Costly Black-Box Optimization with GTperform at Siemens Industrial Turbomachinery

Department of Mathematics, Linköping University

André Malm

Credits:	**16 hp**
Level:	**G2**
Supervisors:	**Nils-Hassan Quttineh**, Department of Mathematics, Linköping University
	Christer von Wowern, RP, Siemens Industrial Turbomachinery AB Finspång
Examiner:	**Torbjörn Larsson**, Department of Mathematics, Linköping University
Linköping:	**January 2022**

Abstract

The simulation program GTperform is used to estimate the machine settings from performance measurements for the gas turbine model STG-800 at Siemens Industrial Turbomachinery in Finspång, Sweden. By evaluating different settings within the program, the engineers try to estimate the one that generates the performance measurement. This procedure is done manually at Siemens and is very time-consuming. This project aims to establish an algorithm that automatically establishes the correct machine setting from the performance measurements.

Two algorithms were implemented in Python: Simulated Annealing and Gradient Descent. The algorithms analyzed two possible objective functions, and objective were tested on three gas turbines located at different locations. The first estimated the machine setting that generated the best fit to the performance measurements, while the second established the most likely solution for the machine setting from probability distributions. Multiple simulations have been run for the two algorithms and objective functions to evaluate the performances.

Both algorithms successfully established satisfactory results for the second objective function. The Simulated Annealing, in particular, established solutions with a lower spread compared to Gradient Descent. The algorithms give a possibility to automatically establish the machine settings for the simulation program, reducing the work for the engineers.

Keywords:

Black-Box Optimization, GTperform, STG-800, Simulated Annealing, Gradient Descent

Acknowledgment

Several people have given valuable contributions and support throughout the thesis, and some are acknowledged below. Additionally, I want to give my sincere thanks to all employees at Siemens' performance division that have provided support in any way. It has been a tremendous personal experience.

Firstly, I want to thank my supervisor Christer von Wovern at Siemens, for his advice throughout the project. He helped me throughout the complete process, from technical aspects to implementation difficulties.

Secondly, my supervisor Nils-Hassan Quttineh at Linköping University for his significant support during many tough periods of this thesis. Without his support, I would never have accomplished this thesis.

Thirdly, my examiner Torbjörn Larsson at Linköping University with the guidance and knowledge throughout the project. Both with a theoretical explanation as well as problem solutions.

Fourthly, to my beloved friends: Erik, Tobias, Mikael, Ludwig, Patrik, Emma och Embla for their support both individually and academically throughout my studies.

Lastly, to my family and spouse, Elina. For their never-ending help. Giving me confidence and support throughout the process.

To all mentioned above, and everyone else that has supported the project in any way: thank you!

Contents

Chapter 1

Scope of Thesis

This thesis is written at Linköping University by André Malm with Christer von Wovern at Siemens Industrial Turbomachinery as supervisor and Nils-Hassan Quttineh as supervisor at Linköping University, and with Torbjörn Larsson as the examiner. In Sections 1.1 to 1.6, we give a brief introduction together with the purpose, objective, method, limitation and structure of the thesis.

1.1 Introduction

Siemens Industrial Turbomachinery manufactures industrial gas turbines for electrical power production in Finspång, Sweden. These are utilised in a broad range of areas, such as mining, oil, gas, textile, or household electricity production. They are developed and manufactured for each specific application and location, where each turbine has its technical requirements regarding power efficiency, reliability, and environmental conditions. The wide range of manufacturing possibilities and customisation makes each machinery unique.

A gas turbine is a highly complex engine that consists of three main components: a compressor, a combustor, and a turbine. Between specific operational hours, the gas turbine needs maintenance to sustain its efficiency and technical requirements. The conditions of the turbine's components are then analysed in detail. If a part within the components shows any signs of wear down, it may need to be replaced to preserve the engine. Such a component may otherwise negatively affect the turbine, increasing the wear down of components, resulting in power inefficiency, shutdowns, and damage to the turbine.

During the maintenance, the performance measurements are sent to the performance division at Finspång, Sweden, to be analysed further. By using a simulation program, GTperform, the machine settings are estimated. These describe the characteristics for the components within the engine, such as efficiency rates, fuel volumes and temperature settings. At each maintenance period, the machine settings are estimated by GTperform, and the solutions obtained are then used to adjust the settings for the actual gas turbine, by manually regulate these for the turbine.

By reverse engineering, the engineers at the performance division in Finspång, tries to establish the machine settings in the GTperform that generate the best fit between the simulated and the actual performance measurements. The engineer accomplishes this by first selecting solutions for the machine settings and simulate the performance measurements. The engineer then decides on a new setting by analysing the setting and the simulated performance, hopefully generating a performance result that is closer to it. This trial and error procedure continued until the employee found a satisfying solution. This method is very time-consuming, as only one solution at a time can be evaluated by GTperform.

1.2 Purpose

The primary reason for the project is that the procedure to establish the optimal machine settings for a gas turbine is performed manually by the GTperform. This is both time consuming and tedious for the engineer, which needs to have experience of the simulation program. Further, to make the correct conclusions about the performance measurements, one needs theoretical knowledge of the machine components and how they are connected with each other. The purpose of the thesis is to implement two algorithms in Python that can automatically estimate the optimal machine settings from given the performance data.

1.3 Objective

The objective of this thesis is to implement a program that can automatically estimate the machine settings from the performance measurements, by GTperform. This is performed by implementing an optimization algorithm that searches between settings to establish an optimal one.

1.4 Method

The selected method for the described problem was to develop two different types of optimization algorithms. The first algorithm uses a stochastic approach called Simulated Annealing, where the machine settings are randomly selected. The second method is Gradient Descent method, which uses the gradient information to estiamte the best solution. The algorithms' performance were then evaluated for three different real-life gas turbines.

1.5 Limitations

Firstly, there is a time complexity constraint for the methods. As the algorithms will be used multiple times each day, the methods need to produce results within ten minutes. Secondly, we restrict the number of settings in the methods and only analyze the most significant and relevant. Lastly, the value of the selected settings must be chosen within give bounded intervals.

1.6 Structure of Thesis

The thesis is structured in the following sections:

- Background Information.
  Presents the general operating principles of a gas turbine, with technical explanations and notations of components. We last define the problem that is studied.
- Mathematical Formulation
  Describes the problem with mathematical formulation and notation.
- Theory
  A literature background of earlier attempts for solving the given problem, with additional information about the two selected methods. We last present earlier work about GTperform.
- Implementation
  Explaines in detail the implementations of the algorithms
- Numerical Results
  Present the results by the two algorithms for the three different gas turbines.
- Discussion
  We discuss the results in further detail by analyzing the performance of each algorithm. This is done by examining the setting solution, time complexity, and the search patterns.
- Conclusion
  Presents the outcome from this thesis.
- Future Work
  A summary of possible future extensions of the work in this thesis.

Chapter 2

Background Information

A general introduction to how gas turbines function is given in Sections 2.1 and 2.2. In Section 2.3 we clarify the technical aspects of the three main components. In Section 2.4 we introduce and explain the specific turbine model in further detail. Further, in Section 2.5 the simulation program GTperform is described, explaining the technical notation and output results. Lastly, in Section 2.6 the problem definition is given.

2.1 Introduction

A gas turbine is a mechanical engine that generates power, either in shaft power or kinetic energy. Turbines that produce shaft power are used to drive electrical power generators, which produce electricity used for industrial purposes, such as gas pipelines, oil rigs, or households. In contrast, a gas turbine designed to produce kinetic energy is used in aircraft to create the thrust to propel it. Both types of engines consist of the same three main parts: a compressor, a combustor, and a turbine, as displayed in Figure 2.2.

2.2 The Gas Turbine Cycle

The gas turbine cycle describes what happens to the fuel as it goes through the gas turbine. The Brayton cycle, first formulated in 1874 and shown in Figure 2.1 describes the physical transformation by the pressure and volume of the gas, as it flows through the gas turbine, which is displayed in Figure 2.2.

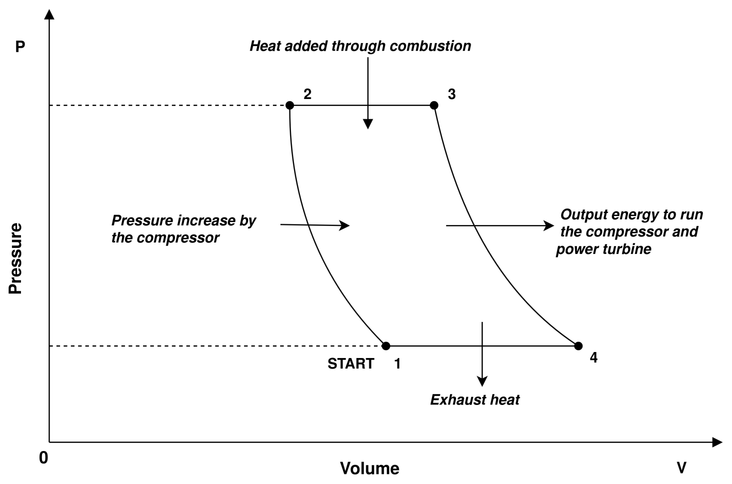

Figure 2.1: The Brayton cycle for gas, with the x-axis displaying the volume and the y-axis showing the pressure.

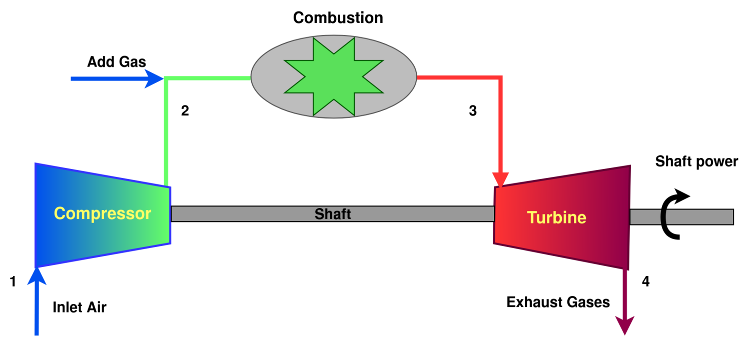

Figure 2.2: The physical state of the gas for the main components of a single shaft gas turbine. The colour scheme displays the temperature transformation as it moves through the components.

In the first step, cold air is drawn into the compressor from the inlet section. From the first to the second steps, the air is compressed by reducing its space. The air then flows from the compressor to the combustor, where a flammable gas mixture is injected under constant pressure, increasing the fuel volume. Lastly, the gas is ignited. The heat transferred from burning the gas increases the internal energy and velocity of the fuel mixture. As the hot pressurized combustion exhaust gases flows through the turbine, it is allowed to expand, which means that its volume increases, as displayed between the third and fourth steps, resulting in a decrease in pressure and temperature. When the exhaust fume changes its physical state, it releases energy to the turbine. This energy is used to power the gas turbine and to generate shaft power.

The difference between a gas turbine and piston engine, such as a car engine, is that in the later the compression och combustion occur after each other at the same location, inside the combustion chamber. For gas turbines, these steps are executed simultaneously in an integrated motion at separate locations. In other words, there is a constant flow of fuel going through the engine and generating a continuous stream of hot exhaust fumes that runs the turbine at a steady rate.

Following the last step, the process is complete. The exhaust gases is released into the atmosphere, which absorbs the excess heat. At this point, the volume of the gas decreases into its initial state. An engine that operates under this procedure is called an open-cycle gas turbine. The compressor extracts unprocessed air at one location and releases the exhausts gases at another point. However, if the exhausted fumes had been reused, the gas turbine operates in a closed-cycle mode, which means that the combustion gases are recycled in an integrated system.

Gas turbines that generate shaft power have additional possibilities to increase the output energy efficiency by combining it with a steam engine. In the last step, the hot exhaust fumes from the turbine go through a second process instead of being released directly to the atmosphere. The gas is here used to boil water into steam by letting it flow through metal cylinders filled with water. The water is vaporized, creating steam that is used to generate additional energy from another generator. Afterwards, the gas is released back into the atmosphere.

2.3 Gas Turbine Components

To get more knowledge about a gas turbine's operational principles, we analyze the compressor, combustion, and turbine in further detail, with specific construction and technical characteristic of each component.

2.3.1 Compressor and Turbine

Gas turbines are divided into two separate groups depending on if they are constructed with a single or a double shaft. A turbine in the former group has a single shaft connecting the turbine and the compressor, as shown in Figure 2.2. The second group of gas turbines are constructed with a twin-spool turbine, as shown in Figure 2.3. For these, the turbine is divided into two sections, where one runs the compressor while the other generates the shaft power. In this configuration, the hot exhaust gases first goes through the larger turbine and then through the smaller power turbine. For this design, the fuel cycle consists of the same principle as in the single shaft engine but with a small modification between the third and fourth steps, as illustrated in Figure 2.2. The turbine is divided it into two separate componetns, as displayed in Figure 2.3. The power turbine between steps $3'$ and 4 generates the shaft power, while the larger turbine drives the compressor between steps 3 and $3'$. By separating the two turbines, the machine becomes more flexible to connect to different types of power generators.

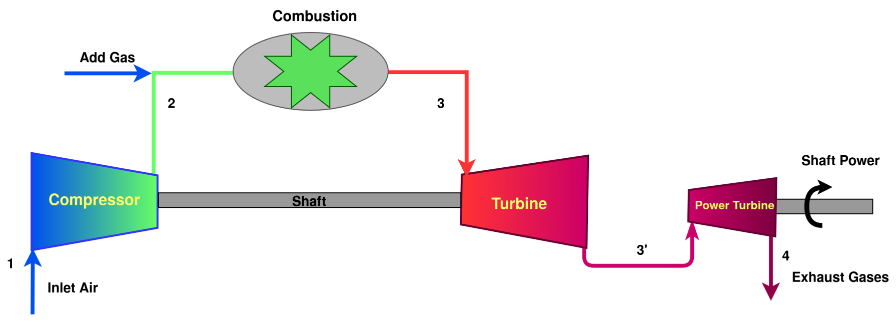

Figure 2.3: The primary components of the twin shaft gas turbine and the physical state of the air from the inled where it is drawn into the engine, to the endpoint where it is exhausted. The colour scheme displays the temperature transformation of the gas as it moves through the components.

The compressor is constructed by metal blades, called airfoils, that are put together into several layers connected to the shaft. As the shaft rotates, the airfoils spin. By arranging the airfoils in decreasing size, starting with the largest, they generate a suction effect that draws gas into the engine and pressurizes the particles by pushing them tighter together. By doing so, the temperature increases, which is displayed by the change of colour from blue to green in Figures 2.1 and 2.3.

The turbine can be viewed as the compressor rotated 180 degrees horizontally, with the airfoils now arranged by increasing size. The volume of the exhaust fumes is now allowed to expand as space within the turbine increases. As the hot exhaust gases flow through the turbine, it pushes the airfoils and creates a spinning motio of the shaft. This motion drives the turbine and the compressor, as it is connected to the same shaft. By maintaining a constant flow of gas through the turbine, the engine keeps on spinning.

To start a gas turbine, one needs to generate enough rotational force to the shaft, connecting the compressor and turbine, such that the combustion can sustain the flow and compression in the engine. That is done by an electric starting engine that is connected to the shaft. This starts the rotational spin of the shaft, and when it has reached a sufficient speed, the starting engine it will stop, and the combustion procedure will start, as the fuel is inserted into the combustor.

2.3.2 Combustor

The combustion section controls the burning of fuel. The gas is required to be combusted in a stable way to generate the optimal energy output. As the gas flows from the compressor it is separated into multiple chambers. Each of these is constructed in the same way and consists of three components: a metal case, a gas injection system and an inner flame tube. The metal case surrounds the injection system and flame tube. As the air flows into one end of the metal case, the injection system infuses a flammable gas mixture to the pressurized air. The fuel mixture is then propelled to one end of the inner flame tube, were burning flames alocated around the tube, ignites the fuel as it flows through the flames. When it does, an combustion occurs of the fuel, which generate hot exhaust gases, that flows through the flame tube and metal case into the turbine.

2.4 SGT-800

We are interested in one particular gas turbine model, the SGT-800 model shown in Figure 2.4. This model is an open-cycled, single shafted gas turbine, as explained in the earlier section and is one of the most manufactured gas turbine models at the factory in Finspång.

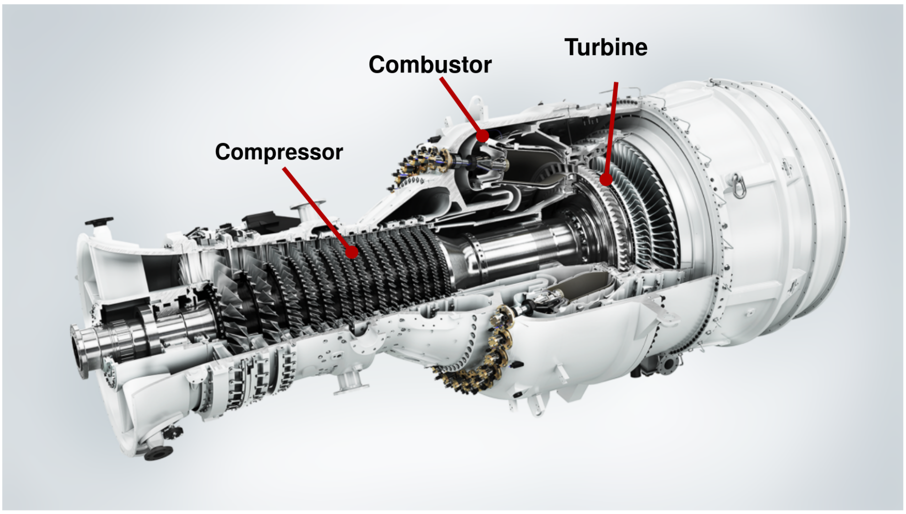

Figure 2.4: The SGT-800 model with its three main components

2.5 The Simulation Program GTperform

GTperform is a simulation program that evaluates the performance parameters, $\boldsymbol{y}$, of a given gas turbine. This is done by solving a complex system of partial differential equations. To establish a gas turbine performance the program evaluates the the machine settings, $\boldsymbol{x}$, for the compressor, combustor, and turbine, together with specific known running parameters, $\boldsymbol{r_c}$, from the cite, as displayed in Figure 2.5. The machine settings have nominal values, but their actual current values are unknown.

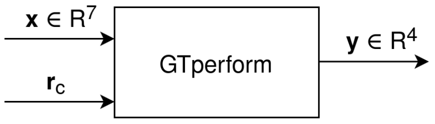

Figure 2.5: Performance parameters from the machine settings and the running conditions.

The running conditions consist of climate conditions and fixed technical parameters for the engine, such as air pressure, temperature, humidity, turbine rotational speed and the gas heating temperature in the combustor, and several more. All these parameters are known to the operator at all times.

The performance parameters are the following.

$$\boldsymbol{y} = \begin{bmatrix} P_{\text{el}} \\ T_3 \\ P_3 \\ T_7 \end{bmatrix} \tag{2.1}$$

The parameter P_{el} describes the output energy that drives the gas turbine, while T_3 and P_3 describe the temperature and pressure at the end of the compressor. Lastly, the T_7 parameter describes the temperature at the end of the turbine. The performance parameters can be measured at all times

The machine settings, which can be modified within GTperform, are the following.

- TIT, the inlet fuel temperature for the turbine.
- C_η, a scale factor for the compressor's efficiency.
- C_m, a scale factor of the capacity of the compressor.
- C_{pd}, the drop in pressure by the combustor.
- T_{fc}, a scale factor of the efficiency for the inlet flow capacity in the turbine.
- T_η, a scaling factor for the efficiency of the turbine.

- I_{1520}, the air temperature before it goes into the turbine.

The machine settings are assumed to have normal probability distributions with known means and standard deviations. The statistics are established for a general turbine and are given by our supervisor at Siemens. Further, we are also given the standard deviations for the measurments of the performance parameters.

Figure 2.6 shows the locations of the performance parameters and the locations for the machine settings.

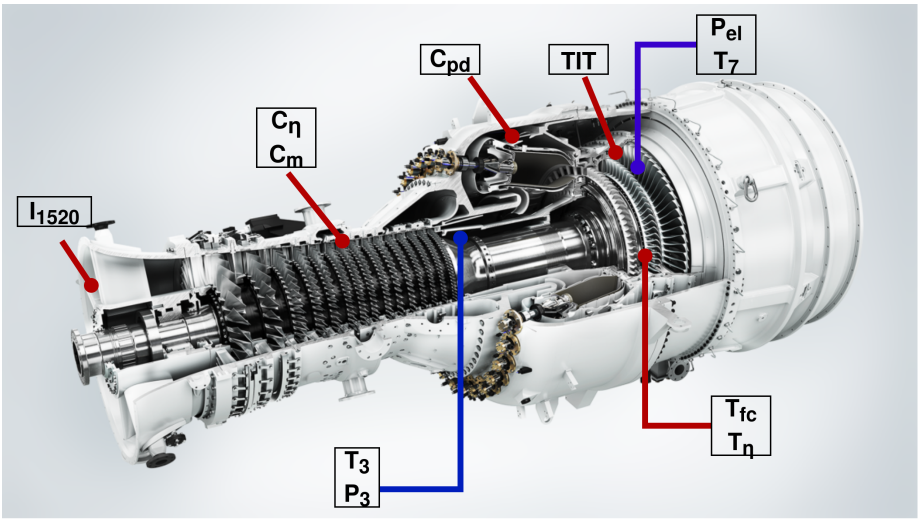

Figure 2.6: Sensors for the measurments of the performance parameters and locations for for the machine settings.

2.6 Problem Definition

To calculate the performance of a gas turbine, the GTperform simulation program is used. This tool is constructed to simulate an actual gas turbine as accurately as possible. A difficulty of the performance parameters is that every gas turbine manufactured is unique, and performance and technical requirments may differ between machines. Given the settings for a specific gas turbine, the simulations provides the anticipated values of the peformance parameters. The results are compared with observed measurements, and deviations from the simulation model are analyzed. The deviations indicate that the assumed machine settings are not correct, but must be modified in order to obtain a simulated performance that better comply with the measured. The correction to reduce the differences is, at the moment, done manually by the engineers at the performance division in Finspång. For several reasons it can be problematic to find the most probable machine settings. The machine settings parameters in GTperform can interact with each other, which means that adjustments can be made in many different ways, and it is difficult to know which one that gives the best solution.

We will henceforth refer to measurements of the performance parameters as peformance measurements. The goal is to implement optimization algorithms that establish the likely machine settings from the performance measurements. The developed optimization algorithm will generate an analyzing tool for the employees to use at the performance division.

This research examines the possibility of automatically estimating the machine settings from the performance measurements by GTperform with Simulated Annealing or Gradient Descent. This will be done by analyzing two different objective functions. The first is to estimate the machine settings in GTperform that generate the performance parameters with the smallest difference to the actual performance measurments. For this objective, the algorithm searches the complete grid of solutions. The second is to establish the machine settings that are the most likely based on their probability distributions.

Chapter 3

Mathematical Formulation

In Section 3.1, we describe the first problem formulation, while in Section 3.2, we describe the second problem formulation.

3.1 First Formulation

We are interested in finding the machine settings $\boldsymbol{x} = [TIT, C_\eta, C_m, C_{\mathrm{pd}}, T_{fc}, T_\eta, I_{1520}]$ which give the lowest residuals between the actual performance measurements y^{m} and the simulated $y^{\mathrm{s}}(\boldsymbol{x})$ obtained from GTperform. The machine settings and the peformance measurments are assumed to be normally distributed. The distributions of the settings are established from technical constraints and conditions. For the machine settings, we denote the distribution mean as μ_{x_i} and the standard deviation as σ_{x_i}. For the performance measurements we denote the standard deviation as σ_{y_j}.

The objective function is defined as minimizing the sum of squared residuals. Each term is multiplied by a scale factor which equals to the inverted value of the variance for the corresponding probability distribution. The scale factor is used because the performance measurements represent physical conditions that are measured in different units, and therefore needs to be scaled differently. The objective function is expressed as follows.

$$z_1 = \min_{\boldsymbol{x}} f(\boldsymbol{x}) = \sum_{i=1}^{4} \frac{1}{\sigma_{y_i}^2} (y_i^{\mathrm{m}} - y_i^{\mathrm{s}}(\boldsymbol{x}))^2 \tag{3.1}$$

$$\text{s.t.} \quad \boldsymbol{x}_{\min} \leq \boldsymbol{x} \leq \boldsymbol{x}_{\max} \tag{3.2}$$

Here, $\boldsymbol{x}_{\min}$ and $\boldsymbol{x}_{\max}$ are lower and upper bounds, respectively, on the machine settings.

3.2 Second Formulation

This is achieved by including the probability distributions of the machine settings in the first problem formulation. This penalizes settings by how far they are from their theoretical mean values. As for the performance measurements, the settings are also measured in different units and need to be appropriately scaled. This is done by multiplying each term with a factor that equal the inverted value of its variance. Settings that lie far from the distribution mean will therefore be more penalized than if the setting is closer to the theoretical mean. We define the objective function for the second task as the following

$$z_2 = \min_{\boldsymbol{x}} g(\boldsymbol{x}) = \sum_{i=1}^{4} \frac{1}{\sigma_{y_i}^2}(y_i^{\mathrm{m}} - y_i^{\mathrm{s}}(\boldsymbol{x}))^2 + \sum_{j=1}^{7} \frac{1}{\sigma_{x_j}^2}(x_j - \mu_{x_j})^2 \tag{3.3}$$

$$\text{s.t.} \quad \boldsymbol{x}_{\min} \leq \boldsymbol{x} \leq \boldsymbol{x}_{\max} \tag{3.4}$$

Chapter 4

Theory

In Section 4.1, the theory of black-box optimization is explained. We give a literature background of two main types of procedures. In Section 4.2, earlier studies of the GTperform are discussed. Lastly, in Sections 4.3 and 4.4, we outline the theory of the two selected methods.

4.1 Black-Box Optimization

Black-box theory refers to an area of optimization where we lack the knowledge of a mathematical model. We can only analyze the output result for a selected input domain. In literature, this is represented by a black box, hence the name black-box optimization [5].

A great overview and introduction to the subject is given by Mario, Yaun, Micheal, and Saman [5]. They present several algorithms for solving the black-box problem, based on earlier work by Rice [6] on algorithm selection. The algorithms are divided into two groups, which are either deterministic or stochastic. Deterministic algorithms are methods that for a particular input will always generate the same output, while stochastic algorithms are constructed based on randomness and probability distributions. Hence, the latter methods can give different outputs for the same input. The algorithms are further categorized into subgroups based on search procedures, theoretical assumptions, and selection criteria. [5].

4.1.1 Deterministic Algorithms

Deterministic algorithms can be divided into three classes: Line Search methods, Trust Region methods, and Pattern Search methods [5].

Lundgren, Rönnqvist and Värbrand in [3, pp. 257-261] describe Line Search methods as iterative algorithms that use gradient or hessian information to establish an optimal point. Depending on if the problem is convex or not, these algorithms either finds a local or a global optimal solution.

The second class is Trust Region methods. These are often used when the time complexity to evaluate the objective function is very high. In these situations, we may only be able to analyze a few solutions. The idea of these methods is to approximate the problem around the current solution by a mathematical expression, creating a smooth curvature that would be easier to evaluate. These algorithms then estimate the optimal solution for the approximate curvature [11].

The third class of deterministic algorithms are Pattern Search methods. At each iteration, these algorithms construct various solutions according to chosen pattern. The solutions are evaluated and the method chooses the point with the lowest objective value, and then continues the search procedure from the new solution. If the method cannot find any better solution, the search pattern changes, and the search process continues with the new pattern. If the method can not find any better solution after a selected number of different search patterns, the algorithm stops [5].

4.1.2 Stochastic Algorithms

Stochastic methods consist of the following subclasses: Random Search methods, Simulated Annealing, and Population-Based algorithms [5].

The Random Search methods sample elements from a selected neighbourhood, using a fixed or adaptive probability distribution. If the new solution has a lower objective value than the old one, we move to the new one. If we cannot find any better solution within a specific number of iterations, the algorithm stops [3, pp. 421-428].

Simulated Annealing (SA) is a heuristic technique that uses a random-search procedure to estimate a global optimum. The concept is based on the movement of atoms in metal as they anneal. In the annealing process, the metal is first heated, making the atoms move in the material. As temperature decreases, so do the energy, causing the atoms to stabilize into a crystal structure. The algorithm jumps between solutions, accepting solutions with worse objective values by a probability distribution which depends of a temperature parameter. In the beginning, the temperature is high, but as the algorithm moves between solutions, it decreases [3, pp. 447-448].

Population-Based algorithms are methods that are inspired by genetic theory. Each solution is here viewed as an individual and individuals are combined in an evolutionary process to produce offsprings of solutions, called children. By natural selection, the children that have better objective value survive. These are then put through the same process, creating new children that are combinations of their parents. The process continues until the algorithm finds an individual, that is, solution with a satisfactory objective value [5].

4.2 Earlier Work on GTperform

Vaske [10], described in great detail the use of GTperform in combination with different types of optimization algorithms, each one tested on simulated data at Siemens in Finspång. The focus was to analyze the possibility of using optimization algorithms to automate the estimation process for a simpler gas turbine model. By investigating both the performance and computational power requirements, they concluded that complex methods were unnecessary. Instead, they concluded that the significant difficulty was the computational time complexity in evaluating each parameter setting in GTperform. The methods that performed well were simple algorithms that did not need to evaluate many parameter settings in GTperform

4.3 Simulated Annealing

The Simulated Annealing algorithm establishes a new OK solution by searching for possible solutions within a given neighbourhood. If the new solution has a lower objective value than the old, it is always accepted. On the other hand, if the new solution has a higher objective value, the algorithm accepts or rejects this point with a probability that depends on a temperature. The probabilities are given by the Metropolis Hasting formula, shown below.

$$\begin{cases} p = \exp\left(\frac{z_{\text{new}} - z_x}{T_i}\right), & \text{if } z_{\text{new}} < z_x \\ 1, & \text{otherwise} \end{cases} \tag{4.1}$$

Here, z_{new} is the value of the objective function of the new solution, z_x is objective value for the current solution, and T_i is the temperature of the system at iteration i [8].

The temperature is decreased with a geometric cooling schedule, updating the temperature by $T_{i+1} = \alpha * T_i$, where the cooling factor $\alpha \in (0, 1)$ is a constant [2].

The algorithm stops when a global or local stopping condition is satisfied. These can be constructed in many different ways. The most common one is a maximum number of solutions that the algorithm can evaluate. One can however end the search if the algorithm finds a solution that has an objective value that is lower than a chosen value [7].

The initial temperature is of most concern for the performance and behaviour of the algorithm. It regulates the search procedure by affecting the Metropolis Hasting probability. A high value gives a higher probability, which means that the algorithm can search areas further away. The problem with this is that the algorithm moves too far off, wasting computational time by evaluating to many poor solutions. On the other hand, having a too low starting temperature for the cooling schedule will have the opposite effect, casing the search to be too restricted and not allowing the algorithm to search at neighbourhoods further away. The temperature must be determined beforehand in such a way that in the beginning, the ability to move to a point far away is large enough [9].

One way of choosing an initial temperature is by first generating N random starting solutions. We then select a new solution within a restricted neighbourhood for each solution and calculate the difference, δ, between its function value and that of the given solution. The initial temperature, T_0, is then given by

$$T_0 = \sum_{i=1}^{N} \frac{\delta_i}{N \log(p)}, \tag{4.2}$$

where the parameter p is the probability wanted for the first iteration, for example 0.7. The idea is that the chose temperature should give probability of accepting a new solution in the neighborhood, independent of where the starting point is generated [9].

Another possible option is to choose the initial based on the objective value, z_0, for the initial solution, according to

$$T_0 = \frac{-0.1 z_0}{\log(0.5)}. \tag{4.3}$$

This temperature will accept solutions with a 10% worse objective value with a probability of 0.5. This inital choice of temperature is done in the first iteration of SA and does not need any additional function evaluations before the algorithm can be started [9].

4.4 Gradient Descent

The steepest descent is a deterministic method that for each iteration computes a search direction, p_n, from a given point, x_n, and decides how far to move along that direction. The new point, x_{n+1} is given by

$$x_{n+1} = x_n + \alpha_n p_n. \tag{4.4}$$

where α_n is the step length taken. The progress of this method depends on the choice of both the direction p_n and the step length α_n [3, pp. 257-261].

If the optimization problem is to minimize an objective function, the search direction p_n must be a descending direction [3, pp. 257-261], that is, such that

$$p_n \nabla f(x_n) < 0. \tag{4.5}$$

A commen choice of search direction is

$$p_n = -B_n^{-1} \nabla f(x_n), \tag{4.6}$$

where B_n is a symmetric and non-singular matrix. In the Gradient Descent method, B_n is the identity matrix, which means that the search direction is the negative gradient at the point x_n [3, pp. 257-261].

4.4.1 Step Length

The selection of the step length α_n is a difficult task. We want to find the value that gives the lowest objective value for the search direction p_k. The difficulty is that there is a trade-off between rate of convergency and time complexity. To achieve high accuracy, we need to calculate and evaluate many different values α to establish the optimal one, which would take a long time. The easiest way is to use a constant value of α. A problem with this is that it will not guarantee that the algorithm converges to a local minimum point, since there is a possibility that the algorithm could jump too short or too far between different iterations, missing the optimal solution. At the start of the procedure, the step length should be larger, but as the procedure converges to minimal points, the distances should become shorter [4, pp. 50-52].

Ahmed et al. [1] describe the possibility to use a quadratic interpolation to establish the optimal, α_n. This procedure consists of three steps. First, select two points along the direction p_n and denote these as x_1 and x_2. Second, evaluate the objective values $f_1(x_1)$ and $f(x_2)$, together with the value $f(x_0)$ at the starting point, x_0.

The points are then interpolated with a quadratic function, and the minimal point x_{min} of the interpolation is established from the derivative. The distance between x_0 and x_{min} describes how far the algorithm wants to move along the direction p_n [1].

4.4.2 Wolfe Conditions

The step length α_n can sometimes become relatively small, generating short moves that give a small decrease of the objective value. This makes the algorithm inefficient, as it does not fully utilize the information by the search direction p_k. We here describe two conditions that the step lenght should satisfy in order to guarentee a sufficient decrease in the objective value. These are called the Wolfe condition [3, pp. 276-278].

The first is the Armijo condition

$$f(x_n + \alpha_n p_n) \leq f(x_n) + c_1 \alpha_n \nabla f(x_n)^T p_n, \tag{4.7}$$

where c_1 is a scalar parameter chosen between 0 and 1. It will establish a minimal level that the objective value must satisfy [4, pp. 52-56].

The Wolfe condition requires α_n to satisfy

$$\nabla f(x_n + \alpha_n p_n) p_n \geq c_2 \nabla f(x_n)^T p_n, \tag{4.8}$$

where c_2 is a constant that must be greater than or equal to c_1 and less than or equal to 1. This condition rules out that the values of α_n are too small. It is only accepting steps giving a directional derivative that is greater than a constant of the previous. Together, these two conditions ensure that α_n gives sufficient decrease in the objective value [4, pp. 50-52]. In conclusion, the Wolfe conditions require that the value α_n must satisfy the following conditions [4, pp. 52-56].

$$f(x_n + \alpha_n p_n) \leq f(x_n) + c_1 \alpha_n \nabla f(x_n)^T p_n \tag{4.9}$$

$$\nabla f(x_n + \alpha_n p_n) p_n \geq c_2 \nabla f(x_n)^T p_n \tag{4.10}$$

$$c_1 \in [0, 1], \quad c_2 \in [c_1, 1] \tag{4.11}$$

Chapter 5

Implementation

The two proposed algorithms are thoroughly explained below. First in Section 5.1 we describe the underlying concepts and overall procedure for Simulated Annealing (SA), while in Section 5.2 we outline the Gradient Descent (GD). The algorithms have been implemented in Python.

5.1 Simulated Annealing

In Section 5.1.1, we describe the higher-level representation of the method. In Sections 5.1.2– 5.1.4 the components are further explained and the pseudocode of the algorithm is described in Section 5.1.5.

5.1.1 Outline of Algorithm

In Figure 5.1, we display the modules of the SA method. It consists of multiple blocks that are aligned and iteratively executed until one of two termination conditions is satisfied.

The algorithm begins by randomly selecting the starting solution, $\boldsymbol{x}_{\text{init}}$, from seven uniform probability distributions, one for each variables. In the first iteration, we set both the solution vector, $\boldsymbol{x}$, and the best found solution, $\boldsymbol{x}_{\text{best}}$, equal to $\boldsymbol{x}_{\text{init}}$. Finally, the initial temperature, T_{init}, is set to 1000 by equation (4.2) in Section 4.3

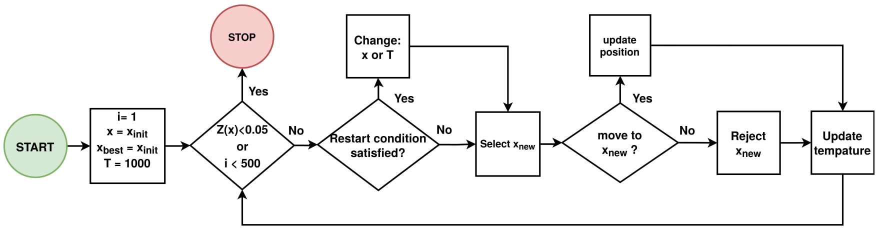

Figure 5.1: The main components of the Simulated Annealing.

There are two possible stopping conditions, either the algorithm stops after 500 iterations or when the objective value for the current solution is less then a close number to zero, in our case 0.05. If the vector $\boldsymbol{x}$ does not fulfill the stopping conditions, the iterative search method continues. The first step is to investigate if $\boldsymbol{x}$ satisfies the restart condition. The restart ability has two significant purposes. Firstly, to secure that the algorithm does not get stuck bu rejecting possible moves. Secondly, it ensures that the algorithm has not moved too far off from the best found solution, and is searching around poor regions with high objective values. If the restart condition is satisfied, the algorithm would either change the vector $\boldsymbol{x}$ to the best-found vector $\boldsymbol{x}_{\text{best}}$ we have established so far or increase the temperature, giving a higher probability for jumping into other regions. Important to clarify is that the restartbility is inactivated for the first 20 iterations and is switched on afterward.

Given the solution $\boldsymbol{x}$, the algorithm calculates the neigbhourhood, explained in Section 5.1.2. Within the neighbourhood the algorithm randomly selects a new solution, $\boldsymbol{x}_{\text{new}}$, and calculates the objective value $z(\boldsymbol{x}_{\text{new}})$ by using GTperform. Whether, the solution is accepted or rejected is explained in more detail in Section 5.1.4.

The last step for each iteration, regardless if we have moved from the current solution or not, is to update the temperature, T_i. Finally we invesigate if the new point satisfies any of the two stopping conditions. If it does, the algorithm stops and writes the best-found solution. Otherwise, it continues the iterative process until one of the two stopping conditions are satisfied.

5.1.2 Neighbourhood

The neighbourhood is defined as a portion of the feasible set. If the problem had been in two variables, x_1 and x_2, the feasible set would look like in Figure 5.2. The larger box illustrates the whole feasible set, while the smaller boxes display the neighbourhoods. The intersections of the feasible set and the neighbourhoods are shown in grey colour, the algorithm is allowed to select the new point only within these regions. For the problem with six variables, the neighborhood and the gray region are hypercubes in six dimensions.

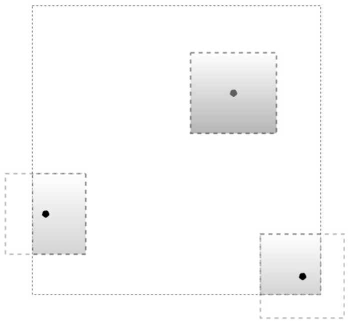

Figure 5.2: Illustrations of the Neighbourhood regions in a two dimensional space for three points.

5.1.3 Restart Ability

The restart functionality makes sure that the algorithm does not move too far away from the current best solution. The large number of possible jumps in combination with the complexity of evaluating the objective function makes it essential for the algorithm not to get stuck within poor regions. This is especially important at the beginning of the algorithm, when the temperature is high. Suppose that the method has not found a solution that has a lower objective value then to the solution x_{best} within 30 iterations. In that case, the search procedure will be restarted from solution x_{best} with the temperature setting left unchanged.

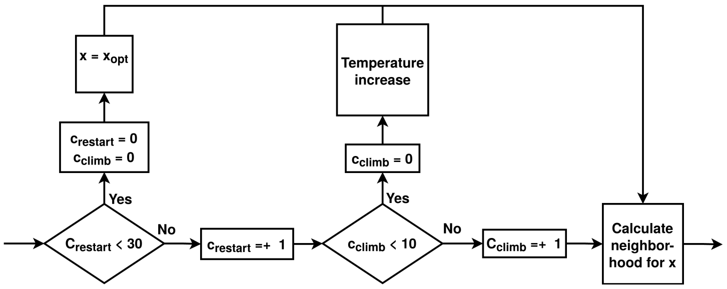

Figure 5.3: Schedule for the reset procedure.

With the restart functionality, we can search along multiple paths from the current best solution to find areas with better solutions. In the beginning, the algorithm tends to search in wider areas, but as temperature decreases, so does the region that is searched. In the later stage of the algorithm, there is a low probability of jumping to worse solutions, and at this point the method will only accept better solutions. To accommodate this behaviour, we add a feature in the restart that will affect the temperature. We add a climbing ability within the restart function so that the algorithm does not get stuck at specific areas because of a too low temperature. If the method rejects ten solution in a row, it updates the temperature, increasing it to promote the possibility to jump and continue the search. In the theory section, we described a method for the initial temperature setting. We will use the same method to increase the temperature by using the expression

$$T_{\text{iter}} = \frac{-z(\boldsymbol{x}_{\text{init}}) \times 0.1}{\log(0.5)}, \tag{5.1}$$

which means that an objective value that is 10% worse than the current one should be accepted with a probability of 0.5. Suppose this temperature increase did not help for ten successive iterations. In that case, the algorithm will update the temperature once more and continues until it either jumps to a new solution or restarts from the best found solution. Further, the algorithm will then set the counters for the restart and climbing ability to zero.

5.1.4 Selection Algorithm

In Figure 5.4, we display the move selection procedure. Firstly, we investigate if the function value $z(\boldsymbol{x}_{\text{new}})$ is smaller than $z(\boldsymbol{x}_{\text{best}})$. If true, set both $\boldsymbol{x}$ and the best found solution $\boldsymbol{x}_{\text{best}}$ equal to $\boldsymbol{x}_{\text{new}}$. Otherwise, evaluate if the value is less than $z(\boldsymbol{x})$. If this is true, then we move to the new point, but update only $\boldsymbol{x}$ and set it to $\boldsymbol{x}_{\text{new}}$. Important to notice is that we are not changing the solution $\boldsymbol{x}_{\text{best}}$.

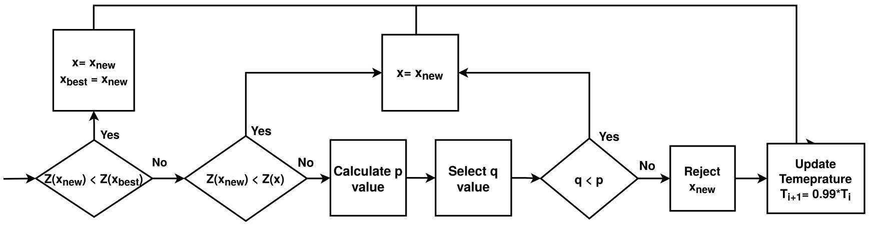

Figure 5.4: A diagram for the selection procedure.

In the second case, if none of the earlier conditions were satisfied, then the objective value $z(\boldsymbol{x}_{\text{new}})$ is higher than both $z(\boldsymbol{x})$ and $z(\boldsymbol{x}_{\text{best}})$. The algorithm then analyses if it should still move to the newly established solution by the acceptance probability p from the current temperature T_i by equation (4.1) in Section 4.3. To do this, we choose a random value q from a continuous uniform distribution between zero and one. If q is less than p, we update and select $\boldsymbol{x}_{\text{new}}$ as our new solution $\boldsymbol{x}$. Lastly, update the temperature setting by multiplying it with a scale factor and restart the procedure.

5.1.5 Pseudocode for SA

Algorithm 1 Simulated Annealing

```
1: Set: $i = 1$
2: Select temperature: $T_\text{i} = 1000$
3: Set: $c_\text{restart} = 0$
4: Set: $c_\text{climb} = 0$
5: Generate a starting vector: $\boldsymbol{x}_\text{init}$
6: Set the best and current point equal to the inital point:
   $\boldsymbol{x}_\text{best} = \boldsymbol{x} = \boldsymbol{x}_\text{init}$
7: Calculate the objective value for the initial point: $z(\boldsymbol{x}_\text{init})$
8: Set the best and current objective values equal to the initial value: $z_\text{best} = z_\text{x} = z(\boldsymbol{x}_\text{init})$
9: while $i < 500$ or $z_{\boldsymbol{best}} < 0.05$ do
10:     if $c_\text{restart} < 30$ then
11:         Set: $\boldsymbol{x} = \boldsymbol{x}_\text{best}$
12:         Set: $z_x = z_\text{best}$
13:         Set: $c_\text{restart} = 0$
14:         Set: $c_\text{climb} = 0$
15:         if $c_\text{climb} < 10$ then
16:             Increase temperature: $T_\text{iter} = \frac{-z(\boldsymbol{x}) \times 0.1}{\log(0.5)}$
17:             Set: $c_\text{climb} = 0$
18:             Set: i = i+1
19:             Break
20:     else
21:         Set: $c_\text{climb} = c_\text{climb} + 1$
22:         Set: $c_\text{restart} = c_\text{restart} + 1$
23:     Calculate neigbourhood $A(\boldsymbol{x})$ for current solution $\boldsymbol{x}$
24:     Chose new point within the neighbourhood: $\boldsymbol{x}_\text{new} \in A(\boldsymbol{x})$
25:     Calculate the $\boldsymbol{x}_\text{new}$ objective value: $z_\text{new} = z(\boldsymbol{x}_\text{new})$
26:     if $z_\text{new} < z_\text{best}$ then
27:         Set: $\boldsymbol{x}_\text{best} = \boldsymbol{x}_\text{new}$
28:         Set: $z_\text{best} = z_\text{new}$
29:         Set: $\boldsymbol{x} = \boldsymbol{x}_\text{new}$
30:         Set: $z_x = z_\text{new}$
31:     else if $z_\text{new} < z_x$ then
32:         Set: $\boldsymbol{x} = \boldsymbol{x}_\text{new}$
33:         Set: $z_x = z_\text{new}$
```

Algorithm 1 Simulated Annealing (continued)

```
34:     else
35:         Calculate acceptance probability: $p = \exp\left(\frac{z_{\text{new}} - z_x}{T_i}\right)$
36:         Choose value from a uniform distribution: $q \in U[0, 1]$
37:         if $q < p$ then
38:             Set: $\boldsymbol{x} = \boldsymbol{x}_{\text{new}}$
39:             Set: $z_x = z_{\text{new}}$
40:     Update temperature: $T_{\text{i}} = 0.99 \times T_{\text{i}}$
41:     Set: i = i+1
42: end while
43: return $\boldsymbol{x}_{\text{best}}$ and $z_{\text{best}}$
```

5.2 Gradient Descent

In Section 5.2.1 we give a high-level representation of the method, while the main parts of the algorithm are explained in more detail in Sections 5.2.2 and 5.2.3. In Section 5.2.4 we illustrate the pseudocode.

5.2.1 Overview of the Gradient Descent

In Figure 5.5 we display Gradient Descent approach.

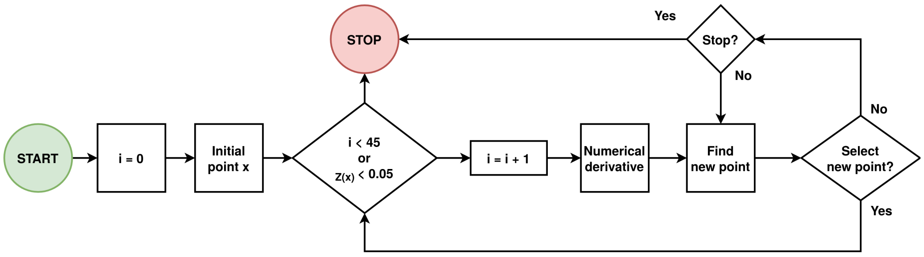

Figure 5.5: Overview of the Gradient Descent

As for the Simulated Annealing method, there are two stopping conditions. The method stops after 45 iterations or if the objective value is smaller than 0.05. If the solution does not fulfill any of these criteria, the algorithm continues the search by calculating the gradient for the solution, as described in Section 5.2.2.

Given the gradient, the algorithm finds the best step length by interpolating three different solutions, as explained in Section 5.2.3. If the new solution satisfies the Armijo and Wolfe conditions with the parameters $c_1 = 0.01$ and $c_2 = 0.05$, the method moves to the new solution. If the solution does not satisfy these conditions, the algorithm analyses if the interpolation should be re-made with other solutions. The interpolation method has an stopping condition, which stops the complete search procedure if it can not find a better solution within seven attempts.

5.2.2 Search Direction

The search direction is determined by a numerical approximation of the gradient. This is done by the forward difference method, expressed as

$$\frac{\partial z}{\partial x_k} = \frac{z(x_1,\ldots,x_k+h,\ldots x_n) - z(x_1,\ldots,x_k,\ldots x_n)}{h}, \quad k=1,\ldots,n, \quad (5.2)$$

with $h = 10^{-5}$. For each partial derivative, the method needs to evaluate one new point, at the distance h. The gradient is given by

$$\nabla z = (\frac{\partial z}{\partial x_1}, \frac{\partial z}{\partial x_2}, \frac{\partial z}{\partial x_3}, \frac{\partial z}{\partial x_4}, \frac{\partial z}{\partial x_5}, \frac{\partial z}{\partial x_6}, \frac{\partial z}{\partial x_7}). \quad (5.3)$$

The gradient is normalized with the euclidean norm, giving the vector

$$\gamma = \frac{1}{\|\nabla z\|^2}\nabla z. \quad (5.4)$$

The steepest descent direction at $\boldsymbol{x}$ is given by $h_x = -\gamma$.

5.2.3 Interpolation Method

In Figure 5.6 we show a flow diagram for the interpolation method.

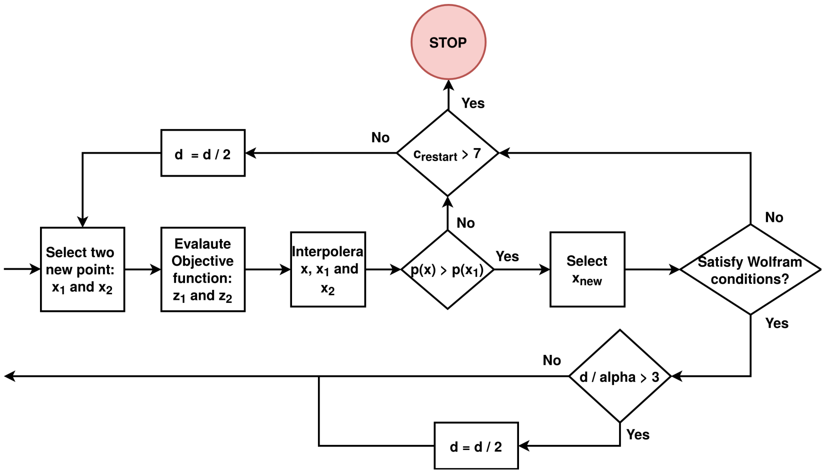
STOP
Yes
No
d = d / 2
$c_{restart}$ > 7
No
Select two new point: x_1 and x_2
Evalaute Objective function: z_1 and z_2
Interpolera x, x_1 and x_2
p(x) > p(x_1)
Yes
Select x_{new}
Satisfy Wolfram conditions?
No
Yes
No
d / alpha > 3
Yes
d = d / 2

Figure 5.6: The interpolation procedure for Gradient Descent

The interpolation starts by selecting two solutions on equal distance along the search direction,

$$\boldsymbol{x}_{\rm n} = \boldsymbol{x} + n d_s \boldsymbol{h}_{\boldsymbol{x}}, \quad n = 1, 2. \tag{5.5}$$

The distance parameter d_s is initially set to 0.01. This parameter can change if the two selected solutions do not give any satisfactory results.

The two solutions are evalauted as $z_1 = z(\boldsymbol{x}_1)$ and $z_2 = z(\boldsymbol{x}_2)$. In the next step we interpolate the three objective values z, z_1 and z_2 by a quadratic expression

$$p(\alpha) = \alpha^2 + b\alpha + c, \tag{5.6}$$

where a, b and c are constant. The interpolation between $\boldsymbol{x}, \boldsymbol{x_1}$ and $\boldsymbol{x_2}$ for the function values z, z_1 and z_2 is displayed by the blue dashed line in Figure 5.7. If $a > 0$, the minimal point of the quadratic interpolation is calculated. Otherwise, the interpolation has no minimal point and it is rejected. In this situation, the algorithm will restart with the distance divided by two. The reason for the bisection is due to efficiency. By halving the distance, the interpolation only needs to evaluate one new solution for the new trial, and can therefore reuse information from the unsuccessful interpolation. The procedure is repeated a maximum of seven times. If it cannot find a solution within this limit, the interpolation procedure stops, and so will the complete algorithm and return the solution $\boldsymbol{x}$ as the best found.

If the interpolation finds a minimum point, α_{min}, it is used to calculate the distance we should move in the search direction. The best solution is established by $\boldsymbol{x}_{\text{best}} = \boldsymbol{x} + \alpha_{\text{min}}\boldsymbol{h}_{\text{x}}$, we then analyze if the objective value $z(x_{\text{best}})$ satisfies the Armijo and Wolfe conditions. If it does not fulfill these criteria, then the algorithm will reject the solution $\boldsymbol{x}_{\text{best}}$ and restarts the interpolation method with the d_s parameter divided by two.

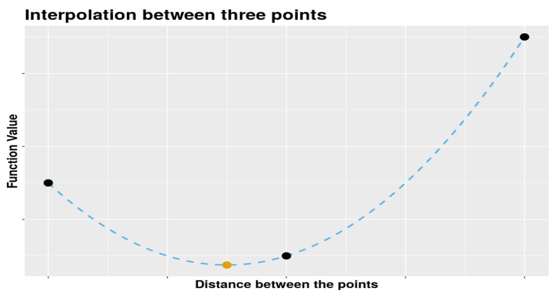

Figure 5.7: Interpolation curve by three points in black with the yellow point equal to the x_{best}.

For each successful move, the algorithm analyses the selected distance α_{min} it moves from the starting point. By looking at the ratio between the value α_{min} and the starting distance d_s, the method analyses the number of times it failed before finding a solution that satisfied the Armijo and Wolfe conditions. If it was too large, then the starting distance was too large and that the method wasted a lot of time trying to interpolate points that were too far away from each other. For this reason, if the ration $\frac{\alpha}{d_s}$ is larger than three for three consecutive successful iterations, we divide the distance d_s by two.

5.2.4 Pseudocode of Gradent Descent

Algorithm 2 Gradient Descent

```
1: Set distance $d = 0.01$
2: Set iteration parameter to zero: $c = 0$
3: Generate a random starting solution: $\boldsymbol{x}$
4: Calculate the objective value for the initial point: $z = z(\boldsymbol{x})$
5: while $c < 45$ or $z < 0.05$ do
6:     $c = c + 1$
7:     Calculate the numerical derivative $\nabla z$
8:     Set the search direction to $h_x = -\frac{1}{\|\nabla z_x\|^2}\nabla z$
9:     Set $\boldsymbol{x_1} = \boldsymbol{x} + d_{\text{iter}}\boldsymbol{h_x}$ and $\boldsymbol{x_2} = \boldsymbol{x} + 2d_{\text{iter}}\boldsymbol{h_x}$
10:    Evaluate objective values $z_1 = z(\boldsymbol{x_1})$ and $z_2 = z(\boldsymbol{x_2})$
11:    Interpolate $p(\alpha) = a\alpha^2 + b\alpha + c$ using the three points $z_x$, $z_1$, $z_2$
12:    if $p(\alpha) > 0$ then
13:        The interpolation curve is a parabola
14:        Find the minimal point $\alpha_{\text{min}}$ of $p(x)$.
15:        Calculate the new best solutions $\boldsymbol{x}_{\text{best}} = \boldsymbol{x} + \alpha_{\text{min}}\boldsymbol{h}_{\text{x}}$
16:        if $z(\boldsymbol{x}_{\text{best}})$ satisfies Armijo and Wolfe conditions then
17:            $\boldsymbol{x} = \boldsymbol{x}_{\text{best}}$
18:            $z = z(\boldsymbol{x}_{\text{best}})$
19:            if the ratio $\frac{d}{\alpha} > 3$ then
20:                if $c_{\text{restart}} > 3$ then
21:                    $d = \frac{d}{2}$
22:                    $c_{\text{restart}} = 0$
23:            else
24:                $c_{\text{restart}} = c_{\text{restart}} + 1$
25:        else
26:            if $c_{\text{restart}} > 7$ then
27:                Stop algorithm
28:                return $\boldsymbol{x}$ solution and $z$
29:            else
30:                $d = \frac{d}{2}$
31:                $c_{\text{temp}} = c_{\text{temp}} + 1$
32: return solution $\boldsymbol{x}$ and objective value $z$
```

Chapter 6

Numerical Results

The numerical results are divided into three sections, one for each gas turbine. Each method was tested 15 times for the two problem formulations. To easier present the results, the residuals for the variables settings and performance measurements are transformed to standard normal distributions, with zero mean and standard deviation one.

6.1 Rioja

For the Rioja dataset, the results are divided into two parts. In Section 6.1.1 we show the results for the first problem formulation, while in Section 6.1.2 we show the outcome for the second problem formulation.

6.1.1 First Problem Formulation

Figures 6.1 and 6.2 show the objective values for Simulated Annealing and Gradient Descent, with the best objective value for each run in Figure 6.3. Figure 6.4 shows boxplots of the residuals for the machine settings and performance measurements.

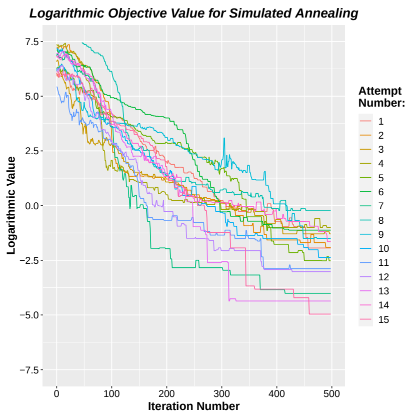

Figure 6.1: The SA process for the 15 runs, where each starting position was randomly selected.

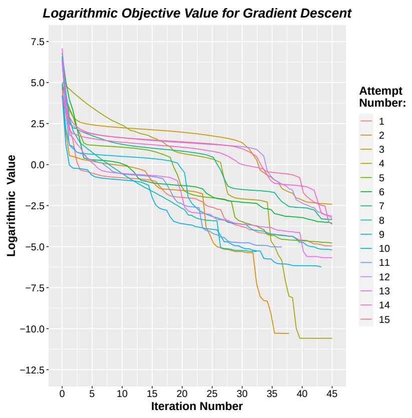

Figure 6.2: The GD process for the 15 runs, where each starting position was randomly selected.

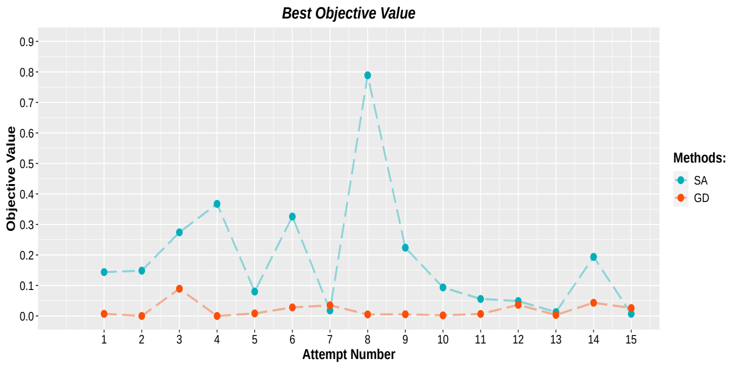

Figure 6.3: The best found objective function values for each of the 15 runs.

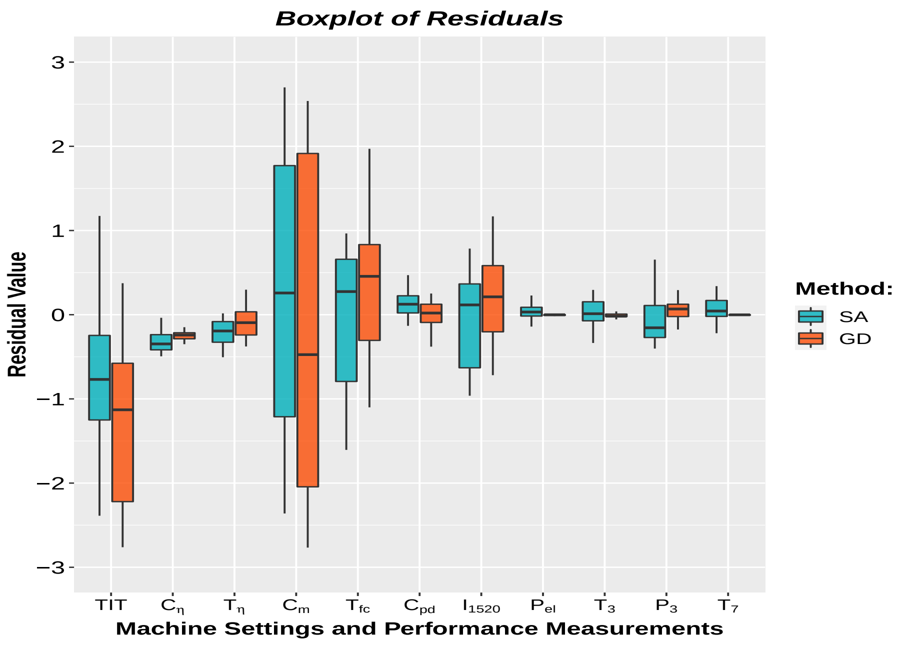

Figure 6.4: Boxplots of the machine settings and performance measurements for the two methods, with red for GD and blue for SA. Within each box, the central mark indicates the median. The bottom and top edges of the box indicate the 25th and 75th percentiles, and the thin lines at the ends of each box represent outliners.

6.1.2 Second Problem Formulation

Figures 6.5 and 6.6 illustrate the objective values for Simulated Annealing and Gradient Descent, with the best objective value for each run in Figure 6.7. Figure 6.8 shows the boxplots of the residuals for the machine settings and performance measurements

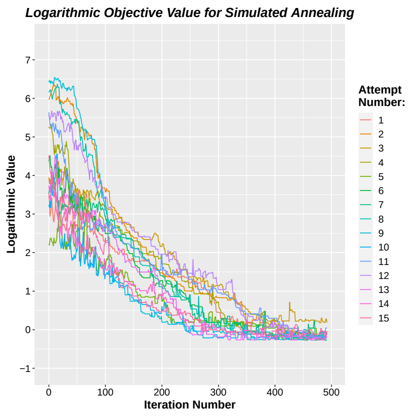

Figure 6.5: The SA process for the 15 runs, where each starting position was randomly selected.

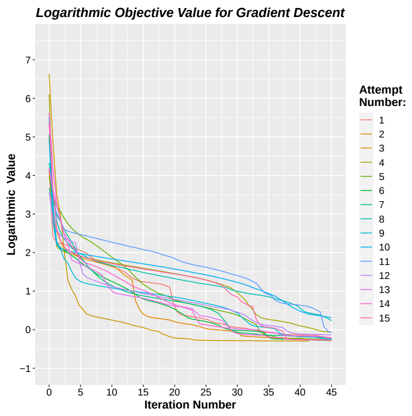

Figure 6.6: The GD process for the 15 runs, where each starting position was randomly selected.

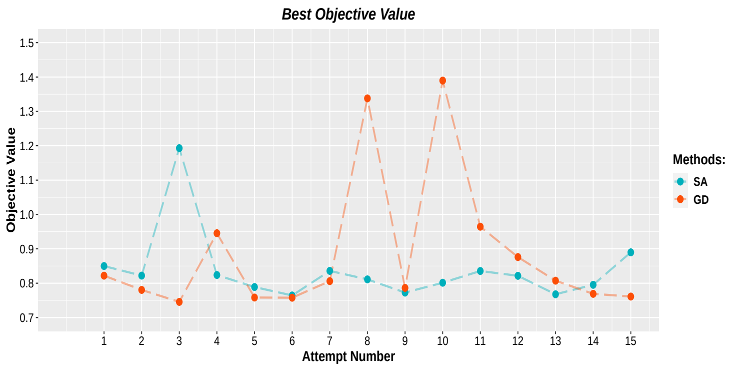

Figure 6.7: The best found objective function values for each of the 15 runs.

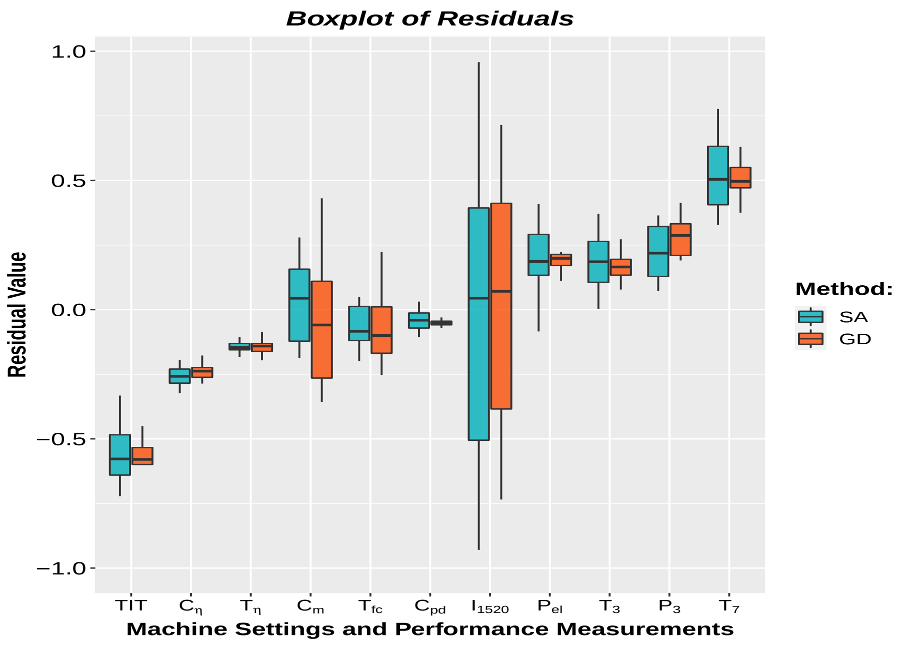

Figure 6.8: Boxplots of the machine settings and performance measurements for the two methods, with red for GD and blue for SA. Within each box, the central mark indicates the median. The bottom and top edges of the box indicate the 25th and 75th percentiles, and the thin lines at the ends of each box represent outliners.

6.2 Holland

For the Holland machinery the results are divided into two parts. In Section 6.2.1 we show the results for the first problem formulation, while in Section 6.2.2 we show the outcome for the second problem formulation.

6.2.1 First Problem Formulation

Figures 6.9 and 6.10 show the objective value for Simulated Annealing and Gradient Descent, with the best objective value for each run in Figure 6.11. Figure 6.12 shows the boxplots of the residuals for the machine settings and performance measurements

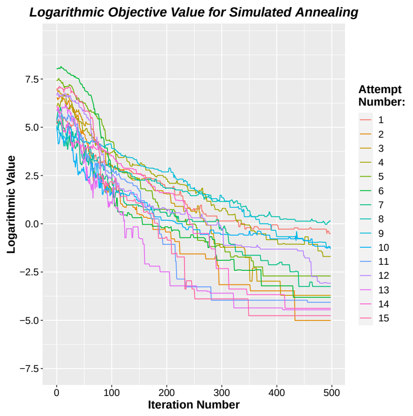

Figure 6.9: The SA process for the 15 runs, where each starting solution was randomly selected.

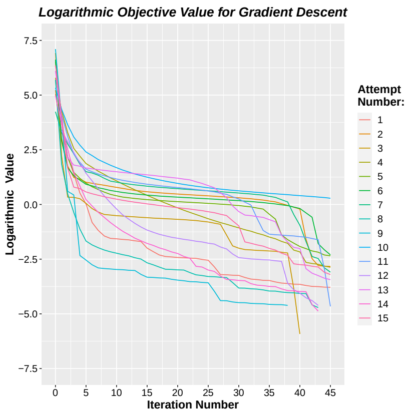

Figure 6.10: The GD process for the 15 runs, where each starting solution was randomly selected.

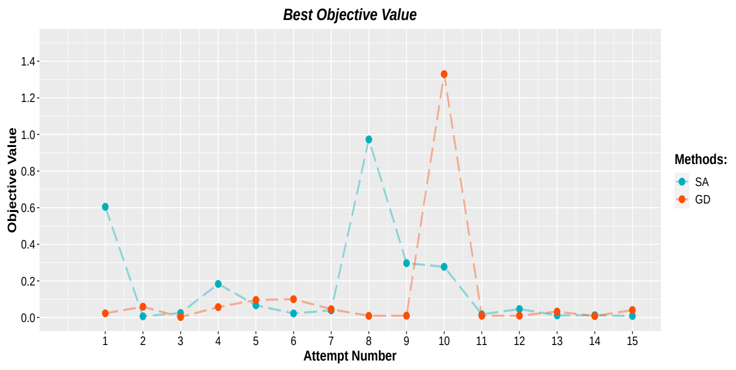

Figure 6.11: The best found objective values for each of the 15 runs.

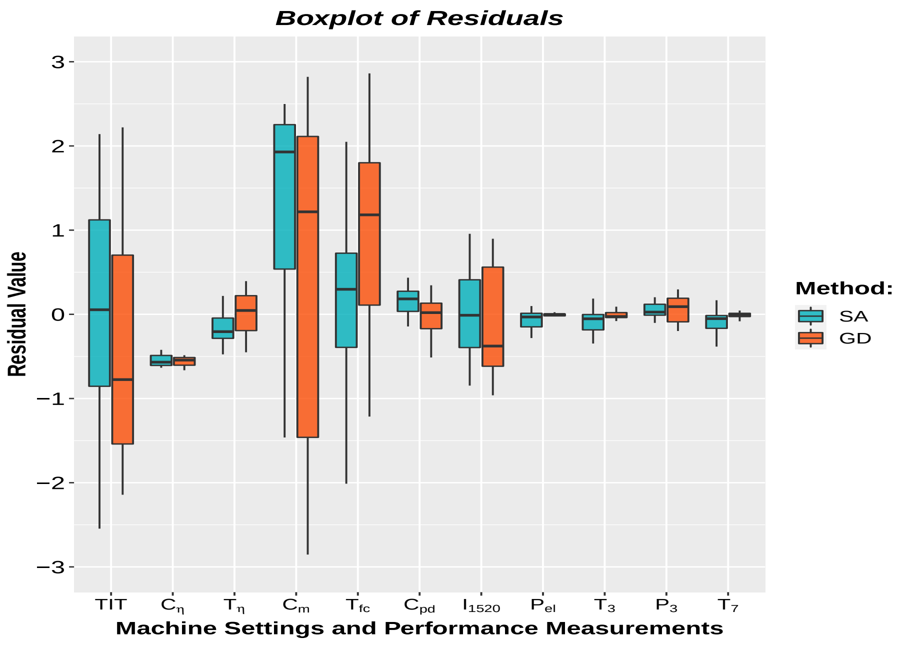

Figure 6.12: Boxplots of the residuals of the machine settings and performance measurements for the two methods, with red for GD and blue for SA. Within each box, the central mark indicates the median. The bottom and top edges of the box indicate the 25th and 75th percentiles, and the thin lines at the ends of each box represent outliers.

6.2.2 Second Problem Formulation

Figures 6.13 and 6.14 show the objective values for Simulated Annealing and Gradient Descent, with the best objective value for each run shown in Figure 6.15. Figure 6.16 shows the boxplots of the residuals for the machine settings and performance measurements.

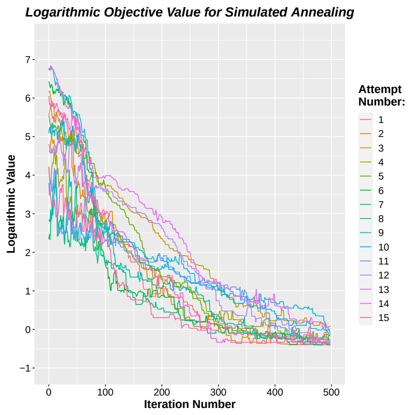

Figure 6.13: The Simulated Annealing process for the 15 runs, where each starting solution was randomly selected.

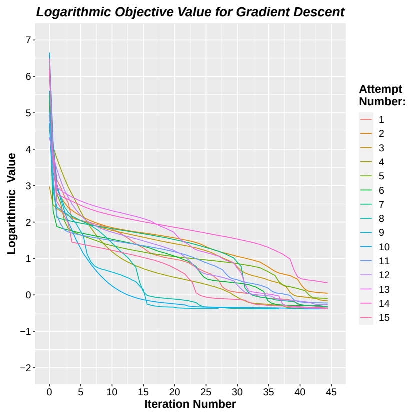

Figure 6.14: The Gradient Descent process for the 15 runs, where each starting solution was randomly selected.

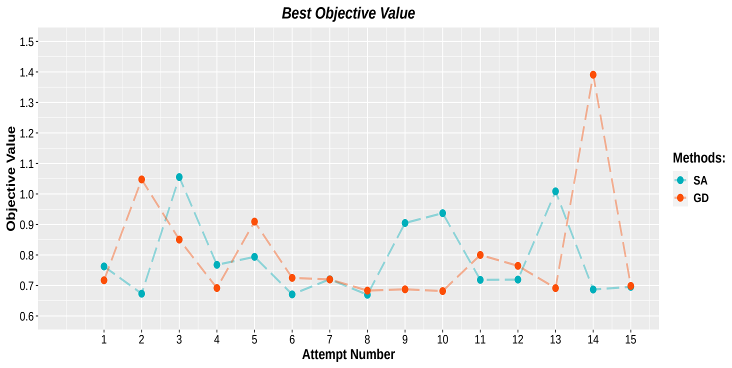

Figure 6.15: The best found objective values for each of the 15 runs.

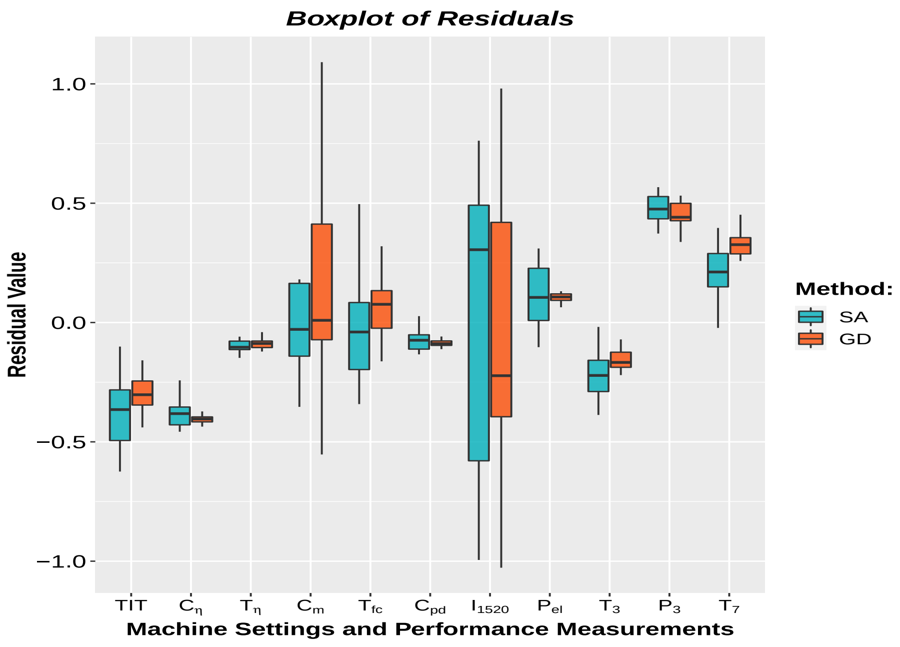

Figure 6.16: Boxplots of the machine settings and performance measurements for the two methods, with red for GD and blue for SA. Within each box, the central mark indicates the median. The bottom and top edges of the box indicate the 25th and 75th percentiles, and the thin lines at the ends of each box represent outliers.

6.3 Delimara

For the Delimara machinery the results are divided into two parts. In Section 6.3.1 we show the results for the first problem formulation, while in the Section 6.3.2 we show the outcome for the second problem formulation.

6.3.1 First Problem Formulation

Figures 6.17 and 6.18 show the objective values for Simulated Annealing and Gradient Descent, with the best objective value for each run in Figure 6.19. Figure 6.20 shows the boxplots of the residuals for the machine settings and performance measurements.

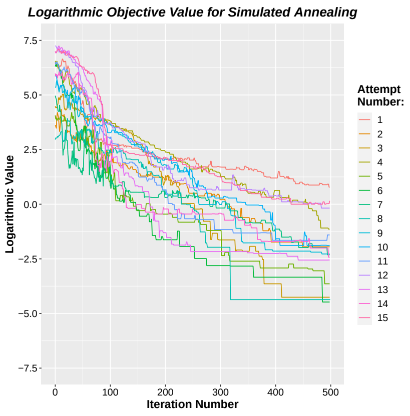

Figure 6.17: The SA process for the 15 runs, where each starting solutions was randomly selected.

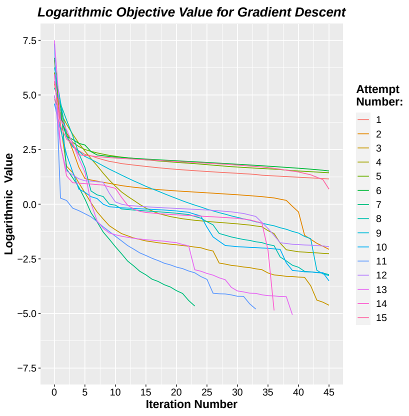

Figure 6.18: The GD process for the 15 runs, where each starting solutions was randomly selected.

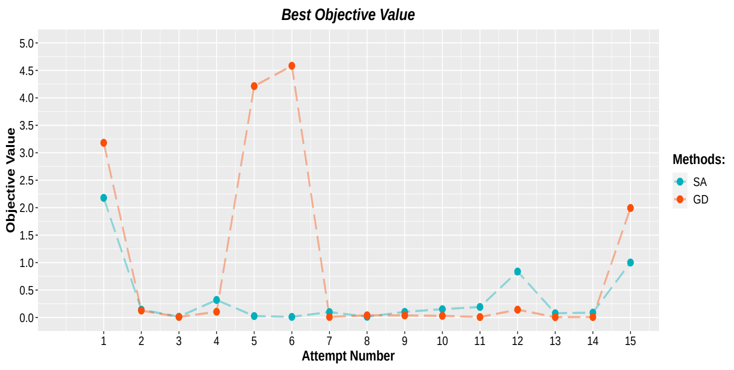

Figure 6.19: The best found objective function values for each of the 15 runs.

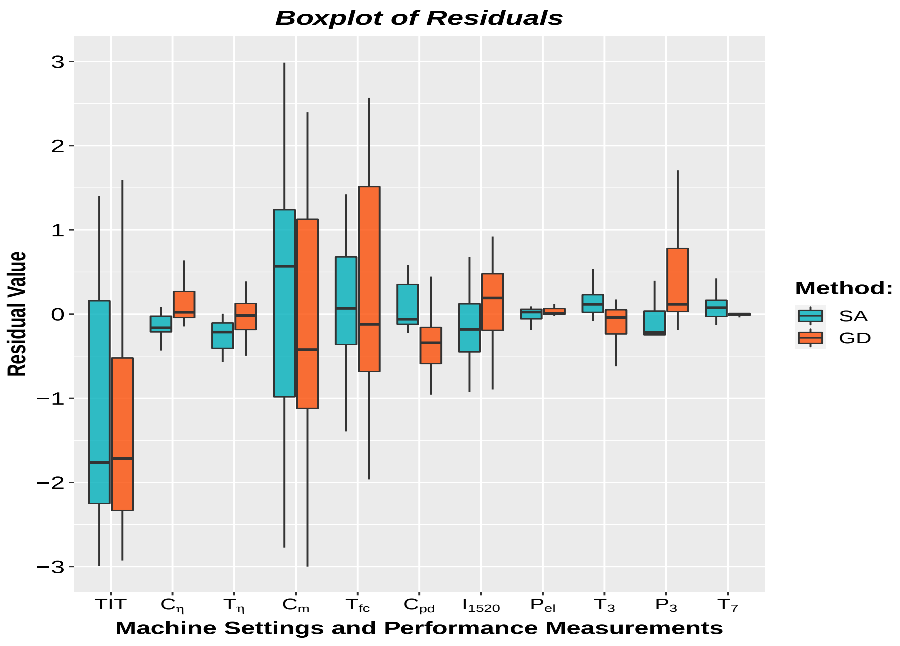

Figure 6.20: Boxplots of the residuals for the machine settings and performance measurements, by the two methods with red for GD and blue for SA. Within each box, the central mark indicates the median. The bottom and top edges of the box indicate the 25th and 75th percentiles, and the thin lines at the ends of each box represent outliers.

6.3.2 Second Problem Formulation

Figures 6.21 and 6.22 show the objective values for the Simulated Annealing and Gradient Descent, with the best objective value for each run illustrated in Figure 6.23. Figure 6.24 shows the boxplots of the residuals for the machine settings and performance measurements

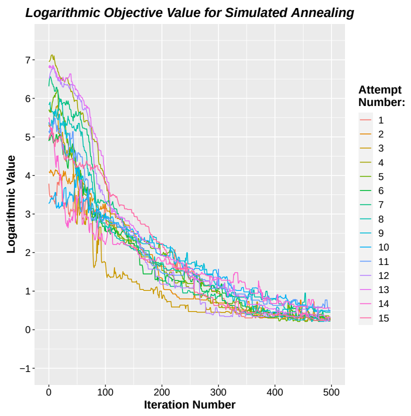

Figure 6.21: The SA process for the 15 runs, where each starting solution was randomly selected.

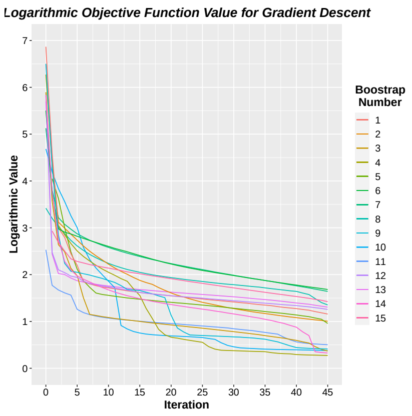

Figure 6.22: The GD process for the 15 runs, where each starting solution was randomly selected.

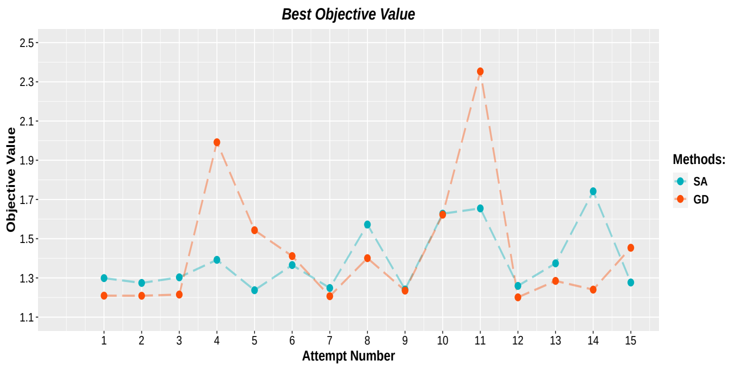

Figure 6.23: The best found objective values for each of the 15 runs.

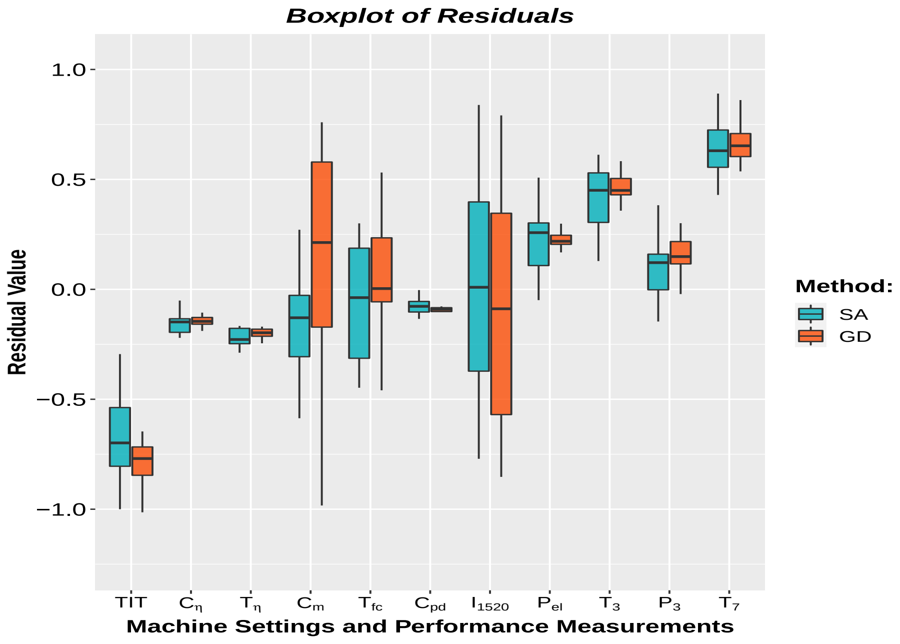

Figure 6.24: Boxplots of the residuals for the machine settings and performance measurements by the two methods, with red color for GD and blue color for SA. Within each box, the central mark indicates the median. The bottom and top edges of the box indicate the 25th and 75th percentiles, and the thin lines at the ends of each box represent outliers.

Chapter 7

Discussion

The discussion is divided into two parts. In Section 7.1 we analyze the performances of the algorithms for the first problem formulation, while in Section 7.2 we examine the performance of the algorithms for the second formulation.

7.1 First Formulation

The best found objective values by the two algorithms for the respective gas turbines are displayed in Figures 6.3, 6.11 and 6.19. The majority of the runs established solutions with low objective values. However, for some attempts the algorithms had difficulties finding satisfactory solutions, since they got stuck at poor local minima.

When comparing the performances between the two algorithms, we notice a significant difference in results between the three gas turbines. We notice that for the Rioja machinery, the Gradient Descent gave the best solutions. The Simulated Annealing had a greater spread between the solutions and produced a poor local minimum in the eighth run. However, for the Delimara turbine, the SA outperformed the GD which gave poor local minima at the first, fifth, sixth, and fifteenth iterations. The objective values of these outliers were higher compared with the outliers for the other two gas turbines. For the Holland turbine, the results were not as clear as for the Rioja and Delimara engine. In this case, the Gradient Descent had better general performance. However, it also had the worst outliers.

By analyzing the overall performance of the algorithms for the three turbines, we notice that there are similar patterns for the two methods. The performance of the SA is shown in Figures 6.1, 6.9 and 6.17. For the first 100 iterations, the algorithm gives large decreases in objective values by finding better solutions without the need to restart. The restart procedure is used in the later stages when the temperature setting is lower and limits the search to a small neighbourhood.

The Gradient Descent gave a significant decrease in the objective value for the first few iterations, as seen in Figures 6.2, 6.10 and 6.18. This is due to large step lengths from the quadratic interpolation method, described in Section 5.2.3. After the first few iterations, the algorithm struggles for many iterations, generating only small changes in the objective value, since it gets stuck at the same neighbourhood for many consecutive iterations.

With the first objective formulation, we are trying to establish the machine settings in GTperform by using only the performance measurements. By analyzing the residual boxplots, displayed in Figures 6.4, 6.12 and 6.20, we can see that there are similarities between the solutions. For all three turbines, the methods have a large spread for the TIT, C_m, T_{fc} and I_{1520} settings, with the mass capacity scale factor for the combustor being the worst, having solutions that range almost within the complete parameter grid The three other machine settings, C_η, c_{pd} and T_η do not fluctuate as much, having minimal residuals, and are all close to the theoretical zero mean. Further, we notice that the residuals for the performance measurements are smaller for the Gradient Descent than for Simulated Annealing. However, Gradient Descent has a larger spread between the machine settings, while Simulated Annealing has the opposite outcome, with higher residuals for the performance measurements and lower for the machine settings.

For most starting solutions, the algorithms find satisfactory solutions within the time limit. Nonetheless, the varying results make the algorithms unpredictable, we obtain a different solution for the machine settings each time we run them. The solution may be highly different, which could be seen from the boxplots, meaning that it is hard to draw any accurate conclusions.

7.2 Second Formulation

The best found objective values for the algorithms are displayed in Figures 6.7, 6.15 and 6.23. We notice that the solutions have much higher objective values than from the solutions the first problem formulation. This is because we add the machine setting terms to the objective function. The algorithms must now balance between the residuals of machine settngs and the residuals for the performance measurements. We notice that the Simulated Annealing shows a more stable performance with smaller deviations and fewer outliers. Nevertheless, the Gradient Descent found acceptable solutions. The unpredictability of its performance however makes it more unreliable, having a higher probability of giving inaccurate information about the machine settings.

Comparing the complete performance of the algorithms in Figures 6.5, 6.6, 6.13, 6.14, 6.21 and 6.22, shows that most runs converge towards the same endpoint. The residuals for the performance measurements, besides the T_3 parameter for the Holland machinery, are all distributed above zero. By further analyzing each solution, we notice that both Gradient Descent and Simulated Annealing have established solutions which deviate from the true performance measurements. This is because the probability distribution for the machine settings pushs the algorithm to find settings that are closer to the theoretical mean values.

The boxplots displayed in Figures 6.8, 6.16 and 6.24, show that the spread of the residuals for the machine settings is within one standard deviation around zero. The settings that had the most significant deviation throughout the three gas turbines were I_{1520}, T_{fc}, and c_m. The first of these regulates the inlet air temperature before it goes into the turbine. The second and third settings represent the scale factor capacities for the compressor and turbine. An interesting finding for all three gas turbines is that that the two scale factors are simultaneously above or below the zero, with a correlation to the I_{1520} temperature settings. Note that if the scale factor median mark is positive, the temperature median is negative for both the Holland and Delimara gas turbines. The opposite effect can be seen for the Rioja engine, with a negative median for the scale factor and a positive median value for the inlet temperature.

If we further compare the algorithms, we notice that Gradient Descent gives the lowest deviations for the performance measurements. Thishowever gives a higher spread for some machine settings, like the C_m parameter, which is more significant for the Holland and Delimara turbines. We notice an opposite effect for the Simulated Annealing, with a higher deviation for the performance measurments and a lower spread for most machine settings. By analyzing the two most fluctuating variables, C_m and T_{fc}, in detail, we notice that they are much lower for the Simulated Annealing than for the Gradient Descent.

Chapter 8

Conclusion

Two optimization algorithms were developed at the performance division at Siemens to estimate the machine settings for the simulation program GTperform. The difficulty aspect is that every single turbine is unique and tuned for its specific purpose. The goal was to reduce the work for the engineers, which have earlier determined these settings by trail and error. The idea originated from an earlier student thesis, which investigated the possibility of estimation parameters in a simpler gas turbine model using optimization and GTperform from simulated data.

Our work was based on two optimization algorithms: Simulated Annealing and Gradient Descent, which were evaluated for performance measurements from three different turbines. We tried two objective function for each turbine and objective function, the algorithms were tested 15 times to establish statistical conclusions about the spread of the solutions found.

The second problem formulation was most successfully accomplished. For this formulation, both algorithms gave a spread of the solution that was smaller than one standard deviation. Comparing the algorithms the Simulated Annealing had a more stable performance and fewer outliers compared to the Gradient Descent. However, the algorithms were not able to achieve any accurate estimation for the first problem formulation. The spread between the solution was for this formulation too large, resulting in unreliable conclussion about the machine settings.

Chapter 9

Future Work

Due to time constraints, several possible topics within this thesis work have not been researched or investigated in any detail. Firstly, we have not compared the performance of local and global optimization algorithms' performance, since we have only used the first type of methods. This could be reasearch by analysing many different starting solutions for the algorithms and investigate how the method converges. Another interesting possiblity to analyse further is to extend the line search algorithm, by adding the second derivative information, for example, using the Newton-Raphson method. Then Line Search method would might faster converge to local minimal points and not get stuck around specific neigbhourhoods for mutiple iterations. The problem is that the algorithm then becomes very computational heavy, increasing the time complexity even further. Another possible change is within the step length procedure, by increasing the number of solutions to be interpolated and the degree of the interpolation polynomial. These suggestions could increase the efficiency of the algorithm and decrease the spread for the machine settings.

Secondly, our Python code could be more efficiently structured and written. There are several time losses and memory inefficiencies within both algorithms. This is due to the operational usage of the GTperform program. Sometimes the program can not keep up with the speed of the algorithms. Having difficulties generating internal files that is need for the process. Mutiple times, the program needs to wait for the files to be created in order to continue the algorithm procedure. If we don not stop and wait, the GTperform would crash, which could crash our algorithms. The GTperform creates a bottlenecks due to is implementation. This problem have not probably been adressed, as the process of finding the machine settings have been done manual by the engineers.

Thirdly, the statistical inference needs to be extended to many more turbines with large sample sizes.

In this thesis we have applied the algorithms to a single gas turbine from each location. It would be interesting to analyze the best solutions for a group of turbines within similar geographical areas and technical constraints. The reason for this is that many companies have multiple gas turbines connected to the same electrical network to fulfil their energy demands. However, analyzing multiple turbines with GTperform would be highly time-consuming, since we have to reset and reconfigure the complete GTperform program for each turbine and machine settings. This might be possible, but not within the 10 minutes time limit. We also woud need to implement and code a higher structual program in Python that would managed the complete process. This would give further knowledge and understanding of the performance of a unique type of turbine in a certain operational enviroment and how it differs from the general performance.

Bibliography

[1] A. Abdel-Rahman. ”A Modified Partial Quadratic Interpolation Method for Unconstrained Optimization”. In: *J. Concretee and Appliable Mathematics* 11 (1) (2013), pp. 136–146.

[2] D. Abramson, M. Krishnamoorthy, H. Dang. ”Simulated Annealing Cooling Schedules for the School Timetabling Problem”. In: *Asia-Pacific Journal of Operational Research* 16 (1) (1997), pp. 1–22.

[3] J. Lundgren, M. Rönnqvist, P. Värbrand. *Optimeringslära.* Vol. Upplag 3:5. Studentlitteratur AB.

[4] J. Nocedal, S.J. Wright. *Numerical Optimization.* Springer, 2006.

[5] M. Acosta, M. Andres, Y. Sun, M. Gerard, S. Kumara. ”Algorithm Selection for Black-Box Continuous Optimization Problems: A Survey on Methods and hallenges”. In: *Information Sciences,* 317 (2015), pp. 224–245.

[6] J.R. Rice. ”The Algorithm Selection Problem”. In: *Advances in Computers* 15 (1976), pp. 65–118.

[7] S. Alizamir, S. Rebennack, P. Pardalos. ”Improving the Neighborhood Selection Strategy in Simulated Annealing Using the Optimal Stopping Problem”. In: *Simulated Annealing.* IntechOpen, 2008. Chap. 18.

[8] S. Ledesma, J. Avina-Cervantes, R. Sanchez. *Practical Considerations for Simulated Annealing Implementation.* IntechOpen, 2008.

[9] B. Walid. ”Computing the Initial Temperature of Simulated Annealing”. In: *Computational Optimization and Applications* 29 (2004), pp. 369–385.

[10] S. Vaske. ”Implementation and Analysis of a GPA Method Usign Optimization Techniques on a Indstrial Gas turbine.” MA thesis. Chalmers University of Technology, 2000.

[11] Y. Ya-Xiang. "A Review of Trust Region Algorithms for Optimization". In: *Proceedings of the Fourth International Congress on Industrial and Applied Mathematics ICM99* (Sept. 1999).