Validation of a simulator for an engine power plant

Master's Thesis in Computer Engineering

Author: Jens Lassus

Supervisor: Jerker Björkqvist

Faculty of Science and Engineering

Åbo Akademi University

2024

Abstract

Simulation software is used extensively in many industries today because it is usually faster and cheaper to develop and test new products this way, since there is no need for a physical prototype. In some cases, it is infeasible to build a physical version just for testing. This could be because it is prohibitively expensive, or because testing over a vast time span is required. Simulation software solves these issues, since it is relatively quick and cheap to reconfigure. Additionally, simulations can be run faster than real time. All these factors contribute to faster and cheaper prototyping.

One scenario in which simulation is very valuable is when building an engine power plant. It has become increasingly important to be able to accurately predict the performance of a power plant already during the sales process, as it can directly affect the revenue of the customer. As this is a very competitive market, a manufacturer can gain an advantage by committing to better performance guarantees than their competitors, but this is risky unless the performance can be accurately predicted. This risk can be mitigated if the manufacturer has access to reliable simulation software. Looking at existing installations is not a complete solution, because engine power plants are purpose-built for their application, and engines perform differently depending on ambient conditions, such as air temperature, air pressure, and humidity, as well as load, fuel methane number, and gas pressure.

One potential concern about simulating the real world is the accuracy of the simulator. If the results of the simulations are used to make important decisions, something about the accuracy of the simulator should be known. The purpose of this thesis is to develop software to speed up the process of evaluating the accuracy of a particular piece of simulation software used for simulating engine power plants. The accuracy can be evaluated by comparing the simulated values against values recorded at a real installation. This feedback from the real world back into the simulator can be thought of as a kind of *digital twin*, the concept of which is explained in section 3.3.

Because the dataset used in this thesis is limited to a single power plant, it is not possible to fully evaluate the simulator against the full range of power plants that it is capable of simulating. However, the software produced as a part of this thesis should make future validation against other datasets faster. Additionally, even with this

limited dataset it is possible to simulate and compare those performance parameters that may vary within one installation, such as the ambient and fuel conditions mentioned above. Over a longer period of time, it is also possible to study the effects aging has on performance.

The process of manually entering each sensor reading into the simulator would be very tedious and prone to human error. For this reason, the practical part of this thesis consists of developing a piece of software that automates the process of feeding input parameters into the simulator, running the simulation, then extracting the results and comparing them to the corresponding real readings from the sensors. The benefit of this is that it makes it easy to test the simulator against datasets from other power plants, as well as comparing different versions of the simulator against each other using the same dataset. Without this piece of software, this process would be very tedious, which would prevent large datasets from being assessed. Another benefit is that it enables plant operators to see if their power plant is behaving as expected. Anomalous readings can be detected by comparing them to their simulated counterparts. The challenge with developing the software is that the simulator was built in Microsoft Excel, with some core parts written in C++, and it was never intended for this use case. Although the causes of inaccuracies in the simulator are discussed briefly, this is not the focus of this thesis.

Table of Contents

1. Introduction

During the development of a product, it is often useful to know how it would perform in a particular configuration, before a physical product is built. In some cases, physical models can be built for testing, but in more complex cases it is often easier to simulate the product using software. This is especially true when power plants are concerned. Power plants are complex systems that are expensive and time consuming to build. In addition, every power plant is its separate project with its own unique requirements and operating environment, which means that often no data exists for how a plant will perform in a specific configuration.

In the past, the primary customers for engine power plants were industries generating power for their own use, to reduce the amount of energy that had to be bought from the power grid. Efficiency was important in terms of fuel savings, but would only affect the cost of production, not the revenue directly. Today, the dominating customer category is grid-scale utility customers, whose core business is selling energy to the power grid. To maximize their revenue, it is in these customers' interest to bid as many kilowatt hours as possible to the energy market. More accurate predictions for performance allows for tighter tolerances, which means that the customer can bid closer to their maximum generating capacity.

Lowering tolerances increases risk. The consequences of not fulfilling the commitments towards the energy market would hit the customer's profitability directly. This is a risk that needs to be mitigated. A common way of doing so is by transferring this risk to the supplier of the power plant in the form of monetary penalties. If the performance parameters guaranteed in the supply contract are not met at the site acceptance test, considerable penalties are materialized, hitting the profitability of the supplier. Consequently, the supplier also needs to mitigate this risk. As the consequences are fixed, and the risk cannot be transferred, the supplier must aim to lower the probability of the risk materializing. The supplier also does not want to promise less than they can deliver, as this could cause them to lose a contract to competition. If nothing else is available, estimates about performance can be made based on previous projects and known characteristics of each component. However, this is tedious manual work that is susceptible to human errors. For these reasons, an accurate and configurable power plant simulator can be an efficient and effective tool for risk management during the sales process.

Based on theory and physical laws, the simulator can calculate how a theoretical power plant would perform in various configurations and environments. This simulation software can also be referred to as a *model*. One drawback with any type of model is that it is not perfectly accurate. It is interesting to know the accuracy of the model, because if the accuracy is unknown, no confidence can be placed in any assumptions made based on simulated data. The process of determining the accuracy of the model is called model *validation*. It is not possible to prove absolute validity of a model. It is only possible to invalidate, or fail to invalidate. A more attainable goal is a high degree of *face validity*, which means that the model appears to be an accurate representation of the system from all outward indications [1].

In this thesis, the accuracy of a piece of simulation software for engine power plants is assessed. The motivation is that by knowing the accuracy of the simulated data, more confidence can be placed in it. This reduces the risk for the supplier of the power plant, for the reasons outlined above.

To assess the accuracy of the simulator, data recorded from a real engine power plant under known conditions is used. Some parameters, for instance load and ambient temperature, are used as input values for the simulator. Other parameters, such as efficiency and exhaust gas temperature, are compared to the corresponding simulated values. The accuracy of the simulation is determined by how close the simulated values are to the recorded values. When doing this for large datasets, this process would be tedious to do by hand. For this reason, the practical part of this thesis focuses on automating the process of importing the data into the simulator interface, converting units, running the simulation, and finally exporting the results into a file. The intentions are to modify the simulator as little as possible, which means using the existing interface that was made in Microsoft Excel.

2. Background

Evaluating the performance of an engine power plant for a particular application is not as simple as looking at the catalogue values. The performance of an internal combustion engine is dependent on many different parameters, some of which cannot be controlled by the operator – instead they depend on ambient conditions, such as temperature, humidity, and pressure. For a customer to be able to somewhat fairly compare the performance between products from different manufacturers, the parameters affecting the performance need to be the same. To enable this, there are internationally accepted standards. One such standard is the ISO 3046 [2], which sets the standard reference conditions at which the performance is stated (unless otherwise stated by the manufacturer).

In a laboratory, such conditions could in theory be produced. However, considering for example the huge amount of airflow required, conditioning the temperature to meet the standard reference conditions may not be viable in practice. Therefore, there is also a need for international standards providing methods for translating performance measured at one set of conditions to those expected at standard reference conditions. This is also provided by ISO 3046.

Engines manufactured for the automotive industry and similar may be sold as off-the-shelf, free-standing products, according to catalogue values. This would make little sense for applications where the equipment is used in utility-scale powerplants. The performance needs to be further translated from catalogue values, i.e. standard reference conditions, to site conditions. This lets the customer make a true evaluation of the expected performance on their installation site, in order to compare different offers.

The simulator that will be evaluated in this thesis was originally created in the 1980s by an employee working as a sales representative in a different time zone than the main office. During the sales process, some calculations needed to be done. These calculations used to be done manually by experts located at the main office. Because of time zone differences, this process was rather slow. To speed up this process, a relatively simple tool for doing these calculations was created in Microsoft Excel. This initial tool was based on the ISO 3046 standard, as well as the company's own performance manuals. The performance manuals describe cooling systems, exhaust gas temperatures and flow rates, as well as corrections for various factors. Some calculations in the simulator are based on physical laws, while others are based on internal research. One example of internal research is the research on condensation in the intercooler.

At the time, only diesel engines were of interest. As gas engines were developed, the need arose to simulate those as well. As the approximations made for diesel engines in ISO 3046 do not fit today's gas engines too well, the standard allows for the manufacturers to use substitute reference conditions to better simulate the performance. Another example where the standard permits manufacturers to utilize their own specifications is the internal mechanical resistance of an engine. For these reasons, the performance manuals of a manufacturer may differ slightly from the ISO 3046 standard.

2.1. Motivation for using a simulator

Conducting a project is always associated with some risk. A risk is the chance that some event that negatively affects the project occurs. This is why *risk management* is an important part of project management. Risk management is the process of identifying the possible events that could affect the project, and assessing the probability of them occurring, as well as how they would affect the project [3]. In 2001, a study designed to answer which tool provides the greatest benefit for the project risk management process was conducted. The highest-ranking tool in the study was simulation [4]. The second and third highest ranking tools were responsibility assignment and risk impact assessment, respectively.

2.2. Motivation for assessing simulator accuracy

By assessing the accuracy of the simulator, the accuracy of the performance manuals is also assessed, since the simulator is based on them. Other benefits include reducing contractual risks and verifying that the plant is operated as intended.

2.2.1. Verifying performance manuals

The performance manuals themselves are not necessarily perfect, for several reasons. One reason is that the simulation software used during development, GT-POWER [5], is more tailored towards small engines, such as those found in a regular car. Additionally, the prototype engines used for testing and for creating the performance manuals may not perfectly represent an engine that is delivered to a customer. Prototype engines often have fewer cylinders than the final product. Furthermore, engine-to-engine variations exist due to component tolerances. It is difficult to know if a particular engine is a good representation of the average engine. The software created alongside this thesis could help verify the correctness of the performance manuals, since the simulator is based on them.

2.2.2. Contractual risks

Another reason to verify the accuracy of the simulator is that it can help reduce contractual risks. When selling a power plant, certain guarantees are given about minimum efficiencies, for example. The performance of the power plant is dependent on external factors, such as temperature, humidity, and pressure. If not carefully considered, the seller may end up promising something that they cannot deliver. Using a simulator, the power plant can be simulated in its target environment before a contract is signed, significantly reducing contractual risks.

The contracts may in some cases include guarantees about performance degradation over time. However, in addition to normal wear and tear, which can be mitigated quite well by regular maintenance, unexpected external factors could also contribute to performance degradation. For example, if the area is very dusty, that dust may build up in the radiators, insulating them and reducing their efficiency. Performance degradation of a real power plant could be measured using the simulator by running it based on the current input parameters and comparing the sensor readings of interest to their theoretical values, calculated by the simulator. Assuming that the simulator is otherwise very accurate, the difference between the theoretical values and the actual values would be due to performance degradation.

From a customer's point of view, knowing the degradation ahead of time is important because it will affect the payback time of the power plant. It does not matter how efficient the new power plant is, if it degrades to below average after a few years. The customer may be willing to pay a higher initial price for a stronger performance guarantee. The seller takes a risk by giving this guarantee, but by having an accurate simulator, this risk is reduced.

2.2.3. Verifying operating configuration

An accurate simulator can be used to compare certain parameters from a real power plant against their calculated values, in order to find cases where the power plant is configured differently than expected. For example, if the exhaust gas temperature differs from the simulated value, this could indicate that the power plant is running in a different configuration than intended.

2.2.4. Verifying using large datasets

Automation enables developers to test the simulator on large datasets. This provides greater coverage of all possible input parameters, which means they can be more confident that it is accurate. A model can never be proven to be correct, but the more different cases it can accurately predict, the more confidence can be placed in it.

3. Theory

This chapter covers some background information and theory about simulation, model validation, digital twins, big data, black box testing, and the ISO 3046 standard.

3.1. Simulation

Simulation and models are closely related and are sometimes used somewhat interchangeably. Shannon defines simulation as “the process of designing a model of a real system and conducting experiments with this model for the purpose of either understanding the behavior of the system and/or evaluating various strategies for the operation of the system” [6].

A model is an approximation of a system, phenomenon, or process. It is simpler than the real system, while still replicating its most important properties. A model can be physical, mathematical, or logical [7, p. 117], [8]. The purpose of a model is to predict how the real system will behave, without the need to interact with the real system. The real system does not even need to exist. For example, by constructing a model before the real system is built, the performance of the proposed system can be evaluated, reducing the risk of failure to meet specifications. The system can also be simulated faster than real-time, allowing the effects of running the system in certain configurations for long periods of time to be analyzed much more quickly and cheaply than using the real system. The model is easy to reconfigure and experiment with, allowing questions concerning what would happen in various scenarios and in different configurations to be answered. In many cases, this would be impossible or impractical to do using a real system. This would necessitate the use of a model. Another advantage of using a model is that training can be done in a safe environment where neither people nor equipment can be harmed, even if operator mistakes are made.

Simulation also has some disadvantages. Unlike an analytical approach, a simulation is run with a set of input parameters and returns a set of output parameters, rather than solved for the optimal set of input parameters. Using this approach, it is impossible to tell if there is a different set of input parameters that would yield a better result, without trying all of them.

3.2. Model validation

One potential problem with models is validity. An inaccurate model is not useful. Depending on the level of inaccuracy, it might not reduce project risks at all. At the very least, engineers and project managers should be aware of the limitations of the model, so that not too much confidence is placed in it. The purpose of this thesis is to speed up the process of assessing the validity of an existing model for engine power plants. The method used in this thesis is to run the simulation using input parameters from a real power plant under known conditions, and then compare the simulated parameters to the corresponding readings from the real power plant.

It should be noted that, similar to a scientific theory, a model can only be proved to be invalid. Its validity cannot be proved. However, by sufficiently testing the model using varied input data and comparing it to known output data, one can be reasonably certain that the model is accurate in most circumstances [1].

3.3. Digital Twins

Complex physical systems, such as power plants, are outfitted with a host of different sensors, continuously recording everything of interest about the system, for example temperature, pressure, voltage, and flow sensors. Some of this data is required for the system to operate, while other data is more interesting from an analytical standpoint. A digital twin takes the idea of a model a step further. While a regular model can represent a real system, after it is created, it is disconnected from reality. A digital twin continuously takes in new data from the real system, enabling real-time simulation. IBM provides a definition of what a digital twin is:

“The digital twin is the virtual representation of a physical object or system across its life-cycle. It uses real-time data and other sources to enable learning, reasoning, and dynamically recalibrating for improved decision making.” [9]

In other words, a digital twin aims to mirror a physical system as closely as possible. Digital twins could, for example, be used to find early indications of a developing fault. An additional benefit is that engineers can diagnose potential problems from anywhere in the world, from the comfort of their desk. This could be especially useful if a problem develops on a ship out at sea, for example.

The concept of a digital twin was initially formulated in 2002 by Michael Grieves at the University of Michigan [10, p. 93], but in practice, digital twins have been used by NASA since the 1960s [11].

3.4. Big Data

Companies today are increasingly data driven, meaning that they record, process, and analyze large quantities of data, for the purpose of improving their business in various ways. When the datasets become extremely large and complex, they are called *Big Data*. What size dataset is considered big constantly changes, due to constantly growing computing capabilities. One way of defining big data is to say that it is data that is too large for traditional relational databases to handle. The variety of data, as well as the rate that new data is received, also play a role [12].

The data used in this thesis is not quite of that scale; it comes from a single power plant and only a relatively small number of parameters are used. Nonetheless, one could imagine that in the future, all sensor data from all power plants are available from a central point. At that point, it could probably be considered big data.

3.5. GAIA-X

GAIA-X is a European initiative for creating a secure, centralized system where data and services can be made available [13]. The goal of this initiative is to promote innovation, by ensuring transparency and interoperability, while remaining secure. GAIA-X also guarantees data sovereignty, meaning that users decide where their data is stored, who may process it, and for what purpose.

Examples of use cases cover many different sectors, including industry, smart living, finance, health, public sector, mobility, agriculture, energy, and geoinformation. A relevant use case for engine power plants might be the gathering and processing of sensor information from components installed at various locations across the world. In cases like these, questions arise regarding the ownership of data, and who is allowed to access it.

3.6. Black Box testing

Black box testing is a category of software testing where the tester knows nothing about the inner workings of the software [14]. All the tester knows is what output the software is supposed to return for some given input parameters. The tester can then try to give that input to the software and see what output it produces. If the output is the same as the expected output, the test is passed.

Although not quite the same, this is analogous to how the simulation software is to be verified. Inspecting the source code in order to find mistakes is not viable, since it consists of tens of thousands of lines of code, some of which are based on expert knowledge. It is also possible that some of the resources used when developing the software are inaccurate. These inaccuracies would not be caught by this method, assuming that the same resources would have to be used to verify the code. This is why the black box method is chosen. The difference is that instead of specifications, there are sensor readings from a physical version of the system to use as a reference.

3.7. ISO 3046

The simulator is based on the ISO 3046 standard [2], which specifies how to calculate corrections for power when ambient temperature, air pressure, or humidity, differ from the reference conditions. The reference conditions are T = 298 K, p = 100 kPa, humidity = 0.3.

Properties of internal combustion engines, for example the rated power, are always declared at certain reference conditions. This includes the ambient air temperature, pressure, and relative humidity. However, real-world applications may not match these conditions. The ISO 3046 standard defines formulas for calculating corrections for, among other things, power and fuel consumption, based on the actual conditions. It is a satellite standard to the ISO 15550 standard, which contains the reference conditions.

The ISO 3046 standard can also be used to calculate derating factors. Derating is the process of reducing the maximum allowed power, based on ambient conditions. Ambient conditions that reduce the maximum power are high temperature, high humidity, and low air pressure.

4. Implementation

For the practical part of this thesis, the objective is to automate the process of running the simulation on input data recorded from a real installation.

4.1. Structure

The simulator consists of a Microsoft Excel document with code written in Visual Basic for Applications (VBA), as well as some core modules written in C++. Two sheets in the Excel document are of importance for the automation software: the table sheet and the report sheet. These can be seen as the import and export sheets, respectively. In the import sheet, there is a column for each simulation case. This is where the automation software needs to insert the input data for the simulation. Similarly, there is a column for each simulation case in the output sheet. These are the columns the automation software needs to read and compare to the measured data.

Additionally, the import sheet contains design data. This data contains static information such as altitude and what auxiliary equipment is installed at the site. This data does affect the simulation, but because it is static, it is not available anywhere in the sensor data. This limitation means that the Excel document needs to be manually configured for a specific site.

Importing and exporting the data from the input and output sheet is nontrivial. The rows of the input and output sheets are labeled, but they follow a different naming system compared to the sensor data. Some required input data must be calculated from multiple parameters in the sensor data. Additionally, the simulator uses SI units, such as Kelvin and Pascal, while the sensor data uses more everyday units, such as Celsius and bar. The solution used here is to map unit conversion functions to each input parameter, where required, as well as mapping each input parameter either directly to a sensor parameter, or to a function that calculates the value based on multiple sensor values.

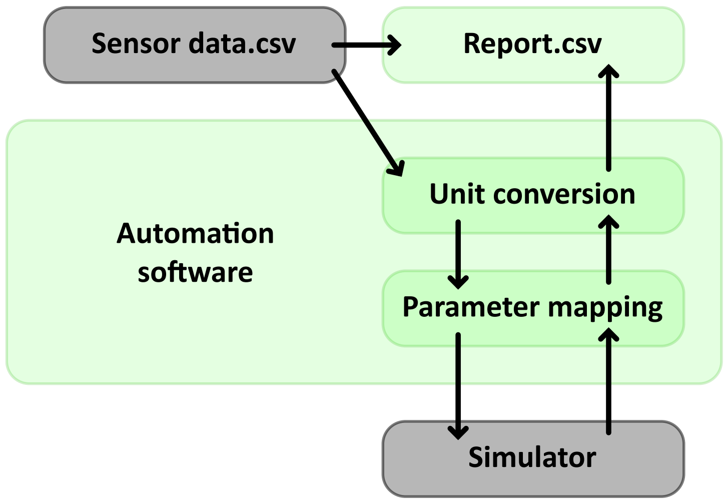


Figure 1

Figure 1 shows the flow of data through the application. The sensor data includes sensor readings from a real power plant, in CSV format. Each row represents a particular time period, and each column represents a sensor. The length of a time period is consistent within a file but can vary between files. Some sensor readings, such as ambient temperature, are used as input to the simulator, while others, for example the heat rate, are used as reference values for comparing to the output data from the simulator. The report is also a CSV file. Each row in the sensor data produces one row in the report. The report contains pairs of columns, where the first column is the calculated value from the simulator, and the second is the corresponding measured value from the power plant.

From a data flow perspective, the automation software converts units and maps parameters back and forth between the different naming schemes and units used in the sensor data and the simulator. The input must be converted to the format expected by the simulator. The output from the simulator must also be converted back to the same format as the input data, so that it can be compared easily. The sensor data and simulator components are darker to indicate that these are already existing, fixed components that the rest of the application must be built around.

4.2. Libraries

The libraries used are chosen according to the needs of the project. If reading or modifying an Excel file were enough, there would be many options for several different programming languages. Python and Openpyxl would be one possible combination. These types of libraries typically just provide an easy way to read and write the Excel file format. The advantage is that Excel does not need to be installed on the machine running the code. The problem is that simply reading or modifying the file is not enough for this project. To run the simulation, some VBA macros in the Excel file need to be invoked, which means that the code needs to interface with a running instance of Excel. This limits the number of options.

4.2.1. Microsoft Office Interoperability Library

The library chosen to interface with Excel is the Microsoft Office Interoperability library [15]. The choice of programming language falls on C#, since this is one of the supported languages, and it is familiar to the author. The Office Interoperability library enables launching Excel as a background task and provides an API for reading and modifying Excel files, as well as running macros. This is required if the goal is to keep the modifications of the simulation software to the minimum.

Microsoft Office applications have COM-based object models. Component Object Model (COM) is a standard for binary interfaces, which allows for the creation of language-independent binary software components. COM objects have well-defined interfaces, through which they communicate with other objects. The Microsoft Office Interoperability library is a COM interoperability library, which allows managed code to interact with COM objects. The important distinction between managed code and COM objects is that managed code executes inside a runtime environment, which manages its memory, while COM objects exist in unmanaged memory. The interoperability library also handles marshaling, which is the process of converting back and forth between equivalent data types, since they might differ between the COM object and the managed code.

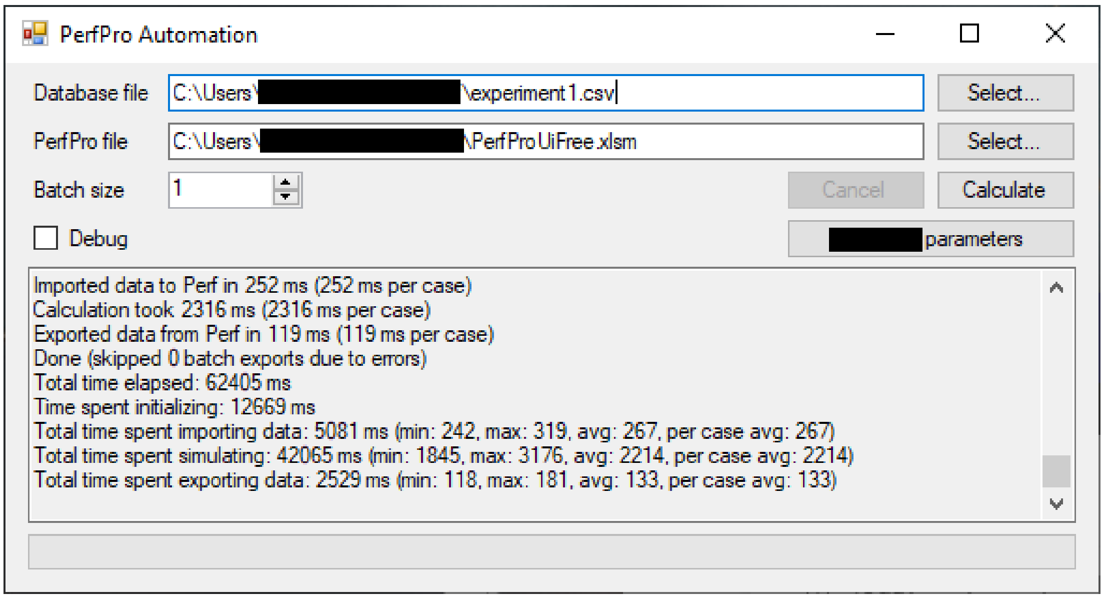
4.2.2. C#

C# (pronounced “See Sharp”) [16] is a strongly typed, general-purpose object-oriented programming language developed by Microsoft. It was chosen for this project partly due to its compatibility with the Microsoft Interoperability Library for Microsoft Excel, and partly because its syntax is very similar to other well-known languages, for example C++ and Java. For this reason, it is easy to learn for anyone familiar with those languages.

The official implementation of C# has two branches. One uses the .NET Framework and is limited to Windows only. The other uses .NET Core and is compatible with multiple platforms. The .NET Framework implementation is used in this project.

4.3. Program structure

The purpose of the software is to make it faster and easier to run many simulations. For this reason, the software is given a Graphical User Interface (GUI). This makes the software more accessible to people without technical knowledge. Figure 2 shows a screenshot of the GUI.



Database file C:\Users\[redacted]\experiment1.csv

PerfPro file C:\Users\[redacted]\PerfProUiFree.xlsm

Batch size 1

☐ Debug

[redacted] parameters

```
Imported data to Perf in 252 ms (252 ms per case)
Calculation took 2316 ms (2316 ms per case)
Exported data from Perf in 119 ms (119 ms per case)
Done (skipped 0 batch exports due to errors)
Total time elapsed: 62405 ms
Time spent initializing: 12669 ms
Total time spent importing data: 5081 ms (min: 242, max: 319, avg: 267, per case avg: 267)
Total time spent simulating: 42065 ms (min: 1845, max: 3176, avg: 2214, per case avg: 2214)
Total time spent exporting data: 2529 ms (min: 118, max: 181, avg: 133, per case avg: 133)
```

Figure 2

The source code is divided into several *classes*, which is a common structure in object-oriented programming languages. Each of these classes handles its own logical part of the problem.

4.3.1. Database

The purpose of the database class is to read the data from a file, translate the column names to names that the simulator can understand, as well as converting units. The database class is also responsible for saving the results to disk.

Initially, this class would import all the data at once into memory. In memory, the data would take up several times the space that it used on disk. This does not scale for large datasets. The final implementation uses a custom class that only keeps an index of the position of each line in the file. This way, it provides reasonably fast random access to each line of the file, without keeping them in memory. This is discussed more in section 4.5.5.

4.3.2. Simulator interface

The simulator class interfaces with the simulator, i.e. Microsoft Excel. This class has methods for importing and exporting data, as well as running the simulation. Many details about mapping the Excel sheets and understanding where data should be read from and written to are irrelevant to the rest of the program. For this reason, it is separated into its own class.

4.3.3. Handling popups

In the VBA code that is part of the simulator, popup messages are used to inform the user when the simulation is done, and about potential errors. These messages cannot be disabled without editing them out of the VBA code. Since the goal was to avoid modifications to the simulator software, and since no other modifications to the VBA code were needed, it would be unfortunate to require code modifications for such a simple issue. The only other solution to this problem was to automatically detect the popup messages and close them. One problem is that the call to the VBA code is synchronous, which means that the entire thread is waiting for the popup window to be closed. The solution was to have a separate thread that uses polling to detect when the window appears, and then click the button to let the main thread continue. Although the solution is not very elegant, one benefit compared to just disabling the popup messages in the VBA code is that error messages can now be captured and displayed in the user interface of the automation software.

4.4. Data

To accelerate the first phase of development, some simulated data was provided as input data. This was done so work could focus on how to interface with the simulator programmatically, without the need to map and convert sensor readings, since these were already fully compatible.

The actual input data used for testing the software comes from a real gas engine power plant. The data is made available through a web interface. The interface allows the user to select the desired parameters to export, the time frame, time aggregation, as well as filters, and makes this data available to download as a CSV file.

4.4.1. Time aggregation

Several choices must be made when selecting what data to use for testing and analysis. The time aggregation is important because the simulator was only made to simulate steady-state scenarios. A time aggregation of one second can be considered dynamic and does not represent the expected output from the simulator well. On the other hand, a time aggregation of one hour might hide too much of what happened during that hour. A middle-ground value of one minute was chosen, since this is a reasonably high time resolution, without including second-to-second variations in power output, for example.

4.4.2. Weather

Another factor to consider is the weather at the time when the data was recorded. The power plant will perform differently depending on the ambient temperature and humidity. Only the absolute humidity is relevant to the engine. This means that in cold weather, the engine will work with dryer air.

4.4.3. Required parameters

The simulator must know the temperature, pressure, and humidity of the air entering the engine, as well as the air surrounding the radiators. Various temperature and humidity sensors are placed around the power plant. Some are on the engine itself, while others are part of a weather station. The pressure is assumed to be constant and was calculated based on the known altitude of the plant.

The simulator must also know the composition of the fuel. While natural gas consists mainly of methane (CH4), some amount of other gases, such as ethane (C2H6) and nitrogen (N2) are also usually present [17]. This power plant has a gas analyzer, so most of these parameters are available. The other components are assumed to be zero.

4.4.4. Missing parameters

Some required input parameters, apart from the design data, are not available in the web interface. These must be assumed to be some reasonable, static value.

4.5. Performance

The speed and memory usage of the software is examined below. When referring to the existing software, i.e. the simulator itself, the term “simulator” is used. This is considered to be a black box. When referring to the new software, i.e. the software built around the simulator to automate it, the term “automation software” is used. When referring to the combination of the two, the term “automated simulator” is used.

4.5.1. Hardware

All benchmarks are run on a laptop running Windows 10, with an Intel Core i5-8365U processor and 8 gigabytes of system memory. The installed version of Microsoft Excel is Microsoft Excel for Microsoft 365 MSO (Version 2109, 32-bit). During benchmarks the charger was plugged in, so battery saving modes should not be a concern. However, a few other factors exist that could contribute to inconsistent results. The CPU has a variable frequency, and a change in temperature could affect the boost frequency of the CPU. Various unpredictable background tasks could also use resources during the benchmark and affect the results.

4.5.2. Motivation

The performance of the automated simulator is measured in how many scenarios per second it can simulate. A scenario is defined here as a set of input parameters from one power plant, aggregated on a specific time frame. The simulator is only designed to simulate steady-state scenarios, in which the power plant has been running in the same configuration long enough for all sensor readings to settle to a steady value. It is unclear how long this time span is, but since there are only a limited number of different steady-state scenarios that can occur during a day, the speed of the automated simulator is not crucial. Currently, it is not significant whether it takes ten seconds or one minute to simulate a full day’s worth of data.

However, because the simulator is only made to simulate steady-state scenarios, it could be of interest to measure how large the inaccuracies in the simulator become when run on data that has been identified as dynamic. In this case, it could become relevant to run the simulator on data with a frequency of up to one hertz. Another reason for wanting to keep up with a data frequency of one hertz would be if this project was developed into a digital twin in the future. Depending on the input data, the simulator may or may not be able to keep up with this data rate, as shown in the results below. This limits what the current implementation can be used for in the future, but it is sufficient for the purposes of this project.

4.5.3. Benchmarking the automation software

Initially, the main goal of benchmarking was to see how much time the automation software adds on top of the simulation time, as well as to establish whether the batch size influences the overall execution time of the automated simulator. The batch size is the number of scenarios that are imported into the simulator at a time. Batch sizes of 1, 10, and 100 were tested. To reduce inaccuracies, the tests were run 10 times each and the fastest time for each batch size was recorded. The same dataset was used for all batch sizes. The dataset consists of 100 rows (scenarios) of data with a time aggregation of one minute. The results are shown in Figure 3. All benchmarked datasets use data recorded when the power plant had been close to maximum power for at least an hour, since it is known that lower power scenarios take longer to simulate, and to ensure that the system is as close to a steady state as possible.

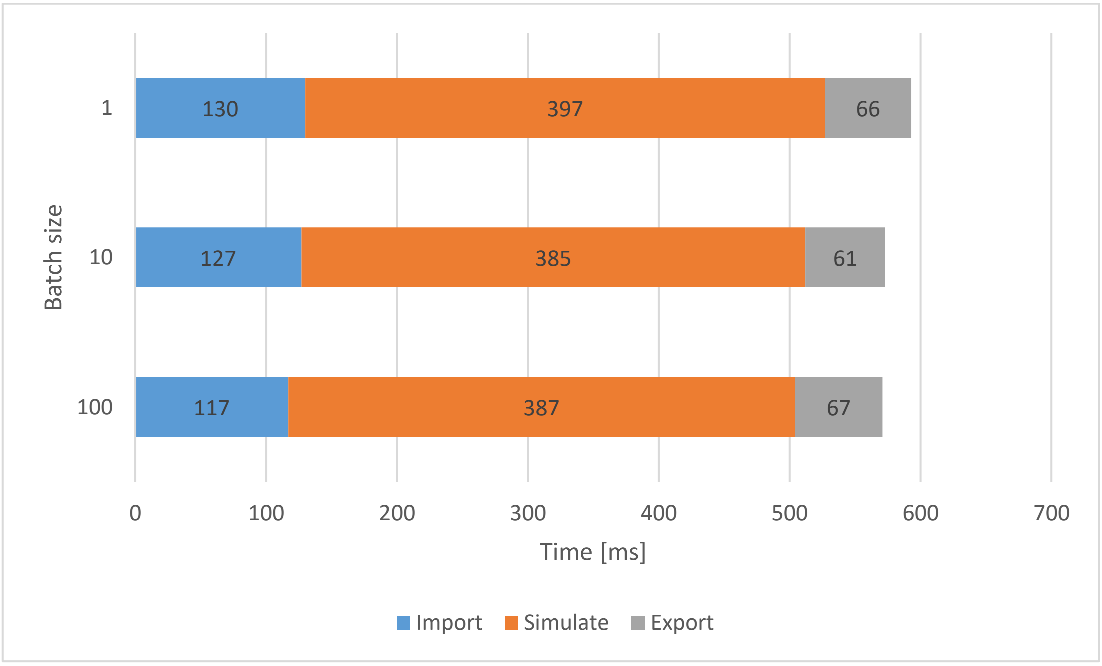


Figure 3

As expected, Figure 3 shows that most of the execution time is used to simulate. Still, the simulation software adds around 50% on top of the simulation time. Ways to optimize these function calls could be investigated in the future. Figure 3 also shows that there is no significant advantage to increasing the batch size. Increasing it from 1 to 10 only reduces the overall execution time per scenario by 20ms, or around 3%. Increasing it from 10 to 100 only reduces the execution time by 2ms, which is within the margin of error. The option to change the batch size is still left in case this changes in the future. It should be noted that a large batch size in theory increases memory usage, but this is an implementation detail of Microsoft Excel. In testing, no significant change in memory usage was noted when comparing a batch size of 1 to a batch size of 100. The memory usage was examined using Windows Task Manager while the simulator was running.

One advantage of keeping the batch size small is that it makes the software seem more responsive to the user. This is because the call to the simulator code is synchronous and does not provide any feedback before the simulations of all scenarios are done. If the batch size is small, the call returns often, and the progress can be reported back to the user. Another advantage of small batch sizes is that in case the simulator encounters an error, the entire batch is discarded. Cancelling the operation is also faster with a smaller batch size, since the automation software waits for the currently simulating batch to be done before terminating.

4.5.4. Benchmarking the simulator

As benchmarking progressed, it became evident that some scenarios were significantly slower to simulate than others. What these scenarios seemed to have in common was the time of the year the data was recorded. Data from July, when temperatures could reach at least 35 °C, was significantly slower to simulate. For this reason, ambient temperature and humidity were suspected to have an impact on the execution time. Two separate tests were run, in order to isolate the effects of changes in temperature from the effects of changes in humidity.

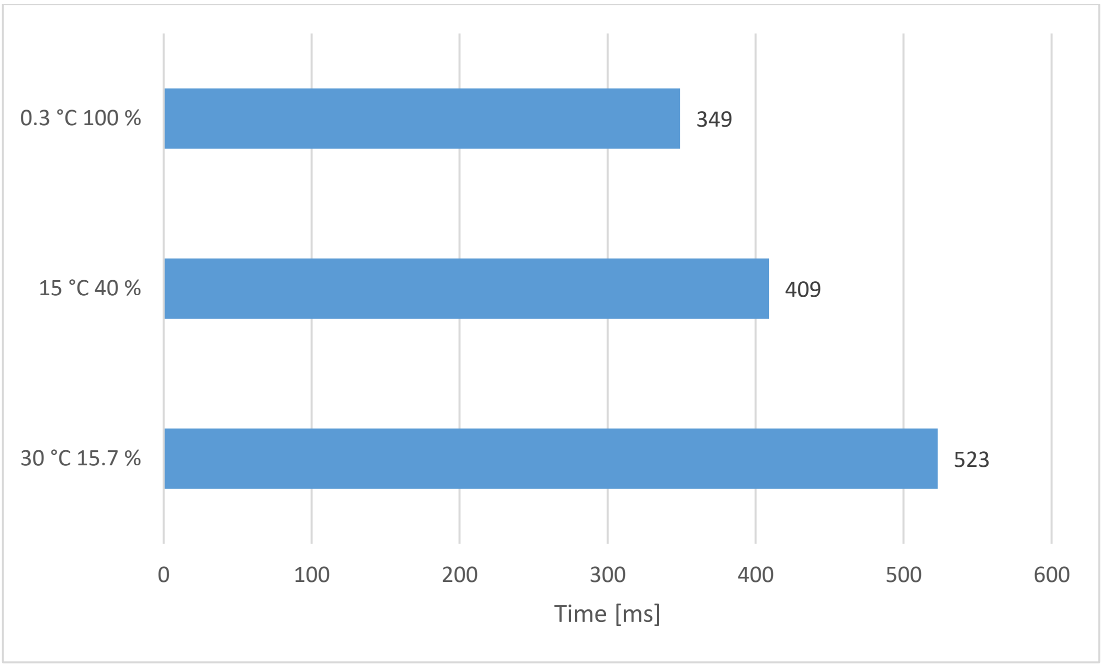


Figure 4

Figure 4 shows how the time to simulate increases as the temperature rises. The first scenario, with a temperature of 0.3 °C, is an unmodified scenario recorded in January. The other two are versions of the same scenario, where only the temperature and relative humidity have been modified. Since only the absolute humidity is relevant to the engine, the relative humidity had to be recalculated to achieve a constant absolute humidity.

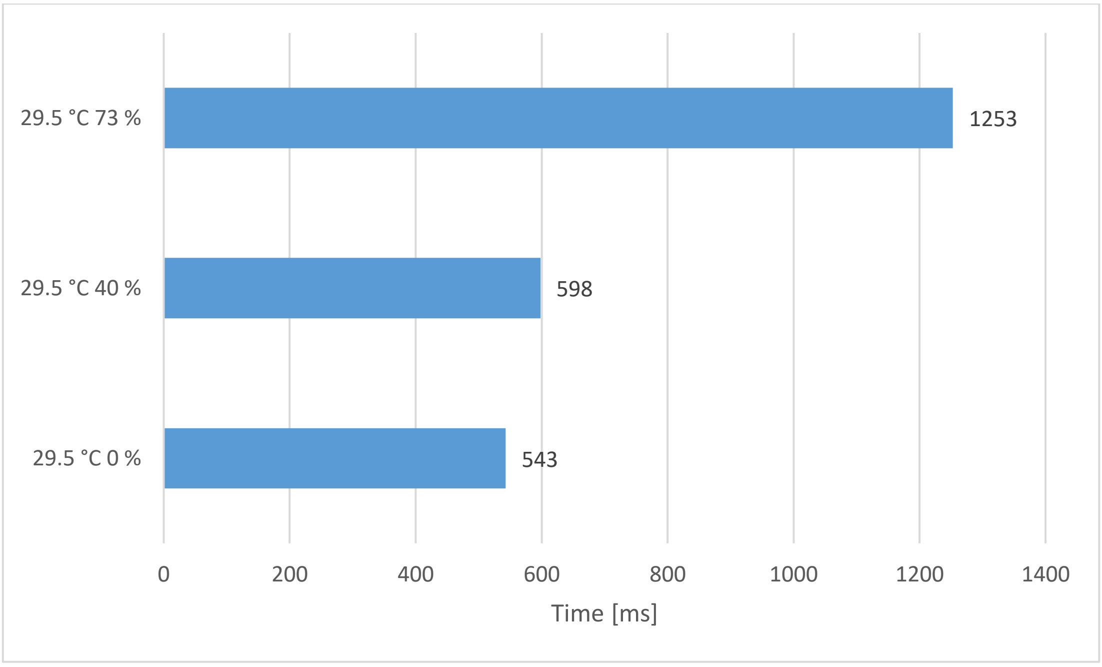


Figure 5

Figure 5 shows how the time to simulate decreases as the humidity decreases. The first scenario, with a relative humidity of 73%, is an unmodified scenario recorded in July. The other two are modified versions of the same scenario, where only the relative humidity has been reduced.

These results show that both the temperature and humidity affect the time to simulate, but an absolute humidity above a certain level seems to have an especially large impact. To see how much impact this has on real data, a dataset from January was compared to a dataset from July. The average execution time for each scenario is shown in Figure 6 and Figure 7, for January and July respectively.

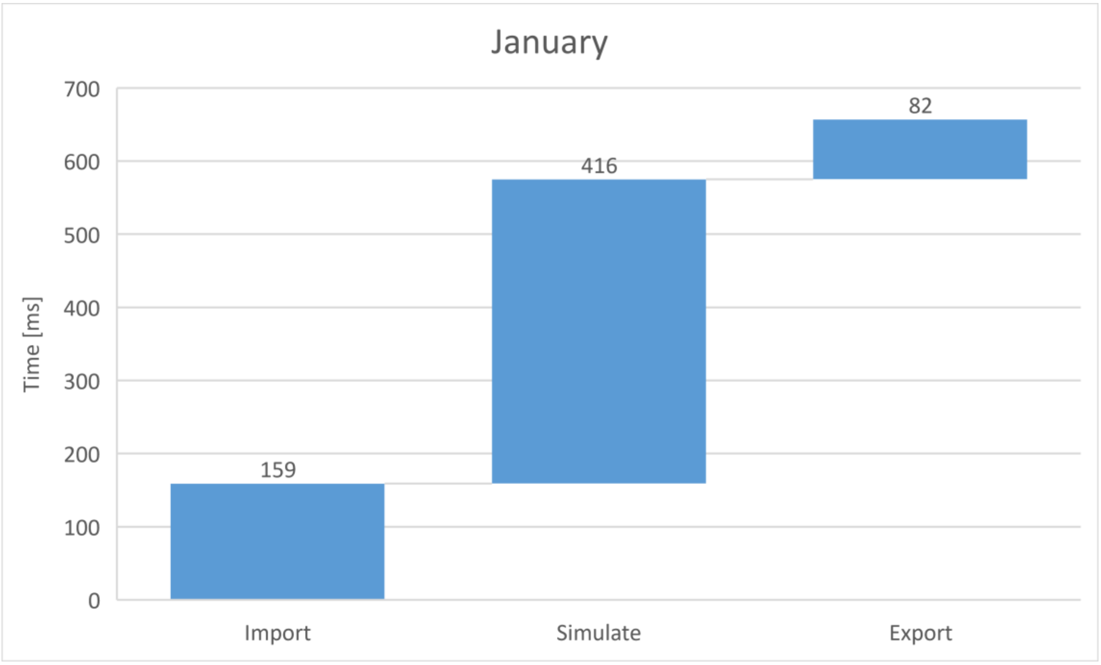


Figure 6

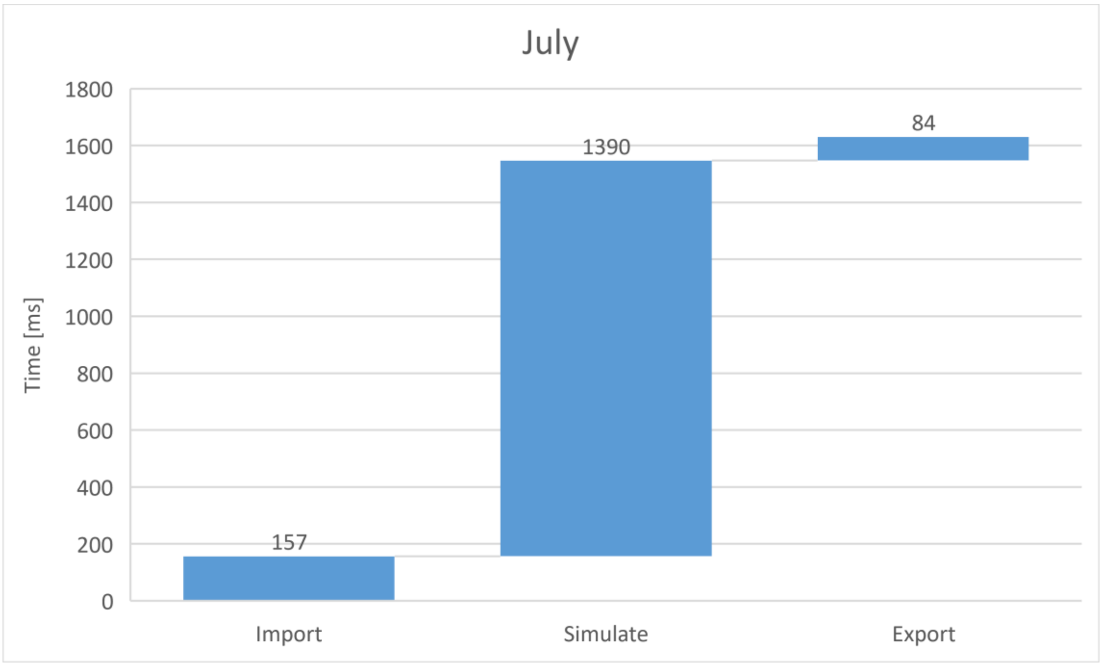


Figure 7

As expected, comparing Figure 6 (January) to Figure 7 (July) reveals that the simulation takes significantly longer in July, while the import and export times stay the same.

4.5.5. Memory optimization

The memory optimization in the automation software is worth considering, more specifically, the way the input data is handled. Initially, all rows of the input data were imported at once before the simulation started. The data was imported into a list of dictionaries, with a dictionary for each row. This provided fast random access to each row, and each data point in the row was accessible by its column name. Additionally, any filtering on a specific column becomes easier if the data is already parsed into rows and columns. This way, accurate progress can be reported to the user, since the total number of rows, after filtering, is known. This worked well for input data sizes up to a few megabytes. One megabyte of data in CSV format can hold roughly 3000 rows. Each row represents one scenario. Three thousand scenarios is equivalent to 125 days worth of data, if aggregated for each hour, and would take roughly 30 minutes to simulate in the best case scenario. Given this information, it seems unlikely that datasets much larger than this would be used with the automation software in its current form. This would indicate that optimizing for memory usage is not necessary. However, in case the simulator was optimized to handle larger datasets in the future, this solution scales poorly. It was noted that when loading a CSV file of around 300 megabytes, the total memory usage of the automation software was roughly 1.6 gigabytes.

The improved data handler is divided into two classes. The first one, called FileLineRandomAccess, provides random access to each line of a text file, without keeping it in memory. It does this by scanning the entire file at first and keeping an index of the position and length of each line. The reason that the length is needed is that it also allows for a filter to be specified, which means that not all lines are necessarily included. If all lines were always included, the length could be found from the position of the next line. Including the length also allows for dynamic handling of both Unix and Windows-style line breaks. Windows-style line breaks consist of two characters, a carriage return and a newline character, whereas line breaks in Linux consist of only a newline character. Reducing the line length by one extra character if a carriage return is detected ensures that both Windows-style and Unix-style line breaks are excluded from the end of the line.

The second class, called CsvReader, builds on top of FileLineRandomAccess to enable random access to the rows of a CSV file. Additionally, the columns in the row can be indexed either by numeric index or by column name. It is also possible to filter the rows on a column. All of this is transparent to the rest of the program. The result is that the data is as easy to access as before, but this solution scales much better when memory usage is concerned. In theory, this solution is slightly slower since the file must be read twice, and since each row is accessed from disk rather than memory. In practice, however, this difference is negligible compared to the execution time of the simulator. Testing revealed that reading one row of data from the disk using the improved method always takes less than 2ms, but it should be noted that the input file was accessed from a solid-state drive (SSD). If accessed from a traditional spinning hard drive, this could take slightly more time. However, even if it were to take ten times longer, it could still be considered insignificant.

5. Results

The strategy for determining how closely the simulator resembles reality is as follows. First, snapshots of readings from a real power plant are taken. Simulations based on the data from these snapshots are then run. Finally, key output parameters from these simulations are compared to the corresponding readings in the snapshots. The two key output parameters selected for comparison are heat rate and exhaust gas temperature. Additionally, the effects of changes in key external factors on these output parameters are examined. These external factors are ambient temperature, ambient humidity, and load. Other parameters that may affect the results are kept as static as possible. Nothing prevents someone from comparing additional parameters, but these are some of the most interesting, for reasons explained in their corresponding sections (5.1 and 5.2).

Parameters other than the ones being compared are kept as static as possible. This limits the number of available data points. Another option would be to take as many data points as there is time to run through the simulator and compare the corresponding measured and calculated values. The problem with this approach is that without context, it is difficult to know if the snapshot was taken during a dynamic or steady-state scenario.

All graphs in this chapter follow the same pattern. They come in pairs, where the first graph shows the calculated and corresponding measured value as absolute values for the parameter in question. The numeric values of the y-axis scale have been intentionally removed for data protection reasons. The second graph shows the measured value relative to the calculated value, in percentages. It is also worth noting that, although the dots in the graphs are joined by lines, this is only to make the graphs more readable. The values are not necessarily in chronological order. The values on the x-axis are also not necessarily evenly spaced.

5.1. Heat rate

Heat rate is the inverse of efficiency. The unit of heat rate here is kilojoules per kilowatt-hours (kJ/kWh). This tells how many kilojoules worth of fuel that must be consumed to generate one kilowatt-hour of energy. A theoretical perfect 100% efficiency would be 3600 kJ/kWh, since 3600 kJ = 1 kWh. The generated energy can be measured as mechanical energy, or as electrical energy, either at the generator terminals or after converting to high voltage. Both converting mechanical energy to electrical energy, and converting low voltage to high voltage, are subject to inefficiencies. Measuring the heat rate after these processes results in a higher measured heat rate (lower efficiency). In this case, the energy is measured as electrical energy at the generator terminals. The heat rate is important, because it tells how much fuel the customer will have to use to generate the electricity that they sell. In other words, it heavily affects their profits. The formula to convert heat rate in kJ/kWh to efficiency in percent is

$$\frac{3600\ \mathrm{kJ/kWh}}{x\ \mathrm{kJ/kWh}} \times 100\%$$

The graph in Figure 8 shows the calculated and measured heat rate at different ambient temperatures as absolute values. The graph in Figure 9 shows the same data, except that it is represented as the measured heat rate relative to the calculated heat rate, in percentages. A value above zero means that the measured heat rate is above the calculated heat rate.

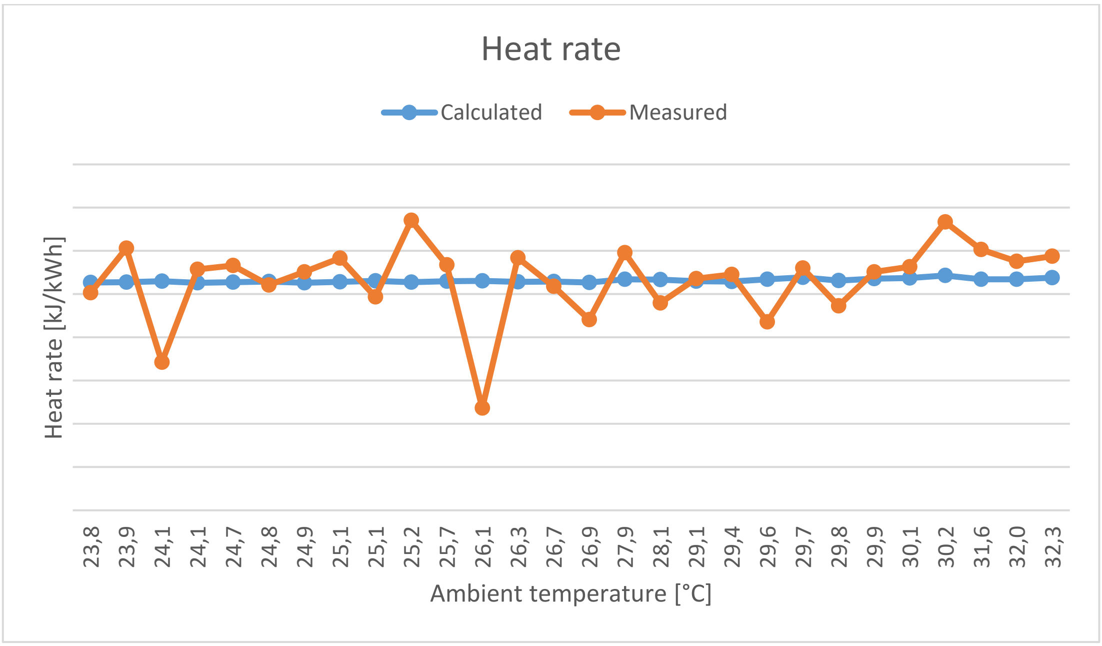


Figure 8

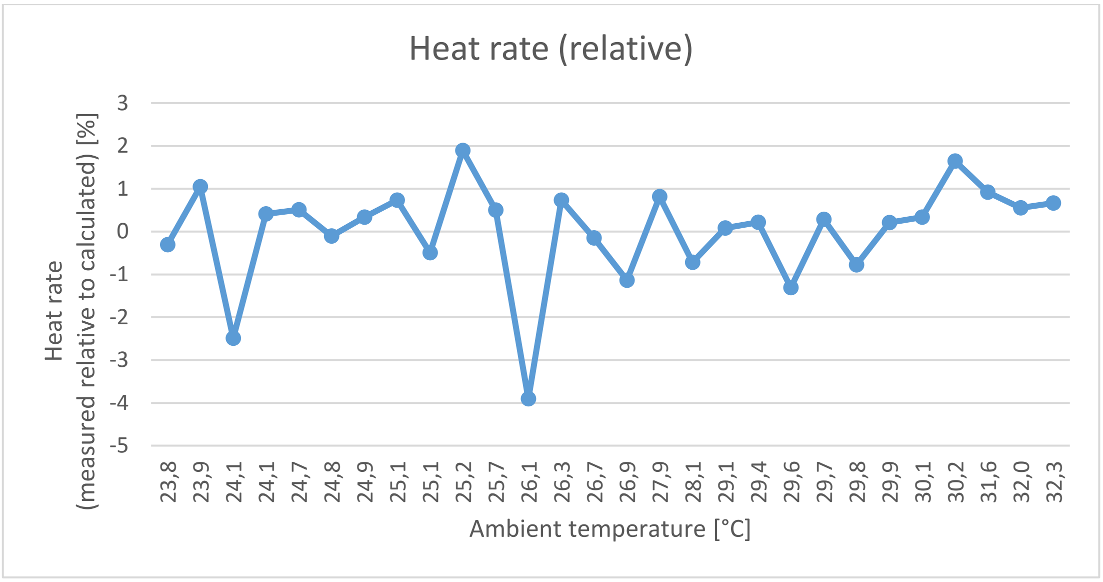


Figure 9

The data point at 26.1 °C in Figure 8 and Figure 9 shows that the measured heat rate is significantly lower than the calculated heat rate. There is also a significant difference at 24.1 °C. It is unlikely that these differences have anything to do with these specific temperatures. What is more likely is that something has caused the power plant to deviate from the expected heat rate. This could be because the power plant was in the process of ramping up or down the load, or due to a factor that was not accounted for in the simulator. It is worth noting that some required input parameters could not easily be extracted from the data and were given a default value in the automation software. If one or several of these parameters deviated significantly for this data point, but were not accounted for, it could explain why the deviation was not present in the calculated data. Another possible reason is that a sensor could be reporting inaccurate values.

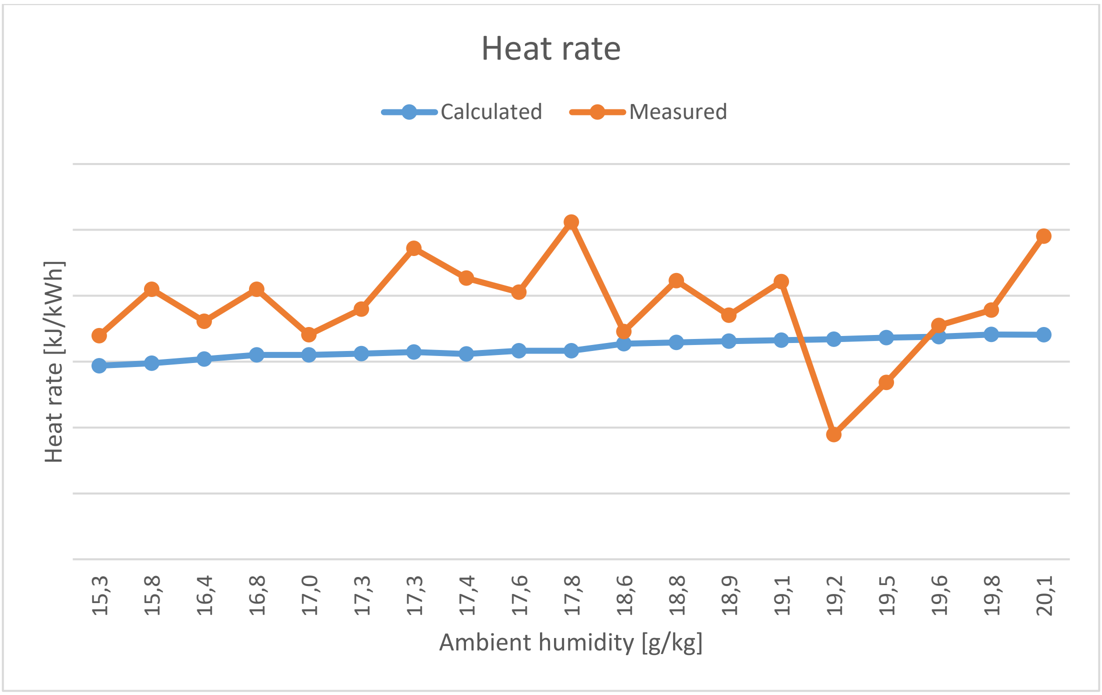


Figure 10

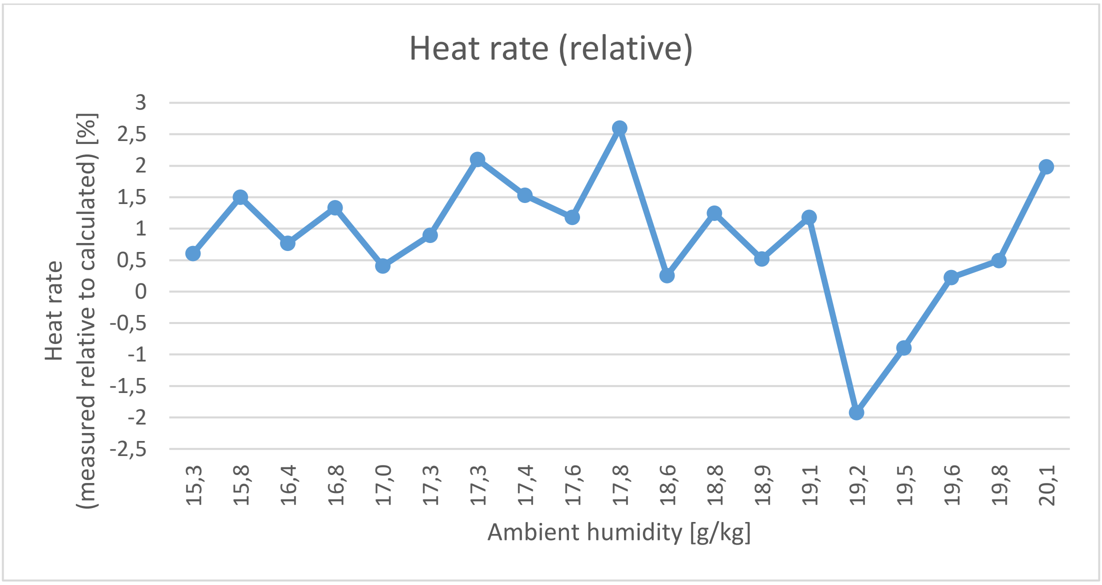


Figure 11

Figure 10 and Figure 11 show the heat rate at different ambient humidity levels, while the ambient temperature is kept between 25 °C and 26 °C. When referring to ambient humidity, a relative measurement ranging from 0% to 100% is commonly used. However, only the absolute humidity, measured in grams of water per kilogram of air, is relevant to the power plant. The problem with this is that the ambient temperature affects the humidity and sets a maximum absolute humidity. This means that limiting the temperature to a small range also greatly limits the range of the humidity. A temperature range of 25 °C to 26 °C was chosen because it includes humidity readings ranging from 15.21 g/kg to 21.26 g/kg.

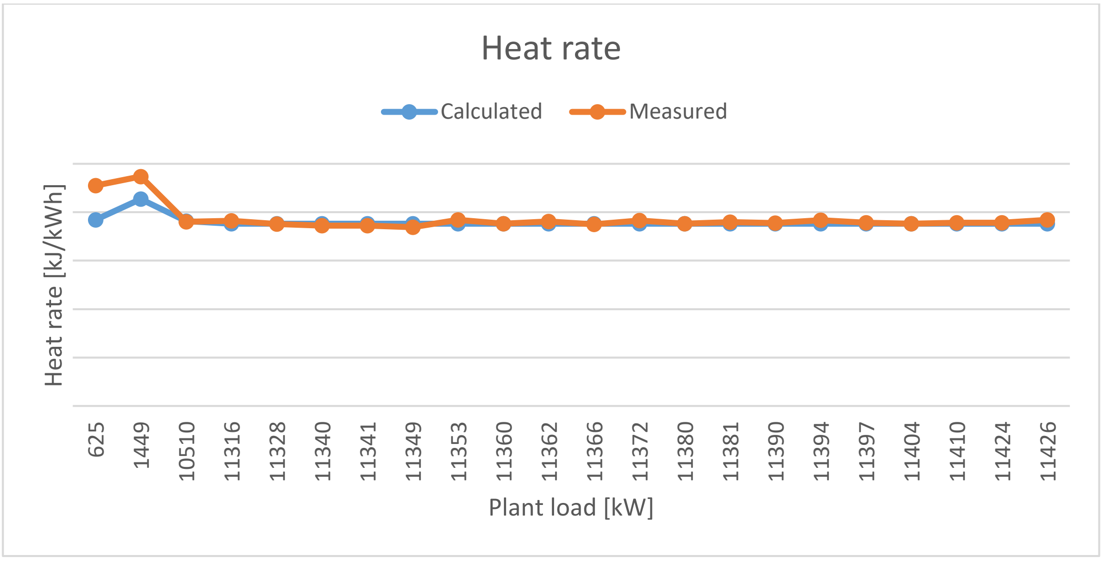


Figure 12

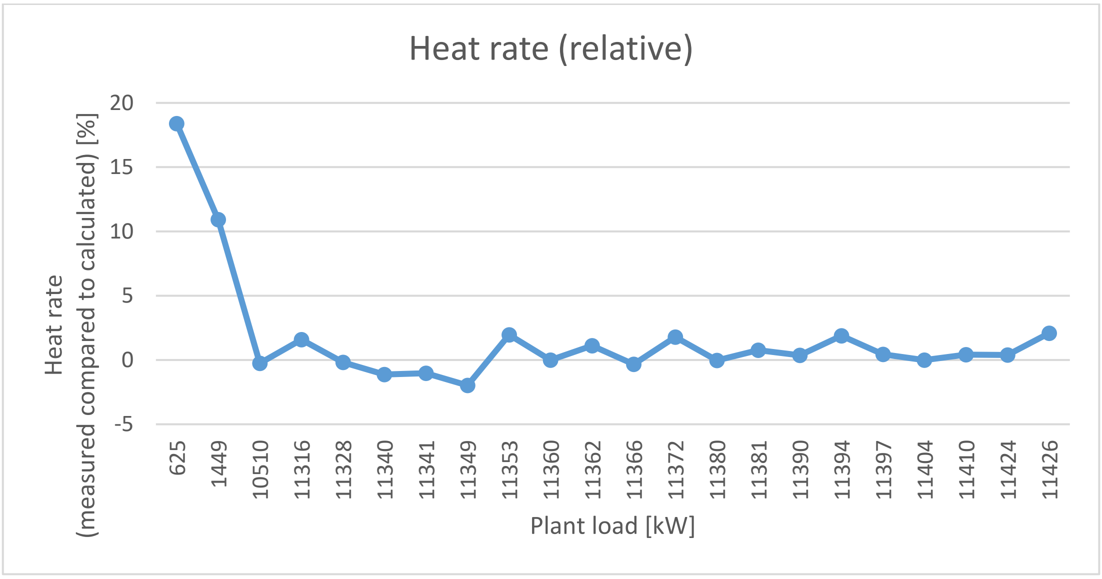


Figure 13

Figure 12 and Figure 13 show the heat rate at different loads. One observation that was made is that the simulator is quite unstable and inaccurate at very low load levels.

5.2. Exhaust gas temperature

The exhaust gas temperature is interesting to examine because it also gives an indication of the efficiency. A higher exhaust gas temperature means a lower efficiency. The temperature is also interesting from a heat recovery perspective. Heat from the cooling system as well as the exhaust may be used for district heating, as well as for producing steam for industrial use, or for driving a steam turbine.

In Figure 14 and Figure 15, the exhaust gas temperature is shown at different ambient temperatures.

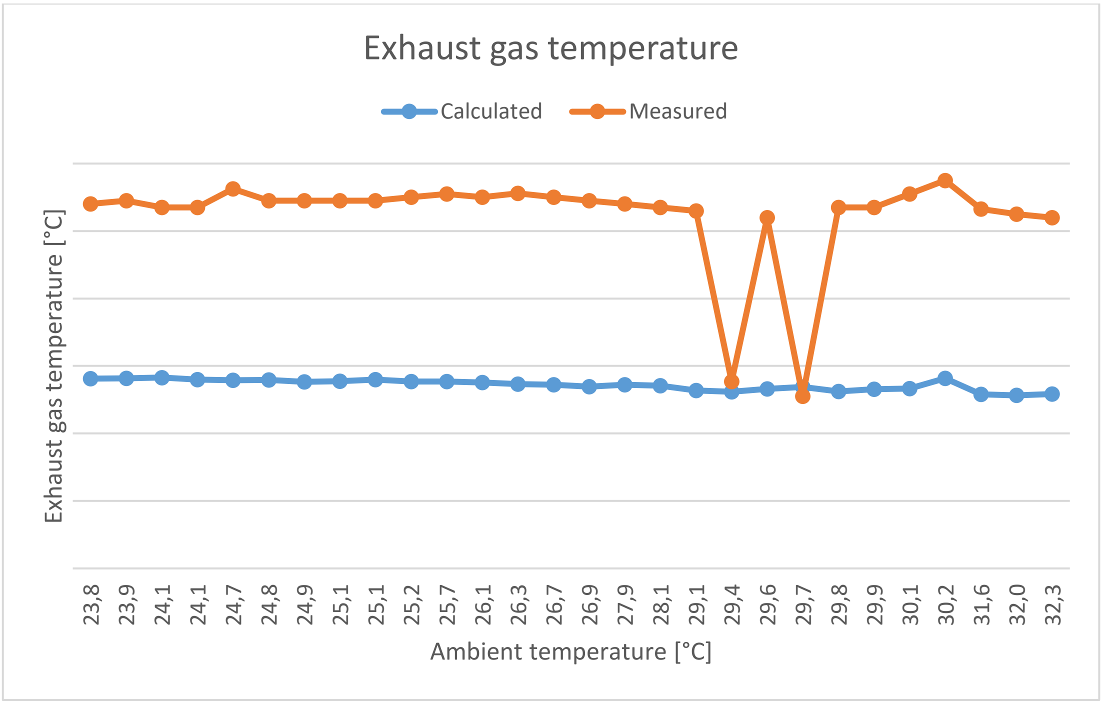


Figure 14

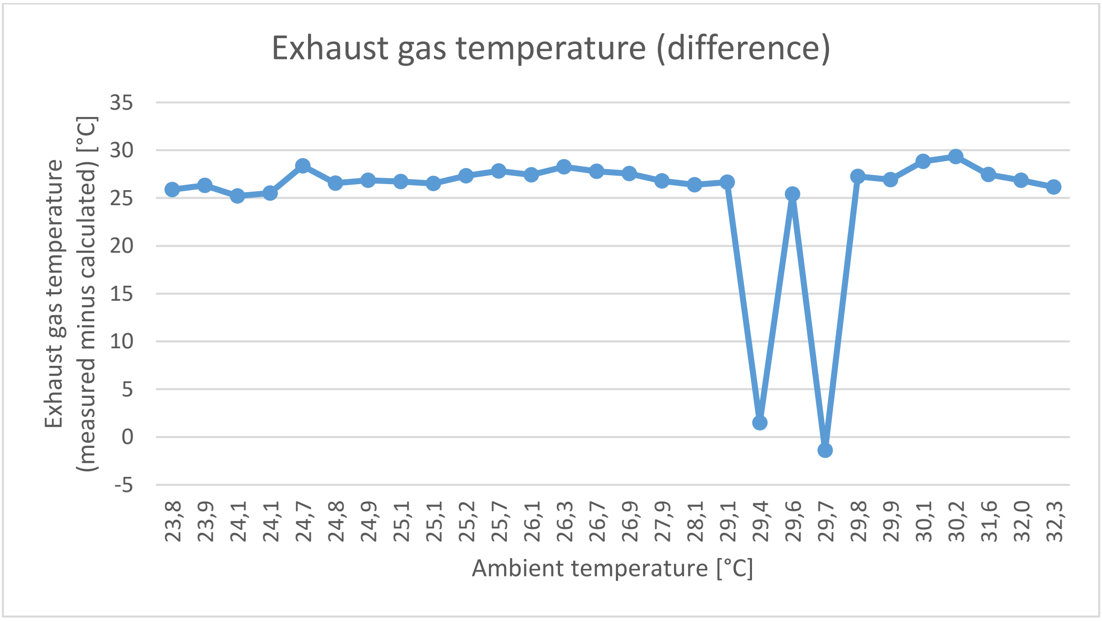


Figure 15

The exhaust temperature should ideally be known within 15 °C. As Figure 15 shows, the measured temperature is consistently between 25 and 30 °C higher than the calculated temperature, ignoring the two outliers. However, if the calculated temperatures were simply increased by around 25 °C, they would be accurate to within 5 °C in this limited dataset, again ignoring the outliers.

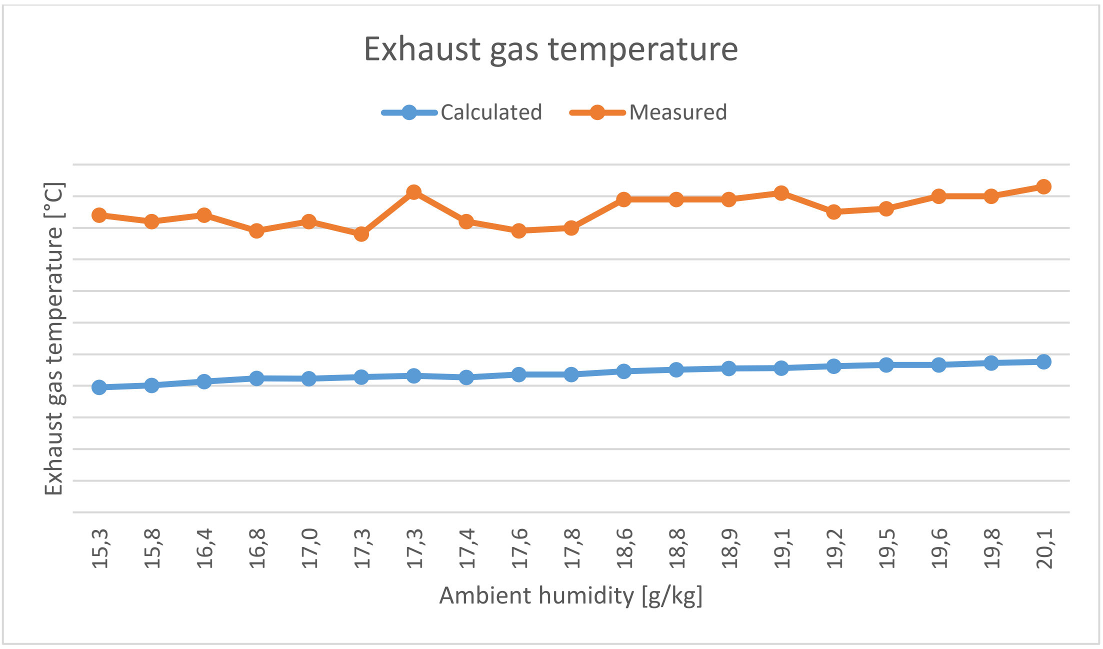


Figure 16

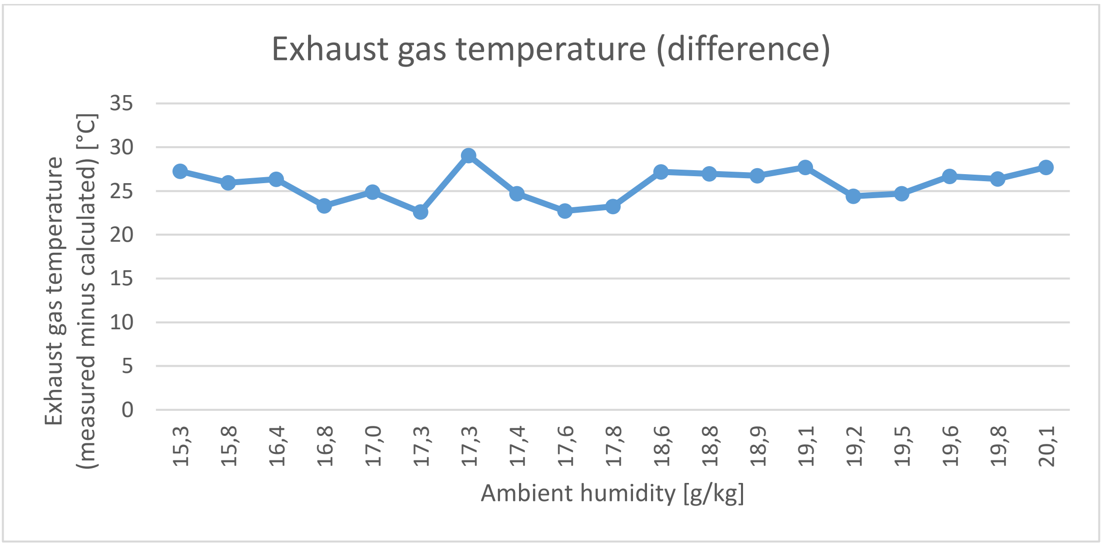


Figure 17

Figure 16 and Figure 17 show the exhaust gas temperature at different ambient humidity levels. As was the case with the ambient temperature, the measured values are consistently around 25 °C higher than the calculated ones.

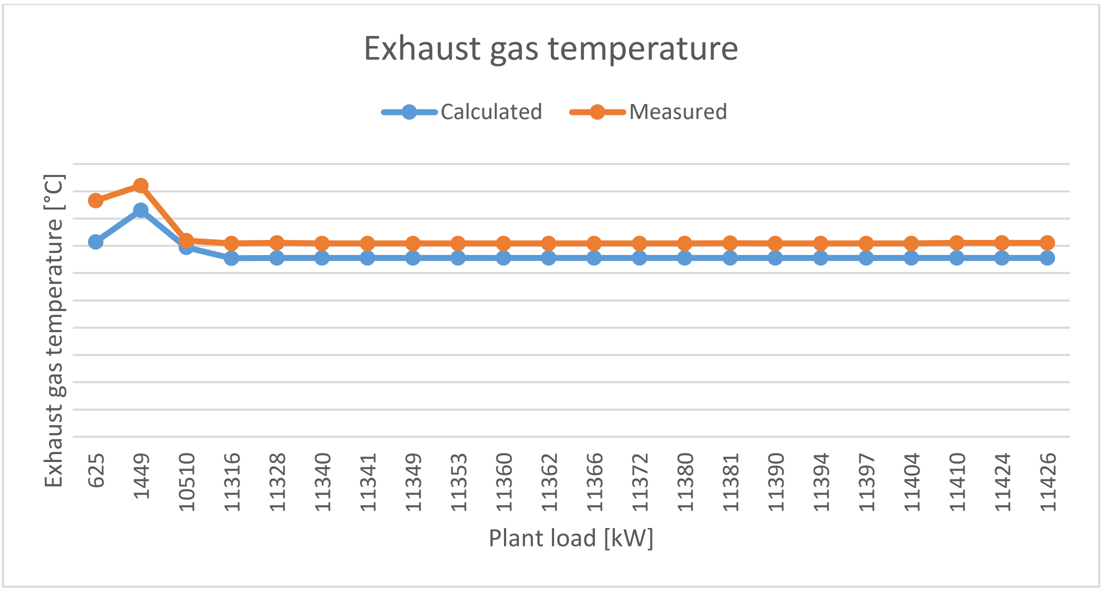


Figure 18

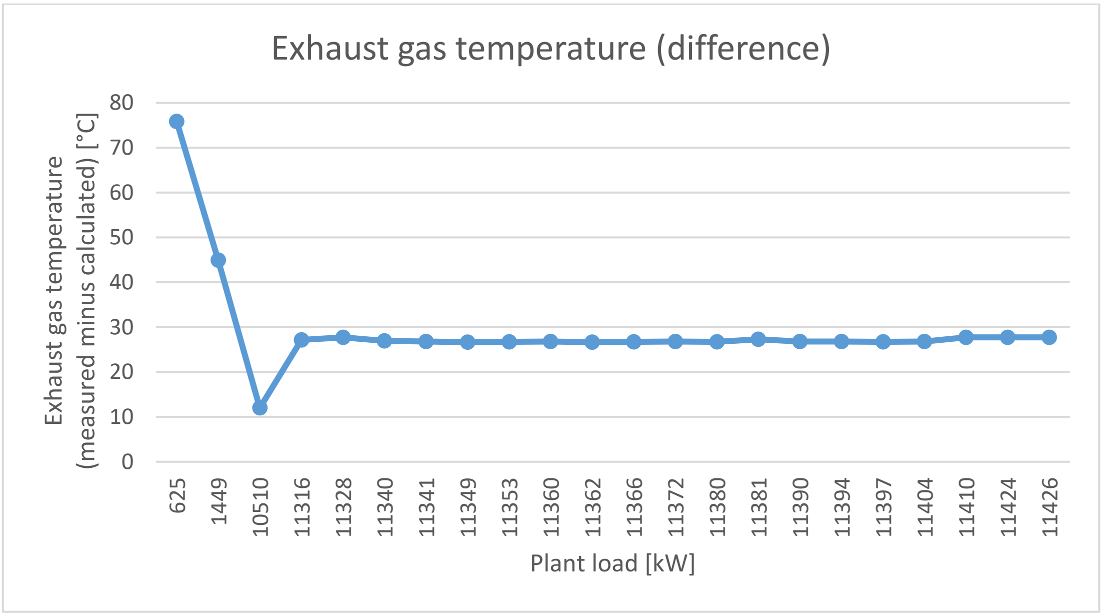


Figure 19

Figure 18 and Figure 19 show the exhaust gas temperature at different load levels. Again, the measured temperatures are consistently higher, ignoring the outliers caused by the instability of the simulator at low load levels.

The inaccuracies shown in the above graphs could be the result of many different uncertainties. For instance, the sensors measuring the Lower Heating Value (energy content of the gas) and the gas flow sensors have some uncertainty. The coolant flow rate in the cooling system is also unknown.

6. Conclusion

The results in the previous chapter show that most of the calculated parameters are reasonably close to their measured counterparts. However, with the simulated readings for the heat rate sometimes deviating with up to 2.5% compared to the measured values, it is not enough for commercial use. The exhaust gas temperatures seem to have a systematic error that causes the simulated values to be around 25 °C lower than the measured values. This may only need a simple offset to be accurate. It could also indicate that the power plant is configured differently compared to the simulator. The graphs showing various parameter comparisons at different plant loads show that the simulator is the most accurate close to the maximum load. This makes sense, since this is how the plants are intended to be operated. At this stage it is difficult to know whether the rest of the inaccuracies can be attributed to the simulator being inaccurate, or something else. Some input parameters were chosen to be static, since no corresponding parameters were found in the real data. If chosen incorrectly, this could lead to inaccuracies in the simulated data even if the simulator itself is accurate.

Ultimately, the graphs shown in the previous chapter are not the most important part of this thesis. This is because they only show a few selected parameters, and because the dataset is taken from a single power plant. It is more important that in the future, similar comparisons can be made more easily due to the automation software that was produced as a part of this thesis work. Once a satisfactory level of accuracy has been achieved for a range of different applications and external factors, the simulator can be trusted even more during the sales process. Additionally, the automated simulator can be used to verify that existing installations are running as expected.

Some areas of the software are still worth improving. To simplify further automation, a command-line API could be implemented. The graphs in the previous chapter were created manually. A web-based tool for visualizing these graphs could be created. This tool could accept CSV files from real installations, run the simulator, then automatically render the graphs. The ultimate version of this tool could have access to the source data through an API and show the graphs in real time.

Summary in Swedish

Validering av en simulator för ett motorkraftverk

Mjukvara för simulering används i stor utsträckning inom många industrier idag. Orsaken är att det ofta är snabbare och billigare att utveckla och testa nya produkter på det här sättet, eftersom man inte behöver tillverka fysiska prototyper. I vissa fall är det orimligt att bygga fysiska versioner endast för att testa, antingen för att det är orimligt dyrt, eller för att man har behov av att testa produkten över en lång tidsperiod. Simuleringsmjukvara löser dessa problem eftersom den är relativt snabb och billig att konfigurera om. Dessutom kan simulatorer köras snabbare än realtid. Alla dessa faktorer bidrar till snabbare och billigare utveckling.

Ett exempel på ett fall där simulering är väldigt värdefullt är då ett motorkraftverk ska byggas. Det har blivit allt viktigare att kunna förutsäga prestandan hos ett kraftverk redan under försäljningsprocessen, eftersom den kan ha direkt inverkan på kundens intäkter. Eftersom det förekommer hård konkurrens på den här marknaden kan en tillverkare uppnå ett övertag genom att ge bättre prestandagarantier än konkurrenterna, men det här är riskfyllt om man inte kan förutsäga prestandan med tillräcklig noggrannhet. Den här risken kan hanteras om tillverkaren har tillgång till en pålitlig simulator. Att titta på existerande kraftverk är ingen fullständig lösning, eftersom kraftverken specialbyggs för deras ändamål. Dessutom påverkas prestandan hos motorerna av yttre faktorer som luftens temperatur, fuktighet och tryck, samt kraftverkets belastning och bränslets metantal och gastryck.

En nackdel med simulatorer är att de inte stämmer helt överens med verkligheten. Ifall resultaten från simulatorn används för att ta viktiga beslut borde man veta något om dess noggrannhet. Avsikten med denna avhandling är att utveckla mjukvara för att snabba upp processen att utvärdera precisionen hos en viss mjukvara som används för att simulera motorkraftverk. Detta görs genom att jämföra de simulerade värdena mot värden som uppmätts i ett riktigt kraftverk. Att mata in återkoppling från ett verkligt kraftverk tillbaka in i simulatorn kan liknas vid en digital tvilling (digital twin), som är en sorts digital representation av ett system genom hela dess livscykel.

I denna avhandling är det inte möjligt att utvärdera simulatorn mot alla sorters kraftverk som den kan simulera, eftersom sensordatan är begränsad till ett kraftverk.

Mjukvaran som utvecklas inom ramarna för denna avhandling borde ändå snabba upp framtida utvärderingar av simulatorn mot data från andra kraftverk. Dessutom kan man även med detta begränsade dataset jämföra de variabler som kan variera inom ett kraftverk, exempelvis väder och bränsleparametrar som nämndes tidigare. Över längre tidsperioder kunde man också studera hur kraftverkets åldrande påverkar dess prestanda.

Att manuellt skriva in alla sensorvärden i simulatorn för hundratals eller tusentals fall skulle ta alltför länge och risken för mänskliga misstag är stor. Därför består den praktiska delen av denna avhandling av att utveckla mjukvara som automatiserar processen att mata in parametrar i simulatorn, köra den, och sedan läsa resultatet och jämföra det med verkliga mätvärden från kraftverket. Detta gör det dessutom lättare att göra om undersökningen med andra kraftverk, samt att jämföra olika versioner av simulatorn med samma dataset. Utan denna mjukvara skulle det här vara extremt tidskrävande, vilket skulle förhindra utvärdering av större dataset. En annan fördel är att det blir lättare att upptäcka om ett kraftverk inte beter sig som förväntat. Avvikande värden kan upptäckas genom att jämföra dessa mot motsvarande värde från simulatorn. Källor till eventuella fel i simulatorn diskuteras kort, men detta är inte fokus för denna avhandling.

Simulatorn baserar sig på ISO 3046-standarden. Standarden beskriver hur värden som effekt och bränsleförbrukning, som bestäms vid ett visst lufttryck, lufttemperatur och luftfuktighet, kan räknas om så att de gäller vid andra förhållanden. Standarden kan också användas för att beräkna en så kallad derating-faktor, som berättar hur mycket man måste sänka maxeffekten på motorn under vissa förhållanden. Omgivningsfaktorer som sänker maxeffekten är exempelvis hög temperatur, hög luftfuktighet eller hög höjd (lågt lufttryck).

Förutom ISO 3046 används också interna prestandamanualer som baserar sig på simuleringar i GT-POWER, samt laboratorietest av motorer. Problemet med detta är att GT-POWER är designat för att simulera förbränningsmotorer av mindre skala, exempelvis de som finns i vanliga personbilar. Motorerna som testas i laboratorier har ofta mindre antal cylindrar än de som levereras till kraftverk, vilket också kan ge upphov till fel.

Simulatorn består av ett Microsoft Excel-dokument, där en stor del av koden består av VBA-kod, samt några kärnmoduler skrivna i C++. De två Excel-arken av intresse kan för detta ändamål ses som inmatningsarket och utmatningsarket. I båda dessa ark finns en rad för varje mätpunkt. Sensordatan från kraftverket består av en CSV-fil, där varje rad motsvarar en tidpunkt, och varje kolumn motsvarar en mätpunkt. Automatiseringen som byggts inom ramen för detta arbete tar datan från kraftverket, en rad åt gången, och matar in sensorvärdena på motsvarande rad i inmatningsarket. Svårigheten i detta steg är att para ihop rätt kolumn i sensordatan med rätt rad i inmatningsarket, eftersom dessa har olika namngivningssystem. För en del mätpunkter måste också en enhetskonvertering göras. Därefter körs simulatorn. Sedan måste mätpunkterna från utmatningsarket konverteras tillbaka på motsvarande sätt, eftersom målet är att rapporten som skapas ska använda samma format som datan från kraftverket.

Resultaten av utvärderingen visar att simulatorn stämmer hyfsat bra överens med de uppmätta värdena, men eftersom exempelvis verkningsgraden har ett fel på upp till 2,5 procent så är den inte tillräckligt exakt för kommersiell användning. Avgastemperaturerna verkar också ha ett systematiskt fel som gör att de simulerade temperaturerna är ca 25 °C lägre än de verkliga temperaturerna. Dessa fel behöver dock inte bero på fel i simulatorn. En möjlig förklaring är att kraftverket är konfigurerat annorlunda än simulatorn. Dessutom gavs några parametrar statiska värden eftersom några motsvarande mätdata inte hittades. Om dessa gavs felaktiga värden kunde det ge upphov till fel i de simulerade värdena. Inexakta sensorvärden för exempelvis gasflödet eller gasens energiinnehåll (LHV) kunde också påverka resultatet. Det viktigaste är ändå att mjukvaran som utvecklades som en del av denna avhandling gör att det är lättare att undersöka vad dessa fel kan bero på.

References

[1] ProModel, “Model Validation.” [Online]. Accessed: Jul. 14, 2021. Available: https://www.promodel.com/onlinehelp/promodel/80/C-03%20-%20Model%20Validation.htm

[2] *Reciprocating internal combustion engines*, ISO Standard 3046, May 2002.

[3] D. V. Pym, “Risk Management,” *PM Netw.*, pp. 33–36, Aug. 1987.

[4] T. Raz and E. Michael, “Use and benefits of tools for project risk management,” *Int. J. Proj. Manag.*, vol. 19, no. 1, pp. 9–17, Jan. 2001, doi: 10.1016/S0263-7863(99)00036-8.

[5] “GT-POWER.” Gamma Technologies, LLC.

[6] R. E. Shannon, “Introduction to Simulation,” in *Proceedings of the 24th Conference on Winter Simulation*, in WSC ’92. New York, NY, USA: Association for Computing Machinery, 1992, pp. 65–73. doi: 10.1145/167293.167302.

[7] B. Lightsey, *Systems engineering fundamentals*. 2001.

[8] A. Maria, “Introduction to modeling and simulation,” in *Proceedings of the 29th conference on Winter simulation - WSC ’97*, Atlanta, Georgia, United States: ACM Press, 1997, pp. 7–13. doi: 10.1145/268437.268440.

[9] M. Armstrong, “Cheat sheet: What is Digital Twin?” [Online]. Accessed: Feb. 28, 2021. Available: https://www.ibm.com/blogs/internet-of-things/iot-cheat-sheet-digital-twin/

[10] M. Grieves and J. Vickers, “Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems,” 2017, pp. 85–113. doi: 10.1007/978-3-319-38756-7_4.

[11] B. D. Allen, “Digital Twins and Living Models at NASA,” Nov. 01, 2021. [Online]. Accessed: Mar. 11, 2024. Available: https://ntrs.nasa.gov/citations/20210023699

[12] Oracle, “What Is Big Data?” [Online]. Accessed: Mar. 11, 2024. Available: https://www.oracle.com/big-data/what-is-big-data/

[13] Gaia-X, “About Gaia-X.” [Online]. Accessed: Mar. 11, 2024. Available: https://gaia-x.eu/what-is-gaia-x/about-gaia-x/

[14] S. Nidhra, “Black Box and White Box Testing Techniques - A Literature Review,” *Int. J. Embed. Syst. Appl.*, vol. 2, pp. 29–50, Jun. 2012, doi: 10.5121/ijesa.2012.2204.

[15] Microsoft, “Interoperability Overview.” [Online]. Accessed: Mar. 11, 2024. Available: https://learn.microsoft.com/en-us/dotnet/csharp/advanced-topics/interop/

[16] Microsoft, “A tour of the C# language.” [Online]. Accessed: Mar. 11, 2024. Available: https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/

[17] Britannica, “Composition and properties of natural gas.” [Online]. Accessed: Apr. 02, 2023. Available: https://www.britannica.com/science/natural-gas/Composition-and-properties-of-natural-gas