Universitetet i Stavanger

**FACULTY OF SCIENCE AND TECHNOLOGY**

# MASTER'S THESIS

| | |
|---|---|
| Study programme/specialisation:<br><br>Master of Science in Petroleum Engineering<br><br>Reservoir Engineering | Spring / Autumn semester, 2020<br><br>Open/Confidential |
| Author: Aizhan Kengessova | …………………………………………<br>(signature of author) |
| Programme coordinator:<br><br>Supervisor(s): Pål Østebø Andersen<br>Dag Chun Standnes<br>Jan Inge Nygård | |
| Title of master's thesis:<br><br>**Prediction efficiency of immiscible Water Alternating Gas Performance by LSSVM-PSO algorithms** | |
| Credits: 30 | |
| Keywords:<br>Water Alternating Gas Injection, Least Squares Support Vector Machines (LSSVM), Particle Swarm Optimization (PSO), Recovery Performance Prediction. | Number of pages: …………………<br><br>+ supplemental material/other: …………<br><br>Stavanger, ……………………….<br>date/year |

Title page for Master's Thesis
Faculty of Science and Technology

# Contents

# Abstract

In this work the aim is developing LSSVM-PSO model capable of capturing the interplay between the most influential parameters (mechanisms) and recovery factor (RF) of WAG process in layered reservoirs. In a previous work 1840 Black Oil Model simulations were run for a 2D model with multiple layers, an injector and a producer, and used to derive a dimensionless number correlating reservoir heterogeneity, WAG hysteresis, gravity, mobility ratio and WAG ratio to predict recovery factor (as measured after 1.5 injected pore volumes). Given that only one parameter, the dimensionless number, was used to correlate RF, a significant data scatter was observed.

In this work the database is expanded by running 824 new simulations using new hysteresis parameters values. The Machine Learning algorithm Least Squares Support Vector Machine (LSSVM) is used to correlate RF with representative input parameters, such as characteristic mobility ratios, gravity numbers, heterogeneity factor and more. The appropriate number of effective input parameters was obtained by reducing the set of independent input parameters to dimensionless groups. Particle Swarm Optimization was used to optimize the LSSVM algorithm parameters.

The trained LSSVM-PSO model could serve as an effective screening tool in uncovering important trends of parameter variation and improve the efficacy of uncertainty analyses.

# Acknowledgements

I would like to express my gratitude to my supervisor, Pål Østebø Andersen for his overall guidance and support in theoretical questions throughout this work. He is an inexhaustible source of good ideas and thoughtful mentor. I am also grateful for my co-supervisor Dag Chun Standnes reviewing this work and for sharing his practical experience in specific questions.  I wish to thank my co-supervisor Jan Inge Nygård for his generous help, time and for giving me the learning opportunity while implementing LSSVM-PSO code on Python. This thesis was not as nearly as possible without his great contribution.

I wish to appreciate all my friends at UiS who spend with me two amazing years in Stavanger and contributed in my growth as a person. Also, I wish to thank my dearest friend and mentor at work Zhakashev Ganiyet who encouraged me to take a Master's degree in Norway at first place.

Last but not least, my master program could not have been completed without unconditional love and support of my family. No words describe my appreciation for their support and love to me.

# Nomenclature

| | | |
|---|---|---|
| $\lambda_D$ | = | mobility of the displacing fluid, $(\text{Pa} * \text{s})^{-1}$ |
| $\lambda_d$ | = | mobility of the displaced fluid, $(\text{Pa} * \text{s})^{-1}$ |
| $\lambda_i$ | = | phase mobility, $(\text{Pa} * \text{s})^{-1}$ |
| $\sigma$ | = | interfacial tension, IFT (N/m) |
| $N_{ca}$ | = | capillary number |
| $\vartheta$ | = | Darcy velocity (m/s) |
| $E_d$ | = | volumetric sweep efficiency |
| $E_v$ | = | volumetric (macroscopic) sweep efficiency |
| $\mu_i$ | = | Viscosity, Pa $*$ s |
| $\rho_i$ | = | phase density, kg/m$^3$ |
| $\Delta\rho$ | = | density difference, kg/m$^3$ |
| $C$ | = | Lands's trapping parameter |
| $F_H$ | = | heterogeneity multiplier |
| $F_G$ | = | gravity multiplier |
| $k_{ri}$ | = | relative permeability |
| $k_{ri}^{max}$ | = | relative permeability endpoints |
| $K_x, K_z$ | | horizontal and vertical absolute permeability, m |
| $L_x$ | = | distance from injector to producer, m |
| $L_y$ | = | width of reservoir, m |
| $L_y$ | = | total height of reservoir, m |
| $M$ | = | mobility ratio |
| $H_\phi$ | = | Pore volume, m$^3$ |
| $h_i$ | = | layer height, m |
| $M_{WAG}$ | = | simple characteristic three phase mobility ratio |
| $M^*$ | = | total injection time |
| $n_i$ | = | Corey exponents, |
| $N_G$ | = | gravity number |
| $r_w$ | = | water volume fraction in a wag cycle |
| $s_i$ | = | local phase saturation |
| $s_{ir}$ | = | residual phase saturation |
| $S_i$ | = | normalized saturation |
| $t$ | = | time, seconds |
| $T^{g-hc}$ | = | gas half-cycle length, seconds |
| $T^{tot}$ | = | Total injection time, seconds |
| $T^{cycle}$ | = | Total WAG cycle length, days |
| $T^{w-hc}$ | = | water half-cycle length, seconds |
| $\alpha$ | = | hysteresis parameter |
| $\tau$ | = | time scale, seconds |
| $\varphi$ | = | porosity |
| $x$ | = | horizontal direction towards producer, m |
| $z$ | = | vertical direction downwards, m |

**Subscripts and Superscripts**

| | | |
|---|---|---|
| $*$ | = | characteristic value, |
| $1pv$ | = | 1 PV |
| $arit$ | = | arithmetic |
| $g$ | = | gas |
| $G$ | = | gravity |
| $harm$ | = | harmonic |
| $i$ | = | phase |
| $j$ | = | layer |
| $o$ | = | oil |
| $res$ | = | residence |
| $init$ | = | initial reservoir conditions |
| $seg$ | = | segregation |
| $T$ | = | total |
| $w$ | = | water |

# Table of Figures

## List of Tables

Chapter 1

# Introduction

## 1.1 Background and Motivation

About two thirds of worldwide oil production belongs to mature fields, and production amount from new discovered fields is on a steady decline (O'Brien et al. 2016). This makes reconsideration of mature fields' potential more relevant. To optimize production of such fields, EOR technologies are widely applied across the world. Thermal and chemical EOR projects dominate in sandstone reservoirs while gas injection and water-based methods are primarily used in carbonates (Manrique et al. 2010).

In *Figure 1. 1* we see a resource overview for the largest oil fields on the Norwegian Continental Shelf (NCS), which comprises produced oil, remaining oil reserves, and (expected) amount of residual oil once planned production stage ends.

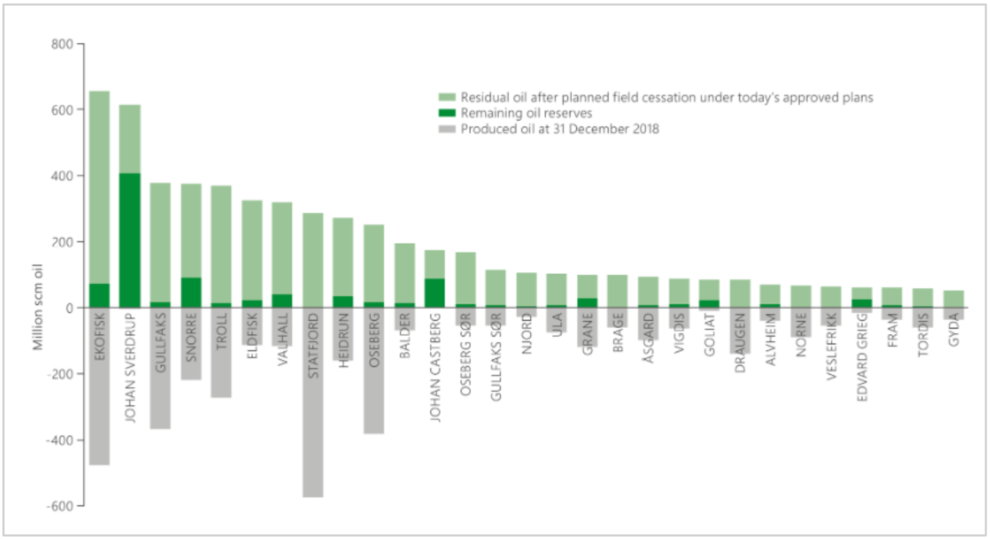


*Figure 1. 1* – Resource overview for fields, information by 31.12.2018 (NPD 2019)

Many large fields on NCS are now in a mature phase and have produced large percentage of their original reserves. In *Figure 1. 2* we see the proportion of remaining oil reserves for a number of fields relative to original. The size of the circles indicates remaining reserves. That means some greener fields as Johan Sverdrup and Johan Castberg in the North and Barents Seas respectively appear on left side of the plot. They are under development and will contribute to a continued high level of production during the 2020s (NPD 2019). Mature fields took place on the right side of the graph and proportion of their remained reserves relative to initial ones has gone lower over the years. However, some fields such Snorre, Valhall, Heidun still have considerable reserves even relative to some of greener fields.

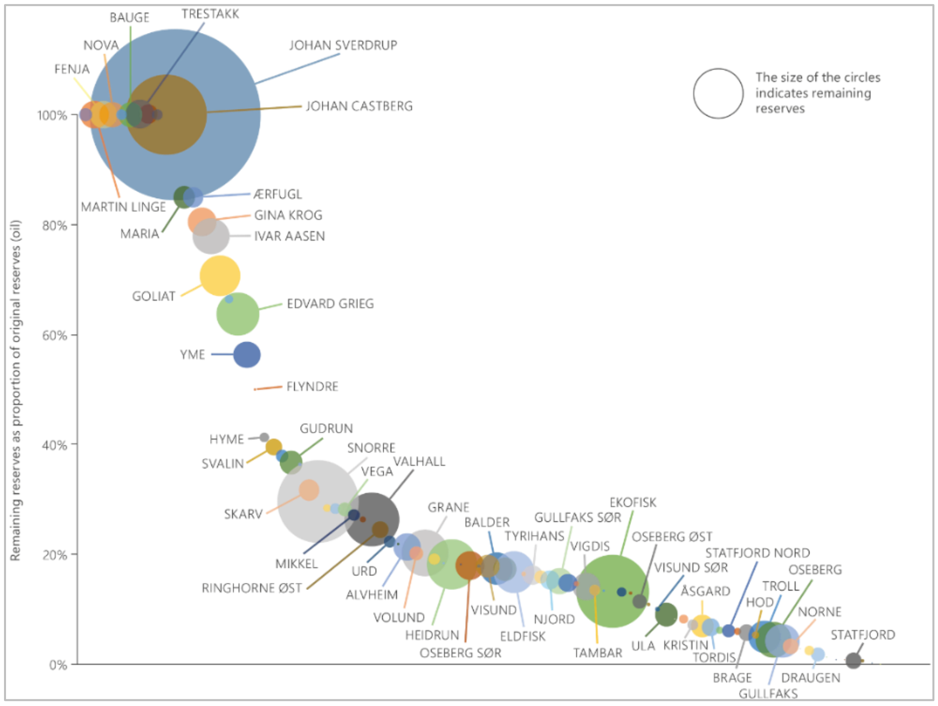


*Figure 1. 2*– Remaining proportion of the original oil reserves and the size of remaining oil reserves (NPD 2019)

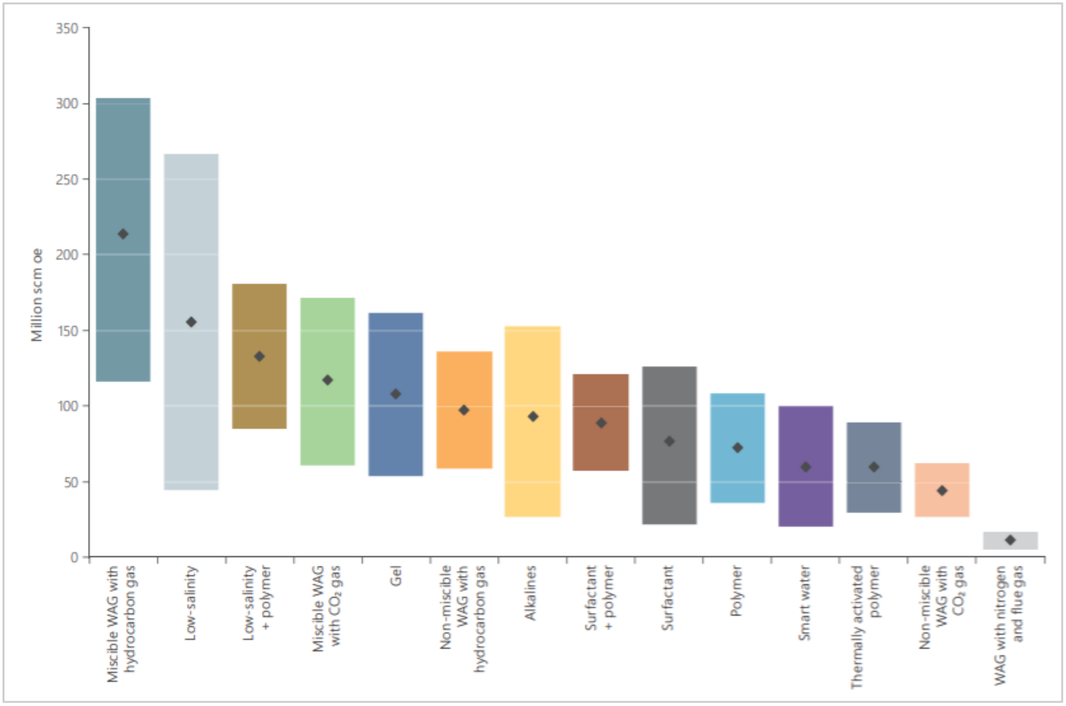


*Figure 1. 3* – Scaled EOR potential for method with uncertainty (NPD 2019)

In 2017 the technical potential in 27 fields was reported by Norwegian Petroleum Directorate (NPD) and that number was expanded till 46 fields in 2019 (NPD 2019). The technical potential of the implementation of EOR methods to adopt to an existing or planned facility on fields were ranked by operator companies. A flat oil price of 60 USD per barrel and discount rate of 7% has been used for economics analysis. Scaled EOR potential for each field is presented in *Figure 1. 3*. Scaling factor which is used for calculating scaled volumes is estimated by combining operational and economic factor with values from 0 to 1 for each of the methods. From the *Figure 1. 3* we can see that gas-based Water Alternating Gas (WAG) injection has high potential mainly for fields with existing equipment for its implementation. Also, low-salinity water and smart water injection are listed as high rank ones.

With situation of oil price of USD 30 per barrel to date 14.04.2020, concentration on mature fields is reasonable. For example, one of the effects of the price decline is that in 2015, many international and independent projects struggled to generate enough cash to generate enough cash to cover their spending and dividends, even as they severely cut spending and greenfield projects. Four of the biggest oil companies (Royal Dutch Shell, BP LC, Exxon Mobil Corp. and Chevron Corp.), outstripped cash flow by more than a combined USD 20 billion during the first half of 2015 (Sarah Kent, Justin Scheck 2015).

Based on facts described above, studying EOR methods and developing tools for diminishing complexity of decision-making process on implementing them are actual these days.

## 1.2 Thesis objectives and novelty

The objectives correspond to answering the following questions:

- How well can the LSSVM-PSO model predict WAG performance in comparison to model of previous work based on one dimensionless number?
- How does convergence behavior depend on the number of selected dataset fractions for LSSVM-PSO model?
- How good is the prediction efficiency when previously constant WAG cycle length changes?
- How can hysteresis parameters be chosen to achieve desired hysteresis effect?

The novelty characteristics of this work can be summarized in these forms:

- Creation of LSSVM algorithm-based model for the WAG efficiency evaluation purpose.
- The trained LSSVM model could be an effective tool in uncovering important trends of parameter variation and improve the efficacy of uncertainty analyses.

Chapter 2

# Theoretical part of work

## 2.1 Introduction to WAG process

Conventional water and gas injection methods are well known as secondary recovery methods. At the same time, those methods can lack of efficiency that lead to major problems in conditions of unfavorable mobility ratio between oil and displacing phase or low displacement effectiveness. During the displacement process of gasflooding gas fingering can be caused by inefficient mobility ratio leading to reduction of sweep efficiency. Also, presence of indications of heterogeneity as fractures, high permeable layers might cause early breakthrough of gas into production wells. Therefore, cyclic injection of water slugs along with gas slugs helps to maintain front stability and improve volumetric sweep efficiency.

One of the important mechanisms in displacing process is gravity segregation. It is very high for gasflooding process due to high difference between densities of the phases, which negatively affects volumetric sweep efficiency even microscopic sweep is higher in zones contacted by the displacing flood than that for waterflooding. For waterflooding gravity segregation has less effect because of less difference between water and oil densities in comparison to previous case. WAG injection process limits the negative effect of gravity segregation and it is not that severe as in pure gasflooding process and still allows to have higher displacement efficiency than in waterflooding process.

Water Alternating Gas (WAG) method was introduced to overcome problems with mobility ratio between oil and injected gas in gas flooding due to low viscosity and high relative permeability of gas (Green and Willhite 2018). Injection of water helps to stabilize the flooding front through enhancing macroscopic sweep efficiency, while injection of gas contributes to improved microscopic sweep efficiency of the contacted reservoir regions (Christensen, Stenby', and Skauge 2001). In other words, WAG utilizes the advantages of two traditional methods whilst minimizing their individual downsides.

WAG method has different types based on process driving mechanism and fluids implementation. The *Figure 2. 1* shows water alternating gas method types based on variation of different attributes.

Based on process, this EOR method is separated into three types. Conventional WAG involves cycles of water and gas alternately injected as shown in *Figure 2. 2.* In hybrid $CO_2$ +WAG, the conventional WAG process is modified with cycles of $CO_2$ injection. Simultaneous

Water and Gas injection (SWAG) encompasses a surface-prepared mixture of water and gas that is injected into the reservoir. Despite their differences SWAG is still classified as a WAG process type.

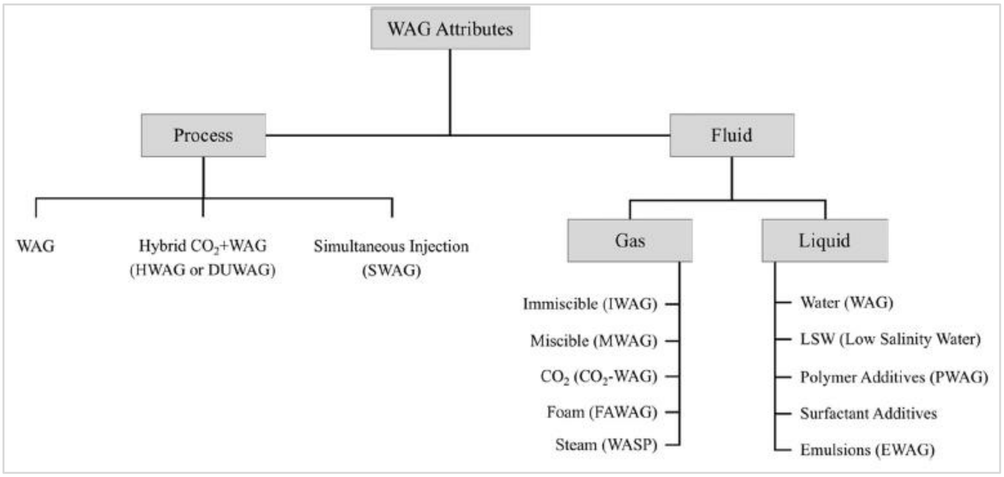


*Figure 2. 1*– Variations of WAG processes based on different attributes (Afzali, Rezaei, and Zendehboudi 2018)

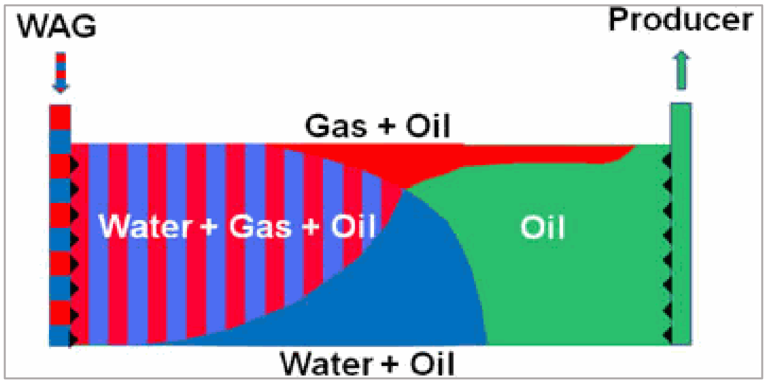


*Figure 2. 2*– Schematic representation of immiscible WAG injection in a reservoir (Bourgeois, Joubert, and Dominguez 2019)

There are multiple WAG types as dependent on fluid type and composition. However, the most relevant classification is whether the injected gas cycles experience miscibility conditions or not. Hence, they are commonly referred to as miscible WAG (MWAG) or immiscible WAG (IWAG) processes. Schematic illustrations of both methods are shown in *Figure 2. 2* and *Figure 2. 3*. The main difference of two figures is presence of miscible zone for MWAG. WAG miscibility is highly dictated by reservoir conditions (temperature, pressure, and depth) and the properties of the displaced phase (oil) and injected fluids (water and gas). As oil and gas approach miscibility, significant mass transfer occurs. While mass transfer in immiscible process is limited to gas being dissolved in oil, in the miscible process both gas and oil have mass transfer with each other, thus ultimately becoming practically the same phase. The conditions that enables miscibility during WAG injection involves increasing the concentration of light components in the injected gas to reach the Minimum Miscibility Concentration (MMC), or by maintaining a sufficiently high pressure above the Minimum Miscibility Pressure (MMP), or a combination of these (Green and Willhite 2018).

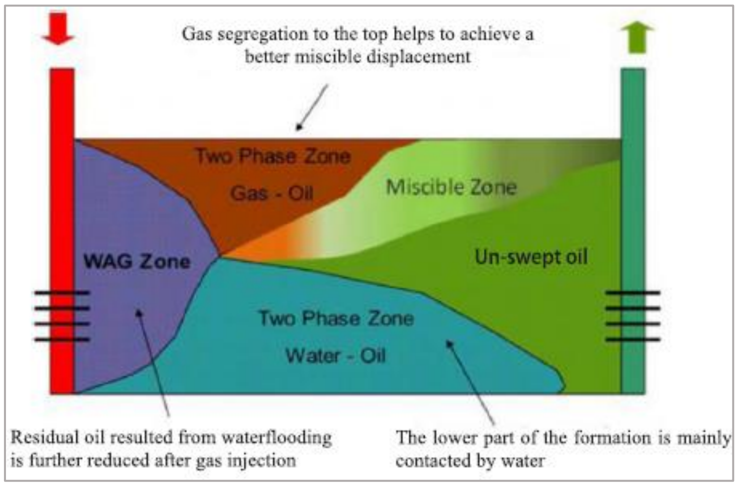

Gas segregation to the top helps to achieve a better miscible displacement
Two Phase Zone
Gas - Oil
Miscible Zone
WAG Zone
Un-swept oil
Two Phase Zone
Water - Oil
Residual oil resulted from waterflooding is further reduced after gas injection
The lower part of the formation is mainly contacted by water

*Figure 2. 3*– Schematic representation of miscible WAG injection in a reservoir (modified after Luis et al.)

To successfully design any EOR strategy, understanding the main mechanisms is highly important not only in terms of resulting recovery factor but also economic feasibility of overall project. The WAG mechanisms, and the underlying processes, can be quite comprehensive in terms of its physics and subsurface uncertainties. Moreover, its overall efficiency depends on when it was implemented, ie. whether it was implemented as a secondary or tertiary process of the lifecycles of the field.

There are mechanisms of WAG process, which can improve oil recovery factor

- Improved volumetric sweep by water following gas.
- Oil viscosity reduction resulting from gas dissolution.
- Oil swelling by dissolved gas.
- Interfacial tension (IFT) reduction.
- Residual oil saturation reduction due to three-phase and hysteresis effects.

The intended purpose of WAG mechanisms is to improve displacement efficiency of reservoir fluids, as compared to single phase flooding processes. This can happen by decreasing the mobility ratio M (normally M>>1 for gas flooding), which is defined as follows:

$$M = \frac{\lambda_D}{\lambda_d} \qquad (1.1)$$

where $\lambda_D$ is the mobility of the displacing fluid (water or gas) and $\lambda_d$ is the mobility of the displaced fluid (e.g., oil). M affects both macro- and micro- sweep efficiencies. This is important parameter as it directly affects to the volumetric (macroscopic) sweep efficiency ($E_v$).

The improvement in displacement efficiency can also happen by increasing the capillary number ($N_{ca}$), which is given by:

$$N_{ca} = \frac{\vartheta \mu}{\sigma} \qquad (1.2)$$

where $\sigma$ is the interfacial tension, IFT (N/m), µ refers to the viscosity of the displacing fluid (Pa.s), and $\vartheta$ is the Darcy velocity (m/s). The capillary number is connected to microscopic (displacement) sweep efficiency ($E_d$), as high $N_{ca}$ contributes to easier displacement of residual oil from the pores (Afzali, Rezaei, and Zendehboudi 2018).

Most of the EOR methods that aim to increase the capillary number is focused on decreasing the interfacial tension between the displacing and displaced fluids. Examples of where this happens is for surfactant and thermal EOR methods. In case of miscible WAG displacement, capillary number can go towards infinity as complete miscibility assumes almost zero interfacial tension between gas and oil. The total oil recovery efficiency ($E$) results from a combination of both microscopic displacement efficiency ($E_d$) and volumetric sweep efficiency ($E_v$):

$$E = E_d * E_v \qquad (1.3)$$

In this thesis IWAG process is considered for simplification purpose. Injecting water and gas in an alternating way will result in complicated saturation behavior in the reservoir, since gas and water saturations will tend to fluctuate as they are cyclically injected. This gives rise to three phase relative permeability behavior (oil, gas and water), which will need to be described through various correlations. The relative permeability can also be cycle dependent. (Larsen and Skauge 1998).

There are variety of reservoir properties and parameters influencing the WAG process efficiency according to literature. Common factors presented in the literature are reservoir heterogeneity, relative permeability, hysteresis, wettability, and gravity. The failure of EOR projects are often connected to reservoirs with high heterogeneity. High stratification of reservoirs makes gas injection process uneconomical in majority of cases because of problems of early gas breakthrough. Properties as flow connection between reservoir layers, stratification, relation of viscous -to-gravity forces mainly control vertical displacement efficiency. The cross flow usually negatively affects the displacement process and recovery factor Gravity segregation in homogeneous models has adversary effect and leads to low recovery efficiency when single phase injection is used. Immiscible WAG is applied in that situation. (Christensen, Stenby', and Skauge 2001). In highly heterogeneous reservoirs, gravity effect can divert flow from high permeable layers to low permeable layers. So, in heterogenous models low gravity effect can be basis of a scenario when low permeable reservoir layers stay upswept. Also, ordinary techniques to calculate relative permeability data is not correct to use in WAG due to its cyclic hysteresis nature. The relative permeability gas, which is non-wetting phase, is more affected by the hysteresis (Afzali, Rezaei, and Zendehboudi 2018). Hysteresis decreases gas mobility and gas-oil mobility ratio also gets lower positively affecting recovery factor. Moreover, hysteresis reduced negative effect of gravity segregation in homogeneous reservoirs. Land (1968) and Carlson (1980) models are widely used to model relative permeability hysteresis. The wettability has been defined as a parameter influencing as it impacts parameters like capillary pressure, relative permeability, dispersion, and electrical properties (resistivity and conductivity). It is the most important for planning tertiary oil recovery as surfactant flooding, miscible injections, alkaline flooding, and hot water flooding.

## 2.2 Existing WAG experience

The first WAG field experience reported in literature was in 1957 in Canada according to Arne Skauge, 2003. He wrote that preliminary portion of early projects including both MWAG and IWAG were applied in territory of Canada, USA and former USSR. Recovery factor in 72 fields reviewed by him are reported to have increased by 5 to 15 % of OIIP (Oil Originally In Place). It was reported that 80% of the USA WAG field projects are positive (Sanchez 1999). In practice it is quite hard to differentiate miscible and immiscible injection because of uncertainties in the process itself when applied on field scale, however many cases were defined as miscible, referring to multiple contact miscibility (Christensen, Stenby', and Skauge 2001).

WAG is a difficult process, which may not be practical in reducing the fluids front instabilities due to high completion costs, operational complexities. In case of alternating injection of water after gas technique, water (higher density) will sweep the bottom part of the reservoir and provides more stabilized flooding front by correcting mobility ratio. This is economically profitable as it lowers gas volume required to be injected into the reservoir in comparison to pure gas flooding method (Afzali, Rezaei, and Zendehboudi 2018).

WAG was successfully applied in many fields of Norwegian Continental Shelf (NSC) as Gullfaks, Statfjord, South Brage, Snorre and Oseberg Øst. WAG is more complicated in terms of design and operational requirements in comparison to traditional water or gas injection. WAG performance is highly sensitive to the injection strategies as injection well pattern, WAG ratio, number of WAG cycles, volume of each cycle, and injection rate and pressure.

Different aspects considered during WAG design are injection gas type, injection pattern and tapering. Gas type is mostly classified into three groups: $CO_2$, hydrocarbons (HC) and non-hydrocarbons. The most popular well pattern is "5 spots" for offshore projects, while for onshore projects the placement of wells can be more flexible. Tapering means to increase or decrease the WAG ratio as more WAG cycles are injected. but in most of the cases that was not planned, but the result of unfavorable change of cycles duration while process management.

One of the operational problems, described in literature (Christensen, Stenby', and Skauge 2001) is *early breakthrough in production wells*, which usually happens as a result of lack of understanding in terms of reservoir geology. Such that, ie. with "wrong" placement of wells, gas channelling occurs. In some cases, failure to maintain high enough pressures lead to loss of miscibility, and consequently quicker breakthrough of gas phase and lower than expected recovery factor. Early breakthrough happened in Snorre field due to uncertainties in geology (Stenmark and O. Andfossen 1995). Structural definition and degree of communication through faults and vertical transmissibilities are the most influential reservoir data for the WAG pilot carried out at Gullfaks (Dalen, Instefjord, and Kristensen 1995). Another is *reduced injectivity of injection wells,* which can happen due to three-phase flow, reduced effect of thermal fractures during gas injection or precipitates (hydrates and asphaltenes) formed in the near well zone. This becomes the reason for the fast drop in pressures in the reservoir. Furthermore, severe corrosion problems are related with $CO_2$ injection WAG. In most cases these have been solved by using high-quality steel (different kinds of stainless steels or ferritic steel), coating the pipes, and by better treatment of the equipment. Asphaltene and hydrate formation can lead to problems both during injection and production. However, the factors influencing the formation are better known for hydrates than for asphaltenes. In addition, temperature differences in water and gas injection in WAG process have resulted in stress related tubing failures at Rangely Weber and Brage fields (A. Skauge and A. Berg 1997).

Summarizing the main parameters influenced to the success or failure of field WAG trails were:

- Lack of experience.
- Uncertainties in geology or poor knowledge about reservoir properties.
- Inappropriate parameters as injection well pattern, WAG ratio, number of WAG cycles, volume of each cycle, and injection rate and pressure.
- Brine composition and salinity are important.
- Five-spot pattern is the most common strategy.
- The most common challenges in the WAG operation are early gas breakthrough, injectivity loss, corrosion, and the chance of asphaltene precipitation and hydrates formation (Christensen, Stenby', and Skauge 2001);
- The most preferred WAG ratio is 1:1 in terms of optimal oil production. However, it doesn't make much influence on WAG performance in mixed wet reservoirs;
- Accurate three phase relative permeability model is required both for miscible and immiscible gas injection processes;
- Wettability controls WAG performance. Optimal values of injection rate, WAG ratio, number of cycles, brine salinity, and polymer additive concentration will be significantly affected by the wettability (Afzali, Rezaei, and Zendehboudi 2018).

## 2.3 Applications of ML in Reservoir Engineering

Machine learning is described as a subfield of computer science that concentrates on solving two types of practical problems by collecting the dataset and algorithmically building a statistical model based on the dataset (Burkov 2019). Machine learning algorithms are categorized as either using supervised, unsupervised or reinforced learning processes. Supervised learning is the first category of ML, which finds relationship between the variables by dealing with labeled datasets. As output and input data features are known initially makes dataset "labeled". ML algorithm uses input data with "X" features and has known corresponding output value as "Y". After algorithm captures patterns in a dataset, it generates a model. Then the model is tested on a new dataset (testing dataset) to predict outputs using the same paternal laws/rules/behavior from the previous dataset (training dataset) for evaluating its predictive power and accuracy (Theobald 2017). Finally, after training and testing parts have been successfully accomplished, the model can be used for prediction in the world with unknown outputs (other datasets).  Simplified schematic illustration of forward model is presented on *Figure 2. 4.*

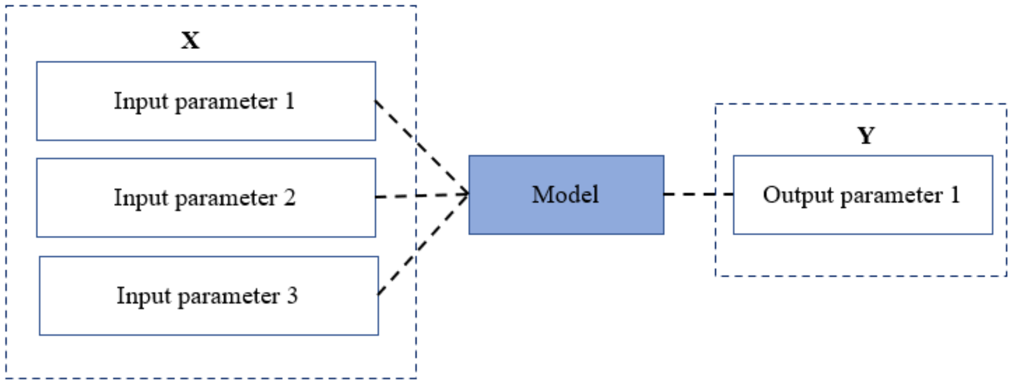


*Figure 2. 4*– Scheme of prediction model

If the data is not fully classified or labeled, unsupervised learning algorithms are implemented that will uncover patterns by itself. The most common technique is *k-means clustering*, which groups data points that have similar features, ie. as illustrated in *Figure 2. 5.*

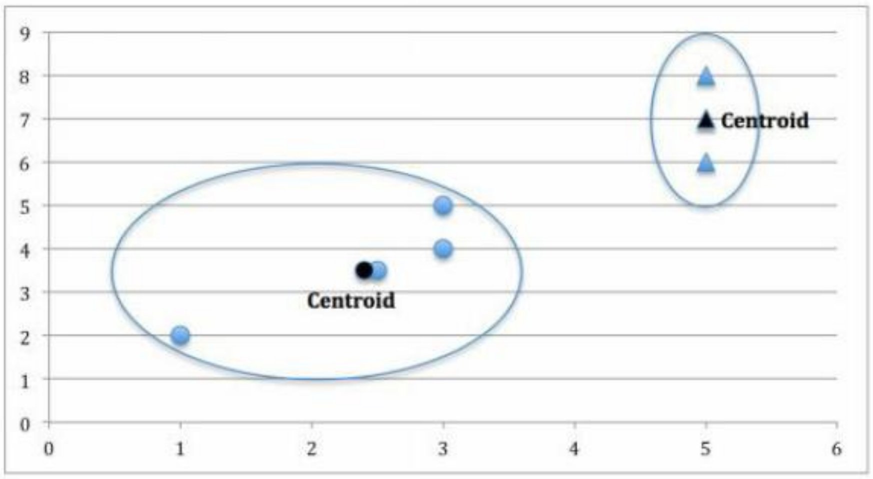


*Figure 2. 5*– k-means clustering algorithm example

Reinforcement learning is the most advanced method among the ML categories, which is due to its key feature of improving non-stop by getting information from the previous iterations. In cases of supervised or unsupervised learning types, the final model is created after training and test parts, which can be considered as an endpoint. Another feature of reinforcement learning is performance assessment set in a way that grades the output as positive or negative depending on the (desired) outcome, as opposed to tagging data as in cases of previously described ML algorithms types. The model learns continuously, so in example of self-driving cars avoiding crash will be evaluated as a positive grade and in case of chess game avoiding losing will be regarded as a positive grade.

Machine learning (ML) tools are becoming more popular in the petroleum industry, especially in geoscience (Lary et al. 2016) and reservoir engineering. The power of ML algorithms can be useful for understanding the trends in complex dataset and provide multivariate (multi-input and one output), nonlinear, nonparametric regression or classification. It can take form of a variety of algorithms as support vector machines (SVM), artificial neural networks (ANN), decision trees (DT), random forests (RF), Genetic Algorithm (GA), case-based reasoning, self-organizing map (SOM) etc.

ML based approach was used in many petroleum and reservoir engineering problems. LSSVM (Least Squares Support Vector Machines) regression with radial basis kernel (RBK) and GA (Generic Algorithms) for optimization was applied in estimation of gas hydrates formation temperature (Baghban et al. 2016).

LSSVM regression with RBK and simplex optimization was used for determining the natural gas density as function of pressure, temperature and molecular weight of gas. (Razavi et al. 2018). They collected 1240 gas density points from the literature. The results showed low error and deviation from actual data, which makes the model useful tool for engineers for estimation of gas density in pipeline and dry gas reservoir calculations.

LSSVM regression with PSO (Particle Swarm Optimization) was implemented for asphaltene precipitation prediction (Chamkalani et al. 2014). The main input parameters to the prediction model were temperature, molecular weight, and dilution rate. The study also discussed three other regression scaling models from works of Rassamdana and Sahimi (1996), Hu and Guo (2001) and Ashoori et al (2003). In comparison to all three other models, prediction model performed more accurately because of ability to fit high non-linearity in process. It was also proposed to integrate LSSVM-PSO model with black oil simulators to increase the accuracy of the prediction.

Multi-classifier LSSVM with RBK was used to capture high non-linear mapping relationship between the well logging data and the lithology categories (Cheng, Guo, and Wu 2010). Kernel parameter and slack variables were optimized using PSO algorithm. The training set consisted of 240 samples and 23 samples were used for testing.

With the rising interest in shale gas reservoirs due to technological and research improvements of last decades, ML tools also started to be widely used in unconventional (non-traditional) reserves development direction. One such work was provided by Tahmasebi, Javadpour, and Sahimi, 2017. As shale gas reservoir projects involve more wells to be drilled, the development complexities rise and in turn the economic prospects of such projects become comprised of riskier investments. As such, identifying the most favorable spots for drilling production wells (sweet spots) is of high importance. Sweet spots were considered in terms of TOC (Total Hydrocarbon Content) and FI (Fracable Index) as high TOC ranging from 2% to 10% correspond to high organic content and FI controls wells gas production capacity drilled on a shale reservoir. Two ML methods as Multiple Linear Regression (MLR) and Neural Networks integrated with fuzzy systems, resulting hybrid machine learning technique (HML) were used to predict TOC and FI of shales based on wells logs. GA was used as an optimization algorithm as it can be used when is discontinuous, non-differentiable, highly nonlinear, and even stochastic. HML technique was observed to make much more accurate predictions for the TOC and FI, when compared with those of the MLR method, However, both of proposed methods could not capture the whole complexity of shale reservoirs as they show highly non-linear behavior. HML method was designed to minimize the required knowledge as ML process can get not so optimistic with rising complexity of models and computational burden following it.

These days ways of efficiently screening approaches for IOR/EOR selection opportunities are widely discussed. ML based methods are also mentioned to be promising tools and Neural Networks, Fuzzy Logic and Expert systems are often proposed for using exploration and production operations (Alvarado et al. 2002).

Chapter 3

# Methodology

## 3.1 WAG efficiency characterization using dimensionless number

A study on WAG performance prediction was provided by Nygård and Andersen, 2020, where 1600 Black Oil Model simulations were run for a 2D model with multiple layers, an injector and a producer. The results were used to derive a dimensionless number correlating reservoir heterogeneity, WAG hysteresis, gravity, mobility ratio and WAG ratio to predict recovery factor (as measured after 1.5 injected pore volumes). Since reservoirs are involved with complicated physical behavior, the idea was to use the knowledge about these mechanisms to analyze the WAG process and related properties to ultimately develop a universal formula for mobility ratio $M^*$ for prediction of recovery factor. A more general parameter $M_{WAG}$ was used as starting point. The development of $M^*$ was done in a stepwise process whereby the model complexity would gradually increase as the model included more mechanisms. The mathematical description of the scaling process is shown in *Table 3.1.*

Some constant parameters, also known as tuning parameters were developed to account for uncertain (missing) knowledge about the processes and correlations, which is the common practice in physics. This will be the basis for further works provided in this thesis.

Table 3. 1– *Summary of mathematical description of mobility ratio based on key parameters and dependencies*

| Initial simplified Mobility ratio |
|---|
| $\boldsymbol{M_{WAG}} = \left(\frac{r_w}{M^*_{w/o}} + \frac{1-r_w}{M^*_{g/o}}\right)^{-1}$ (3.1) where, Oil/Water: $M^*_{w/o} = \frac{\lambda^*_w}{\lambda^*_{ow}} = \frac{\mu_o}{\mu_w}\frac{k^{max}_{rw}}{k^{max}_{row}}\frac{(n_{ow}+1)}{(n_w+1)}\frac{\left(1-\frac{s_{wr}}{s_{w,max}}\right)}{\left(1-\frac{s_{orw}}{s_{ow,max}}\right)}$ (3.2) Oil/Gas: $M^*_{g/o} = \frac{\lambda^*_g}{\lambda^*_{og}} = \frac{\mu_o}{\mu_g}\frac{k^{max}_{rg}}{k^{max}_{rog}}\frac{(n_{og}+1)}{(n_g+1)}\frac{\left(1-\frac{s_{gr}}{s_{g,max}}\right)}{\left(1-\frac{s_{org}}{s_{og,max}}\right)}$ (3.3) |
| Heterogeneity scaling |

$$\boldsymbol{F_H} = \frac{\overline{K}_x^{arit}}{\overline{K}_x^{harm}} \geq 1 \qquad (3.4)$$

where,

$$\overline{K}_x^{arit} = \frac{1}{L_z}\sum_{j=1}^{N_L} h_j K_{x,j} \quad (3.5), \qquad \overline{K}_x^{harm} = L_z, \left(\sum_{j=1}^{N_L} \frac{h_j}{K_{x,j}}\right)^{-1} \quad (3.6), \qquad L_z = \sum_{j=1}^{N_L} h_j \quad (3.7)$$

Gravity scaling

$$\boldsymbol{F_G^{w/o}} = \frac{1 + a_1\left(N_G^{w/o}\right)^{a_2}}{1 + a_1(F_H - 1)\left(N_G^{w/o}\right)^{a_2}} \quad (3.8), \qquad \boldsymbol{F_G^{g/o}} = \frac{1 + a_1\left(N_G^{g/o}\right)^{a_2}}{1 + a_1(F_H - 1)\left(N_G^{g/o}\right)^{a_2}} \quad (3.9)$$

where,

$$N_G^{w/o} = \frac{t_{res}^{w/o}}{t_{seg}^{w/o}} \quad (3.10), \qquad N_G^{g/o} = \frac{t_{res}^{g/o}}{t_{seg}^{g/o}} \quad (3.11)$$

$$t_{reg}^{w} = \frac{L_x L_y \sum_{j=1}^{N_L} \phi_j h_j}{Q_w} \quad (3.12), \qquad t_{reg}^{g} = \frac{L_x L_y \sum_{j=1}^{N_L} \phi_j h_j}{Q_g} \quad (3.13)$$

$$t_{seg}^{w/o} = \frac{H\phi}{K_z^{harm}\Delta\rho_{wo} g}\left(\frac{1}{\lambda_w^*} + \frac{1}{\lambda_{ow}^*}\right) \quad (3.14), \qquad t_{seg}^{g/o} = \frac{H\phi}{K_z^{harm}\Delta\rho_{go} g}\left(\frac{1}{\lambda_g^*} + \frac{1}{\lambda_{og}^*}\right) \quad (3.15)$$

Hysteresis scaling

$$\boldsymbol{s_{gr}^{wag}} = s_{gr}(1 - r_w) + r_w s_{gr}^{hyst} \quad (1.18)$$

$$\boldsymbol{k_{rg,M}^{wag}} = \left(\frac{1 - r_w}{k_{rg}^{max}} + \frac{r_w}{k_{rg,M}^{max,hyst}}\right)^{-1} \quad (3.16), \qquad \boldsymbol{k_{rg,N_G}^{wag}} = \left(\frac{1 - r_w}{k_{rg}^{max}} + \frac{r_w}{k_{rg,N}^{max,hyst}}\right)^{-1} \quad (3.17)$$

where,

$$s_{gr}^{hyst} = s_{gr} + \frac{s_{g,max} - s_{gr}}{1 + C\left(s_{g,max} - s_{gr}\right)} \quad (3.18)$$

$$k_{rg,M}^{max,hyst} = \frac{k_{rg}^{max}}{1 + b_1 F_H^{b_2}\alpha} \quad (3.19), \qquad k_{rg,N_G}^{max,hyst} = \frac{k_{rg}^{max}}{1 + b_3 F_H^{b_4}\alpha} \quad (3.20)$$

Mobility terms:

$$\lambda_{g,M}^* = \frac{1}{\mu_g}\left(1 - \frac{s_{gr}^{wag}}{s_{g,max}} + \right)\frac{k_{rg,M}^{max}}{n_g + 1} \quad (3.21), \qquad \lambda_{g,N_G}^* = \frac{1}{\mu_g}\left(1 - \frac{s_{gr}^{wag}}{s_{g,max}} + \right)\frac{k_{rg,N_G}^{max}}{n_g + 1} \quad (3.22)$$

$\lambda_{g,M}^*$ and $\lambda_{g,N_G}^*$ replace $\lambda_g^*$ in $M_{g/o}^*$ for (3.3) and $t_{seg}^{g/o}$ for (3.15), respectively. $s_{gr}^{wag}$ replaces $s_{gr}$ and $k_{rg,M}^{max,hyst}$ or $k_{rg,N_G}^{max,hyst}$replaces $k_{rg}^{max}$in $M_{g/o}^*$, respectively (3.3).

Scaled Mobility ratio M*

$$M^* = \left(\frac{r_w}{M_{w/o}^* F_H F_G^{w/o}} + \frac{1 - r_w}{M_{g/o}^* F_H F_G^{g/o}}\right)^{-1} \quad (3.23)$$

*Table 3. 2* – The tuning parameters that were determined from scaled simulation results

| | | | | | |
|---|---|---|---|---|---|
| $a_1$ , - | 3 | $b_1$ , - | 1 | $b_3$ , - | 10 |
| $a_2$ , - | 0.5 | $b_2$ , - | 0.5 | $b_4$ , - | 2 |

Nygård and Andersen, 2020 collected all of their simulation results as in *Figure 3. 1*, were Recovery Factor (RF) was plotted against $M_{WAG}$ (left) and $M^*$ (right). Comparing them, we see that by only using $M_{WAG}$ there is high variation of RF for a given value of of $M_{WAG}$ (a range of 0.40 for $M_{WAG} \approx 1$ and a range of 0.20 for $M_{WAG} \approx 100$). This is because it does not account for heterogeneity, gravity or hysteresis, whereas dimensionless scaled $M^*$effects accounts for these effects, which is why we can observe that the data is much more collected, with RF scatter ranges between 0.15 and 0.25. The values of M cover three orders of magnitude (original values of $M_{WAG}$ cover two) indicating the shift along the axis to compensate for the stated effects.

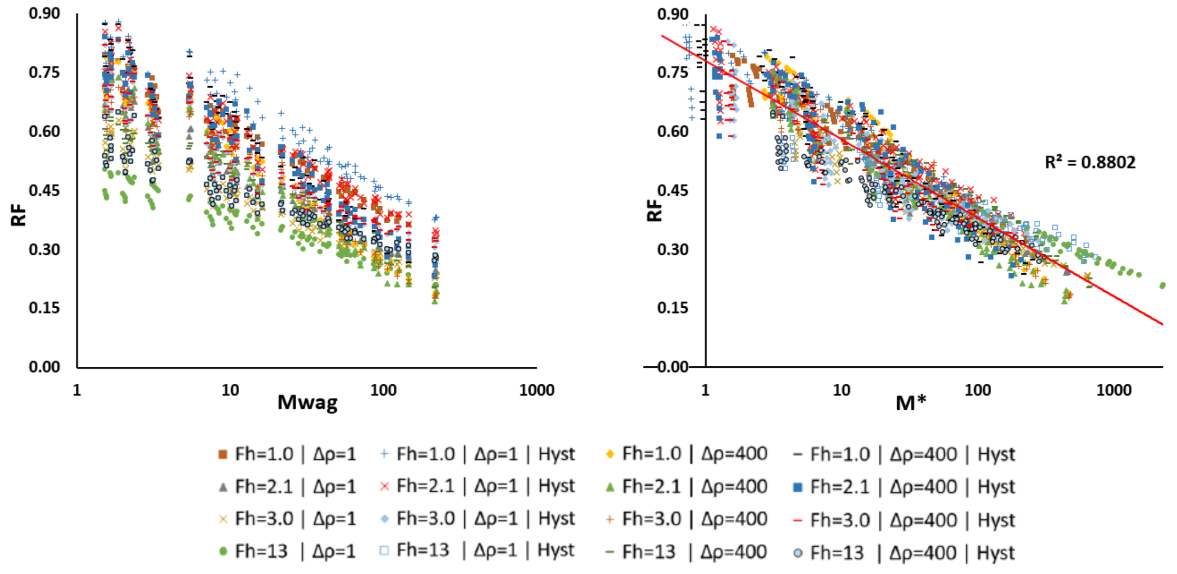


*Figure 3. 1*– Overview of all simulation results plotted vs $M_{WAG}$ (left) and $M^*$ (right) (Nygård and Andersen 2020)

$M^*$ was more effective in correlating the RF trends (*Figure 3. 1*), we can still observe significant data scatter. Therefore, one of the objectives of the current work is to reduce this scatter further by applying a model.

## 3.2 Work principle of Machine Learning and Optimization algorithms to be applied on the problem

### 3.2.1. Least Squares Support Vector Machines (regression)

Support Vector Machines (SVM) is a supervised ML algorithm that has been widely used in classification and nonlinear function estimation. However, the major disadvantage of SVM is its higher computational load for the constrained optimization programming. This drawback has been lowered with the Least Squares Support Vector Machine (LSSVM), which solves linear equations instead of a quadratic programming problem.

Support vector machine (SVM) was developed by Vapnik, 1995 and was originally used to solve classification problems by building hyperplanes in multidimensional spaces that separated data which belong to different class labels. After proving itself useful the area of its application was extended to cover regression problems. The solution of the SVM is unique and absent from local minimums under some limited conditions. The algorithm maps the input vector, x, into a high dimensional feature space, z, by building optimal separating hyperplanes in this higher dimensional space. The mathematical description by is provided below.

The support vector machine (SVM) is generally known as a strong mathematical approach to create accurate and comprehensive correlation between the variables (or parameters) of a certain mathematical problem.
A modified version of SVM named least squares-SVM(LS-SVM) was introduced by Suykens and Vandewalle (1999). Like SVM, LS-SVM has a variety of applications in both regression and classification cases. LSSVM normally lowers the run time and exhibits more adaptivity.

Key differences with between SVM and LSSVM:

- ε - insensitive cost replaced by quadratic error cost.
- Inequality constraint replaced by equality constraint.

The given finite sample data is represented as an array of $D = \{(x_1, y_1), \dots \dots, (x^n, y^n)\}, x_i \in R^n, y_i \in R$ (Hou, Yang, and An, 2009).
In LSSVM, the regression is expressed as a feature space representation (Suykens 2002):

$$y_i = w^T \varphi(x_i) + b \; where \; x_i \in R^p \; and \; y_i \in R \qquad (3.1)$$

For a given training set $\{x_i \, , y_i\}_{i=1}^{N}$ the optimization problem is described as:

$$\min_{w,e} J(\omega, e) = \frac{1}{2} w^T w + \gamma \frac{1}{2} \sum_{i=1}^{N} e_i^2 \qquad (3.2)$$

$$y_k = w^T \varphi(x_i) + b + e_i, \qquad i = 1, \dots., N$$

$$L(w, b, e; \alpha) = J(w, e) - \sum_{k=1}^{N} \alpha_i \{w^T \varphi(x_i) + b + e_i - y_i\} \qquad (3.3)$$

with Lagrange multipliers $\alpha_i$.

Conditions for optimality:

$$\begin{cases} \dfrac{dL}{dw} = 0 \rightarrow w = \displaystyle\sum_{k=1}^{N} \alpha_i \varphi(x_i) \\ \dfrac{dL}{db} = 0 \rightarrow \displaystyle\sum_{k=1}^{N} \alpha_i = 0 \\ \dfrac{dL}{de_i} = 0 \rightarrow \alpha_i = \gamma e_i, \qquad i = 1, \dots, N \\ \dfrac{dL}{d\alpha_i} = 0 \rightarrow = \; w^T \varphi(x_i) + b + e_i - y_i = 0, i = 1, \dots, N \end{cases} \qquad (3.3)$$

Solution is

$$\left[\begin{array}{c|c} 0 & \vec{1}^T \\ \vec{1} & \Omega + \gamma^{-1} I \end{array}\right] \left[\begin{array}{c} b \\ \hline \alpha \end{array}\right] = \left[\begin{array}{c} 0 \\ \hline y \end{array}\right] \qquad (3.3)$$

with

$$y = [y_1; \dots; y_N], \vec{1} = [1; \dots; 1], \alpha = [\alpha_1; \dots; \alpha_N]$$

and by applying Mercer's condition:

$$\Omega_{kj} = \varphi(x_i)^T \varphi(x_j) = K(x_i, x_j) \ , \ \ i, j = 1, \dots, N \qquad (3.4)$$

Resulting LS-SVM model for function estimation:

$$y(x) = \sum_{k=1}^{N} \alpha_k K(x_i, x) + b \qquad (3.5)$$

The final form of LSSVM is given by

$$\begin{bmatrix} 0 & 1 & \cdots & 1 \\ 1 & K(x_1, x_1) + \frac{1}{\gamma} & \dots & K(x_1, x_N) \\ \vdots & \vdots & \ddots & \vdots \\ 1 & K(x_N, x_1) & \cdots & K(x_N, x_N) + \frac{1}{\gamma} \end{bmatrix}_{(N+1)X(N+1)} \begin{bmatrix} b \\ \alpha_1 \\ \vdots \\ \alpha_N \end{bmatrix}_{(N+1)X1} = \begin{bmatrix} 0 \\ y_1 \\ \vdots \\ y_N \end{bmatrix}_{(N+1)X1} \qquad (3.6)$$

$K(x_1, x_1)$ is a kernel function. Radial Basis Kernel function was selected for the problem in this thesis:

$$K(x_i, x_j) = \exp\left( - \frac{\|x_i - x_,\|^2}{\sigma^2} \right) \qquad (3.7)$$

**Particle swarm optimization (PSO)**

The LSSVM regularization parameter, $\gamma$, and kernel parameter, $\sigma^2$ can be determined through optimization technique such as Generic Algorithm (GA), Particle Swarm Optimization (PSO), and Simulated Annealing (SA) by minimizing the objective function. In this study, Root Mean Square Error (RMSE) between simulated "*real*" values and model-predicted values from LSSVM is considered as objective function with PSO routine.

The Particle Swarm optimization is a meta-heuristic algorithm, which was inspired by the social behavior of birds. It is an example of swarm intelligence, that can be used for optimizing the LSSVM algorithm. The basic principle of PSO's work is described according to Bozorg-Haddad, Solgi, and Loáiciga (2017).

**Creating the population of particles.** In an *N*-dimensional optimization problem a particle is specified as an array of size 1 × *N*, which is each possible solution of the optimization problem as a particle:

$$Particle = X = (x_1, x_2, \dots, x_i, \dots, x_N) \qquad (3.8)$$

where $X = a$ possible solution of the optimization problem, $x_i = i$th decision variable of solution $X$, and $N =$ number of decision variables. The PSO algorithm starts by generating a matrix of particles

$$Swarm = \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_j \\ \vdots \\ X_M \end{bmatrix} = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,i} & \cdots & x_{1,N} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,i} & \cdots & x_{2,N} \\ & & & \vdots & & \\ x_{j,1} & x_{j,2} & \cdots & x_{j,i} & \cdots & x_{j,N} \\ & & & \vdots & & \vdots \\ x_{M,1} & x_{M,2} & \cdots & x_{M,i} & & x_{M,N} \end{bmatrix} \quad (3.9)$$

where $X_j$ = $i$th solution $x_{j,i}$ = $i$th decision variables of the $j$th solution, and $M$ = population size. Each particle moves through the decision space based on the individual best ($Pbest$) and global best ($Gbest$) positions:

$$Pbest = (p_{j,1}, p_{j,2}, \dots, p_{j,i}, \dots, p_{j,N}), \quad j = 1,2,\dots,M \quad (3.10)$$

where $Pbest_j$ = the best position of the $j$th particle and $p_{j,i}$= the best position of the $j$th particle in the $i$th dimension. The graphical illustration of the concept is shown in Figure 3. 2.

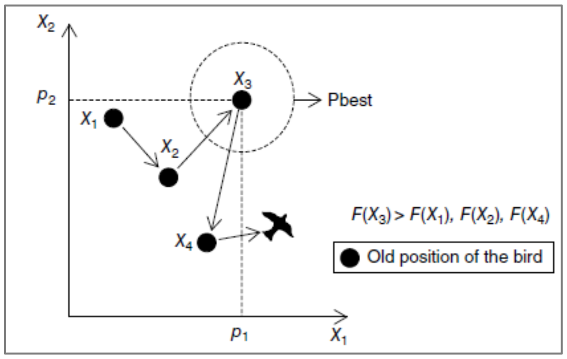


*Figure 3. 2* – The best individual position in a two-dimensional maximization problem (Bozorg-Haddad, Solgi, and Loáiciga (2017)

$Gbest$ is an array 1× $N$ whose elements define the best position achieved in the swarm:

$$Gbest = (g_1, g_2, \dots, g_i, \dots, g_N), \qquad all\ j \qquad (3.11)$$

where $Gbest$ = the best position in the swarm's history and $g_1$ = the best position in the swarm's history in the $i$th dimension. The graphical illustration of the concept is shown in *Figure 3. 3.*

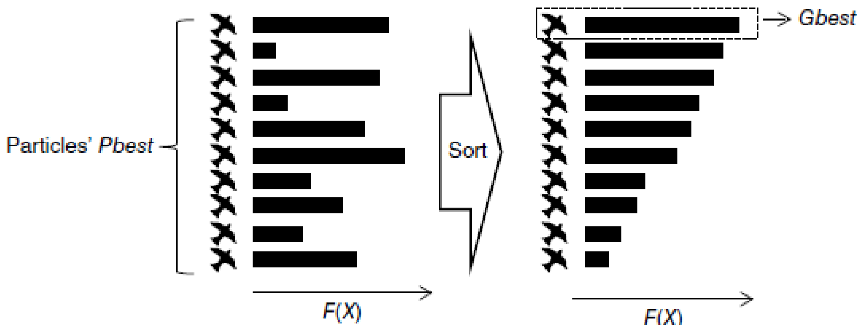


*Figure 3. 3*– The global best position in a maximization problem (Bozorg-Haddad, Solgi, and Loáiciga (2017)

**Calculation of velocities.** To update the position of each particle the particles' velocities are used, which are calculated based on $Gbest$ and $Pbest$.

The previous velocity of the $j$th particle ($V_j$) is:

$$V_j = \left(v_{j,1}, v_{j,2}, \dots, v_{j,i}, \dots, v_{j,N}\right), \qquad j = 1,2,\dots,M \qquad (3.12)$$

where $v_{j,i}$ = the velocity of the $j$th particle in the $i$th dimension that is calculated as follows:

$$v_{j,i}^{(new)} = w \times v_{j,i} + C_1 \times Rand \times \left(p_{j,i} - x_{j,i}\right) + C_2 \times Rand \times \left(g_i - x_{j,i}\right) \quad (3.13)$$

$$j = 1,2,..,M, \quad i = 1,2,\dots,N$$

where
$v_{j,i}^{(new)}$ = the new velocity of the $j$th particle in the $i$th dimension;
$v_{j,i}$ = the previous velocity of the $j$th particle in the $i$th dimension;
$w$ = inertia weight parameter;
$Rand$ = a random value in the range [0,1];
$C_1$ = cognitive parameter, and $C_2$ =social parameter ($C_1$ and $C_2$ control the movement of $Pbest$ and $Gbest$ toward an optimal point. $C_1 = C_2 = 2$ can be used.. Movement along different directions towards $Gbest$ and $Pbest$ is possible if $C_1$ and $C_2$ are larger than one.

The particle's velocity is limited by lower and upper bounds in the following manner:

$$V_i \le v_{j,i}^{(new)} \le V_i^{(u)}, \quad j = 1,2,\dots,M, \quad i = 1,2,\dots,N \quad (3.14)$$

where $V_i^{(L)}$ and $V_i^{(u)}$corresponds to the lower and upper bound of the velocity along the $i$th dimension, respectively.

The inertia weight parameter may change as the algorithm progresses as follows:

$$w_o = w_0 - \left[(w_0 - w_T) \times \frac{t}{T}\right], \quad t = 1,2,\dots,T \qquad (3.15)$$

where $w_o$ refers to initial inertia weight, $w_T$ is inertia weight for the last iteration, and $T$ is total number of iterations. The values of $w$ changes through the iterations.
The inertia weight parameter w has an important role in swarm convergence and affects the velocity of individual particles in the swarm. Large or small values of $w$ cause searching in a wide or narrow space, respectively (Bozorg-Haddad, Solgi, and Loáiciga (2017).

**Updating of particles positions.**

$$X_j^{(new)} = \left(x'_{j,1}, x'_{j,2}, \dots, x'_{j,i}, \dots, x'_{j,N}\right), \qquad j = 1,2,\dots,M \quad (3.16)$$

$$x'_{j,i} = x_{j,i} + v_{j,i}^{(new)}, \quad j = 1,2,\dots,M, \quad i = 1,2,,\dots,N \quad (3.17)$$

where $X_j^{(new)}$ = $j$th new solution and $x'_{j,i}$ = new value of $i$th decision variable of the jth new solution. The $M$ newly generated solutions replace all the old solutions.

Goodness of fit can be determined by one of these methods:

**1.** The coefficient of determination ($R^2$):

$$R^2 = 1 - \frac{\sum_{i=1}^{n}\left(y_{obs,i} - y_{model,i}\right)^2}{\sum_{i=1}^{n}\left(\left(\frac{1}{n}\sum_{i=1}^{n} y_{obs,i}\right) - y_{model,i}\right)^2} \qquad (3.18)$$

The values of $R^2$ range from 0 to 1, corresponding to the worst and the best fit.

**2.** Root Mean Square Error (RMSE), which is square root of Mean Squared Error (MSE):

$$MSE = \frac{\sum_{i=1}^{n}\left(y_{obs,i} - y_{model,i}\right)^2}{n} \qquad (3.19)$$

$$RMSE = \sqrt{MSE} \qquad (3.60)$$

$y_{exp}$ and $y_{model}$ are the experimental and modeled values, respectively.

## 3.3 Workflow

The proposed workflow for building LSSVM-PSO model is presented in *Figure 3. 4.* Initially data analysis was provided to build the dataset which was used for creating LSSVM-PSO model. Original dataset from previous work by was expanded (see more in Chapter 0).

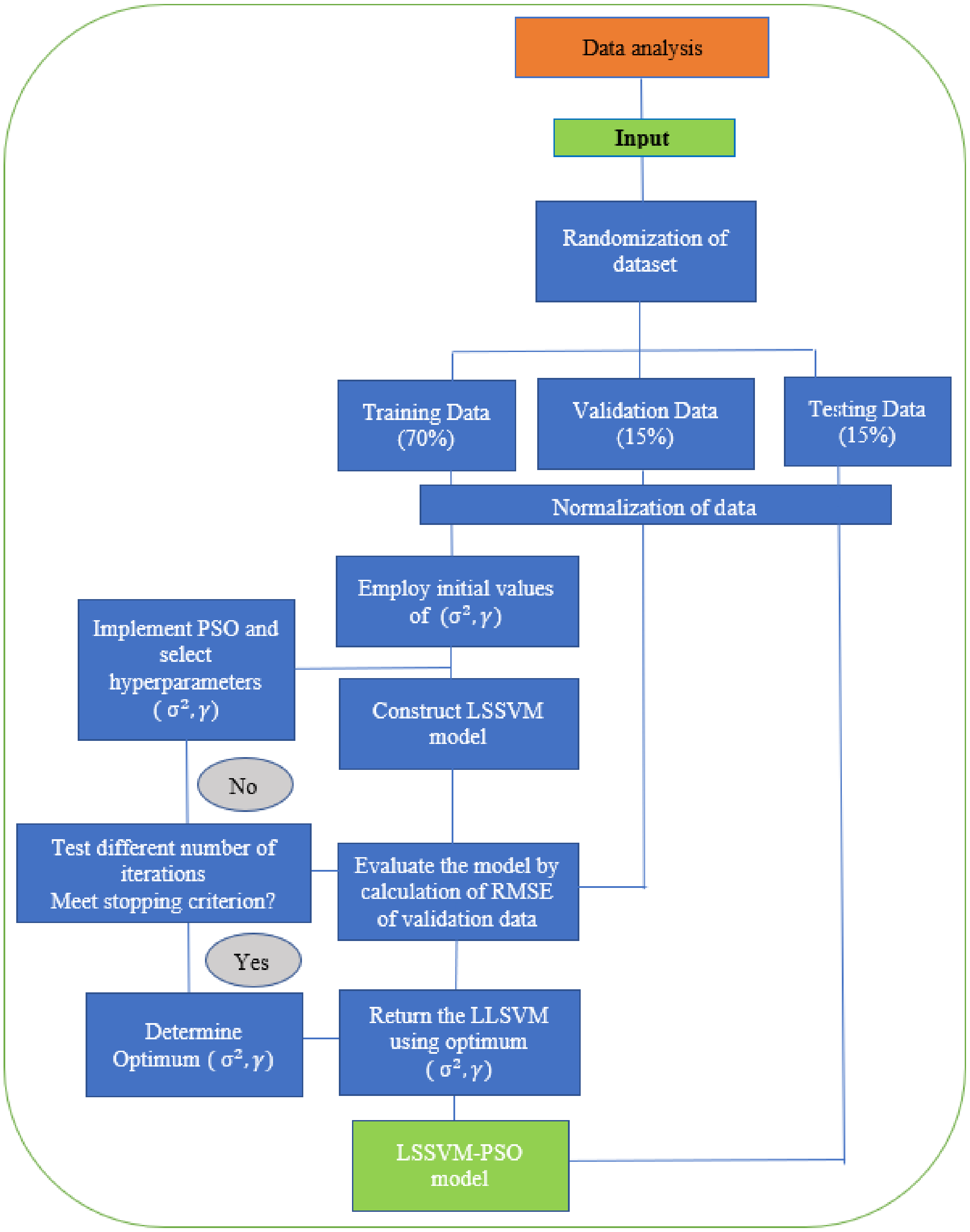

Data analysis
Input
Randomization of dataset
Training Data (70%)
Validation Data (15%)
Testing Data (15%)
Normalization of data
Employ initial values of $(\sigma^2, \gamma)$
Implement PSO and select hyperparameters $(\sigma^2, \gamma)$
Construct LSSVM model
No
Test different number of iterations Meet stopping criterion?
Evaluate the model by calculation of RMSE of validation data
Yes
Determine Optimum $(\sigma^2, \gamma)$
Return the LLSVM using optimum $(\sigma^2, \gamma)$
LSSVM-PSO model

*Figure 3. 4*– LSSVM-PSO model building workflow

Afterwards the randomized expanded dataset was divided into training, validation and testing parts by fractions of 70, 15 and 15 %, respectively. Normalization of data was applied to keep input data compatible to calculated RMSE values for each part of the dataset. Random initial values were selected for hyperparameters $(\sigma, \gamma)$ values. The training dataset was utilized to train the LSSVM model by capturing patterns and rule in it. PSO was applied on LSSVM algorithm for optimizing hyperparameter values by minimizing Root Mean Square Error (RMSE) between real validation data output and predicted output by LSSVM. Instead of using RMSE value as stopping criteria the optimization algorithm was tested with different numbers of iterations. That was a decision made based on lack of knowledge about value of RMSE for validation dataset when optimization process can be stopped. The optimal number of iterations was defined to be 50 with 10 particles based on stabilization of hyperparameters and RMSE curves. $\sigma, \gamma$ values were selected by analyzing RMSE curves behavior when different hyperparameters are were implemented. Testing data is used to observe the model for predictive capability.

Then LSSVM is returned using the optimum values for hyperparameters. LSSVM-PSO model is in the output.

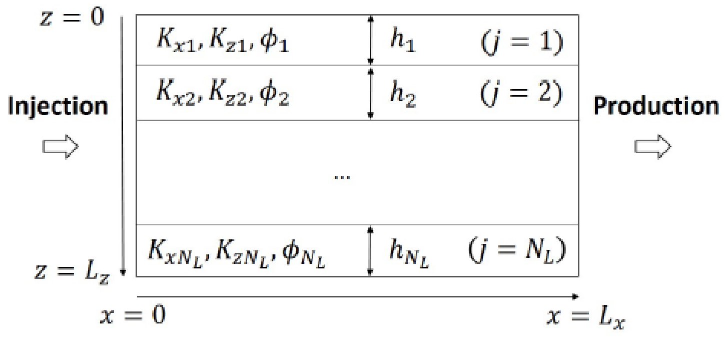
Chapter 4

# Creating the dataset and building LSSVM-PSO model

## 4.1 Identification of the dataset

The reservoir model (system) that is used for simulations is stratified and contains layers that align with the horizontal x-axis, which goes from injector to producer well. This is illustrated in *Figure 4. 1*, where layering is distinguished by differences in horizontal permeability values. The specific layered setup can be read from *Table 4. 3*. The layers are internally homogeneous (uniform height, porosity, and permeability). The z-axis points downwards along the direction of gravity, normal to the injector-producer path. Both wells are perforated along the entire reservoir section. The applied grid properties and operational parameters are provided in *Table 4. 1*, while fluid flow properties and input relative permeability functions can be seen from

*Table 4.* 2 and *Figure 4. 2*, respectively. The properties described herein remain the same throughout the study.



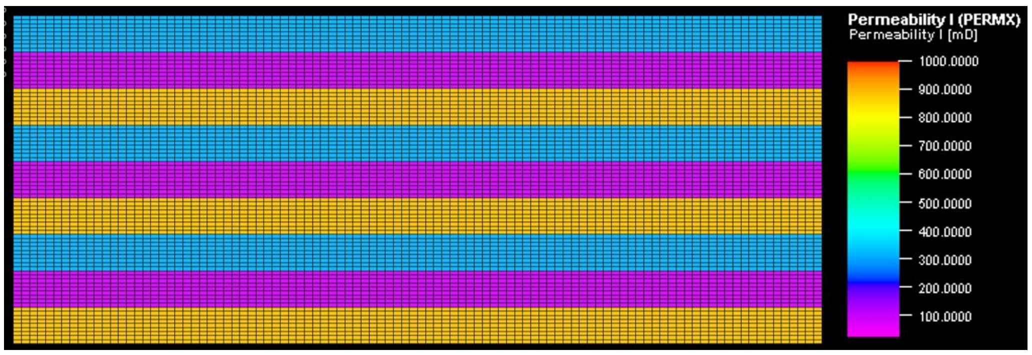


*Figure 4. 1*– System geometry scheme (upper) and illustration of permeability in reservoir model (lower)

*Table 4. 1*– Rock/grid properties and operational parameters

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $N_x$, | - | 100 | $L_x$, | m | 1000 | $\varphi_j$,- | 0.30 | $Q_w$, m$^3$/d | 1014.6 | $T^{w-hc}$, d | 45 |
| $N_y$, | - | 1 | $L_y$, | m | 100 | $h_j$, m | 3 | $Q_g$, m$^3$/d | 1014.6 | $T^{g-hc}$, d | 45 |
| $N_z$, | - | 81 | $L_z$, | m | 81 | $N_L$,- | 9 | | | $T^{tot}$, PVs | 1.5 |

*Table 4. 2*– Reservoir flow properties

| | | | |
|---|---|---|---|
| $k_{row}^{max} = k_{row}(S_{wi})$, - | 0.25 | $n_{ow}$ , - | 2 |
| $k_{rog}^{max} = k_{rog}(S_{gi})$, - | 0.25 | $n_{og}$ , - | 2 |
| $k_{rw}^{max} = k_{rw}(1 - S_{orw})$, - | 0.05 | $n_w$ , - | 2 |
| $k_{rg}^{max} = k_{rg}(1 - S_{org})$, - | 0.005 | $n_g$ , - | 2 |
| $S_{oi}$, - | 0.842 | $S_{orw}$, - | 0.20 |
| $S_{wi} = S_{wr}$ ,- | 0.158 | $S_{org}$, - | 0.10 |
| $S_{gi} = S_{gr}$, - | 0.00 | | |

*Table 4. 3*– Specification of model heterogeneities. Patterns are indicated from to (j = 1) layer.

| Model | $K_{x,j}$ $[mD]$ | $K_{z,j\ i}$ $[mD]$ | $F_H$ |
|---|---|---|---|
| 1 (base) | [300] x9 | [300] x9 | 1.0 |
| 2 | [300, 100, 900] x3 | [300, 100, 900] x3 | 2.1 |
| 3 | [500, 50] x4, [500] | [500, 50] x4, [500] | 3.0 |
| 4 | [1000, 20] x4, [1000] | [1000, 20] x4, [1000] | 12.9 |

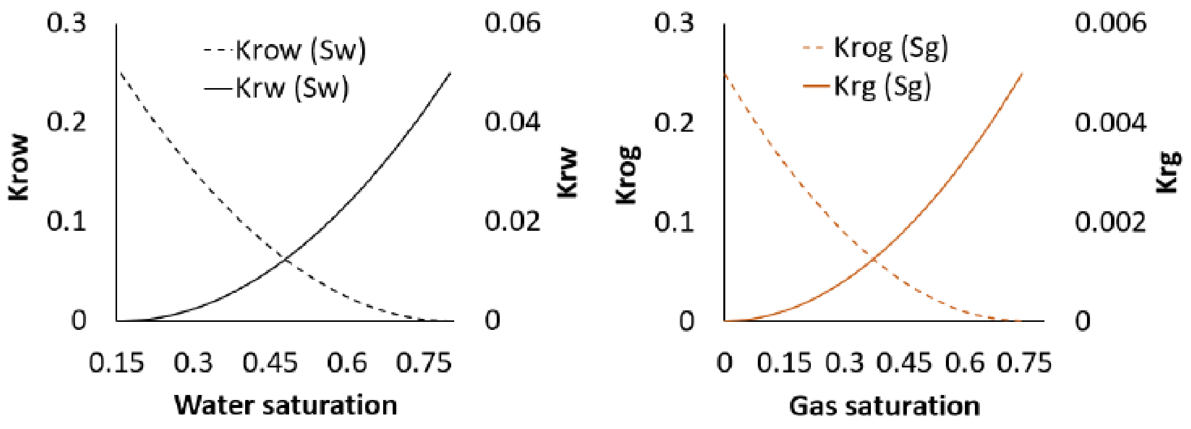


*Figure 4. 2*– Input oil-water (left) and gas-oil (right) relative permeability functions

The main assumptions made regarding input parameters are:

- Compressibility and miscibility effects are ignored in the study
- WAG is applied as a non-tertiary method, without any prior injection stage.

The data from 1648 WAG simulations of previous work (Nygård and Andersen 2020) was used, together with 192 single-phase injection simulations, as an initial dataset. Both in previous and current work, Eclipse was used for these simulation runs. The parameter combinations that was used to produce these results (total 1840 simulations) are shown in

*Table 4.* 4, which cover changes to heterogeneity, gravity and hysteresis effects.

*Table 4. 4*– Overview table of simulation experiments

| Purpose | Types | Changing parameters values (in different combinations) | | | | | | | Number of runs |
|---|---|---|---|---|---|---|---|---|---|
| | | $F_H$ | **$\Delta\rho$, kg/m$^3$** | $r_w$ | $C$ | $\alpha$ | $\mu_w$ | $\mu_g$ | |
| **Scaling heterogeneity** | | 1.00 | 1 | 0.33 | 1000000 | 0 | 20 | 4 | *460* |
| | | 2.09 | | 0.50 | | | 4 | 1 | |
| | | 3.00 | | 0.67 | | | 1 | 0.5 | |
| | | 12.86 | | 0 | | | 0.5 | 0.25 | |
| | | | | 1 | | | | | |
| **Scaling gravity** | *no heterogeneity* | 1 | 400 | 0.33 | 1000000 | 0 | 20 | 4 | *115* |
| | | | | 0.50 | | | 4 | 1 | |
| | | | | 0.67 | | | 1 | 0.5 | |
| | | | | 0 | | | 0.5 | 0.25 | |
| | | | | 1 | | | | | |
| **Scaling gravity** | *with heterogeneity* | 2.09 | 400 | 0.33 | 100000 | | 20 | 4 | *345* |
| | | 3.00 | | 0.50 | | | 4 | 1 | |
| | | 12.86 | | 0.67 | | | 1 | 0.5 | |
| | | | | | | | 0.5 | 0.25 | |
| **Scaling WAG hysteresis** | *No gravity+ heterogeneity* | 1 | 1 | 0.33 | 1 | 0 | 20 | 4 | *460* |
| | | 2.09 | | 0.50 | | 2.5 | 4 | 1 | |
| | | 3.00 | | 0.67 | | | 1 | 0.5 | |
| | | 12.86 | | 0 | | | 0.5 | 0.25 | |
| | | | | 1 | | | | | |
| | *with gravity +heterogeneity* | 1 | 400 | 0.33 | 1 | 0 | 20 | 4 | *460* |
| | | 2.09 | | 0.50 | | 2.5 | 4 | 1 | |
| | | 3.00 | | 0.67 | | | 1 | 0.5 | |
| | | 12.86 | | 0 | | | 0.5 | 0.25 | |
| | | | | 1 | | | | | |
| *Total* | | | | | | | | | *1840* |

## Approach used for selection of input parameters for LSSVM-PSO model

The main idea for selecting input parameters was to consider all mechanisms that was observed to have significant influence on recovery factor of WAG process. It was done in two phases:

1. Formation of dataset based on simulations provided in previous study and main processes used for scaling (Nygård and Andersen 2020);
2. Extending the dataset after dataset analysis by running additional simulations, to cover parameter combinations that in previous work was not fully explored

For having the dataset sufficiently representing the key processes in the WAG process and limiting number input parameters relating to the same output, effect of some process parameters must be grouped into dimensionless numbers. That will allow the LSSVM model input parameters to be not dependent on unit systems and efficiently represent maximum information with minimum space and computational power consumed. The parameters for the initial dataset with justification of their purpose is presented in Table 4. 5.

*Table 4. 5*– Overview table of simulation experiments

| # | Parameter | Symbol | Purpose | Type | Scaling formula |
|---|---|---|---|---|---|
| 1 | Heterogeneity factor | Log$_{10}$Fh | Describes effect of changing heterogeneity | Scaled dimensionless | $\boldsymbol{logF_H} = log \frac{\bar{K}_x^{arit}}{\bar{K}_x^{harm}}$ (4.1)<br>$\bar{K}_x^{arit} = \frac{1}{L_z}\sum_{j=1}^{N_L} h_j K_{x,j}$ (4.2)<br>$\bar{K}_x^{harm} = L_z, \left(\sum_{j=1}^{N_L} \frac{h_j}{K_{x,j}}\right)^{-1}$ (4.3), $L_z = \sum_{j=1}^{N_L} h_j$ (4.4) |
| 2 | Hysteresis parameter | $\alpha$ | Contribution of hysteresis | Single dimensionless | Used directly |
| 3 | Land trapping parameter | Log$_{10}$C | | | Used directly |
| 4 | WAG ratio | rw | Input of altering water or gas fraction. | | Used directly |
| 5 | Mobility ratio of water and gas | Log$_{10}$(Mw/o) | Mobility of oil, gas and water | Scaled dimensionless | $\boldsymbol{M^*_{w/o}} = \frac{\lambda^*_w}{\lambda^*_{ow}} = \frac{\mu_o}{\mu_w}\frac{k_{rw}^{max}}{k_{row}^{max}}\frac{(n_{ow}+1)}{(n_w+1)}\frac{\left(1-\frac{S_{wr}}{S_{w,max}}\right)}{\left(1-\frac{S_{orw}}{S_{ow,max}}\right)}$ (4.5) |
| 6 | Mobility ratio of gas and oil | Log$_{10}$(Mg/o) | | | $\boldsymbol{M^*_{g/o}} = \frac{\lambda^*_g}{\lambda^*_{og}} = \frac{\mu_o}{\mu_g}\frac{k_{rg}^{max}}{k_{rog}^{max}}\frac{(n_{og}+1)}{(n_g+1)}\frac{\left(1-\frac{S_{gr}}{S_{g,max}}\right)}{\left(1-\frac{S_{org}}{S_{og,max}}\right)}$ (4.6) |
| 7 | Gravity number of water/gas | Log$_{10}$(Ng.w/o) | | | $\boldsymbol{N_G^{w/o}} = \frac{t_{res}^{w/o}}{t_{seg}^{w/o}}$ (4.7) $t_{reg}^{w} = \frac{L_x L_y \sum_{j=1}^{N_L}\phi_j h_j}{Q_w}$ (4.8)<br>$t_{seg}^{w/o} = \frac{H\phi}{K_z^{harm}\Delta\rho_{wo} g}\left(\frac{1}{\lambda^*_w}+\frac{1}{\lambda^*_{ow}}\right)$ (4.9) |
| 8 | Gravity number of gas/oil | Log$_{10}$(Ng.g/o) | | | $\boldsymbol{N_G^{g/o}} = \frac{t_{res}^{g/o}}{t_{seg}^{g/o}}$ (4.10) $t_{reg}^{g} = \frac{L_x L_y \sum_{j=1}^{N_L}\phi_j h_j}{Q_g}$ (4.11)<br>$t_{seg}^{g/o} = \frac{H\phi}{K_z^{harm}\Delta\rho_{go} g}\left(\frac{1}{\lambda^*_g}+\frac{1}{\lambda^*_{og}}\right)$ (4.12) |

Logarithm was used for some of the input parameters presented in *Table 4. 5* to keep compatibility with other parameters values.

To understand the distribution of selected input parameters and to estimate its quality, paired scatter plots of initial input parameters was created *(Figure 4. 3)*. They show the relation of parameters between each other in all combinations. The plotted parameters are given by the top column box and the right-most row box, which correspond to the plotted property on the y- and x-axis, respectively.

For most parameters there is a uniform distribution in their value ranges, especially for combinations concerning M and Ng parameters, as seen in *Figure 4. 3*. In the previous study, hysteresis parameters C and $\alpha$ were selected primarily to cover two edge cases, namely "no hysteresis" ($\alpha$ =2.5, C=0) and "strong hysteresis" ($\alpha$ =0, C=100000) scenarios. This is why we see parameter combinations for C and $\alpha$ not giving as good distribution as for other parameters. We could use this opportunity to extend the dataset with some intermediate values for C and alpha, that are within the previously used edge case values.

When preparing the input matrix for LSSVM, there was a challenge in defining appropriate values for mobility and gravity numbers for the cases of single-phase injection of gas or water. This was due to that at rw=1 values for Ng.g/o and Mg/o did not exist, similarly for Ng.w/o and Mw/o at rw=0. This would create "gaps" in the dataset (the matrix used in ie. Eq. 3.6) and require the LSSVM to handle those scenarios in some way, which might have made it biased to our choices. Instead, to preserve LSSVM functionality as-is, "fake" values were introduced while keeping the RF value identical to that of the non-fake values, at the rw=1 and rw=0 endpoints. Basically, it meant that a scatter of "fake" values was used together with the same RF value, to try to make the LSSVM disregard somewhat the existence of ie. Ng.g/o and Mg/o at rw=1. This is explained in detail below.

For each datapoint where Ng.g/o and Mg/o (or Ng.w/o and Mw/o) did not exist, four datapoints were used. These were created in the following manner, where i=w or i=g:

- Log(Mi/o_ref)+1, Log(Ng.i/o_ref)+1
- Log(Mi/o_ref) -1, Log(Ng.i/o_ref)+1
- Log(Mi/o_ref)+1, Log(Ng.i/o_ref)-1
- Log(Mi/o_ref)-1, Log(Ng.i/o_ref)-1

Where phase-specific reference values from that of the WAG cases were used since both gas and water exist there. Letting the rw approach but never become rw=1, we could select the correct reference values for the gas phase. Similarly, for the water phase. While the actual single-phase injection simulations ran at exactly rw=1 or rw=0, this allowed some reference value selection to complete the 8-input dataset where data for both WAG, gas and waterflooding exists.

Ultimately, this meant that the dataset was extended with 576 "fake" datapoints created based on 192 non-fake datapoints for rw=1 and rw=0, and hence 768 datapoints in total represent water and gas flooding.

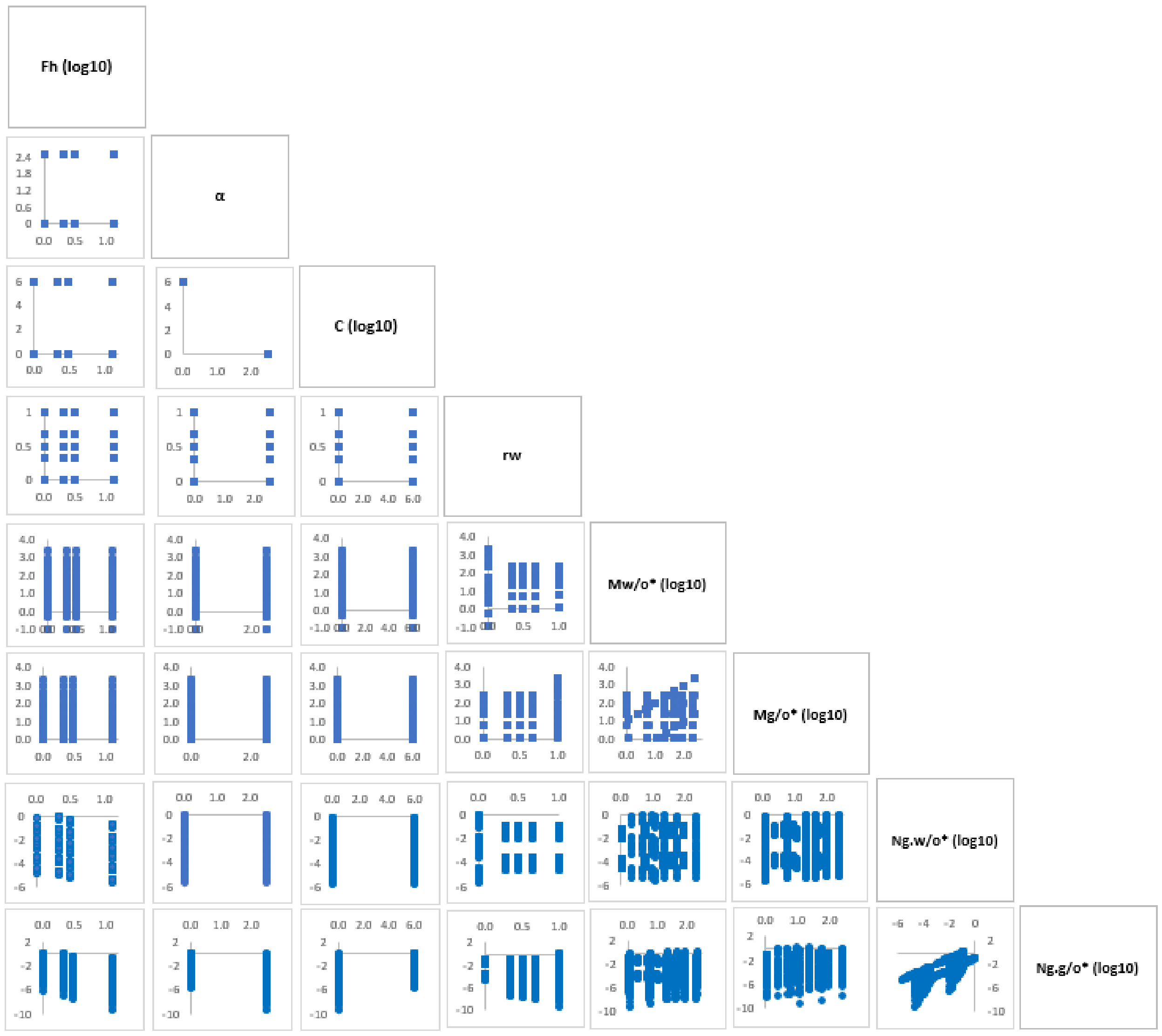


*Figure 4. 3*– Paired scatter plots of the input parameters

In total *Figure 4. 3* contains overall 1840+576 =2416 datapoints.

## 4.2 Extension of the dataset

As it was mentioned before in previous work hysteresis effect was used in terms of "no hysteresis" and "strong hysteresis". In simulation hysteresis effect was set using WAGHYSTR keyword in Eclipse (E100) and in case of absence of that effect this keyword was not used. For scaling process based on theoretical knowledge extremely high C=1000000 and $\alpha = 0$ were used for "no hysteresis" cases. However, in this work hysteresis effect will be varied to train the ML model. Previously set value of C for "no hysteresis" case stands out most among other parameters ranges as it is large number even in logarithmic scale (*Figure 4. 4*). So, perhaps it is possible to identify a lower value of C (than 1000000) that practically has the same effect of C hysteresis effect vanishing, where this new C would be used for new simulations. The formula used to calculate trapped gas saturation $s_{g,hyst}$ is based on combined Land (1968) and Carlson (1981) elements.

By setting limit to $s_{g,max} - s_{gr} = 0.05$ (5%) in (4.13) and $s_{g,hyst}$ was plotted against different C values (*Figure 4. 4*). From the plot it can be seen that $s_{g,hyst}$ approaches zero by increasing Land's parameter values and by accepting the accuracy when trapping of gas becomes less than 1%, the highest value for C was set to be 1000 or logC = 3.

$$s_{g,hyst} = s_{gr} + \frac{s_{g,max} - s_{gr}}{1 + C(s_{g,max} - s_{gr})} \qquad (4.13)$$

$$C = \frac{1 - s_{wc}}{(s_{gr})_{max}} - 1 \qquad (4.14)$$

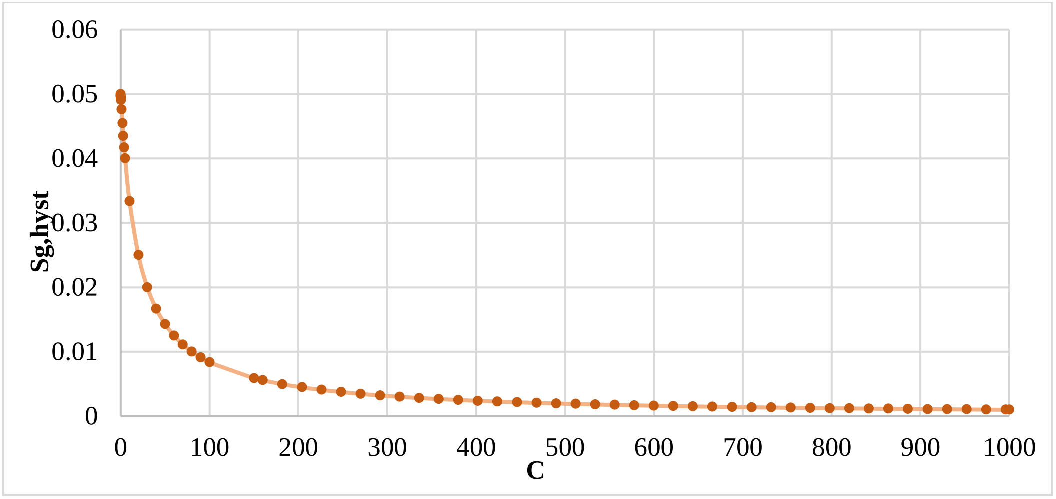


*Figure 4. 4*– Impact of C value on Sg,hyst

To identify C and $\alpha$ values combination which can effectively represent desired degree of hysteresis effect and avoid running too many new simulations, the impact of those values on degree of hysteresis effect must be studied.

To study influence of hysteresis in terms of different $\alpha$ and C values sensitivity test was carried out. Overall 308 cases with 77 (C, $\alpha$) combinations were run on Eclipse based on extreme scenarios of "homogeneous" (Fh=1) and "highly heterogeneous" (Fh=12.9), "no gravity" and "with gravity" with 1:1 WAG, when $\mu_w = 4$ and $\mu_g = 4$ . That was provided to understand behavior of recovery efficiency in from low hysteresis to moderate and highly hysteresis cases. C and $\alpha$ values tested for all cases are shown in *Table 4. 6* and *Figure 4. 5* (black dots). Results were plotted relating hysteresis values to scaled RF (color bar), where the lowest RF (purple) and the highest RF (red) for each special scenario correspond to 0 and 1, respectively (Figure *4. 5). The* recovery factor (RF) is used as an indicator of how strong the hysteresis is, based on C and alpha choices. However, one could plausibly use Sor instead if these and other parameter values are easily available for export (and analysis) from all reservoir sections. To exclude influence of other factors such as gravity and heterogeneity, scaled RF values was used. C values were plotted using logarithmic scale to keep compatibility with $\alpha$values. The values that were used in all combinations with existing parameters are highlighted with white circles (C=1000, $\alpha = 0$ and C=0, $\alpha = 2.5$).

Various combinations of C and $\alpha$ can be seen from the plots in *Figure 4. 5*. The behavior in different cases from *Figure 4. 5 (a-c)* show overall similar trends, changing from no hysteresis to strong hysteresis zones which correspond to low and high scaled RF values, respectively.

*Figure 4. 5(a)* illustrates homogeneous (Fh=1) reservoir, no gravity case where improvement in scaled RF values are gradual from high C and low $\alpha$ values to low C and high $\alpha$ values. In case of "homogeneous" (Fh=1) reservoir with gravity effect in *Figure 4. 5(b)*, green zone corresponding to moderate hysteresis effect is smaller than in (a) and does not exist in upper right corner of the map. That means that some of the positive effects of hysteresis are neutralized by gravity effect and higher hysteresis effect is needed to get similarly higher RF values, in comparison to lower hysteresis effect cases.

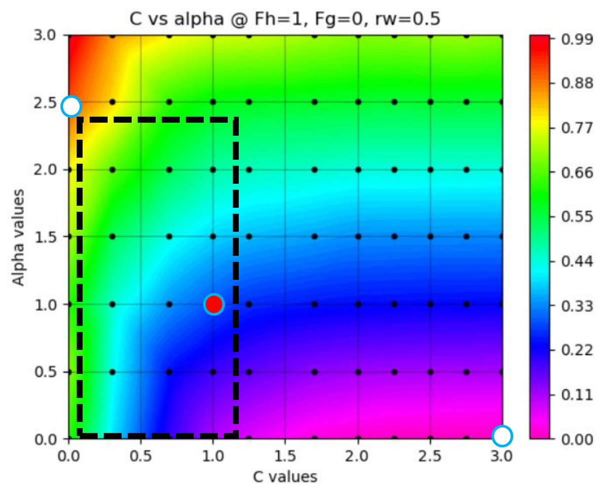
*Figure 4. 5 (c)* represents heterogeneous (Fh=12.9) reservoir - no gravity case which is similar to *Figure 4. 5 (a)* but with thinner moderate and high hysteresis zones. In "heterogeneous" (Fh=12.9) reservoir with gravity case in *Figure 4. 5 (d)* low hysteresis zone (light blue) is repeated twice, which makes it

We assume that the red and purple zones correspond to the higher and lower hysteresis effect zones, where existing extreme and no hysteresis values were already explored. Two zones unexplored in previous studies are the moderate (green) and moderate-to-low (blue) hysteresis effect, we are interested in those ones. All patterns of previous maps for individual cases are well overlapping in *Figure 4. 5 (c)*. The only difference between the maps plotted for individual cases is that the area of various color zones is changing to thinner or thicker, or slightly shifting their locations when the characteristics of WAG process vary in terms of heterogeneity and gravity. This confirms possibility to select one combination of hysteresis parameters representing all individual cases.

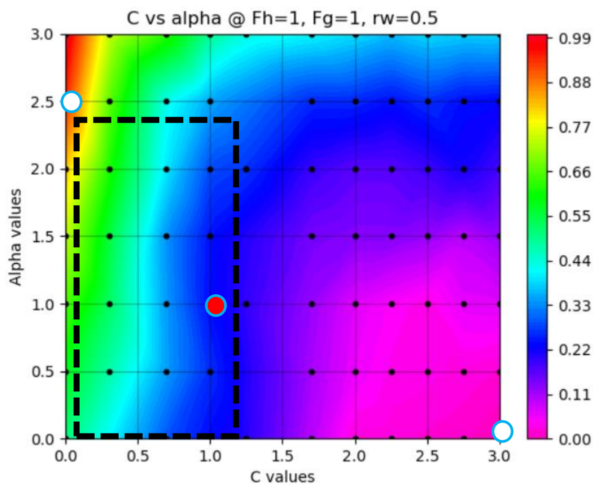
To narrow the area of interest of the hysteresis parameter values, we compare with values used in the literature, such as C in range 0.7-16.7 and $\alpha$ in range 0.01-2.8 as seen from *Table 4. 7*. That narrows the area of the study to within black dashed line square. Finally values for adding to the existing dataset are C= 10 (logC=1) and $\alpha$=1 (red point in *Figure 4. 5).*

*Table 4. 6*– C and $\alpha$ values used for test simulations

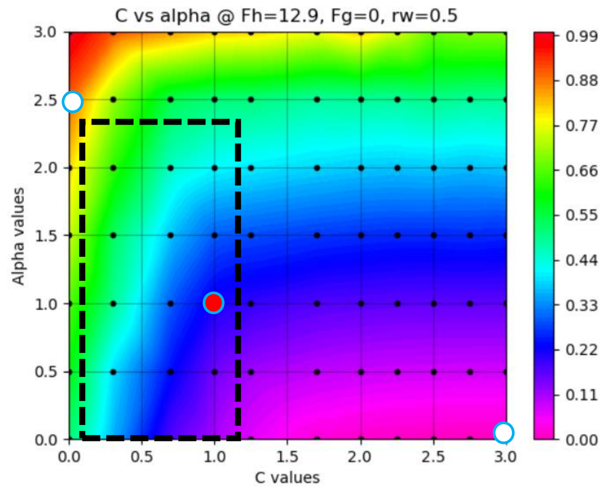
| # | Parameters | |
|---|---|---|
| | C | $\alpha$ |
| 1 | 1 | 0 |
| 2 | 2 | 0.5 |
| 3 | 5 | 1.0 |
| 4 | 10 | 1.5 |
| 5 | 17.8 | 2.0 |
| 6 | 50 | 2.5 |
| 7 | 100 | 3.0 |
| 8 | 178 | |
| 9 | 316 | |
| 10 | 562 | |
| 11 | 1000 | |
| Overall combinations C and $\alpha$ | **77** | |



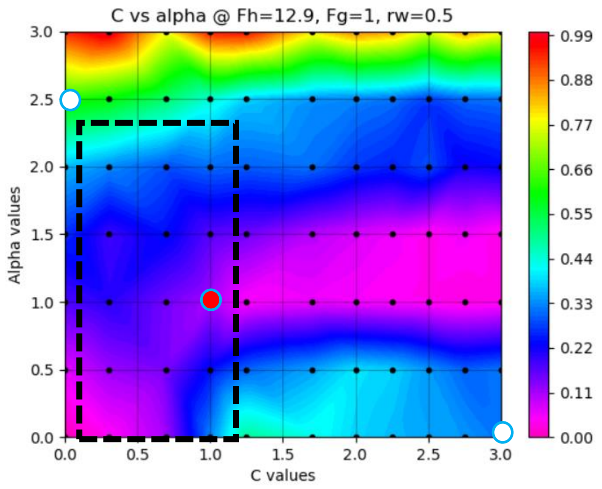
a. Homogeneous without gravity effect



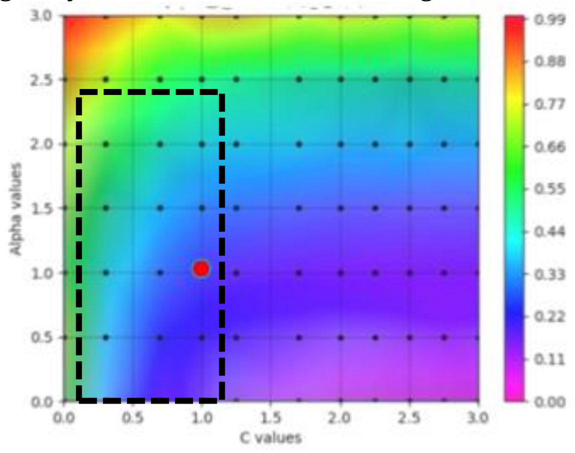
b. Homogeneous with gravity effect



c. Heterogeneous without gravity effect



d. Heterogeneous with gravity effect



e. overlapped maps from a to d

*Figure 4. 5*– Study of hysteresis influence on recovery factor using scaled RF values (color bar), C values

*Table 4. 7*– C and $\alpha$ values used in literature

| C | $\alpha$ | Reference |
|---|---|---|
| History-matched: 16.7 | History-matched :1.8 | (Mahzari and Sohrabi 2016) |
| Experimental: 7 | Experimental: 0.5 | |
| 2.7, 1.4 | 2.8,2.4 | (Talabi et al. 2019) |
| 0.7-2.2 | 0.01 | (Spiteri and Juanes 2006) |

### 4.2.1. Quality check of the results

Quality check was provided using cumulative gas and water injection volumes in reservoir conditions and injection rates to have constant reference for comparing cases. Combinations, which were removed in the previous work and have not been included in the previous dataset due to not meeting quality check criteria are shown in  Overall number of combinations when altering 5 different combinations of $rw$, $\mu_g$, $\mu_w$, 4 different values of heterogeneity factor and 2 values of density difference is 80.

Table 4. 8– Removed combinations of viscosity values in the dataset *Table 4. 8*. Overall number of combinations when altering 5 different combinations of $r_w$, $\mu_g$, $\mu_w$, 4 different values of heterogeneity factor and 2 values of density difference is 80.

*Table 4. 8*– Removed combinations of viscosity values in the dataset

| # | $r_w$ | $\mu_g$ | $\mu_w$ | *Fh* | **C,** $\alpha$ *combination* | $\Delta\rho$ |
|---|---|---|---|---|---|---|
| 1 | 0.33 | 20 | 20 | 1 | 0, 1000 | 1 |
| 2 | 0.5 | 20 | 4 | 2.1 | 2.5, 1 | 400 |
| 3 | 0.5 | 20 | 20 | 3 | | |
| 4 | 0.67 | 20 | 4 | 12.9 | | |
| 5 | 0.67 | 20 | 20 | | | |
| Combinations | 5 | | | 4 | 1 | 2 |
| Overall number simulations | 80 | | | | | |

To avoid the same problems with new runs the same combinations of viscosity values with new $\alpha$ and C values were not used. Overall, 824 simulation runs were performed with new C and $\alpha$ values combination with existing parameters values. All new runs do meet quality check criteria as all of them have the same cumulative gas and water injection volumes curves in reservoir conditions (*Figure 4. 6)* and injection rates (*Figure 4. 7).* The cumulative injection volumes value is 3.7MMm$^3$ and injection rate is 1014Sm$^3$/d.

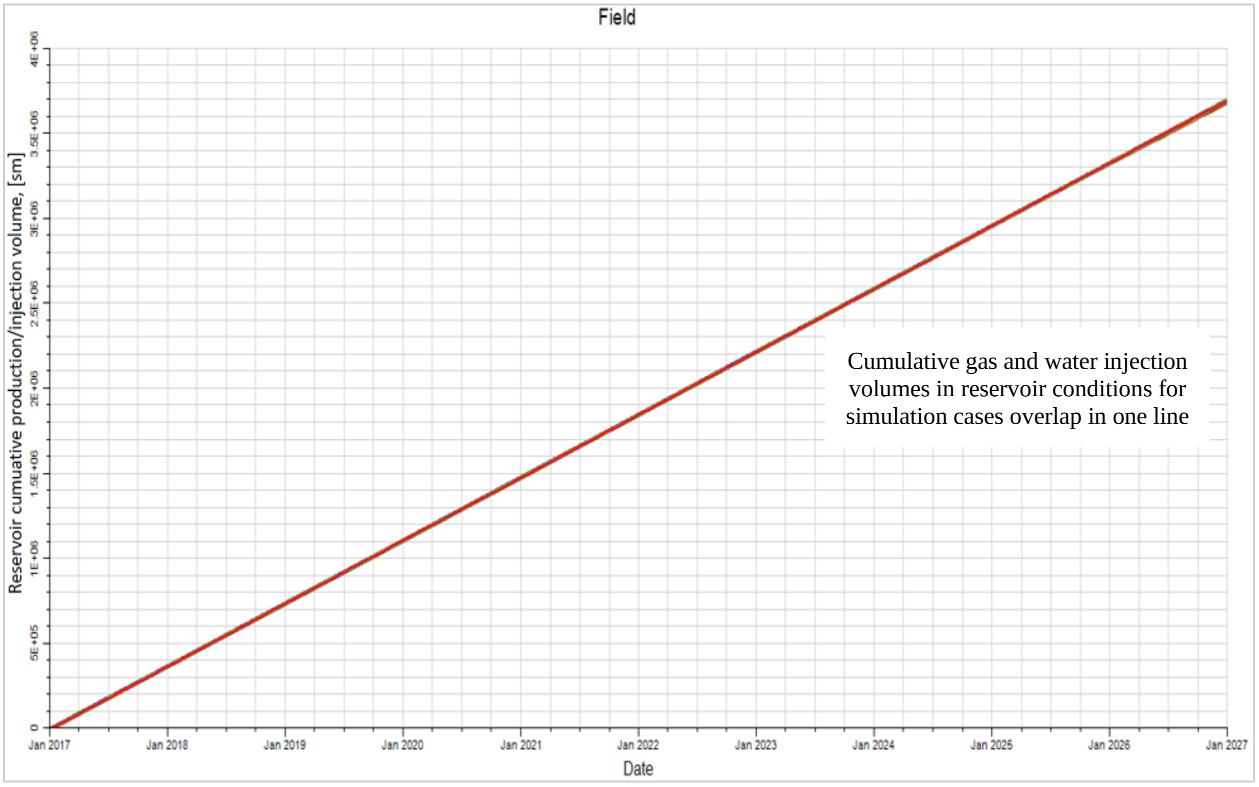


*Figure 4. 6*– Cumulative gas and water injection volumes in reservoir conditions for new simulation cases

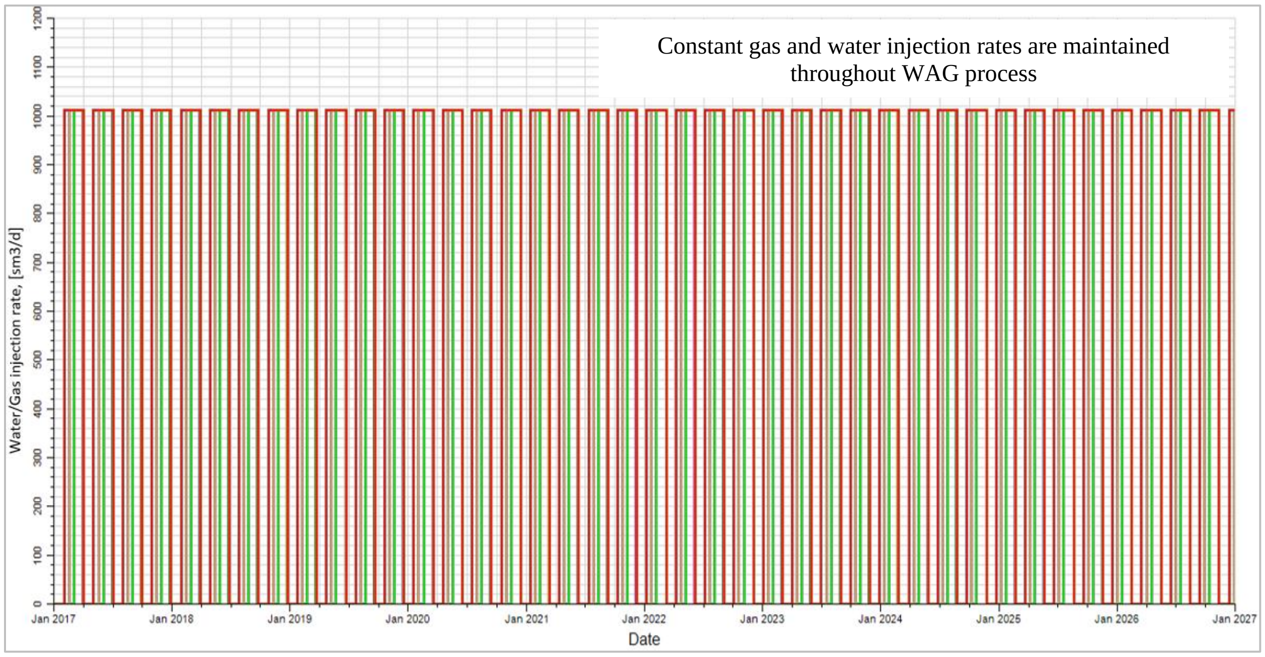


*Figure 4. 7*– Water/Gas injection rates for new simulation cases

824 simulation runs generated 824 new datapoints. With purpose of teaching LSSVM-PSO model to disregard C and $\alpha$ values when single phase flooding is applied 768 datapoints (with real and “fake” values) of previous study wad added with changing logC =3 to logC=1 and $\alpha$=0 to

$\alpha$ =1 values. Recovery factors are kept the same as in single phase flooding hysteresis does not exist and C and $\alpha$ can be combination of any values. So, the dataset was expanded to additional 1592 data points (*Figure 4. 8*). Red points were generated from new runs and blue points are from existing dataset. Some points are created by new runs overlap with existing ones for some parameters, which can be seen from the red boarders outside of blue points.

The *Figure 4. 8* has all 4008 data points including 2416 datapoints regarding to previous work's dataset (2416 datapoints) and 1592 datapoints created with new simulation runs. The dataset was randomized in order to ensure a good distribution of the data, and then it was split into training (80%) and testing (20%) parts. Afterwards normalization the dataset was normalized so that the LSSVM could operate with input values within 0 to 1, rather than specific low or specific high non-normalized input values.

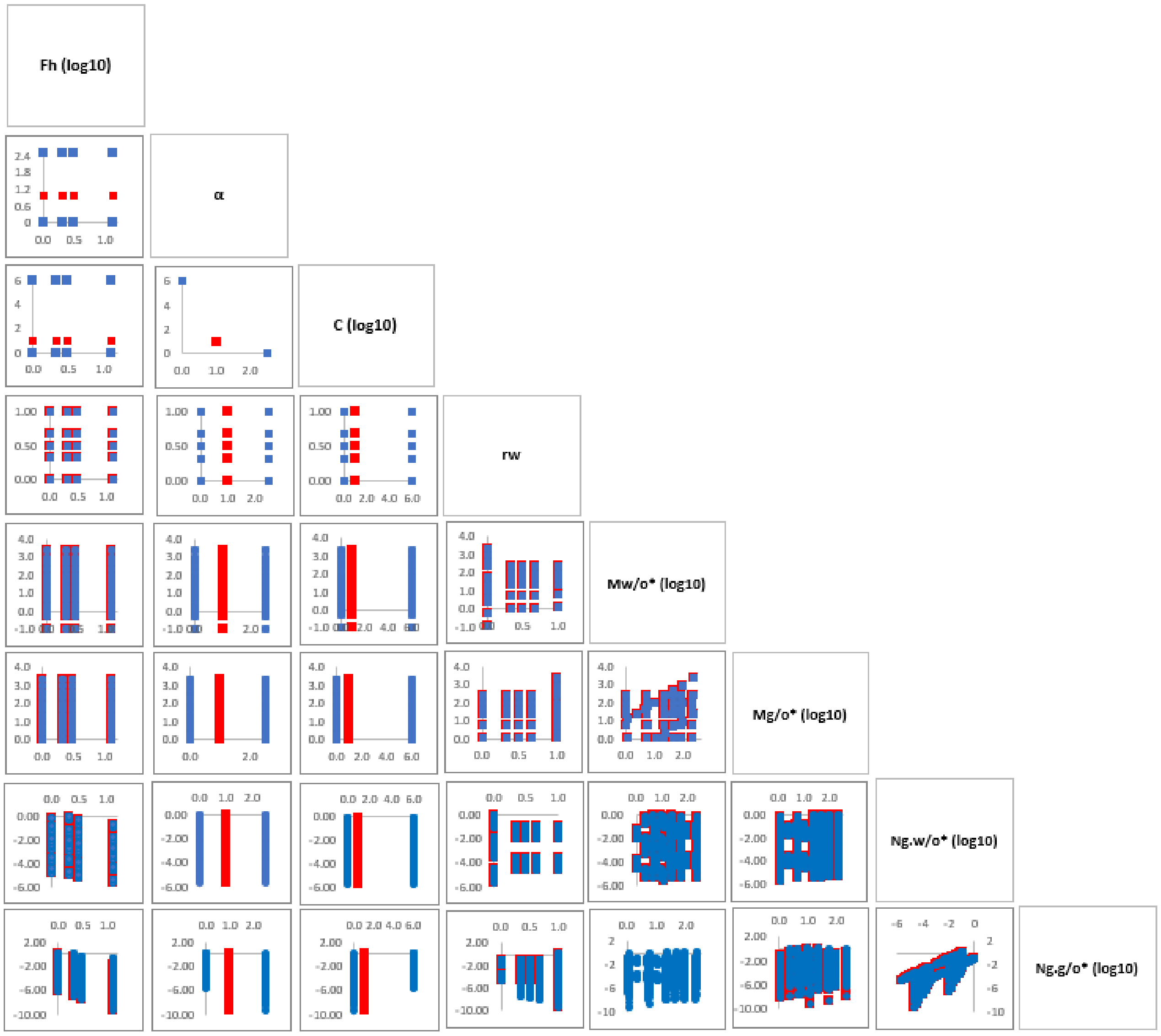


*Figure 4. 8*– Paired scatter plots of the input parameters (extended dataset)

## 4.3 Development of LSSVM-PSO algorithm from scratch

LSSVM with RBK algorithm code was written from scratch on Python 3.7.7 using Visual Studio Code editor. The LSSVM code integrated with PSO algorithm is presented in Appendix.

In this work the PSO algorithm is used to optimize $\sigma$ and $\gamma$ values for the LSSVM algorithm. As it was mentioned before the PSO algorithm is sensitive to the initial value of inertia weight parameter ($w$), $C_1$ and $C_2$, which play significant role in swarm convergence. Specifically, $w$ controls the size of the searching space. To make the algorithm more robust in that regard, random selection was used from range [0,2] for $C_1, C_2$ and [0,1] for $w$.

One of the difficult parts in setting up PSO is defining the stopping criteria for the process to be finished. It is hard to know what value for RMSE will be acceptable for $\sigma$ and $\gamma$ values before running the LSSVM-PSO code on the dataset. Therefore, there was no RMSE stopping criteria, and instead only the number of iterations that would be run. The PSO objective function was set to minimize the RMSE error between the predicted and real RF values of the training dataset. This leads to overfitting issues as illustrated in *Figure 4. 9*, where we can see that there is a good match with the prediction from the training data, but a poor prediction based on the testing data.

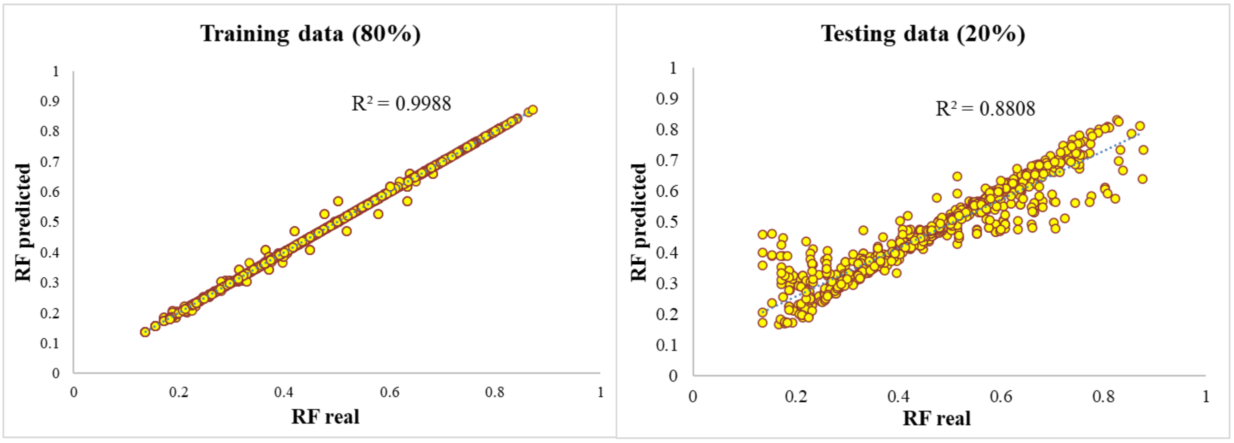


*Figure 4. 9* – Data overfitting: RF predicted vs. RF real

To understand which parameter was crucial in facing this kind of problem behavior of hyperparameters and their influence on the error for testing data was studied. By running PSO cases with 5, 10, 20 particles, the same behavior of $\gamma$ was observed. The algorithm starts developing very high values corresponding to lower $\sigma$, which ensures good fit on training data but poor prediction quality as it was demonstrated in *Figure 4. 9.*

$\sigma$ controls the width of the ε-insensitive zone, used to fit the training data. According the tests, it becomes either very low or stabilizes around 0.5-0.6 when model becomes overfitted.

As $\gamma$ balances the model complexity and the training error. The higher $\gamma$ gives better fit on training dataset, but at some stage, it stops influencing much on prediction accuracy if $\sigma$ remains constant. The *Figure 4. 9* shows how increasing $\gamma$ value affects fitting in training and testing dataset when $\sigma$ is kept constant at 0.5. Very little improvement in fitness of the results is observed between cases when $\gamma = 10$ and $\gamma = 100$. So, that means no need in choosing very high $\gamma$ value.

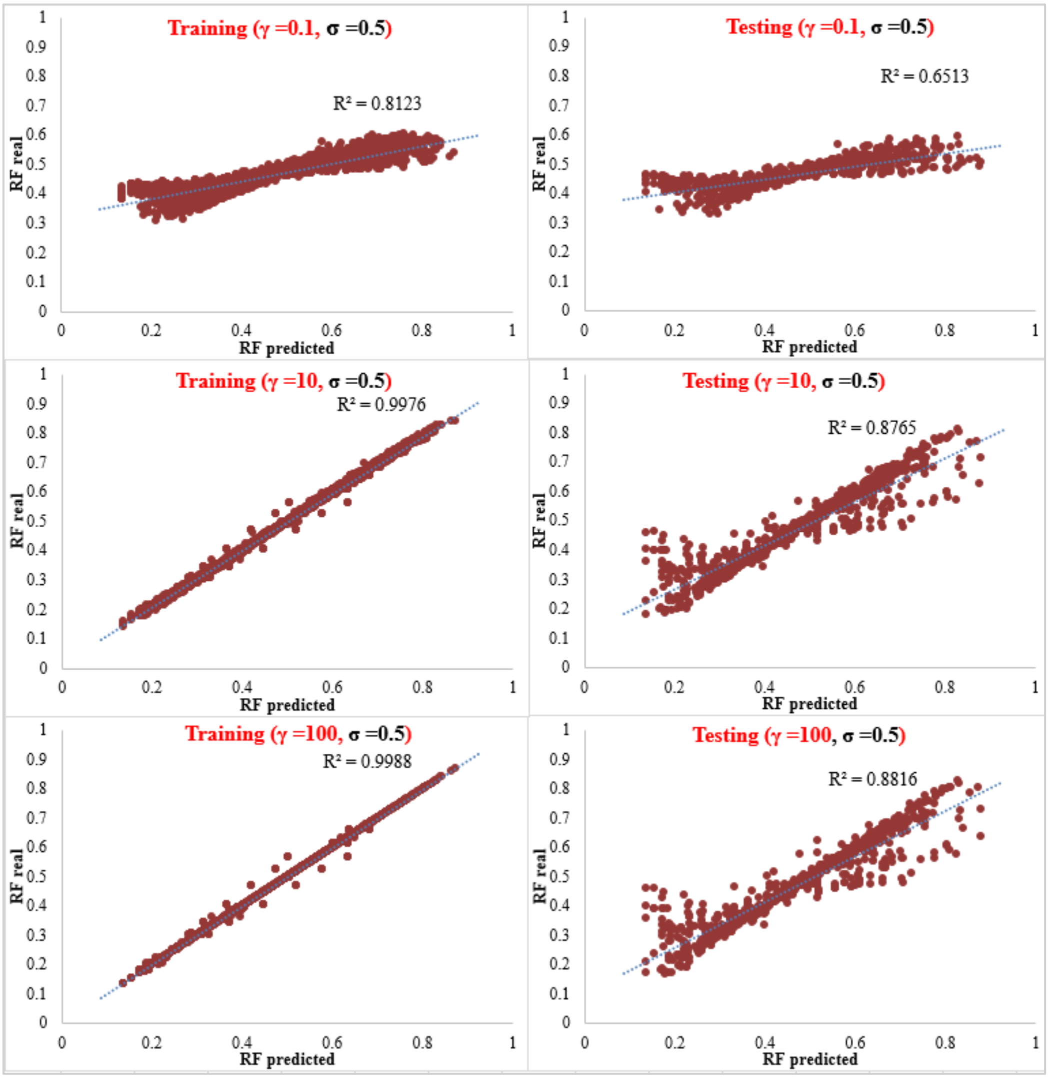


*Figure 4. 10* – RF predicted vs. RF real for different $\gamma$ values with constant $\sigma$

Previously the training (80 %) dataset was only validated against itself, which lead to overfitting issues. To solve this, we split the dataset instead by three fractions: training (70%), validation (15%) and testing (15%). The LSSVM model is trained on the training data, where the PSO objective function now instead is set to minimize the RMSE error between the predicted and real RF values of the validation dataset. The prediction error of the testing part is also calculated, but not used. This is to observe how it develops in comparison to the controlled prediction errors, as can be seen from *Figure 4. 11*. The behavior of various key parameters optimization process such as RMSE for training, validation and testing datasets; Difference between RMSE of training and validation dataset; γ and σ values is presented in *Figure 4. 11*, *Figure 4. 12*, *Figure 4. 13.* The algorithm was run 3 times to compensate for randomness of $C_1$, $C_2$ and w value.

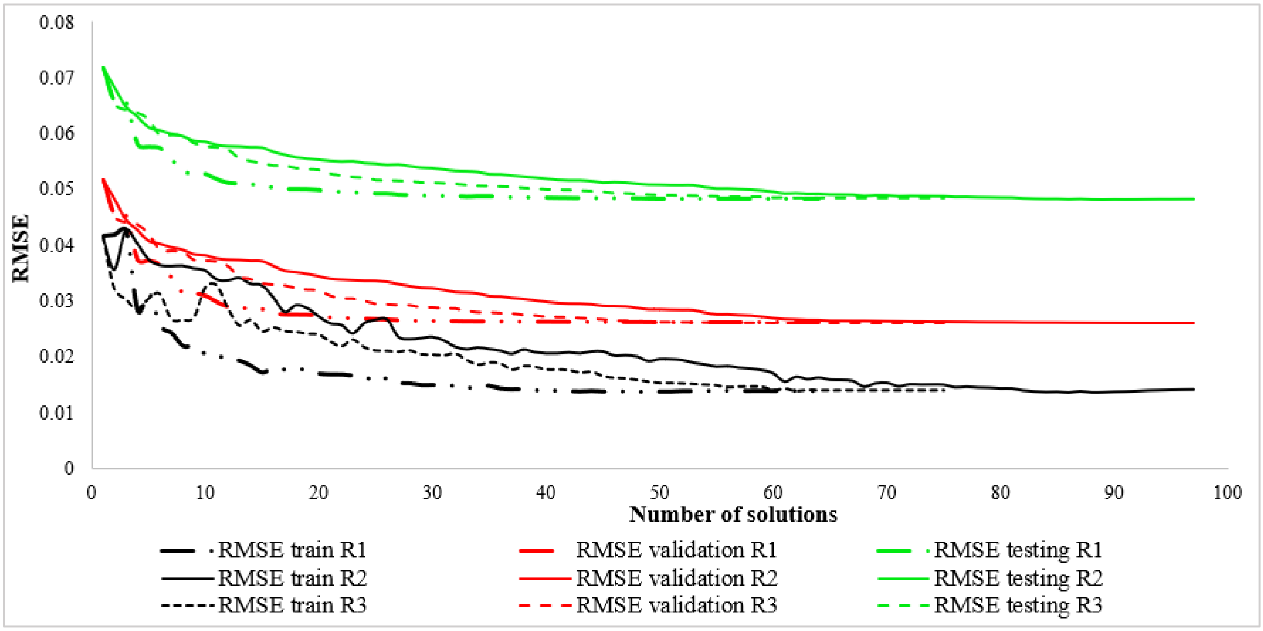


*Figure 4. 11* – Comparison of RMSE for training, validation and testing with 50 iterations, 10 particles runs

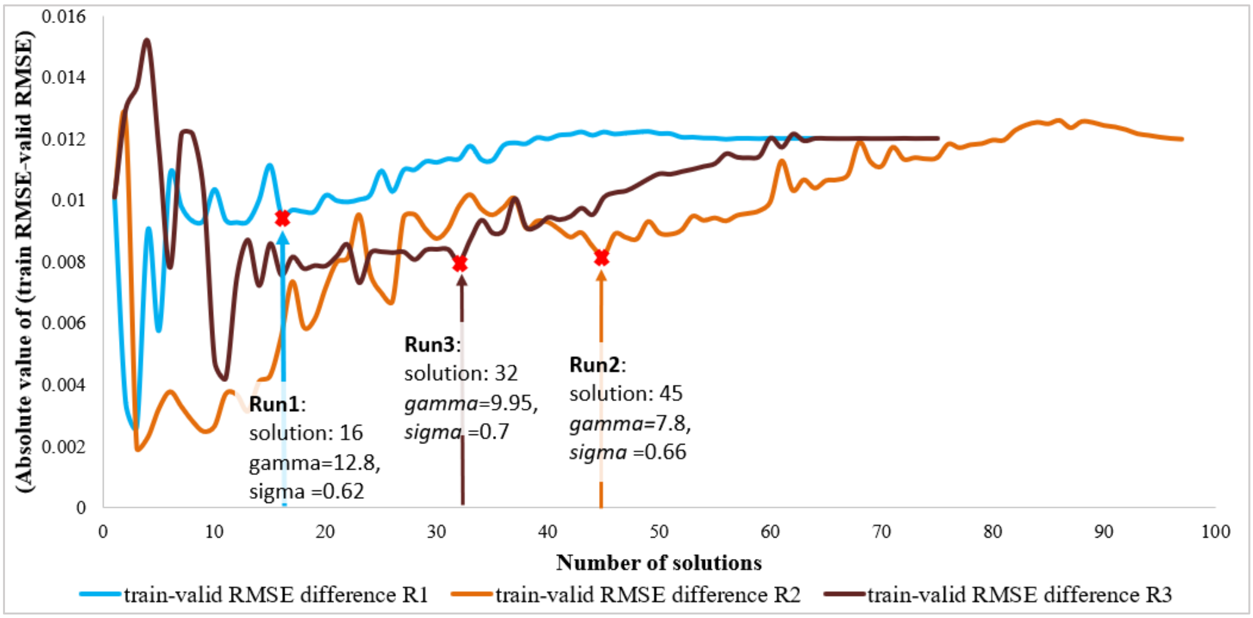


*Figure 4. 12* – Difference between RMSE of raining and validation dataset for 3 runs (50 iterations, 10 particles)

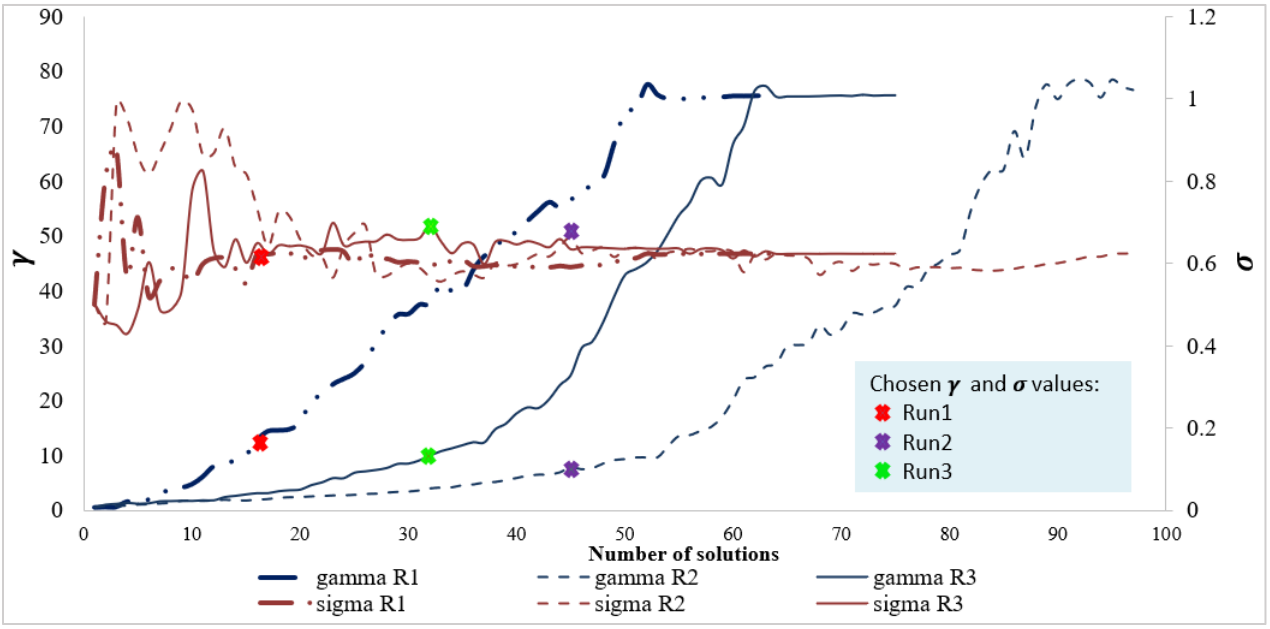


*Figure 4. 13* – Evolution of $\gamma$ and $\sigma$ values for 3 runs (50 iterations, 10 particles)

From the *Figure 4. 11* it is quite clear that all errors are following the same downwards trends. By minimizing the RMSE for a smaller part of the dataset, we see that the RMSE for training and testing parts will be also minimized, until they all eventually stabilize at a relatively constant RMSE value. The observed downwards trend in all of them seems to have been made possible due to the data being well distributed (or randomized) across the training, validation and testing datasets. It is likely that, whatever pattern or behavior that underlines all of these, that it has been preserved for all of the fractioned dataset. These similar patterns therefore make it possible for the non-linear model to improve prediction efficiency on the 15 % testing data that it has not "seen", even though the dataset was trained and validated on other parts of the dataset. Moreover, despite random $C_1$, $C_2$ and w values used for individual particles (and for each iteration) in PSO algorithm, they converge (individually) to roughly the same values of RMSE values. In other words, the results for the training, validation and testing parts seem representative, despite the random features introduced to the PSO algorithm. According to the *Figure 4. 13* $\gamma$ increases to higher value than $\sigma$. However, they both stabilize to the end of runs at value $\gamma = 78$ and $\sigma = 0.62$. As it was demonstrated before, when $\gamma$ reaches some value at constant $\sigma$, the error stops evolving

The whole purpose of studying PSO performance was to choose optimal values for value $\gamma$ and $\sigma$, which will improve non-linear model accuracy. Decision on optimal values of hyperparameters was made based on analysis on $|RMSE_{train} - RMSE_{valid}|$ evolution for three PSO algorithm runs (*Figure 4. 12*). The graphs in *Figure 4. 12* have 3 stages that happen at different number of solutions as they have random $C_1$, $C_2$ and w values: 1. decrease in difference between RMSE of training dataset and validation dataset (all three RMSE value follow downward trend in *Figure 4. 11*); 2. Stabilization stage when the curves are constant, so RMSE for both datasets are decreasing; 3. The difference starts rising until it stabilizes. The points before the differences in RMSE for training and validation dataset start rising correspond to solution numbers 16, 45 and 32 for runs

1, 2 and 3, respectively. The points marked by red "x" signs. Corresponding $\gamma, \sigma$ can be observed from *Figure 4. 13 and to confirm* correctness of our setups.

Table 4. 9. From those values in to confirm correctness of our setups.

Table 4. 9 $\gamma = 9.95,\ \sigma = 0.7$ were taken as optimal as they correspond to the lowest value of $|RMSE_{train} - RMSE_{valid}|$. I also can be seen that those values regard to the lowest $RMSE_{train}$, but we should stay unbiased to that. That is shown only for observation purpose to confirm correctness of our setups.

*Table 4. 9* - Properties of chosen optimal $\gamma,\ \sigma$ from each of runs

| Runs | $\gamma$ | $\sigma$ | $\lvert RMSE_{train} - RMSE_{valid}\rvert$ | $RMSE_{train}$ |
|---|---|---|---|---|
| Run 1 | 12.8 | 0.62 | 0.009564 | 0.050268731 |
| Run 2 | 7.8 | 0.66 | 0.008215 | 0.051256 |
| Run 3 | 9.95 | 0.7 | 0.008028 | 0.050855 |

As a part of study small experiment was provided with value of w, which was set to be decreasing through the iterations **(2.34, Chapter 2).** The results are demonstrated in *Figure 4. 14*.

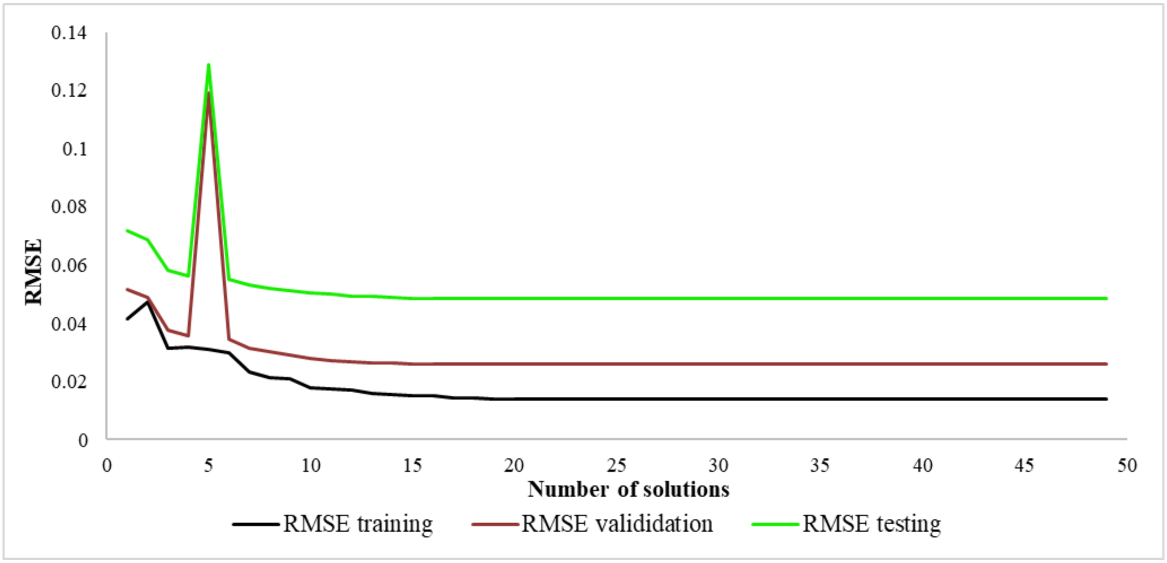


*Figure 4. 14*– RMSE for training, validation, and testing data with decreasing w

With systematic lowering of w value, the convergence is accelerated, and the algorithm finds the lowest error value for validation and training dataset 4-5 times faster. However, there is behavior of the graphs in the few iterations (spike), which makes this plot less convenient even it gives similar results with cases when w was selected randomly in each iteration. It is quite hard to see the process of swarm convergence using PSO, as it was shown in *Figure 4. 11*. As it was mentioned before, w influences on size of the searching space. That could be the reason why it converges faster with systematic decreasing inertia weight.

Chapter 5

# Results and discussions

## 5.1 Analysis of results generated by LSSVM-PSO model

Predicted recovery factor values using LSSVM-PSO model plotted against real recovery factor values are demonstrated in *Figure 5. 1*. Training, validation, and testing dataset parts are included in the same graph. Output generated by LSSVM-PSO model shows overall good fit of real data with low scatter. Goodness of fit is estimated using the coefficient of determination $R^2$.

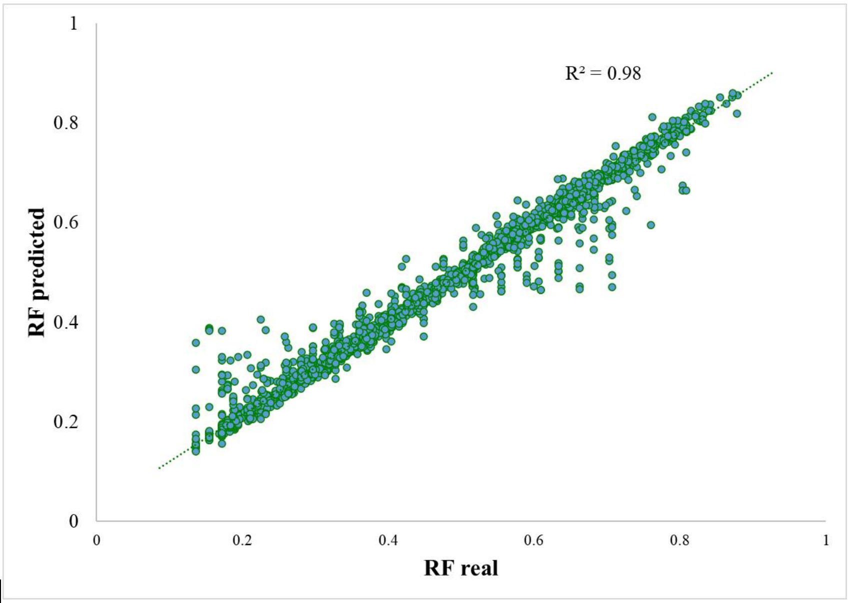


*Figure 5. 1* – RF predicted by LSSVM-PSO vs. real RF for the whole dataset

In previous work by Nygård and Andersen 2020 the dimensionless number M* was obtained by manual stepwise scaling of key mechanisms and properties influencing WAG process, so it can be linearly related to RF. However, some scatter was observed when plotting the M* values against the real RF values. Initial idea of this work was developing a non-linear model (LSSVM-PSO) trained on many input parameters related to WAG simulation cases data which potentially can perform better than previous model based on only one input parameter.

The performance LSSVM-PSO model in this work using 8 dimensionless input parameters was expected to show better accuracy in predicting RF values for WAG process as

LLSVM algorithm has more freedom in relating the given parameters to each other, so more complicated patterns can be captured, if they exist.

Direct comparison of two models' accuracy is not possible as the LSSVM-PSO model is not linear and more complex. To provide comparison on accuracy of two models, LSSVM algorithm was implemented using M* as a single input parameter for RF (output) values to get the most optimal line relating real RF values to M* values. The dataset with one input and one output parameter was divided to training, testing and validation parts, in similar way as 8 input parameter dataset was divided before. Then M*-based (1 parameter) model was used to predict of RF values for each of those dataset parts and results were plotted against real RF values. They are presented in *Figure 5. 2* on the first column. Prediction results for 8 parameter LSSVM-PSO model is shown in the same figure (the column).

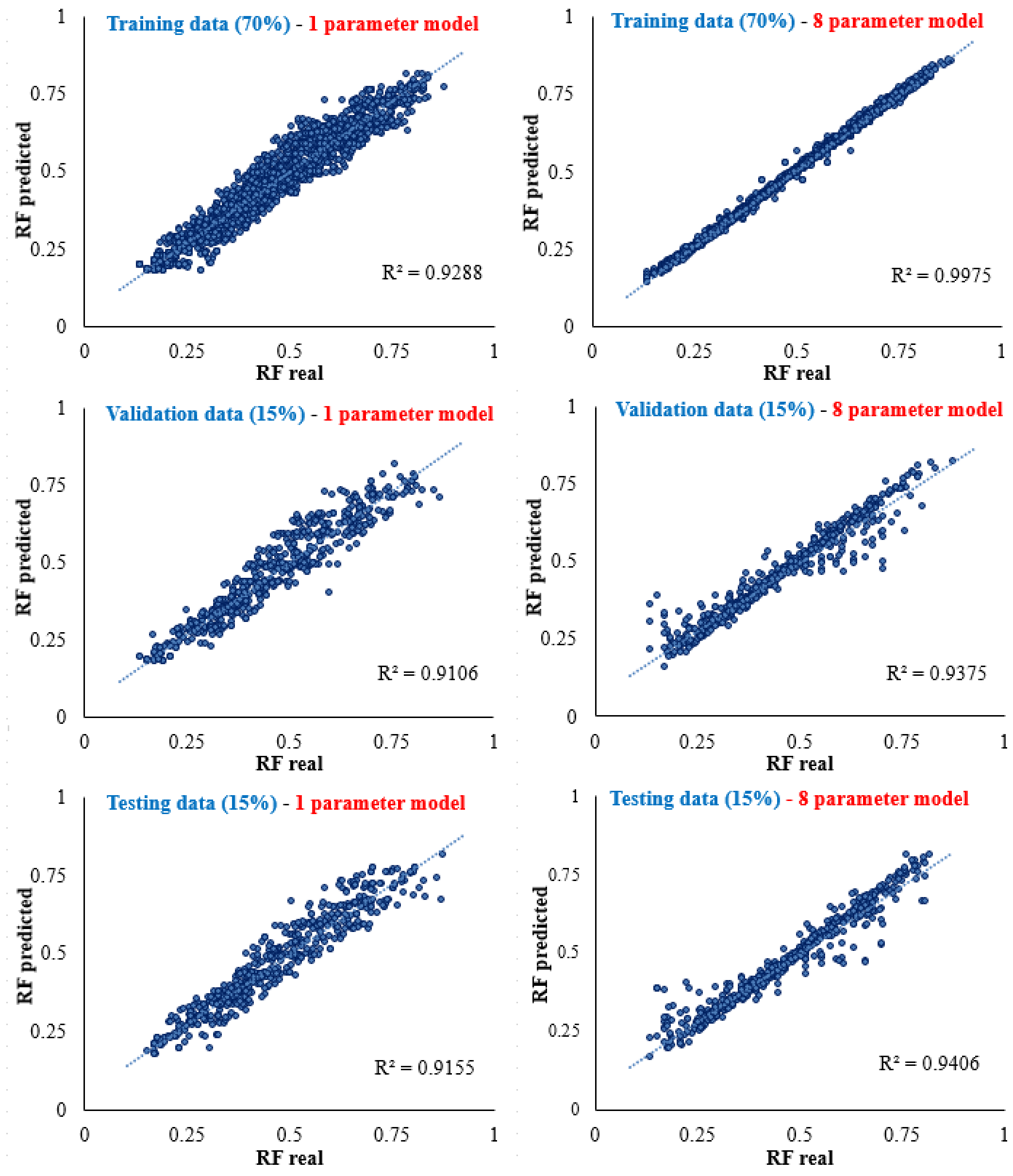


*Figure 5. 2* - Comparison of 8 parameter LSSVM-PSO and 1 parameter (M*) models results

As it can be seen from *Figure 5. 2* the LSSVM-PSO model (8 parameter) shows very good match to training data, which is expected outcome because the model was trained on that part of the dataset. The one parameter model has less accuracy reflected in higher scatter, corresponding to $R^2 = 0.9288$. Scatter has increased when moving to validation and training part for 8 parameter model, leading to a lower $R^2$ value. However, for M*-input model the prediction power has not much changed with using another part of the dataset.

Overall, two models demonstrated good prediction capability level, with slightly better performance of 8 input model. 8 parameters model was able to capture the main patterns in the dataset as it showed convenient prediction power when applied to testing dataset.

To observe prediction efficiency of the LSSVM-PSO model for each of the individual cases plots *Figure 5. 3* were created.

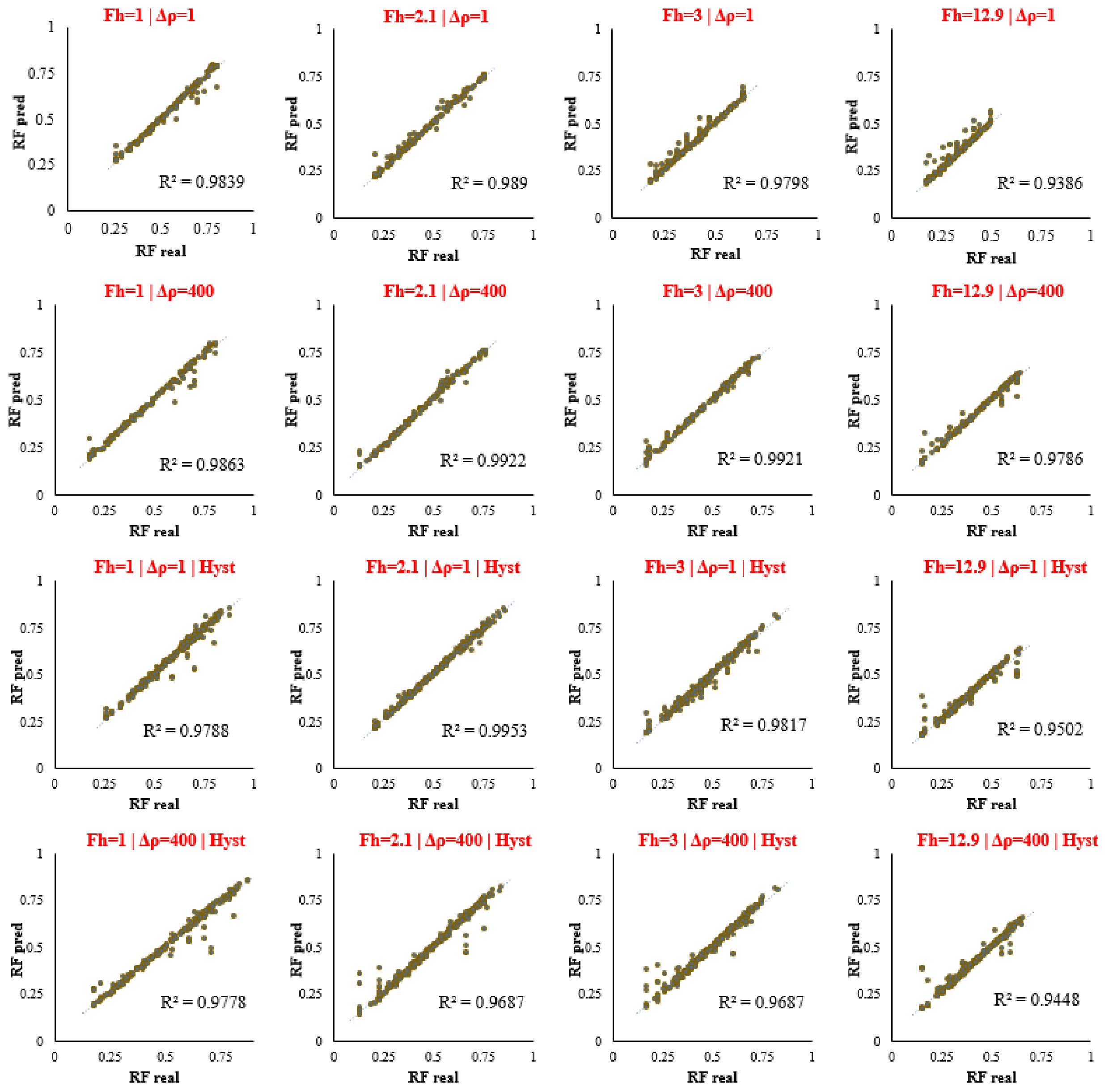


*Figure 5. 3*– RF predicted by LSSVM-PSO vs. real RF for the whole dataset for individual cases

*Figure 5. 3* demonstrates that non-linear model was able to capture the trends precise enough in all cases. Less accuracy is observed for in highly heterogeneous cases in comparison to less heterogeneous and homogenous ones, but it is still high level of predictive capability. That means the model is applicable for all considered cases.

## 5.2 Testing LSSVM-PSO model on a new dataset

### 6.2.1. Introducing a special case by adding a new parameter

In the previous study by limitations were fixed horizontal inclination and fixed WAG cycle frequency (Nygård and Andersen 2020), which can be interesting parameters to be studied. In this work a new parameter WAG cycle frequency was considered to study its effect on overall process and understand the scope of applicability of created LSSVM-PSO model. The model was trained on the dataset, which is related to constant WAG cycle length equal to 90 days. Also, it was not introduced in input parameters, consequently, was not part of the dataset.

Values 45 and 180 days for WAG cycle length the same for gas and water phases were implemented in special cases with new simulations. Adding a new parameter with 2 values, that will be changed in new simulations is always challenging decision due to rising number of overall combinations with other parameters. Based on that in new simulation cases only 2 values of heterogeneity factor ($F_H$) representing "homogeneous" and "highly heterogeneous" reservoirs were utilized. All unique values of other parameters are kept the same as in the dataset simulations used as the basis of LSSVM-PSO dataset and used when defining combinations with new parameter values. Overall, 1648 runs were performed. Values for all parameters can be seen from *Table 5. 1.*

*Table 5. 1*– Parameters used in extra cases

| Changing parameters values introduced in previous dataset | | | | | | | New parameter altered | Overall number of runs with all combinations |
|---|---|---|---|---|---|---|---|---|
| **$F_H$** | **$\Delta\rho$, kg/m$^3$** | **$r_w$** | **$C$** | **$\alpha$** | **$\mu_w$** | **$\mu_g$** | **$T^{cycle}$, days** | |
| 1.00 | 1 | | *1* | *2.5* | 20 | 20 | *45* | *1648* |
| 12.86 | 400 | 0.33 | 1000000 | 0 | 4 | 4 | *180* | |
| | | 0.50 | | | 1 | 1 | | |
| | | 0.67 | | | 0.5 | 0.5 | | |
| | | 0 | | | 0.25 | 0.25 | | |
| | | 1 | | | 0.1 | 0.1 | | |

All new runs do meet quality check criteria as all of them have the same cumulative gas and water injection volumes curves in reservoir conditions and injection rates equal to 1014Sm$^3$/d. The cumulative injection volumes value is 3.7MMm$^3$.

### 6.2.2. Results of testing LSSVM-PSO model on a new dataset

The LLSVM-PSO model was tested on new dataset with 1648 datapoints. The results plotted in a form of output of LSSVM-PSO (predicted RF) vs. results of simulations (real RF) can be observed from *Figure 5. 4.*

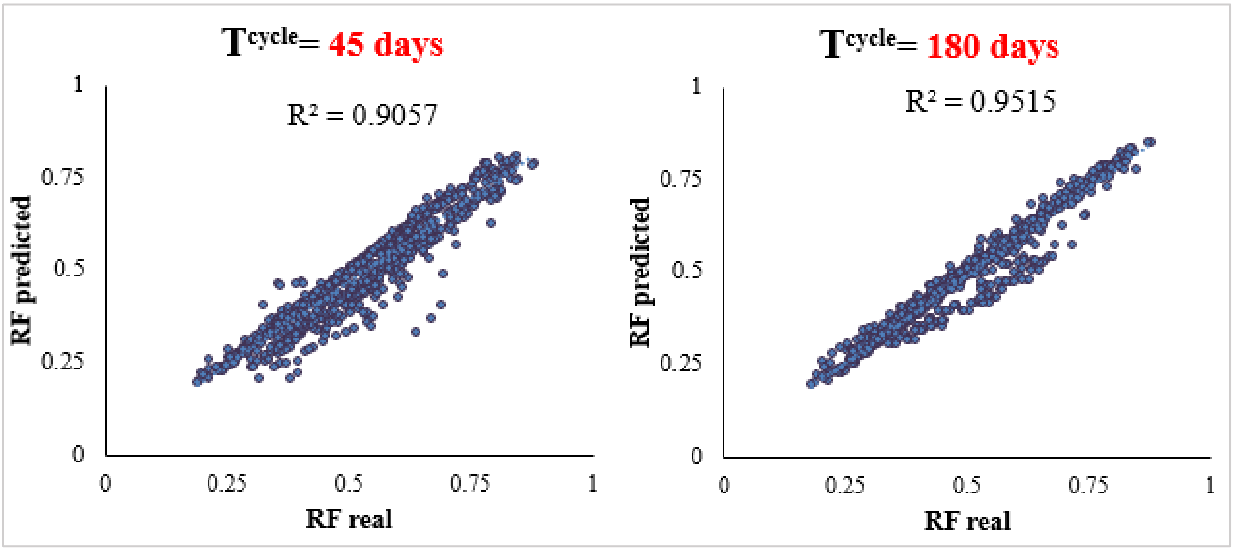


*Figure 5. 4* – Comparison of real RF and predicted (using LSSVM-PSO model) RF values for cases when WAG cycles legth are 180 (left) and 45 days (right)

Apparently non-linear model was able to capture the main trends in WAG process as it was able to perform prediction with decent accuracy on completely new dataset that was created by altering a new parameter, not introduced to it before.

To explore the match above (*Figure 5. 4)* in a more detailed manner, we plot the different cases individually, as seen from *Figure 5. 5 and Figure 5. 6.*

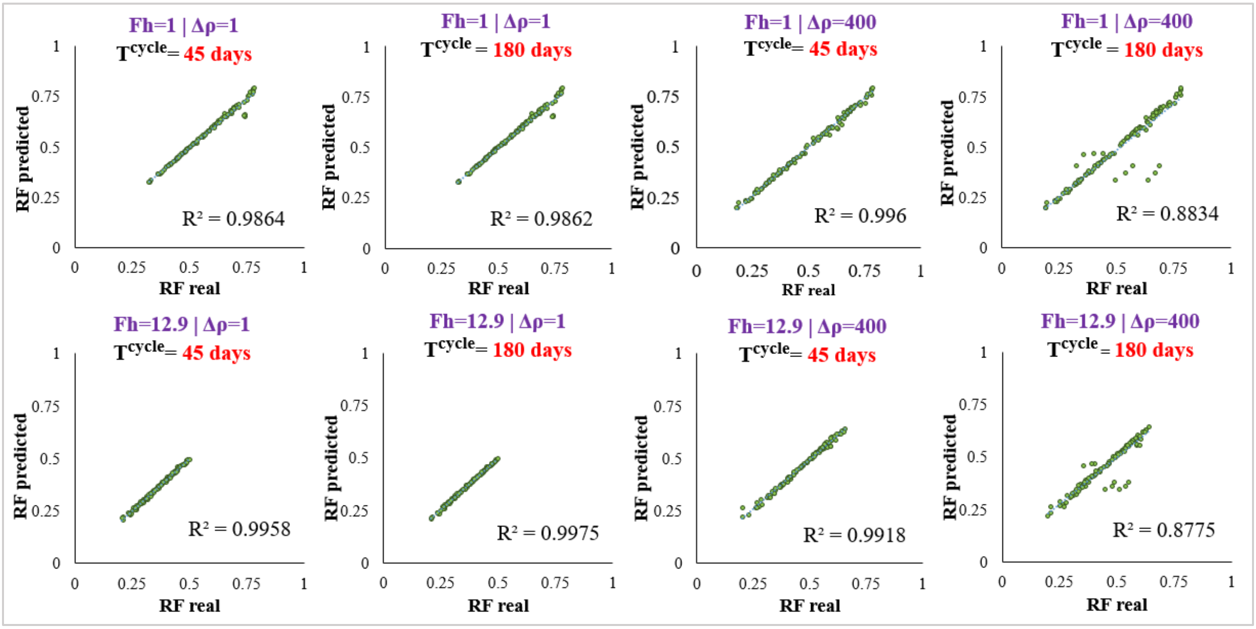


*Figure 5. 5* - Comparison of real RF and predicted RF values no hysteresis cases when WAG cycles length are 180 and 45 days

*Figure 5. 5* illustrates individual cases for homogeneous and heterogenous reservoirs in presence (columns 3,4) and absence (columns 1,2) of gravity effect. The LSSVM-PSO model can be very precise in prediction recovery factor of WAG applied in no hysteresis conditions regardless the change in WAG cycle length expect the cases corresponding to gravity cases when $T^{cycle}$=180 days. Those ones were predicted with relatively low accuracy, and higher scatter in comparison to the other ones in the same figure.
When the conditions get more complicated with hysteresis effect as in *Figure 5. 6*, the predictive power of the model decreases, and it becomes less convenient.

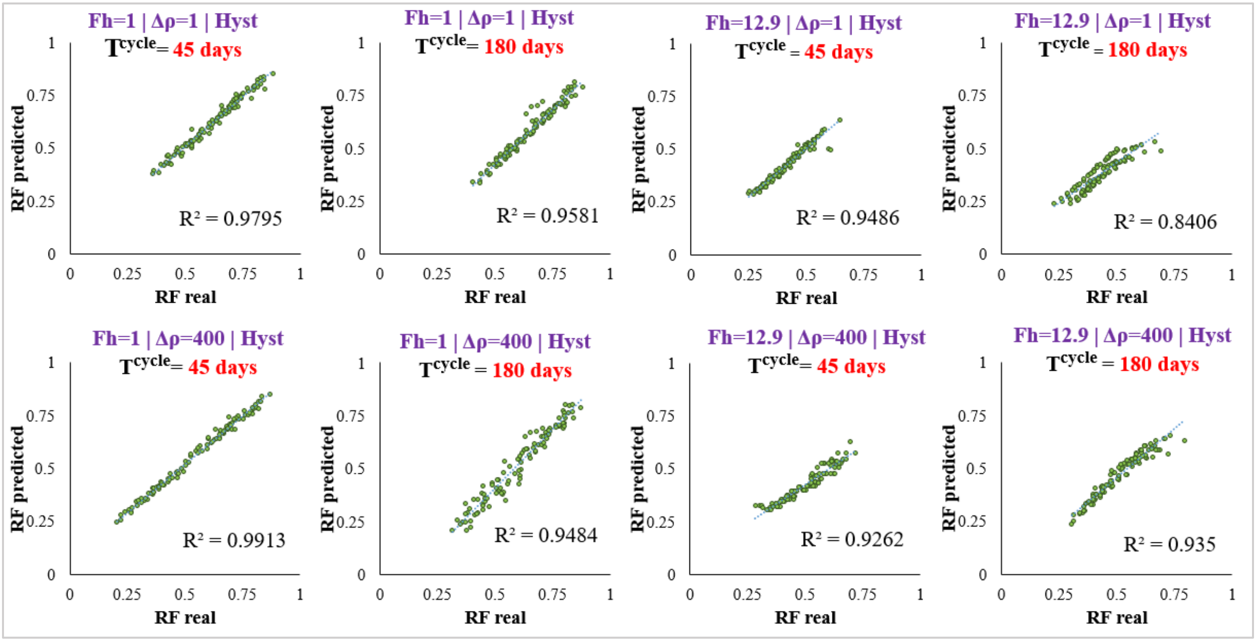


*Figure 5. 6*– Comparison of real RF and predicted RF values with hysteresis cases when WAG cycles length are 180 and 45 days

From combination of *Figure 5. 5* and *Figure 5. 6* For case when $T^{cycle}$=45 days, overall good match is observed from all individual cases, which leads to a suggestion that there is not much effect on WAG efficiency when WAG cycle length was changed from 180 to 45 days, in other words, cycle frequency was lowered two times. However, in case of $T^{cycle}$=90 days, it shows higher scatter in prediction results for "with hysteresis" models, especially for heterogeneous ones. That probably can refer to that influence of WAG half cycle on WAG efficiency is not linear and LSSVM-PSO model has to be trained on cases with new input parameter to improve prediction results.

Chapter 7

# Conclusion

The overall conclusions to this work can be summarized as this:

- In previous work performed by Nygård and Andersen 2020 dimensionless number M* was developed by incorporating key parameters of WAG process and relating to RF linearly. The dataset from the previous work was extended by running a new combination of C and $\alpha$, which was selected by studying impact of those parameters on hysteresis degree based on analysis of 77 combinations of (C, $\alpha$) in different cases of reservoir conditions. Input parameters for LSSVM were created based on grouping WAG process parameters into 8 dimensionless numbers. The final dataset that was used for developing LSSVM-PSO model contained 4008 data points, corresponding to labeled 8 input parameters and one output parameter (RF).
- It is important to select enough dataset fractions when using PSO to optimize LSSVM.
- Using only 2 dataset fractions (ie. training 80 % and testing 20 % parts) resulted in overfitness issues, which was due to that the model only was checked against itself.
- Overfitness issues was resolved by using 3 dataset fractions (ie. training 70 %, validation 15 % and testing 15 % parts), where the trained model checked itself against validation dataset.
- The LSSVM-PSO model's predictive power was compared to that of the previous model, with the goal of checking how well the main trends were captured. The non-linear model proved itself to be effective on WAG performance efficiency prediction, however previous model has shown also compatible match with real data.
- The LSSVM model was tested on new simulations where the previously constant WAG cycle length was changed. The model was still able to predict the main trends but had some trouble especially with doubled WAG cycle when heterogeneity and hysteresis effects were applied.
- Having tested the model on WAG cycle length, which it was not calibrated or trained against, proves that it could potentially be used in a broader scope as well. More work should be made to discover how universal the model could be.

# Appendix

LSSVM-PSO code

```
import os
import numpy as np
import numpy
import pandas as pd
import matplotlib.pyplot as plt
import time
import datetime
import multiprocessing
import concurrent.futures
import random
import math
from sklearn.utils import shuffle
from sklearn import preprocessing


# from numba import jit, autojit, prange
from multiprocessing import cpu_count
import cProfile, pstats, io
from pstats import SortKey

np.set_printoptions(linewidth=200, edgeitems=5)
pd.set_option("display.max_columns", 500)
pd.set_option("display.width", 2000)

# pylint: disable=unused-variable
# pylint: disable=unbalanced-tuple-unpacking
# pylint: disable=anomalous-backslash-in-string
# pylint: disable=abstract-class-instantiated


class LSSVM:
    def __init__(self, gamma, sigma):
        lists = [], [], [], [], []
        self.x_train, self.y_train, self.train_range, self.train_Mwag, self.train_Mstar = 
lists
        self.x_valid, self.y_valid, self.valid_range, self.valid_Mwag, self.valid_Mstar = 
lists
        self.x_test, self.y_test, self.test_range, self.test_Mwag, self.test_Mstar = lists

        self.gamma, self.sigma = gamma, sigma
        self.alphas, self.bias = [], []
        self.train_prediction = []
        self.valid_prediction = []
        self.test_prediction = []

    def import_and_separate_data(self, path, sheets, fractions):
        data = pd.read_excel(path, sheet_name=sheets[0])
        dataframe = shuffle(data)
        frac_rows_valid = math.ceil(fractions[1]*len(dataframe))
        frac_rows_test = math.ceil(fractions[2]*len(dataframe))
        frac_rows_train =len(dataframe) - frac_rows_valid-frac_rows_test
        valid =dataframe[0:frac_rows_valid]
        test = dataframe[frac_rows_valid:frac_rows_valid + frac_rows_test]
        train = dataframe[frac_rows_valid + frac_rows_test:]
```

```python
        len_train, len_valid, len_test = len(train), len(valid), len(test)
        all_train, all_valid, all_test = len_train, len_valid, len_test
        print(f"train:{len_train}, valid:{len_valid}, test:{len_test}")
        print(f"Cumulative train:{all_train}, valid:{all_valid}, test:{all_test}")

        # train, valid, test= np.array_split(dataframe, (fractions[:-1].cumsum() * len(dataframe)).astype(int))
        train_init = SplitFractionedData([], train, sheets[0], [], [], [], 0, {}, 7)
        input_tr, rf_tr, logMwag_tr, logMstar_tr, Ranges_tr, number_tr = train_init
        valid_init = SplitFractionedData([], valid, sheets[0], [], [], [], 0, {}, 7)
        input_v, rf_v, logMwag_v, logMstar_v, Ranges_v, number_v = valid_init
        test_init = SplitFractionedData([], test, sheets[0], [], [], [], 0, {}, 7)
        input_ts, rf_ts, logMwag_ts, logMstar_ts, Ranges_ts, number_ts = test_init
        print(number_tr, number_v, number_ts)

        train_sep, valid_sep, test_sep = [], [], []
        for (i, sheetname) in enumerate(sheets[1:]):
            data = pd.read_excel(path, sheet_name=sheetname)
            data = data.replace(np.nan, '', regex=True)
            dataframe = shuffle(data)
            frac_rows_valid = math.ceil(fractions[1]*len(dataframe))
            frac_rows_test = math.ceil(fractions[2]*len(dataframe))
            frac_rows_train =len(dataframe) - frac_rows_valid-frac_rows_test
            valid =dataframe[0:frac_rows_valid]
            test = dataframe[frac_rows_valid:frac_rows_valid + frac_rows_test]
            train = dataframe[frac_rows_valid + frac_rows_test:]
            len_train, len_valid, len_test = len(train), len(valid), len(test)
            all_train, all_valid, all_test = all_train + len_train, all_valid + len_valid, all_test + len_test

            #training
            train_sep, number_tr = SplitFractionedData(
                input_tr, train, sheetname, rf_tr, logMwag_tr, logMstar_tr, number_tr, Ranges_tr, 666,
            )
            input_tr, rf_tr, Ranges_tr, logMwag_tr, logMstar_tr = train_sep
            #training
            valid_sep, number_v = SplitFractionedData(
                input_v, valid, sheetname, rf_v, logMwag_v, logMstar_v, number_v, Ranges_v, 999
            )
            input_v, rf_v, Ranges_v, logMwag_v, logMstar_v = valid_sep
            #testing
            test_sep, number_ts = SplitFractionedData(
                input_ts, test, sheetname, rf_ts, logMwag_ts, logMstar_ts, number_ts, Ranges_ts, 666,
            )
            input_ts, rf_ts, Ranges_ts, logMwag_ts, logMstar_ts = test_sep

        self.x_train, self.y_train, self.train_range, self.train_Mwag, self.train_Mstar = train_sep
        self.x_valid, self.y_valid, self.valid_range, self.valid_Mwag, self.valid_Mstar = valid_sep
        self.x_test, self.y_test, self.test_range, self.test_Mwag, self.test_Mstar = test_sep

    def normalize_data(self):
        scalar = preprocessing.MinMaxScaler()
        self.x_train = scalar.fit_transform(self.x_train)
        self.y_train = scalar.fit_transform(self.y_train)
        self.x_valid = scalar.fit_transform(self.x_valid)
        self.y_valid = scalar.fit_transform(self.y_valid)
        self.x_test = scalar.fit_transform(self.x_test)
        self.y_test = scalar.fit_transform(self.y_test)


    def import_sheet_indexes(self, base_path, filenames, sheets):
        file_train, file_valid, file_test = filenames["train"], filenames["valid"], filenames["test"]
        path_train = os.path.realpath(f"{base_path}\\{file_train}.xlsx")
        path_valid = os.path.realpath(f"{base_path}\\{file_valid}.xlsx")
        path_test = os.path.realpath(f"{base_path}\\{file_test}.xlsx")
        data_train, data_valid, data_test = pd.read_excel(path_train), pd.read_excel(path_valid), pd.read_excel(path_test)
        data_train, data_valid, data_test = (
            data_train.replace(np.nan, "", regex=True),
            data_valid.replace(np.nan, "", regex=True),
            data_test.replace(np.nan, "", regex=True),
        )
        train_indexes, valid_indexes, test_indexes = {}, {}, {}
        for sheet in sheets:
            current_train = [int(item) for item in data_train[sheet].tolist() if str(item) != ""]
            current_valid = [int(item) for item in data_valid[sheet].tolist() if str(item) != ""]
            current_test = [int(item) for item in data_test[sheet].tolist() if str(item) != ""]
            train_indexes[sheet], valid_indexes[sheet], test_indexes[sheet] = current_train, current_valid, current_test
        self.train_range, self.valid_range, self.test_range = train_indexes, valid_indexes, test_indexes

    def export_sheet_indexes(self, base_path, filenames):
        file_train, file_valid, file_test = filenames["train"], filenames["valid"], filenames["test"]
        path_train = os.path.realpath(f"{base_path}\\{file_train}.xlsx")
        path_valid = os.path.realpath(f"{base_path}\\{file_valid}.xlsx")
        path_test = os.path.realpath(f"{base_path}\\{file_test}.xlsx")
        indexes_train, indexes_valid, indexes_test = pd.DataFrame({}), pd.DataFrame({}), pd.DataFrame({})

        for column in self.train_range.keys():
            dataframe = pd.DataFrame(data=self.train_range[column], columns=[column])
            indexes_train = pd.concat([indexes_train, dataframe], axis=1)

        for column in self.valid_range.keys():
            dataframe = pd.DataFrame(data=self.valid_range[column], columns=[column])
            indexes_valid = pd.concat([indexes_valid, dataframe], axis=1)

        for column in self.test_range.keys():
            dataframe = pd.DataFrame(data=self.test_range[column], columns=[column])
            indexes_test = pd.concat([indexes_test, dataframe], axis=1)

        with pd.ExcelWriter(path_train) as writer:
            indexes_train.to_excel(writer, index=False, sheet_name="Indexes_train")

        with pd.ExcelWriter(path_valid) as writer:
            indexes_valid.to_excel(writer, index=False, sheet_name="Indexes_test")

        with pd.ExcelWriter(path_test) as writer:
            indexes_test.to_excel(writer, index=False, sheet_name="Indexes_test")

    def export_train_valid_test_data(self, base_path, filenames):
        file_train, file_valid, file_test = filenames["train"], filenames["valid"], filenames["test"]
        path_train = os.path.realpath(f"{base_path}\\{file_train}.xlsx")
        path_valid = os.path.realpath(f"{base_path}\\{file_valid}.xlsx")
        path_test = os.path.realpath(f"{base_path}\\{file_test}.xlsx")
        columns = ["logFh", "α", "logC", "logMg/o*", "logMw/o*", "logNg,w/o", "logNg,g/o", "rw"]
        columns += ["logMwag", "logM*", "RF"]
        #training files
        train_data = np.hstack((self.x_train, self.train_Mwag, self.train_Mstar, self.y_train))
        train = pd.DataFrame(data=train_data, columns=columns)
        #validation files
        valid_data = np.hstack((self.x_valid, self.valid_Mwag, self.valid_Mstar, self.y_valid))
        valid = pd.DataFrame(data=valid_data, columns=columns)
        #testing files
        test_data = np.hstack((self.x_test, self.test_Mwag, self.test_Mstar, self.y_test))

        test = pd.DataFrame(data=test_data, columns=columns)


        with pd.ExcelWriter(path_train) as writer:
            train.to_excel(writer, index=False, sheet_name="All data train")

        with pd.ExcelWriter(path_valid) as writer:
            valid.to_excel(writer, index=False, sheet_name="All data valid")

        with pd.ExcelWriter(path_test) as writer:
            test.to_excel(writer, index=False, sheet_name="All data test")

    def export_after_lssvm_prediction(self, base_path, filenames):
        file_train, file_valid, file_test = filenames["train"], filenames["valid"], filenames["test"]
        path_train = os.path.realpath(f"{base_path}\\{file_train}.xlsx")
        path_valid = os.path.realpath(f"{base_path}\\{file_valid}.xlsx")
        path_test = os.path.realpath(f"{base_path}\\{file_test}.xlsx")
        columns = ["logFh", "α", "logC", "logMg/o*", "logMw/o*", "logNg,w/o", "logNg,g/o", "rw"]
        bias = np.zeros((np.shape(self.x_train)[0], 1))
        bias[0, 0] = self.bias
        print(bias)
        # temp_list = [self.x_train, self.train_Mwag, self.train_Mstar, self.y_train, self.train_prediction, self.b_alpha[1:, 0], bias]
        train_results = np.hstack((self.x_train, self.train_Mwag, self.train_Mstar, self.y_train, self.train_prediction, self.b_alpha[1:, 0], bias))
        train = pd.DataFrame(
            data=train_results, columns=columns + ["logMwag", "logM*", "RF", "RFpred", "alphas", "bias"],
        )

        valid_results = np.hstack((self.x_valid, self.valid_Mwag, self.valid_Mstar, self.y_valid, self.valid_prediction))
        valid = pd.DataFrame(data=valid_results, columns=columns + ["logMwag", "logM*", "RF", "RFpred"])

        test_results = np.hstack((self.x_test, self.test_Mwag, self.test_Mstar, self.y_test, self.test_prediction))
        test = pd.DataFrame(data=test_results, columns=columns + ["logMwag", "logM*", "RF", "RFpred"])

        with pd.ExcelWriter(path_train) as writer:
            train.to_excel(writer, index=False, sheet_name="All data train results")

        with pd.ExcelWriter(path_valid) as writer:
            valid.to_excel(writer, index=False, sheet_name="All data valid results")

        with pd.ExcelWriter(path_test) as writer:
            test.to_excel(writer, index=False, sheet_name="All data test results")

    def export_after_lssvm_prediction_as_sheets(self, base_path, filenames, sheets):
        file_train, file_valid, file_test = filenames["train"], filenames["valid"], filenames["test"]
        path_train = os.path.realpath(f"{base_path}\\{file_train}.xlsx")
        path_valid = os.path.realpath(f"{base_path}\\{file_valid}.xlsx")
        path_test = os.path.realpath(f"{base_path}\\{file_test}.xlsx")
        columns = ["logFh", "α", "logC", "logMg/o*", "logMw/o*", "logNg,w/o", "logNg,g/o", "rw"]
        bias = np.zeros((np.shape(self.x_train)[0], 1))
        bias[0, 0] = self.bias
        train_results = np.hstack(
            (
                self.x_train,
                self.train_Mwag,
                self.train_Mstar,
                self.y_train,
                self.train_prediction,
                self.b_alpha[1:, 0],
                bias,
            )
        )
        train = pd.DataFrame(
            data=train_results, columns=columns + ["logMwag", "logM*", "RF", "RFpred", "alphas", "bias"],
        )

        valid_results = np.hstack((self.x_valid, self.valid_Mwag, self.valid_Mstar, self.y_valid, self.valid_prediction))
        valid = pd.DataFrame(data=valid_results, columns=columns + ["logMwag", "logM*", "RF", "RFpred"])

        test_results = np.hstack((self.x_test, self.test_Mwag, self.test_Mstar, self.y_test, self.test_prediction))
        test = pd.DataFrame(data=test_results, columns=columns + ["logMwag", "logM*", "RF", "RFpred"])

        with pd.ExcelWriter(path_train) as writer:
            for item in sheets:
                indexes = self.train_range[item]
                data = pd.DataFrame(
                    data=train_results[indexes, :],
                    columns=columns + ["logMwag", "logM*", "RF", "RFpred", "alphas", "bias"],
                )
                data.to_excel(writer, index=False, sheet_name=item)

        with pd.ExcelWriter(path_valid) as writer:
            for item in sheets:
                indexes = self.valid_range[item]
                data = pd.DataFrame(
                    data=valid_results[indexes, :], columns=columns + ["logMwag", "logM*", "RF", "RFpred"],
                )
                data.to_excel(writer, index=False, sheet_name=item)

        with pd.ExcelWriter(path_test) as writer:
            for item in sheets:
                indexes = self.test_range[item]
                data = pd.DataFrame(
                    data=test_results[indexes, :], columns=columns + ["logMwag", "logM*", "RF", "RFpred"],
                )
                data.to_excel(writer, index=False, sheet_name=item)

    def use_already_separated_data(self, path_train, path_valid, path_test,  x_cols, y_cols, col_Mwag, col_Mstar, mode):
        if mode == "single":
            prep_train = pd.read_excel(path_train)
            prep_valid = pd.read_excel(path_valid)
            prep_test = pd.read_excel(path_test)
            self.x_train, self.x_valid, self.x_test = np.mat(prep_train[x_cols]), np.mat(prep_valid[x_cols]), np.mat(prep_test[x_cols])
            self.y_train, self.y_valid, self.y_test = (
                np.mat(prep_train[[y_cols[0]]]),
                np.mat(prep_valid[[y_cols[0]]]),
                np.mat(prep_test[[y_cols[0]]]),
            )
            self.train_Mwag, self.valid_Mwag, self.test_Mwag = (
                np.mat(prep_train[col_Mwag]),
                np.mat(prep_valid[col_Mwag]),
                np.mat(prep_test[col_Mwag]),
            )
            self.train_Mstar, self.valid_Mstar, self.test_Mstar = (
                np.mat(prep_train[col_Mstar]),
                np.mat(prep_valid[col_Mstar]),
                np.mat(prep_test[col_Mstar]),
            )

    def calculate(self):
        self.alphas, self.bias = np.mat(np.zeros((np.shape(self.x_train)[0], 1))), 0
        length = np.shape(self.x_train)[0]
        kernel = np.mat(np.zeros((length, length)))
        with concurrent.futures.ProcessPoolExecutor(max_workers=cpu_count()) as executor:

            results = [executor.submit(kernelTrans, self.x_train, self.x_train[i], sigma, i) for i in range(length)]
            for f in concurrent.futures.as_completed(results):
                kernel[:, f.result()[1]] = f.result()[0]
        # Prepare matrix parts
        leftOnes = np.mat(np.ones((length, 1)))
        innerMatrix = kernel + np.identity(length) * (1 / gamma)
        zeroEntry = np.mat(np.zeros((1, 1)))
        topOnes = leftOnes.T

        # Create final matrices
        topPart = np.hstack((zeroEntry, topOnes))
        botPart = np.hstack((leftOnes, innerMatrix))
        matrix = np.vstack((topPart, botPart))
        solution = np.vstack((zeroEntry, self.y_train))

        # Calculate bias and alpha values
        b_alpha = matrix.I * solution
        self.bias = b_alpha[0, 0]
        self.alphas = b_alpha[1:, 0]
        self.b_alpha = b_alpha

    def predict(self, x_ref, x_check):
        m = np.shape(x_check)[0]
        predict_result = np.mat(np.zeros((m, 1)))
        with concurrent.futures.ProcessPoolExecutor(max_workers=cpu_count()) as executor:
            results = [executor.submit(kernelTrans, x_ref, x_check[i, :], sigma, i) for i
in range(m)]
            for f in concurrent.futures.as_completed(results):
                Kx = f.result()[0]
                predict_result[f.result()[1], 0] = Kx.T * self.alphas + self.bias
        return predict_result

    def train(self):
        self.train_prediction = self.predict(self.x_train, self.x_train)

    def validate(self):
        self.valid_prediction = self.predict(self.x_train, self.x_valid)

    def check(self):
        self.test_prediction = self.predict(self.x_train, self.x_test)


class Particle:
    def __init__(self, values, setup, index, index_max, it_max):
        # Runs only at initialization of particle
        self.position = []
        self.velocity = []
        self.best_positions = []
        self.best_error = -1.0
        self.error_train = -1.0
        self.error_valid = -1.0
        self.error_test = -1.0
        self.dimensions = len(values)
        self.w_range = setup[2]
        self.c1_range = setup[0]
        self.c2_range = setup[1]
        self.w = 0.0
        self.c1 = 0.0
        self.c2 = 0.0
        self.c1c2s = [[0.0, 0.5],[0.5,1.0],[1.0,1.5],[1.5,2.0]]
        self.best_w = 0.0
        self.best_c1 = 0.0
        self.best_c2 = 0.0
        self.setup = setup
        self.index = index
        self.index_max = index_max
        self.iterations = it_max

        for i in range(self.dimensions):
            self.velocity.append(random.uniform(-1, 1))
            self.position.append(values[i])

    def check_fitness(self, error_function, param, iteration):
        self.error_train, self.error_valid, self.error_test = error_function(self.position
, param)[:]
```

```
        print(f"I{iteration+1} of {self.iterations}. P{self.index+1} of {self.index_max}. Train Error: {self.error_train}. Validation error: {self.error_valid}. Testing error: {self.error_test}.")

        # Initial values are best automatically
        if self.best_error == -1:
            self.best_positions = self.position
            self.best_error = self.error_valid

        # Did I improve my score?
        if self.error_valid < self.best_error and self.best_error>=0:
            self.best_w = self.w
            self.best_c1 = self.c1
            self.best_c2 = self.c2
            self.best_positions = self.position
            self.best_error = self.error_valid

    def update_velocity(self, global_positions):
        self.w = random.uniform(self.w_range[0], self.w_range[1])
        c1_range = random.choice(self.c1c2s)
        c2_range = random.choice(self.c1c2s)
        self.c1 = random.uniform(c1_range[0], c1_range[1])
        self.c2 = random.uniform(c2_range[0], c2_range[1])

        for i in range(0, self.dimensions):
            r1 = random.random()
            r2 = random.random()

            current_position = self.position[i]
            current_velocity = self.velocity[i]
            my_best_position = self.best_positions[i]
            best_global_position = global_positions[i]

            inertia = self.w * current_velocity
            ego_velocity = self.c1 * r1 * (my_best_position - current_position)
            collective_velocity = self.c2 * r2 * (best_global_position - current_position)

            self.velocity[i] = inertia + ego_velocity + collective_velocity

    def update_positions(self):
        for i in range(0, self.dimensions):
            self.position[i] = self.position[i] + self.velocity[i]


class PSO:
    def __init__(self, minimization_function, values, param, setup, number_of_particles, max_iterations):
        self.best_global_error = -1.0
        self.best_global_error_train = -1.0
        self.best_global_error_valid = -1.0
        self.best_global_error_test = -1.0
        self.best_global_positions = []
        self.best_global_setup = []
        self.best_global_w = 0.0
        self.best_global_c1 = 0.0
        self.best_global_c2 = 0.0
        self.max_iterations = max_iterations
        self.error_fx = minimization_function
        self.nparticles = number_of_particles
        self.logs, self.swarm = [], []
        self.param = param
```

```python
        self.max_iter = max_iterations
        for i in range(number_of_particles):
            particle = Particle(values, setup[i], i, number_of_particles, max_iterations)
            self.swarm.append(particle)

    def optimize(self):
        # Begin optimization
        i = 0
        count = 0
        start_time = datetime.datetime.now()
        while i < self.max_iterations:
            swarm = []
            with concurrent.futures.ProcessPoolExecutor(max_workers=cpu_count()) as executor:
                results = [
                    executor.submit(remote_check_fitness, self.param, self.swarm[pindex], self.error_fx, i)
                    for pindex in range(self.nparticles)
                ]
                for f in concurrent.futures.as_completed(results):
                    d = f.result()
                    swarm.append(f.result())
            self.swarm = swarm

            for pindex in range(self.nparticles):
                particle = self.swarm[pindex]
                if self.best_global_error == -1:
                    self.best_global_positions = list(particle.position)
                    self.best_global_error = particle.best_error
                    self.best_global_error_train = particle.error_train
                    self.best_global_error_valid = particle.error_valid
                    self.best_global_error_test = particle.error_test
                    strings = ["-" for _ in range(9)]
                    values = [count + 1] + strings + [i, self.max_iter, self.best_global_error_train]
                    values += [self.best_global_error_valid, self.best_global_error_test]
                    values += self.best_global_positions
                    self.do_logging(values)
                    count += 1

                # Check if current particle is best globally
                if (
                    (particle.best_error < self.best_global_error)
                    and particle.position[0] > 0
                    and particle.position[1] > 0
                ):
                    self.best_global_positions = list(particle.best_positions)
                    self.best_global_error = particle.best_error
                    self.best_global_error_train = particle.error_train
                    self.best_global_error_valid = particle.error_valid
                    self.best_global_error_test = particle.error_test
                    self.best_global_setup = particle.setup
                    self.best_global_w = particle.best_w
                    self.best_global_c1 = particle.best_c1
                    self.best_global_c2 = particle.best_c2
                    time_passed = datetime.datetime.now() - start_time
                    seconds = time_passed.total_seconds()
                    values = [count + 1] + particle.w_range + [particle.w]
                    values += particle.c1_range + [particle.c1]
                    values += particle.c2_range + [particle.c2]
                    values += [i, self.max_iter, self.best_global_error_train]
                    values += [self.best_global_error_valid, self.best_global_error_test]
                    values += self.best_global_positions
                    self.do_logging(values)
                    print("-------------------------------")
                    print(f"Count: {count+1}")
                    print(f"Global training error: {self.best_global_error_train}")
                    print(f"Global validation error: {self.best_global_error_valid}")
                    print(f"Global testing error: {self.best_global_error_test}")
                    print(f"Global positions: {self.best_global_positions[0]}, {self.best_global_positions[1]}")
                    print(f"Global setup: {self.best_global_setup}")
                    print(f"w: {self.best_global_w:.2f}, c1: {self.best_global_c1:.2f}, c2: {self.best_global_c2:.2f}")
                    print(f"Time passed: {time_passed}. Seconds: {seconds}")
                    print("-------------------------------")
                    count += 1

            for pindex in range(self.nparticles):
                self.swarm[pindex].update_velocity(self.best_global_positions)
                self.swarm[pindex].update_positions()
            i += 1
        final_time_passed = datetime.datetime.now() - start_time
        final_seconds = final_time_passed.total_seconds()
        print("-------------------------------")
        print(f"Count: {count}")
        print(f"Final positions: {self.best_global_positions[0]}, {self.best_global_positions[1]}")
        print(f"Final training error: {self.best_global_error_train}")
        print(f"Final validation error: {self.best_global_error_valid}")
        print(f"Final testing error: {self.best_global_error_test}")
        print(f"Final setup: {self.best_global_setup}")
        print(f"w: {self.best_global_w:.2f}, c1: {self.best_global_c1:.2f}, c2: {self.best_global_c2:.2f}")
        print(f"Time passed: {time_passed}. Seconds: {final_seconds}")
        print("-------------------------------")

    def do_logging(self, data):
        self.logs.append(data)


def SplitFractionedData(dataset, sheetdata, sheetname, rf, logMwag, logMstar, lastrow, sheetindex, option):
    sheetdata = sheetdata.replace(np.nan, '', regex=True)
    inputs = np.mat(sheetdata[["logFh", "α", "logC", "logMg/o*", "logMw/o*", "logNg,w/o", "logNg,g/o", "rw"]])
    rf_new = np.mat(sheetdata[["RF"]])
    logMwag_new = np.mat(sheetdata[["logMwag"]])
    logMstar_new = np.mat(sheetdata[["logM*"]])
    sheetindex[sheetname] = list(range(lastrow, lastrow + len(sheetdata)))
    lastrow = lastrow + len(sheetdata)

    if option == 7:
        return inputs, rf_new, logMwag_new, logMstar_new, sheetindex, lastrow
    else:
        rf = np.vstack((rf, rf_new))
        logMwag = np.vstack((logMwag, logMwag_new))
        logMstar = np.vstack((logMstar, logMstar_new))
        dataset = np.vstack((dataset, inputs))

        return [dataset, rf, sheetindex, logMwag, logMstar], lastrow


def kernelTrans(X, A, sigma, index):
    K = kernelTransInner(X, A, sigma)
    return K, index


def kernelTransInner(X, A, sigma):
    temp = X - A
    factor = 1 / (-1 * sigma ** 2)
    K = np.exp(np.array([np.inner(x, x) for x in temp]) * factor).reshape(len(X), 1)
    return K

def calculatev2(param, gamma, sigma):
    # Getting the data
    xparam, yparam = param[0], param[1]
    xtrain = np.memmap(xparam[0], dtype=xparam[1], shape=xparam[2], mode="r")
    ytrain = np.memmap(yparam[0], dtype=yparam[1], shape=yparam[2], mode="r")

    # Initializing
    length = np.shape(xtrain)[0]
    kernel = [kernelTrans(xtrain[:], xtrain[i], sigma, i)[0][:, 0] for i in range(length)]

    # Prepare full matrix
    matrix = np.ones((length + 1, length + 1))
    matrix[0, 0] = 0
    matrix[1:, 1:] = kernel + np.identity(length) * (1 / gamma)
    solution = np.zeros((length + 1, 1))
    solution[1:, 0] = ytrain[:, 0]

    # Calculate bias and alpha values
    b_alpha = np.dot(np.linalg.inv(matrix), solution)
    bias = b_alpha[0, 0]
    alphas = b_alpha[1:, 0]
    b_alpha = b_alpha
    return alphas, bias

def predict(alphas, b, xref_xcheck, sigma):
    xref = np.memmap(xref_xcheck[0][0], dtype=xref_xcheck[0][1], shape=xref_xcheck[0][2],
mode="r")
    xcheck = np.memmap(xref_xcheck[1][0], dtype=xref_xcheck[1][1], shape=xref_xcheck[1][2]
, mode="r")
    m = np.shape(xcheck)[0]
    predict_result = np.mat(np.empty((m, 1)))
    for i in range(m):
        Kx = kernelTrans(xref, xcheck[i, :], sigma, i)[0][:, 0]
        predict_result[i, 0] = np.dot(Kx, alphas) + b
    return predict_result

def prediction_error(values, param):
    y_train = np.memmap(param[1][0], dtype=param[1][1], shape=param[1][2], mode="r")
    y_valid = np.memmap(param[3][0], dtype=param[3][1], shape=param[3][2], mode="r")
    y_test = np.memmap(param[5][0], dtype=param[5][1], shape=param[5][2], mode="r")
    gamma, sigma = values
    alphas, bias = calculatev2(param, gamma, sigma)
    y_pred_train = predict(alphas, bias, [param[0], param[0]], sigma)
    y_pred_valid = predict(alphas, bias, [param[0], param[2]], sigma)
    y_pred_test = predict(alphas, bias, [param[0], param[4]], sigma)
    error_train = np.sqrt(np.sum(np.square(y_train-y_pred_train))/len(y_train))
    error_valid = np.sqrt(np.sum(np.square(y_valid-y_pred_valid))/len(y_valid))
    error_test = np.sqrt(np.sum(np.square(y_test-y_pred_test))/len(y_test))
    return [error_train, error_valid, error_test]


def remote_check_fitness(param, particle, function, iteration):
    particle.check_fitness(function, param, iteration)
    return particle


def setup(particles):
    #c1c2s = [[0.0, 2.0]]
    c1c2s = [[0.0, 0.5],[0.5,1.0],[1.0,1.5],[1.5,2.0]]
    ws = [[0.0, 1.0]]
    setup = []
    for i in range(len(c1c2s)):
        c1 = c1c2s[i]
        for j in range(len(c1c2s)):
            c2 = c1c2s[j]
            for k in range(len(ws)):
                w = ws[k]
                addition = [c1, c2, w]
                for _ in range(particles):
                    setup.append(addition)
    return setup


if __name__ == "__main__":
    print("--------------------Parameter Setup------------------")

    print("-------------------Save LSSVM Model-----------------")

    fractions = np.array([0.7, 0.15, 0.15])

    base_path = "C:\\Users\\aizha\\PycharmProjects\\HelloWorld\\Output LSSVM\\Results"

    base = "C:\\Users\\aizha\\PycharmProjects\\HelloWorld\\Input LSSVM"
    filename = "Dataset_v13"
    path = os.path.realpath(f"{base}\{filename}.xlsx")
    Sheets = [
        "Fh=1 | Δρ=1",
        "Fh=2.1 | Δρ=1",
        "Fh=3 | Δρ=1",
        "Fh=12.9 | Δρ=1",
        "Fh=1 | Δρ=400",
        "Fh=2.1 | Δρ=400",
        "Fh=3 | Δρ=400",
        "Fh=12.9 | Δρ=400",
        "Fh=1 | Δρ=1 | Hyst",
        "Fh=2.1 | Δρ=1 | Hyst",
        "Fh=3 | Δρ=1 | Hyst",
        "Fh=12.9 | Δρ=1 | Hyst",
        "Fh=1 | Δρ=400 | Hyst",
        "Fh=2.1 | Δρ=400 | Hyst",
        "Fh=3 | Δρ=400 | Hyst",
        "Fh=12.9 | Δρ=400 | Hyst",
    ]

    gamma, sigma = 9.95, 0.7

    start_time_original = datetime.datetime.now()
    lssvm = LSSVM(gamma, sigma)


    print("--------------------------------Import and split the data------------------------")
    start_time = datetime.datetime.now()
    lssvm.import_and_separate_data(path, Sheets, fractions)
    time_passed = datetime.datetime.now() - start_time
    deltatime = datetime.timedelta(seconds=time_passed.total_seconds())
    print(f"Duration: {deltatime}")
    print("--------------------------------Import and split the data: done------------------")


    print("--------------------------------Exporting split data-----------------------------")
    filenames_split = {"train": f"combined_train_{fractions[0]}ratio_v13", "valid": f"combined_valid_{fractions[1]}ratio_v13", "test": f"combined_test_{fractions[2]}ratio_v13"}
    start_time = datetime.datetime.now()
    lssvm.export_train_valid_test_data(base_path, filenames_split)
    time_passed = datetime.datetime.now() - start_time
    deltatime = datetime.timedelta(seconds=time_passed.total_seconds())
    print(f"Duration: {deltatime}")
    print("--------------------------------Exporting split data: done-----------------------")


    print("--------------------------------Exporting indexes--------------------------------")
    filenames_indexes = {"train": "0.7_train_indexes_v13", "valid": "0.15_valid_indexes_v13", "test": "0.15_test_indexes_v13"}
    start_time = datetime.datetime.now()
    lssvm.export_sheet_indexes(base_path, filenames_indexes)
    time_passed = datetime.datetime.now() - start_time
    deltatime = datetime.timedelta(seconds=time_passed.total_seconds())
    print(f"Duration: {deltatime}")
    print("--------------------------------Exporting indexes: done--------------------------")


    print("--------------------------------Importing indexes--------------------------------")
    start_time = datetime.datetime.now()
    lssvm.import_sheet_indexes(base_path, filenames_indexes, Sheets)
    time_passed = datetime.datetime.now() - start_time
    deltatime = datetime.timedelta(seconds=time_passed.total_seconds())
    print(f"Duration: {deltatime}")

    print("--------------------------------Using constant reference data--------------------")
    filenames_split = {"train": f"combined_train_{fractions[0]}ratio_v13", "valid": f"combined_valid_{fractions[1]}ratio_v13", "test": f"combined_test_{fractions[2]}ratio_v13"}
    file_train, file_valid, file_test = filenames_split["train"], filenames_split["valid"], filenames_split["test"]
    path_train = os.path.realpath(f"{base_path}\\{file_train}.xlsx")
    path_valid = os.path.realpath(f"{base_path}\\{file_valid}.xlsx")
    path_test = os.path.realpath(f"{base_path}\\{file_test}.xlsx")
    colm_x = ["logFh", "α", "logC", "logMg/o*", "logMw/o*", "logNg,w/o", "logNg,g/o", "rw"]
    start_time = datetime.datetime.now()
    lssvm.use_already_separated_data(path_train,path_valid, path_test, colm_x, ["RF"], ["logMwag"], ["logM*"], "single")
    time_passed = datetime.datetime.now() - start_time
    deltatime = datetime.timedelta(seconds=time_passed.total_seconds())
    print(f"Duration: {deltatime}")

    print("--------------------------------Starting LSSVM run-------------------------------")

    print("--------------------------------Normalizing constant reference data--------------")

    start_time = datetime.datetime.now()
    lssvm.normalize_data()
    time_passed = datetime.datetime.now() - start_time
    deltatime = datetime.timedelta(seconds=time_passed.total_seconds())
    print(f"Duration: {deltatime}")

    print("--------------------------------Normalizing constant reference data: done--------")

    #Make mmap files
    optimize_these_numbers = [0.5, 0.5]
    xtrain, ytrain = lssvm.x_train[:, :], lssvm.y_train[:, :]
    xvalid, yvalid = lssvm.x_valid[:, :], lssvm.y_valid[:, :]
    xtest, ytest = lssvm.x_test[:, :], lssvm.y_test[:, :]

    xtrain_shape, ytrain_shape = np.shape(xtrain), np.shape(ytrain)
    xvalid_shape, yvalid_shape = np.shape(xvalid), np.shape(yvalid)
    xtest_shape, ytest_shape = np.shape(xtest), np.shape(ytest)

    xtrain_name, ytrain_name = "mmap_xtrain", "mmap_ytrain"
    xvalid_name, yvalid_name = "mmap_xvalid", "mmap_yvalid"
    xtest_name, ytest_name = "mmap_xtest", "mmap_ytest"

    path_mmap_xtrain = os.path.realpath(rf"{base_path}\{xtrain_name}")
    path_mmap_xvalid = os.path.realpath(rf"{base_path}\{xvalid_name}")
    path_mmap_xtest = os.path.realpath(rf"{base_path}\{xtest_name}")
    path_mmap_ytrain = os.path.realpath(rf"{base_path}\{ytrain_name}")
    path_mmap_yvalid = os.path.realpath(rf"{base_path}\{yvalid_name}")
    path_mmap_ytest = os.path.realpath(rf"{base_path}\{ytest_name}")

    mmap_xtrain = np.memmap(path_mmap_xtrain, dtype="float64", shape=xtrain_shape, mode="w+")
    mmap_xvalid = np.memmap(path_mmap_xvalid, dtype="float64", shape=xvalid_shape, mode="w+")
    mmap_xtest = np.memmap(path_mmap_xtest, dtype="float64", shape=xtest_shape, mode="w+")

    mmap_ytrain = np.memmap(path_mmap_ytrain, dtype="float64", shape=ytrain_shape, mode="w+")
    mmap_yvalid = np.memmap(path_mmap_yvalid, dtype="float64", shape=yvalid_shape, mode="w+")
    mmap_ytest = np.memmap(path_mmap_ytest, dtype="float64", shape=ytest_shape, mode="w+")

    mmap_xtrain[:], mmap_ytrain[:] = xtrain[:], ytrain[:]
    mmap_xvalid[:], mmap_yvalid[:] = xvalid[:], yvalid[:]
    mmap_xtest[:], mmap_ytest[:] = xtest[:], ytest[:]

    del mmap_xtrain
    del mmap_xvalid
    del mmap_xtest
    del mmap_ytrain
    del mmap_yvalid
    del mmap_ytest

    param = [
        [path_mmap_xtrain, "float64", xtrain_shape],
        [path_mmap_ytrain, "float64", ytrain_shape],
        [path_mmap_xvalid, "float64", xvalid_shape],
        [path_mmap_yvalid, "float64", yvalid_shape],
        [path_mmap_xtest, "float64", xtest_shape],
        [path_mmap_ytest, "float64", ytest_shape],
    ]

    print("--------------------------------PSO run----------------------------------------
--")
    n_particles, n_iterations = 10, 50
    setup = setup(n_particles)
    start_time = datetime.datetime.now()
    pso = PSO(
        prediction_error,
        optimize_these_numbers,
        param,
        setup,
        number_of_particles=n_particles,
        max_iterations=n_iterations,
    )
    try:
        pso.optimize()
    except KeyboardInterrupt:
        print("sun")

    time_passed = datetime.datetime.now() - start_time
    deltatime = datetime.timedelta(seconds=time_passed.total_seconds())
    print(f"Duration: {deltatime}")
    columns = ["Count", "w_min", "w_max", "w", "C1_min", "C1_max", "C1", "C2_min"]
    columns += ["C2_max", "C2", "Iteration", "max_iteration", "Global train Error"]
    columns += [ "Global validation error", "Global test error", "gamma", "sigma"]
    df = pd.DataFrame(data=np.mat(pso.logs), columns=columns)
    filename = f"PSO_log_iter{n_iterations}_par{n_particles}_v13_rand_w03"
    path = os.path.realpath(f"{base_path}\{filename}.xlsx")
    with pd.ExcelWriter(path) as writer:
        df.to_excel(writer, sheet_name="PSO Logs", index=False)

    print("--------------------------------PSO run: done----------------------------------
--")

    print("--------------------------------LSSVM run--------------------------------------
--")
    start_time = datetime.datetime.now()
    lssvm.calculate()
    time_passed = datetime.datetime.now() - start_time
    deltatime = datetime.timedelta(seconds=time_passed.total_seconds())
    print(f"Duration: {deltatime}")
    print("--------------------------------LSSVM run: done--------------------------------
--")

    print("--------------------------------Training run-----------------------------------
--")
    start_time = datetime.datetime.now()
    lssvm.train()
    time_passed = datetime.datetime.now() - start_time
    deltatime = datetime.timedelta(seconds=time_passed.total_seconds())
    print(f"Duration: {deltatime}")
    print("--------------------------------Trainig run: done----------------------------------")

    print("--------------------------------Validation process-------------------------------")
    start_time = datetime.datetime.now()
    lssvm.validate()
    time_passed = datetime.datetime.now() - start_time
    deltatime = datetime.timedelta(seconds=time_passed.total_seconds())
    print(f"Duration: {deltatime}")
    print("--------------------------------Validation process: done-------------------------")

    print("--------------------------------Testing process----------------------------------")
    lssvm.x_train = lssvm.x_train[:,:]
    lssvm.y_train = lssvm.y_train[:,:]
    start_time = datetime.datetime.now()
    lssvm.check()
    time_passed = datetime.datetime.now() - start_time
    deltatime = datetime.timedelta(seconds=time_passed.total_seconds())
    print(f"Duration: {deltatime}")
    print("--------------------------------Testing process: done----------------------------")

    print("--------------------------------Exporting the results----------------------------")
    filenames_combined = {"train": f"0.7_train_results_v13_{gamma}_{sigma}","valid": f"0.15_valid_results_v13_{gamma}_{sigma}", "test": f"0.15_test_results_v13_{gamma}_{sigma}"}
    lssvm.export_after_lssvm_prediction(base_path, filenames_combined)
    filenames_split = {"train": f"split_0.7_train_results_v13_{gamma}_{sigma}", "valid": f"split_0.15_valid_results_v13_{gamma}_{sigma}", "test": f"split_0.15_test_results_v13_{gamma}_{sigma}"}
    start_time = datetime.datetime.now()
    lssvm.export_after_lssvm_prediction_as_sheets(base_path, filenames_split, Sheets)
    time_passed = datetime.datetime.now() - start_time
    deltatime = datetime.timedelta(seconds=time_passed.total_seconds())
    print(f"Duration: {deltatime}")
    print("--------------------------------Exporting the results: done----------------------")
    print("-------------------------------- Summary ----------------------------------------")
    time_passed_total = datetime.datetime.now() - start_time_original
    deltatime = datetime.timedelta(seconds=time_passed_total.total_seconds())
    print(f"Duration: {deltatime}")
    print("---------------------------------- End ------------------------------------------")
```

## References

Afzali, Shokufe, Nima Rezaei, and Sohrab Zendehboudi. 2018. 'A Comprehensive Review on Enhanced Oil Recovery by Water Alternating Gas (WAG) Injection'. *Fuel* 227: 218–46.

Alvarado, Vladimir et al. 2002. 'Selection of EOR/IOR Opportunities Based on Machine Learning'. *SPE 13th European Petroleum Conferece*: 11.

Baghban, Alireza et al. 2016. 'Phase Equilibrium Modelling of Natural Gas Hydrate Formation Conditions Using LSSVM Approach'. *Petroleum Science and Technology* 34(16): 1431–38.

Bourgeois, M., T. Joubert, and V.E. Dominguez. 2019. 'Analysis of 3-Phase Behavior in WAG Injections for Various Wettabilities'. 2019(1): 1–16.

Bozorg-Haddad, Omid, Mohammad Solgi, and Hugo A. Loáiciga. 2017. *Meta-Heuristic and Evolutionary Algorithms for Engineering Optimization*. Newark, UNITED STATES: John Wiley & Sons, Incorporated. http://ebookcentral.proquest.com/lib/uisbib/detail.action?docID=5015534.

Burkov, Andriy. 2019. '"All Models Are Wrong, but Some Are Useful." — George Box'. : 152.

Chamkalani, Ali et al. 2014. 'Integration of LSSVM Technique with PSO to Determine Asphaltene Deposition'. *Journal of Petroleum Science and Engineering* 124: 243–53.

Cheng, Guojian, Ruihua Guo, and Wenhai Wu. 2010. 'Petroleum Lithology Discrimination Based on PSO-LSSVM Classification Model'. In *2010 Second International Conference on Computer Modeling and Simulation*, Sanya, China: IEEE, 365–68. http://ieeexplore.ieee.org/document/5421448/ (February 4, 2020).

Christensen, R, E H Stenby', and A Skauge. 2001. 'Review of WAG Field Experience'. : 14.

Dalen, Vilgeir, Rune Instefjord, and Reidar Kristensen. 1995. 'A WAG Injection Pilot in the Lower Brent Formation at the Gullfaks Field'. *Geological Society, London, Special Publications* 84(1): 143–52.

Green, Don W., and G. Paul Willhite. 2018. *Enhanced Oil Recovery*. Second edition. Richardson, Texas, USA: Society of Petroleum Engineers.

Hou, Likun, Qingxin Yang, and Jinlong An. 2009. 'An Improved LSSVM Regression Algorithm'. In *2009 International Conference on Computational Intelligence and Natural Computing*, Wuhan, China: IEEE, 138–40. http://ieeexplore.ieee.org/document/5231009/ (April 1, 2020).

Larsen, J.A., and Arne Skauge. 1998. 'Methodology for Numerical Simulation With Cycle-Dependent Relative Permeabilities'. *SPE Journal* 3(02): 163–73.

Lary, David J., Amir H. Alavi, Amir H. Gandomi, and Annette L. Walker. 2016. 'Machine Learning in Geosciences and Remote Sensing'. *Geoscience Frontiers* 7(1): 3–10.

Mahzari, Pedram, and Mehran Sohrabi. 2016. 'An Improved Approach for Estimation of Flow and Hysteresis Parameters Applicable to WAG Experiments'. : 20.

Manrique, Eduardo Jose et al. 2010. ‘EOR: Current Status and Opportunities’. In *SPE Improved Oil Recovery Symposium*, Tulsa, Oklahoma, USA: Society of Petroleum Engineers. http://www.onepetro.org/doi/10.2118/130113-MS (February 4, 2020).

NPD. 2019. ‘Resource 2019 - Fields and Discoveries. Big Opportunities’. *Norwegian Petroleum Directorate*. https://www.npd.no/en/facts/publications/reports2/resource-report/resource-report-2019/fields/ (April 16, 2020).

Nygård, Jan Inge, and Pål Østebø Andersen. 2020. ‘Simulation of Immiscible Water-Alternating-Gas Injection in a Stratified Reservoir: Performance Characterization Using a New Dimensionless Number’. *SPE Journal*. http://www.onepetro.org/doi/10.2118/200479-PA (April 17, 2020).

O’Brien, Jeremy et al. 2016. ‘Maximizing Mature Field Production - A Novel Approach to Screening Mature Fields Revitalization Options’. In *SPE-180090-MS*, Vienna, Austria: Society of Petroleum Engineers, 11. https://doi.org/10.2118/180090-MS.

Razavi, Razieh et al. 2018. ‘Utilization of LSSVM Algorithm for Estimating Synthetic Natural Gas Density’. *Petroleum Science and Technology* 36(11): 807–12.

Sanchez, Nestor L. 1999. ‘Management of Water Alternating Gas (WAG) Injection Projects’. In *SPE-53714-MS*, SPE: Society of Petroleum Engineers, 8. https://doi.org/10.2118/53714-MS.

Sarah Kent, Justin Scheck. 2015. ‘Cash Crunch Clouds Future for Oil Firms: Spending on New Projects, Share Buybacks and Dividends Outstrips Cash Flow’. *Wall Street Journal*. https://www.wsj.com/articles/cash-crunch-clouds-future-for-oil-firms-1445816429?mod=wsj_nview_latest (April 17, 2020).

Skauge, A., and E. A. Berg. 1997. ‘Immiscible WAG Injection in the Fensfjord Formation of the Brage Oil Field’. https://www.earthdoc.org/content/papers/10.3997/2214-4609.201406788.

Skauge, Arne. 2003. ‘REVIEW OF WAG FIELD EXPERIENCE’. *1st International Conference and Exhibition Modern Challenges in Oil Recovery*: 11.

Spiteri, Elizabeth J., and Ruben Juanes. 2006. ‘Impact of Relative Permeability Hysteresis on the Numerical Simulation of WAG Injection’. *Journal of Petroleum Science and Engineering* 50(2): 115–39.

Stenmark, H., and P. O. Andfossen. 1995. ‘Snorre WAG Pilot - A Case Study’. https://www.earthdoc.org/content/papers/10.3997/2214-4609.201406924.

Suykens, Johan A. K., ed. 2002. *Least Squares Support Vector Machines*. River Edge, NJ: World Scientific.

Talabi, Oluwole A., Jaime E. Moreno, Ruppa K. Malhotra, and Yunlong Liu. 2019. ‘Practical Upscaling of WAG Hysteresis Parameters from Core to Full-Field Scale Part II’. In *SPE/IATMI Asia Pacific Oil & Gas Conference and Exhibition*, Bali, Indonesia: Society of Petroleum Engineers. http://www.onepetro.org/doi/10.2118/196286-MS (June 12, 2020).

Theobald, Oliver. 2017. ‘Machine Learning for Absolute Beginners’. : 52.

Vapnik, Vladimir N. 1995. *The Nature of Statistical Learning Theory*. Berlin, Heidelberg: Springer-Verlag.