BASIC RESERVOIR HISTORY MATCHING BY REINFORCEMENT LEARNING FUNDAMENTALS

BY

Khalid Labib Alsamadony

A Thesis Presented to the
DEANSHIP OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

PETROLEUM ENGINEERING

December 2019

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN- 31261, SAUDI ARABIA

DEANSHIP OF GRADUATE STUDIES

This thesis, written by **Khalid Labib Alsamadony** under the direction of his thesis advisor and approved by his thesis committee, has been presented and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE IN PETROLEUM ENGINEERING.**

Dr. Qinzhuo Liao
(Advisor)

Dr. Dhafer A. Al Shehri
Department Chairman

Dr. Abeeb Awotunde
(Member)

Dr. Salam A. Zummo
Dean of Graduate Studies

Dr. Abdulazeez Abdulraheem
(Member)

22/12/19

Date

Dedication

I dedicate this work to my mother and father

ACKNOWLEDGMENTS

I would like to thank Dr. Qinzhuo Liao for his support, Dr. Abeeb Awotunde and Dr. Abdulazeez Abdulraheem for the valuable discussions, their time and support. I am thankful to KFUPM and the college of petroleum engineering and geoscience (CPG) for this great opportunity. I am grateful to my family for the continuous encouragement.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

HM	:	History Matching
AHM	:	Automatic History Matching
RL	:	Reinforcement Learning
TD	:	Temporal Difference
EnKF	:	Ensemble Kalman Filter
PCA	:	Principle Component Analysis
RL	:	Reinforcement Learning
R	:	Rewards
S	:	Set of states
A	:	Set of actions
ε	:	Probability of taking a random action
α	:	Step size (Learning rate)
γ	:	Discount-rate parameter
RMSE	:	Root Mean Square Error

ABSTRACT

Full Name : KHALID LABIB ABDELMEGID ALSAMADONY

Thesis Title : BASIC RESERVOIR HISTORY MATCHING BY REINFORCEMENT LEARNING FUNDAMENTALS

Major Field : Petroleum Engineering

Date of Degree : January 2020

History Matching (HM) is the main step in any reservoir simulation study to get reliable predictions from simulation models. Due to the disadvantages of manual HM, several automatic HM (AHM) methods were introduced in the past. However, each one has its own advantages and limitations depending on the properties of the HM problem. Reinforcement learning (RL) is a family of methods that learns by interacting with the environment to select the best action at a particular situation. Tabular temporal difference (TD) methods are considered as RL basics, and include Q-learning, SARSA, expected SARSA and double Q-learning. RL was successfully applied in various fields such as computer science. In petroleum engineering, it was implemented to improve reservoir management by optimizing water flooding operation and steam injection.

This study aims to introduce RL as an AHM method by applying some basic RL methods to solve a simple HM problem and to evaluate the potential of RL for HM. Additionally, the results of the Ensemble Kalman Filter (EnKF) are presented to compare the performance of tabular TD methods to one of the available AHM method in the literature.

The results show that all the applied tabular temporal difference methods can solve the studied HM problem. Although they are inefficient compared to the EnKF, they achieved comparable results to those of the EnKF in matching the observations and the true models.

ملخص الرسالة

الاسم الكامل: خالد لبيب عبد المجيد السمدوني

عنوان الرسالة: مطابقة ماضي المكامن باستخدام مبادئ التعليم المعزز

التخصص: هندسة البترول

تاريخ الدرجة العلمية: يناير 2020

مطابقة الماضي هي الخطوة الأساسية في أي مشروع محاكاة للمكامن، للتمكن من الحصول على نتائج مستقبلية صحيحة من نماذج المحاكاة. هنالك العديد من الطرق الآلية لمطابقة الماضي التي تم تطويرها سابقا بسبب مساوئ مطابقة الماضي اليدوية، ولكن هذة الطرق الآلية لها مميزات وعيوب بناء على خصائص مسألة مطابقة الماضي.

التعليم المعزز يحتوي على مجموعة من الطرق التي تتعلم عن طريق التفاعل مع بيئتها مثل الفرق الزمني الذي هو من أساسيات هذا التعليم. هناك العديد من المجالات التي شهدت نجاح تطبيق طرق التعليم المعزز مثل علوم الحاسب وهندسة المترول، فمثلا في هندسة البترول تم تطبيق هذه الطرق بنجاح في إدارة المكامن عن طريق الوصول للضخ الأمثل للماء والبخار في المكامن.

هذه الدراسة تهدف الي استخدام بعض مبادئ التعليم المعزز في مسألة بسيطة لمطابقة الماضي لتقييم امكانية استخدامها مستقبلا كطريقة مطابقة ماضي آلية، ولذلك تم تطبيق إحدى الطرق المستخدمة في مطابقة الماضي وهي مجموعة مرشح كالمان للمقارنة بينهم.

تشير نتائج هذه الدراسة الي تمكن جميع طرق التعليم المعزز المستخدمة من مطابقة الماضي ولكنها تعاني من بطء شديد مقارنة بمجموعة مرشح كالمان

CHAPTER 1

INTRODUCTION

1.1 Problem definition

History Matching (HM) is necessary to update simulation model with true inputs, thus reliable forecast, planning and reservoir management can be attained. HM is an inverse and data assimilation problem that can be described in an optimization or Bayesian framework. Forward problem (e.g. reservoir simulator) calculates outputs using model inputs. Oppositely, inverse problem infers model inputs from observations (measurements). For example, forward problem is the calculation of reservoir pressures and rates (model outputs) based on its permeability, and porosity values (formation properties or model inputs). On the other hand, inverse problem (history matching) is the estimation of formation properties (model inputs) of a reservoir model using rates and pressures measurements from production, injection, and observation wells. HM goal is to find reservoir model inputs vector **m** that satisfies $\mathbf{f(m) = d_{obs}}$, where f ( ) is the reservoir simulator, f(m) is the model outputs vector and $\mathbf{d_{obs}}$ is the observations vector. In other words, HM aims to minimize the mismatch $|f(m) - \mathbf{d_{obs}}|$ between model outputs (reservoir simulator results) and observations, so that reservoir model can be reliable and accurate for future predictions.

1.2 Manual and Automatic History Matching

HM can be classified to manual HM and automatic HM (AHM), which sometimes called assisted HM particularly for the methods that need expert judgment from time to time during HM process. In manual HM, reservoir engineers modify model inputs (manual trial and error) to minimize the mismatch using a workflow, engineering judgment and experience. In AHM, efficient mathematical representation of reservoir model (parametrization) is usually implemented besides the programmed algorithm (AHM method) that minimizes the mismatch (objective function) automatically. These papers (Oliver & Chen, 2010) & (Rwechungura, Dadashpour, & Kleppe, 2011) & (Cancelliere, Verga, & Viberti, 2011) reviewed several parametrization and HM methods.

1.2.1 Manual History Matching

Manual HM is the earliest HM method and used to be the only HM method. Flexibility is its main advantage. However, it has many disadvantages such as dependence on individual judgment, higher cost, geological features loss, time consuming, unreliable on long term (over years of history) particularly with many reservoir model variables and observations, especially if they are continuously measured. Due to these drawbacks, AHM was introduced.

1.2.2 Parametrization and Automatic History matching Methods.

Parametrization methods are used to reduce number of variables to efficiently represent reservoir model. They usually have smaller number of variables (basis vectors) than the actual reservoir. In some parametrization methods, the retained basis vectors (model variables) may change during history matching process. Many parametrizations methods were implemented to represent reservoir models (geological features) such as zonation, pilot point, spline bases, wavelet bases, principle component analysis (or Karhunen-Loeve expansion), discrete component analysis, sensitivity matrix decomposition (e.g. by subspace iteration or Lanczos method), Gramian decomposition, multiscale estimation, truncated plurigaussian and level set methods. (Oliver & Chen, 2010)

Algorithms that have been used as AHM methods can be classified to gradient and non-gradient. Examples of gradient methods are Quasi-Newton methods (e.g. BFGS & LBFGS), conjugate gradient, Levenberg-Marquardt, Gauss-Newton, and steepest descent. Gradient methods require calculating the gradients of reservoir simulator (fluid flow) equations and usually converge faster. However, they are difficult to be adjusted to different reservoir model variables and simulators because of gradient calculations, and may not be suitable for problems with discontinuous and/or discrete variables. Examples of non-gradient methods are the ensemble Kalman filter (EnKF), evolutionary algorithms (e.g. evolutionary strategies & genetic algorithms), gradual deformation, neighborhood algorithm, and simulated annealing. Non-gradient methods are usually easier to implement, adaptable to different reservoir model variables and simulators, but they require more forward (reservoir simulator) runs, thus usually more computationally expensive, and

slower to converge. (Oliver & Chen, 2010) & (Rwechungura, Dadashpour, & Kleppe, 2011)

1.2.3 Difficulties

HM process requires running the forward model (reservoir simulator) several times, thus it is computationally expensive, especially for a reservoir that has enormous number of model variables and observation data to be matched. HM problem has many solutions (ill-posed problem) because number of reservoir model inputs are extremely more than the number of wells. Additionally, to achieve geologically reasonable solutions, constraints are usually applied on HM such as known geostatistical models (prior information (regularization)), and multiple HM models (solutions) are needed for uncertainty quantification since HM problem does not have a unique solution. Additional challenges are that the types of data (collected from each well over time) are quite small, and the general relation between reservoir model inputs and model outputs is non-linear and non-local. Complications are also in the decision on the number and type of inputs to be retained and modified in HM process. This is because the sensitivity of model outputs (at observation locations) to the model inputs is complicated, changes with time (with different flow regimes and paths) and depends on operating conditions such as well constant rate, well constant bottom-hole pressure, well shut-in and new wells. Furthermore, Due to huge amount of reservoir model variables and data, efficient mathematical representation (e.g. Parametrization or dimension reduction) is essential (Oliver & Chen, 2010) & (Rwechungura, Dadashpour, & Kleppe, 2011). Finally, HM solutions that have the best matching of observation (the smallest error) are not guaranteed to represent the actual

reservoir model (the true reservoir model inputs), and might have false predictions. (Tavassoli, Carter, & King, 2004)

Despite the tremendous research effort on AHM over the past decades, no agreement has been reached on the best AHM, i.e. no method is widely accepted as the best because each one has its own advantages and disadvantages. For example, the superior AHM method in solving a particular HM problem might be worse in some other HM problems. Therefore, there is no method that is considered robust for all HM applications. Consequently, the best AHM method is completely based on the HM problem properties (Oliver & Chen, 2010) & (Rwechungura, Dadashpour, & Kleppe, 2011) & (Cancelliere, Verga, & Viberti, 2011). However, the EnKF can be considered as one of the best AHM method in some HM applications (Oliver & Chen, 2010). For instance, the EnKF was able to solve a real HM problem (real North Sea field), and had better results than manual HM (Haugen, et al., 2008).

Many optimization algorithms are available in the literature that can be used to minimize the objective functions (mismatch between simulator results and observations), but introducing one of them as new AHM method is not trivial because none of them is guaranteed to be adequate as an AHM method due to the complexity of reservoir model variables, simulators and observations. Additionally, in case one of them exhibited acceptable HM results, its algorithm and parameters usually need to be optimally

configured for each HM problem, which requires a sufficient background about this algorithm. (Cancelliere, Verga, & Viberti, 2011)

1.3 Objective

The main objective of this study is to evaluate the potential of reinforcement learning (RL) methods to be used in AHM. This is done by applying four basic RL methods (Q-learning, SARSA, Double Q-learning & expected SARSA) to solve a simple HM problem. These methods are evaluated based on their accuracy, number of simulation runs needed, computational cost, and compared to the performance of the EnKF.

CHAPTER 2

LITERATURE REVIEW

2.1 What is Reinforcement Learning?

Reinforcement learning (RL) consists of an agent that learns by interacting with its environment (by trial and error). This agent learns what to do in different situations to receive the maximum reward without being explicitly told about the correct actions (Sutton & Barto, 2018, p. 1). It can also be defined as "A RL agent acts in an environment, observing its state and receiving rewards. From its perceptual and reward information, it must determine what to do" (Poole & Mackworth, 2017).

Examples of RL are presented in the next sentences. A newborn gazelle calf struggles to stand properly on its feet, but later learns, can walk, and run. Further example is a mobile robot that collects trash from rooms and needs to decide if it should search a room for trash or return to the recharging station. The robot learns and makes this decision (search or charge battery) based on its battery charge level, its location, and how quickly it can reach the station.

2.1.1 Characteristics of Reinforcement Learning

RL problems have common Characteristics. The first Characteristic is the interaction between the agent and its environment. This interaction affects the future state of the agent, e.g. the next location of the robot. Therefore, it affects the available opportunities for the agent and the actions that can be selected later. As a consequence, the agent might plan for the future by considering the delayed and indirect consequences of its action (choice) in a particular state, to take the optimum action. Secondly, the agent must observe the status of its environment (e.g. The robot must monitor its charge level and location) regularly because the outcomes of actions are not fully known previously. Thirdly, goal and rewards are explicitly defined, so the agent can evaluate its progress toward the goal using its direct sensations and the received feedbacks, e.g. when the batteries are empty, the robot knows. Fourthly, the agent uses the experiences to improve its actions and performance over time, e.g. later the robot is able to do its tasks without running down the battery. (Sutton & Barto, 2018, p. 5)

2.1.2 Elements of Reinforcement Learning

Elements of RL are an agent, an environment, a policy, a reward signal, a value function, and a model of environment, which is only used in some methods. A policy determines the learning agent behavior (at a given state and time) by mapping the environment states to actions. There are diverse types of policies, e.g. look up table (tabular methods), function, and search process. Additionally, policies may deterministic or stochastic (actions have probabilities). A reward signal is the primary and immediate evaluation -given by the

environment to define RL goal and may be stochastic function- of the learning agent behavior, actions, and states, thus it alters the agent policy, and is used to construct the value function. A value function is the long-term evaluation of actions and states, i.e. a value function estimates the total accumulations of rewards that agent expects from actions and states. It should be noted that a high value state does not mean that it has a high reward because a state that has low reward can be followed by high rewards that makes this state has high value. Therefore, the agent selects its actions based on the value function (not the reward signal) to receive the maximum rewards over time. The method of estimating the values (value function) is the most important part in any RL algorithm. At the same time, it is difficult to determine these values because they must be estimated and continuously updated from the sequences of observations and experiences that the agent faces over time. Lastly, a model of the environment (only used in some methods) is the inference of how the environment will behave, that is the inference of the next state and reward of the environment giving an action and a state. The advantage of using a model is the ability to plan by considering future states and actions before facing them. (Sutton & Barto, 2018, pp. 6-7)

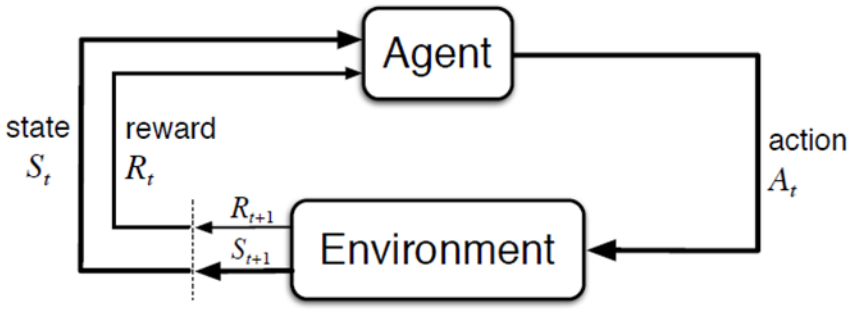

Figure 1 The interaction of agent and environment (Sutton & Barto, 2018, p. 48)

An agent is a learner that makes decision, takes actions, interacts, and affects its outside world (its environment). An environment is anything outside the agent boundary (not included in the agent's actions or unavailable choice for the agent), that agent can affect but cannot directly change or control, e.g. the agent can go right, left, up, down but cannot directly control if it will crash into something or not, but its choice of a direction will affect the outcome. The boundary between the agent and the environment represents only the limitations of the agent absolute control and not the knowledge of the agent about the environment, i.e. the agent usually knows at least something about the environment and in some cases everything about the environment, but it does not have complete control over the environment. In summary, anything that that agent cannot change arbitrarily is considered part of the environment. (Sutton & Barto, 2018, pp. 47-50)

Figure 1 represents a diagram of RL framework, which includes an environment that presents a new state and a reward as a response to an agent's action. Actions are the decisions that the agent learns how to make, and states are the useful information that helps the agent in making these decisions (actions). Representation of actions and states greatly affects the performance of any reinforcement learning method, and it might be difficult to choose an optimum representation because currently it is more art than science. (Sutton & Barto, 2018, pp. 47-50)

2.1.3 Examples of Reinforcement learning problems

In this section, three examples are given to provide an idea about designing actions, states and rewards. The first example is about applying RL to optimize a bioreactor. The actions might be temperature and stirring rate. The states might be ingredients in the container, and target chemical to be produced by the bioreactor. The rewards might be the production rate of the useful (target) chemical. In this example, action is a vector that has temperature and stirring rate and state is a list or a vector that has symbolic and sensor information. These representations of actions and states are common in RL. However, rewards are usually single numbers.

The second example is about controlling a robot motion in a pick and place task, in which the objective is to learn fast and smooth movements. The actions could be the applied voltage to each motor in each joint of the robot. The states could be the latest information about each joint angle and velocity. The reward is given as +1 for each successful completed job of picking and placing an object. Small negative rewards that are functions of movements jerkiness might be given over time steps to encourage smooth movements.

The third example is about how a robot searches for empty cans in an office by making decisions based on its battery charge level. The states could be simplified to only high and low charge levels, e.g. S= [high, low]. In reinforcement learning, each state might have different sets of actions as explained in the next sentences. The set of actions for high level charge might be search for a can or wait for someone to bring a can to preserve battery,

e.g. A(high) = [search, wait]. On the other hand, the set of actions for low level charge might be search or wait or go to its recharging station, e.g. A(low) = [search, wait, recharge]. The rewards are: large negative number in case battery died, positive if it collected a can, and zero most of the time. (Sutton & Barto, 2018, pp. 51-52)

2.2 How is Reinforcement learning different?

RL has similarities with many methods and can be integrated or work with some of them. For instance, RL can be confused with other machine learning types, evolutionary methods, and decision-making algorithms. Therefore, the differences between RL and other methods need to be discussed.

2.2.1 Differences between Reinforcement Learning and other types of Machine Learning

Supervised learning uses a training set of data that is labeled by an external supervisor knowledge, such that each example has properties of a situation that corresponds (linked) to a correct action, e.g. identifying situation category (classification problem). The goal of supervised learning is to develop (train) generalized and extrapolated correct responses to data that were not in the training set. However, learning from interactions cannot be adequately achieved relying only on supervised learning because examples of correct actions (correct responses of supervised learning model) for all situations encountered by the agent cannot be practically obtained in many problems. Moreover, In RL the most valuable learning occurs in the unknown situations, i.e. uncharted territory.

Unsupervised learning is finding the hidden structure of unlabeled data. Finding hidden structure of an agent's experience can be useful in RL, though unsupervised learning alone cannot solve RL problems. RL might be confused as a type of unsupervised learning because RL does not depend on examples of correct actions (external knowledge), however RL is about maximizing rewards and does not try to find the hidden structure of data. (Sutton & Barto, 2018, p. 2)

RL can be regarded as the third paradigm of machine learning that is different from supervised and unsupervised learning for the following reasons. RL is related to artificial intelligence that is based on fewer and general principles rather than by the immense amount of facts and procedures (Sutton & Barto, 2018, p. 4). RL has challenge of the trade-off between exploitation and exploration, that issue is not available in supervised and unsupervised learning (Sutton & Barto, 2018, p. 3). Exploit-explore dilemma is when the agent must decide whether to follow its previously known good actions or explore other actions. Further challenge, which is only in RL, is the blame attribution problem or credit assignment problem where the action responsible for punishment or reward is not obvious because the action happened long time before. For example, an action might be initially seen as suboptimal, but later may lead to the highest reward in the future (Poole & Mackworth, 2017).

The differences between RL and other machine learning paradigm are outlined in the next four points. RL has only reward signal with no supervisor to tell directly if an action is good or not. Secondly, the feedback or the outcome of doing an action is delayed, i.e. it is not instantaneously available that in some cases reward signal might be positive for an action, but later perceived as the worst action due to its final catastrophic result. Thirdly, RL works in dynamic environment where time is crucial factor. Therefore, the data (interactions between agent and environment) at a time step is correlated to the next time step (sequential data). This implies that assuming independent and identically distributed random variable (IID) is not valid in RL. Fourthly, agent's actions influence the next data it receives, i.e. two agents in exactly the same environment can receive completely different data (e.g. states and rewards), hence learn different things. (Silver D., 2015)

2.2.2 Differences between Reinforcement Learning and Evolutionary Methods

Evolutionary methods usually start by generating random population (individuals) or static (fixed) policies where each one interacts over long time with separate realization of the environment. Then, individuals (policies) with highest fitness or most rewards are selected with random combination and modification, which forms the next generation of population (policies), and the process repeats. One advantage of evolutionary methods is in solving problems, where the agent cannot sense the complete state of the environment (Sutton & Barto, 2018, pp. 7-8). Additionally, evolutionary methods can work effectively with small space policies, or by generating policies with efficient structure where good policies are easy or common to obtain. Otherwise, it might be too slow and inefficient for practical

applications especially with large space policies because for m actions and n states, there are m^n policies. Small games can have about 10^{150} policies, which are more than the particles in the universe (Poole & Mackworth, 2017).

RL might be confused with evolutionary methods because both can solve same problems, have common features, and can work together. However, they differ in how they work to find a solution as explained in the subsequent lines. The first difference is that most of RL methods estimates value functions, while evolutionary methods do not use value functions. Secondly, evolutionary methods do not learn while interacting because they ignore useful details of each interaction with the environment and overlook RL problem structure. For instance, they neglect that the generated policies are functions from states to actions, and they do not recognize the states that an individual experienced during its lifetime or which actions were taken. In other words, Evolutionary algorithms waste the information from experiences because they judge the policy as a whole by waiting until the end when the agent has finished. On the other hand, RL methods learn the components of the policy, thus they can distinguish between good and bad actions in each state separately, and do not wait until the end of the task. Therefore, RL methods can be immensely more efficient by exploiting the detailed information of each interaction. These details might lead to worse performance in case the states are misperceived, though in many cases they increase the efficiency. (Sutton & Barto, 2018, pp. 7-8) & (Poole & Mackworth, 2017)

2.2.3 Differences between Reinforcement Learning and Decision-Making Methods

Many RL methods are different from some other decision-making methods (e.g. dynamic programming) in the following aspects. RL can solve problems where the environment dynamics (expected transitions) are not available because it can learn from transitions of actual experiences (Sutton & Barto, 2018, p. 66). This is the case for many complex problems where we do not know the expected outcome of an action unless we tried it in the environment. For example, in a bioreactor the effect of temperature on the rate of production is unknown without running the experiment, and in history matching the effect of increasing permeability values to get a better match is unknown without running the reservoir simulator. Furthermore, RL does not solve the system of equations that is based on exhaustively searching and looking at all possibilities. However, it finds approximate solutions by utilizing the transitions experience (samples). These Approximate solutions are extremely important to solve problems with enormous number of states, where the optimal policy (exact or best solution) is computationally expensive. For example, backgammon game has about 10^{20} states, which would take thousands of years by the fastest computers to find the optimal policy. Additional difference is that RL approximates solutions in distinctive way by focusing more on states that are encountered frequently, while compromising other states with low probability of occurring, which have minor effect on the total rewards received by the agent. (Sutton & Barto, 2018, pp. 66-67)

RL transforms dynamic programming methods into practical algorithms suitable for huge problems. This is done by using samples to efficiently represent the environment dynamics and using function approximation methods to efficiently represent value functions. Moreover, they are compatible with each other in a way that function approximation methods utilize samples that are concentrated on a subset of state and action spaces. The design, analysis, and application of any RL method relies on recognizing the connection between dynamic programming, sampling, and function approximation. (Szepesvári, 2010, p. Preface)

2.3 Why Reinforcement learning?

RL witnessed several accomplishments in different applications and exceeded the expectations in some of them. Szepesvári (2010, p. 63) listed several publications that discusses the successful applications of RL in many fields such as learning in games, networking, operation research, maintenance problems, pricing, finance, scheduling, fleet management, inventory control and robotics. Moreover, RL sits in the intersection of many different sciences (Silver D. , 2015), see Figure 2.

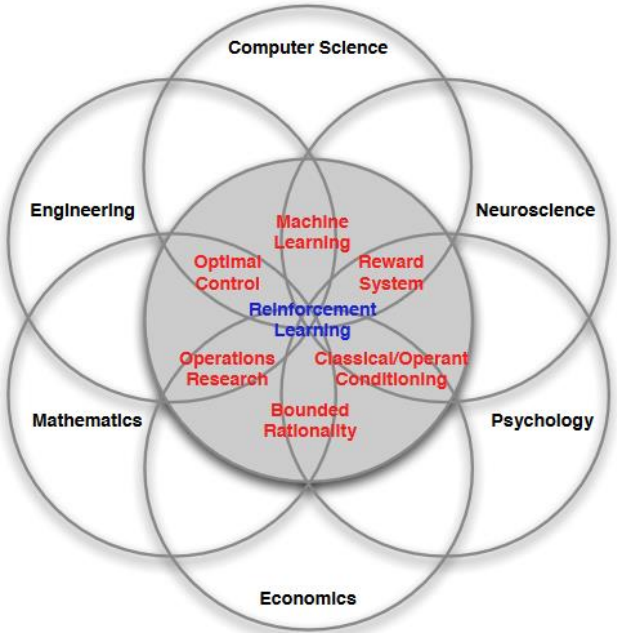

Figure 2 Reinforcement learning relation to different sciences (Silver D. , 2015)

2.3.1 Reinforcement Learning in Games

The following two examples show the breakthroughs of RL in learning in games. The Deep Q-learning outperformed previous algorithms in Atari games, and reached at least 75% or more of human professional player level in 29 games using the same program (same algorithm & network architecture & hyper parameters) across all games, which used only game score and display pixels without incorporating any prior knowledge about each game (Mnih, et al., 2015). Moreover, AlphaGo program outperformed all previous Go programs with a winning rate of 99.8% and defeated the champion of Go game for the first time in history. Go game has huge search space; It has approximately 250^{150} possible sequences of moves, that is why it was thought that it would take at least a decade for a computer program to defeat a professional human player (Silver, et al., 2016).

2.3.2 Reinforcement Learning in Petroleum Engineering

The applications of RL methods in petroleum engineering were focused on reservoir management and optimization. The first introduction of RL in petroleum engineering was in 2018, where it was applied to optimize steam injection.

A RL method (SARSA) was implemented to optimize steam injection in heavy oil simulation model based on a real reservoir located in northern Alberta. The design of actions, states and reward were as the following. Action were to increase, decrease or not change injection rate, such that in case 1 the rate of change was 10 m^3/d, and in case 2 was 5 m^3/d. Reward was the net present value (NPV), whereas the environment states were cumulative oil production, cumulative water production, and cumulative steam injection, and the function was approximated by six radial basis kernels. The RL method was computationally more efficient than differential evolution; it required a smaller number of simulations to converge. Additionally, it achieved higher NPV and lowered cumulative steam-oil ratio compared to differential evolution optimization algorithm and typical field injection strategies. Table 1 summarizes the performance of the two RL cases, and differential evolution by CMG CMOST at the end of 250 days of production. Figure 3 presents the net present value of each method over the production life of 250 days. (Guevara, Patel, & Trivedi, 2018)

Table 1 Performance comparison between two cases of reinforcement learning (RL) and Differential Evolution at the end of 250 days of production (Guevara, Patel, & Trivedi, 2018)

Case	No. of simulations required to achieve convergence	Simulation time[1] [hours]	Net present value [$]
RL (Case 1)	40	12.17	2.5e+06
RL (Case 2)	140	66.94	3.4e+06
Differential Evolution (CMOST)	350	192.75	1.1e+06

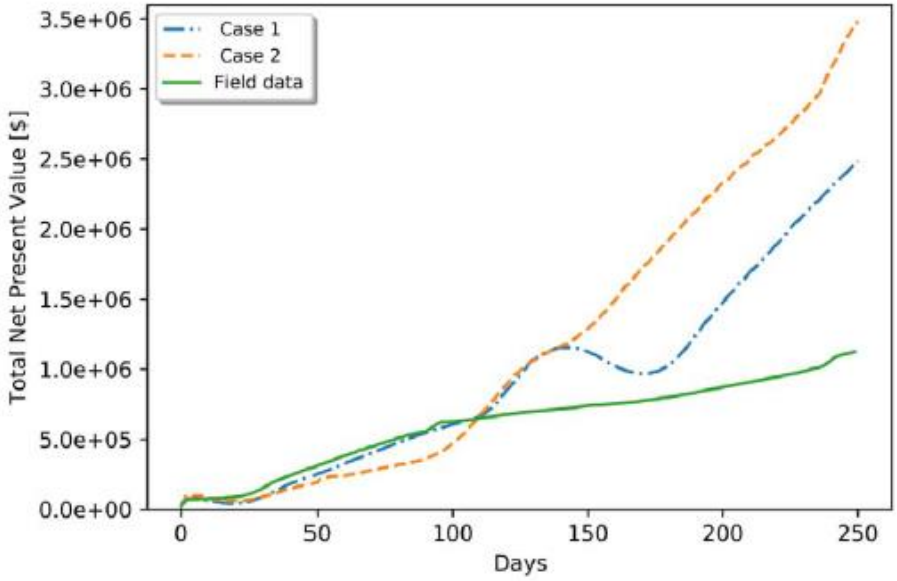

Figure 3 NPV comparison between RL (case 1 &2) and field operation over production life of 250 days (Guevara, Patel, & Trivedi, 2018)

In other research paper, a RL based algorithm achieved the objectives and operational targets in optimizing water flooding in a synthetic oil reservoir model (Egg model). This RL method resulted in the maximum cumulative net present value (NPV) over 14 years compared to reactive control and gradient based control, see Figure 4. The design of this RL method included two rewards function: instantaneous and delayed reward function for learning different production policies and objectives. Actions were injection rates of each

injection well, and states were percentage of the current remaining oil in the reservoir, and water cuts for each well, such that the function was approximated by fuzzy inference system. Since the total production and injection rates were seen as redundant data, they were not included in the set of states. (Hourfar, Bidgoly, Moshiri, Salahshoor, & Elkamel, 2019)

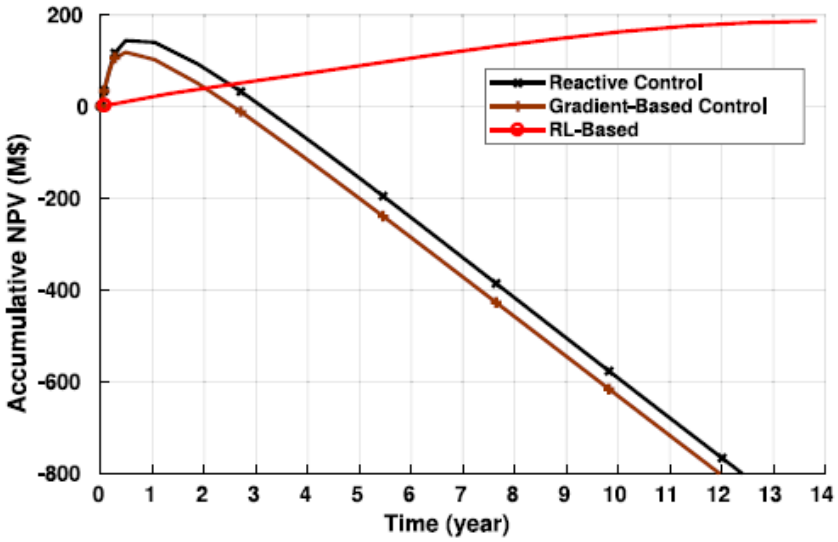

Figure 4 NPV over 14 years of production of reactive control, gradient-based control and RL method (Hourfar, Bidgoly, Moshiri, Salahshoor, & Elkamel, 2019)

Another study applied four deep RL algorithms (DQN, DDQN, Dueling DDQN & DDPG) to optimize water flooding in a 3-phase reservoir. Only DQN reached the highest NPV (7.31 MM), which was found by particle swarm optimization (PSO), however DQN needed three times the simulation runs needed by PSO to converge to this highest NPV. On the other hand, DDPG converged to local optimum NPV (7.19 MM) with only third of PSO simulation runs. Table 2 summarizes the results of this work, and Figure 5 compares the

methods based on the attained NPV over the reservoir simulation runs. In these RL methods, reward was the profit, action was the injection rate, states were oil, water, and gas production rates in addition to the reservoir pressures, such that the function was approximated by deep neural networks. (Ma, Yu, She, & Gu, 2019)

Table 2 Convergence and NPV of DQN, DDQN, dueling DDQN, DDPG and PSO. (*) did not converge (Ma, Yu, She, & Gu, 2019)

Algorithms	Number of numerical simulation runs needed to converge (10^6)	Final expected NPV (10^6 US$)
DQN	0.18	7.31
DDQN	*	7.07
dueling DDQN	0.06	7.19
DDPG	0.02	7.19
PSO	0.06	7.31

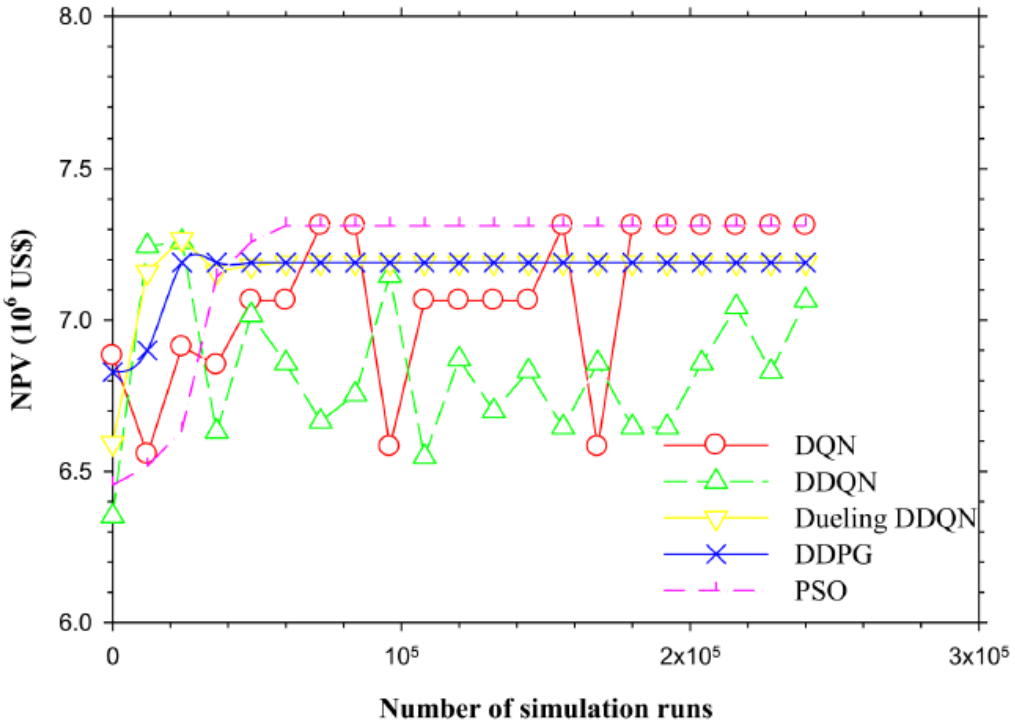

Figure 5 NPV versus simulation runs (Ma, Yu, She, & Gu, 2019)

In a study, a 2D simple reservoir model, which has one production well, was optimized by RL method called proximal policy optimization (PPO), where the function was approximated by a neural network. PPO method had the recovery factor as the reward, changing well bottom hole pressure (BHP) as the action, pressure and water saturation as the states. (Miftakhov, 2019)

2.4 Classifications of Reinforcement Learning Methods

Temporal difference (TD) learning is the central part of RL. TD can be combined with other methods: with planning by learning the dynamic model of the environment from the generated samples (agent-environment interaction); with Monte Carlo by updating value functions after more than one step and before the end, e.g. after every five steps; and with both (planning & Monte Carlo).

Examples of TD control methods are Q-learning, SARSA, expected SARSA and double Q-learning; examples of TD with planning are dyna-Q, dyna-Q+ and Prioritized sweeping; and examples of TD with Monte Carlo are n-steps and eligibility traces.

RL methods can be classified based on the type of the function approximation used: tabular, linear, and non-linear (neural network). Moreover, they can be divided based on the type of value estimated: Q (S, A) action-state value (control problem) and V(S) state value (prediction problem). There is also a distinction between off policy (e.g. Q-Learning) and on-policy (e.g. SARSA). Off policy does not update its value function with the current policy (the performed action), but with the best action (max) based on the current value function. In contrast, on policy updates the value function with the value of actions that are being performed. Another classification of RL methods can be based on the type of the method used to select actions, i.e. the tradeoff between exploration and exploitation, e.g. greedy and ε-greedy.

2.5 What is the Recommended Reinforcement Learning Method?

There are many RL methods available in the literature, that might make anyone interested in applying RL ask about the recommended RL method that achieves the best results. This section briefly discusses the results and performance of various RL methods. Moreover, it stresses that in general there is no RL method that is better than another in all applications, but one that is better in a specific application or has an overall better performance in most of applications. Furthermore, four publications are cited to support the previous sentences, and to show that more advanced methods are not necessarily better than fundamental ones.

A study compared tabular and linear function approximation in a grid world game (predator-prey). Surprisingly, tabular RL methods outperformed their variants that are based on linear function approximation in most cases. Even though tabular ones had partial state representation, and linear function approximation used full state representation besides its ability to generalize. The study elucidated that accurate representation of partial state is better than inaccurate representation of full state. Additionally, it was concluded that it might be better to try solving complex problems with a basic method before attempting an advanced method. (Schrum, 2008)

Geramifard, et al. (2013) compared the performance of four different function approximation methods (representations) for three RL methods applied on four benchmark domains. The compared representations were tabular, fixed sparse (linear), radial basis

functions and incremental feature dependency discovery. RL methods based on Tabular representation had the best performance in two domains: grid world and inverted pendulum (continuous state). It was observed that tabular representation resulted in the best performance for smaller domains, and that the selection and design of representation can have more significant impact on performance than the choice of the RL method itself.

Various impressive achievements in RL field were caused by implementing neural network as a function approximation, however they (non-linear function approximation) are exposed to divergence. On the other hand, linear function approximation methods are guaranteed to converge for some RL methods, easier to implement, and faster to train because of having less parameters. In a research paper, simple policy approximations, such as linear and radial basis function (RBF), were compared to neural network, where they were used to represent policies of a RL method named trust region policy optimization (TRPO), that was applied in six continuous control tasks. Linear and RBF policies were comparable to the top results previously achieved in these tasks, and more efficient in learning in four of these tasks. Furthermore, RBF policy had the best results in five tasks. (Rajeswaran, Lowrey, Todorov, & Kakade, 2017)

Hessel, et al. (2017) Integrated the previous independent improvements of DQN methods into one algorithm that demonstrated superior performance and data efficiency in 57 games (Atari 2600) compared to other RL methods such as DQN, DDQN, prioritized DDQN, Dueling DDQN, Distributional DQN, noisy DQN and A3C. Despite that, this state of art

algorithm had worse scores than other RL methods in some of these games. Furthermore, DQN – that has the worst overall performance- outperformed this best algorithm in some games such as bowling and venture.

2.6 Applied Methods

This section introduces the methods that are applied in the next chapter (chapter 3: results and conclusion).

2.6.1 Principle Component Analysis (PCA)

PCA (or Karhunen-Loeve expansion) is a global parametrization method that represents any Gaussian random field to number of independent random variables, and creates many realizations that honor and preserve the geostatistical correlation. PCA is used as a dimension reduction method because it orders the eigenvectors (f_n) of the geostatistical covariance matrix by their eigenvalues (λ_n), such that in decomposition (truncation) process only the top eigenvectors are retained, which are able to approximate the original field (Chang & Zhang, 2014). In PCA equation 2.1, $Y(\boldsymbol{x}, \omega)$ is the representation of the random variable field, which is a function of location *x* and the probability space $\xi_n(\omega)$, whose values are independently sampled from standard Gaussian distribution *N*(0, 1) to form a realization. Each set of ξ_n values (eigenvector parameters) forms a unique realization that has the same field covariance. *N* is number of retained eigenvectors, which is smaller than number of the original field eigenvectors in case of truncation.

$$Y(x, \omega) = \sum_{n=1}^{N} \sqrt{\lambda_n}\, f_n(x) \xi_n(\omega) \qquad (2.1)$$

2.6.2 The Ensemble Kalman Filter (EnKF)

The EnKF is a data assimilation method that uses ensembles (many samples of inputs, e.g. permeability field realizations). These input model ensembles are forecasted (forward simulation) to get the output ensembles (e.g. pressures or rates, each output ensemble corresponds to input ensemble).Then, Kalman gain translates the difference between output ensembles and observations into values that modify (update) all ensembles (inputs and outputs) to match the observations (Gu & Oliver, 2005).

The EnKF consists of the following steps. Firstly, reservoir simulator uses the initial or the updated input model ensembles *m* (e.g. sets of ξ_n values from PCA) to calculate output ensembles (*f*(*m*)), as in equation 2.2, where *f* is the reservoir simulator and superscript *f* refers to forecast. Secondly, covariance matrix and Kalman gain are calculated by equations 2.3 & 2.4, where P is the ensembles covariance matrix, *Ne* is the number of ensembles, subscripts *i* and *j* refer to ensemble index, *T* is the matrix transpose and (y-bar) indicates the mean, *K* is Kalman gain, C_D is covariance of observation error, ($P\ H^T$) is the input-output and output-output covariance, and ($H\ P\ H^T$) is the ouput-ouput covariance (Oliver & Chen, 2010). Finally, the model inputs and outputs are updated (assimilated) by equation 2.5, where (H y^f) contains only the output ensembles(*f*(*m*) or simulator results), d is the observation, and superscript *a* refers to assimilation.

$$y^f = [(m), f(m)] \qquad (2.2)$$

$$P = \frac{1}{N_e - 1} \sum_{i,j=1}^{N_e} (y_i^f - \overline{y})(y_j^f - \overline{y})^T \qquad (2.3)$$

$$K = P\,H^T(H\;\;P\;\;H^T + C_D)^{-1} \qquad (2.4)$$

$$y^a\; = y^f\; + K * (d - H\;\;y^f\;) \qquad (2.5)$$

2.6.3 Tabular Temporal Difference Methods

Q-learning (TD control algorithm) is regarded as one of the early achievements of RL (Watkins, 1989). The procedures of Q-learning algorithm are shown in Figure 6.

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

Figure 6 Q-learning algorithm from (Sutton & Barto, 2018)

All tabular TD control methods have Q table (state-action value) in common, which they utilize to select the optimum actions for the states. The main difference between them is how they calculate and update the Q values. Moreover, they have some common parameters such as step-size (α), discount rate (γ) and epsilon (ε). Step-size (α) affects the learning rate. Discount rate (γ) affects the weight (present value) of future rewards, as γ gets closer to zero future rewards are discounted, and the agent becomes more shortsighted. Epsilon (ε) represents the exploration probability, i.e. the probability of selecting random action without exploiting what was learnt as the best action. These parameters should be

optimally selected for each environment (application) and for each algorithm independently. For instance, Figure 7 compares the performance of tabular TD method in cliff-walking environment with different step-size (α) values, and all of them have the same ε-greedy policy (ε=0.1). Cliff-walking environment **(**Figure 8**)** consists of an agent that have to reach to a goal without going into the cliff region. This figure also highlights the difference between on-policy (SARSA) that learnt the safer but longer path, and off-policy (Q-learning) that learnt the optimal (shortest) path.

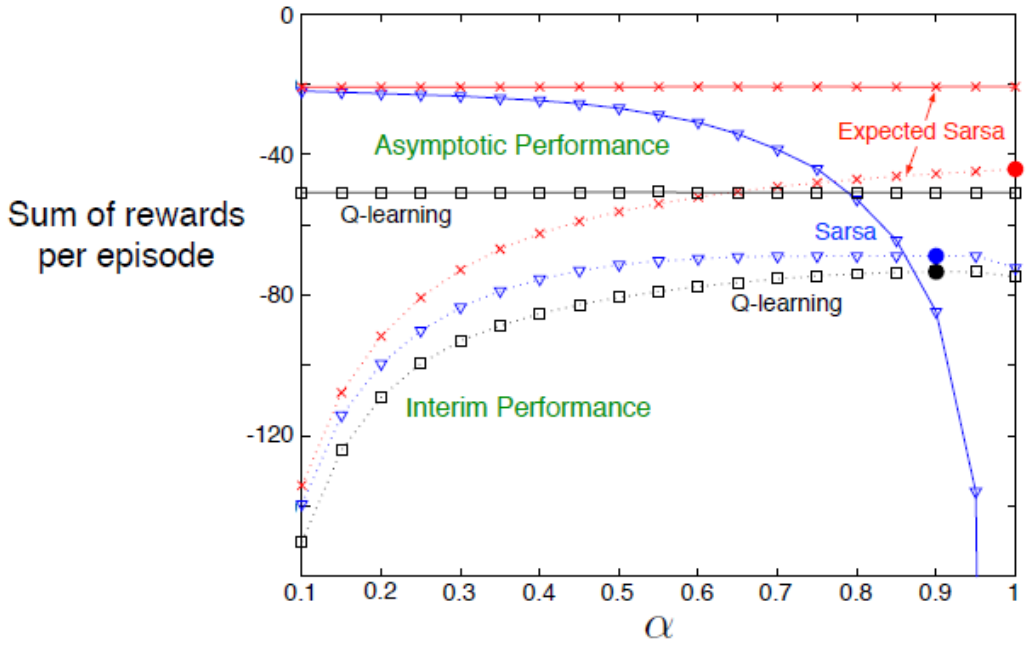

Figure 7 Rewards of TD control methods in cliff-walking environment for interim performance (average of first 100 episodes) and asymptotic performance (100000 episodes), circles represent the best α value for each method (Seijen, Hasselt, Whiteson, & Wiering, 2009)

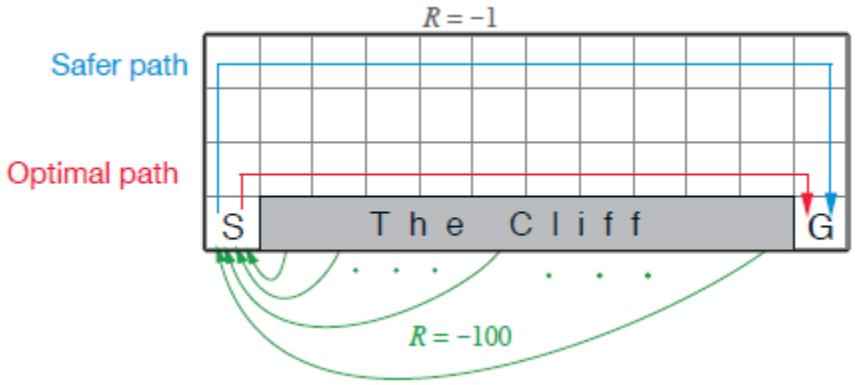

Figure 8 Cliff walking environment, S (location of the agent), and G (location of the goal) (Sutton & Barto, 2018)

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

Figure 9 SARSA algorithm

(Rummery & Niranjan, 1994) introduced SARSA algorithm that is shown in Figure 9. (John, 1994) introduced Expected SARSA, in which its update of Q (S, A) is the expected value, which considers the probability of each action under the current policy. Although it is computationally more complex, it generally has slightly better performance than SARSA. The calculation of Q values in Expected SARSA is done by equation 2.2.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \mathrm{E}_\pi[Q(S_{t+1}, A_{t+1}) \mid S_{t+1}] - Q(S_t, A_t) \quad \text{Eq (2.1)}$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t)] \quad \text{Eq (2.2)}$$

The concept of double learning was introduced by (Hasselt, 2010), and Figure 10 presents the double Q-learning algorithm. This algorithm eliminates the maximization bias that is in the previous methods, which occurs when the true values of actions are all zeros, but they have distribution or uncertainty that some of them have positive and negative values, which cause the maximum of their estimate to be positive instead of the maximum true value that is zero. To resolve this issue, double learning uses two estimates (two Q-tables), such that only one of them is updated per time step. Thus, it requires twice the memory, but there is no increase in the computation per time step.

Double Q-learning, for estimating $Q_1 \approx Q_2 \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q_1(s, a)$ and $Q_2(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, such that $Q(terminal, \cdot) = 0$
Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using the policy ε-greedy in $Q_1 + Q_2$
    Take action A, observe R, S'
    With 0.5 probabilility:
      $Q_1(S, A) \leftarrow Q_1(S, A) + \alpha\Big(R + \gamma Q_2\big(S', \arg\max_a Q_1(S', a)\big) - Q_1(S, A)\Big)$
    else:
      $Q_2(S, A) \leftarrow Q_2(S, A) + \alpha\Big(R + \gamma Q_1\big(S', \arg\max_a Q_2(S', a)\big) - Q_2(S, A)\Big)$
    $S \leftarrow S'$
  until S is terminal

Figure 10 Double Q-learning algorithm

CHAPTER 3

RESULTS AND CONCLUSION

3.1 Methodology

A simple HM problem is used to compare the applied methods, which contains the following. A single phase 1D reservoir model has length of 50, and two fixed pressure boundary conditions at both ends. This HM problem considers one type of observation (pressure) to be matched, and one type of reservoir model variable (permeability) to be changed to match the observation. The number of observations is 50, and the number of model inputs is 10. These model inputs are changed to modify the permeability values of the reservoir model to match the 50 pressure observations. The number of observations is large and equal to the size of the reservoir model to ensure that any method has enough information to exactly match the true observations. The number of model inputs used is only 10 and not 50 because principle component analysis (PCA) is used as a parameterization method to decompose the permeability field from 50 eigenvectors to 10 eigenvectors. Since the correlation length used in PCA is 5, 10 eigenvectors are sufficient to retain most of the permeability field (about 80%). All methods have to history match 100 randomly generated true realizations to evaluate their average performance. These 100 true realizations are generated by all permeability field eigenvectors (50), while the initial reservoir model, that is used to match the true realizations, is generated by only the first ten eigenvectors of permeability field, and the parameters of these ten eigenvectors are

modified by each method to match each true realization. It is crucial for all methods to have the same initial model inputs values and the same 100 realizations for HM to achieve a fair comparison between them. Thus, all methods initially have the same 10 parameter values of the used 10 eigenvectors, and they are restored to their initial values after the end of each true realization HM. For instance, the EnKF model input ensembles are restored to their initial values at the beginning of each realization HM process. Additionally, the 100 true realizations that are history matched are the same for all methods. As a result, at the beginning all methods have the same average pressure and permeability RMSE (Root-mean-square of error) of the 100 true realizations. Furthermore, the parameters of all methods are initialized after the HM end of each realization, e.g. the Q-table values are changed to zeros after matching one realization. In other words, RL methods are not allowed to learn from the previous realization HM. The appendix contains field permeability and pressure figures for one true realization and the reservoir model that is modified to match the 100 true realizations.

This HM problem is adequate to filter out the potential RL methods for HM. The simplicity of the reservoir model makes it extremely fast for simulation runs to be computed, easier to debug and integrate with the EnKF and RL algorithms compared to other reservoir models. The EnKF is applied to solve this HM problem so that the applied RL methods, which are the tabular variants of Q-learning, SARSA, double Q-learning, and expected SARSA, can be compared to one of the used AHM methods in the literature. Thus, the thesis objective can be attained, which is focused on evaluating some basic RL methods for HM.

For comparison, two plots are presented for each method: average RMSE of permeability and pressure versus the number of time steps (iterations). The first data point (at first time step) in the RMSE figures of all methods does not represent the first iteration or first time step of the method, but the initial average RMSE of the 100 realizations before history matching process, hence the first data point in pressure and permeability RMSE figures are the same in all methods. Additionally, CPU time taken by each method per one-time step (iteration) is stated.

equation 3.1 is the RMSE, then equation 3.2 calculates the average RMSE of the 100 realizations, which are the values (black dots) in the average pressure and permeability RMSE figures. While, the blue error bars in permeability and pressure RMSE plots are the standard errors, which are calculated by equation 3.3.

$$RMSE\ of\ each\ Realization = \sqrt{\frac{\sum_{i=1}^{n}(Predicted_i - Observation\ or\ true\ value_i)^2}{number f samples\ (n{=}50)}} \qquad \text{Eq (3.1)}$$

$$Average\ RMSE \text{ of all realizations} = \frac{\sum_{i=1}^{n} RMSE_i}{number of realizations (n = 100)} \qquad \text{Eq (3.2)}$$

$$Standard\ Error = \frac{\text{Standard Deviation (RMSE of all realizations)}}{\sqrt{number\ of\ realizations\ (n = 100)}} \qquad \text{Eq (3.3)}$$

3.2 The Ensemble Kalman Filter

In the study, the EnKF method is presented as a benchmark for RL methods. The results of the EnKF are based on 15 ensembles and 0.1 observation error (C_D). The EnKF method generates many solutions (15 ensembles), but the results in this section are the average solution of these ensembles. At the beginning, observation error was 0.0001, although EnKF matched the pressure observations of the 100 realizations, it had extreme error values in matching of true permeability fields, hence observation error was changed to 0.1. Since the EnKF method contains 15 ensembles, each time step of the EnKF involves 15 reservoir simulation runs, which are computed in parallel.

The average pressure and permeability RMSE at the first-time step are the values before the EnKF HM. Before HM process, the average RMSE of 100 pressure realizations was 0.123, and the average RMSE of 100 permeability realizations was 0.179.

In Figure 11, the EnKF decreased pressure RMSE to less than 0.06 after one time step (15 simulation runs). The minimum achieved average pressure RMSE by the EnKF is 0.013 at the end.

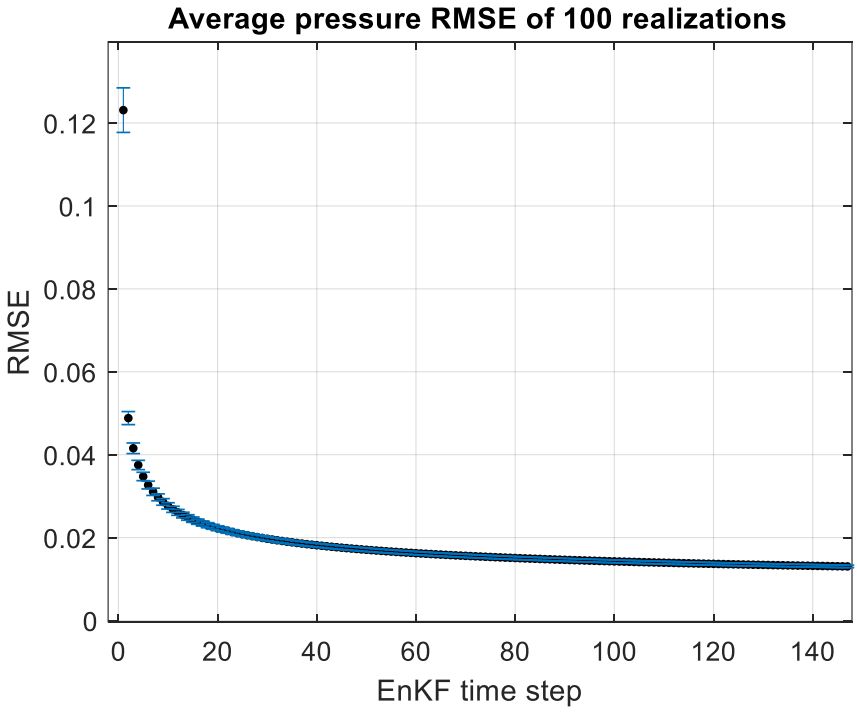

Figure 11 The EnKF average RMSE of 100 realizations of observations (pressure values)

In Figure 12, the EnKF decreased permeability RMSE to less than 1.7 after one time step (15 simulation runs). The minimum achieved average permeability RMSE by the EnKF is about 1.5 at the end.

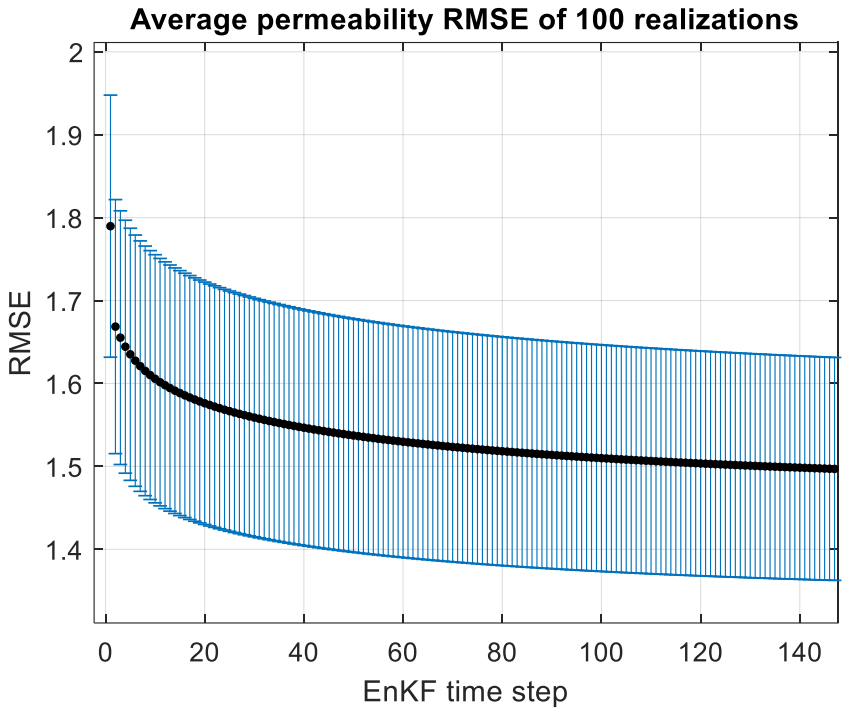

Figure 12 The EnKF average RMSE of 100 realizations of true models (permeability values)

3.3 History Matching in Reinforcement Learning Framework

In this section, HM is defined from RL perspective, and suggestions on how to design states, actions and reward are presented. In RL, an agent is in the place of the petroleum engineer, who tries to change the reservoir model parameters to match the production history, where the environment is the reservoir simulator or model, that the petroleum engineer modifies, or the agent interacts with. After the modification of the reservoir model inputs, plots that have both the reservoir model outputs and history data or the mismatch between them are usually analyzed to decide on the next changes to the reservoir model inputs. Thus, these plots (useful information) are like the states that agent utilizes to take an action.

Consequently, the action that the agent can take, is to change reservoir model inputs such as permeability or porosity values. The States could be the error or mismatch between the reservoir model outputs and observations, e.g. error in pressure values or in production rates. It is critical for the state to distinguish between positive error (e.g. output value is more than observation) and negative error (e.g. output value is less than observation), i.e. using absolute value error might be confusing for the agent. The reward signal might be based on the change in state or change in error. Such that, if the error in increasing, the reward will be negative, and if the error is decreasing, the reward will be positive.

3.4 Q-table Design

This section discusses the design of actions, states, reward, and Q-table. In the studied HM problem, the action is to change permeability values by modifying the parameter of each eigenvector (of 10 eigenvectors), and the state is the error in each pressure value of the 50 observations. Therefore, both actions and states are continuous. Because Tabular TD methods rely on look up table (Q-table) for their decision making, they work only with discrete actions and states. Thus, the continuous actions and states need to be discretized to form the Q table, which is the main component of tabular TD algorithms. There are numerous approaches to discretize states and actions, though one has to balance between efficiency or size of Q table, and ensuring that enough information is clearly communicated to the agent to select the best actions.

Actions are increase, decrease or do not change, so each eigenvector parameter has three actions. Therefore, we have 30 (3 X 10) actions. The increase or decrease of permeability values is done by adding to or subtracting from each eigenvector parameter a small percent (e.g. 1%) of its initial value. The selected actions are applied on all 10 eigenvector parameters at each time step, which modify the permeability values of the reservoir model, which affect the model outputs (pressure values).

The states are the observations error, that are calculated as the relative absolute error percentage ((predicted pressure (model output) – observation pressure)/observation pressure X 100)), where each observation error is discretized to positive error or negative error, hence we have 100 (2 X 50) states, i.e. two states for each error of the 50 pressure observations. As a result, the size of Q table or matrix is ((100 rows) X (30 columns)) that is 3000 values, such that rows represent the states and columns represent the actions.

The reward function is a vector whose 50 elements are calculated as change of the absolute error of each observation (absolute value of (old error) – absolute value of (new error)). If the new error is larger, the reward will be negative value, otherwise the reward will be positive value.

Table 3 illustrates the designed Q-table for only two observations and two eigenvector parameters (reservoir model inputs). The size of this Q table is (4 states X 6 actions), that is 2 states for each observation (2 X 2 states), and 3 actions for each eigenvector parameter (3 X 2 actions). Assume the initial state of observation one is positive error, and observation two state is negative error, then based on that the agent decided to increase parameter 1 (of eigenvector 1) and to not change parameter 2 (of eigenvector 2). Accordingly, the reward value (change of the absolute error) that is calculated from observation 1 will be used to update elements 11 and 16, and reward value of observation 2 will be used to update elements 41 and 46.

Table 3 Sample of the designed Q table for tabular temporal difference methods

		first eigenvector parameter (parameter 1)			second eigenvector parameter (parameter 2)		
		Increase	Decrease	None	Increase	Decrease	None
Observation 1	S (+ve error)	11	12	13	14	15	16
	S (-ve error)	21	22	23	24	25	26
Observation 2	S (+ve error)	31	32	33	34	35	36
	S (-ve error)	41	42	43	44	45	46

To select actions based on maximizing Q(s,a), the sum of values of each action in the current states is calculated. Assume that we are in the same states of the previous example (S(+ve) for observation 1 and S(-ve) for observation 2), to decide which action the agent will select for parameter 1, the maximum value of these element sums: (11+41), (12+42), and (13+43) is selected. For example, if the sum of elements 11&41 (11+41) is the maximum, the agent will increase parameter 1 by a specified value (e.g. 1% of the initial parameter 1 value). In case two or more sums are equal, the agent will randomly select from them. For example, if (12+42) is equal to (13+43), the agent will randomly select to decrease or not change parameter 1, or if they all ((11+41), (12+42), (13+43)) are equal, the agent will randomly select to increase or decrease or not change parameter 1. Similarly, for parameter 2, value of element 14 is added to value of element 44, (15+45), and (16+46),

then the action that has maximum value is selected. All the tabular TD methods are based on this Q-table.

3.5 Q-Learning

The figures of the tabular Q-learning method are generated using discount factor equal to 1, step size (learning rate) equal to 0.95 and epsilon (exploration probability) equal to 0.2. Additionally, the percent of change done on each eigenvector parameter (model input) is 0.01 (1%) of its initial value. Although increasing the percent of this change (1%) leads to quicker matching of pressure observations, it might cause divergence of matching permeability fields.

Each time step of Q-learning involves only one simulation run. Before Q-learning HM process, the average RMSE of 100 pressure realizations was 0.123, and the average RMSE of 100 permeability realizations was 0.179.

In Figure 13, Q-learning decreased pressure RMSE to 0.06 after 1065 time steps (1065 simulation runs). The minimum achieved average pressure RMSE by the Q-learning is 0.0288 at the end. This means that it could not reach the minimum average pressure RMSE of the EnKF.

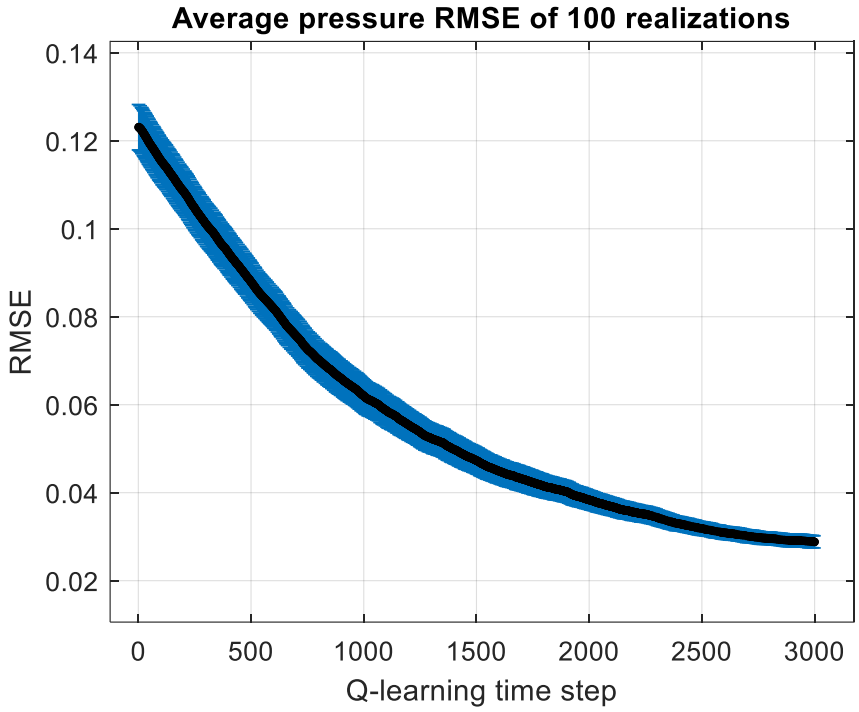

Figure 13 Q-learning average RMSE of 100 realizations of observations (pressure values)

In Figure 14, Q-learning decreased permeability RMSE to 1.7 after 704 time steps (704 simulation runs). The minimum achieved average permeability RMSE by Q-learning is 1.588 at the end, hence it could not reach the minimum RMSE achieved by the EnKF.

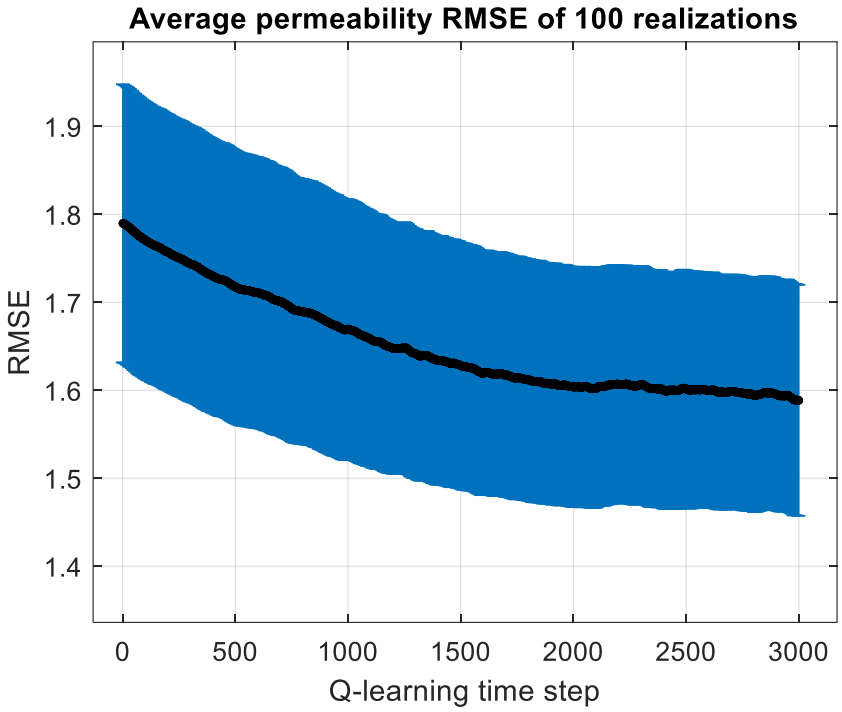

Figure 14 Q-learning average RMSE of 100 realizations of true models (permeability values)

3.6 SARSA

The figures of the tabular SARSA method are generated using the same values used for Q-learning, which are discount factor equal to 1, step size (learning rate) equal to 0.95 and epsilon (exploration probability) equal to 0.2. Additionally, percent of change done on each eigenvector parameter (reservoir model input) is 0.01 (1%) of its initial value.

Each time step of SARSA involves only one simulation run. Before SARSA HM process, the average RMSE of 100 pressure realizations was 0.123, and the average RMSE of 100 permeability realizations was 0.179.

In Figure 15, SARSA decreased pressure RMSE to 0.06 after 1700 time steps (1700 simulation runs). The minimum achieved average pressure RMSE by SARSA is 0.04233 at the end, which is larger than the minimum of Q-learning and the EnKF.

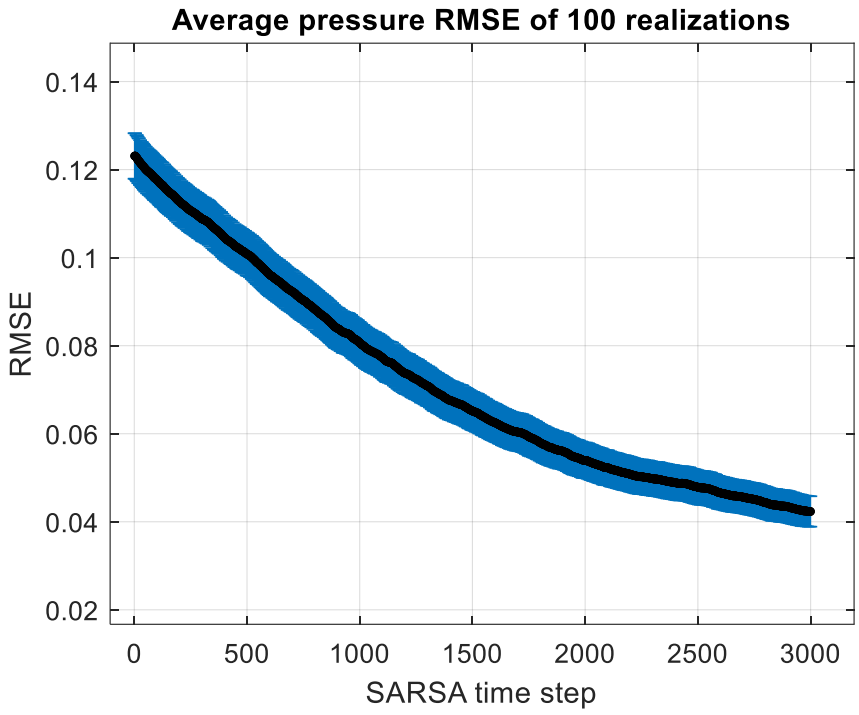

Figure 15 SARSA average RMSE of 100 realizations of observations (pressure values)

In Figure 16, SARSA decreased permeability RMSE to 1.7 after 1308 time steps (1308 simulation runs). The minimum reached average permeability RMSE by SARSA is 1.635 at the end, which is larger than the RMSE of Q-learning and the EnKF.

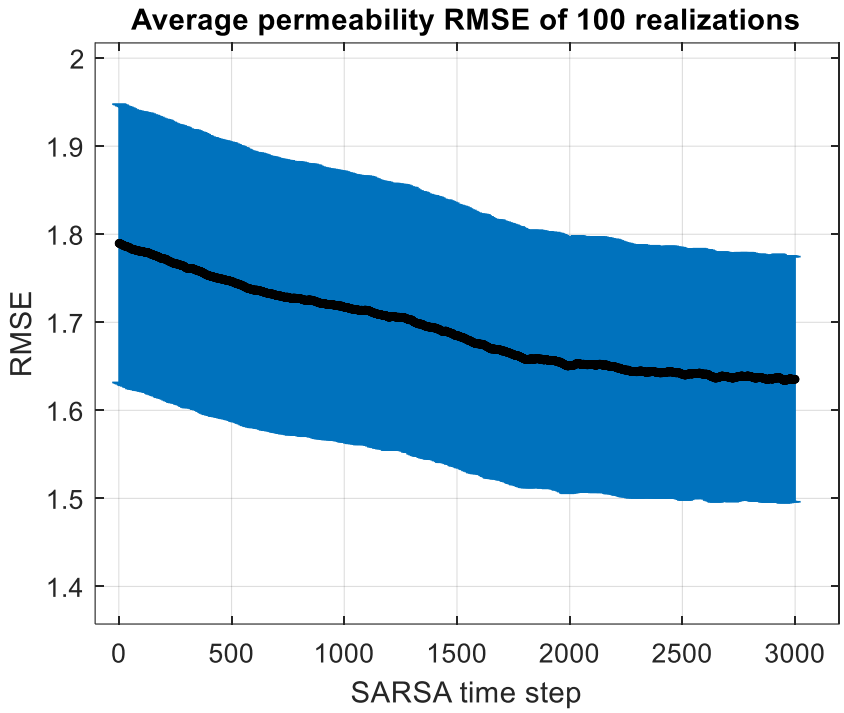

Figure 16 SARSA average RMSE of 100 realizations of true models (permeability values)

3.7 Double Q-learning

The results of the double Q-learning method are generated using the same values used for Q-learning and SARSA, which are discount factor equal to 1, step size (learning rate) equal to 0.95, and epsilon (exploration probability) equal to 0.2. Additionally, percent of change done on each eigenvector parameter (model input) is 0.01 (1%) of its initial value.

Each time step of double Q-learning involves only one simulation run. Before double Q-learning HM process, the average RMSE of 100 pressure realizations was 0.123, and the average RMSE of 100 permeability realizations was 0.179.

In Figure 17, double Q-learning decreased average pressure RMSE to 0.06 after 803 time steps (803 simulation runs). The minimum achieved average pressure RMSE by double Q-learning is 0.02541, which is less than the minimum of SARSA and Q-learning, but larger than the EnKF.

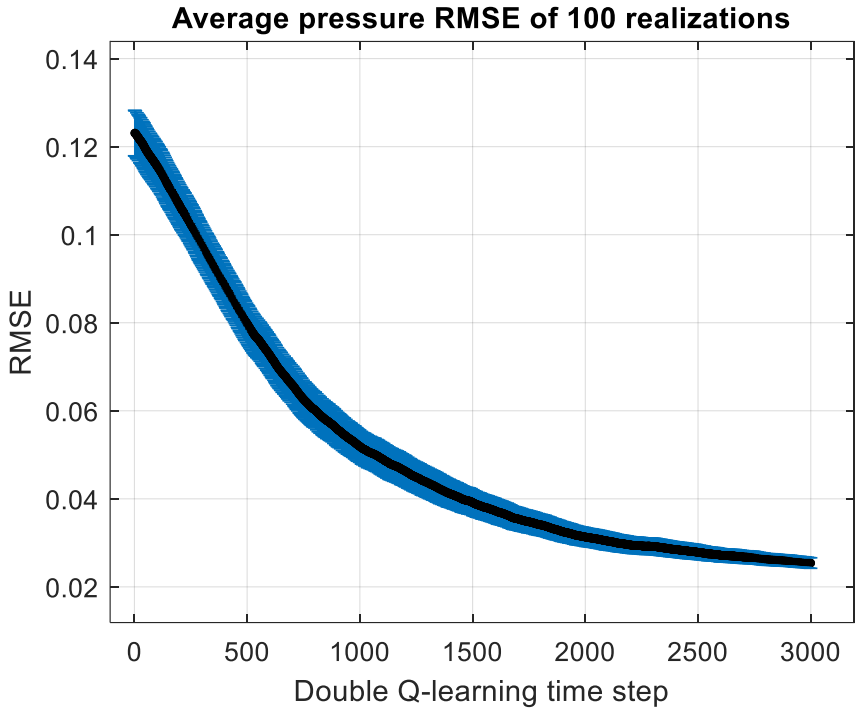

Figure 17 Double Q-learning average RMSE of 100 realizations of observations (pressure values)

In Figure 18, double Q-learning decreased average permeability RMSE to 1.7 after 600 time steps. The minimum average permeability RMSE by double Q-learning is 1.572, which is smaller than Q-learning and SARSA, but larger than the RMSE of the EnKF.

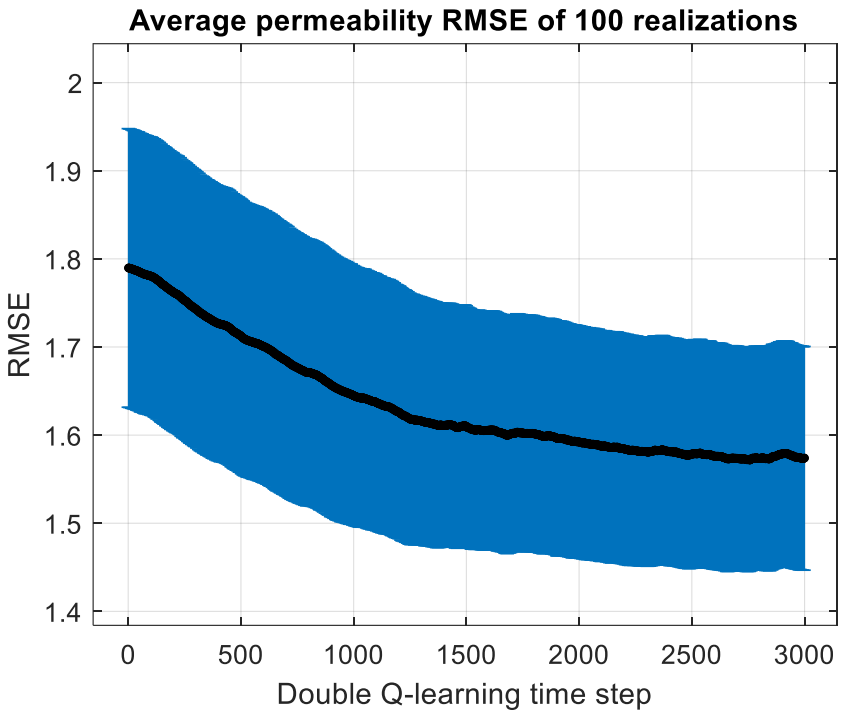

Figure 18 Double Q-learning average RMSE of 100 realizations of true models (permeability values)

3.8 Expected SARSA

The figures of the expected SARSA method are generated using the same values used for Q-learning, SARSA and double Q-learning, which are discount factor equal to 1, step size (learning rate) equal to 0.95 and epsilon (exploration probability) equal to 0.2. Additionally, the percent of change done on each eigenvector parameter (model input) is 0.01 (1%) of its initial value.

Each time step of double Q-learning involves only one simulation run. Before expected SARSA HM process, the average RMSE of 100 pressure realizations was 0.123 and the average RMSE of 100 permeability realizations was 0.179.

In Figure 19, expected SARSA decreased average pressure RMSE to 0.06 after 792 time steps (792 simulation runs). The minimum average pressure RMSE of expected SARSA is 0.02724, which is less than the minimum of SARSA and Q-learning, but larger than the double Q-learning and EnKF.

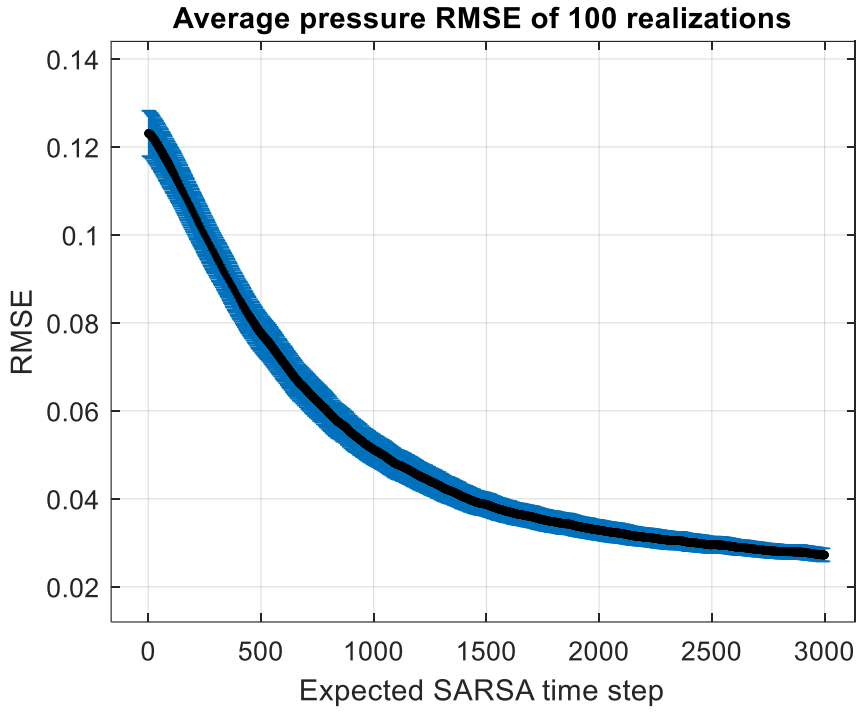

Figure 19 Expected SARSA average RMSE of 100 realizations of observations (pressure values)

In Figure 20, expected SARSA decreased average permeability RMSE to 1.7 after 550 time steps. The minimum average permeability RMSE reached by expected SARSA is 1.574, which is smaller than Q-learning, SARSA, approximately equal to double Q-learning, and larger than the RMSE of the EnKF.

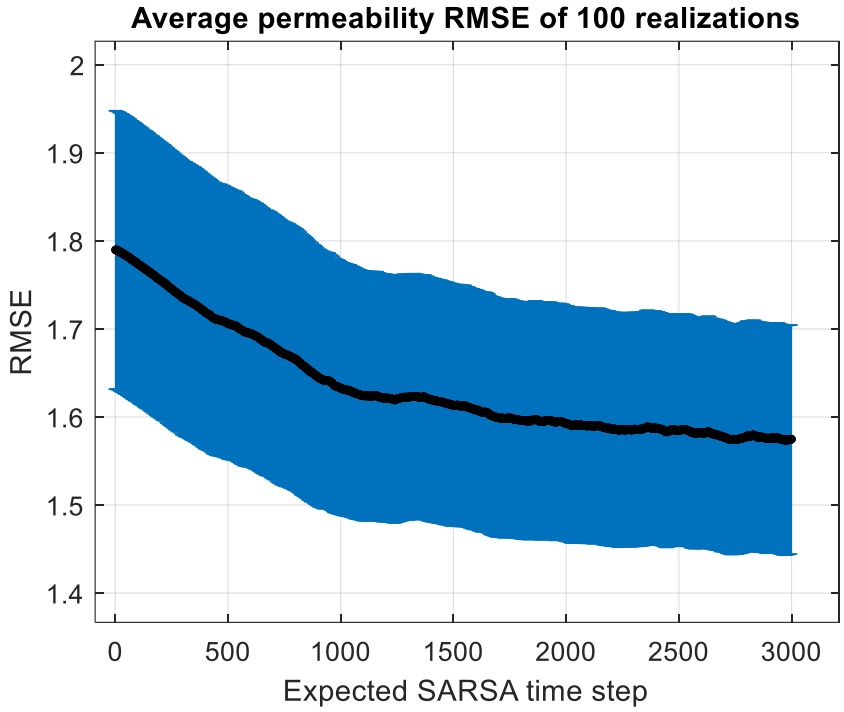

Figure 20 Expected SARSA average RMSE of 100 realizations of true models (permeability values)

Expected SARSA used slightly larger percent of change (action) without diverging the match of permeability values. In the next two figures, the percent of change was modified to 0.02 (2%) of the initial value of each eigenvector parameter.

In Figure 21, expected SARSA decreased average pressure RMSE to 0.06 after 420 time steps (420 simulation runs). The minimum average pressure RMSE of expected SARSA is 0.02264 at the end, which is less than the minimum of Q-learning, SARSA, double Q-learning, and expected SARSA (1%), but larger than the minimum of the EnKF.

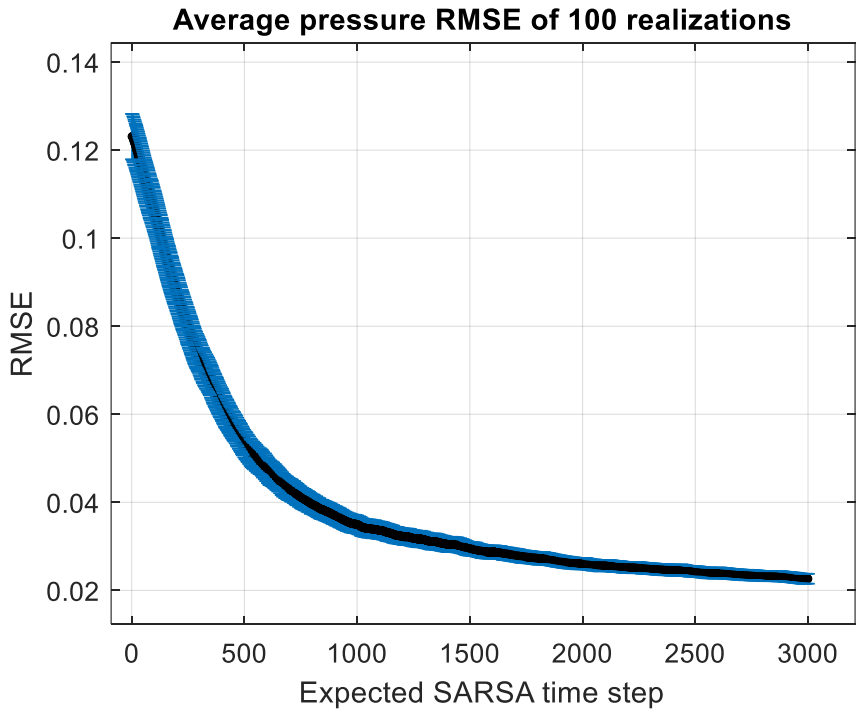

Figure 21 Expected SARSA (2%) average RMSE of 100 realizations of observations (pressure values)

In Figure 22, expected SARSA decreased average permeability RMSE to 1.7 after 350 time steps. The minimum average permeability RMSE of expected SARSA is about 1.6, which is smaller than SARSA, but larger than Q-learning, double Q-learning, Expected SARSA (1%), and the EnKF.

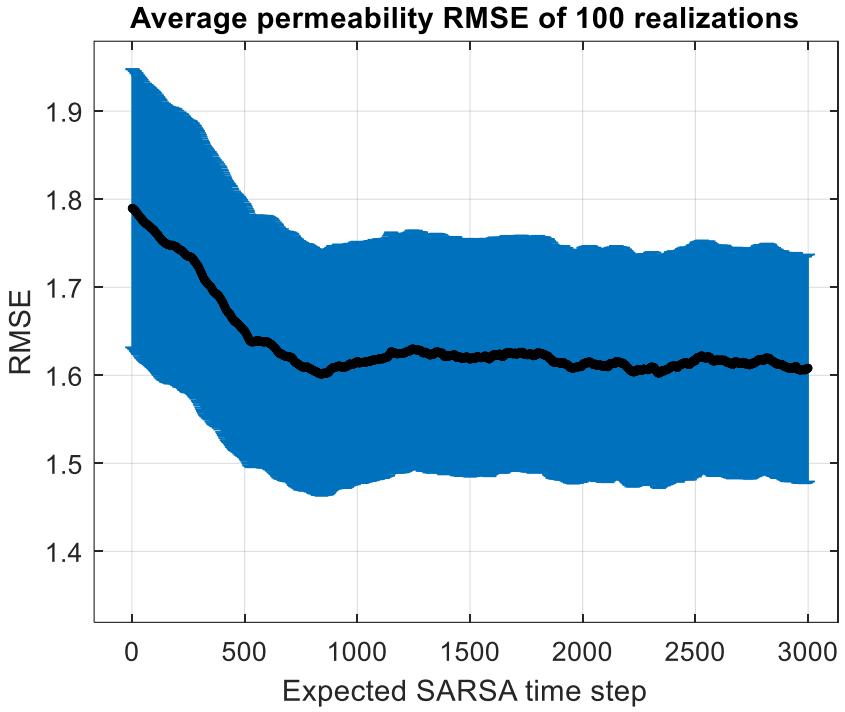

Figure 22 Expected SARSA (2%) average RMSE of 100 realizations of true models (permeability values)

3.9 Summary

This section summarizes the highlights of the thesis outcomes. To compare between the average pressure RMSE of the tabular TD methods, Figure 23 is shown. This figure shows that expected SARSA (2%) is the most efficient and accurate, followed by expected SARSA and Double Q-learning, which have similar performance. On the other hand, SARSA has the largest pressure RMSE values and the least efficiency.

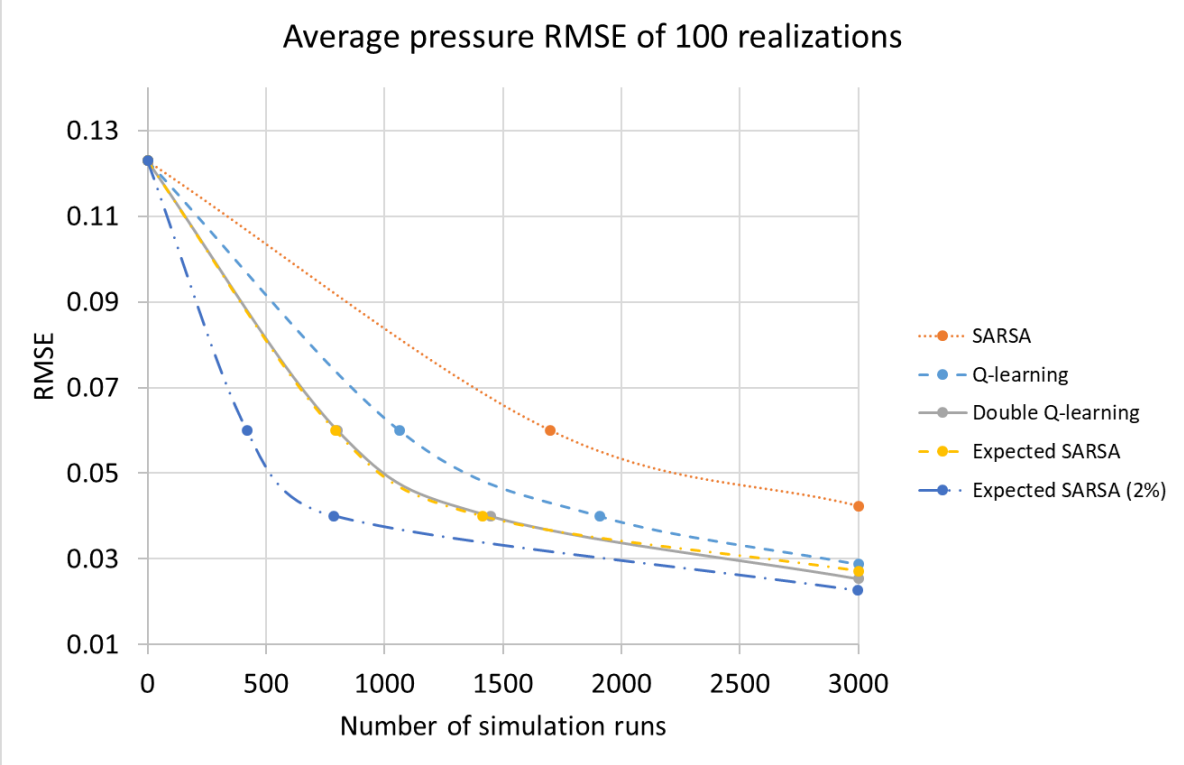

Figure 23 Tabular TD methods average RMSE of 100 realizations of observations (pressure values)

Additionally, Figure 24 compares tabular TD methods in average permeability RMSE, where expected SARSA (2%) exhibits the fastest decrease in permeability RMSE values, but later it flattens out after 836 simulation runs. It seems that the expected SARSA and Double Q-learning are somehow the best in matching permeability values compared to other tabular TD methods. Moreover, SARSA has the worst efficiency and permeability RMSE, which was the same case in pressure RMSE.

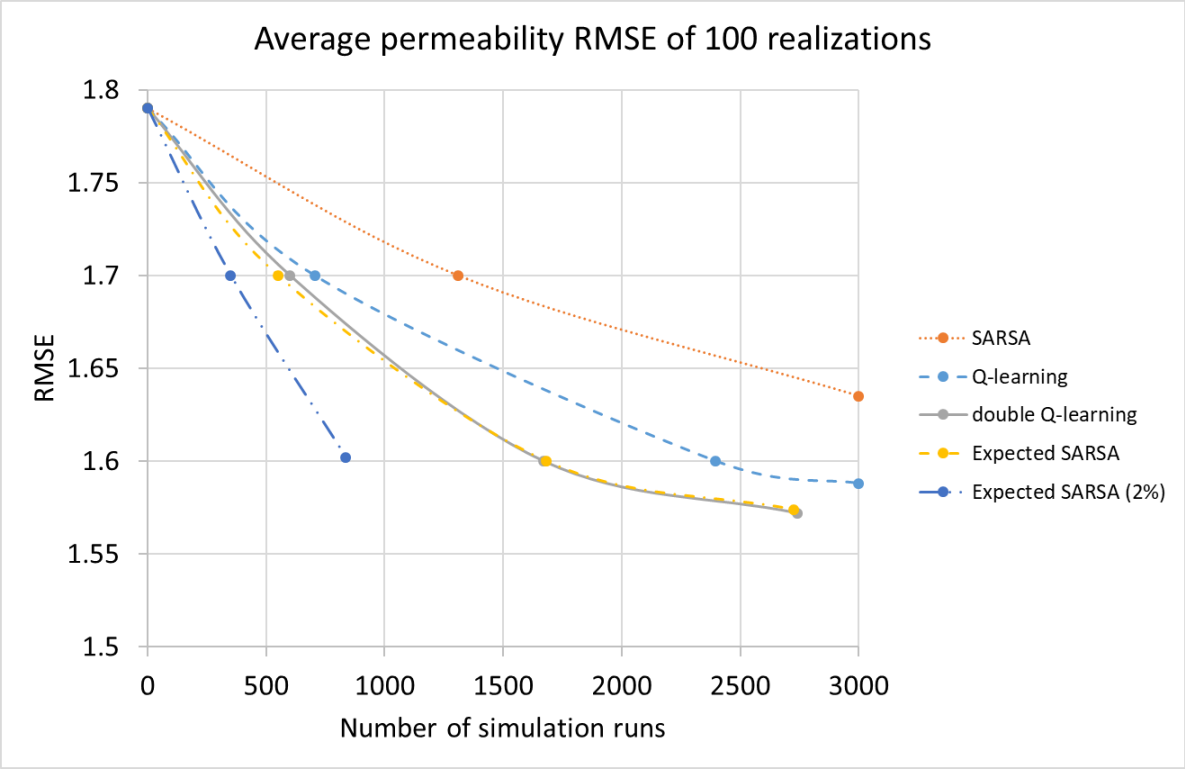

Figure 24 Tabular TD methods average RMSE of 100 realizations of true models (permeability values)

Table 4 All methods average RMSE of 100 realizations of observations (pressure values), and Table 5 All methods average RMSE of 100 realizations of true models (permeability values) Both tables indicate that the EnKF method is more accurate and extremely more efficient than the tabular TD methods.

Table 4 All methods average RMSE of 100 realizations of observations (pressure values)

EnKF		Q-learning		SARSA	
Pressure RMSE	**N simulation runs**	**Pressure RMSE**	**N simulation runs**	**Pressure RMSE**	**N simulation runs**
0.123	0	0.123	0	0.123	0
0.05	15	0.06	1065	0.06	1700
0.04	30	0.04	1910	0.04233	3000
0.02	420	0.0288	3000		
0.013	2235				
Double Q-learning		**Expected SARSA**		**Expected SARSA (0.02)**	
Pressure RMSE	**N simulation runs**	**Pressure RMSE**	**N simulation runs**	**Pressure RMSE**	**N simulation runs**
0.123	0	0.123	0	0.123	0
0.06	803	0.06	792	0.06	420
0.04	1450	0.04	1415	0.04	786
0.02541	3000	0.02724	3000	0.02264	3000

Table 5 All methods average RMSE of 100 realizations of true models (permeability values)

EnKF		Q-learning		SARSA	
Permeability RMSE	**N simulation runs**	**Permeability RMSE**	**N simulation runs**	**Permeability RMSE**	**N simulation runs**
1.79	0	1.79	0	1.79	0
1.68	15	1.7	704	1.7	1308
1.6	165	1.6	2396	1.635	3000
1.5	2235	1.588	3000		
Double Q-learning		**Expected SARSA**		**Expected SARSA (0.02)**	
Permeability RMSE	**N simulation runs**	**Permeability RMSE**	**N simulation runs**	**Permeability RMSE**	**N simulation runs**
1.79	0	1.79	0	1.79	0
1.7	600	1.7	550	1.7	350
1.6	1670	1.6	1682	1.602	836
1.572	2740	1.574	2725		

Figure 25, Figure 26 and Table 6 All methods minimum average RMSE of 100 realizations of observations (pressure values) and true models (permeability values) show the minimum average pressure and permeability RMSE of all methods. The EnKF has the minimum pressure and permeability RMSE. Comparing only tabular TD methods, Expected SARSA (2%) has the minimum pressure RMSE, and Double Q-learning has the minimum permeability RMSE.

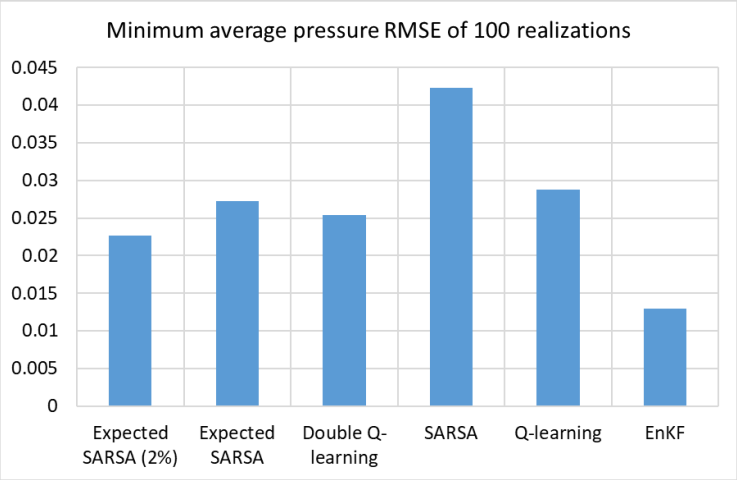

Figure 25 All methods minimum average RMSE of 100 realizations of observations (pressure values)

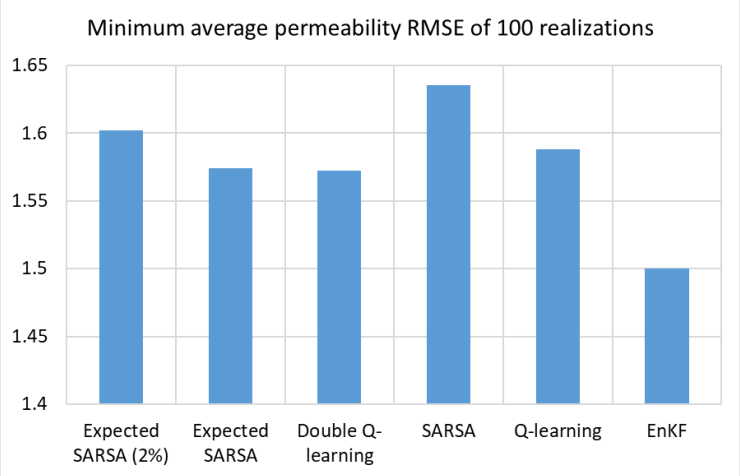

Figure 26 All methods minimum average RMSE of 100 realizations of true models (permeability values)

Table 6 All methods minimum average RMSE of 100 realizations of observations (pressure values) and true models (permeability values)

	EnKF	Q-learning	SARSA	Double Q-learning	Expected SARSA	Expected SARSA (0.02)
Min Pressure RMSE	0.013	0.0288	0.04233	0.02541	0.02724	0.02264
Min Permeability RMSE	1.5	1.588	1.635	1.572	1.574	1.602

Based on CPU time taken by each method, SARSA needed the least time, while the EnKF took the largest time. However, the time step of the EnKF involves 15 simulation runs (ensembles). Furthermore, expected SARSA required the largest CPU time compared to other tabular TD methods. Table 7 CPU time of 100-time steps for 100 realizations (seconds)shows the CPU time taken by all methods.

Table 7 CPU time of 100-time steps for 100 realizations (seconds)

CPU time (seconds)	
EnKF	4.7
Q-learning	2.32
SARSA	1.7
Double Q-learning	2.85
Expected SARSA	3.22

3.10 Conclusion

This study showed that RL can solve basic HM problems. Tabular TD methods were able to match 50 observations of 100 realizations by adjusting 10 reservoir model inputs. Even though they are compatible only with discrete actions and states, they lowered the RMSE of the HM problem that has only continuous variables. Furthermore, they do not assume or require any condition on HM problems, e.g. having continuous or discrete variable, linear or non-linear, and gaussian or non-gaussian model. Thus, RL is general and should be applicable to different varieties of HM problems. Moreover, RL can be considered as the most similar method to manual HM; it mimics what a petroleum engineer is trying to do to solve reservoir HM problems.

On the other hand, the applied tabular TD methods (in this study) lack some characteristics that make them far from being considered for practical HM problems. One of their cons is the lack of parallel computation, which is available in the form of ensembles in the EnKF, which improves efficiency, and provides multiple solutions that are needed for uncertainty quantification, which is critical because of the non-uniqueness nature of HM solution. Additional disadvantage is that the optimum design and representation of the value function are not straight forward and require experimenting. Further drawback is the possible need of trial and error process to optimize RL algorithms parameters (step size, exploration & discount factor) to improve their performance. Additionally, all tabular TD methods could not reach the minimum RMSE of the EnKF. Finally, they were extremely inefficient; they needed many simulations runs to start solving a basic HM problem.

References

Cancelliere, M., Verga, F., & Viberti, D. (2011). Benefits and Limitations of Assisted History Matching. Society of Petroleum Engineers.

Chang, H., & Zhang, D. (2014). History matching of statistically anisotropic fields using the Karhunen-Loeve expansion-based global parameterization technique. *Computational Geosciences*, 265-282.

Geramifard, A., Walsh, T. J., Tellex, S., Chowdhary, G., Roy, N., & How, J. P. (2013). A Tutorial on Linear Function Approximators for Dynamic Programming and Reinforcement Learning. *Foundations and Trends® in Machine Learning, 6*(4), 375-451. doi:10.1561/2200000042

Gu, Y., & Oliver, D. S. (2005). History Matching of the PUNQ-S3 Reservoir Model Using the Ensemble Kalman Filter. *Society of Petroleum Engineers*.

Guevara, J. L., Patel, R. G., & Trivedi, J. J. (2018). Optimization of Steam Injection for Heavy Oil Reservoirs Using Reinforcement Learning. Society of Petroleum Engineers. .

Hasselt, H. V. (2010). Double Q-learning. *Advances in Neural Information Processing Systems 23 (NIPS 2010)* (pp. 2613--2621). Curran Associates, Inc.

Haugen, V. E., Naevdal, G., Natvik, L.-J., Evensen, G., Berg, A. M., & Flornes, K. M. (2008). History Matching Using the Ensemble Kalman Filter on a North Sea Field Case. *Society of Petroleum Engineers*.

Hessel, M., Modayil, J., Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., . . . Silver, D. (2017). Rainbow: Combining Improvements in Deep Reinforcement Learning. *AAAI.*

Hourfar, F., Bidgoly, H. J., Moshiri, B., Salahshoor, K., & Elkamel, A. (2019). A reinforcement learning approach for waterflooding optimization in petroleum reservoirs. *Engineering Applications of Artificial Intelligence,* 98-116.

John, G. H. (1994). When the best move isn't optimal:Q-learning with exploration. *In Proceedings of the National Conference on Artificial Intelligence.* Cambridge, MA: AAAI/MIT Press.

Ma, H., Yu, G., She, Y., & Gu, Y. (2019). Waterflooding Optimization under Geological Uncertainties by Using Deep Reinforcement Learning Algorithms. Society of Petroleum Engineers. doi:10.2118/196190-MS

Miftakhov, R. (2019, March 5). *Deep Reinforcement Learning for Petroleum Reservoir Optimization.* Retrieved from Linkedin: https://www.linkedin.com/pulse/deep-reinforcement-learning-petroleum-reservoir-ruslan-miftakhov

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., . . . Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 529-533.

Oliver, D. S., & Chen, Y. (2010). Recent progress on reservoir history matching: A review. *Computational Geosciences*.

Poole, D., & Mackworth, A. (2017). *Artificial Intelligence: Foundations of Computational Agents* (2nd ed.). Cambridge University Press.

Rajeswaran, A., Lowrey, K., Todorov, E. V., & Kakade, S. M. (2017). Towards Generalization and Simplicity in Continuous Control. *Neural Information Processing Systems Conference (NIPS)* (pp. 6550-6561). Curran Associates, Inc.

Rummery, G. A., & Niranjan, M. (1994, January 1). *On-Line Q-Learning Using Connectionist Systems.* Retrieved from https://core.ac.uk/display/21255097

Rwechungura, R. W., Dadashpour, M., & Kleppe, J. (2011). Advanced History Matching Techniques Reviewed. Society of Petroleum Engineers.

Schrum, J. (2008, November). *Competition Between Reinforcement Learning Methods in a Predator-Prey Grid World.* Retrieved from The University of Texas at Austin: http://nn.cs.utexas.edu/?schrum:tech08

Seijen, H. V., Hasselt, H. V., Whiteson, S., & Wiering, M. (2009). A theoretical and empirical analysis of Expected Sarsa. *IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*.

Silver, D. (2015). *Introduction to Reinforcement Learning.* Retrieved from UCL Course on Reinforement Learning: http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Driessche, G. V., . . . Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, 484-489.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning:An Introduction.* MIT Press.

Szepesvári, C. (2010). *Algorithms for reinforcement learning.* Morgan & Claypool Publishers.

Tavassoli, Z., Carter, J. N., & King, P. R. (2004). Errors in History Matching. *Society of Petroleum Engineers*.

Watkins, C. (1989). *Learning From Delayed Rewards.* PhD Thesis, University of Cambridge, England.

Appendix

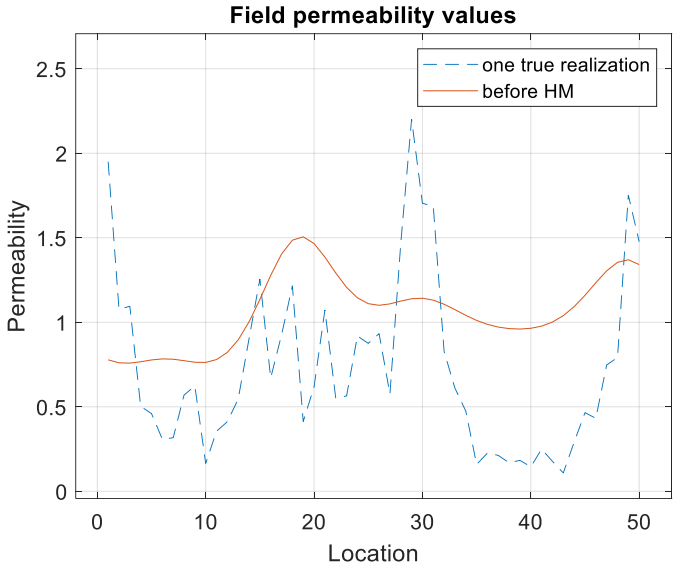

Figure 27 The true realization permeability model is generated by 50 eigenvectors, while the before history matching permeability model is created by only 10 eigenvectors

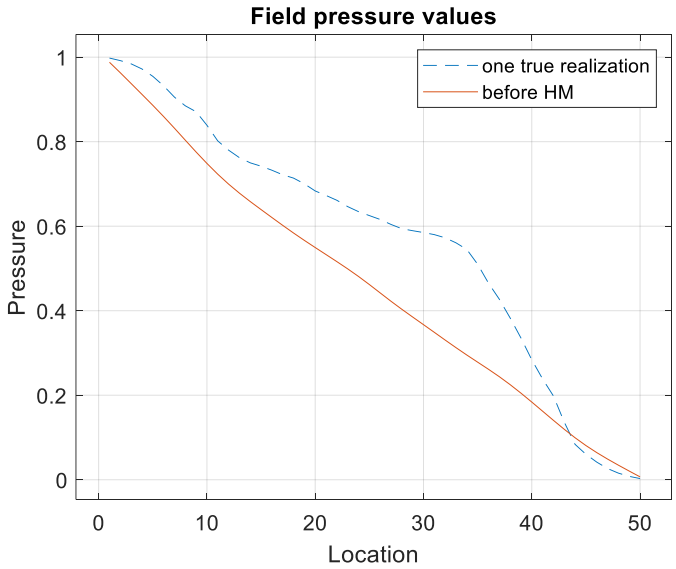

Figure 28 Pressure values are calculated from the permeability values of the earlier figure

Vitae

Name	: Khalid Labib Alsamadony
Nationality	: Egyptian
Email	: khalidalsamadony@outlook.com
Academic Background	: BS & MSc in Petroleum Engineering, KFUPM
Journal Publication	: Sauerer, B., Craddock, P. R., Aljohani, M. D., Alsamadony, K. L., & Abdallah, W. (2017). Fast and accurate shale maturity determination by Raman spectroscopy measurement with minimal sample preparation. *International Journal of Coal Geology*, *173*, 150–157. doi: 10.1016/j.coal.2017.02.008