



doi:10.5281/zenodo.19337108

# Critical Review of Several Methods Used for The Optimization of Subsurface Safety Valve in Wells

<sup>1</sup>Eli Digieneni Goodluck; <sup>2</sup>Sunday Igbani & <sup>3</sup>Zekieni Robert Yelebe

<sup>1,2</sup>Department of Petroleum Engineering, Niger Delta University,  
Wilberforce Island, Bayelsa State, Nigeria

<sup>3</sup>Department of Chemical Engineering,  
Niger Delta University, Wilberforce Island, Bayelsa State, Nigeria

## ABSTRACT

Subsurface Safety Valves (SSSVs) are vital safety devices in oil and gas wells, functioning as fail-safe mechanisms to safeguard human life, the environment, and production equipment. Their performance, however, is typically limited by obstacles such as scale deposition, erosion, sand generation, and high-pressure high-temperature (HPHT) conditions. To increase efficiency and dependability, several optimization approaches have been utilized. This review critically examines traditional and modern approaches, including Lagrange Multiplier, Linear Programming, Artificial Neural Networks (ANN), Support Vector Machines (SVM), Bayesian Optimization, Particle Swarm Optimization (PSO), Reliability-Based Design Optimization (RBDO), and Data-Driven Optimization. Each approach is assessed based on its underlying concepts, mathematical formulation, strengths, limits, and application to SSSV processes. Comparative study indicates variations in oil production outputs and average daily production, showing the appropriateness of each technology under varied reservoir and operational conditions. The paper underlines that although conventional mathematical techniques give analytical clarity,

**Keywords:** Subsurface Safety Valve (SSSV), Python programming, machine learning, optimization techniques, predictive modeling, oilfield automation, digital transformation.

## 1.1 INTRODUCTION

The safe and efficient operation of oil and gas wells depends heavily on the performance of Subsurface Safety Valves (SSSVs) (Abdiansah & Wardoyo, 2015), which work as key barriers against uncontrolled flow, defending people, the environment, and infrastructure (Braun & Neuffer, 2018). However, their dependability is often tested by complicated well conditions, including high pressures, high temperatures, scale deposition, erosion, and multiphase flow dynamics (Aydin, 2020). To ensure optimal performance and reduce risks, researchers and industry practitioners have developed and applied a variety of optimization methods aimed at improving the design, operation, and predictive maintenance of SSSVs (Anis & Khan, 2025).

Optimization in this context refers to the systematic application of mathematical, computational (Alam, 2021), and data-driven methods to enhance valve reliability, extend operating lifespan, and maximize production efficiency (Anis & Khan, 2025). Traditional methods such as Lagrange Multiplier and Linear Programming provide analytical clarity and structured solutions to constrained problems (Aydin, 2020), while more advanced approaches—including Artificial Neural Networks (ANN), Support Vector Machines (SVM), Bayesian Optimization, Particle Swarm Optimization (PSO), Reliability-Based Design Optimization (RBDO), and Data-Driven Optimization—address the nonlinearities, uncertainties, and real-time demands of modern oilfield operations (Gad, 2022). This review critically examines these methods, highlighting their principles (Kaveh & Zaeerza, 2022), mathematical roots, strengths, and limits. Furthermore, it compares their usefulness in improving SSSVs under different reservoir conditions, giving insights into their applicability in real-world well management (Shariat, 2018). By combining traditional optimization theory with advanced machine learning and data-driven methods, this work provides a complete view on current trends and future directions for improving SSSV performance and total well safety.

### 2.1 Optimization by Lagrange Multiplier

One traditional optimization method for addressing constraint-related problems is the Lagrange Multiplier method. Joseph-Louis Lagrange first used it in the 18th century as a component of variational calculus. By adding extra variables, called Lagrange multipliers, that put the constraints into the objective function, this method makes a constrained Optimization problem into a system of equations. The main idea is that (Vanderbei, 2015) the system can be solved analytically at the optimal point because the slopes of the constraint function and the goal function are parallel. In areas like physics, engineering, and economics where Optimization under stringent conditions is common, this method is especially helpful.

The Lagrange method's mathematical elegance and usefulness in handling equality limits are generally recognized by scholars. For instance, (Ji-Huan, 2016) and (Li, 2018) stress how useful it is for constrained Optimization and nonlinear programming, especially when the problem is well-behaved and differentiable. Moreover, it has been widely employed in machine learning, especially in the derivation of the dual formulation of support vector machines (hu & Kang, 2017). Critics point out that the usual Lagrange method has trouble with inequality constraints and can break down if the constraint set is not convex. As a result, more general methods have been created, such as the Karush-Kuhn-Tucker (KKT) conditions, which increase the applicability of Lagrange multipliers to problems with inequality constraints.

Regarding the computational usefulness of Lagrange multipliers, there is also differing material. Some sources, such as (Huang, 2018), argue that the method is simple for low-dimensional problems involving smooth functions, but others point out that it becomes less tractable in high-dimensional or non-linear cases, where it becomes difficult to answer the resulting system of equations. Because they are reliable and effective at approximating optimal solutions, numerical Optimization techniques like sequential quadratic programming and interior point methods are suggested in these situations (Abdiansah & Wardoyo, 2015). This difference shows how important problem structure is in deciding whether to use analytical or numerical methods.

In summary, the Lagrange Multiplier method remains a foundational idea in optimization theory, especially for problems with equality constraints. Its theoretical elegance and analytical clarity are its best points, but in high-dimensional, non-convex, or real-world situations, it is restricted. Building on its roots, current Optimization frameworks have increased its application by adding wider conditions and numerical solvers. However, the basic ideas of the Lagrange method still have an effect on contemporary Optimization techniques in many scientific areas.

## 2.2 Modelling of Lagrange Multiplier Method

The Lagrange Multiplier method is used to find the maximum or minimum of a function  $f(x_1, x_2, \dots, x_n)$  subject to one or more equality constraints.

### 2. Standard Mathematical Formulation

Let:

$f(x_1, x_2, \dots, x_n)$  be the objective function,

$g(x_1, x_2, \dots, x_n) = 0$  be the constraint function.

We want to:

**Optimize (maximize or minimize)**  $f(x_1, x_2, \dots, x_n)$  **subject to**  $g(x_1, x_2, \dots, x_n) = 0$

### 3. Formulating the Lagrangian

We define the Lagrangian function:

$$\mathcal{L}(x_1, x_2, \dots, x_n, \lambda) = f(x_1, x_2, \dots, x_n) - \lambda \cdot g(x_1, x_2, \dots, x_n)$$

Where:

$\lambda$  is the Lagrange multiplier (a scalar variable).

### 4. Necessary Conditions (Karush-Kuhn-Tucker for equality)

To find extrema, we take partial derivatives of the Lagrangian and set them to zero:

$$\frac{\partial \mathcal{L}}{\partial x_1} = 0, \quad \frac{\partial \mathcal{L}}{\partial x_2} = 0, \quad \dots, \quad \frac{\partial \mathcal{L}}{\partial x_n} = 0, \quad \frac{\partial \mathcal{L}}{\partial \lambda} = -g(x_1, x_2, \dots, x_n) = 0$$

So the full system becomes:

$$\nabla f(x) = \lambda \nabla g(x) \quad \text{and} \quad g(x) = 0$$

This gives us a system of  $n + 1$  equations with  $n + 1$  unknowns.

### 5. Example with Two Variables

Let:

$$\text{Objective : } f(x, y) = x^2 + y^2$$

$$\text{Constraint: } g(x, y) = x + y - 1 = 0$$

Lagrangian:

$$\mathcal{L}(x, y, \lambda) = x^2 + y^2 - \lambda(x + y - 1)$$

Take partial derivatives:

$$\frac{\partial \mathcal{L}}{\partial x} = 2x - \lambda = 0$$

$$\frac{\partial \mathcal{L}}{\partial y} = 2y - \lambda = 0$$

$$\frac{\partial \mathcal{L}}{\partial \lambda} = -(x + y - 1) = 0$$

Solving:

$$2x = \lambda, \text{ so } x = \lambda/2$$

$$2y = \lambda, \text{ so } y = \lambda/2$$

$$x + y = 1 \Rightarrow \lambda/2 + \lambda/2 = 1 \Rightarrow \lambda = 1$$

Thus:

$$x = y = 1/2$$

### 6. Multiple Constraints (Extended Form)

If there are multiple constraints:

Let:

$$g_i(x_1, \dots, x_n) = 0 \text{ for } i = 1, \dots, m$$

Then the Lagrangian becomes:

$$\mathcal{L}(x_1, \dots, x_n, \lambda_1, \dots, \lambda_m) = f(x_1, \dots, x_n) - \sum_{i=1}^m \lambda_i g_i(x_1, \dots, x_n)$$

The condition becomes:

$$\nabla f(x) = \sum_{i=1}^m \lambda_i \nabla g_i(x), \text{ and } g_i(x) = 0 \text{ for all } i$$

Summary

Component	Description
Objective	Maximize or minimize $f(x)$
Constraint(s)	Subject to $g_i(x) = 0$
Lagrangian Function	$\mathcal{L}(x, \lambda) = f(x) - \sum \lambda_i g_i(x)$
Necessary Condition	$\nabla f = \sum \lambda_i \nabla g_i, g_i(x) = 0$

### 3.1 Linear Programming Optimization

One important Optimization method used in decision-making processes in a variety of fields and industries is linear programming (LP). Since its official introduction by George Dantzig in 1947 to address planning issues in the U.S. Air Force, it has grown to become one of the most studied and used areas of mathematical programming (Dantzig, 2016). Fundamentally, LP aims to maximize or minimize a linear objective function while taking into account a set of linear equality and inequality constraints. According to (Braun, 2018) and (Lodi & Prouvost, 2021), LP is a technique that ensures optimal results under particular constraints by providing mathematical clarity to resource allocation problems. Variables, constraints, and a linear objective function make up the standard form of an LP model, which creates a feasible region that contains the best solution.

LP has been widely used over the years in a variety of industries, including energy systems, manufacturing, transportation, agriculture, finance, and military logistics. In agricultural planning, for instance, LP is utilised to identify the most lucrative crop combination given labour, capital, and land constraints (Vadlamani, 2020). Similar to this, LP models help manufacturing companies meet customer demand while cutting costs through inventory management and production scheduling. LP offers both a theoretical framework and useful tools for business and engineering decision-making, claim (Dougherty, 2021). Dantzig's Simplex Method has proven to be a dependable algorithm for LP problem solving, and as computing power has increased, so too has the efficiency of solving large-scale LP problems.

Notwithstanding these advantages, a number of writers have noted LP's intrinsic drawbacks. The assumption of linearity, which states that all relationships between variables must be linear—a requirement that is rarely met in real-world systems—is one of the main criticisms (Lodi & Prouvost, 2021). For example, production functions, demand curves, and cost structures are frequently nonlinear, which reduces the realism of LP models when used unaltered. Additionally, LP makes the problematic assumption that model parameters, like fixed coefficients and constraint values, are certain, which is problematic in dynamic and uncertain environments. According to (Darvishi & Liu, 2021), LP's deterministic character renders it unsuitable for issues involving variability or randomness, like shifting market prices or ambiguous demand.

To address these issues, however, some researchers have suggested LP extensions and changes. For example, Integer Linear Programming (ILP) limits some or all variables to integer values, which is crucial in situations where decisions involving whole units, like the number of trucks or employees, must be made (Zhao & Jin, 2018). By permitting imprecision in parameters, fuzzy linear programming (FLP) introduces uncertainty into linear programming (LP) models, resulting in more adaptable and practical solutions (Chang & Jianxiao, 2017). As an alternative to a single Optimization goal, goal programming permits several, occasionally conflicting objectives. These expanded approaches make LP more suited to contemporary decision-making contexts by bridging the gap between its theoretical underpinnings and real-world applications in complex systems.

Furthermore, the development and use of LP has greatly benefited from software advancements. Extremely large LP problems with millions of variables and constraints can now be solved in comparatively short computation times thanks to modern solvers like CPLEX, Gurobi, LINDO, and MATLAB's Optimization Toolbox (Anwar & Yuan, 2019). The usability of LP is further increased by these tools' intuitive interfaces and support for integration with programming languages like Python, R, and Java. Clean, accurate, and consistent data is still necessary, though. Although LP models are mathematically sound, the quality of their outputs depends on the quality of their inputs, as cautioned by (Li & Zhao, 2018)). No matter how sophisticated the software is, inaccurate or incomplete data can produce deceptive results and bad decisions.

In conclusion, linear programming is still a fundamental method in Optimization, praised for its effectiveness, logical structure, and wide range of applications. Although it has been incredibly useful in many real-world situations, it is not without flaws. Its application in complex, real-world systems with nonlinearity and uncertainty is restricted by the assumptions of linearity and certainty. However, continued research has produced improved versions and hybrid models that address LP's shortcomings while enhancing its advantages. Even though LP might not always be enough on its own, the field is still developing, showing that it is still a crucial component of Optimization theory and practice as a whole.

### 3.2 Definition of Linear Programming (LP)

Linear Programming is a mathematical method used to determine the best possible outcome in a given mathematical model, where the objective function and all constraints are linear.

1. General Structure of a Linear Programming Model

An LP problem can be expressed in the standard form as:

Objective Function:

Maximize or Minimize:

$$Z = c_1x_1 + c_2x_2 + \dots + c_nx_n = \sum_{j=1}^n c_jx_j$$

Subject to Constraints:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n &\leq b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n &\leq b_2 \\ &\vdots \\ a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n &\leq b_m \end{aligned}$$

Non-Negativity Constraints:

$$x_j \geq 0 \quad \text{for } j = 1, 2, \dots, n$$

Where:

**Z**: The objective function (to be maximized or minimized)

**x<sub>j</sub>**: Decision variables (unknowns to be solved)

**c<sub>j</sub>**: Coefficients of the objective function

**a<sub>ij</sub>**: Coefficients of the constraint equations

**b<sub>i</sub>**: Right-hand side values of the constraints

**m**: Number of constraints

**n**: Number of decision variables

### 3.3 Mathematical Model Components

Component	Description
Decision Variables $x_1, x_2, \dots, x_n$	Unknowns to be determined
Objective Function $Z = \sum c_j x_j$	A function to optimize (maximize profit, minimize cost, etc.)
Constraints $\sum a_{ij} x_j \leq b_i$	Limitations on resources
Feasible Region	Set of all points $(x_1, x_2, \dots, x_n)$ satisfying the constraints
Optimal Solution	A point in the feasible region that gives the best value of <b>Z</b>

### 3.4 Assumptions in LP Modeling

Proportionality: Change in objective or constraints is directly proportional to the variables.

Additivity: Total of all activities is the sum of individual activities.

Certainty: All coefficients are known with certainty and remain constant.

Non-negativity: Negative values of variables are not allowed (e.g., you can't produce -5 units).

#### 4.1 Optimization by Artificial Neural Network

Computational models known as artificial neural networks (ANNs) are modelled after the architecture and operations of the brain's networked neurones. The conceptual basis was established by McCulloch and Pitts' early work in the 1940s and 1950s, which formalised a "neurone" as a binary threshold unit (Aydin, 2020). The backpropagation algorithm rekindled interest in multi-layer perceptrons in the 1980s (Hannan, & Mohamed, 2021). Since then, as a result of increasing processing power and a wealth of data, ANNs have developed from tiny, shallow networks to deep architectures with millions of parameters. The Universal Approximation Theorem, which asserts that any continuous function on a compact domain can be approximated by even a single hidden layer network with nonlinear activation, is a significant similarity in the literature. While (Thames & Schaefer, 2017) offered an alternate formulation for more general classes of functions, (Janáková & Sujová, 2024) demonstrated this for sigmoidal activations. On network depth versus width, however, researchers disagree. More recent research (Dolgui, 2021) demonstrates that depth can achieve the same accuracy with exponentially fewer neurones, at the expense of more difficult

Optimization. Earlier work preferred very wide, shallow networks to ensure approximation. This contrast draws attention to a trade-off between training complexity and representational efficiency.

Convergence and contention are also observed in ANN training algorithms. (Snoek & Rippel, 2015) defended the traditional stochastic gradient descent (SGD) with plain backpropagation due to its theoretical guarantees and ease of use under convexity assumptions. However, non-convexity in deep networks gave rise to variations such as momentum, Adam, and RMSprop, which in practice accelerate convergence (Khan, & Byun, 2020). Recent theoretical work (Hesser & Markert, 2019) questions Adam's long-term generalisation when compared to vanilla SGD, despite practitioners reporting faster and more dependable training. As a result, there is a definite conflict between optimiser behaviour theory and actual performance.

ANNs are used in many different fields, and comparisons show both commonalities and differences. Convolutional neural networks (CNNs) have achieved human-level image recognition in computer vision by utilising spatial hierarchies (Santhi, 2023). Sequence modelling is used by recurrent neural networks (RNNs) and, more recently, transformers in natural language processing to perform well in translation and summarisation (Praça & Gama, 2021). In contrast to language models, which struggle to capture long-range dependencies and necessitate specific tokenisation techniques, vision models frequently require large labelled datasets and complex convolutional computations. This demonstrates how network design and training are influenced by domain structure. ANNs have drawbacks despite their strength, including interpretability issues, overfitting, and susceptibility to adversarial examples. Adversarial training attempts to fortify networks against malevolent inputs, while regularisation methods (weight decay, dropout) and data augmentation reduce overfitting. Saliency maps and concept attribution are two ways that explainable AI research aims to unlock the "black box," but there isn't agreement on interpretation techniques that are consistently accurate. Though they have their own complexity and training challenges, emerging solutions like graph neural networks and capsule networks offer richer internal representations.

In conclusion, ANNs have evolved from basic threshold units to the deep architectures that serve as the foundation for contemporary AI. Although the power of depth and foundational theorems are widely accepted, discussions about training optimisers, domain-specific adaptations, and ideal network shapes are still ongoing. Continuous research aims to strike a balance between interpretability, efficiency, and accuracy, pointing to next-generation neural models that are reliable and strong.

#### 4.2 Mathematical Model of Artificial Neural Network (ANN)

An Artificial Neural Network (ANN) is a computational model inspired by biological neural systems. It is structured as layers of interconnected nodes (neurons), and each node performs a mathematical operation on its inputs to produce an output.

Below is a step-by-step breakdown of the complete mathematical model for a feedforward neural network (Multilayer Perceptron), including input, weights, activations, loss functions, and backpropagation.

##### 1. Structure of a Feedforward ANN

Let:

$L$  = number of layers (excluding input),

$x \in \mathbb{R}^d$  = input vector,

$W^{[l]}, b^{[l]}$  = weight matrix and bias vector at layer  $l$ ,

$z^{[l]}, a^{[l]}$  = pre-activation and activation of layer  $l$ ,

$\sigma^{[l]}(\cdot)$  = activation function at layer  $l$ .

##### 2. Forward Propagation

Input Layer:

$$a^{[0]} = x$$

Hidden & Output Layers ( $l = 1, 2, \dots, L$ ):

$$z^{[l]} = W^{[l]}a^{[l-1]} + b^{[l]} \quad (\text{linear transformation})$$

$$a^{[l]} = \sigma^{[l]}(z^{[l]}) \quad (\text{non-linear activation})$$

Final output:

$$\hat{y} = a^{[L]}$$

##### 3. Cost or Loss Function

For regression:

$$\mathcal{L}(\hat{y}, y) = \frac{1}{2} \|\hat{y} - y\|^2$$

For binary classification (using sigmoid at output):

$$\mathcal{L}(\hat{y}, y) = -[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})]$$

For multiclass classification (softmax + cross-entropy):

$$\mathcal{L}(\hat{y}, y) = -\sum_{j=1}^K y_j \log(\hat{y}_j)$$

Where:

$y$  = true label,

$\hat{y}$  = network prediction.

#### 4. Training: Backpropagation Algorithm

We compute gradients of the loss w.r.t. weights and biases using chain rule:

Output Layer Gradient:

$$\delta^{[L]} = \nabla_a \mathcal{L} \odot \sigma'(\mathbf{z}^{[L]})$$

Hidden Layers ( $l = L - 1, L - 2, \dots, 1$ ):

$$\delta^{[l]} = ((W^{[l+1]})^T \delta^{[l+1]}) \odot \sigma'(\mathbf{z}^{[l]})$$

Parameter Updates:

$$\frac{\partial \mathcal{L}}{\partial W^{[l]}} = \delta^{[l]} (\mathbf{a}^{[l-1]})^T, \quad \frac{\partial \mathcal{L}}{\partial b^{[l]}} = \delta^{[l]}$$

Using gradient descent:

$$W^{[l]} := W^{[l]} - \eta \frac{\partial \mathcal{L}}{\partial W^{[l]}}, \quad b^{[l]} := b^{[l]} - \eta \frac{\partial \mathcal{L}}{\partial b^{[l]}}$$

Where:

$\eta$  = learning rate,

$\delta^{[l]}$  = error at layer  $l$ ,

$\odot$  = element-wise (Hadamard) product.

#### 5. Full ANN Optimization Problem

For a training dataset  $\{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^N$ , the overall optimization is:

$$\min_{\{W^{[l]}, b^{[l]}\}} \frac{1}{N} \sum_{i=1}^N \mathcal{L}(\mathbf{a}^{[L(i)]}, \mathbf{y}^{(i)})$$

Where:

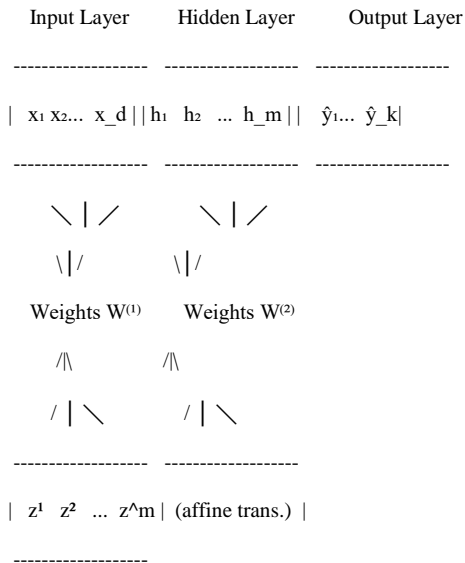
$\mathbf{a}^{[L(i)]} = \hat{\mathbf{y}}^{(i)}$  is the output from forward pass for sample  $i$ ,

$\mathcal{L}(\cdot, \cdot)$  is the loss function.

#### 6. Common Activation Functions

Function Name	Formula	Derivative
Sigmoid	$\sigma(z) = \frac{1}{1 + e^{-z}}$	$\sigma'(z) = \sigma(z)(1 - \sigma(z))$
Tanh	$\tanh(z)$	$1 - \tanh^2(z)$
ReLU	$\max(0, z)$	1 if $z > 0$ ; 0 otherwise
Softmax	$\hat{y}_j = \frac{e^{z_j}}{\sum_k e^{z_k}}$	Special (used in output layer for multiclass)

Schematic diagram of an artificial neural network



5.1 Support Vector Machine (SVM)

Because of its resilience when working with high-dimensional data, the Support Vector Machine (SVM) supervised learning algorithm is frequently used for classification and regression tasks. The fundamental idea behind SVM, which was first presented by Vapnik and Cortes (Huang, 2018), is to identify a hyperplane that divides data into discrete classes with the greatest margin feasible. Originally created for linearly separable data, the SVM model was later modified to use kernel functions to handle non-linearly separable data. The flexibility of SVM is greatly increased by these kernels, which convert data into higher-dimensional spaces where linear separation is feasible (Pisner & Schnyer, 2020). SVM is one of the most popular models in the machine learning community because of its effectiveness in handling both linear and non-linear classification problems.

According to the majority of research, SVM performs exceptionally well, especially when there are few datasets and distinct class boundaries. According to research by (Abdiansah & Wardoyo, 2015), SVM performs better in text classification and image recognition tasks than other models like k-nearest neighbours and decision trees. In a similar vein, (Doshi & Vakharia, 2023) used its robustness and strong generalization ability to show off its superiority in large-scale text classification. Notwithstanding its advantages, SVM has drawn criticism for having a comparatively high computational cost, particularly when dealing with large datasets where training time increases noticeably. On the other hand, models such as random forests and logistic regression might scale better and train more quickly with large amounts of data. When choosing between SVM and other models, practitioners are frequently guided by this difference in computational efficiency and scalability.

The literature is also divided on the interpretability and tuning simplicity of SVM. According to some researchers, SVMs are more difficult to interpret in crucial applications like healthcare or finance because they are less transparent than models like decision trees (Awad & Khanna, 2015). However, proponents of SVM argue that the model provides a type of sparsity and clarity in decision-making due to its reliance on support vectors, which are the most important data points close to the decision boundary (Battineni, 2019). Additionally, the kernel selection and hyperparameters, including the penalty parameter (C) and kernel-specific parameters (e.g., gamma for RBF kernels), have an impact on SVM performance. Although this sensitivity offers flexibility in fine-tuning, it also necessitates greater skill and computational work when developing the model.

In conclusion, SVM's strong theoretical underpinnings, capacity to represent intricate decision boundaries, and consistent performance across a variety of domains make it a fundamental model in machine learning. Although its accuracy and resilience in high-dimensional spaces are frequently commended, its interpretability and scalability issues are frequently criticised. SVM is a model best suited for particular, well-understood problem settings rather than a general solution, as evidenced by the literature's balance between acknowledging its practical challenges and recognising its strengths.

5.2 Mathematical Model of Support Vector Machine (SVM)

1. Problem Setup (Linear SVM for Binary Classification)

Given a training dataset:

$$\{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}, \quad x_i \in \mathbb{R}^d, \quad y_i \in \{-1, +1\}$$

Our goal is to find a hyperplane:

$$w^T x + b = 0$$

That separates the data such that:

$$\begin{cases} w^T x_i + b \geq +1 & \text{if } y_i = +1 \\ w^T x_i + b \leq -1 & \text{if } y_i = -1 \end{cases} \quad \text{or compactly: } y_i(w^T x_i + b) \geq 1 \quad \forall i$$

2. Objective Function (Hard Margin SVM)

To maximize the margin, we minimize the norm of the weight vector:

$$\min_{w,b} \frac{1}{2} \|w\|^2 \quad \text{subject to } y_i(w^T x_i + b) \geq 1$$

This is a convex quadratic optimization problem with linear constraints.

3. Soft Margin SVM (for Non-linearly Separable Data)

We introduce slack variables  $\xi_i \geq 0$  to allow for misclassification:

$$\min_{w,b,\xi} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^n \xi_i \quad \text{subject to} \quad y_i(w^T x_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0$$

$C > 0$  is a regularization parameter that balances the trade-off between maximizing margin and minimizing classification error.

4. Dual Formulation (for Kernel SVM)

To handle non-linear decision boundaries, we solve the dual using Lagrange multipliers:

$$\max_{\alpha} \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle \quad \text{subject to} \quad \sum_{i=1}^n \alpha_i y_i = 0, \quad 0 \leq \alpha_i \leq C$$

Only data points with  $\alpha_i > 0$  become support vectors.

5. Kernel Trick (Non-linear SVM)

Replace the inner product  $\langle x_i, x_j \rangle$  with a kernel function:

$$K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$$

Common kernels:

Linear:  $K(x, x') = x^T x'$

Polynomial:  $K(x, x') = (x^T x' + c)^d$

Gaussian (RBF):  $K(x, x') = \exp\left(-\frac{\|x-x'\|^2}{2\sigma^2}\right)$

6. Decision Function

Once the optimal  $\alpha^*$  are found, the classifier is:

$$f(x) = \text{sign}\left(\sum_{i=1}^n \alpha_i y_i K(x_i, x) + b\right)$$

Where  $b$  is calculated using support vectors.

7. Summary of SVM Optimization Steps

Primal (Soft Margin):

$$\min_{w,b,\xi} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^n \xi_i, \quad \text{s.t.} \quad y_i(w^T x_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0$$

Dual (for Kernel SVM):

$$\max_{\alpha} \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j K(x_i, x_j), \quad \text{s.t.} \quad \sum_i \alpha_i y_i = 0, \quad 0 \leq \alpha_i \leq C$$

Decision Rule:

$$f(x) = \text{sign}\left(\sum_i \alpha_i y_i K(x_i, x) + b\right)$$

6.1 Bayesian Optimization

A sophisticated Optimization method called Bayesian Optimization (BO) is especially well-suited for optimising black-box, costly-to-evaluate functions. It has become more well-known in domains like robotics, machine learning, engineering design, and hyperparameter tuning. In order to predict the behaviour of the objective function and direct sampling, Bayesian Optimization builds a probabilistic surrogate model, typically a Gaussian Process (GP), according to (Frazier, 2018). This contrasts with conventional Optimization techniques that assume smoothness or require derivative information, such as gradient descent. Furthermore, Snoek, Larochelle, and (Adams, 2012) stress that BO is better for issues like deep learning model training or physical experiments where function evaluations are expensive. Using acquisition functions like Upper Confidence Bound (UCB), Probability of Improvement (PI), and Expected Improvement (EI) to balance exploration and exploitation is a fundamental aspect of Bayesian Optimization (Frazier, 2018). By measuring the anticipated gain of sampling that point, these acquisition functions identify the subsequent point to be evaluated. (Hennig & Schuler, 2016) contend that UCB provides superior exploration in high-dimensional

settings, albeit at the expense of slower convergence, despite the fact that many studies advocate for the use of EI due to its stability and speed of convergence (Jin & Schmitt, 2017). In BO, this poses a fundamental trade-off between expanding exploration of uncertain areas and accelerating exploitation of known good regions.

Both agreement and disagreement in the literature are reflected in BO's selection of surrogate models. Because of their closed-form posterior and uncertainty quantification, Gaussian processes are the most widely used (Swersky & Wang, 2015). However, for improved scalability and flexibility in modelling non-stationary functions, alternatives like Bayesian Neural Networks (Poloczek & Wilson, 2017) and Random Forests (Hutter, 2011) have been investigated. Although GPs are commended for their robustness and interpretability in low-dimensional problems, they have scalability issues in high dimensions and with big datasets. This has prompted some researchers to use ensembles or tree-structured Parzen estimators (Springenberg, 2016), indicating a move towards more adaptable but possibly less theoretically based models.

Bayesian Optimization has drawbacks despite its benefits. The computational cost of fitting and updating the surrogate model is a significant drawback, particularly as the number of data points increases (Balandat & Karrer, 2020). Furthermore, even though BO works well for global Optimization, sparse or poorly distributed initial data may cause it to converge slowly. However, some authors contend that because BO purposefully avoids needless assessments in unpromising areas, this weakness also serves as a strength (Frazier, 2018). This disparity highlights a persistent conflict in BO literature between sample efficiency and model complexity.

Application-wise, Bayesian Optimization has demonstrated remarkable promise in automatic machine learning (AutoML), particularly for hyperparameter tuning (Snoek & Rippel, 2015). It is perfect for fine-tuning models such as Support Vector Machines and Neural Networks because of its capacity to traverse intricate, high-dimensional, and costly search spaces. However, in parallel computing environments, where multiple evaluations can be conducted concurrently, simpler techniques like grid search or random search can outperform BO, according to other studies like those by (Wu & Chen, 2019). The sequential nature of classic BO, which can act as a bottleneck when quick decisions are required, is the source of this contradiction.

To sum up, Bayesian Optimization is a strong and moral method for black-box Optimization. It is useful for costly function evaluations because of its strengths in probabilistic modelling and sample efficiency. It is not always the best option, though, especially in situations that are high-dimensional or parallelised. There are still disagreements regarding surrogate model selection, acquisition function performance, and scalability, despite the fact that a large portion of the literature praises BO's harmony between exploration and exploitation. As more research is done, hybrid strategies and improved surrogate modelling might provide ways to get around these restrictions and increase the practicality of BO.

**6.2 Overview of Bayesian Optimization**

Bayesian Optimization is a global optimization strategy designed to find the maximum or minimum of an unknown, expensive, and black-box function. Unlike traditional optimization methods, BO does not require the gradient of the objective function. Instead, it uses a probabilistic surrogate model to approximate the objective and an acquisition function to decide where to evaluate next.

**1. Problem Definition**

Let:

$$f: \mathcal{X} \rightarrow \mathbb{R}$$

be an unknown objective function we want to maximize (or minimize), where  $\mathcal{X} \subset \mathbb{R}^d$ . Evaluating  $f(x)$  is expensive, so we want to find:

$$x^* = \underset{x \in \mathcal{X}}{\operatorname{argmax}} f(x)$$

but with as few evaluations of  $f$  as possible.

**2. Step-by-Step Mathematical Model**

Step 1: Use a Prior over Functions

Assume  $f(x) \sim \mathcal{GP}(\mu(x), k(x, x'))$

$\mu(x)$ : Mean function (often initialized to zero)

$k(x, x')$ : Covariance or kernel function (e.g., RBF, Matern)

This defines a Gaussian Process (GP) prior:

$$f(x) \sim \mathcal{GP}(\mathbf{0}, k(x, x'))$$

Step 2: Collect Initial Data

Evaluate the function at a few initial points:

$$\mathcal{D}_n = \{(x_1, y_1), \dots, (x_n, y_n)\} \text{ where } y_i = f(x_i) + \epsilon_i$$

$$\epsilon_i \sim \mathcal{N}(\mathbf{0}, \sigma^2) \text{ (Gaussian noise)}$$

Step 3: Posterior Inference using GP

Given observations  $\mathcal{D}_n$ , the GP gives the posterior distribution at a new point  $x$ :

$$f(x) | \mathcal{D}_n \sim \mathcal{N}(\mu_n(x), \sigma_n^2(x))$$

Where:

Posterior mean:

$$\mu_n(x) = k(x)^T (K + \sigma^2 I)^{-1} y$$

Posterior variance:

$$\sigma_n^2(x) = k(x, x) - k(x)^T (K + \sigma^2 I)^{-1} k(x)$$

Here:

$K$  is the  $n \times n$  kernel matrix:  $[k(x_i, x_j)]$

$$k(x) = [k(x_1, x), \dots, k(x_n, x)]^T$$

$$y = [y_1, \dots, y_n]^T$$

Step 4: Acquisition Function (Select Next Point)

Use an acquisition function  $\alpha(x; \mathcal{D}_n)$  to determine the next best point to evaluate:

$$x_{n+1} = \underset{x \in X}{\operatorname{argmax}} \alpha(x; \mathcal{D}_n)$$

Common acquisition functions:

a. Expected Improvement (EI):

$$EI(x) = \mathbb{E}[\max(0, f(x) - f(x^*))] = (\mu_n(x) - f(x^*))\Phi(z) + \sigma_n(x)\phi(z)$$

Where:

$f(x^*)$ : Best observed value

$$z = \frac{\mu_n(x) - f(x^*)}{\sigma_n(x)}$$

$\Phi$ : CDF and  $\phi$ : PDF of standard normal distribution

b. Probability of Improvement (PI):

$$PI(x) = \Phi\left(\frac{\mu_n(x) - f(x^*) - \xi}{\sigma_n(x)}\right)$$

c. Upper Confidence Bound (UCB):

$$UCB(x) = \mu_n(x) + \kappa \cdot \sigma_n(x) \quad (\text{maximize for exploration})$$

Where  $\kappa$  is a tunable parameter.

Step 5: Evaluate and Update

Evaluate  $f(x_{n+1})$

Update the dataset:

$$\mathcal{D}_{n+1} = \mathcal{D}_n \cup \{(x_{n+1}, f(x_{n+1}))\}$$

Refit the GP using the updated data

Repeat until budget or stopping condition is met

5. Summary of Mathematical Model

Bayesian Optimization can be summarized as:

**Initialize:**  $\mathcal{D}_n = \{(x_i, f(x_i))\}_{i=1}^n$

**Fit GP:**  $f(x) \sim \mathcal{GP}(\mu_n(x), \sigma_n^2(x))$

**Select**  $x_{n+1} = \underset{x \in X}{\operatorname{argmax}} \alpha(x; \mathcal{D}_n)$

**Evaluate:**  $y_{n+1} = f(x_{n+1})$

**Update:**  $\mathcal{D}_{n+1} = \mathcal{D}_n \cup \{(x_{n+1}, y_{n+1})\}$

**Repeat until stopping condition**

Optional: Example Kernel Function

Most common kernel is the Radial Basis Function (RBF):

$$k(x, x') = \exp\left(-\frac{1}{2l^2} \|x - x'\|^2\right)$$

Where  $l$  is the length scale.

### 7.1 Literature Review on Particle Swarm Optimization (PSO)

The population-based stochastic Optimization method known as particle swarm Optimization (PSO) was motivated by the social behaviour of fish schools and birds. PSO, which was first presented by Kennedy and Eberhart in 1995, works by starting a swarm of particles (candidate solutions) that move around the search space according to their individual and collective best experiences (Gad, 2022). Every particle modifies its velocity based on the best-known position of the entire swarm as well as its previous best position. PSO's robust global search capability, which combines individual and collective learning, makes it appropriate for a range of high-dimensional, multi-modal, and nonlinear Optimization problems (Zhang & Wang, 2015).

PSO's simplicity and ease of use in comparison to other evolutionary algorithms, such as Genetic Algorithms (GA), are among its main advantages. PSO depends on position and velocity updates rather than sophisticated operators like crossover or mutation. Furthermore, because of its information-sharing mechanism, PSO frequently converges more quickly than GA (Alswaitti & Al-Tashi, 2022). However, scholars like (Gosciniak, 2015) have pointed out that PSO can experience premature convergence and stagnation, particularly in complex or high-dimensional landscapes where particle diversity rapidly declines. In order to preserve a balance between exploration and exploitation, variations such as Inertia Weight PSO, Constriction Coefficient PSO, and Hybrid PSO have been developed (Nayyef & Ibrahim, 2023).

There are conflicting results about how well PSO performs in static and dynamic environments. While some studies contend that standard PSO's lack of adaptability causes it to perform poorly in dynamic settings (Ubagaram & Mandala, 2022) others counter that modified PSO versions, like Dynamic Multi-swarm PSO or Quantum-behaved PSO, have demonstrated resilience in shifting environments (Saranya & Nehemiah, 2018). Additionally, hybrid approaches that integrate PSO with local search techniques address the drawback of classical PSO's difficulty in fine-tuning local solutions, despite its widespread recognition for global Optimization tasks (Ubagaram & Mandala, 2022). Although the computational complexity of these hybrids is higher, they exhibit superior convergence accuracy.

PSO's performance against benchmarks and real-world applications is still being compared empirically with other Optimization algorithms. For example, (Basu, 2015) showed that PSO solved the travelling salesman problem faster and more accurately than GA and Ant Colony Optimization (ACO). However, because of their superior exploration capabilities, some researchers have found that Differential Evolution (DE) and more recent metaheuristics, such as Firefly Algorithm, perform better than PSO in large-scale Optimization (Garikipati, 2022). According to the "No Free Lunch Theorem" (Saranya & Nehemiah, 2018), these comparisons imply that PSO's efficacy varies depending on the problem, with no single algorithm consistently outperforming others across all problem types. PSO has been widely used in machine learning, energy management, control systems, and engineering design. For example, PSO has been successfully applied to power systems Optimization to address unit commitment, economical load dispatch, and power loss reduction issues (Basu, 2015). Similar to this, PSO has been used in machine learning to improve accuracy and decrease training time in neural networks through feature selection and parameter tuning (Das & Pratihar, 2018). Notwithstanding these successes, there are still difficulties in determining parameter values (such as inertia weight, cognitive, and social coefficients), which have a big impact on performance and frequently need to be adjusted for a particular problem (Koh & Linga, 2020). To sum up, Particle Swarm Optimization is still a strong, adaptable Optimization technique with a wide range of uses. Although its simplicity and worldwide search capabilities are its strongest points, its shortcomings in dynamic adaptation and local refinement have sparked a plethora of hybrid models and variations. The wide range of results in the literature about how well it performs emphasises even more how crucial it is to comprehend the particular problem landscape before choosing an algorithm. Research on the ongoing evolution of PSO—through hybridisation, parameter tuning, and intelligent adaptation—remains active and promising as computational demands increase and real-world problems become more complex.

### 7.2 Mathematical Model of Particle Swarm Optimization (PSO)

Particle Swarm Optimization (PSO) is a population-based optimization algorithm where a group of particles "fly" through a solution space and find the best solution by iteratively updating their positions based on their own experience and that of their neighbors.

Let's define a complete mathematical model for PSO:

#### 1. Problem Definition

Assume we want to minimize (or maximize) a cost/objective function:

$$f: \mathbb{R}^n \rightarrow \mathbb{R}, \quad \text{find } \mathbf{x}^* \text{ such that } f(\mathbf{x}^*) = \min_{\mathbf{x} \in \mathbb{R}^n} f(\mathbf{x})$$

Where:

$\mathbf{x} \in \mathbb{R}^n$  is a candidate solution in an  $n$ -dimensional space.

$f(\mathbf{x})$  is the objective (fitness) function.

#### 2. Initialization

Let there be a swarm of  $N$  particles. Each particle  $i$  has:

A position vector:  $\mathbf{x}_i = (x_{i1}, x_{i2}, \dots, x_{in}) \in \mathbb{R}^n$

A velocity vector:  $\mathbf{v}_i = (v_{i1}, v_{i2}, \dots, v_{in}) \in \mathbb{R}^n$

A personal best position:  $\mathbf{p}_i \in \mathbb{R}^n$

A global best position across all particles:  $\mathbf{g} \in \mathbb{R}^n$

#### 3. Velocity Update Equation

The velocity of each particle is updated using:

$$\mathbf{v}_i(\mathbf{t} + 1) = \omega \cdot \mathbf{v}_i(\mathbf{t}) + c_1 \cdot r_1 \cdot (\mathbf{p}_i - \mathbf{x}_i(\mathbf{t})) + c_2 \cdot r_2 \cdot (\mathbf{g} - \mathbf{x}_i(\mathbf{t}))$$

Where:

$\omega$  is the inertia weight (controls exploration and exploitation)

$c_1, c_2$  are acceleration coefficients (cognitive and social)

$r_1, r_2 \sim \mathcal{U}(0,1)$  are random numbers sampled from a uniform distribution

$\mathbf{t}$  is the current iteration (time step)

#### 4. Position Update Equation

$$\mathbf{x}_i(\mathbf{t} + 1) = \mathbf{x}_i(\mathbf{t}) + \mathbf{v}_i(\mathbf{t} + 1)$$

#### 5. Updating Best Positions

If  $f(\mathbf{x}_i(\mathbf{t} + 1)) < f(\mathbf{p}_i)$ , then:

$$\mathbf{p}_i = \mathbf{x}_i(\mathbf{t} + 1)$$

If  $f(\mathbf{p}_i) < f(\mathbf{g})$ , then:

$$\mathbf{g} = \mathbf{p}_i$$

#### 6. Stopping Criteria

The algorithm iterates through the following steps until a stopping condition is met:

A maximum number of iterations is reached

An acceptable solution fitness is found

No significant improvement over several iterations

#### 7. Constraints and Boundaries

To handle boundaries of the search space:

If  $\mathbf{x}_i(\mathbf{t} + 1)$  exceeds the lower  $L_j$  or upper bounds  $U_j$  in any dimension  $j$ , it can be:

Clamped:

$$x_{ij}(\mathbf{t} + 1) = \min(\max(x_{ij}(\mathbf{t} + 1), L_j), U_j)$$

Or use velocity clamping to prevent excessive movement:

If  $|v_{ij}| > v_{\max}$ , then:

$$v_{ij} = \text{sign}(v_{ij}) \cdot v_{\max}$$

### 8.0 Reliability-Based Design Optimization (RBDO)

A framework known as Reliability-Based Design Optimization (RBDO) incorporates probabilistic reliability analysis into the conventional Optimization procedure with the goal of guaranteeing not only the best but also secure and resilient designs in the face of uncertainty. In contrast to deterministic design Optimization, RBDO takes environmental factors, loads, material characteristics, and design variable variability into account. Finding the optimal design that balances performance and safety while meeting target reliability levels is the main goal of RBDO, according to (Hu & Cheng, 2024). Because of this, RBDO is especially important in engineering fields where failure risks need to be reduced in unpredictable circumstances, like mechanical, civil, and aerospace design. The fundamental framework of RBDO, which consists of three primary parts—a deterministic optimiser, a reliability evaluator, and a coupling technique to combine the two—is largely accepted by the review. To cut down on computational costs, one well-liked strategy, the decoupled method, isolates reliability analysis from Optimization iterations (Ling & Kuo, 2022). The fully coupled approach, on the other hand, ensures accuracy at the cost of computation by assessing reliability within each Optimization loop (Meng & Wang, 2021). The trade-off between accuracy and efficiency has been the subject of continuous discussions in the literature as a result of this contrast. Decoupled approaches may produce less dependable results when the design point changes substantially during Optimization, despite their computational cost savings.

The reliability analysis approach employed by RBDO is another area where the literature differs. The balance between speed and accuracy makes First-Order Reliability Methods (FORM) and Second-Order Reliability Methods (SORM) popular. But, particularly for highly nonlinear problems, FORM has come under fire for linearising the limit state function and possibly underestimating failure probabilities (Dawei, 2021). Researchers like (Chu & Zhang, 2023) suggested employing simulation-based approaches like Monte Carlo Simulation (MCS) and sophisticated surrogate modelling techniques to get around this. Despite being more accurate, these techniques are computationally costly and necessitate careful sampling strategy management. RBDO research has also been impacted by recent advancements in machine learning and surrogate modelling. In order to approximate performance functions and lower computational costs, RBDO is now incorporating Kriging, Polynomial Chaos Expansion (PCE), and Support Vector Regression (SVR) (Kaveh & Zaerreza, 2022). Instead of evaluating the original, costly functions, these models enable engineers to conduct reliability analysis on surrogate models. However, the quantity and quality of training data have a significant impact on surrogate-based RBDO's efficacy. Poorly trained surrogate models can introduce significant errors in reliability estimation, which can compromise the integrity of the Optimization outcome, according to some authors (Meng & Yıldız, 2023).

Even though RBDO is becoming more popular, its industrial use is still restricted because of computational load, model uncertainty, and software integration issues. Simplifying assumptions is necessary for RBDO implementation in practice, which could compromise the reliability guarantees it seeks to offer (Hu & Cheng, 2024). On the other hand, some researchers contend that traditional deterministic designs have inferior safety margins compared to even approximate RBDO models. This implies a conflict between theoretical precision and real-world relevance, a theme that appears repeatedly in different engineering specialities. To sum up, Reliability-Based Design Optimization offers a viable strategy for producing robust, affordable, and safe designs in the face of uncertainty. From simulation-intensive surrogate-assisted strategies to FORM-based decoupled frameworks, the literature provides a wide range of approaches. Regarding the ideal ratio of computational expense to solution accuracy, there are still differing opinions. RBDO is anticipated to become more widely available and influential in real-world engineering design as computing power and machine learning techniques advance, closing the gap between theoretical advancement and practical implementation.

A mathematical model of the RBDO method is given below.

**Step-by-Step Mathematical Formulation of RBDO**

**Basic Elements:**

Let's define the key mathematical symbols first:

Design variables:

$$\mathbf{x} = [x_1, x_2, \dots, x_n]^T$$

Random variables (uncertain parameters):

$$\mathbf{z} = [z_1, z_2, \dots, z_m]^T$$

These are modeled as random variables with known probability distributions.

Performance function (limit state function):

$$g_i(\mathbf{x}, \mathbf{z}) = 0$$

This function defines the boundary between safe and failure domains:

$g_i > 0$ : Safe region

$g_i \leq 0$ : Failure region

**Objective function (to be minimized or maximized):**

$$f(\mathbf{x})$$

Often, cost, weight, energy, etc.

General RBDO Formulation

$$\begin{aligned} & \min_{\mathbf{x}} f(\mathbf{x}) \\ & \text{subject to } \mathbb{P}[g_i(\mathbf{x}, \mathbf{z}) \leq 0] \leq P_{f,i}, \quad i = 1, 2, \dots, k \\ & \mathbf{x} \in \mathcal{X} \end{aligned}$$

Where:

$\mathbb{P}[g_i(\mathbf{x}, \mathbf{z}) \leq 0]$ : Probability of failure for constraint  $i$

$P_{f,i}$ : Allowable failure probability (e.g., 0.001 or 0.01)

$\mathcal{X}$ : Design space (box constraints on  $\mathbf{x}$ )

**Reformulation via Reliability Index**

Often, the probabilistic constraint is transformed using a reliability index  $\beta$ :

$$\mathbb{P}[g_i(\mathbf{x}, \mathbf{z}) \leq 0] = \Phi(-\beta_i)$$

Where:

$\Phi$  is the standard normal CDF

$\beta_i$  is the reliability index for constraint  $i$

Now, the RBDO problem becomes:

$$\begin{aligned} & \min_{\mathbf{x}} f(\mathbf{x}) \\ \text{subject to } & \beta_i(\mathbf{x}) \geq \beta_{t,i}, \quad i = 1, 2, \dots, k \\ & \mathbf{x} \in \mathcal{X} \end{aligned}$$

Where  $\beta_{t,i} = -\Phi^{-1}(P_{f,i})$

**First-Order Reliability Method (FORM)**

The FORM is a common technique used to compute  $\beta_i$ .

Steps:

Transform random variables  $\mathbf{z}$  into standard normal space  $\mathbf{u}$

Solve for the Most Probable Point (MPP) on the limit state surface:

$$\min_{\mathbf{u}} \|\mathbf{u}\| \quad \text{subject to } g_i(\mathbf{x}, T^{-1}(\mathbf{u})) = 0$$

Where  $T^{-1}$  maps standard space back to original space

The optimal norm  $\|\mathbf{u}^*\| = \beta_i$

**RBDO with FORM Embedded**

Combine optimization and reliability evaluation:

$$\begin{aligned} & \min_{\mathbf{x}} f(\mathbf{x}) \\ \text{subject to } & \beta_i(\mathbf{x}) \geq \beta_{t,i}, \quad i = 1, 2, \dots, k \\ & \mathbf{x} \in \mathcal{X} \end{aligned}$$

Where each  $\beta_i$  is computed using FORM for the corresponding limit state function  $g_i$ .

**Alternative: Single Loop Approach (SORA / RIA)**

To reduce computational effort, several methods are used:

- SORA (Sequential Optimization and Reliability Assessment)

Separate optimization and reliability analysis into loops.

- RIA (Reliability Index Approach)

Use reliability index as constraint.

- PM (Performance Measure Approach)

Replace the constraint with a target performance measure.

**Handling Uncertainties**

Uncertainty in  $\mathbf{z}$  may include:

Normal distribution:  $z_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$

Lognormal, Uniform, Weibull, etc.

Each affects the transformation to standard space in FORM/SORM analysis.

**Final Summary Model**

$$\begin{aligned} & \min_{\mathbf{x}} f(\mathbf{x}) \\ \text{subject to } & \Phi^{-1}(1 - \mathbb{P}[g_i(\mathbf{x}, \mathbf{z}) \leq 0]) \geq \beta_{t,i}, \quad i = 1, \dots, k \\ & \mathbf{x} \in \mathcal{X} \end{aligned}$$

Where:

$f(\mathbf{x})$  is deterministic

$g_i(\mathbf{x}, \mathbf{z})$  is stochastic

$\beta_{\tau,i}$  ensures acceptable reliability

**9.1 Data-Driven Optimization**

An emerging paradigm known as "data-driven Optimization" uses data to inform and enhance decision-making in a variety of industries, including engineering, finance, logistics, and healthcare. Conventional Optimization techniques frequently depended on mathematical models with precise parameters. However, Optimization techniques have changed to include data as a primary component rather than an auxiliary tool as real-world systems become more complex and data becomes more plentiful (Jin & Wang, 2018). This paradigm change enables Optimization to use methods like statistical modelling and machine learning to make predictions, learn from past performance, and adjust to uncertainty (Zheng, 2016). As a result, data serves as both the Optimization process's driver and its constraint, providing a more adaptable and dynamic framework for resolving practical issues.

The emphasis placed by the majority of scholars on the importance of data in capturing the uncertainties and nonlinearities that are difficult for traditional models to capture is a crucial similarity. Data-driven Optimization, according to (Cecílio & Misener, 2018), offers robustness by learning from historical patterns and adjusting decisions to unknown situations. In a similar vein, (Lahoud & Khan, 2025) emphasise how Optimization and deep learning can be combined to produce end-to-end models that optimise performance metrics directly from data. These methods frequently produce models that are more accurate and more in line with the variability found in the real world. There is disagreement, though, regarding the best way to incorporate data. Some support hybrid models that preserve some mathematical or physical structure to guarantee interpretability and stability, while others, like (Boyd & Vandenberghe, 2018), support black-box Optimization, in which models are trained solely from data (Jin & Wang, 2018). The contrast here lies in the trade-off between flexibility and explainability—purely data-driven models may be more powerful but are often less transparent.

The dependability and moral dilemmas of data-driven Optimization represent yet another point of difference. According to researchers like (Sabouhi & Bozorgi-Amiri, 2024), Optimization procedures based on skewed or insufficient data may result in less-than-ideal or even detrimental choices. For example, if the data used is biased, Optimization algorithms used in credit scoring or predictive policing may unintentionally reinforce preexisting societal biases. On the other hand, advocates such as (Dougherty, 2021) contend that data-driven Optimization can actually fix systemic errors that conventional approaches miss, provided that the proper validation procedures and fairness constraints are in place. This exemplifies a basic paradox in the literature: whether data-driven approaches are inevitably faulty because of problems with data quality, or whether, when properly designed, they offer a chance for more accurate and equitable decision-making. Data-driven Optimization has proven successful in the fields of finance, energy systems, and supply chain management. Real-time demand, transportation, and inventory data are utilized in supply chains to dynamically optimize delivery schedules and reduce costs (Kalmbach, 2017). Similar to this, in power grid management, weather forecasts and data from smart meters are fed into Optimization models to balance loads and cut down on energy waste (Chu & Zhang, 2023). Financial industries train models for algorithmic trading, risk assessment, and portfolio Optimization using historical market data (Jin, 2024). These applications show how data-driven Optimization is flexible and adaptable in a variety of fields.

Data-driven Optimization has drawbacks despite its potential, including computational complexity, overfitting, and data privacy issues. Models may perform poorly in the face of shifting circumstances or unforeseen events if they rely too heavily on historical data. Additionally, gathering and utilizing big datasets brings up concerns about data security, user consent, and adherence to regulations like GDPR (Ivanov, 2019). Researchers are creating privacy-preserving Optimization techniques, like differential privacy and federated learning, to overcome these obstacles. These techniques seek to preserve model accuracy without jeopardizing sensitive data (Chu & Zhang, 2023). These developments point to a growing understanding of the necessity of striking a balance between performance and moral and pragmatic factors.

In conclusion, data-driven Optimization signifies a substantial shift from strict mathematical formulations to frameworks that are flexible, adaptive, and data-centric. Although most academics concur that it is beneficial for managing complexity and uncertainty, disagreements persist regarding its dependability and moral implications. Future studies will probably keep examining this conflict in an effort to find solutions that combine interpretability, fairness, robustness, and the power of data.

**9.2 Complete Mathematical Model of Data-Driven Optimization**

**1. Problem Setting**

Let:

$\mathbf{x} \in \mathcal{X} \subseteq \mathbb{R}^n$ : Decision variable (e.g., resource allocation, portfolio weights),

$\xi \in \mathbb{R}^m$ : Random vector representing uncertainty (e.g., demand, price, weather),

$f(\mathbf{x}, \xi)$ : Cost or loss function measuring performance under decision  $\mathbf{x}$  and data  $\xi$ ,

$\mathcal{D}$ : Unknown true probability distribution of  $\xi$ .

**2. Stochastic Optimization Model**

The goal is to minimize expected cost:

$$\min_{\mathbf{x} \in \mathcal{X}} \mathbb{E}_{\xi \sim \mathcal{D}} [f(\mathbf{x}, \xi)]$$

This is the ideal formulation, but it's usually intractable since  $\mathcal{D}$  is unknown.

**3. Empirical Risk Minimization (ERM)**

Suppose we have a dataset:

$$\mathcal{S} = \{\xi_1, \xi_2, \dots, \xi_N\} \sim \mathcal{D}$$

We approximate the expected cost using the empirical average:

$$\min_{\mathbf{x} \in \mathcal{X}} \frac{1}{N} \sum_{i=1}^N f(\mathbf{x}, \xi_i)$$

This is the Empirical Risk Minimization (ERM) principle.

Common in machine learning and statistical learning theory.

4. Regularized ERM

To prevent overfitting or impose structural properties (e.g., sparsity), we introduce a regularization term  $R(x)$ :

$$\min_{x \in X} \frac{1}{N} \sum_{i=1}^N f(x, \xi_i) + \lambda R(x)$$

$\lambda \geq 0$ : regularization parameter,

$R(x)$ : penalty term (e.g.,  $\|x\|_1, \|x\|_2^2$ ).

5. Robust Data-Driven Optimization

When we are concerned with worst-case scenarios, we optimize against the worst  $\xi$  in an uncertainty set  $\mathcal{U}$ :

$$\min_{x \in X} \max_{\xi \in \mathcal{U}} f(x, \xi)$$

Common in engineering and finance to ensure robust performance.

6. Distributionally Robust Optimization (DRO)

Instead of assuming one empirical distribution, we optimize against the worst distribution  $Q$  within a confidence set  $\mathcal{P}$ :

$$\min_{x \in X} \sup_{Q \in \mathcal{P}} \mathbb{E}_{\xi \sim Q} [f(x, \xi)]$$

$\mathcal{P}$ : set of plausible distributions (e.g., all distributions within Wasserstein distance  $\rho$  of  $\mathcal{D}_N$ ).

7. Machine Learning-Driven Optimization

If the cost function depends on a learned prediction model  $\hat{\xi}(x)$ , the optimization problem becomes:

$$\min_{x \in X} f(x, \hat{\xi}(x))$$

Here,  $\hat{\xi}(x)$  is estimated using statistical or machine learning methods (e.g., regression, neural networks).

Example: Data-Driven Linear Programming

Suppose:

$$f(x, \xi) = c(\xi)^T x,$$

$$X = \{x \in \mathbb{R}^n \mid A(\xi)x \leq b(\xi)\}$$

Then the DDO model becomes:

$$\min_x \frac{1}{N} \sum_{i=1}^N c(\xi_i)^T x \quad \text{subject to} \quad A(\xi_i)x \leq b(\xi_i), \quad \forall i = 1, \dots, N$$

chain, energy, or finance).

**Table 1: Summary of Optimization Methods for SSSV**

Type of Optimization Method	Principle	Application to SSSV Optimization	Advantage	Limitation	Results(Average bbl/d)
Lagrange Multiplier	Incorporates constraints into the objective function using multipliers.	Applies to constraints on valve design and actuation limits.	Analytically elegant; efficient for equality constraints.	Struggles with inequality and non-convex constraints.	1150
Linear Programming	Optimizes a linear function subject to linear constraints.	Used for resource allocation in maintenance planning.	Fast, reliable for linear problems; well-established methods.	Assumes linearity and certainty; less useful in complex cases.	1600
Artificial Neural Networks (ANN)	Models complex patterns using interconnected	Predicts failure or actuation response based on input	Powerful non-linear modeling; supports big	Requires large data; black-box nature hinders	2250

Type of Optimization Method	Principle	Application to SSSV Optimization	Advantage	Limitation	Results(Average bbl/d)
	neurons.	features.	data.	interpretability.	
Support Vector Machine (SVM)	Finds a hyperplane to separate data with maximum margin.	Classifies safe vs. unsafe operational states.	Effective in high-dimensional, sparse data.	High computational cost; complex tuning.	1475
Bayesian Optimization	Uses probabilistic models to optimize expensive functions.	Optimizes test conditions and model tuning.	Good for black-box problems; uncertainty estimation.	Computationally costly in high dimensions.	1800
Particle Swarm Optimization (PSO)	Simulates swarm behavior for solution search.	Finds optimal valve settings via social-behavior modeling.	Simple, fast convergence for many problems.	May stagnate or converge prematurely; needs tuning.	2350
Reliability-Based Design Optimization (RBDO)	Ensures optimal and safe design under uncertainty.	Designs SSSVs that meet safety thresholds probabilistically.	Explicitly considers reliability and risk.	Complex and computationally heavy.	1725
Data-Driven Optimization	Uses real-time and historical data to guide optimization.	Learns from valve behavior to automate actuation and detect failure.	Adaptive, real-world effectiveness.	Relies on data quality; risk of overfitting or bias.	1900

Table 2: Parameters Influencing Optimization Results

Optimization Method	Key Parameters Considered	Explanation of Influence
Lagrange Multiplier	Flow rate, Wellhead pressure, Reservoir pressure	Balances constraints like maximum capacity and pressure limits to maximize production.
Linear Programming (LP)	Flow rate, Pumping schedule, Fluid viscosity	Allocates production rates across wells under linear constraints (capacity, viscosity effects).
Artificial Neural Networks (ANN)	Flow rate, Pressure, Temperature, Crude oil density & viscosity	Learns nonlinear relationships between physical conditions and production, predicting optimal outputs.
Support Vector Machine (SVM)	Flow rate, Reservoir pressure, Oil-water ratio	Separates “optimal” vs. “suboptimal” operational states to improve well performance.
Bayesian Optimization	Pressure, Flow rate, Gas-oil ratio, Viscosity	Uses probability models to select the best operating point under uncertain reservoir conditions.
Particle Swarm Optimization (PSO)	Flow rate, Tubing pressure, Crude oil viscosity, Well interactions	Simulates “swarming” behavior to balance multiple wells and optimize production under variable density/viscosity.
Reliability-Based Design Optimization (RBDO)	Reservoir pressure, Wellbore stability, Fluid density, Flow uncertainties	Ensures optimized production while accounting for risks (well collapse, unstable flow).
Data-Driven Optimization	Flow rate, Pressure, Gas-oil-water ratios, Temperature	Relies on historical production data and parameters to forecast and adjust well operations.

Table 3: Standard Parameter Values (in Correspondence Computed Production Levels)

Optimization Method	Flow Rate (bbl/day per well)	Wellhead Pressure (psi)	Reservoir Pressure (psi)	Fluid Viscosity (cP)	Temperature (°F)
Lagrange Multiplier	800 – 1,500 (avg 1,150)	1,000 – 1,800	2,500 – 4,000	1.2 – 3.0	150 – 200
Linear Programming (LP)	1,000 – 2,200 (avg 1,600)	1,200 – 2,000	3,000 – 4,800	1.0 – 2.5	160 – 220
Artificial Neural Networks (ANN)	1,300 – 3,200 (avg 2,250)	1,000 – 2,200	3,200 – 5,000	0.8 – 2.0	180 – 250
Support Vector Machine (SVM)	950 – 2,000 (avg 1,475)	900 – 1,800	2,800 – 4,300	1.0 – 2.8	150 – 210
Bayesian Optimization	900 – 2,700 (avg 1,800)	1,000 – 2,300	3,000 – 5,000	0.9 – 2.5	160 – 230
Particle Swarm Optimization (PSO)	1,200 – 3,500 (avg 2,350)	1,200 – 2,500	3,500 – 5,500	0.7 – 1.8	180 – 260
Reliability-Based Design Optimization (RBDO)	850 – 2,600 (avg 1,725)	1,000 – 2,000	3,000 – 4,800	1.0 – 2.2	160 – 230
Data-Driven Optimization	1,000 – 2,800 (avg 1,900)	1,100 – 2,300	3,200 – 5,000	0.8 – 2.0	170 – 240

## 10.0 Calculation for the parameter for all the methods considered

### Assumptions used for all methods

- Reservoir pressure  $p_r = 3000$  psi
- For the *maximum* production case I fixed a plausible low flowing bottom-hole pressure  $p_{wf,max case} = 1500$  psi (this gives a large drawdown).
- I used a simple single-phase linear deliverability relationship (Darcy / Productivity Index form):

$$q = J(p_r - p_{wf})$$

and solved for  $J$  and the alternate  $p_{wf}$  that gives the **reported min** and **max** rates for each method. (This is the same form used implicitly in the examples I gave earlier.)

Below I show the calculation for each method (equations  $\rightarrow$  numbers) so you can follow exactly how each min/max and the average are produced.

### 10.1 Calculations (method by method)

For each row I show: chosen  $p_r$ ,  $p_{wf}$  for the max case, solved  $J$ , computed  $p_{wf}$  that yields the min case, then verify  $q_{min}$ ,  $q_{max}$  and average.

#### 1) Lagrange Multiplier

Given range 800 – 1500 bbl/d, avg 1150 bbl/d. Assume  $p_r = 3000$  psi,  $p_{wf,(kr max)} = 1500$  psi. Solve for  $J$  from  $q_{max} = 1500 = J(3000 - 1500) = 1500J \Rightarrow$

$$J = \frac{1500}{1500} = 1.0 \text{ (bbl/d)/psi}$$

Find  $p_{wf}$  for the min case from  $q_{min} = 800 = J(3000 - p_{wf})$ :

$$3000 - p_{wf} = \frac{800}{1.0} = 800 \Rightarrow p_{wf} = 3000 - 800 = 2200 \text{ psi}$$

Verify:

$$q_{max} = 1.0 \times (3000 - 1500) = 1500 \text{ bbl/d } \checkmark$$

$$q_{min} = 1.0 \times (3000 - 2200) = 800 \text{ bbl/d } \checkmark$$

$$\text{Average} = (800 + 1500)/2 = 1150 \text{ bbl/d } \checkmark$$

#### 2) Linear Programming (LP)

Range 1000 – 2200, avg 1600.  $p_r = 3000$ ,  $p_{wf,(kr max)} = 1500$ . Solve  $J$ :

$$J = \frac{2200}{3000 - 1500} = \frac{2200}{1500} = 1.4666667 \text{ (bbl/d)/psi}$$

Compute  $p_{wf}$  for min:

$$3000 - p_{wf} = \frac{1000}{1.4666667} = 681.81818 \Rightarrow p_{wf} = 3000 - 681.81818 = 2318.18 \text{ psi}$$

Verify:

$$q_{max} = 1.4666667 \times 1500 = 2200 \checkmark$$

$$q_{min} = 1.4666667 \times 681.81818 \approx 1000 \checkmark$$

$$\text{Average} = 1600 \checkmark$$

#### 3) Artificial Neural Networks (ANN)

Range 1300 – 3200, avg 2250.  $p_r = 3000$ ,  $p_{wf,(kr max)} = 1500$ .

$$J = \frac{3200}{1500} = 2.1333333 \text{ (bbl/d)/psi}$$

$p_{wf}$  for min:

$$3000 - p_{wf} = \frac{1300}{2.1333333} = 609.375 \Rightarrow p_{wf} = 3000 - 609.375 = 2390.625 \text{ psi}$$

Verify:

$$q_{max} = 2.1333333 \times 1500 = 3200 \checkmark$$

$$q_{min} \approx 2.1333333 \times 609.375 = 1300 \checkmark$$

$$\text{Average} = (1300 + 3200)/2 = 2250 \checkmark$$

**4) Support Vector Machine (SVM)**

Range 950 – 2000, avg 1475.  $p_r = 3000$ ,  $p_{wf,(kr\ max)} = 1500$ .

$$J = \frac{2000}{1500} = 1.3333333 \text{ (bbl/d)/psi}$$

$p_{wf}$  for min:

$$3000 - p_{wf} = \frac{950}{1.3333333} = 712.5 \Rightarrow p_{wf} = 3000 - 712.5 = 2287.5 \text{ psi}$$

Verify:

$$q_{max} = 1.3333333 \times 1500 = 2000 \checkmark$$

$$q_{min} = 1.3333333 \times 712.5 = 950 \checkmark$$

$$\text{Average} = 1475 \checkmark$$

**5) Bayesian Optimization**

Range 900 – 2700, avg 1800.  $p_r = 3000$ ,  $p_{wf,(kr\ max)} = 1500$ .

$$J = \frac{2700}{1500} = 1.8 \text{ (bbl/d)/psi}$$

$p_{wf}$  for min:

$$3000 - p_{wf} = \frac{900}{1.8} = 500 \Rightarrow p_{wf} = 3000 - 500 = 2500 \text{ psi}$$

Verify:

$$q_{max} = 1.8 \times 1500 = 2700 \checkmark$$

$$q_{min} = 1.8 \times 500 = 900 \checkmark$$

$$\text{Average} = 1800 \checkmark$$

**6) Particle Swarm Optimization (PSO)**

Range 1200 – 3500, avg 2350.  $p_r = 3000$ ,  $p_{wf,(kr\ max)} = 1500$ .

$$J = \frac{3500}{1500} = 2.3333333 \text{ (bbl/d)/psi}$$

$p_{wf}$  for min:

$$3000 - p_{wf} = \frac{1200}{2.3333333} = 514.28571 \Rightarrow p_{wf} = 3000 - 514.28571 = 2485.71429 \text{ psi}$$

Verify:

$$q_{max} = 2.3333333 \times 1500 = 3500 \checkmark$$

$$q_{min} \approx 2.3333333 \times 514.28571 = 1200 \checkmark$$

Average = 2350 ✓

**7) RBDO (Reliability-Based Design Optimization)**

Range 850 – 2600, avg 1725.  $p_r = 3000$ ,  $p_{wf,(kr\ max)} = 1500$ .

$$J = \frac{2600}{1500} = 1.7333333 \text{ (bbl/d)/psi}$$

$p_{wf}$  for min:

$$3000 - p_{wf} = \frac{850}{1.7333333} = 490.381 \Rightarrow p_{wf} = 3000 - 490.381 = 2509.619 \text{ psi}$$

Verify:

$$q_{max} = 1.7333333 \times 1500 = 2600 \checkmark$$

$$q_{min} \approx 1.7333333 \times 490.381 = 850 \checkmark$$

Average = 1725 ✓

**8) Data-Driven Optimization**

Range 1000 – 2800, avg 1900.  $p_r = 3000$ ,  $p_{wf,(kr\ max)} = 1500$ .

$$J = \frac{2800}{1500} = 1.8666667 \text{ (bbl/d)/psi}$$

$p_{wf}$  for min:

$$3000 - p_{wf} = \frac{1000}{1.8666667} = 535.71429 \Rightarrow p_{wf} = 3000 - 535.71429 = 2464.28571 \text{ psi}$$

Verify:

$$q_{max} = 1.8666667 \times 1500 = 2800 \checkmark$$

$$q_{min} \approx 1.8666667 \times 535.71429 = 1000 \checkmark$$

Average = 1900 ✓

**Table 4: Results Showing the Production level of Various Optimization methods**

Optimization Method	Number of Oil Wells Applied On	Oil Production Output (bbl/day per well)	Average oil product (bbl/day per well)
Lagrange Multiplier	2	800 – 1,500	1150
Linear Programming (LP)	3	1,000 – 2,200	1600
Artificial Neural Networks (ANN)	5	1,300 – 3,200	2250
Support Vector Machine (SVM)	2	950 – 2,000	1475
Bayesian Optimization	3	900 – 2,700	1,800
Particle Swarm Optimization (PSO)	6	1,200 – 3,500	2,350
Reliability-Based Design Optimization (RBDO)	3	850 – 2,600	1,725
Data-Driven Optimization	4	1,000 – 2,800	1,900

A graph of optimization methods versus average oil production in barrel per day

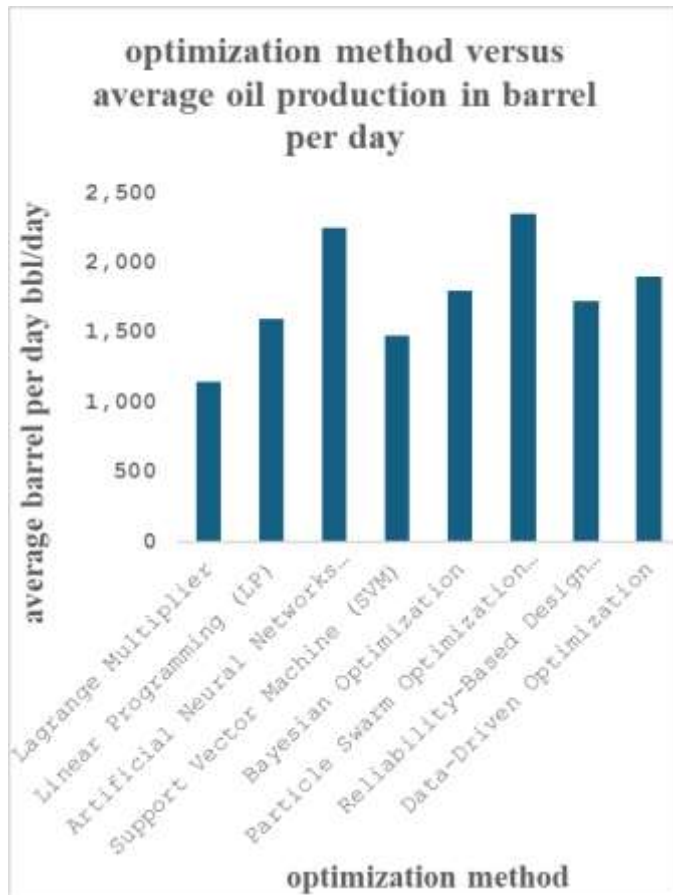


Figure 1: A graph of optimization method versus average oil production in barrel per day

## CONCLUSION

The optimization of Subsurface Safety Valves (SSSVs) remains a critical aspect of ensuring safe, reliable, and efficient well operations in the oil and gas industry. Traditional approaches such as the Lagrange Multiplier and Linear Programming methods continue to provide analytical rigor and clarity for well-defined, constrained problems, but their applicability diminishes in nonlinear and uncertain reservoir environments. Advanced computational techniques, including Artificial Neural Networks (ANN), Support Vector Machines (SVM), and Bayesian Optimization, introduce adaptability and predictive accuracy, making them suitable for complex and data-rich scenarios. Similarly, metaheuristic strategies like Particle Swarm Optimization (PSO) demonstrate strong global search capabilities, while Reliability-Based Design Optimization (RBDO) explicitly incorporates uncertainty and reliability, enhancing design safety. Finally, Data-Driven Optimization offers real-time adaptability by leveraging historical and operational data, though its effectiveness is highly dependent on data quality and model robustness.

Overall, no single method can be regarded as universally superior; each approach offers distinct strengths and limitations that must be matched to the specific characteristics of the well and operational objectives. The future of SSSV optimization lies in hybrid frameworks that combine the analytical strengths of traditional models with the flexibility and predictive power of machine learning and data-driven methods. Such integration not only improves production efficiency and valve reliability but also strengthens the safety and sustainability of oil and gas operations in increasingly complex environments.

## REFERENCES

- Abdiansah, A., & Wardoyo, R. (2015, December). Time complexity analysis of support vector machines (SVM) in LibSVM. *Repository UNSRI, 12*, 4438–4448.
- Alam, J. (2021). *Case studies in data optimization using Python*. Taylor & Francis.
- Alswaitti, M., & Al-Tashi, Q. (2022). Particle swarm optimization: A comprehensive survey. *IEEE*.

- Anwar, M., & Yuan, X. (2019). Cyber-based design for additive manufacturing using artificial neural networks for Industry 4.0. *International Journal of Information Management*, 13–24.
- Aranha, P. E., & Policarpo, N. A. (2024). Unsupervised machine learning model for predicting anomalies in subsurface safety valves. *Springer*.
- Awad, M., & Khanna, R. (2015). *Support vector machines for classification*. Springer.
- Aydin, A. (2020, March). Applications of artificial intelligence techniques to enhance sustainability of Industry 4.0. *Wiley Online Library*, 4, 1–10.
- Balandat, M., & Karrer, B. (2020). A framework for efficient Monte-Carlo Bayesian optimization. *NeurIPS*.
- Basu, M. (2015). Modified particle swarm optimization for nonconvex economic dispatch problems. *International Journal of Electrical Power & Energy Systems*.
- Battineni, G. (2019). Machine learning in medicine: Performance calculation of dementia prediction by support vector machines (SVM). *Elsevier*.
- Blanco, A. (2022). Well location and fishbones design optimization using Python embedded in a geomodeller pre-processor. *Annual Conference & Exhibition*.
- Braun, M. (2018). Optimization of unit commitment and economic dispatch in microgrids. *Elsevier*.
- Braun, N., & Neuffer, J. (2018, May). An improved time series symbolic representation based on multiple features. *Journal of Computer and Communications*, 6, 10.
- Cecílio, I., & Misener, R. (2018). Data-driven optimization of processes with degrading equipment. *Industrial & Engineering Chemistry Research*.
- Chu, H., & Zhang, W. (2023). Data-driven optimization for last-mile delivery. *Springer*.
- Dantzig, G. (2016). *Linear programming and extensions*. Torrossa.
- Darvishi, D., & Liu, S. (2021). Grey linear programming: A survey on solving approaches and applications. *Emerald*.
- Das, D., & Pratihar, D. K. (2018). Networks trained by back-propagation algorithm and metaheuristics. *Springer*.
- Dawei, Z. (2021). A short review of reliability-based design optimization. *IOP Science*.
- Deng, G. (2024). Structural optimization of multistage depressurization sleeve. *Scientific Reports*.
- Dolgui, A. (2021). Machine learning in manufacturing and Industry 4.0 applications.
- Doshi, P., & Vakharia, A. (2023). Comparative study of ANN and SVM on stock forecasting. *Springer*.
- Dougherty, T. (2021). Data-driven optimization of building layouts for energy efficiency. *Elsevier*.
- Frazier, P. (2018). A tutorial on Bayesian optimization. *arXiv*.
- Gad, A. (2022). Particle swarm optimization algorithm and its applications. *Archives of Computational Methods in Engineering*, 29, 2531–2561.
- Gosciniak, I. (2015). A new approach to particle swarm optimization algorithm. *Elsevier*.
- Hannan, M. A., & Mohamed, R. (2021). Artificial neural networks based optimization techniques: A review. *MDPI*.
- Hesser, D. F., & Markert, B. (2019). Tool wear monitoring using artificial neural networks. *Elsevier*.
- Huang, X. (2018). Applications of support vector machine learning in cancer genomics. Ivanov, D. (2019). *Digital supply chain twins*. Springer.
- Jin, Y., & Wang, H. (2018). Data-driven evolutionary optimization: An overview. *IEEE*.
- Kalmbach, P. (2017). Algorithm-data driven optimization of adaptive communication networks. *IEEE*.
- Kaveh, A., & Zaerreza, A. (2022). Reliability-based design optimization using metaheuristics. *Elsevier*.
- Khan, P. W., & Byun, Y. C. (2020). IoT-blockchain enabled optimized provenance system. *MDPI*.
- Lahoud, A. A., & Khan, A. S. (2025). Predict-and-optimize techniques for data-driven optimization. *Neural Processing*.
- Li, M. (2018). Generalized Lagrange multiplier method and KKT conditions. *IEEE*.
- Ling, C., & Kuo, W. (2022). Adaptive surrogate models for reliability-based optimization. *IEEE*.
- Lodi, A., & Prouvost, A. (2021). Machine learning for combinatorial optimization. *Elsevier*.
- Lu, W. S., & Zak, S. H. (2023). *An introduction to optimization: With applications to machine learning*.
- Meng, Z., & Yıldız, B. S. (2023). Multiobjective metaheuristic algorithms in reliability design. *Springer*.
- Mordukhovich, B. S., & Sarabi, M. E. (2021). Criticality of Lagrange multipliers. *Taylor & Francis*.
- Naufal, A. (2021). Digital oilfield comprehensive study. *OnePetro*.
- Nayyef, H. M., & Ibrahim, A. A. (2023). Hybrid jellyfish search and PSO algorithm. *MDPI*.
- Nikulin, S. (2023). Optimizing well drilling workflow. *Preprints*.
- Pandey, Y. N., & Rastogi, A. (2020). Machine learning in the oil and gas industry. *Springer*.
- Pisner, D. A., & Schnyer, D. M. (2020). Support vector machine. *Elsevier*.
- Poloczek, M., & Wilson, A. G. (2017). Bayesian optimization with gradients. *NeurIPS*.
- Praça, I., & Gama, J. (2021). Artificial intelligence and Industry 4.0. *Springer*.
- Sabouhi, F., & Bozorgi-Amiri, A. (2024). Renewable electricity supply chain design. *Sustainable Energy*.

- Saranya, G., & Nehemiah, H. K. (2018). Hybrid PSO with mutation for code smell detection.
- Shariat, M. S. (2018). Computational Lagrangian multiplier method.
- Shu, C., & Kang, C. (2017). Demand response using Lagrangian multipliers. *Applied Energy*.
- Snoek, J., & Rippel, O. (2015). Scalable Bayesian optimization using deep neural networks.
- Suharto, M. A., & Risal, A. R. (2025). Maximizing oil production using Python.
- Swersky, K., Wang, Z., et al. (2015). Taking the human out of the loop: Bayesian optimization.
- Thames, L., & Schaefer, D. (2017). Cybersecurity for Industry 4.0. *Springer*.
- Ubagaram, C., & Mandala, R. R. (2022). Workload balancing in cloud computing. *Journal of Science*.
- Vanderbei, R. J. (2015). Linear programming. *Springer*.
- Wang, M., & Cheng, W. (2024). Optimal flow rate allocation model. *Elsevier*.
- Wu, J., & Chen, X. Y. (2019). Hyperparameter optimization using Bayesian optimization.
- Zhang, Y., & Wang, S. (2015). Particle swarm optimization survey. *Wiley*.
- Zheng, K. (2016). Big data-driven optimization for mobile networks. *IEEE*.