Study programme / specialisation:
M. Sc. In Computational Engineering

The spring semester, 2022

Open / ~~Confidential~~

Author: Nidaa Raji

[illegible]
(signature author)

Course coordinator:

Supervisor(s):
Aojie Hong (UiS)
Kurt Petvipusit (Equinor)

Thesis title:

Utilizing machine learning algorithms in the ensemble-based optimization (EnOpt) method for enhancing gradient estimation.

Credits (ECTS): 30

Keywords:
Gradient
Optimization
Ensemble optimization
EnOpt
Machine Learning

Pages:100

+ appendix: 14

Stavanger, 15 June/2022.

Table of Contents

Acknowledgment 11

Abstract 12

Chapter 1: Introduction 14

- Background and motivation: 14
- Novelty of the work: 15
- Machine learning algorithms: 17
  - Gradient Boosting Regressor: 17
  - Adaptive Boosting (AdaBoost) Regressor: 17
  - Random Forest: 18
- Optimization test functions: 20

Chapter 3: Experiments on EnOpt-ML 24

- Introduction: 24
- Workflow: 24
- Base case configuration: 26
- Changing the starting point 32
  - Starting point [-5 2]: 32
  - Starting point [3 -1]: 37
- Changing initial covariance matrix 42
  - Initial covariance matrix: 0.2I 42
  - Initial covariance matrix: 0.4I 42
- Changing the sample size N 48
  - N = 5 48
  - N = 100 48
- Changing the Number of the predicted samples by the machine learning model 55
  - Nml = 2 55
  - Nml = 1000 55
- Different training iterations 61
- Reduction of sample size N throughout the iterations 63
  - Reduced sample size N used at iteration 4 (after 3rd iteration)to N=5 63
  - Reduced sample size N used at iteration 4 (after 3rd iteration) to N=10 63
- Machine learning algorithms 75
  - Adaptive boosting regressor: 75

Random Forest: 75
Chapter 4: Improvement example 81
On Himmelblau 81
On Buckin function number 6 86
On Rosenbrock 91
Chapter 5: Final remarks and Conclusions 96
Observations/ remarks: 96
Conclusion: 97
Appendix 99
EnOpt and EnOpt-ML codes 99
Objective Functions for test: 99
Machine Learning algorithms 100
Function for testing (EnOpt and EnOpt-ML) 100
Testing with different configurations On test functions 104
Compute F means and prepare data for plotting 106
PLOT simulation means 106
MEANS of different simulations *(SEPARATE PLOTS for EnOpt and EnOpt-ML)* 107
min and max values for plots 108
Plotting Gradient 108
Plotting samples 109
MSE for EnOpt-ML : ML predicted samples (Nml) 110
Create output files 110
Creating excel file with summary table *(useful when doing multiple simulations with numerous simulations)* 111
References: 113

List of Tables

Table 1 Optimization results for base configuration case (single simulation)........................28
Table 2 Base case configuration summary output values (1000 simulations)........................30
Table 3 Optimization results for case starting point [-5 2] (single simulation)...................33
Table 4 Case starting point [-5 2] : summary output values (1000 simulations)..................35
Table 5 Optimization results for Starting point [3 -1] (single simulation) 38
Table 6 Case starting point [3 -1]: summary output values (1000 simulations)...................40
Table 7 Optimization results for case: covariance 0.2I (single simulation) 43
Table 8 Optimization results for case: covariance 0.4I (single simulation) 43
Table 9 Case: covariance 0.2I : summary output values (1000 simulations) 46
Table 10 Case: covariance 0.4I : summary output values (1000 simulations) 46
Table 11 Optimization results for case N=5: (single simulation)...49
Table 12 Optimization results for case N=100 single simulation): 49
Table 13 Case N=5: summary output values (1000 simulations)...52
Table 14 Case N=100: summary output values (1000 simulations).....................................52
Table 15 Optimization results for case Nml = 20 (single simulation).................................56
Table 16 Optimization results for case Nml = 1000 (single simulation).............................56
Table 17 Case Nml = 20: summary output values (1000 simulations)................................59
Table 18 Case Nml = 1000: summary output values (1000 simulations)............................59
Table 19 Optimization results for Reduced sample size N = 5 used at iteration 4 (single simulation) 64
Table 20 Optimization results for Reduced sample size N = 10 used at iteration 4 (single simulation) 64
Table 21 Reduced sample size N = 5 used at iteration 4 summary output values (1000 simulations)..67
Table 22 Reduced sample size N = 10 used at iteration 4 summary output values (1000 simulations)..67
Table 23 Optimization results for case reduced sample size N = 0 used at iteration 4 (single simulation) 70
Table 24 Optimization results for case reduced sample size N = 0 used at iteration 5 (single simulation) 70
Table 25 Case reduced sample size N = 0 used at iteration 4 : summary output values (1000 simulations)..73

Table 26 Case reduced sample size N = 0 used at iteration 5 : summary output values (1000 simulations)..73
Table 27 Optimization results for AdaBoost case (single simulation).................................76
Table 28 Optimization results for Random Forest case (single simulation)..........................76
Table 29 AdaBoost case: summary output values (1000 simulations)..................................78
Table 30 Random Forest case: summary output values (1000 simulations) 78
Table 31 Optimization results for   improvement example (single simulation).....................83
Table 32 Improvement example: summary output values (1000 simulations).......................84
Table 33 Optimization results for Bukin function 6 (single simulation)...............................87
Table 34 Bukin function 6 : summary output values (1000 simulations)...............................89
Table 35 Optimization results for   Rosenbrock (single simulation)......................................92
Table 36 Rosenbrock: summary output values (1000 simulations)......................................94

List of Figures

Figure 1Workflow chart of EnOpt-ML....19

Figure 2 Himmelblau. Test function for optimization....20

Figure 3 Bukin function number 6. Test function for optimization....21

Figure 4 Rosenbrock. Test function for optimization (evaluated in the interval x ∈ [-2.048, 2.048], y ∈ [-2.048, 2.048])....22

Figure 5 Rosenbrock. test function for optimization (evaluated in the interval x ∈ [-5, 10], y ∈ [-5, 10])....22

Figure 6 Base configuration gradient (single simulation)....27

Figure 7 Base configuration EnOpt and EnOpt-ML samples and ML model predicted samples....27

- Figure 8 Base confirmation cases objective function optimization results (1000 simulations)....29

- Figure 9 Base configuration case: EnOpt and EnOpt-ML means of simulations....29

Figure 10Base configuration case: Boxplot of optimized objective function....30

Figure 11 Base case simulations' results of EnOpt and EnOpt-ML with respect to true minimum values....31

Figure 12 case starting point [-5 2] gradient (single simulation)....32

Figure 13 Case starting point [-5 2] , EnOpt and EnOpt-ML samples and ML model predicted samples....33

Figure 14 Case starting point [-5 2] : objective function optimization (1000 simulations)..34

Figure 15 case starting point [-5 2] : EnOpt and EnOpt-ML means of simulations....34

Figure 16 Case starting point [-5 2] : Boxplot of optimized objective function....35

Figure 17 Case starting point [-5 2] : simulations' results of EnOpt and EnOpt-ML with respect to true minimum values....36

Figure 18 Case - Starting point [3 -1]: gradient (single simulation)....37

Figure 19 Starting point [3 -1]: EnOpt and EnOpt-ML samples and ML model predicted samples....38

Figure 20 Case starting point [3 -1]: objective function optimization (1000 simulations)...39

Figure 21 Case starting point [3 -1]: EnOpt and EnOpt-ML means of simulations....39

Figure 22 Case starting point [3 -1]: : Boxplot of optimized objective function....40

Figure 23 Case starting point [3 -1]: simulations' results of EnOpt and EnOpt-ML with respect to true minimum values....41

Figure 24 Case: covariance 0.2I : gradient (single simulation) 42
Figure 25 Case: covariance 0.2I : EnOpt and EnOpt-ML samples and ML model predicted samples 42
Figure 26 Case: covariance 0.4I : gradient (single simulation) 42
Figure 27 Case: covariance 0.4I : EnOpt and EnOpt-ML samples and ML model predicted samples 42
Figure 28 Case: covariance 0.2I : objective function optimization results (1000 simulations) 44
Figure 29 Case: covariance 0.4I : objective function optimization results (1000 simulations) 44
Figure 30 Case: covariance 0.2I : EnOpt and EnOpt-ML means of simulations 44
Figure 31 Case: covariance 0.4I : EnOpt and EnOpt-ML means of simulations 44
Figure 32 Case: covariance 0.2I : Boxplot of optimized objective function 45
Figure 33 Case: covariance 0.4I : Boxplot of optimized objective function 45
Figure 34 Case: covariance 0.2I : simulations' results of EnOpt and EnOpt-ML with respect to true minimum values 47
Figure 35 Case: covariance 0.4I : simulations' results of EnOpt and EnOpt-ML with respect to true minimum values 47
Figure 36 Case N=5: gradient (single simulation) 48
Figure 37 Case N=5: EnOpt and EnOpt-ML samples and ML model predicted samples 48
Figure 38 Case N=100: gradient (single simulation) 48
Figure 39 Case N=100: EnOpt and EnOpt-ML samples and ML model predicted samples . 48
Figure 40 Case N=5: objective function optimization results (1000 simulations) 50
Figure 41 Case N=100: objective function optimization results (1000 simulations) 50
Figure 42 Case N=5: EnOpt and EnOpt-ML means of simulations 50
Figure 43 Case N=100: EnOpt and EnOpt-ML means of simulations 50
Figure 44 Case N=5: Boxplot of optimized objective function 51
Figure 45 Case N=100: Boxplot of optimized objective function 51
Figure 46 Case N=5: simulations' results of EnOpt and EnOpt-ML with respect to true minimum 53
Figure 47 Case N=100: simulations' results of EnOpt and EnOpt-ML with respect to true . 54
Figure 48 Case Nml = 20: gradient (single simulation) 55
Figure 49 Case Nml = 20: EnOpt and EnOpt-ML samples and ML model predicted samples 55

Figure 50 Case Nml = 1000: gradient (single simulation) 55
Figure 51 Case Nml = 1000 EnOpt and EnOpt-ML samples and ML model predicted samples 55
Figure 52 Case Nml = 20: (top-right), case Nml = 1000(bottom right), EnOpt (left): objective function optimization results (1000 simulations) 57
Figure 53 Case Nml = 20: (top-right), case Nml = 1000(bottom right): EnOpt and EnOpt-ML means of simulations...57
Figure 54 Case Nml = 20: Boxplot of optimized objective function....................................58
Figure 55 Case Nml = 1000: Boxplot of optimized objective function................................58
Figure 56 Case Nml = 20: simulations' results of EnOpt and EnOpt-ML with respect to true minimum values..60
Figure 57 Case Nml = 1000: simulations' results of EnOpt and EnOpt-ML with respect to true minimum values...60
Figure 58 Objective function means over 1000 simulations for EnOpt, and EnOpt-ML with ML model trained at different iterations..61
Figure 59 means of 1000 simulations for EnOpt and EnOpt-ML with ML model trained at different iterations...62
Figure 60 Reduced sample size N = 5 used at iteration 4 gradient (single simulation).........63
Figure 61 Reduced sample size N = 5 used at iteration 4 EnOpt and EnOpt-ML samples and ML model predicted samples...63
Figure 62 Reduced sample size N = 10 used at iteration 4 gradient (single simulation)......63
Figure 63 Reduced sample size N = 10 used at iteration 4 EnOpt and EnOpt-ML samples and ML model predicted samples...63
Figure 64 Reduced sample size N = 5 used at iteration 4 objective function optimization results (1000 simulations)..65
Figure 65 Reduced sample size N = 10 used at iteration 4 objective function optimization results (1000 simulations)..65
Figure 66 Reduced sample size N = 5(left)/N=10(right) used at iteration 4 EnOpt and EnOpt-ML means of simulations..65
Figure 67 Reduced sample size N = 5 used at iteration 4 Boxplot of optimized objective function 66
Figure 68 Reduced sample size N = 10 used at iteration 4 Boxplot of optimized objective function 66

Figure 69 Reduced sample size N = 5 used at iteration 4 : simulations' results of EnOpt and EnOpt-ML with respect to true minimum values 68
Figure 70 Reduced sample size N = 10 used at iteration 4 : simulations' results of EnOpt and EnOpt-ML with respect to true minimum values 68
Figure 71 Case reduced sample size N = 0 used at iteration 4 gradient (single simulation): 69
Figure 72 Case reduced sample size N = 0 used at iteration 4 EnOpt and EnOpt-ML samples and ML model predicted samples 69
Figure 73 case reduced sample size N = 0 used at iteration 5: gradient (single simulation) .69
Figure 74 Case reduced sample size N = 0 used at iteration 5 : EnOpt and EnOpt-ML samples and ML model predicted samples 69
Figure 75 Case reduced sample size N = 0 used at iteration 4 : objective function optimization results (1000 simulations) 71
Figure 76 Case reduced sample size N = 0 used at iteration 5 : objective function optimization results (1000 simulations) 71
Figure 77 Case reduced sample size N = 0 used at iteration 4 : EnOpt and EnOpt-ML means of simulations 72
Figure 78 Case reduced sample size N = 0 used at iteration 5 : EnOpt and EnOpt-ML means of simulations 72
Figure 79 Case reduced sample size N = 0 used at iteration 4 : Boxplot of optimized objective function 72
Figure 80 Case reduced sample size N = 0 used at iteration 5 : Boxplot of optimized objective function 73
Figure 81 Case reduced sample size N = 0 used at iteration 4 : simulations' results of EnOpt and EnOpt-ML with respect to true minimum values 74
Figure 82 Case reduced sample size N = 0 used at iteration 5 : simulations' results of EnOpt and EnOpt-ML with respect to true minimum values 74
Figure 83 AdaBoost case: gradient (single simulation) 75
Figure 84 AdaBoost case: EnOpt and EnOpt-ML samples and ML model predicted samples 75
Figure 85 Random Forest case: gradient (single simulation) 75
Figure 86 Random Forest case: EnOpt and EnOpt-ML samples and ML model predicted samples 75
Figure 87 AdaBoost case: objective function optimization results (1000 simulations) 76

Figure 88 Random Forest case: objective function optimization results (1000 simulations) 77
Figure 89 AdaBoost case: EnOpt and EnOpt-ML means of simulations 77
Figure 90 Random Forest case: EnOpt and EnOpt-ML means of simulations.....................77
Figure 91 AdaBoost case: Boxplot of optimized objective function 78
Figure 92 Random forest case : Boxplot of optimized objective function 78
Figure 93 AdaBoost case: simulations' results of EnOpt and EnOpt-ML with respect to true minimum values...79
Figure 94 Random Forest case: simulations' results of EnOpt and EnOpt-ML with respect to true minimum values...80
Figure 95 Improvement example: gradient (single simulation)..81
Figure 96 Improvement example EnOpt and EnOpt-ML samples and ML model predicted samples..82
Figure 97 Improvement example objective function optimization results (1000 simulations) 83
Figure 98 Improvement example EnOpt and EnOpt-ML means of simulations 83
Figure 99 improvement example Boxplot of optimized objective function 84
Figure 100 Improvement example simulations' results of EnOpt and EnOpt-ML with respect to true minimum values 85
Figure 101 Bukin function 6: gradient (single simulation)..86
Figure 102 Bukin function 6: EnOpt and EnOpt-ML samples and ML model predicted samples 87
Figure 103 Bukin function 6 objective function optimization results (1000 simulations) 88
Figure 104 Bukin function 6 EnOpt and EnOpt-ML means of simulations 88
Figure 105 Bukin function 6 Boxplot of optimized objective function.................................89
Figure 106 Bukin function 6 : simulations' results of EnOpt and EnOpt-ML with respect to true minimum values...90
Figure 107 Rosenbrock: gradient (single simulation)..91
Figure 108 Rosenbrock: EnOpt and EnOpt-ML samples and ML model predicted samples 92
Figure 109 Rosenbrock: objective function optimization results (1000 simulations) 93
Figure 110 Rosenbrock: : EnOpt and EnOpt-ML means of simulations...............................93
Figure 111 Rosenbrock: Boxplot of optimized objective function.......................................94

Figure 112 Rosenbrock: simulations' results of EnOpt and EnOpt-ML with respect to true minimum values..95

Acknowledgment

This master thesis is part of the requirements to fulfill the degree in the **Master of Science in Computational Engineering** at the University of Stavanger.

Above all, I want to express my gratitude to my parents, Fatima and Abdessamad, as well as my sister, Lamyae, for their constant encouragements.

I'd like to express my gratitude to both Prof. Aojie Hong, my internal supervisor, for his insightful suggestions, feedback and guidance and Kurt Petvipusit, my External supervisor from Equinor, for sharing some of his valuable time and expertise.

I'd also like to thank my wonderful friends, especially Vitalijus , for their continuous moral support.

Last but not least, I'd like to extend my appreciation to everyone who contributed, whether near or far, to my education at the University of Stavanger, especially my professors.

Abstract

High or even prohibitive computational cost is one of the key limitations of robust optimization using the Ensemble-based Optimization (EnOpt) approach, especially when a computationally demanding forward model is involved (e.g., a reservoir simulation model). It is because, in EnOpt, many realizations of the forward model are considered to represent uncertainty, and many runs of forward modeling need to be performed to estimate gradients for optimization. This work aims to develop, investigate, and discuss an approach, named EnOpt-ML in the thesis, of utilizing machine learning (ML) methods for speeding up EnOpt, particularly for the gradient estimation in the EnOpt method.

The significance of any deviations is investigated on three different optimization test functions: Himmelblau, Bukin function number 6 and Rosenbrock for their different characteristics. A thousand simulations are performed for each configuration setting to do the analyses, compare means and standard deviations of the ensembles. Singled out cases are shown as examples of gradient learning curves differences between EnOpt and EnOpt-ML, and the spread of their samples over the test function.

Objectives:

Objective1: Building of a code with a main function that would allow easy configurations and tweaking of parameters of EnOpt, Machine learning (ML) algorithms and test function or objective functions in general (with two variables). Codes necessary for test functions, ML algorithms, plotting and simulation data saving files are defined outside of that main function.

The code is attached in the *Appendix*.

Objective2: Testing and analysis of results to detect any special improvement with EnOpt-ML compared to EnOpt. The use of Himmelblau as a primary test function was with a modification of specific parameters, one at a time, starting with a base configuration case for possible comparisons. After gathering traits of effects of those configurations, an example where the improvement could show interesting were presented and then applied to the other two test functions and analyzed.

The main objective then has been to reduce the number of times the objective function is evaluated while not considerably reducing the optimization quality.

EnOpt-ML yielded slightly better results when compared to EnOpt under the same conditions when fixing a maximum objective function evaluations through the number of samples and the iteration at which this number is reduced.

Chapter 1: Introduction

Background and motivation:

Companies in different sectors implement optimization solutions to make informed decisions to increase profitability, improve efficiency and reduce costs. The purpose of optimization is to find the “best available” solution(s) of an objective function using a set of properties that are held constant throughout an analysis (i.e., control variables).

There are two approaches to optimization: gradient free and gradient based. The gradient free optimization approach has slower convergence but is capable of finding a global optimum whereas the gradient based approaches have a fast convergence, but the optimum can be either local or global. This thesis work will consider a gradient based optimization.

When the optimization problem at hand contains uncertain data, we should consider this incertitude. Robust optimization (RO) is a field of optimization theory that deals with optimization problems and accounts for uncertainty that can be represented as deterministic variability in the value of the parameters of the problem or its solution (Fonseca et al. 2017). RO used by Essen et al. (2009) on water flooding optimization with an adjoint-based method, for example, required access to reservoir simulator codes (usually inaccessible) and proved computationally intensive. A method independent of the simulator that is easier to implement is the Ensemble-based Optimization (EnOpt), it was proposed by Lorentzen et al. (2006) and Nwaozo (2006). EnOpt was then used by Chen et al. (2009) for RO where the gradient is approximated between the randomly perturbed control variables (random samples) and their objective function values.

The accuracy of EnOpt estimation relies on the sample size of the perturbed controls and the covariance matrix that generates those samples. The perturbed controls are commonly generated with a multivariate normal distribution with a pre-defined mean vector and covariance matrix. As one objective function evaluation needs to be performed for each set of perturbed controls, the number of objective function evaluations and the computational intensiveness of optimization increases with the number of sampled sets of perturbed controls.

This thesis aims to develop, investigate, and discuss an approach, named EnOpt-ML in the thesis, of utilizing machine learning (ML) methods for speeding up the EnOpt method. In EnOpt-ML, sets of perturbed controls and their corresponding objective values are used to train a ML model, and then the trained ML model is used for gradient estimation.

For investigating the performance of EnOpt-ML, various objective functions, configurations of EnOpt-ML, and ML algorithms are experimented. The main objective of the experiments is to identify an optimal combination of an EnOpt-ML configuration and ML algorithm that can reduce the number of objective function evaluations and computational cost.

Novelty of the work:

The key novelty of the thesis work is that the EnOpt-ML approach is proposed to incorporate ML in EnOpt for enhancing gradient estimation and speeding up optimization, and the approach is tested and discussed under various settings. To the best of the author's knowledge, this approach has not been addressed specifically in the literature.

Outline of the thesis:

The rest of the thesis is structured as follows:

- Chapter 2 presents key concepts/theories, equations/algorithms, and models used for the thesis work.
- Chapter 3 introduces the workflows of various experiments on EnOpt-ML and analyses the experiments results.
- Chapter 4 tests the optimal EnOpt-ML setting, determined based on the analysis in Chapter 3, on three objective functions.
- Chapter 5 summarizes the thesis work with a general discussion and conclusions.

Chapter 2: Concepts and approaches

In this chapter, principal concepts and components of the algorithm used throughout this thesis are presented.

Robust optimization (RO):

Material uncertainties should be considered for achieving high-quality decision making. The optimal values of control (or decision) variables for a risk-neutral decision maker are those that optimize the mean (or expected value) of the decision maker's objective function over the uncertainty. The optimal control variables are expected to be robust to any possible realization of the uncertainties, and thus optimizing the mean of the objective function under uncertainty is also referred to as Robust Optimization (RO).

Ensemble optimization (EnOpt):

Ensemble optimization (EnOpt) is a stochastic optimization method that uses an ensemble of realizations sampled via Monte Carlo simulation to account for and express uncertainties. Its goal is to maximize or minimize the mean of the objective function across the ensemble. It is a robust optimization (RO) strategy that is both simple and practical to implement (Hong et al. 2017).

As a gradient-based optimization algorithm, EnOpt requires a starting point (initial mean) and a covariance matrix to define the search area for gradient estimation. In EnOpt, the values of control variables are updated/improved iteratively towards their optimal values. In each iteration, an ensemble of perturbed controls is generated and evaluated (i.e., the objective value corresponding to one combination of the values of control variables is calculated), then the gradient is estimated based on the perturbed controls and their corresponding objective values, and finally the estimated gradient is used to update the values of the control variables. The iteration of search for the optimum continues until some stopping criteria are met. EnOpt is a promising RO approach, but when based on rich grid-based reservoir models with hundreds of realizations for example, it is computationally demanding (Hong et al 2017).

EnOpt can be sensitive to the user-defined starting point (i.e., the choice of initial mean) when there are several local optima, and it may not lead to the global optimum. The covariance matrix in EnOpt specifies a search area – changing the covariance matrix allows to widen/shrink the search area– which corresponds to a more global/local search strategy (Fonseca et. al. 2014). Despite its cheap computing cost, previous research has demonstrated that the EnOpt can efficiently and satisfactorily improve the objective function (Chen et al., 2009; Chen and

Oliver, 2010, 2012). However, EnOpt may not yield appropriate findings, for various geological models for example, unless the variation in the ensemble models is small enough, according to Fonseca et al. (2017).

Stochastic gradient descent:

Stochastic gradient descent (SGD) is a widely used algorithm for optimization . The basic idea of SGD is that it, in each iteration, searches for a smaller value of an objective function in a search direction with a certain learning rate (step size at each iteration), and such iteration repeats for smaller and smaller objective values to approach an unknown minimal (usually, near minimal) objective value until a certain criterion or criteria are fulfilled for stopping the iteration.

Machine learning algorithms:

The algorithms for the three ML regressors used in this thesis were made available by Sklearn. Sklearn (Pedregosa et al. 2011) is a free software machine learning library for the Python programming language that includes a number of tools for machine learning and statistical modeling.

Gradient Boosting Regressor:

Gradient boosting (GBR) is a machine learning approach for prediction studies that generates a model in the form of an ensemble. For optimizing the different levels of stages, GBR simplifies the random differences of loss function. Later, the gradient boosting algorithm was developed to optimize the cost function and iteratively select function points in the negative gradient direction (Friedman 2001). Gradient boosting combines weak “learners” (ML algorithms that perform slightly better than a random guess) into a single strong learner (models with good accuracy) in an iterative fashion.

There are two types of Gradient boosting: classifier and regressor. It is a regressor in this thesis because we are predicting continuous values.

Adaptive Boosting (AdaBoost) Regressor:

Adaptive boosting Regressor is similar to GBR with the distinction being that the AdaBoost is, as its name suggest, adaptively adjusting to errors on weak hypotheses (hypotheses concentrating on a specific feature) (Freund et.al. 1995). AdaBoost does not need prior knowledge of the accuracies of the weak hypotheses as it adapts to them and generates a weighted majority hypothesis in which the weight of each weak hypothesis is a function of its accuracy (Freund et.al. 1995).

Random Forest:

Random Forest is an ensemble of decision trees in which the outputs of all trees are aggregated to give one final prediction, which is, in regression , the average of the individual tree predictions (Svetnik et al. 2003). While growing the trees, Random Forest adds more randomization to the model.

EnOpt-ML:

EnOpt-ML is a modified EnOpt method that uses machine learning algorithms to reduce the computational cost of EnOpt. It trains the ML model on the samples from the real objective function and predicts more samples that are also used in the gradient estimation. EnOpt-ML method is explained through the diagram in figure 1 :

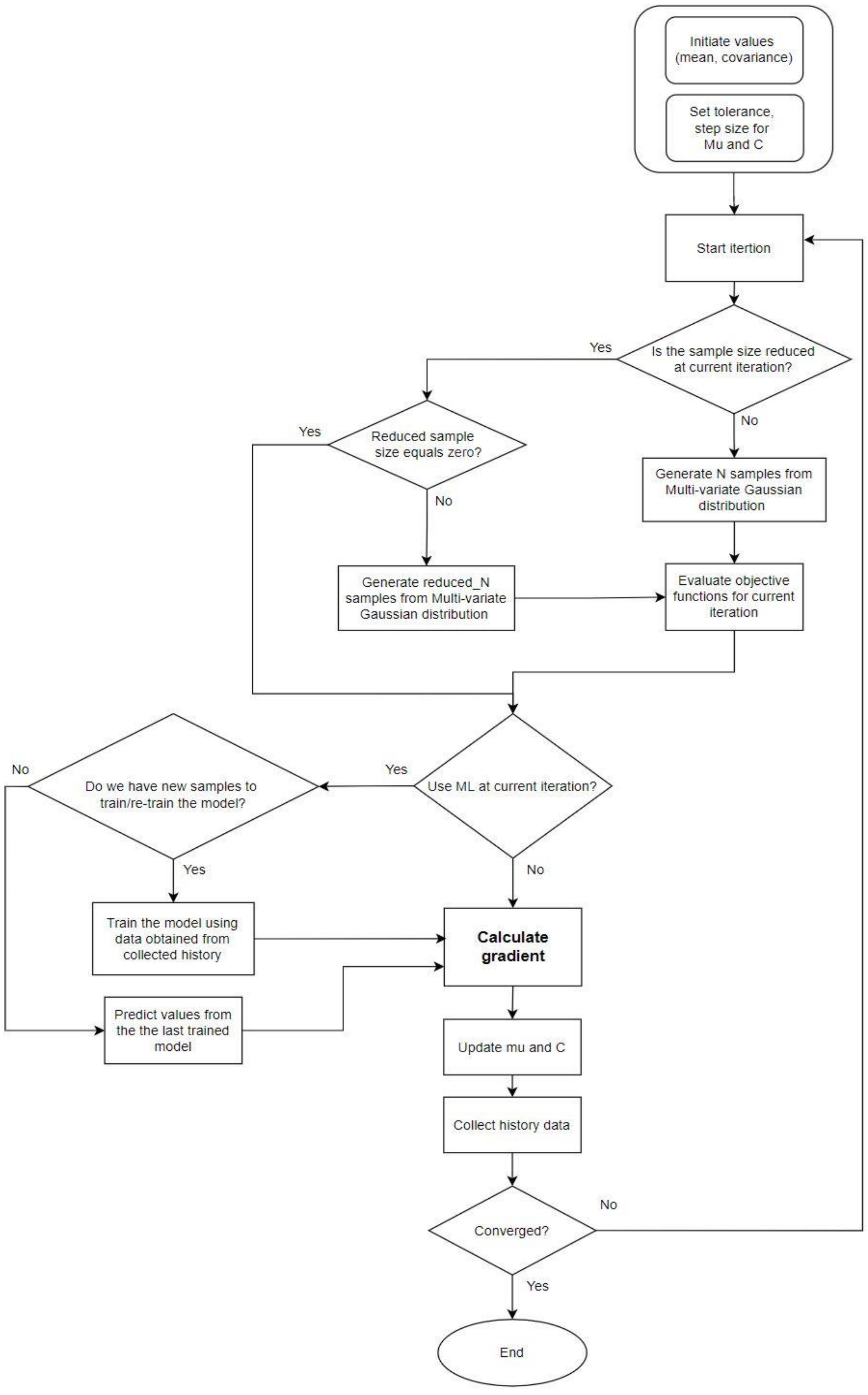

Initiate values (mean, covariance)
Set tolerance, step size for Mu and C
Start itertion
Is the sample size reduced at current iteration?
Yes
No
Reduced sample size equals zero?
Yes
No
Generate N samples from Multi-variate Gaussian distribution
Generate reduced_N samples from Multi-variate Gaussian distribution
Evaluate objective functions for current iteration
Use ML at current iteration?
Yes
No
Do we have new samples to train/re-train the model?
No
Yes
Train the model using data obtained from collected history
Predict values from the the last trained model
Calculate gradient
Update mu and C
Collect history data
Converged?
No
Yes
End

Figure 1Workflow chart of EnOpt-ML

Optimization test functions:

Presented in the following are the three optimization test functions used in this thesis: Himmelblau, Rosenbrock and Bukin function number 6. Himmelblau is used in chapters 3 and 4 while Rosenbrock and Bukin function number 6 are only used in chapter 4.

Himmelblau: f(x, y) = (x2 + y – 11)2 + (x + y2 -7)

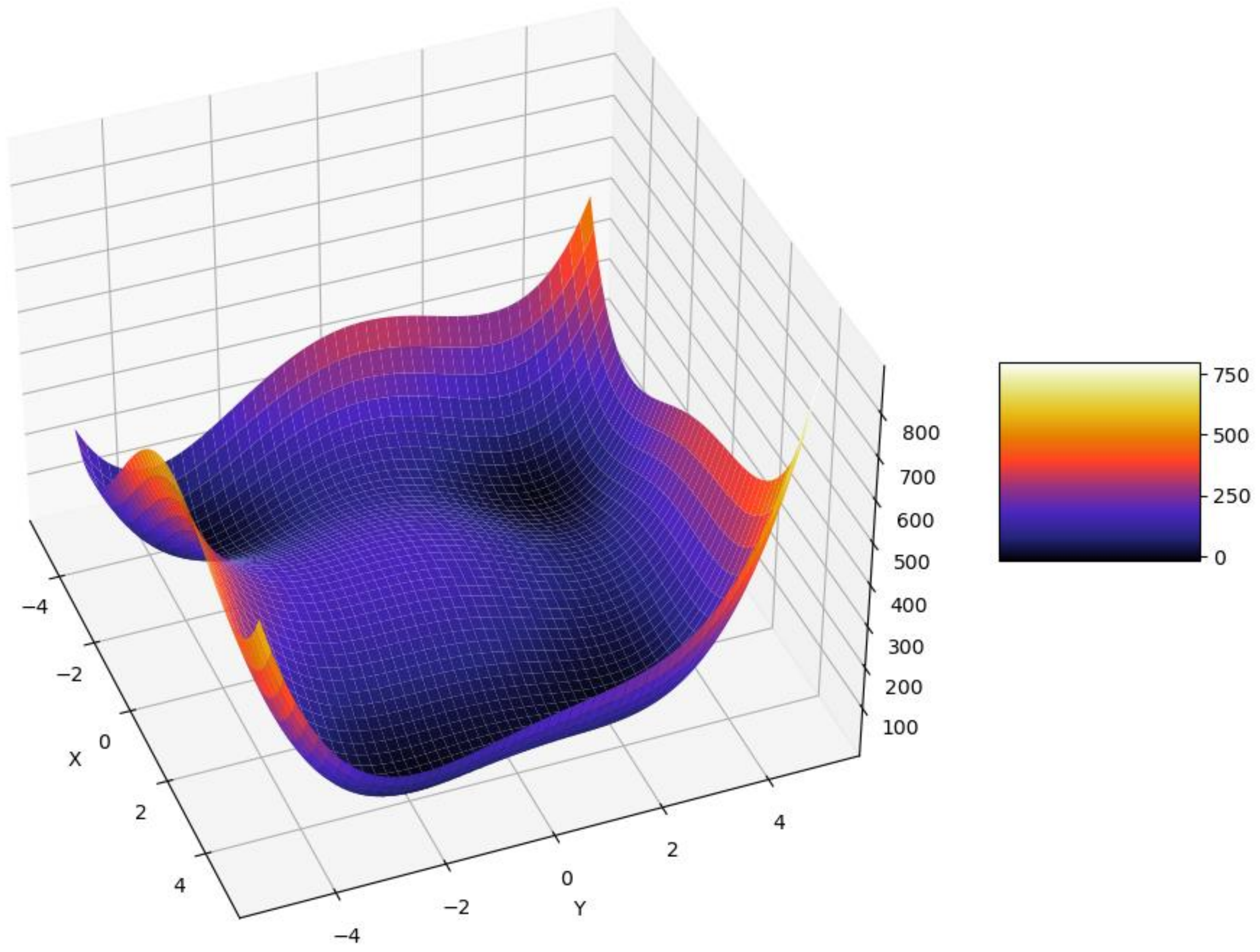


Figure 2 Himmelblau. Test function for optimization

The Himmelblau function was chosen for the challenges that different minima would present:

- four identical local minima at different (x, y) locations:

f(x,y) = 0 at (x,y) = (3, 2)

f(x,y) = 0 at (x,y) = (−2.805118, 3.283186)

f(x,y) = 0 at (x,y) = (−3.779310, −3.283186)

f(x,y) = 0 at (x,y) = (3.584458, −1.848126)

- Evaluated on the rectangle x ∈ [-5, 5], y ∈ [-5, 5].

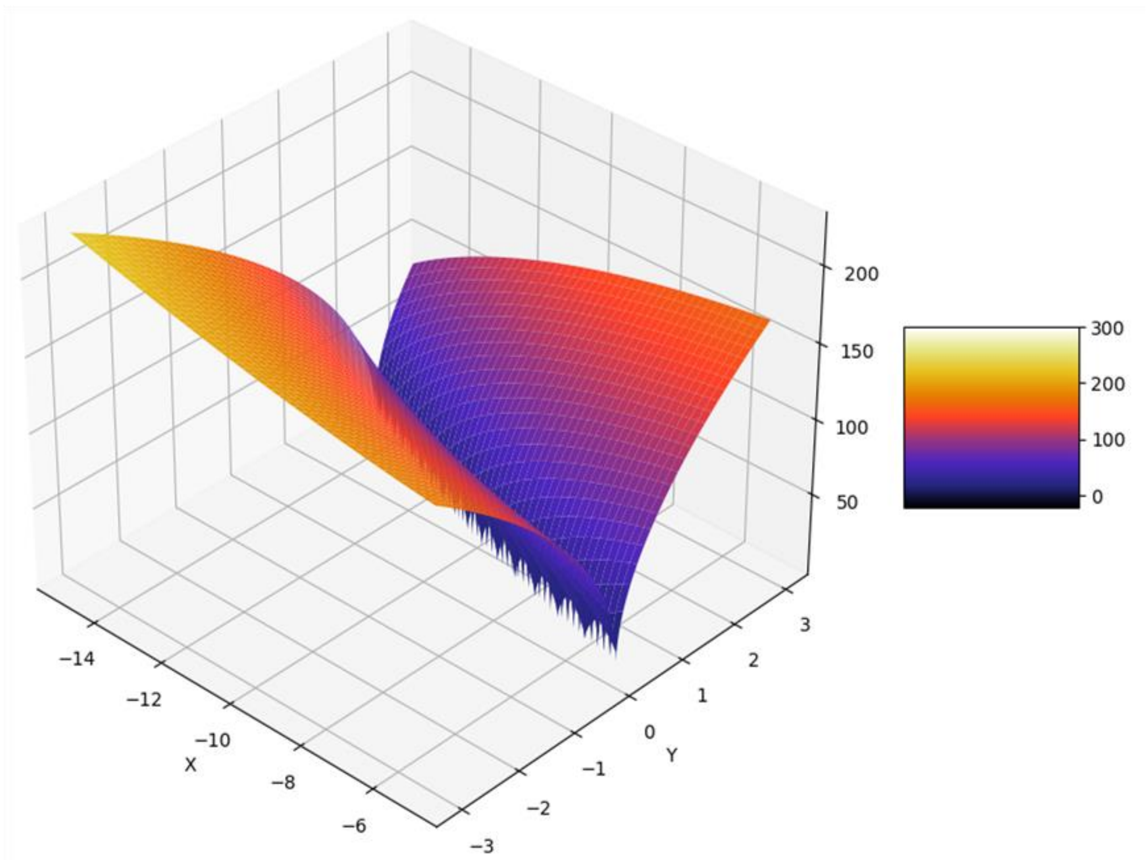


Figure 3 Bukin function number 6. Test function for optimization

Bukin's function # 6: $f(x, y) = 100 \sqrt{|y - 0.01 x^2|} + 0.01|x+10|$

The Bukin function number 6 has been chosen to test the optimization on an example for function with a ridge region.

<u>Some Characteristics of the Bukin function number 6:</u>

- One global minimum:

$f(x,y) = 0$ at $(x,y) = (-10, 1)$

- Numerous local minima all of which are located in a ridge.
- Evaluated on the rectangle $x \in [-15, -5]$, $y \in [-3, 3]$.

Rosenbrock: $f(x, y) = (x - 1)^2 + 10(y - x^2)^2$

(also known as: Rosenbrock's valley or Rosenbrock's banana function)

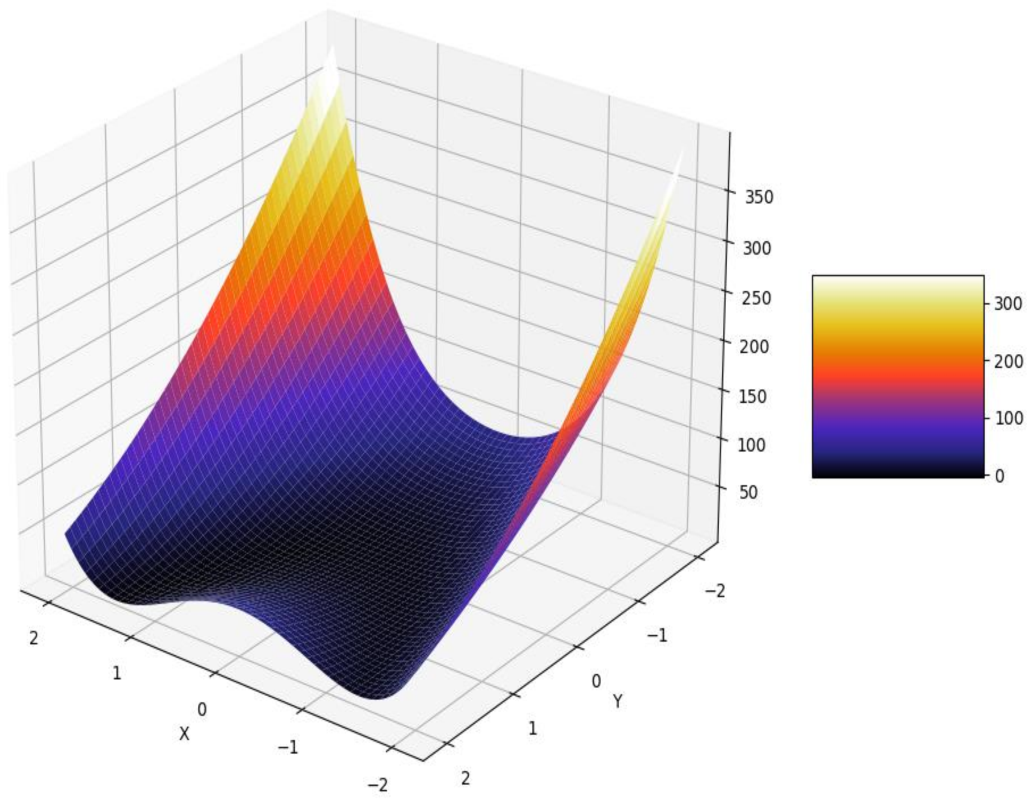


Figure 4 Rosenbrock. Test function for optimization (evaluated in the interval x ∈ [-2.048, 2.048], y ∈ [-2.048, 2.048])

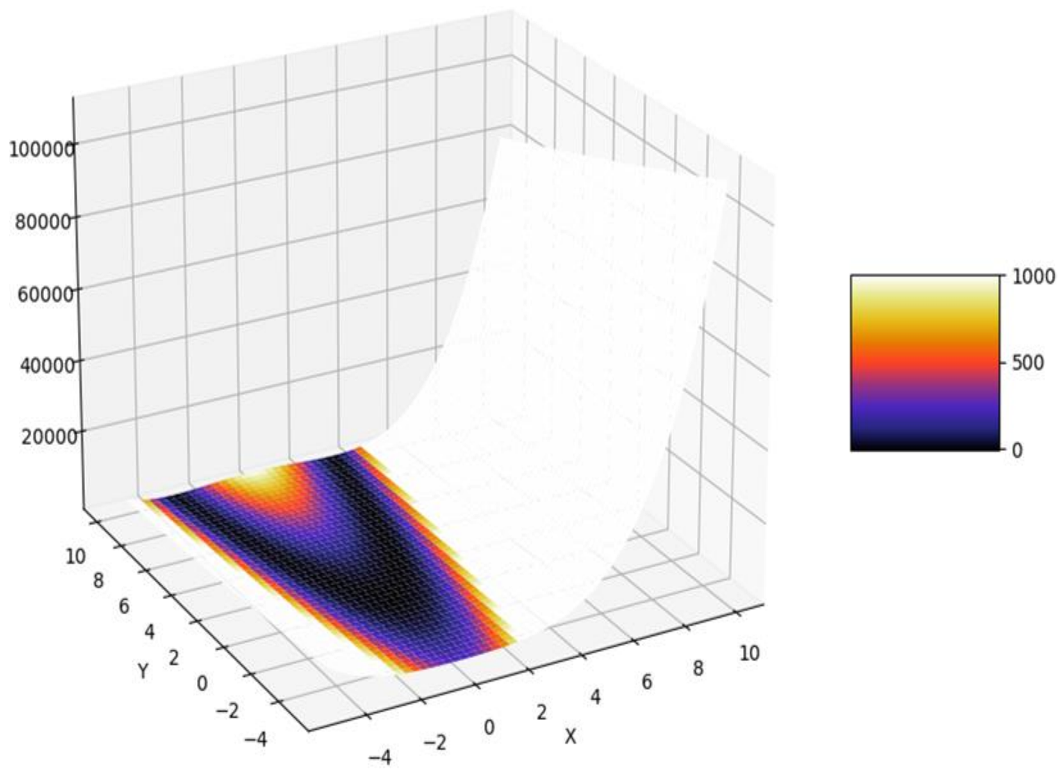


Figure 5 Rosenbrock. test function for optimization (evaluated in the interval x ∈ [-5, 10], y ∈ [-5, 10])

The Rosenbrock function has been chosen to test the optimization on an example for function with a valley-like region.

<u>Some Characteristics of the Rosenbrock function:</u>

- One global minimum:

f(x,y) = 0 at (x,y) = (1, 1)

- The function is unimodal, with a narrow, parabolic valley as the global minimum. Despite the ease with which this valley can be found, convergence to the minimum is not easy (Picheny et al., 2012)
- Evaluated on the rectangle $x \in [-5, 10]$, $y \in [-5, 10]$

Chapter 3: Experiments on EnOpt-ML

Introduction:

The main goal of this chapter is to investigate the impact of machine learning algorithms on stochastic gradient descent (SGD). The Himmelblau function is used for optimization testing in the present chapter. The algorithms that were written in the Python programming language (version 3.8.8) and are available in Appendix A. A function called "run(.)", in the thesis Python code, has been created to facilitate the change in parameters for testing of EnOpt and EnOpt-ML. As long as a result in optimization is less than 10% of the range of the function, it is considered "good enough" in these chapters.

Workflow:

1. Definition of the starting point (initial mean) and initial covariance matrix that the optimization needs to start from,
2. In the function run(.) defined in Appendix A, several parameters are varied for testing different EnOpt and EnOpt-ML configurations. Those parameters are:
    - Mean (mu) and covariance (C): are mandatory fields where mu is the starting point and C is the initial covariance matrix.
    - Initial sample size (init_N)
    - Reduced sample size (reduced_N): in case of reduction of sample size, this parameter defines the number of samples used at a defined iteration and in the following iterations. in the case that reduced_N= 0, No more evaluations on the original objective function are performed.
    - Number of samples predicted by the machine learning model (Nml)
    - The objective function used (of)
    - The maximum number of iterations (max_iter) as a stopping criterion, for the optimization
    - Train iteration (train_iter): the number of iterations after which the ML model starts the learning process and being used.
    - Reduce at iteration (reduce_at_iter): iteration at which the number of samples is reduced (i.e., when 'reduced_N' is used)
    - Machine learning algorithm (ML): the machine learning algorithm that will be used.

    Note: run(.) is not the only function used in the process, but it is the main function for optimization with EnOpt and EnOpt-ML in this thesis work. Other functions specify, for instance, the test functions and the machine learning algorithms needed/used. In addition, there is a sequence of *for* loops, outside of run(.) that goes through different configurations and simulations on this main function(see Appendix A).

- Other configurations that have to be specified for the different simulations:
    - » seed: the *seed()* method in Python is used to get the random number generator started, this method needs a starting value which we specify with this configuration. If the *seed* value is specified and unchanged, we will always get the same values of the optimizations so long as the other parameters are unchanged. If the *seed = None*, it will indicate that we are getting different samples (because of the unfixed seed) at each rerun of the function. The latter is useful when we need to perform multiple simulations for more robust results.
    - » sims: the number of simulations performed on the function. When *sims >1*, the *seed* has to be set to *None*

3. Analysis of results and recommended practices of using EnOpt-ML.

Base case configuration:

In the base configuration -configuration from which sensitivity analyses are going to be derived- the optimization test function is Himmelblau on which it is applied both EnOpt and EnOpt-ML with the use of machine learning based perturbed sample evaluation approach on the gradient. The latter will be referred to in this thesis as EnOpt-ML. the GBR is used in this configuration case as a ML technique.

The initial mean is at (0, 0) , the initial covariance matrix is [0.1 0 0 0.1], and the rest of the configuration is as follows:

config_sims = 1

(config_sims= 1 for a single simulation. config_sims>1 for simulations repeated a number config_sims with different generated samples)

config_init_N = [20]

config_reduced_N = [5]

config_max_iter = [100]

config_train_iter = [999, 1]

(config_train_iter = 999 for EnOpt (never reaching train iteration for ML), and config_train_iter = 1 is for the use of ML at 2nd iteration)

config_reduce_at_iter = [999]

(config_reduce_at_iter =999 is so that config_reduced_N is not used/reached)

config_seed = 1000

(the seed = 1000 (randomly chosen value) is set for single simulation cases for comparison purposes with other configurations. When seed = None, the simulations are repeated with different seeds (generating different samples) for multiple simulations.

After running the base configuration for optimization, the following results are obtained:

- <u>for a single simulation:</u>

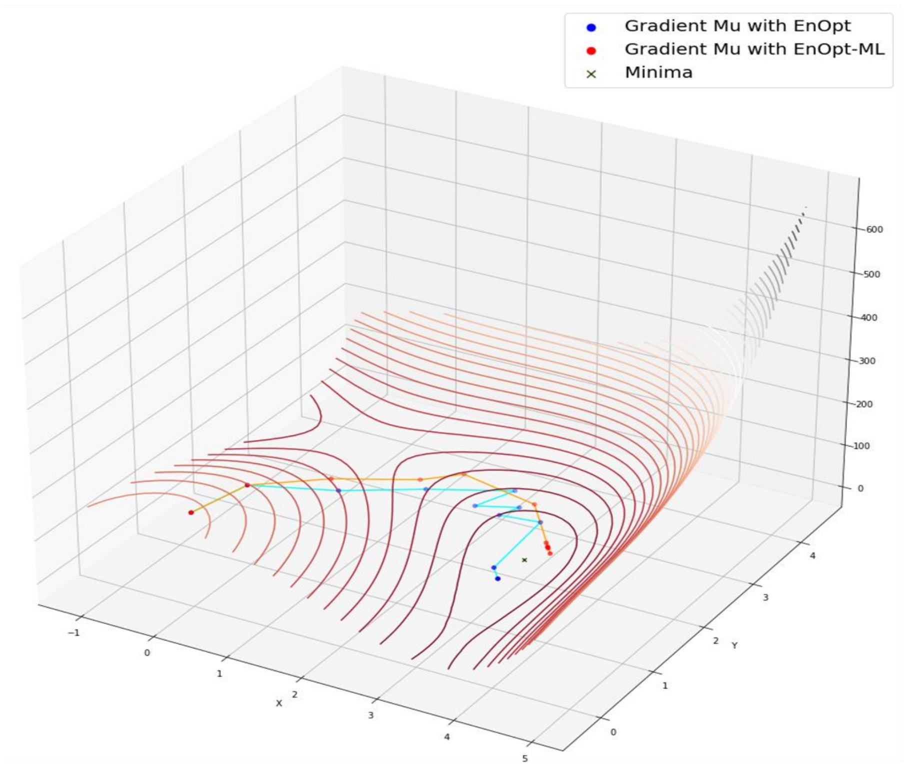


Figure 6 Base configuration gradient (single simulation)

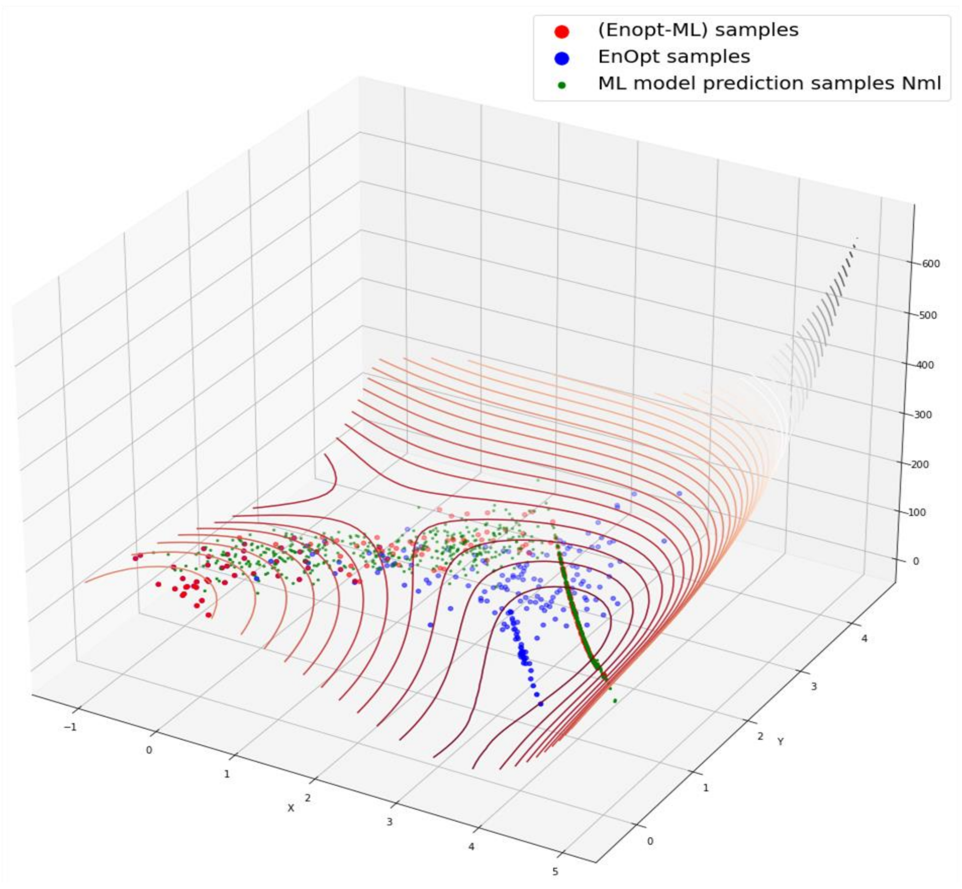


Figure 7 Base configuration EnOpt and EnOpt-ML samples and ML model predicted samples

Mean square error (MSE): 34.39

MSE of EnOpt-ML refers to the mean square error between the trained ML model predictions of the objective function and the actual objective function. It will be abbreviated as MSE in the rest of this thesis' following chapters.

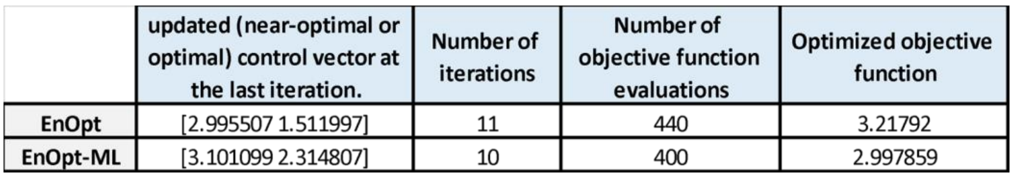
The MSE of the GBR shows a relatively small value. A small value of MSE indicates a good fit between the model predicted samples and the test function.

in (fig. 6) EnOpt_ML gradient moved in the same way as EnOpt gradient for the first 5 iterations. After the 5th iteration, EnOpt-ML gradient continued in a linear-like trajectory while EnOpt gradient was zigzagging. This EnOpt-ML behavior could be due to that, in EnOpt-ML, the samples used to predict the ML samples are collected from all previous iterations and, after a number of iterations, new samples will have less effect on the training of the ML model.

The samples of both methods have the distribution of samples shown in (fig. 7), it offers a picture of the change in covariance matrix and its effect on the sampling as the covariance matrix decreases in size. Because of this, In the last iterations, (earlier in EnOpt-ML than with EnOpt) the samples are distributed in narrower trajectory (fig.7).

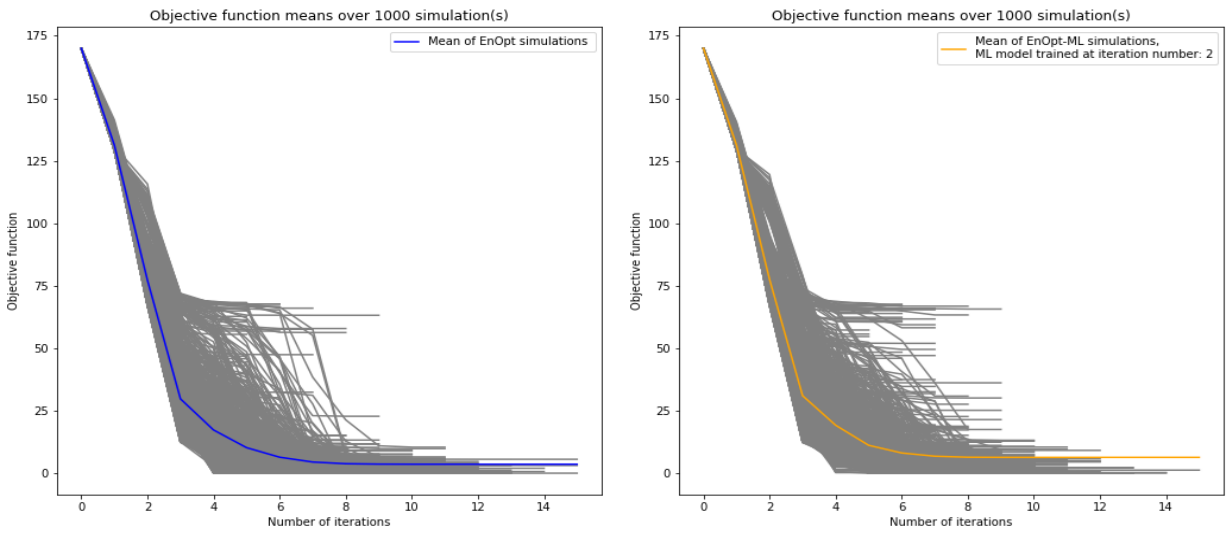
In this simulation, EnOpt has performed one extra iteration compared to EnOpt_ML (the latter evaluated 40 fewer objective function). Both the optimal control vectors and the optimized objective functions were very close in value both to each other (table 1) and to the local minima (optimal vector being [3 2] and optimal objective function equal to 0).

Table 1 Optimization results for base configuration case (single simulation)



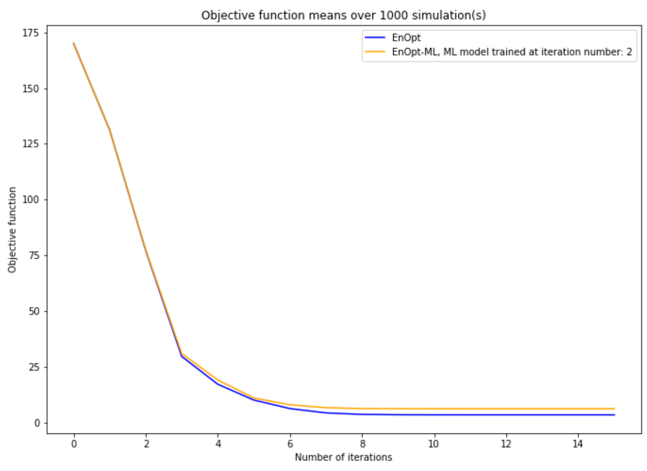
	updated (near-optimal or optimal) control vector at the last iteration.	Number of iterations	Number of objective function evaluations	Optimized objective function
EnOpt	[2.995507 1.511997]	11	440	3.21792
EnOpt-ML	[3.101099 2.314807]	10	400	2.997859

- <u>1000 simulations:</u>



-

- *Figure 8 Base confirmation cases objective function optimization results (1000 simulations)*



-

- *Figure 9 Base configuration case: EnOpt and EnOpt-ML means of simulations*

For the 1000 simulations case, EnOpt-ML resulted in a slightly higher objective function mean and standard deviation through the simulations in average(table 2). EnOpt on the other hand performed an average number of objective function evaluations 8% higher than EnOpt-ML. the simulations of EnOpt and EnOpt-ML show very close values throughout the iterations (fig. 9). Figure 8 shows all the different 1000 simulations (in gray) in each method, where we can appreciate .that the standard deviation between the simulations of EnOpt and EnOpt-ML are not very different ; 9.24for EnOpt and 10.77 for EnOpt-ML (table 2). The mean of EnOpt and

EnOpt-ML objective functions are very similar along the iterations (fig. 9) and the results show a slightly better but almost half the value of optimization in EnOpt than EnOpt-ML (table 2)

The quartiles from the boxplot (fig. 10) show that the maximum, minimum, 25% and 50% quartiles in the objective function values obtained are very close to one another, but the 75% quartile is double the value in EnOpt-ML with respect to EnOpt.

In (table 2), we see that the number of objective function evaluation is very close with 8% higher value for EnOpt compared to EnOpt-ML.

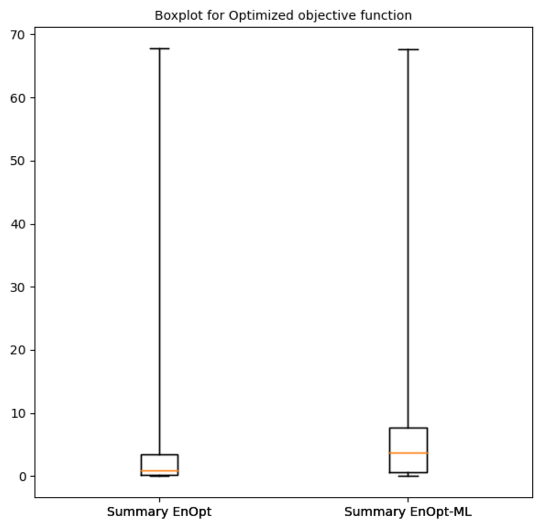


Figure 10Base configuration case: Boxplot of optimized objective function

Table 2 Base case configuration summary output values (1000 simulations)

EnOpt	iterations	Number of objective function evaluations	Optimized objective function	EnOpt-ML	iterations	Number of objective function evaluations	Optimized objective function
mean	8.34134134	333.6536536	3.643425409	mean	7.685685686	307.4274274	6.380151037
std	1.79416852	71.76674086	9.241104092	std	1.588190796	63.52763186	10.77345233
min	4	160	0.000012	min	4	160	0.000009
25%	7	280	0.2061915	25%	6.5	260	0.592244
50%	8	320	0.866076	50%	8	320	3.674649
75%	9	360	3.3699605	75%	9	360	7.69015
max	15	600	67.753509	max	15	600	67.598196

As introduced in Chapter 2, the Himmelblau test function has four local minima. The simulations with both EnOpt and EnOpt-ML with an initial vector [0 0], yielded values that mostly fall in the lower-right quadrant of (fig. 11). Few simulations produced values closer to the minima in the top-right quadrant and even fewer values were close to the lower-left one. No values were close to the minimum on the top-left area of (fig. 11).

Another observation is that the blue dots, representing EnOpt gradient optimization values, land closer to the top-right minima, while the red dots, EnOpt-ML gradient optimization values, are mostly stuck on the slope leading to that minimum.

More EnOpt-ML simulations got closer to the lower-left minima than EnOpt-ML.

The distribution of EnOpt and EnOpt-ML simulation (fig. 11) plot of results shows that EnOpt and EnOpt-ML can end up on distinct optimal values.

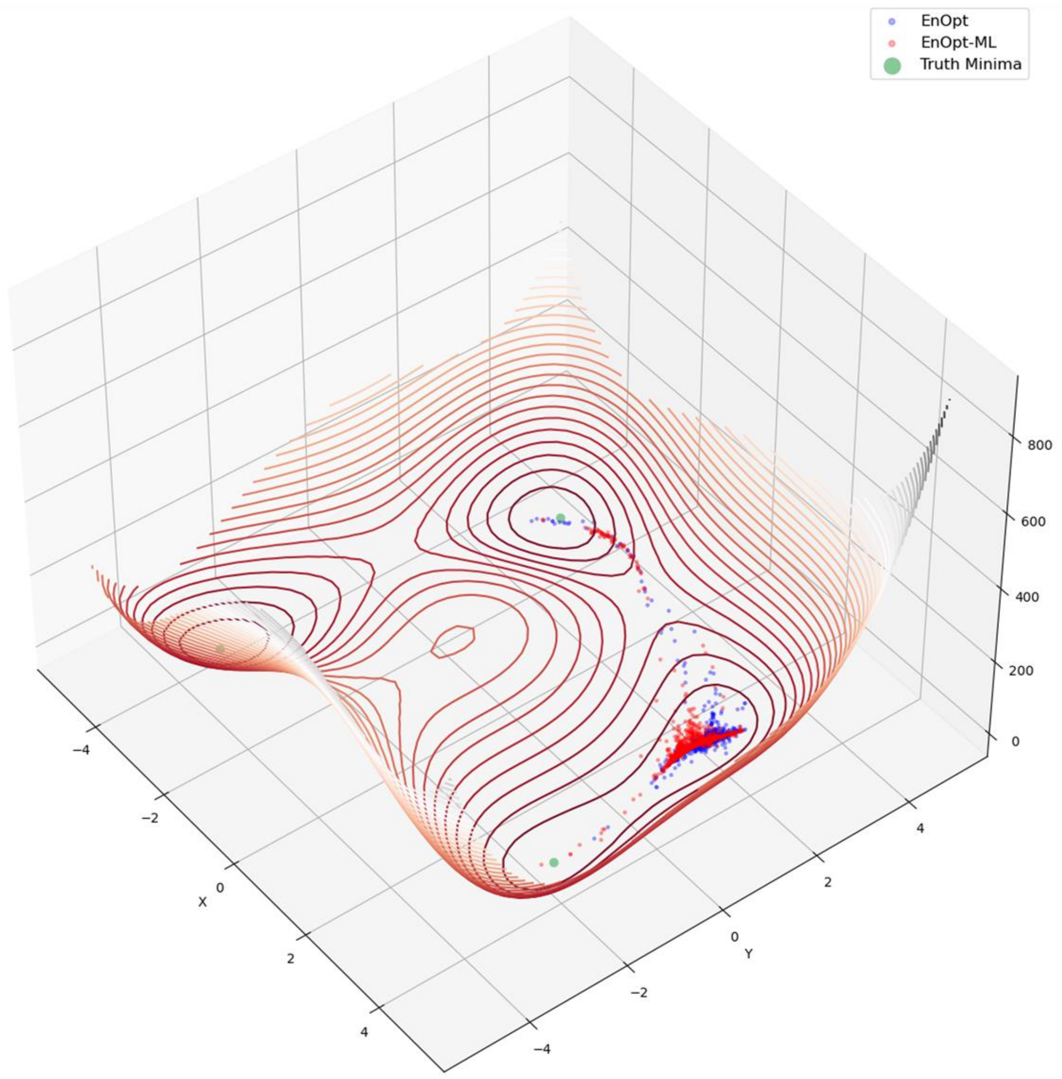


Figure 11 Base case simulations' results of EnOpt and EnOpt-ML with respect to true minimum values

Changing the starting point

The initial point is known to affect the EnOpt results. The choice of the initial point is made to get an example of starting on a steep region versus a starting on a flat region.

Starting at [-5 2] is for the case of steep region start, while starting at [3 -1] is for the flat region start case.

Note: all other configurations are kept unchanged.

Starting point [-5 2]:

- Single simulation:

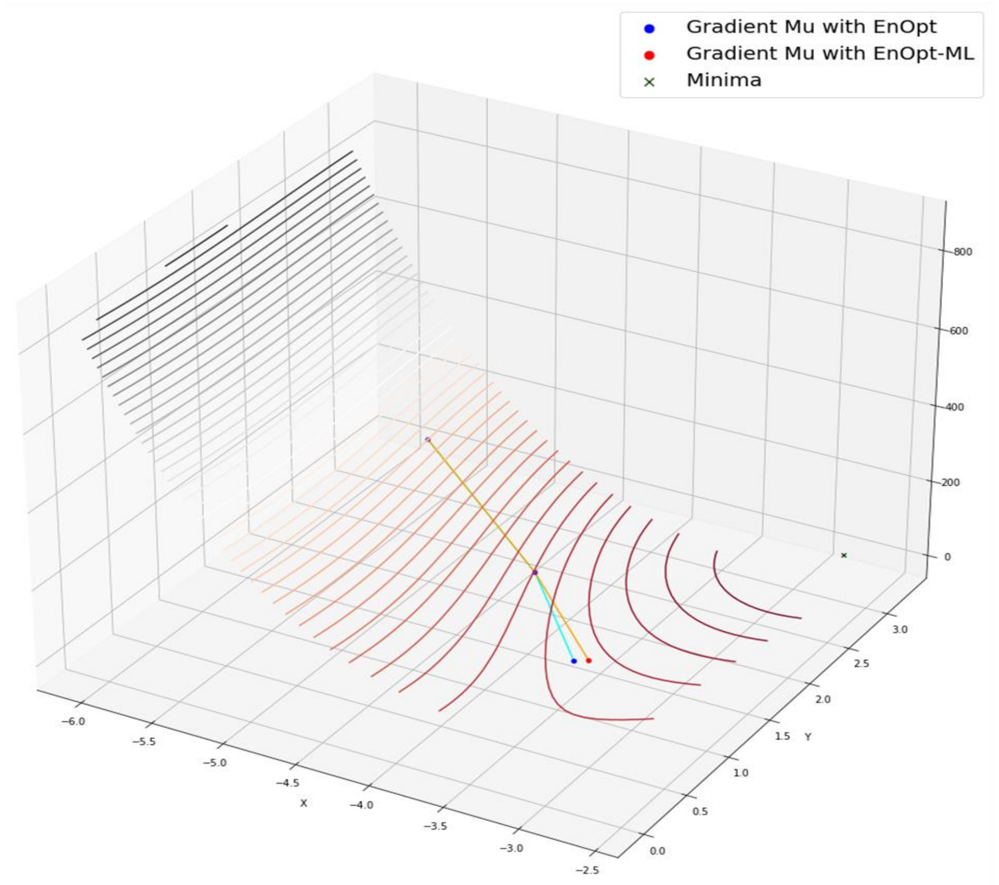


Figure 12 case starting point [-5 2] gradient (single simulation)

MSE: 44.77

The MSE is small, this mean that the errors on the ML model are small.

When we start at [-5 2] on this simulation, The objective function optimization stops after two iterations for both EnOpt and EnOpt-ML (fig. 12). The closest of the four minima of the Himmelblau is not approached enough and this is due to the flatter region on the final iteration.

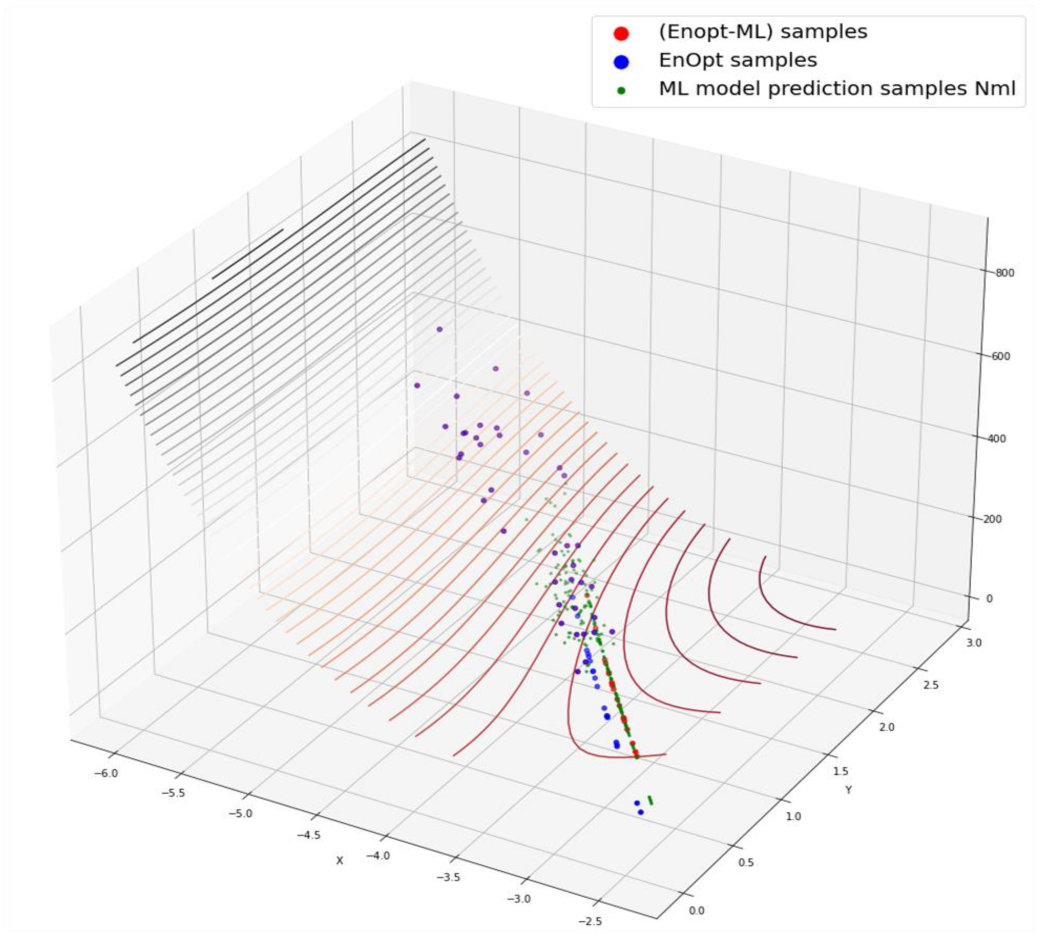


Figure 13 Case starting point [-5 2] , EnOpt and EnOpt-ML samples and ML model predicted samples

Figure 13 shows the distribution of the samples of EnOpt and EnOpt ML. we see a linear-like distribution of samples after the second as the covariance matrix became smaller faster. The covariance did not allow the samples to be taken from a wider area and it was then not possible for the gradient to go closer to the closest local minimum.

Very few iterations were performed due to the fact that the tolerance set as stopping criterion is 10^{-3} was reach early on the iteration process.

Table 3 Optimization results for case starting point [-5 2] (single simulation)

	updated (near-optimal or optimal) control vector at the last iteration.	Number of iterations	Number of objective function evaluations	Optimized objective function
EnOpt	[-3.364432 0.949934]	3	120	91.141739
EnOpt-ML	[-3.302379 1.014721]	3	120	86.830536

A slight improvement is seen when using EnOpt-ML In comparison with EnOpt (table 3). EnOpt yielded 91.14 while EnOpt-ML yielded 86.83 as an optimal value. Both values are high compared to the actual minimum that is zero and to the base configuration case (≈ 3 for both EnOpt and EnOpt-ML (table 1)).

The same number of objective function evaluations were performed on both methods (120 times for EnOpt and EnOpt-ML, table 3)

- <u>1000 simulations:</u>

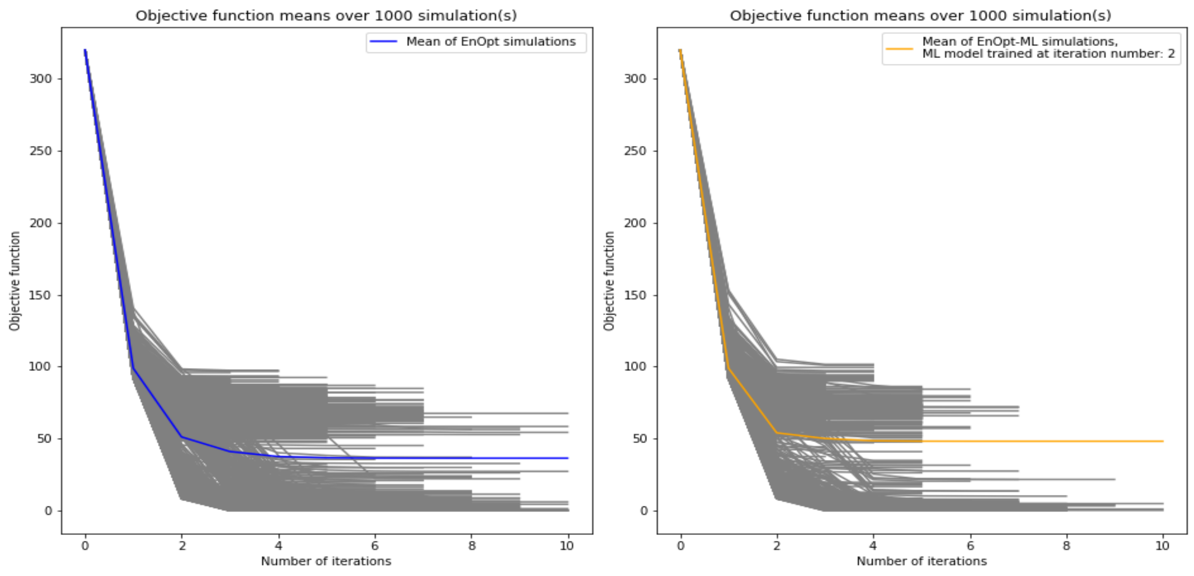


Figure 14 Case starting point [-5 2] : objective function optimization (1000 simulations)

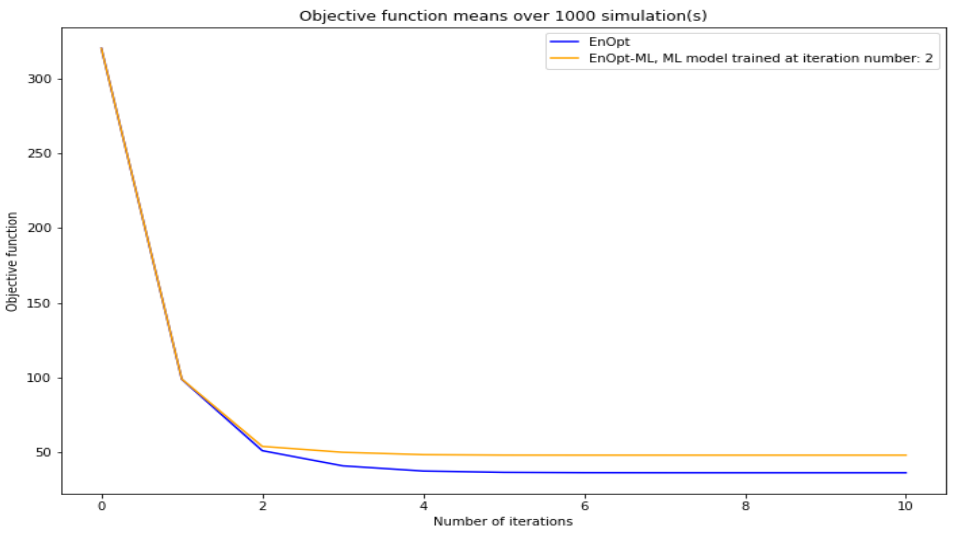


Figure 15 case starting point [-5 2] : EnOpt and EnOpt-ML means of simulations

When the simulation is repeated 1000 times with different random seeds, the means of objective functions are showing a better optimization with EnOpt compared to EnOpt.-ML (fig. 15). EnOpt gave in average 36.47 while EnOpt-ML gave 48.31 which is approximately 20% higher. They both have almost the same standard deviation (33.58 EnOpt versus 33 for EnOpt-ML, table 4). The optimized objective functions also show a higher mode (50% quartile, figure 22). 25% quartile value (fig22) is also closer to the minimum on the boxplot for EnOpt than it is for EnOpt-ML.

In (fig. 14) we see the mean of the different simulations tend to stop iterating in earlier iterations in EnOpt-ML compared to EnOpt.

42 more objective function evaluations were obtained, in average, with EnOpt compared to EnOpt ML (table 4).

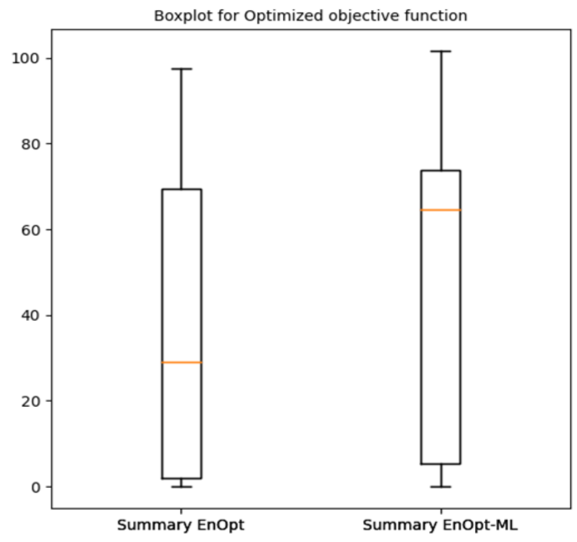


Figure 16 Case starting point [-5 2] : Boxplot of optimized objective function

Table 4 Case starting point [-5 2] : summary output values (1000 simulations)

EnOpt	iterations	Number of objective function evaluations	Optimized objective function	EnOpt-ML	iterations	Number of objective function evaluations	Optimized objective function
mean	5.40540541	216.2162162	36.46503814	**mean**	4.361361361	174.4544545	48.31079311
std	1.73575243	69.43009704	33.58563399	**std**	1.474753656	58.99014623	33.00700032
min	3	120	0.000255	**min**	3	120	0.002446
25%	4	160	1.940359	**25%**	3	120	5.3171915
50%	5	200	29.05168	**50%**	4	160	64.675557
75%	7	280	69.359249	**75%**	5	200	73.72312
max	10	400	97.475792	**max**	10	400	101.6413

In summary, we can say that we got 20% less objective function evaluations in EnOpt-ML than in EnOpt, but the Optimizations results are 20% higher in EnOpt-ML compared to EnOpt.

The plot of 1000 simulations results (fig. 17) show that, with this starting point [-5 2], none of the optimizations (be it with EnOpt or EnOpt-ML) have landed in any other minima than the one on the top-right quadrant of this figure.

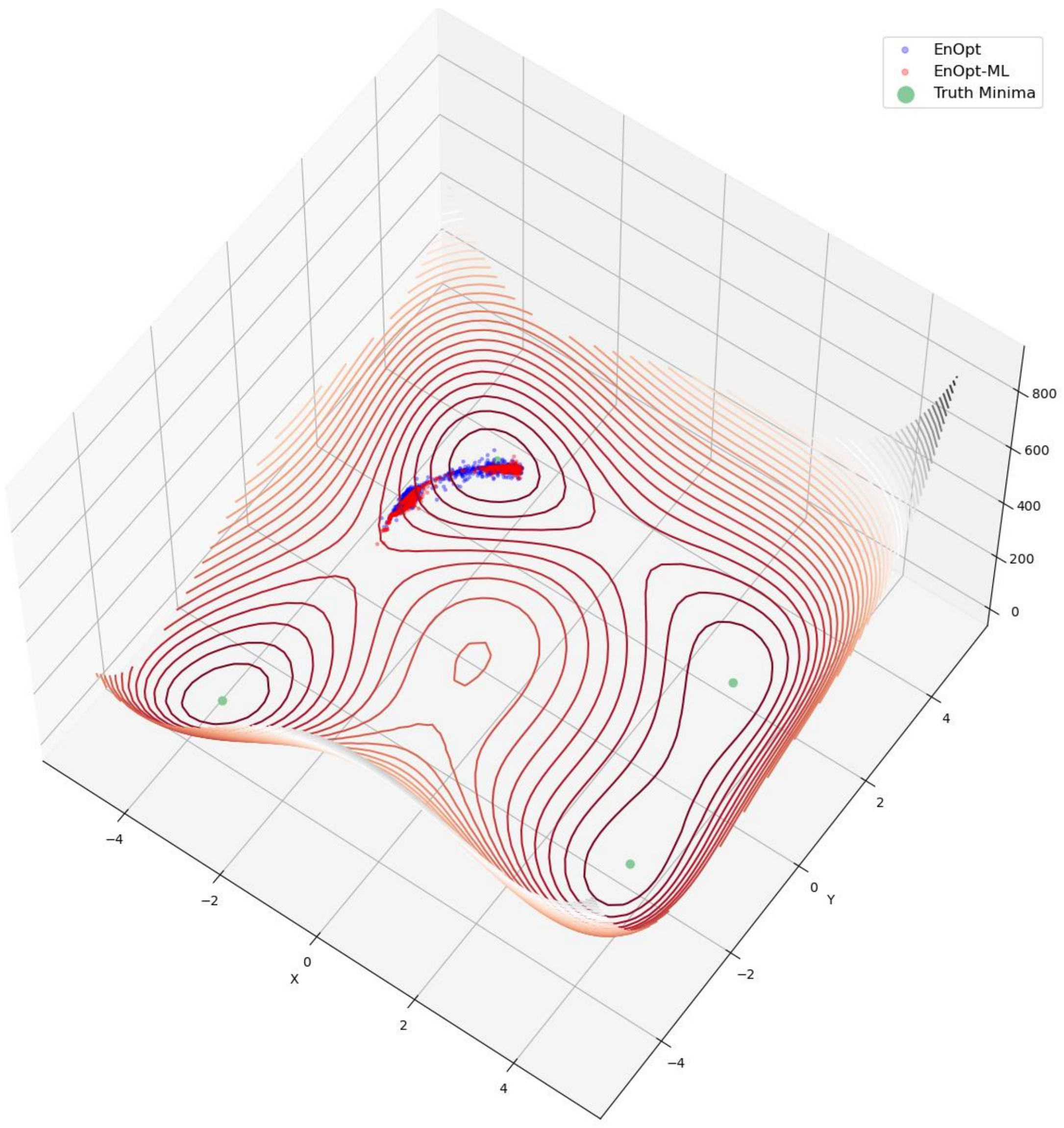


Figure 17 Case starting point [-5 2] : simulations' results of EnOpt and EnOpt-ML with respect to true minimum values

Starting point [3 -1]:

<u>Single simulation:</u>

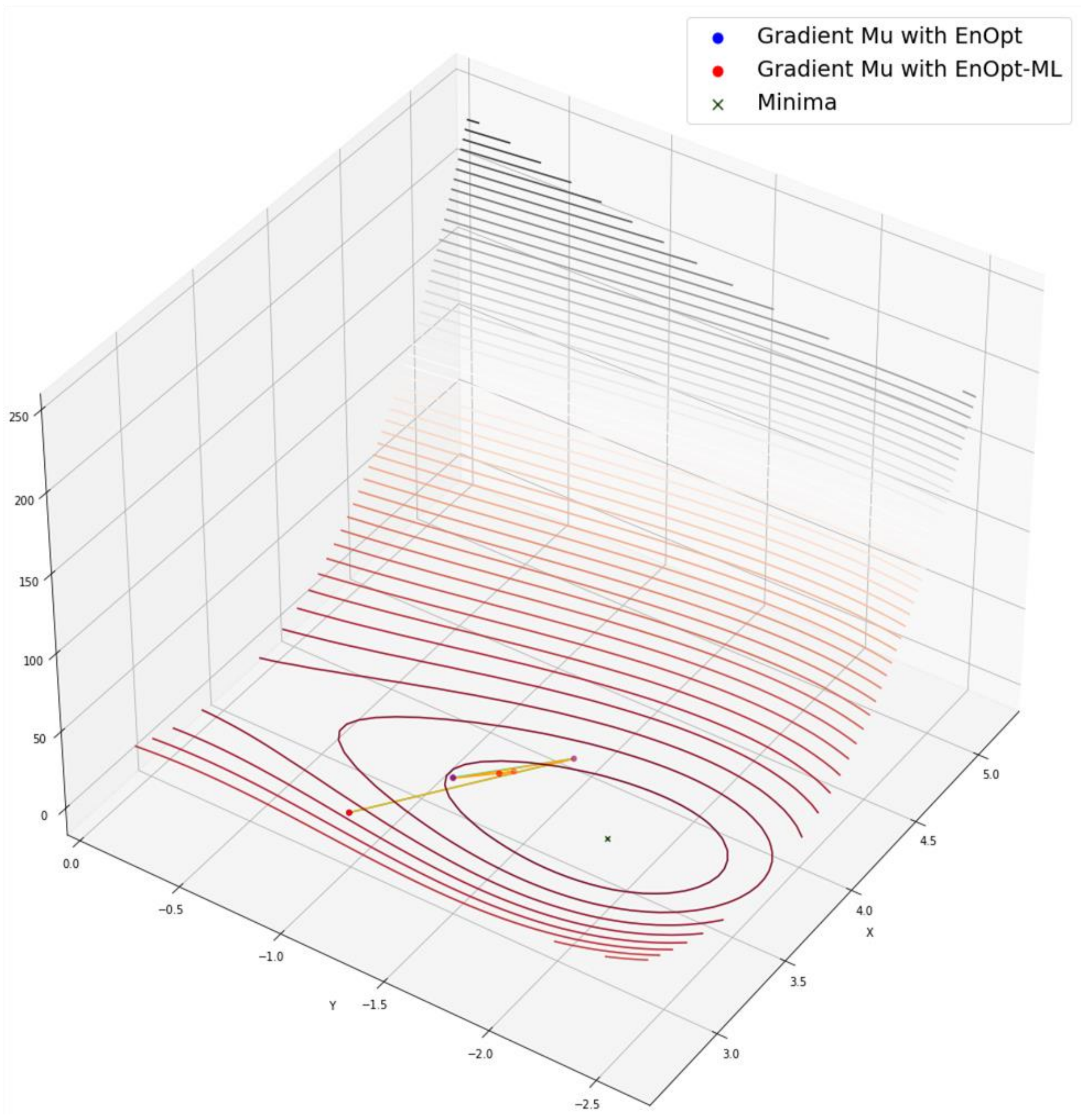


Figure 18 Case -   Starting point [3 -1]: gradient (single simulation)

MSE: 6.17

MSE is very low which shows a good ML model prediction.

Almost the same behavior in this case with starting point [3 -1](fig. 18) as the previous case as we are again in a relatively flat region. But since we started on a lower slope, the simulations gave better optimization results , EnOpt -ML(3.81) and EnOpt (4.71).

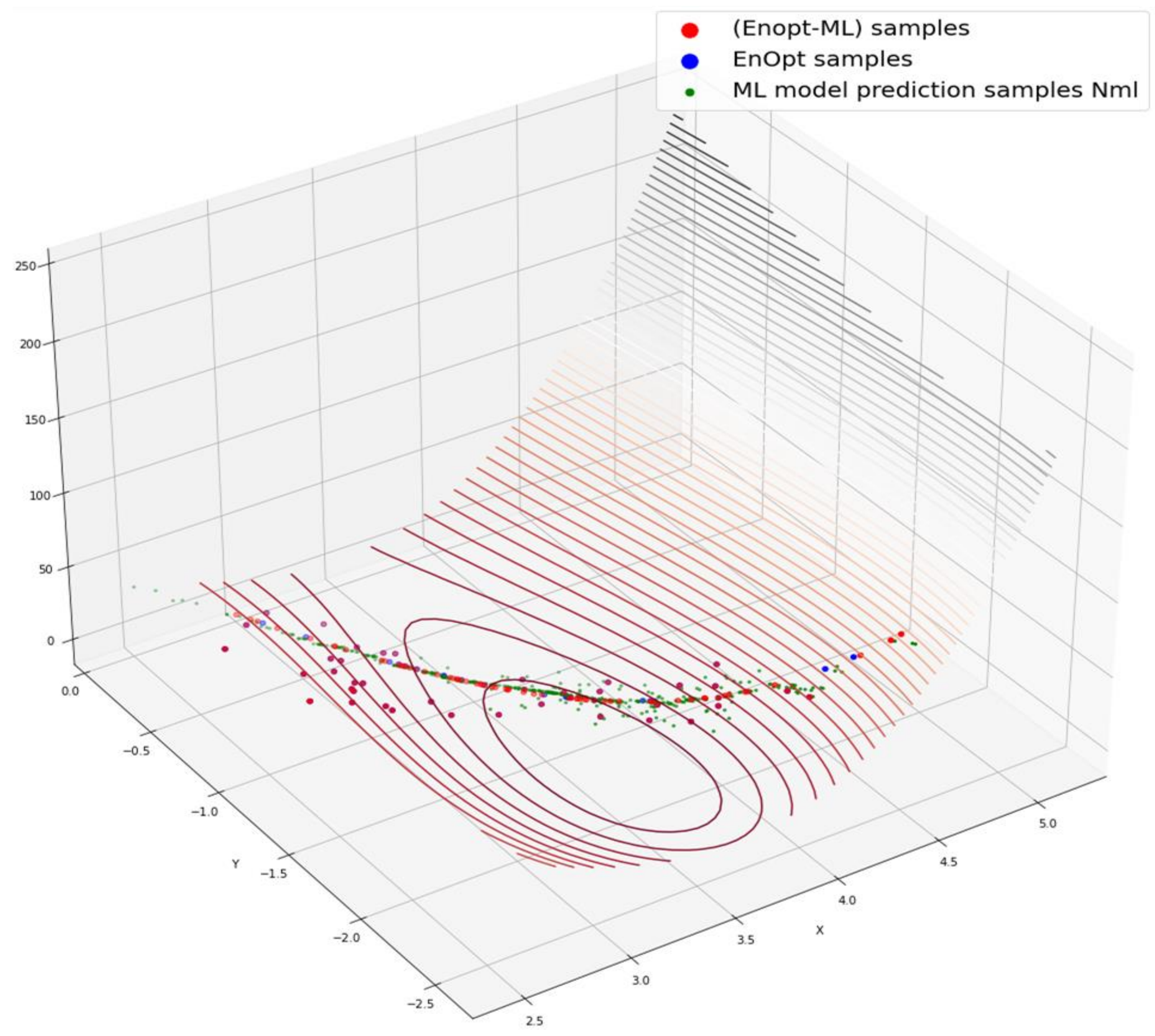


Figure 19 Starting point [3 -1]: EnOpt and EnOpt-ML samples and ML model predicted samples

The number of objective function evaluations is clearly lower in EnOpt-ML than in EnOpt by 80 times (table 5). This makes EnOpt-ML as the difference is very small in the optimization results between the two methods.

Table 5 Optimization results for Starting point [3 -1] (single simulation)

	updated (near-optimal or optimal) control vector at the last iteration.	Number of iterations	Number of objective function evaluations	Optimized objective function
EnOpt	[3.508528 -1.152152]	3	120	4.707815
EnOpt-ML	[3.641242 -1.290166]	5	200	3.808361

- <u>1000 simulations:</u>

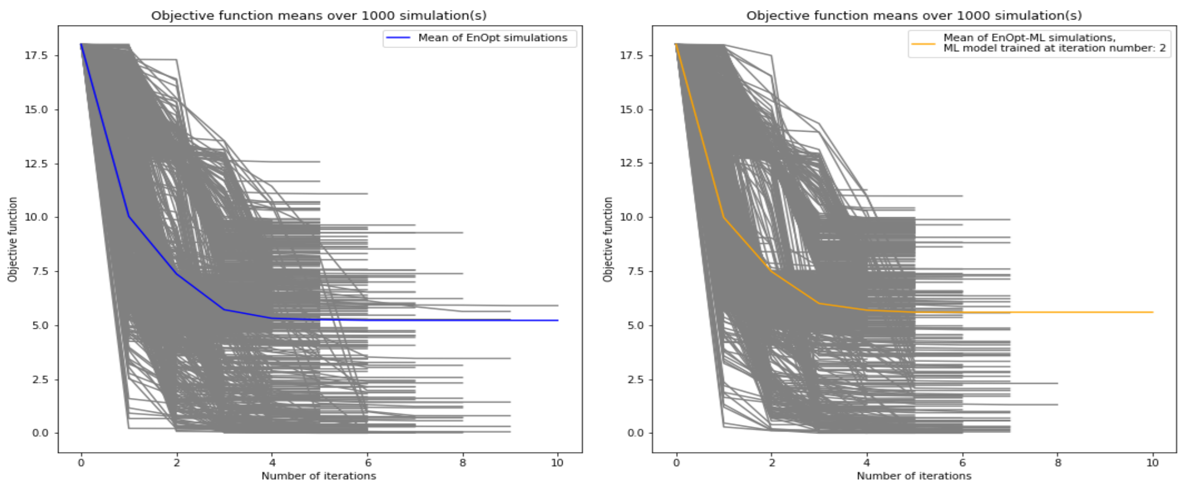


Figure 20 Case starting point [3 -1]: objective function optimization (1000 simulations)

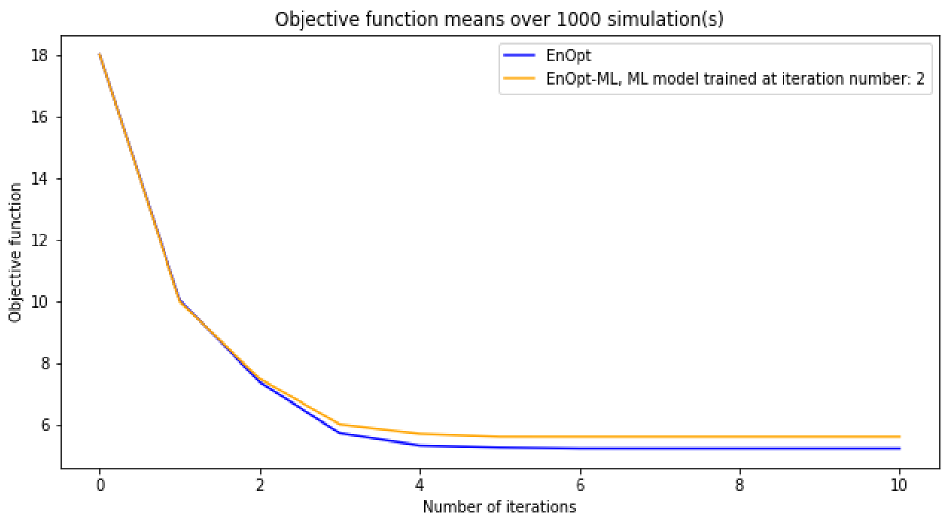


Figure 21 Case starting point [3 -1]: EnOpt and EnOpt-ML means of simulations

There are 10 times smaller standard deviations in the simulations with the starting point being in a flatter area compared to starting on a higher steep slope in the previous case (Enopt (2.55 in current case versus 33.5 in the previous one) EnOpt-ML (2.5 in current case versus 33 in previous case)). The mean over the simulation means (fig. 21) shows lower EnOpt values which is better than EnOpt-ML results, but this difference is very small (last optimal objective function means on EnOpt 5.21 in comparison EnOpt-ML resulted in 5.59 (table 6).

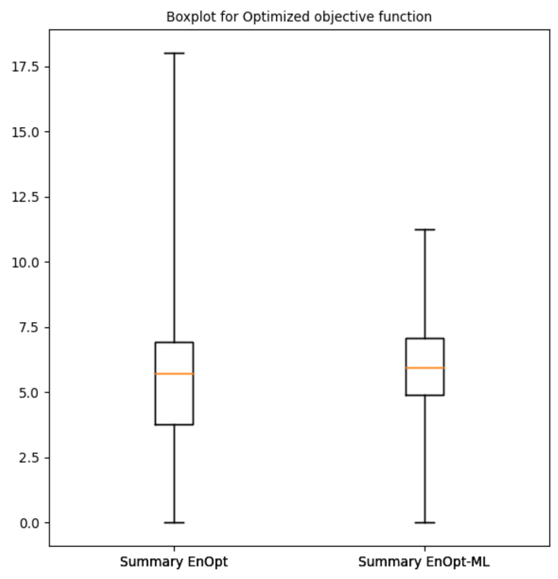


Figure 22 Case starting point [3 -1]: : Boxplot of optimized objective function

Table 6 Case starting point [3 -1]: summary output values (1000 simulations)

EnOpt	iterations	Number of objective function evaluations	Optimized objective function	EnOpt-ML	iterations	Number of objective function evaluations	Optimized objective function
mean	4.41941942	176.7767768	5.218195975	mean	4.407407407	176.2962963	5.593292677
std	1.29329366	51.73174623	2.75426896	std	1.038914757	41.55659029	2.503022661
min	1	40	0.002813	min	2	80	0.000846
25%	4	160	3.772761	25%	4	160	4.8850815
50%	4	160	5.741116	50%	4	160	5.954247
75%	5	200	6.939228	75%	5	200	7.0822095
max	10	400	17.999858	max	8	320	11.26578

Both optimizations run a same proportion of objective functions when starting in an already flat region, as in this case. The difference is that EnOpt-maximum ML's number of iterations was reduced by two, as seen in the table 6.

The quartiles on (fig. 22) show a higher maximum value of the objective function in EnOpt than in EnOpt-ML. (fig. 22) also shows that the EnOpt-ML has a higher 25% quartile value optimization, EnOpt-ML tends then to have higher objective function in these simulations than EnOpt but with a mode that is very close between the two methods (5.47 in EnOpt versus 5.95 in EnOpt-ML, table 6). The objective function evaluation mean over the simulations

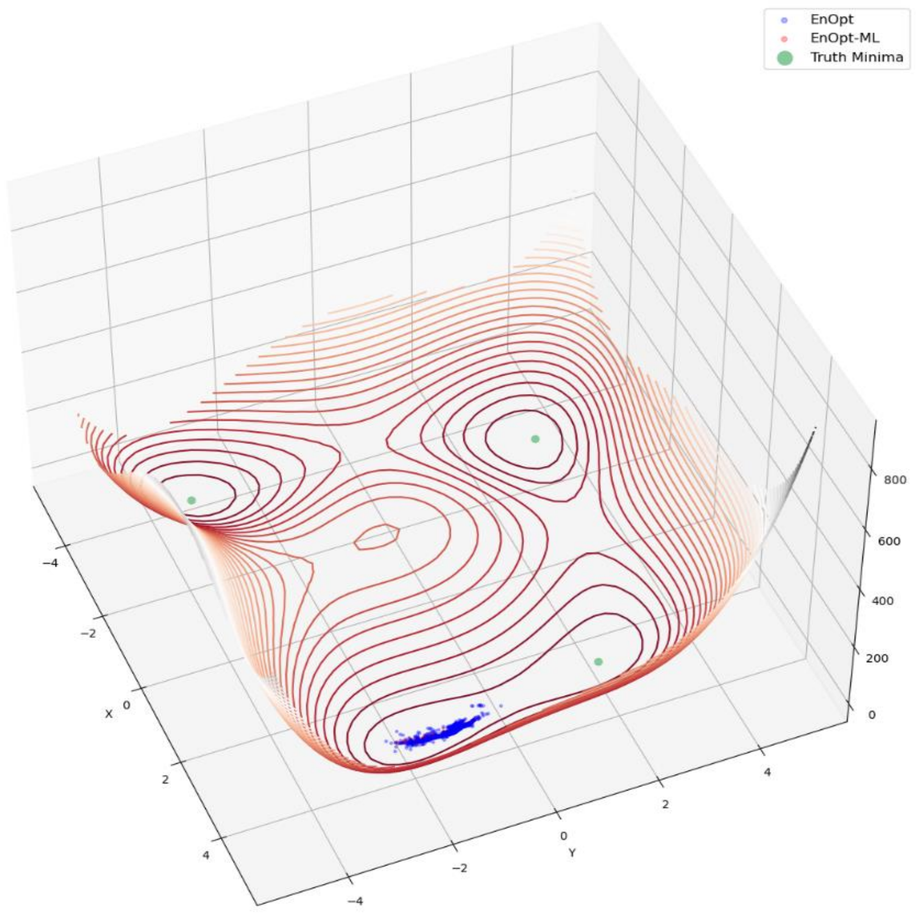


Figure 23 Case starting point [3 -1]: simulations' results of EnOpt and EnOpt-ML with respect to true minimum values

The simulations yielded optimization results (fig. 23) on the same area as the minima on the lower-left quadrant in all the 1000 simulations.

Changing initial covariance matrix

Initial covariance matrix: 0.2I

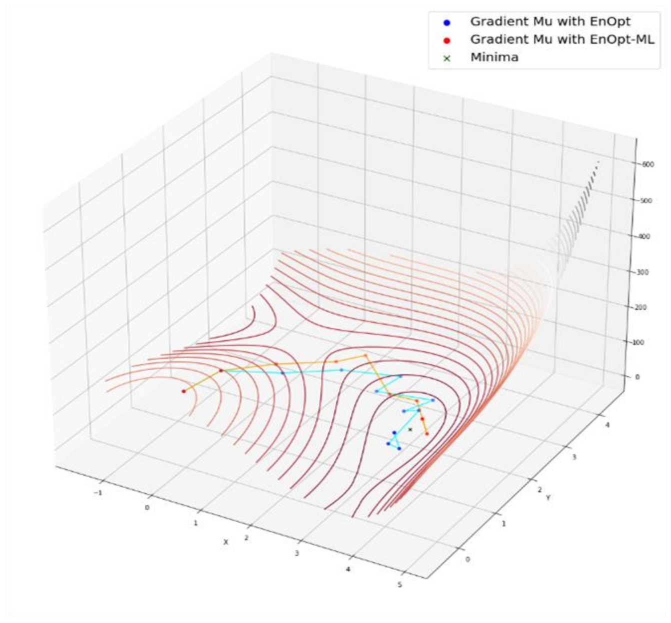


Figure 24 Case: covariance 0.2I : gradient (single simulation)

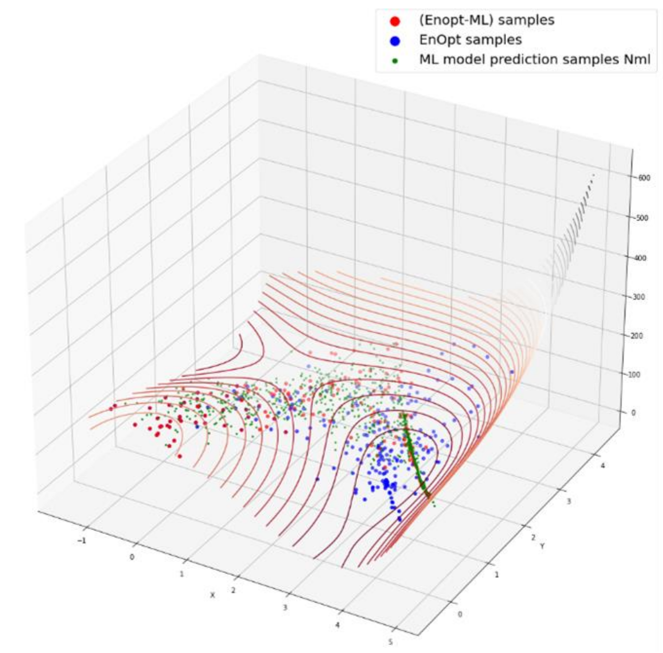


Figure 25 Case: covariance 0.2I : EnOpt and EnOpt-ML samples and ML model predicted samples

MSE: 63.65

Initial covariance matrix: 0.4I

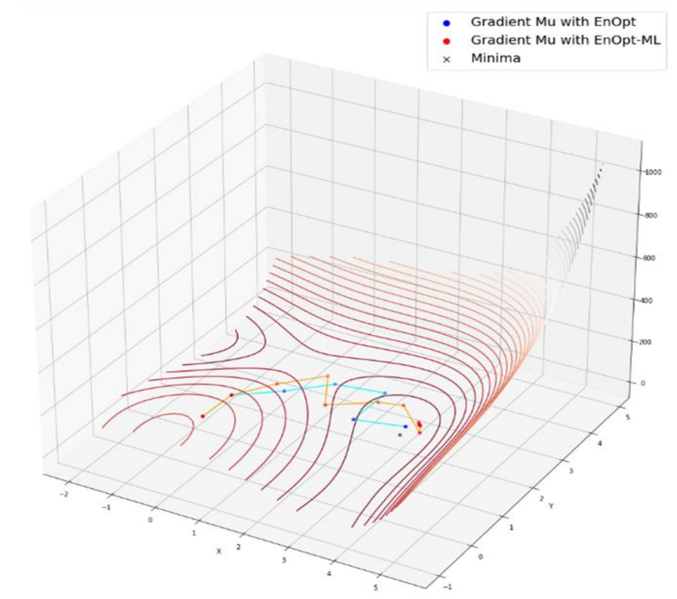


Figure 26 Case: covariance 0.4I : gradient (single simulation)

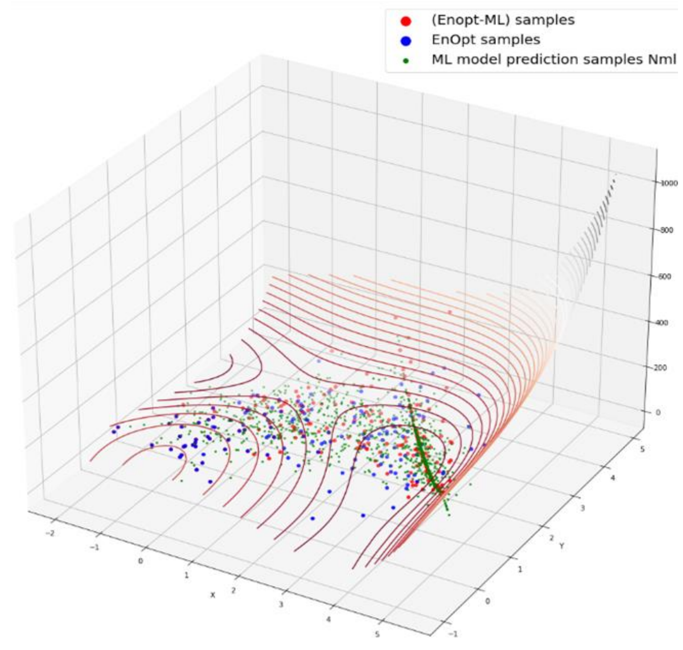


Figure 27 Case: covariance 0.4I : EnOpt and EnOpt-ML samples and ML model predicted samples

MSE: 178.77

The MSE is more than double the deviation for I×0.4 than I×0.2 covariance matrices and has led to an optimum that is 5.58 when I×0.2 EnOpt-ML lead to 2.06 which is slightly better in this case.

We also observe that, in this configuration, the EnOpt-ML for I×0.4 performed in more iterations than EnOpt (13 versus 7) while the EnOpt-ML for I×0.2 performed in fewer iterations than EnOpt (6 versus 12). The base configuration case had 11 iterations in EnOpt and 10 for EnOpt-ML, which are only one iteration difference (tables 7 and 8).

Figures 24 and 26 show the difference in spread of function sampling, we can observe that in the case where the covariance is higher, the EnOpt-ML and EnOpt samples are taken in the later iterations on the same narrow region which is not what we have seen with the base case configuration or in the case where the covariance is 0.2I where the show parallel narrow scattering.

Table 7 Optimization results for case: covariance 0.2I (single simulation)

	updated (near-optimal or optimal) control vector at the last iteration.	Number of iterations	Number of objective function evaluations	Optimized objective function
EnOpt	[2.840681 1.796319]	12	480	2.156101
EnOpt-ML	[3.028928 2.307775]	9	360	2.067858

Table 8 Optimization results for case: covariance 0.4I (single simulation)

	updated (near-optimal or optimal) control vector at the last iteration.	Number of iterations	Number of objective function evaluations	Optimized objective function
EnOpt	[2.942099 2.26195]	7	280	1.127203
EnOpt-ML	[3.159395 2.399548]	13	260	5.583819

The number of objective function evaluation increased in EnOpt from the configuration with a Covariance matrix of 0.1 I (440, table 1) to a covariance of 0.2I (480, table 7), but then it decreased in the case of a covariance matrix of 0.4I (280, table 8). On the other hand, for EnOpt-ML, the number of objective function evaluation only kept decreasing with higher covariances (400 to 360 to 260 in covariances 0.1I, 0.2I and 0.4I respectively, tables 1, 7 and 8). The optimal values of the objective function on all covariances are not very different between the base case configuration and the covariance of 0.2I (approximately 3 with covariance 0.1I, and approximately 2 with covariance 0.4I). for the case where the covariance is 0.4I, the value is four times higher in the result of EnOpt-ML in comparison with EnOpt (1.12 in EnOpt versus 5.58 in EnOpt-ML, tables 1, 7 and 8)

- <u>1000 simulations:</u>

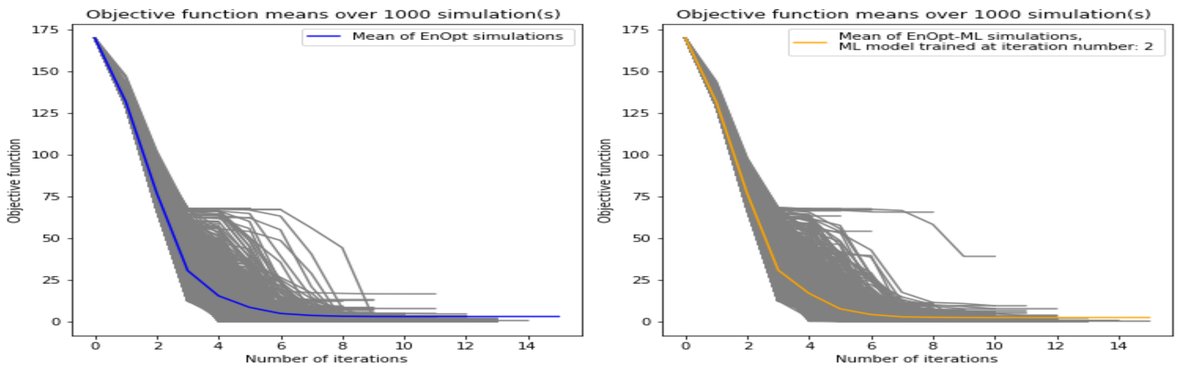


Figure 28 Case: covariance 0.2I : objective function optimization results (1000 simulations)

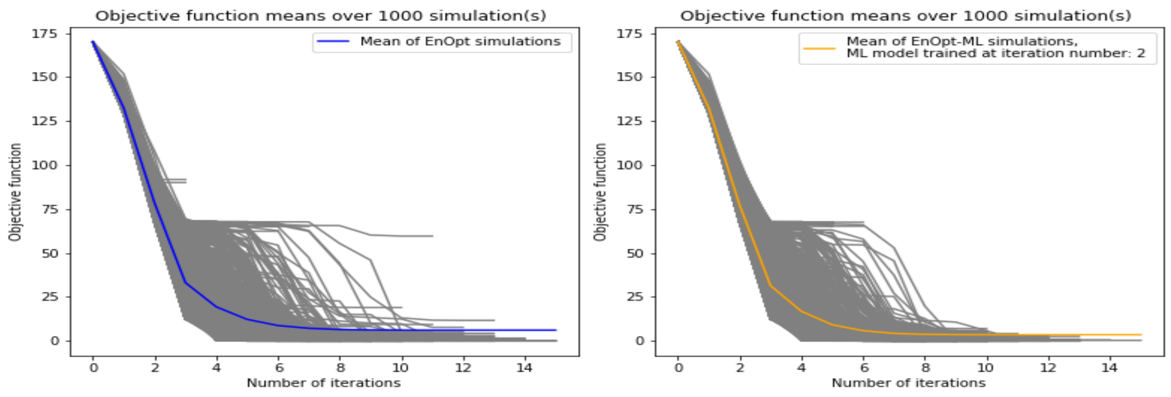


Figure 29 Case: covariance 0.4I : objective function optimization results (1000 simulations)

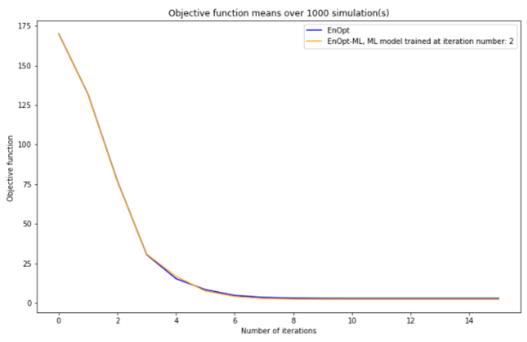


Figure 30 Case: covariance 0.2I : EnOpt and EnOpt-ML means of simulations

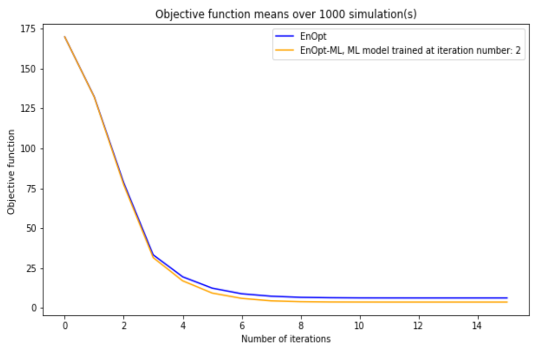


Figure 31 Case: covariance 0.4I : EnOpt and EnOpt-ML means of simulations

The means gathered from the 1000 simulations show very close optimization values throughout the iterations with respect to the methods (EnOpt or EnOpt-ML) and with respect to the covariances (0.1I and 0.4I),(fig. 30 and 31). The iterations mean is around 8 for all cases which indicates that there is not much difference in the number of times the objective function has been evaluated.

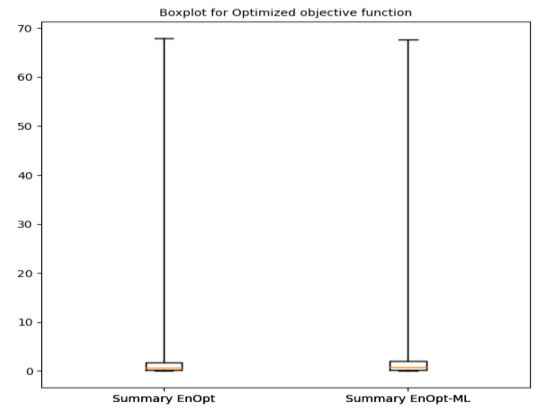


Figure 32 Case: covariance 0.2I : Boxplot of optimized objective function

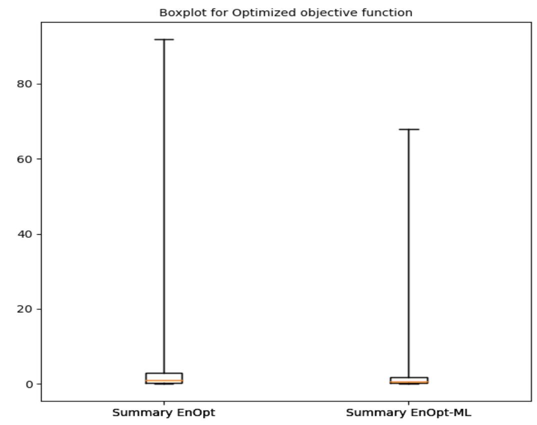


Figure 33 Case: covariance 0.4I : Boxplot of optimized objective function

In figures 32 and 33, the quartiles look very close to each other for the two present configuration cases as well as EnOpt and EnOpt-ML with the only visible overall difference of EnOpt-ML computed with a covariance matrix equal to 0.4I, where the maximum value obtained through all simulations is lower.

When comparing the EnOpt to EnOpt-ML of I×0.4 in those simulations, we get about one third less on the value of EnOpt-ML results with slightly fewer evaluations of the objective function from the EnOpt evaluations (tables 9 and 10). For I×0.2, same observation, but with smaller difference on the optimized objective values and the number of objective function evaluation. The standard deviation of the different simulations is higher on the 0.4I covariance for both optimal values and objective function number of evaluation (tables 9 and 10).

Comparing to the base case configuration, the EnOpt-ML overperforms EnOpt when the initial covariance is greater. However, for EnOpt-ML, I×0.2 did best in terms of objective function optimization both on the means and the standard deviation of the simulations in comparison with both covariances I×0.4 and I×0.1. (tables 2, 9 and 10).

In summary, the values obtained on the means for both present cases show that EnOpt-ML performed better with larger covariance matrix (see also fig. 32 and 33). And on the contrary, the objective function had more evaluations in EnOpt-ML than in EnOpt where the covariance is 0.4I in comparison with the 0.2 covariance matrix case (tables 9 and 10).

Table 9 Case: covariance 0.2I : summary output values (1000 simulations)

EnOpt	iterations	Number of objective function evaluations	Optimized objective function	EnOpt-ML	iterations	Number of objective function evaluations	Optimized objective function
mean	8.03503504	321.4014014	3.03806546	**mean**	8.062062062	322.4824825	2.420361895
std	1.86513572	74.60542861	9.945434183	**std**	1.604774987	64.19099946	7.627771851
min	4	160	0.000022	**min**	4	160	0.00006
25%	7	280	0.1506085	**25%**	7	280	0.181601
50%	8	320	0.555473	**50%**	8	320	0.736363
75%	9	360	1.686584	**75%**	9	360	1.987503
max	14	560	67.884046	**max**	15	600	67.681479

Table 10 Case: covariance 0.4I : summary output values (1000 simulations)

EnOpt	iterations	Number of objective function evaluations	Optimized objective function	EnOpt-ML	iterations	Number of objective function evaluations	Optimized objective function
mean	7.58258258	151.6516517	6.154339948	**mean**	7.986986987	159.7397397	3.558531807
std	2.30598614	46.11972281	15.15484148	**std**	1.922035189	38.44070379	11.83664817
min	3	60	0.000034	**min**	4	80	0.000018
25%	6	120	0.2426775	**25%**	6	120	0.16454
50%	7	140	0.95156	**50%**	8	160	0.655106
75%	9	180	2.9547915	**75%**	9	180	1.861418
max	15	300	91.763569	**max**	15	300	67.798513

Figures 34 and 35, show that with a higher covariance matrix, there are more chances that the EnOpt/ EnOpt-ML can fall into other minima. These two examples also have a concentration of simulation results on the lower-right minimum of the figure, few on the top – right minimum, one outlier for covariance equal to 0.4I and none for the rest of the minima. Of course, the wider the covariance matrix the more possibilities it is to sample from areas that may lead to different directions.

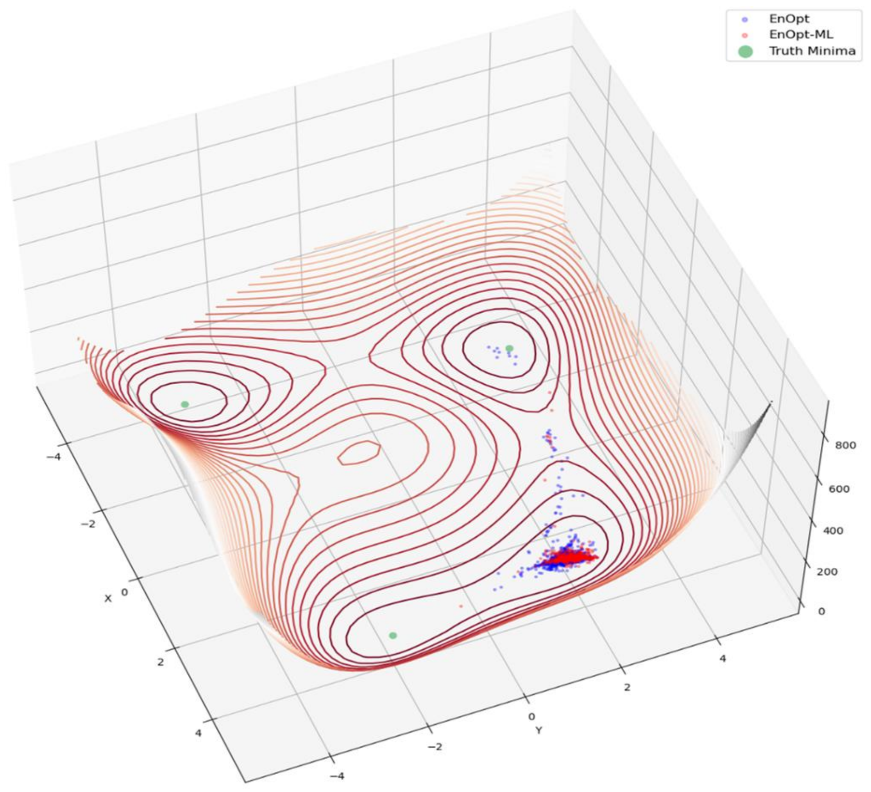


Figure 34 Case: covariance 0.2I : simulations' results of EnOpt and EnOpt-ML with respect to true minimum values

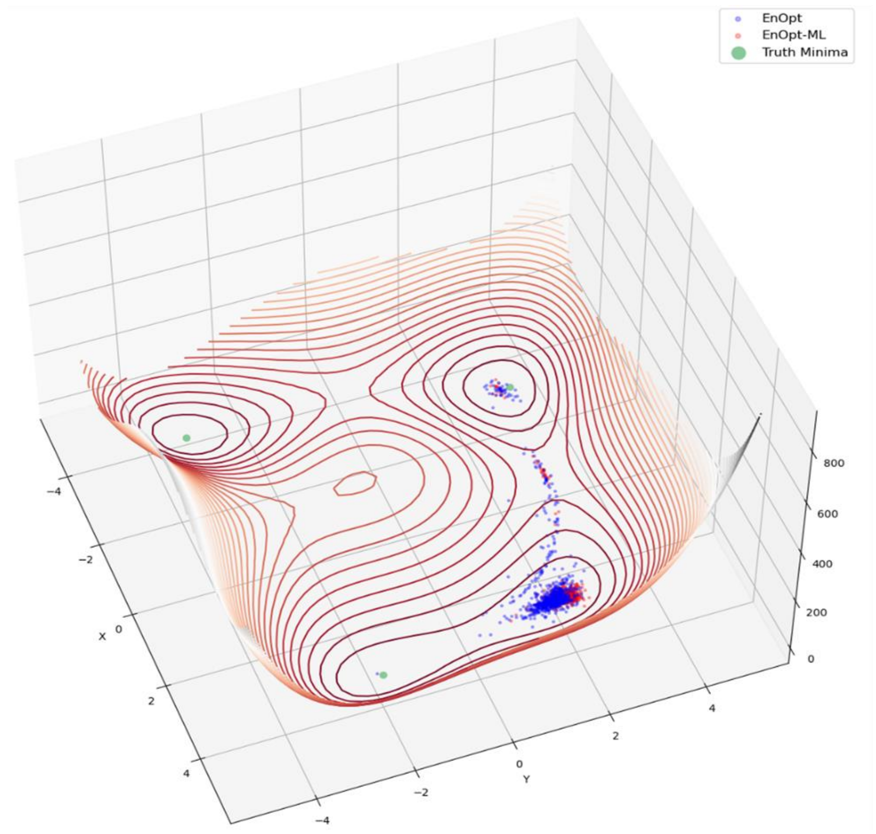


Figure 35 Case: covariance 0.4I : simulations' results of EnOpt and EnOpt-ML with respect to true minimum values

Changing the sample size N

In EnOpt, The number of samples N is very important to the performance of the optimizations, but also, it is a factor in the objective function number of evaluations, meaning, it is best to generate high number of samples for finding better near optimal values, but it also means that the computational time is going to increase. In this subchapter, EnOpt-ML is going to be performed to explore its impact

- Single simulation

N = 5

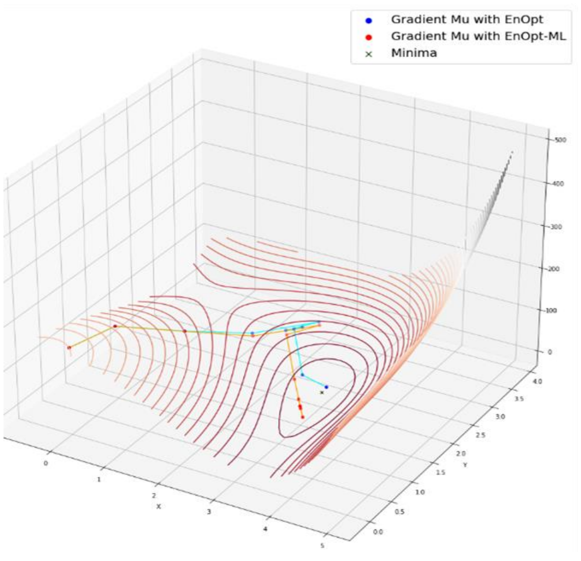


Figure 36 Case N=5: gradient (single simulation)

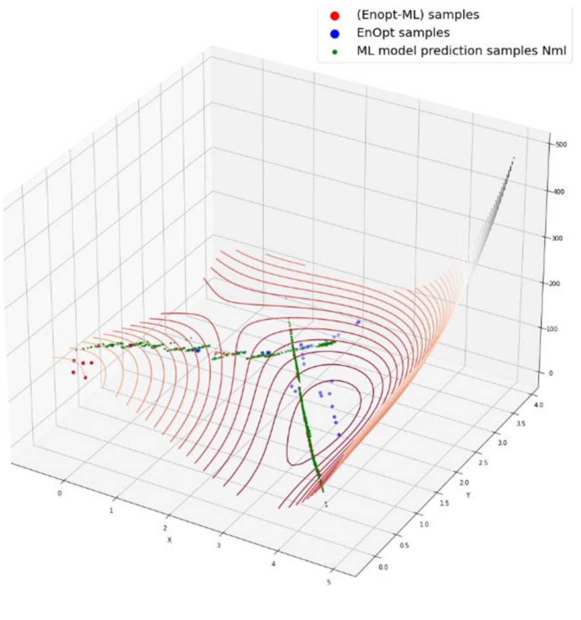


Figure 37 Case N=5: EnOpt and EnOpt-ML samples and ML model predicted samples

MSE: 40.86

N = 100

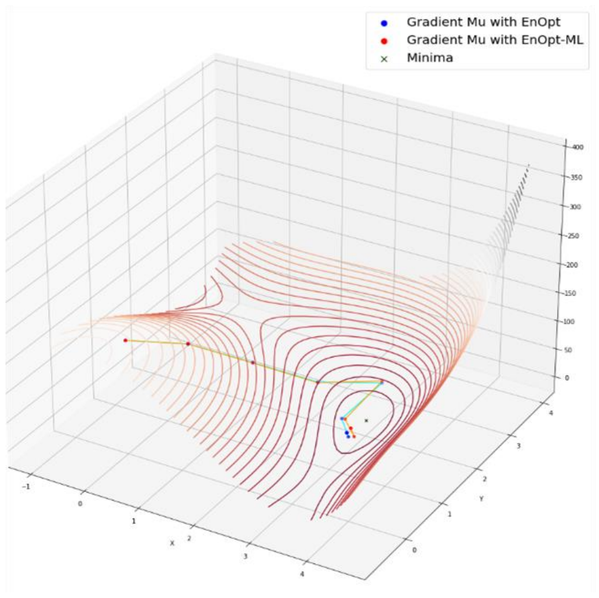


Figure 38 Case N=100: gradient (single simulation)

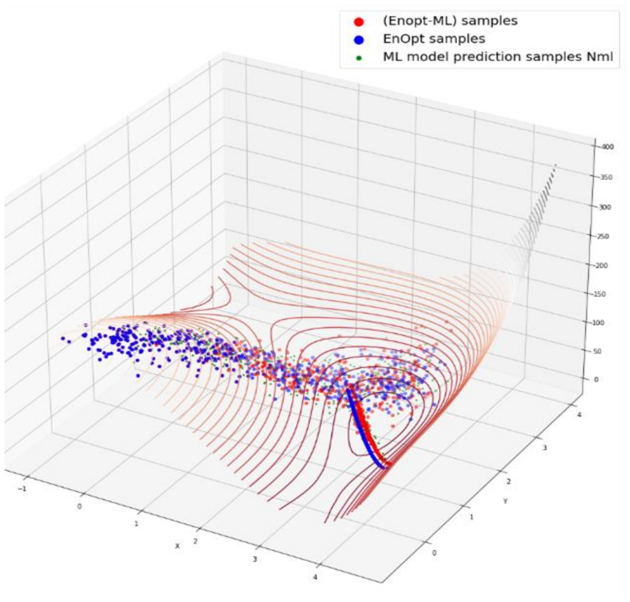


Figure 39 Case N=100: EnOpt and EnOpt-ML samples and ML model predicted samples

MSE: 19.42

The base configuration with a lower number of samples N = 5, compared to a higher number of samples N = 100, has more wiggly gradients both with EnOpt and EnOpt-ML(fig. 36 and 38). The increase of N number of samples also improves the MSE as the machine learning algorithm also benefits from the data it is training on.The samples on figures 37 and 39 also show that a smaller number of samples would fail to collect features in the function if it had some data that would have redirected the gradient to a different slope.

Table 11 Optimization results for case N=5: (single simulation)

	updated (near-optimal or optimal) control vector at the last iteration.	Number of iterations	Number of objective function evaluations	Optimized objective function
EnOpt	[2.985587 2.136792]	10	100	0.306666
EnOpt-ML	[2.947249 1.560185]	11	110	3.187601

Table 12 Optimization results for case N=100 single simulation):

	updated (near-optimal or optimal) control vector at the last iteration.	Number of iterations	Number of objective function evaluations	Optimized objective function
EnOpt	[2.958271 1.539858]	9	1800	3.293155
EnOpt-ML	[2.928709 1.691949]	8	1600	1.994666

The objective function is clearly evaluating more times in N=100 than N= 5, but curiously in this simulation (and contrary of most simulations See table 11), in the case where the sample size is 5, EnOpt performed better than in the case where the sample size was 100 (0.3 for N equal to 5 versus 3.29 for N equal to 100, tables 11 and 12). EnOpt-ML evaluated more times the objective function than EnOpt, but it computed 200 fewer times than EnOpt when N=100 . when N= 100, EnOpt-ML yielded a value a slightly smaller than EnOpt on the same configuration.

Compared to the base case with a single simulation, N=20, EnOpt yielded very close values to the case with N=100 while we obtained slightly better value with EnOpt-ML where the sample size are higher (3.18 with N=5, 2.99 with N=20 and 1.99 with N=100). But since the difference is small between the results with N=20 and N=100 considering we need a lot more objective function evaluations, we can say that N=20 did best on EnOpt-ML front.

- <u>1000 simulations:</u>

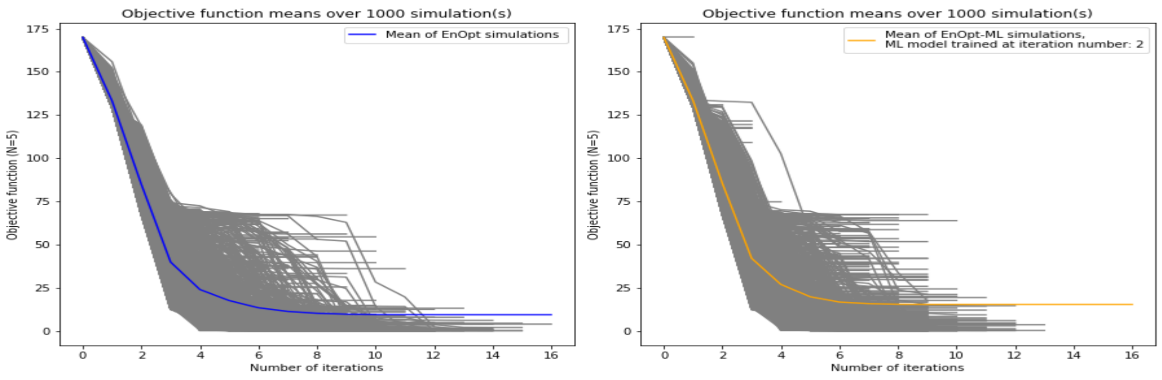


Figure 40 Case N=5: objective function optimization results (1000 simulations)

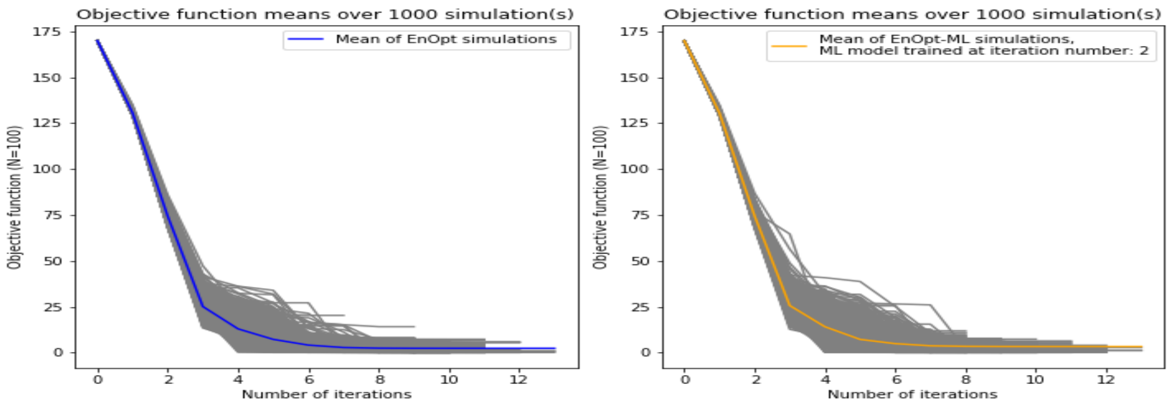


Figure 41 Case N=100: objective function optimization results (1000 simulations)

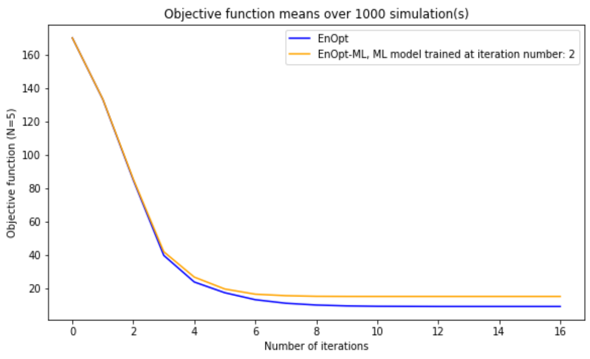


Figure 42 Case N=5: EnOpt and EnOpt-ML means of simulations

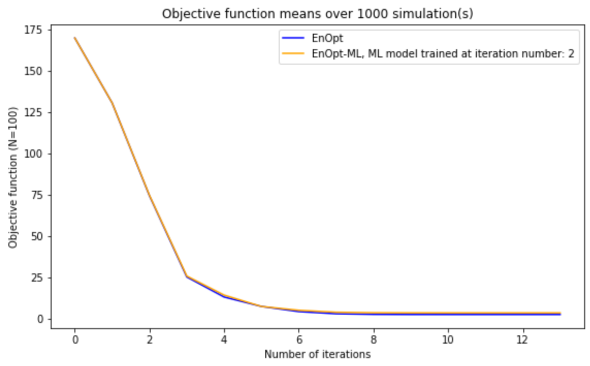


Figure 43 Case N=100: EnOpt and EnOpt-ML means of simulations

For both cases discussed in this subchapter, EnOpt-ML has, in average, fewer objective function evaluations than EnOpt with optimization results that are fairly close to each other although closer for the case where N = 100.

The main objective is to observe the part that machine learning algorithm model plays when used on the gradient. EnOpt-ML grants around 222 fewer evaluations of the objective function (tables 13 and 14) when using higher number of samples (N = 100), while being still pretty close to the value obtained with EnOpt. In addition, it appears that the bigger the sample size the closest are the results of EnOpt and EnOpt-ML.

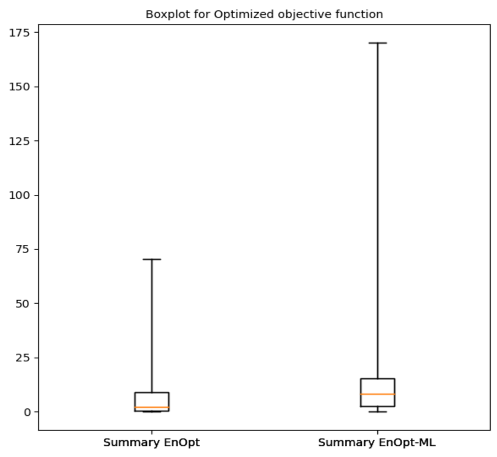


Figure 44 Case N=5: Boxplot of optimized objective function

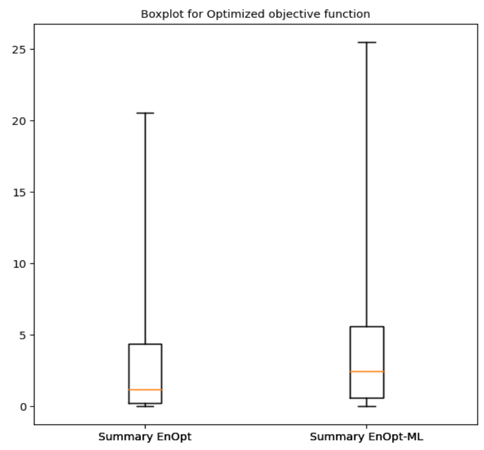


Figure 45 Case N=100: Boxplot of optimized objective function

The quartiles (figures 44 and 45) show how the case of higher number of samples (N=100), the mean of the optimized objective functions of the simulations is way better than where the sample size is very small. The highest optimized objective function obtained was EnOpt-ML where N=5 (15.35 which is about 5 times higher than where N=100, tables 13 and 14)

Table 13 Case N=5: summary output values (1000 simulations)

EnOpt	iterations	Number of objective function evaluations	Optimized objective function	EnOpt-ML	iterations	Number of objective function evaluations	Optimized objective function
mean	7.88588589	78.85885886	9.337499209	mean	6.956956957	69.56956956	15.35452064
std	2.21467844	22.1467844	16.45689647	std	1.718445594	17.18445594	20.89051922
min	4	40	0.000061	min	1	10	0.00265
25%	6	60	0.427817	25%	6	60	2.699147
50%	8	80	2.241446	50%	7	70	8.343788
75%	9	90	9.112715	75%	8	80	15.2397295
max	16	160	70.426997	max	13	130	170

Table 14 Case N=100: summary output values (1000 simulations)

EnOpt	iterations	Number of objective function evaluations	Optimized objective function	EnOpt-ML	iterations	Number of objective function evaluations	Optimized objective function
mean	8.86086086	1772.172172	2.337241181	mean	7.743743744	1548.748749	3.308826613
std	1.37748124	275.4962482	2.487727094	std	1.597969152	319.5938304	2.994426332
min	5	1000	0.000061	min	5	1000	0.000279
25%	8	1600	0.2047365	25%	6	1200	0.597889
50%	9	1800	1.165795	50%	8	1600	2.455063
75%	10	2000	4.374845	75%	9	1800	5.588925
max	13	2600	20.537546	max	13	2600	25.480254

We can deduce the same thing for EnOpt-ML comparison on all sample size cases as we can for single simulations: because the difference between the results with N=20 and N=100, where we need a much higher number of objective function evaluations (1548.75, table 14), we can say that the N=20 case with EnOpt-ML method is good enough.

The same can be said about EnOpt as, in fact, it was better with more samples but the counterpart of computational intensity, and since the difference is small in optimized value. we can say that the sample size of 20 is good enough and best of the three cases (9.33 for N=5, 3.64 for N=20 and 2.33 for N=100, tables 2, 13 and 14) for the purpose of balancing optimization results with reduction of the objective function evaluation.

In Figure 46, the simulations performed with N=5 landed mostly on the true minimum on the lower-right quadrant but also on the top-right and the lower-left quadrants' true minima (and some in between those three) with more EnOpt-ML optima on the lower-left true minimum quadrant than EnOpt and more EnOpt optima on the top-right true minimum than EnOpt-ML optima. none of the optimizations' simulations landed on the top-left true minimum.

In figure 47, the simulations performed with N=100 landed all on the same true minimum on the lower right quadrant.

We notice that, when compared to the base case (fig.11), the larger the sample size, the more simulations end up on the same minimum.

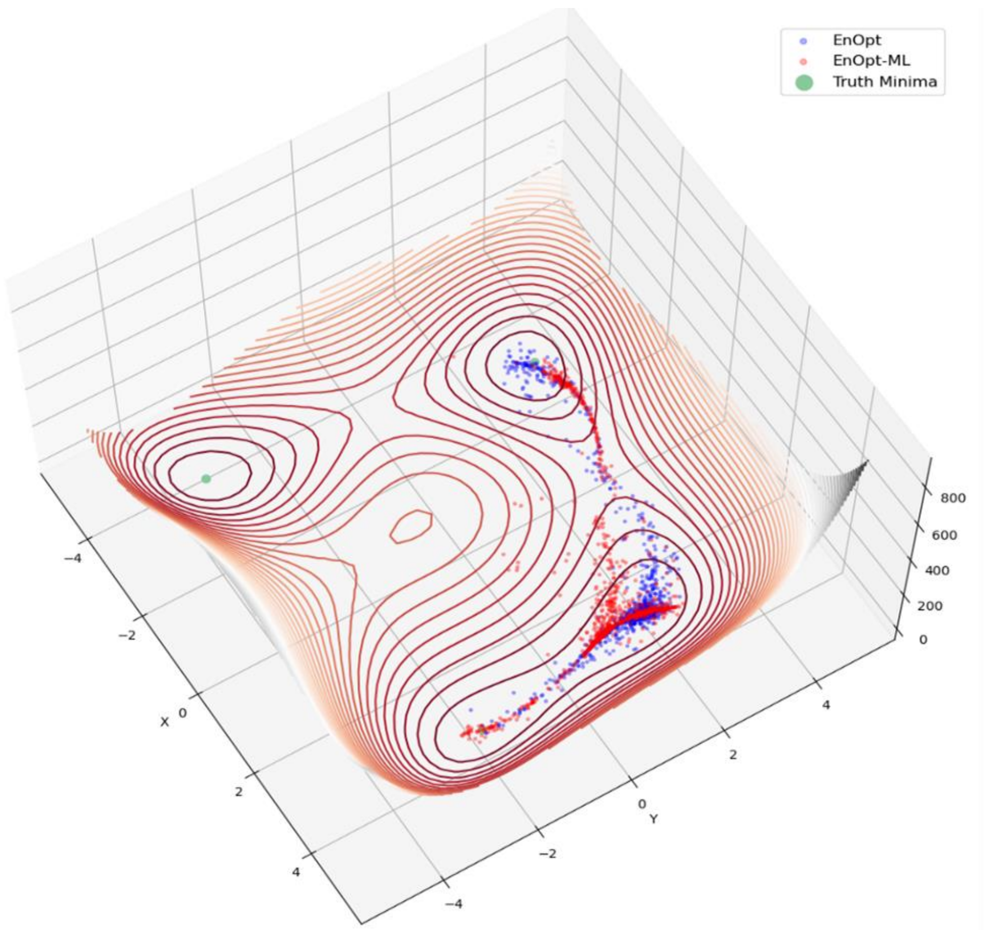


Figure 46 Case N=5: simulations' results of EnOpt and EnOpt-ML with respect to true minimum

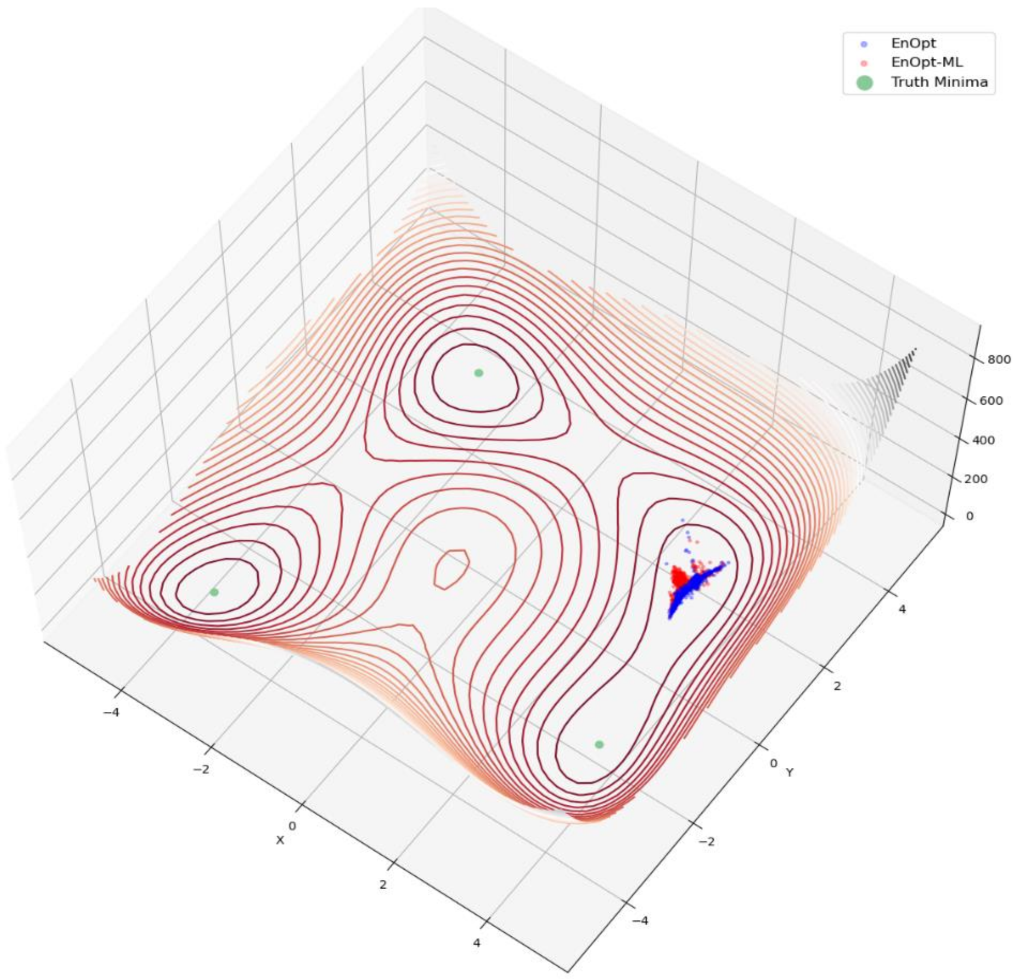


Figure 47 Case N=100: simulations' results of EnOpt and EnOpt-ML with respect to true

Changing the Number of the predicted samples by the machine learning model

The predicted samples from the ML model can also affect the optimization of EnOpt-ML due to the errors they can present. In this subchapter, the number of those predicted samples is changed from Nml=100 (base configuration case) to Nml = 20 and Nml=1000 to explore the effect it can have on the EnOpt-ML method.

- Single simulation

Nml = 2

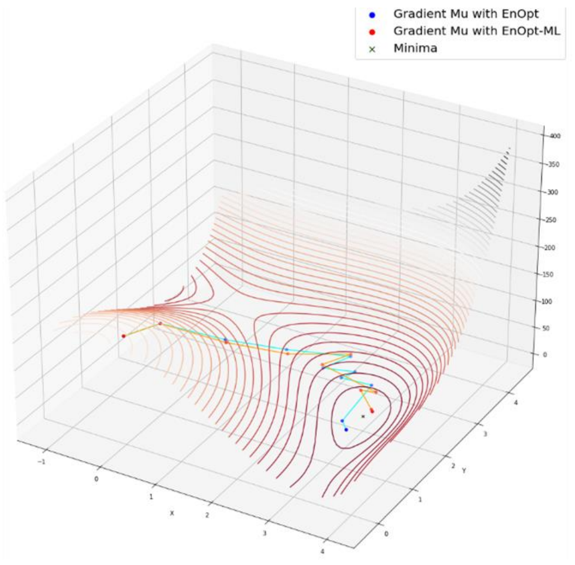


Figure 48 Case Nml = 20: gradient (single simulation)

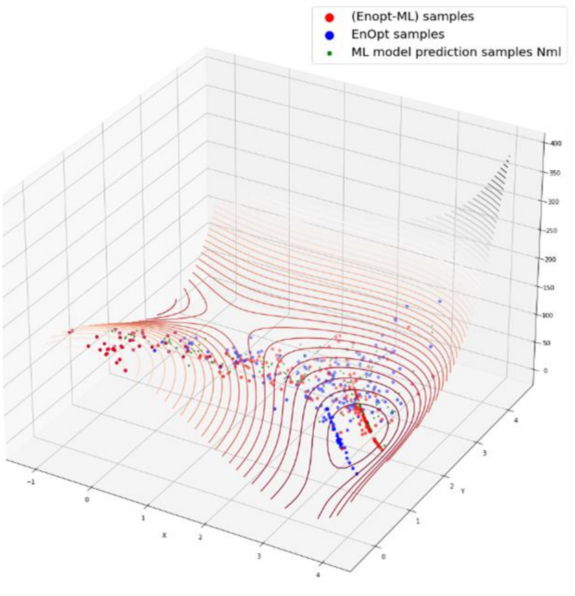


Figure 49 Case Nml = 20: EnOpt and EnOpt-ML samples and ML model predicted samples

MSE : 95.05

Nml = 1000

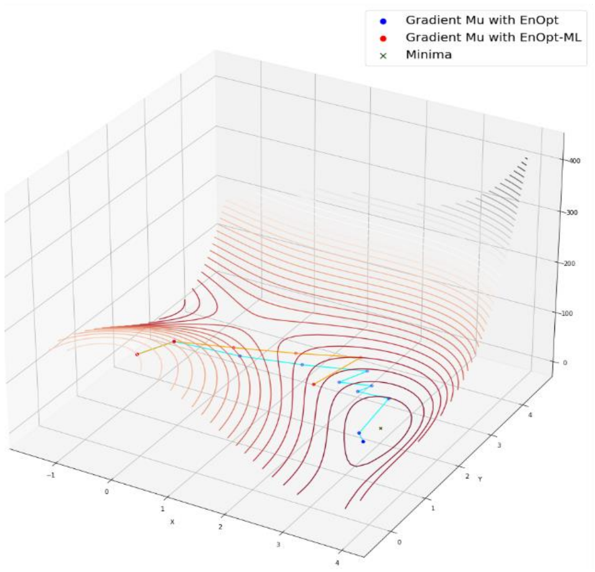


Figure 50 Case Nml = 1000: gradient (single simulation)

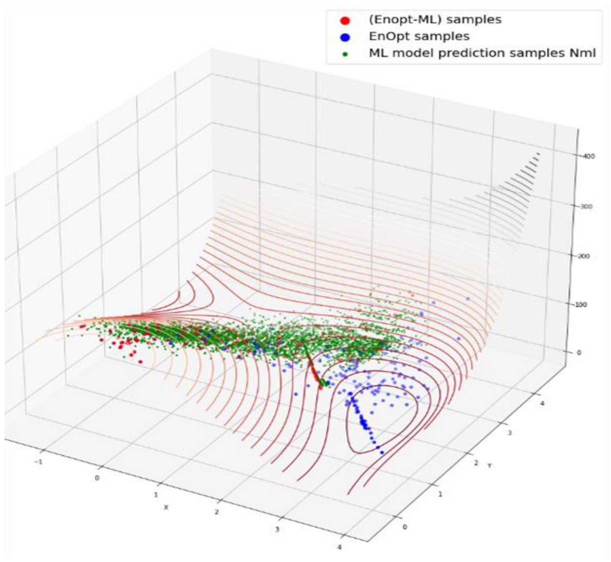


Figure 51 Case Nml = 1000 EnOpt and EnOpt-ML samples and ML model predicted samples

MSE : 85.99

When changing the number of samples predicted by the machine learning model,   we can see that with smaller Nml , we get better optimization value (for this single simulation) compared to the base case configuration. In fact , with even higher number of predicted samples used we ended up with a declined performance (34.98 for Nml= 1000 versus 0.9 for Nml=20, tables 15 and 16) . This is due to that the Nml samples do not necessarily fall in the Himmelblau surface. The MSE (Mean Square Error)   is higher on Nml=20 case than the Nml=1000 case which may be misleading, but this is the effect of the very different number of samples used to calculate the MSE.

Table 15 Optimization results for case Nml = 20 (single simulation)

	updated (near-optimal or optimal) control vector at the last iteration.	Number of iterations	Number of objective function evaluations	Optimized objective function
EnOpt	[2.995507 1.511997]	10	440	3.21792
EnOpt-ML	[3.072356 2.159561]	11	400	0.900504

Table 16 Optimization results for case Nml = 1000 (single simulation)

	updated (near-optimal or optimal) control vector at the last iteration.	Number of iterations	Number of objective function evaluations	Optimized objective function
EnOpt	[2.995507 1.511997]	10	440	3.21792
EnOpt-ML	[1.70808 2.188727]	6	240	34.98751

In EnOpt-ML, the difference in the number of times the objective function was evaluated has dropped with the use of Nml=1000 by 160 times compared to Nml=20 case (tables 15 and 16), while both cases of Nml= 20 and the base configuration case of Nml=100 evaluated the objective function an equal number of times.

- <u>1000 simulations</u>

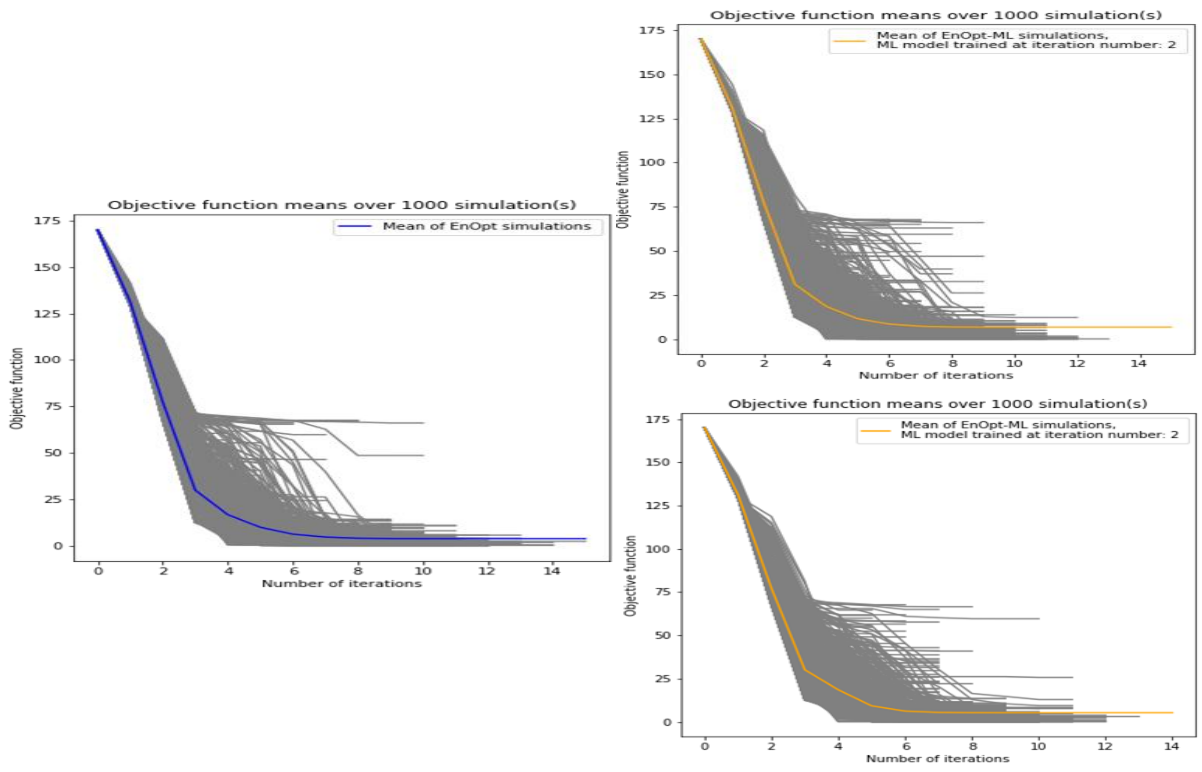


Figure 52 Case Nml = 20: (top-right), case Nml = 1000(bottom right), EnOpt (left): objective function optimization results (1000 simulations)

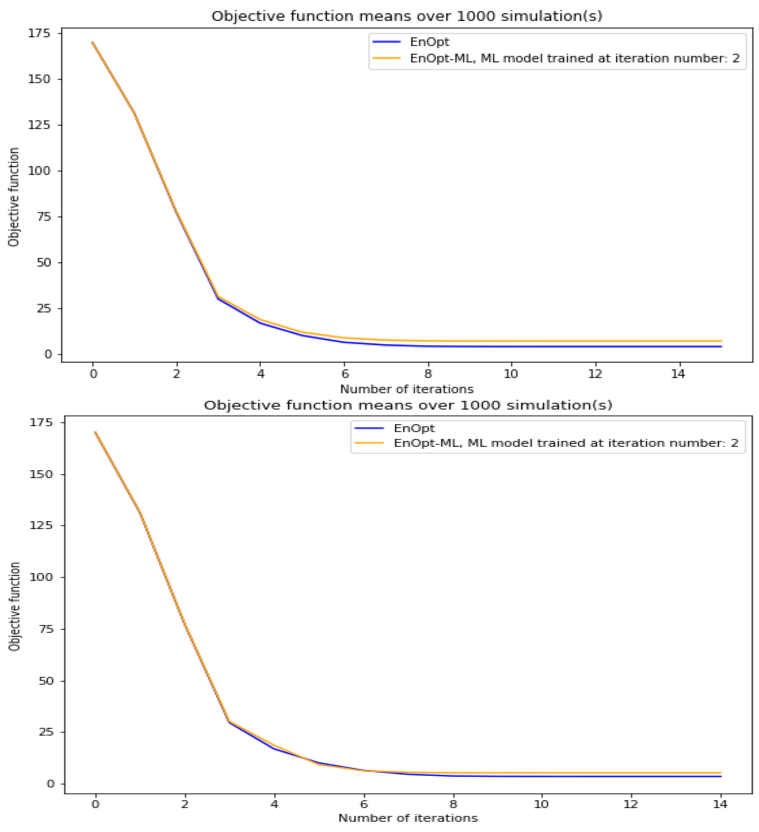


Figure 53 Case Nml = 20: (top-right), case Nml = 1000(bottom right): EnOpt and EnOpt-ML means of simulations

The deviations between the different cases' simulations are almost the same. The EnOpt result does not change since its algorithm does not include the use of Nml, so the tables included are only for EnOpt-ML. for the different Nml.

The mean of the simulations means lead to very close values with slightly lower value for Nml=1000. This is contrary of the single simulation where the difference was pronounced with Nml=1000's result being higher (tables 16 and 18).

The EnOpt in both cases performed only very slightly better than EnOpt-ML (3.64 versus 6.82 and 5.31). The results between Nml= 20 and Nml=1000 led to very similar results with just a little higher 75% quartile value on Nml=20 case configuration (fig. 54 and 55, table).

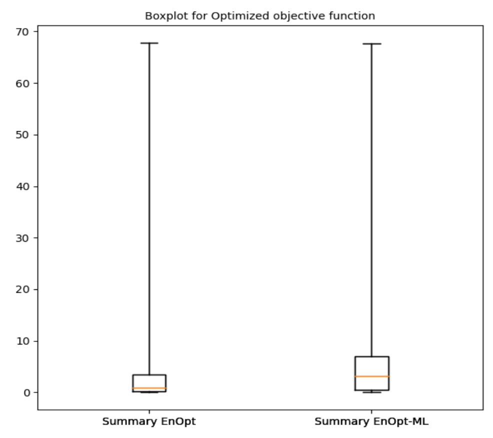


Figure 54 Case Nml = 20: Boxplot of optimized objective function

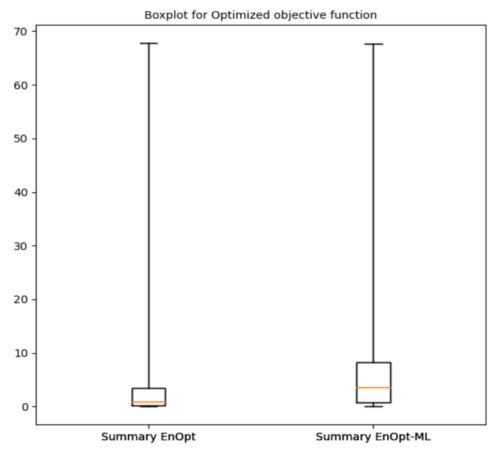


Figure 55 Case Nml = 1000: Boxplot of optimized objective function

The average number of objective function evaluation has doubled with an Nml=1000 compared with the case of Nml=20 while the optimization results' mean over the simulations are not so different (tables 17 and 18).

Table 17 Case Nml = 20: summary output values (1000 simulations)

EnOpt-ML	iterations	Number of objective function evaluations	Optimized objective function
mean	7.497497497	299.8998998	6.823029092
std	1.623889707	64.95558826	10.93917618
min	4	160	0.000714
25%	6	240	0.782544
50%	7	280	3.539252
75%	9	360	8.24247
max	13	520	67.717864

Table 18 Case Nml = 1000: summary output values (1000 simulations)

EnOpt-ML	iterations	Number of objective function evaluations	Optimized objective function
mean	7.616616617	599.7997996	5.308727566
std	1.480049072	129.9111765	8.994107474
min	5	320	0.00197
25%	6	480	0.442765
50%	8	560	3.104132
75%	9	720	6.915578
max	13	1040	67.701877

In comparison with the base configuration case, Nml=100, the average of simulations of optimization results are very close to the ones with Nml= 20 (tables 2 and 17)

The distribution of EnOpt and EnOpt-ML simulation (fig. 56 and 57) plot of results shows that EnOpt and EnOpt-ML can end up on distinct optimal values. The majority of them for both plots (fig. 56 and 57) are in the lower-right quadrant of the figures 56 and 57. Few simulations produced values that were closer to the top-right quadrant's true minimum, and even fewer were close to the lower-left quadrant's true minimum (in Nml= 20 ,it was barely approached). No optimization approached the true minima in the top-left corner of the figures (fig. 56 and 57).

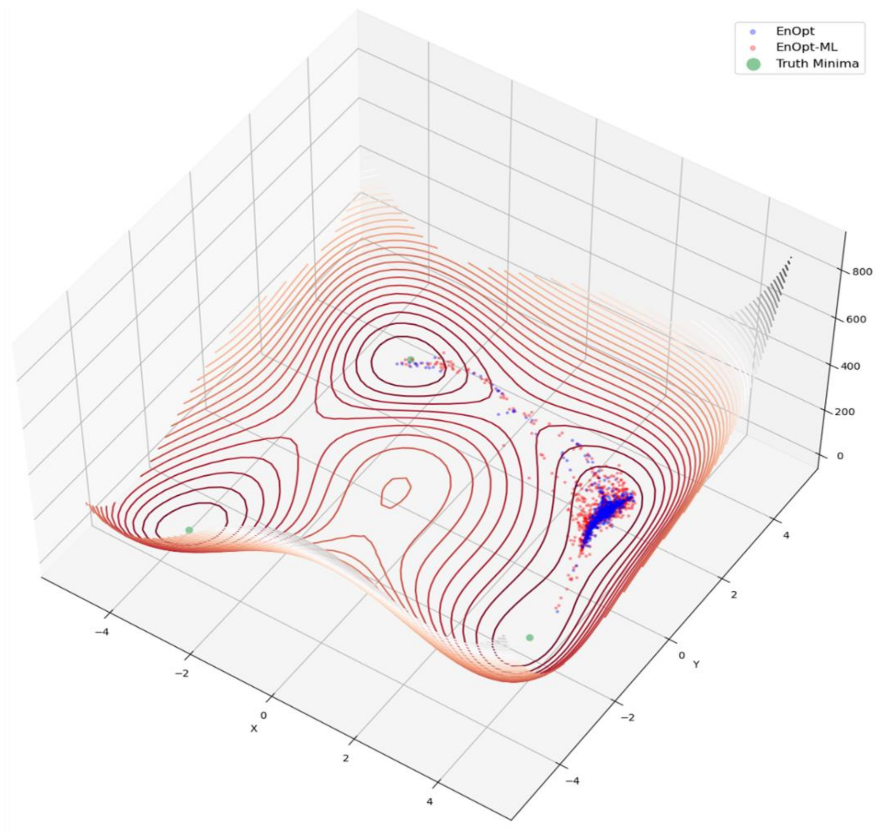


Figure 56 Case Nml = 20: simulations' results of EnOpt and EnOpt-ML with respect to true minimum values

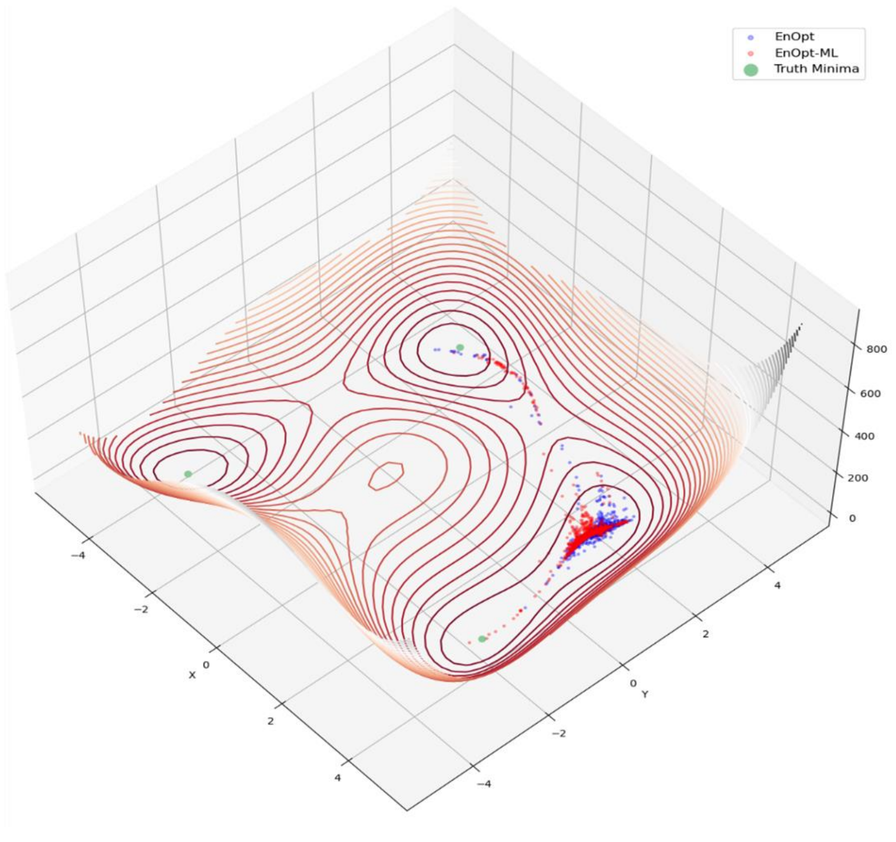


Figure 57 Case Nml = 1000: simulations' results of EnOpt and EnOpt-ML with respect to true minimum values

Different training iterations

In the process of using machine learning model on the gradient to observe its contribution/ effect on the optimization, we need to decide which data will be included in the model prediction samples. Here it is discussed whether it has an impact and if it does, how important of an impact it has on the optimization.

The following figures(fig 58) present the different iterations at which machine learning started being included in the algorithm (including the predicted samples in the gradient calculations).

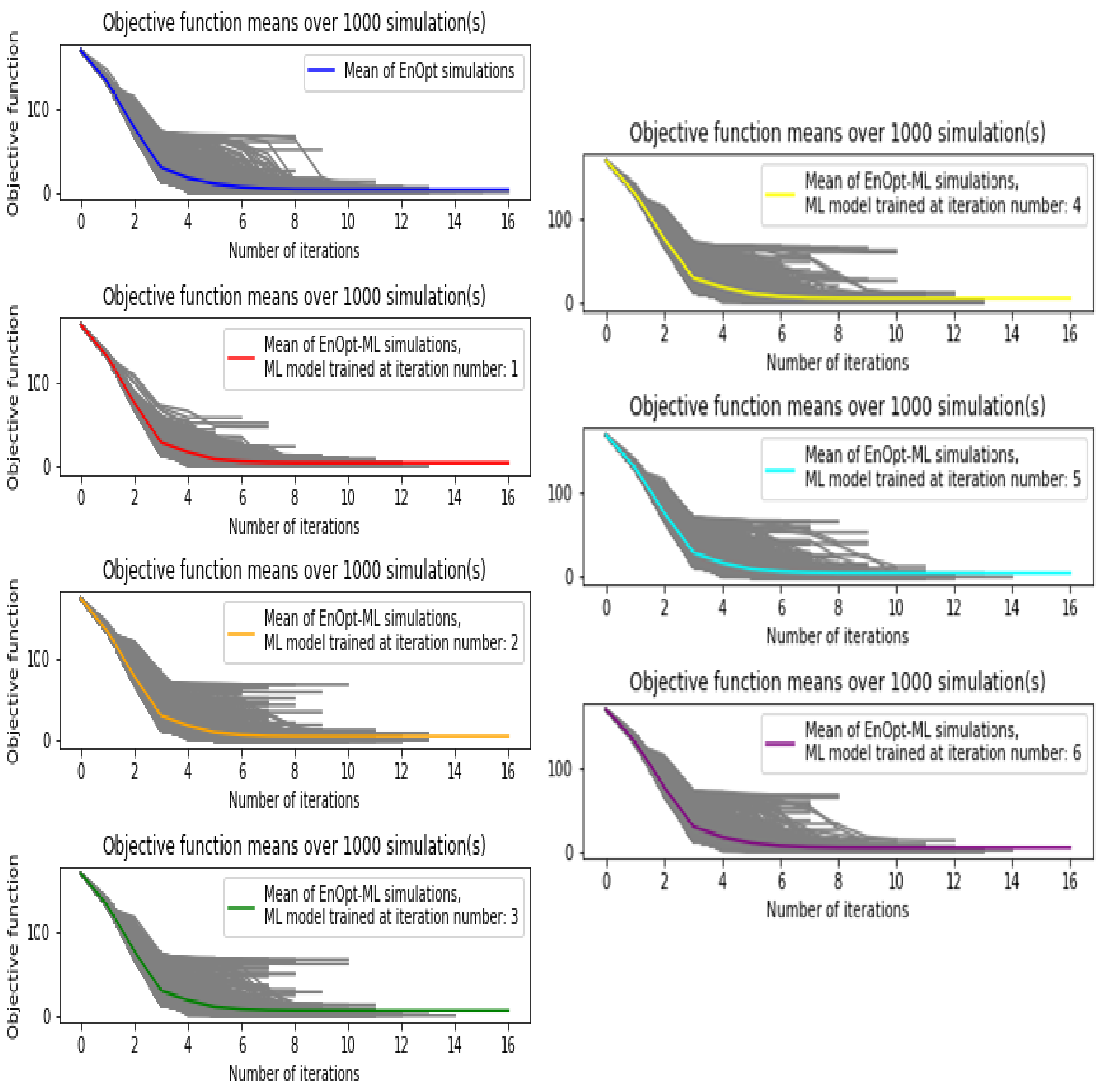


Figure 58 Objective function means over 1000 simulations for EnOpt, and EnOpt-ML with ML model trained at different iterations

When it is trained at first iteration (fig. 58), we can see that there is smaller deviation for the ensemble of simulations, but it has on the rest of the cases a shape (envelope of the simulations standard deviation) that start looking almost unchanged overall.

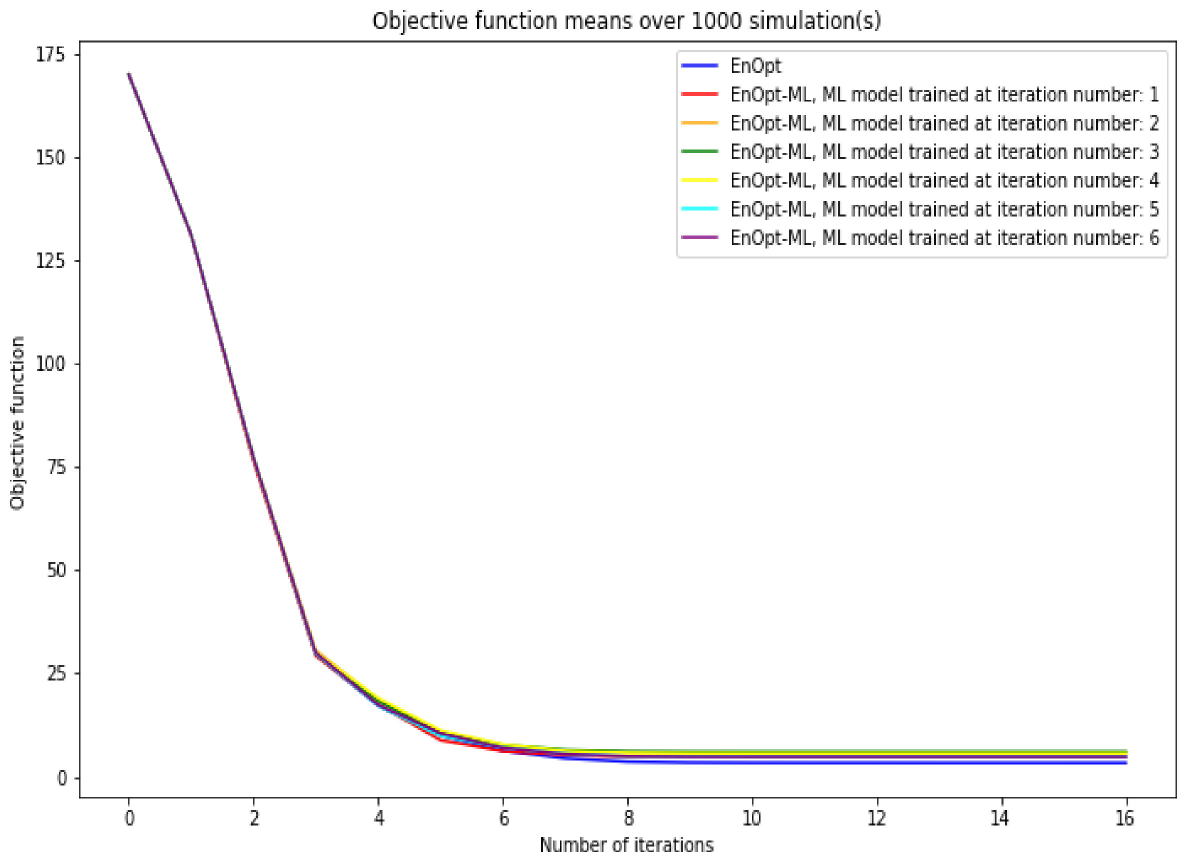


Figure 59 means of 1000 simulations for EnOpt and EnOpt-ML with ML model trained at different iterations

Comparing all the simulations' means on different trained iterations (fig. 59), we can observe that:

1. EnOpt lead to the lowest objective function in comparison with all the EnOpt-ML versions
2. The training at later iterations (5 and 6 here) led to better results compared to training at earlier stages.
3. The test function optimizations with all the different start training iteration cases are very close to one another .

Reduction of sample size N throughout the iterations

Reduced sample size N used at iteration 4 (after 3rd iteration)to N=5

- Single simulation

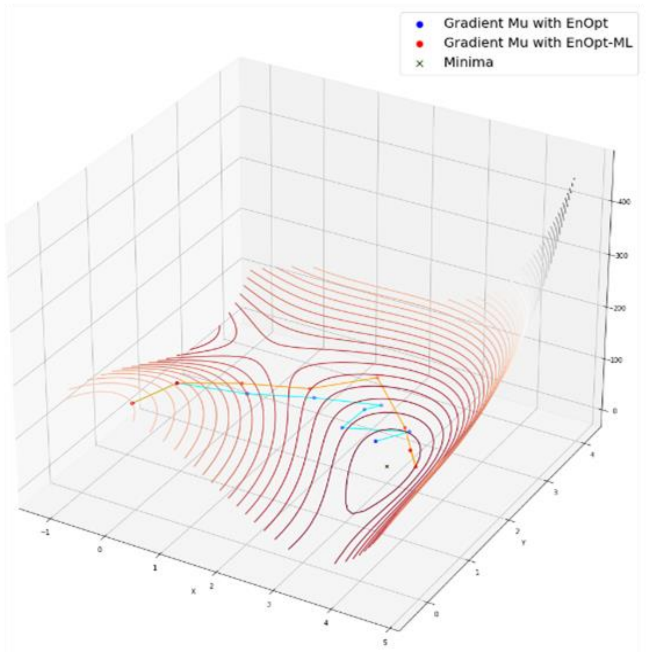


Figure 60 Reduced sample size N = 5 used at iteration 4 gradient (single simulation)

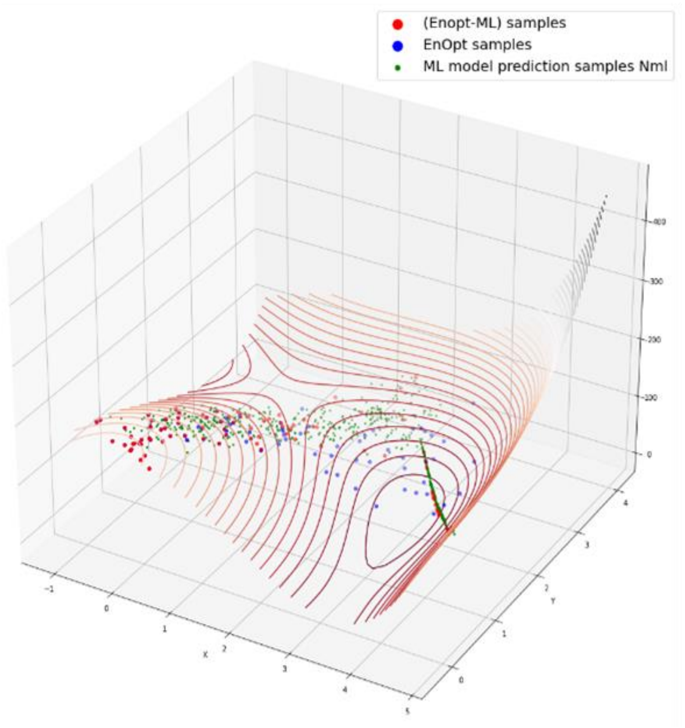


Figure 61 Reduced sample size N = 5 used at iteration 4 EnOpt and EnOpt-ML samples and ML model predicted samples

MSE: 52.55

Reduced sample size N used at iteration 4 (after 3rd iteration) to N=10

- Single simulation

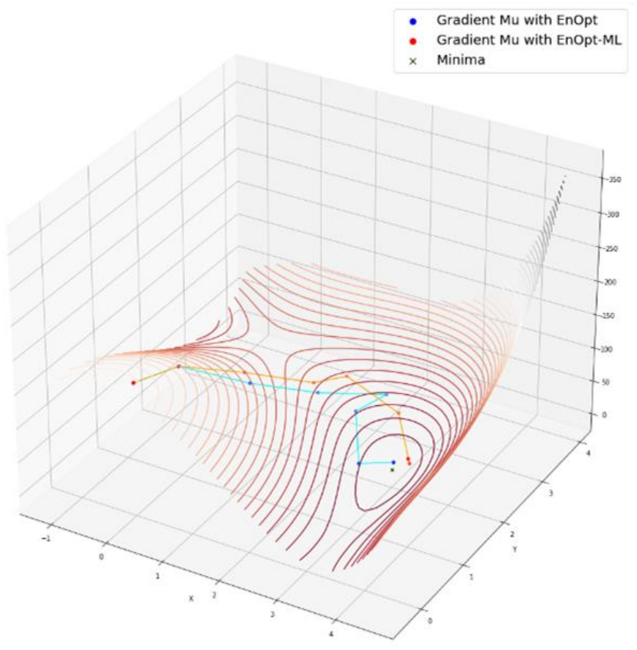


Figure 62 Reduced sample size N = 10 used at iteration 4 gradient (single simulation)

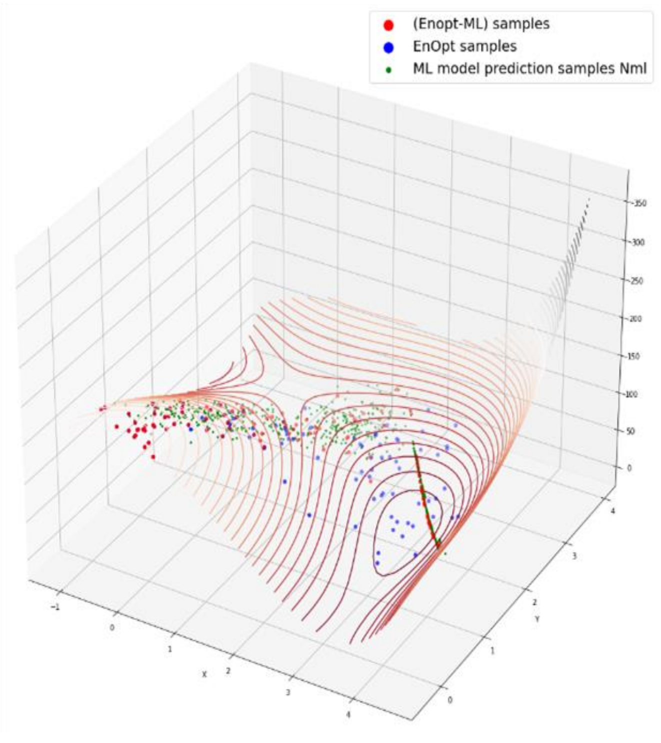


Figure 63 Reduced sample size N = 10 used at iteration 4 EnOpt and EnOpt-ML samples and ML model predicted samples

MSE: 35.49

In an attempt to reduce the number of the objective function evaluations, the sample size is reduced to check if it is possible to get a near-optimum value while at the same time running fewer objective function evaluations. The goal is also to observe if EnOpt-ML gives better result than EnOpt when the sampling on the original function is reduced or stopped.

In the case of a single simulations presented here, comparing the reduction to N=5 and N=10 we can see that the performance is better where the sample size is only reduced to 10 (fig. 60 and 62). There is a small difference between the EnOpt and EnOpt-ML results; in the reduction of sample size to N=5 , we get slightly better optimization on the EnOpt-ML front(4.99 in EnOpt-ML versus 6.26 in EnOpt, table 19). With the sample size reduced only to N=10, EnOpt had better optimization results than EnOpt-ML(0.39 in EnOpt compared 2.31to EnOpt-ML). The comparison of EnOpt-ML in the two cases, shows a double value of the optimized function when N=5 but with 50 fewer objective function evaluations (table 20) which is a good tradeoff between number of objective function evaluations and the optimal value.

Table 19 Optimization results for Reduced sample size N = 5 used at iteration 4 (single simulation)

	updated (near-optimal or optimal) control vector at the last iteration.	Number of iterations	Number of objective function evaluations	Optimized objective function
EnOpt	[2.529272 2.364079]	9	180	6.262037
EnOpt-ML	[3.122646 2.40612]	8	170	4.994715

Table 20 Optimization results for Reduced sample size N = 10 used at iteration 4 (single simulation)

	updated (near-optimal or optimal) control vector at the last iteration.	Number of iterations	Number of objective function evaluations	Optimized objective function
EnOpt	[2.92347 2.151243]	8	220	0.395203
EnOpt-ML	[3.087847 2.279605]	8	220	2.313051

- <u>1000 simulations:</u>

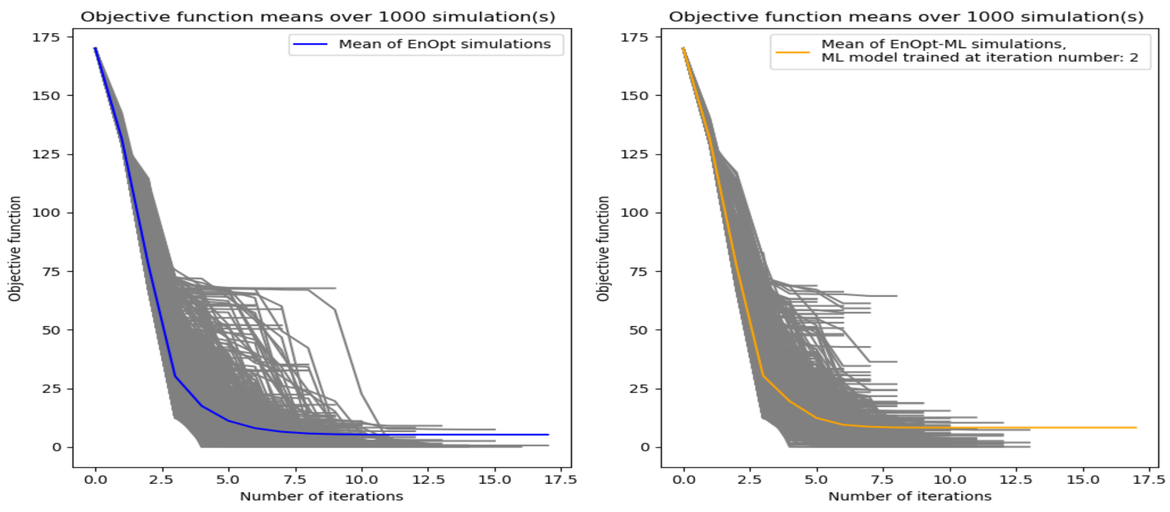


Figure 64 Reduced sample size N = 5 used at iteration 4 objective function optimization results (1000 simulations)

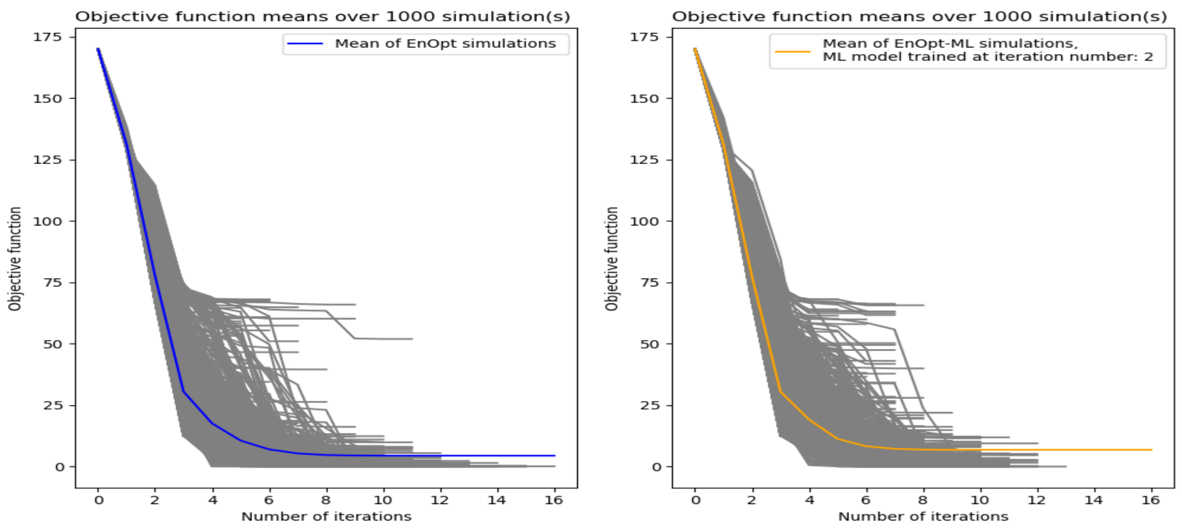


Figure 65 Reduced sample size N = 10 used at iteration 4 objective function optimization results (1000 simulations)

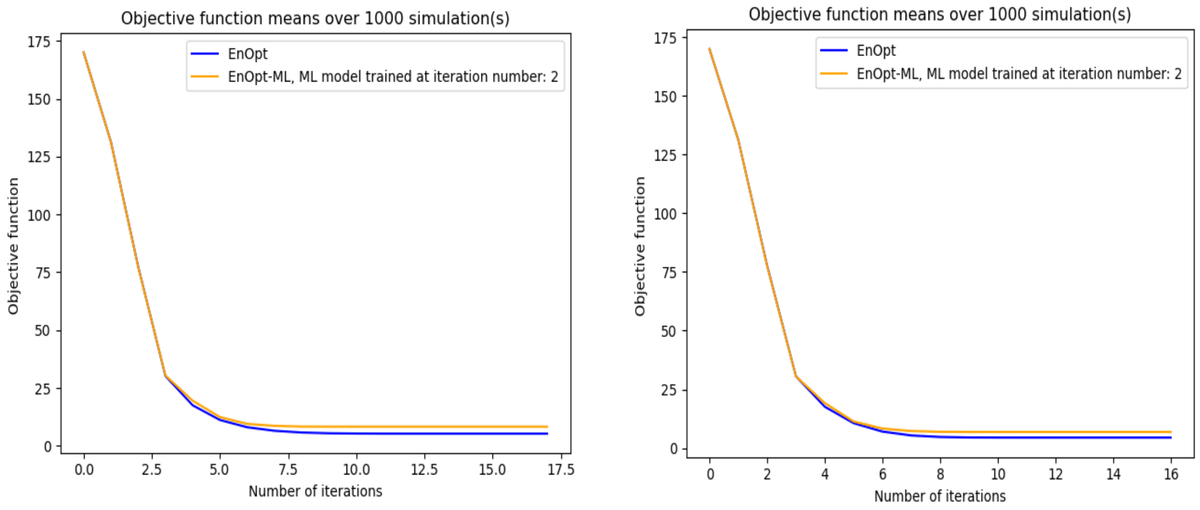


Figure 66 Reduced sample size N = 5(left)/N=10(right) used at iteration 4 EnOpt and EnOpt-ML means of simulations

From figures 61 and 63, we can see from the distribution of the different simulations (in gray), that more simulations of EnOpt and EnOpt-ML optimizations have stopped earlier than for the base configuration case.

The means (fig. 66) show that EnOpt had gotten slightly better value of optimization than EnOpt in both configurations.

At the fourth iteration, we have fewer function evaluations (see tables 21 and 22)when the sample size N has been reduced to 5 (168 for EnOpt and 161 for EnOpt-ML) and a smaller deviation (21 for EnOpt and 17 for EnOpt-ML). When N is reduced to 10 at the 4th iteration, the objective function is evaluated a greater number of times (226 for EnOpt and 208 for EnOpt-ML) with higher standard deviations too (37 for EnOpt and 32 for EnOpt-ML). The optimization value is, on the other hand, not significantly different on the means or standard deviations between the simulations on both cases of configuration.

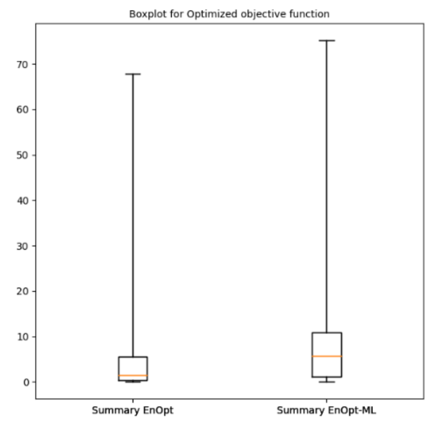


Figure 67 Reduced sample size N = 5 used at iteration 4 Boxplot of optimized objective function

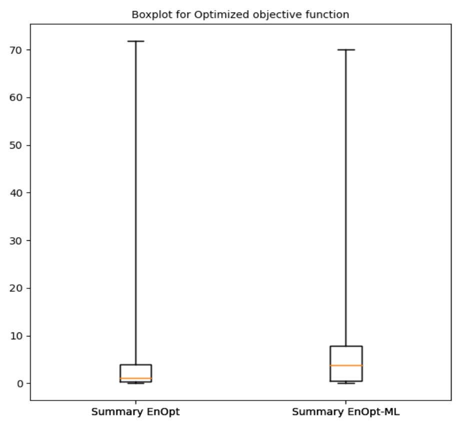


Figure 68 Reduced sample size N = 10 used at iteration 4 Boxplot of optimized objective function

The (fig 67 and 68) show that the variability is a little higher in EnOpt-ML values of the optimized objective function than on EnOpt for both of the cases where the 50% quartile is centered for EnOpt-ML but is very close to the 25% quartile for EnOpt.

Table 21 Reduced sample size N = 5 used at iteration 4 summary output values (1000 simulations)

EnOpt	iterations	Number of objective function evaluations	Optimized objective function	EnOpt-ML	iterations	Number of objective function evaluations	Optimized objective function
mean	7.85085085	168.5085085	5.232045866	mean	7.128128128	161.2812813	8.261514465
std	2.10562201	21.05622014	10.38866691	std	1.734825305	17.34825305	10.87836492
min	4	130	0.000064	min	3	120	0.000217
25%	6	150	0.336717	25%	6	150	1.1033665
50%	8	170	1.316315	50%	7	160	5.729338
75%	9	180	5.290503	75%	8	170	9.9790485
max	17	260	67.698888	max	13	220	70.197382

Table 22 Reduced sample size N = 10 used at iteration 4 summary output values (1000 simulations)

EnOpt	iterations	Number of objective function evaluations	Optimized objective function	EnOpt-ML	iterations	Number of objective function evaluations	Optimized objective function
mean	8.2982983	225.965966	4.391659362	mean	7.438438438	208.7687688	6.839160729
std	1.89110011	37.8220022	10.58703219	std	1.605340525	32.10681049	10.54530504
min	4	140	0.000001	min	4	140	0.000165
25%	7	200	0.24728	25%	6	180	0.793791
50%	8	220	1.020039	50%	7	200	3.781651
75%	10	260	3.594646	75%	8	220	8.0955925
max	16	380	68.10367	max	13	320	68.151035

Figures 69 and 70 do not show any interesting differences compared to each other and to the base case configuration.

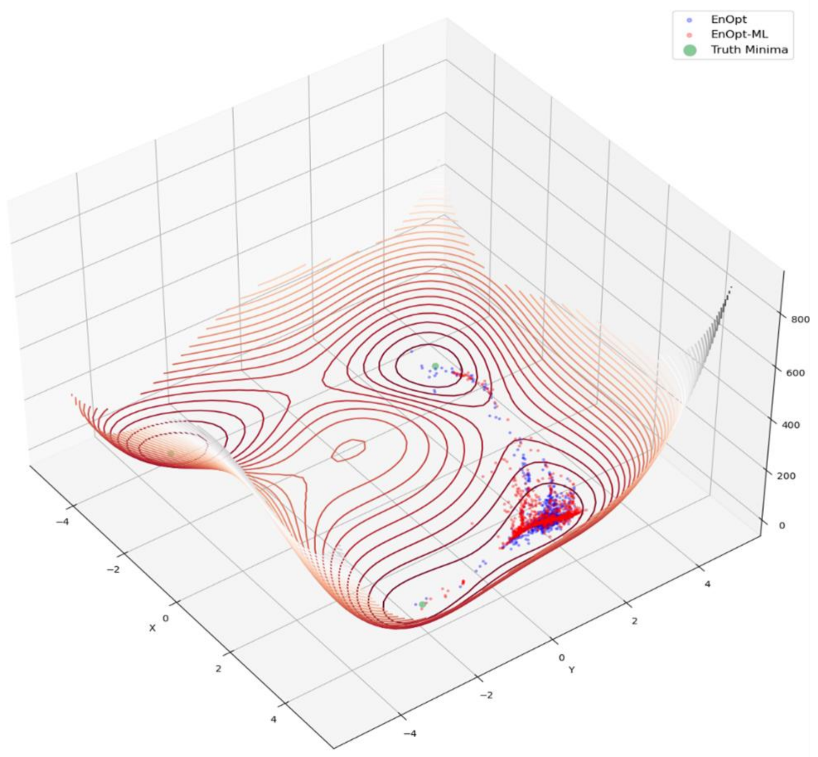


Figure 69 Reduced sample size N = 5 used at iteration 4 : simulations' results of EnOpt and EnOpt-ML with respect to true minimum values

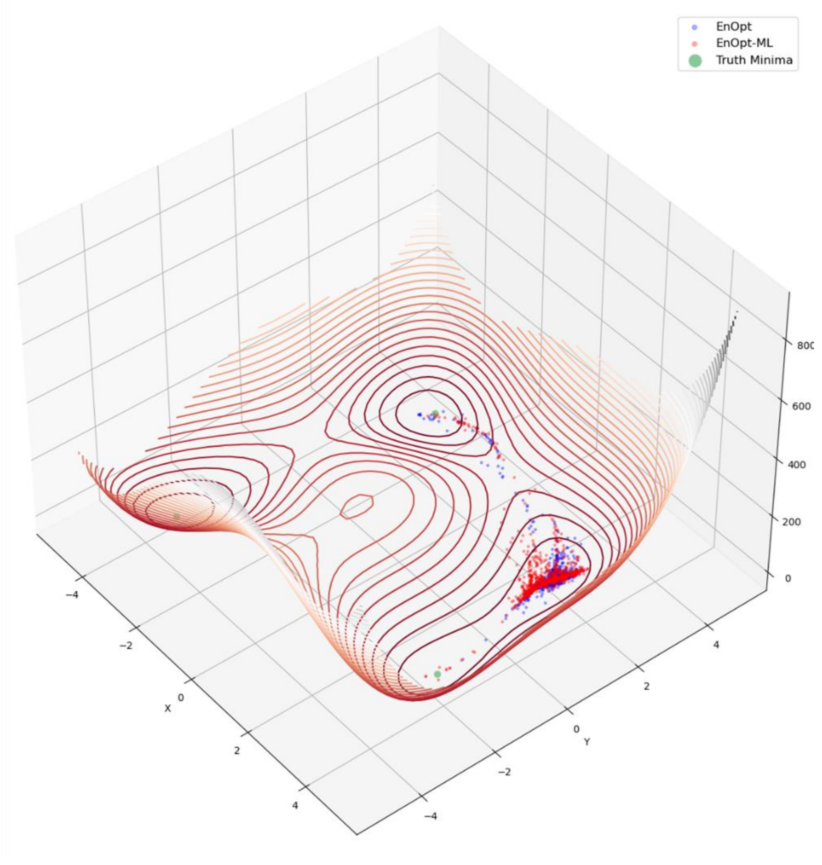


Figure 70 Reduced sample size N = 10 used at iteration 4 : simulations' results of EnOpt and EnOpt-ML with respect to true minimum values

Since the results of the reduction of sample size are interesting from the number of objective function evaluations point of view, a comparison with change in at what iteration we reduce N number of samples and the reduction to N=0 (i.e., sampling from the objective function is stopped) for both the cases involved is made in the following comparison:

- Reduced sample size N used at iteration 4 (after 3rd iteration) to N=0
- Single simulation:

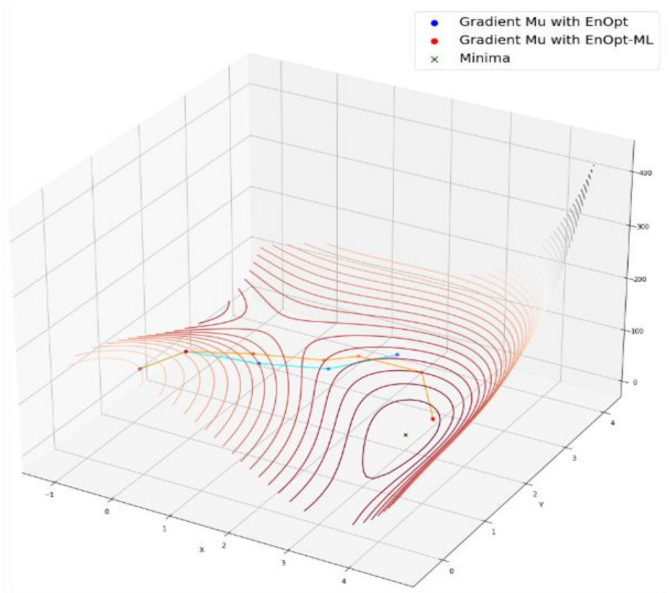


Figure 71 Case reduced sample size N = 0 used at iteration 4 gradient (single simulation):

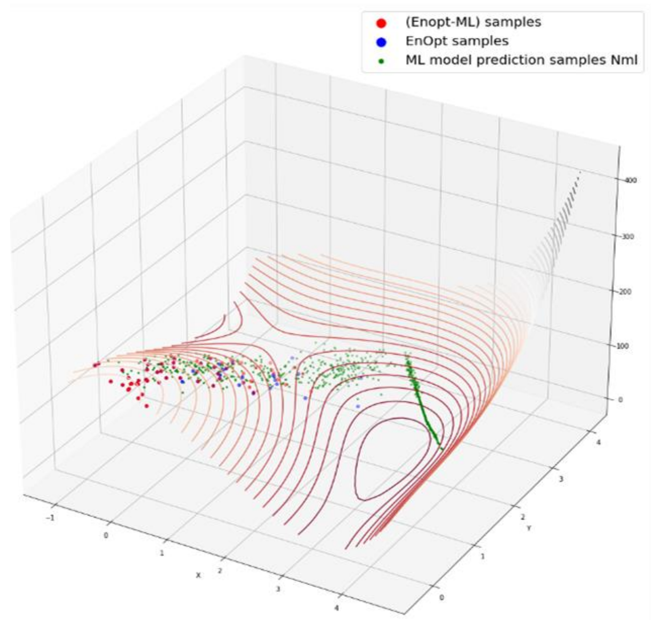


Figure 72 Case reduced sample size N = 0 used at iteration 4 EnOpt and EnOpt-ML samples and ML model predicted samples

MSE: 629.65

- Reduced sample size N used at iteration 5 (after 4th iteration) to N=0
- Single simulation:

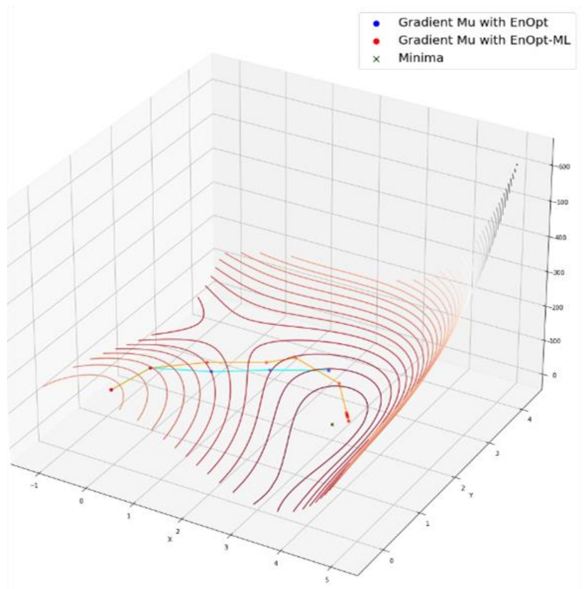


Figure 73 case reduced sample size N = 0 used at iteration 5: gradient (single simulation)

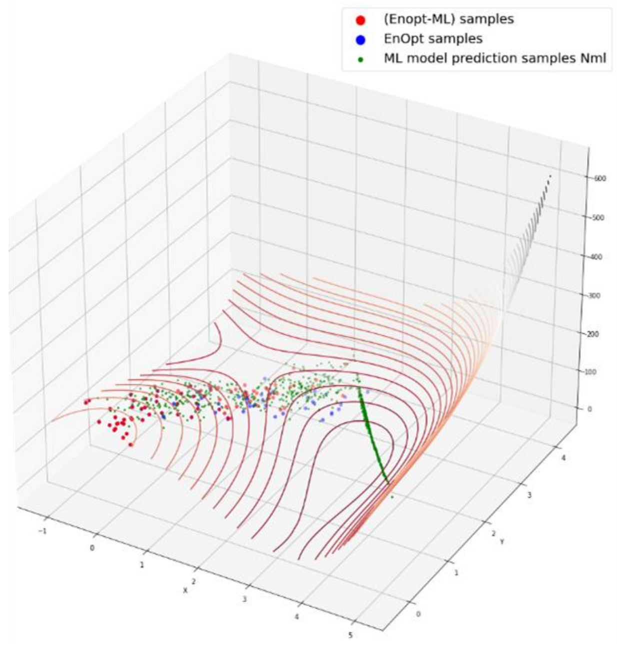


Figure 74 Case reduced sample size N = 0 used at iteration 5 : EnOpt and EnOpt-ML samples and ML model predicted samples

MSE: 709.66

The MSE in these configurations is much higher than we obtained in previous cases. It is due to that the ML model is predicting on the same samples since the stopping of objective function sampling.

Table 23 Optimization results for case reduced sample size N = 0 used at iteration 4 (single simulation)

	updated (near-optimal or optimal) control vector at the last iteration.	Number of iterations	Number of objective function evaluations	Optimized objective function
EnOpt	[1.328095 2.466779]	3	120	45.995224
EnOpt-ML	[3.165828 2.433299]	8	120	6.473882

Table 24 Optimization results for case reduced sample size N = 0 used at iteration 5 (single simulation)

	updated (near-optimal or optimal) control vector at the last iteration.	Number of iterations	Number of objective function evaluations	Optimized objective function
EnOpt	[2.173977 3.000148]	4	160	28.146441
EnOpt-ML	[3.088504 2.296758]	10	160	2.557661

The 1st case (N = 0 from Iteration 4) shows that EnOpt-ML led to better optimization value in comparison with EnOpt (tables 23 and 24) and so happened also for the 2nd case (N = 0 from Iteration 5). The second case has better optimization as it gathers more data from the extra iteration compared to where we stop sampling from the objective function at the previous iteration. For this simulation we can say that the optimization of reduced N at iteration 4 to N=0 yielded good enough optimization result (6.47 for EnOpt-ML) for EnOpt-ML if we compare it to the case where EnOpt run all the simulation on N=20 from the base configuration (3.21 for EnOpt and 2.99 for EnOpt-ML).

Reduced sample number to N=0 is equivalent to stopping all sampling from the objective function which would limit the number of objective function evaluations.

- <u>1000 simulations:</u>

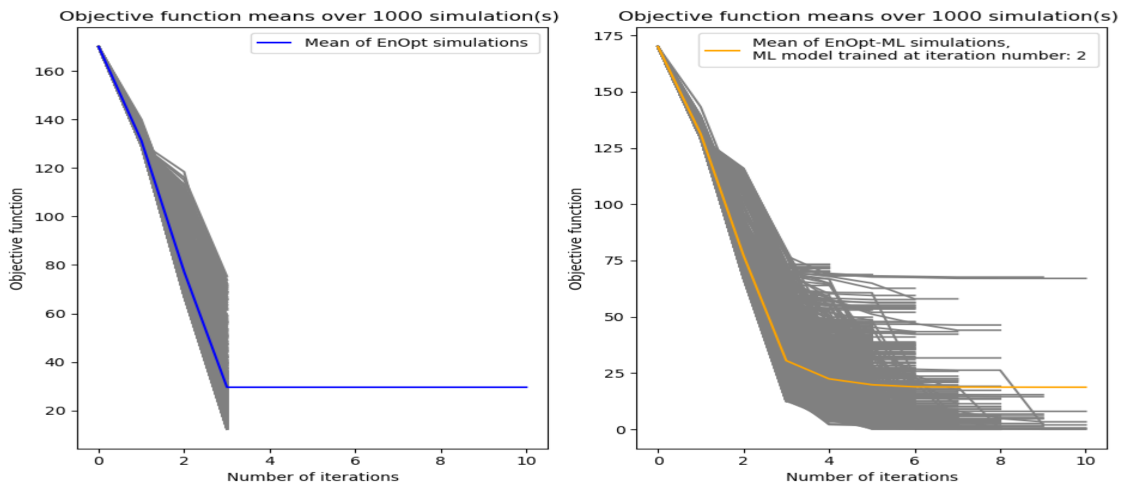


Figure 75 Case reduced sample size N = 0 used at iteration 4 : objective function optimization results (1000 simulations)

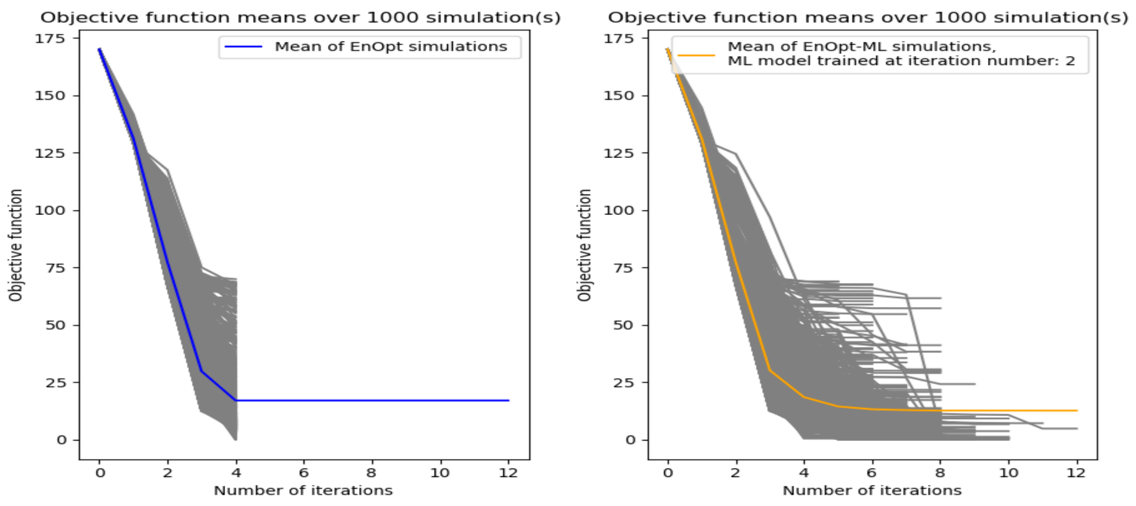


Figure 76 Case reduced sample size N = 0 used at iteration 5 : objective function optimization results (1000 simulations)

When stopping the sampling at 4th iteration, the mean on EnOpt-ML is not far from where it stopped at EnOpt (18.73 in EnOpt-ML versus 29.59 in EnOpt in average and the standard deviation are not very different tables 25 and 26). The optimizations are again not much dissimilar in value for all the cases.

In the case where we stop sampling at 5th iteration, the optimizations in EnOpt and EnOpt-ML are not very different (12.63 for EnOpt-ML versus 17.09 EnOpt, tables 25 and 26)

From the comparison of the mean values over the simulations' means, the EnOpt-ML is slightly better for the reason that we stop gathering EnOpt samples and so EnOpt does not improve after that(fig. 77 and 78). The boxplots(fig. 79 and 80) the 25% quartile of EnOpt in the case of stopping the sampling at the 4th iteration is higher than the 50% quartiles of EnOpt-

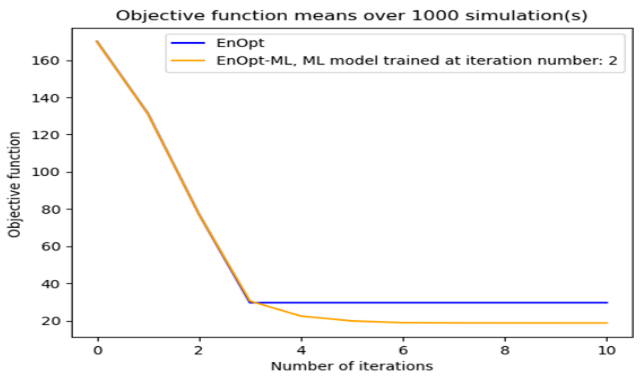


Figure 77 Case reduced sample size N = 0 used at iteration 4 : EnOpt and EnOpt-ML means of simulations

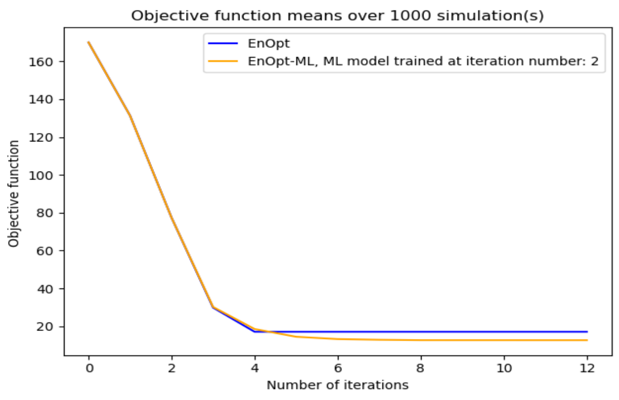


Figure 78 Case reduced sample size N = 0 used at iteration 5 : EnOpt and EnOpt-ML means of simulations

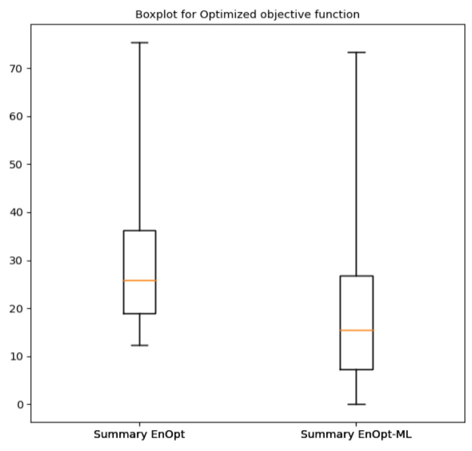


Figure 79 Case reduced sample size N = 0 used at iteration 4 : Boxplot of optimized objective function

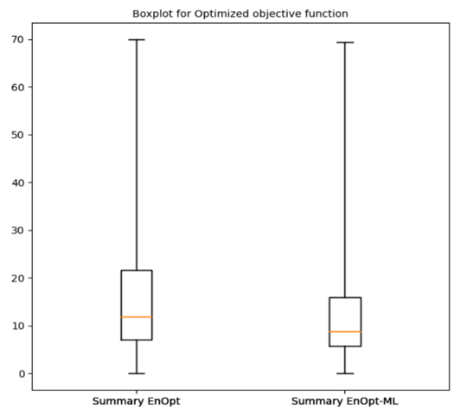


Figure 80 Case reduced sample size N = 0 used at iteration 5 : Boxplot of optimized objective function

ML of the same case and it is also higher than the 50% quartile of EnOpt and EnOpt-ML of the case where we stop the sampling at 5th iteration. This is showing that EnOpt in that case is poorly performing (fig. 81 and 82)

Table 25 Case reduced sample size N = 0 used at iteration 4 : summary output values (1000 simulations)

EnOpt	iterations	Number of objective function evaluations	Optimized objective function	EnOpt-ML	iterations	Number of objective function evaluations	Optimized objective function
mean	3	120	29.59701926	mean	5.445445445	120	18.73164531
std	0	0	14.02911777	std	1.167691648	0	14.71484949
min	3	120	12.270873	min	4	120	0.003434
25%	3	120	18.9094395	25%	5	120	7.3218875
50%	3	120	25.910577	50%	5	120	15.505178
75%	3	120	36.1877085	75%	6	120	26.826503
max	3	120	75.313342	max	10	120	73.31098

Table 26 Case reduced sample size N = 0 used at iteration 5 : summary output values (1000 simulations)

EnOpt	iterations	Number of objective function evaluations	Optimized objective function	EnOpt-ML	iterations	Number of objective function evaluations	Optimized objective function
mean	4	160	17.09531449	mean	5.947947948	159.95996	12.63735449
std	0	0	14.81086978	std	1.221588249	1.265543994	12.10253269
min	4	160	0.035556	min	3	120	0.005086
25%	4	160	7.1192285	25%	5	160	5.752444
50%	4	160	11.82127	50%	6	160	8.860696
75%	4	160	21.6730175	75%	6	160	15.90257
max	4	160	69.934194	max	12	160	69.432725

In Figures 81 and 82 we can appreciate a move towards the middle of the lower right quadrant of the figure from stopping at iteration 4 to stopping at iteration 5 for EnOpt minimums obtained in the multiple simulations. We do not appreciate that translation as much when it comes to EnOpt-ML.

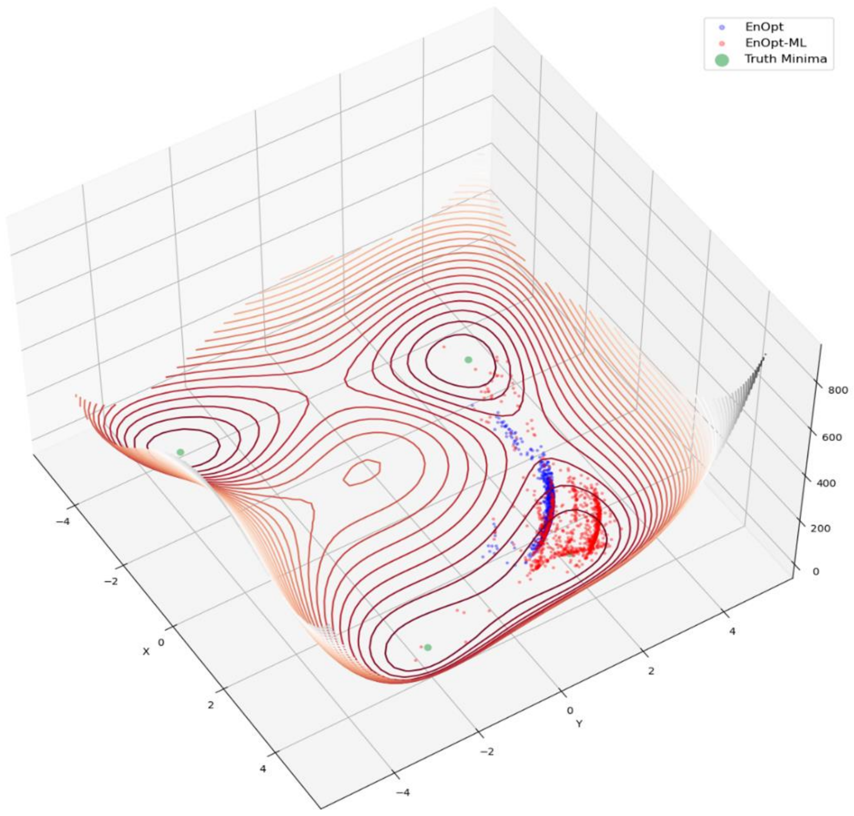


Figure 81 Case reduced sample size N = 0 used at iteration 4 : simulations' results of EnOpt and EnOpt-ML with respect to true minimum values

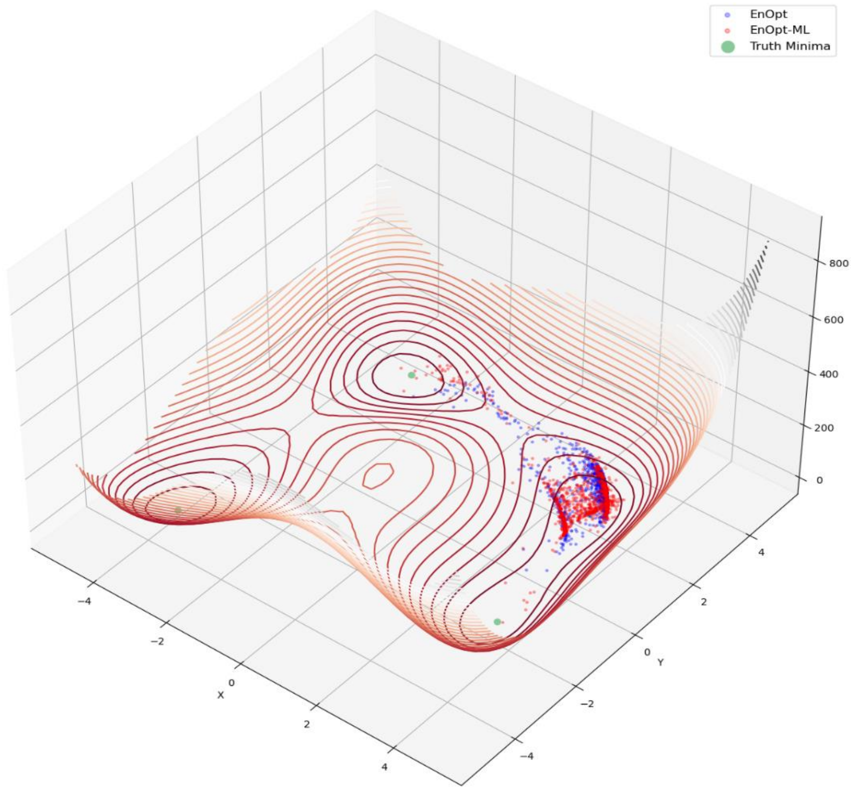


Figure 82 Case reduced sample size N = 0 used at iteration 5 : simulations' results of EnOpt and EnOpt-ML with respect to true minimum values

Machine learning algorithms

There are various machine learning algorithms that could improve the performance of the gradient in EnOpt. As introduced in chapter 2, in this chapter two ML algorithms are used. AdaBoost and Random Forest

Adaptive boosting regressor:

- Single simulation:

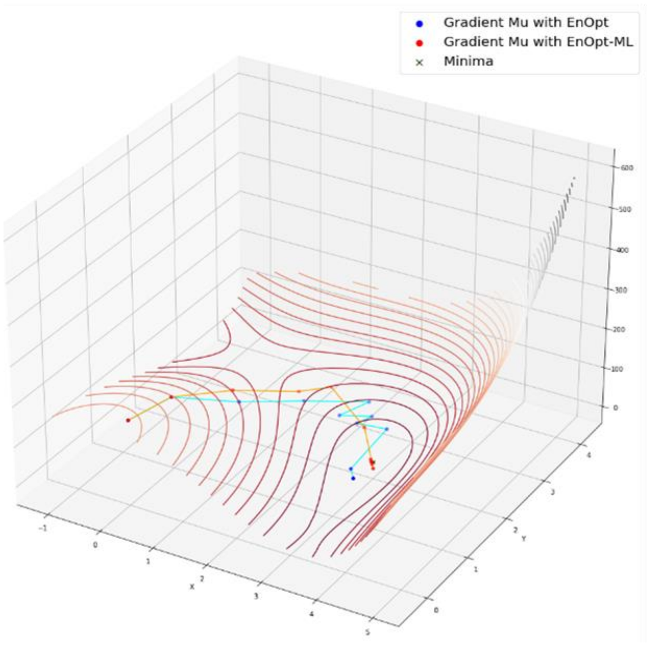


Figure 83 AdaBoost case: gradient (single simulation)

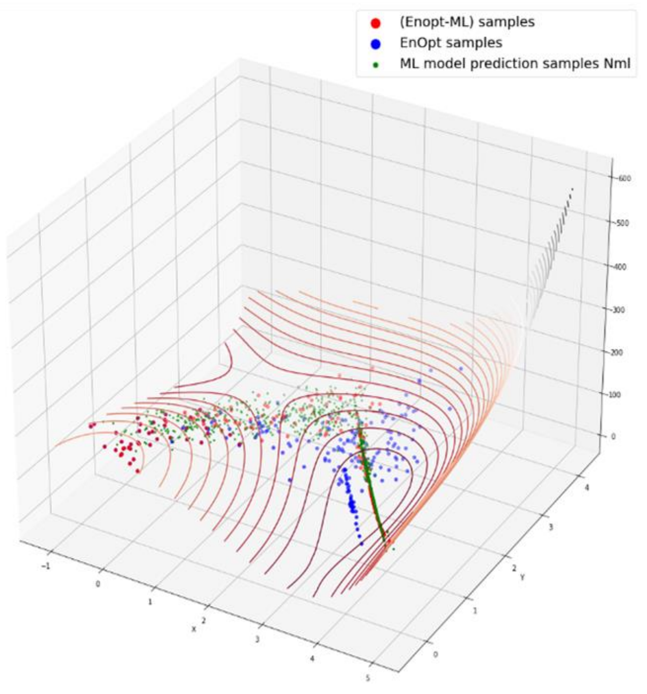


Figure 84 AdaBoost case: EnOpt and EnOpt-ML samples and ML model predicted samples

MSE: 56.4

Random Forest:

- Single simulation:

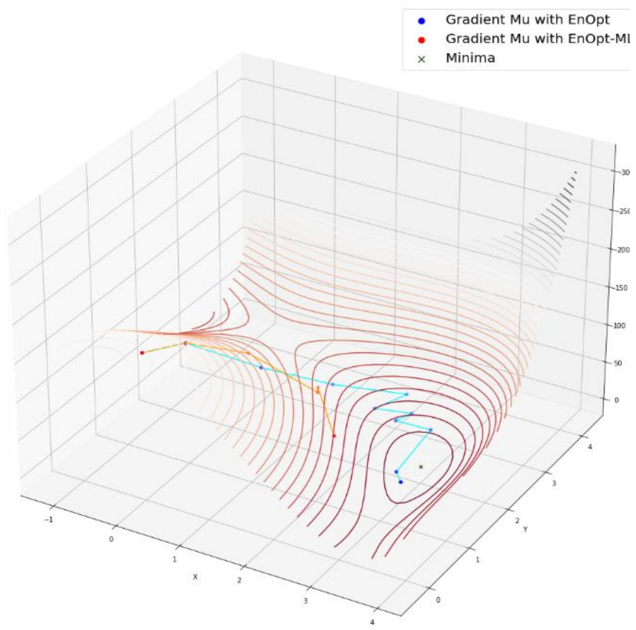


Figure 85 Random Forest case: gradient (single simulation)

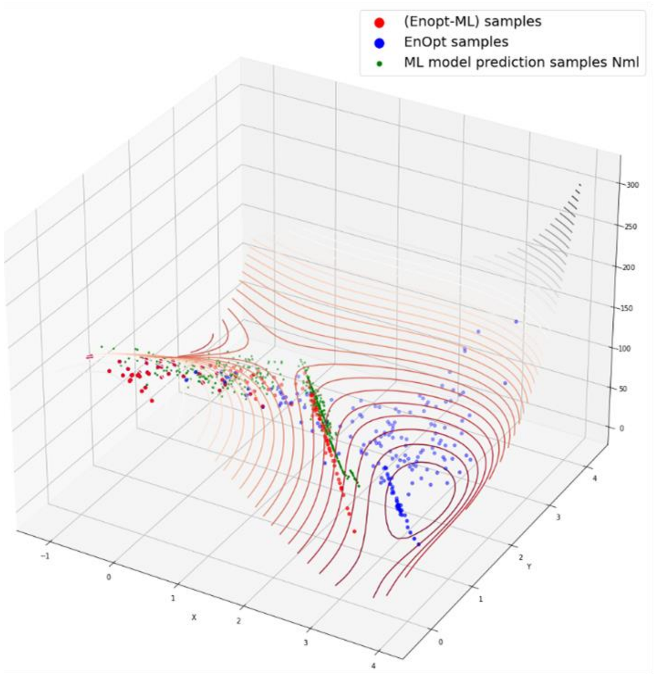


Figure 86 Random Forest case: EnOpt and EnOpt-ML samples and ML model predicted samples

MSE: 368.46

EnOpt-ML with Random forest stopped iterating early in the optimization and on an optimized value that is pretty high (39.44 table 28) compared to both AdaBoost (0.03, table 27) and GBR (2.99 table 1).

The AdaBoost Regressor performed best compared to the base configuration with GBR and the Random Forest regressor which performed the worst and got stuck pretty early along the slope. The AdaBoost regressor actually performed even better than EnOpt for this single simulation.

The MSE of the Random Forest shows it cannot fit the training data well as the MSE 368.46, in the contrary of MSE of AdaBoost 56.4 and the MSE of GBR with 34.39.

Table 27 Optimization results for AdaBoost case (single simulation)

	updated (near-optimal or optimal) control vector at the last iteration.	Number of iterations	Number of objective function evaluations	Optimized objective function
EnOpt	[2.995507 1.511997]	11	440	3.21792
EnOpt-ML	[2.975953 1.986702]	10	400	0.030587

Table 28 Optimization results for Random Forest case (single simulation)

	updated (near-optimal or optimal) control vector at the last iteration.	Number of iterations	Number of objective function evaluations	Optimized objective function
EnOpt	[2.995507 1.511997]	11	440	3.21792
EnOpt-ML	[1.969806 1.495179]	6	240	39.447073

- <u>1000 simulations:</u>

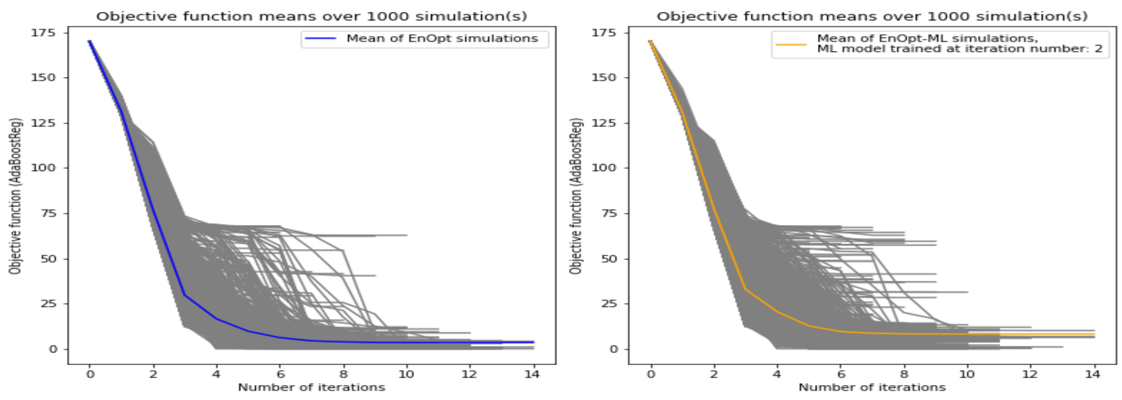


Figure 87 AdaBoost case: objective function optimization results (1000 simulations)

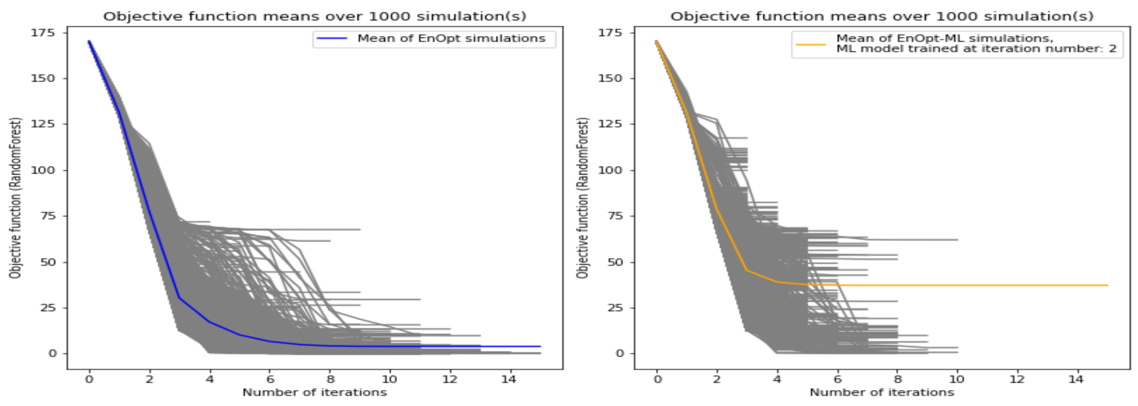


Figure 88 Random Forest case: objective function optimization results (1000 simulations)

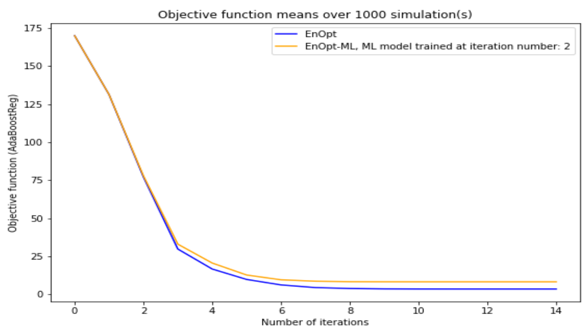


Figure 89 AdaBoost case: EnOpt and EnOpt-ML means of simulations

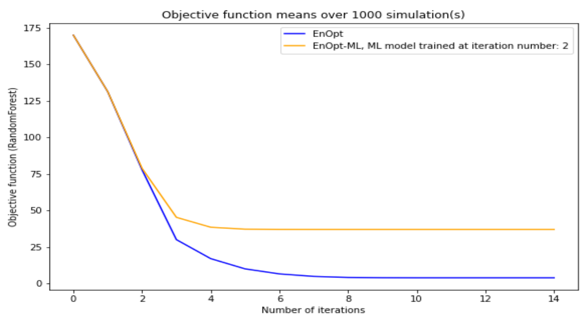


Figure 90 Random Forest case: EnOpt and EnOpt-ML means of simulations

Most of the simulations on Random Forest end earlier in the iteration process (an average of 5 iterations) while the means are around 38.21(table 30). For AdaBoost on the other hand, the optimum mean is almost 5 times lower and ends in average around the 7th iteration. The AdaBoost regressors results are much closer to the GBR (6.38 for GBR versus 8.19 for AdaBoost, tables 2 and 29)from the simulations obtained here.

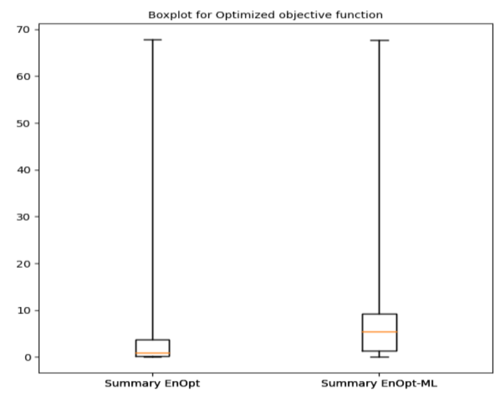


Figure 91 AdaBoost case: Boxplot of optimized objective function

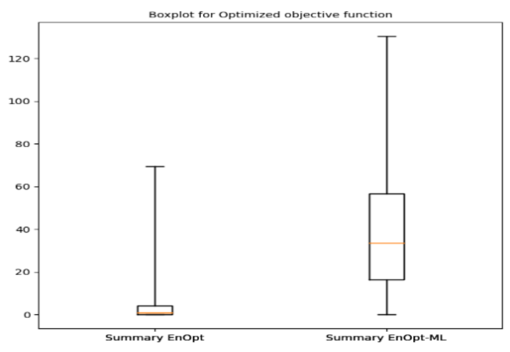


Figure 92 Random forest case : Boxplot of optimized objective function

Table 29 AdaBoost case: summary output values (1000 simulations)

EnOpt-ML	iterations	Number of objective function evaluations	Optimized objective function
mean	7.342342342	293.6936936	8.193356594
std	1.591220564	63.64882258	11.98542666
min	4	160	0.000287
25%	6	240	1.2906725
50%	7	280	5.452069
75%	8	320	9.1903405
max	14	560	67.718445

Table 30 Random Forest case: summary output values (1000 simulations)

EnOpt-ML	iterations	Number of objective function evaluations	Optimized objective function
mean	4.691691692	187.6676677	38.21004506
std	1.204134927	48.16539708	24.95133907
min	2	80	0.000265
25%	4	160	16.40944
50%	4	160	33.570848
75%	5	200	56.79608
max	11	440	130.385887

From the figures 91 and 92 are an illustration of how poor is the efficacy of Random Forest regressor in this case in comparison with GradBoost regressor, AdaBoost regressor and EnOpt. as we can see that EnOpt-ML with Random Forest has very large quartiles compared to EnOpt-ML with AdaBoost and also with EnOpt.

From figure 93, we see that the AdaBoost regressor as an ML used for EnOpt-ML has the simulations' optima's distributed very similarly to when GBR was used. Figure 94, on the other hand, shows that Random forest's EnOpt-ML simulations yielded in general poor results that only very few simulations led to optimizations that are closer to one of the minima.

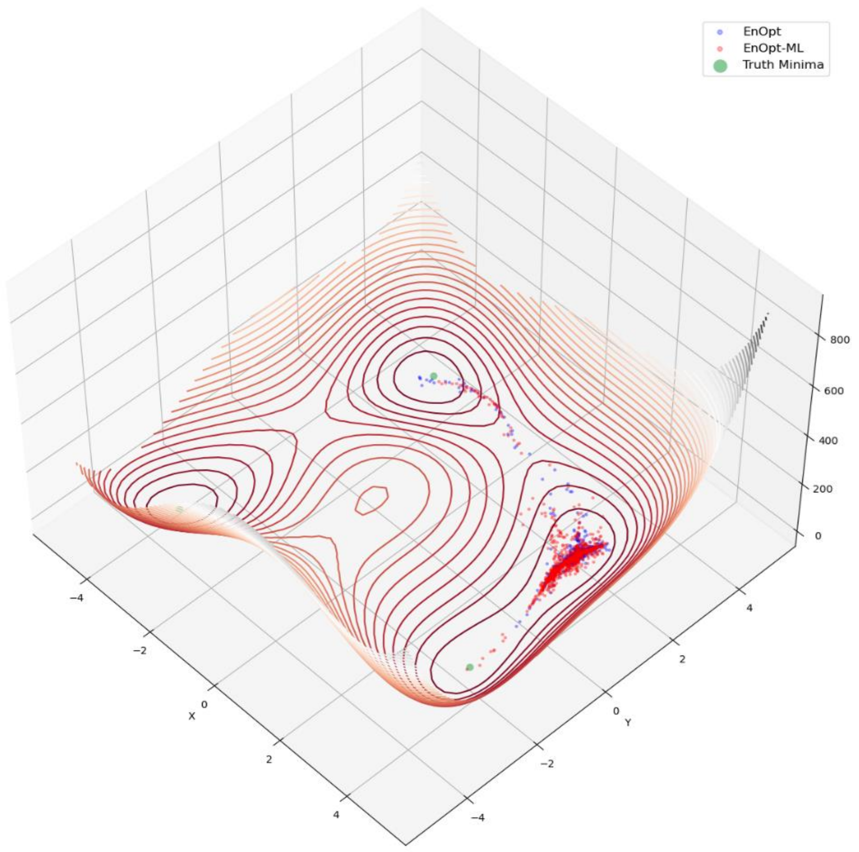


Figure 93 AdaBoost case: simulations' results of EnOpt and EnOpt-ML with respect to true minimum values

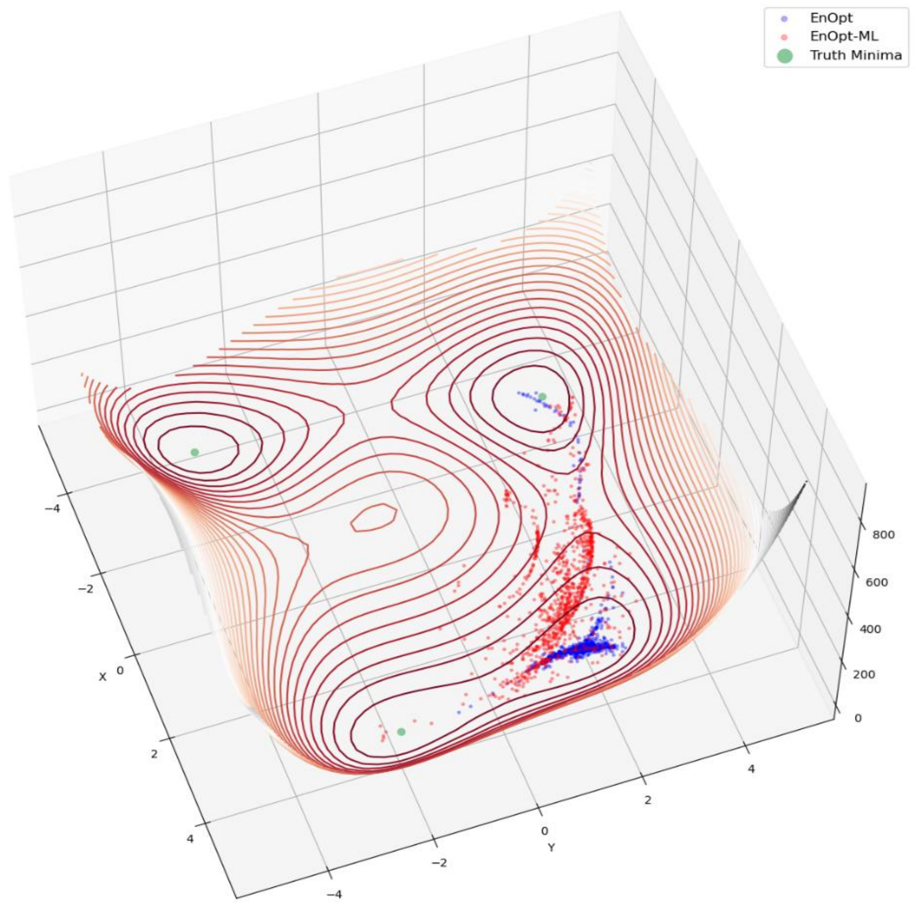


Figure 94 Random Forest case: simulations' results of EnOpt and EnOpt-ML with respect to true minimum values

Chapter 4: Improvement example

Based on the previous cases observations, a collection of 'best' performers is applied to Himmelblau, Rosenbrock and Bukin function number 6 for optimization testing. Some interconnections in terms of optimizing through the reduction of time the objective function has been evaluated and the overall average of performed iterations are taken into account. These two latter functions will be an example to observe the possibility of generalization for the use of ML algorithms to improve the gradient estimation when also at a given iteration, we reduce the number of samples added.

The "best" configuration is as the following:

- Number of samples N = 20,
- Initial Covariance matrix [0.2 0 0 0.2],
- Machine learning model, GBR trained at 2nd iteration,
- Reduced N = 0 (stop sampling from objective function) reduced at 5th iteration.
- Nml = 500

On Himmelblau

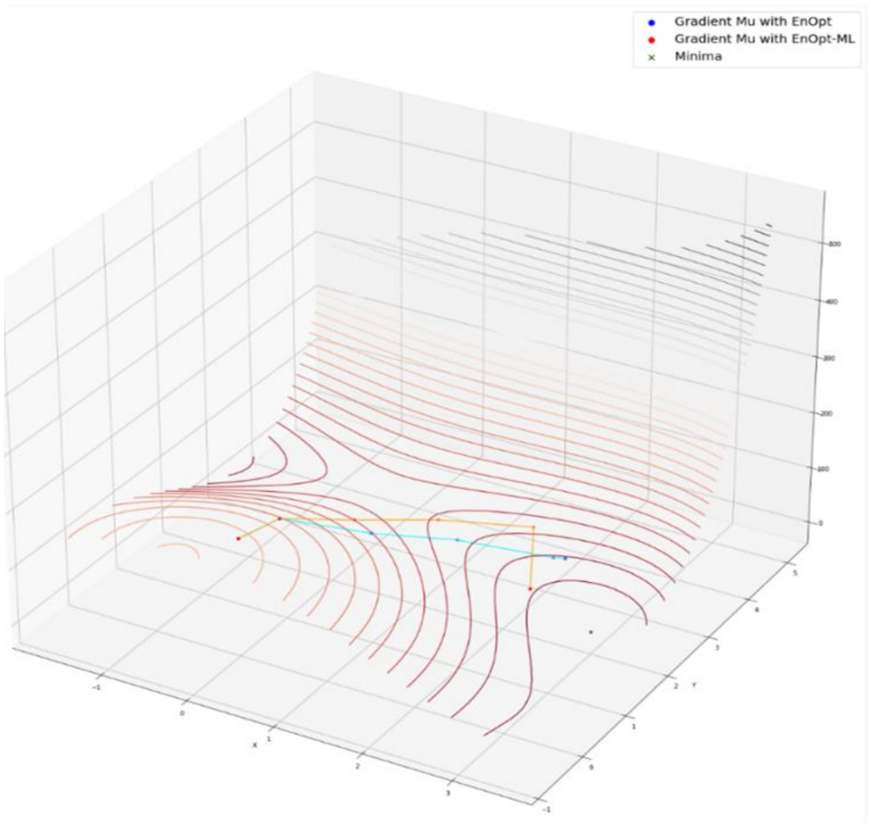


Figure 95 Improvement example: gradient (single simulation)

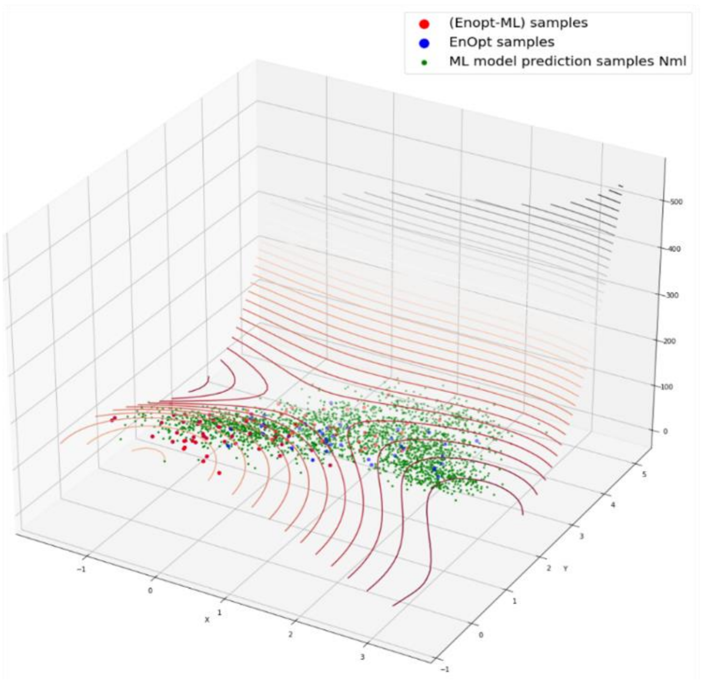


Figure 96 Improvement example EnOpt and EnOpt-ML samples and ML model predicted samples

MSE: 454.9

In this case, a simulation with 20 samples in each iteration has been performed until the 4th iteration and at the 5th iteration there were no new samples used. The relatively large number of model-predicted samples Nml in combination with the stopping of sample collection at 5th iteration explains the MSE obtained (454.9) because the ML model is not learning on any new data, hence the errors between the predicted samples and the actual test function become more pronounced.

This case is a recollection of what was observed in the previous chapter and could improve the optimization to get a 'good enough' result with smaller number of evaluations of the objective function (because of fewer iterations) through the optimization.

There are, of course, many other configurations that could give even better results. Here in this thesis, this case will mainly serve as an example to what could be done with respect to reducing the objective function evaluations by using machine learning algorithms on the gradient estimation.

The total number of iterations specified for reducing the sample size to zero, is doubled (for evaluating upon update) then multiplied by the number of samples used up to that point, is usually the objective function run total. It is not always the case, as the optimization may stop before the iteration, at which point we put N=0. This means that by determining the setting, we can determine the maximum number of times the function will be executed ahead of time.

Table 31 Optimization results for improvement example (single simulation)

	updated (near-optimal or optimal) control vector at the last iteration.	Number of iterations	Number of objective function evaluations	Optimized objective function
EnOpt	[2.098966 2.970574]	4	160	28.523781
EnOpt-ML	[2.145439 2.332841]	6	160	16.86338

After stopping the sampling, EnOpt-ML improved the results of the gradient by more than 1.5 times(table 31). The result obtained by EnOpt at that stop iteration.

- <u>1000 simulations:</u>

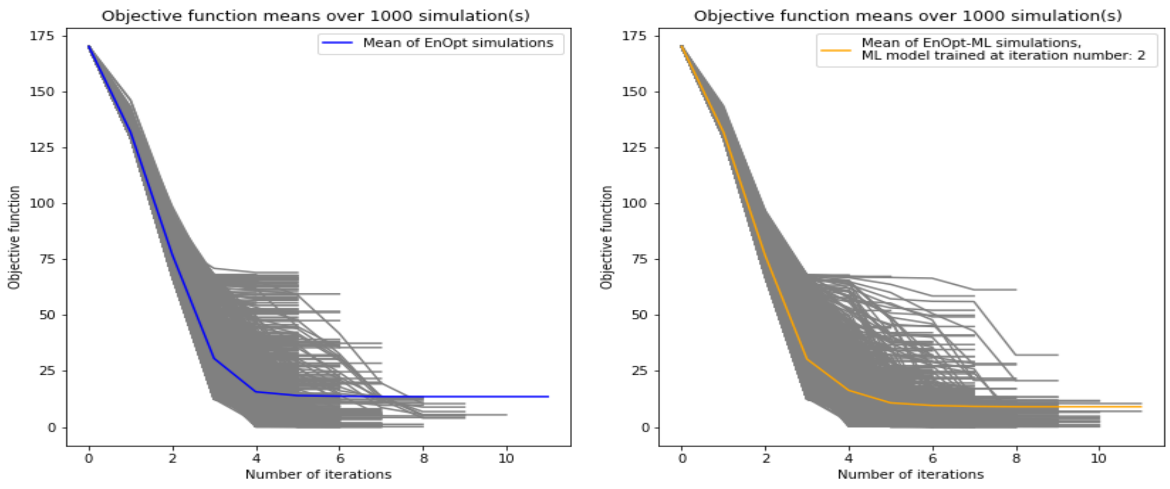


Figure 97 Improvement example objective function optimization results (1000 simulations)

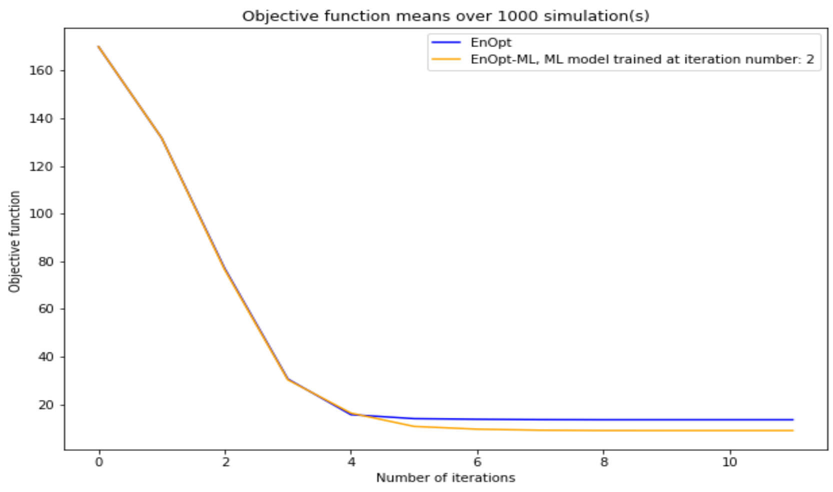


Figure 98 Improvement example EnOpt and EnOpt-ML means of simulations

The 1000 simulation in this configuration has shown an improvement in the optimization with the use of the EnOpt-ML (mean of 8.09 compared to 14.99 for EnOpt, table 32) even when reducing/stopping the sampling. The mean resulted from these simulations is satisfactory especially that the function run only 160 times which is fewer than what we observed in the base case configuration (333.65 as average from EnOpt and 307.42 from EnOpt-ML, table 2) without

Figure 99 shows medians with close values in the EnOpt and EnOpt-ML boxplots, with the EnOpt boxplot skewed towards the 25% quartile while having a 75% quartile that is significantly higher than the EnOpt-ML boxplot.

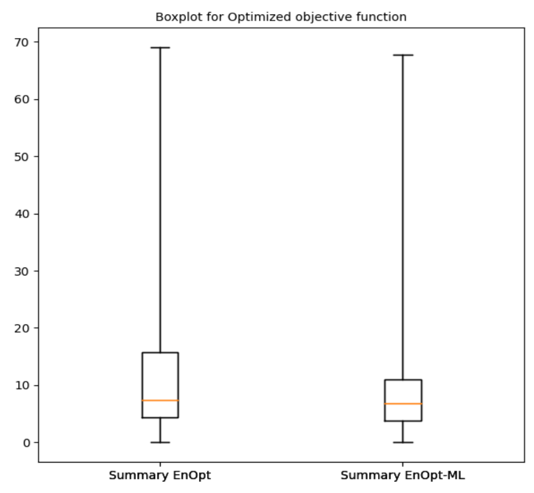


Figure 99 improvement example Boxplot of optimized objective function

Table 32 Improvement example: summary output values (1000 simulations)

EnOpt	iterations	Number of objective function evaluations	Optimized objective function	EnOpt-ML	iterations	Number of objective function evaluations	Optimized objective function
mean	4	160	14.99233674	mean	6.318318318	160	8.096639766
std	0	0	15.65234561	std	1.356633954	0	7.807887614
min	4	160	0.030159	min	5	160	0.000551
25%	4	160	5.054908	25%	5	160	3.1054145
50%	4	160	8.585471	50%	6	160	6.399106
75%	4	160	19.368154	75%	7	160	9.908607
max	4	160	67.749574	max	13	160	66.536961

The plot (fig. 100) shows that most simulations yield optimized values that are around the true minimum on the lower-right of the figure.

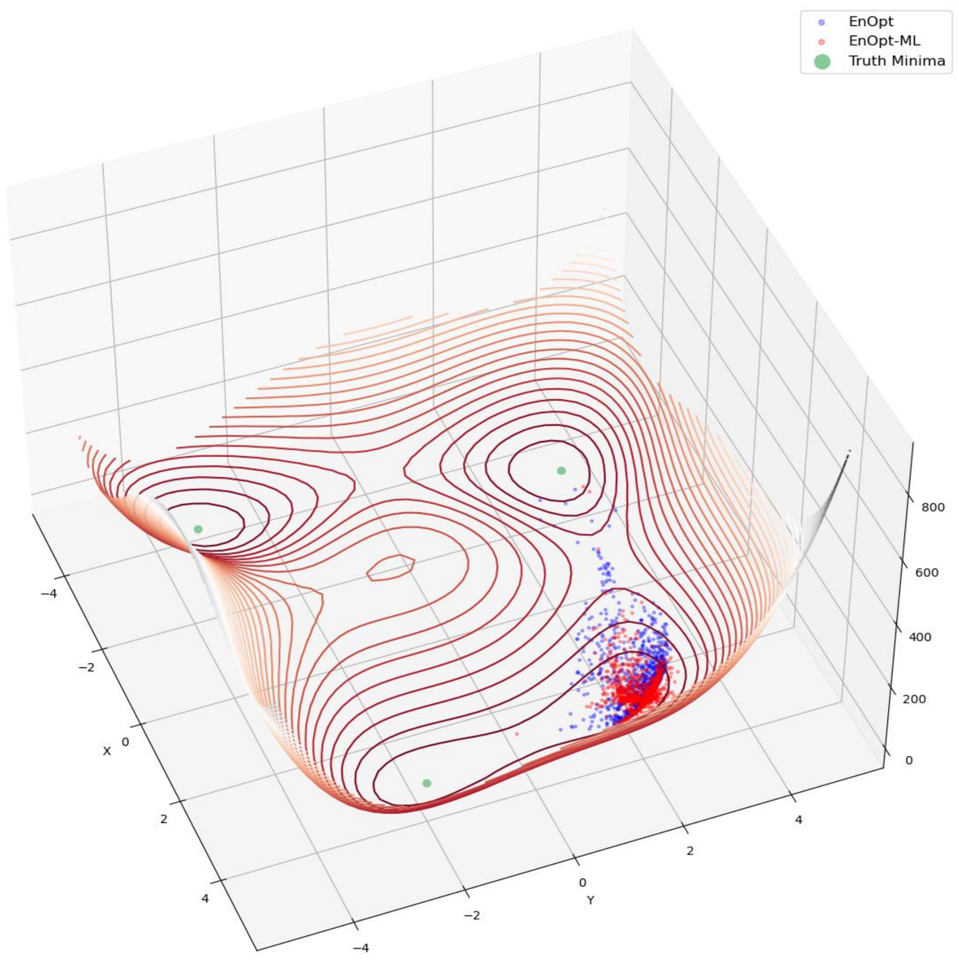


Figure 100 Improvement example simulations' results of EnOpt and EnOpt-ML with respect to true minimum values

On Buckin function number 6

The configuration here has been deviated from the Himmelblau in the previous subchapter configuration by:

- test function : Bukin function number 6,
- starting point [-12 , 2].

- Single simulation

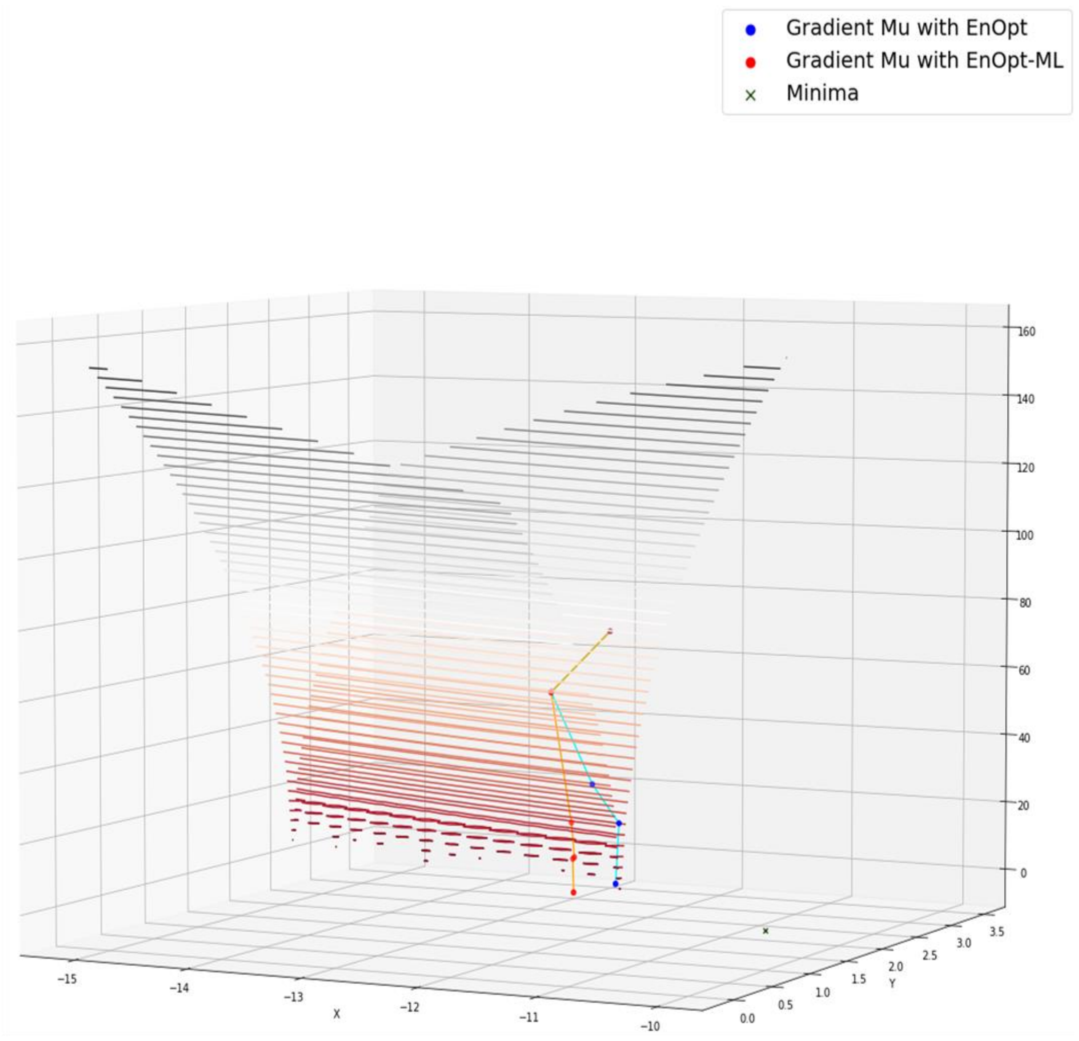


Figure 101 Bukin function 6: gradient (single simulation)

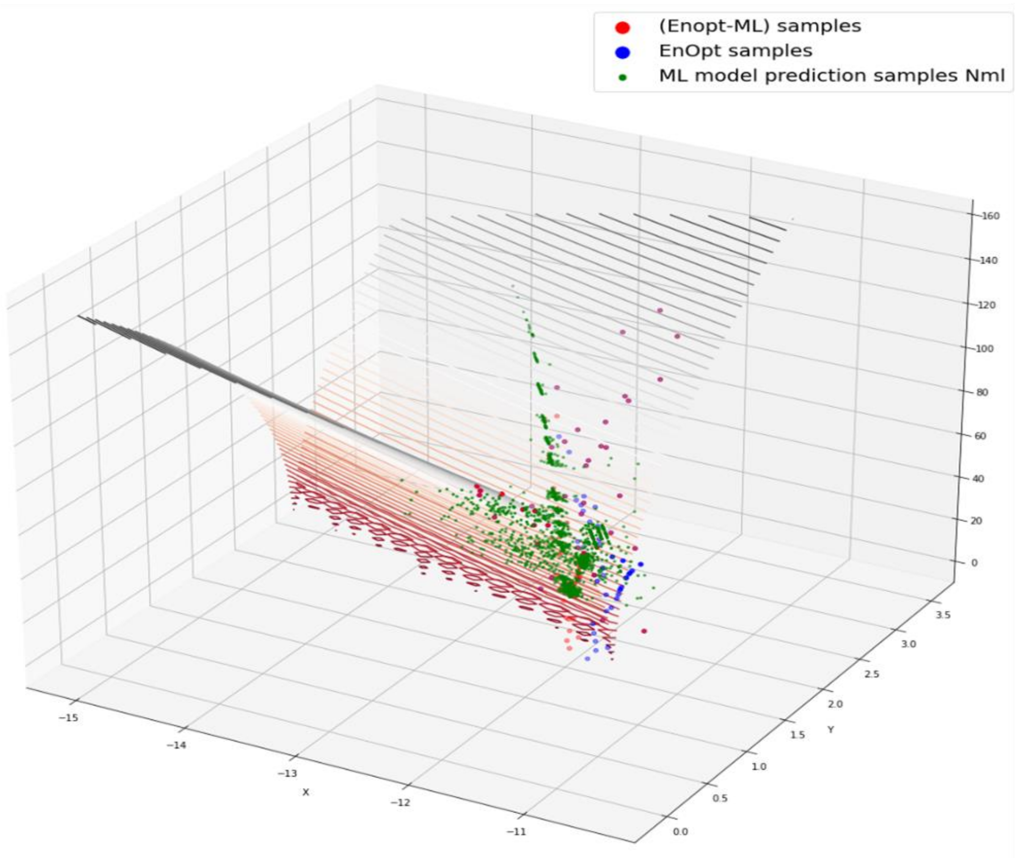


Figure 102 Bukin function 6: EnOpt and EnOpt-ML samples and ML model predicted samples

MSE: 64.62

MSE of the ML model (64.62) is lower than the one obtained with Himmelblau (454.9); this is due to the shape of Bukin function number 6.

The Buckin function n6 is characterized by many local minima and one global one. We can see that the two algorithms lead to different minima. Here, EnOpt-ML fell into a lower minimum (3.16, table 33) than EnOpt-ML (7.23, table 33). The objective function evaluations are set to result in 160 evaluations.

Since there are many local minima, neither EnOpt nor EnOpt-ML approached the global minima (fig. 101).

Table 33 Optimization results for Bukin function 6 (single simulation)

	updated (near-optimal or optimal) control vector at the last iteration.	Number of iterations	Number of objective function evaluations	Optimized objective function
EnOpt	[-11.495603 1.326695]	4	160	7.23008
EnOpt-ML	[-11.928068 1.4218]	6	160	3.162951

- <u>1000 simulations</u>:

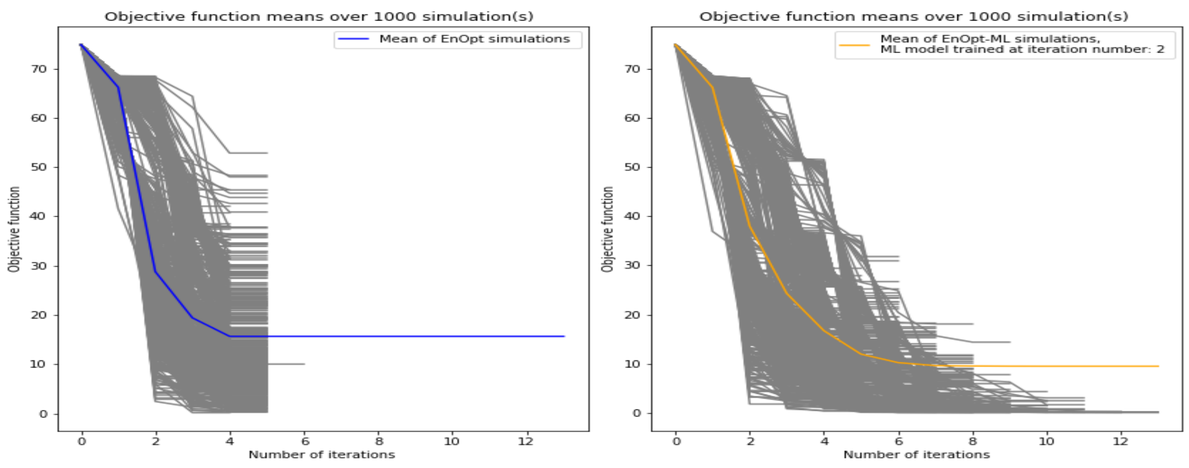


Figure 103 Bukin function 6 objective function optimization results (1000 simulations)

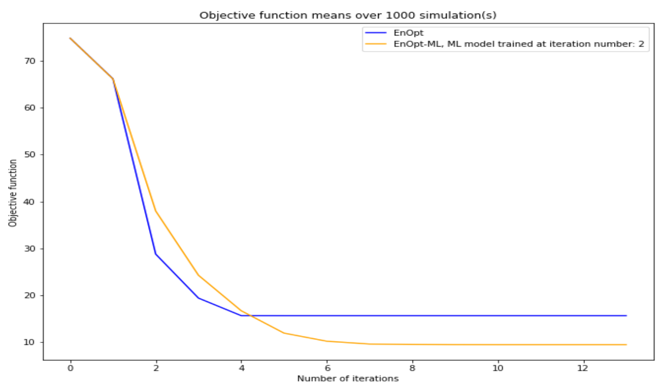


Figure 104 Bukin function 6 EnOpt and EnOpt-ML means of simulations

The simulations show a slight improvement on the results with the use of EnOpt-ML compared to the use of EnOpt both in terms of means and standard deviation of the ensemble of simulations(fig. 103 and 104). The minimum number of iterations (lower than 160 in average for both EnOpt and EnOpt ML, table 34) suggest that the reduction in N could be applied even sooner with a possibility to reduce the number of objective function evaluations and still be able to get a ‘good’ enough optimization results. If the objective function is expensive to calculate, it may be more difficult to choose the iteration at which we can stop or reduce the N number of samples to still be able to get a good optimization result.

Figure 105 shows that the median of simulations from EnOpt method (15, table 34)results is higher than the 75% quartile of the simulations with EnOpt-ML method (13.4, table 34).

In figure 106 we can see that there were some simulations with EnOpt and EnOpt-ML that were getting closer to the global minimum. The figure also shows that simulations led to different local minima as they are very close and on the same ridge.

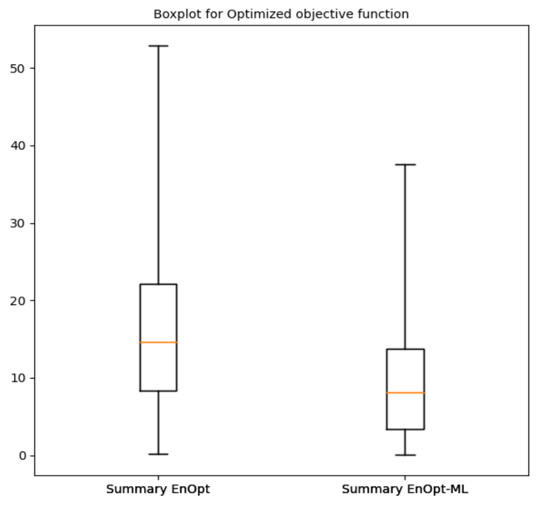


Figure 105 Bukin function 6 Boxplot of optimized objective function

Table 34 Bukin function 6 : summary output values (1000 simulations)

EnOpt	iterations	Number of objective function evaluations	Optimized objective function	EnOpt-ML	iterations	Number of objective function evaluations	Optimized objective function
mean	3.69169169	147.6676677	16.00829415	**mean**	5.640640641	156.7567568	9.261639346
std	0.4641896	18.56758401	9.557876696	**std**	1.756177185	10.92385248	7.087733572
min	2	80	0.388513	**min**	3	120	0.035795
25%	3	120	8.722706	**25%**	4	160	3.4182745
50%	4	160	15.008802	**50%**	5	160	8.051084
75%	4	160	21.6203955	**75%**	7	160	13.403765
max	4	160	67.502876	**max**	13	160	35.133651

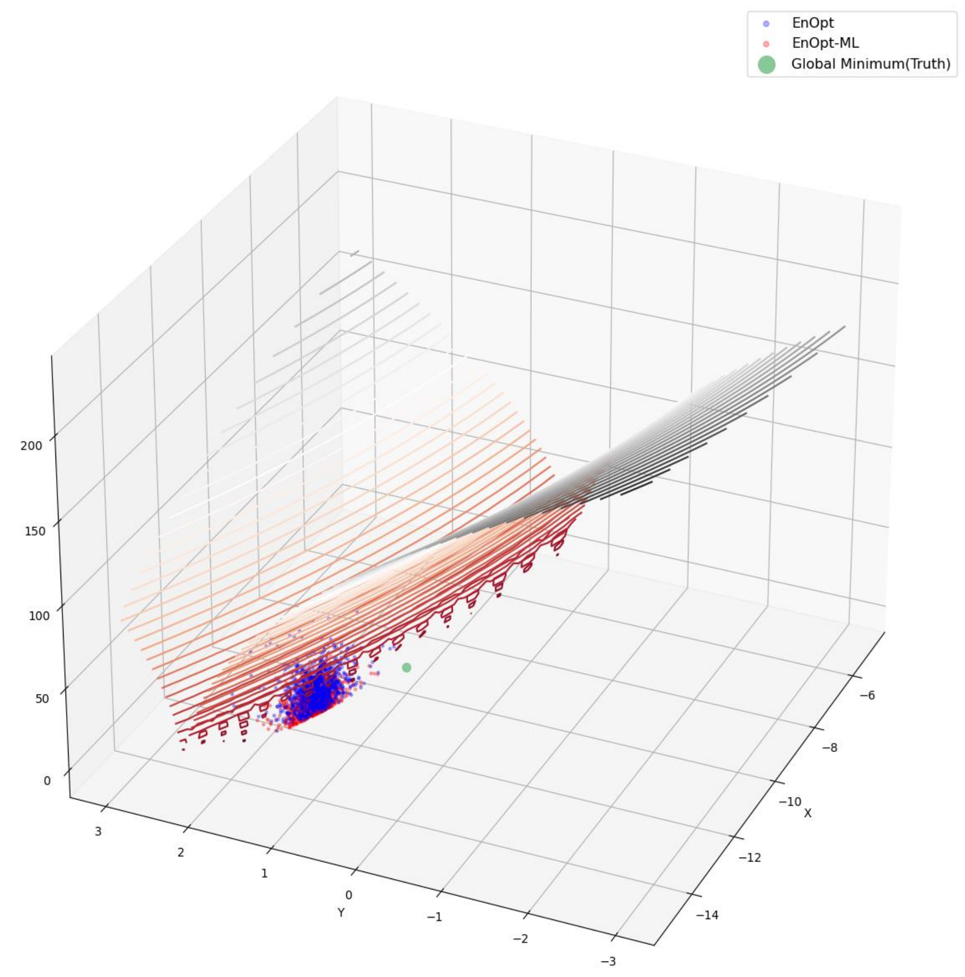


Figure 106 Bukin function 6 : simulations' results of EnOpt and EnOpt-ML with respect to true minimum values

On Rosenbrock

The configuration here has been deviated from the Himmelblau in the previous subchapter configuration by:

- test function : Rosenbrock,
- starting point [8, -2].

- Single simulation

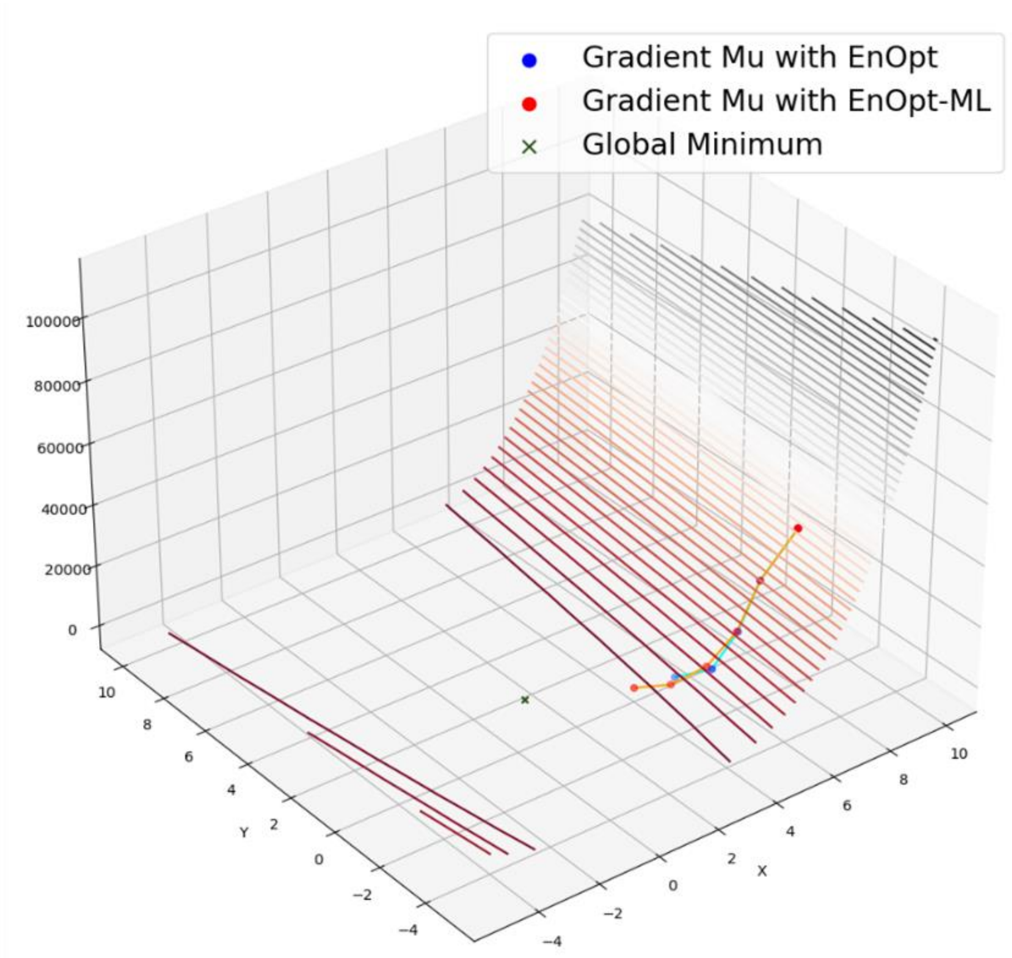


Figure 107 Rosenbrock: gradient (single simulation)

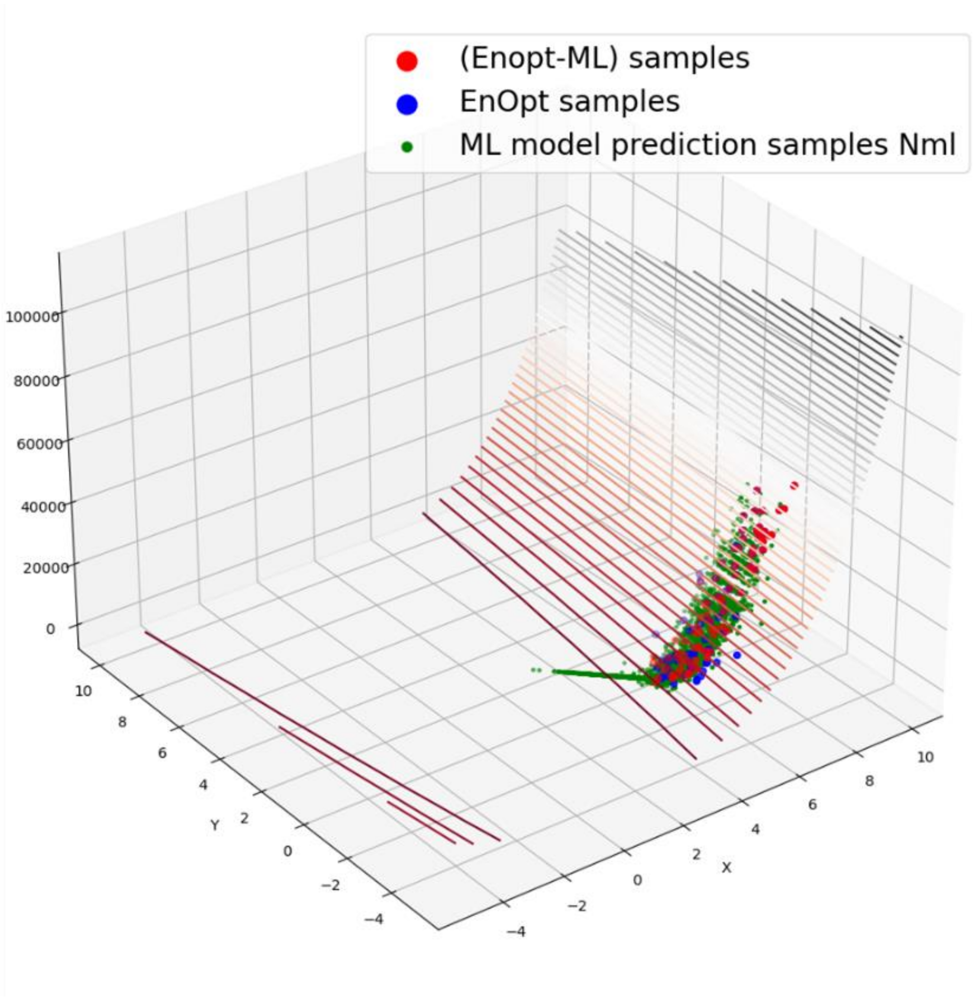


Figure 108 Rosenbrock: EnOpt and EnOpt-ML samples and ML model predicted samples

MSE: 5197560.11

The MSE is high for the higher scale (compared to the other two test functions) and the different function shape. The Rosenbrock function has a very high maxima in the chosen interval of evaluation. The chosen starting point is at almost 45000. At this scale (global minimum being 0), a 1714.56 (in EnOpt-ML, table 35)is not a bad optimization result for this specific simulation in comparison with the other test functions used in this thesis. EnOpt on the other hand did poorly (4520, table 35) which means that using the ML model with the gradient helped getting better optimization value.

Table 35 Optimization results for Rosenbrock (single simulation)

	updated (near-optimal or optimal) control vector at the last iteration.	Number of iterations	Number of objective function evaluations	Optimized objective function
EnOpt	[4.479895 -1.164367]	4	160	4520.861785
EnOpt-ML	[3.534575 -0.576345]	6	160	1714.559687

- <u>1000 simulations</u>:

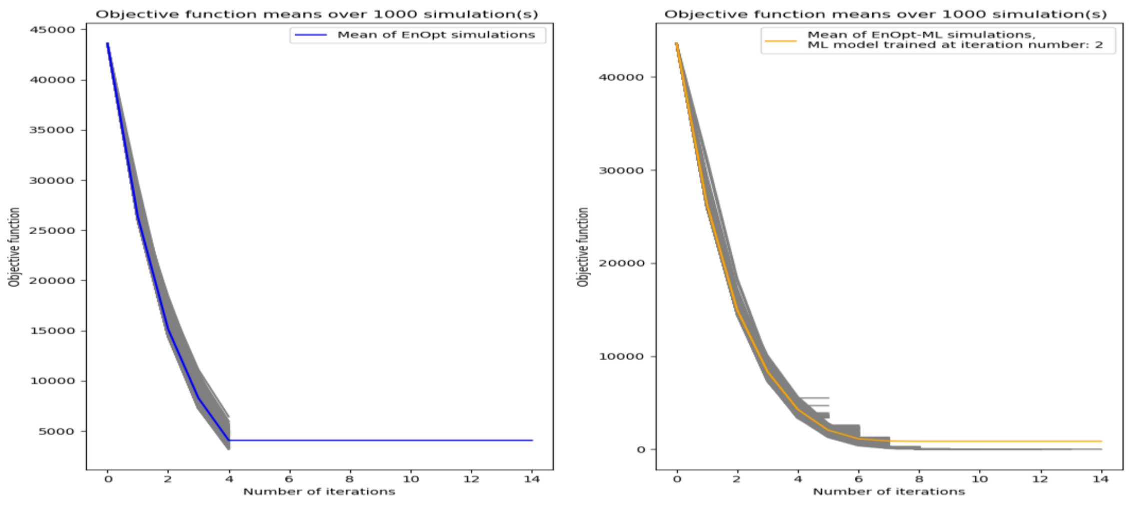


Figure 109 Rosenbrock: objective function optimization results (1000 simulations)

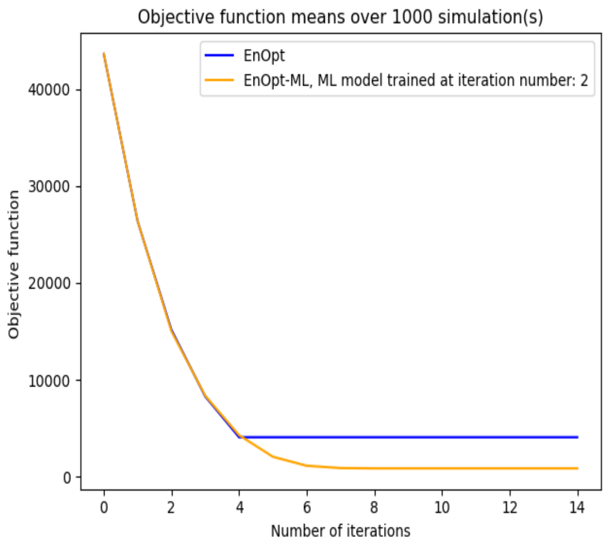


Figure 110 Rosenbrock: : EnOpt and EnOpt-ML means of simulations

EnOpt-ML performed better than EnOpt (it is still only slightly better scale-wise) knowing that there are no more samplings on the true objective function after the 4th iteration for both methods.

Figure 111 shows how much all simulations with EnOpt-ML yielded better optimized values of the objective function for all quartiles compared to EnOpt simulations with medians of 683.02 for EnOpt-ML compared to 3923.13 for EnOpt (table 36).

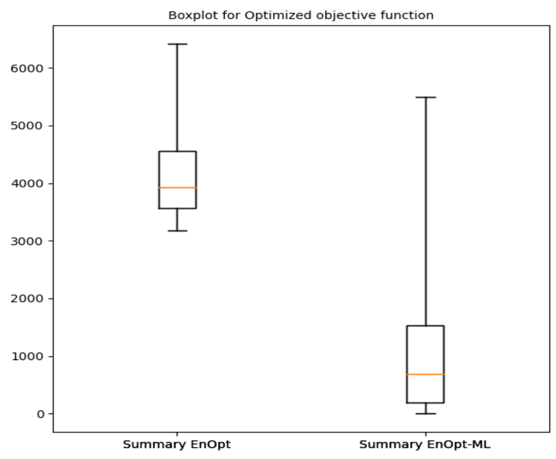


Figure 111 Rosenbrock: Boxplot of optimized objective function

Table 36 Rosenbrock: summary output values (1000 simulations)

EnOpt	iterations	Number of objective function evaluations	Optimized objective function	EnOpt-ML	iterations	Number of objective function evaluations	Optimized objective function
mean	4	160	4066.763985	mean	7.437437437	160	849.815354
std	0	0	595.5870513	std	1.447767904	0	839.0688255
min	4	160	3175.408546	min	5	160	0.000226
25%	4	160	3569.132415	25%	6	160	185.156716
50%	4	160	3923.454653	50%	7	160	683.025654
75%	4	160	4561.259274	75%	8	160	1527.768429
max	4	160	6414.039478	max	14	160	5500.400997

When plotting the simulations' results for both EnOpt and EnOpt-ML (fig. 112), we can see that few EnOpt-ML simulations results approached pretty much the global minimum while all EnOpt simulations optimizations stayed behind all EnOpt-ML simulations' optimized values

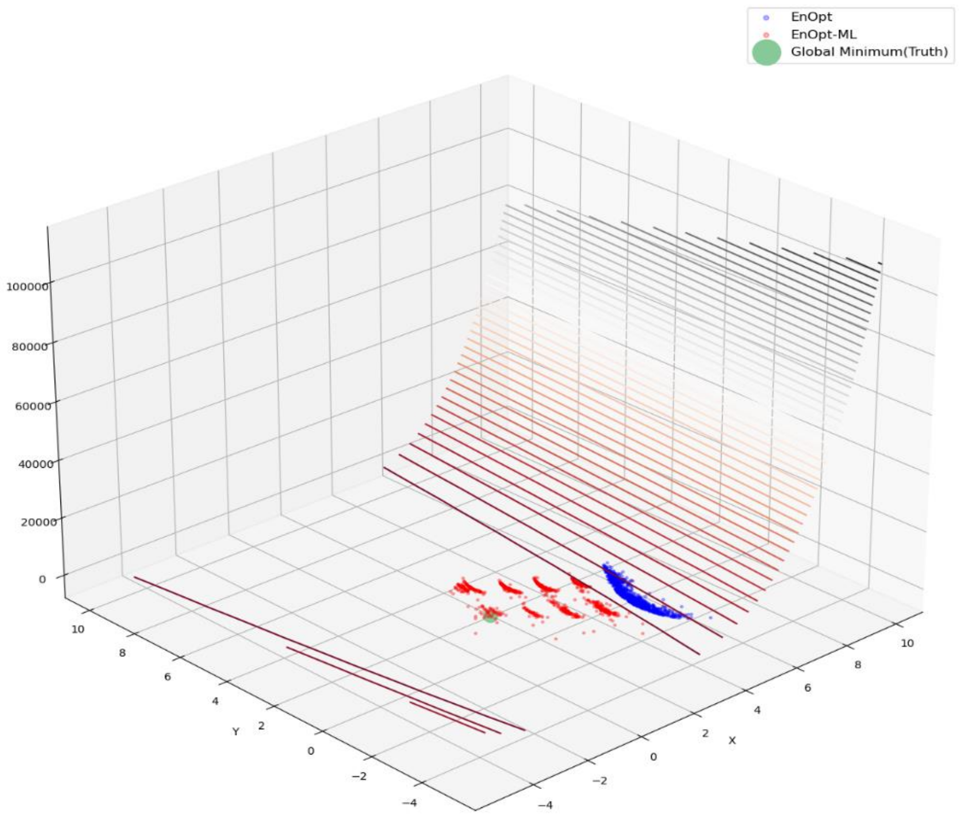


Figure 112 Rosenbrock: simulations' results of EnOpt and EnOpt-ML with respect to true minimum values

Chapter 5: Final remarks and Conclusions

The objective of this thesis work is to investigate whether using ML to include all information from previous iterations for gradient estimation in StoSAG can improve the computational efficiency for optimization. Throughout the thesis, the EnOpt-ML approach, where a ML model is trained using control vectors and objective values from all previous StoSAG iterations and used for gradient estimation, has been developed and applied for optimization. The optimization results of EnOpt-ML and that of EnOpt are compared and discussed, under various configurations and on challenging objective functions.

Observations/ remarks:

1. For Himmelblau test function:
   - The starting point of the optimization effects both EnOpt and EnOpt-ML although they may land on different minima.
   - Increasing the sample size improves both EnOpt and EnOpt-ML, the latter requires fewer evaluations on the objective function and has good enough results.
   - Increasing the Nml (number of samples created through the predictions of the machine learning algorithms used) does not yield significant improvement on the optimization results.
   - Increase in the initial covariance matrix gave slightly better means (smaller means for minimization) and smaller standard deviations on the side of EnOpt-ML compared to EnOpt.
   - Training ML model at later iterations gives even better results than if it is trained from the start.
   - We can reduce the number of samples N at certain iteration level for both EnOpt and EnOpt-ML and that reduces the objective function evaluations for both and does result in fairly good optimization results for EnOpt-ML compared to EnOpt. The later we reduce the sample size N, the better the optimization results are, but the specified value of N gives us an upper hand over the maximum number of function evaluation we are willing to run. Note: reducing N to 0 is not equivalent to stopping the iterations at the given step for EnOpt-ML (only in EnOpt) as the iterations continue with the Nml predicted samples.

- EnOpt-ML requires fewer objective function evaluations than EnOpt, in average. Especially with higher covariance matrix values.
- Among the three machine learning algorithms used in this thesis, Gradient Boosting regressor and Ada Boosting regressor achieve significantly better optimization results than Random Forest regressor.

2. For Bukin function number 6: (a ridge function)
   - EnOpt-ML achieved faintly better optimization results than EnOpt in the 1000 simulations as average.
   - This function has many local minima which lie along a ridge, and EnOpt and EnOpt-ML did not land at the same minimum.
3. For Rosenbrock test function: (valley function)
   - The function has a global minimum lying in a narrow parabolic valley. it descends very slowly on its way to it compared to at the side of the function.
   - EnOpt-ML achieved better results than EnOpt.
   - None of the two optimization results were very satisfying, but this can also be attributed to the fact that the function has a much higher values in the chosen evaluation interval in comparison with the two other test functions in this thesis.

Conclusion:

As the objective, this thesis work has developed, implemented, investigated, and discussed an approach – EnOpt-ML – of using ML in the stochastic gradient descent for improving the computational efficiency of gradient estimation and optimization. For some specific cases, EnOpt-ML does improve the computational efficiency in terms of fewer objective function evaluations and achievement of a near-optimum, compared to EnOpt. However, the experiment results do not provide clear evidence on that EnOpt-ML overperforms EnOpt in general. The use of fewer samples from EnOpt means feeding fewer data to the training of machine learning models. Thus, although the reduction in the number of samples reduces the number of objective function evaluations, it also negatively impacts the training of ML in EnOpt-ML, as the more training data, the more accurate a trained ML model will be. The increase in sample size N has led, on average (in the examples studied), to fewer objective function evaluations with EnOpt-ML than objective function evaluations with EnOpt.

We can control the maximum number of objective function evaluations we want to perform (and then the objective value will be predicted using the trained ML model) to reduce the computational intensiveness. However, we should keep in mind that the earlier we reduce the sample size or stop sampling, the less “optimal” the results might be.

In conclusion, this thesis work has set up the stage for utilizing ML in SGD for speeding up optimization. However, more analysis and further research should be conducted to investigate the effect of ML and better utilize it. Recommendations for future works are, for example, testing more different configurations of EnOpt-ML, using other ML algorithms, modifying the way ML is used, test on more computational demanding objective functions (e.g., reservoir simulation models), and include uncertainty (i.e., robust optimization).

Appendix

EnOpt and EnOpt-ML codes

```
%matplotlib inline
import numpy                    as     np
import matplotlib.pyplot        as     plt
import scipy.stats              as     ss
from   numpy.random             import multivariate_normal as multinormal
from   numpy.linalg             import eig
import statistics
from   numpy                    import random
from   pandas                   import DataFrame
import timeit
from   mpl_toolkits             import mplot3d
from   mpl_toolkits.mplot3d     import Axes3D
import math
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error
```

Objective Functions for test:

```
def of_HimmelBlau(M1, M2):
    truth        = [3, 2]  # (or an other depending on starting mu)
    x            =  M1
    y            =  M2#M[1]
    return (x**2 + y - 11)**2 + (x + y**2 - 7)**2

def of_Rosenbrock(M1, M2):
    x            =  M1
    y            =  M2#M[1]
    b            = 10 #choice  b=10 .
    return (x-1)**2 + b*(y-x**2)**2

def of_BukinFunc6(M1, M2): #Bukin function N.6
    x            =  M1
    y            =  M2#M[1]
    return 100*(abs(y-0.01*x**2)**(1/2))+ 0.01*abs(x+10)
```

Machine Learning algorithms

```
def ML_GradBoostReg(Xs, Ys):
    param = {'n_estimators': 100,'max_depth': 1, 'learning_rate': 0.1, 'criterion': 'mse'}
    model = GradientBoostingRegressor(**param)
    model.fit(Xs, Ys)
    return model

def ML_RandForest(Xs, Ys):
    param = {'n_estimators': 100,'max_depth': 1}
    model = RandomForestRegressor(**param)
    model.fit(Xs, Ys) #reshape(-1, 1)
    return model

def ML_AdaBoostReg(Xs, Ys):
    param = {'n_estimators': 100,'random_state':0}
    model = AdaBoostRegressor(**param)
    model.fit(Xs, Ys) #.reshape(-1, 1)
    return model
```

Function for testing (EnOpt and EnOpt-ML)

```
def run(mu, C, init_N=20, reduced_N=5, Nml=100, of=None, max_iter=100,
        train_iter=2, reduce_at_iter=100, ML=None, seed=None, show_plot=False,
        show_results= False, plot_func=None, minx=-6, maxx=6, miny=-6, maxy=6):

    current_iter = 0
    converged    = 0
    model        = None

    tol                 =  10**(-3)                 # Tolerance
    stepMu              =  1                        # Stepsize for the mean  (mu)
    stepC               =  0.1                      # Stepsize for the covariance matrix
(C)
    Fold                =  of(mu[0], mu[1])         # Objective function at initial point
mu

    if seed != None:
        np.random.seed(seed)                        # seed: which seed or random seed?

    X1s      = np.array([])
    X2s      = np.array([])
    Fs       = np.array([])
    X1sML     = np.array([])
    X2sML     = np.array([])
    XsML      = np.array([])
    FsML      = np.array([])

    of_runs   = 0

    histMu    = []

    # add initial mu
    histMu.append(mu.copy())
```

```
    while converged     == 0:
          force_stop     = 1     # force stop gradient calculation
                              #when N=0 in non-MLsimulation
        # Run objective function

        if current_iter < reduce_at_iter:
            N = init_N
        else:
            if reduced_N > 0:
                N = reduced_N

        if current_iter < reduce_at_iter or reduced_N > 0:
           force_stop  = 0
            of_runs    += N
            X = multinormal(mu, C, N)                         # N: Number of control samples
            X1s         = np.concatenate((X1s, X[:,0].copy())) # history X1
            X2s         = np.concatenate((X2s, X[:,1].copy())) # history X2
            Xs          = np.array([X1s, X2s])
            F           = list(map(lambda x:of(x[0], x[1]), X))
            Fs          = np.concatenate((Fs, F))              # history F

        if current_iter >= train_iter:
            force_stop  = 0
            N           = Nml

            # Use data for prediction
            if current_iter < reduce_at_iter or reduced_N > 0:
                model       = ML(Xs.T, Fs)                     # Training model

            X               = multinormal(mu, C, Nml)
                            # Nml: Number of control samples added for ML
            F               = model.predict(X)
            # separate variables for ML history
            X1sML           = np.concatenate((X1sML, X[:,0].copy())) # history X1_ML
            X2sML           = np.concatenate((X2sML, X[:,1].copy())) # history X2_ML
            XsML            = np.array([X1sML, X2sML])
            FsML            = np.concatenate((FsML, F))

    If  force_stop  == 0
# Gradient calculation
        gradMu           = np.mean([[i - Fold for i in F][i] * (X-mu)[i] for i in
range(N)], 0)


        gradMu          /= np.linalg.norm(gradMu)               #normalize gradient
        gradC            = np.zeros((2,2))
```

```
        for k in np.arange(0, N):
            # updating covariance gradient

            gradC       += (F[k] - Fold) * (np.matmul((np.asmatrix(X[k] - mu)),
(np.asmatrix(X[k] - mu)).T) - C) / N

        gradC           /= np.linalg.norm(gradC)               # normalize gradient
        muNew            = mu - stepMu * gradMu                # updating the mean ((-)
since we are minimizing)
        Cnew             = C  - stepC  * gradC                 # updating the covariance

    # Must have positive definite covariance C:
        E                = np.linalg.eigvals(Cnew)             # eigenvalues of the new
covariance matrix

        if (min(E))     <= 0:
            Cnew         = Cnew + np.dot(np.abs(min(E))+ 10**(-6), (np.eye(2)))

    # Checking if the F new point has reduced the objective:
        Fnew             = of(muNew[0], muNew[1])
        delta            = Fold - Fnew

    # Gradient descent
        while delta      < 0:
            stepMu       = 0.5 * stepMu
            muNew         = mu - stepMu * gradMu
            Fnew          = of(muNew[0],muNew[1])
            delta         = Fold - Fnew

        # New values:
        mu                = muNew
        C                 = Cnew
        Fold              = Fnew

        histMu.append(mu.copy())

        # End of iteration
        current_iter    += 1
        if current_iter >= max_iter or delta<tol or force_stop == 1:
            converged     = 1

            #diff_Truth = of(mu[0], mu[1])-of(truth[0], truth[1])
               # the above would be used if the true minimum is different than 0



        # Prepare data for plotting
            histFlatMu      = np.array(histMu)
            histObjectiveMu = of(np.array(histFlatMu)[:,0], np.array(histFlatMu)[:,1])
            x               = np.linspace(minx, maxx, 200)
            y               = np.linspace(miny, maxy, 200)
            meshX, meshY    = np.meshgrid(x, y)
            Z               = of(meshX, meshY)

        # Plot
            if show_plot:
                fig             = plt.figure(figsize  =(12, 12))
                ax              = plt.axes(projection ='3d')
                ax.contour3D(meshX, meshY, Z, 50, cmap='RdGy')
                ax.plot(histFlatMu[:,0], histFlatMu[:,1], of(histFlatMu.T[0],
histFlatMu.T[1]),   color ='purple')
                ax.scatter(np.array(histFlatMu)[:,0], np.array(histFlatMu)[:,1],
histObjectiveMu, color ='red')
                plt.show()

        # Show results
            if show_results:
                print('Finished at itereration number: \n', current_iter)
                print("The mean at last point of iteration: \n", np.round(mu, 6))
                print("Objective function at the mean at last point of iteration: \n",
round(of(mu[0], mu[1]), 6) , '\n' )

                #print('The difference to the true value is: \n', round(diff_Truth, 6))
                # this would only be used if the true minimum is not 0 so to compare

       # Initialize learning step
        stepMu = 1

    return {
        "train_iter": train_iter,
        "iterations": current_iter,
        "init_N": init_N,
        "reduced_N": reduced_N,
        "reduce_at_iter": reduce_at_iter,
        "Nml": Nml,
        "max_iter": max_iter,
        "The mean at last iteration": np.round(mu, 6),
        "Number of times the objective function run": of_runs,
        "Optimized objective function": round(of(mu[0], mu[1]), 6),
        "X": Xs,
        "F": Fs,
        "Mu": histFlatMu,
        "XsML": XsML,
        "FsML": FsML
            }
```

Testing with different configurations On test functions

```
%matplotlib qt
import time

config_mu            = [ 0, 0]          # Different starting means:
                                        # [ 0, 0], [-5,2], [ 3, -1]   for himmelblau,
                                        # [-6,-2], [-12,2]            for bukin fct 6
                                        # [-6,-2], [8,-2]             for rosenbrok
config_C             = 0.1 * np.eye(2)  # initial covariance matrix
config_sims          = 1000             # number of simulations  [1, 1000]
config_init_N        = [20]             # number of samples (or samples before
                                          reduction   if it applies)
config_reduced_N     = [0]              # reduced sample size used with
'config_reduce_at_iter'
config_of            = [of_HimmelBlau]  # [of_HimmelBlau, of_Rosenbrock ,of_BukinFunc6]
config_max_iter      = [100]            # 100 as stopping criteria
config_train_iter    = [999, 1]         # the iterations ML starts training its model on.
```

```
                                                            999 when ML algorithm is not used
config_reduce_at_iter = [999]            # the iteration at which 'config_reduced_N'
                                                            is used. 999 when 'config_reduced_N' not used
config_ML             = [ML_GradBoostReg] # [ML_GradBoostReg, ML_RandForest,
ML_AdaBoostReg]
config_seed           = None              # None for random; 1000 for base
                                                           configuration case
stats = []

for init_N in config_init_N:
    for reduced_N in config_reduced_N:
        for of in config_of:              # of = objective funct.
                                                              = optimization test function
            for max_iter in config_max_iter:
                for train_iter in config_train_iter:
                    for reduce_at_iter in config_reduce_at_iter:
                        for ML in config_ML:
                            for s in range(0, config_sims):
                            # for simulating with different random seeds
                                start_time              = time.time()
                                run_stats               = run(
                                    mu                  = config_mu,
                                    C                   = config_C,
                                    init_N              = init_N,
                                    reduced_N           = reduced_N,
                                    of                  = of,
                                    max_iter            = max_iter,
                                    train_iter          = train_iter,
                                    reduce_at_iter      = reduce_at_iter,
                                    ML                  = ML,
                                    seed                = config_seed,
                                    Nml                 = 500,
                                    #minx               =-10,  #-15, # change when changing
test function
                                    #maxx                = 50,  #-5, # change when changing
test function
                                    #miny                =-10,  #-3,  # change when changing
test function
                                    #maxy                = 50,  # 3,  # change when changing
test function
                                    show_plot           = False,
                                    show_results        = False)

                                run_time                = time.time() - start_time
                                run_stats["run_time"] = run_time
                                run_stats["Sim"]      = s
                                stats.append(run_stats)
```

COMPUTE F MEANS AND PREPARE DATA FOR PLOTTING

```
# set colors in desired order (for simulation means to show, basically)
colors = ['blue',  'orange','red', 'green' ,'yellow', 'cyan', 'purple']

# fill missing values due to different end iterations to be able to compute the means
def fill_list_with_last_element(x, n):
    temp = x.tolist()
    while len(temp) < n:
        temp.append(temp[-1])
    return np.array(temp)

# find longest running simulation (max iterations)
iters = []
for s in stats:
    iters.append(s['iterations'])
f_max_len = np.max(iters) + 1

f_sims = []
f_sims_filled = []
f_means = []

# compute means over simulations
for i in range(len(config_train_iter)): # [999, 1, 2, 3, 4]
    #
    f_sims.append([])
    f_sims_filled.append([])
```

```
    for j in range(config_sims):
        k = config_sims*i+j                                 # find sim index
        f = of(stats[k]['Mu'][:,0], stats[k]['Mu'][:,1])     # calculate Obj.Funct.from MUs
        f_filled = fill_list_with_last_element(f, f_max_len) # fill list
        f_sims[i].append(f)
        f_sims_filled[i].append(f_filled)
    f_means.append(np.mean(f_sims_filled[i], axis=0))        # append means
```

PLOT simulation means

```python
fig, ax = plt.subplots()

for i in range(len(f_means)):
    Y = f_means[i]
    X = range(0, len(Y))

    if config_train_iter[i] == 999:
        plot_label = f"EnOpt "
    else:
        plot_label = f"EnOpt-ML, ML model trained at iteration number: 
{config_train_iter[i]+1}"

    plt.plot(X, Y, label=plot_label,color=colors[i])

plt.xlabel('Number of iterations')
plt.ylabel('Objective function ')
plt.title(f"Objective function means over {config_sims} simulation(s)")
plt.legend()
plt.show()
```

MEANS of different simulations *(SEPARATE PLOTS for EnOpt and EnOpt-ML)*

```python
fig, ax = plt.subplots(1, 2)
axs = ax.ravel()

for i in range(len(f_means)):
    Y = f_means[i]
    X = range(0, len(Y))

    if config_train_iter[i] == 999:
        plot_label = f"Mean of EnOpt simulations "
    else:
        plot_label = f"Mean of EnOpt-ML simulations, \nML model trained at iteration 
number: {config_train_iter[i]+1} "


    for j in range(len(f_sims[i])):
        axs[i].plot(range(0, len(f_sims[i][j])), f_sims[i][j], color="gray")
    axs[i].plot(X, Y, label=plot_label, color=colors[i])
    axs[i].set_xlabel('Number of iterations')
    axs[i].set_ylabel('Objective function')
    axs[i].set_title(f"Objective function means over {config_sims} simulation(s) ")
    axs[i].legend()
```

```
plt.show()
```

min and max values for plots

```
#
# RUNS ONLY WITH single simulation of both EnOpt and EnOpt-ML
if len(stats) == 2:

    # min and max values
    X1_s = [*stats[0]['X'][0], *stats[1]['XsML'][0]] # concat X1s from all simulations
    X2_s = [*stats[0]['X'][1], *stats[1]['XsML'][1]] # concat X2s from all simulations

    X1min = np.min(X1_s) * 1.1
    X2min = np.min(X2_s) * 1.1
    X1max = np.max(X1_s) * 1.1
    X2max = np.max(X2_s) * 1.1
```

Plotting Gradient

```
# RUNS ONLY WITH single simulation of both EnOpt and EnOpt-ML

if len(stats) == 2:
    #Show Mu's
    stats0 = stats[0]
    stats1 = stats[1]

    x               = np.linspace(X1min, X1max, 50)
    y               = np.linspace(X2min, X2max, 50)
    meshX, meshY    = np.meshgrid(x, y)
    Z               = of(meshX, meshY)

    #
    fig             = plt.figure(figsize  =(20, 20))
    ax              = plt.axes(projection ='3d')
    ax.contour3D(meshX, meshY, Z, 50, cmap='RdGy')

    #Gradient with Enopt
    EO_Grad_curve   = ax.plot(stats0['Mu'][:,0], stats0['Mu'][:,1],
                              of(stats0['Mu'].T[0], stats0['Mu'].T[1]), color ='cyan')
    EO_Grad_Mu      = ax.scatter(np.array(stats0['Mu'])[:,0], np.array(stats0['Mu'])[:,1],
                              of(np.array(stats0['Mu'])[:,0], np.array(stats0['Mu'])[:,1]),
color ='blue')

    #Gradient with EnOpt+ML
    EO_ML_Grad_curve= ax.plot(stats1['Mu'][:,0], stats1['Mu'][:,1],
                             of(stats1['Mu'].T[0], stats1['Mu'].T[1]), color ='orange')
    EO_ML_Grad_Mu   = ax.scatter(np.array(stats1['Mu'])[:,0], np.array(stats1['Mu'])[:,1],
                             of(np.array(stats1['Mu'])[:,0], np.array(stats1['Mu'])[:,1]),
color ='red')

# Plotting the minima for reference:
    minima1  = ax.scatter(3, 2, of(3, 2), color = '#274e13',marker='x') # minimum 1
                                                                      for Himmelblau
    #minima2  = ax.scatter(-2.805118, 3.283186, of(-2.805118, 3.283186), color =
'#274e13',marker='x')# minimum 2 for Himmelblau
    #minima3  = ax.scatter(-3.779310,-3.283186, of(-3.779310,-3.283186), color =
'#274e13',marker='x')# minimum 3 for Himmelblau
    #minima4  = ax.scatter(3.584458,-1.848126, of( 3.584458,-1.848126), color  =
'#274e13',marker='x')# minimum 4 for Himmelblau
    #minima5  = ax.scatter( 1       , 1       , of( 1       , 1       ), color =
'#274e13',marker='x')# global minimum of Rosenbrok
    #minima6  = ax.scatter(-10      , 1        , of(-10      , 1      ), color =
'#274e13',marker='x')# global minimum of Bukin function N#6


    plt.legend(( EO_Grad_Mu,  EO_ML_Grad_Mu,  minima1),      # minima 1,2,3,4,5 or 6
                                                           depending on function
              ( 'Gradient Mu with EnOpt', 'Gradient Mu with EnOpt-ML', 'Minima'),
               markerscale=2, fontsize=20)
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.show()
```

Plotting samples

```
# RUNS ONLY WITH single simulation of both EnOpt and EnOpt-ML
if len(stats) == 2:

    # X's and F's
    fig             = plt.figure(figsize  =(15, 15))
    ax              = plt.axes(projection ='3d')
    ax.contour3D(meshX, meshY, Z, 50, cmap='RdGy')

    # EnOpt Samples
    EO_samples      = ax.scatter(stats0['X'][0], stats0['X'][1],
                                 stats0['F'], color ='blue',marker='o')
    # EnOpt-ML Samples
    EO_ML_samples   = ax.scatter(stats1['X'][0], stats1['X'][1],
                                 stats1['F'], color ='red', marker='o')
    # Samples from ML model prediction
    ML_samples      = ax.scatter(stats1['XsML'][0],
                                 stats1['XsML'][1], stats1['FsML'], color ='green',
marker='.')

    plt.legend((EO_ML_samples, EO_samples, ML_samples) ,
               ('(Enopt-ML) samples', 'EnOpt samples','ML model prediction samples Nml'),
               markerscale=3, fontsize=20)



    plt.xlabel('X')
    plt.ylabel('Y')
    plt.show()
```

MSE for EnOpt-ML : ML predicted samples (Nml)

(N.B: for EnOpt only, no predictions since it is mapped on the objective function)

```
# RUN ONLY WITH 2 SIMULATONS:
if len(stats) == 2:

    # Given values
    Y_true2 = of(stats1['XsML'][0], stats1['XsML'][1]) # objective function on the X and
                                                        Y used in model prediction by ML

    # calculated values
    Y_pred2 = stats1['FsML']                            # ML predicted

    # Calculation of Mean Squared Error (MSE)
    # https://www.geeksforgeeks.org/python-mean-squared-error/
    MSE = mean_squared_error(Y_true2,Y_pred2)

    stats1['MSE'] = MSE                                 # Append MSE to the stats
    print('MSE: ', round(MSE, 2))
```

Create output files

```
from tabulate import tabulate
header     = stats[0].keys()
```

```
rows      =  [x.values() for x in stats]
#print(tabulate(rows, header, tablefmt="tsv"))

content   = tabulate(rows, header, tablefmt="csv")
text_file = open("output.csv","w")
text_file.write(content)
text_file.close()
```

Creating excel file with summary table *(useful when doing multiple simulations with numerous simulations)*

```
import pandas as pd
import subprocess
from datetime import datetime

if len(stats) == 2:

    df                 = pd.DataFrame(data=stats)

    now                = datetime.now()
    current_time       = now.strftime("%y_%m_%d_%H_%M_%S")

    file_name          = f'stats_{current_time}.xlsx'

    df.to_excel(file_name)

    df = df.drop(['train_iter', 'init_N', 'Nml', 'max_iter'], axis=1)
                      #  dropping unnecessary summary columns

else:
    now                = datetime.now()
    current_time       = now.strftime("%y_%m_%d_%H_%M_%S")
    # all stats
    file_name = f'stats_all.xlsx'

    df                  = pd.DataFrame(data=stats)

    df.to_excel(file_name)

    # df0 for EnOpt
    df0                 = pd.DataFrame(data=stats[0:config_sims-1])

    now                = datetime.now()
    current_time       = now.strftime("%y_%m_%d_%H_%M_%S")
    file_name         = f'stats_{current_time}.xlsx'
    file_name_summary = f'stats0_{current_time}_summary.xlsx'

    df.to_excel(file_name)

    df0 = df0.drop(['train_iter', 'init_N', 'Nml', 'max_iter', 'reduced_N',
'reduce_at_iter', 'Sim'], axis=1) #  dropping unnecessary summary columns

    df0_summary        = df0.describe()
    df0_summary        = df0_summary.drop(index=('count'))  # drop rows
    df0_summary.to_excel(file_name_summary)

    # df1 for EnOpt-ML
    df1                = pd.DataFrame(data=stats[config_sims: 2*config_sims-1])

    current_time      = now.strftime("%y_%m_%d_%H_%M_%S")

    file_name_summary = f'stats1_{current_time}_summary.xlsx'

    df1.to_excel(file_name)

    df1 = df1.drop(['train_iter', 'init_N', 'Nml', 'max_iter', 'reduced_N',
'reduce_at_iter', 'Sim'], axis=1)

    df1_summary        = df1.describe()
    df1_summary        = df1_summary.drop(index=('count'))
    df1_summary.to_excel(file_name_summary)
```

References:

Anzar. (2021). Forecasting of Daily Demand's Order Using Gradient Boosting Regressor. In Progress in Advanced Computing and Intelligent Engineering (pp. 177–186). Springer Singapore. https://doi.org/10.1007/978-981-33-4299-6_15

Bagalkot, Keprate, & Orderløkken. (2021). *Combining Computational Fluid Dynamics and Gradient Boosting Regressor for Predicting Force Distribution on Horizontal Axis Wind Turbine.* https://doi.org/https://doi.org/10.3390/vibration4010017

Chen, Y., Oliver, D. S., and Zhang, D. (2009). *Efficient ensemble-based closed-loop production optimization*. SPE J. 14, 634–645. doi: 10.2118/112873-PA

Chen, Y., and Oliver, D. S. (2010). *Ensemble-based closed-loop optimization applied to Brugge field*. SPE Reserv. Eval. Eng. 13, 56–71. doi: 10.2118/118926-PA

Chen, Y., and Oliver, D. S. (2012). *Localization of ensemble-based control-setting updates for production optimization.* SPE J. 17, 122–136. doi: 10.2118/125042-PA

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, Édouard Duchesnay (2011). Scikit-learn: Machine Learning in Python. 12(85):2825−2830. Journal of Machine Learning Research (2011). https://doi.org/10.48550/arXiv.1201.0490

Fonseca RM, Kahrobaei S, Van Gastel LJT, Leeuwenburgh O, Jansen J.D (2015). *Quantification of the impact of ensemble size on the quality of an ensemble gradient using principles of hypothesis testing*. Paper 173236-MS presented at the 2015 SPE Reservoir Simulation Symposium, Houston, USA; 22-25 February 2015.

Fonseca, Chen, B., Jansen, J. D., & Reynolds, A. (2016). *A Stochastic Simplex Approximate Gradient (StoSAG) for optimization under uncertainty.* International Journal for Numerical Methods in Engineering, 109(13), 1756–1776. https://doi.org/10.1002/nme.5342

Friedman J. H. (2001) *Greedy function approximation: a gradient boosting machine*. The Annals of Statistics 29 (5), 1189–1232.

J.E. Nwaozo.Dynamic *Optimization of a Water Flood Reservoir.* University of Oklahoma (2006)

Li, Li, W., & Xu, Y. (2018). *Human Age Prediction Based on DNA Methylation Using a Gradient Boosting Regressor*. *Genes*, *9*(9), 424. https://doi.org/10.3390/genes9090424

Picheny, V., Wagner, T., & Ginsbourger, D. (2012). A benchmark of kriging-based infill criteria for noisy optimization.

R.J. Lorentzen, A.M. Berg, G. Naevdal, E.H. Vefring. *A new approach for dynamic optimization of waterflooding problems.* Amsterdam, Netherlands Proceedings of the SPE Intelligent Energy Conference and Exhibition (2006) Apr. 11–13, No. SPE 99690

Svetnik, Liaw, A., Tong, C., Culberson, J. C., Sheridan, R. P., & Feuston, B. P. (2003). *Random Forest: A Classification and Regression Tool for Compound Classification and QSAR Modeling.* Journal of Chemical Information and Computer Sciences, 43(6), 1947–1958. https://doi.org/10.1021/ci034160g

van Essen, G.M., Zandvliet, M.J., Van den Hof, P.M.J., Bosgra, O.H., Jansen, J.D (2009).: *Robust waterflooding optimization of multiple geological scenarios*. SPE J. 14(1), 202–210 (2009)