MS. SPE Reservoir Simulation Symposium. Houston, U.S.A.

Goldberg, D. E. (1989). Genetic Algorithms in Search Optimization and Machine Learning. Boston, MA, USA: Addison Wesley.

Gómez S., Gosselin O. Barker J.W. (1999). Gradient-Based History matching With A Global Optimization Method. SPE Number: 56756, Annual Technical Conference and Exhibition, Houston, USA.

Leitão, H. C. and Schiozer, D. J. (1999). A New Automated History Matching Algorithm improved by Parallel Computing. SPE Number: 53977. SPE Latin American and Caribbean Petroleum Engineering Conference, Caracas, Venezuela.

Mantica, S., Cominelli, A., Mantica, G. (2002). Combining Global and Local Optimization Techniques for Automatic History Matching Production and Seismic Data. SPE Number: 78353-PA. SPE Journal, 7(2), 123-130.

Mühlenbein, H. (1997). The equation for response to selection and its use for prediction. In Evolutionary Computation, 5(3), 303–346.

Reynolds, A., Abdollahzadeh, A., Christie, M.A., Corne, D., Williams, G., Davies, B. (2011). A Parallel BOA-PSO Hybrid Algorithm for History Matching. IEEE Congress on Evolutionary Computation (CEC). New Orleans, LA, USA.

Robles, V., Peña, J.M., Perez, M.S., Herrero, P., Cubo, O. (2005). Extending the GA-EDA hybrid algorithm to study diversification and intensification in GAs and EDAs. Proceedings of the 6th International Symposium on Intelligent Data Analysis IDA, 3646, 339-350. Berlin, Germany: Springer-Verlag GmbH.

Romero, C., Carter, J., Gringarten, A., & Zimmerman, R. (2000). A Modified Genetic Algorithm for Reservoir Characterisation. SPE Number: 64765-MS. International Oil and Gas Conference and Exhibition. Beijing, China.

Santana, R., Larrañaga, P., & Lozano, J.A. (2007). Combining variable neighbourhood search and estimation of distribution algorithms in the protein side chain placement problem. Journal of Heuristics. 14: 519-547.

Schulze-Riegert, R., Krosche, M., & Mustafa, H. (2009). Data Assimilation Coupled to Evolutionary Algorithms - A Case Example in History Matching. SPE Number: 125512-MS. SPE/EAGE Reservoir Characterization and Simulation Conference. Abu

Dhabi, UAE.

Williams, G. J. J., Mansfield, M., MacDonald, D. G., Bush, M. D. (2004). Top-Down Reservoir Modelling. SPE Number: 89974. SPE Annual Technical Conference and Exhibition held in Houston, Texas, U.S.A.

Zhang, Q., Sun, J. & Tsang, E. (2005). EDA+GA: Evolutionary Algorithm with Guided Mutation for the Maximum Clique Problem, IEEE Trans. on Evolutionary Computation, 9 (2), pp 192-200.

CHAPTER 6:

GAUSSIAN-BASED ESTIMATION OF DISTRIBUTION ALGORITHMS

> "It is not knowledge, but the act of learning, not possession but the act of getting there, which grants the greatest enjoyment."
>
> Carl Friedrich Gauss (1777 - 1855)

6.1 Introduction

In history matching, most of the optimisation variables, e.g. porosity and permeabilities are real-valued. The EDAs that so far have been applied to history matching normally work with discrete variables and one must discretise the variable search space into a number of bins when these algorithms are used for the real-valued problems. The discretisation can be potentially impractical, as some values close to each other in the parameter's continuous domain may become far away in the discrete domain.

Another problem is that discretisation does not reflect the evolution process. Some regions of the parameter space are searched thoroughly and will be densely covered with high quality solutions whereas others contain mostly low quality solutions. Although some efforts have been made to adapt representation of continuous domains and split bins into two (e.g. Chen et al., 2006), the intuitive approach is to use EDAs that work directly with a population of real-valued vectors.

While the EDAs described in this study so far can be adapted to handle larger alphabets, adapting them to handle real-valued problems is more difficult. Although univariate EDAs exist (see for example Rudlof & Koppen, 1996) and others have been created

during this thesis, creating a true multivariate real-valued EDA capable of handling multi-modal search spaces is implemented in the current chapter.

This chapter introduces four Gaussian-based EDAs for history matching. For modelling and sampling, the first algorithm uses single univariate Gaussian distribution, the second algorithm uses a single multivariate Gaussian model, the third uses multiple univariate Gaussian models (or univariate Gaussian mixture models), and the fourth uses multiple multivariate Gaussian models. We use these real-valued EDAs for the history matching problem of a synthetic model, PUNQ-S3.

The chapter is organised as follows: the Methodology section introduces the basic EDA procedure and gives a description of the four Gaussian-based EDAs used in this work. The Application section presents the result of test function applications, introduces the field description and history matching problems and presents the results of this history matching application. In the discussion section, we investigate the results and discuss the generality of the findings.

6.2 Gaussian-based EDAs

A discussed earlier e.g. see Chapters 3 and 4), a large number of EDAs have been introduced and used for optimisation problems (e.g., see Pelikan et al., 2000). These algorithms primarily differ in the type of probabilistic model they use. Gaussian models are the most popular choice of probability distribution. In this chapter, we use three Gaussian-based algorithms which allow direct representation of real-valued problems. The central assumption for all these algorithms is that the misfit landscape is normally distributed. Gaussian-based algorithms are described in the following sections:

6.2.1 Incremental Single Univariate Gaussian Estimation of Distribution Algorithm

The Incremental Univariate Single Gaussian Estimation of Distribution Algorithm (iSUGEDA) extends the concept of incremental learning for univariate marginal estimation of distribution to real-valued problems by representing variables of optimisation problems using a single Gaussian distribution. Thus, the algorithm borrows ideas from three algorithms: *PBILc* of Sebag and Ducoulombier (1998), the incremental

univariate marginal distribution algorithm (*iUMDA)* of Mühlenbein (1997), and the univariate marginal distribution algorithms for continuous domains (*UMDAc)* of Larrañaga et al. (1999).

Univariate Gaussian distribution uses a normal (Gaussian) function for expressing a distribution in one dimension. According to the *central limit theorem*, the average of identically distributed random parameters is approximately normally distributed, regardless of their original distributions. This makes the Gaussian distribution one of the most common probability distributions. The univariate Gaussian distribution has the following probability density function:

$$\mathcal{N}(\bar{x}_i, \sigma_i) = \frac{1}{\sqrt{2\pi}\sigma_i} e^{-\frac{1}{2}(\frac{x_i - \bar{x}_i}{\sigma_i})^2} \tag{6.1}$$

where $\boldsymbol{x} = (\boldsymbol{x_1}, \ldots, \boldsymbol{x_n})$ is the vector of n parameters and subscript i iterates over parameters, and $\boldsymbol{\mathcal{N}}$ is the Gaussian distribution function represented by the mean value $\bar{\mathbf{x}}_\mathbf{i}$ and the standard deviation $\boldsymbol{\sigma_i}$ of each parameter.

Sebag and Ducoulombier (1998) were the first to introduce a univariate EDA based on the Gaussian distribution. They extended PBIL to real-valued spaces by using a single Gaussian model for the distribution of the population and named their algorithms PBILc. Larrañaga et al. (1999) extended the univariate marginal distribution algorithm (UMDA) to continuous domains by replacing the basic histogram model with a single Gaussian distribution model. Thus, both PBILc and HEDAc evolve the following joint distribution, which is used in iSUGEDA:

$$JPD(x) = \prod_{i=1}^{n} \mathcal{N}(\bar{x}_i, \sigma_i) \tag{6.2}$$

For sampling new solutions from a univariate Gaussian distribution with mean $\bar{x}$ and standard deviation $\boldsymbol{\sigma}$, one can first sample from an independent univariate normal deviate, $Z = \boldsymbol{\mathcal{N}}(\mathbf{0}, \mathbf{1})$, then transform the samples to the desired distribution:

$$X = \bar{X} + \sigma Z \tag{6.3}$$

where ***X*** is the vector of samples. The samples that are within the allowed bound for the parameters need to be checked.

In each generation, unlike HEDAc, PBILc uses an incremental learning mechanism to update the parameters ($\bar{x}_i$ and σ_i) of the univariate Gaussian model for each parameter as follows:

$$\bar{x}_i^{\,t} = (1-\alpha).\bar{x}_i^{\,t-1} + \alpha.(x_i^{\,best1} + x_i^{\,best2} - x_i^{\,worst})$$
$$\sigma_i^{\,t} = (1-\alpha).\sigma_i^{\,t-1} + \alpha.(\sqrt{\frac{\sum_{j=1}^{P}(x_i^j - \bar{x}_i)^2}{P}}) \quad (6.4)$$

where superscript ***t*** represents current generation, superscript ***best1***, ***best2***, and ***worst*** are respectively first best, second best, and worst solutions in the set of ***P*** selected solutions. Superscript ***j*** iterates over these solutions. In the equation, standard deviation ($\boldsymbol{\sigma_i}$) is updated using the spread of solutions $\boldsymbol{\alpha}$ is the learning factor between 0.0 and 1.0 and determined after a tuning study.

In iSUGEDA, we used the same equation as (6.4) for updating standard deviation, but the mean is updated using the incremental learning mechanism of iUMDA (Mühlenbein 1997) as:

$$\bar{x}_i^{\,t} = (1-\alpha).\bar{x}_i^{\,t-1} + \alpha.(\bar{x}_i^{\,P}) \quad (6.5)$$

where $\bar{\boldsymbol{x}}_{\boldsymbol{i}}^{\boldsymbol{P}}$ is the mean of parameter ***i*** in the set of ***P*** selected solutions.

iSUGEDA is a straightforward and efficient EDA for real-valued problem, but it has a serious deficiency for multimodal problems: like PBILc, it uses only a single normal distribution for each variable and it is only able to capture distributions accurately that are all centred around a single point in the search space. Many history matching applications have a potential problem with the single Gaussianity assumption. In addition, like PBILc and iHEDA, which is not able to model interaction between parameters, since sampling from the joint probability distribution assumes that all the parameters are independent.

6.2.2 Single Multivariate Gaussian Estimation of Distribution Algorithm

One can use a multivariate Gaussian distribution as the probabilistic model of EDA to get around the problem of not considering the interaction between parameters in iSUGEDA. By using a multivariate Gaussian distribution, we will be able to estimate the means and covariance matrices of the parameters from the selected promising solutions of the population and use them to generate new child solutions. A covariance matrix is able to explain the shape of the learnt distribution.

The multivariate Gaussian distribution is a generalisation of the univariate Gaussian distribution to higher dimensions. It describes a set of possibly correlated real-valued random variables, each of which clusters around a mean value. The multivariate Gaussian distribution has the following density function:

$$\mathcal{N}(\bar{x}_i, \Sigma) = \frac{1}{\sqrt{(2\pi)^2}\sqrt{|\Sigma|}} e^{-\frac{1}{2}(x_i-\bar{x}_i)^T\Sigma^{-1}(x_i-\bar{x}_i)} \quad (6.6)$$

where $\boldsymbol{\Sigma}$ is the covariance matrix of the multivariate Gaussian distribution and it must be symmetric and positive definite, $|\boldsymbol{\Sigma}|$ is the determinant of the covariance matrix, superscript $\boldsymbol{T}$ represents transpose matrix function, and $\boldsymbol{\Sigma^{-1}}$ is the inverse of the covariance matrix. Thus, if we assume the joint probability distribution of parameters is a multivariate Gaussian distribution, we will have:

$$JPD(x) = \prod_{i=1}^{n} \mathcal{N}(\bar{x}_i, \Sigma) \quad (6.7)$$

The estimation of the Gaussian networks algorithm (EGNA) (Larrañaga, Etxeberria, Lozano, & Pena, 1999) was one of the first attempts to create an EDA for a continuous domain, based on multivariate Gaussian distribution. EGNA uses a Gaussian network to model the interactions between variables in the selected population of solutions in each generation. The Gaussian network structure is learnt by using a separate local search with the objective of optimising the Bayesian-Dirichlet score.

Real-coded Estimation of Distribution Algorithm (RECEDA) is another multivariate Gaussian-based EDA that uses Cholesky decomposition for sampling from the

multivariate Gaussian distribution. In RECEDA, first the mean vector ($\bar{\boldsymbol{X}}$) and covariance matrix ($\boldsymbol{\Sigma}$) of parameters are calculated from the selected solutions, and then, if the covariance matrix is real, symmetric and positive definite, it can be decomposed using Cholesky decomposition to get a lower triangular matrix with strictly positive diagonal entries, $\boldsymbol{L}$, so that $\boldsymbol{L}.\boldsymbol{L}^{\boldsymbol{T}} = \boldsymbol{\Sigma}$. Finally, new solutions are created by transforming an independent univariate normal deviate, $Z = \boldsymbol{\mathcal{N}}(\mathbf{0}, \mathbf{1})$:

$$X = \bar{\mathrm{X}} + L.Z \tag{6.8}$$

where $\boldsymbol{X}$ is the vector of samples and $\boldsymbol{L}$ is a lower triangular matrix obtained from the Cholesky decomposition of the covariance matrix. Again, new samples need to be checked that are within the allowed bound for the parameters.

We introduce the Single Multivariate Gaussian Estimation of Distribution Algorithm (SMGEDA). SMGEDA uses Eigendecomposition for sampling from the multivariate Gaussian distribution. In each generation similarly to in RECEDA, we obtain the mean vector and covariance matrix of the parameter values in the selected solutions. Then we compute eigenvalues and eigenvectors of the covariance matrix such that $\boldsymbol{\Sigma}.\boldsymbol{v} = \boldsymbol{\lambda}\boldsymbol{v}$, where $\boldsymbol{v}$ is eigenvector and $\boldsymbol{\lambda}$ is a scalar, eigenvalue of matrix $\boldsymbol{\Sigma}$. Then we draw the desired number of samples from an independent univariate normal deviate $Z = \boldsymbol{\mathcal{N}}(\mathbf{0}, \mathbf{1})$**.** Finally samples are transferred to multivariate Gaussian distribution using:

$$X = \bar{\mathrm{X}} + (v.\sqrt{\boldsymbol{\lambda}}I).Z \tag{6.9}$$

Although Cholesky decomposition is more computationally straightforward, the matrix $\boldsymbol{L}$ changes for different orderings of the elements of the random vector $\boldsymbol{Z}$**,** while the Eigendecomposition approach gives matrices that are related by simple re-orderings. Another benefit of Eigendecomposition is that eigenvalues can be used to repair an ill-posed covariance matrix, as discussed in the next section.

6.2.3 Multiple Univariate Gaussian Estimation of Distribution Algorithm

The problem of single Gaussianity in iSUGEDA leads us to use a finite Gaussian mixture model as density estimator, instead of a single Gaussian model. This allows the univariate algorithms to deal with multimodal distributions and explore different regions of the search space simultaneously. A survey in the literature reveals the following two algorithms based on the mixture of Gaussian models.

Gallagher et al. (1999) introduced the first EDA, which uses Gaussian mixtures. They extended PBIL to real-valued spaces by using an Adaptive Gaussian mixture model instead of a single probability vector. They used a mixture of Gaussian distribution models with a recursive update rule, i.e. models are modified and improved gradually as a new point is sampled.

The mixed iterated density estimation evolutionary algorithm (mIDEA) (Bosman & Thierens, 2001) is another EDA that also uses mixtures of Gaussian distributions. The probabilistic model building in mIDEA starts by k-means clustering the parameters, then a probability distribution is fitted over each cluster for each parameter. Finally, a weighted sum over the individual distributions of each cluster is taken as the final probability distribution of the parameter. The weight factors are the ratio of the number of solutions in each cluster to the total number of solutions. Within the IDEA framework, Bosman & Thierens (2001) used joint normal kernel distribution, where a single Gaussian distribution is placed around each selected solution.

Using a similar concept, we propose MUGEDA, in which, we use a mixture Gaussian probability distribution model based on the probabilistic distance clustering (Ben-Israel, 2006; Iyigun & Ben-Israel, 2008) for each parameter. The final probability distribution is a weighted sum of multiple probability distributions. The weight factors must be positive and sum to 1 to ensure that mixture model is still a probability distribution. The joint probability distribution of parameters is expressed as:

$$JPD(x) = \prod_{i=1}^{n}\left(\sum_{k=1}^{K} \beta_k \mathcal{N}_k(\bar{x}_{ik}, \sigma_{ik})\right) \tag{6.10}$$

where $\boldsymbol{k}$ iterates over the number of mixture models ($\boldsymbol{K}$), $\boldsymbol{\beta_k}$ is the weight factor for mixture model $\boldsymbol{k}$, such that $\boldsymbol{\beta_k} \geq \mathbf{0}$ and $\sum_{k=1}^{K} \boldsymbol{\beta_k} = \mathbf{1}$, $\boldsymbol{\mathcal{N}_k}(\bar{\boldsymbol{x}}_{\boldsymbol{ik}}, \boldsymbol{\sigma_{ik}})$ is the Gaussian distribution model for model $\boldsymbol{k}$, with mean $\bar{\boldsymbol{x}}_{\boldsymbol{ik}}$ and standard deviation $\boldsymbol{\sigma_{ik}}$.

In each generation of MUGEDA, we perform clustering of the solutions in the population by probabilistic distance clustering of each parameter; a Gaussian probability distribution is fitted over each cluster using the mean and variance of the parameter in the solutions. Thus, each cluster represents a local probabilistic model and multiples of clusters create the mixture distribution model. Weight factors are calculated from the spread of clusters:

$$\beta_{\mathrm{k}} = \frac{S_k}{\sum_{k=1}^{K} S_k} \; ; \; S_k = \frac{1}{\mathrm{n_k}} \sum_{\mathrm{i}=1}^{\mathrm{n_k}} \sum_{p=1}^{D} \left(x_{pi} - c_{pk}\right)^2 \tag{6.11}$$

where $\boldsymbol{S_k}$ is the spread of cluster $\boldsymbol{k}$, and subscript $\boldsymbol{i}$ iterates over the number of solutions in cluster $\boldsymbol{k}$, $\boldsymbol{n_k}$.

It is not possible to use incremental learning in Gaussian mixture-based EDAs, since the mixture models are created from scratch in each generation and no connection can be created between mixture models in two consecutive generations. The algorithm uses one-dimensional clustering for constructing mixture models and new solutions are sampled from univariate normal distribution of the variables: thus final joint probability distribution is the product of marginal probabilities and interactions between variables are not considered.

6.2.4 Multiple Multivariate Gaussian Estimation of Distribution Algorithms

There are two attempts reported in the literature in which a mixture of joint Gaussian probability distribution has been used as the probability model in EDAs. Larrañaga, Etxeberria, Lozano, & Pena (1999) extended EGNA to support multivariate Gaussian mixtures by creating a Gaussian network to model the interactions between variables in the selected population of solutions in each generation. Bosman & Thierens (2000) reported good results using mIDEA, when extended to use a mixture of joint normal distributions.

Using the full covariance matrix in multivariate Gaussian-based EDAs has a potentially serious problem. The computation error for the covariance matrix increases when sample size is relatively small in relation to problem size. In this case, the covariance matrix could not be a positive semi-definite and will have negative eigenvalues because of the finite precision and computational power of the computer. For univariate Gaussian-based EDAs, this is not a problem, as they only use diagonal elements of the covariance matrix (variances), which are always assured to be non-negative.

We tried a multiple multivariate Gaussian EDA (MMGEDA) using a mean and covariance matrix of clusters obtained by Probabilistic Distance Clustering (PDC), (see section 2.6.3 of chapter 2), in the same way we used in the MUGEDA. We acknowledge that learning such a general probability distribution is quite difficult and a large number of samples is required for satisfactory accuracy. With a relatively small number of parent solutions in high dimensional history matching problems, clusters may get a very small number of solutions. In these situations, the covariance matrix repairing techniques may be required to be used as the covariance matrix is likely to become ill-posed, and sampling from an ill-posed covariance matrix is not possible.

The covariance matrix repairing (CMR) technique can resolve the ill-posed covariance matrix problem for such EDAs. An ill-posed covariance matrix is a common problem in the statistical learning field, and is usually solved by regularisation techniques, e.g. adding a positive value to the diagonal of the matrix, or shrinking the covariance matrix towards an identity matrix. In EDAs, such changes must ensure that they do not affect the distribution of solutions in the following generations. To make a covariance matrix positive and semi-definite, we used the CMR approach introduced by Dong & Yao (2007) as follows:

1. In each generation, do clustering of the parents.
2. For each cluster $\boldsymbol{j}$:
   2.1. Compute mean $\bar{\boldsymbol{x}}_{\boldsymbol{j}}$ and covariance matrix $\boldsymbol{\Sigma}_{\boldsymbol{j}}$ of solutions in the cluster.
   2.2. Obtain eigenvalues and eigenvectors of $\boldsymbol{\Sigma}_{\boldsymbol{j}}$ using eigendecomposition, initialise $\boldsymbol{k}$=1.
   2.3. Check for the minimum value of eigenvalues, $(\boldsymbol{\lambda}_{\boldsymbol{min}})$, if $\boldsymbol{\lambda}_{\boldsymbol{min}}$ is negative, do following until $\boldsymbol{\lambda}_{\boldsymbol{min}}$ becomes positive:

2.3.1. Add an identity matrix with values of $|\boldsymbol{\lambda_{min}}|.\mathbf{k}$ to $\boldsymbol{\Sigma_j}$.

2.3.2. Increase $\boldsymbol{k}$ by a factor, e.g. $\boldsymbol{k} = \boldsymbol{k} * \mathbf{1.5}$.

2.3.3. Recalculate minimum eigenvalue of $\boldsymbol{\Sigma_j}$.

6.3 Applications

6.3.1 Test Functions

All of the four Gaussian-based algorithms were initially validated with three well-known test functions: Sphere, Rastrigin, and Griewank functions. The aim was to apply and tune algorithms on three typical test functions for the objective function's multimodality and dependency between variables.

6.3.1.1 Sphere test function

The sphere function, first used by De Jong (1975), is one of the simplest and most standard unimodal test functions for a convergence study of the numerical optimisation algorithms. The function is continuous and unimodal. In addition, it has no dependencies between variables, and is therefore a good test candidate for single univariate EDAs. The global minimum value for the Sphere test function obtained for any dimensionality is zero if all variables are set to zero. The function is as follows:

$$f(x) = \sum_{i=1}^{n} x_i^2 \tag{6.12}$$

where $\boldsymbol{x_i} \in [-\mathbf{5.12}, \mathbf{5.12}]$, and in this chapter we take n=5.

6.3.1.2 Rastrigin function

The Rastrigin function, the equation (6.13) and Figure 6.1, is based on the Sphere function with the addition of cosine modulation to generate frequent local minima. It was first proposed by Rastrigin (Törn & Zilinskas, 1989) in the form of a 2-dimensional function then generalised as an n-dimensional function. Variables are independent and it is a highly nonlinear and symmetric function, with many local minima in the form of

valleys. However, it has one global optimum, ***f(x)=0***, at the centre point, ***x=[0, ..., 0]***. Many optimisation algorithms, especially gradient-based methods, have difficulty with finding the global optimum in the Rastrigin function. We use this function as an example for problems with multimodal objective function landscape.

$$f(x) = 10n \sum_{i=1}^{n} (x_i^2 - 10\cos(2\pi x_i)) \tag{6.13}$$

where, $\boldsymbol{x} = (\boldsymbol{x_1}, \dots, \boldsymbol{x_n})$, $\boldsymbol{n} = \mathbf{5}$, and $\boldsymbol{x_i} \in [-\mathbf{5.12}, \mathbf{5.12}]$.

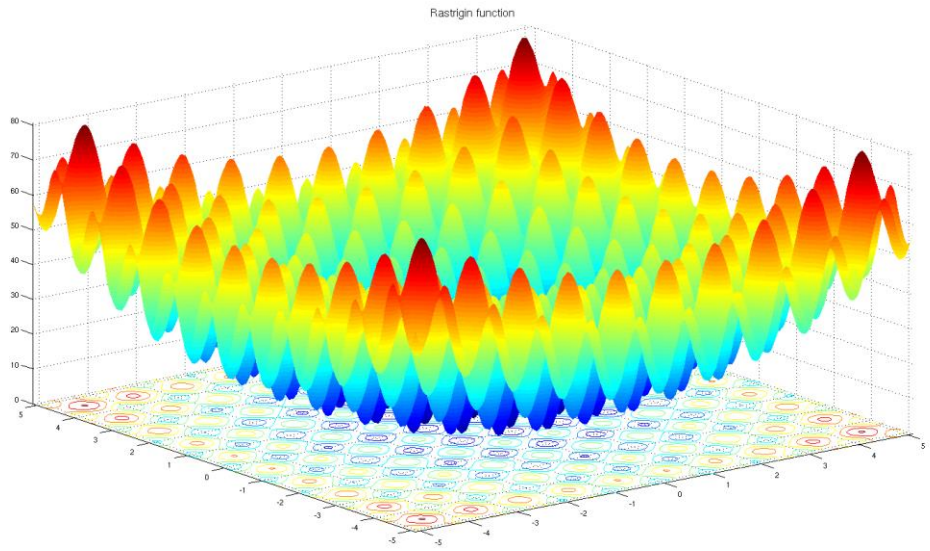


Figure 6.1: A Rastrigin function for two variables.

6.3.1.3 Griewank function

The Griewank function (Griewank 1981) is similar to the Rastrigin function, with many widespread, regularly distributed local minima. It has a product term that introduces interdependence among the variables. Thus, it can show the failure of the univariate algorithms. This function is given by:

$$f(x) = \frac{1}{4000} \sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1 \tag{6.14}$$

where $\boldsymbol{n} = \mathbf{5}$, $\mathbf{and}\ \boldsymbol{x_i} \in [-\mathbf{600}, \mathbf{600}]$. Again, the global optimum, *f(x) =0*, can be obtained, at centre point, $\boldsymbol{x_i} = (\mathbf{0}, \dots, \mathbf{0})$.

6.3.1.4 Test function results

We tested Gaussian-based EDAs on these three test functions, using a total of 5,000 function evaluations as the stopping criterion for the algorithms; we also fixed the total number of children to be generated in each generation to 100. Due to the stochastic nature of the algorithms, each experiment was repeated 10 times and the average of 10 trials used for the comparison study. Table 6.1 shows a summary of the control parameter tuning. Best values found for control parameters of each algorithm when applied to each of the test functions are shown. Our criteria for the performance of the algorithms were minimum objective function values, with 95% significance level t-test standard error as error bars, and generational minimum objective function, representing the convergence speed of the algorithm.

Table 6.1: Summary of algorithms' tuning on test functions. Bold numbers show best values that are significantly better than other values.

Algorithm	Control parameter	Tested values	Best value on SP	Best value on RR	Best value on GW
iSUGEDA	Number of parents	100, 200, 300, 500	**100**	100	**100**
	Learning rate	0.3, 0.5, 0.7, 0.9, 0.95, 1.0	0.95	0.95	1.0
SMGEDA	Number of parents	100, 200, 300, 500	**100**	100	**100**
	Decomposition type	CD and EV	EV	EV	EV
MUGEDA	Number of parents	100, 200, 300, 500	**100**	**100**	**100**
	Number of clusters	2, 5, 10, 20	**2**	**10**	**10**
MMGEDA	Number of parents	100, 200, 300, 500	**100**	**100**	**100**
	Number of clusters	2, 5, 10, 20	**2**	**10**	**10**

Figure 6.2 shows the average of the minimum objective function and objective function convergence for 10 trials of the Sphere test function. One can see that single Gaussian-based algorithms (SUGEDA and SMGEDA) converge well and even marginally outperform multiple Gaussian-based algorithms. SUGEDA performs reasonably on SP, as no dependency between the variables exists. SUGEDA is a special case of SMGEDA where all non-diagonal elements of the covariance matrix are zero; thus, it is always expected that both algorithms produce similar results for a univariate problem, although SMGEDA involves more computations for the covariance matrix computation and decomposition.

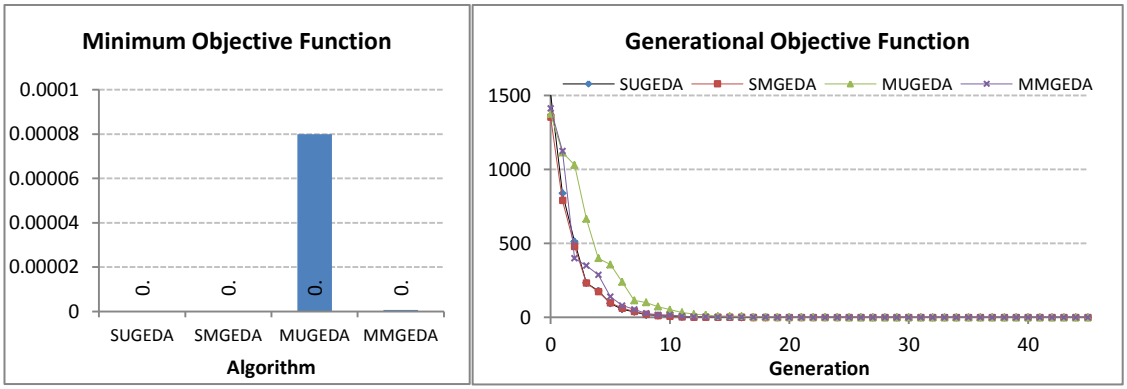


Figure 6.2: Results of the objective function convergence for the sphere test function; the average minimum objective function (left) and generational minimum objective function (right) of 10 trials. Single models outperform mixture models.

As discussed earlier, the Rastrigin test function is a test for multimodality. Figure 6.3 shows results of 4 algorithms on the Rastrigin function. One can see that from the generation 20, multimodal-enabled models (MUGEDA and MMGEDA) outperform single models, in terms of convergence to minimum objective function value.

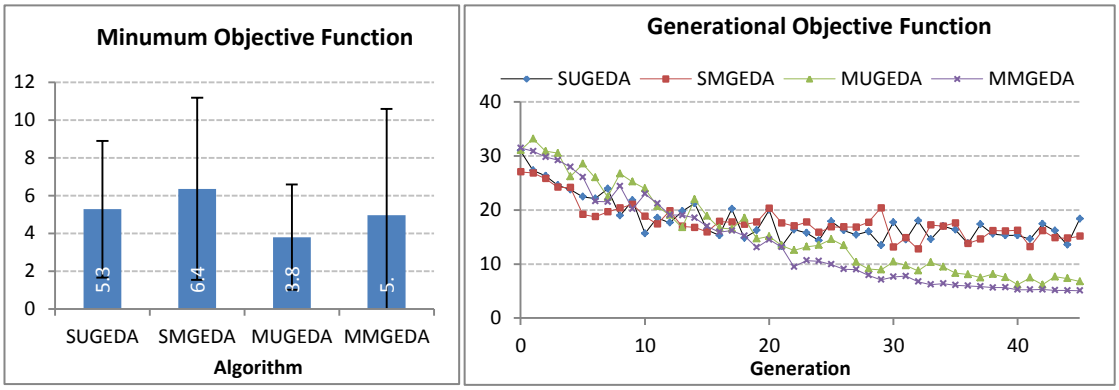


Figure 6.3: Objective function convergence results for the Rastrigin test function. The average minimum objective function (left) is marginally improved while the generational minimum objective function (right) is significantly improved for the mixture models from generation 20.

The final test function was the Griewank test function. Apart from multiple local minima, there is some level of interaction between variables in the Griewank test function. We hence used it as a test for multivariate Gaussian EDAs. As indicated in Figure 6.4, between the two multiple Gaussian-based models, MMGEDA can benefit from these interactions to improve the local optima and convergence speed.

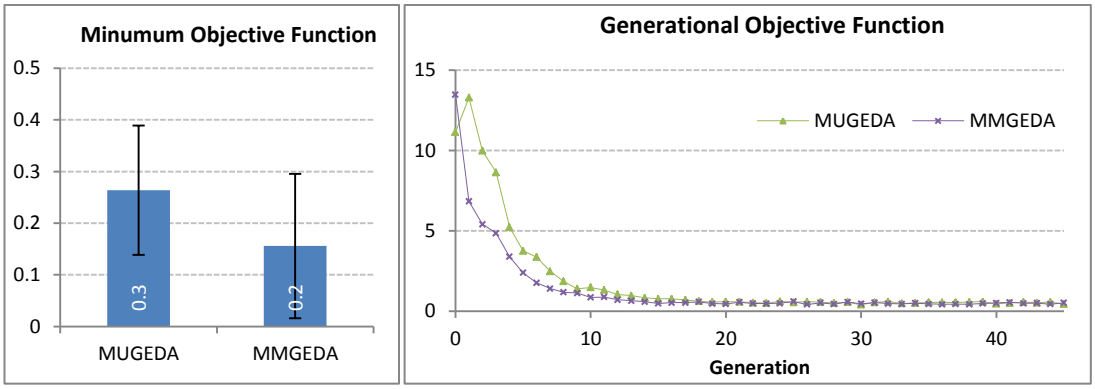


Figure 6.4: Griewank function results using two multiple Gaussian-based models. MMGEDA outperforms MUGEDA for lowest objective function (left) and convergence speed (right). Results are averaged for 10 independent trials.

6.3.2 History Matching of PUNQ-S3

Having tested and tuned Gaussian-based algorithms on selected test functions, we applied them to the history matching problem of the PUNQ-S3 reservoir (Floris et al., 2001). Our aim was to study the convergence performance of different Gaussian-based EDAs on a well-known synthetic case for history matching.

In Chapter 3, three EDAs were applied to PUNQ-S3 model, which naturally work with discrete variable ranges, thus one should discretise the continuous variables e.g. porosity and porosity/permeability correlation coefficients in PUNQ-S3 model to make them discrete. In current chapter, we use Gaussian-based EDAs for history matching PUNQ-S3 model, which naturally work with continuous variables in PUNQ-S3 model. Thus one can expect better performance of Gaussian-based EDAs compared to histogram-based and BOA EDAs in continuous (real-valued) applications.

Initially, all four algorithms were tuned for their best control parameters. These algorithms are all stochastic: thus, we repeated each experiment 10 times and averaged the results for misfit to minimise the effect of randomness. In history matching, function evaluations (simulation runs) are generally highly expensive, thus one must use the lowest possible number of evaluations. Since the parameterisation used for PUNQ-S3 in this chapter includes 38 uncertainty parameters, the number of new solutions to be created in each generation was fixed at 50 (>38) to give the algorithms sufficient degrees of freedom. We also set the stopping criterion to be the total of 3,000 function evaluations; with 150 random solutions in the initial population, there will be 57 generations in total.

6.3.2.1 Results of SUGEDA on PUNQ-S3

The first algorithm tested and tuned on PUNQ-S3 was SUGEDA. By fixing the total number of generations and number of child solutions in each generation, SUGEDA still has two other control parameters that must be tuned: the number of parents in each generation and the learning rate for the incremental learning mechanism.

Figure 6.5 shows averaged minimum misfit and misfit convergence of SUGEDA for three possible choices of the number of parents. As the figure shows, the setup of 50 parents per generation gives best minimum misfit and generational misfit convergence.

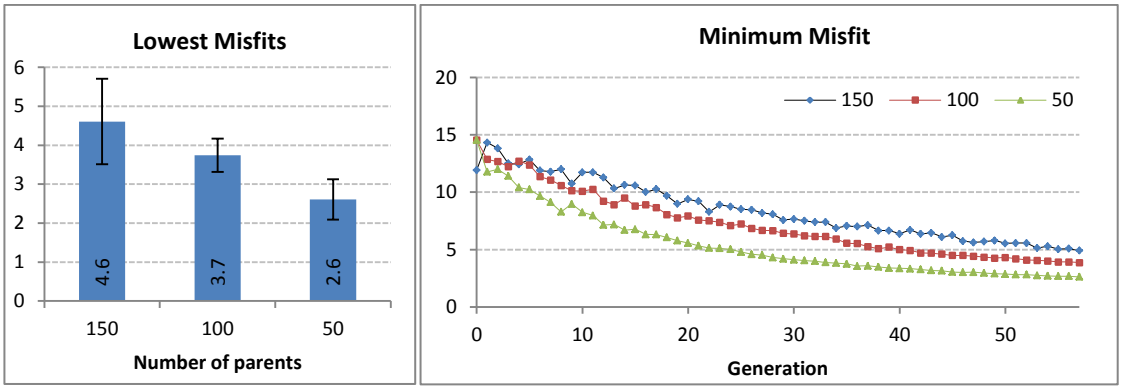


Figure 6.5: Averaged minimum misfit (left) and generational minimum misfit (right) for different numbers of parents in SUGEDA. 50 parents per generation gives the best misfit convergence.

As SUGEDA uses an incremental learning mechanism, the second tuning parameter was the learning rate. We tested 6 different learning rates, 1.0, 0.95, 0.9, 0.7, 0.5, 0.3. A learning rate of 1.0 means that the algorithm does not consider the probabilistic model of the previous generation and only uses the model learnt from the parent solutions in the current generation. As Figure 6.6 shows, SUGEDA is not very sensitive to the learning rate. However, a learning rate of 0.95 was slightly preferred..

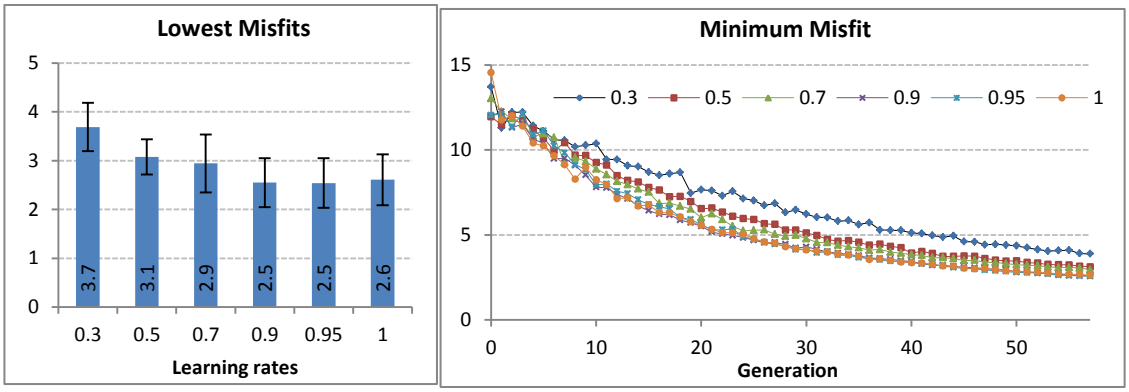


Figure 6.6: Averaged minimum misfit (left) and generational minimum misfit (right) for different values of the learning rate in SUGEDA. Although 0.95 yields slightly better misfit convergence, all higher values, 1.0, 0.95, and 0.9, give similar results.

6.3.2.2 Results of SMGEDA on PUNQ-S3

For SMGEDA, we tuned the number of parents in each generation and the sampling algorithm for the multivariate Gaussian distribution. Figure 6.7 shows the misfit convergence for three different values of the number of parents. As the figure shows, 150 parents per generation setup yields the best convergence. From this figure, it is evident that a setup of 50 parents per generation suffers from the loss of diversity and seems to become trapped in local optima.

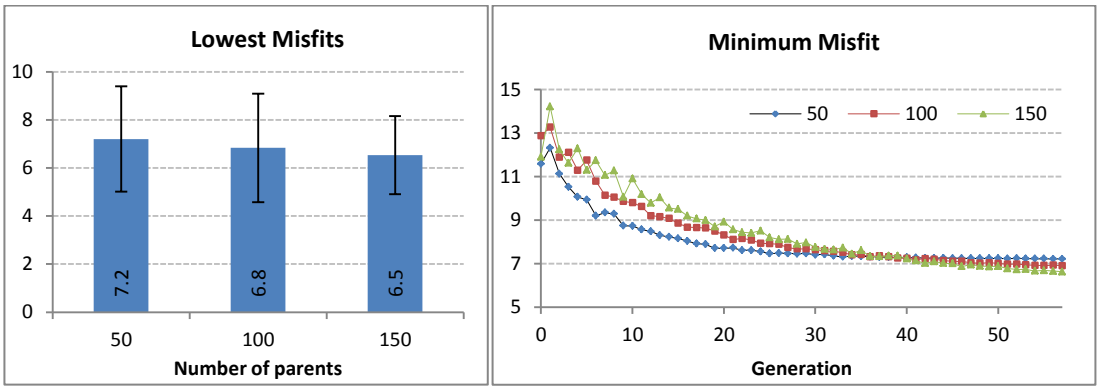


Figure 6.7: Averaged minimum misfit (left) and generational minimum misfit (right) for different numbers of parents in SMGEDA. 150 parents per generation gives the best misfit convergence for the algorithm. Where using 50 parents per generation, the algorithm seems to be trapped in local minima due to the loss of diversity.

We also tested two different decomposition methods for sampling from the multivariate Gaussian probability distribution: Cholesky and Eigendecomposition. Results obtained on PUNQ-S3 using both decomposition methods were very similar.

6.3.2.3 Results of MUGEDA on PUNQ-S3

We tested and tuned MUGEDA on PUNQ-S3 model for its two control parameters: the number of parents and number of clusters (mixture models). Figure 6.8 shows minimum misfit and misfit convergence plots of SMGEDA for three different values for the number of parents. As the figure shows, the 150 parents per generation setup is a slightly better option, in terms of minimum misfit and misfit convergence.

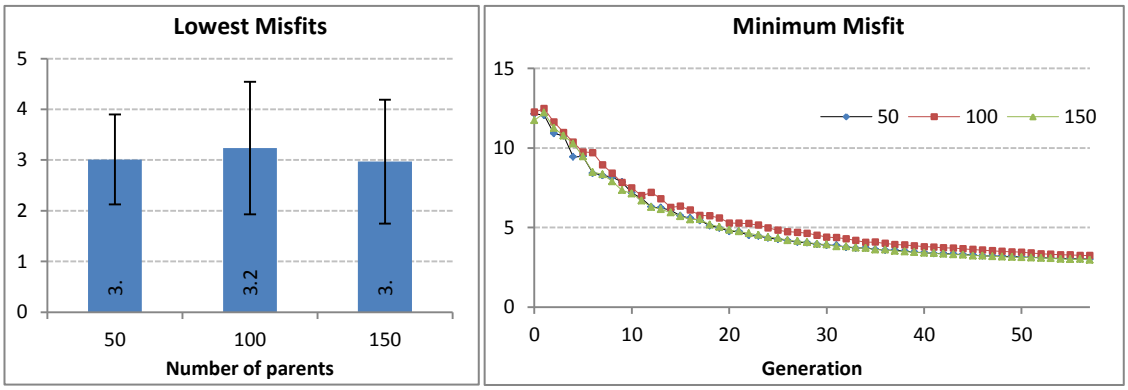


Figure 6.8: Averaged minimum misfit (left) and generational minimum misfit (right) for different numbers of parents in MUGEDA. 150 parents per generation gives slightly best misfit convergence for the algorithm, although the improvement is not significant.

In MUGEDA, first, one-dimensional probabilistic distance clustering of the parent solutions was performed. Then, the spread, mean and standard deviation of the clusters were obtained and used to sample new child solutions. We tested three values for the number of clusters (mixture models): 3, 5, and 10. The convergence result of 10 clusters per generation was also just marginally better than the other two values, and the improvement was not significant, as shown by the error bars.

6.3.2.4 Results of MMGEDA on PUNQ-S3 model

The last algorithm to be test and tuned on PUNQ-S3 model was MMGEDA. MMGEDA requires higher number of parents. Otherwise, clusters may get a small number of members and the covariance matrix calculations and multivariate sampling techniques

become unstable. Therefore, only parent size of 150 was considered.

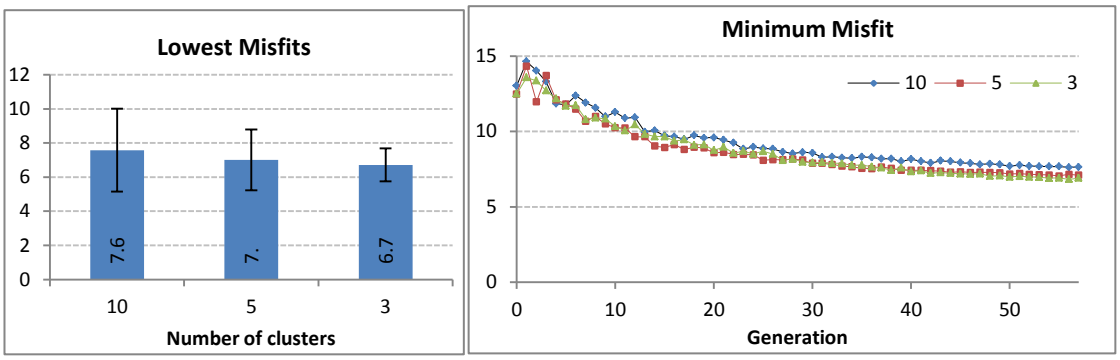


Figure 6.9 shows the minimum misfit and misfit convergence of MMGEDA for three different values for the number of clusters. From this figure, it is obvious that lower numbers of clusters are preferred.

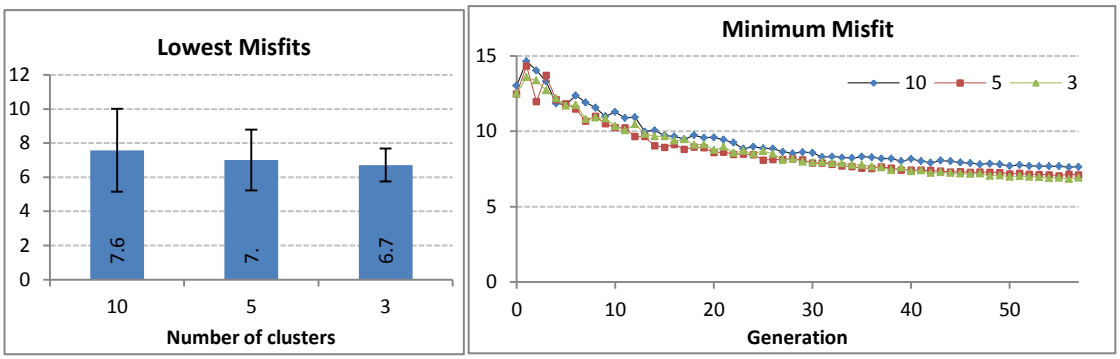


Figure 6.9: Averaged minimum misfit (left) and generational minimum misfit (right) for different numbers of clusters in MMGEDA. Here, 3 and 5 clusters of the parent solutions converge slightly better than 10 clusters.

6.3.2.5 Comparative study of Gaussian-based EDAs on PUNQ-S3

Having tuned the control parameters for all four Gaussian-based EDAs, in this section we present the comparative study of the algorithms on PUNQ-S3 model. Again, we used 3,000 function evaluations with 50 child solutions per generation for all three algorithms. For each algorithm, we took the best value of the tuned parameters. Figure 6.10 shows the minimum misfit and misfit convergence plots of the algorithms. As the figure shows, MUGEDA outperforms other three Gaussian-based algorithms.

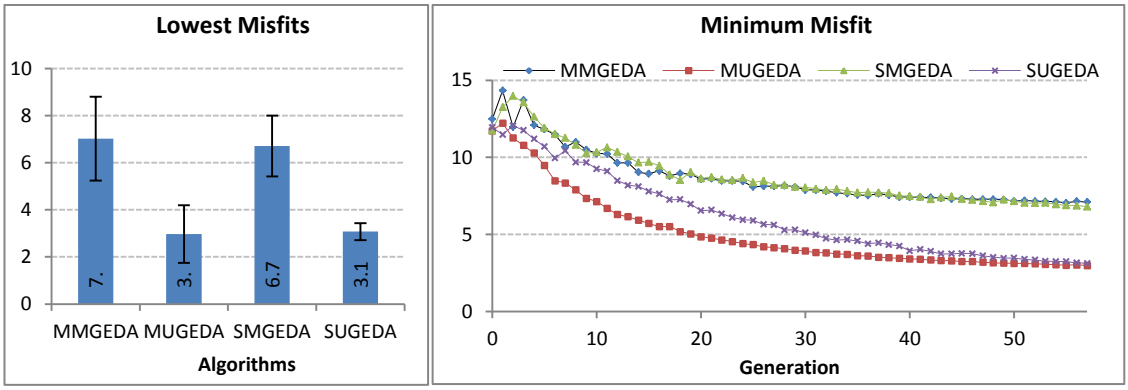


Figure 6.10: Comparison of four Gaussian-based EDAs for the minimum misfit (left) and generational minimum misfit (right) obtained by each algorithm; MUGEDA outperforms the other three algorithms.

6.4 Discussion

In this chapter, our goal was not to recommend a specific EDA that outperforms other EDAs in in an overall sense. Test function applications showed that it is possible to find problems which are most suitable to any given algorithm at hand and vice versa. The Gaussian distribution function can be used as a probability model in EDAs for unimodal continuous history matching problems or multimodal continuous problems, where a mixture of Gaussians is used. If the unimodel Gaussian-based EDA is preferred, independent component analysis can be used with univariate marginal EDA to tackle the interrelations among the variables (Zhang et al., 2000).

Gaussian-based EDAs can handle multivariate problems. Multi-variety can be introduced to these models by computing the full covariance matrix and using it for sampling new solutions. Multivariate models need to be traded-off with the considerable increase in computational expenses they often demand. Another important issue that must be taken into consideration is that due to the very high computational requirements of the function evaluations in high-dimensional history matching problems, it is usually not possible to take large population sizes. Thus, when using mixture models, one should estimate the models using a minimal number of solutions. The resulting Gaussian models, especially in the multivariate case, will not be robust enough, even when a covariance matrix repair is used. This might explain the relatively poor results of multivariate EDAs on PUNQ-S3 (Figure 6.10), along with the fact that in the PUNQ-S3 case, there is not strong dependency between search variables.

Taking multiple Gaussian models for PUNQ-S3 improves diversity and makes the algorithm more explorative. Thus, it accelerates the convergence in the early stages of the search (Figure 6.10). However, due to the strongly elitist nature of the EDAs introduced in this chapter, this improvement is weakened in later stages, whereas a stronger pressure is put on the regions with the global optimum. One must be aware of possible overfitting when a large number of mixture models is used.

6.5 References

Ben-Israel, A. (2006). Probabilistic Distance Clustering. DIMACS Technical Report.

Bosman, P. A., & Thierens, D. (2000). Mixed IDEAs. Utrecht University Technical Report UU-CS-2000-45, Utrecht University, Utrecht, Netherlands.

Bosman, P. A., & Thierens, D. (2001). Advancing continuous IDEAs with mixture distributions and factorization selection metrics. In Proceedings of the Optimisation by Building and Using Probabilistic Models. OBUPM Workshop at the Genetic and Evolutionary Computation Conference (GECCO), 208–212. San Francisco, CA, USA.

Chen, C., Liu, W., & Chen, Y. (2006). Adaptive discretization for probabilistic model building genetic algorithms. In Proceedings of the 8th annual conference on Genetic and evolutionary computation (GECCO), 1103–1110. New York, NY, USA.

De Jong, K.A. (1975). An analysis of the behaviour of a class of genetic adaptive systems. PhD thesis, University of Michigan. Michigan, USA.

Dong, W., and Yao, X. (2007). Covariance Matrix Repairing in Gaussian Based EDAs. IEEE Congress on Evolutionary Computation (CEC). Singapore.

Floris, F., Bush, M., Cuypers, M., Roggero, F., & Syversveen, A.R. (2001). Methods for Quantifying the Uncertainty of Production Forecasts. Petroleum Geoscience , 7, 87-96.

Gallagher, M., Frean, M., Downs, T. (1999). Real-valued Evolutionary Optimisation using a Flexible Probability Density Estimator. Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), 840-846. Orlando, Florida, USA.

Griewank, A.O. (1981). Generalized descent for global optimisation, Journal of Optimisation Theory and Applications, 34, 11-39.

Iyigun, C., and Ben-Israel, A. (2008). Probabilistic Distance Clustering Adjusted for Cluster Size. Probability in the Engineering and Informational Sciences 22, 603-621.

Larrañaga, P., & Lozano, J. A. (Eds.) (2002). Estimation of distribution algorithms: A new tool for evolutionary computation. Berlin, Germany: Springer-Verlag.

Larrañaga, P., Etxeberria, R., Lozano, J. A. & Pena, J.M. (1999). Optimisation by Learning and Simulation of Bayesian and Gaussian Networks. Technical Report KZZA-IK-4-99, Department of Computer Science and Artificial Intelligence, University of the Basque

Country.

Larrañaga, P., Etxeberria, R., Lozano, J. A. & Pena, J.M. (2000). Optimisation in continuous domains by learning and simulation of Gaussian networks. Workshop Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), 201–204. Las Vegas, Nevada, USA.

Mühlenbein, H. & Paaß, G. (1996). From recombination of genes to the estimation of distributions I. binary parameters. In Parallel Problem Solving from Nature - PPSN IV. Lecture Notes in Computer Science, 1411, 178–187. Springer.

Mühlenbein, H. (1997). The equation for response to selection and its use for prediction. In Evolutionary Computation, 5(3), 303–346.

Pelikan, M., Goldberg, D. E., & Lobo, F.G. (2000). A Survey of Optimisation by Building and Using Probabilistic Models. Computational Optimisation and Applications, 21: 5–20.

PUNQ-S3 Test Case. (2010). Retrieved 7 1, 2010, from Department of Earth Science and Engineering Department Website - Imperial College: http://www3.imperial.ac.uk/earthscienceandengineering/research/perm/punq-s3model

Rudlof, S., & Koppen, M. (1996). Stochastic Hill Climbing with Learning by Vectors of Normal Distributions. In T. Furuhashi, Proceedings of the First Online Workshop on Soft Computing (60-70).

Törn, A., Zilinskas, A., (1989). Global Optimisation. Lecture Notes in Computer Science, Nº 350. Berlin, Germany: Springer-Verlag.

Zhang, Q., Allinson, N.M., & Yin, H. (2000). Population Optimization Algorithm Based on ICA. In proceeding of IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks, 33-36. San Antonio, TX, USA.

CHAPTER 7:

DIVERSITY-BASED ADAPTIVE EVOLUTIONARY ALGORITHMS FOR HISTORY MATCHING

"We must alter theory to adapt it to nature, but not nature to adapt it to theory."

Claude Bernard (1865 - 1927)

7.1 Introduction

The subject of performance assessment of the population-based evolutionary algorithms (PBEAs) has already developed into a new research topic in the Evolutionary Computation (EC) community, but not yet in history matching and uncertainty quantification.

The notion of performance assessment involves two aspects: the quality of the solution and the time spent to find such a solution. The basic approach to deal with performance assessment of a PBEA in the EC community is to use quantitative performance measures and statistical analysis of these measures. The choice and the design of the appropriate performance measures are challenging topics in EC (e.g. Lacevic and Amaldi, 2010).

Like any other optimisation algorithm, an EA is controlled by certain parameters which determine the performance of algorithm. For implementing an EA algorithm, it is necessary to specify how to represent the solution (reservoir model), how to represent the environment (misfit), how to select solutions fittest to the environment (selection),

how selected solutions (parents) will be changed to generate new solutions (children) and the condition in which the optimisation course will be terminated (stopping criteria). The value of these control parameters mainly determines the quality and the efficiency of the search (Eiben et al, 1999).

As we saw in the previous chapters, choosing suitable parameters is essential for the performance of the algorithm. The effectiveness, efficiency, and robustness of an algorithm are determined by the parameter choice, which is generally problem-dependent and requires extensive user experience. Even an experienced user may not be able to determine optimal parameter values which would work well on different problems. One can set up these parameters for optimum performance of the algorithm in two ways: parameter tuning and parameter adaptation. Tuning is a process, in which, the user tries to find good values of the parameters, based on a common practice approach, before running the algorithm. Then the algorithm is tuned using these values, which remain fixed during the evolution.

Parameter adaptation in evolutionary algorithms aims to tune and optimise the control parameters of the algorithms over the course of evolution. Eiben et al. (1999) classify parameter adaptations into three types: deterministic, in which parameters are changed according to a deterministic function, adaptive, in which parameters changed with feedback from the algorithm and self-adaptive, in which the adaptation method is encoded into the problem representation and treated as a problem parameter.

There are many adaptive EAs introduced by the computer science community and tested on standard test functions. For a survey of diversity-based evolutionary algorithms refer to Gouvêa & Araújo (2010). Santana et al. (2008) presented a general framework for introducing adaptation in EDAs with the possibility of changing the probabilistic model during the evolution. Liu et al. (2007) introduced an entropy-driven parameter estimation mechanism for EAs. Rasmus (2002) introduced a diversity-guided evolutionary algorithm (DGEA) which uses well-known distance-to-average-point measure to alternate between phases of mutation and phases of recombination and selection in a genetic algorithm.

In last two decades, EAs have been extensively applied to history matching, but there is still a lack of knowledge on how to find reasonably good parameter values for these algorithms.

In all reported history matching applications in which, although parameter tuning is used to set the control parameters, one cannot find a single set of control parameters that are suitable for different history matching problems. Furthermore, manual parameter tuning is time consuming, and since EAs are stochastic and usually run until a predefined stopping criterion, finding the optimum value of the control parameters is not guaranteed using the tuning.

In this chapter, first we discuss the population diversity and different measures for it. Then, we present a comparative study of these measures. Finally we use one of the measures for the control parameter estimation of an evolutionary algorithm. For this purpose, we use a diversity-based adaptation mechanism for histogram-based estimation of distribution algorithm (HEDA).

7.2 Diversity Measures

Population diversity is a key measurement in population-based evolutionary algorithms such as genetic algorithm (GA) and estimation of distribution algorithm (EDA). Diversity is an estimate of the levels and types of variety of individuals in a population. Maintaining ahigh enough population diversity is essential to avoid premature convergence and achieve a better match quality (Burke et al., 2004). Often, the reason behind premature convergence is that the individuals in the gene pool are too alike, so the algorithm has not been able to extract new information, which can lead the search toward to a local optimum and prevent it from reaching the global optimum.

Several methods have been introduced and used for estimating diversity measures. Wineberg and Oppacher (2003) showed that determining the distance between all possible pairs of genes/organism in the population is the basic method that underlies all of the diversity measures. The distance is calculated either in genotypic space or phenotypic space. Genotypic measures estimate the variation in the variable genomes, while phenotypic measures estimate variance in the fitness.

In real-coded optimisation problems, genotypic measures are more common. Genotypic diversity measures are either genome-based or gene-based. Genome-based measures treat the genome as a whole and usually estimate a distance using deterministic formulae to differentiate between two genomes. Gene-based measures evaluate one gene at a time and calculate the average of these values. The sum of the Hamming distance between all possible pairs of the population is a widely used genotypic diversity measure for binary variables (Horn, 1997). For real-valued variables, Euclidean distance (Lacevic and Amaldi, 2010) and distance to population centroid (Morrison and De Jong, 2002) are widely used.

In EAs, identical genotypes produce the same fitness, and hence the genotype diversity is closely connected to the phenotype diversity. Phenotypic measures are preferred over genotypic measures for real-coded parameter optimisation problems because, in genotypic measures, all bitwise diversity is treated the same, but in fact, different bit positions can result in different phenotypic diversity. A popular phenotypic measure is the use of Shannon Entropy on fitness frequencies (Rosca 1995).

The petroleum engineering community still widely uses visual and qualitative approaches for performance assessment of PBEAs in history matching and uncertainty quantification. Convergence time to a specific minimum misfit or minimum misfit after a certain number of function evaluations are common approaches for performance comparison studies of PBEAs in petroleum literature. Examining diversity just by looking at the population plots can lead to misinterpretations due to the possible unreliability of the human eye, and one must appreciate mathematical techniques for diversity/convergence interpretations.

Diversity measures are defined and designed based on the concepts of exploration and exploitation. The goal in any search algorithm is to achieve a balance between the exploration and exploitation properties of the algorithm. A good search algorithm should be able to combine both tasks. Exploitation is required to ensure refinement of the current solution and finding the global optimum. Exploration is required because local minima are usually present, and every part of the parameter search space should be searched thoroughly to avoid becoming trapped in these local minima.

In this chapter, four diversity measures are used for real-coded PBEAs. These include two genotypic distance-based measures, the Pair-wise Hamming distance-based and moment of inertia-based diversity measures, and two phenotypic measures, the proportional entropy-based and crowding entropy-based measures. These measures are used to show the performance of different PBEAs when used for solving the minimisation problem of a test function and the history matching problem of a synthetic model, PUNQ-S3, and a real North Sea reservoir model, the Koma field. We visualize and analyse the convergence and diversity of applied algorithms. We aim the diversity measures to enable us, firstly, to conduct performance comparison of different PBEAs, and secondly, to control the performance of a PBEA by getting feedback about its behaviour and to use this information to improve the search in the next generations of the algorithm.

7.2.1 Fitness-based Diversity Measures

A common method for visualising the performance of the evolutionary algorithm is to plot fitness value as an indicator of the algorithm's convergence throughout the evolution. In population-based evolutionary algorithms one can monitor statistical measures of the misfit, i.e. mean and standard deviation of misfit value of the solutions in the generation or box-plot, e.g. minimum, P10, P50, P90, and maximum values.

7.2.2 Distance-based Diversity Measures

The diversity concept is closely related to the concept of distance between individuals in the population. Two widely used distance functions for diversity measures are: Hamming distance and Euclidean distance.

7.2.2.1 Pairwise Hamming Distance Measure (HDM)

Hamming distance was originally used by Hamming (1950) for detection and correction of errors in digital communication. Pairwise Hamming distance is one of the first measures introduced for the population diversity. It is obtained by summing all pairwise distances between individuals (Horn, 1997). Pairwise Hamming distance for a population is defined as:

$$HDM = \sum_{j=1}^{P-1} \sum_{k=j+1}^{P} \left(\sum_{i=1}^{N} \left| x_{ij} - x_{ik} \right| \right) \tag{7.1}$$

where:

- *HDM* is Pair-wise Hamming distance,
- *i* is the subscript for the variables number in solution *x*,
- *N* is the total number of variables in solution *x*,
- *j* and *k* are the subscripts for the solution number in the population,
- *P* is the total number of solutions in the population,
- *x* is the variable value.

When using HDM, it should be noticed that all variable ranges, x_i, should be normalised to a number between 0 and 1. Another important issue with HDM is its relatively high computational demands; the computation of the measure is quadratic with the size of the population and it needs *P(P-1)/2* calculations.

7.2.2.2 Inertia based Diversity Measure (IDM)

The concept of the moment of inertia is a measure of an object's resistance to changes to its rotation, which first was introduced by Leonhard Euler in 1765, in classical mechanics. Morrison and Jong (2002) used this concept for measurement of population diversity. For this measure, first, the centroid of the population is calculated for all equally weighted solutions in the population; then the moment of inertia based diversity measure of these solutions about the calculated centroid is given by:

$$\begin{cases} c_j = \dfrac{\sum_{i=1}^{N} x_{ij}}{N} \\ IDM = \dfrac{\sum_{j=1}^{P} \sum_{i=1}^{N} (x_{ij} - c_j)^2}{PN} \end{cases} \tag{7.2}$$

where IDM is the inertia based diversity measure, *i* is the subscript for the number of variables number in solution *x*, *N* is the total number of variables in solution *x*, *x* is the variable value, c_i is the centroids of the variable values in the solutions, *j* is the subscript

for the solution number in the population, P is the total number of solutions in the population.

Like HDM, IDM needs equally weighted solution parameters and one must normalise all the parameters to a real number between 0 and 1. IDM has less computational requirements than HDM as the total computational time for the centroid and inertia-base measure calculations is $4(NP) + N$ calculations.

7.2.3 Entropy-based Diversity Measures

Entropy is a term to measure the amount of disorder of a system. Shannon entropy is a concept from information theory, used to quantify the uncertainty associated with a random variable. Shannon entropy of a random variable x with probability mass function $p(x)$ can be used to measure the average information content that is missing when the value of x is unknown (Shannon, 1948). Shannon entropy for discrete variables is defined as:

$$H(x) = -\sum_{i=1}^{n} p_i . \ln(p_i) \tag{7.3}$$

where $H(X)$ is Shannon entropy, p_i is the probability of observing value i for variable x, and we always have $\sum p_i = 1$, n is the number of bins for discrete variable x, ln is the natural logarithm, though logarithms in base 2 and 10 are also sometimes used.

With this definition, a fair coin has entropy of one while a biased coin has an entropy lower than one. This is due to a higher probability of returning one side which makes prediction easier and reduces uncertainty. In a similar concept, an unknown variable with a large variety of different values will have higher Shannon entropy, compared to another unknown variable with fewer values. In this context, Shannon entropy can be used as a measure of diversity: the greater the Shannon entropy, the higher the diversity.

7.2.3.1 Proportional Entropy-based Measure (PEM)

Rosca (1995) used Shannon entropy as a population diversity measure by first placing fitness values in the population into fitness classes. Thus in the Shannon entropy

equation (7.3), n is the number of fitness classes, and p_i is the ratio of number of fitness values in class i to total number of the fitness values in the population (P).

Several methods have been introduced in the literature to determine different classes of the fitness. Liu et al. (2007) used a linear method to split the fitness values by dividing the range between the best and the worst fitness into n predefined number of classes. They also used Gaussian distribution to spread out the fitness class using the average and the standard deviation of the fitness values in the population. In the Gaussian method, the lower and upper bounds of the classes are obtained from *average+/-i*standard deviation* where i is an integer number equal to or less than the number of fitness values in the population.

Liu et al. (2007) also introduced a proportional method, which does not require linear or Gaussian characteristics of the fitness function. In the current work, we used the proportional method to define the number of classes, so that fitness classes represent unique fitness values in the current population. If all solutions in the population have different fitness value, e.g. in a history matching problem, the number of fitness classes, n, is equal to the population size, P. Probability values in the Shannon entropy equation (7.3), p_i, are calculated as:

$$p_i = \frac{f_i}{\sum_{i=1}^{n} f_i} \tag{7.4}$$

where i is the class number, f_i is the fitness value of the solution and n is the total number of classes. For two or more solutions with the same fitness value, only one solution is kept as the fitness class and others are eliminated, as they do not contribute to the population's diversity.

7.2.3.2 Crowding Entropy-based Measure (CEM)

Wang et al. (2010) proposed another entropy-based measure based on the concept of the crowding distance (Deb, 2001). The crowding distance is an estimation of the density of solutions obtained by calculating the average distance of two solutions on either side of a solution along the fitness function. The crowding distance is widely used for ranking of the solutions in multiobjective algorithms by estimating density along each of the

objectives in the objective function space. The crowding entropy of the population is defined as:

$$CEM = \frac{1}{f_{max} - f_{min}} \sum_{j=1}^{P} \left[u_j . log_2 \left(\frac{u_j}{u_j + l_j} \right) + l_i . log_2 \left(\frac{l_j}{u_j + l_j} \right) \right] \quad (7.5)$$

where *CE* is the crowding entropy, f_{max} is the maximum value of the fitness function in the population, f_{min} is the minimum value of the fitness function in the population, j is the subscript for the solution number in the population, P is the total number of solutions in the population, u_j is the distance of solution j from its upper adjacent solution, and l_j is the distance of solution j from its lower adjacent solution.

To have equally weighted fitness values, each fitness value should be normalized before calculating the crowding entropy.

7.2.4 Applications

We applied four diversity measures to the optimisation problem of the Rosenbrock test function, the history matching problem of the PUNQ-S3 synthetic case, and the history matching of Koma field.

7.2.4.1 Rosenbrock function results

For the Rosenbrock test function, we used a GA with 5,000 function evaluations in 48 generations and 200 solutions in an initial random population. In each generation, the best 100 solutions with the minimum fitness value were selected and a simulated binary crossover (SBX) was applied to the 50 random pairs of selected solutions, resulting in 100 new child solutions. A simulated binary mutation operator was applied with a probability of 0.1 to all the generated solutions. The result was 100 new solutions in each generation. The algorithm stopped after 48 generations. The results for the different diversity measures were compared to the convergence of the algorithm for the minimum fitness value, the mean and standard deviation of the fitness values, and 25, 50 and 75 quartiles.

The results show that all four measures reflect the diversity/convergence in the search procedure (Figure 7.1). IDM from the distance based measures and CEM from the

entropy-based measures show similar convergence to the convergence of fitness values (Figure 7.2).

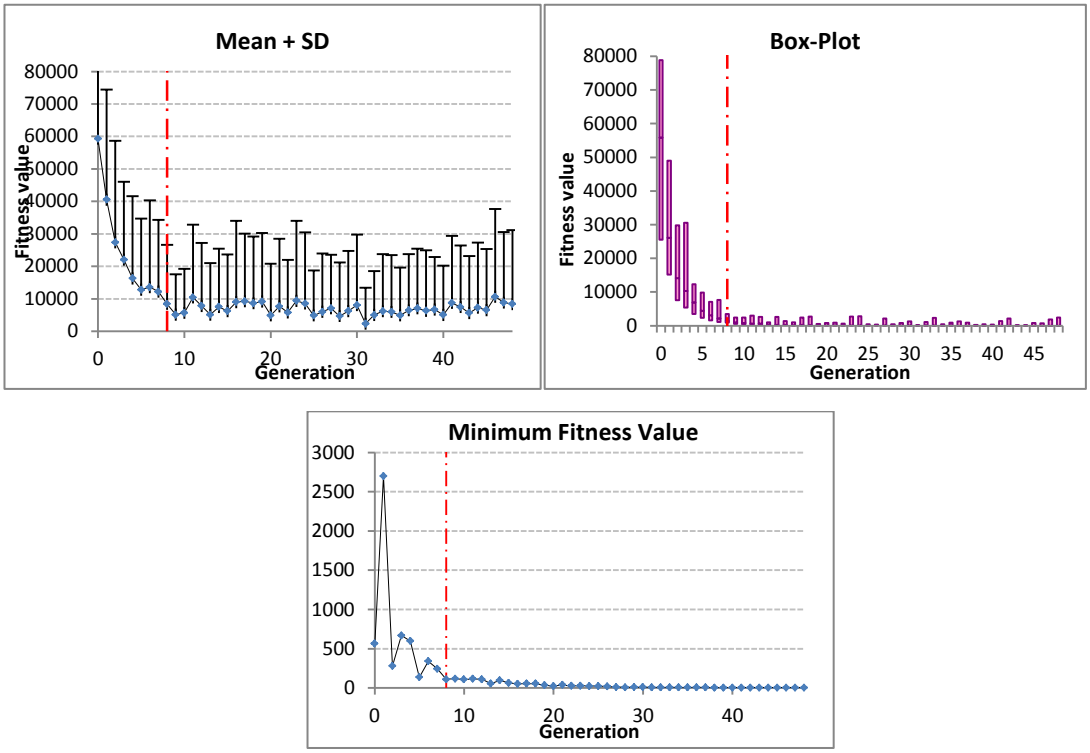


Figure 7.1: Results of the fitness convergence for the Rosenbrock function application. The mean and standard deviation (top-left), box-plot of 25, 50, and 75 quartiles (top-right), and minimum fitness value (bottom) all show the convergence of the GA algorithm in generations 8 to 9.

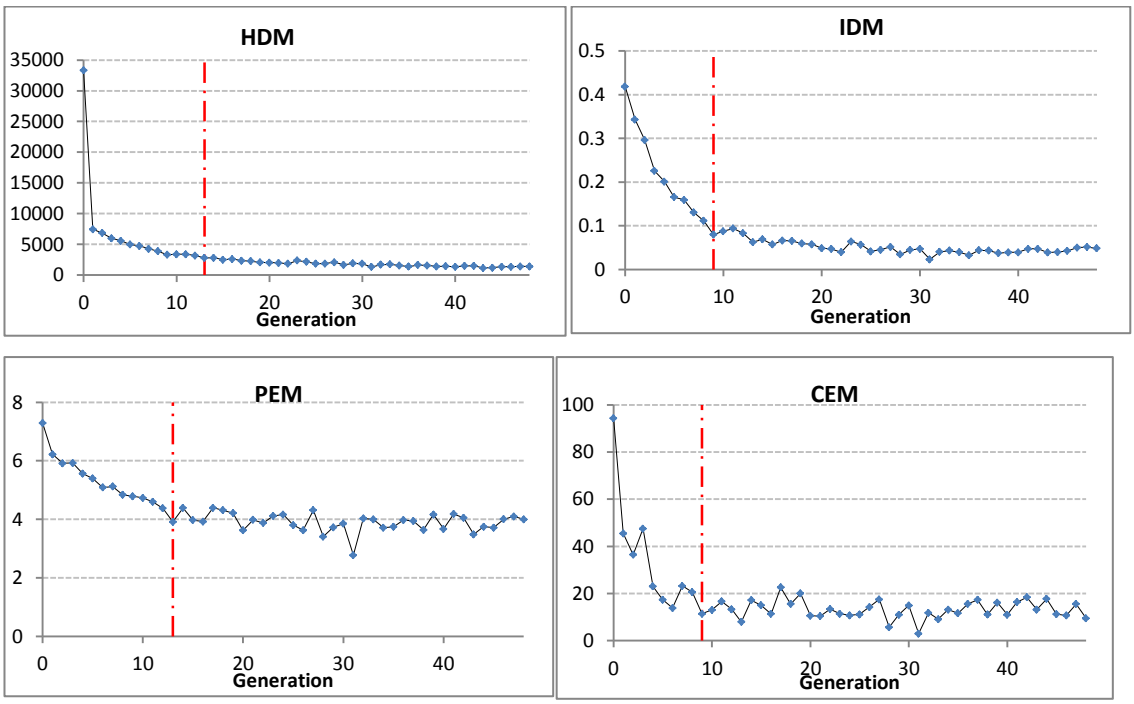


Figure 7.2: Results of the four diversity measures for the Rosenbrock function application; HDM (top-left), IDM (top-right), PEM (bottom-left), and CEM (bottom-right) all show loss of the diversity throughout the evolution. The convergence estimated by HDM and PEM is at generation 13 while by IDM and CEM is at generation 9.

7.2.4.2 PUNQ-S3 model Results

We applied all four previously described diversity measures for performance assessment of BHEDA, BOA, Particle Swarm Optimisation (PSO), and Hybrid Bayesian and Particle Swarm Optimisation Algorithm (BOA/PSO) to Reynolds et al. (2011) algorithms when used in history matching of PUNQ-S3 model (Boss 1999). All of these algorithms were tuned for their optimal control parameters discussed in the previous studies. We repeated each tuned algorithm 10 times and averaged for the misfit and diversity measures to minimize the effects of randomness. For more details on BHEDA and BOA, refer to Chapter 3 and on PSO and BOA/PSO. We used 3,000 function evaluations (simulation runs) in an initial generation of 150 solutions and 57 generations of each 50 new solutions.

Figure 7.3 shows the box-plot and the convergence points of the misfit for different algorithms. Unlike BOA and HEDA, the initial generation in PSO and hybrid BOA/PSO is not completely random. A random swarm of 25 solutions is used to generate another 125 solutions to complete the initial population, so that the box-plot of misfit is narrower in the first generation compared to BOA and HEDA.

Figure 7.4 shows minimum misfit and mean misfit obtained by all four algorithms. The results for the diversity measures are given in Figure 7.5. As seen in Figure 7.5, hybrid BOA/PSO has converged at generation two, while all of the four measures show a flat curve above zero for the distance based measures and a flat curve at the same level as the other algorithms. This indicates a good characteristic for a PBEA, converging while maintaining the diversity. For the PSO box-plot, convergence is seen at generation 6, while HDM shows almost zero diversity and IDM drops to zero after the convergence stage.

A similar response is shown by PEM for PSO, indicated by a lower level of entropy. CEM exhibits a sharp growth in the later stages of the evolution. All of these are indications of a strong loss of diversity and a premature convergence, which can also be seen from the minimum misfit chart. BOA and BHEDA have converged gently, with a gradual decline in distance based measures. They have maintained diversity until very late stage of the search, where an increase in the entropy based measures can be seen.

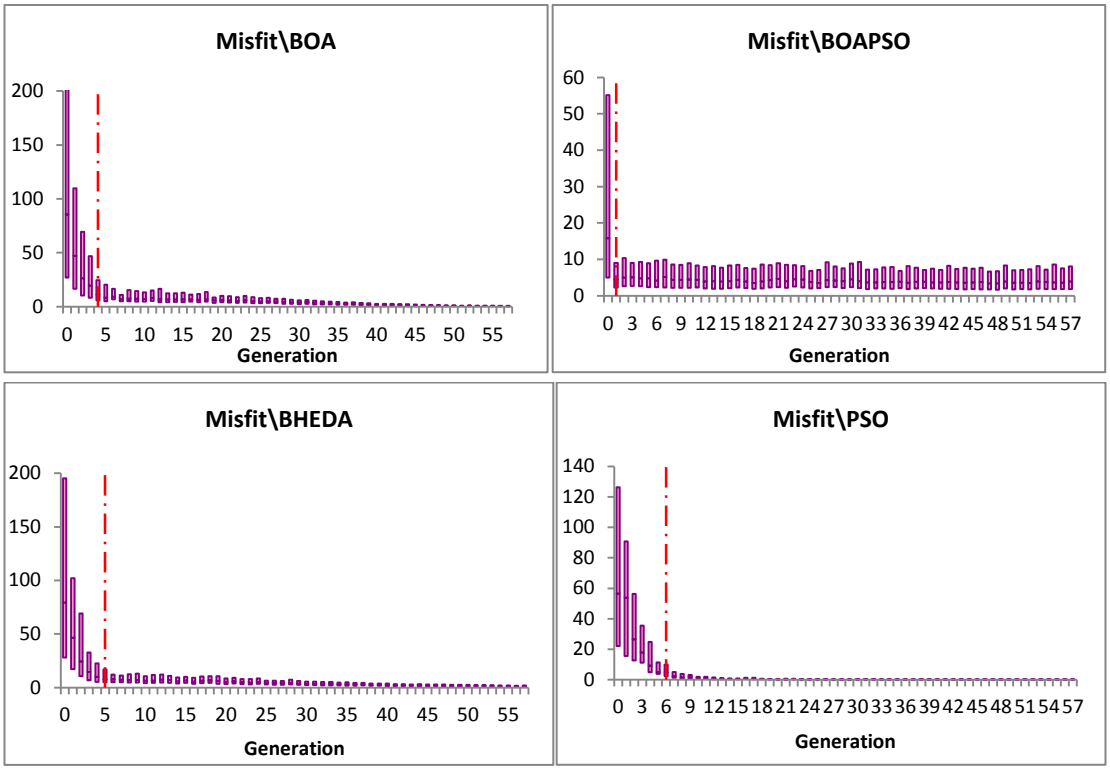


Figure 7.3: Box-plot (25, 50, and 75 quartiles) of the misfit convergence for the PUNQ-S3 achieved by BOA (top-left), hybrid BOA/PSO (top-right), BHEDA (bottom-left), and PSO (bottom-right). In PSO and BOA/PSO initial population is not completely random as a random swarm of 25 solutions is used to generate another 125 solutions to complete the initial population. For BOA convergence is at generation 4, for BOA/PSO at generation 1, for PSO at generation 6, and for BHEDA at generation 5.

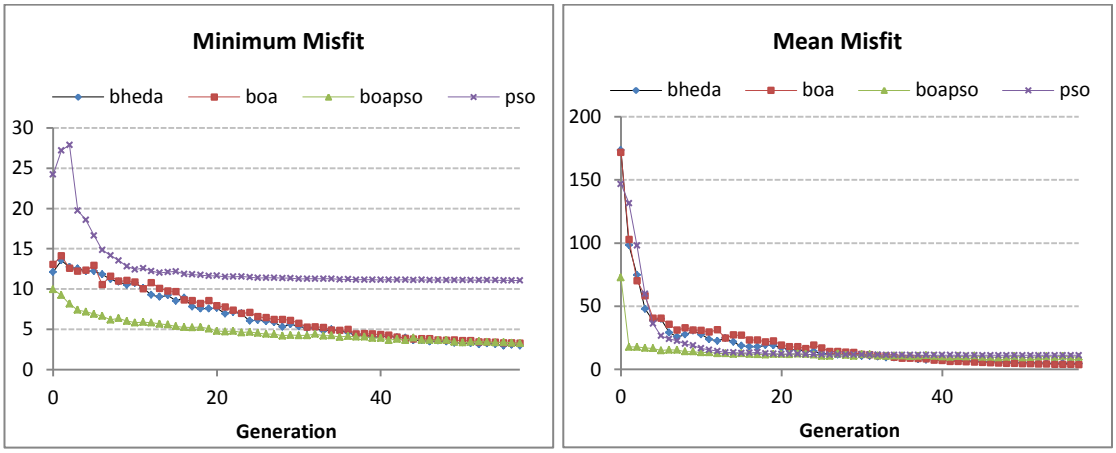


Figure 7.4: The minimum misfit (left) and the mean misfit (right) charts for the PUNQ-S3 achieved by different algorithms. From generation 10, PSO seems to be trapped in local minima.

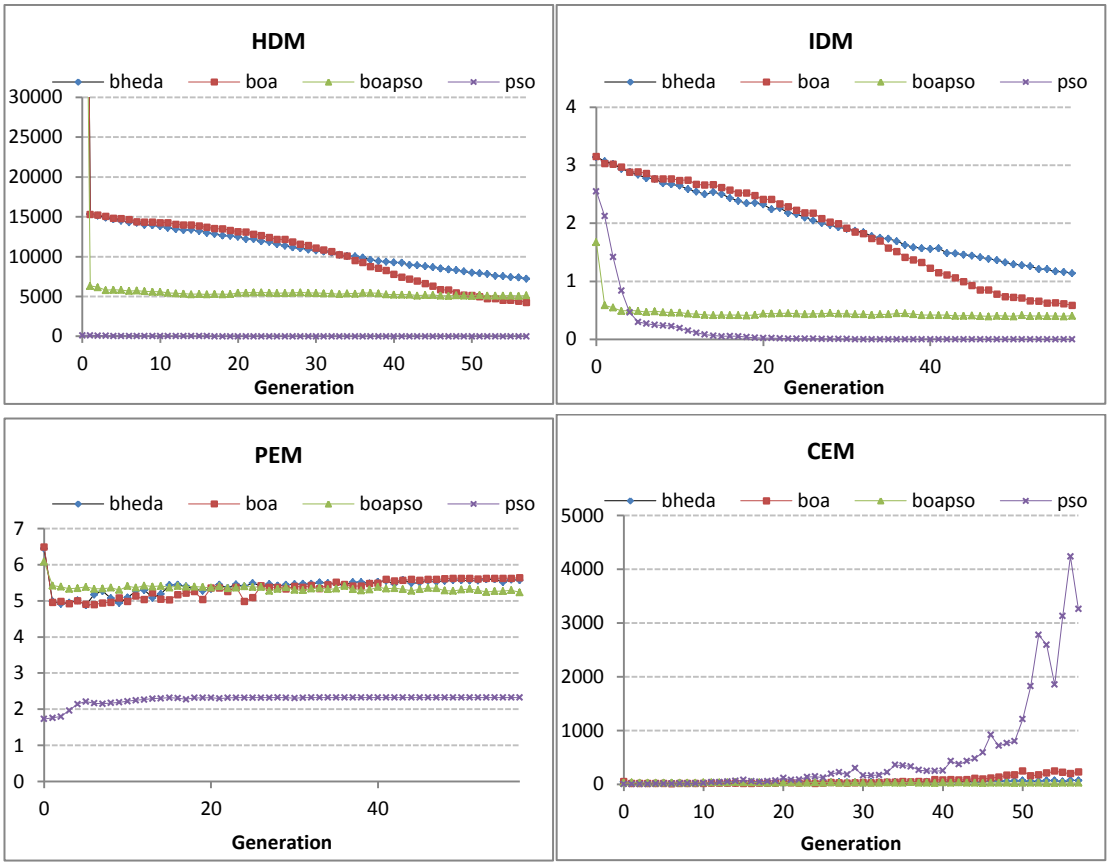


Figure 7.5: Diversity measures obtained for the PUNQ-S3 by different algorithms; HDM (top-left), IDM (top-right), PEM (bottom-left), and CEM (bottom-right) all show loss of diversity and convergence, but at different generations. From generation 13, PSO suffers from extreme loss of diversity and appears to be trapped in local minima.

7.2.4.3 Koma field Results

We applied BOA and PSO to history matching of the Koma and recorded the results for all four previously described diversity measures. All the algorithms were tuned for the optimal control parameters and repeated 5 times, and then averaged for misfit and diversity measures. For more details of the PSO algorithm used in this application, see Reynolds et al. (2011).

We used 1,000 simulation runs for the two algorithms. PSO was set up with swarm size of 50 and 20 generations. For BOA, we used an initial population of 150 random solutions. Then in each generation, 100 solutions were selected to generate 50 new solutions from the Bayesian network model. Maximum 6 edges were allowed for the network and parameter values were discretised using 8 bins.

Figure 7.6 illustrates the misfit convergence for two algorithms, BOA and PSO, when applied to the Koma field. Both algorithms have converged to lower values of the

misfit, but we can see that, overall, PSO outperforms BOA in terms of minimum and mean misfit. Figure 7.7 shows results for different diversity measures. PSO achieved lower distance base measures than BOA, but similar a entropy-based measure. Combined with better misfit convergence, one can conclude that PSO has achieved a better balance between exploration and exploitation.

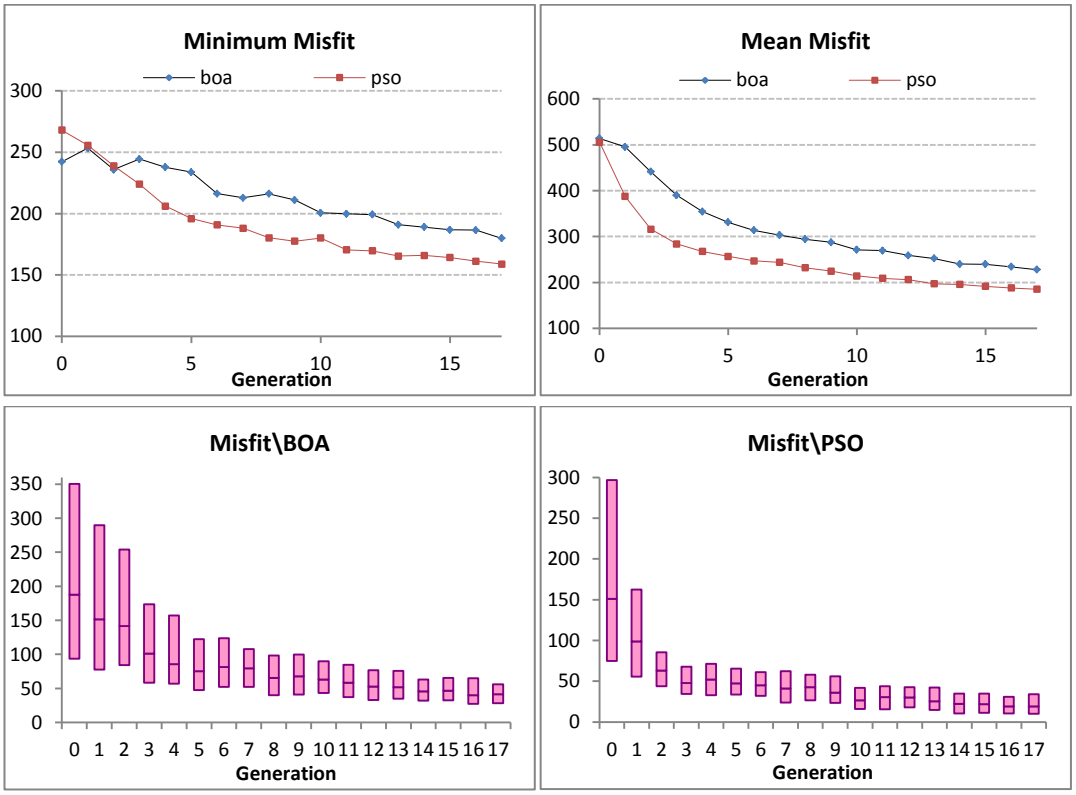


Figure 7.6: The minimum misfit (top-left), the mean misfit (top-right), box-plot of the misfit convergence for the Koma field achieved by BOA (bottom-left) and hybrid PSO (bottom-right). PSO has better convergence than BOA.

7.3 Diversity-based Adaptive EDA

A good search algorithm should be able to achieve and maintain the balance between the exploration and exploitation properties of the search. Exploration is required since local minima are usually available in the parameter space and every region of the search space should be searched aggressively to avoid becoming trapped in possible local minima. Having explored all regions, the algorithm is required to perform exploitation to ensure refinement of the current solution and finding the global optimum.

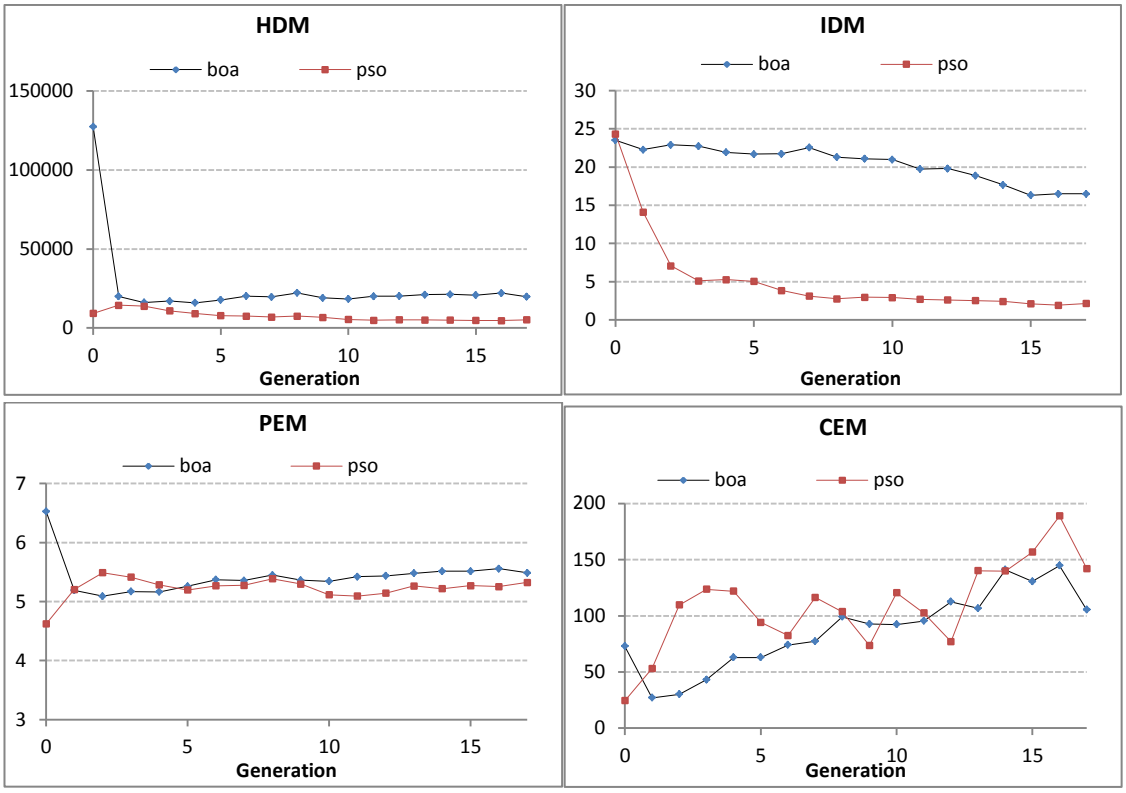


Figure 7.7: HDM (top-left), IDM (top-right), PEM (bottom-left) and CEM (bottom-right) diversity measures of Koma field history matching. PSO has a lower amount of the distance-based measure than BOA. In Koma field, PSO yields a similar entropy-based measure to BOA.

The balance between exploration and exploitation in each generation of the search is reflected in population diversity. Loss of diversity shows an exploitative search that may lead the algorithm to a non-optimal result, that is the algorithm is converged to a local minimum. Moreover, extreme diversity indicates an overly explorative search, in which the convergence is insufficient. To keep the balance, it is crucial to monitor the diversity and keep it at acceptable levels. As discussed in the previous section, there are various measures that can be used to quantify diversity. We take the inertia-based diversity measure (IDM) to monitor the diversity and keep it at an acceptable level in an EDA search. The result is an adaptive EDA which adapts its search mechanism to balance convergence and diversity.

For IDM parameter values of the solutions must be equally weighted first, then the centroid of the population is calculated; finally, the moment of inertia-based diversity measure is calculated, see equation (7.2).

where IDM is the inertia-based diversity measure, j is the subscript for the normalised variables number in solution x, N is the total number of variables in solution, x is the

variable value, c_j is the centroid of the variable, i is the subscript for the number of solutions in the population, P is the total number of solutions in the population.

In the current work, unlike the original definition by Morrison & Jong (2002), we weighted IDM with the number of variables and the number of solutions,. This is to obtain a measure that is independent of the number of solutions in the population or the number of variables in the solution. Figure 7.8 shows IDM for an ideal case, where four solutions are spread evenly across the search space.

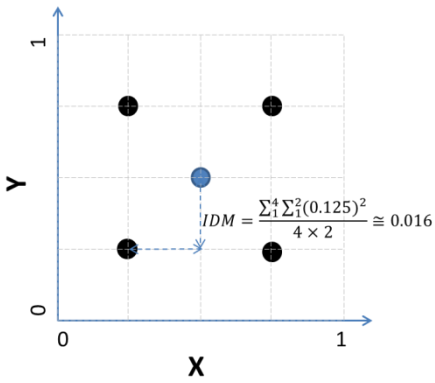


Figure 7.8: IDM for a population of 4 solutions (black dots, centroid is the blue dot) in a problem with two search variables (X and Y). Variable ranges are continuous and normalised [0,1), and hence IDM, when weighted for the number of solutions (4) and the number of variables (2), becomes 0.016.

Taking IDM for the population diversity has advantages and some disadvantages, but usually, its advantages outweigh its disadvantages. As mentioned earlier, solution parameters need to be equally weighted, and one must normalise all the parameters to a real number between 0 and 1. Moreover, IDM has difficulty with specific problems in which the search converges to multiple, separate regions in the parameter space, for which, actual convergence may not be seen by only looking at IDM. As discussed earlier, IDM calculation requires less computational expense than HDM.

7.3.1 Parameter adaptation

Ursem (2002) used population diversity to guide the search with a genetic algorithm by alternating between phases of mutation and phases of recombination and selection. Here, we use a similar concept for parameter adaptation in HEDA; we use the IDM diversity measure to alternate between the exploring and exploiting behaviour of the algorithm.

The exploration and exploitation behaviours are set by choosing an appropriate number of parents in each generation. In the exploration mode, to ensure a global search, the parent population size is set equal to a number considerably larger than the number of children (here, the size of the initial population) while in the exploitation mode, it is set equal to a much smaller number (here, the number of children), which ensures better convergence.

We call this adaptive AHEDA, which is indeed an HEDA with diversity-based parameter adaptation. In AHEDA, the search is started in *exploit* mode, and then in each generation, if IDM is less than a threshold value (D_l), diversity is increased by switching the search mode to *explore* so that the algorithm can escape from possible entrapment. Thereafter, whenever IDM is greater than another threshold value (D_h), the search is switched back to *exploit* mode to accelerate the convergence. This parent size adaptation is continued until the stopping criterion is met. The pseudo-code for AHEDA is as follows:

1. Generate and evaluate initial (t = 0) random population P_0, and set the search mode to *exploit*.
2. Set the low and high threshold diversity values, respectively D_l and D_h.
3. Select parents from the population, P_t.
4. Build the histogram model and use it to generate the children.
5. Evaluate the children and update the population with them.
6. If *IDM* of the children is less than D_l , set the search mode as *explore*,
7. Else, if *IDM* is greater than D_h, set the search mode to *exploit*.
8. Continue from step 3 (t = t +1) until the termination condition is met.

Threshold values for IDM (D_l and D_h) are respectively chosen equal to 10% and 50% of the IDM value in the initial random population. The initial random population is very diverse and has high IDM. The algorithm starts with the *exploit* mode, in which the number of parent solutions (selected in the selection and survival steps) is equal to the number of child solutions, generated using the histogram model. The algorithm converges and hence IDM decreases. When IDM falls by 90%, the algorithm is set to explore mode, in which much larger number of parent solutions are taken, and hence the IDM increases. In the unlikely event of IDM exceeding the 50% of IDM value of the

initial population, the algorithm is set back to *exploit* mode to accelerate the convergence and the process continues.

7.3.2 Application 1: test functions

In the previous chapters, we used fit-to-purpose test functions. In Chapter 5, the Rosenbrock test function was used to test the ability of the algorithms to escape from the local optima. In Chapter 6, Rastrigin test function was used to test algorithms on problems with many local minima.

We tested AHEDA initially on above test functions, Rosenbrock and Rastrigin, which are designed for testing of the optimisation algorithms on problems with local minima. If an algorithm is not tuned well or is not adaptive enough, it can easily get trapped in local minima of these functions.

7.3.2.1 Rastrigin test function application

A successful history matching aims to explore all possible local minima in the search space, along with the global minimum, which together represent the uncertainty in the reservoir model parameters. We applied HEDA and AHEDA to the Rastrigin function to test whether they are able to explore multiple minima, a common feature of history matching problems. For ease of plotting, we use a Rastrigin function with three decision variables, i.e. $\boldsymbol{n = 3}$ in equation (6.13). We know that this function has many local minima and one global optimum, $f(x)=0$, at the centre point, $x=[0, 0, 0]$.

In the Rastrigin function application, the stopping criterion is set as 2,000 function evaluations, with a fixed number of children in each generation, equal to 50. The initial population size was 300, so the number of parents could be chosen between as 50 and 300. Another control parameter of these algorithms is the bin size, for which the algorithm was tuned and the best value found to be 25. The results of the two algorithms were compared for misfit convergence and parameter estimation in four stages of the evolution: generations 0, 11, 22, and 34.

HEDA was run with three different values for the parent size (50, 150, and 300). AHEDA was run with two search modes: exploitative (parent size of 50) and explorative (parent size of 300). To minimise the effect of the seed on results, all four

experiments were forced to start from the same initial population of 300 random solutions. Our results for the Rosenbrock function application show that with parent size of 150 and 300, AHEDA is able to better converge to the global minimum than HEDA while it has maintained diversity and local minima up to the end of evolution (Figure 7.9).

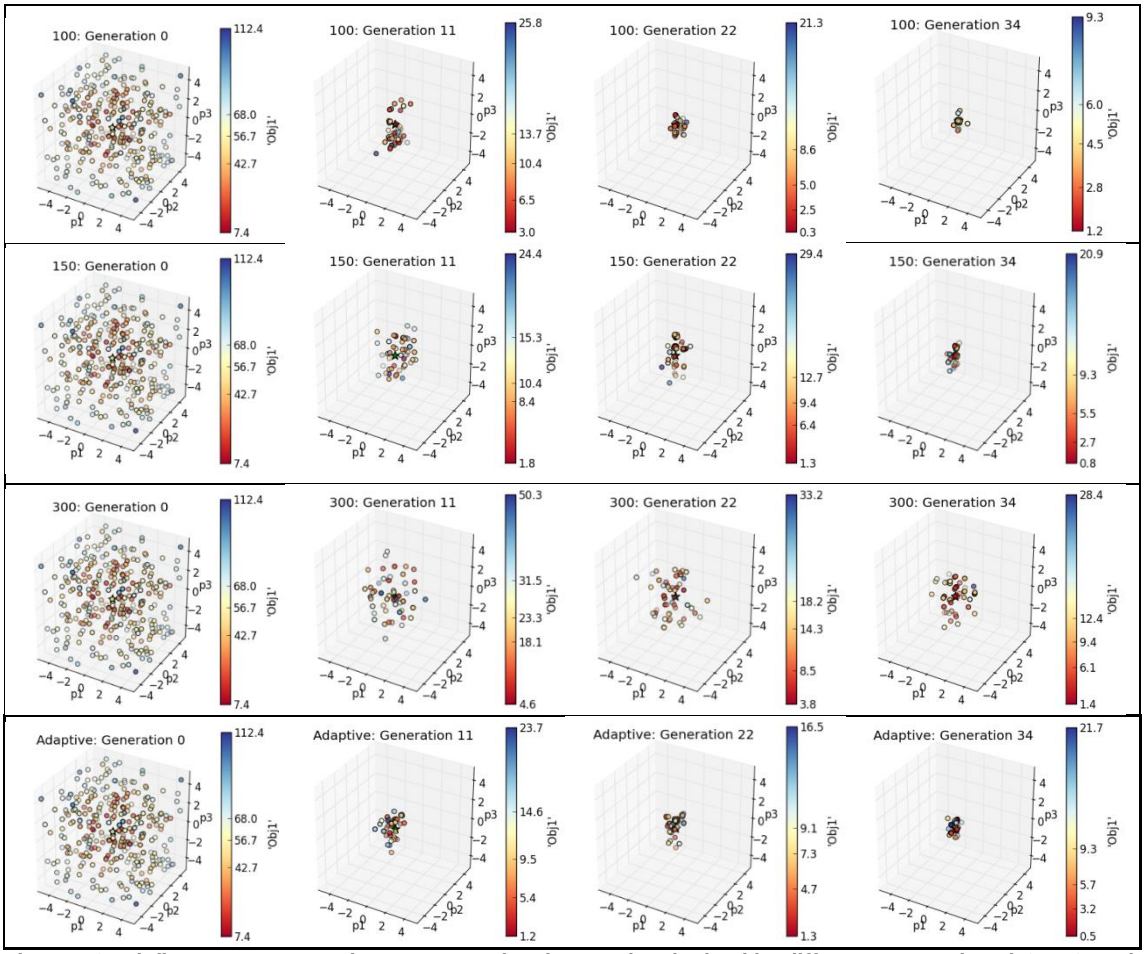


Figure 7.9: Misfit convergence and parameter estimation results obtained by different parent sizes (50, 150, and 300) of HEDA and AHEDA for the Rastrigin function application in four selected generations (0, 11, 22, and 34). The coloured bar shows boxplot misfits (minimum, P10, P50, P90, and maximum) and the green star shows the global optimum. AHEDA better converges to the global minimum compared to HEDA, with parent size of 150 and 300, while it has also better maintained the diversity compared to the parent size of 50.

7.3.2.2 Rosenbrock test function application

A common problem of the evolutionary algorithms applied in history matching is premature convergence. We use the Rosenbrock function to test the ability of the algorithms to escape from the entrapment. The Rosenbrock function often serves as a test case for this situation, where the algorithms are easily trapped in the valley because of the function's shape. Here, we use the Rosenbrock function with three decision variables. The function has its global optimum, $f(x_i) = 0$, at $x_i = [1,1,1]$.

Apart from the parent size, which largely controls the convergence and diversity of the algorithm, we used the same control parameters for HEDA and AHEDA. We repeated each experiment 10 times and averaged for minimum fitness and convergence speed. The results are shown in Figure 7.10 and Figure 7.11. Misfit convergence, diversity, and parameter estimation results in the figure show that AHEDA has better converged to the global minimum than HEDA regardless of the parent size. It can be seen that that AHEDA has also better maintained diversity compared to HEDA.

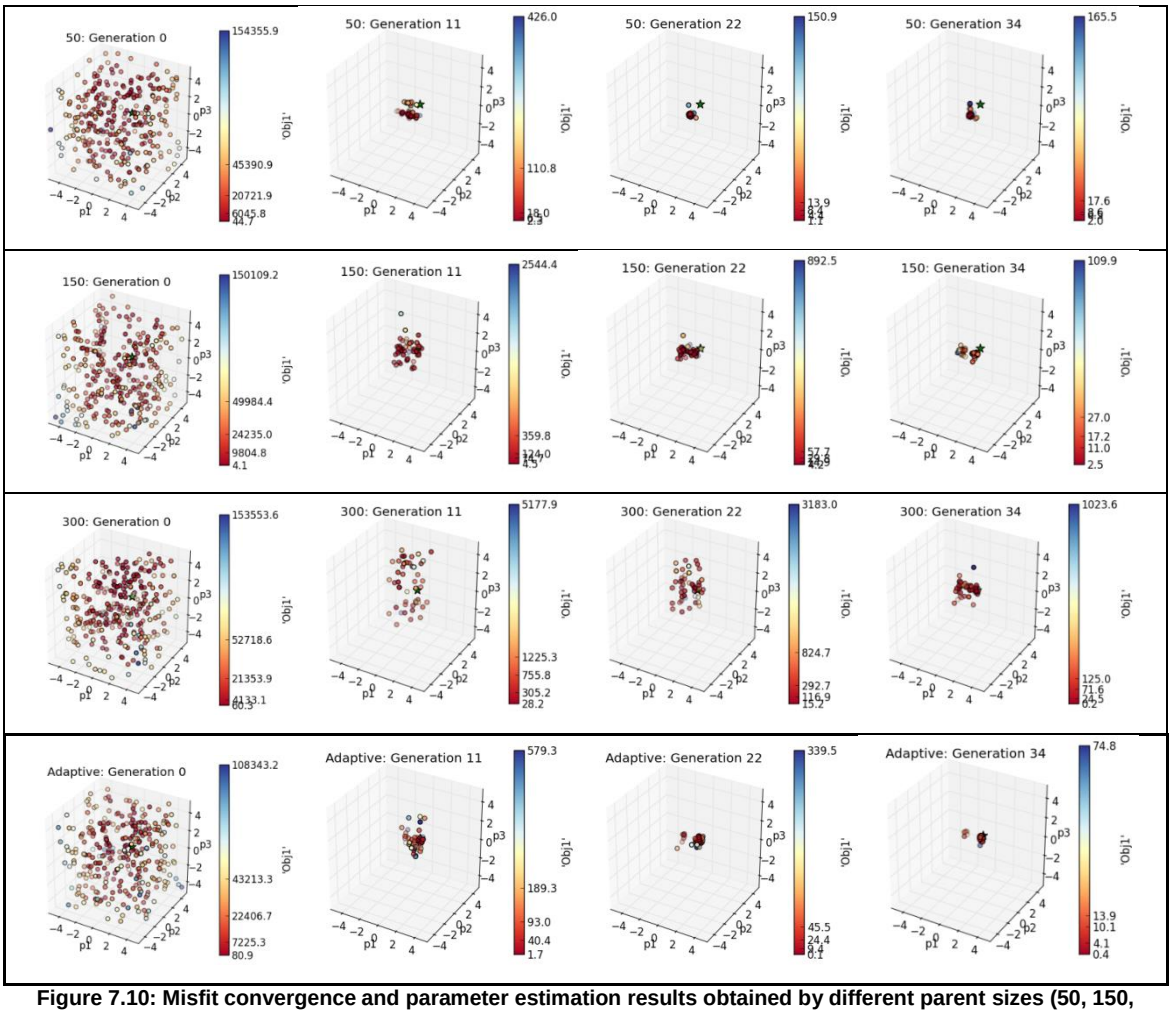


Figure 7.10: Misfit convergence and parameter estimation results obtained by different parent sizes (50, 150, and 300) of HEDA and AHEDA in four selected generations (0, 11, 22, and 34) of the Rosenbrock function application.The coloured bar shows boxplot misfits (minimum, P10, P50, P90, and maximum) and the green star shows the global optimum. AHEDA has better converged to the global minimum than HEDA, with any parent size, while it has also better maintained the diversity. HEDA with parent size of 50 has become trapped in the local minimum, while AHEDA has found both local and global optimums.

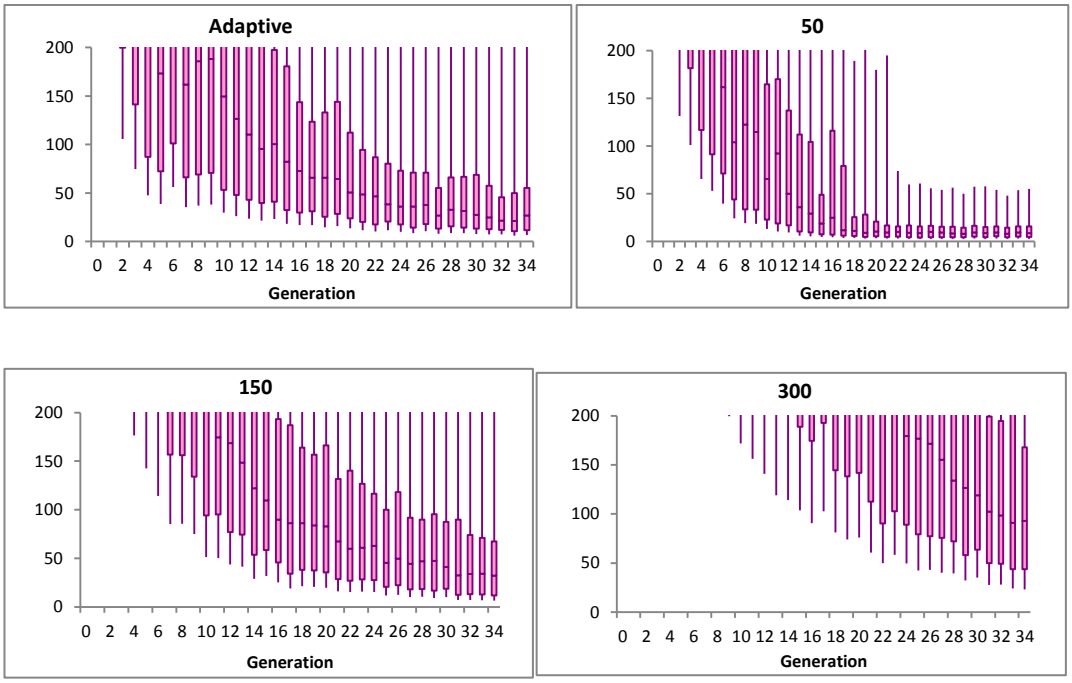


Figure 7.11: Misfit convergence box-plots (minimum, P10, P50, P90, and maximum) for misfit under 200 for the average of 10 runs by different parent size (50, 150, and 300) HEDA and AHEDA in Rosenbrock application. Adaptive parent size has converged better than parent sizes 150 and 300, especially at the early stages. Ithas also maintained better diversity than parent size 50 at the later stages of the search.

7.3.3 Experiment 2: History matching of the IC-Fault model

Following analysis of the test function results, we applied AHEDA to the hard history-match problem of the IC-Fault model (Carter et al., 2006). We aimed to evaluate the performance of the diversity-based adaptive mechanism for HEDA on a difficult history matching problem which has steep minima in its misfit landscape. History matching a problem with steep minima in misfit landscape requires strong explorative properties and high level of diversity.

In Chapter 5, we applied incremental histogram-based EDA (iHEDA), simulated binary GA (SBGA) and a hybrid SBGA/iHEDA to history matching problem of IC-Fault model. The experimentation showed that SBGA can have difficulty of converging in early convergence, while iHEDA can suffer from low diversity. The hybrid algorithm was an attempt to achieve a better search mechanism by taking the advantages of two algorithms, so that it could improve the search performance.

In current chapter, we aim to improve the convergence and diversity of search in IC-Fault model by using the diversity-based adaptive EDA, instead of hybridisation in

Chapter 5. The adaptive mechanism is aimed to enable the algorithm to escape from the steep minima in IC-Fault model.

In the IC-Fault application, we defined the stopping criterion as a total of 1,000 simulation runs. This number is probably affordable in terms of the number of simulations in most assisted history matching problems. The number of bins in the histogram model was also fixed at 20, which allows precise enough discretisation of the parameter space. The initial population size was chosen as 120, which allows an adequate scanning of the search space in the initial generation. Another control parameter is the number of children, which in this case was fixed to 40, considerably larger than the number of bins. Therefore it leaves total of 22 generations, excluding the initial random generation.

Our previous studies (Chapter 3) with histogram-based EDAs showed that using the same number of parent solutions as child solutions (a ratio of one) causes an exploitative search to be performed, while increasing the ratio to two or three, makes the search more explorative.

AHEDA starts the search in exploitative mode, i.e. the number of parents is equal to the number of children (here 40). Then when the IDM is less than a threshold (here 0.01), it switches to the explorative mode by increasing the number of parents to initial population size (here 120). As Figure 7.12 shows, in IC-Fault model application, AHEDA switches two times between exploitative and explorative search mode. The IDM thresholds here are 0.01 and 0.03 respectively for switching to explorative and exploitative modes.

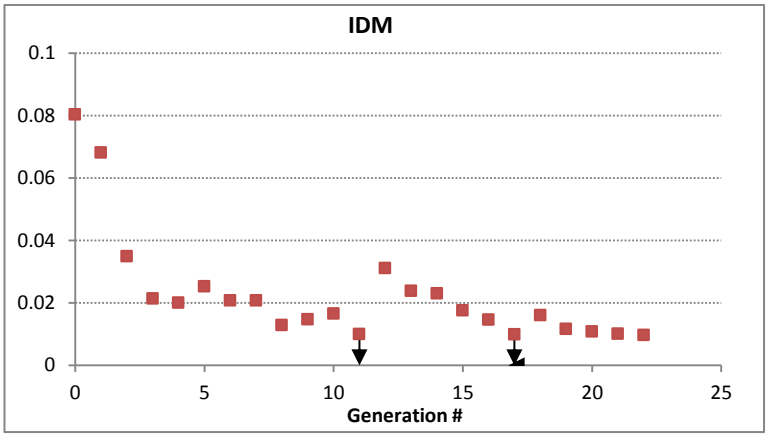


Figure 7.12: IDM in AHEDA application on IC-Fault model. The algorithm switches two times between exploitative and explorative modes, as shown by the vertical arrows.

Each experiment was run 10 times and results were averaged for the comparative study of HEDA and AHEDA. Figure 7.13 shows the minimum misfit and convergence speed of the algorithms. AHEDA is preferred due to lower minimum misfit and better convergence for an average of 10 trials.

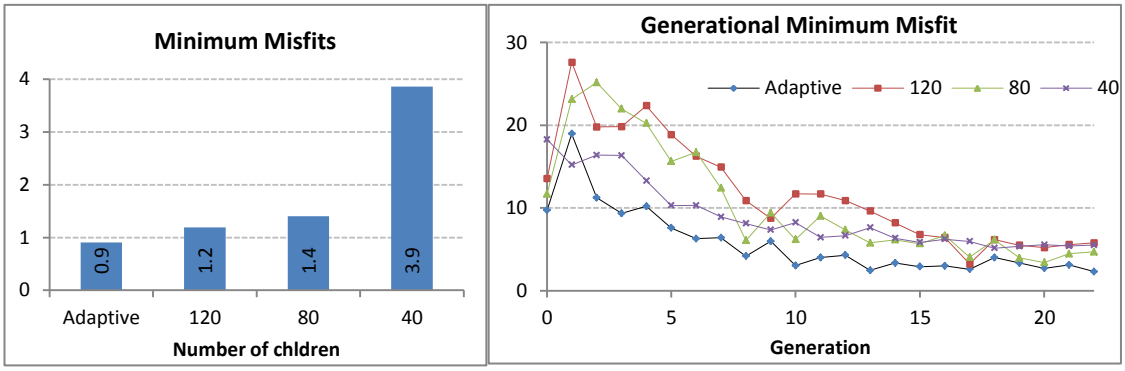


Figure 7.13: Minimum misfit found at entire evolution (left) and per generation (right) for different values of the number of parents in HEDA. AHEDA is preferred due to lower minimum misfit and better convergence.

Misfit convergence and parameter estimation results for an average run of three different parent sizes (40, 80, and 120) in HEDA and AHEDA were studied in four selected generations (0, 7, 14, and 22) (Figure 7.14). The figure shows that AHEDA has better convergence to the global minimum than HEDA, with any parent size. It has also better maintained the population diversity at the later stages of the search.

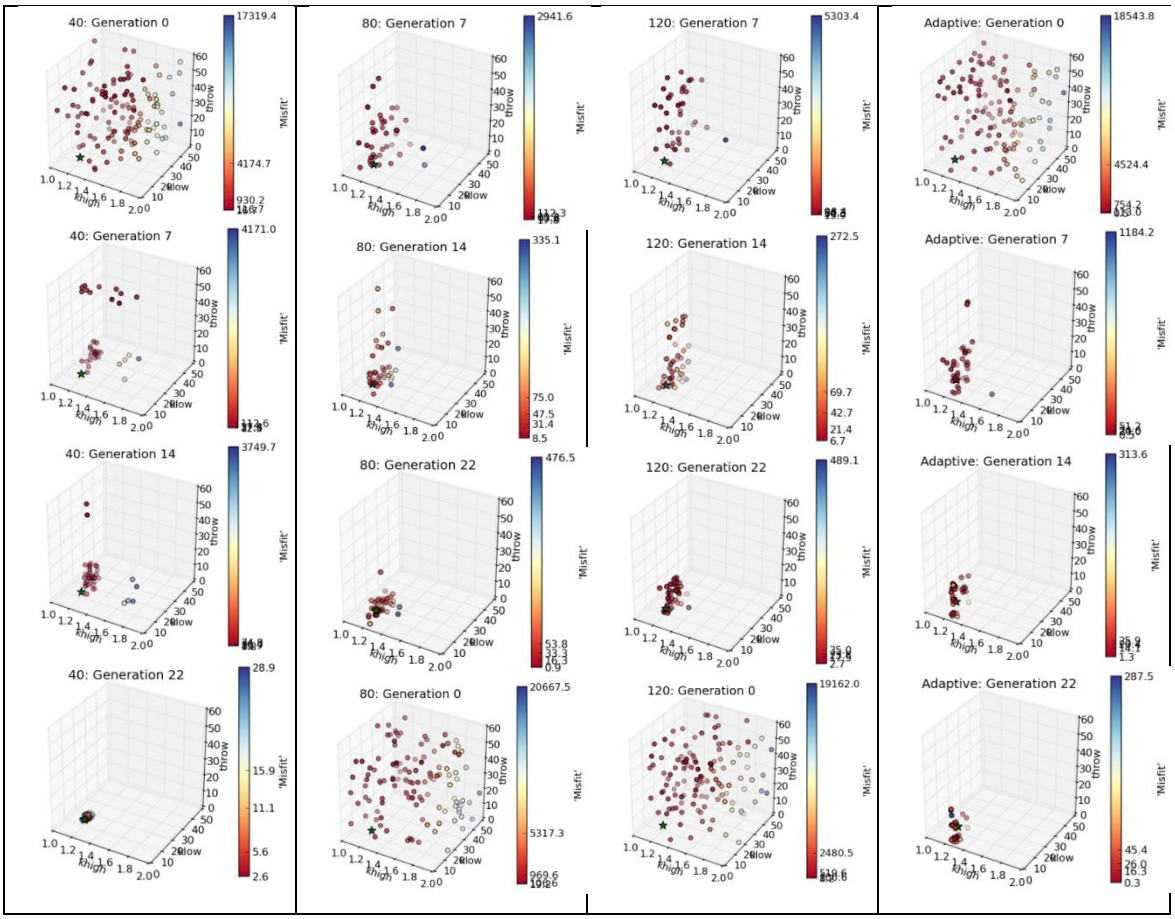


Figure 7.14: Misfit convergence and parameter estimation results obtained by different parent sizes (40, 80, and 120) of HEDA and Adaptive HEDA in four selected generations (0, 7, 14, and 22) of the Rosenbrock application. The coloured bar shows boxplot misfits (minimum, P10, P50, P90, and maximum) and the green star shows the truth case. AHEDA has obtained best convergence to the global minimum while it has also maintained the diversity by keeping search in three areas (clusters of the solutions).

7.4 Discussion

Distance is the most widely used concept to differentiate solutions in the search space and entropy represents the amount of disorder of the population. In principle, an increase in the distance or entropy represents an increase in the diversity. Diversity measures can show valuable information about the convergence and performance of the algorithms.

7.4.1 Interpretation of the diversity measure curves

An important issue regarding the use of diversity measures is how to interpret the diversity/convergence from the curves. Figure 7.15 shows typical situations in distance-based and entropy-based measures. Convergence requires a reduction in the distance based measure, but the curve must not touch or become too close to the horizontal axis,

as this represents zero diversity and trapping in local minima. In an ideal search, the entropy-based measure (1/diversity) shows a slight decrease or constant entropy throughout the generations. When the search has fallen into an entrapment, the diversity decreases significantly, and the curve shows a sharp increase in entropy.

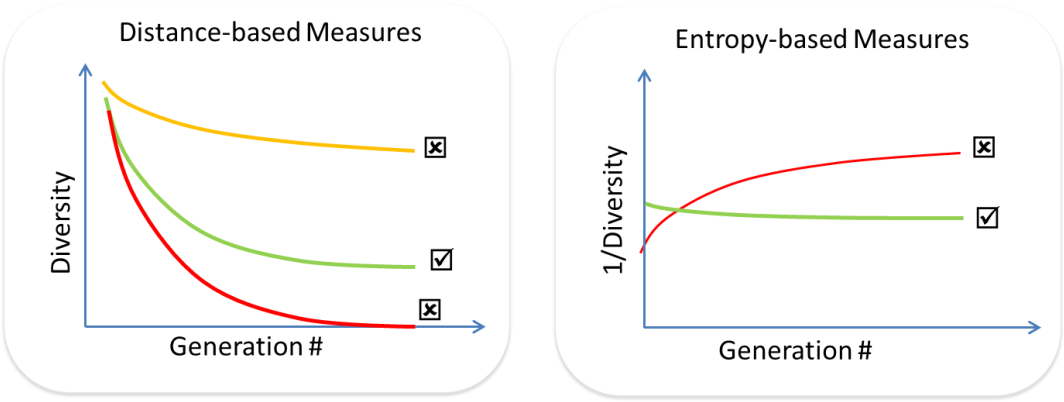


Figure 7.15: Typical situations in distance-based measures (left) and entropy-based measures (right). For the distance-based measures, the yellow curve may indicate not enough convergence or convergence to multiple areas, the red curve shows an entrapment for the search, and the green curve shows the desired search. For the entropy-based measures, the green curve shows balanced convergence/diversity while the red curve shows loss of diversity.

If the search converges to multiple regions around the search space, the distance-based measure may not properly show the convergence (yellow curve in Figure 7.15); however, the entropy-based measures will show convergence and hence should be used in conjunction with the distance-based measure.

7.4.2 Choice of diversity measure for use in adaptation

One important issue regarding the adaptation mechanism in AHEDA is the choice of diversity measure. Between the two groups of measures, the distance-based group is chosen, since these are easier to implement. The entropy-based measures show fluctuations, which need to be smoothed using smoothing techniques, such as moving average, if used for adaptation.

Between the two distance-based measures, IDM was chosen for use in the adaptation mechanism, as its calculation is straightforward and only needs $4(NP) + N$ calculations, which is more efficient compared to the Hamming distance measure (HDM). It should be noted that the computational complexity of HDM is not considerable comparing to expensive reservoir simulation runs.

7.5 References

Boss, C. (1999). Production forecasting with Uncertainty Quantification. Technical Report. Netherlands Institute of Applied Geoscience, TNO.

Burke, E.K., Gustafson, S., Kendall, G. (2004). Diversity in genetic programming: an analysis of measures and correlation with fitness. IEEE Transactions on Evolutionary Computation. 8(1), 47-62.

Carter, J.N., Ballester, P.J., Tavassoli, Z., King, P.R. (2006). Our calibrated model has poor predictive value: An example from the petroleum industry. The Fourth International Conference on Sensitivity Analysis: Reliability Engineering & System Safety, 91(10-11), 1373–1381.

De Jong, K. A. (1975). An Analysis of the Behaviour of a Class of Genetic Adaptive Systems, Ph.D. thesis, University of Michigan, Ann Arbor, Michigan.

Deb, K. (2001). Multi-objective Optimization Using Evolutionary Algorithms. Wiley, Chichester.

Ding, N. & Zhou, Sh., D. (2008). Linkages Detection in Histogram-Based Estimation of Distribution Algorithm. In Chen, Y.-p. & Lim, M.-H. (Eds.): Linkage in Evolutionary Computation. SCI 157, 25–40. Berlin Heidelberg: Springer-Verlag.

Ding, N., Zhou, Sh., D., & Sun, Z. Q. (2007). Histogram-Based Estimation of Distribution Algorithm: A Competent Method for Continuous Optimization. Journal of Computer Science and Technology, 23(1): 35-43.

Eiben, A. E., Hinterding, R., & Michalewicz, Z. (1999). Parameter control in evolutionary algorithms. IEEE Trans. Evol. Comput., vol. 3(2), 124–141.

Gouvêa Jr, M.M. and Araújo, A.F.R. (2010). Diversity-Based Adaptive Evolutionary Algorithms. In Korose, P. (Ed.): New Achievements in Evolutionary Computation. 318-334.

Hamming, R. W. (1950). Error detecting and error correcting codes. Bell System Technical Journal, 29 (2), 147–160

Horn, J. (1997). The Nature of Niching: Genetic Algorithms and the Evolution of Optimal, Cooperative Populations. Ph.D. Thesis, University of Illinois-Champaign.

Lacevic, B., Amaldi, E. (2010). On population diversity measures in Euclidean space. IEEE Congress on Evolutionary Computation (CEC). 1-8. Barcelona.

Larrañaga, P. and Lozano, J.A. (2001). Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation. Boston, USA: Kluwer Academic Publisher.

Liu, Sh., Mernik, M., Bryant, B. R. (2007). Entropy-Driven Parameter Control for Evolutionary Algorithms. Informatica, 31, 41–50.

Morrison, R. W. and Jong, K. A. D. (2002). Measurement of population diversity. In Selected

Papers from the 5th European Conference on Artificial Evolution, Springer, pages 31–41, London, UK.

Reynolds, A., Abdollahzadeh, A., Christie, M.A., Corne, D., Williams, G., Davies, B. (2011). A Parallel BOA-PSO Hybrid Algorithm for History Matching. IEEE Congress on Evolutionary Computation (CEC). New Orleans, LA, USA.

Rosca, J. (1995). Entropy-driven adaptive representation. In Rosca, J., editor, Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications, pages 23-32, Tahoe City, CA, USA.

Rosenbrock, H. H. (1960). An automatic method for finding the greatest or least value of a function. The Computer Journal 3: 175–184.

Santana, R., Larrañaga, P., and Lozano, J.A. (2008). Adaptive Estimation of Distribution Algorithms. In Cotta, C., et al. (Eds.): Adaptive and Multilevel Metaheuristics, SCI 136, 177–197. Berlin Heidelberg: Springer-Verlag.

Shannon, C.E. (1948). A mathematical theory of communication. Bell System Technical Journal, 27:379-423, 623-656.

Ursem, R.K. (2002). Diversity-Guided Evolutionary Algorithms. Proceedings of the Congress on Evolutionary Computation, 1633-1640. IEEE Press.

Wang, Y., Wu, L.H., Yuan, X.F. (2010). Multi-objective self-adaptive differential evolution with elitist archive and crowding entropy-based diversity measure. Soft Computing, 14, 193–209.

Wineberg, M., Oppacher, F. (2003). The Underlying Similarity of Diversity Measures Used in Evolutionary Computation, Proc. Genetic and Evol. Comput. Conf. 1493-1504.

CHAPTER 8:

ADAPTIVE ALGORITHMS FOR UNCERTAINTY QUANTIFICATION

> "As far as the laws of mathematics refer to reality, they are not certain; and as far as they are certain, they do not refer to reality."
>
> Albert Einstein (1879 - 1955)

8.1 Introduction

In reservoir engineering, uncertainty quantification of prediction parametersfuture reservoir behaviour involves determining the uncertainty of input variables and a simulation model which computes prediction parametersreservoir performancefor any given set of input variables. The uncertainty of input reservoir parameters is shown by a probability distribution function, which if sampled on a large enough scale, can be used to determine the probability distribution function of the prediction parameters. Figure 8.1 illustrates a general framework for uncertainty quantification of a predictive parameter such as cumulative oil production.

If a large number of uniform samples/realisations is available, one can use a regular Monte Carlo integration to obtain the probability distribution of prediction parameters and credible intervals (P10, P50, and P90) for decision.

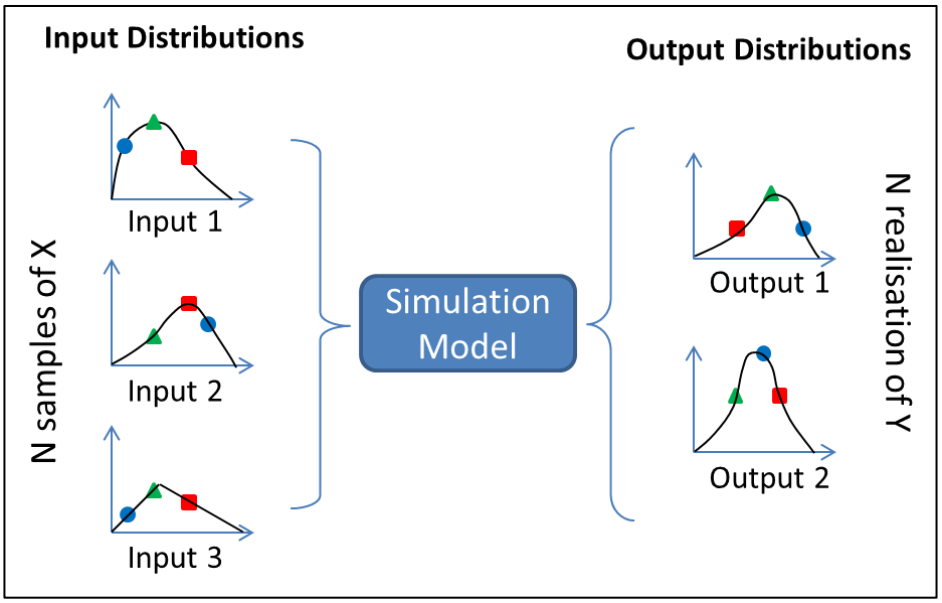


Figure 8.1: A general framework for uncertainty quantification of prediction parameters. Many samples from the distribution function of the input parameters are sampled and simulated to obtain the distribution function of outputs (prediction parameters). The coloured circle, triangular and square coloured marks each represent a sample/realisation.

In reservoir uncertainty quantification, samples are usually non-uniform and limited. Reservoir simulations are expensive, and a limited number of forward simulations can be done in an uncertainty quantification study. In addition, history matching using optimisation algorithms makes use of importance sampling, i.e. certain areas of the uncertainty parameter space are sampled more frequently.

The Bayes factor is the posterior probability ratio of two models; for an ensemble of models obtained in history matching with optimisation algorithms, the prior probability and the denominator in equation (2.6) are alike and the Bayes factor of the two models ($M_1$and M_2) is, in fact, the ratio of their likelihoods.

$$BF = \frac{p(M_1|D)}{p(M_2|D)} = \frac{p(D|M_1)}{p(D|M_2)} \tag{8.1}$$

In some reservoir applications, e.g. reservoir development optimisation and planning, model selection is carried out when a limited number of model realisations must be selected. Although one can use the best history-matched model, which has a higher Bayes factor with regard to all other models, for reservoir forecasting and planning, a single model is insufficient to be used as the basis for important decision making in reservoir planning and management, as it does not allow uncertainty and estimation of risk to be taken into account.

The Bayesian model averaging is used for uncertainty quantification, in which the response quantity is estimated under each model, and then estimates are averaged according to their likelihood ratio.

In the past two decades, various techniques have been introduced for estimating posterior distribution, hence quantifying the uncertainty as discussed in the following paragraphs. These techniques can be classified as approximate or Monte Carlo sampling. In these approximate techniques, we try to approximate the posterior distribution with a limited number of simulations or objective function calculations. The credible interval obtained by approximate techniques may be too narrow and unrealistic (Baker & Cuypers, 2000). Monte Carlo sampling techniques are more common for searching the parameter space. This would need numerous forward simulation runs, which are computationally expensive. To work around this problem, one can use proxy models instead of forward simulations, but these may introduce significant modelling errors.

Design of Experiments and Response Surface Methods are also widely used for uncertainty quantification (e.g. see Damsleth et al., 1994, Manceau et al., 2001 and Montgomery, 2001). These statistical approaches are used, firstly, to identify uncertain parameters that most affect the response parameters, via a minimum number of simulations, secondly, to fit a surface, typically a linear or quadratic model, of response variables that can be used as input to a simple Monte Carlo sampling.

The PUNQ project (Floris et al., 1999) proposed some types of approximate and sampling methods for quantifying uncertainty. The first class comprised Maximum Likelihood solution plus local characterisation of the likelihood function (ML+), or Maximum *A Posteriori* solution plus local characterisation of the objective function (MAP+), when the prior term is included (Roggero, 1997). Another class was Multi-ML and Multi-MAP, depending on which objective function is used, for which it is assumed the objective function is truly multimodal and different history-matched models located at the distinct optima show the uncertainty. Multi-ML+ and Multi-MAP+ forms another class and is a combination of the two previous class, i.e. local characterisation of the objective function around distinct local optima is used. Floris et al. (1999) also presented the results of Monte Carlo sampling of the full posterior

distribution. The final approach in the PUNQ project was Oliver et al. (1996), which aimed to estimate the complete posterior using an optimisation technique to sample both the prior and the observation data, and hence, to reduce the number of simulation runs needed.

Another sampling method is to use Markov Chain Monte Carlo (MCMC) to estimate the posterior distribution from the ensemble of models obtained in history matching. In this method, the full parameter search space is sampled, but instead of running forward simulations for each sample, the likelihood function is approximated from the existing ensemble of models (e.g. see Subbey et al., 2003). However, the shape of the posterior distribution is very high-dimensional, non-Gaussian and multimodal. Another issue with this approach is that the normalising constant (in equation 1) is unknown and evaluating this constant is not a trivial issue.

The chapter is organized as follows. First, we discuss a Bayesian framework for history matching and uncertainty quantification, followed by a literature review of the methods used for model selection and uncertainty quantification. In the methodology section, we discuss clustering and describe three clustering techniques used in this work. In the application section, we present the results of field applications. Finally, in the discussion section, we discuss the results and make some concluding remarks.

8.2 Methodology

We introduce the methods and techniques used including k-nearest neighbours approximation, probabilistic distance clustering, the Neighbourhood Algorithm with Bayesian inference, and the Metropolis-Hasting algorithm with an adaptive multivariate Gaussian proposal.

8.2.1 K-NN Approximation

For inference from an ensemble, a continuous approximate distribution must be created from a finite ensemble of models. The approximate distribution could be used to approximate samples in an arbitrary position in the search space. In a high dimensional space such an approximation technique practically becomes an interpolation problem.

k-nearest neighbours (K-NN) approximation is a regression method to evaluate the features of a new datapoint by simply examining the *k* closest datapoints in the parameter space to the new datapoint and assigning a value equal to the inverse-distance weighted average of these *k* neighbours' feature values. In this chapter, we use this approach to approximate a newly sampled model's posterior probability (or misfit value) from its k-nearest neighbour models in the ensemble instead of forward simulation of the model.

An important aspect of K-NN approximation is choosing an appropriate value for *k*. The most basic form takes *k*=1. However this yields a pretty unstable approximation (high variance and sensitive to the data) and the approximation can often be done more consistently by increasing *k* (Hand et al. 2001). However, increasing *k* may introduce bias, since it may include data points in the space which are not necessarily very close to the new data point, especially in high-dimensional space.

We choose *k* using a data-adaptive strategy in K-NN approximation. We try various *k* values (usually 1 to 10) to approximate the misfit value of each model in the ensemble from that of its closest neighbours, and we measure the mean squared error (MSE) of the ensemble. Then we plot MSE, as the performance criterion of the approximation, against *k*. By increasing *k*, MSE drops to a minimum or, from a particular *k* value onwards, the drop is not significant. In other words, adding more complexity to approximation (increasing *k*) does not significantly improve the approximation. This *k* is selected as the best choice for K-NN approximation.

Another aspect of K-NN approximation is to use an appropriate distance metric when measuring the closeness of the data points. Two main distance measures are Euclidean and Mahalanobis distance. In the current work, we use Euclidean distance, equation (2.20), which is one of the most widely used similarity measures. More details on this measure can be found in the Chapter 2.

8.2.2 Adaptive clustering algorithms

MCMC sampling is a time consuming process, especially if a single walk (or chain) is used, and the parameter space is high-dimensional and a large number of samples are going to be taken. Multiple independent walks can accelerate the sampling process by

an order of magnitude. These multiple walks can start from randomly selected points in the search space. An alternative to random starting points is to use a clustering algorithm to perform a clustering in the parameter space, and then start multiple walks from the best model (with lowest misfit) of each cluster.

In Chapter 2, we reviewed clustering analysis and described three algorithms: k-mean clustering (KMC) (MacQueen, 1967), hierarchical agglomerative clustering (HAC) (Manning et al., 2008) and probabilistic-distance clustering (PDC) (Ben-Israel, 2006). Then we applied these three algorithms to the clustering problem of the Iris dataset and showed that PDC outperforms other two algorithms; it was therefore selected to be used in the present chapter.

A difficult problem in clustering is the number of clusters, since we often have no idea of how many clusters exist. There is no general formula to find the optimal number of clusters applicable to any given data set. A rule of thumb is to use $\sqrt{\frac{N}{2}}$ as the number clusters, where N is the number of models. In the following section, we introduce a heuristic method to determine the number of clusters through the algorithm itself.

PDC(Iyigun & Ben-Israel, 2008) starts from a predefined number of clusters. In the case of lack of prior information regarding the optimal number of clusters, one can do multiple runs of clustering, with different $\boldsymbol{k}$ numbers, then compare the results and choose the best number for k, based on a given criterion. A common criterion in PDC is the sum of the constant value in equation (2.22), for all models (Ben-Israel, 2006). This constant is called the joint distance function, denoted by JDF and expressed as follows.

$$JDF = \sum_{i=1}^{N} \frac{\prod_{j=1}^{K} d(m_i, c_j)}{\sum_{l=1}^{K} \prod_{j \neq l}^{K} d(m_i, c_j)} \tag{8.2}$$

We use an iterative procedure to determine the number of clusters in PDC; i.e. we start from $\boldsymbol{k}$ =2, calculate joint distance function, JDF and iterate (k+=1) until JDF drops to a significance level (e.g. 90%, in Figure 8.2).

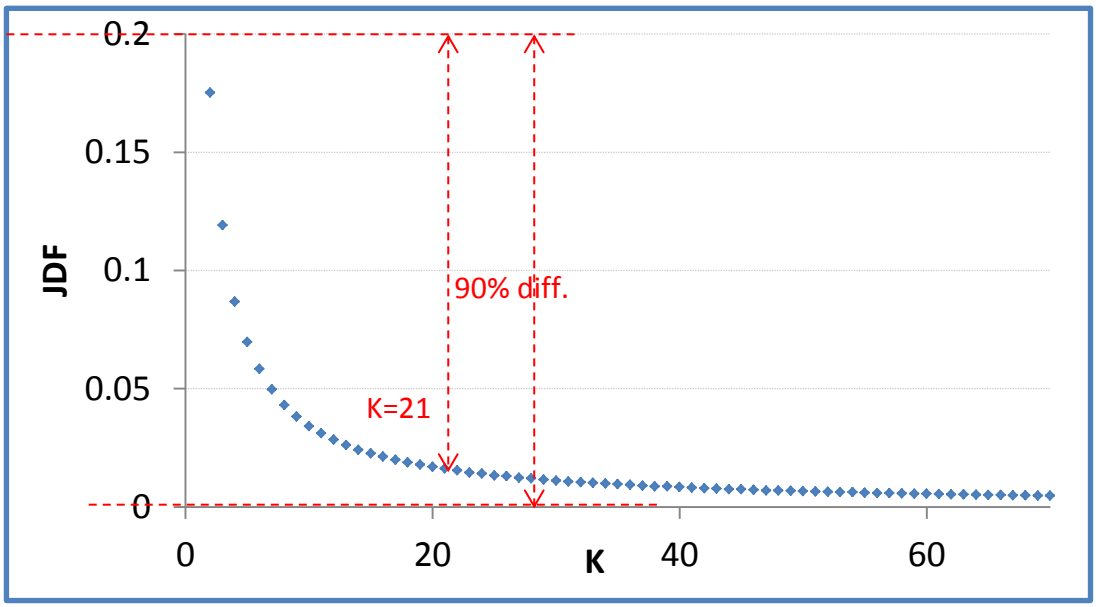


Figure 8.2: Total Joint Distance Function used to determine the number of clusters in PDC. Here, a 90% drop in JDF is seen at k=21.

8.2.3 Neighbourhood Algorithm with Bayesian inference

The Neighbourhood Algorithm with Bayesian inference (NAB) is an MCMC-based uncertainty quantification method introduced by Sambridge (1999). In NAB, one can create a random walk using a Gibbs sampler that has the posterior probability $p(M|D)$ as its stationary distribution. The random walk should be long enough so that the resulting samples closely approximate samples from $p(M|D)$. The quality of the samples improves as a function of the number of random steps in the walk. These samples can be used directly for parameter inference and prediction.

In NAB, a neighbourhood approximation based on splitting the space into Voronoi cells is used to avoid using forward simulation of the resampled models. NAB allocates the likelihood of each model in the ensemble to the points located in the Voronoi cell around the model. NAB accounts for the sampling density by taking a random deviate from the variable range and assigning the probability of the Voronoi cell which the sample falls into.

Sambridge (1999) also proposed multiple independent random walks, each starting from a different point (e.g. better fitting models) in model space, to reduce the computation time. He also equipped NAB with a scaling mechanism of the parameter ranges in which a scale factor (e.g. max minus min value) is applied to the parameter values of the

ensemble before sampling. NAB has been used in many uncertainty quantification studies, among them Subbey et al. (2003) and Mohamed et al. (2010).

8.2.4 Adaptive proposal

In Chapter 2, we reviewed MCMC sampling algorithms and among them the Metropolis-Hastings (M-H) algorithm. If properly tuned and configured, this algorithm can perform a quick and flexible sampling for estimating posterior distributions in parameter estimation problems such as history matching and uncertainty quantification.

In this chapter, we use a multivariate Gaussian probability function as the proposal distribution for sampling the unknown target distribution of uncertainty parameters, which is approximated by an ensemble of calibrated models obtained in history matching. As discussed in Chapter 2, the Metropolis algorithm refers to a special case of the M-H algorithm, in which a symmetrical proposal distribution is used. Since multivariate Gaussian distribution is symmetrical, the right term for our sampling algorithm is a Metropolis algorithm.

The Metropolis algorithm (and M-H in general) is often used in situations where the target distribution is unknown, the prior knowledge is quite limited, or calculating the marginal distributions is a difficult task to undertake in Gibbs sampling (Gelfand & Smith, 1990). In such situations, the tuning of the proposal distribution is really crucial.

The tuning of the proposal can be done offline or online; in the former mode, the best parameters for the proposal distribution are found using a trial and error procedure and applied to the problem; in the latter, an adaptive scheme can be used to tune the proposal distribution by utilising the information obtained during the sampling and, thus, automatically adapting the proposal distribution to the target distribution. This adaptive proposal strategy was first proposed by Haario et al. (1999).

The proposal distribution is a multivariate Gaussian probability distribution, which has two control parameters, mean and covariance matrix. The mean is taken as the current sample, and the covariance matrix is either tuned with the initial value of ensemble, or adapted during the sampling, using an adaptive proposal strategy similar to the strategy proposed by Haario et al. (1999).

If a Gaussian proposal distribution is used, the covariance matrix is tuned during the burn-in period, by calculating the acceptance rate, which is the ratio of accepted samples to total proposed samples (accepted plus rejected). The desired acceptance rate varies for different target distributions; however, the ideal acceptance rate has been theoretically obtained to be around 50% for one dimensional Gaussian target distribution, reducing to about 23% for high-dimensional Gaussian target distributions (Roberts et al., 1997).

The adaptive proposal of Haario et al. (1999) takes three setup parameters for adapting the covariance factor: an initial value for the covariance matrix, a memory parameter H, which is the number of previous samples that the covariance matrix is updated with, and a frequency parameter U, which is a certain number of samples after which the updating happens every U times. The proposal distribution of the new sample X_{t+1} is as follows:

$$Q(\mathrm{X}_{t+1}|\mathrm{X}_t) = \mathcal{N}(\mathrm{X}_t, f_\Sigma \Sigma_H) \tag{8.3}$$

where X_t is current sample, Σ_H is the covariance matrix of the last H samples (i.e. $\{\mathrm{X}_{t-H+1}, \mathrm{X}_{t-H+2}, \dots, \mathrm{X}_t\}$), and f_Σ is covariance scaling factor.

Unlike in Haario et al. (1999), here the adaptation only takes place in the burn-in period, during which the samples are not collected and counted toward the final posterior distribution. After the burn-in period, the proposal distribution is fixed, and sampling is continued with the adapted covariance matrix.

Gelman et al. (1996) used a heuristic approach to choose the initial scaling factor, supported by theoretical optimisation of mixing properties and empirical tests of the M-H sampling when the target and proposal distributions are both Gaussian. Their approach relates the covariance factor f_Σ only to the number of dimensions d through:

$$f_\Sigma(d) = \frac{2.4^2}{d} \tag{8.4}$$

8.2.5 Adaptive Metropolis-Hasting with K-NN Approximation

In this section, we introduce a procedure of uncertainty quantification based on adaptive Metropolis-Hasting sampling, which employs the multivariate Gaussian proposal

probability distribution with the mean of the current sample and covariance matrix, as adapted in the burn-in period. The algorithm is referred to as Adaptive Metropolis-Hastings (AMH) throughout this thesis.

The AHM is as fast as any other M-H algorithm, since in the acceptance/rejection step, as it collects the current sample for each rejected proposal. This makes the MCMC algorithm more efficient than NAB, a Gibbs sampler with a rejection step, which collects only accepted proposals.

Instead of simple neighbourhood approximation, as in NAB, AMH uses a more robust and sophisticated K-NN approximation, in which the posterior probability of an arbitrary sample in the search space is approximated by the weighted average of the k nearest neighbours' posterior probabilities. The weight factors are the normalized inverse of a distance measure between the sample and its neighbours.

AMH is also equipped with PD clustering, to select optimum starting points of the random walks. If multiple independent walks are taken in parallel, a huge efficiency can be achieved in practice. The PD clustering of input ensemble is done in the normalized (scaled to 0-1 range) variable space and a random walk is initiated from the best fitting model of each cluster.

The general workflow for AMH is as following:

1. If multiple walks are chosen, perform PD clustering of the input ensemble in normalized variable space.
2. Start multiple parallel random walks, $t = 0$, from the best fitting model of each cluster, $X^{(0)}$, initialize the desired set of samples with zero probabilities $\mathbb{S} = \{0.0\ for\ \mathrm{X}_{(\mathrm{j})}\ in\ cluster\}$. Assign probability 1.0 in set $\mathbb{S}$ to best fitting model of the cluster.
3. Repeat the following steps in each walk with the current sample of $X^{(t)}$ for a desired number of iterations, that is, the total number of desired samples divided by the number of clusters:

3.1. Draw a random sample, $X^{(t+1)}$, from the multivariate Gaussian distribution $\mathcal{N}(\mathrm{x}, \Sigma)$ with the mean of the current model, $X^{(t)}$, and covariance of the input ensemble, Σ; i.e.:

$$\mathcal{N}(\mathrm{x}, \Sigma) = \frac{1}{\sqrt{(2\pi)^2}\sqrt{|\Sigma|}} \mathrm{e}^{-\frac{1}{2}\left(X-X^{(t)}\right)^{\mathrm{T}}\Sigma^{-1}\left(X-X^{(t)}\right)} \tag{8.5}$$

Here, a fixed or an adaptive covariance matrix, as discussed in the earlier section, can be used.

3.2. Accept sample $\mathrm{X}^{(t+1)}$ and assign probability of $\mathrm{w_k}$ to the indices of neighbours, $\mathrm{X_{(k)}}$, in set $\mathbb{S}$, if a uniform random deviate, $\mathrm{r} = \mathrm{U}(0,1)$, satisfies the following (8.6) and walk is not in burn-in period, else repeat from step 3.1.

$$r \leq \frac{\mathrm{P}\left(X^{(t+1)}\right)}{\mathrm{P}\left(X^{(t)}\right)} \tag{8.6}$$

where $\mathrm{X}^{(t)}$ is the current sample; since the ratio of two posterior probabilities appears in (8.6), one can use the likelihood ratio instead; even further the log of likelihoods can be used to avoid numerical underflow errors in the computer, i.e.:

$$\log(r) \leq \log\left(\mathrm{P}\left(X^{(t+1)}\right) - \mathrm{P}\left(X^{(t)}\right)\right) \tag{8.7}$$

With the likelihood model defined by equation (2.13), the negative log of likelihoods is equivalent to the misfit. The misfit of the drawn sample, $\mathrm{M}\left(\mathrm{X}^{(t+1)}\right)$, can be approximated using the K-NN approximation, in the form of weighted mean of misfits of the K nearest neighbours in the ensemble, where the weight factors are normalized inverse of distance, i.e.:

$$\mathrm{M}\left(X^{(t+1)}\right) = \frac{1}{\mathrm{K}}\sum_{\mathrm{k}=1}^{\mathrm{K}}\left(\mathrm{w_k}\mathrm{M}\left(\mathrm{X_{(k)}}\right);\ \text{where}\ \mathrm{w_k} = \frac{\frac{1}{\mathrm{d_k}}}{\sum_{\mathrm{k}=1}^{\mathrm{K}}\frac{1}{\mathrm{d_k}}}\right. \tag{8.8}$$

where k iterates over the number of nearest neighbours, d_k is a distance measure (here Euclidian) between sample $X^{(t+1)}$ and its k-th neighbour in the ensemble, and $M(X_{(k)})$ is the misfit of neighbour k.

The burn-in period is a portion of the total samples (e.g. 50%) from the start, which are taken but not collected and counted toward the final set of samples.

8.3 Applications

In this section, we present the results of ensemble-based uncertainty quantification of three cases: a test function (a standard bivariate Gaussian distribution) and two field applications, the synthetic IC-Fault and PUNQ-S3 models.

8.3.1 Bivariate Gaussian distribution application

All the uncertainty quantification methods were initially applied to a bivariate Gaussian distribution function. The aim was to investigate the performance of the algorithms on a well-known distribution function for which the true probability distribution is known.

8.3.1.1 Function description

The multivariate Gaussian distribution is the statistical distribution with probability density function of:

$$P(x|\bar{x},\Sigma) = \frac{1}{\sqrt{(2\pi)^2}\sqrt{|\Sigma|}} e^{-\frac{1}{2}(x-\bar{x})^T\Sigma^{-1}(x-\bar{x})} \tag{8.9}$$

where x is the vector of variables having the multivariate Gaussian distribution, $\bar{x}$ is the mean vector, $\boldsymbol{\Sigma}$ is the covariance matrix of the multivariate Gaussian distribution and it must be symmetric and positive definite; $|\boldsymbol{\Sigma}|$ is the determinant of the covariance matrix, superscript $\boldsymbol{T}$ represents transpose matrix function, and $\boldsymbol{\Sigma^{-1}}$ is the inverse of the covariance matrix.

If the mean and covariance matrix of the Gaussian probability distribution are unknown, the misfit (log likelihood) function for a single observation x would be:

$$\mathrm{Ln(L)} = -\frac{\mathrm{k}}{2}\ln(2\pi) - \frac{1}{2}\ln|\boldsymbol{\Sigma}| - \frac{1}{2}(\mathrm{x}-\bar{\mathrm{x}})^{\mathrm{T}}\Sigma^{-1}(\mathrm{x}-\bar{\mathrm{x}}) \qquad (8.10)$$

where k is the number of variables.

The bivariate Gaussian probability density function (PDF) has several useful and elegant properties and, for this reason, and as it is suitable for three dimensional visualisation, it is a commonly employed model in many statistical analyses. Here, a 2-dimensional case was assumed, bivariate Gaussian distribution, where $\bar{\boldsymbol{x}} = (\mathbf{0}, \mathbf{0})$ and $\boldsymbol{\Sigma} = \begin{bmatrix} \mathbf{1} & \mathbf{0.5} \\ \mathbf{0.5} & \mathbf{1} \end{bmatrix}$.

An ensemble of 3000 models (x vectors) have been generated using the same process normally used for history matching. i.e. two uniform variables were taken, both in the range of (-10,10); then a random initial population of 300 models was created and evaluated the models for their negative log likelihood value (i.e. misfit), as in equation (8.10). For the next generation, the best 200 solutions were selected and, using the chosen evolutionary algorithm (here HEDA), 100 new models were generated. The process was repeated for 27 generations so that a total of 3000 models were obtained. Figure 8.3 shows the inertia diversity and the convergence of the misfit boxplots for the ensemble. As the figure shows, after scanning the parameter space, the search algorithm has converged to a close neighbourhood of the global optimum (p1=0, p2=0). This is also shown by the 3d scatterplot and density map misfit/parameters (Figure 8.4).

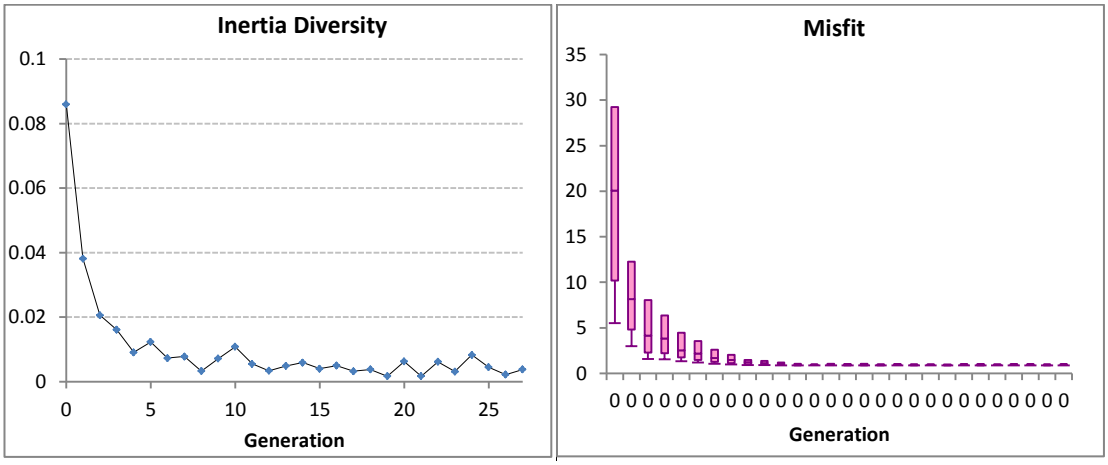


Figure 8.3: Inertia diversity (left) and boxplot of misfit (right) for 2500 models sampled by HEDA in bivariate Gaussian distribution application (two parameters p1 and p2). The search algorithm has converged to the global optimum.

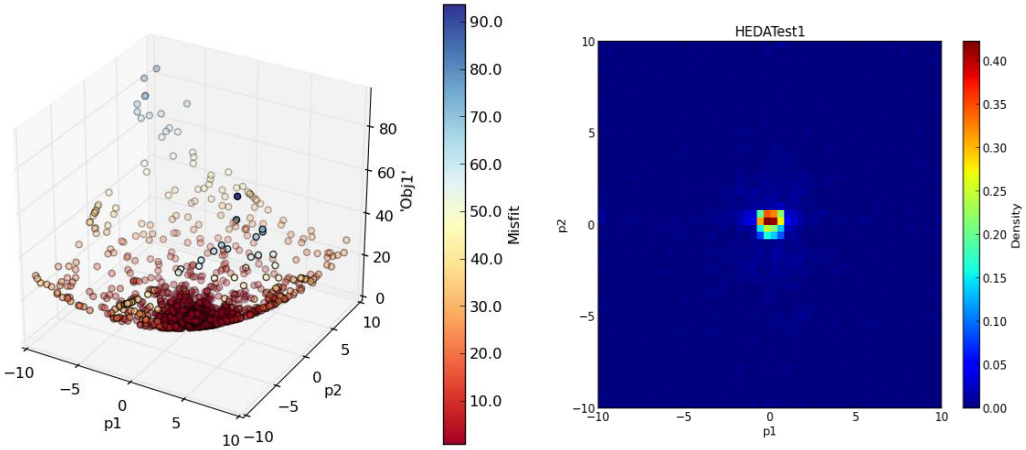


Figure 8.4: 3D scatterplot (left) and density map (right) of misfit and parameter values for 2500 models sampled by HEDA in bivariate Gaussian distribution application (two parameters p1 and p2). The search algorithm has converged to the global optimum.

8.3.1.2 K-NN approximation

We determined the best *k* value in K-NN approximation of the 3000 model ensemble, using the adaptive strategy explained in the K-NN section. For various *k* values, the mean squared error (MSE) was calculated and plotted on a bar chart (Figure 8.5). It can be seen that if *k=4*, the MSE does not decrease significantly. Thus adding more complexity to the approximation by increasing k does not improve the approximation significantly. Therefore *k=4* was selected for the later experiments.

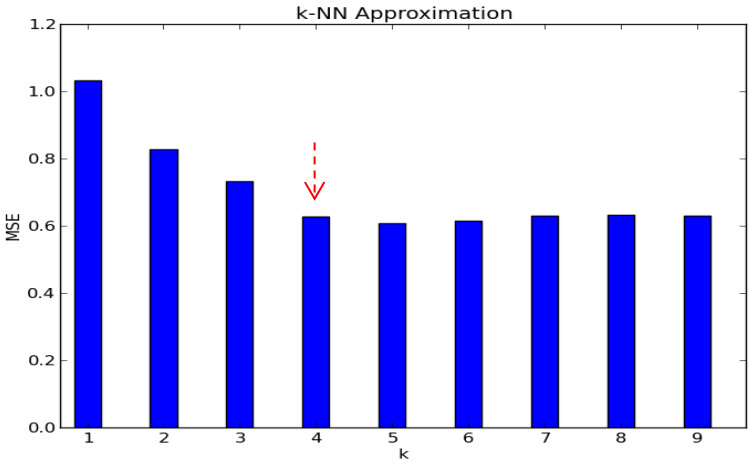


Figure 8.5: MSE versus k in K-NN approximation of ensemble obtained by the search algorithm for bivariate Gaussian probability density function. K=4 is best choice for the approximation.

8.3.1.3 Database results for bivariate Gaussian distribution

For the bivariate Gaussian distribution, a database of 150,000 samples was created by drawing random samples from the assumed parameter region of (-10, 10) and for each sample the misfit (negative log-likelihood) was calculated, as in equation (8.10). Figure

8.6 shows the uniform-like sampling density and misfit landscape of these 150,000 random samples in the bivariate Gaussian distribution. Since the sampling is uniform, the true probability density can be estimated by the likelihood, and hence by the misfit surface.

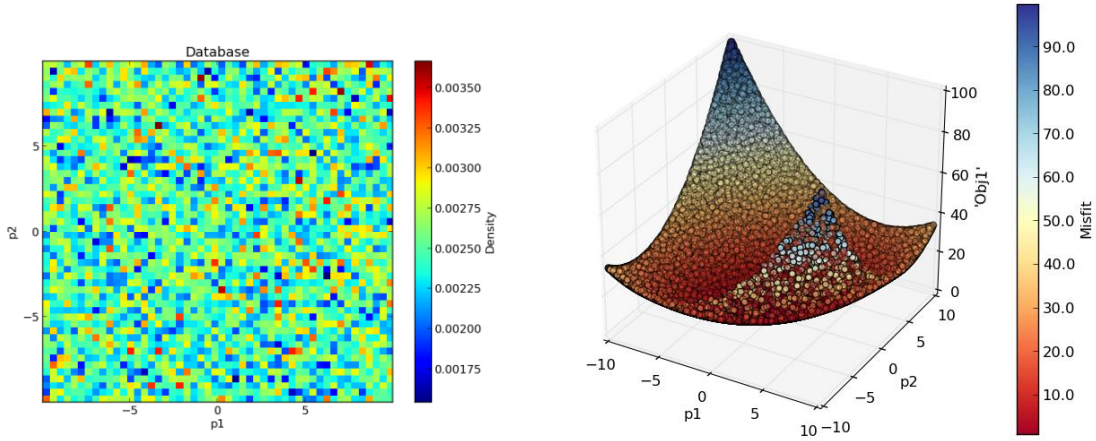


Figure 8.6: Heat maps (left) and 3D scatterplot (right) showing sampling density and misfit (negative log-likelihood) of 150,000 random samples in database of bivariate Gaussian distribution.

8.3.1.4 Estimation of probability distribution function

The ensemble of 3000 models, each model with a corresponding misfit value obtained by the HEDA was used for MCMC sampling of 150,000 new models, estimating the probability density of the bivariate Gaussian distribution.

We sampled the posterior probability distribution of the bivariate Gaussian distribution using two studied algorithms, NAB and AMH. For both algorithms, a single walk from the best misfit model was performed. NAB uses neighbourhood approximation (n=1), while AMH uses K-NN approximation with k=4. Figure 8.7 demonstrates the heat-map of sampling density, which estimates the probability density function. As the figure shows, AMH better estimates the peak and the correlated elliptic shape than NAB, when compared to the shape estimated by the database results.

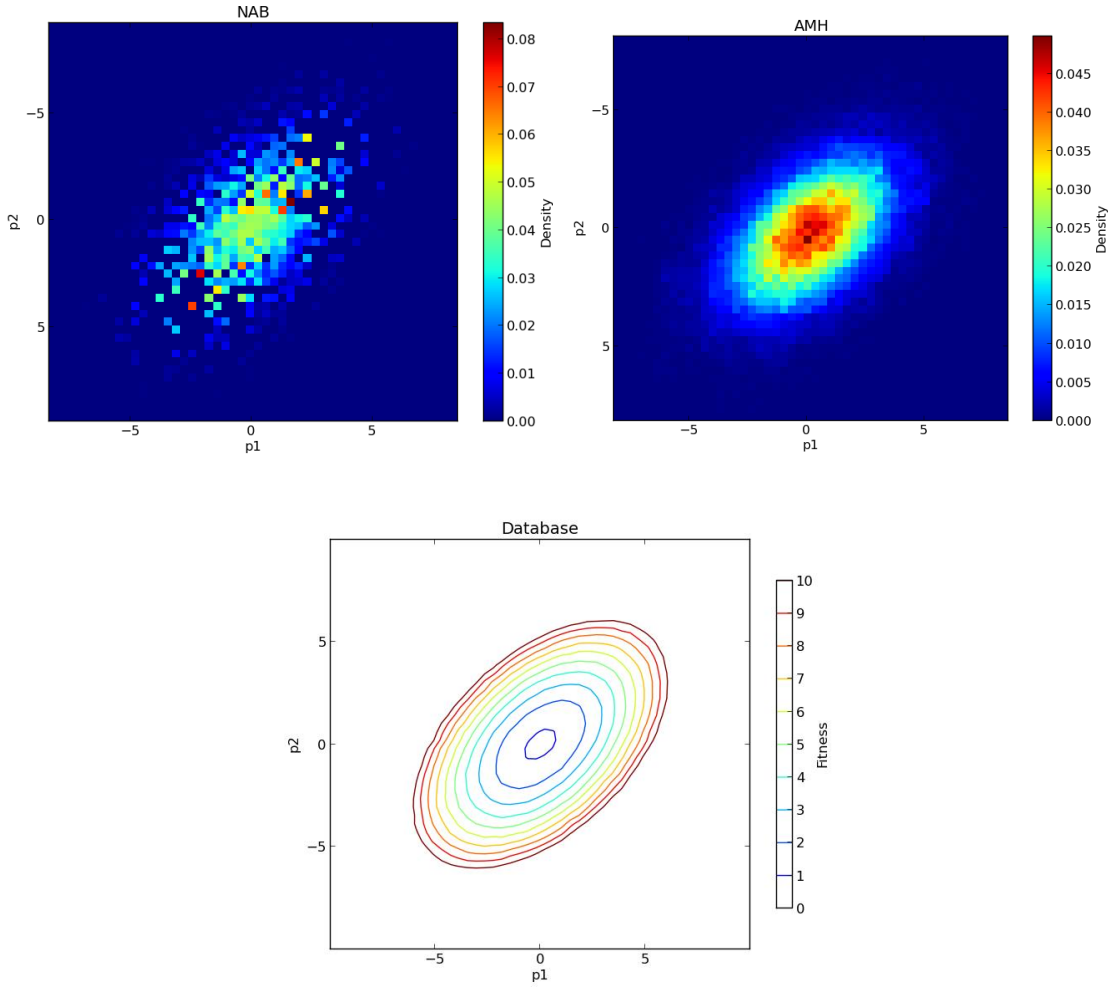


Figure 8.7: The estimated (heat-map) density of sampling in bivariate Gaussian probability density function by NAB (left) and AMH (right) and the true (contour-map) obtained for the database (bottom). AMH better captured the peak and smooth, correlated, elliptic shape of the true surface.

8.3.1.5 MCMC sampling convergence

The predictive parameter for uncertainty quantification was defined as the sum of the two variables (p1+p2) in the bivariate Gaussian distribution. The CDF of this variable was estimated using NAB and AMH sampling algorithms and reported in this section.

Figure 8.8 shows the convergence of the predictive parameter's moving average during the MCMC sampling of 150,000 samples by each of the algorithms. As the figure shows, AMH has converged better to the true value of the predictive parameter (zero). It should be mentioned that samples were not collected during the burn-in period.

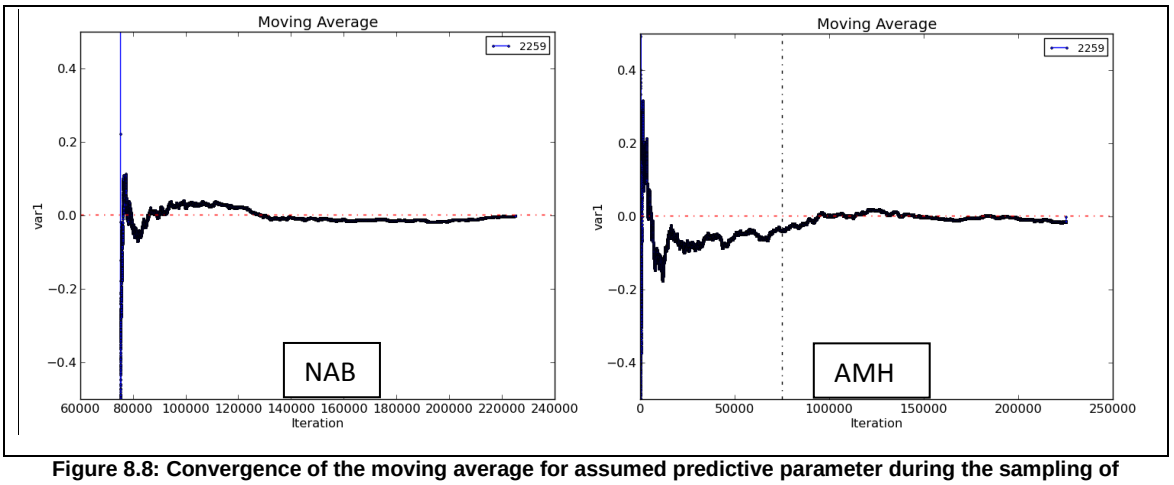


Figure 8.8: Convergence of the moving average for assumed predictive parameter during the sampling of 150,000 samples by NAB (left) and AMH (right). Both algorithms performed a single walk, starting from the best misfit model (2259). The dotted vertical line in AMH and the vertical blue line in NAB show the burn-in period (75,000 samples). The burned-in samples were not available for NAB, thus are not shown. The horizontal line shows the true value of the predictive parameter. After the burn-in period, AMH gets stable quicker compared to NAB.

We also looked at the trace-plots. Trace-plots show the value of parameters throughout the sampling. Another important chart is the autocorrelation function (ACF). It is the cross-correlation of a parameter trace-plot with itself, which shows the similarity between samples as a function of the sample separation (lag) between them. The trace-plots of the bivariate Gaussian distribution variables (p1 and p2) are shown in Figure 8.9 for NAB and in Figure 8.10 for AMH. One can see that both algorithms sample similar parameter ranges.

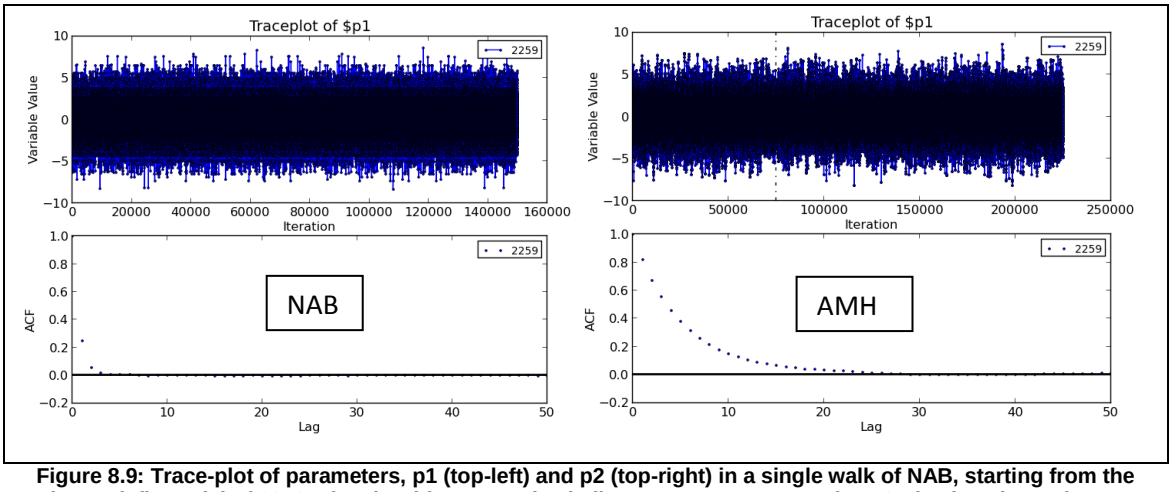


Figure 8.9: Trace-plot of parameters, p1 (top-left) and p2 (top-right) in a single walk of NAB, starting from the best misfit model of 2259. The algorithms sample similar parameter ranges. The ACF is also shown for p1 (bottom-left) and p2 (bottom-right), which becomes zero after 5 lags in NAB and 20 lags inAMH.

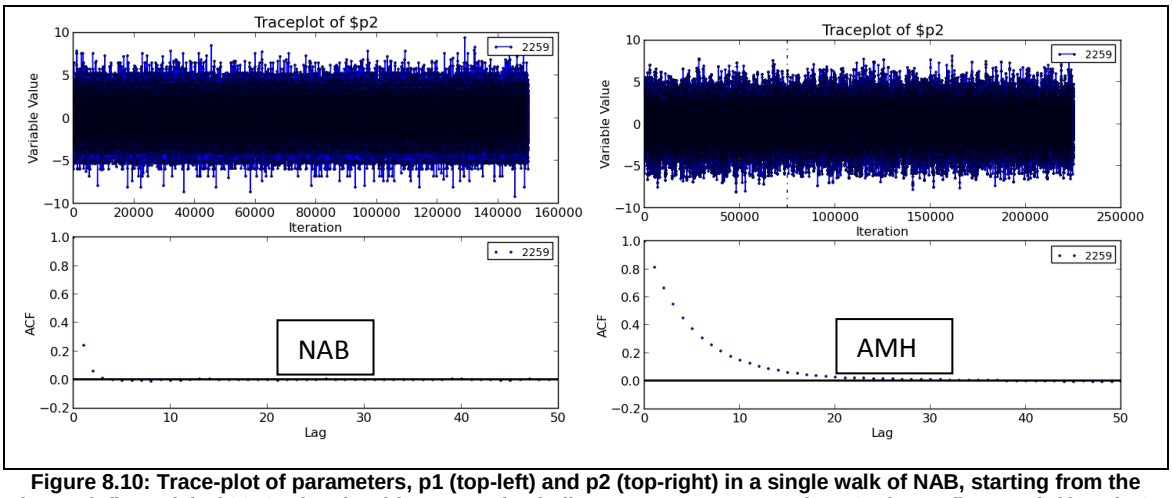


Figure 8.10: Trace-plot of parameters, p1 (top-left) and p2 (top-right) in a single walk of NAB, starting from the best misfit model of 2259. The algorithms sample similar parameter ranges. The ACF for p1 (bottom-left) and p2 (bottom-right) becomes zero after 5 lags in NAB and 20 lags in AMH.

8.3.1.6 Comparison with database results

The last figure in this section demonstrates the CDF calibration curve of the predictive parameter's (p1+p2) for NAB and AMH versus the database results (Figure 8.11). One can see from the figure that both algorithms (NAB and AMH) estimated the CDF very close to the database results.

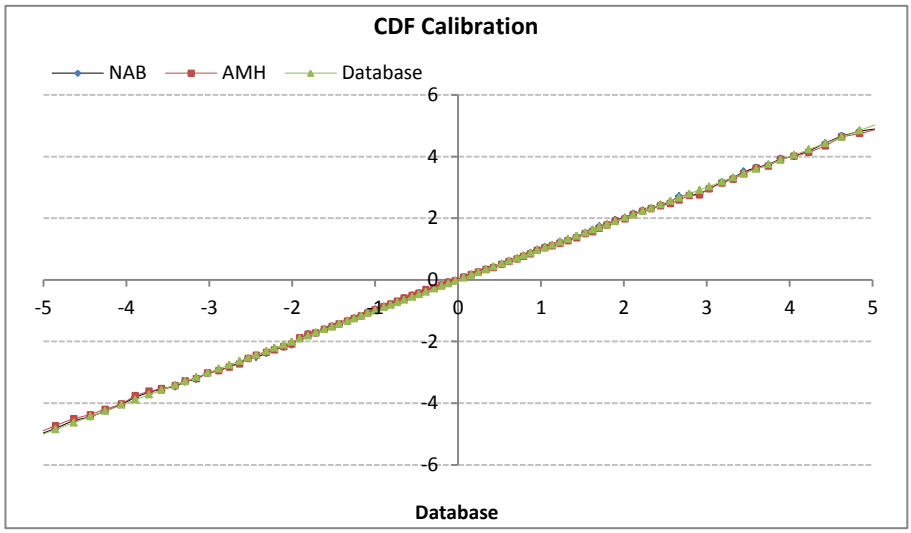


Figure 8.11: CDF calibration curves of predictive parameter's (p1+p2) for NAB and AMH versus database results. NAB and AMH both obtained true CDF for the database results.

8.3.2 IC-Fault model application

Following the testing of NAB and AHM on the bivariate Gaussian test function, we applied them to uncertainty quantification of the IC-Fault model. The IC-Fault model is

a synthetic reservoir case with a complicated response surface and many local minima. It was selected since it has a truth case and a database composed of a large number of forward simulation models (~160,000) sampled using a normal Monte Carlo algorithm, i.e. different random combinations of three uncertainty parameters were taken, and a forward simulation was done for each sample (Tavassoli et al., 2004). We take this database as the reference case for the comparative study of NAB and AMH.

In chapters 5 and 7, we used different evolutionary algorithms for history matching of IC-Fault model. In this chapter, we use the ensemble obtained by these algorithms for resampling by MCMC-based algorithms to perform the uncertainty quantification of predictive parameter in IC-Fault model (FOPT).

Tavassoli et al. (2004) showed that the best history-matched model may not give the best forecast. This statement by itself reveals the importance of ensemble-based uncertainty quantification used in this chapter. A best history-matched case is a single case obtained within a specific timeframe and limited computational resources, thus it is not guaranteed to be the global solution and the forecast from such single model is not usually robust.

In this chapter, we use ensemble-based uncertainty quantification methods to obtain robust predictions for IC-Fault model. As discussed earlier, history matching is an inverse problem with non-unique solutions. The presence of local minima in the misfit landscape of IC-Fault model is an indication of presence of multiple acceptably history-matched models, which are sufficiently different from each other. A robust forecast takes into account all these models. This is ensured by using uncertainty quantification techniques which involves resampling an ensemble of models obtained in history matching.

One can quantify the uncertainty around each of the misfit components. However here we try to quantify the uncertainty of a selected predictive parameter, total oil production of the field (FOPT). FOPT is the most important prediction of the reservoir performance, and it is the basis for many reservoir development decisions.

If we analyse the database for models with misfit under 25 (Figure 8.12), one can see a complex twisting, ribbon-like shape of the low misfit models in parameter search space.

The green dashed line in the parallel coordinate chart and the green star in the 3D scatter chart show the truth case.

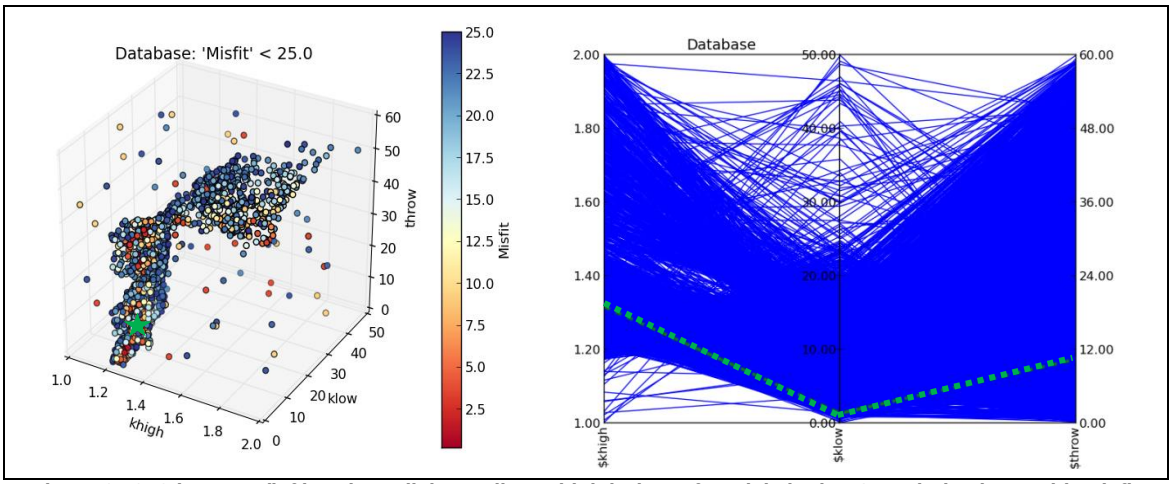


Figure 8.12: 3d scatter (left) and parallel coordinate (right) plots of models in the IC-Fault database with misfit under 25.0. The green dashed line in the parallel coordinate chart shows the truth case.

As discussed earlier in this chapter, we perform uncertainty quantification of prediction parameters using MCMC resampling of an ensemble of models obtained from a direct EA search. Therefore, it is necessary to study the effect of search algorithm choice and ensemble size on uncertainty quantification results.

8.3.2.1 Search algorithms

As discussed in the chapter 3 to 7, in history matching, different EAs perform search on the uncertain variable space differently. i.e. some algorithms are by nature more explorative (e.g. GA) while others are more exploitative (e.g. EDAs). Although, one can tune EAs for their optimum control parameters manually or using explicit adaptive algorithms (e.g. AHEDA in Chapter 7), so that, they could achieve a more balanced exploration and exploitation search.

In current chapter, we selected two direct search algorithms for history matching of the IC-Fault model, simulated binary GA (SBGA) and incremental histogram-based EDA (iHEDA). For more details on these algorithms refer to Chapter 5. Regardless of the search algorithm used to obtain the ensemble, a high level of ensemble diversity is required to perform a robust resampling by MCMC algorithms. SBGA is implemented with simulated binary crossover and mutation operators, which respectively ensure exploitation and exploration in the search if properly set by the control parameters.

As any other EDA, iHEDA has strong exploitative properties due to the use of probabilistic model. Therefore, as discussed in Chapter 5, if iHEDA used in the steep minima problems, it may get trapped in local minima. This was shown by previous tests on IC-Fault (Chapter 5). In such situations, the mutation operator is the most important operator which can help algorithms to escape from the local minima.

Thus, to improve the diversity of ensemble, we equipped iHEDA with a mutation operator and in current chapter will be referred to as imHEDA. In each generation of imHEDA, a set of children are generated from the incremental histogram model of the parent solutions (see sections 5.3.1 and 5.3.3), then the generated solutions undertake a mutation crossover according the procedure described in the section 5.2.2 of Chapter 5.

Both algorithms (SBGA and imHEDA) were tuned and forced to start from the same initial population. They were run two times exactly, with same initial population and tuning parameters. Control parameters used in two trials of the algorithms are given in the following table.

Table 8.1: Control parameters of the algorithms in IC-Fault application.

Parameter/ Algorithm	Size of initial population	Number of parents	Number of children	Number of generations	Number of bins	Learning rate	Mutation probability
imHEDA	30	20	10	47	20	0.7	0.3
SBGA	30	10	10	47	---	---	0.5

Minimum misfit, convergence, and diversity results are shown in Figure 8.13. Although the same initial population and control parameters are used, minimum misfit, convergence and diversity behave differently in the two trials of each algorithm, due to their stochastic nature.

With different misfit and diversity results, one cannot expect the same ensemble of models, even in two trials of the same algorithm with different seed numbers. SBGA and imHEDA explore the search space differently; to examine that, we looked at the models with misfit under 25.0 in the ensembles obtained by imHEDA (Figure 8.14 and Figure 8.15) and SBGA (Figure 8.16 and Figure 8.17).

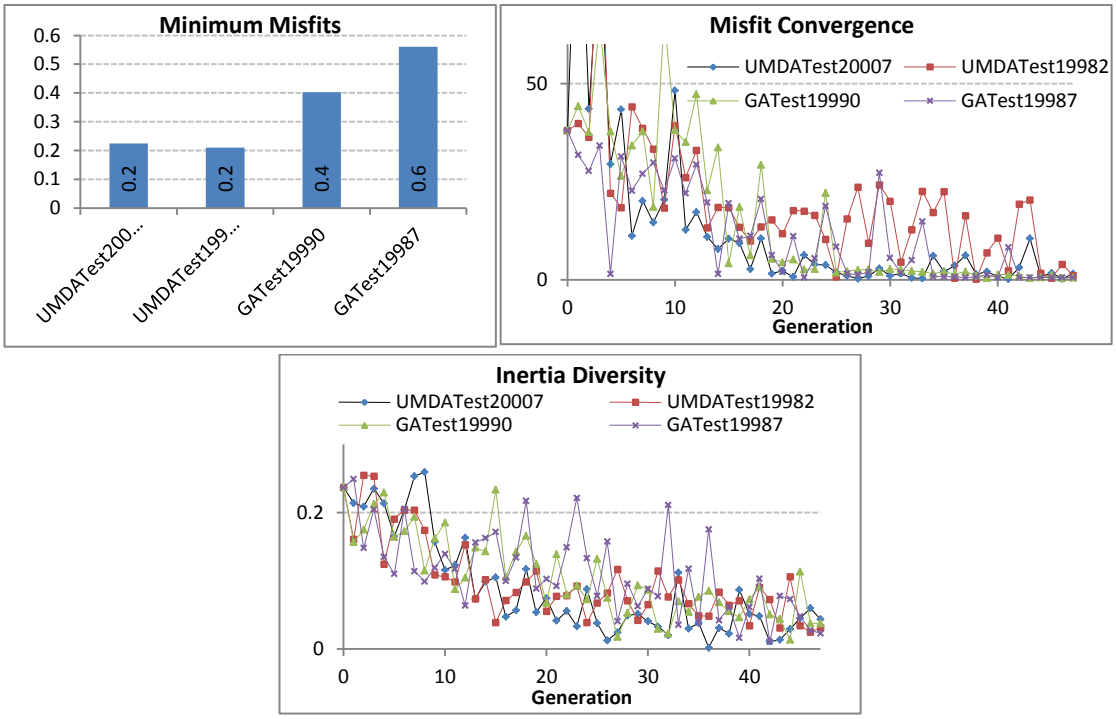


Figure 8.13: Minimum misfit (top-left), misfit convergence (top-right), and inertia diversity (below) plots for two trials of SBGA and two trials of imHEDA; differenttrials relate to a different seed numbers.

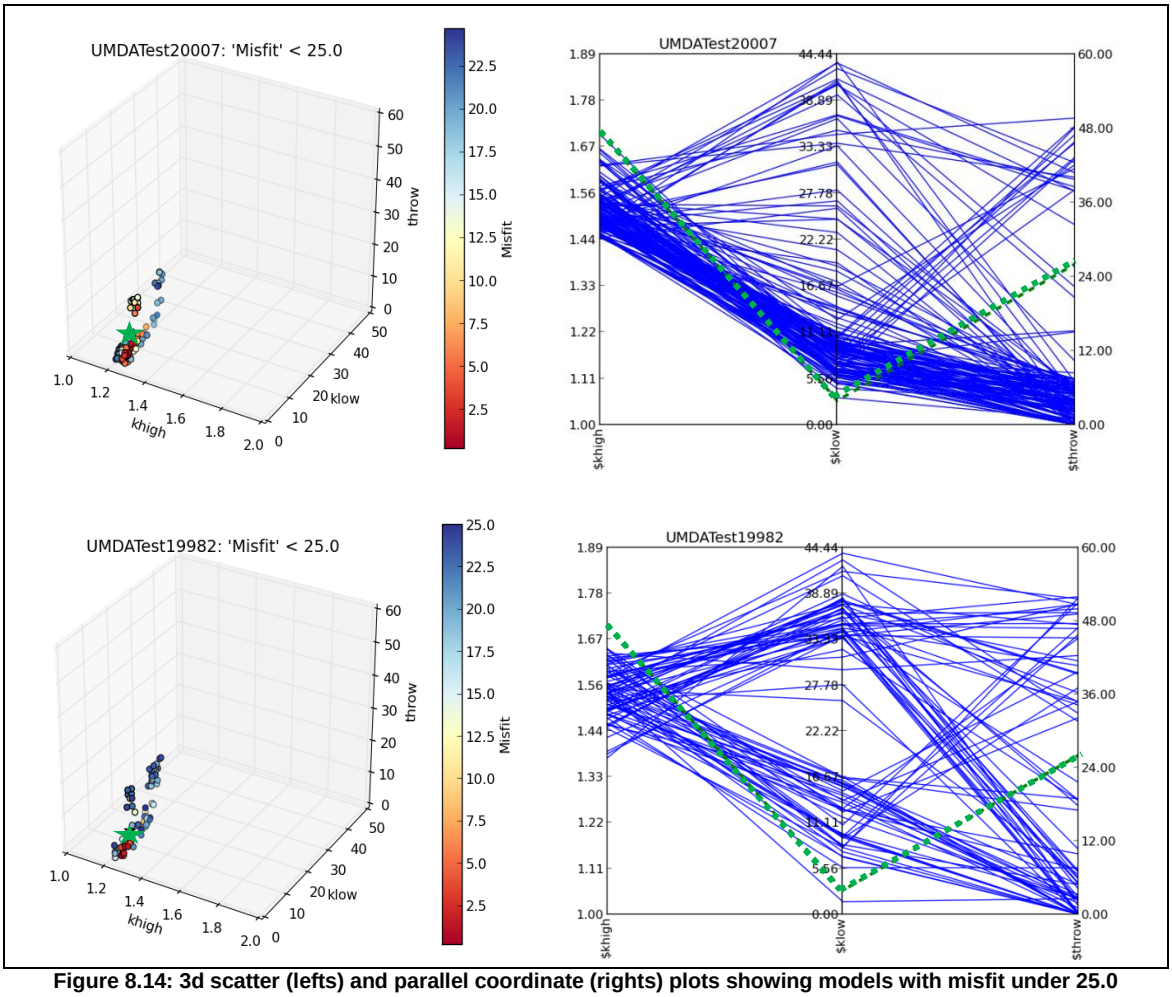


Figure 8.14: 3d scatter (lefts) and parallel coordinate (rights) plots showing models with misfit under 25.0 obtained by two trials (top and bottom) of imHEDA on IC-Fault model.

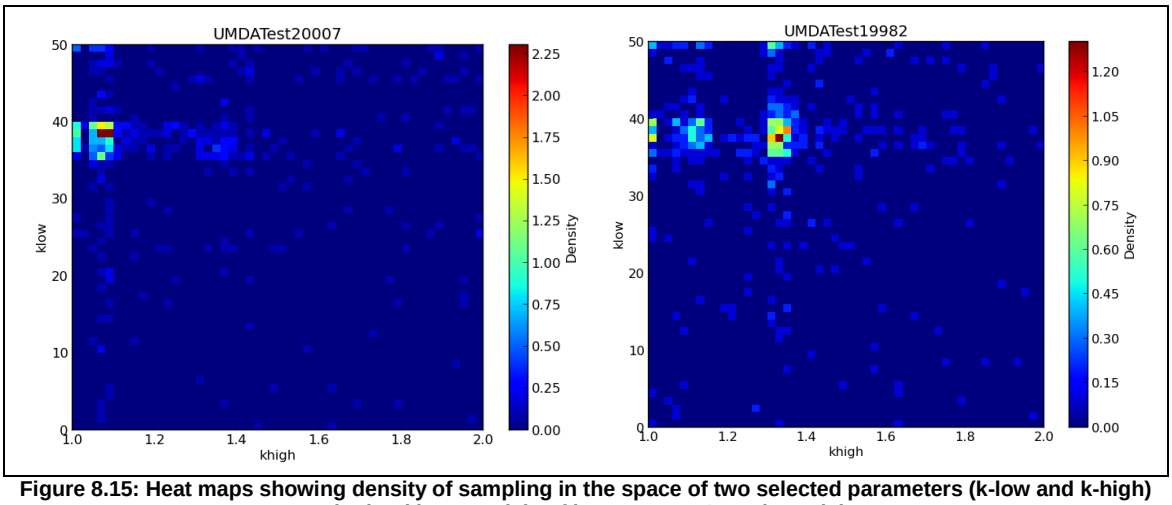


Figure 8.15: Heat maps showing density of sampling in the space of two selected parameters (k-low and k-high) obtained by two trials of imHEDA on IC-Fault model.

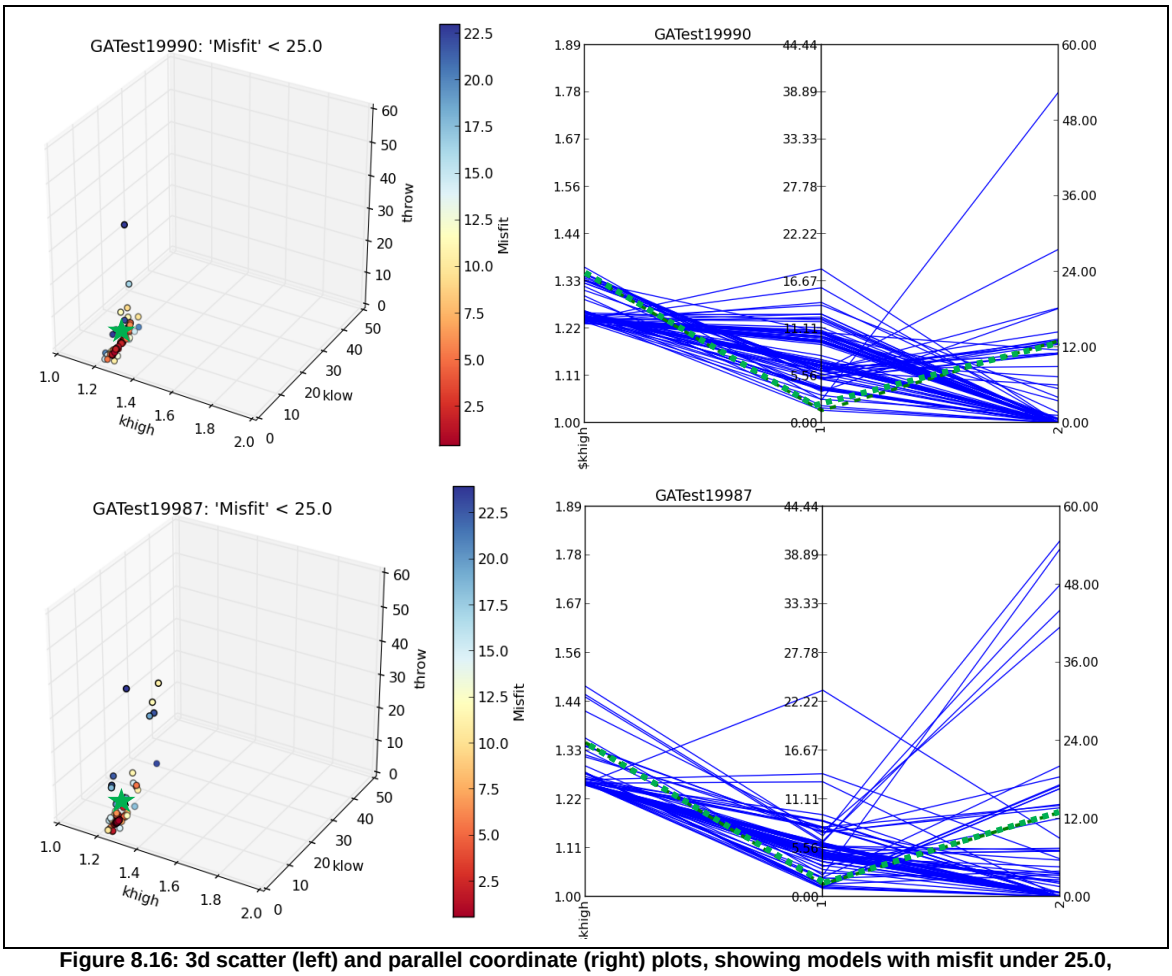


Figure 8.16: 3d scatter (left) and parallel coordinate (right) plots, showing models with misfit under 25.0, obtained by two trials of SBGA on IC-Fault model.

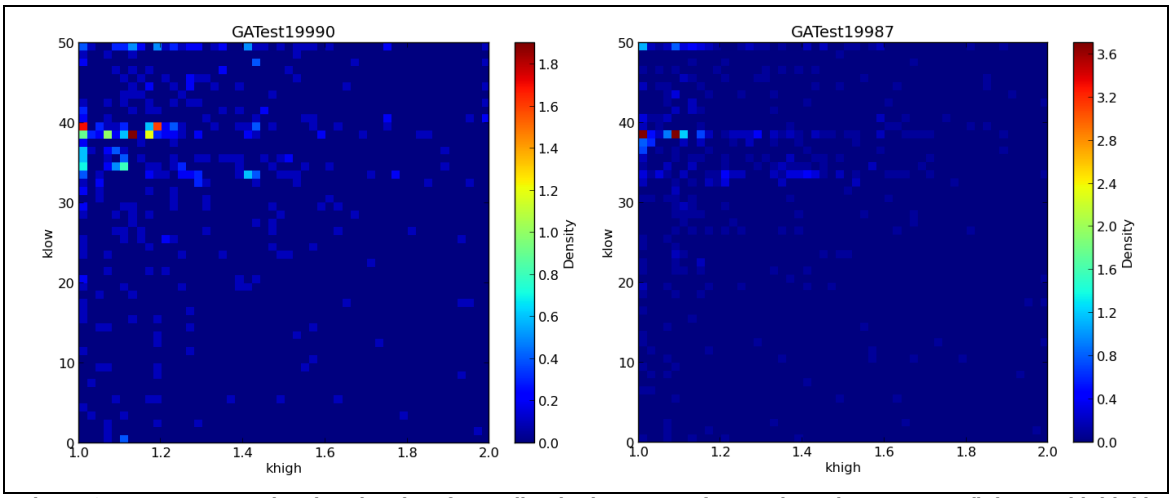


Figure 8.17: Heat maps showing density of sampling in the space of two selected parameters (k-low and k-high) obtained with by trials of SBGA on IC-Fault.

One can see that ensembles obtained by two trials of same algorithm are different in terms of both well-fitting models and density of the sampling. Hence, one should expect different results for the Bayesian inference of ensembles obtained by those trials, regardless of the algorithm used for uncertainty quantification. This reveals the importance of the direct search algorithm used for model calibration and generating the ensemble. Hence, the first factor affecting the performance of ensemble-based uncertainty quantification methods is the ensemble itself.

8.3.2.2 Ensemble size

A sensitivity study of ensemble size on the sampling algorithms was performed. For this purpose, two imHEDA experiments were run to obtain two different ensemble sizes. The first ensemble was composed of 500 models, obtained by 47 generations of imHEDA, each containing 10 solutions. The second ensemble was created by another imHEDA experiment, in which 2500 models were generated in 247 generations, each containing 10 solutions. The distribution of models with misfit under 25.0 in these two ensembles is shown in Figure 8.18 and Figure 8.19.

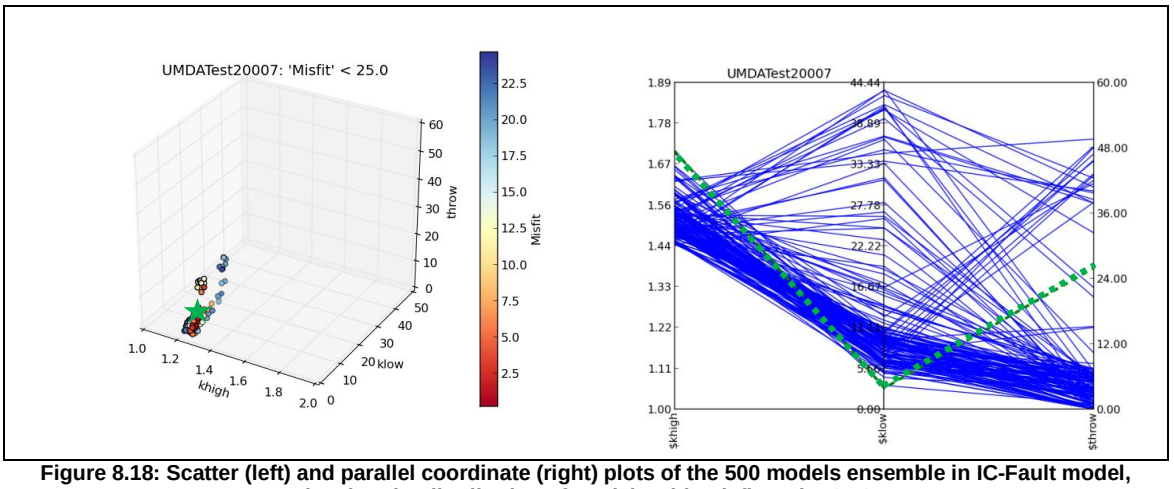


Figure 8.18: Scatter (left) and parallel coordinate (right) plots of the 500 models ensemble in IC-Fault model, showing the distribution of models with misfit under 25.0.

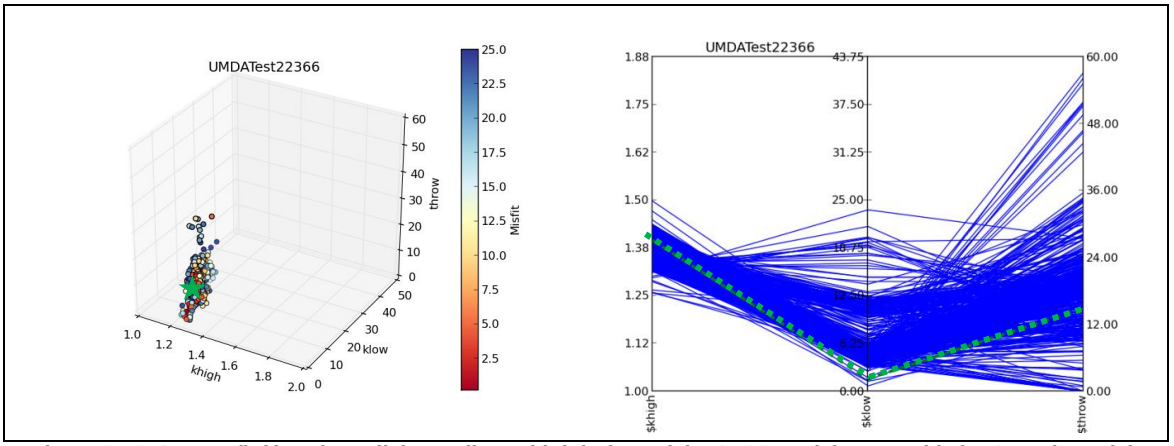


Figure 8.19: Scatter (left) and parallel coordinate (right) plots of the 2,500 models ensemble in IC-Fault model, showing the distribution of models with misfit under 25.0.

To study the effect of ensemble size on sampling algorithms, we used NAB for sampling 150,000 new samples from the above ensembles and the inference results from these two ensembles were compared, using the convergence of moving average and parameter traceplots.

Figure 8.20 shows two ensembles as picked by imHEDA and their corresponding 150,000 models sampled by NAB. From the figure, it is evident that the ensemble size and distribution of the models in the ensemble hugely affects the results of ensemble-based uncertainty quantification. The ensemble should be sufficiently large and diverse for the posterior sampling to capture all the regions of importance and achieve a reliable quantification of the uncertainty.

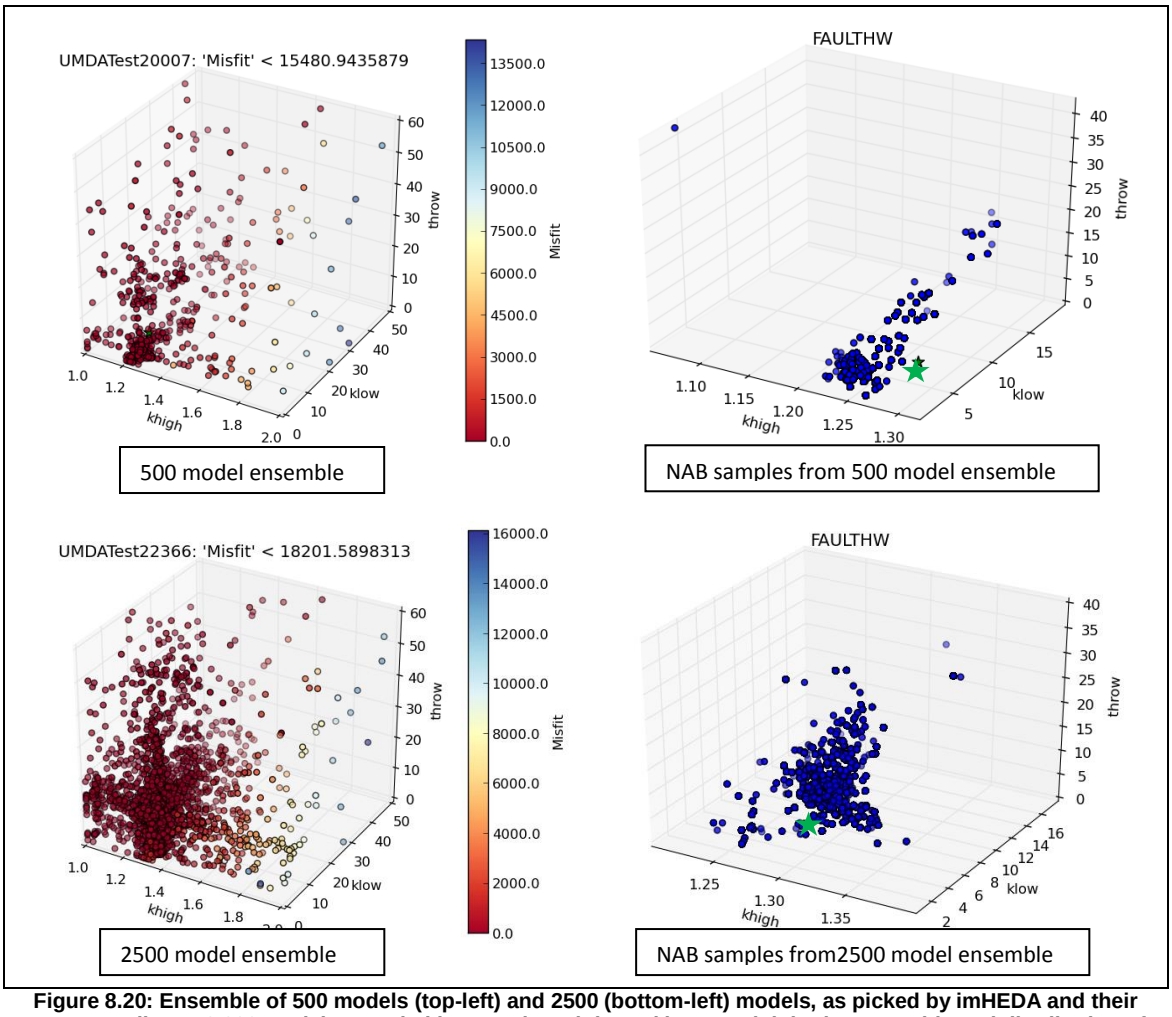


Figure 8.20: Ensemble of 500 models (top-left) and 2500 (bottom-left) models, as picked by imHEDA and their corresponding 150,000 models sampled by NAB (top-right and bottom-right). The ensemble and distribution of the models in the ensemble is the first important factor of ensemble-based uncertainty quantification.

Figure 8.21 shows the moving average of the predictive parameter (FOPT) using NAB sampling of these two ensembles. When results of two different ensemble sizes are compared with the prediction of the database of the IC-Fault model (P50 of 83,500 STB for FOPT after 3651 days of production), the figure from the ensemble of 2,500 models (84,500 STB), is less biased than that from the 500 models ensemble, which is 86,300 STB. The reason is a larger ensemble size better and more diversely covers the search space and approximation from such ensemble will be more robust.

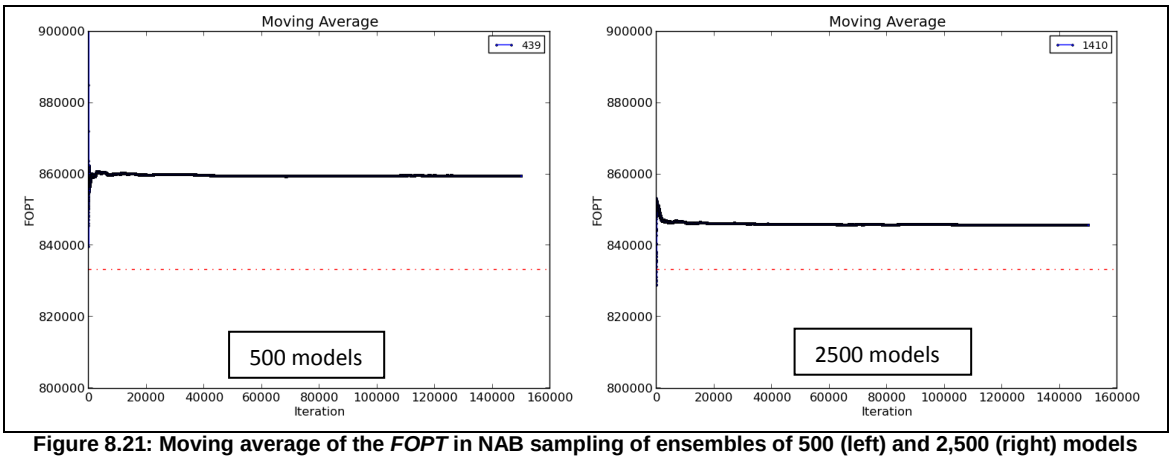


Figure 8.21: Moving average of the *FOPT* in NAB sampling of ensembles of 500 (left) and 2,500 (right) models generated by imHEDA. The red-dotted line shows P50 of Database. P50 prediction from the ensemble of 2,500 models is closer to that of the database compared to the ensemble of 500.

We also analysed the trace-plots of three parameters, *throw* (Figure 8.22), *K-high* (Figure 8.23), and *K-low* (Figure 8.24) and their autocorrelation function values for the two ensemble sizes of 500 and 2500 models. The trace-plot and autocorrelation function of the three parameters does not vary considerably between the two ensemble sizes, although the 2500 model ensemble shows a wider spread of parameter range for the *throw* and *K-high*.

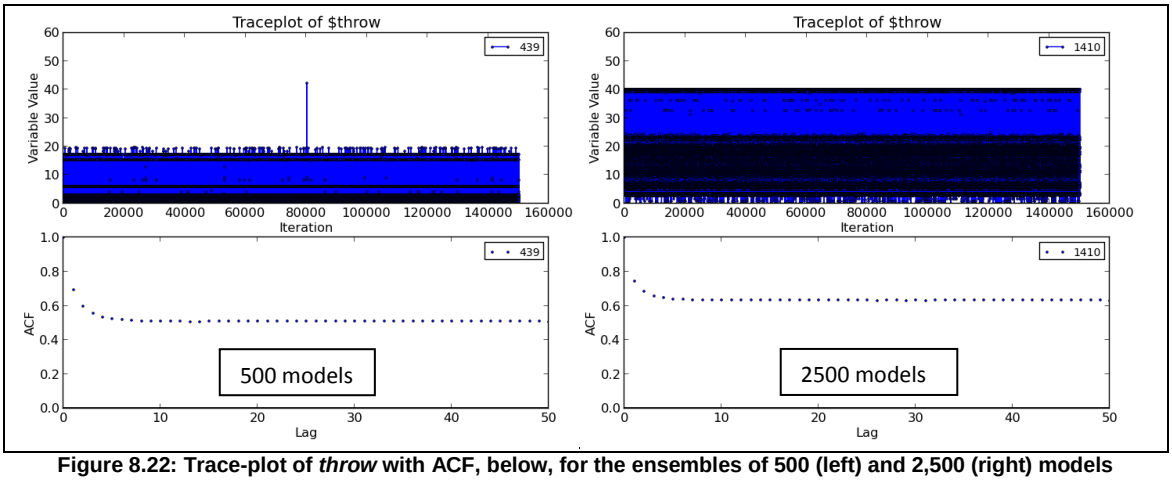


Figure 8.22: Trace-plot of *throw* with ACF, below, for the ensembles of 500 (left) and 2,500 (right) models generated by imHEDA. NAB sampling from the ensemble of 2,500 models has slightly wider parameter range and higher ACF than from the ensemble of 500 models.

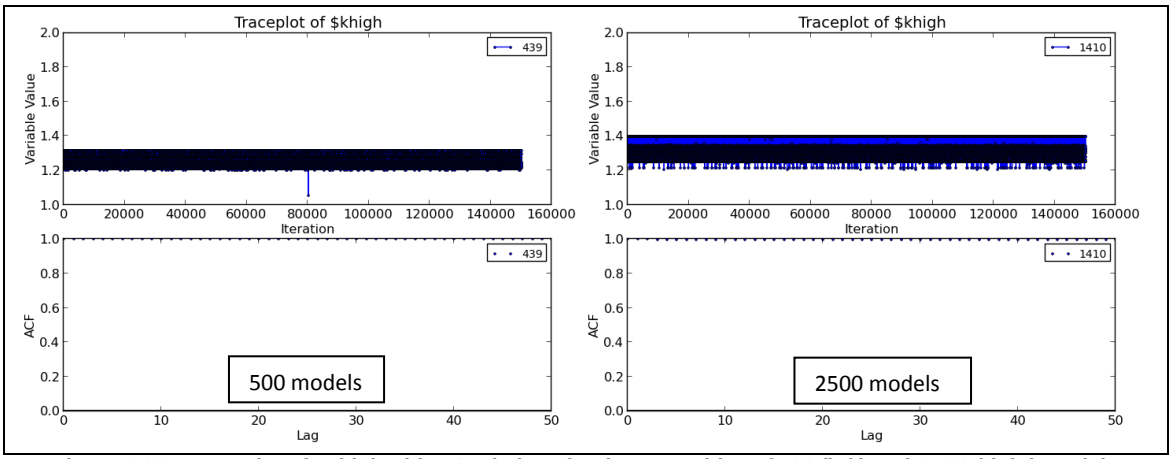


Figure 8.23: Trace-plot of *K-high* with ACF, below, for the ensembles of 500 (left) and 2,500 (right) models generated by imHEDA. NAB sampling from the ensemble of 2,500 models has wider parameter range than the ensemble of 500 models.

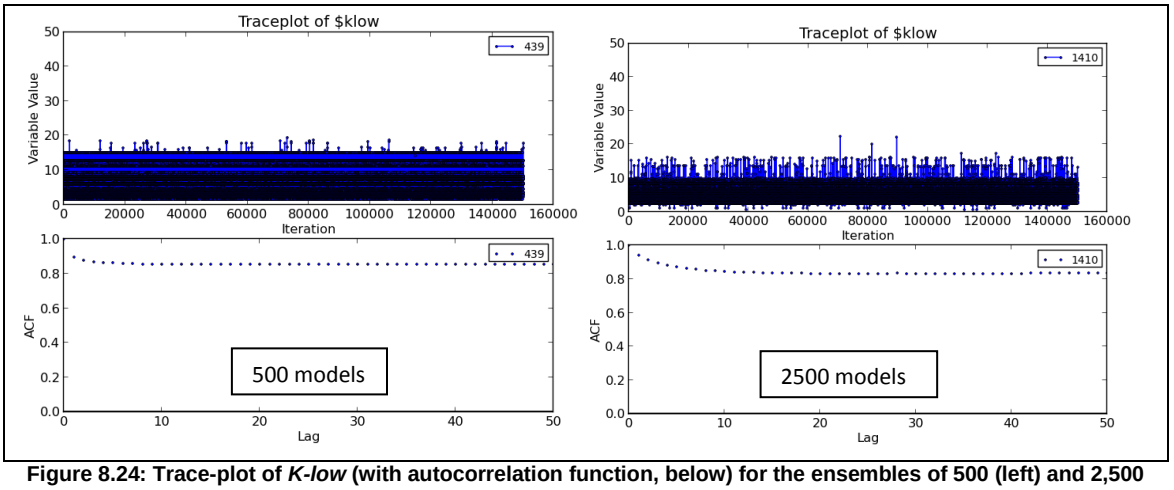


Figure 8.24: Trace-plot of *K-low* (with autocorrelation function, below) for the ensembles of 500 (left) and 2,500 (right) models generated by the search algorithm imHEDA. NAB sampling from the ensemble of 500 models has slightly wider parameter range and higher ACF than ensemble of 500 models.

Finally, we looked at the CDF calibration curves of the predictive parameter FOPT versus the database. As mentioned earlier, and as Figure 8.25 shows, in terms of the closeness of inference results to the database, the ensemble of 2500 models is a better choice than the ensemble of 500 models. In the subsequent experiments, we only use the ensemble with 2500 models, to minimise the effect of ensemble size on the performance of sampling algorithms.

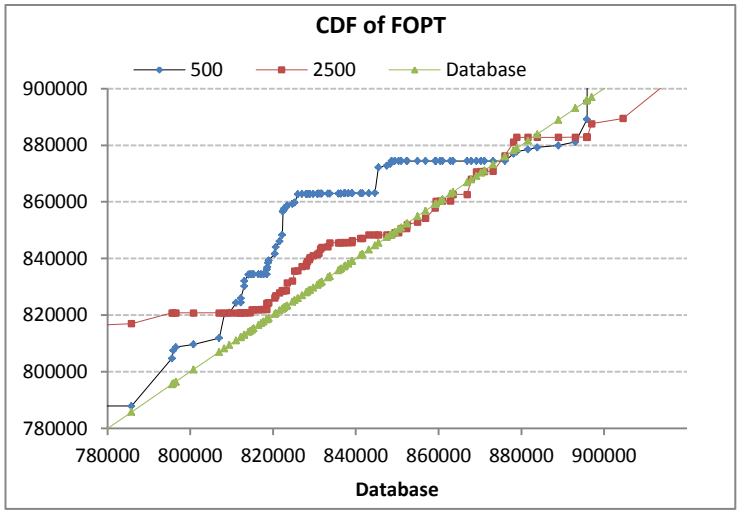


Figure 8.25: CDF ccalibration versus database curves of 500 and 2,500 model ensembles in IC-Fault model. The ensemble of 2500 models resulted in a prediction very close to the database.

8.3.2.3 K-NN approximation

For the IC-Fault model, MSE was plotted against various *k* values, when each model in the ensemble of 2500 models was approximated using its k nearest neighbours. As Figure 8.26 shows, *k=3* corresponds to the minimum MSE, and hence was used in the K-NN approximation. This was also confirmed by CDF calibration curves of FOPT in the IC-Fault application. Figure 8.27 shows results of AMH with different k values. As the figure shows, k=3 results in the closest CDF compared to the database.

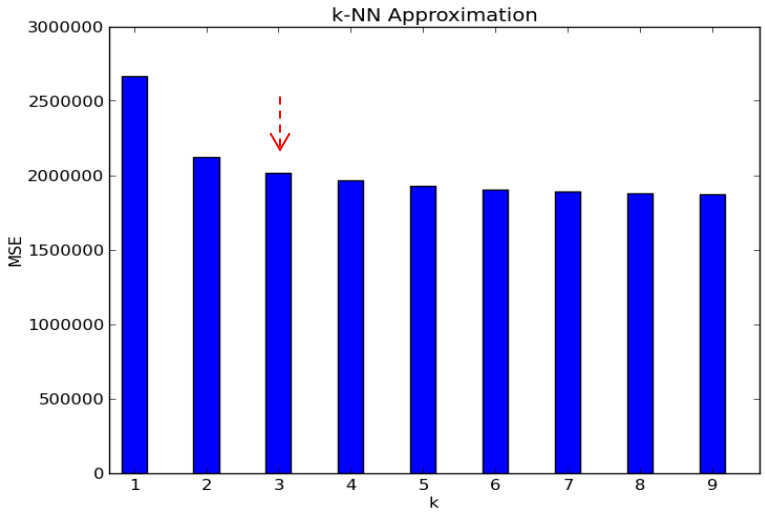


Figure 8.26: MSE versus k in K-NN approximation of ensemble obtained by the search algorithm for IC-Fault. k=3 is the best choice for the K-NN approximation.

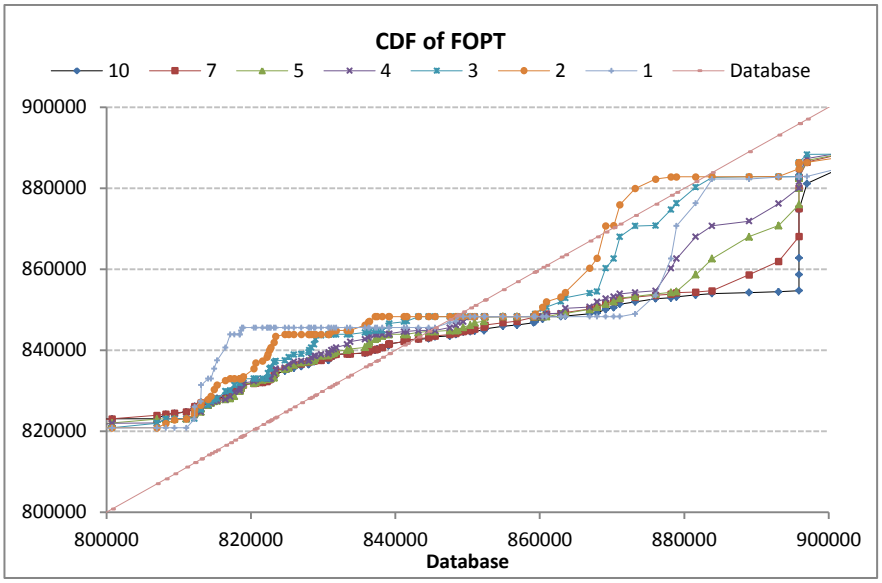


Figure 8.27: CDF calibration curves of FOPT in IC-Fault application, obtained by AHM with different k values in K-NN approximation. K=3 result is closest to the database.

8.3.2.4 Multiple walks

In the IC-Fault model application, two options were tested for choosing the starting points of multiple walks in AMH sampling. The options are either to select the starting points randomly or perform a probabilistic-distance (PD) clustering and take the best fitting model of each cluster as a starting point.

Figure 8.28 shows samples in parameter space, as sampled by the AMH algorithm from either random starting points (RD) or the best model of different clusters obtained by PD clustering.

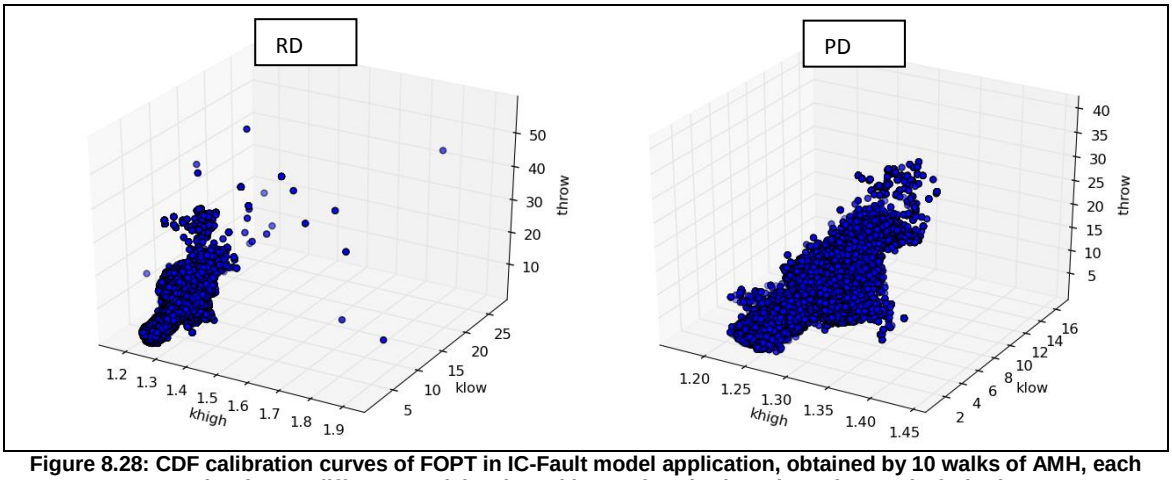


Figure 8.28: CDF calibration curves of FOPT in IC-Fault model application, obtained by 10 walks of AMH, each starting from a different model, selected by random (RD) or clustering analysis (PD). Clustering provides better starting points than the diverse sampling.

The trace-plots of the parameters were compared for two cases: random multiple walks and multiple walks from PD clustering. As shown in Figure 8.29, these plots and their respective ACFs were very similar for *K-high* and *K-low*. However for *throw* the trace-plots and autocorrelation functions were slightly improved.

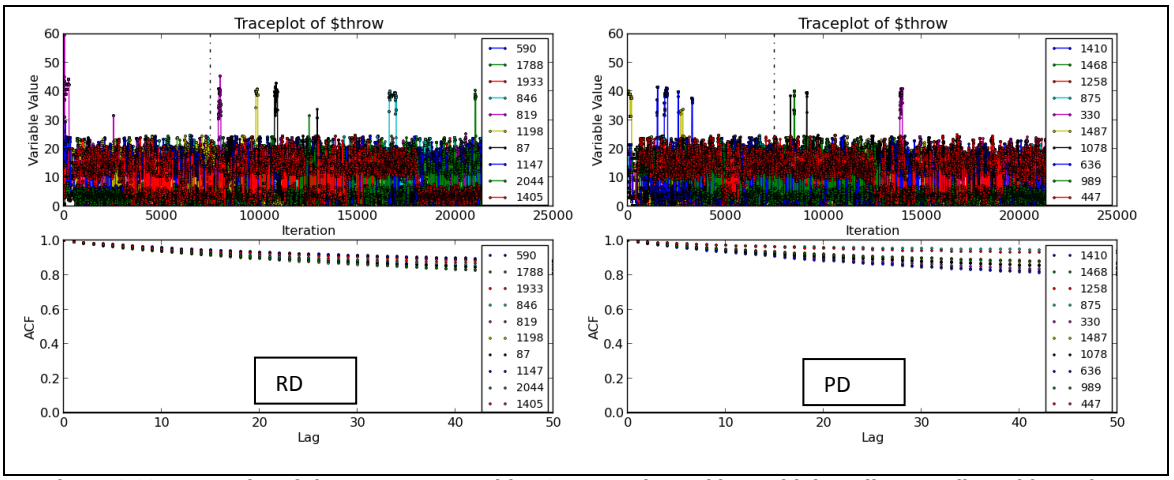


Figure 8.29: Trace-plot of *throw* parameter with ACF, as estimated by multiple walks sampling with random (left) and PD clustering (right) starting points. PD clustering starting points have resulted in slightly wider parameter range and lower ACF than random starting points.

Figure 8.30 shows CDF calibration curves of FOPT for these two options. One can see that starting points from PD clustering have improved the results, as they are closer to the database than those from random starting points.

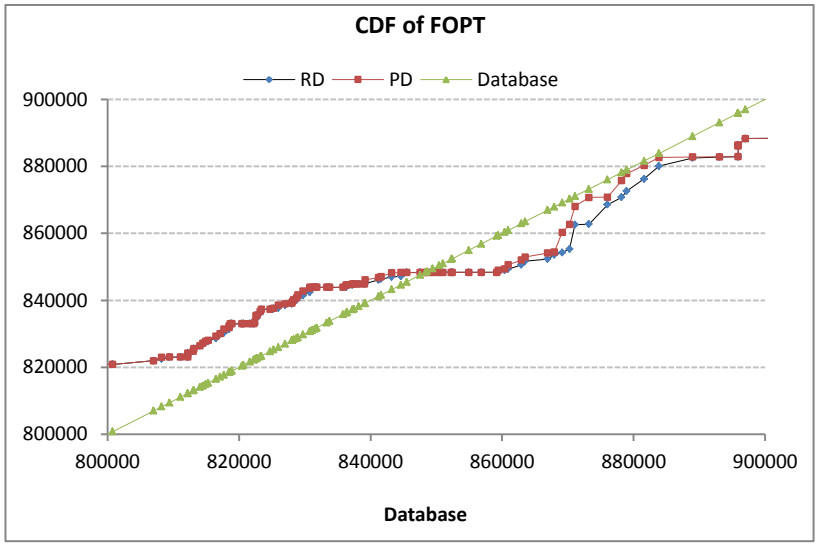


Figure 8.30: CDF calibration curves of FOPT in IC-Fault model application, obtained by 10 walks of AHM, each starting from a different model, selected by random (RD) or clustering analysis (PD). Clustering improves the results for uncertainty quantification, as the resulting CDF is closer to CDF obtained from the database.

8.3.2.5 Covariance matrix tuning

Initially, we used a fixed covariance matrix for the proposal distribution in M-H; then it was tuned by a simple trial and error process. The starting value for the covariance matrix could be the covariance matrix of the models in the ensemble. Then the covariance matrix was tuned by using a multiplication factor and by monitoring the acceptance rate.

Figure 8.31 shows the acceptance rate of M-H for different covariance matrix factors, which are multiplied to the initial covariance matrix of the ensemble in the IC-Fault model. Factors 0.01 and 0.005 show an acceptable range of acceptance rate (between 23-50%). In the figure, D is equal to $2.38^2/d$ and corresponds to the heuristic approach of Gelman et al. (1996), where d is the number of dimensions. In the IC-Fault model, d is equal to 3, thus D is equal to 1.8. Figure 8.32 shows CDF calibration curves of FOPT for different covariance factors in the IC-Fault model application. One can see that results obtained for 0.001 and 0.005 are closer to the database compared to those for other factors, although all the factors greater than 0.0001 have resulted in CDFs sufficiently close to the database. The covariance matrix factor of 0.005 is the best choice here, as it is supported by the acceptance rate and the CDF calibration plots both.

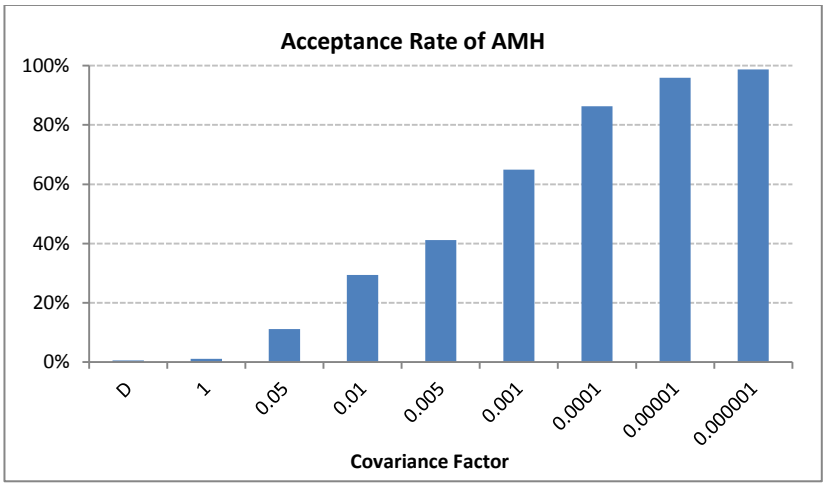


Figure 8.31: Acceptance rate of Metropolis-Hasting algorithm with multivariate Gaussian proposal distribution of mean current value and covariance of the ensemble multiplied, by a factor. Results for factors 0.01 and 0.005 fall in the acceptable range of acceptance rate (23% to 50%).

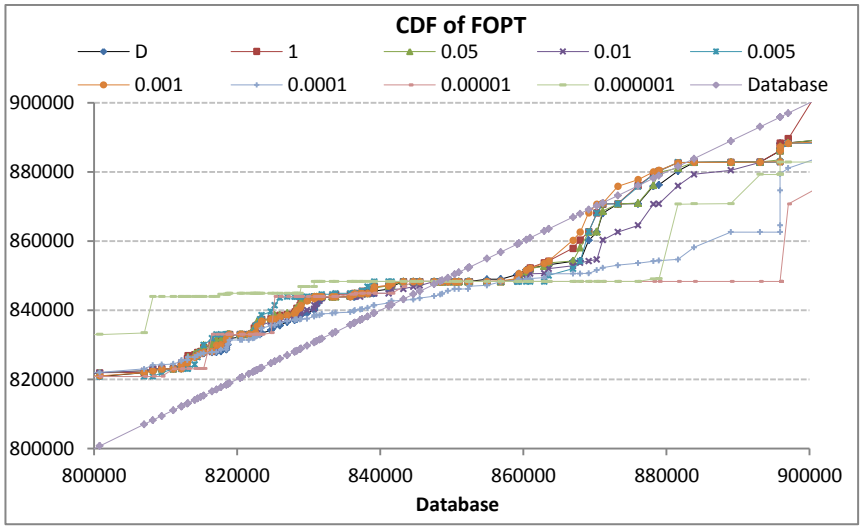


Figure 8.32: CDF calibration curves of FOPT in IC-Fault model application, obtained by M-H with initial covariance matrix of ensemble's covariance matrix and different covariance factors used for tuning. Results for 0.001 and 0.005 are closer to database than other factors.

Another option for the initial covariance matrix is to take the identity matrix *i* and then multiply it by a factor during the tuning. Figure 8.33 shows the acceptance rate of M-H for different covariance matrix factors, which are multiplied to initial covariance matrix of identity *i*. In this case, factors 0.0001 and 0.0005 show an acceptable range of acceptance rate (between 23-50%). Figure 8.32 shows CDF calibration curves of FOPT in which the result obtained for 1i matches the database results best.

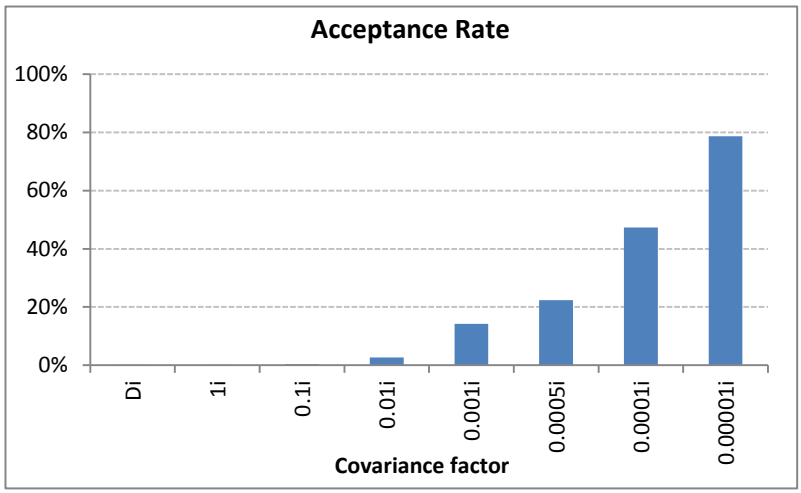


Figure 8.33: Acceptance rate of Metropolis-Hasting algorithm with the multivariate Gaussian proposal distribution of mean current value and covariance of ensemble, multiplied by a factor. Results for factors 0.01 and 0.005 fall in the acceptable range of acceptance rate (23% to 50%).

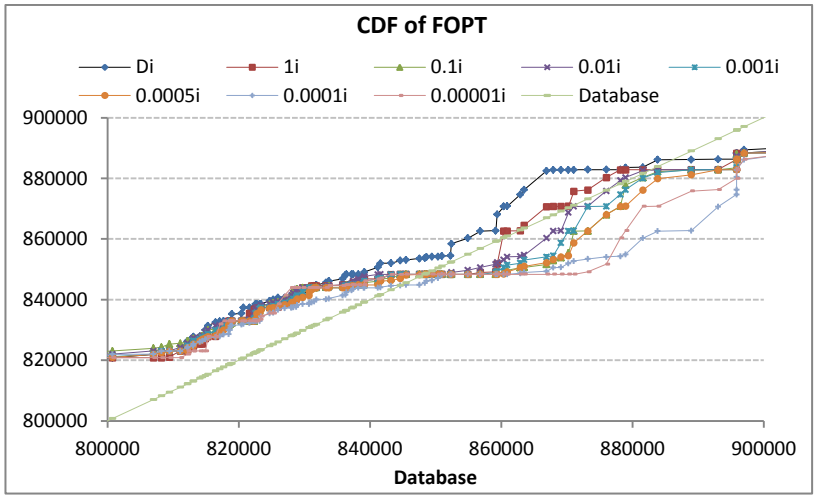


Figure 8.34: CDF calibration curves of FOPT in IC-Fault model application, obtained by M-H with initial covariance matrix of identity and different covariance factors used for tuning. Result for 1i matches database result best.

8.3.2.6 Adaptive Covariance matrix

In the previous section, two initial covariance matrix factors and several covariance factors for the multivariate Gaussian proposal distribution in M-H were tested. The results showed that the acceptance rate of M-H with fixed covariance matrix and the CDF obtained for the predictive parameter are highly sensitive to the choice of initial covariance matrix, as well as the covariance factor value used for tuning. Covariance matrix factor tuning is time-consuming and results obtained for a case cannot be applied to another.

To address this problem, an adaptive strategy is applied for the covariance matrix in multivariate Gaussian proposal distribution of AMH. The strategy tunes the covariance matrix during the sampling in the burn-in period, and hence provides the M-H sampling with a proper proposal distribution without the need for tuning. The adaptation is done in the burn-in period, by recalculating the covariance matrix from the last certain number of accepted samples (here 1000), as discussed in the Methodology section 8.2.4.

The initial covariance matrix can be chosen as the covariance of the ensemble multiplied by a factor. Figure 8.35 shows the acceptance rate of AMH for different covariance factors. In this case, the acceptance rate was not very sensitive, and regardless of the factor value, the acceptance rate was around 20% (between 13-26%). The results for the calibration curves (Figure 8.36) of covariance factors in AMH are

very close to each other and better fit the database compared to the fixed covariance matrix case.

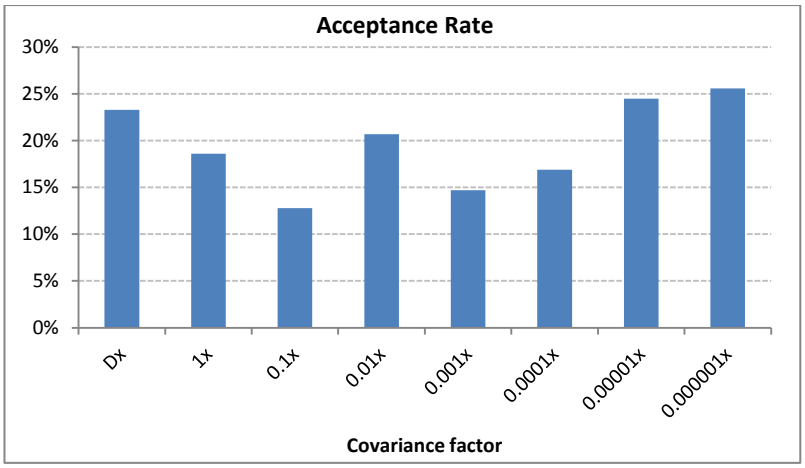


Figure 8.35: Acceptance rate of AMH with initial covariance matrix of ensemble and different covariance factors. The acceptance rates are all around 20%.

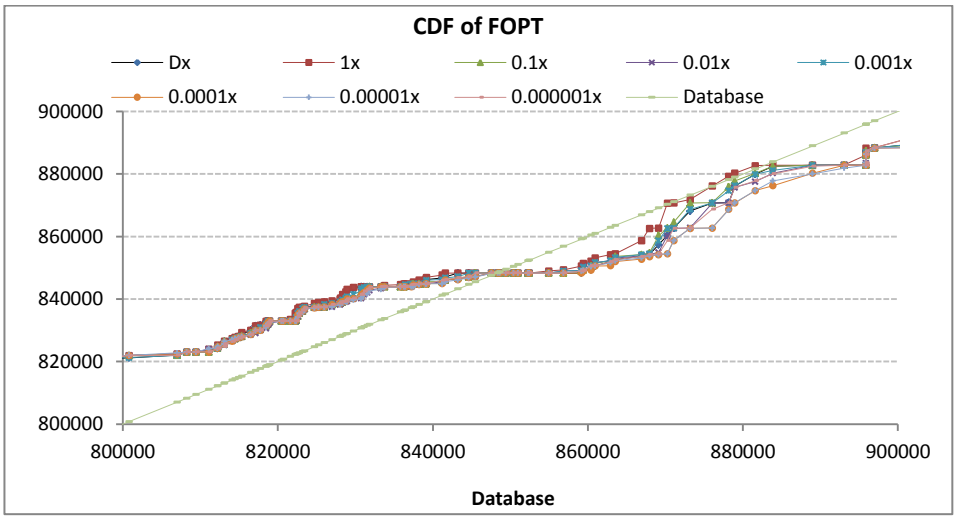


Figure 8.36: CDF calibration curves for FOPT in IC-Fault model application, obtained by AHM with initial covariance matrix of ensemble and different covariance factors. Results better match database than those from the fixed covariance (Figure 8.34).

Even if AMH starts from the identity covariance matrix, the acceptance rate, as shown in Figure 8.37, is around 20% (between 15-25%) and less sensitive to the covariance factor in M-H. However, results are very similar to AMH with initial covariance matrix of ensemble, which shows AMH is not sensitive to the choice of the initial covariance matrix (Figure 8.38).

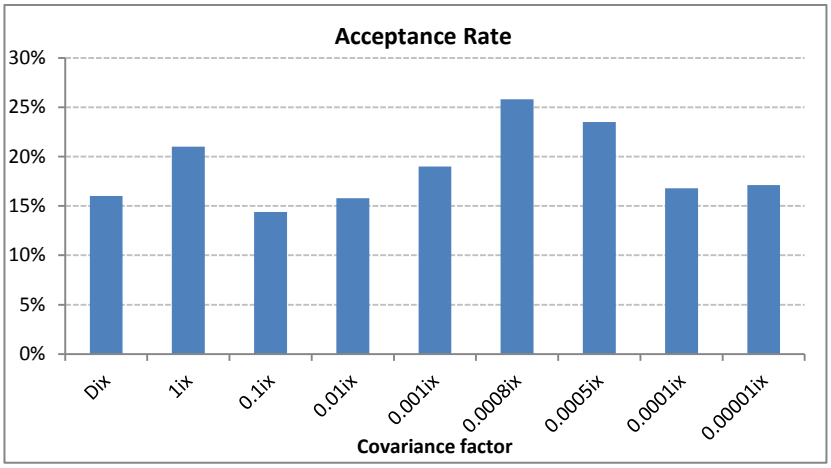


Figure 8.37: Acceptance rate of AMH with initial covariance matrix of identity and different covariance factors. The acceptance rates are all around 20%.

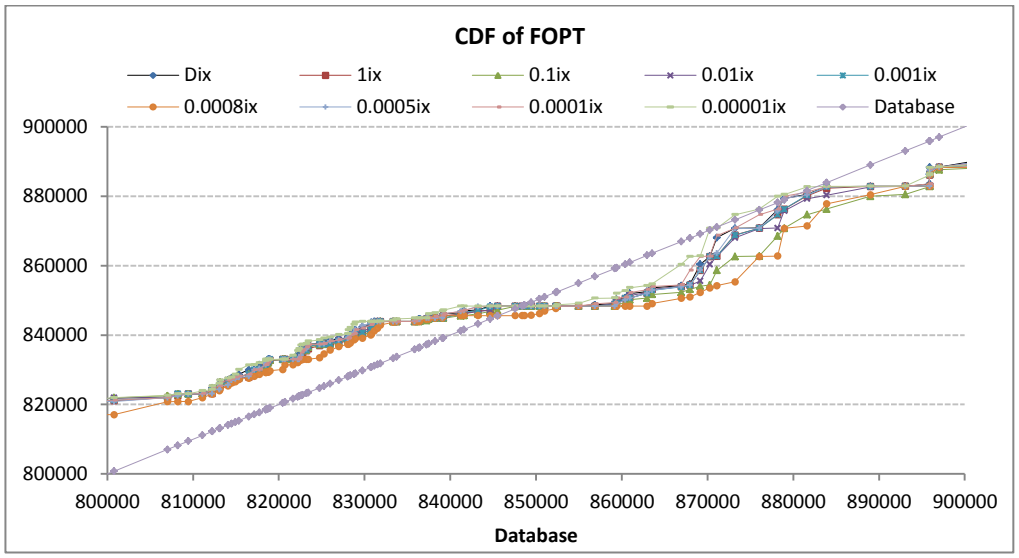


Figure 8.38: CDF calibration curves of FOPT in IC-Fault model application, obtained by AHM with initial covariance matrix of identity and different covariance factors. Results better match database than fixed covariance (Figure 8.34).

8.3.2.7 Comparison of convergence diagnosis and mixing

In this section, the results of MCMC sampling, including the convergence plots, moving average of FOPT and trace-plots of the uncertain parameters, are compared for a single MCMC sampling trial of each of the NAB and AMH algorithms. Both algorithms sample from the same ensemble of 2,500 models obtained by history matching, using the search algorithm imHEDA. AMH was set up with the adaptive proposal and an initial covariance value of the ensemble, and then adapted during the burn-in period.

Figure 8.39 shows the spatial distribution of 150,000 models sampled by two MCMC algorithms, NAB and AMH, in the ICF application. As NAB uses the nearest neighbour

approximation, it only resamples models in the ensemble. However, AMH uses K-NN (here k=3) approximation and, as the figure shows, it has resampled a much wider area in parameter space.

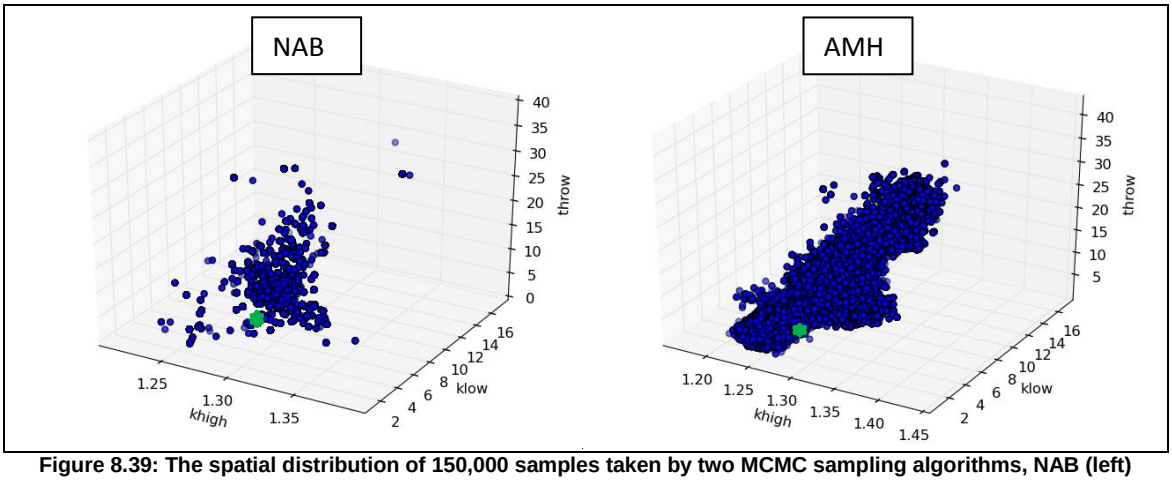


Figure 8.39: The spatial distribution of 150,000 samples taken by two MCMC sampling algorithms, NAB (left) and AMH (right) in IC-Fault model application. The truth case is shown with a green star. NAB has resampled the ensemble while AMH has sampled a much wider area.

Next we looked at the convergence of mean FOPT. Figure 8.40 demonstrates the results of moving average of the predictive parameter. The AMH plot shows the burn-in period with a horizontal dotted line, after which the algorithm has reached its stationary status. Both algorithms have converged to a similar value of FOPT, which is slightly higher than the database result, shown by the red-dotted line.

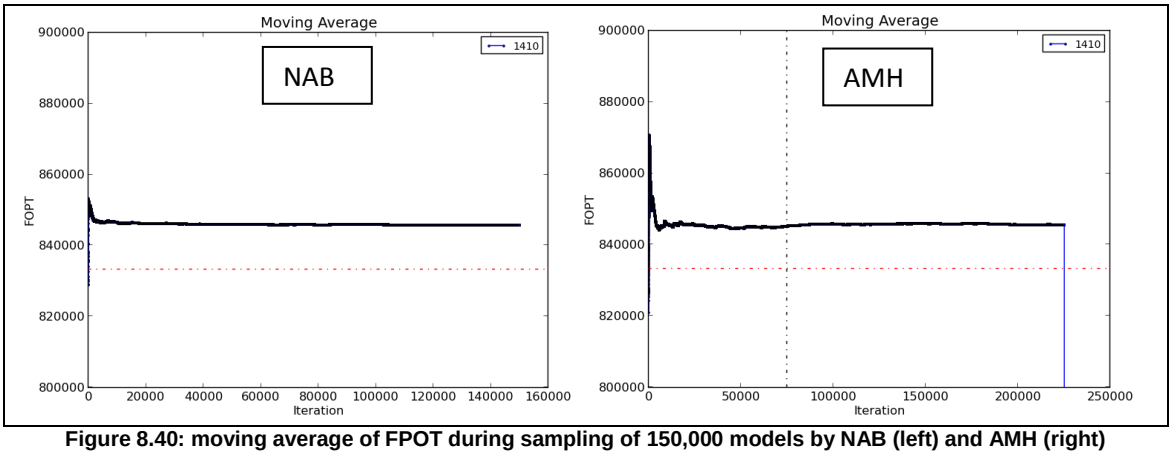


Figure 8.40: moving average of FPOT during sampling of 150,000 models by NAB (left) and AMH (right) algorithms in ICF application. The database is shown with a horizontal red-dotted line; the vertical black-dotted line shows the end of burn-in period in AMH. Both algorithms have converged to a similar value.

Figure 8.41, Figure 8.42, and Figure 8.43 show the trace-plot and ACF of three uncertainty parameters, *throw*, *K-high*, and *K-low* respectively. The figures show that

NAB has slightly obtained better parameter estimation with lower ACF, especially for *throw* and *K-low* parameters.

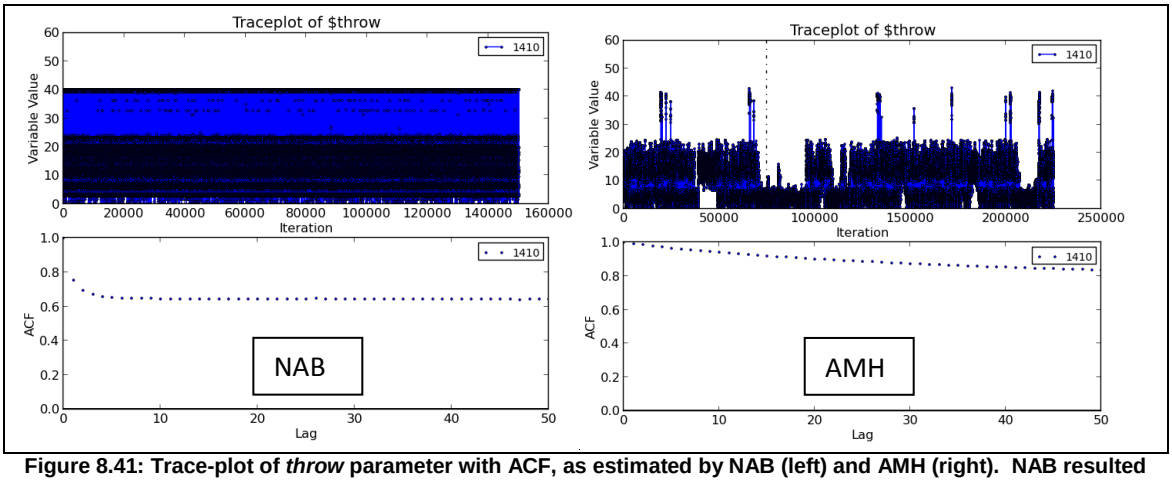


Figure 8.41: Trace-plot of *throw* parameter with ACF, as estimated by NAB (left) and AMH (right). NAB resulted in slightly wider parameter range and lower ACF than AMH.

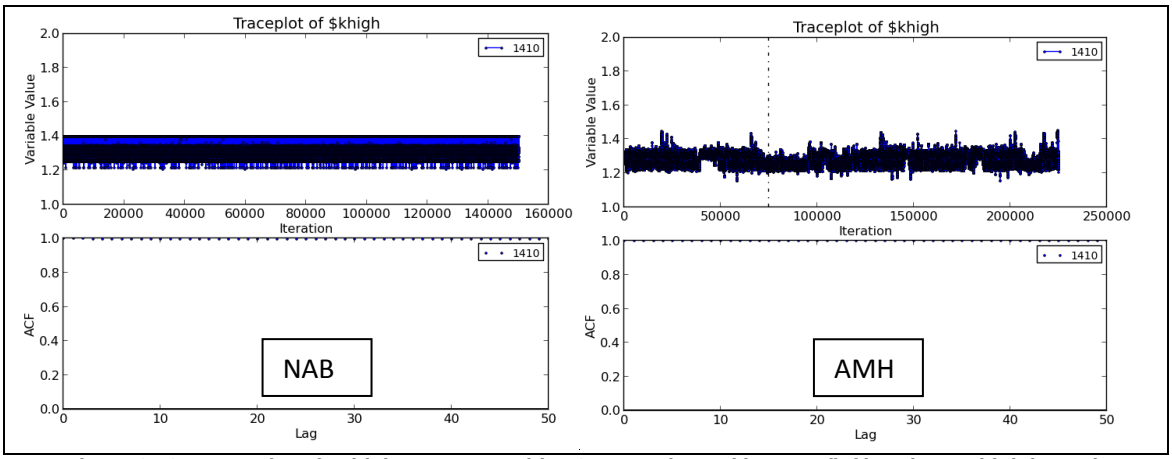


Figure 8.42: Trace-plot of *K-high* parameter with ACF as estimated by NAB (left) and AMH (right). Both algorithms resulted in very narrow parameter range and high ACF.

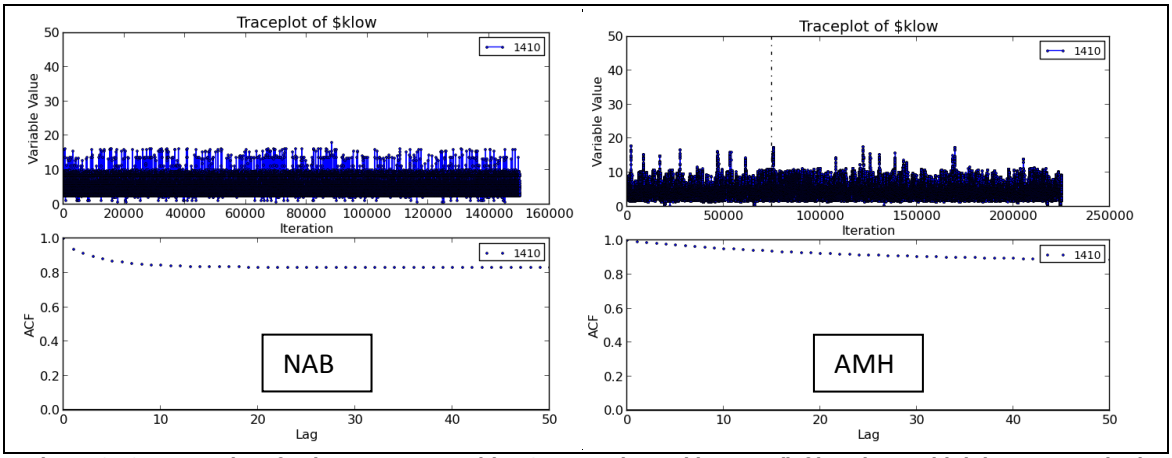


Figure 8.43: Trace-plot of *K-low* parameter with ACF as estimated by NAB (left) and AMH (right). NAB resulted in slightly wider parameter range and lower ACF than AMH.

Finally, Figure 8.44 shows the box-plot and CDF calibration versus the database plot of the ultimate FOPT, as obtained by the NAB and AMH algorithms. Although both algorithms have obtained credible intervals comparable to the database, one can see that, in this case, NAB has obtained predictions closer to the database compared to the AMH prediction.

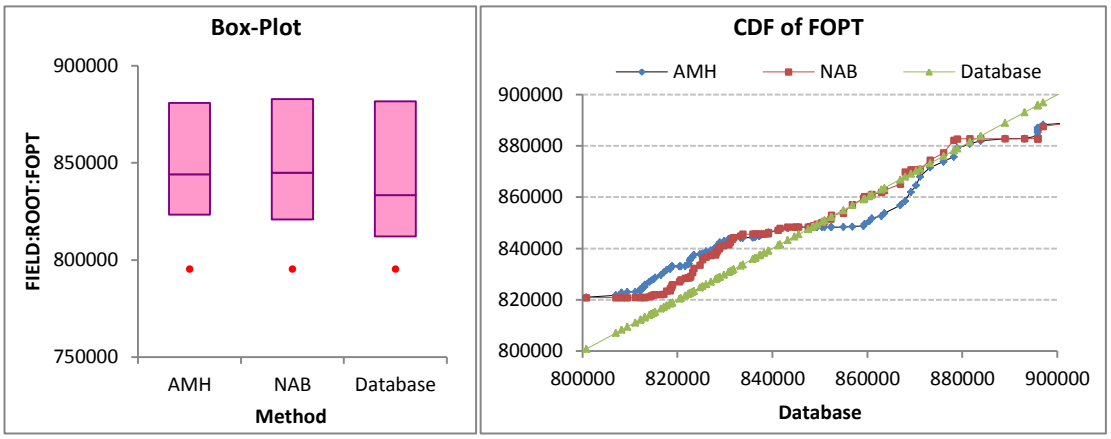


Figure 8.44: Box-plot (left) and CDF calibration plot (right) of predictive parameter, ultimate FOPT, in IC-Fault model, as obtained by NAB and AMH. the truth case is shown by red dots. NAB has obtained predictions closer to database than AMH.

8.3.3 Uncertainty quantification of PUNQ-S3

To check the validity of these results for the IC-Fault model, we also tested NAB and AMH algorithms on the PUNQ-S3 model (Boss, 1999; Barker & Cuypers, 2000). The PUNQ-S3 model is a different history match problem, with a different misfit landscape compared to the IC-Fault model. It has a large number of parameters (in current parameterisation 38). Therefore it is not possible to create a large number of realisations to be used as a reference case. However, it has a truth case model, which is going to be used as the reference case for a comparative study of the uncertainty quantification methods.

8.3.3.1 History matching

We used two algorithms, SBGA and imHEDA, for history matching and obtaining an ensemble of models in the PUNQ-S3 model application. Both algorithms were tuned for their best control parameters and both started from the same initial population. Each tuned algorithm was run five times with exactly the same initial population and control parameters but different random seed numbers. PUNQ-S3 model needs less exploration,

as implemented by mutation, but more exploitation than IC-Fault model. Control parameters used in two trials of the algorithms are shown in Table 8.2.

Table 8.2: Control parameters for the search algorithms in PUNQ-S3 application.

Algorithm	Size of initial population	Number of parents	Number of children	Number of generations	Number of bins	Learning rate	Mutation probability
imHEDA	150	50	50	57	20	0.9	0.1
GA	150	50	50	57	---	---	0.1

The minimum misfit and convergence results for history matching using the two algorithms are shown in Figure 8.45. Although both algorithms obtained acceptable misfit convergences, imHEDA has achieved a slightly better minimum misfit and its ensemble was thus selected for use in uncertainty quantification.

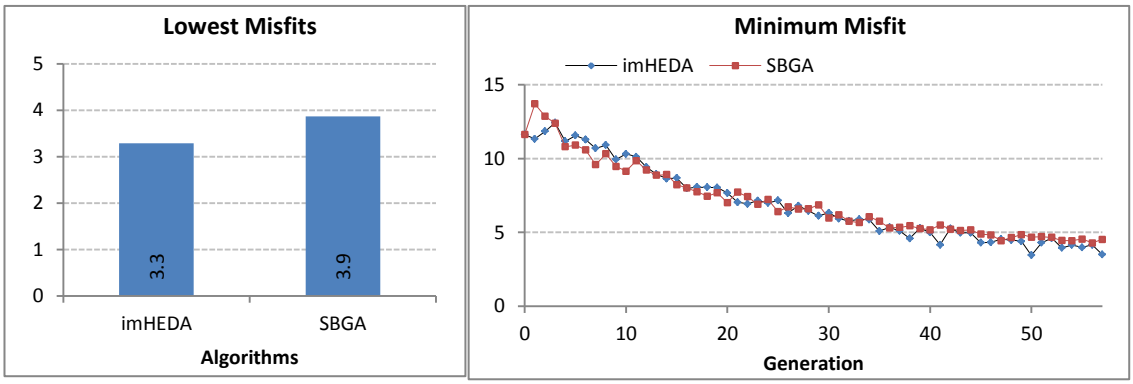


Figure 8.45: Minimum misfit (left) and misfit convergence (right) plots for two search algorithms, SBGA and imHEDA, in PUNQ-S3 application as averaged for 5 trials with different seeds. Both algorithms showed acceptable misfit convergence. imHEDA slightly achieved a better minimum misfit and its ensemble was selected for use in the uncertainty quantification.

8.3.3.2 Uncertainty quantification

In the PUNQ-S3 model application, we first looked at the uncertainty quantification results obtained by two ensemble-based algorithms, NAB and AMH, using an ensemble of 3,000 simulation runs obtained by the search algorithm imHEDA. Due to the stochastic nature of our sampling algorithms, samplings were repeated for 5 trials, each with a different seed number. The results were averaged and are shown in Figure 8.46. It appears that AMH resulted in a P50 (the middle line in the box-plot) and a CDF closer to the truth case (3.87 mmbbl).

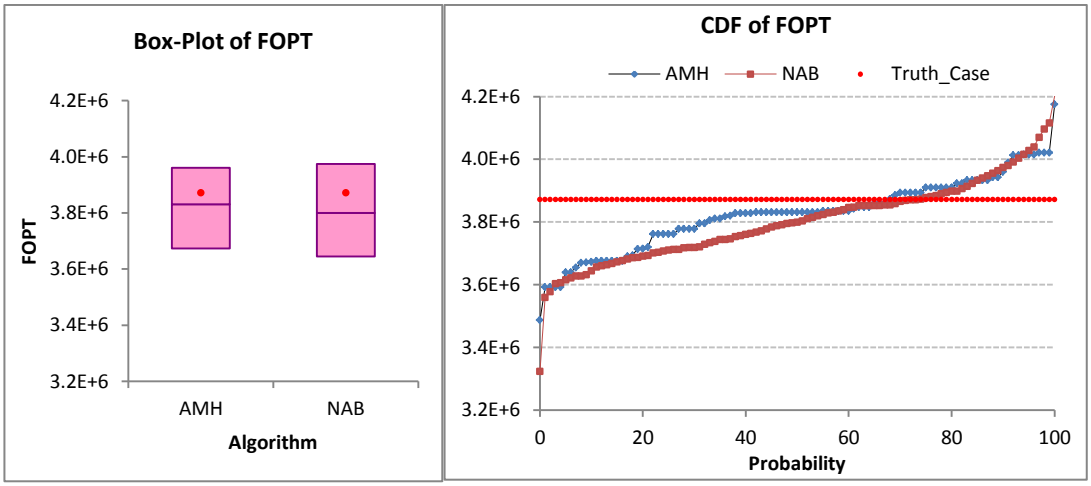


Figure 8.46: Box-plot (left) and CDF plot (right) of predictive parameter for the last time period of FOPT in PUNQ-S3 model, obtained by two MCMC sampling algorithms, NAB and AMH. Truth case is shown by the red-dotted line. AMH has obtained predictions closer to the truth case compared to NAB.

We also analysed the convergence of the average FOPT during the sampling. Figure 8.47 shows the results of the FOPT moving average in sampling with NAB and AMH. The results from AMH have converged to a value closer to the database result than those from NAB.

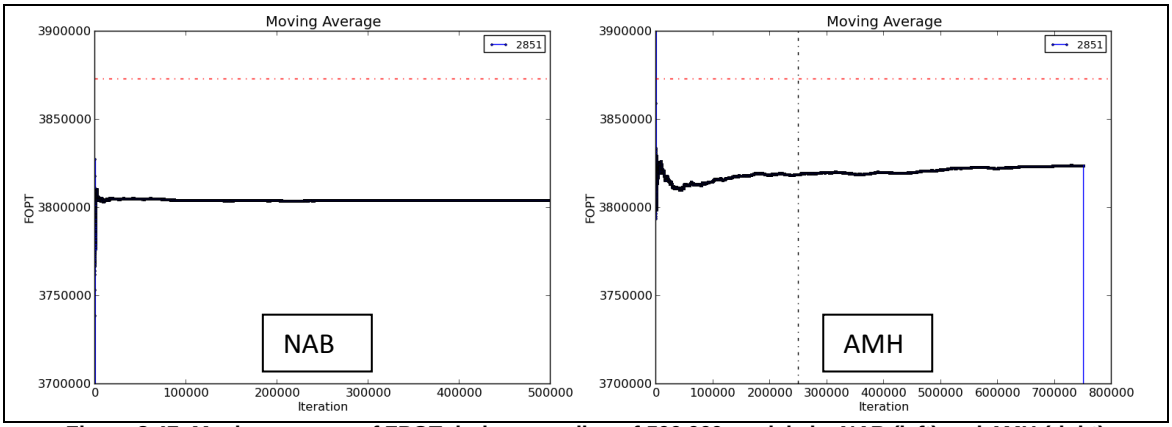


Figure 8.47: Moving average of FPOT during sampling of 500,000 models by NAB (left) and AMH (right) algorithms in PUNQ-S3 application. The truth case is shown with a horizontal red-dotted line; the vertical black-dotted line shows the end of burn-in period in AMH. AHM converged to a value closer to truth case than NAB.

8.4 Discussion

The main goal of this chapter is to analyse the factors affecting the performance of the ensemble-based sampling techniques for uncertainty quantification, and based on these findings to introduce and employ a new ensemble-based uncertainty quantification method which is efficient, robust, and not sensitive to control parameters, as it adapts the proposal distribution to the structure of the problem.

To perform a comparison study of the uncertainty quantification methods, several trials of ensembles and methods are required. As pointed out in the previous sections, due to the stochastic nature of the search algorithms used to create ensembles, one cannot repeat results with two trials of the same algorithm, even with same initial population and control parameters.

Results obtained in all three applications, multivariate Gaussian function, IC-Fault and PUNQ-S3 models, showed that credible intervals obtained by different ensemble-based uncertainty quantification methods primarily depend on the ensemble itself, which should be large enough to be used for uncertainty quantification.

The results also depend on the amount of the information contained in the ensemble; if little information is encoded in the ensemble, one would expect poor results. The complexity of the history matching problem, encoded in the misfit landscape, and the algorithm used for history matching determine the number and distribution of the models with low misfits in the ensemble. This in turn determines the amount and importance of the extracted information in the sampling.

8.5 References

Barker, J. W., and Cuypers M. (2000). Quantifying Uncertainty in Production Forecasts: Another Look at the PUNQ-S3 Problem. SPE Number: 62925. Proceedings of SPE Annual Technical Conference and Exhibition, 1-4 October, Dallas, Texas.

Ben-Israel, A. (2006). Probabilistic Distance Clustering. DIMACS Technical Report.

Boss, C. (1999). Production forecasting with Uncertainty Quantification. Technical Report. Netherlands Institute of Applied Geoscience, TNO.

Damsleth, E., Hage, A., Volden, R. (1992). Maximum Information at Minimum Cost: A North Sea Field Development Study with Experimental Design. Journal of Petroleum Technology, 44(12), 1350-1360. SPE.

Floris, F. J. T., Bush, M. D., Cuypers, M., Roggero, F., and Syversveen, A. R. (1999). Comparison of Production Forecast Uncertainty Quantification Methods−An Integrated Study. Proceedings of 1st Conference on Petroleum Geostatistics, Toulouse, France.

Gelfand, A.E. & Smith, A.F.M. (1990). Sampling-based approaches to calculate marginal densities. *Journal of the American Statistical Association*, 85(410), 398-409.

Gelman, A.G., Roberts, G.O. & Gilks, W.R. (1996). Efficient Metropolis Jumping Rules, in Bernardo, J.M., Berger, J.O., David, A.F. & Smith, A.F.M., (eds), Bayesian Statistics V. 599-608, Oxford: Oxford University press.

Haario, H., Saksman, E., & Tamminen, J. (1999). Adaptive proposal distribution for random walk Metropolis algorithm. *Computational Statistics*. 14(3), 375.

Hand, D., Mannila, H., Smyth, P. (2001). Principles of Data Mining. Cambridge, USA: The MIT Press.

Iyigun, C., and Ben-Israel, A. (2008). Probabilistic Distance Clustering Adjusted for Cluster Size. Probability in the Engineering and Informational Sciences 22, 603-621.

MacQueen, J. B. (1967). Some Methods for classification and Analysis of Multivariate Observations. Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability. Berkeley: University of California Press.

Mahalanobis, P. Ch. (1936). On the generalised distance in statistics. Proceedings of the National Institute of Sciences of India, 2 (1): 49–55.

Manceau, E., Mezghani, M., Zabalza-Mezghani, I., Roggero, F. (2001). Combination of Experimental Design and Joint Modeling Methods for Quantifying the Risk Associated With Deterministic and Stochastic Uncertainties – An Integrated Test Study. SPE paper: 71620, SPE ATCE, New Orleans, 30. September – 3 October.

Manning, D., Raghavan, P., and Schütze, H. (2008). Introduction to Information Retrieval. Cambridge, UK: Cambridge University Press.

Montgomery, D.C. (2001). Design and Analysis of Experiments. New York: John Wiley & Sons, Inc.

Roberts, G.O., Gelman, A., Gilks, W.R. (1997). Weak convergence and optimal scaling of random walk Metropolis algorithms. *Ann. Appl. Probab.* 7(1), 110–120.

Roggero, F. (1997). Direct Selection of Stochastic Model Realizations Constrained to Hsitory Data. SPE Number: 3873. SPE Annual Technical Conference and Exhibition, 5-8 October, San Antonio, Texas.

Subbey, S., Christie, M. and Sambridge, M. (2003). A strategy for Rapid Quantification of Uncertainty in Reservoir Performance Prediction. SPE Number: 79678. SPE Reservoir Simulation symposium, Houston, Texas, USA.

CHAPTER 9:

SUMMARY AND CONCLUSIONS

> "I am turned into a sort of machine for observing facts and grinding out conclusions."
>
> Charles Darwin (1809 –1882)

This chapter concludes the thesis by summarizing the chapters, drawing the main conclusions and key findings, and listing the research contributions of the thesis. Finally possible future directions for research related to this thesis are recommended.

9.1 Summary

In this section, a summary of results and specific conclusions of each main chapter is presented.

9.1.1 Literature review and clustering algorithms

In Chapter 2, first, a literature review was presented on reservoir simulation models including an analytical model, well model, streamline simulation, and full-field simulation. Then we discussed the procedure of manual history matching and the steps required to obtain a good match.

Chapter 2 also described automatic/assisted history matching in more detail, with an overview of the different optimisation techniques used for history matching. We reviewed evolutionary algorithms and their components required in assisted history matching.

Finally in Chapter 2, a survey of uncertainty quantification techniques was presented and two main Markov chain Monte Carlo techniques were reviewed that can be used for

sampling from the posterior density function. We also summarised three different clustering algorithms that can be used for grouping of models in the ensemble based on their distance-based similarity in the parameter space. Results of the Iris dataset clustering in Chapter 2 showed that probabilistic distance clustering outperforms k-mean and hierarchical agglomerative algorithms, and hence this method was used throughout the thesis.

9.1.2 Estimation of distribution algorithms

Chapter 3 described the theory and explored the application of Estimation of Distribution Algorithms (EDAs), a modern history matching technique that to date has been applied only briefly in the petroleum industry. EDAs are a set of algorithms in the evolutionary computation field, characterized by the use of explicit probability distribution models in optimization. Unlike genetic algorithms, EDAs sample from learnt probability distributions of the full posterior probability distribution.

We described three EDAs, two histogram based EDAs, including basic histogram (BH) and equal area histogram (EAH), which make use of the histogram as their probabilistic model, and the Bayesian Optimisation Algorithm (BOA), which employs a more sophisticated Bayesian Network to approximate higher order marginals.

In order to evaluate the performance of EDAs, they were applied to the synthetic PUNQ-S3 model and the following results were observed:

- All three EDAs are capable of being employed in history matching and uncertainty quantification problems.
- In an explorative set-up, the multivariate model (BOA) converges faster than univariate models, and fewer generations are needed to obtain a specified misfit value. In an exploitative set-up, the BOA is more likely to become trapped in local minima while univariate models (BH and EAH) perform better.
- The degree of model complexity is another issue to be taken into account. The complexity of inducing multivariate models is higher than that of univariate models and multivariate models require more CPU time. However, in our problems the time required to generate the probability models is negligible

compared to the time required to evaluate a population (simulation runs), even in the more sophisticated case of BOA. On the other hand, the increased sophistication of BOA allows time to be saved by permitting effective optimization in fewer solution evaluations.

- With respect to predicted uncertainty bounds, univariate histogram-based models perform better than the multivariate BOA model in an exploitative set-up, as ensemble models created by them are more diverse, and it is proved that posterior sampling from a diverse ensemble leads to narrower credible intervals and closer to the truth case.
- BH, EAH, and BOA combined with a NAB posterior probability sampler provides predicted uncertainty bounds that honour truth case results and these results are comparable with other modern algorithms published in the literature.

The second case study was a real North Sea field. Here, we were able to compare BOA with a commercialised, tuned GA. We found that:

- BOA consistently achieved lower misfits, and hence better fitting models over all random starting conditions.
- BOA was, on average, 1.5 times faster than the highly tuned GA when using algorithm parameters that had been tuned for PUNQ-S3 model.
- BOA provided better matches than GA on over 70% of the match quality components.

9.1.3 Multiobjective estimation of algorithms for history matching

In Chapter 4, multiobjective optimisation in general and a well-known multi-objective sorting algorithm, NSGA2 were presented. In multiobjective optimisation, the aim is to find the optimal trade-off between the objectives in the form of a diverse set of Pareto optimal solutions. NSGA2 performs sorting of the population based on the Pareto front rank of solutions; within solutions with same Pareto rank, it sorts solutions based on the crowding distance to the neighbouring solutions in the sorted objective space. Many real-world optimisation problems contain two or more competing objectives. We examined history matching as a multiobjective problem and combined the NSGA2

sorting technique with an elitist BOA to create a multiobjective BOA (MBOA) and use it in history matching problem of two case studies.

MBOA was initially applied for history matching of the PUNQ-S3 model. The result was compared with the result of the single objective BOA and we observed that using multiple objectives does not significantly improve the minimum misfit convergence. However, it better maintained the diversity of solutions than a single objective. We also tested different ways of splitting overall misfit into objectives. We showed that objectives based on the geography of the wells in reservoir are more favourable.

In the second case study, we used MBOA to history match Koma field in North Sea. Like the PUNQ-S3 model, the Koma field results demonstrated that the multiobjective scheme of BOA does not considerably improve convergence speed and quality of the matches. For Koma field, the diversity of MBOA was not even superior to a single objective BOA.

Finally we can conclude:

- The use of multiobjective schemes of EAs for history matching does not substantially improve the misfit convergence speed nor reduce the number of simulations required for achieving a similar match in comparison with the single objective scheme.
- A multiobjective scheme of EAs in history matching is less sensitive to control parameters and multiobjectivisation of a single overall misfit can potentially increase the diversity of the solutions in the population, since it helps the overall search process to avoid premature convergence of the population and trapping in local minima. It should be noticed that this diversity does not necessarily represent local optima for the overall misfit in the parameter search space.
- The use of multiobjective algorithms for history matching can have other added benefits. One is to study how challenging it is to match a component of the misfit, e.g. wells and regions, with the current parameterisation of a model. Another is to identify conflicting misfit objectives regarding different parts of the reservoir. In this case, these conflicting objectives reveal that the total energy of the reservoir is not enough to obtain a good match overall. Thus, the model

parameterisation should be improved or if staying with current parameterization, acceptable match can be achieved by trade-off between conflicting objectives.

9.1.4 Hybrid SBGA/iHEDA

In Chapter 5, first, two new algorithms were employed: a real-coded Genetic Algorithm based on the simulated binary crossover and mutation operators (SBGA) and an incremental Univariate Marginal Estimation of Distribution Algorithm based on the histogram model (iHEDA).

Then, we proposed a novel hybrid algorithm (SBGA/iHEDA) based on SBGA and iHEDA. In SBGA/iHEDA new solutions are created by a parallel, cooperative children generation scheme, which, uses crossover and mutation operators of GAs as a mechanism to create new solutions from the promising solutions of the entire population and it uses a histogram model built by iHEDA with the promising solutions of the entire population and samples the model to generate new solutions.

We initially applied and tuned the algorithms on the Rosenbrock function, to obtain best guess for the control parameters in history matching applications. We executed SBGA, iHEDA, and hybrid SBGA/iHEDA to solve two history matching problems, the synthetic IC-Fault model and the Teal South reservoir model. The experimental results of the performance study demonstrated that:

- All three algorithms, SBGA, iHEDA, and hybrid SBGA/iHEDA can be employed in history matching, although they may behave differently for different problems.
- Experimental results on a test function, a synthetic case, and a real reservoir model showed that hybridising SBGA with EDAs can improve a weakness of SBGA, slow convergence in early stages of the search.The hybridising also improves a problem of EDAs, not enough diversity and exploration power throughout the search.
- When applied to the Rosenbrock test function and the difficult IC-Fault model, hybridisation improved the search compared to both, pure SBGA and iHEDA.

- When applied to the Teal South reservoir, although compared to iHEDA, hybridisation improved the search. SBGA still resulted in slightly better convergence.
- We used two approaches for hybridizing, fixed and adaptive participation. It appears that fixed 50/50, 70/30 (SBGA/iHEDA), and adaptive participation showed better performance than other fixed participation schemes. However, the participation ratio of 50/50 (0.5) is proposed for future applications.

9.1.5 Gaussian-based EDAs

Chapter 6 introduced four different Gaussian-based EDAs which can be used in history matching problems with real-valued variables. These EDA algorithms include incremental univariate Gaussian-based EDA, multivariate Gaussian-based EDA, multiple univariate Gaussian-based EDA, and multiple multivariate Gaussian-based EDA. All four algorithms can be used for the continuous history matching problem.

We visualised and analysed convergence and diversity of the applied algorithms. Our experimental results show that, depending on the nature of the problems with regard to multimodality and multi-variety, these algorithms are able to solve optimisation problems in history matching efficiently and accurately without the need for discretising variables.

In Chapter 6, three novel features were used with the Gaussian-based EDA. We used an incremental learning mechanism for single univariate Gaussian EDA and probabilistic-distance clustering (PDC) for splitting the set of selected solutions into multiple Gaussian models. PDC is less sensitive than some other clustering methods, for example k-mean clustering, to both initialisation and outliers. We also used eigendecomposition for sampling multivariate distributions. The Eigenvalues obtained from the eigendecomposition were used for covariance matrix repair when needed.

We showed that the use of multiple Gaussian models for the multimodal test functions and the PUNQ-S3 model resulted in significant improvement in the objective function convergence, compared to single Gaussian models. The reason is that multiple models allow simultaneous exploration of multiple regions of the search space for each variable. Use of the multivariate Gaussian EDAs requires relatively large population

sizes. In the case of significant dependency between variables, these algorithms can improve the convergence considerably.

9.1.6 Diversity-based adaptive EDA

Chapter 7 introduced four different estimators of population diversity, which can be used for diversity/convergence studies and monitoring the performance of the algorithms in history matching, using the population-based evolutionary algorithms. These diversity measures were two distance-based measures calculated from the uncertainty parameter search space and two entropy-based measures calculated from the fitness values.

Results showed that the inertia-based diversity measure (IDM), from the distance-based measures, and the crowding entropy-based measure (CEM), from the entropy-based measures, show better diversity/convergence of the algorithms and are recommended for future studies. When these measures were used in PUNQ-S3 application, hybrid BOA/PSO had the best convergence, while it maintained diversity. In contrast, PSO had a premature convergence, clearly shown by the entropy-based measures. In the Koma field application, PSO outperformed BOA for diversity/convergence balance.

Diversity, by itself, is a measure of similarity/dissimilarity between individuals of a population. We use a distance-based diversity measure to control and adapt control parameters of a HEDA, firstly in a minimisation problem of two standard test functions and secondly in a history matching problem of a synthetic IC-Fault model. We analysed convergence and diversity results of the applied adaptive mechanism versus the base algorithm and showed that the diversity measure enables us to control the performance of HEDA and improve the search in the next generations of the algorithm.

Chapter 7 then introduced an adaptation mechanism based on the population diversity for the control parameters of the evolutionary algorithms used in history matching. The diversity measure used for the parameter adaptation was the inertia-based distance measure (IDM). In principle, an increase in the distance between individuals in the population represents an increase in the diversity. The diversity measure was used to balance exploration and exploitation performance of a population based evolutionary

algorithm in history matching. The algorithm adapts to the nature of the problem and hence outperforms the algorithms with fixed control parameters even if they are tuned.

Our experiments showed that although HEDA with a lower number of children theoretically better converges, it is more likely to be trapped in local minima in different trials of the algorithm. Choosing a higher number will reduce the trapping chance, but it will also delay the convergence of the algorithm. The optimum solution is achieved with the diversity-based adaptive HEDA, in which, at the early stages of the search, the convergence is achieved by taking lower numbers of parents, but as soon as the search is stacked at an entrapment, the diversity is increased by taking a larger number of parents. Our results showed that the proposed adaptive mechanism outperforms HEDA, in terms of minimum misfit, convergence speed, and uniqueness of the well-matched solutions. AHEDA is able to perform a balanced search, in which the diversity/convergence balance is maintained throughout the evolution.

9.1.7 Adaptive uncertainty quantification

Uncertainty quantification plays a crucial role in providing high quality and robust decisions for reservoir management. A set of diverse fitting models that represent the correct sampling of the posterior allow us to estimate posterior distribution, thus, to quantify uncertainty in performance prediction.

In Chapter 8, we studied factors affecting the performance of the ensemble-based uncertainty quantification methods. We also proposed a novel uncertainty quantification method based on Metropolis-Hasting sampling with an adaptive multivariate proposal distribution (AMH).

The results of ensemble-based sampling algorithms for uncertainty quantification showed that, if the ensemble size is not large enough or ensemble is not diverse enough, the models in the ensemble may exhibit similar characteristics. Therefore, MCMC sampling based on approximation from such ensembles cannot estimate the true posterior distribution. This is because optimisation algorithms may perform the refinement of low misfit models and do not explore the search space enough. Furthermore, the algorithms might have been trapped in local minima during the search,

in which case, model selection from resulted ensemble is not a good approximation of the true posterior probability.

The proposed method (AMH) involves inference from an ensemble of simulation models obtained by direct search algorithms in the history matching phase to approximate the posterior probability of the uncertainty parameters in model space and, consequently, the predictive parameter of the reservoir simulation model. AMH is not sensitive to the initial covariance matrix and either of the ensemble's covariance or the identity matrix can be used as the initial covariance matrix, which is going to be updated during the burn-in period, whenever a certain number of accepted samples become available.

When compared to NAB, another ensemble-based uncertainty quantification method, AMH with adaptive covariance matrix, was shown to be competitive to NAB. AMH provided results similar to the probability distribution function of multivariate Gaussian function function and slightly closer to the truth case than NAB in the PUNQ-S3 model application, although it was slightly outperformed by NAB in the IC-Fault model application, with regard to the closeness of forecasted CDF to the database result.

AMH performs well in problems with a misfit landscape and probability density that can be estimated with a Gaussian model (e.g. multivariate Gaussian function and PUNQ-S3 model), while it may struggle in problems with sharp minima and those than cannot be effectively estimated with Gaussian models.

9.2 Major research contributions

1) First application of two histogram-based EDA (BH and EAH) and Bayesian Optimisation Algorithm (BOA) of the class of Estimation of Distribution Algorithms (EDAs) for reservoir history-matching (Chapter 3).

2) Application of multiobjective sorting algorithms and integration with EDAs for multiobjective optimisation (MOO) of history matching problems (Chapter 4).

3) Development and application of a real-coded simulated binary genetic algorithm (SBGA) for history matching optimisation (Chapter 5).

4) Implementation of incremental learning mechanism and mutation operator for history matching with histogram based algorithm (imHEDA) (Chapters 5 and 8).

5) Development, implementation and application of a parallel cooperative hybrid algorithm for history matching based on fixed and adaptive participation of iHEDA and SBGA algorithms (Chapter 5).

6) Development and application of four Gaussian-based EDAs based on univariate/multivariate and single/multiple distribution models of continuous parameter space in history matching (Chapter 6).

7) Introduction of four diversity measures, including two distance-based measures in parameter space and two entropy-based measures in fitness space, for convergence/diversity analysis of population-based evolutionary algorithms (Chapter 7).

8) Development and application of a novel, diversity-based adaptive HEDA, based on inertia diversity measure, for implementing a balanced diversity/convergence search in history matching (Chapter 7).

9) Development and application of a novel, ensemble-based uncertainty quantification method, based on Metropolis sampling with adaptive multivariate Gaussian proposal distribution, k-nearest neighbours approximation for avoiding forward simulations, and probabilistic distance clustering for choosing optimum starting points of multiple independent walks in sampling (Chapter 8).

10) Successful application of developed and implemented history matching algorithms and uncertainty quantification methods on several test functions, synthetic reservoir models and real reservoir models (Chapter 3 to 8).

9.3 Conclusions

A reservoir model is created from a set of uncertain variables. The reservoir model needs to be matched to available historical data. But, as we discussed in the chapter 2, multiple realisations of the reservoir model yield same historical performance. Therefore it is important to obtain an ensemble of acceptably history-matched models

using optimisation algorithm (e.g. ones introduced in this thesis). If these algorithms perform a balanced search in terms of exploration and exploitation, one can be ensured that the ensemble represents the likely uncertainty in geology and behaviour of the reservoir.

According to the no-free-lunch-theorem, there is no single algorithm which performs well in all history matching problems. In current work, a portfolio of flexible and adaptive algorithms were developed and applied to history matching and uncertainty quantification of petroleum reservoirs. These algorithms yield good convergence speed and ensemble diversity. In addition, they yield insights into reservoir behaviour. The global search performance results and the ensemble produced using applied history matching algorithms showed they are highly competitive in terms of convergence, diversity and distribution. Furthermore, the comparative results showed that the proposed adaptive algorithm for uncertainty quantification can not only obtain a good estimate of probability distribution of predictive parameters, but also has less computational cost.

As we showed in the previous chapters, improvement of the convergence speed to achieve a single good history-matched model is important in real field applications, since a single trial of assisted-history matching run requires days or sometimes months of a cluster of computers, which is usually limited in terms of availability and the number of CPUs.

In uncertainty quantification studies using an ensemble of history-matched models, not only the convergence speed of the direct search algorithms (e.g. EDAs) is important, but also as we showed in the chapter 8, the diversity of models in ensemble is a crucial factor determining the performance and robustness of the resampling algorithms (e.g. AMH).

An adaptive probabilistic-distance clustering technique was used in the thesis for creating a Gaussian probability model in EDAs for history matching and grouping of the models in ensemble and creating Gaussian mixtures for uncertainty quantification. The clustering technique is adaptive in the sense that it works out the number of mixtures from the data and hence adapts to the structure of parameter space.

For adapting the learning and variation mechanism in EAs, we introduced and used EDAs, an implicitly adaptive class of EAs, for history matching. The learning in EDAs is explicit, i.e. they can directly identify and mix building blocks from the set of promising solutions through the use of probabilistic models. This adaptive variation mechanism helps EDAs to outperform other EAs (e.g. GAs) in terms of efficiency and quality of the matches.

Table 9.1: Advantages and disadvantages of three studied EDAs.

Algorithm	Advantage	Disadvantage
BHEDA	Can work better in exploitative search thus it requires less function evaluations.	Doesn't handle interactions; uses fix discretisation.
EAHEDA	Better handles continuous variables.	Doesn't handle interactions.
BOA	Handles interactions, if set up explorative enough, yields better convergence.	More complex structure and implementation, more computationally expensive, requires more function evaluations.

We tried adapting the sorting and selection mechanism in EAs by using multiobjective optimisation. Although history matching is not a multiobjective optimisation problem by nature, a trade-off between match components may be observed in some history matching problems. In these cases, multiobjectivisation of the overall misfit can possibly help to solve these problems more effectively and efficiently by inducing more diversity and making the algorithm less vulnerable to improper choice of control parameters.

Table 9.2: Advantages and disadvantages of single and multi-objective algorithms.

Algorithm	Advantage	Disadvantage
Single-objective	Better suits history-matching problems.	Performance is not great if there is trade-off between objective functions.
Multi-objective	Better suits development optimisation problems; less vulnerable to parameter tuning and loss of diversity.	Performance is not great if there is no trade-off between objective functions. Practically only works with two objectives.

The algorithm choice is another important aspect of EAs. As the 'no-free-lunch' theorem states, an algorithm performs better than others on a set of problems. To ensure that an algorithm is performing an acceptable search on a set of different problems, one can make them adaptive. We examined the hybridisation of algorithms as a way of adapting the search mechanism and algorithm choice. Hybridisation combines advantages of the algorithms (see Table 9.3), and hence can create more flexible and powerful algorithms. The hybrid GA/EDA provided more diversity and exploration power throughout the search.

Table 9.3: Advantages and disadvantages of SBGA, iHEDA, and their hybrid algorithm.

Algorithm	Advantage	Disadvantage
iHEDA	Preserving meaningful patterns, implicitly adaptive probability models.	Too much exploitative.
SBGA	Handling complex interactions.	Insufficient early convergence.
Hybrid SBGA/iHEDA	Handling interactions, implicitly adaptive, balanced exploration and exploitation, less liable to tuning.	Requires more complex structure and implementation.

Solution representation was another aspect of EAs which was targeted using Gaussian-based EDAs, adapted for multi-modality and multi-variety in a misfit landscape. These algorithms work naturally with continuous representations and complex, multimodal, multivariate problems, and hence can adapt to the structure of problems in history matching.

Table 9.4: Advantages and disadvantages of Guassian-based EDAs.

Algorithm	Advantage	Disadvantage
iSUGEDA	Natively supports continuous variables.	Doesn't handle multiple minima in misfit surface and interaction between variables.
SMGEDA	Natively supports continuous variables. Handles interactions between variables. Natively supports continuous variables.	Doesn't handle multiple minima in misfit surface.
MSGEDA	Natively supports continuous variables. Handles multiple minima in misfit surface.	Doesn't handle interaction between variables.
MMGEDA	Natively supports continuous variables. Handles multiple minima in misfit surface and interactions between variables..	Requires large population size.

Explicit-adaptive EAs make use of diversity measures to control and adapt the search mechanism for optimum diversity and convergence performance. Our developed diversity-based, explicitly adaptive EDA for history matching outperformed a non-adaptive EDA by maintaining the balance between diversity and convergence throughout the search.

Table 9.5: Advantages and disadvantages of AHEDA.

Algorithm	Advantage	Disadvantage
AHEDA	Performs a balanced explorative-exploitative search. No need for tuning the number of parents.	Uses a diversity threshold.

Finally, we developed and used an adaptive algorithm for ensemble-based uncertainty quantification which uses the multivariate Gaussian function as the proposal

distribution. The comparative results showed that the proposed algorithm, as a Metropolis-based algorithm, is more efficient than Gibbs samplers. It is robust for the problems which can be effectively represented by Gaussian mixtures. Lastly, it is not sensitive to the control parameters, as it adapts the proposal distribution to the structure of the sampling problem.

Table 9.6: Advantages and disadvantages of AMH comparing to NAB.

Algorithm	Advantage	Disadvantage
AMH	Proposals from adaptive Gaussian distribution; thus yields better performance in problems with Gaussian misfit landscape (e.g. PUNQ-S3 model); good acceptance rate (20-30%).	Performance in steep minima problems is poor.
NAB	Proposals from marginal log-likelihoods; yields better performance in steep minima problems (e.g. IC-Fault model).	Acceptance rate is low (usually under 3%).

9.4 Recommendations for application in industry

In this section, a summary of our recommended procedure for assisted history matching and uncertainty quantification is presented. Our recommendations for applying the findings of thesis in industrial applications are embedded.

I. Create the initial reservoir model and parameterise the uncertainties:

   a. Create the initial reservoir model based on the available static and dynamic data.

   b. Discuss and agree the set and the ranges of uncertain parameters with the multi-disciplinary team which includes people in charge of the data used for building the initial model.

II. Obtain the observation data and define the misfit function:

   a. Collect and quality control the observation data, including oil, water, and gas rates, plus average and bottom-hole pressures. Checkout for outliers and inconsistency in the allocated rates. If the model is controlled on total voidage only match two (e.g. oil, water) out of three rates.

b. Consider measurement error levels for the observation data, which is treated as the tollerance level for misfit value. In lack of knowledge about the actual measurement errors, the errors can be estimated from the standard deviation of a Gaussian model fitted to the differences of observation data and simulation results of a manualy-matched base case model.

c. Consider a misfit function, e.g. equation (3.4). The preference and the importance of the data types (e.g. oil rate, BHP) in history matching can be introduced in terms of weighting factors in the equation.

III. Perform an assisted history matching using an EA search to get an ensemble of solutions. A solution here is a vector of uncertain variables, which represents the simulation model.

a. Design and set EA control parameters. Consider a flexible or an adaptive EA for assisted history matching:

1. Flexible here means it can be tuned for the optimum performance on a particular problem by modifying different control parameters.

2. Adaptive algorithms are equipped with an adaptation mechanism for control parameters.

3. A good practice is to tune all of the algorithm's control parameters on a small synthetic model except the parent size, for which, the adaption mechanism introduced in Chapter 8 is recommended. The parent size is most important which majorly determines the balance between exploration and exploitation.

b. If tuning is not possible, our recommendations for control parameters are:

1. For the total of n uncertain variables in the model, consider at least $n+1$solutions per generation (population size).

2. Depending on the computational expenses, consider 20 to 50 iterations (generations). If the convergence is not satisfactory, extend the search to more iteration.

3. Set the number of selected solutions in each generation (parents' size) to be initially twice (e.g. in BOA) or equal (e.g. in HEDA) the number of children (population size). If possible, tune the parents' size to get a balanced exploration and exploitation by looking at the convergence and diversity plots.

4. Set the number of bins used for the discretisation of continuous variables (e.g. porosities) to a number between 10 and 25 (the half of the population size).

c. Perform assisted history matching using EAs; A typical workflow for EA is:

1. Sample the initial population (150 solutions) randomly from the prior range of the uncertain variables and run the simulation runs.

2. Sort the solutions and select the parent solutions in terms of their misfit value. For history matching multi-objective sorting algorithms are not recommended (Chapter 4).

3. Create a probablistic model for each or entire variables. Examples of the probablistic model are histogram model (in HEDA), Bayesian network (in BOA), and Gaussian model (Chapter 6).

4. Sample child solutions from the created probablistic model.

5. Run the simulation model with new values for uncertain variables, then merge created solutions with the initial population. A generation of soluions just created.

6. Continue from the step 2, until 60 generations are completed.

IV. Analyse the ensemble of history-matched models and use it for uncertainty quantification of predictive parameters:

a. Analyse the history matching results to gain insights into reservoir model:

    1. Compare prior (before history matching) and posterior (after history matching) probability distribution of each uncertain variable. Discuss new distributions with G&G and people provided the prior range.

    2. Analyse the ensemble of history-matched models for patterns and trends. The clustering algorithms (see Chapters 2 and 8) for clustering the ensemble in the uncertain variables' space can be used to find these patterns. Each cluster (or pattern) can possibly represent a local minimum region in the search space, in another words, a geological/behavioural scenario of the reservoir model.

b. Perform uncertainty quantification of predictive parameters (e.g. COP) using a MCMC sampling algorithm:

    1. Make sure the ensemble of history-matched models is large enough (e.g. 3,000). The larger ensemble size, the more stable inference results obtained in different trials of the sampling algorithm. The ensemble size becomes enough when the convergence of the search algorithm has happened.

    2. Make sure the diversity of ensemble is high enough. The diversity can be examined using different measures discussed in Chapter 7.

    3. Consider a MCMC sampling algorithm in Bayesian inference (e.g. NAB or AMH in Chapter 8) to resample a larger number of samples (e.g. 1e6). As the forward simulations are usually expensive, consider an approximation technique instead, e.g. k-NN.

    4. Use multiple MCMC walks each starting from the best model of each cluster in ensemble (clustered in variable space) using parallel processing or multi-threading techniques. This can be helpful when resampling with MCMC becomes expensive in high-dimensional problems.

5. Check for convergence of the sampling algorithm using moving average plot of the predictive parameter or trace-plots of the variables (see Chapter 8).

9.5 Future directions

Recommendations for continuation and future work in the direction of this thesis are presented in this section:

9.5.1 Linkage effect in histogram-based EDAs

A drawback of the marginal histogram-based EDAs is that they do not consider possible interaction between variables and lack the ability to detect variable linkages. This becomes serious in the problem with correlated variables. Zhang et al. (2000) used independent component analysis with univariate marginal EDA to tackle the interrelations among the variables. Ding & Zhou (2008) used two methods, a probabilistic graphical model and space transformation, for linkage detection in marginal histogram models. These methods can be used to improve the performance of the histogram-based EDAs studied in this thesis, when applied to problems with the linkage effect.

9.5.2 Adaptive bin width strategies

In many history matching problems, such as those with steep optima, the number of bins should be large enough to capture the local optima. Population size is usually limited due to the computational complexity of simulation runs in history matching. Adaptive bin width strategies allow the basic histogram model to perform a quality search using small bin size in a relatively small number of populations.

Ding et al. (2007) used two strategies for adapting the bin size in histogram-based EDA, the surrounding effect and the shrinking strategy. These two strategies can be applied to the histogram-based EDAs employed in this thesis. The surrounding effect adds a small portion of the neighbouring bins' probability to each bin probability. This correction improves the performance of the basic histogram model where the population size is not large enough. The shrinking strategy improves the search by shrinking the width of the

promising variable bin for the next generation. It allows the algorithm to find quality solutions in such situations, even with a small bin size.

9.5.3 Adaptive Gaussian based EDAs

In this thesis, we presented and employed four different Gaussian-based EDAs based on the single or multiple Gaussian models and univariate or multivariate models. An extension of this work could be a single Gaussian-based algorithm, which could adapt to the structure of the problem by taking appropriate type (univariate or multivariate) and number of Gaussian mixture models.

Heuristic methods can be used to estimate the level of dependency between variables. Therefore the interaction between variables can be determined by the algorithm itself. Another technique is to use principal component analysis (PCA) to obtain orthogonal variables from the original variables, then perform optimisation, using univariate models in the transformed space.

The number of models in Gaussian mixtures can be determined using the clustering technique discussed in Chapter 8. Therefore, this adaptive Gaussian-based EDA will be able to adapt to the shape of the misfit landscape in terms of multi-modality as the algorithm determines the number of clusters by itself.

9.5.4 Other applications for multiobjective optimisation

In this thesis, we showed that the history matching problem with the common misfit definition in assisted history matching is not a multiobjective problem by nature; however, there are other added benefits of multiobjectivisation of single overall misfit in history matching. Multiobjective EAs can be used for reservoir development optimisation, which is more naturally stated as a multiobjective problem and better fits the multiobjective scheme of EAs. For a development optimisation problem, the possible competing objectives can be defined as the investment cost and recovery income or even short term and long term production.

9.5.5 Mixed histogram-Gaussian based EDAs

Typically, history matching involves both continuous and discrete uncertain parameters. Examples of continuous variables are porosity and permeability and examples of discrete variables are open/close status of faults or rocktype, PVT, or equilibrium region numbers. As discussed earlier, histogram models inherently support discrete variables but they discretise continuous variables into a certain number of bins. In contrast, Gaussian-based models inherently support continuous variables and discrete variables are considered as continuous, and after the optimization the results of discrete parameters are rounded up to the nearest possible discrete value. A mixed histogram-Gaussian based EDA could support both discrete and continuous variables natively.

9.5.6 Diversity-based adaptive mechanism for other EAs

In Chapter 7, we used an adaptation mechanism for a histogram-based EDA based on the inertia diversity measure. A similar adaptation mechanism can be used based on other diversity measures, or for control parameters of any other population-based evolutionary algorithm which determine the diversity of the population in each generation. A good example of these control parameters are the crossover and mutation probability in standard GA.

9.5.7 Other hybrid algorithms

In this thesis, we employed a hybrid algorithm, combining a histogram-based EDA (HEDA) and a simulated binary GA (SBGA), to improve the performance of the HEDA in terms of convergence speed, balance between exploration and exploitation, etc. We tested a single participation function, which was parallel cooperative search; other participation functions are still open for further investigation.

In another work (Reynolds et al., 2011), we proposed and employed a hybrid BOA-PSO for history matching. There are other potential hybrid global algorithms, such as hybrid HEDA/PSO or hybrid BOA/SBGA, which can be developed and applied to history matching. Finally, hybridization of EDAs with local optimisation algorithms can be attractive for history matching.

9.5.8 NAB-like uncertainty quantification with K-NN approximation

In chapter 8, we presented AMH, a novel MCMC-based algorithm based on Metropolis-Hasting sampling with adaptive multivariate proposal distribution. To avoid further simulation runs, AMH was equipped with a k-nearest neighbour approximation. In addition, it was equipped with the probabilistic-distance clustering to select optimum starting points of the multiple independent walks.

Another ensemble-based uncertainty quantification method can be achieved by replacing the M-H sampling taking proposals from the multivariate Gaussian in AMH with the Gibbs sampler taking proposals from marginal probabilities. This method shares the sampling method and proposal distribution with NAB, but instead of simple neighbourhood approximation, as in NAB, it uses a more robust and sophisticated k-nearest neighbour (K-NN) approximation. As in AMH, here probabilistic-distance (PD) clustering can be used to select optimum starting points for the multiple walks.

APPENDIX A: ABBREVIATIONS

ACF	Auto-Correlation Function
AHEDA	HEDA with diversity-based parameter adaptation
AMH	Adaptive Metropolis-Hastings
ANN	Artificial Neural Network
ARMC	Acceptance-Rejection Monte Carlo
BHEDA	Basic Histogram Estimation of Distribution Algorithm
BHP	Bottom-Hole flowing Pressure
BMA	Bayesian Model Averaging
BMDA	Bivariate Marginal Distribution Algorithm
BOA	Bayesian Optimisation Algorithm
BOA/PSO	Hybrid Bayesian and Particle Swarm Optimisation Algorithm
BT	Break-through Time
CDF	Cumulative Distribution Function
CEM	Crowding Entropy-based Measure
cGA	compact Genetic Algorithm
CMR	Covariance Matrix Repairing
COMIT	Combining Optimizers with Mutual Information Tress
DEMOPR	Differential Evolution based on MOGA Pareto Ranking
DGEA	Diversity-Guided Evolutionary Algorithm
EAHEDA	Equal-Area Histogram Estimation of Distribution Algorithm
EBNA	Estimation of Bayesian Network Algorithm
EC	Evolutionary Computation
EDA	Estimation of Distribution Algorithm
EGNA	Estimation of the Gaussian Networks Algorithm
EnKF	Ensemble Kalman Filter
FDA	Factorized Distribution Algorithm
FOPT	Total Oil Production of the Field
GA	Genetic Algorithm
GOC	Gas-Oil Contact
GOR	Gas-Oil Ratio

HAC	Hierarchical Agglomerative Clustering
HDM	Pairwise Hamming Distance Measure
HMC	Hamiltonian Monte Carlo
IDM	Inertia-based Diversity Measure
iHEDA	incremental Histogram-based Estimation of Distribution Algorithm
imHEDA	incremental mutation-enabled Histogram-based Estimation of Distribution Algorithm
iSUGEDA	incremental Univariate Single Gaussian Estimation of Distribution Algorithm
iUMDA	incremental Univariate Marginal Distribution Algorithm
JDF	Joint Distance Function
KMC	K-Mean Clustering
K-NN	K-Nearest Neighbours
LSQR	Sparse Equations and Least Squares
MBOA	Multiobjective Bayesian Optimisation Algorithm
MCMC	Markov Chain Monte-Carlo
M-H	Metropolis-Hastings
mIDEA	mixed Iterated Density Estimation Evolutionary Algorithm
MIMIC	Mutual Information Maximizing Input Clustering
MMGEDA	Multiple Multivariate Gaussian Estimation of Distribution Algorithm
MOEA	Multiobjective EA Evolutionary Algorithm
MOGA	Multiobjective Genetic Algorithm
MOPSO	Multi-Objective Particle Swarm Optimisation based on NSGA
MSE	Mean Squared Error
MUGEDA	Multiple Univariate Gaussian Estimation of Distribution Algorithm
NAB	NA-Bayes
NAB	Neighbourhood Algorithm with Bayesian inference
NPGA	Niched Pareto Genetic Algorithm
NSGA	Non-dominated Sorting Genetic Algorithm
NSGA2	Elitist Non-dominated Sorting Genetic Algorithm
NTG	Net-to-Gross
PAES	Pareto Archived Evolution Strategy
PAVE	AVErage gridblock Pressure

PBEA	Population-Based Evolutionary Algorithm
PBIL	Population-Based Incremental Learning
PBILc	Population-Based Incremental Learning extended to continuous spaces
PCE	Polynomial Chaos Expansion
PDC	Probabilistic Distance Clustering
PDF	probability Density Function
PEM	Proportional Entropy-based Measure
PI	Productivity Index
PMBGA	Probabilistic Model Building Genetic Algorithm
PNX	Parent-centric Normal Crossover
PVT	Pressure-Volume-Temperature
QGP	Production Gas rate
QWP	Production Water rate
QOP	Production Oil rate
RD	RanDom starting points
RECEDA	REal-coded Estimation of Distribution Algorithm
RFT	Repeat Formation Tester
SBGA	Simulated Binary Genetic Algorithm
SBGA/iHEDA	Hybrid SBGA and iHEDA algorithm
SBX	Simulated Binary Crossover
SHCLVND	Stochastic Hill Climbing with Learning by Vectors of Normal Distributions
SMGEDA	Single Multivariate Gaussian Estimation of Distribution Algorithm
SPEA	Strength Pareto Evolutionary Algorithm
UMDA	Univariate Marginal Distribution Algorithm
UMDA	Univariate marginal distribution Algorithm
UMDAc	Univariate Marginal Distribution Algorithms for continuous domains
WCT	Water-CuT ratio
WOC	Water-Oil Contact

APPENDIX B: SOURCE CODE SUMMARY

In this Appendix, source code documentation of the Python scripts written for this thesis is provided. The developed source codes include 13 Python scripts. Following table shows a summary of the scripts and their code statistics. Source code files are not included in the thesis as they blow up the number of pages of this thesis. This appendix contains the description of each module (script), its usage, created and last modified dates followed by a referenced list of classes and functions, for which, a short description and parameters and return values are provided.

I worked on and authored all the scripts myself during my work on this thesis. I originated, designed and developed the idea and structure for most of the modules including **`Alg`**, **`Analyser`**, **`Forecaster`**, **`Cluster`**, **`Grapher`**, **`Stats`**, and **`TF`**. For the rest of modules (**`Runner`**, **`SimReaders`**, **`ObsReaders`**, **`RunSimMisfit`**, **`Utils`** and **`MisfitCalculator`**), the idea and initial code was from Prof. Christie's research group (Uncertainty Quantification Group at Heriot-Watt University), however the code has been substantially modified and developed further for new features and functionalities.

Following class and function documentation was automatically generated by running Python's **Sphinx** and **rst2pdf** modules. An HTML version of the following documentation is placed in the source code directory of the group's cluster computer.

Module / Line Count	Total	Source	Comment	Blanks
Alg.py	1576	1027	341	208
Analyser.py	980	694	193	93
Cluster.py	232	154	58	20
Forecaster.py	987	734	170	83
Grapher.py	626	512	68	46
MisfitCalculator.py	225	132	68	25
ObsReaders.py	446	287	109	50
Runner.py	671	474	107	90
RunSimMisfit.py	213	126	56	31
SimReaders.py	406	252	103	51
Stats.py	388	167	163	58
TF.py	129	61	42	26
Utils.py	311	161	106	44
Total	7190	4781	1584	825

1) Alg Module

This module initiates and runs history matching jobs using different sorting and optimisation algorithms. :Explain: The module works in MPI mode, so that, on root node the history matching job is performed and simulation jobs are going to be sent to and received by other nodes in **`mpi_controller()`**. **`Experiment`** reads checks (**`Experiment.check_args()`**), and sets algorithm parameters. Then it creates (**`Experiment.create_init_pop()`**) the initial population of solutions. **`Population`** class contains a list of solutions (reservoir simulations) which can be run and then sorted by a criterion (e.g. objective function value). **`SORT`** class and its derived classes sort a population using a single (**`SOT`**) or multiobjective (**`WSA`**, **`AWSA`**, **`NSGA2`**, **`SPEA2`**) sorting algorithms. **`Solution`** class represents a single simulation run. Optimisation and search algorithms are derived from **`ALG`**, which include **`HEDA`** ,:class:*.EAHEDA*,:class:*.GA*, **`GAHEDA`**, as well as Gaussian-based EDAs (**`SUGEDA`**,:class:*.SMGEDA*,:class:*.MUGEDA*,:class:*.MMGEDA*, and **`AGEDA`**).

Results of each simulation run stored in a directory (leap*). At the end of experiment 'misfitparams.dat' files are gathered in a tab separated misfit (*.tsm) file. In addition, diversity and convergence results per generation are output to a tab separated generations (*.tsg) file.

Usage: The module runs on standalone and calls another module 'Runner.py -f <ensemble>.def' for initialisation and setup of algorithm parameters.

Created: AA110513, *Last Modified:* AA130217.

`Alg.`**`mpi_controller`** ()
main driver for mpi; it scatters and gathers tasks from 'root' node (rank!=0) to other nodes (rank!=0).

`Alg.`**`exit`** ()
exit system for main node (rank==0)and others(rank!=0).

class `Alg.`**`Population`** (*`num`*, *`n_vars`*, *`pars`*, *`simu`*, *`alg=''`*)
Represents a population of solutions.

`append (`*`solution`*`)`
appends a new solution to the population.

`extend (`*`new_population`*`)`
extends a population with solutions from a new population.

`sort (`*`criteria=-1`*`)`
sorts a population using a criterion, e.g. the index of objective functions.

`length ()`
returns the number of solutions in a population.

populate (*pars_list*)
creates solution objects from a parameters value list to run simulation either on cluster or serial machine.

Parameters: **pars_list** -- parameters value list of population.

Returns: None.

class Alg.**SORT** (*gen*, *objs*)
base class for sorting algorithms.

Parameters:
- **gen** -- generation number.
- **objs** -- dictionary of objective functions.

Returns: None

scale_objs ()
scales up and normalises the objective values for multi-objective crowding calculations.

nondominated_set ()
obtains nondominated set of solutions for multi-objective algorithms.

fast_nondominated_sort2 ()
Discovers Pareto fronts in a population, based on non-domination criterion.

fast_nondominated_sort (*first_front_only=False*)
discovers Pareto fronts in a population, based on non-domination criterion (second implementation).

Parameters: **first_front_only** -- boolean for either to return only first front or not.

Returns: list of fronts with different ranks.

class Alg.**SOT** (*gen*, *objs*)
single objective selectio n based on tournament.

run (*n_select*, *solutions*)
runs single objective tournament selection.

class Alg.**WSA** (*gen*, *objs*)
Weighted Sum Approach (WSA) Using randomly generated weights and Elitism by Ishibuchi and Murata (1996).

run (*n_select*, *solutions*)
runs Weighted Sum Approach (WSA) sorting algorithm for multiobjective optimisation.

Parameters:
- **population** -- population of solutions.

• **n_select** -- number of selected.

Returns: list of sorted solutions, which are parents for the next generation.

class Alg.**AWSA** (*gen*, *objs*)

Adaptive Weighted Sum Approach (AWSA) adapts weight factors from the previous generation.

run (*n_select*, *solutions*)

runs Adaptive Weighted Sum Approach (AWSA) sorting algorithm for multiobjective optimisation.

Parameters:

• **population** -- population of solutions.

• **n_select** -- number of selected solutions.

Returns: None.

class Alg.**PEF** (*gen*, *objs*)

Entropy-based Sorting Algorithm (Deb, Pratab, Agarwal, and Meyarivan, 2002).

run (*n_select*, *solutions*, *k=None*)

runs entropy-based multiobjective sorting algorithms.

Parameters:

• **n_select** -- number of selected solutions.

• **solutions** -- solutions to be sorted.

• **k** -- number of nearest neighbor solutions.

Returns: list of sorted solutions, which are parents for the next generation.

class Alg.**NSGA2** (*gen*, *objs*)

The improved Non-dominated Sorting Genetic Algorithm (Deb, Pratab, Agarwal, and Meyarivan, 2002).

run (*n_select*, *solutions*)

runs NSGA2 multiobjective sorting algorithm.

Parameters:

• **n_select** -- number of selected solutions.

• **solutions** -- solutions to be sorted.

Returns: list of sorted solutions, which are parents for the next generation.

crowding_distance_assigment ()

crowding_distance_assigment: Initializes, then assigns distance for each individual in solutions.

crowding_entropy_assigment ()

crowding_entropy_assigment: Initialize, then assigns distance for each individual in the solutions.

static **crowded_comparasion_operator** (*x*, *y*)

crowded_comparasion_operator: A static method which oerrides the < and > operators in the sort() function.

Parameters:
- **x** -- here is the first solution
- **y** -- here is the second solution

Returns: None

class Alg.**SPEA2** (*gen*, *objs*)

The improved Strength Pareto Evolutionary Algorithm (Zitzler, Laumanns and Thiele 2001).

run (*n_select*, *solutions*)

runs improved Strength Pareto Evolutionary Algorithm (Zitzler, Laumanns and Thiele 2001).

Parameters:
- **n_select** -- number of selected solutions.
- **solutions** -- solutions to be sorted.

Returns: list of sorted solutions, which are parents for the next generation.

randomizedSelect (*A*, *k*, *length*)

randomizedSelect: select the kth smallest element from array without sorting it.

Parameters:
- **A** -- list of solutions to be sorted
- **k** -- the index of k
- **length** -- length of array

Returns: the kth smallest element of array.

class Alg.**Solution** (*pars*, *num*, *gen*, *n_vars*, *simu*)

Solution: contains definition of single solution in population.

evaluate ()

evaluator function, creates a temp dir, runs simulator and return objectives to other nodes (rank!=0).

Param : None

Returns: objectives value string

scale_objectives (*obj_bounds*)

scale objective values required for the distance calculations.

Parameters: **obj_bounds** -- bound of objective function values.

Returns: None

dominated (*other*, *maximise=False*)
Method checks whether the individual is dominated other.

Parameters:
- **other** -- another solution
- **maximise** -- boolean either problem is a Maximisation or not (minimisation).

Returns: True or False

class Alg.**ALG** (*gen*, *pars*, *args*)
base class for optimisation algorithms.

Parameters:
- **pars** -- parameters objs
- **args** -- algorithm's control parameters

Returns: None

sbx_crossover (*first_parent*, *second_parent*)
performs Simulated Binary Crossover (SBX) [1995 by Deb and Agrawal] between two individual parents.

Parameters:
- **first_parent** -- first individual parent.
- **second_parent** -- second individual parent.

Returns: a list with two result children

opb_crossover (*first_parent*, *second_parent*)
Execute a one point binary crossover on the input individuals in place (SBX) [1995 by Deb and Agrawal].

Parameters:
- **first_parent** -- first individual parent.
- **second_parent** -- second individual parent.

Returns: A tuple of two child individuals

sbx_mutation (*child*)
performs real polynomial simulated binary mutation for the crossovered children.

Parameters: **child** -- Individual to be mutated.

Returns: the child itself after mutation.

gss_mutation (*child*, *mu*, *sigma*)
performs real Gussian mutation for the crossovered children.

Parameters:
- **child** -- Individual to be mutated.
- **mu** -- Mean around the individual of the mutation.

- **sigma** -- Standard deviation of the mutation.

Returns: the child itself after mutation.

normalize (*child*)

guarantees that any parameter value created does not cross parameter ranges.

Parameters: **child** -- the child to be checked

Returns: the child itself after mutation.

modeler (*parents*, *Laplace_corerection=False*)

creates a histogram model for each parameter out of parent solutions.

Parameters:

- **parents** -- the list of parents
- **Laplace_corerection** -- boolean perform Laplace correction or not.

Returns: list of list of probebilities

learning (*model*, *r_Learning*)

learning from the past generation's model.

Parameters:

- **model** -- original model.
- **l_rate** -- learning rate.

Returns: updateed model.

surrounding (*model*)

adds little probability form bins with probabilities to surrounding bins.

Parameters: **model** -- original model

Returns: updateed model

shrinking (*model*)

shrinking strategy, shrink bin with maximum probability to two bins.

Parameters: **model** -- original model.

Returns: updated model (self.par_ranges is also updated).

univariate_guassian_modeler (*pars_data*)

calculates and return the mean and standard deviation of the parents.

Parameters: **pars_data** -- the list of parameter values of the parents.

Returns: mean and standard deviationof the parents.

univariate_guassian_generator (*n_children*, *model*, *p=''*)

generates and return child solutions using the 'univariate Gaussian model' i.e. mean and sigma.

Parameters:

n_children -- number of children to be generated

- **model** -- probabilistic model created from the parent solution in the generation
- **p** -- parameter serial number. list of generated child solutions.

multiple_gaussian (*n_children*, *model*)

generates and return child population using the 'multivariate Gaussian model' i.e. mean and sigma of Gaissian mixtures.

Parameters:
- **n_children** -- number of children to be generated
- **model** -- probabilistic model created from the parent solution in the generation.

Returns: list of generated child solutions

multivariate_guassian_modeler (*pars_data*)

calculates and return the 'mean' and 'covariance' of the parents.

Parameters: **pars_data** -- the parameter values of the parents.

Returns: the covariance vector of the parents.

multivariate_guassian_generator (*n_children*, *(mean*, *cov)*)

generator: generates and return the multivariate Gaussian mixture from the parent solution.

Parameters:
- **n_children** -- number of children to be generated
- **mean** -- the mean vector of the parents.

Return cov: the covariance matrix of the parents.

Returns: list of generated child solutions.

generator (*n_children*, *model*)

generates and return child population from the histogram model created from the parent solution.

Parameters:
- **n_children** -- number of children to be generated
- **model** -- probabilistic model created from the parent solution in the generation

Returns: list of generated child solutions.

PDC (*data*, *n_cluaters=''*, *cutoff=0.001*, *signif_value=0.9*)

performs probabilistic distance clustering (PDC) of a list of solutions.

Parameters:
- **data** -- the array of solutions' parameters value.
- **n_cluaters** -- number of clusters.
- **cutoff** -- cut-off value for shift in cluster centers.
- **signif_value** -- significance value for determining the number of clusters.

Returns: the list of indeces of solutions in clusters, the list of cluster centers and the list of membership probabilities.

class Alg.**HEDA** (*gen*, *pars*, *args*)

basic Histogram-based Estimation of Distribution Algorithm (HEDA); it is equiped with incremental learning, mutation probability, and two adaptive bin-width strategies, surrounding effect and Shrinkage.

run (*parents*, *n_Children*)

runs HEDA algorithm.

Parameters:
- **parents** -- parent solutions parameters.
- **n_Children** -- number of children to be created.

Returns: the list of generated children parameters.

class Alg.**SUGEDA** (*gen*, *pars*, *args*)

Single Univariate Gaussian-based EDA with incremental learning mechanism (iSUGEDA).

run (*parents*, *n_Children*)

runs SUGEDA algorithm.

Parameters:
- **parents** -- parent solutions parameters .
- **n_Children** -- number of children to be created.

Returns: generated children parameters.

class Alg.**SMGEDA** (*gen*, *pars*, *args*)

Single Multivariate Guassian Estimation of Distribution Algorithm (SMGEDA).

run (*parents*, *n_Children*)

runs SMGEDA algorithm.

Parameters:
- **parents** -- parent solutions parameters .
- **n_Children** -- number of children to be created.

Returns: generated children parameters.

class Alg.**MUGEDA** (*gen*, *pars*, *args*)

Multiple Univariate Guassian Estimation of Distribution Algorithm (MUGEDA).

run (*parents*, *n_Children*)

runs MUGEDA algorithm.

Parameters:
- **parents** -- parent solutions parameters .
- **n_Children** -- number of children to be created.

Returns: generated children parameters.

class Alg.**MMGEDA** (*gen*, *pars*, *args*)
Multiple Multivariate Guassian Estimation of Distribution Algorithm (MMGEDA).

run (*parents*, *n_Children*)
runs MMGEDA algorithm.

Parameters:
- **parents** -- parent solutions parameters .
- **n_Children** -- number of children to be created.

Returns: generated children parameters.

class Alg.**AGEDA** (*gen*, *pars*, *args*)
Adaptive Guassian Estimation of Distribution Algorithm (AGEDA).

run (*parents*, *n_Children*)
runs AGEDA algorithm.

Parameters:
- **parents** -- parent solutions parameters.
- **n_Children** -- number of children to be created.

Returns: generated children parameters.

class Alg.**EAHEDA** (*gen*, *pars*, *args*)
Equal Area Histogram EDA algorithm (EAHEAD).

run (*parents*, *n_Children*)
runs Equal Area Histogram EDA (EAHEDA) algorithm.

Parameters:
- **parents** -- parent solutions parameters.
- **n_Children** -- number of children to be created.

Returns: generated children parameters.

modeler (*solutions*)
calculates and return the parameter ranges with equal probability from the parents.

Parameters: **solutions** -- the list of parent solutions.

Returns: the parameter ranges.

generator (*n_children*, *model*)
generates and return child population from the ranges.

Parameters:
- **n_Children** -- number of children to be created.
- **model** -- the histogram model created from the parents.

Returns: generated children parameters.

class Alg.**GA** (*gen*, *pars*, *args*)
Contains Genetic Algorithm (GA).

run (*parents*, *n_Children*)
runs Genetic Algorithm.

Parameters:
- **parents** -- parent solutions parameters.
- **n_Children** -- number of children to be created.

Returns: generated children parameters.

class Alg.**GAHEDA** (*gen*, *pars*, *args*)
GAHEDA: Hybrid Genetic Algorithm / Histogram-based EDA algorithm.

run (*parents*, *n_Children*)
runs hybrid HEDA / GA.

Parameters:
- **parents** -- parent solutions parameters.
- **n_Children** -- number of children to be created.

Returns: generated children parameters.

get_parts (*r_Participation*)

calculates the number of solution needs to be generated from GA (i.e. for a participation rate).

Parameters: **r_Participation** -- GA participation rate

Returns: number of GA solution.

class Alg.**Experiment** (*path*)
Experiment: main driver class for an history-matching experiment.

Parameters: **path** -- the directory path of experiment.

Returns: None

check_args ()
checks arguments, algorithm control parameters etc for errors.

read_adapts (*alg_params*)
read arguments for self.adaptive algorithm control parameters.

Parameters: **alg_params** -- algorith parameters.

Returns: updated algorithm parameters

create_init_pop (*gen*, *n_initial_pop*)
read initial population from either a file, job or tdrm stracture indictaed in 'ensemble.def' by "init_population_in".

Parameters:

- **gen** -- generation number.
- **n_initial_pop** -- number of solutions in initial population.

Returns: initial population.

`parse_arguments ()`
pares arguments passed to this module.

2) Analyser Module

This class module analyses and prepares results of history matching and uncertainty quantification for post-processing and graphing tools. **`Parameter`** represents a single uncertainty parameter. **`Run`** represents a single reservoir simulation run, while a **`Measure`** represents a population of them and contains different diversity and convergence measures (e.g. **`Measure.hamming_diversity()`**, **`Measure.inertia_diversity()`**,**`Measure.proportional_entropy()`**, **`Measure.crowding_entropy()`**). **`Analyse`** and class contains additional pre/post processing functions for history matching and uncertainty quantification, such as:

- **`Analyse.run_pareto()`** sorts solutions based on their Pareto ranks.
- **`Analyse.run_pareto()`** reads parameter definition file and creates **`Parameter`** objects.
- **`Analyse.run_sort()`** sorts leap directories.
- **`Analyse.run_read()`** reads and sorts leap directories and outputting to *.tsm files.
- **`Analyse.run_misfit()`** calculates generational misfit measures from a *.tsm file and outputs them to a *.tsg file.
- **`Analyse.run_pmd()`** calculates the histogram model of parameters and outputs to a *.tsd file.
- **`Analyse.run_select()`** selects models from an ensemble of history-matched models using clustering.
- **`Analyse.run_single()`** calculates and outputs misfit values and its calculation components to a *.tsr file.
- **`Analyse.run_transfer()`** transfers created tab separated files (e.g. *.tsm, *.tsg, *.tsr) job in directories to a single folder, '<root_directory>_runs'.
- **`Analyse.run_icf()`** reads full IC-Fault database and creates *.tsm file out of it. It uses **`ICF_Reader`** class for reading and storing database results.
- **`Analyse.run_obs()`** gets observation and simulation data for forecasted runs.

- **`Analyse.run_rdat()`** extracts Date-COP table from *r.dat file of VIP and store in a file.

Usage: The module runs on standalone and accepts up to 4 command line arguments. For help on arguments type in command line **`Analyser.py`** -h

Module author: Asaad Abdollahzadeh <Asaad.Abdollahzadeh@gmail.com>

Created: AA110513, *Last Modified:* AA130217.

class `Analyser.`**`Parameter`** (*`num`*, *`n_bins`*, *`name`*, *`type`*, *`value`*, *`par_normalize=False`*)

contains definition of single parameter in a solution.

Parameters:

- **num** -- parameter number
- **n_bins** -- number of bins
- **name** -- parameter name
- **type** -- parameter type, e.g. discrete, continuous and binary
- **value** -- parameter values or range
- **par_normalize** -- boolean for either normalize the range or not.

Returns: None

class `Analyser.`**`Run`** (*`pars`*, *`num`*, *`gen`*, *`n_vars`*, *`simu`*)

contains definition of single run in population.

Parameters:

- **pars** -- list of parameter values
- **num** -- run number
- **gen** -- generation number
- **n_vars** -- list of the number of misfit components in each objective.
- **simu** -- simulator type, e.g. Eclipse or VIP.

Returns: None

class `Analyser.`**`Measure`** (*`num`*, *`n_vars`*, *`pars`*, *`simu`*, *`alg=''`*)

Calculate diversity measures for a population of solutions. It is a base class for population class.

Parameters:

- **num** -- population number.
- **n_vars** -- list of the number of misfit components in each objective.
- **pars** -- list of parameter values

- **simu** -- simulator type, e.g. Eclipse or VIP.
- **alg** -- algorithm type.

Returns: None

hamming_diversity ()
measures diversity for a population using the pairwise Hamming distance.

inertia_diversity ()
measures diversity for a population using the 'Moment of inertia calculation' or 'distance-to-average-point measure' .

proportional_entropy (*k=-1*)
measures entropy of a population using the 'Misfit proportional'.

entropy2 ()
measures entropy of a population using the 'parameter bins proportional'.

crowding_entropy (*k=-1*)
measures entropy of a population using crowding distance.

class Analyser.**Analyse** (*mykey*, *predict_var='FIELD:ROOT:FOPT'*)
class to analyse history-matching results.

Parameters:
- **predict_var** -- predictive variable.
- **mykey** -- a placeholder for an argument.

Returns: None

make_uqs_db (*path*)
creates 'uqs_db' and 'obs_db' files for a job directory.

Parameters: **path** -- directory path of the job.
Returns: none

run_pareto (*job_dirs*)
sorts solutions based on thier Pareto ranks.

Parameters: **job_dirs** -- job directory path
Returns: None

read_pars (*job_dirs*, *n_bins=20*)
reads parameter file range ('par_ranges.dat').

Parameters:
- **job_dirs** -- the name of the directory containing 'uqs_db' file.
- **n_bins** -- number of bins.

Returns: pars: parameter objects.

run_sort (*job_dirs*)

sorts the solutions (leap* directories).

Parameters: **job_dirs** -- job directory path(s)

Returns: None

run_read (*path*, *read_leaps=False*, *with_np=True*, *write_tsm=False*, *sort_by='Solution'*, *predict_var=''*)

read and stors history-matchin result files <job>.tsm

Parameters:

- **path** -- file path of the misfitparam file (*.tsm)
- **read_leaps** -- boolean to read from l'leap' directories or not.
- **with_np** -- boolean to read with Numpy or not.
- **write_tsm** -- boolean to write-out the *.tsm file or not.
- **sort_by** -- sort solutions by what.
- **predict_var** -- predictive variable

Returns: tuple of headers and parameter ranges and numpy array of tsm data.

run_misfit (*job_dirs*, *from_tsm=False*)

calculates generational misfit measures and outputs them to <job>.tsg

Parameters:

- **job_dirs** -- job directory name(s)
- **from_tsm** -- read from '*.tsm' file or 'leap' directories.

Returns: None

run_pmd (*job_dirs*)

calculates and outputs posterior probability distribution function to <job>.tsp file.

Parameters: **job_dirs** -- job directory name(s)

Returns: None

run_select (*job_dirs*)

selects models from an ensemble of history-matched models.

Parameters: **job_dirs** -- directory containing models

Returns: list of selected model numbers.

run_single (*job_dirs*)

calculates and outputs misfit values and its calculation components to <job>.tsr file for a single run directory.

Parameters: **job_dirs** -- job directories

Returns: None

run_transfer (*job_dirs*)
Transfers *.tsv files from jobs in root directory to a single folder, '<root_directory>_runs'.

Parameters: **job_dirs** -- job directory path(s).
Returns: None

run_icf (*path*)
reads full IC-Fault database and creates *.tsm file from it.

Parameters: **path** -- directory path containing 'ic_fault.data_base'.
Returns: None

run_obs (*path*)
Gets obs and sim data for forecasted runs.

Parameters: **path** -- directory path.
Returns: None

run_rdat (*job_dirs*)
extract Date-COP table from *r.dat file of VIP and store in a file.

Parameters: **job_dirs** -- path of the *r.dat file
Returns: none

class Analyser.**ICF_Reader**
Creates 'SimReader' object for simulation result files of ICF database.

3) Cluster Module

This script contains classes and functions for clustering of data points. **Clustering** is main class for cluster analysis, in which clusters are initialised, checked (**Clustering.check_clusters()**), output to *.tsc file (**Clustering.out()**). The simillarity measurs such as **Clustering.inter_spreads()** and **Clustering.intra_spread()** are used to determine the number of clusters. Three clustering techniques used here are k-means clustering (**Clustering.km()**), Hierarchical Agglomerative clustering (**Clustering.ha()**) and Probabilistic Distance Clustering (**Clustering.pd()**).

Usage: The module runs standalone from a history matching job with *.tsm results file as first argument and clustering algorithm type (km, ha, or pd) as second argument. In addition, **Clustering** object can be created and used from other modules.

Module author: Asaad Abdollahzadeh <Asaad.Abdollahzadeh@gmail.com>

Created: AA111028, *Last Modified:* AA130218.

class Cluster.**Clustering** (*data*, *alg='pd'*, *n_clusters=''*, *metric='euclidean'*, *out_tsc=False*, *cutoff=0.001*, *signif_value=0.9*)

Clustering base class.

Parameters:

- **data** -- results of history matching data read from *.tsm file.
- **alg** -- clustering algorithm, e.g. 'km', 'pd', or 'ha'.
- **n_clusters** -- number of clusters, '' means determined by the algorithm
- **metric** -- distance metric or similarity measure.
- **out_tsc** -- write out clusters to output file, *.tsc.
- **cutoff** -- cutoff value for convergence (shift in clustering centers).
- **signif_value** -- significance value for determining the number of clusters in PDC.

Returns: None

check_clusters ()

checks for empty and unity clusters; remove empty clusters and merge unity to nearest cluster.

out (*job_dir*, *pars=False*, *par_data=False*)

outputs clusters, write clustering results to an ascii *.tsc file.

Parameters:

- **job_dir** -- dob directory path
- **pars** -- list of parameters
- **pars_data** -- parameter value of solutions (points)

Returns: index of minimum penalty.

inter_spreads ()

calculate the spreads within clusters.

intra_spread ()

calculates between clusters spread.

km ()

performs adaptive k-means clustering (KMC) ; self-determines the number of clusters (k).

ha ()

performs Hierarchical Agglomerative clustering (HAC); self-determines the number of clusters (k).

pd ()

performs Probabilistic Distance Clustering (PDC). can self-determine the number

of clusters.

4) Forecaster Module

This script performs forecasting. Pre-requisite is a successful history matching job. class **`Forecast`** initialises and starts a forecasting run. Function **`Forecast.creator()`** of it is class generator for chosen forecasting type. Forward simulation runs for forecasting, if needed, can be written in a 'qsub' job (**`write_qjob()`**) and then submitted (**`submit_job()`**).

`Calc` is base class for uncertainty quantification (UQ) classes. It initialises UQ algorithms, calculate and output the credible intervals (**`Calc.calculate_credible_intervals()`**), and gets simulation data from ensemble of simulation runs (**`Calc.get_sim_data()`**). Following classes (UQ algorithms) derived from **`Calc`**:

- **`NAB`** performs Neighbourhood-Algorithm with Bayes (NAB).
- **`ML`** maximum likelihood using a single model.
- **`DA`** Database Averaging forecasting using large number of models.
- **`MML`** Maximum Likelihood forecasting using multiple models.
- **`MCMC`** Markov Chain Monte Carlo (MCMC) Sampling using K-Nearest-Neighbours approximation.

Usage: The module runs on standalone and accepts up to 11 command line arguments. For help on arguments type: **`Forecaster.py`** -h

Module author: Asaad Abdollahzadeh <Asaad.Abdollahzadeh@gmail.com>

Created: AA100702, *Last Modified:* AA130217.

`Forecaster.`**`spawn`** (*`f`*)
auxiliary function for passing task to nodes in MPI interface.

class `Forecaster.`**`Calc`** (*`path`*, *`sampler`*, *`n_models`*, *`clustering_alg`*, *`n_clusters`*, *`k_nn`*, *`f_cov`*, *`n_burnin`*, *`perfix`*, *`diagnose`*)
base class for UQ algorithms.

Parameters:

- **path** -- path of the job
- **sampler** -- sample type
- **n_models** -- number of models for MC sampling
- **clustering_alg** -- clustering algorithm type, i.e. random (rd), k-means (km), probabilistic-distance (pd), ...
- **n_clusters** -- number of clusters

- **k_nn** -- number of nearest neighbours
- **f_cov** -- covariance factor value for M-H sampling.
- **n_burnin** -- number of solutions in burn-in period.
- **perfix** -- prefix for the output file
- **diagnose** -- verbose for output '*.tsp' files, which contain history of sampling.

Returns: None

calculate_credible_intervals (*data*, *probs*, *text=''*)

calculates credible intervals (Min, P10, P50, P90, and Max) and output them to a *.tsf file.

Parameters: **data** -- simulation results data for samples probs: list of probabilities for each run selected for forecasting text: perfix for header line in *.tsf and *.tsp files.

Returns: none

output_single_run (*data*, *header=''*)

outputs a single run results, i.e. Data & P50 for truth case runs.

Parameters:
- **data** -- simulation results data for samples
- **header** -- header line

Returns: none

get_sim_data (*path*, *run_list=[]*)

extracts simulation data for runs.

Parameters:
- **path** -- directory path for simulation results
- **run_list** -- list of runs for which simulation data is extracted.

Returns: None

get_sim_data2 (*path*, *n_models=None*)

extracts simulation data for runs in IC-Fault database.

Parameters:
- **path** -- directory path for simulation results.
- **n_models** -- number of models to be extracted.

Returns: None

class Forecaster.**NAB** (*path*, *sampler*, *n_models*, *clustering_alg*, *n_clusters*, *k_nn*, *f_cov*, *n_burnin*, *perfix*, *diagnose*)

Neighbourhood-Algorithm with Bayes: Vernonoi cell approximation and Gibbs sampling of likelihoods.

run ()
runs NAB program and calculate credible interval for variables in objective function from a set of runs.

nad_to_misfitparams ()
gets 'misfitparams.dat' ascii file from 'na.nad' binary file.

misfitparams_to_nad (*forward_path*)
Converts 'misfitparams.dat' of non-na algorithms to 'nad' file.

Parameters: **forward_path** -- directory path for files created for/by NAB program.

Returns: None

write_nab_in (*nab_file*)
write out 'nab.in' file needed by 'nab' program.

Parameters: **nab_file** -- path+name for input file for NAB ('nab.in').

Returns: None

class Forecaster.**ML** (*path*, *sampler*, *n_models*, *clustering_alg*, *n_clusters*, *k_nn*, *f_cov*, *n_burnin*, *perfix*, *diagnose*)
Maximum Likelihood forecasting using a single model. Sampler types here are: 'mlb': single best model of ensemble, 'mct': single base (truth) case model (mct).

class Forecaster.**DA** (*path*, *sampler*, *n_models*, *clustering_alg*, *n_clusters*, *k_nn*, *f_cov*, *n_burnin*, *perfix*, *diagnose*)
Database Averaging forecasting using large number of models , e.g. the ensemble estimates the posterior. sampler type is 'daa': a large set of uniform samples.

class Forecaster.**MML** (*path*, *sampler*, *n_models*, *clustering_alg*, *n_clusters*, *k_nn*, *f_cov*, *n_burnin*, *perfix*, *diagnose*)
Maximum Likelihood forecasting using multiple models, currently handles following sampler types: 'mmb': Multiple Maximum Likelihood from n Best Misfitmodels, 'mmc': Multiple Maximum Likelihood from Best Misfit model of k different clusters,

class Forecaster.**MCMC** (*path*, *sampler*, *n_models*, *clustering_alg*, *n_clusters*, *k_nn*, *f_cov*, *n_burnin*, *perfix*, *diagnose*)
Base class for Markov Chain Monte Carlo (MCMC) Sampling using K-Nearest-Neighbours approximation. It handels following sampler types: 'mcm': Gibbs sampling using marginal probabilities, 'mcs': Standard Monte Carlo sampling, 'mhg': Metropolis-Hasting sampling with proposals from the multivariate Gaussian distributions, 'arm': Acceptance-Rejection sampling using proposals from histogram model of best likelihood's clusters, 'nnm': One-at-a-time MH using proposals from marginal probability distributions

knn_approximate(*points*)
performs k-nn approximation.

Parameters: **points** -- array of points (each point with a list of parameter values).

Returns: index array of k-nn solutions, inverse of distance based probabilities (weights), mean distance of k-nn, and estimated log likelihood.

run()

runs MCMC sampling with Nearest neighbour approximation and probabilistic distance clustering for starting points of walks.

class Forecaster.**Forecast** (*path*)

This class initialises and starts a forecasting run.

Parameters: **path** -- directory path of forecasting job.

Returns: None

extend_dataFile()

Extends Simulation Data Files for forecasting purpose.

creator (*sampler*, *n_models*, *clustering_alg*, *n_clusters*, *k_nn*, *f_cov*, *n_burnin*, *perfix*, *diagnose*)

main driver method for Forecasting.

Parameters:

- **sampler** -- sample type, currently 'mlb', 'daa','mlt','nab','mmb','mmc','mcm','nnm','mhg', 'arc', 'mcs' are supported.
- **n_models** -- number of models for MC sampling
- **clustering_alg** -- clustering algorithm type, i.e. random (rd), k-means (km), probabilistic-distance (pd), ...
- **n_clusters** -- number of clusters
- **k_nn** -- number of nearest neighbours
- **f_cov** -- covariance factor value for M-H sampling.
- **n_burnin** -- number of solutions in burn-in period.
- **perfix** -- perfix for the output file
- **diagnose** -- verbose for output '*.tsp' files, which contain history of sampling.

Returns: sampler class

Forecaster.**submit_job** (*qsub_filename*)

submits qsub_file with the run order for the queue.

Parameters: **qsub_filename** -- qsub input file

Returns: None

Forecaster.**write_qjob** (*qsub_filename*, *job_dirs*, *sampler*, *n_models*, *clustering_alg*, *n_clusters*, *n_knn*, *f_cov*, *n_burnin*, *n_repeats*, *diagnose*)

writes a Forecaster.py qsub job.

Parameters:

- **qsub_filename** -- qsub file name
- **job_dirs** -- list of job directories.
- **sampler** -- sample type
- **n_models** -- number of models for MC sampling
- **clustering_alg** -- clustering algorithm type, i.e. random (rd), k-means (km), probabilistic-distance (pd), ...
- **n_clusters** -- number of clusters
- **k_nn** -- number of nearest neighbours
- **f_cov** -- covariance factor value for M-H sampling.
- **n_burnin** -- number of solutions in burn-in period.
- **perfix** -- prefix for the output file
- **diagnose** -- verbose for output '*.tsp' files, which contain history of sampling.

Returns: None

5) Grapher Module

This module reads following tab separated ascii results files and creates charts as selected by arguments in the form of *.png images. Supported files are:

- tab separated misfit result files (*.tsm)
- tab separated forecast result files (*.tsf)
- tab separated sampling result files (*.tsp)
- tab separated clustering result files (*.tsc)

Usage: The script runs as standalone with some arguments passed in command line. Help on supported command line arguments are shown by 'Grapher.py -h' command.

The class **Charts** can be created and used to make charts from other modules.

Module author: Asaad Abdollahzadeh <Asaad.Abdollahzadeh@gmail.com>

Created: AA120414, *Last Modified:* AA130217.

class Grapher.**Charts**

main class for charts. Member functions return a specific chart type.

bar (*xData*, *yData*, *err_bars=None*, *names=None*, *x_title=None*, *y_title=None*, *title=None*, *width=0.35*, *ylim=None*, *hline=None*, *vline=None*, *save_path=None*, *acorr=None*)

returns a bar chart. Chart data, axis and chart title, vertical and horizontal lines and axis limits and save path for image are passed to the function.

line (*xData*, *yData*, *names=None*, *axis_titles=None*, *title=None*, *xlim=None*, *ylim=None*, *hline=None*, *vline=None*, *save_path=None*, *acorr=None*)

returns a line chart. Chart data and series names, axis and chart title, vertical and horizontal lines and axis limits and save path for image are passed to the function.

normal (*xData*, *yData*, *xTitle=None*, *yTitle=None*, *bar_title=True*, *par_ranges=None*, *mean=None*, *cov=None*, *bins1=(250, 250)*, *bins2=(100, 100)*, *title=None*, *save_path=None*)

returns contour and heatmap plots for bivariate Gaussian distribution. Chart data and series names, axis and chart title, parameter ranges, bin sizes= for axes and save path for image are passed to the function.

matrix (*npArr*, *alpha*, *title=''*, *save_path=None*, *bar_title=None*)

returns a matrix chart. Matrix data, chart title and save path for image are passed to the function.

scatter (*npArr*, *color='b'*, *title=''*, *xlim=None*, *ylim=None*, *hline=None*, *vline=None*, *save_path=None*, *bar_title=None*)

returns a scatter chart. Chart data (x and y arrays), axis and chart title, third axis (color array), vertical and horizontal lines and axis limits and save path for image are passed to the function.

scatter2 (*xData*, *yData*, *names=None*, *xlabel=None*, *ylabel=None*, *title=''*, *xlim=None*, *ylim=None*, *hline=None*, *vline=None*, *save_path=None*, *bar_title=None*)

second implementation for scatter chart. x and y arrays, axis and chart title, third axis (color array), vertical and horizontal lines and axis limits and save path for image are passed to the function.

contour (*xData*, *yData*, *zData*, *x*, *y*, *n_bin=50*, *title=None*, *save_path=None*, *bar_title=None*)

returns a contour chart. x, y and z arrays, x and y axis titles, number of bins, chart title and save path for image are passed to the function.

heatmap (*xData*, *yData*, *x*, *y*, *title=None*, *bins=(50, 50)*, *save_path=None*, *bar_title=None*)

returns a heatmap chart. x and y arrays, x and y axis titles, chart title, number of bins for two axes and save path for image are passed to the function.

threeDscatter (*npArr*, *npArr2=None*, *color='b'*, *marker='o'*, *s=25*, *figsize=(8, 6)*, *dpi=80*, *axis_titles=None*, *title=None*, *limits=None*, *bar_title=None*, *save_path=None*, *cm=None*, *norm=None*, *ticks=None*)

returns a three-dimensional scatter chart. 3D array (x, y and z arrays), second optional 3D array, fourth array (color), marker type, figure size and dpi, axes and chart titles, and save path for image are passed to the function.

threeDsurface (*X*, *Y*, *Z*, *points=None*, *title=None*, *zlim=None*, *bar_title=None*, *save_path=None*, *rstride=1*, *cstride=1*, *cmap=<matplotlib.colors.LinearSegmentedColormap object at 0x00000000124D7C50>*, *linewidth=0*, *antialiased=False*, *vmin=None*, *vmax=None*)

returns a three-dimensional surface chart. x, y and z arrays, axes and chart titles, z axis limits and save path for image are passed to the function.

parallel_coordinates (*data_sets*, *data_sets2=None*, *color=None*, *title=None*, *xtitles=None*, *cm=None*, *limits=None*, *save_path=None*)

returns a parallel coordinate chart for multiple variables. Data arrays, axes and chart titles, color array, axes limits and save path for image are passed to the function.

class Grapher.**Graph** (*job_dirs*, *file_type*, *chart_list*, *chart_args*, *perfix*, *show*)

main class for reading tab seperated result files (*.ts*) and creating different charts using above 'Charts' class.

chart_tf (*n_bins=1000*)

plots Rastrigin test function.

chart_tsm ()

reads tab separated misfit result files (*.tsm) and creates runtime, misfit, parameter values vs. generations (or runs) chart.

chart_tsp ()

reads tab separated sampling result files (*.tsp) and creates probability distribution function (pdf), mean convergence, parameter traceplots and their autocorrelation function for MCMC sampling.

chart_tsc2 ()

reads tab separated clustering result files (*.tsc), second implementation, three-d scatter, parallel coordinate and pca charts of clustering results.

chart_tsc ()

reads tab separated clustering result files (*.tsc), first implementation, three-d scatter and parallel coordinates plots

6) MisfitCalculator Module

This module calculates misfit for a single objective function. Class **`MisfitCalculator`** performs this calculation using **`MisfitCalculator.CalcMisfit()`** and one of the PUNQ (**`MisfitCalculator.update_misfit()`**), IC-Fault (**`MisfitCalculator.update_misfit_ICF()`**) or Teal-South (**`MisfitCalculator.update_misfit_TS()`**)misfit definition types.

Usage: The module runs as standalone from a history matching job as current directory and a leap* run as first argument. In addition, **`MisfitCalculator`** object can be created and used from other modules.

Module author: Asaad Abdollahzadeh

<Asaad.Abdollahzadeh@gmail.com> Created: 18/08/2008 by Mike Christie, *Last Modified:* AA130225.

class `MisfitCalculator.`**`MisfitCalculator`** (*`obj`*, *`dbase`*, *`obs_db`*,*`sim_db`*)

Misfit calculator class; contains codes for misfit calculations from observation and simulation objects.

Parameters:

- **obj** -- objective function name.
- **dbase** -- shelve database created from 'ensemble*.def' file.
- **obs_db** -- observation data in form of ObsReader object.
- **sim_db** -- simulation data in form of SimReader object.

Returns: None

`CalcMisfit(`*`out_dir=""`*`)`

calculates misfit.

Parameters: **out_dir** -- output misfit calculation tables into *.tsr file or not.

Returns: None

`penalize_control_rat()`

penalty for wells controlled by rate that not producing observed value (due to BHP constraint).

Parameters: None

Returns: misfit penalty

update_misfit (*data_obs*, *data_sim*, *data_sig*, *data_wgt*)

calculates misfit for a single data point for PUNQ-like misfit definition.

Parameters:

- **data_obs** -- observation value
- **data_sim** -- simulation value
- **data_sig** -- sigma value
- **data_wgt** -- weigth value

Returns: misfit value

update_misfit_TS (*data_obs*, *data_sim*, *data_sig*, *data_wgt*)

calculates misfit for a single data point for Teal-South-like misfit definition.

Parameters:

- **data_obs** -- observation value
- **data_sim** -- simulation value
- **data_sig** -- sigma value
- **data_wgt** -- weigth value

Returns: misfit value

update_misfit_ICF (*data_obs*, *data_sim*, *data_sig*, *data_wgt*)

calculates misfit for a single data point for IC-Fault-like misfit definition.

Parameters:

- **data_obs** -- observation value
- **data_sim** -- simulation value
- **data_sig** -- sigma value
- **data_wgt** -- weigth value

Returns: misfit value

7) ObsReaders Module

This script reads observation data files required for misfit calculation scripts. **ObsFactory** class creates an ObsReader object based on simulator type (e.g. ECL, VIP). **HistDB** is base class for observation file reader, from

which, **ECL_Reader** and **VIP_Reader** classes are derived which read Eclipse and VIP observation files respectively.

Usage: The module can be run as standalone from a history matching job ('uqs_df' shelf file) or ObsReader object can be created and used from other modules (e.g. **Runner.py**).

Module author: Asaad Abdollahzadeh <Asaad.Abdollahzadeh@gmail.com>

Created: 18/08/2008 by Mike Christie, *Last Modified:* AA130225.

ObsReaders.**line_iterator** (*lines*)

create an iterator from readlines() list.

Parameters: **lines** -- lines of string.

Returns: lines iterator

class ObsReaders.**ObsFactory**

class factory based on simulator type.

Parameters:

- **simulator** -- simulator name.
- **obs_file_list** -- list of observation files including path.
- **obj_functions** -- dictionary of objective functions as in 'ensemble*.def' file.

Returns: class for reading observation files.

class ObsReaders.**HistDB**

base class for observation file reader.

ObsVars ()

returns list of variables in the observation database created from observation files.

Parameters: **lines** -- lines of string.

Returns: list of variables

ObsValue (*var*, *date*)

returns the value of observation variable in a specified date/time.

Parameters:

- **variable** -- variable name
- **date** -- observation date or time

Returns: value of variable in specified date/time.

class ObsReaders.**ECL_Reader** (*file_list*, *obj_functions*)

class reads ECL observation files.

Parameters:

- **file_list** -- the list of observation file.
- **obj_functions** -- dictionary of objective functions as in the 'ensemble*.def' file.

Returns:

None

ReadSigFile (***lines***)

reads and stores observation data from a 'PUNQ' like observation file.

Parameters: **lines** -- the list lines as read by python's readlines().

Returns: None

ReadObsFile (***lines***)

reads and stores observation data from an Eclipse observation files (*.obs).

Parameters: **lines** -- the list lines as read by python's readlines().

Returns: None

ObsDates (***var***)

returns dates/times from an observation dictionary created for Eclipse observation files (*.obs).

Parameters: **var** -- the variable name in form of 'classtype:class_name:var_name'.

Returns: None

SigValue (***var***, ***time***)

returns sigma-weigth tuple.

Parameters:

- **var** -- the variable name in form of 'classtype:class_name:var_name'.
- **time** -- specified observation date/time.

Returns: tuple of sigma and weigth.

class ObsReaders.**VIP_Reader** (*file_list*, *obj_functions*)

class reads VIP observation files.

Parameters:

- **file_list** -- the list of observation file.

• **obj_functions** -- dictionary of objective functions as in the 'ensemble*.def' file.

Returns: None

ReadVIPObsFile (*obs_file*)

reads and stores observation data from an VIP observation files (*.obs).

Parameters: **obs_file** -- the observation file.

Returns:

None

ObsSum (*new_var*, *myVars*, *dates*)

return observation result value for a sum variable, e.g liquid rate.

Parameters:

• **new_var** -- variable name for new sum variable.

• **myVars** -- list of variables to be sum.

• **dates** -- dates for which new variable to be computed.

ObsDates (*var*)

returns dates/times from an observation dictionary created for VIP observation
files.

Parameters: **var** -- the variable name in form of 'classtype:class_name:var_name'.

Returns: None

SigValue (*var*, *date*)

returns sigma-weight tuple.

Parameters:

• **var** -- the variable name in form of 'classtype:class_name:var_name'.

• **date** -- specified observation date/time.

Returns: tuple of sigma and weight

8) Runner Module

This class initialises and starts an experiment for history-matching (covered in this thesis) and development optimisation (not covered in this thesis). **Runner** is main class for initialising the experiments. It also:

• reads in and executes the experiment configuration files (e.g. 'ensemble.def')
in **Runner.ensemble_exec()**,

- creates a Python shelve database (i.e. 'uqs_db') out of them in **Runner.write_uqs_db()**,
- writes out NA input file ('write_na_in') in **Runner.write_na_in()**,
- creates a Python shelve database (i.e. 'obs_db') from the observation files in **Runner.setup_obs_db()**,
- creates a text file for uncertainty parameter ranges (i.e. 'par_ranges.dat') from the configuration file in **Runner.setup_pars()**,
- runs select_models of **Analyser.py** and a text file for model/probability of selected model used in devopt in **Runner.setup_models()**,
- read initial population to resume from another experiment in **Runner.aquire_init_population()**,
- creates qsub bash file for history matching experiments in **Runner.write_job()**,
- creates qsub bash file for development optimisation experiments in **Runner.write_dev_job()**,
- and finally submits created Qsub job file in **Runner.submit_job()**. **Usage:** The script runs as standalone with some arguments passed in command line. Help on supported command line arguments are shown by 'Runner.py -h' command.

Module author: Asaad Abdollahzadeh <*Asaad.Abdollahzadeh@gmail.com*> *Created:* 18/08/2008 by Mike Christie, *Last Modified:* AA130225.

class Runner.**Runner**

This class initialises and starts a run for history-matching. It gives default values to all required details for a run. These can be over-ridden by the ensemble.def file.

ensemble_exec (***path***, ***ensemble_def***)

reads in the ensemble.def file and executes each line to over-ride defaults configured in initialise.

Parameters:
- **path** -- the path of the job file
- **ensemble_def** -- name of the ensemble file in the path directory

Returns: none

write_uqs_db (*path*, *dbase_file*)
creates a 'uqs_db' file based on the values in 'ensemble.def'.

Parameters:
- **path** -- the path of the job file.
- **dbase_file** -- the name of job specification and database file (e.g. uqs_db).

Returns:

None

write_na_in ()
creates the in file for an NA run base on the values in 'ensemble.def'

setup_obs_db (*path*)
creates a 'obs_db' file based on the observation reader object, which reads in the
history file(s) in 'ensemble.def'.

Parameters: **path** -- the path of the job file

Returns:

None

setup_pars (*path*, *file_name*)
creates a 'par_ranges.dat' file based on parameter ranges in 'ensemble.def'.

Parameters:
- **path** -- the path of the job file.
- **file_name** -- the output filename (e.g. 'par_ranges.dat').

Returns:

None

setup_models (*path*)
creates 'models.dat' file based on results of a history-matching job.

Parameters: **path** -- the path of the history-matching job

Returns:

None

aquire_init_population ()
read initial population from either a file, job or tdrm structure indicated in
'ensemble.def' by "init_population_in".

write_job (*path*, *qsub*, *qsub_filename*)

creates qsub_file with the run order for the queue.

Parameters:
- **path** -- the path of the job file
- **qsub** -- Qsub job queuing type (e.g. 'PBS' or 'SGE').
- **qsub_filename** -- Qsub job file name

Returns: None

write_alg_cmd ()
writes algorithm's command line.

write_dev_job (*path*, *base_dir*, *qsub_filename*)
creates qsub_file for a development optimisation job.

Parameters:
- **path** -- the path of the job file
- **base_dir** -- base directory path for the devopt job
- **qsub_filename** -- Qsub job file name

Returns: none

submit_job (*qsub_filename*, *n_repeat*, *wait_job*)
submits created Qsub job file for a history matching.

Parameters:
- **path** -- the path of the job file
- **n_repeat** -- number of repeat for a history matching job with exact same algorithm parameters.
- **qsub_filename** -- Qsub job file name

Returns: none

9) RunSimMisfit Module

This script is called by sampler/optimisation algorithm after sampling done, then it configures simulation run, calculates misfit and pass it to the Sampler. pre-requisite are UQ sampler and Simulator .

ObsFactory class creates an ObsReader object based on simulator type (e.g. ECL, VIP). **HistDB** is base class for observation file reader, from which, **ECL_Reader** and **VIP_Reader** classes are derived which read Eclipse and VIP observation files respectively.

Usage: The module cab run as standalone from a history matching job ('uqs_df' shelf file) or **MisfitCalculator** object can be created and used from other modules (e.g. **Runner.py**).

Module author: Asaad Abdollahzadeh

<Asaad.Abdollahzadeh@gmail.com> Created: AA100626, *Last Modified:* AA130217.

class RunSimMisfit.**SimMisfit**
(*path*) Calculate objective function for a simulation run.

Parameters: **path** -- directory path of the experiment.

Returns: None

run_tf (*temp_dir*)
reads 'params.in' file and calculate misfit for each objective in test functions

Parameters: **temp_dir** -- temp directory containing 'params.in' file

Returns: None

run (*temp_dir*)
reads 'params.in' file, runs the simulator and calculate misfit for each objective in
simulation runs.

Parameters: **temp_dir** -- temp directory containing 'params.in' file

Returns: None

calculate_misfits ()
Creates MisfitCalcCreater objects and calculate misfit for each objective.

create_misfit (*v_dict*, *run_time*)

Creates 'misfit.dat' and 'misfitparams.dat' files and outputs the objective functions to them.

Parameters:
- **v_dict** -- dictionary of parameters-values.
- **run_time** -- runtime of the simulation run.

Returns: None

10) SimReaders Module

This script reads simulation results files required for misfit calculation scripts. Class **sim_reader** is the base class, from which, reader class are derived for different simulators (e.g. ECL, VIP). Class eclipse**_reader** reads Eclipse summary spreadsheet files (*.RSM) and class *:class:`.vip_reader` reads VIP summary spreadsheet files (*.sss).

Usage: The module can be run as standalone from a history matching job ('uqs_db' shelf file) or sim_reader object can be created and used from other modules (e.g. **RunSimMisfit**).

Module author: Asaad Abdollahzadeh <Asaad.Abdollahzadeh@gmail.com>

Created: 18/08/2008 by Mike Christie, *Last Modified:* AA130225.

SimReaders.**iterator** (*lines*)
iterates a list of lines.

class SimReaders.**sim_reader** (*file_list*, *count_zero_history=True*)
Base class for sim reader; reads simulation results based on the extensions 'RSM'
(Simulator type)

Parameters:
- **file_list** -- Simulation result file list.
- **count_zero_history** -- ether account for the zero vale history results.

Returns: SimReader object

SimValue (*var*, *time*)
returns simulation result value for a variable and date.

Parameters:
- **var** -- variable name in form of 'classtype:class_name:var_name'.
- **time** -- time/date.

Returns: the value of variable at specified time/date.

SimLastDates ()
returns last date in simulation result file.

SimSum (*new_var*, *myVars*, *dates*)
returns simulation result value for a sum variable, e.g. liquid rate.

Parameters:
- **new_var** -- variable name for new sum variable.
- **myVars** -- list of variables to be sum.
- **dates** -- dates for which new variable to be computed.

Returns: None

SimVars ()
returns list of variables in simulation result file.

InterpSimValue (*var*)

returns interpolated value of a variable in dates not reported in simulation result
file.

Parameters: **var** -- variable name in form of 'classtype:class_name:var_name'.

Returns: interpolated values

var_check (*class_type*, *var_name*)

check if a vriable exists in simulation result discionary.

Parameters:

- **class_type** -- variable class type e.g. 'weel' and 'field'.
- **var_name** -- variable name in form of 'classtype:class_name:var_name'.

Returns: None

class SimReaders.**eclipse_reader** (*file_list*, *count_zero_history=True*)

Creates 'SimReader' object for simulation result files with extensions 'RSM'
(Eclipse Simulator)

Parameters: **file_list** -- Simulation result file list

Returns: SimReader object

CreateResultsDB (*file_name*)

reads simulation result from an Eclipse *.RSM file.

Parameters: **file_name** -- Simulation result file name including path

Returns: None

Create_Header_Table (*line_iter*)

reads header line from Eclipse simulation result and creates a header list.

Parameters: **line_iter** -- iterator of the Simulation result file.

Returns: headers list

Read_Results_Page (*headers*, *line_iter*)

reads data from Eclipse simulation result files and creates a dictionary of
variable-date-values.

Parameters:

- **headers** -- list of headers in RSM files(dates, variables etc)

• **line_iter** -- iterator of the Simulation result file.

Returns: None

SimDates (*var*)

returns list of dates for a variable.

Parameters: **var** -- variable (in form of 'classtype:class_name:var_name').

Returns: sorted list of dates

class SimReaders.**vip_reader** (*file_list*, *count_zero_history=True*)

class SSS_Reader: Creates 'SimReader' object for simulation result files with
extensions 'SSS' (VIP Simulator).

Parameters: **file_list** -- Simulation result file list

Returns: SimReader object

CreateResultsDB (*file_name*)

reads simulation result from an VIP *.SSS file.

Parameters: **file_name** -- Simulation result file name including path

Returns: None

Create_Header_Table (*line*, *class_type*)

reads header line from VIP simulation result and creates a header list.

Parameters: **line_iter** -- iterator of the Simulation result file.

Returns: headers list

Create_Units (*col_ref*, *line*)

returns list of dates for a variable.

Parameters:

• **col_ref** -- dictionary of headers and thier respected
column reference
number.

• **line** -- line containing the units.

Returns: list of variable units

Read_Results_Page (*line_iter*, *class_type*, *headers*)

reads data from VIP simulation result files and creates a dictionary of
variable-date-values.

Parameters:

• **headers** -- list of headers in SSS files(dates, variables etc)

- **class_type** -- class type of the file; e.g. field, wells and etc.
- **line_iter** -- iterator of the Simulation result file.

Returns: None

SimDates (*var*)
returns list of dates for a variable.

Parameters: **var** -- variable (in form of 'classtype:class_name:var_name').

Returns: sorted list of dates

11) Stats Module

This script contains a collection of statistical and mathematical functions which use
Numpy or/and **Scipy** modules.

Usage: These are common utility functions used across other scripts. The whole module (**Stats.py**) is imported.

Module author: Asaad Abdollahzadeh <Asaad.Abdollahzadeh@gmail.com>

Created: AA120720, *Last Modified:* AA130217.

Stats.**peakdet** (*v*, *delta*, *x=None*)
detects peaks in a list of values

Parameters:
- **v** -- vector of values
- **delta** -- a positive value
- **x** -- second list to return value from; if None the index of 'v' are returened.

Returns: VCAD distance

Stats.**VCAD** (*a*, *b*)
Returns the cosine of the angle between two Points.

Parameters:
- **a** -- first point
- **b** -- second point

Returns: VCAD distance

Stats.**getDistance** (*a*, *b*)
Get the Euclidean distance between two Points

Parameters:
- **a** -- first pointp
- **b** -- second point

Returns: Sqrt of sum distance: sqrt(sum((a[i]-b[i])^2) for all i)

Stats.**elbow_distance** (*K*, *N*, *F*, *F_start*, *F_end=0.0*) Get elbow distance of a curve

Parameters:

- **K** -- The number of points
- **F** -- the curve list of points
- **N** -- the ultimate number of points
- **F_start** -- first value of curve function
- **F_end** -- ultimate value of curve function

Returns: elbow distance from the curve

Stats.**PCA** (*A*, *cutoff=10.0*)

performs principal components analysis (PCA) on the n-by-p data matrix A Rows of A correspond to observations, columns to variables.

Parameters:

- **A** -- A n-by-p data matrix
- **cutoff** -- cutoff value

Returns: eigenvectors (is a p-by-p matrix, each column containing coefficients for one principal component), eigenvalues (a vector containing the eigenvalues of the covariance matrix of A), and the score(the principal component scores; that is, the representation of A in the principal component space).

Stats.**COV** (*data_arr1*, *data_arr2=None*, *rowvar=1*, *bias=0*)

returns covariance matrix between two arrays (using Numpy python library)

Parameters:

- **data_arr1** -- first 1-D or 2-D array containing multiple variables and observations.
- **data_arr2** -- second 1-D or 2-D array containing multiple variables and observations.
- **rowvar** -- if non-zero (default), then each row represents a variable, otherwise, each column represents a variable.
- **bias** -- if zero (default), normalisation by 'N-1' (default), if '1', then normalization is by 'N'

Returns: The covariance matrix of the variables.

Stats.**KLD** (*data_arr1*, *data_arr2*)

Kullback-Leibler Distance between two arrays (using Scipy python library).

Parameters:
- **data_arr1** -- first vector
- **data_arr2** -- second vector

Returns: KLD distance of two vectors

Stats.**KDE** (*data_arr*)

The Gussian kernel density of a numpy array (using Scipy python library).

Parameters: **data_arr** -- array of data

Returns: Array of KDE estimates

Stats.**DIST** (*data_arr1*, *data_arr2=None*, *metric='euclidean'*) Computes distance between each pair of data vectors in two Cartesian arrays (using Scipy python library).

Parameters:
- **data_arr1** -- data array 1
- **data_arr2** -- data array 2
- **metric** -- distance metric type, e.g. 'euclidean'

Returns: Array of distances

Stats.**MDI** (*points*, *values*, *grid*, *method='nearest'*) Multivariate data interpolation (griddata)(using Scipy python library).

Parameters:
- **points** -- numpy data array of NxD
- **values** -- values at the points
- **grid** -- regular grid for which interpolation should be done.
- **method** -- iterpolation method, i.e. 'linear', 'nearest', 'cubic'

Returns: interpolated_data: interpolated values at the regular grid

Stats.**KNN** (*data_arr*, *data_arr2=None*, *metric='euclidean'*, *k=3*) calculates and return the 'mean' of the a list of real values (using Scipy python library).

Returns: data_arr: first numpy data array of NxD

Parameters:
- **data_arr2** -- second numpy data array of NxD
- **distance metric function type** (*metric-*) -- 'euclidean', 'sqeuclidean', 'seuclidean', 'cityblock', 'mahalanobis'
- **k** -- number of nearest neighbours

Returns: index array of k nearest neighbours, distance matrix of k-nn, inverse of distance based probabilities (weights).

`Stats.`**`HDD`** (*`data_arr`*, *`n_bins=10`*, *`density=True`*, *`ranges=None`*)

creates histogram modelfor an numpy array.

Parameters:

- **data_arr** -- numpy data array of MxN
- **n_bins** -- numberof bins
- **density** -- boolean for output densities or frequencies
- **ranges** -- array of lower and upper bound tuples; if None calculated from the data

Returns: histogram model and related bin edges.

`Stats.`**`AIC`** (*`data_arr`*, *`scale=True`*)

Bayesian model selection based on Akaike's Information Criterion (AIC) for a set of models it tries minimizes the Kullback-Leibler distance: AIC = -2*log-likelihhod+2p (p is number of parameters) (Akaike 1973)

Parameters:

- **data_arr** -- 1D array of values
- **scale** -- True: scale data or not

Returns: likelihood probabilities

`Stats.`**`curvature`** (*`points`*)

calculates curvature of a set of 3 points with consecutive x value of 1 unit difference.

Parameters: **points** -- three points

Returns: radious of curvature

`Stats.`**`percentile`** (*`my_list`*, *`percent`*, *`key=<function <lambda> at 0x0000000007486048>`*)

Finds the percentile of a list of values.

Parameters:

- **my_list** -- a list of values
- **percent** -- a float value from 0.0 to 10.0.
- **key** -- optional key function to compute value from each element of N.

Returns: the percentile of the values.

`Stats.`**`mean`** (*`my_list`*)

calculates and return the 'mean' of the a list of real values.

Parameters: **my_list** -- the list

Returns: mean

`Stats.`**`divide_list`** (*`my_list`*, *`n`*)

Produces an iterator over subsections of maximum length n of the list.

Parameters:
- **my_list** -- the list
- **n** -- length of sublists

Returns: list of sublists

`Stats.`**`weighted_average`** (*`list1`*, *`list2`*)

returns average of list1 weighted by list2.

Parameters:
- **list1** -- the list to be averaged
- **list2** -- the list used for weighting

Returns: average number

`Stats.`**`histogram`** (*`my_list`*, *`ranges`*)

calculates and return the histogram model of a list of real values

Parameters:
- **my_list** -- the list
- **ranges** -- histogram bin ranges

Returns: my_dic: dictionary of list members and frequencies

`Stats.`**`counter`** (*`my_list`*)

counts a list for its items and return dictionary 'item:count'

Parameters: **my_list** -- the list

Returns: my_dic: dictionary of list items and counts

`Stats.`**`bootstrap`** (*`sample`*, *`n_samples=500`*, *`statfunc=<function mean at 0x0000000007486128>`*)

Performs resampling from sample with replacement, gathers statistic in a list computed by statfunc on the each generated sample.

Parameters:
- **sample** -- input sample of values: independent and identically (iid) distributed with unknown statistical distribution.
- **nsamples** -- number of samples to generate
- **samplesize** -- sample size of each generated sample

- **statfunc** -- statistical function to apply to each generated sample.

Returns: list of the mean of the resampled bootstrap 'samples'.

`Stats.`**`rollete_wheel`** (*`in_list`*, *`p`*)
performs rollete wheel sampling.

Parameters:
- **in_list** -- list of real values to be sampled.
- **p** -- quartile value, e.g. 25, 50 and 75.

Returns: quantile probability distribution.

`Stats.`**`linear_interpolate`** (*`x`*, *`y`*, *`order=1`*)
returns a function that does linear interpolation of data in two vectors.

Parameters:
- **x** -- first vector
- **y** -- second vector

Returns: function which handles linear interpolation

12) TF Module

This script contains standard test functions used in history matching and uncertainty quantification. Functions include:

- **`TF.ZDT()`** performs Zitzler-Deb-Thieless(ZDT) test suit (functions 1-4).
- **`TF.Sphere()`** performs Sphere test function.
- **`TF.Sphere()`** performs Sphere test function.
- **`TF.Rastrigin()`** performs Rastrigin test function.
- **`TF.Griewank()`** performs Griewank test function.
- **`TF.Rosenbrock()`** performs Rosenbrock test function.
- **`TF.Ackley()`** performs Ackley test function.
- **`TF.mvg()`** performs Bivariate Gaussian (BVG) test function.

Usage: These are common utility functions used across other scripts. The whole module (**`Utils`**) is imported in and used by the modules.

Module author: Asaad Abdollahzadeh <Asaad.Abdollahzadeh@gmail.com>

Created: AA111027, *Last Modified:* AA130225.

`TF.`**`ZDT`** (*`x`*, *`id`*)
Zitzler-Deb-Thieless(ZDT) test functions for multiobjective optimisation. Currently
functions 1-4 are implemented.

Parameters:
- **x** -- vector of parameter values
- **id** -- test function number

Returns: string containing the value of multiple objectives

`TF.Sphere` (*x*)
returns Sphere (SP) or De Jong's test function value for vector 'x'; it has many local
minima and one global minimum at (0,0), no interaction between variables.

`TF.Rastrigin` (*x*, *A=10*)
returns Rastrigin (RR) test function: has many local minima and one global
minimum at (0,0), no interaction between variables.

`TF.Griewank` (*x*)
returns Griewank (GW) test function for vector 'x'; it consists of many local optima
and has weak interaction between variables.

`TF.Rosenbrock` (*x*)
returns Rosenbrock (RB) test function: highly nonlinear and symmetric around
quite a flat curved valley. Variables are strongly correlated.

`TF.Ackley` (*x*)
returns Ackley (AC) test function for vector 'x'; it is highly nonlinear and
symmetric around quite a flat curved valley. Variables are strongly correlated.

`TF.mvg` (*x*)
returns Mulitivariate Gaussian (MVG) test function for vector 'x'; it is used as a test
funtion for MCMC sampling algorithms.

`TF.TF_driver` (*TF*, *par_line*)
main driver for the test functions.

Parameters:
- **TF** -- test function type
- **par_line** -- string containing the input parameters

Returns: string containing the value of single/multiple objectives.

13) Utils Module

This script contains a collection of common functions used by other modules.

Usage: These are common utility functions used across other scripts. The whole module (`Utils`) is imported in and used by the modules.

Module author: Asaad Abdollahzadeh <Asaad.Abdollahzadeh@gmail.com>

Created: AA100503, *Last Modified:* AA130225.

`Utils.`**`mean`** (*`my_list`*)
calculates and return the 'mean' of a list of real values.

Parameters: **my_list** -- the list

Returns: the mean value

`Utils.`**`sigma`** (*`my_list`*)
calculates and return the 'standard deviation' of a list of real values.

Parameters: **my_list** -- the list

Returns: the sigma value

`Utils.`**`normalise`** (*`my_list`*, *`l_bound=0.0`*, *`u_bound=10.0`*)
normalises a list based on a predefined lower and upper bound values.

Parameters:

- **my_list** -- the list
- **l_bound** -- lower bound
- **u_bound** -- upeer bound

Returns: normalised list

`Utils.`**`filter`** (*`my_list`*, *`filter_list`*)
filters a list for items in another list.

Parameters:

- **my_list** -- the list
- **filter_list** -- the list of items to be removed from my_list

Returns: filtered list

`Utils.`**`curvature`** (*`points`*)
calculates curvature of a set of 3 points with consecutive x value of 1 unit
difference.

Parameters: **points** -- three points

Returns: radious of curvature

`Utils.`**`percentile`** (*`my_list`*, *`percent`*, *`key=<function <lambda> at 0x0000000006FE0C88>`*)
finds the percentile of a list of values.

Parameters:

- **my_list** -- a list of values
- **percent** -- a float value from 0.0 to 10.0.
- **key** -- optional key function to compute value from each element of N.

Returns: the percentile of the values

Utils.**copyfiles** (*dir_from*, *dir_to*, *file_list*)
copies a list of files from one directory to another. If the copy instruction fails, the error is output.

Parameters:
- **dir_from** -- The directory to copy the files from
- **dir_to** -- The directory to copy the files to
- **file_list** -- The list of files to be copied

Returns: None

Utils.**process_files** (*cur_dir*, *temp_dir*, *edit_file_list*, *variable_dict*, *search_char_s*, *search_char_e*, *python_inc_list*=**[**, **]**)
reads in files that will be edited, processes, and writes out to temp directory.

Parameters:
- **cur_dir** -- current directory path
- **temp_dir** -- temp directory path
- **edit_file_list** -- Edit files list
- **variable_dict** -- Variables name/value pairs dictionary
- **srch_char_s** -- Start search character, usually '{'
- **srch_char_e** -- End search character, usually '}'
- **python_inc_list** -- list of included Python script files.

Returns: total_misfit: Total misfit of Objective Function

Utils.**process_line** (*line*, *variable_dict*, *search_char_s*, *search_char_e*)
takes a line of text from edit_files and processes it for the parameters.

Parameters:
- **line** -- text line
- **variable_dict** -- dictionary of the parameters/values
- **search_char_s** -- search start character '{'
- **search_char_e** -- search end character '}'

Returns: start point of processed text and the processed line.

Utils.**update_dict** (*par_list*, *infile*)
reads in a dictionary file and loads them into the dict.

Parameters:
- **par_list** -- the current parameter list
- **infile** -- the file to be read (already in open state)

Returns: new dictionary with values for the parameters

`Utils.`**`run_simulator`** (*`sim_command`*, *`sim_data_file`*, *`solution_number`*, *`max_sim_time`*)

runs the simulator.

Parameters:

- **sim_command** -- Simulator run command, gettin from uqs_db
- **sim_data_file** -- Simulation data file name without extension
- **solution_number** -- solution (run) serial number
- **max_sim_time** -- maximum runtime for simulations

Returns: None

`Utils.`**`ic_fault`** (*`throw`*)

in IC-Fault model, it runs 'runD' program to generate 'ZCORN.INPUT' file
(includes file for IC-Fault model), creates text to include created 'ZCORN.INPUT' file.

Parameters: **throw** -- string value of the throw of ICF fault

Returns: text to include 'ZCORN.INPUT' plus runs 'runD' program to create 'ZCORN.INPUT'

`Utils.`**`choose_discrete_var`** (*`range`*, *`list`*)

choose a discrete variable from a list.

Parameters:

- **range** -- Range
- **list** -- list to select from.

Returns: list: list

`Utils.`**`parse_range`** (*`astr`*)

parses a string for directory paths

Parameters: **astr** -- string to be parsed

Returns: sorted list of absolute directory path

`Utils.`**`kdtree`** (*`point_list`*, *`depth=0`*)

creates a kde using an efficient kdetree algorithms.

Parameters:

- **point_list** -- list of points
- **depth** -- the maximum depth of tree

Returns: node of tree.