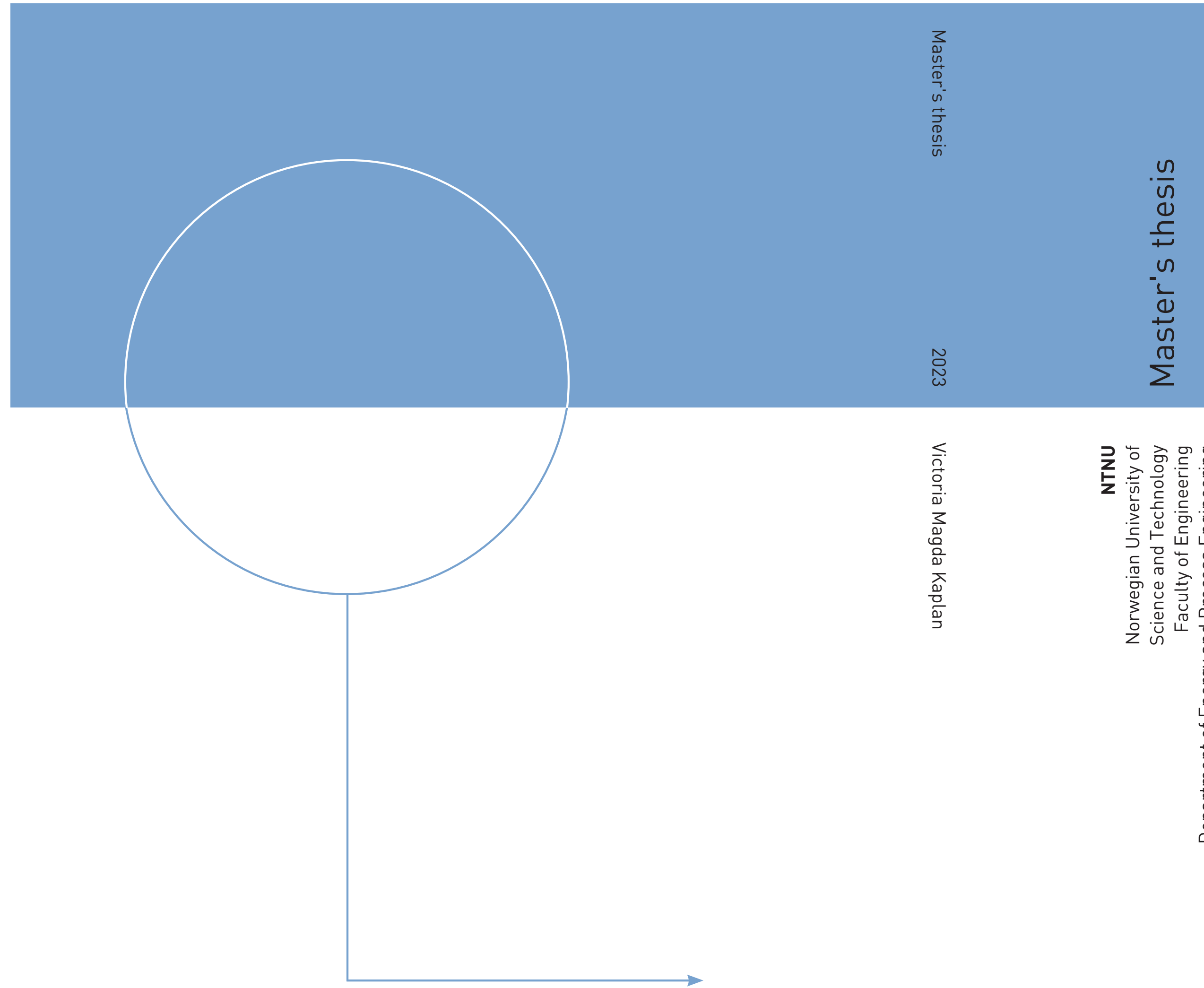


Victoria Magda Kaplan

Digital Twin Model for Gas Turbine Power Generation Forecasting

MSc. Natural Gas Technology
Supervisor: Lars Olof Nord
Co-supervisor: Even Solbraa

July 2023

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of
Science and Technology

Norwegian University of
Science and Technology

Digital Twin Model for Gas Turbine Power Generation Forecasting

MSc. Natural Gas Technology

Supervisor: Lars Olof Nord

Co-supervisor: Even Solbraa

Victoria Magda Kaplan

MSc. Natural Gas Technology

Submission date: July 2023

Supervisor: Lars Olof Nord

Co-supervisor: Even Solbraa

Norwegian University of Science and Technology
Department of Energy and Process Engineering

Victoria Magda Kaplan

Digital Twin Model for Gas Turbine Power Generation Forecasting

Master's thesis in Natural Gas Technology
Supervisor: Lars Olof Nord
Co-supervisor: Even Solbraa
June 2023

Norwegian University of Science and Technology
Faculty of Engineering
Department of Energy and Process Engineering

ABSTRACT

In recent years, digital twin models have gained significant interest as powerful tools for predicting the behavior of complex processes and types of equipment, assisting in damage prevention, increasing productivity, and facilitating decision-making in various industries. This study focuses on developing a digital twin of a specific gas turbine for power forecasting by accurately modeling its behavior under changing ambient conditions.

Despite its promising task, it comes with equal challenges: to generate a precise digital twin needs to start with an accurate model. From this point forward, the digital twin can correctly simulate the behavior and assist the industry with decision-making challenges.

A comprehensive gas turbine model was developed using Python, enabling the estimation of power generation based on varying ambient temperatures. The model underwent verification and validation procedures to ensure its accuracy and reliability. Furthermore, it was integrated with a weather forecast API, allowing the prediction of the gas turbine's power output under different weather scenarios.

The methodology employed in building the digital twin model is described in detail, including the equations utilized and the setup for the simulation software. The verification and validation process is thoroughly discussed to emphasize the credibility of the digital twin model for real-world applications.

The results obtained from the digital twin simulations demonstrate its potential capability to provide precise and reliable power generation forecasts for the studied gas turbine.

This thesis contributes to the advancement of digital twin models for gas turbines and showcases the potential of such models in power forecasting and decision-making in the energy industry. The outcomes also emphasize the importance of accurate modeling and data integration to maximize the benefits of digital twin technology for industrial applications.

SAMMENDRAG

De siste årene har digitale tvillingmodeller fått betydelig interesse som kraftige verktøy for å forutsi oppførselen til komplekse prosesser og typer utstyr, bistå med skadeforebygging, øke produktiviteten og lette beslutningstaking i ulike bransjer. Denne studien fokuserer på å utvikle en digital tvilling av en spesifikk gassturbin for kraftprognoser ved å nøyaktig modellere dens oppførsel under skiftende omgivelsesforhold.

Til tross for den lovende oppgaven, kommer den med like utfordringer: å generere en presis digital tvilling må starte med en nøyaktig modell. Fra dette tidspunktet kan den digitale tvillingen simulere atferden på riktig måte og hjelpe industrien med beslutningsutfordringer.

En omfattende gassturbinmodell ble utviklet ved bruk av Python, som muliggjør estimering av kraftproduksjon basert på varierende omgivelsestemperaturer. Modellen gjennomgikk verifikasjons- og valideringsprosedyrer for å sikre nøyaktighet og pålitelighet. Videre var den integrert med et værvarslings-API, som muliggjorde forutsigelse av gassturbinens kraftuttak under forskjellige værscenarier.

Metodikken som brukes for å bygge den digitale tvillingmodellen er beskrevet i detalj, inkludert likningene som ble brukt og oppsettet av simuleringsprogramvaren. Verifikasjons- og valideringsprosessen er grundig diskutert for å understreke troverdigheten til den digitale tvillingmodellen for applikasjoner i den virkelige verden.

Resultatene oppnådd fra de digitale tvillingsimuleringene viser dens potensielle evne til å gi presise og pålitelige kraftproduksjonsprognoser for den studerte gassturbinen.

Denne masteroppgaven bidrar til å fremme digitale tvillingmodeller for gassturbiner og viser potensialet til slike modeller i kraftprognoser og beslutningstaking i energiindustrien. Resultatene understreker også viktigheten av nøyaktig modellering og dataintegrasjon for å maksimere fordelene med digital tvillingteknologi for industrielle applikasjoner.

PREFACE

This master thesis was written under the guidance of Lars Olof Nord and Even Solbraa at the Department of Energy and Process Engineering, Norwegian University of Science and Technology.

I would like to express my deepest appreciation to both my supervisors, Lars and Even, for their support throughout the entire process. Their guidance during challenging phases of the work was invaluable. I am also immensely grateful to Sviatoslav Eroshkin, whose assistance and continuous encouragement since the beginning of the project have been essential. Your support and expertise have truly made a profound difference.

I extend my heartfelt thanks to all my friends who have accompanied me on this arduous journey of pursuing a master's degree. To my friends Dhruva, Denisse, Lukas, Marina, Maria, Marthine, Marco, Mohamamd, especially. Thank you for all the laughter and support during the studies. I am very grateful to have made friends in my masters that I will carry to my personal life.

Moreover, I dedicate this thesis to my family, whose belief in me even when I doubted myself has been an unwavering pillar of support. Your early phone calls from Brazil and Poland, filled with encouragement and love, have meant the world to me. I consider myself incredibly fortunate to have such an extraordinary level of unconditional support from each of you

CONTENTS

Abstract **i**

Sammendrag **ii**

Preface **iii**

Contents **v**

List of Figures **v**

List of Tables **vii**

Abbreviations **ix**

1 Introduction **1**

2 Theory **3**
- 2.1 Gas turbines 3
- 2.2 Object orientated programming 5
- 2.3 Weather forecast Application Programming Interface - API 7
- 2.4 Digital Twin 8

3 Methods **13**
- 3.1 Gas Turbine Performance Calculations 13
  - 3.1.1 Compressor 13
  - 3.1.2 Combustor 16
  - 3.1.3 Turbines 17
  - 3.1.4 Gas Turbine 18
  - 3.1.5 Off design performance 19
- 3.2 Python Model 22
  - 3.2.1 Combustion 25
  - 3.2.2 Off design performance 28
- 3.3 Verification and Validation 29
- 3.4 HYSYS model 30
- 3.5 Thermoflow Model 31
- 3.6 Weather Forecast API 32
- 3.7 Digital Twin 34

4 Results and Discussion 35
4.1 Design Model . . . 35
4.2 Off-design scenarios . . . 37
4.3 Verification . . . 37
4.4 Validation . . . 43
4.4.1 Limitations of the model . . . 50
4.5 Weather API connection . . . 51
4.6 Digital Twin . . . 53

5 Conclusion and Further Work 57
5.1 Conclusion . . . 57
5.2 Future Work . . . 58

References 59

Appendices: 63

LIST OF FIGURES

2.1.1 Pressure - volume diagram of Brayton cycle. [5] 4
2.1.2 Cross section of gas turbine SGT A-35 for 34 and 38 MW showing the main parts: 1 - compressor, 2 - combustor and 3 - expander. Adapted from [6] 4
2.4.1 Digital twin model framework, extracted from [15]. 10

3.1.1 Gas Turbine main components. 13
3.1.2 Compressor characteristic.[18] 14
3.1.3 Compressor control volume. 15
3.1.4 Turbine characteristic.[18] 17
3.1.5 Turbine control volume. 18
3.1.6 Off design calculation procedure. 21
3.4.1 Aspen HYSYS model. 30

4.3.1 Power Output variation with Ambient Temperature. Green represents the values generated from HYSYS model, and the blue line are the outputs from the Python mode. 37
4.3.2 Efficiency of gas turbine versus Ambient Temperature. Blue line represents the values extracted from HYSYS model, and orange represent the Python model results. 38
4.3.3 Gas Turbine Efficiency vs. Power Output at 0° C Ambient Temperature. 38
4.3.4 Gas Turbine Efficiency vs. Power Output at 15° C Ambient Temperature. 39
4.3.5 Gas Turbine Efficiency vs. Power Output at 30° C Ambient Temperature. 39
4.3.6 Turbine Exhaust Temperature T_{amb} = 0 °C. 40
4.3.7 Turbine Exhaust Temperature T_{amb} = 15 °C. 40
4.3.8 Turbine Exhaust Temperature T_{amb} = 30 °C. 41
4.3.9 Compressor Outlet Temperature - T_{amb} = 0 °C. 41
4.3.10 Compressor Outlet Temperature - T_{amb} = 15 °C. 42
4.3.11 Compressor Outlet Temperature - T_{amb} = 30 °C. 42
4.4.1 Siemens SGT A35 RB overview. [24] 43
4.4.2 Siemens SGT A35 RB simplified diagram. 44
4.4.3 Air flow for off design conditions comparison. 48
4.4.4 Power and efficiency output results for Thermoflow and Python. . . 49

4.4.5 Gas turbine power results comparison. 49
4.4.6 Gas turbine efficiency results comparison. 50
4.4.7 Exhaust turbine temperature results comparison. 51
4.5.1 Average temperature weather historical data for the geographical location selected. 52
4.6.1 Power output estimation for the time frame selected. 55
4.6.2 Power output forecast. 56
4.6.3 Efficiency of gas turbine forecast. 56

LIST OF TABLES

3.5.1 Fuel gas specification inputted in GTPRO. 31

4.1.1 Model comparison setup. * Air composition of 79.81 % N_2 and 20.09 % of O_2. ** for this first verification, the fuel composition was simplified to 100 % CH_4. 35
4.1.2 Comparison of Python and HYSYS model outputs. 36
4.4.1 Design conditions of SGT A35 - extracted from GTPRO and [24]. *Power Output at Generator Terminal. 44
4.4.2 Air composition, extracted from GTPRO (considering 0% relative humidity). 45
4.4.3 GT PRO inputs to GasTurb. * Booster Compressor is the Intermediate Pressure Compressor represented in the gas turbine setup, and HP stands for high pressure compressor. 46
4.4.4 Iterations setup on GasTurb. 47
4.4.5 Comparison of Thermoflow and Python model for design conditions. GasTurb setup: Booster Turboshaft HP Spool 47
4.5.1 Geographical coordinates of the offshore oil field. * Coordinates written in DMS (degrees, minutes, seconds) format. **Coordinates written in decimal format. 51
4.6.1 OLS Regression Results. 54

NOMENCLATURE

Abbreviations

API	Application Programming Interface	
JSON	JavaScript Object Notation	
GT	Gas Turbine	
LHV	Lower Heating Value	kJ/kg
LCV	Lower Calorific Value	kJ/kg
OOP	Object Oriented Programming	
HPC	High Pressure Compressor	
HPT	High Pressure Turbine	
IPC	Intermediate Pressure Compressor	
PT	Power Turbine	

Greek Letters

η	Efficiency	

Latin Letters

A_c	Cross-Section Area	m^2
C_p	Specific Heat Capacity	kJ/kgK
F	Fuel Flow Rate	kg/s
h	Enthalpy	kJ/kg

$\dot{m}$	Mass Flow	kg/hr
ΔP	Pressure difference	kPa
P	Pressure	kPa
r	Pressure Ratio	
T	Temperature	K
W	Work	MW

Subscripts

amb	Ambient
c	Combustor
comp	Compressor
in	Inlet
is	Isentropic
n	Number of stages
new	New
out	Outlet
pol	Polytropic
ref	Reference
t	Total
turb	Turbine

CHAPTER ONE

INTRODUCTION

In a world with an emerging population, industry expansion, and increased reliance and dependency on technology, the energy demand grows at an unprecedented pace. Lately, geopolitical issues highlighted just how important it is to generate not only energy addressing environmental concerns but in a way that it is reliable and available.

Amidst this energy crisis, finding innovative solutions to deliver power becomes of the utmost relevance. One promising approach lies in the optimization of gas turbine operation, an equipment already established in the industry.

Gas turbines are critical components in power generation facilities, and their efficient operation is vital for meeting energy demands and securing power availability. Compact equipment providing reliable energy with low emissions, high efficiency, and the ability to operate with a wide range of different fuels, gas turbines have been the choice for many industries for power generation.

Nonetheless, the inherent complexities of gas turbine systems, combined with ever-changing operating conditions present challenges to precisely forecast performance and predict potential issues in advance. Inaccuracy to simulate off-design performance and the effect of environmental conditions is a challenge for gas turbine performance evaluation.

In recent years, digital twin technology emerged as a powerful tool for modeling and simulating the behavior of physical systems in a virtual environment. By creating virtual copies of real systems, digital twins offer a powerful tool to model and simulate the behavior of gas turbines in a simulation environment.

From evaluating its performance, not only it can predict when degradation occurs but predicts when maintenance needs to be done, as well as when peak power production will occur. This technology has the potential to transform the way gas turbines operate, by optimizing and ensuring its safe and efficient performance, in the face of high energy demands and environmental regulations.

Also, project power generation accurately also gives improved estimates for energy availability, essential in a world where the energy demand increasing by as much as 15% from 2022 to 2050 [1].

Thus, this master thesis aims to develop a robust digital twin model of a generic gas turbine to predict the power generation accurately and evaluate the effect of environmental conditions on the equipment's performance. The model, built using Python, will be connected to a weather forecast API for validation

against historical data and for creating the digital twin for future power generation forecasts.

The work is structured as follows:

1. Literature review of gas turbines and their thermodynamic model.
2. Introduction to the concepts used for building the Python model, like object-oriented programming, weather forecast API, and digital twin concepts.
3. The method for building the Python model for design and off-design performance calculations of the gas turbine model.
4. Verification of the model with established software (Aspen HYSYS), software-to-software validation with another commercial software (Thermoflow ® Suite), and validation with actual data.
5. Analysis of verification and validation to ensure a reliable model.
6. Connection of the model with weather forecast API, validate the model with historical data, and for giving an estimation of power generation.
7. Conclusions and further work.

CHAPTER
TWO

THEORY

2.1 Gas turbines

The development of gas turbines dates back to 1791 when an Englishman named John Barber put up a patent for the first-ever gas turbine. However, it wasn't until 1903, when Norwegian inventor Ægidius Elling created a gas turbine able to generate more power than needed to run its components, which was a groundbreaking achievement at a time when knowledge about aerodynamics was limited. From there, gas turbines have evolved significantly both in performance and size reduction. Nowadays, gas turbines are the most versatile turbomachinery equipment in operation.[2]

Gas turbines, in essence, are power plants, converting chemical energy from fuel into mechanical energy, and produce considerable amounts of energy when compared to its size and weight. An increase in the use of gas turbines in the last decades in industries ranging from the power industry, utilities, as well as the petrochemical industry is due to its high efficiency, compactness, and low emissions. The flexibility to operate with various fuels also make gas turbines an obvious choice for power generation in offshore applications. [3]

The principle of operation of a gas turbine is based on the Brayton cycle, a thermodynamic cycle that follows four processes: compression, heating, expansion, and cooling, and is represented in figure 2.1.1. During an ideal thermodynamic cycle, when ambient air passes through the compressor, its pressure and temperature rise significantly, as seen on path $1 \rightarrow 2$. In an ideal process, entropy is assumed constant in a so-called *isentropic* process.

Heat is introduced into the system through the combustion of the fuel upon contact with the compressed air, along the constant pressure path $2 \rightarrow 3$. The total entropy and temperature of the system increase from the exothermic reaction during the isobaric process.

The hot and pressurized gases produced by combustion flow through the turbine, connected to the same shaft as the compressor. As the gases expand and lose pressure, they spin the turbine blades and generate mechanical energy. After spinning the turbine, the excess energy is used to generate work, such as generating electricity or propelling a vehicle. [4]

Finally, the exhaust gases are cooled and leave the system, completing the thermodynamic cycle. This process is represented by curve $3 \rightarrow 4$.

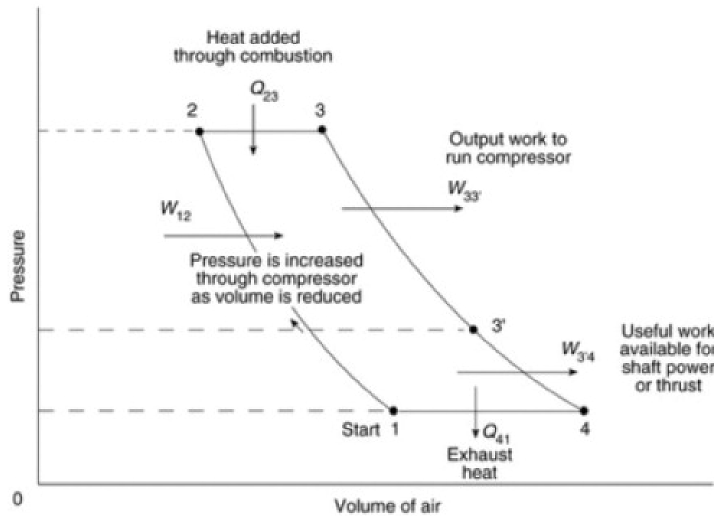


Figure 2.1.1: Pressure - volume diagram of Brayton cycle. [5]

A gas turbine main equipments are compressor, combustor, and turbine. Each part is connected into one or more shafts, collectively named gas generator. There, air entering the compressor is pressurized, heated, and used as an oxidizer of the fuel entering the combustor, where it is continuously burned generating gases with high temperatures and pressures, often called flue gas or exhaust gas. Then, the exhaust gas is expanded through a turbine producing mechanical work, and exits the turbine at lower pressure and temperature. The gas residual energy can be used for different applications, such as power generation, heating, and cooling. Figure 2.1.2 illustrates a real gas turbine schematic.

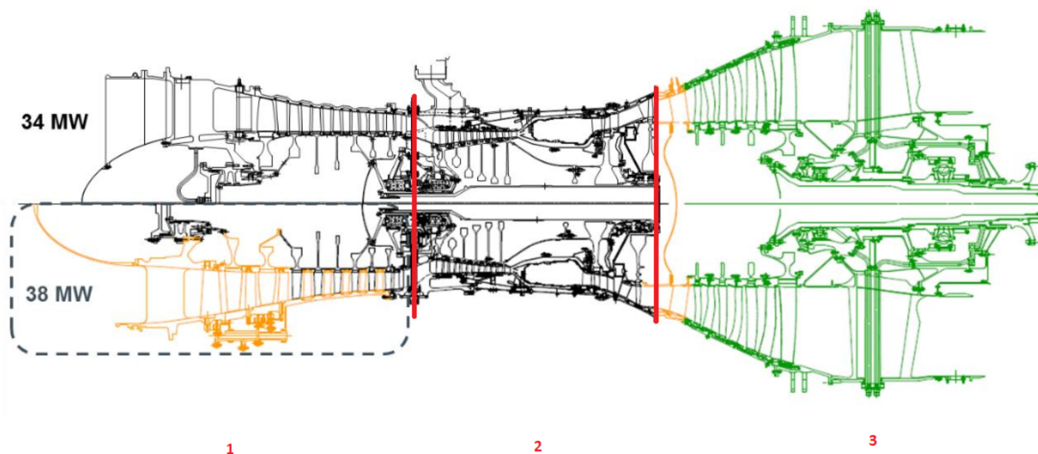


Figure 2.1.2: Cross section of gas turbine SGT A-35 for 34 and 38 MW showing the main parts: 1 - compressor, 2 - combustor and 3 - expander. Adapted from [6]

Gas turbines are classified in two categories: aeroderivative and industrial gas turbines. Initially developed for military use with focus on improving powering jet engines, aeroderivative gas turbines were later adapted for industrial use and

power generation. Nowadays, the distinction between those two categories is not so clear anymore, because aeroderivative gas turbines were modified for industrial use due to compactness, and high pressure ratio.

These gas turbines connect compressor and turbine into a single shaft, but also can be connected with multiple shafts running at different speeds, increasing operational flexibility of the equipment, especially during part-load operation, where the power generation is not at its maximum capacity.

Additionally, there are several ways for a gas turbine to produce energy. In a so-called simple cycle, a gas turbine can be power-, or mechanical driven. Power-driven turbines are connected to a generator, producing electricity that can be used for domestic or industrial applications, like in the aircraft industry.

Inside the simple cycle, there is also the mechanical-driven gas turbines, used to provide mechanical power directly into use, without the need of electricity generation. It can be used for powering equipments such as pumps, compressors, or other machinery in industrial applications. These gas turbines have a simpler design when compared to power-driven gas turbines due to the fact that it does not require the complex electrical systems for power generation.

Furthermore, some gas turbines can produce thermal energy that can be used for heating applications, as well as other industrial processes. Another type of gas turbine combines both thermal and electrical generation, in a combined heat and power (CHP) system, more commonly referred to as cogeneration systems, which increase overall efficiency of the system.

2.2 Object orientated programming

In a software development context, objects are representations of real-world systems within the virtual environment. They encompass data and associated behaviors, enabling executing actions with or on them. Much like physical objects, virtual objects can be manipulated and used for various functions [7].

Knowing what an object means, object-oriented means that an object has a direction towards something. Object-oriented works are focused on modeling objects. It is a way of describing complex physical systems as a collection of interacting objects through their data, behavior, as well as governing equations.

Object Oriented Programming (OOP) is a form of programming in Python that allows programmers to create objects that have methods and attributes. Methods is the name used in the object-orient context in the same manner as functions that use information about the object, as well as the object, defined inside a class. It returns results according to a formula or altering the object itself. OOP allows users to create their objects and methods and create a repeatable and organized code. In Python code, everything is an object, meaning that everything is an instance of a class.

As a program code grows more in size and complexity, it becomes necessary to structure it into group-related functions and data together, without interfering or being modified by unrelated functions and datasets. Creating distinct code blocks inside the main code makes it easier to understand, make modifications, as well as code reuse. This approach reduces code development time. [8]

Object-oriented programming is one of the many ways possible to organize a code. By writing all the functions, variables, and parameters necessary into aggregate data structures, and dictionaries for ad-hoc grouping of related data, using separate *namespaces*, the code has more structure and can be easily modified, and reclaimed.

However, the main advantage of object orientation, is that it is possible to combine data and functions which act upon that data into a single structure. This way makes it clear to find the related parts of the code, physically defined similarly to one another and also makes it easier to access the data only through that object's methods. This principle is called *encapsulation*, where the data inside an object should only be accessible through the object's functions, or methods.

Encapsulation is the main principle of object orientation, where it clusters methods and data operating into a single object, and it can only be accessed when calling out the method. This code then has several advantages, where the method is defined in a single logical place. This is where the data is kept and the data inside the object is not modified by any external code, guaranteeing data integrity. Finally, when using a method, one is interested only in the result of the method, not in the details to use it. This way is possible to change to using another object that could be entirely different structured, but the code does not change because of the same interface.

Despite encapsulation not being enforced by Python language, properties of an object are conventionally named by starting with an underscore. In Java language, for example, it is recommended to write *set* and *get* methods for all attributes, even if the setter method assigns the value of the passed parameter and the getter returns the attribute. [8]

These set-and-get methods are also applied for parameters that require some calculation. Other attributes that do not require calculation are directly written in the code, or given as external inputs for the code.

Since not all functions apply to all kinds of data provided in this code, user-defined objects were used in the form of classes. Classes are a data type, similar to string or list, where functions and related data act *upon* the data, located in one place. This way, the code is more organized and easier to program as it gets more complex. Data values stored inside an object are called *attributes*, and functions associated with the object operating on those attributes are named *methods*. *Set* and *get* methods are also applied for the calculated attributes.

With different classes written in a code, there are two main types of relationships between classes: composition and inheritance. Composition is the process of clustering objects altogether by making some objects attributes of other objects, forming a so-called *"has-a"* relationship. For example, consider a class named "Car" that has composition relationships with the "Engine" and "Wheel" classes. The "Car" class contains instances of Wheel and Engine working and without them, the object Car does not exist.

In composition, a class may have one or more variables of other classes, referred to as its components. They are essential for the class to run and cannot work independently. Meaning that, if the owner object ceases to exist in the code, its attributes will also disappear.

This is relevant because it enables code reuse, modularity, and further building complex code by combining simpler objects. It also promotes the creation of

more meaningful and specialized classes by hierarchically organizing them, or by interconnecting components. Changes in the components of the class are affected only locally, without impacting the entire system, improving the flexibility and maintainability of the code.

Inheritance is another fundamental concept of OOP, which allows classes to inherit methods, properties, and behavior from another class. Now, the relation between classes is in the form of *"is-a"*, where a derived or subclass inherits the characteristics from a base or parent class.

The example considered uses the class "Vehicle" as the base class, and the class "Car" and "Boat" as subclasses. It demonstrates the *"is-a"* relationship, where both the car and boat *are* vehicles. Each class is a vehicle, in this case, inheriting common behavior from the parent class, but each one has its specific methods and characteristics.

This way, the subclass automatically contains all the methods and variables of the parent class, allowing for easy code reuse. The subclass can override the behavior from the parent class, and also introduce new variables and methods that are specific to itself. However, inheritance needs to be used carefully while following the principles of encapsulation and abstraction. Incorrect use of inheritance leads to complex code maintenance and unforeseen errors. It is crucial to carefully analyze the relationship between the classes to determine if inheritance is the best approach for code design.[8]

Designing your objects can be done in a way as to represent real things, such as process equipment or chemical reaction. One can also create objects that do not have a physical representation but is still plausible to organize attributes and methods together. Code objects do not necessarily have to be comparable to real-world equivalents.

2.3 Weather forecast Application Programming Interface - API

Present everywhere nowadays, from mobile phones to weather forecasting, API stands for application programming interface. It is a software intermediary that allows for software to communicate with each other. APIs are an accessible way to extract and share data among organizations.

The term API is generally used to describe connectivity interfaces to an application. When using an application on your mobile, for example, it is connected to the internet, and data is sent to a server. The server then collects this data, interprets it, performs the necessary actions, and sends it back to the phone. The application then interprets the data and presents it in a readable and user-friendly way. [9]

For weather forecasts, a weather API allows weather data to be inquired into data scripts and code. It contains several weather measurements, such as wind speed, temperature, pressure, and humidity. More advanced APIs can also contain near real-time current conditions reporting, and years of worldwide weather reports.[10]

There is a variety of weather APIs available, each with its unique features and levels of data access. Some are free, while others require a subscription or

payment. The choice of the API will depend on the specific needs of an application or a project, as well as the desired level of data precision and reliability.

Weather APIs can be used in various applications, from providing weather information for outdoor activities and events to optimizing decision-making processes in industries, such as agriculture, transportation, and renewable energy. Overall, it is a powerful tool for accessing and sharing weather data, offering several benefits for individuals, as well as for industries alike.

2.4 Digital Twin

Digital twins can be defined as virtual copies of processes, equipment, and systems modeled using their data, functions, and capabilities related to the physical object.

Even before the term *digital twin* was coined in 2003 by Michael Grieves [11], one of the first digital twins registered came in the form of NASA's solution for allowing the damaged spacecraft Apollo 13 to return and bring its crew safely to Earth.

By using the data from failure scenarios in the simulators used for training the astronauts, NASA mission controllers quickly adapted to the real failure scenario and were able to adjust the operation and trajectory of the spacecraft to land safely and with no casualties.

Despite not being exact digital twins, these simulators were probably the first use of digital twins, matching the actual conditions of the spacecraft in a way that multiple possibilities were tested to bring the astronauts home.

The concept has lately been explored by several different fields, from using data collected from smart watches to track human health, dynamic simulation of weather forecasts, and going all the way to process monitoring for traffic control and also for oil and gas processing to avoid major spills, its interest increased at the same time as the methods for data monitoring improved. Operational flexibility combined with the use of Artificial Intelligence or Machine learning tools also contributes to pushing this new technology further.

Not only give live information about a real process, digital twin systems can be applied to improve decision-making, and also make predictions on how the process can perform in the future.[12]

Articles by Rasheed et al [13] and [12] describe the values of a digital twin, and the most relevant are described below:

- Visibility: Able to monitor operations in real-time, as well as it's interconnected systems;

- Efficiency and safety: minimize human intervention in dangerous operations;

- Maintenance: possibility to detect early faults much in advance, enabling better maintenance scheduling;

- Risk assessment: by having a virtual copy of a real asset, different scenarios are simulated in the digital twin to observe the system behavior at unusual conditions, as well as the correlated mitigation strategies;

- Synergy and collaboration: by interconnecting different systems, software, and data into different areas of the business, the decision-making time reduces;
- Personalization: modifying the digital twin according to each process, and asset, with great detail enables faster evaluation of changes in operation based on market and process data;
- Documentation: the digital twin can also be used for storing data, ready and available for stakeholders.

Despite its promising outcomes, the evolution of digital twins needs to be accompanied by the development of more precise models of the physical asset.

The physical realism of the model depends on accurate sensor technology, physics-based simulators, and data-driven models. The data-driven models ensure that the model is continuously updated according to reality, adapting to new changes in operation.

This constant model upgrade is also accompanied by hybrid analysis and modeling, where the digital twin is built with a combination of simulated and real measured data, increasing the sensibility and trustworthiness of the model.

Additionally, a highly reliable model needs to be accompanied by progress on different fronts at the same tie, from continuous model improvement to human-machine interface and data security, which are not the focus of this study.

There are distinct ways of building a digital twin model, and three of them are described as follows:

Physics-based modeling

A physics-based modeling incorporates the observation of a physical phenomenon and translating its behavior in the form of mathematical equations, to ultimately set results.

However, the number of suppositions necessary to translate the physical observation into several equations ends up not fulfilling all the physics requirements.

The physics-based modeling approach has the great advantage of being generally less biased than data-driven models because they are governed by the laws of nature. The use of numerical simulators derives from those physics equations, to solve those equations numerically.

Despite this approach being more reliable because it is based on actual equations, the use of numerical simulators can lead to numerical instability, where it may not be possible to encompass all the necessary historical data, in the long term.

However, the advances in to use of high-fidelity numerical simulators in the last two decades allow them to establish predictive digital twins, as stated by Rasheed et al.[12]

Data driven modeling

In this type of modeling, real-data information collected from measurement instruments and historical data from the physical process is analyzed by machine learning algorithms using statistical techniques to create the digital twin.

These models are used when it is necessary for critical real-time monitoring and control of a process, or when the system is complex and difficult to explicitly model using only equations.

This way, the data is treated as an expression of both known and unknown physical phenomena, and by developing a data-driven model, one can account for the full physics, fulfilling the gaps from the pure physics model. [14]

Hybrid model

Hybrid digital twin models are a combination of the two models above, in a complementary way.

Here, the physical model sets the foundation for the model, where it is adjusted and refined by the inputs gathered from real-time sensors and measurements based on real operating conditions.

This model combines the advantages of both approaches, by providing a solid and accurate foundation from the physics-based model, and the versatility of real-time insights from the data-driven models. The theory and practical approach combined have gained popularity in recent years. [15]

A framework extracted from [15] illustrating the creation of a digital twin of a gas turbine is shown below, which summarizes the steps used for creating a hybrid digital twin model in figure 2.4.1:

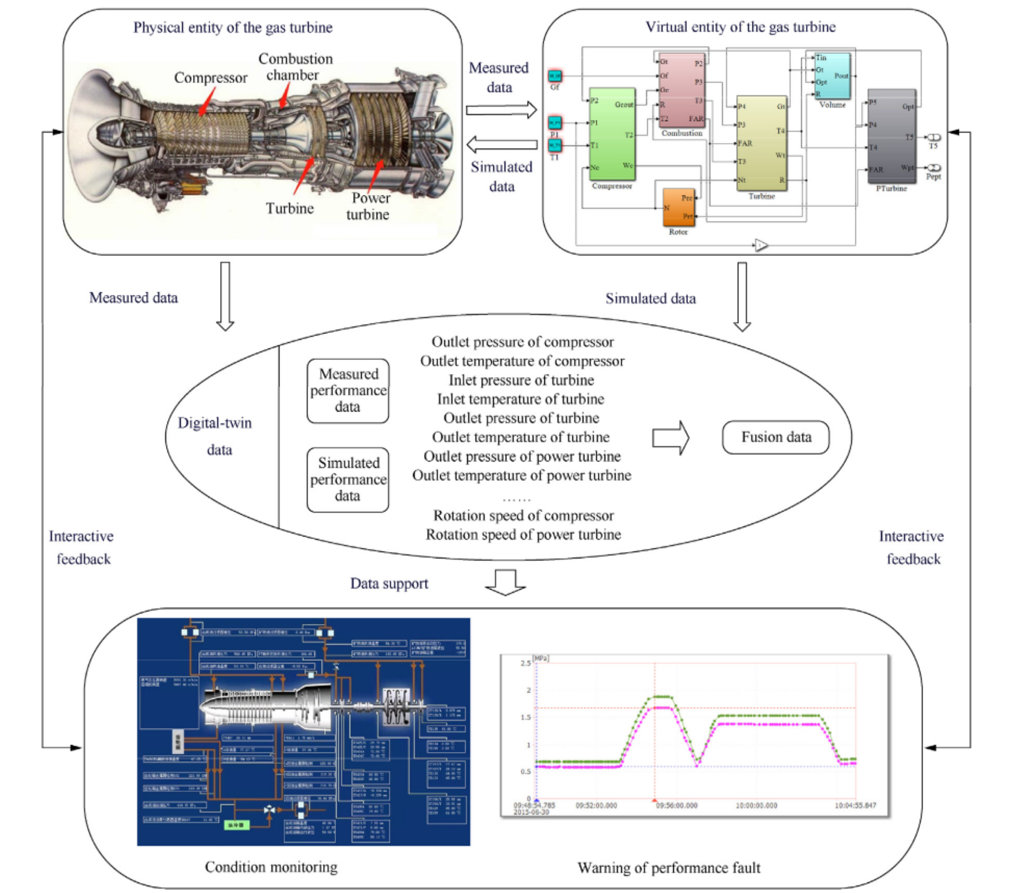


Figure 2.4.1: Digital twin model framework, extracted from [15].

The alternative approaches to building a digital twin need to be carefully selected but always focused on the ultimate goal, which is to have very few differences in the behavior of the real object, requiring a special dedication to building the model.

CHAPTER
THREE

METHODS

3.1 Gas Turbine Performance Calculations

For evaluating the performance of a gas turbine, there are different methods, varying in levels of precision. One of those could be by matching the compressor and turbine performance characteristics, which are particular to each gas turbine, and are described in more detail in chapter 3.1.5. However, the main drawback of using these charts is that they are specific for each manufacturer, meaning that may not be easily accessible.

For this study, a simplified model of a two-shaft industrial gas turbine is used for developing the model in Python. The thermodynamics of the main components described in 2 is described as follows, where the exhaust gases are used to the turbine coupled with an electricity generator, as seen in figure 3.1.1. Temperatures and pressures are represented as T_o and P_{ref} and the subscript i = *1,2,3,4* will be used throughout this work for representing each part of the engine.

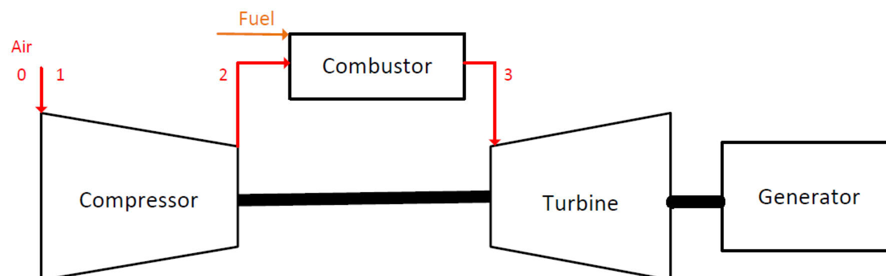


Figure 3.1.1: Gas Turbine main components.

3.1.1 Compressor

Compressor performance calculation is an essential part of gas turbine analysis, as it allows for estimating compressor efficiency, as well as operating characteristics. To better understand these calculations, it is important to understand the role of the compressor in the gas turbine system.

A compressor is a device that pressurizes air, increasing the temperature and pressure of the fluid, meaning that it requires power input to provide energy for

the fluid. Increased pressure of air intake increase combustion process and power extraction process after combustion more efficiently.[16] Performance analysis of gas turbines ignores inlet pressure losses and assumes, assuming that the inlet is ideal[17]. This means that ambient pressures and temperatures are equal to the inlet of the compressor, $P_0 = P_1$ and $T_0 = T_1$.

Compressor performance is represented by a characteristic map showing the relation of efficiency η_c, corrected shaft rotational speed $\frac{\omega}{\sqrt{T_{01}}}$, corrected mass flow rate $\frac{\dot{m}\sqrt{T}}{P}$ and pressure ratios $\frac{P_{0e}}{P_{01}}$.

Characteristics graphs are generated from experimental data relating pressure ratio with some parameters such as temperature or mass flow, at different speeds. Corrected mass flow rate is represented by $\frac{\dot{m}\sqrt{T}}{P}$, where it accounts for the changes in mass flow according to different operating conditions. It considers changes in temperature, pressure, and humidity and is a way to standardize the actual flow rate to a set of reference conditions. A compressor characteristic is represented in figure 3.1.2

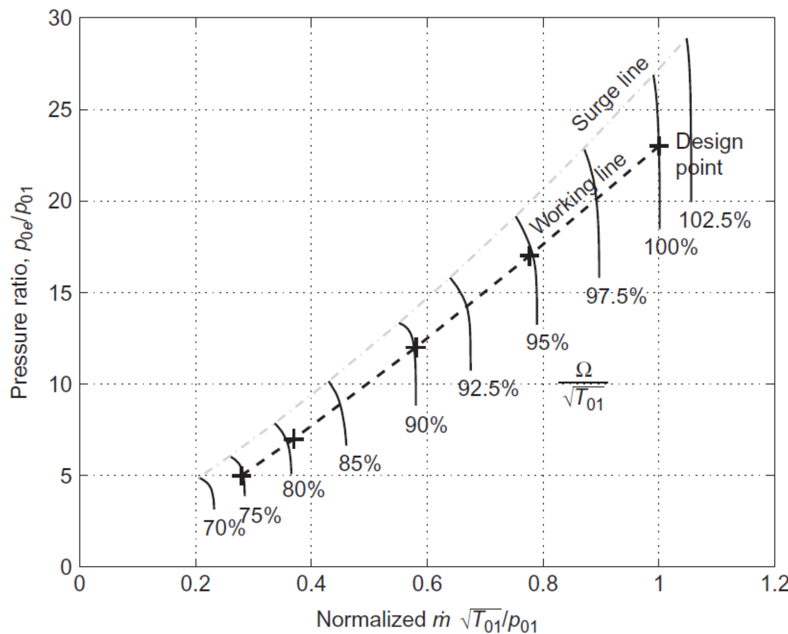


Figure 3.1.2: Compressor characteristic.[18]

For large axial compressors, the lines of constant reduced speed can be considered vertical, and the isentropic efficiency η_{is} is constant. These assumptions are valid for both design and off-design modeling. Using the pressure ratio given as an output of the compressor map, the isentropic outlet temperature T_{2s} of a compressor is determined using the pressure ratio from the pressure temperature isentropic relation. The actual outlet temperature of the compressor is then calculated afterwards, as follows:

$$T_{2s} = T_1 \left(\frac{P_2}{P_1}\right)^{(\frac{k-1}{k})}$$
$$T_2 = T_1 + \frac{T_{2s} - T_1}{\eta_{is,c}} \quad (3.1)$$

Where kappa (k $= \frac{C_p}{C_v}$ is calculated using neqsim) and $\eta_{is,c}$ is the isentropic efficiency of the compressor.

An isentropic efficiency η_{is} is defined as the ratio between ideal and actual enthalpy change during the compression process:

$$\eta_{is} = \frac{h_{2s} - h_1}{h_2 - h_1} \approx \frac{T_{2s} - T_1}{T_2 - T_1} \quad (3.2)$$

Polytropic efficiency η_{pol} of the compressor can be defined as the isentropic efficiency of the compressor divided into several stages, and the relation between those two is given by:

$$\eta_{is} = \frac{\left(\frac{P_2}{P_1}\right)^{\frac{k-1}{k}} - 1}{\left(\frac{P_2}{P_1}\right)^{\frac{k-1}{k \cdot \eta_{p,c}}} - 1} \quad (3.3)$$

Using figure 3.1.3 for illustrating the boundary limits for the compressor calculations, the compressor work equation will be derived below.

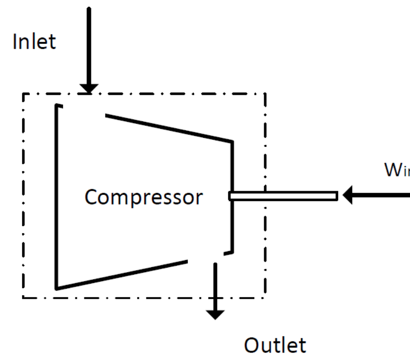


Figure 3.1.3: Compressor control volume.

Starting with the principle of mass conservation and the first law of thermodynamics, the mass and complete energy balance of the compressor are:

$$\frac{dm_{cv}}{dt} = \dot{m}_{in} - \dot{m}_{out} \quad (3.4)$$

$$\frac{dE_{cv}}{dt} = \dot{Q}_{cv} - \dot{W}_{shaft} + \dot{m}_i \left(h_i + \frac{c_i^2}{2} + gz_i\right) - \dot{m}_{out} \left(h_e + \frac{c_e^2}{2} + gz_e\right) \quad (3.5)$$

Where cv subscript represents the control volume represented in figure 3.1.3. Assuming steady-state flow, the mass balance then is $m_{in} = m_{out}$, and $\frac{dE_{cv}}{dt} = 0$. Assuming negligible kinetic and potential energy changes in the system, the differences between velocities and elevation z is also not considered.

With these simplifications, the energy equation then becomes:

$$\dot{W}_{shaft} - \dot{Q}_{cv} = \dot{m}(h_{out} - h_{in}) \tag{3.6}$$

In an adiabatic process, Q_{cv} can also be considered zero. Enthalpy h change is a function of heat capacity at constant pressure and temperature change:

$$dh = C_p \cdot dT \tag{3.7}$$

And C_p is the specific heat capacity at constant pressure. Assuming that C_p is constant during the compression process, the enthalpy equation then is:

$$\Delta h = C_p \Delta T \tag{3.8}$$

Where Δ T is the difference between inlet and outlet temperatures. Replacing h with the equation 3.8, the energy necessary to power the compressor is then given by:

$$W_{comp} = \dot{m} C_p (T_2 - T_1) \tag{3.9}$$

And C_p is the specific heat of air.

3.1.2 Combustor

The combustor function is to burn a mixture of air and fuel, and to deliver the resulting exhaust gases to the turbine at an uniform temperature. Thermal energy of the fuel/air mixture is increased during the combustion process.

Fuel is mixed with air, which supplies the oxygen necessary for the combustion process. Mass and energy balance in the combustor is the given below. Here, steady-state conditions are assumed, as well as heat capacity at constant pressure of air and fuel streams, as well as the combustion gases stream.

$$\dot{m}_3 = \dot{m}_1 + \dot{m}_{fuel} \tag{3.10}$$

$$\dot{E}_{in} = \dot{E}_{out} \tag{3.11}$$

$$m_1 C_{p,1} \delta h_1 + m_{fuel} \cdot LHV = \dot{m}_3 C_{p,3} \delta h_3 \tag{3.12}$$

Since enthalpy change is simplified for $C_p \Delta T$, the outlet temperature of combustor, or turbine inlet temperature (TIT), is determined using the following energy balance:

$$\dot{m}_1 C_p (T_2 - T_0) + \dot{m}_f \cdot LHV = \dot{m}_3 C_{p3} (T_3 - T_0) \tag{3.13}$$

where T_0 is the ambient temperature, LHV stands for lower heating value of the gas, and C_{p3} represents the specific heating value at constant pressure of the resulting gases of combustion.

3.1.3 Turbines

The function of a turbine is the exact opposite of the compressor, which is to expand the fluid expands, releasing energy. In this case, work is *generated* by the turbine, and this energy can be used for power generation, or even compressor drive. Kinetic energy from the expanding gases which flow from the combustion chamber is converted into shaftpower to drive the compressor, as well as other equipments.

Turbine performance can be equivalent to the compressor, where it is also represented by characteristics showing the relations between pressure ratios and normalized flow capacity, as represented in figure 3.1.4. It is important to observe that the constant speed lines turn flat at a certain point. When this happens, the flow inside the turbine is considered choked. Choked flow is a restricting condition that arises when an increase in pressure difference along a turbine does not alter the inlet mass flux.

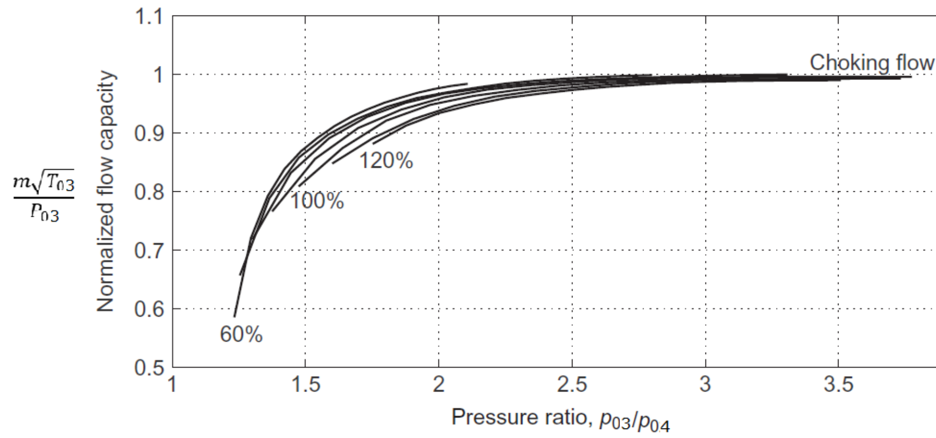


Figure 3.1.4: Turbine characteristic.[18]

The outlet temperature of the turbine is calculated using the same approach as for the compressor, given by:

$$T_{4s} = T_3 \cdot \left(\frac{P_4}{P_3}\right)^{\frac{k-1}{k}} \\ T_4 = T_3 - \eta_{is,t} \cdot (T_3 - T_{4s}) \tag{3.14}$$

Where T_{4s} is the isentropic outlet temperature of the turbine, kappa (k $= \frac{C_p}{C_v}$ is calculated using neqsim) and $\eta_{is,t}$ is the isentropic efficiency of the turbine, and T_3 is the turbine inlet temperature.

An isentropic efficiency η_{is} is defined as the ratio between ideal and actual enthalpy change during the expansion process:

$$\eta_{is} = \frac{h_{3s} - h_4}{h_3 - h_4} \approx \frac{T_3 - T_4}{T_3 - T_{4s}} \tag{3.15}$$

Similarly to the compressor, the isentropic and polytropic efficiency of the turbine is related as follows:

$$\eta_{is,t} = \frac{1 - \left(\frac{P_4}{P_3}\right)^{\frac{k-1}{k}}}{1 - \left(\frac{P_4}{P_3}\right)^{\frac{k-1}{k \cdot \eta_{p,t}}}} \tag{3.16}$$

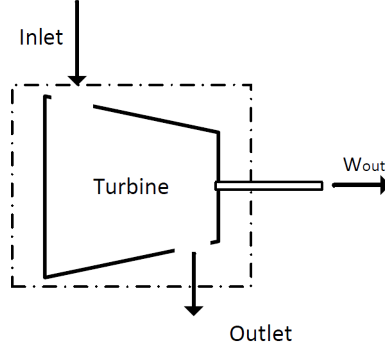


Figure 3.1.5: Turbine control volume.

The mass and energy balances shown in 3.1.1 are also valid for the turbine. The difference lies in the work direction. Since work is being *produced* by the turbine, it is released into the system, as shown in 3.1.5 above. The simplified energy balance then becomes:

$$\dot{Q}_{cv} - \dot{W}_{shaft} = \dot{m}(h_{out} - h_{in}) \tag{3.17}$$

Assuming adiabatic expansion, and using the enthalpy equation for constant C_p and temperature difference, the work produced by the turbine is given by:

$$W_{turb} = \dot{m}_3 \cdot C_{p,3}(T_3 - T_4) \cdot \eta_m \tag{3.18}$$

And m_3 is the sum of the mass flow of air and fuel, $C_{p,3}$ is the specific heat at constant pressure of the combustion gases, and η_m is the mechanical efficiency of the turbine.

3.1.4 Gas Turbine

In the overall energy balance of the gas turbine, the power consumed by the compressor (W_{comp}) and the power generated by the turbine (W_{turb}) needs to be subtracted from one another to result in the net power produced by the gas turbine. The net power output of the gas turbine becomes:

$$W_{GT} = W_{turb} - W_{comp} \tag{3.19}$$

There are several ways for accounting for the efficiency of a gas turbine [19]. Here, the efficiency based on the lower heating value of the fuel is calculated, and the formula is given below:

$$\eta_{GT} = \frac{W_{GT}}{\dot{m}_f \cdot LHV} \tag{3.20}$$

where W_{GT} is the power generated by the gas turbine in MW, m_f is the mass flow of fuel in kg/s, and LHV is the lower heating value of the fuel in MJ/kg.

3.1.5 Off design performance

Gas turbines are designed for specific conditions of temperature, pressure, and fuel demand, but once it goes into operation, these conditions change and can affect their overall performance.

To determine how a gas turbine will operate under specific conditions, a set of equations are used to calculate its performance at a particular speed, pressure ratio, and mass flow. When the turbine runs at steady speed, and all components operate in equilibrium, it's possible to create an equilibrium running diagram with multiple equilibrium points. They are used to determine performance curves for power thrust and specific fuel consumption.

The set of equations above is used to determine a gas turbine operation during a single *design* point, where the equipment is running at a particular speed, pressure ratio, and mass flow. When the turbine runs at a steady speed and its components are operating in equilibrium. A set of several equilibrium points forms an equilibrium running diagram, which can then be used to determine performance curves for power thrust and specific fuel consumption, at different speeds.

However, the challenge lies in when the operation conditions deviate from design, or when determining the limits of operation and power output of the engine, usually referred to as *off design performance*. This is important not only to establish the operational boundaries of a gas turbine but also to ensure correct operation and for the manufacturer to guarantee the performance of the engine within those boundaries.[4]

Two main analyses are performed: one, the reduction in specific fuel consumption with regards to reduced power output, the effects of ambient conditions in the overall gas turbine performance, as well as for calculating emissions.

Reduced specific fuel consumption with a reduction in power generation is important to understand what will be the power output of the gas turbine, and how can it affect overall efficiency, while remaining within the required operational conditions, to avoid damaging the equipment. For example, when at constant speed the inlet flow rate of the compressor is reduced, it can lead the operating point towards surge, which may cause internal wear of the compressor and even equipment shutdown if it stays during this operational mode for longer periods.

Ambient conditions also have a significant impact on gas turbine performance, since it affects overall gas turbine power generation and efficiency. It is known that cold ambient temperatures contribute to the higher power output from the turbine and that higher temperatures have the opposite effect. Since such equipment needs to have guaranteed performance by the manufacturer, the wide range of ambient conditions on which it can operate also needs to be analyzed to ensure safe operation and performance within acceptable conditions.[20].

A basic method of determining the off-design performance of simple gas turbines will be described below. Off-design calculations have to be in accordance with mass flow, speed, and work between the various components.

Some assumptions are made for this model:

- Initially, the model was designed for a single-shaft gas turbine, coupled with a generator at constant speed;
- Air is an ideal gas;
- No pressure drop at the inlet of the compressor, i.e. $P_0 = P_1$;
- Constant reduced flow rate at the compressor, i.e, vertical lines on compressor map;
- Fully open inlet guide vanes (IGV) in the compressor are considered in this model, meaning that the compressor dimensions are fixed and are fixed at maximum inlet area;
- Choked flow on turbine;
- Polytropic efficiency of turbine and compressor are assumed constant.

With the assumption that the speed lines on the compressor map are constant, combined with the continuity equation, and using ideal gas to model air, the following relation is obtained, relating mass flow with temperature, pressure, and cross-section area A_c of the compressor, for design conditions (*ref* index) an for another condition. Since the area of the compressor is constant, and the pressures are also assumed constant, the simplified equation is also shown below:

$$\frac{\dot{m}}{\dot{m}_{ref}} = \frac{P}{P_{ref}} \frac{R_{ref}}{R} \frac{T_{ref}}{T} \frac{A_c}{A_{c,ref}} \tag{3.21}$$

$$\frac{\dot{m}}{\dot{m}_{ref}} = \sqrt{\frac{T_{ref}}{T}} \tag{3.22}$$

Inside the combustion chamber, pressure losses need to be taken into account to evaluate if there is fouling deposition, which can affect its performance. The relation of pressure losses of design conditions to new operating points is described as:

$$\frac{\Delta P_c}{\Delta P_{c,ref}} = \left(\frac{\dot{m}_3}{\dot{m}_{3,ref}}\right)^{1.8} \left(\frac{T_3 P_{3,ref}}{T_{3,ref} P_3}\right)^{0.8} \tag{3.23}$$

For the turbine, by assuming choked flow, the reduced flow is constant, as shown in 3.1.4. This means that a relation between reference reduced speed and the speed at different conditions can be established as well:

$$\frac{\dot{m}\sqrt{T_{01}}}{P_{01}}, \frac{\dot{m}_{ref}\sqrt{T_{01,ref}}}{P_{01,ref}} = constant \tag{3.24}$$

Relating the two conditions, the choked nozzle equation is established:

$$\frac{P_3}{P_{3,ref}} = \frac{\dot{m}_3}{\dot{m}_{3,ref}} \sqrt{\frac{T_3 \cdot MW_{3,ref}}{T_{3,ref} \cdot MW_3}} \tag{3.25}$$

During off-design operation, outlet pressures of the compressor and turbine vary according to the inlet conditions, and the design values cannot be used for evaluating gas turbine performance during this operation mode. For this, an iteration process is established for evaluating gas turbine power output and will be further implemented in the Python model.

The iteration process is schematized below in figure 3.1.6:

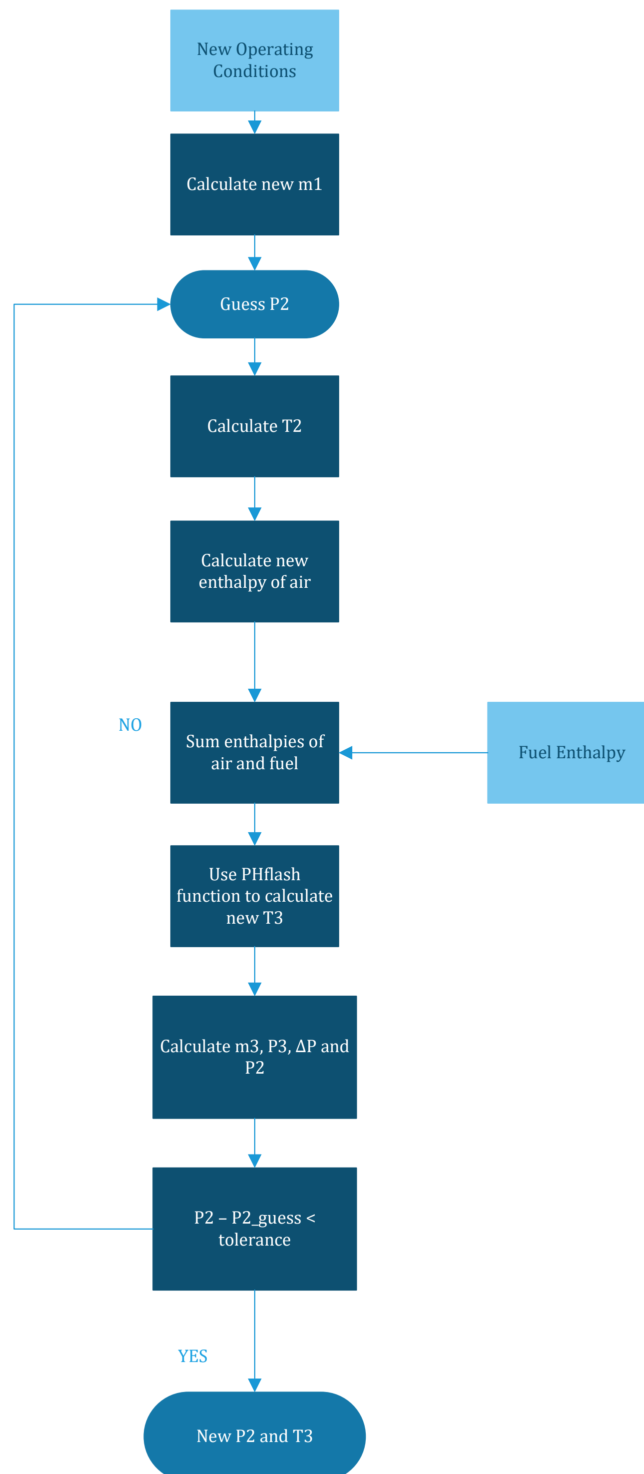


Figure 3.1.6: Off design calculation procedure.

Here, the new mass flow of air is calculated using the formula 3.22, according to the ambient temperature measured. During off-design operation, the compres-

sor and turbine operate at different pressure ratios, so it is not possible to use design values for performance calculations. The iteration process starts with a first estimate of the outlet pressure of compressor P_2, which is used to calculate the outlet temperature of compressor T_2.

These new conditions are used to calculate the new enthalpy of air, which will be used for further calculations. For a given mass flow of fuel, enthalpy is also calculated. These enthalpies are calculated using NeqSim in Python code. Then, the enthalpy of air and fuel are summed and used inside the PHflash function of NeqSim, to obtain the value for outlet temperature of combustor T_3.

PHflash is a function in NeqSim used to perform a phase equilibrium calculation at a given pressure and enthalpy. It takes a thermodynamic system object at a certain pressure and enthalpy and returns the corresponding equilibrium phase composition and properties. For purposes of simplification, PHflash was used for this calculation.

With a new value of T_3, the new outlet pressure of the combustor P_3 and pressure drop inside the combustor ΔP_c are also calculated, using equations 3.25 and 3.23 respectively.

Finally, a new value of the outlet pressure of the compressor is calculated, with the formula below:

$$P_{2,new} = P_3 + \Delta P_c \tag{3.26}$$

Resulting in a new value for the outlet pressure. If the difference between the guessed value and the calculated value is below tolerance, the iteration stops. If not, $P_{2,new}$ is the new guessed value of P_2. The iteration process continues until the tolerance criteria are reached.

3.2 Python Model

The equations illustrated in 4 need a suitable system and/or software to be able to perform calculations accordingly. An object-oriented program such as Python can perform all the necessary calculations, in a practical and accessible manner. Python holds several libraries that can create a prototype of almost any program. Since in this work the focus is to model a gas turbine, other libraries need to be used to supplement the model. Therefore, the physical properties of the fluids entering the gas turbine system will be calculated using Neqsim. In this master thesis, NeqSim library is used for calculating physical properties of the chemical compounds modeled. NeqSim Python is an interface to NeqSim Java library and is used for calculating fluid behaviour, phase equilibrium and process simulation. But first, it is necessary to download NeqSim Python library using pip install command in Python:

```
1 #install neqsim
2 !pip install neqsim
```

Although Neqsim already has built-in functions for some components of a process simulation such as stream, compressor or expander, a decision was made to implement them as methods directly in the Python code. Neqsim is exclusively being

used for calculating fluid properties. This way, it gives more control and flexibility over the code, where it is possible to easily change parameters and the methods, without delving into the Neqsim codebase.

As a stream is defined in the Python code, with its respective mass flow, temperature and pressure, a TPflash is performed inside each stream, as well as the *initProperties()* methods. Both functions are Neqsim built-in functions used before reading the physical properties of a defined stream. TPflash function calculates the equlibrium phase distribution and composition based on temperature and pressure. From there, the physical properties of the fluid, as well as the phase composition is determined.

After TPflash, the *initProperties()* method is used to ensure the correct initialization of the fluid properties in the code, to obtain accurate and consistent values. It sets up the necessary internal data structures and variables to enable the calculation of various properties of the fluid mixture, such as enthalpy, entropy, and density. This step ensures that the fluid properties are correctly initialized and ready for subsequent calculations and analysis.

The method for calculating the properties inside the Stream class is shown below:

```
1  class Stream():
2     def calc(self):
3
4             self.stream_fluid.setTemperature(self.temperature, "K")
5             self.stream_fluid.setPressure(self.pressure/1e5, "bara")
6             self.stream_fluid.setTotalFlowRate(self.flow_rate, 'kg/hr')
7         TPflash(self.stream_fluid)
8         self.stream_fluid.initProperties()
9
```

In order to write the code in a more structured and organized form, the gas turbine components, compressor, combustor, turbine, and also stream were separated into different **classes**, which are simply objects inside Python. This way, all the methods related to each element of the gas turbine is separated in classes for better clarification and traceability of the code.

The structure of the objects follows the same rules: the first method in every class is the `__init__` method. It starts the variables inside each class with a default value, and it ensures that when a new instance of the members of the class is created it avoids class data sharing among instances. The `__init__` method for the Stream class is shown as an example as follows:

```
1  class Stream():
2     def __init__(self):
3         self.temperature = None #K
4         self.pressure = None #Pa
5         self.flow_rate = None #kg/hr
6         self.stream_fluid= None #Neqsim fluid
7
8
```

Since there are several methods and attributes shared between different classes inside the code, when creating different objects of the same class, the methods and attributes not created inside a class can be overwritten, and that may cause errors in the results.

Next, the methods also used in each of the classes above the **get** and **set** for each parameter inside a class. An example of these methods inside compressor class is described below, and more details are in the whole code located in the Appendix:

```
 class Compressor():
    def set_work(self, work:float, units:str) -> float:
        '''
        Method to set compressor work. Returns compressor work in MW units. If
    the unit is not identified, an error will appear.
        '''
        if units == "MW":
            self.work = work
        elif units == "kW":
            self.work = work*1e3
        elif units == "W":
            self.work = work*1e6
        else:
            print(f"No units found for work in {self}.")
            return self.work

    def get_work(self, units:str) -> float:
        '''
        Method to check if the correct unit for work is defined. If not, an
    error will appear.
        '''
        if units == "MW":
            return self.work
        else:
            print(f"No units found for {self} compressor work.")
            return None


```

The *set* methods take in two arguments - the desired parameter and its respective unit. The method then converts the parameter to the base unit defined in the code, and sets the parameter with the correct base unit to the specific unit. Then, the *get* method takes in one argument, and if the parameter is with the correct unit, it returns its value. Otherwise, an error will appear. Overall, these functions are used to retrieve attributes of an object in a standardized way, which help to ensure consistency and accuracy of the calculations involved in each class. For every parameter relevant for the calculations of gas turbine performance, the *set* and *get* method are used.

Furthermore, the *calc* method is used. Every calculation performed inside a class is compiled inside this method, and it also verifies which variable is set. If the necessary number and type of parameters is defined, all the defined calculations are performed when the method is called. An extract of the *calc* method inside the turbine class is shown below:

```
 class Turbine():
    def calc(self):
        self.success = False
        if self.isentropic_efficiency is not None:
            if self.deltaP is not None:
                self.calc_inlet_pressure()
```

```
7                     self.calc_outlet_temperature()
8                     self.calc_outlet_stream()
9                     self.calc_polytropic_efficiency()
10                    self.calc_work()
11                    self.success = True
12
```

3.2.1 Combustion

There are several ways to model combustion inside gas turbines. From more complex to more simplified, all these methods have the ultimate goal: try to reproduce the behavior of burning a combustible fuel when mixed with air, and predict the resulted products of the combustion.

Since the focus of this work was not going in depth into the combustion mechanisms that occur inside the gas turbine, a simplification was made using Neqsim. The function PHflash was used in order to obtain the turbine inlet temperature - TIT. PH Flash is a method inside NeqSim library that calculates the distribution of components in different phases to a state of minimal Gibbs Free Energy point, in a liquid-vapor mixture.

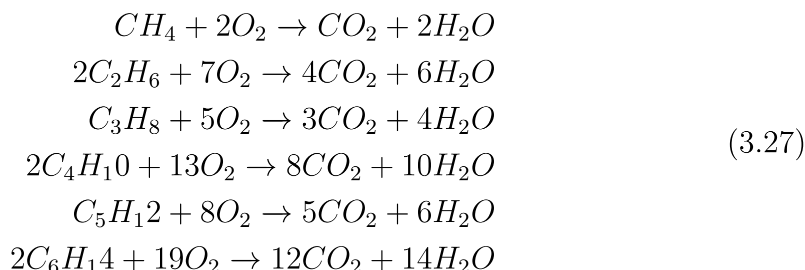
In the PHflash, fluid and enthalpy of the mixture initial guess is given, and a series of iterations are made in this method to estimate the fluid mixture composition that satisfies the minimum Free Gibbs Energy condition.

In the Python code, the initial value of the enthalpy is calculated by reading the enthalpy value of the compressed air stream and added with the enthalpy of the fuel stream. This calculation helps estimate the composition of the combustion products based on the given enthalpy.

The fuel stream enthalpy was determined by multiplying its Lower Calorific Value (LCV) with the mass flow in kg/s. LCV value is calculated using iso6976 method from NeqSim, which uses the ISO 6976 standard properties table to provide the physical properties of a gas mixture. By inputting fuel composition, and reference conditions of temperature of the volume and combustion, the Lower Calorific Value (LCV), Superior Calorific Value, as well as other properties of the fuel mixture, can be read using the iso6976 method. For this work, the LCV value only was used.

Furthermore, the fluid used in the PHflash calculation needed to be determined. First, it was assumed that the fluid was only composed of the air mixture, which is a reasonable assumption given the fact that the mass flow of air entering a gas turbine is significantly higher than the fuel flow. This approach was used for verification using Aspen HYSYS software.

However, the approach does not apply when using validation of the model using Thermoflow. Therefore, an estimation of the combustion products was made. Assuming complete combustion of the hydrocarbons present in the the natural gas mixture, the following stoichiometric reactions were written in the python code:



$$
\begin{aligned}
CH_4 + 2O_2 &\rightarrow CO_2 + 2H_2O \\
2C_2H_6 + 7O_2 &\rightarrow 4CO_2 + 6H_2O \\
C_3H_8 + 5O_2 &\rightarrow 3CO_2 + 4H_2O \\
2C_4H_10 + 13O_2 &\rightarrow 8CO_2 + 10H_2O \\
C_5H_12 + 8O_2 &\rightarrow 5CO_2 + 6H_2O \\
2C_6H_14 + 19O_2 &\rightarrow 12CO_2 + 14H_2O
\end{aligned}
\tag{3.27}
$$

For each reaction, the same procedure was performed to determine the outlet composition. The methane reaction was used to exemplify the procedure as follows:

1. Calculate the available moles of O_2 from the air stream;
2. Calculate the available moles of methane from the fuel stream;
3. Determine the stoichiometric ratio required for complete combustion;
4. Define the limiting reactant: the limiting reactant is consumed completely and is the one that determines the extent of the reaction. Here, the *min* function is used;
5. Calculate the number of moles of the combustion products;
6. Calculate the remaining moles of oxygen in the system;
7. Calculate the mole fractions of CO_2, H_2O and O_2 for this reaction;

It is important to notice that the molar fraction of nitrogen is calculated separately at the end of the reactions, due to the fact that it is an inert that is not consumed in none of the reactions.

The code script below exemplifies the procedure for the methane reaction. All the reactions written above were also implemented in the code.

```
1           def calc_chemical_reaction(self) -> float:
2
3           #Methane reaction
4
5           #Calculate available mols of CH4
6           mols_CH4 = (fuel_dictionary['methane'] * total_moles_fuel)
7
8           #Determine stoichiometric ratio
9           CH4_limit = mols_CH4 * (2/1) #2 mols of H2O / 1 mol CH4
10          O2_limit_methane = mols_O2_air * (2/2) #2 mols of H2O / 1 mol CH4
11
12          #  Determine Limiting Reactant
13          limiting_reactant_methane = min(CH4_limit, O2_limit_methane)
14
15          #Calculate mols of combustion products
16          mols_CO2_methane = limiting_reactant_methane * (1/1) #1 mol CO2 / 1
     mol CH4
17          mols_H2O_methane = limiting_reactant_methane * (2/1) #2 mols H2O /
    1 mol CH4
```

```
18
19            #Calculate remaining moles of oxygen
20            reacted_O2_methane = limiting_reactant_methane * (2/1) #2 moles of
    O2 are needed for 1 mol of CH4
21            mols_O2_not_reacted_methane = mols_O2_air - reacted_O2_methane # 2
    mols of CH4 are needed for 1 mol of methane
22
23            total_moles_methane = mols_CO2_methane + mols_H2O_methane +
    mols_O2_not_reacted_methane + mols_O2_air + N2_ng  + mols_N2_air *
    fuel_dictionary['methane']
24            O2_methane = (mols_O2_not_reacted_methane / total_moles_methane) *
    fuel_dictionary ['methane']
25            CO2_methane = (mols_CO2_methane/total_moles_methane) *
    fuel_dictionary ['methane']
26            H2O_methane = (mols_H2O_methane/total_moles_methane) *
    fuel_dictionary['methane']
```

After the composition of the outlet stream of the combustor is calculated, it is applied in the 'combustor fluid' stream setup. This fluid is used together with the total enthalpy of the combustor inlet as inputs of the PHflash method. The total flow rate is calculated by summing the flow rates of the inlet air stream and the fuel stream.

Finally, the outlet temperature of the combustor is calculated after the PHflash calculation. The script for calculating the LCV value of the fuel mixture, as well as the turbine inlet temperature are shown below:

```
1        def calc_LHV(self):
2            ''''
3            Method for determining the lower calorific value of the fuel
    mixture
4            '''
5            iso6976 = ISO6976(self.get_fuel_inlet_stream().get_fluid())
6            iso6976.setReferenceType('mass')
7            iso6976.setVolRefT(float(15.0))
8            iso6976.setEnergyRefT(float(15.0))
9            iso6976.calculate()
10           self.LHV = round((iso6976.getValue("InferiorCalorificValue")*1e3)
    ,3) #J/kg
11
12
13       def calc_outlet_temperature(self) -> float:
14           ''''
15           Method to calculate the outlet temperature of combustor - Turbine
    Inlet Temperature (TIT)
16
17           First, the enthalpy of the air and fuel stream are calculated, and
    then added.
18
19           The fuel used in the PHflash is based on an estimation of the
    composition of the combustor exhaust. Complete combustion is assumed.
20
21           The PHflash method from neqsim was used for calculating the outlet
    temperature, based on the exhaust fluid and total enthalpy of the inlet of
    the combustor.
22           '''
23           enthalpy_air = self.get_inlet_stream().get_fluid().getEnthalpy()
24           enthalpy_fuel = self.get_LHV() * self.get_fuel_inlet_stream().
```

```
    get_flow_rate("kg/hr")/3600
25          enthalpy = enthalpy_air + enthalpy_fuel
26
27          #combustion_fluid = self.get_inlet_stream().get_fluid().clone()
28          combustion_fluid = self.reaction_fluid            combustion_fluid.
    setPressure(self.get_inlet_stream().get_pressure("Pa")/1e5, "bara")
29          combustion_fluid.setTemperature(self.get_inlet_stream().
    get_temperature("K"), 'K')
30          combustion_fluid.setTotalFlowRate(self.get_inlet_stream().
    get_flow_rate("kg/hr") + self.get_fuel_inlet_stream().get_flow_rate("kg/hr"
    ), "kg/hr")
31          combustion_fluid.initProperties()
32          TPflash(combustion_fluid)
33          PHflash(combustion_fluid, enthalpy)
34          self.outlet_temperature = combustion_fluid.getTemperature('K')
```

3.2.2 Off design performance

After running a design case for the gas turbine, the conditions in this model are used for reference for the off design calculations, indicated in the code by the subscript *ref*. The off design iteration process is implemented in Python code as follows:

```
1   while (True):
2
3       iteration = 0
4       max_iterations = 1000
5
6       #Update P2, T3 and MW - Recycle
7       if P2_new > 0 and T3_new > 0:
8           P2_guess = P2_new
9           T3_guess = T3_new
10      else:
11          P2_guess = my_compressor.get_outlet_stream().get_pressure("Pa")
12          T3_guess = my_combustor.get_outlet_stream().get_temperature("K")
13          tolerance_T = 5 #K
14          tolerance = 0.1  #Pa
15
16      while (True):
17
18              T2 = T1 * ((P2_guess / P1 ) ** (R / (Cp * np)))
19
20              #Calculate air fluid properties at T2 and P2_guess
21              air = fluid("srk")
22              air.addComponent("nitrogen", 0.7981)
23              air.addComponent("oxygen", 0.2019)
24              air.setPressure(P2_guess,'Pa')
25              air.setTotalFlowRate(off_design_flow_rate,'kg/hr')
26              air.setMixingRule(2)
27
28              TPflash(air)
29              air.initProperties()
30              enthalpy_air = air.getEnthalpy('J')
31              enthalpy_fuel = LHV * mfuel
32
33              #Total enthalpy
```

```
34                  enthalpy = enthalpy_air + enthalpy_fuel
35
36                  #combustion_fluid = off_design_compressor.get_inlet_stream().
    get_fluid().clone()
37                  PHflash(air, enthalpy)
38                  T3_new = air.getTemperature('K')
39
40                  #Calculating P2 based on new  values of T3 and mfuel
41
42                  off_design_m3 = mfuel*3600 + off_design_flow_rate
43                  P3 = reference_P3 * (off_design_m3 / reference_m3) * sqrt(
    T3_new/ reference_T3)
44                  delta_P = reference_delta_P * (off_design_m3 / reference_m3)
    **1.8 * ((T3_new * reference_P3)/(reference_T3 * P3))**0.8
45                  P2 = P3*(1 + delta_P)
46                  P2_new = P2
47                  iteration = iteration + 1
48                  diff_guess = abs(P2_guess - P2_new)
49
50                  if diff_guess <= tolerance and abs(T3_new - T3_guess) <
    tolerance_T:
51                      break
52
53                  P2_guess = P2_new
54                  T3_guess = T3_new
55                  off_design_compressor.set_outlet_pressure(P2_new, "Pa")
56                  off_design_compressor.calc()
```

Here, the while loop uses the design conditions of outlet pressure of compressor and outlet temperature of combustor as initial guesses for the iteration process. Then, the equations indicated in 2 for the off design conditions calculate the remaining parameters, and resulting in a new outlet pressure of the compressor.

The iteration process stops once the tolerance criteria is met, where the difference between the guessed value and the calculated value is below the tolerance. This pressure is then used for calculating the performance of the compressor at off design conditions, as well as the remaining components of the gas turbine.

3.3 Verification and Validation

Having a reliable model that is able to represent the real behaviour of the gas turbine sets the foundation for further developing the digital twin.

After creating the model in Python, the results are compared in two different ways, to ensure that it is able to represent the correct behavior of a gas turbine, within acceptable deviation. The first part is the verification of the model, which refers to the process of evaluating a software system or component to determine whether it was built correctly, and if it meets the requirements defined prior to the software or component construction, as written in [21].

In this study, the Python model is verified against HYSYS software, to confirm that the equations are correctly written, and that the results match the ones obtained through the HYSYS model.

Also, to verify the model at different operating conditions, the mass flow of fuel, as well as the ambient temperature were modified within a determined range, and

the results were plotted together in the same graph to see the differences between those models.

After verifying the model with HYSYS, the model was ready for validation. In addition to verification, validation is the process of assessing a system or component to ascertain if the model is able to predict the real-world behavior of what is intended to simulate. For this study, two validations were made: software-to-software validation, and validation against real data of an already installed gas turbine, to be selected later.

Software-to-software validation is important to verify if the accuracy and reliability of the model, as well as adding an additional layer of robustness for the model, proving once again that the equations are correctly written in the code. For this validation Thermoflow GTPRO software is used, due to the fact that it is also an already established software for evaluating gas turbine and turbomachinery and it is based on actual data of several manufacturers of these turbomachines.

Verification and validation of the model is critical to ensure that the model is able to predict gas turbine behavior in a way that is reliable and accurate, in a wide range of operating conditions.

3.4 HYSYS model

Aspen HYSYS is an operation unit simulator, used for simulating several operational units in the most varied industries. It is a flexible tool for simulating several different kinds of operation units applied for a wide range of industries following a simple and fast procedure. Despite being a simulator widely used for unit operations, when it is necessary to increase the level of detail of the analysis of complex equipments such as engines or gas turbines, other tools are proven to be more accurate. In this work, HYSYS is used for verification of the model due to its high thermodynamic accuracy prediction.

The gas turbine model chosen for this work was a single spool gas turbine, as described in 2. In HYSYS each main component is created separately, and are connected by one single shaft, as shown below. Since in Aspen HYSYS the combustion modelling is not comprised in the simulator, the combustor part of the gas turbine is simplified by using a Gibbs Reactor, where complete combustion reaction of methane is assumed and set in the reactor, where the products of the reaction are only CO_2 and H_2O.

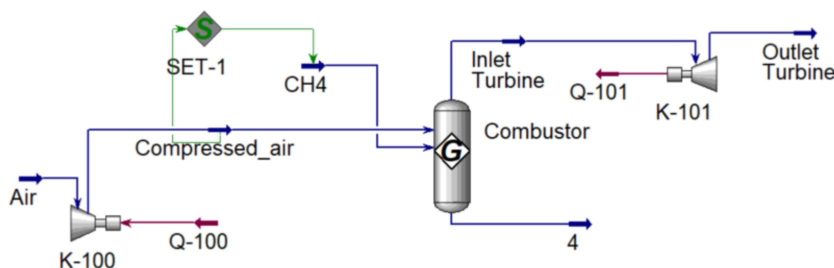


Figure 3.4.1: Aspen HYSYS model.

3.5 Thermoflow Model

Software-to-software validation of the Python model was done using an already established software called Thermoflow, where its development is more focused in engines, turbomachines, gas turbines, and steam cycles in general.Thermoflow is a software suite that comprises several softwares used for modeling and analyzing thermodynamic systems like power plants, cogeneration facilities and other energy-generation related processes. Of all the softwares included in Thermoflow, two of them were used: GT PRO® and GT MASTER ®.

GT PRO ® is a design program for gas turbines, combined cycle, and cogeneration plants. It is based on real gas turbine data from several manufacturers, included in an embedded library with several of them, making it possible to simulate a selected gas turbine in simple or combined cycle, as well as being able to specify the fuel and ambient conditions. User-input thermodynamic criteria combined with manufacturer's data allows GT PRO to generate design parameters for the major equipment.

Once the GT PRO® model is established, another software from Thermoflow is used for evaluating the equipment at off-design. GT MASTER ® takes the design conditions from GT PRO®, and uses this information to evaluate off design performance, taking into consideration variations in ambient conditions, part load performance, and control set-points. It simplifies making several runs, or case studies, starting from the same base case. The base case is simulated in GT PRO®, and the off design simulation is done using GT MASTER®.[19]

After selecting the gas turbine from the software library, one might be able to input specific fuel composition, as well as use the default value. Here, the fuel gas molar composition was given, and was inputted in GT PRO. The fuel data specification is given by the industry:

Component	Molar Percentage (%)
Methane	81.6
Ethane	8.93
Propane	4.24
i-Butane	0.93
n-Butane	1.41
n-Pentane	0.34
n-Hexane	0.35
Nitrogen	0.36

Table 3.5.1: Fuel gas specification inputted in GTPRO.

Once the GTPRO® model was built and run, the design conditions of the gas turbine were established, for the given fuel, defined previously. Thermoflow generates the efficiency, power output, as well as mass flow of air and fuel for the design point.

Off design performance was assessed in Thermoflow GT MASTER®. In these batch runs, ambient temperature varied from 0 °C to 20 °C, and the outputs results are saved in an excel file.

Then, comparison between GT MASTER ® results, and the Python model are made to validate the Python model, in a so-called software-to-software validation. Once the model is validated, it is ready to be used in the digital twin model of the gas turbine.

3.6 Weather Forecast API

Two weather forecast APIs are used in this study, one for historical weather data, and the other for weather forecast data, which will be used for the construction of the digital twin.

Meteostat API will be used for historical weather data retrieval. Meteostat is one of the largest vendors of open weather and climate data, and provides simple access to historical weather data, and it will be used for retrieving historical weather data.

The Meteostat library for Python provides fast and easy access to historical weather data using pandas. This historical observations come from different sources, usually governmental organizations, and are combined through Meteostat's bulk data interface.

First, one must first determine the geographical coordinates of where the gas turbine is placed. Then, a time frame period needs to be defined for observations of the ambient conditions of the geographical location [22]. From there, weather data such as ambient temperature, pressure, and wind speed can be extracted from the API in JSON (JavaScript Object Notation) format.

Afterwards, a list in Python is generated for storing all the necessary information, which will be used for further construction of the digital twin. Here, the connection with the API weather database is shown:

```
1   %pip install meteostat
2
3   #Historical Weather Data
4
5   from datetime import datetime
6   import matplotlib.pyplot as plt
7   from meteostat import Point, Hourly
8
9   #Set time period
10  start = datetime(2023,1,1,12)
11  end = datetime(2023,5,1,12)
12
13  #Create point for Heidrun Platform location
14  Heidrun_platform = Point(65.33, 2.33,16.0)
15
16  #Get Hourly data
17  data = Hourly(Heidrun_platform, start, end)
18
19  data = data.fetch()
20  # data.plot(y= ['tavg','tmin', 'tmax'])
21  # plt.show()
22
```

```
23     data.plot(y=['temp'])
24     plt.show()
25
26     #Extract average temperature values and store them in a list
27
28     tavg_list = []
29     for row in data.itertuples():
30         tavg_list.append(row.temp)
```

For weather forecast data, the open-source API developed by the Norwegian Meteorological Institute named *Locationforecast/2.0 API* is used, in collaboration with *NRK* - Norwegian Broadcasting Corporation. It can provide weather forecast data for any geographical location in the world, using geographical coordinates (latitude and longitude) as input.

This API offers a range of weather parameters, including air temperature, pressure, wind speed and direction, and humidity, among others. It also provides a weather forecast for up to nine days in advance, with hourly updates for the first two days, and three-hourly updates for the remaining forecast period. Additionally, it provides support for historical data, allowing users to retrieve weather data for any location, for a given date and time in the past.

To use this API, an API key needs to be obtained from the Norwegian Meteorological Institute. Once having the key, it is possible to make API requests using HTTP/HTTPS protocols and retrieve data in the chosen format.

Overall, the *Locationforecast/2.0* API is a versatile and complete tool for accessing weather data, which can be used in several applications, from research to operational purposes.

Locationforecast will be used for retrieving weather forecast data for the digital twin construction. The process begins by creating an username to have access to the API, and then using the geographical coordinates of the selected place to retrieve the data. Then, an access access key is generated by the API as an URL link, allowing the user to obtain the necessary data [23].

For this study, the extracted variables needed were the ambient temperature, as well as the day and time. The code snippet of the API connection is shown below:

```
1
2 #Locationforecast API Connection
3
4 import requests
5 from datetime import datetime, timedelta
6
7 # API URL
8 url = "https://api.met.no/weatherapi/locationforecast/2.0/compact?altitude=30&
    lat=65.33&lon=7.32"
9
10 # Define User-Agent header
11 headers = {
12     'User-Agent': 'vmkaplan/PowerGeneration github.com/vmkaplan/PowerGeneration
    ',
13     'Contact': 'victoriakaplan@hotmail.com'
14 }
15
16 # Specify the time range for which you want to retrieve weather data
```

```
17 now = datetime.now()
18 start_time = now
19 end_time = start_time + timedelta(days=10)
20
21 # Make GET request to API
22 response = requests.get(url, headers=headers)
23
24 # Extract JSON response from the response body
25 data = response.json()
26
27 # Extract time and temperature values
28 timeseries = data['properties']['timeseries']
29 current_time = now.strftime('%Y-%m-%dT%H:%M:%SZ')
30
31 time_period = []
32 forecast_temperature = []
33 time_plot = []
34
35 for ts in timeseries:
36     time = ts['time']
37     temperature = ts['data']['instant']['details']['air_temperature']
38
39     # Check if the time is equal to or greater than the current time
40     if time >= current_time:
41
42         time_object = datetime.strptime(time, '%Y-%m-%dT%H:%M:%SZ')
43         day = time_object.day
44         month = time_object.month
45         year = time_object.year
46
47         time_plot.append(f'{day}-{month}-{year}')
48         time_period.append(time)
49         forecast_temperature.append(temperature)
```

3.7 Digital Twin

The last part of this work is the actual construction of the digital twin. After the verification and validation of the model, as well as the creation of the API connection with the Python code, the digital twin can be built.

Following the principles of a hybrid digital twin described in chapter 2, the approach is used in this study, combining physical model with a data-driven model.

In a real operation, the weather forecast is connected to the live process, and it is used as an input of the Python code. Air and fuel stream conditions are also input parameters of the code, as it is assumed that they are measured parameters of the real process.

Afterwards, the output results of power and efficiency of the gas turbine will be the results of the digital twin model.

CHAPTER
FOUR

RESULTS AND DISCUSSION

4.1 Design Model

The gas turbine equations described in chapter 3 were implemented in Python using object oriented programming, with the methods also described in chapter 3. After building a running model of the gas turbine, verification and validation was performed, for both design and off design performance.

Verification of the model is made for confirming that the equations used for thermodinamically describe the compressor, combustor, and expander behavior is correct, with all the simplifications as described in chapter 3.

In the verification case, a design case was formulated and the parameters were inserted in the Python and HYSYS model. The design case is shown in table 4.1.1, which are the basis for building the *design* case of the simple spool gas turbine.

	Value
Air mass flow*	500 kg/s
Ambient Temperature	288.15 K
Ambient Pressure	1.013 bar
Compressor Pressure Ratio	10.7
Compressor Isentropic Efficiency	85.8 %
Methane mass flow**	5 kg/s
Combustor Pressure Loss	1.5 %
Turbine pressure loss	45 mbar
Turbine Isentropic Efficiency	88.4 %

Table 4.1.1: Model comparison setup. * Air composition of 79.81 % N_2 and 20.09 % of O_2. ** for this first verification, the fuel composition was simplified to 100 % CH_4.

The comparison between the output results from Python and HYSYS are

shown below in table 4.1.2. The difference of the variables between both softwares, and Python model outputs is calculated the following way, using equation 4.1:

$$\text{Variable} = \frac{\Delta(\text{Software} - \text{Python})}{\text{Software}} * 100 \tag{4.1}$$

Where *Software* is the results taken from HYSYS and Thermoflow software, and *Python* represents the values calculated from the Python model. This formula is used for all the comparison results, and further discussion.

Parameter	Python	HYSYS	Difference
Outlet Temperature Compressor - K	611	601	1.58
Polytropic Efficiency Compressor	0.896	0.894	0.23
Cp Inlet Stream Compressor - J/kgK	1013	1009	0.4
Compressor Work - MW	163.58	163.5	0.05
Turbine Inlet Temperature (TIT) - K	1007	1026	1.85
Outlet Temperature Turbine - K	612	622	1.66
Polytropic Efficiency Turbine	0.849	0.847	0.16
C_p Inlet Stream Turbine - J/kgK	1171	1181	0.85
Turbine Work - MW	230.6	229.77	0.36
Gas Turbine Power Output - MW	66.07	65.47	0.92
Gas Turbine Overall Efficiency - %	26.41	26.17	0.93

Table 4.1.2: Comparison of Python and HYSYS model outputs.

The main differences between those models are related to the outlet temperature of compressor and turbine. This is related to the equations used for calculating these outlet temperatures, which are using the isentropic temperature, as shown in 2. Instead of using enthalpies in the calculations of the actual outlet temperatures, isentropic temperatures and kappa values read from Neqsim are used to determine the outlet temperature.

The other difference is also related to the outlet temperature of the combustor comparison, which affects the calculation of the exhaust temperature of the turbine. Since this calculation in Python is performed by a simplification using Neqsim, as stated in chapter 2, the accuracy of the calculation of this temperature is affected by this simplification.

Since the equations and calculations performed by Aspen HYSYS to determine these temperatures are not publicly disclosed, it becomes difficult to point out precisely the equations used. This way, the exact reasons for these differences in temperature results remain uncertain.

4.2 Off-design scenarios

Since the main goal of this study is to observe the impact of ambient temperature over the power output and efficiency and gas turbines, the model has to run at off-design conditions as well. Here, three off-design scenarios were used, where the only parameter changing was the ambient temperature. The temperatures selected were 0 °C, 15° C, and 30° C.

Each ambient temperature scenario is defined as a case study, and the results are presented in the following graphs. The gas turbine power output varied as a function of the mass flow of methane, in order to better observe if the impact of ambient temperature is significant in a gas turbine power generation.

Even though at 15° C the gas turbine is operating at design conditions, this temperature was also used in the off design calculation performance to verify if the iteration procedure described in 3 is valid, and can be used as a method for calculating off-design performance of the gas turbine.

4.3 Verification

For the verification against HYSYS to be accurate and reliable, key parameters were selected for comparison with different mass flow rates of fuel, following the procedure for the case study setup described in section 4.2 above.

It can be seen in figure 4.3.1 the gas turbine power generation of the gas turbine is higher than for higher ambient temperatures, which is behavior consistent with reality. Colder ambient temperatures increase ambient air density, allowing more mass flow of air to enter the combustion chamber for the same compressor speed, increasing the total mass flow of the air-fuel mixture, consequently increasing power output.

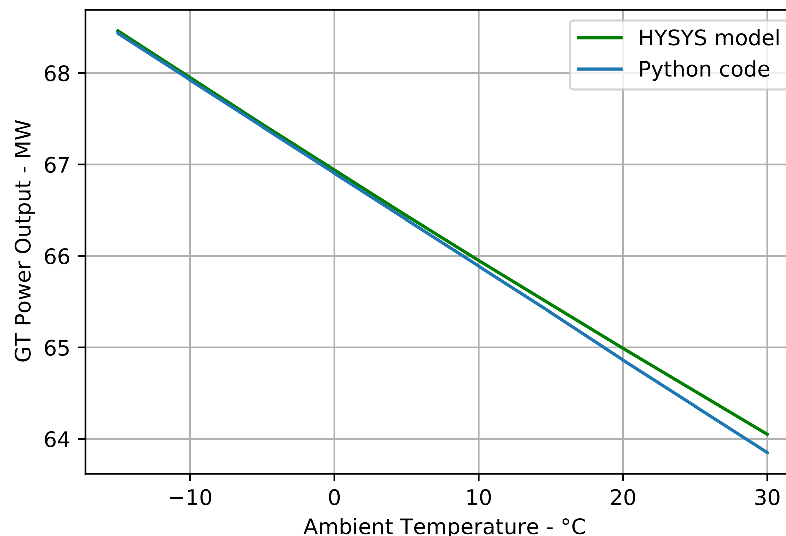


Figure 4.3.1: Power Output variation with Ambient Temperature. Green represents the values generated from HYSYS model, and the blue line are the outputs from the Python mode.

The gas turbine efficiency is also higher for colder temperatures. As seen in

figure 4.3.2, the higher power output the gas turbine, the efficiency grows proportionally, for the same mass flow of fuel.

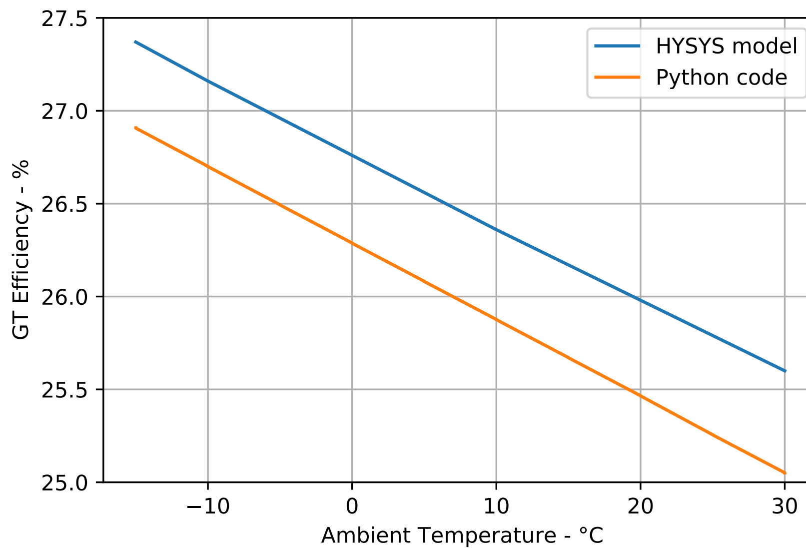


Figure 4.3.2: Efficiency of gas turbine versus Ambient Temperature. Blue line represents the values extracted from HYSYS model, and orange represent the Python model results.

On all three cases, the results of power output of the gas turbine and efficiency were plotted against each other. From figures 4.3.3, 4.3.4 and 4.3.5 it is possible to see that the blue curve representing the Python model generates similar values when compared to HYSYS model outputs, as seen in the red curves. Therefore, the equations implemented in the Python model are consistent with HYSYS and can be used for modeling the gas turbine.

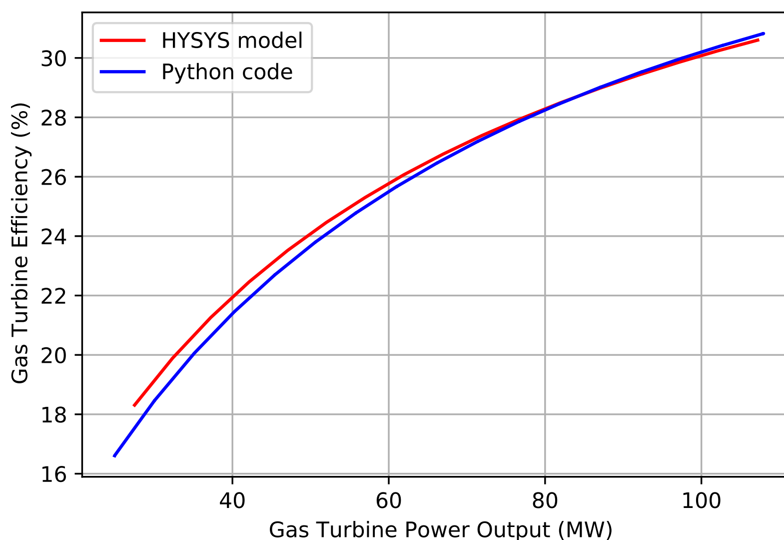


Figure 4.3.3: Gas Turbine Efficiency vs. Power Output at 0° C Ambient Temperature.

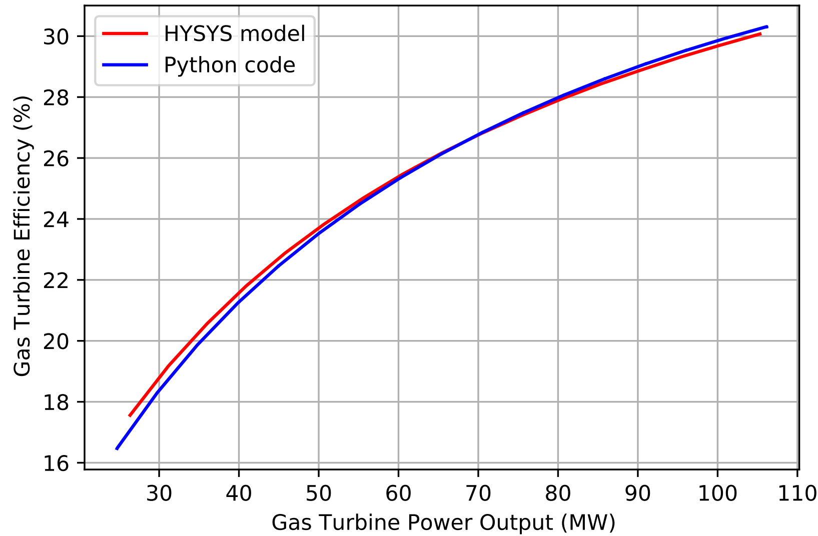


Figure 4.3.4: Gas Turbine Efficiency vs. Power Output at 15° C Ambient Temperature.

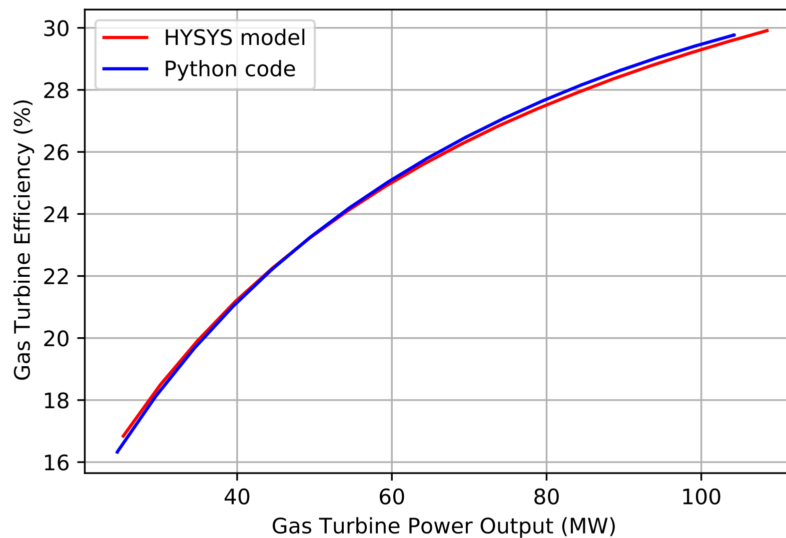


Figure 4.3.5: Gas Turbine Efficiency vs. Power Output at 30° C Ambient Temperature.

Nonetheless, from turbine exhaust temperatures comparison in figures 4.3.6, 4.3.7, and 4.3.8 the differences are perceptible, but still within reasonable difference. The yellow lines representing the Python model show lower values in comparison with the blue lines, showing HYSYS model outputs. It is also possible to observe that the variation decreases the higher the ambient temperature, which could be to the thermodynamic behavior of the gas turbine.

Gas turbines are influenced by the properties of the working fluid, mainly air, and its thermodynamics. At high temperatures, the specific heat capacity (C_p) of air decreases, which consequently affects the temperature rise in the combustion process of the gas turbine.

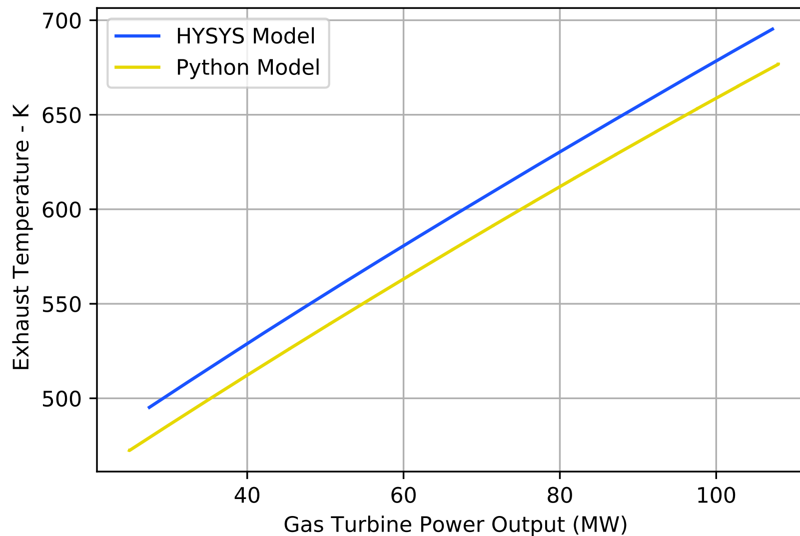


Figure 4.3.6: Turbine Exhaust Temperature $T_{amb} = 0$ °C.

And since the combustion process is simplified using the PHflash method from Neqsim as stated in chapter 3, the turbine exhaust temperature is also influenced by this temperature. The air density decreases at higher temperatures, which can lead to more consistent combustion characteristics, lowering the difference between the two simulations.

It also shows that the combustion simplification approach in both HYSYS and Python are persistent, and that the PHflash method can be used for estimating the combustion outlet temperature, or turbine inlet temperature (TIT).

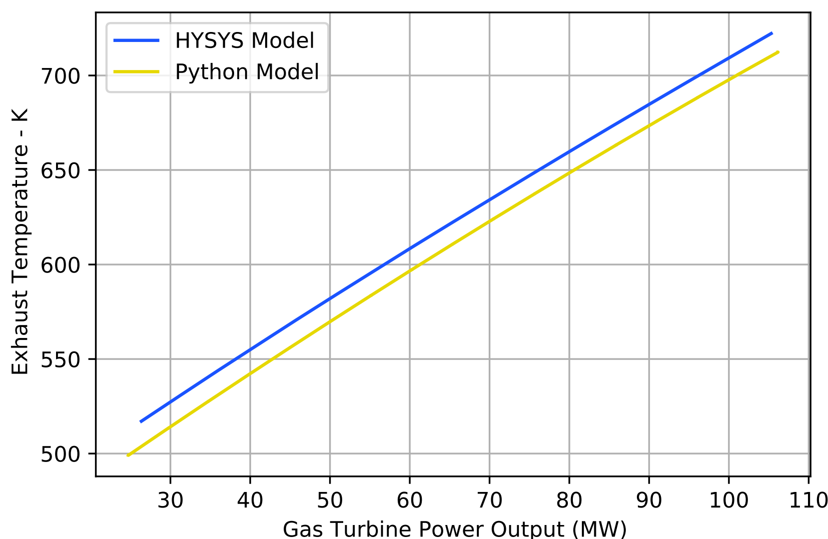


Figure 4.3.7: Turbine Exhaust Temperature $T_{amb} = 15$ °C.

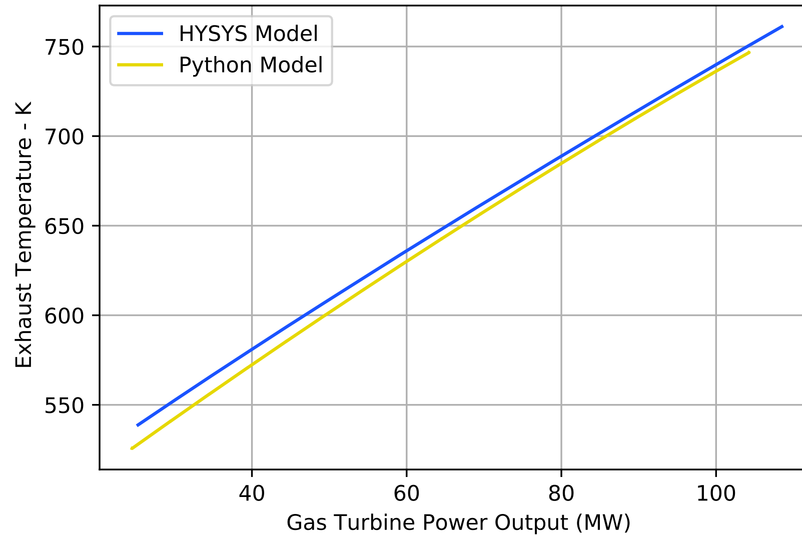


Figure 4.3.8: Turbine Exhaust Temperature $T_{amb} = 30$ °C.

Moreover, the turbine exhaust temperature is also tied with the compressor performance. Higher ambient temperatures tend to lower the compressor efficiency, consequently increasing the turbine exhaust temperature.

Additionally, when comparing the outlet temperature of the compressor, the Python model yields higher values for all three cases in this comparison.

This discrepancy may stem from differences in the methods employed by the two models to calculate the outlet temperature. The Python code employs a more straightforward approach using the isentropic temperature to calculate the outlet temperature. Although the exact methodology used by Aspen HYSYS remains unclear, it might involve solving the energy equation for the compressor.

It is not clear up to this point the exact method used by Aspen HYSYS, but it is relevant to notice this difference in the output results, which can affect the work further.

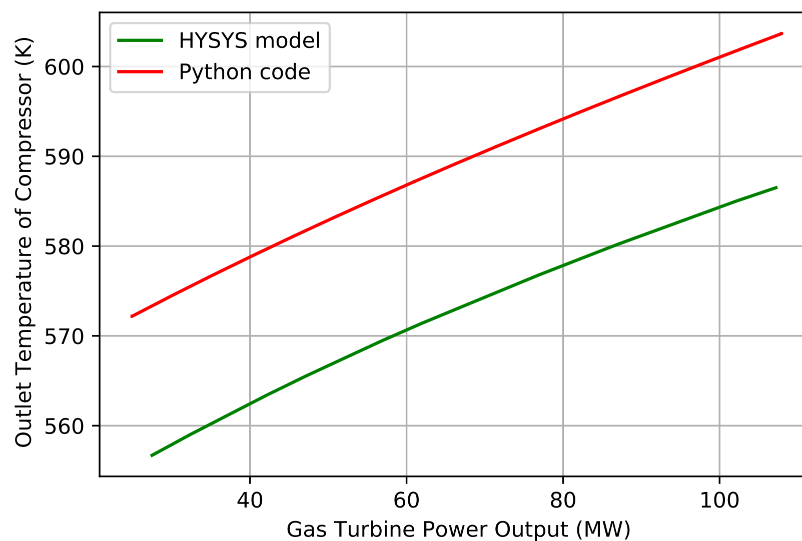


Figure 4.3.9: Compressor Outlet Temperature - $T_{amb} = 0$ °C.

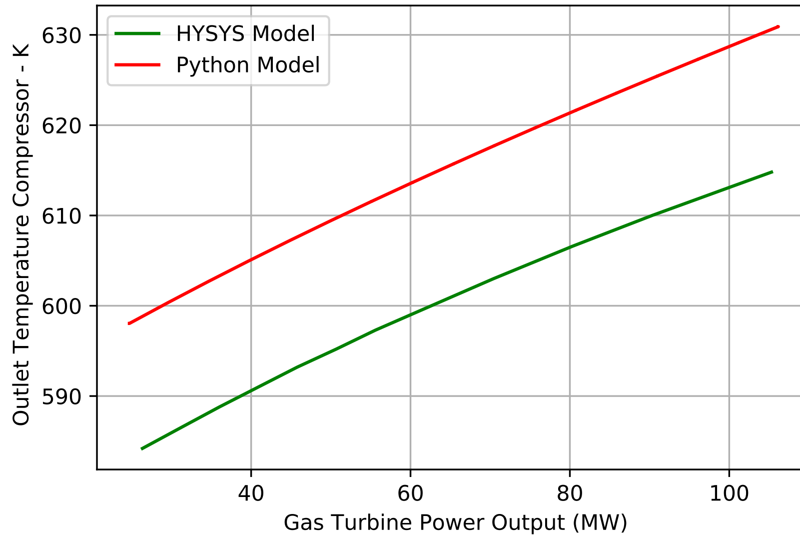


Figure 4.3.10: Compressor Outlet Temperature - $T_{amb} = 15$ °C.

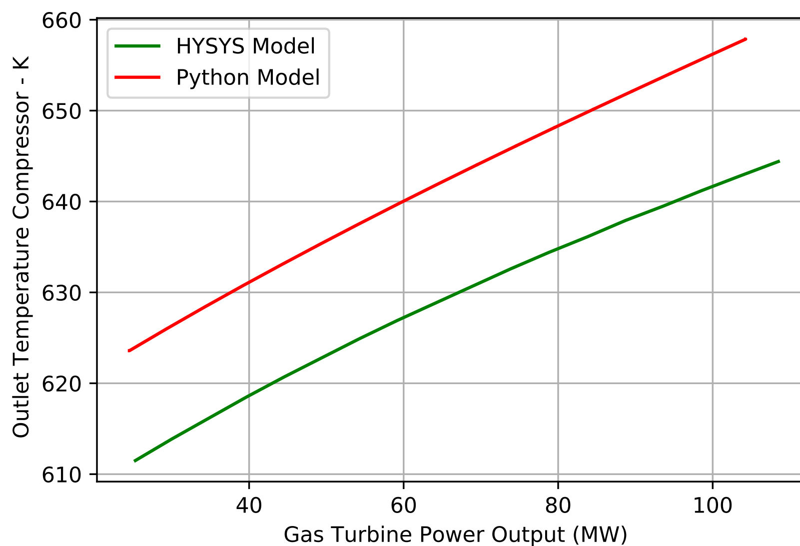


Figure 4.3.11: Compressor Outlet Temperature - $T_{amb} = 30$ °C.

Overall, the model verification proved that the equations used are coherent and can be used for modelling a gas turbine. The Python model established exhibits a robust and consistent behavior for projection of gas turbine power output and efficiency, for different ambient temperature conditions.

4.4 Validation

Once the verification of the Python was established, the validation of the model could be done. This way, the model could be a robust representation of a gas turbine, and then can be compared to another software for further validation.

The software-to-software validation was divided into two parts: validation using design conditions, and then moving on to off-design performance study.

Validation of the software model needed to begin with a selection of a real gas turbine, used in power generation applications. Here, information about the Siemens SGT A-35 was used for the definition of the parameters described below, with a combination of manufacturer data from Thermoflow and company data.

With a compact design, while still maintaining high power generation capacity, the Siemens SGT-A35 is a reliable gas turbine designed for power generation and mechanical drive applications. It meets up the quality standards for security and reliability in the gas industry for both onshore and offshore applications, as well as for several other industries.

It is an aeroderivative gas turbine that can withstand a wide range of operating conditions, and can be easily installed, reducing maintenance costs and downtime. For power generation, can both be used in simple cycle or combined cycle generation, explained in detail on 2, while also used in mechanical drive applications, with power outputs ranging from 30 MW to 39 MW.

The Siemens SGT-A35 RB is a three-shaft gas turbine consisting of two compressors and two turbines, as well a combustion chamber. This configuration allows the equipment to have more operational flexibility, during part-load performance especially, while still maintaining high efficiency and low emissions.

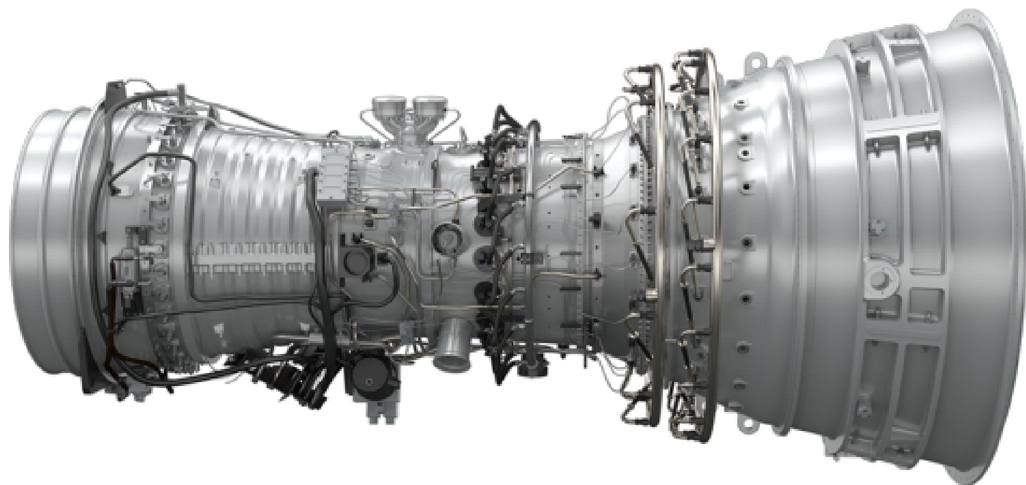


Figure 4.4.1: Siemens SGT A35 RB overview. [24]

A simplified diagram for the gas turbine showing the main parts is shown in figure

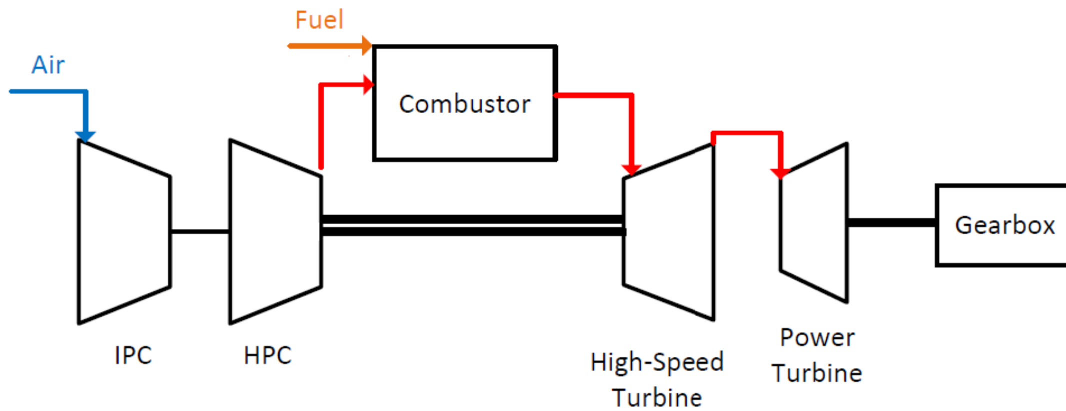


Figure 4.4.2: Siemens SGT A35 RB simplified diagram.

In this case, the SGT A-35 RB include the Siemens RB 211 gas generator, together with the RT 62 gas turbine. Here, air passes through the air plenum into the intermediate-pressure compressor (IPC), and then is directed to the high pressure compressor (HPC) for additional compression, where it is then sent into the combustion chamber.

There, the air mixed with the fuel is ignited, producing a high pressure, high temperature stream of hot gases. These high stream energy drives the high-speed power turbine, generating electricity. The hot gases are directed to the low-speed gearbox mounted turbine RT 62, where the energy is used to drive a mechanical load connected to the gearbox.[6]

Within mechanical drive applications, there are six different types of SGT A-35 gas turbine, varying in power generation, frequency, and presence of Dry Low Emissions (DLE), a specific combustion system designed for lowering and drying NO_x emissions. For the purposes of this study, the performance data of the gas turbine used as design conditions is shown below:

Component	Molar Percentage (%)
Pressure Ratio -	21.7
Exhaust Flow - t/h	345
LHV Efficiency - %	36.32
Gross Power Output - MW*	29.07
Exhaust Temperature - C	506
LHV Heat Rate - kJ/kWh	9912

Table 4.4.1: Design conditions of SGT A35 - extracted from GTPRO and [24]. *Power Output at Generator Terminal.

In order to properly model the gas turbine, additional variables needed to be

determined. One of the variables was the compression ratio of the two-compressor system. Since the isentropic efficiency was the same for all compression stages, the pressure ratio in each stage was calculated using the formula [25]:

$$r = r_t^{\frac{1}{n}} \tag{4.2}$$

Where r is the pressure ratio in each stage, r_t is the total pressure ratio of the compressor system, and n is the number of stages. Here, company data stated that the intermediate pressure compressor(IPC) consisted of 7 compression stages, while the high pressure compressor(HPC) was built of 6 compression stages.

By applying equation 4.2 the pressure ratio for IPC compressor was 5.24, and 4.19 for the HPC compressor.

Air composition from Thermoflow was given and used in the Python code, as shown below:

Component	Molar Percentage (%)
Nitrogen	75.80
Oxygen	14.10
CO_2	3.35
Argon	0.91
Water	5.82

Table 4.4.2: Air composition, extracted from GTPRO (considering 0% relative humidity)

The comparison between Thermoflow results and Python code revealed more uncertainties than expected. Despite Thermoflow providing accurate results of a gas turbine based on manufacturer data, many parameters are not publicly displayed, making a simulation of a gas turbine a challenge. Therefore it was deemed necessary to obtain such parameters using a different approach, necessary for running the model in Python.

For the Python model construction, the main parameters not shown in Thermoflow were the isentropic efficiencies of compressor and turbine, necessary to determine outlet temperatures, among other variables stated in the chapter 2. To determine the isentropic efficiencies, two approaches were selected: one was by estimating initial values of those efficiencies and iterating them manually until the power output and efficiency matched the results from Thermoflow. The second option was to use another simplified software for better estimation of these efficiencies.

It was quickly realized that manual random iteration was not efficient nor accurate, therefore the software GasTurb 14 was used for isentropic efficiency determination. GasTurb is a software used for gas turbine calculations, where it can provide fast and accurate prediction of gas turbine performance based on generalized compressor and turbine maps. It requires few inputs to generate data, and it was used to provide an estimate of the compressors and turbines efficiencies that are part of the gas turbine system.

For the simulation setup in GasTurb was made using Simple Cycle design, using the Booster Turboshaft HP Spool design, representing the three-shaft gas

turbine, connected with a power turbine, as shown in figure 4.4.2. The following inputs from Thermoflow were used for the GasTurb model:

Property	Value
Ambient Pressure - kPa	101.325
Ambient Temperature - K	288.15
Absolute Inlet Pressure Loss - kPa	0.249
Absolute Exhaust Pressure Loss - kPa	1.245
Pressure Ratio Booster Compressor *	5.17
Pressure Ratio HP Compressor *	4.14
Burner Exit Temperature - K	1500
Fuel LHV - MJ/kg	46.7
Burner Pressure Ratio	0
Generator Efficiency	0.9801
Mechanical Efficiency	0.9902

Table 4.4.3: GT PRO inputs to GasTurb. * Booster Compressor is the Intermediate Pressure Compressor represented in the gas turbine setup, and HP stands for high pressure compressor.

The only variables missing for completing the simulation were the compressors and turbines isentropic efficiencies. The initial estimate of those efficiencies was 0.88 for compressors, and 0.9 for turbines. The simulation was made according to the following procedure for discovering relations between those efficiencies to match the parameters based on Thermoflow.

First, the parameters from 4.4.3 were used in GasTurb, as well as an initial estimate of the compressors and turbine efficiencies. Then, using the iterations feature from GasTurb, the following iterations were performed until the isentropic efficiency values for each component converged.

The iteration setup shown in table 4.4.4 shows the variables modified, and the target values. Target values in this model are based on design conditions from Thermoflow. Once the efficiency values converged, they were used as inputs into the Python model, and the outputs were compared with Thermoflow outputs.

Variable	Target
Inlet Corrected Mass Flow *	Exhaust Flow
Isentropic HPC Efficiency	Shaft Power Delivered
Isentropic Efficiency Power Turbine	Power Turbine Exhaust Temperature
Burner Exit Temperature	Fuel Flow

Table 4.4.4: Iterations setup on GasTurb.

The iterations estimated that the isentropic efficiencies of the booster compressor, HP compressor, HP turbine and power turbine to be, respectively, 0.89, 0.84, 0.887, and 0.887. Here, the isentropic efficiencies of both turbines are assumed to be the same. The Python model comparison with the Thermoflow model, as well as the differences for the design conditions is shown below:

Variable	Thermoflow Output	Python Model	Difference - %
Gas Turbine Efficiency - %	36.32	36.33	0.04
Gross Power Output - MW	29.07	29.01	0.19
Exhaust Temperature - K	779.15	721.15	11.46

Table 4.4.5: Comparison of Thermoflow and Python model for design conditions. GasTurb setup: Booster Turboshaft HP Spool

Gross power output is the gas turbine power output at the generator terminal, considering all efficiencies, mechanical and from the generator.

During design conditions comparison, two compressors and two turbines were considered in the analysis. For the off-design evaluation, the analysis was simplified into one compressor and one turbine, in a single-spool shaft as illustrated in figure 3.1.1. This saves computation time on the off-design iteration loop , and was also studied to evaluate if it was possible to model a gas turbine accurately by doing this simplification.

The GasTurb procedure was followed for the off design condition, only now the gas turbine is assumed into a single-spool shaft. The isentropic efficiency of the compressor and turbine then converged to the values of 0.84 and 0.87. Assuming constant polytropic efficiency for the off design calculations, as written in chapter 3, the off design calculations were made using the same iteration procedure as for the HYSYS verification. The comparison of the main outputs are shown in the following graphs.

Starting with the air flow rate, in figure 4.4.3 it is possible to see the differences in both results from those models. Since Thermoflow calculates the air flow rate based on compressor performance map, the flow rate relation with the ambient temperature is not as linear as it is determined in the Python model. The different methodologies for calculating the air flow rate for off design operating conditions explain the difference in results of those models.

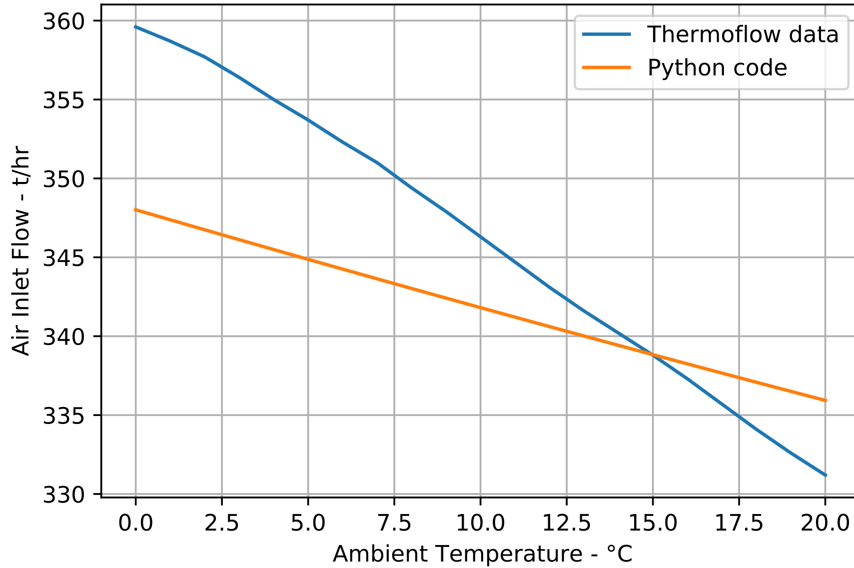


Figure 4.4.3: Air flow for off design conditions comparison.

Nevertheless, the results show that in both cases, the inlet air flow decreases as the ambient temperature increases, which makes the results from Python consistent with Thermoflow outputs, despite the difference on the values. It shows that, for a robust model of a gas turbine, the off design method for determining the mass flow of air as shown in chapter 3 in the system can be used.

The model in Python also follows the same pattern as Thermoflow results for power output and efficiency, as seen on figure 4.4.4. Here, it also shows that the efficiency is directly proportional with the power output of the gas turbine, and it grows smaller for high power outputs.

However, despite following the same pattern, there are still differences in the results which need to be further investigated. One of the reasons for these differences is from the methods used for calculating power and efficiency, where in the power calculation the temperatures are used, as well as specific heat capacity C_p is a property of the fluid, read from Neqsim.

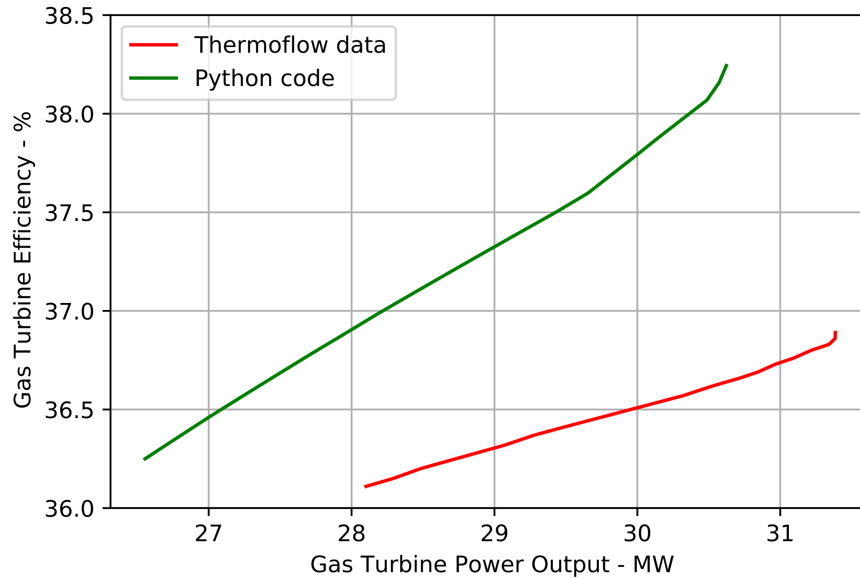


Figure 4.4.4: Power and efficiency output results for Thermoflow and Python.

To further investigate the reasons behind these variations, the power and efficiency calculated were plotted separately, against the ambient temperature.

From figure 4.4.5, the power outputs results show that the Python and GT MASTER® models have similar values for power output. The deviance decreases as the ambient temperature increases, and both models generate approximately the same value for the design conditions, as shown in table 4.4.5. This is consistent for the design point, and the behavior can be expanded to the off-design conditions.

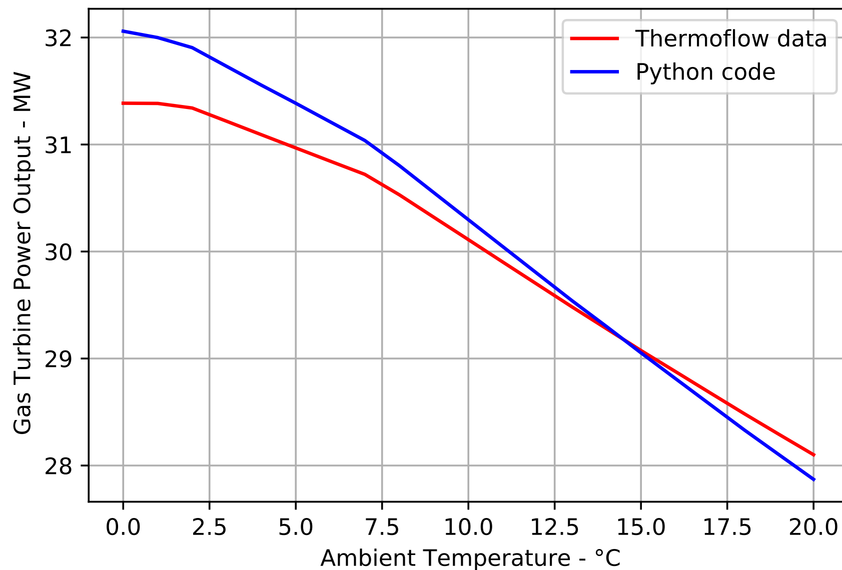


Figure 4.4.5: Gas turbine power results comparison.

Nonetheless, the efficiency results shows higher discrepancies, where the Python code gives higher values for efficiency when compared to Thermoflow. This is connected to the power output difference, where the method for determining it uses

the temperature difference across the turbine and compressor, as previously explained in chapter 2.

Given the number of simplifications and assumptions used for the construction of this model, the differences were considered acceptable, and the model has been successfully validated against Thermoflow software suite.

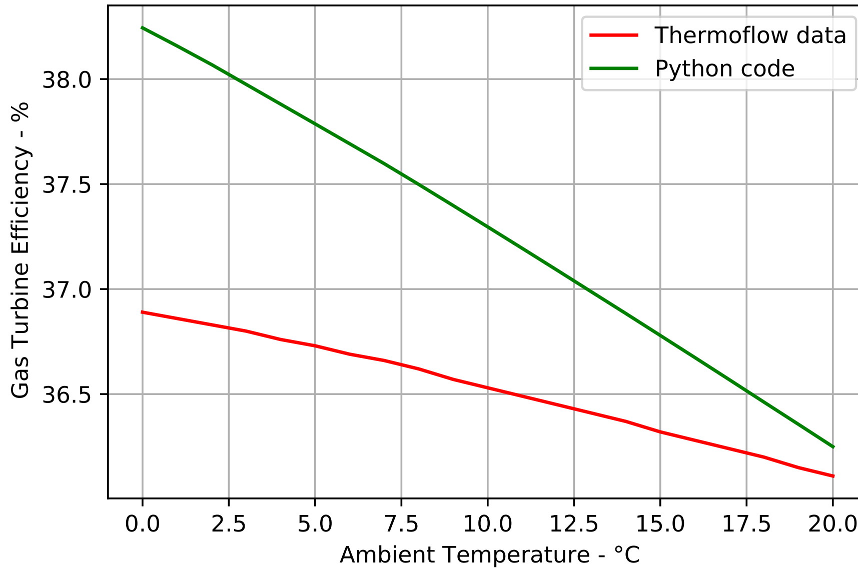


Figure 4.4.6: Gas turbine efficiency results comparison.

Those results from this validation now provide evidence that the Python model developed is robust, because it can be used for different off-design conditions of operation. Since in this case the off-design scenarios are based on changes in ambient temperature only, the model proved to simulate accurate results, and can be used inside the digital twin model.

4.4.1 Limitations of the model

The exhaust temperature from both models are different from each other, and this difference follows the off design performance evaluation.

Despite including iterations inside the turbine class to improve temperature calculations, the temperature results from Python still have great deviations when compared to Thermoflow outputs, where it is a limitation of the model.

Uncertainties on calculations of turbine exhaust temperature were observed during the comparison between Thermoflow and Python model. This can be explained due to the Python model uses isentropic temperatures and assumes constant properties of the fluid to calculate the actual outlet temperature of the turbine, whereas Thermoflow bases itself on real data given by the manufacturer.

There, most probably the enthalpies were considered in the calculations, as well as solving directly the energy equation. It can also impact on efficiency calculations of the gas turbine, justifying the larger difference when compared with Thermoflow.

This was a limitation observed on the model, therefore the turbine exhaust temperature was not a parameter used for validation of the model.

The comparison results for exhaust turbine temperature can be seen on figure 4.4.7.

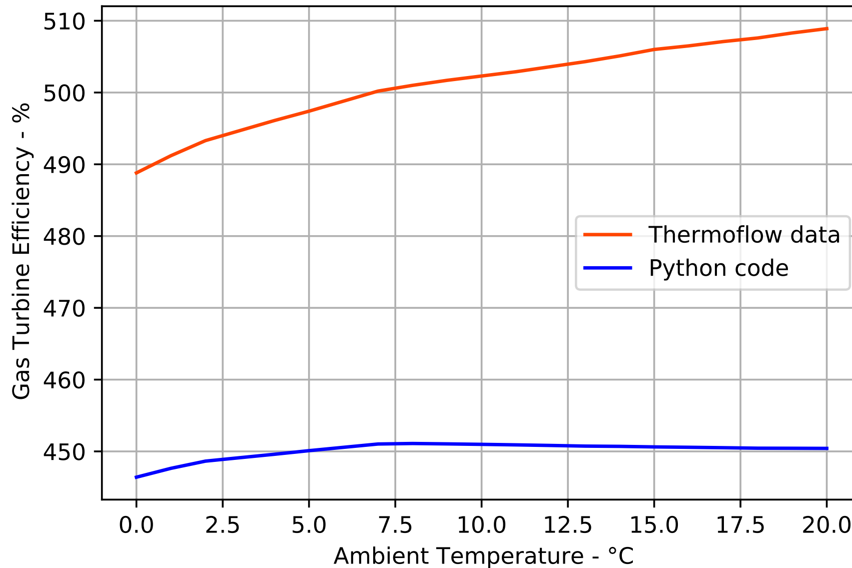


Figure 4.4.7: Exhaust turbine temperature results comparison.

4.5 Weather API connection

As stated in 3, the relation with the API forecast begins with the geographical coordinates of the place, as well as a definition of a time period. The geographical place chosen was of an offshore oil field located in the Norwegian Sea, and the coordinates were extracted from GeoHack website [26].

The coordinates are shown below in table 4.5.1:

Latitude	Longitude
65° 19' 33" N	7° 19' 3" E *
65.325833	7.3175 **

Table 4.5.1: Geographical coordinates of the offshore oil field. * Coordinates written in DMS (degrees, minutes, seconds) format. **Coordinates written in decimal format.

The time period chosen was from 1st of January of 2023 until 1st of May of the same year. This large time period was chosen to better observe the influence of ambient conditions in the gas turbine performance.

Once the coordinates and time stamp are defined, the data can be extracted from Meteostat API. From there, the average temperature values need to be defined in the API, and are stored into a list to be further added as input to the digital twin. The data extraction made in Python is shown in the code snippet, and the temperature values for the time frame selected are plotted in figure 4.5.1 below:

```
#Historical Weather Data

from datetime import datetime
import matplotlib.pyplot as plt
from meteostat import Point, Hourly

#Set time period
start = datetime(2023,1,1,12)
end = datetime(2023,5,1,12)

#Create point for Heidrun
Heidrun_platform = Point(65.33, 2.33,16.0)

#Get hourly dat
data = Hourly(Heidrun_platform, start, end)
data = data.fetch()
data.plot(y=['temp'])
plt.show()

#Extract tavg values and store them in a list
tavg_list = []
for row in data.itertuples():
    tavg_list.append(row.temp)

print(tavg_list)
```

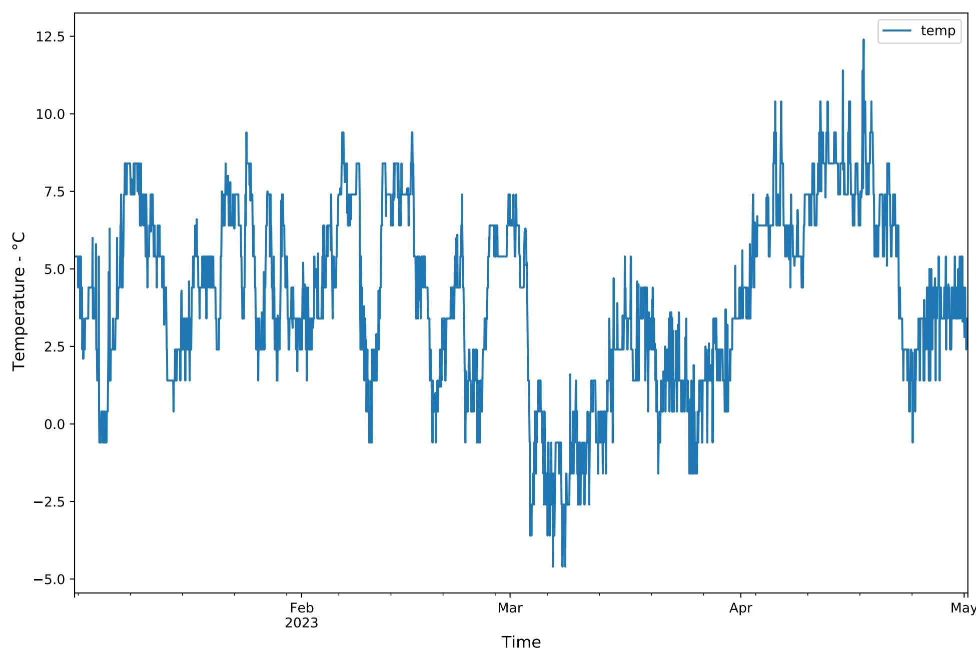


Figure 4.5.1: Average temperature weather historical data for the geographical location selected.

4.6 Digital Twin

It is necessary to establish a connection with physical equipment to create a digital copy of the object while attempting to simulate its behavior. This connection can be made in the form of data gathering from measurements on-site, measuring accurately in defined intervals of critical parameters to ensure the best overview of equipment operation.

Here, the data source for simulating a digital twin was the Thermoflow database. Since it is software based on manufacturer data (as stated in 3), it was used to extract data from the gas turbine based on experiments and simulations.

For the construction of this digital twin, the main inputs for the model are the ambient temperature and the fuel flow rate entering the combustion chamber. The assumption of a gas turbine operating at full load, with constant pressure drops at both inlet and exhaust, is maintained in this case, as well as no pressure drop inside the combustion chamber.

Considering that the source for ambient temperature data comes from the weather forecast API, the fuel flow rate data needs to be established. Since in this work real gas turbine field data was not used, it was decided to find a relation between fuel flow rate and ambient temperature.

A linear regression was performed using results from GT MASTER for different ambient temperatures, resulting in different flow rates of fuel. The linear regression was made in Python by using the statsmodel library. Statsmodel library is a Python module that contains methods for the estimation of different statistical models, among other features for statistical data exploration.

Using Statsmodel, the regression mode has different methods for data analysis. The OLS method (Ordinary Least Squares) was used for finding a relation between fuel intake in a gas turbine and ambient temperature.

First, the Python module is installed to access its methods. Later, the data set extracted from off-design scenarios run on GT MASTER for each variable is defined into two Python lists. The model is then fit into the data using the OLS method, and the results for the regression are shown after. The code snippet summarizing the statistical analysis is shown below:

```
1  import statsmodels.api as sm
2  import numpy as np
3  import pandas as pd
4  import matplotlib.pyplot as plt
5
6  # Linear regression relating fuel intake with temperature
7
8  fuel_flow_rates = [1.82, 1.82, 1.82, 1.82, 1.81, 1.80, 1.80, 1.79, 1.78, 1.77,
       1.76, 1.75, 1.74, 1.73, 1.72, 1.71, 1.70, 1.69, 1.68, 1.67, 1.67] #kg/s
9
10 ambient_temperatures = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ,11 ,12 ,13 ,14 ,15
       ,16 ,17 ,18 ,19 ,20] #C
11
12 # Fit the model
13 X = sm.add_constant(ambient_temperatures)
14 model = sm.OLS(fuel_flow_rates, X)
15 results = model.fit()
16
17 # Plot the data and the fitted line
```

```
plt.scatter(ambient_temperatures, fuel_flow_rates)
plt.show()
```

The summary table for the OLS regression is shown below in table 4.6.1:

Parameter	Coefficient
R^2	0.980
Adjusted R^2	0.979
F-statistic	953.7
Prob (F-statistic)*	1.06e-17

Table 4.6.1: OLS Regression Results.

These indicators confirm that the regression is appropriate. The R-square (R^2) value measures how well the data fit the model, indicating the proportion of the variance of the dependent variable (in this case the fuel flow) that can be explained by the variation of the independent variable (ambient temperature). A high R-squared value suggests that the regression is an appropriate fit for the data while capturing most of the variability.

The Adjusted R-squared value (Adj. R^2) considers the number of predictors in the model while adjusting R^2 value accordingly. Here, the adjusted R^2 value is 0.979, close to the R^2 value, indicating that the fuel flow rate has a strong relationship with the ambient temperature.

Now, the F-statistic and Prob F-statistic (P-value) test the overall significance of the model, assessing whether there is a significant linear relationship between the independent and dependent variables. The p-value allows observing extreme F-statistic values. Here, with a high F-statistic value (953.7) and a small p-value (1.06e-17) indicates that the regression model is statistically significant.

Based on these indicators, the regression model is a good fit for the data. Then, the overall equation relating fuel intake in a gas turbine and ambient temperature is shown below:

$$F = 1.8403 - 0.0086 * T \tag{4.3}$$

Where F is the fuel flow rate in kg/s, and T is the temperature in °C. With all the external sources, the digital twin model can be run. The ambient temperature data and the equation relating fuel flow with ambient temperature are used as inputs of the model, and the Python code for off-design scenarios is run for this set of data. The code below shows part of the structure of how this was set in the Python environment:

```
    ambient_temperature_list = tavg_list #List with data from the weather forecast API
    time_period = timestamps

    for i in range(len(ambient_temperature_list)):
        t = ambient_temperature_list[i]
        fuel_flow = 1.8403 - 0.0086 * t
```

```
7           fuel_flow_rates.append(fuel_flow)
8       fuel_inlet = fuel_flow_rates
9
10      for t, mfuel in zip (ambient_temperature_list, fuel_inlet):
11      #Off-design loop continued ....
```

The power output estimation for the time frame selected is shown below. This time period selected is used for retrieving historical data only. It is possible to observe that the power generation is impacted by the ambient conditions.

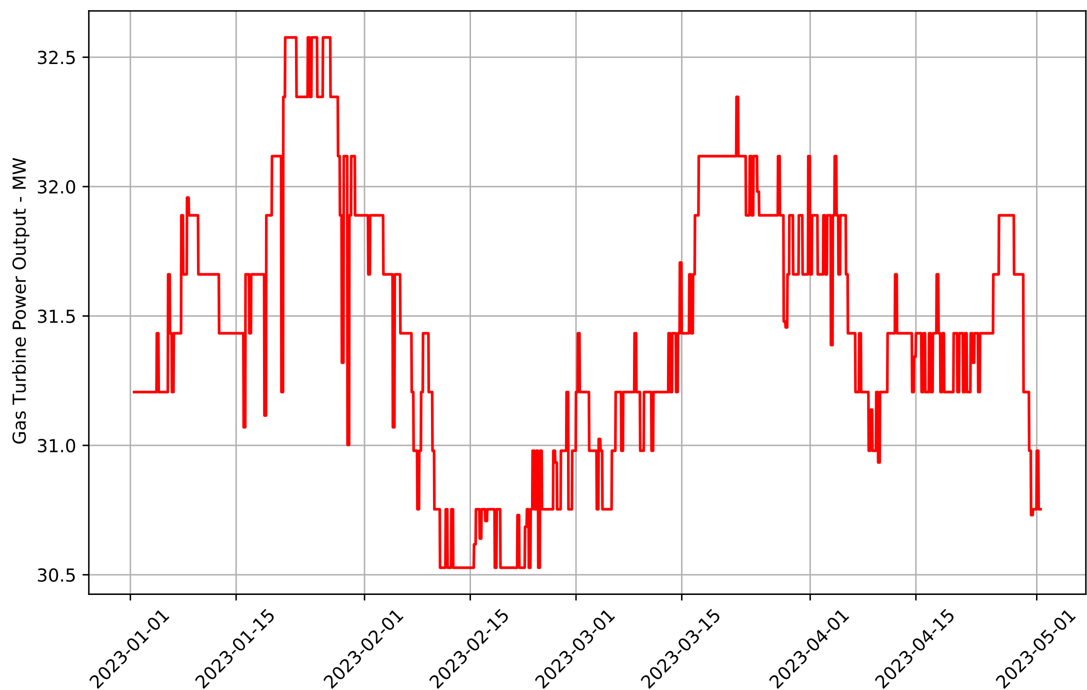


Figure 4.6.1: Power output estimation for the time frame selected.

The same behavior can be seen on figure 4.6.2, but now the model is connected with Locationforecast Weather API in order to forecast the power generation of the gas turbine:

It can be seen from both scenarios that the ambient conditions hold a significant impact on gas turbine operation and performance. The digital twin model is a tool where it makes possible to observe these phenomena, and can be expanded further into evaluating other scenarios.

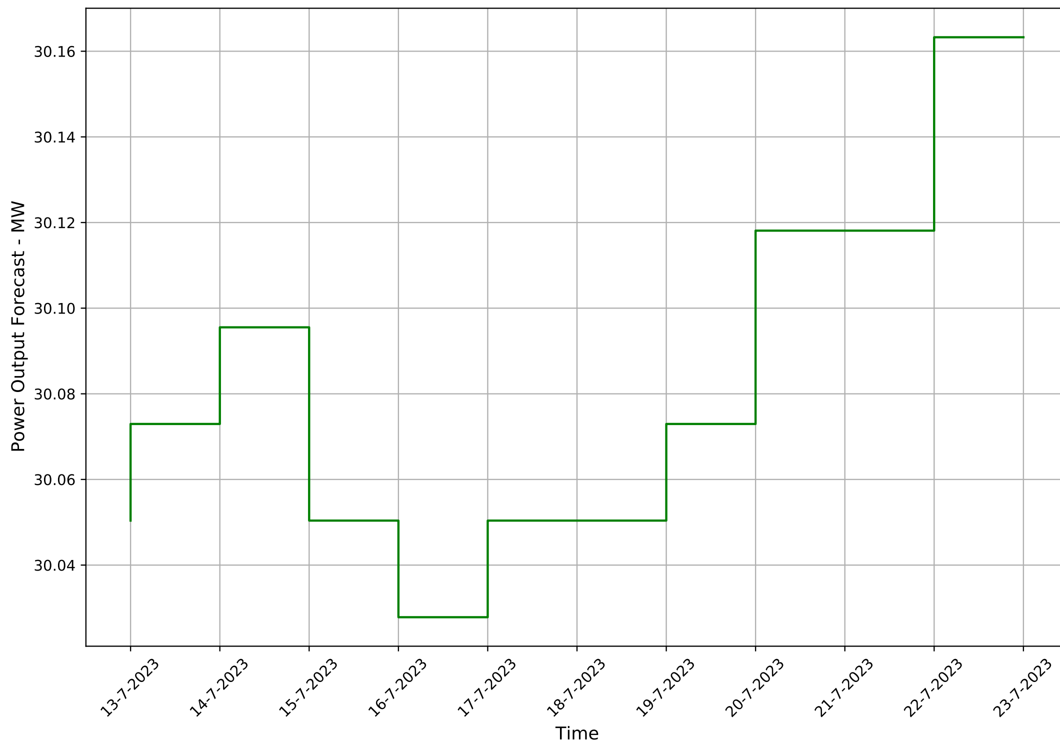


Figure 4.6.2: Power output forecast.

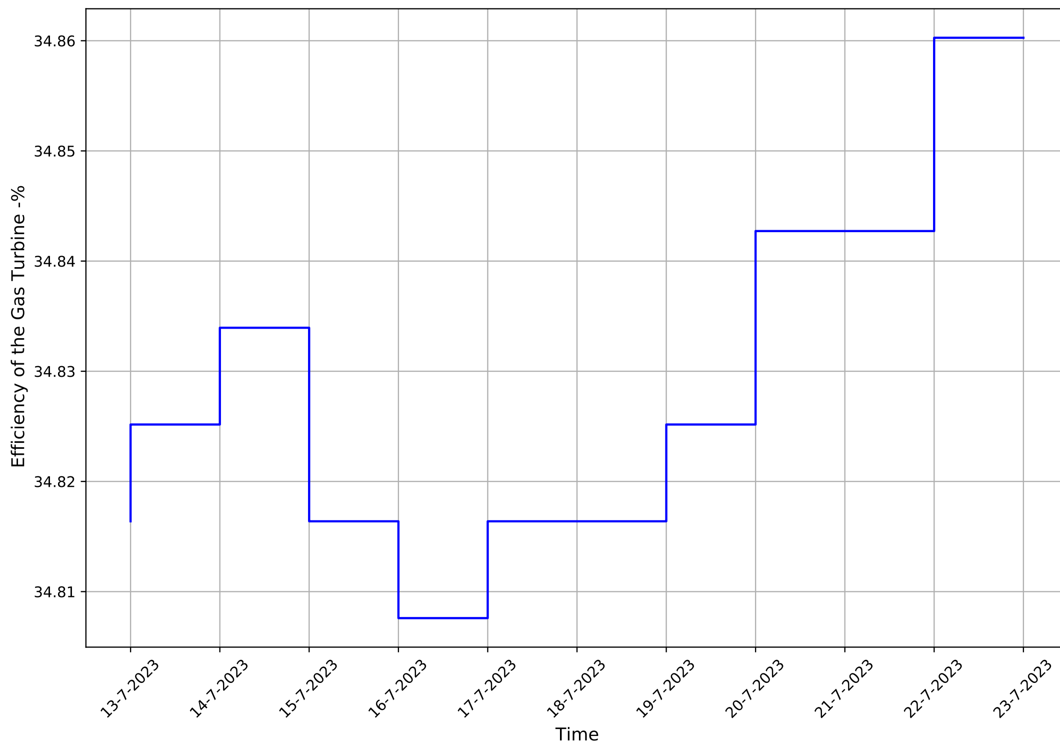


Figure 4.6.3: Efficiency of gas turbine forecast.

CHAPTER

FIVE

CONCLUSION AND FURTHER WORK

5.1 Conclusion

This project outlines the development of a digital twin of a gas turbine for power generation forecast. The literature review showed how a simplified gas turbine can be modeled, and it was successfully verified and validated against two established software.

The model developed in Python was demonstrated to be robust and relatively accurate, with its given simplifications and assumptions. The off-design procedure for performance evaluation proved to be effective and generated good results.

The verification and validation procedures also were effective for the model validation. This was a key part of the study to ensure that the model could generate more precise and accurate results, to further be used to represent a real operating gas turbine.

The off-design procedure simplified from Saravanamuttoo et al. [4] proved to be a good simplified approach to calculate the main parameters of the off-design performance of the gas turbine, despite its limitations on the turbine exhaust temperature calculations.

The approach used for establishing a correlation from fuel data from Thermoflow ® and ambient temperature instead of using field data was effective for the digital twin model and was able to generate good results.

This study made it possible to see how a digital twin can be used in an industrial scenario, and indicate how much ambient temperature affects the performance of a gas turbine.

It also highlighted that gas turbines' best operating conditions are at colder temperatures, where the density of the gaseous fluid increases, allowing more power to be generated for the same fuel input. Higher power outputs consequently increase gas turbine efficiency, and this study shows those effects.

From a verified and validated model, the digital twin provides a good estimation of a gas turbine power forecast. The hybrid model approach proved to account for the effects of both the physical and the data-driven model to create a robust and reliable digital twin.

5.2 Future Work

The complexity of gas turbine operation depends on a broader scope than the one used in this study. To have a more precise functioning digital twin that mimics almost exactly the behavior of a real gas turbine, fewer simplifications need to be performed.

The use of other software to achieve unknown variables can be evaluated in further works to determine its accuracy, and possibly use another method for calculating these variables, like with the aid of compressor and turbine performance curves, relating the speed of the turbomachine and efficiency, along with volume flow.

Compressor and turbine maps can be applied in the off-design methodology, which takes into consideration changes in speed, as well as part-load performance. In the lack of specific maps, using a generalized compressor and turbine characteristics could be another way of improving off-design calculations of the gas turbine.

Perform validation of the model with real field data. This way, the reliability of the model increases and is capable of providing an even more trustworthy estimation of power generation forecasting.

Challenges in developing precise digital twins also rely on accurate sensor measurements, as well as a precise model. For future work, establishing a reliable digital twin tool model that can be connected with live process data will bring the model one step closer to having fewer differences from the real gas turbine.

REFERENCES

[1] EIA - Energy Information Administration. "Annual Energy Outlook - AEO 2023". In: (2023).

[2] Fred Landis. *Development of gas turbines*. Accessed on: 18.05.2023. URL: %5Curl%7Bhttps://www.britannica.com/technology/gas-turbine-engine/Development-of-gas-turbines%7D.

[3] Mehervan P. Boyce. *Gas Turbine Engineering Handbook*. Gulf Professional Publishing, 2002. ISBN: 0-88415-732-6.

[4] HIH Saravanamuttoo, GFC Rogerds, and H. Cohen. *Gas Turbine Theory*. Pearson Education, Ltd., 2001. ISBN: 978-81-7758-902-3.

[5] L.S. Langston. "Turbines, Gas". In: *Reference Module in Earth Systems and Environmental Sciences*. Elsevier, 2014. ISBN: 978-0-12-409548-9. DOI: https://doi.org/10.1016/B978-0-12-409548-9.09044-8. URL: https://www.sciencedirect.com/science/article/pii/B9780124095489090448.

[6] C. Balestrino. "Evolutionary improvements of Siemens SGT-A35 gas turbine". In: *Gas turbines for energy network symposium* (2019).

[7] Dusty Phillips. *Python 3 Object Oriented Programming*. Packt Publishing, 2010.

[8] University of Cape Town and indivual contributors. *Object-Oriented Programming in Python Documentation - Release 1*. 2017.

[9] Ma-Keba Frye. "What is an API?" In: (). URL: https://www.mulesoft.com/resources/api/what-is-an-api.

[10] URL: %5Curl%7Bhttps://www.visualcrossing.com/resources/documentation/weather-api/what-is-a-weather-pi/#:~:text=A%5C%20weather%5C%20API%5C%20is%5C%20an,%5C%2C%5C%20well%5C%2Ddefined%5C%20programming%5C%20interface.%7D.

[11] Michael Grieves. "Digital Twin: Manufacturing Excellence through Virtual Factory Replication". In: (Mar. 2015).

[12] Adil Rasheed, Omer San, and Trond Kvamsdal. *Digital Twin: Values, Challenges and Enablers*. 2019. arXiv: 1910.01719 [eess.SP].

[13] Oracle. *Digital Twins for IoT Applications: A Comprehensive Approach to Implementing IoT Digital Twins*. Accessed on: 01.06.2023. URL: %5Curl%7Bhttps://infotech.report/Resources/Whitepapers/a3323742-5fee-4df3-89fd-d9aeaa012ba6_digital-twins-for-iot-apps-wp-3491953.pdf%7D.

[14] Csaba Ruzsa. "Digital twin technology - external data resources in creating the model and classification of different digital twin types in manufacturing". In: *Procedia Manufacturing* 54 (2021). 10th CIRP Sponsored Conference on Digital Enterprise Technologies (DET 2020) – Digital Technologies as Enablers of Industrial Competitiveness and Sustainability, pp. 209–215. ISSN: 2351-9789. DOI: https://doi.org/10.1016/j.promfg.2021.07.032. URL: https://www.sciencedirect.com/science/article/pii/S2351978921001682.

[15] Minghui HU et al. "Digital twin model of gas turbine and its application in warning of performance fault". In: *Chinese Journal of Aeronautics* 36.3 (2023), pp. 449–470. ISSN: 1000-9361. DOI: https://doi.org/10.1016/j.cja.2022.07.021. URL: https://www.sciencedirect.com/science/article/pii/S1000936122001583.

[16] Alexios Alexiou. "Development of Gas Turbine Performance Models Using a Generic Simulation Tool". In: vol. 1. June 2005. DOI: 10.1115/GT2005-68678.

[17] Elias Tsoutsanis and Nader Meskin. "Dynamic performance simulation and control of gas turbines used for hybrid gas/wind energy applications". In: *Applied Thermal Engineering* 147 (Jan. 2019), pp. 122–142. ISSN: 1359-4311. DOI: 10.1016/J.APPLTHERMALENG.2018.09.031.

[18] S.L Dixon and C.A. Hall. *Fluid Mechanics and Thermodynamics of Turbomachinery*. Elsevier Inc, 2014. ISBN: 978-0-12-415954-9.

[19] Thermoflow Inc. *Thermoflow Software Suite User's Manual*. 2022.

[20] Saturday Ebigenibo and Nweke Promise. "Off-design performance analysis of gas turbines". In: *Global Journal of Engineering and Technology Advances* 4 (Aug. 2020), p. 4. DOI: 10.30574/gjeta.2020.4.2.0046.

[21] "IEEE Standard Glossary of Software Engineering Terminology". In: *IEEE Std 610.12-1990* (1990), pp. 1–84. DOI: 10.1109/IEEESTD.1990.101064.

[22] Accessed on: 11.05.2023. URL: %5Curl%7Bhttps://dev.meteostat.net/%7D.

[23] URL: %5Curl%7Bhttps://api.met.no/%7D.

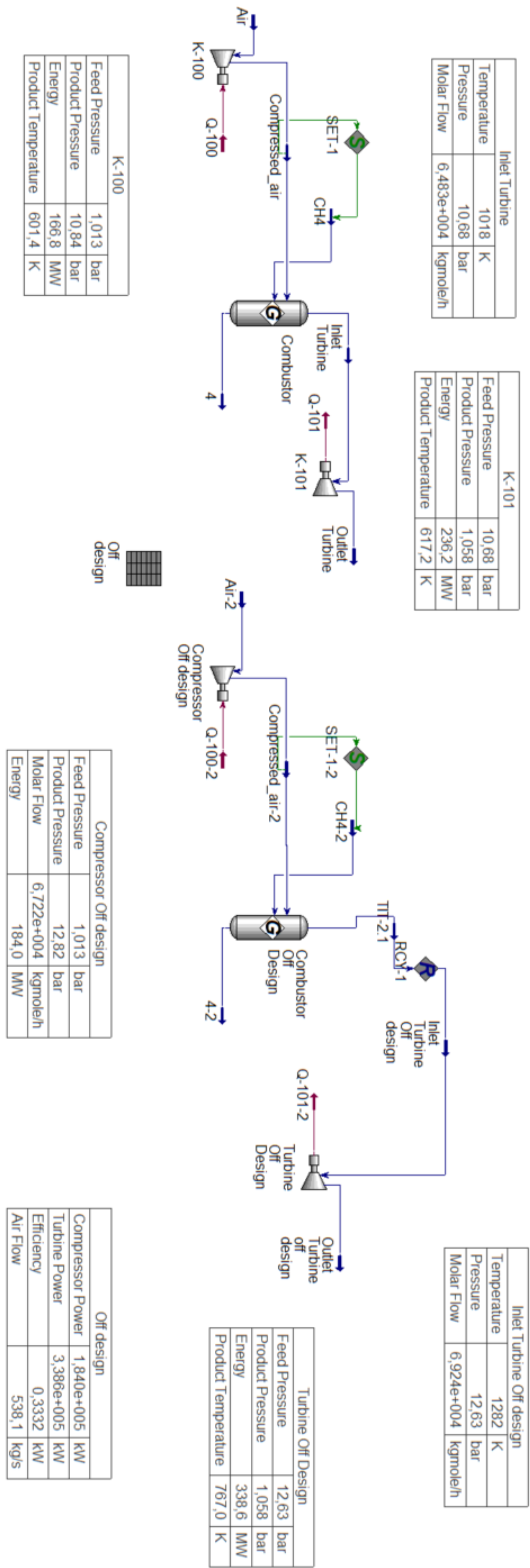
[24] "Siemens Energy Gas turbine portfolio". In: (). URL: https://assets.siemens-energy.com/siemens/assets/api/uuid:2ead6ba9-ceea-4053-a079-a0496124af45/gas-portfolio-brochure.pdf.

[25] Ignacio López-Paniagua et al. "Step by Step Derivation of the Optimum Multistage Compression Ratio and an Application Case". In: *Entropy* 22 (6 June 2020), p. 678. ISSN: 1099-4300. DOI: 10.3390/e22060678. URL: https://www.mdpi.com/1099-4300/22/6/678.

[26] *Heidrun Oil Field.* Accessed July 7, 2023. URL: https://geohack.toolforge.org/geohack.php?pagename=Heidrun_oil_field¶ms=65_19_33_N_7_19_3_E_type:landmark.

APPENDICES

HYSYS model



Python Code

```python
    from neqsim.thermo.thermoTools import fluid, printFrame
    from neqsim.thermo import PSflash, TPflash, PHflash
    from neqsim.process import clearProcess
    from neqsim.standards import ISO6976
    from math import log10, log, sqrt
    import matplotlib.pyplot as plt
    from scipy.optimize import fsolve, minimize
    import copy
    import numpy as np
    import pandas as pd

class Stream():
    '''
    Object to define a stream in the model

    Neqsim will be used to calculate all the fluid properties

    '''
    def __init__(self):
        self.temperature = None #K
        self.pressure = None  #Pa
        self.flow_rate = None #kg/hr
        self.stream_fluid = None #neqsim fluid

    def set_fluid(self,stream_fluid):
        '''
        This function is used to assign an object with the fluid from neqsim

        '''
        self.stream_fluid = stream_fluid

    def get_fluid(self):

        return self.stream_fluid

    def set_temperature(self,temperature:float,units:str) -> float:
        '''
        Function to set temperature of the stream

        @param self stream object
        @param float temperature: temperature of the stream
        @param str units: standarized value of a given physical property

        returns:float temperature in K
        '''
        if (units=="K"):
            self.temperature = temperature
        elif (units =="C"):
            self.temperature = temperature + 273.15
        elif (units == 'F'):
            self.temperature = (9/5)*temperature+32
        else:
            print(f"No units found for temperature in stream {self}")
            self.temperature = None

```

```
56      def get_temperature(self,units:str) -> float:
57          if units == "K":
58              return self.temperature
59          else:
60              print(f"No units found for temperature in stream {self}")
61              return None
62
63
64      def set_flow_rate(self,flow_rate:float,units:str) -> float:
65          '''
66          Function to set flow rate. Returns the flow rate in kg/hr
67          '''
68          if (units == 'kg/hr'):
69              self.flow_rate = flow_rate
70          elif (units == 'm3/hr'):
71              self.flow_rate = flow_rate*self.stream_fluid.getDensity()
72          elif (units == 'kg/s'):
73              self.flow_rate = flow_rate*3600
74          # elif (units == 'MSm3/day'):
75          #     self.flow_rate = flow_rate*1e6*self.density_std/24
76          else:
77              print(f"No units found for flow rate in stream {self}")
78              return None
79
80      def get_flow_rate(self, units:str) -> float:
81          if units == 'kg/hr':
82              return self.flow_rate
83          else:
84              print(f"No units found for flow rate in stream {self}")
85              return None
86
87      def set_pressure(self,pressure:float, units:str) -> float:
88          '''
89          Function to set pressure of the stream
90
91          Returns the pressure of the stream in Pa.
92          '''
93          if(units == 'Pa'):
94              self.pressure = pressure
95          elif(units == 'bara'):
96              self.pressure = pressure*1e5
97          elif(units == 'kPa'):
98              self.pressure = pressure/1e3
99          else:
100             print(f"No units found for pressure in stream {self}")
101             self.pressure = None
102
103     def get_pressure(self,units:str) -> float:
104         if units == "Pa":
105             return self.pressure
106         else:
107             print("No units found")
108             return None
109
110     def calc(self):
111         '''
112         Calculation method inside the stream class. It verifies if the
        parameters have the correct units, and then later performs a TPflash to
```

```
    calculate the physica properties.
113
114         The properties will be calculated using neqsim.
115
116         '''
117         if self.temperature == None:
118             print(f"ERROR: temperature was not set to the STREAM {self}")
119         else:
120             self.stream_fluid.setTemperature(self.temperature, "K")
121
122         if self.pressure == None:
123             print(f"ERROR: pressure was not set to the STREAM {self}")
124         else:
125             self.stream_fluid.setPressure(self.pressure/1e5, "bara")
126         if self.flow_rate == None:
127             print(f"ERROR: flow_rate was not set to the STREAM {self}")
128         else:
129             self.stream_fluid.setTotalFlowRate(self.flow_rate, 'kg/hr')
130         TPflash(self.stream_fluid)
131         self.stream_fluid.initProperties()
132
133 class Compressor():
134     '''
135     Object to define compressor design parameters
136
137     Calculate isentropic efficiency, outlet temperature of compressor
138
139     Neqsim will be used for calculating the fluid properties, and assign it to
    a fluid stream
140
141     '''
142     def __init__(self):
143         self.pressure = None
144         self.temperature = None
145         self.inlet_stream = None #Stream class
146         self.outlet_stream = None #Stream class
147         self.pressure_ratio = None #no units
148         self.isentropic_efficiency = None #no units
149         self.polytropic_efficiency = None #no units
150         self.work = None #MW
151         self.outlet_temperature = None #K
152         self.outlet_pressure = None #Pa
153         self.isentropic_exponent = None #no units
154         self.polytropic_exponent = None #no units
155         self.correction_factor = None #no units
156         self.temperature_isentropic = None #K
157         self.inlet_temperature = None #K
158         self.flow_rate = None #kg/hr
159         self.pressure_air_filter = None #Pa
160
161     def set_inlet_stream(self, inlet_stream:Stream) -> Stream:
162         '''
163         Function to assign inlet stream for the compressor
164         '''
165         self.inlet_stream = inlet_stream
166
167     def get_inlet_stream(self) -> Stream:
168         return self.inlet_stream
```

```
169
170     def set_outlet_stream(self, outlet_stream: Stream) -> Stream:
171         '''
172         Function to assign outlet stream for the compressor
173         '''
174         self.outlet_stream = outlet_stream
175
176     def get_outlet_stream(self) -> Stream:
177         return self.outlet_stream
178
179     def calc_outlet_stream(self) -> Stream:
180         '''
181         Function to calculate parameters of outlet stream of the compressor
182
183         Shallow copy of inlet stream, copying neqsim fluid, and flow rate,
    according to the mass balance: flow_rate inlet_stream = flow_rate
    outlet_stream
184
185         Outlet temperatures and pressures of the stream will be calculated on
    calc_outlet_temperature and pressure method, and then assigned to the
    outlet stream.
186
187         Neqsim will then calculate the outlet stream properties.
188
189         '''
190         self.outlet_stream = copy.copy(self.inlet_stream)
191         self.outlet_fluid = self.inlet_stream.get_fluid().clone()
192         if self.get_outlet_pressure("Pa") == None:
193             print("COMPRESSOR ERROR: pressure is NONE")
194         if self.get_outlet_temperature("K") == None:
195             print("COMPRESSOR ERROR: temperature is NONE")
196         else:
197             self.outlet_stream.set_fluid(self.outlet_fluid)
198             self.outlet_stream.set_temperature(self.get_outlet_temperature("K")
    , "K")
199             self.outlet_stream.set_pressure(self.get_outlet_pressure("Pa"), "Pa
    ")
200             self.outlet_stream.calc()
201
202     def set_outlet_pressure(self, outlet_pressure:float, units:str) -> float:
203         if units == "Pa":
204             self.outlet_pressure = outlet_pressure
205         elif units == "bara":
206             self.outlet_pressure = outlet_pressure*1e5
207         else:
208             print(f"Error: Pressure units not found for compressor {self}.")
209
210     def get_outlet_pressure(self,units:str) -> float:
211         if units == "Pa":
212             return self.outlet_pressure
213         else:
214             print(f"No units found for pressure for the STREAM {self}")
215             return None
216
217     def calc_outlet_pressure(self) -> float:
218         self.outlet_pressure = self.inlet_stream.get_pressure("Pa")*self.
    get_pressure_ratio()
219
```

```
220     def set_outlet_temperature(self, temperature:float, units:str) -> float:
221         if units == "K":
222             return self.temperature
223         elif units == "C":
224             self.outlet_temperature = self.outlet_temperature +273.15
225         elif units == "F":
226             self.outlet_temperature =(9/5)*self.outlet_temperature+32
227         else:
228             print("No units found.")
229             return self.outlet_temperature
230 
231     def get_outlet_temperature(self,units:str) ->float:
232         if units == "K":
233             return self.outlet_temperature
234         else:
235             print("No units found for temperature.")
236             return None
237 
238     def get_kappa(self, temperature:float, pressure: float) -> float:
239         temp_stream = copy.copy(self.inlet_stream)
240         temp_fluid = self.inlet_stream.get_fluid().clone()
241         temp_stream.set_temperature(temperature, "K")
242         temp_stream.set_pressure(pressure, "Pa")
243         temp_stream.set_fluid(temp_fluid)
244         temp_stream.calc()
245         return temp_stream.get_fluid().getKappa()
246 
247     def calc_outlet_temperature(self):
248         steps = 200
249         Pr = self.get_pressure_ratio()
250         T1 = self.inlet_stream.get_temperature("K")
251         Pin = self.get_inlet_stream().get_pressure("Pa")
252         temperature_isentropic  = 288.15
253         i = 0
254         for i in range(steps):
255             kappa = self.inlet_stream.get_fluid().getKappa()
256             Pr = self.get_pressure_ratio()**(i/steps)
257             temperature_isentropic = T1 * (Pr) **((kappa-1)/kappa)
258             Pin = Pin * Pr
259         self.outlet_temperature = T1 + ((temperature_isentropic - T1)/self.
    get_isentropic_efficiency())
260 
261     def get_temperature_isentropic(self, units:str) -> float:
262         if units == "K":
263             return self.temperature_isentropic
264         else:
265             print(f"No units found for temperature in stream {self}")
266             return None
267 
268     def set_pressure_ratio(self, pressure_ratio:float) -> float:
269         self.pressure_ratio = pressure_ratio
270 
271     def get_pressure_ratio(self) -> float:
272         return self.pressure_ratio
273 
274     def calc_pressure_ratio(self):
275         self.pressure_ratio = self.get_outlet_pressure("Pa")/self.
    get_inlet_stream().get_pressure("Pa")
```

```
276
277     def set_isentropic_efficiency(self, isentropic_efficiency) -> float:
278         self.isentropic_efficiency = isentropic_efficiency
279
280     def get_isentropic_efficiency(self) -> float:
281         return self.isentropic_efficiency
282
283     def calc_isentropic_efficiency(self):
284         kappa = self.inlet_stream.get_fluid().getKappa()
285         k = kappa
286         Pr = self.get_pressure_ratio()
287         np = self.get_polytropic_efficiency()
288         self.isentropic_efficiency = 1/((Pr**((k-1)/k)-1)/(Pr**((k-1)/k*np)-1))
289
290     def set_polytropic_efficiency(self, polytropic_efficiency:float) -> float:
291         self.polytropic_efficiency = polytropic_efficiency
292
293     def get_polytropic_efficiency(self) ->float:
294         return self.polytropic_efficiency
295
296     def calc_polytropic_efficiency(self) -> float:
297         '''
298         Function to calculate polytropic efficiency.
299
300         Neqsim will be used for calculating the isentropic properties using a
    PSflash, as well as the inlet and outlet properties of the fluid.
301
302        '''
303         def polytropic_efficiency_error(np):
304             P1 = self.get_inlet_stream().get_pressure("Pa")
305             P2 = self.get_outlet_pressure("Pa")
306             k = self.inlet_stream.get_fluid().getKappa()
307             isentropic_efficiency = self.get_isentropic_efficiency()
308             difference = ((P2/P1)**((k-1)/k) - 1)/((P2/P1)**((k-1)/k*np) -1) -
    isentropic_efficiency
309             np = np_guess
310             return difference
311         np_guess = self.get_isentropic_efficiency()
312         np_solution = fsolve(polytropic_efficiency_error, np_guess)
313         self.polytropic_efficiency = 1/np_solution[0]
314
315
316     def set_work(self, work:float, units:str) -> float:
317         if units == "MW":
318             self.work = work
319         elif units == "kW":
320             self.work = work*1e3
321         elif units == "W":
322             self.work = work*1e6
323         else:
324             print(f"No units found for work in {self}.")
325             return self.work
326
327     def get_work(self, units:str) -> float:
328         if units == "MW":
329             return self.work
330         else:
331             print(f"No units found for {self} compressor work.")
```

```
332             return None
333
334     def calc_work(self):
335         '''
336         Function to calculate compressor work.
337
338         A PS Flash is used to calculate the isentropic properties of the outlet
         stream.
339
340         Neqsim will read the properties of inlet and outlet stream fluids to
        perform the calculations.
341
342         '''
343         #Properties of inlet fluid
344         self.density_in = self.get_inlet_stream().get_fluid().getDensity("kg/m3
        ")
345         self.pressure_in = self.get_inlet_stream().get_pressure("Pa")
346         self.flow_rate = self.get_inlet_stream().get_flow_rate("kg/hr")
347         self.entropy_in = self.get_inlet_stream().get_fluid().getEntropy("J/K")
348         self.enthalpy_in = self.get_inlet_stream().get_fluid().getEnthalpy("J/
        kg")
349         Cp = self.get_inlet_stream().get_fluid().getCp("J/kgK")
350
351         MW = self.get_inlet_stream().get_fluid().getMolarMass()
352
353         #Outlet Fluid
354         self.density_out = self.get_outlet_stream().get_fluid().getDensity("kg/
        m3")
355         self.pressure_out = self.get_outlet_stream().get_pressure("Pa")
356         self.volume_exponent = log(self.pressure_out/self.pressure_in)/log(self
        .density_out/self.density_in)
357         n = self.volume_exponent
358
359         #Isentropic Properties of outlet fluid - using a PS Flash
360         outlet_pressure = self.get_outlet_stream().get_pressure("Pa")
361         PSflash(self.get_outlet_stream().get_fluid(), self.entropy_in, "J/K")
362         self.get_outlet_stream().get_fluid().initProperties()
363         self.enthalpy_out_isentropic = self.get_outlet_stream().get_fluid().
        getEnthalpy("J/kg")
364         self.density_out_isentropic = self.get_outlet_stream().get_fluid().
        getDensity("kg/m3")
365         numerator = self.enthalpy_out_isentropic - self.enthalpy_in
366         denominator = (n / (n - 1)) * ((self.pressure_out/self.
        density_out_isentropic) - (self.pressure_in/self.density_in))
367         CF = numerator / denominator
368         self.work = (self.get_inlet_stream().get_flow_rate("kg/hr")/3600 * Cp *
         (self.get_outlet_stream().get_temperature("K") - self.get_inlet_stream().
        get_temperature("K")))/1e6
369
370     def set_steps(self, steps:int) -> int:
371         self.steps = steps
372
373     def get_steps(self):
374         return self.steps
375
376     def calc(self) -> float:
377         '''
378         Function for verifying if the necessary variables are defined for the
```

```
    compressor. If two variables are given, the other ones are calculated.
379
380         If not, an error will appear.
381         '''
382         self.success = False
383         #if self.steps is not None:
384
385         if self.isentropic_efficiency is not None:
386             if self.pressure_ratio is not None:
387                 self.calc_outlet_pressure()
388                 self.calc_polytropic_efficiency()
389                 self.calc_outlet_temperature()
390                 self.calc_outlet_stream()
391                 self.calc_work()
392                 #self.calc_compression_in_steps()
393                 #self.calc_work()
394                 self.success = True
395
396             elif self.outlet_pressure is not None:
397                 self.calc_pressure_ratio()
398                 self.calc_outlet_temperature()
399                 self.calc_outlet_stream()
400                 self.calc_polytropic_efficiency()
401                 self.calc_work()
402                 #self.calc_compression_in_steps()
403                 self.success = True
404
405             elif self.outlet_temperature is not None:
406                 self.calc_outlet_pressure()
407                 self.calc_pressure_ratio()
408                 self.calc_outlet_stream()
409                 self.calc_polytropic_efficiency()
410                 self.calc_work()
411                 #self.calc_compression_in_steps()
412                 self.success = True
413
414             elif self.polytropic_efficiency is not None:
415                 self.calc_pressure_ratio()
416                 self.calc_outlet_pressure()
417                 self.calc_outlet_temperature()
418                 self.calc_outlet_stream()
419                 self.calc_work()
420                 #self.calc_compression_in_steps()
421                 self.success = True
422
423             elif self.work is not None:
424                 self.calc_outlet_pressure()
425                 self.calc_pressure_ratio()
426                 self.calc_outlet_temperature()
427                 self.calc_outlet_stream()
428                 self.calc_polytropic_efficiency()
429                 #self.calc_compression_in_steps()
430                 self.success = True
431             else:
432                 print(f"Error. Only isentropic efficiency is defined for
    compressor {self}.")
433
434         elif self.set_outlet_pressure is not None:
```

```
435            if self.pressure_ratio is not None:
436                #self.calc_outlet_temperature()
437                self.calc_outlet_stream()
438                self.calc_isentropic_efficiency()
439                self.calc_polytropic_efficiency()
440                self.calc_work()
441                self.success = True
442
443            elif self.isentropic_efficiency is not None:
444                self.calc_pressure_ratio()
445                self.calc_outlet_temperature()
446                self.calc_outlet_stream()
447                self.calc_isentropic_efficiency()
448                self.calc_polytropic_efficiency()
449                self.calc_work()
450                self.success = True
451
452            elif self.outlet_temperature is not None:
453                self.calc_pressure_ratio()
454                self.calc_outlet_stream()
455                self.calc_isentropic_efficiency()
456                self.calc_polytropic_efficiency()
457                self.calc_work()
458                self.sucess = True
459
460            elif self.polytropic_efficiency is not None:
461                self.calc_pressure_ratio()
462                self.calc_isentropic_efficiency()
463                self.calc_outlet_temperature()
464                self.calc_outlet_stream()
465                self.calc_work()
466                self.sucess = True
467
468            elif self.work is not None:
469                self.calc_outlet_temperature()
470                self.calc_pressure_ratio()
471                self.calc_outlet_stream()
472                self.calc_isentropic_efficiency()
473                self.calc_polytropic_efficiency()
474                self.sucess = True
475
476            else:
477                print(f"Error. Only outlet pressure is defined for compressor {
    self}.")
478
479        elif self.pressure_ratio is not None:
480            if self.outlet_pressure is not None:
481                print("Compressor Error. Another variable needs to be specified
    .")
482                self.success = False
483
484            elif self.outlet_temperature is not None:
485                self.calc_outlet_pressure()
486                self.calc_outlet_stream()
487                self.calc_isentropic_efficiency()
488                self.calc_polytropic_efficiency()
489                self.calc_work()
490                self.success = True
```

```
491
492             elif self.isentropic_efficiency is not None:
493                 self.calc_outlet_pressure()
494                 self.calc_outlet_temperature()
495                 self.calc_outlet_stream()
496                 self.calc_isentropic_efficiency()
497                 self.calc_polytropic_efficiency()
498                 self.calc_work()
499                 self.sucess = True
500
501             elif self.polytropic_efficiency is not None:
502                 self.calc_outlet_pressure()
503                 self.calc_outlet_temperature()
504                 self.calc_isentropic_efficiency()
505                 self.calc_outlet_stream()
506                 self.calc_work()
507                 self.success = True
508
509             elif self.work is not None:
510                 self.calc_outlet_pressure()
511                 self.calc_outlet_temperature()
512                 self.calc_outlet_stream()
513                 self.calc_isentropic_efficiency()
514                 self.calc_polytropic_efficiency()
515                 self.success = True
516
517             else:
518                 print(f"Compressor Error. Only pressure ratio is defined for
    compressor {self}.")
519
520         elif self.outlet_temperature is not None:
521             if self.pressure_ratio is not None:
522                 self.calc_outlet_pressure()
523                 self.calc_outlet_stream()
524                 self.calc_isentropic_efficiency()
525                 self.calc_polytropic_efficiency()
526                 self.calc_work()
527                 self.sucess = True
528
529             elif self.outlet_pressure is not None:
530                 self.calc_pressure_ratio()
531                 self.calc_outlet_stream()
532                 self.calc_isentropic_efficiency()
533                 self.calc_polytropic_efficiency()
534                 self.calc_work()
535                 self.sucess = True
536
537             elif self.isentropic_efficiency is not None:
538                 self.calc_outlet_pressure()
539                 self.calc_pressure_ratio()
540                 self.calc_outlet_stream()
541                 self.calc_polytropic_efficiency()
542                 self.calc_work()
543                 self.success = True
544
545             elif self.polytropic_efficiency is not None:
546                 self.calc_isentropic_efficiency()
547                 self.calc_pressure_ratio()
```

```
548                 self.calc_outlet_pressure()
549                 self.calc_outlet_stream()
550                 self.calc_work()
551                 self.success = True
552 
553             elif self.work is not None:
554                 self.calc_outlet_pressure()
555                 self.calc_pressure_ratio()
556                 self.calc_outlet_stream()
557                 self.calc_isentropic_efficiency()
558                 self.calc_polytropic_efficiency()
559                 self.success = True
560 
561             else:
562                 print(f"Compressor Error. Only outlet temperature is defined
        for compressor {self}.")
563 
564         elif self.work is not None:
565             if self.outlet_temperature is not None:
566                 self.calc_outlet_pressure()
567                 self.calc_pressure_ratio()
568                 self.calc_outlet_stream()
569                 self.calc_isentropic_efficiency()
570                 self.calc_polytropic_efficiency()
571                 self.success = True
572 
573             elif self.outlet_pressure is not None:
574                 self.calc_pressure_ratio()
575                 self.calc_outlet_temperature()
576                 self.calc_outlet_stream()
577                 self.calc_isentropic_efficiency()
578                 self.calc_polytropic_efficiency()
579                 self.sucess = True
580 
581             elif self.pressure_ratio is not None:
582                 self.calc_outlet_pressure()
583                 self.calc_outlet_temperature()
584                 self.calc_outlet_stream()
585                 self.calc_isentropic_efficiency()
586                 self.calc_polytropic_efficiency()
587                 self.success = True
588 
589             elif self.isentropic_efficiency is not None:
590                 self.calc_outlet_temperature()
591                 self.calc_outlet_pressure()
592                 self.calc_pressure_ratio()
593                 self.calc_polytropic_efficiency()
594                 self.success = True
595 
596             elif self.polytropic_efficiency is not None:
597                 self.calc_pressure_ratio()
598                 self.calc_outlet_pressure()
599                 self.calc_outlet_temperature()
600                 self.calc_outlet_stream()
601                 self.calc_isentropic_efficiency()
602                 self.success = True
603             else:
604                 print(f"Compressor error. Only duty is defined for compressor {
```

```
    self}.")
605
606         else:
607             print(f"Error. Two variables need to be defined for compressor {
    self}.")
608         # else:
609         #     print("Error. Please define number of steps for calculation
    accuracy.")
610
611
612
613
614 class Combustor():
615         '''
616         Object to calculate turbine inlet temperature (TIT) - outlet of
    combustor.
617
618         Chemical reaction included to estimate composition of outlet stream.
619
620         Energy balance on the combustor.
621
622         '''
623
624         def __init__(self):
625
626             self.inlet_stream = None #Stream class
627             self.fuel_inlet_stream = None #kg/hr
628             self.outlet_stream = None #Stream class
629             self.outlet_temperature = None #K
630             self.outlet_pressure = None #Pa
631             self.inlet_pressure = None
632             self.outlet_fluid = None
633             self.deltaP = None
634             self.temperature = None
635             self.pressure = None
636             self.mixed_stream = None
637             self.ambient_temperature = 288.15 #K
638             self.work = None #MW
639             self.LHV = None #MJ/kg
640             self.mixed_stream = None #Stream class
641             self.excess_air = 1
642             self.isentropic_efficiency  = 1
643
644         def set_deltaP(self, deltaP) -> float:
645             self.deltaP = deltaP
646
647         def get_deltaP(self):
648             return self.deltaP
649
650         def set_ambient_temperature(self, ambient_temperature):
651             self.ambient_temperature = ambient_temperature
652
653         def set_inlet_stream(self,inlet_stream:Stream) -> Stream:
654             self.inlet_stream = inlet_stream
655
656         def get_inlet_stream(self) -> Stream:
657             return self.inlet_stream
658
```

```
659     def set_fuel_inlet_stream(self,fuel_inlet_stream:Stream) -> Stream:
660         self.fuel_inlet_stream = fuel_inlet_stream
661 
662     def get_fuel_inlet_stream(self):
663         return self.fuel_inlet_stream
664 
665     def get_outlet_stream(self):
666         return self.outlet_stream
667 
668     def set_outlet_pressure(self, outlet_pressure:float, units:str) ->
    float:
669         if units == "Pa":
670             self.outlet_pressure = outlet_pressure
671         elif units == "bara":
672             self.outlet_pressure = outlet_pressure*1e5
673         else:
674             print(f"Error: Pressure units not found for compressor {self}."
    )
675 
676     def get_outlet_pressure(self,units:str) -> float:
677         if units == "Pa":
678             return self.outlet_pressure
679         else:
680             print(f"No units found for pressure for the STREAM {self}")
681         return None
682 
683     def calc_outlet_pressure(self) -> float:
684         self.outlet_pressure = self.inlet_stream.get_pressure("Pa")*(1 - (
    self.get_deltaP()/100))
685 
686     def set_outlet_temperature(self, temperature:float, units:str) -> float
    :
687         if units == "K":
688             return self.temperature
689         elif units == "C":
690             self.outlet_temperature = self.outlet_temperature + 273.15
691         elif units == "F":
692             self.outlet_temperature =(9/5) * self.outlet_temperature + 32
693         else:
694             print(f"No units found for outlet temperature of combustor {
    self}.")
695         return self.outlet_temperature
696 
697     def get_outlet_temperature(self,units:str) ->float:
698         if units == "K":
699             return self.outlet_temperature
700         else:
701             print("No units found for temperature.")
702             return None
703 
704     def get_LHV(self):
705         return self.LHV
706 
707     def calc_LHV(self):
708         ''''
709         Method for determining the lower calorific value of the fuel
    mixture
710         '''
```

```
711             iso6976 = ISO6976(self.get_fuel_inlet_stream().get_fluid())
712             iso6976.setReferenceType('mass')
713             iso6976.setVolRefT(float(15.0))
714             iso6976.setEnergyRefT(float(15.0))
715             iso6976.calculate()
716             self.LHV = round((iso6976.getValue("InferiorCalorificValue")*1e3)
    ,3) #J/kg
717 
718         def calc_heat_input(self):
719             self.heat_input = self.get_fuel_inlet_stream().get_flow_rate("kg/hr
    ")/3600 * self.get_LHV()
720 
721         def get_outlet_stream(self):
722             return self.outlet_stream
723 
724         def get_chemical_reaction_fluid(self) -> Stream:
725             return self.reaction_fluid
726 
727         def calc_chemical_reaction(self) -> float:
728             '''
729             Method for estimating outlet composition of the combustor stream.
730 
731             Complete combustion is assumed.
732 
733             Physical properties are calculated using neqsim.
734             '''
735 
736             #Air components - create a dictionary with air components, and the
    respective molar fractions
737             number_components_air = self.get_inlet_stream().get_fluid().
    getNumberOfComponents()
738             name_components_air  = [self.get_inlet_stream().get_fluid().
    getComponent(i).getName() for i in range(number_components_air)]
739             molar_fractions_air = [self.get_inlet_stream().get_fluid().
    getComponent(i).getx() for i in range(number_components_air)]
740             air_dictionary = {}
741 
742             for i in range(number_components_air):
743                 air_dictionary[name_components_air[i]] = molar_fractions_air[i]
744 
745             MW_O2 = self.get_inlet_stream().get_fluid().getComponent('oxygen').
    getMolarMass() *1000 #g/mol
746             MW_N2 = self.get_inlet_stream().get_fluid().getComponent('nitrogen'
    ).getMolarMass() *1000 #g/mol
747 
748             #Air Composition
749             total_moles_air = self.get_inlet_stream().get_fluid().
    getTotalNumberOfMoles()
750             mols_O2_air = (air_dictionary['oxygen'] * total_moles_air)
751             mols_N2_air = (air_dictionary['nitrogen'] * total_moles_air)
752 
753 
754             #Fuel
755             number_components_fuel = self.get_fuel_inlet_stream().get_fluid().
    getNumberOfComponents()
756             name_components_fuel = [self.get_fuel_inlet_stream().get_fluid().
    getComponent(i).getName() for i in range(number_components_fuel)]
757             molar_fractions_fuel = [self.get_fuel_inlet_stream().get_fluid().
```

```
getComponent(i).getx() for i in range(number_components_fuel)]
758         fuel_dictionary = {}
759 
760         for i in range(number_components_fuel):
761             fuel_dictionary[name_components_fuel[i]] = molar_fractions_fuel
[i]
762 
763         fuel_flow_rate = self.get_fuel_inlet_stream().get_flow_rate("kg/hr"
)
764         MW_CH4 = self.get_fuel_inlet_stream().get_fluid().getComponent('
methane').getMolarMass()* 1000 #Molar mass methane in g/mol
765         total_moles_fuel = self.get_fuel_inlet_stream().get_fluid().
getTotalNumberOfMoles()
766 
767         MW_N2 = self.get_fuel_inlet_stream().get_fluid().getPhase(0).
getComponent('nitrogen').getMolarMass()*1000
768         MW_CO2 = self.get_fuel_inlet_stream().get_fluid().getComponent('CO2
').getMolarMass()*1000
769 
770         MW_NG = self.get_fuel_inlet_stream().get_fluid().getMolarMass() *
1000
771         x_N2_f = self.get_fuel_inlet_stream().get_fluid().getComponent('
nitrogen').getx()
772         x_CO2_f = self.get_fuel_inlet_stream().get_fluid().getComponent('
CO2').getx()
773 
774         #N2 mass on NG mix
775         if x_N2_f is not None:
776             x_N2_ng = (fuel_dictionary['nitrogen'] * MW_N2)/MW_NG
777             m_N2_ng = x_N2_ng * fuel_flow_rate #kg/hr
778             mols_N2_ng = m_N2_ng / (MW_N2 / 1000)
779             N2_ng = mols_N2_ng * fuel_dictionary['nitrogen']
780 
781         else:
782             pass
783 
784         if x_CO2_f is not None:
785             x_CO2_ng = (fuel_dictionary['CO2'] * MW_CO2)
786             m_CO2_ng = x_CO2_ng * fuel_flow_rate #kg/hr
787             mols_CO2_ng = m_CO2_ng / (MW_CO2/1000)
788             CO2_ng = mols_CO2_ng * fuel_dictionary['CO2']
789         else:
790             pass
791 
792         #Methane reaction
793 
794         #Calculate available mols of CH4
795 
796         mols_CH4 = (fuel_dictionary['methane'] * total_moles_fuel)
797 
798         #Determine stoichiometric ratio
799         CH4_limit = mols_CH4 * (2/1) #2 mols of H2O / 1 mol CH4
800         O2_limit_methane = mols_O2_air * (2/2) #2 mols of H2O / 1 mol CH4
801 
802         #  Determine Limiting Reactant
803         limiting_reactant_methane = min(CH4_limit, O2_limit_methane)
804 
805         #Calculate mols of combustion products
```

```
806         mols_CO2_methane = limiting_reactant_methane * (1/1) #1 mol CO2 / 1
     mol CH4
807         mols_H20_methane = limiting_reactant_methane * (2/1) #2 mols H20 /
    1 mol CH4
808
809         #Calculate remaining moles of oxygen
810         reacted_O2_methane = limiting_reactant_methane * (2/1) #2 moles of
    O2 are needed for 1 mol of CH4
811         mols_O2_not_reacted_methane = mols_O2_air - reacted_O2_methane # 2
    mols of CH4 are needed for 1 mol of methane
812
813         total_moles_methane = mols_CO2_methane + mols_H20_methane +
    mols_O2_not_reacted_methane + mols_O2_air + N2_ng  + mols_N2_air *
    fuel_dictionary['methane']
814         O2_methane = (mols_O2_not_reacted_methane / total_moles_methane) *
    fuel_dictionary ['methane']
815         CO2_methane = (mols_CO2_methane/total_moles_methane) *
    fuel_dictionary ['methane']
816         H20_methane = (mols_H20_methane/total_moles_methane) *
    fuel_dictionary['methane']
817
818         if 'ethane' in fuel_dictionary:
819             #Ethane Reaction
820             mols_C2H6 = (fuel_dictionary['ethane'] * total_moles_fuel)
821             C2H6_limit = mols_C2H6 * (3/1) #3 mols H20 / 1 mol C2H6
822             O2_limit_ethane = mols_O2_air * (3/3.5) #3 mols H20 / 3.5 mols
    O2
823
824             limiting_reactant_ethane = min(C2H6_limit, O2_limit_ethane)
825             mols_CO2_ethane = limiting_reactant_ethane * (2/1) #2 mols CO2
    / 1 mol C2H6
826             mols_H20_ethane = limiting_reactant_ethane * (3/1) #3 mols H20
    / 1 mol C2H6
827             mols_O2_not_reacted_ethane = mols_O2_air - mols_C2H6 * 3.5
828             total_moles_ethane = mols_CO2_ethane + mols_H20_ethane +
    mols_O2_not_reacted_ethane + N2_ng + mols_N2_air * fuel_dictionary ['ethane
    ']
829             O2_ethane = (mols_O2_not_reacted_ethane/ total_moles_ethane ) *
     fuel_dictionary ['ethane']
830             CO2_ethane = (mols_CO2_ethane / total_moles_ethane) *
    fuel_dictionary ['ethane']
831             H20_ethane = (mols_H20_ethane/total_moles_ethane) *
    fuel_dictionary['ethane']
832         else:
833             pass
834
835         if 'propane' in fuel_dictionary:
836
837             #Propane Reaction
838
839             mols_C3H8 = (fuel_dictionary['propane'] * total_moles_fuel)
840             C3H8_limit = mols_C3H8 * (4/1) #4 mols H20 / 1 mol C3H8
841             O2_limit_propane = mols_O2_air * (4/5) #4 mols H20 / 5 mols O2
842             limiting_reactant_propane = min(C3H8_limit, O2_limit_propane)
843             mols_CO2_propane = limiting_reactant_propane * (3/1) #3 mols
    CO2 / 1 mol C3H8
844             mols_H20_propane = limiting_reactant_propane * (4/1) #4 mols
    H20 / 1 mol C3H8
```

```
845                 mols_O2_not_reacted_propane = mols_O2_air - mols_C3H8 * 5
846                 total_moles_propane = mols_CO2_propane + mols_H2O_propane +
        mols_O2_not_reacted_propane + N2_ng * fuel_dictionary ['propane']
847                 O2_propane = (mols_O2_not_reacted_propane / total_moles_propane
        ) * fuel_dictionary['propane']
848                 CO2_propane = (mols_CO2_propane/total_moles_propane) *
        fuel_dictionary['propane']
849                 H2O_propane = (mols_H2O_propane/total_moles_propane) *
        fuel_dictionary['propane']
850             else:
851                 pass
852 
853             if 'n-butane' in fuel_dictionary:
854 
855                 # n-Butane Reaction
856                 mols_C4H10 = (fuel_dictionary['n-butane'] * total_moles_fuel)
857                 C4H10_limit = mols_C4H10 * (5/1) #5 mols H2O / 1 mol n-C4H10
858                 O2_limit_butane = mols_O2_air * (5/6.5) #5 mols H2O / 6.5 mols
        O2
859                 limiting_reactant_butane = min(C4H10_limit, O2_limit_butane)
860                 mols_CO2_butane = limiting_reactant_butane * (4/1) #4 mols CO2
        / 1 mol C4H10
861                 mols_H2O_butane = limiting_reactant_butane * (5/1) #5 mols H2O
        / 1 mol C4H10
862                 mols_O2_not_reacted_butane = mols_O2_air - mols_C4H10 * 6.5
863                 total_moles_butane = mols_CO2_butane + mols_H2O_butane +
        mols_O2_not_reacted_butane + N2_ng * fuel_dictionary ['n-butane']
864                 O2_nbutane = (mols_O2_not_reacted_butane / total_moles_butane)
        * fuel_dictionary['n-butane']
865                 CO2_nbutane = (mols_CO2_butane/total_moles_butane) *
        fuel_dictionary['n-butane']
866                 H2O_nbutane = (mols_H2O_butane/total_moles_butane) *
        fuel_dictionary['n-butane']
867             else:
868                 pass
869 
870 
871             if 'i-butane' in fuel_dictionary:
872                 # i-Butane Reaction
873                 mols_C4H10 =(fuel_dictionary['i-butane'] * total_moles_fuel)
874                 iC4H10_limit = mols_C4H10 * (5/1) #5 mols H2O / 1 mol i-C4H10
875                 O2_limit_ibutane = mols_O2_air * (5/6.5) #5 mols H2O / 6.5 mols
         O2
876 
877                 limiting_reactant_ibutane = min (iC4H10_limit, O2_limit_ibutane
        )
878                 mols_CO2_ibutane = limiting_reactant_ibutane * (4/1) #4 mols
        CO2 / 1 mol C4H10
879                 mols_H2O_ibutane = limiting_reactant_ibutane * (5/1) #5 mols
        H2O / 1 mol C4H10
880                 mols_O2_not_reacted_ibutane = mols_O2_air - mols_C4H10 *6.5
881                 total_moles_ibutane = mols_CO2_ibutane + mols_H2O_ibutane +
        mols_O2_not_reacted_ibutane + N2_ng * fuel_dictionary ['i-butane']+ N2_ng *
         fuel_dictionary ['i-butane']
882                 O2_ibutane = (mols_O2_not_reacted_ibutane / total_moles_ibutane
        ) * fuel_dictionary['i-butane']
883                 CO2_ibutane = (mols_CO2_ibutane/total_moles_ibutane) *
        fuel_dictionary['i-butane']
```

```
884                 H2O_ibutane = (mols_H2O_ibutane/total_moles_ibutane) *
        fuel_dictionary['i-butane']
885             else:
886                 pass
887 
888             if 'n-pentane' in fuel_dictionary:
889                 # n-Pentane Reaction
890                 mols_C5H12 = (fuel_dictionary['n-pentane'] * total_moles_fuel)
891                 C5H12_limit = mols_C5H12 * (6/1) # 6 mols H2O / 1 mol C5H12
892                 O2_limit_pentane = mols_O2_air * (6/8) #6 mols H2O / 8 mols O2
893 
894                 limiting_reactant_pentane = min(C5H12_limit, O2_limit_pentane)
895                 mols_CO2_pentane = limiting_reactant_pentane * (5/1) #5 mols
        CO2 / 1 mol C5H12
896                 mols_H2O_pentane = limiting_reactant_pentane * (6/1) #6 mols
        H2O / 1 mol C5H12
897                 mols_O2_not_reacted_pentane = mols_O2_air - mols_C5H12*8
898                 total_moles_pentane = mols_CO2_pentane + mols_H2O_pentane +
        mols_O2_not_reacted_pentane + N2_ng * fuel_dictionary ['n-pentane']
899                 O2_npentane = (mols_O2_not_reacted_pentane /
        total_moles_pentane) * fuel_dictionary['n-pentane']
900                 CO2_npentane = (mols_CO2_pentane/total_moles_pentane) *
        fuel_dictionary['n-pentane']
901                 H2O_npentane = (mols_H2O_pentane/total_moles_pentane) *
        fuel_dictionary['n-pentane']
902             else:
903                 pass
904 
905             if 'n-hexane' in fuel_dictionary:
906                 # n-Hexane Reaction
907                 mols_C6H14 = (fuel_dictionary['n-hexane'] * total_moles_fuel)
908                 C6H14_limit = mols_C6H14 * (7/1) #7 mols H2O / 1 mol C6H14
909                 O2_limit_hexane = mols_O2_air * (7/9.5) #7 mols H2O / 9.5 mols
        O2
910                 limiting_reactant_hexane = min(C6H14_limit, O2_limit_hexane)
911 
912                 mols_CO2_hexane = limiting_reactant_hexane* (6/1) #6 mols CO2 /
         1 mol C6H14
913                 mols_H2O_hexane = limiting_reactant_hexane * (7/1) #7 mols H2O
        / 1 mol C6H14
914                 mols_O2_not_reacted_hexane = mols_O2_air - mols_C6H14 * 9.5
915                 total_moles_hexane = mols_CO2_hexane + mols_H2O_hexane  +
        mols_O2_not_reacted_hexane + N2_ng * fuel_dictionary ['n-hexane']
916                 O2_nhexane = (mols_O2_not_reacted_hexane / total_moles_hexane)
        * fuel_dictionary['n-hexane']
917                 CO2_nhexane = (mols_CO2_hexane/total_moles_hexane) *
        fuel_dictionary['n-hexane']
918                 H2O_nhexane = (mols_H2O_hexane/total_moles_hexane) *
        fuel_dictionary['n-hexane']
919             else:
920                 pass
921 
922             #Calculating the new fractions of the product
923 
924             O2_out = O2_methane  + O2_propane + O2_nbutane + O2_ibutane +
        O2_npentane + O2_nhexane + O2_ethane
925             CO2_out = CO2_methane + CO2_ethane + CO2_propane + CO2_ibutane +
        CO2_nbutane + CO2_npentane + CO2_nhexane
```

```
926         H2O_out = H2O_methane + H2O_ethane + H2O_propane + H2O_ibutane + H2O_nbutane + H2O_npentane + H2O_nhexane
927         N2_out = N2_ng + mols_N2_air
928 
929         total_moles_out = O2_out + N2_out + CO2_out + H2O_out
930 
931         self.O2_outlet = O2_out / total_moles_out
932         self.N2_outlet = N2_out / total_moles_out
933         self.CO2_outlet = CO2_out / total_moles_out
934         self.H2O_outlet = H2O_out / total_moles_out
935 
936         self.reaction_fluid = fluid('srk')
937         self.reaction_fluid.addComponent('oxygen', self.O2_outlet)
938         self.reaction_fluid.addComponent('nitrogen', self.N2_outlet)
939         self.reaction_fluid.addComponent('CO2', self.CO2_outlet)
940         self.reaction_fluid.addComponent('H2O', self.H2O_outlet)
941         self.reaction_fluid.initProperties()
942 
943     def calc_outlet_temperature(self) -> float:
944         ''''
945         Method to calculate the outlet temperature of combustor - Turbine Inlet Temperature (TIT)
946 
947         First, the enthalpy of the air and fuel stream are calculated, and then added.
948 
949         The fuel used in the PHflash is based on an estimation of the composition of the combustor exhaust. Complete combustion is assumed.
950 
951         The PHflash method from neqsim was used for calculating the outlet temperature, based on the exhaust fluid and total enthalpy of the inlet of the combustor.
952         '''
953         enthalpy_air = self.get_inlet_stream().get_fluid().getEnthalpy()
954         enthalpy_fuel = self.get_LHV() * self.get_fuel_inlet_stream().get_flow_rate("kg/hr")/3600
955         enthalpy = enthalpy_air + enthalpy_fuel
956 
957         combustion_fluid = self.get_inlet_stream().get_fluid().clone()
958         #combustion_fluid = self.reaction_fluid
959         combustion_fluid.setPressure(self.get_inlet_stream().get_pressure("Pa")/1e5, "bara")
960         combustion_fluid.setTemperature(self.get_inlet_stream().get_temperature("K"), 'K')
961         combustion_fluid.setTotalFlowRate(self.get_inlet_stream().get_flow_rate("kg/hr") + self.get_fuel_inlet_stream().get_flow_rate("kg/hr"), "kg/hr")
962         combustion_fluid.initProperties()
963         TPflash(combustion_fluid)
964         PHflash(combustion_fluid, enthalpy)
965         self.outlet_temperature = combustion_fluid.getTemperature('K')
966 
967     def calc_outlet_stream(self):
968         chemical_reaction = fluid ('srk')
969         chemical_reaction.addComponent('oxygen', self.O2_outlet)
970         chemical_reaction.addComponent('nitrogen', self.N2_outlet)
971         chemical_reaction.addComponent('CO2', self.CO2_outlet)
972         chemical_reaction.addComponent('H2O', self.H2O_outlet)
```

```
973         self.outlet_stream = Stream()
974         self.outlet_stream.set_fluid(chemical_reaction)
975 
976         if self.get_outlet_pressure("Pa") == None:
977             print("COMBUSTOR ERROR: pressure is NONE")
978         if self.get_outlet_temperature("K") == None:
979             print("COMBUSTOR ERROR: temperature is NONE")
980         else:
981             self.outlet_stream.set_fluid(chemical_reaction)
982             self.outlet_stream.set_temperature(self.get_outlet_temperature(
    "K"), "K")
983             self.outlet_stream.set_pressure(self.get_outlet_pressure("Pa"),
     "Pa")
984             self.outlet_stream.set_flow_rate(self.get_inlet_stream().
    get_flow_rate("kg/hr") + self.get_fuel_inlet_stream().get_flow_rate("kg/hr"
    ),"kg/hr")
985             self.outlet_stream.calc()
986 
987 
988     def calc_outlet_stream_old(self):
989             self.outlet_stream = copy.copy(self.inlet_stream)
990             self.outlet_fluid = self.inlet_stream.get_fluid().clone()
991             if self.get_outlet_pressure("Pa") == None:
992                 print("COMBUSTOR ERROR: pressure is NONE")
993             if self.get_outlet_temperature("K") == None:
994                 print("COMBUSTOR ERROR: temperature is NONE")
995             else:
996                 self.outlet_stream.set_fluid(self.outlet_fluid)
997                 self.outlet_stream.set_temperature(self.
    get_outlet_temperature("K"), "K")
998                 self.outlet_stream.set_pressure(self.get_outlet_pressure("
    Pa"), "Pa")
999                 self.outlet_stream.set_flow_rate(self.get_inlet_stream().
    get_flow_rate("kg/hr") + self.get_fuel_inlet_stream().get_flow_rate("kg/hr"
    ),"kg/hr")
1000                self.outlet_stream.calc()
1001
1002    def set_excess_air(self, excess_air):
1003        self.excess_air = excess_air
1004
1005    def calc(self):
1006        '''
1007        Function to verify if the necessary variables were set for
    calculating the combustor.
1008
1009        If not, an error will appear.
1010
1011        '''
1012        self.success = False
1013        if self.fuel_inlet_stream is not None:
1014            if self.deltaP is not None:
1015                self.calc_LHV()
1016                self.calc_outlet_pressure()
1017                self.calc_chemical_reaction()
1018                self.calc_outlet_temperature()
1019                self.calc_heat_input()
1020                self.calc_outlet_stream()
1021                self.success = True
```

```
1022                else:
1023                    print("Combustor Error. Only fuel inlet stream was defined.
    ")
1024            elif self.deltaP is not None:
1025                if self.fuel_inlet_stream is not None:
1026                    self.calc_LHV()
1027                    self.calc_outlet_temperature()
1028                    self.calc_chemical_reaction()
1029                    self.calc_heat_input()
1030                    self.calc_outlet_pressure()
1031                    self.calc_outlet_stream()
1032                    self.success = True
1033                else:
1034                    print("Combustor error. Only pressure loss inside the
    combustor was defined.")
1035            elif self.outlet_temperature is not None:
1036                if self.fuel_inlet_stream and self.deltaP is not None:
1037                    self.calc_LHV()
1038                    self.calc_heat_input()
1039                    self.calc_chemical_reaction()
1040                    self.calc_outlet_pressure()
1041                    self.calc_outlet_stream()
1042                    self.success = True
1043                else:
1044                    print("Combustor error. Only outlet temperature of
    combustor was defined.")
1045            else:
1046                print("Combustor Error. Fuel inlet stream and combustor
    pressure loss need to be defined.")
1047
1048 class Turbine():
1049     '''
1050     Object to define turbine design parameters
1051
1052     Calculate isentropic efficiency outlet temperature of turbine, polytropic
         efficiency, as well as turbine work.
1053
1054     Neqsim will be used for calculating the fluid properties.
1055     '''
1056     def __init__(self): #To avoid class data shared among instances
1057         self.inlet_stream = None #Stream Class
1058         self.outlet_stream = None #Stream Class
1059         self.outlet_temperature = None #K
1060         self.outlet_pressure = None #Pa
1061         self.inlet_pressure = None #Pa
1062         self.polytropic_efficiency = None #no units
1063         self.isentropic_efficiency = None #no units
1064         self.temperature_isentropic = None #K
1065         self.deltaP = None #no units
1066         self.pressure_ratio = None #no units
1067         self.atmospheric_pressure = 1.013e5 #Pa, for now, this is constant.
    Later, change it to variable, so add set and get methods for it
1068         self.work = None
1069         self.mechanical_efficiency = None #no units
1070
1071     def set_inlet_stream(self, inlet_stream:Stream) -> Stream:
1072         self.inlet_stream = inlet_stream
1073
```

```
1074     def get_inlet_stream(self) -> Stream:
1075         return self.inlet_stream
1076
1077     def get_outlet_stream(self) -> Stream:
1078         return self.outlet_stream
1079
1080     def set_outlet_temperature(self, outlet_temperature:float, units:str) ->
     float:
1081             if units == "K":
1082                 return self.outlet_temperature
1083             elif units == "C":
1084                 self.outlet_temperature = self.outlet_temperature + 273.15
1085             elif units == "F":
1086                 self.outlet_temperature = (9/5) * self.outlet_temperature + 32
1087             else:
1088                 print("No units found.")
1089             return self.outlet_temperature
1090
1091     def get_outlet_temperature(self,units:str) ->float:
1092             if units == "K":
1093                 return self.outlet_temperature
1094             else:
1095                 print("No units found for temperature.")
1096                 return None
1097
1098     def set_atmospheric_pressure(self, atmospheric_pressure:float, units:str)
     -> float:
1099         if units =='Pa':
1100             return self.atmospheric_pressure
1101         elif units == 'bara':
1102             self.atmospheric_pressure = self.atmospheric_pressure*1e5
1103         else:
1104             print("No units found.")
1105
1106     def get_atmospheric_pressure(self, units:str) -> float:
1107         if units == 'Pa':
1108             return self.atmospheric_pressure
1109         else:
1110             print("Error. No units found for atmospheric pressure")
1111             return None
1112
1113     def calc_outlet_temperature(self) -> float:
1114         '''
1115         Function to calculate turbine outlet temperature
1116         '''
1117         steps = 150 #Fixed
1118         Pr = self.get_outlet_pressure("Pa")/self.get_inlet_stream().
     get_pressure("Pa")
1119         T3 = self.get_inlet_stream().get_temperature("K")
1120         Pin = self.get_inlet_stream().get_pressure("Pa")
1121         self.temperature_isentropic = 288.15
1122         i = 0
1123         for i in range(steps):
1124             k = self.get_inlet_stream().get_fluid().getKappa()
1125             Pr = (self.get_outlet_pressure("Pa")/self.get_inlet_stream().
     get_pressure("Pa"))**(i/steps)
1126             self.temperature_isentropic = self.get_inlet_stream().
     get_temperature("K") * (Pr) ** ((k-1)/k)
```

```
1127            self.outlet_temperature = self.get_inlet_stream().get_temperature("K")
        - ((self.get_inlet_stream().get_temperature("K") - self.
        temperature_isentropic) * self.get_isentropic_efficiency())
1128
1129
1130        def set_outlet_pressure(self, outlet_pressure:float, units:str) -> float:
1131                if units == "Pa":
1132                    self.outlet_pressure = outlet_pressure
1133                elif units == "bara":
1134                    self.outlet_pressure = outlet_pressure*1e5
1135                else:
1136                    print(f"Error: Pressure units not found for compressor {self}."
        )
1137
1138        def get_outlet_pressure(self,units:str) -> float:
1139                if units == "Pa":
1140                    return self.outlet_pressure
1141                else:
1142                    print(f"No units found for pressure for the STREAM {self}")
1143                return None
1144
1145        def calc_outlet_pressure(self):
1146            self.outlet_pressure = self.get_atmospheric_pressure("Pa") + self.
        get_deltaP("Pa")
1147
1148        def set_deltaP(self, deltaP:float, units:str) -> float:
1149            if units == 'Pa':
1150                self.deltaP = deltaP
1151            elif units == 'bara':
1152                self.deltaP = deltaP * 1e5
1153            elif units == 'mbar':
1154                self.deltaP = deltaP * 100
1155            else:
1156                print(f"Error: Pressure units not found for turbine {self}.")
1157
1158        def get_deltaP(self, units:str) -> float:
1159            if units == 'Pa':
1160                return self.deltaP
1161            else:
1162                print(f"No units found for pressure differential in turbine {self}.
        ")
1163
1164        def set_isentropic_efficiency(self, isentropic_efficiency:float) -> float:
1165            self.isentropic_efficiency = isentropic_efficiency
1166
1167        def get_isentropic_efficiency(self) -> float:
1168            return self.isentropic_efficiency
1169
1170        def calc_isentropic_efficiency(self):
1171            k = self.get_inlet_stream().get_fluid().getKappa()
1172            P3 = self.get_inlet_stream().get_pressure("Pa")
1173            P4 = self.get_outlet_pressure("Pa")
1174            np = self.get_polytropic_efficiency()
1175            Pr = P4/P3
1176            self.isentropic_efficiency = (1-Pr **((k-1)/k*np))/(1-Pr**((k-1)/k))
1177            def isentropic_efficiency_error(ni):
1178                P1 = self.get_inlet_stream().get_pressure("Pa")
1179                P2 = self.get_outlet_pressure("Pa")
```

```
1180            k = self.get_inlet_stream().get_fluid().getKappa()
1181            polytropic_efficiency = self.get_polytropic_efficiency()
1182            difference = (1 - (P2/P1)**((k-1)*ni/k))/(1 - (P2/P1)**((k-1)/k)) -
     polytropic_efficiency
1183            ni = ni_guess
1184            return difference
1185        ni_guess = self.get_polytropic_efficiency()
1186        ni_solution = fsolve(isentropic_efficiency_error, ni_guess)
1187
1188
1189    def set_polytropic_efficiency(self, polytropic_efficiency:float) -> float:
1190        self.polytropic_efficiency = polytropic_efficiency
1191
1192    def get_polytropic_efficiency(self) -> float:
1193        return self.polytropic_efficiency
1194
1195    def calc_polytropic_efficiency(self):
1196        '''
1197        Function to calculate turbine polytropic efficiency.
1198
1199        Neqsim will be used for calculating the isentropic properties using a
    PSflash, as well as the inlet and outlet properties of the fluid.
1200
1201       '''
1202        def polytropic_efficiency_error(np):
1203            P1 = self.get_inlet_stream().get_pressure("Pa")
1204            P2 = self.get_outlet_stream().get_pressure("Pa")
1205            k = self.inlet_stream.get_fluid().getKappa()
1206            isentropic_efficiency = self.get_isentropic_efficiency()
1207            difference = (1 - (P2/P1)**((k-1)*np/k))/(1 - (P2/P1)**((k-1)/k)) -
     isentropic_efficiency
1208            np = np_guess
1209            return difference
1210
1211        np_guess = self.get_isentropic_efficiency()
1212        np_solution = fsolve(polytropic_efficiency_error, np_guess)
1213        self.polytropic_efficiency = np_solution[0]
1214
1215    def set_mechanical_efficiency(self, mechanical_efficiency:float) -> float:
1216        self.mechanical_efficiency = mechanical_efficiency
1217
1218    def get_mechanical_efficiency(self):
1219        return self.mechanical_efficiency
1220
1221    def set_work(self, work:float, units:str) -> float:
1222        if units == "MW":
1223            self.work = work
1224        elif units == "kW":
1225            self.work = work*1e3
1226        elif units == "W":
1227            self.work = work*1e6
1228        else:
1229            print(f"No units found for work in {self}.")
1230            return self.work
1231
1232    def get_work(self, units:str) -> float:
1233        if units == "MW":
1234            return self.work
```

```
        else:
            print(f"No units found for {self} turbine work.")
            return None

    def calc_work(self):
        '''
        Function to calculate turbine work.

        A PS Flash is used to calculate the isentropic properties of the outlet
 stream.

        Neqsim will read the properties of inlet and outlet stream fluids to
perform the calculations.

        '''
        self.Cp = self.get_inlet_stream().get_fluid().getCp("J/kgK")
        self.work = self.get_mechanical_efficiency() * (self.get_inlet_stream()
.get_flow_rate("kg/hr")/3600 * self.Cp * (self.get_inlet_stream().
get_temperature("K") - self.get_outlet_stream().get_temperature("K")))/1e6


    def calc_outlet_stream(self) -> Stream:
                self.outlet_stream = copy.copy(self.inlet_stream)
                self.outlet_fluid = self.inlet_stream.get_fluid().clone()
                if self.get_outlet_pressure("Pa") == None:
                    print("TURBINE ERROR: pressure is NONE")
                if self.get_outlet_temperature("K") == None:
                    print("TURBINE ERROR: temperature is NONE")
                else:
                    self.outlet_stream.set_fluid(self.outlet_fluid)
                    self.outlet_stream.set_temperature(self.
get_outlet_temperature("K"), "K")
                    self.outlet_stream.set_pressure(self.get_outlet_pressure("
Pa"), "Pa")
                    self.outlet_stream.calc()

    def calc(self):
        '''
        Function for verifying if the necessary variables are defined for the
turbine. If two variables are given, the other ones are calculated.

        If not, an error will appear.
        '''
        self.success = False

        if self.isentropic_efficiency is not None and self.
get_mechanical_efficiency() is not None:
            if self.deltaP is not None:
                self.calc_outlet_pressure()
                self.calc_outlet_temperature()
                self.calc_outlet_stream()
                self.calc_polytropic_efficiency()
                self.calc_work()
                self.success = True

            elif self.outlet_pressure is not None and self.
get_mechanical_efficiency() is not None:
                self.calc_outlet_temperature()
```

```
1284                self.calc_outlet_stream()
1285                self.calc_polytropic_efficiency()
1286                self.calc_work()
1287                self.success = True
1288
1289            elif self.outlet_temperature is not None and self.
    get_mechanical_efficiency() is not None:
1290                self.calc_outlet_pressure()
1291                self.calc_outlet_stream()
1292                self.calc_polytropic_efficiency()
1293                self.calc_work()
1294                self.success = True
1295
1296            elif self.polytropic_efficiency is not None and self.
    get_mechanical_efficiency() is not None:
1297                self.calc_outlet_pressure()
1298                self.calc_outlet_temperature()
1299                self.calc_outlet_stream()
1300                self.calc_work()
1301                self.success = True
1302
1303            elif self.work is not None and self.get_mechanical_efficiency() is
    not None:
1304                self.calc_outlet_pressure()
1305                self.calc_outlet_temperature()
1306                self.calc_outlet_stream()
1307                self.calc_polytropic_efficiency()
1308                self.success = True
1309            else:
1310                print(f"Error. Only isentropic efficiency is defined for
    turbine {self}.")
1311
1312        elif self.polytropic_efficiency is not None:
1313            if self.outlet_pressure is not None:
1314                self.calc_isentropic_efficiency()
1315                self.calc_outlet_temperature()
1316                self.calc_outlet_stream()
1317                self.calc_work()
1318                self.success = True
1319
1320        elif self.deltaP is not None and self.get_mechanical_efficiency() is
    not None: #change here, from outlet P to deltaP for Thermoflow comparison
1321            if self.pressure_ratio is not None:
1322                self.calc_outlet_temperature()
1323                self.calc_outlet_stream()
1324                self.calc_isentropic_efficiency()
1325                self.calc_polytropic_efficiency()
1326                self.calc_work()
1327                self.success = True
1328
1329            elif self.isentropic_efficiency is not None and self.
    get_mechanical_efficiency() is not None:
1330                self.calc_outlet_temperature()
1331                self.calc_outlet_stream()
1332                self.calc_polytropic_efficiency()
1333                self.calc_work()
1334                self.success = True
1335
```

```
1336            elif self.outlet_temperature is not None and self.
        get_mechanical_efficiency() is not None:
1337                self.calc_outlet_stream()
1338                self.calc_isentropic_efficiency()
1339                self.calc_polytropic_efficiency()
1340                self.calc_work()
1341                self.sucess = True
1342
1343            elif self.polytropic_efficiency is not None and self.
        get_mechanical_efficiency() is not None:
1344                self.calc_outlet_pressure()
1345                self.calc_isentropic_efficiency()
1346                self.calc_outlet_temperature()
1347                self.calc_outlet_stream()
1348                self.calc_work()
1349                self.sucess = True
1350
1351            elif self.work is not None and self.get_mechanical_efficiency() is
        not None:
1352                self.calc_isentropic_efficiency()
1353                self.calc_outlet_temperature()
1354                self.calc_outlet_stream()
1355                self.calc_polytropic_efficiency()
1356                self.sucess = True
1357
1358            else:
1359                print(f"Error. Only outlet pressure is defined for turbine {
        self}.")
1360
1361        elif self.outlet_temperature is not None and self.
        get_mechanical_efficiency() is not None:
1362            if self.pressure_ratio is not None and self.
        get_mechanical_efficiency() is not None:
1363                self.calc_outlet_pressure()
1364                self.calc_outlet_stream()
1365                self.calc_isentropic_efficiency()
1366                self.calc_polytropic_efficiency()
1367                self.calc_work()
1368                self.sucess = True
1369
1370            elif self.inlet_pressure is not None and self.
        get_mechanical_efficiency() is not None:
1371                self.calc_outlet_pressure()
1372                self.calc_outlet_stream()
1373                self.calc_isentropic_efficiency()
1374                self.calc_polytropic_efficiency()
1375                self.calc_work()
1376                self.sucess = True
1377
1378            elif self.isentropic_efficiency is not None and self.
        get_mechanical_efficiency() is not None:
1379                self.calc_outlet_pressure()
1380                self.calc_outlet_stream()
1381                self.calc_polytropic_efficiency()
1382                self.calc_work()
1383                self.success = True
1384
1385            elif self.polytropic_efficiency is not None and self.
```

```
get_mechanical_efficiency() is not None:
1386                self.calc_isentropic_efficiency()
1387                self.calc_outlet_pressure()
1388                self.calc_outlet_stream()
1389                self.calc_work()
1390                self.success = True
1391
1392            elif self.work is not None and self.get_mechanical_efficiency() is
not None:
1393                self.calc_outlet_pressure()
1394                self.calc_outlet_stream()
1395                self.calc_isentropic_efficiency()
1396                self.calc_polytropic_efficiency()
1397                self.success = True
1398
1399            else:
1400                print(f"Turbine Error. Only outlet temperature is defined for
turbine {self}.")
1401
1402        elif self.work is not None and self.get_mechanical_efficiency() is not
None:
1403            if self.outlet_temperature is not None:
1404                self.calc_outlet_pressure()
1405                self.calc_outlet_stream()
1406                self.calc_isentropic_efficiency()
1407                self.calc_polytropic_efficiency()
1408                self.success = True
1409
1410            elif self.outlet_pressure is not None and self.
get_mechanical_efficiency() is not None:
1411                self.calc_outlet_pressure()
1412                self.calc_outlet_temperature()
1413                self.calc_outlet_stream()
1414                self.calc_isentropic_efficiency()
1415                self.calc_polytropic_efficiency()
1416                self.sucess = True
1417
1418            elif self.pressure_ratio is not None and self.
get_mechanical_efficiency() is not None:
1419                self.calc_outlet_pressure()
1420                self.calc_outlet_temperature()
1421                self.calc_outlet_stream()
1422                self.calc_isentropic_efficiency()
1423                self.calc_polytropic_efficiency()
1424                self.success = True
1425
1426            elif self.isentropic_efficiency is not None and self.
get_mechanical_efficiency() is not None:
1427                self.calc_outlet_temperature()
1428                self.calc_outlet_pressure()
1429                self.calc_polytropic_efficiency()
1430                self.success = True
1431
1432            elif self.polytropic_efficiency is not None and self.
get_mechanical_efficiency() is not None:
1433                self.calc_outlet_pressure()
1434                self.calc_outlet_temperature()
1435                self.calc_outlet_stream()
```

```
1436                 self.calc_isentropic_efficiency()
1437                 self.success = True
1438             else:
1439                 print(f"Turbine error. Only duty is defined for turbine {self}.
     ")
1440         else:
1441             print(f"Error. Two variables need to be defined for turbine {self}.
     ")
1442
1443
1444
1445 class GasTurbine():
1446     '''
1447     Object to calculate GT parameters, such as overall efficiency and total
     work.
1448
1449     Compressor and Turbine parameters will be used for the method inside this
     class
1450     '''
1451     def __init__(self):
1452         self.gt_work = None #MW
1453         self.mechanical_efficiency = None #No units
1454         self.generator_efficiency = None #no units
1455         self.gt_efficiency = None #no units
1456         self.gt_specific_power = None #MJ/kg
1457         self.turbine = None #Class
1458         self.compressor = None #Class
1459         self.combustor = None #Class
1460         self.gross_power_output = None
1461         self.gross_efficiency = None
1462
1463
1464     def set_turbine(self, turbine:Turbine) -> Turbine:
1465         self.turbine = turbine
1466
1467     def get_turbine(self):
1468         return self.turbine
1469
1470     def set_compressor(self, compressor:Compressor) ->  Compressor:
1471         self.compressor = compressor
1472
1473     def get_compressor(self):
1474         return self.compressor
1475
1476     def set_combustor(self, combustor:Combustor) -> Combustor:
1477         self.combustor = combustor
1478
1479     def get_combustor(self):
1480         return self.combustor
1481
1482     def get_gt_work(self, units:str) -> float:
1483         if units == 'MW':
1484             return self.gt_work
1485         else:
1486             print("No units found for Gas Turbine work.")
1487
1488     def set_gt_work(self, gt_work:float, units:str) -> float:
1489         if units == 'MW':
```

```
1490            self.gt_work = gt_work
1491
1492    def calc_gt_work(self):
1493            self.gt_work = (self.turbine.get_work("MW") - self.compressor.
    get_work("MW")) * self.mechanical_efficiency *self.generator_efficiency
1494
1495    def set_gt_specific_power(self, gt_specific_power:float) -> float:
1496        self.gt_specific_power = gt_specific_power
1497
1498    def get_gt_specific_power(self):
1499        return self.gt_specific_power
1500
1501    def calc_gt_specific_power(self):
1502        self.gt_specific_power = self.get_gt_work("MW")/(self.compressor.
    get_inlet_stream().get_flow_rate("kg/hr")/3600)  #MJ/kg of air
1503
1504    def set_gt_efficiency(self, gt_efficiency:float) -> float:
1505        self.gt_efficiency = gt_efficiency
1506
1507    def calc_gt_efficiency(self):
1508        self.gt_efficiency = (self.get_gt_work("MW")/(self.combustor.
    get_fuel_inlet_stream().get_flow_rate("kg/hr")/3600 * self.combustor.
    get_LHV()/1e6)) * 100
1509
1510    def get_gt_efficiency(self):
1511        return self.gt_efficiency
1512
1513    def get_gross_power_output(self, units:str):
1514        if units == "MW":
1515            return self.gross_power_output
1516        else:
1517            print("No units found for gross power output of turbine.")
1518
1519    def calc_gross_power_output(self):
1520        self.gross_power_output = (self.turbine.get_work("MW") - self.
    compressor.get_work("MW")) /(self.generator_efficiency * self.
    mechanical_efficiency)
1521
1522    def calc_gross_efficiency(self):
1523        self.gross_efficiency = self.get_gross_power_output("MW")/(self.
    combustor.get_fuel_inlet_stream().get_flow_rate("kg/hr")/3600 * self.
    combustor.get_LHV()/1e6)
1524
1525    def get_gross_efficiency(self):
1526        return self.gross_efficiency
1527
1528
1529
1530    def calc(self):
1531        '''
1532        Function to verify if all the necessary variables are defined before
    performing GT calculations.
1533
1534        If not, an error will appear.
1535        '''
1536        self.success = False
1537
1538        if self.turbine.get_work("MW") and self.compressor.get_work("MW") is
```

```
    not None:
1539                self.calc_gt_work()
1540                self.calc_gt_efficiency()
1541                self.calc_gt_specific_power()
1542                self.calc_gross_power_output()
1543                self.calc_gross_efficiency()
1544                self.success = True
1545
1546        elif self.gt_work is not None:
1547                self.calc_gt_efficiency()
1548                self.calc_gt_specific_power()
1549                self.calc_gross_power_output()
1550                self.calc_gross_efficiency()
1551                self.success = True
1552
1553        else:
1554            print(f"Gas Turbine error. Verify if generator efficiency,
    compressor and turbine work are defined for {self} gas turbine.")
1555
1556
1557 #Design Calculations
1558
1559 fluid_package = "srk"
1560 air = fluid(fluid_package)
1561 air.addComponent("nitrogen", 0.7981)
1562 air.addComponent("oxygen", 0.2019)
1563 air.addComponent("methane", 0.0)
1564 air.addComponent("H2O", 0.0)
1565 air.addComponent("hydrogen", 0.0)
1566 air.addComponent("CO2", 0.0)
1567 air.setMixingRule(2)
1568
1569 air_stream = Stream() #create object string
1570 air_stream.set_temperature(288.15,"K")
1571 air_stream.set_pressure(1.013, "bara")
1572 air_stream.set_flow_rate(500,"kg/s")
1573 air_stream.set_fluid(air) #assign an air stream with a fluid (read the fluid
    from neqsim)'
1574 air_stream.calc()
1575
1576 methane_fluid = fluid(fluid_package)
1577 methane_fluid.addComponent("nitrogen", 0.0)
1578 methane_fluid.addComponent("oxygen", 0.0)
1579 methane_fluid.addComponent("argon", 0.0)
1580 methane_fluid.addComponent("methane", 1.0)
1581 methane_fluid.addComponent("hydrogen",0.0)
1582 methane_fluid.addComponent("H2O", 0.0)
1583 methane_fluid.addComponent("CO2", 0.0)
1584 methane_fluid.addComponent('propane', 0.0)
1585 methane_fluid.addComponent('ethane', 0.0)
1586 methane_fluid.addComponent('n-butane', 0.0)
1587 methane_fluid.addComponent('i-butane', 0.0)
1588 methane_fluid.addComponent('n-hexane', 0.0)
1589 methane_fluid.addComponent('n-pentane', 0.0)
1590 methane_fluid.setMixingRule(2)
1591
1592 my_compressor = Compressor()
1593 my_compressor.set_inlet_stream(air_stream)
```

```
1594 my_compressor.set_isentropic_efficiency(0.858)
1595 my_compressor.set_pressure_ratio(10.7)
1596 my_compressor.set_steps(100)
1597 my_compressor.calc()
1598
1599 methane_stream = Stream()
1600 methane_stream.set_temperature(288.15,"K")
1601 methane_stream.set_pressure(my_compressor.get_outlet_stream().get_pressure("Pa"
     ), "Pa")
1602 methane_stream.set_flow_rate(5,"kg/s")
1603 methane_stream.set_fluid(methane_fluid) #assign an air stream with a fluid (
     read the fluid from neqsim)
1604 methane_stream.calc()
1605
1606 my_combustor = Combustor()
1607 my_combustor.set_inlet_stream(my_compressor.get_outlet_stream())
1608 my_combustor.set_fuel_inlet_stream(methane_stream) #assign methane stream as
     fuel inlet stream
1609 my_combustor.set_deltaP(1.5)
1610 my_combustor.calc()
1611
1612 my_turbine = Turbine()
1613 my_turbine.set_inlet_stream(my_combustor.get_outlet_stream())
1614 my_turbine.set_isentropic_efficiency(0.884)
1615 my_turbine.set_deltaP(45e-3, "bara")
1616 my_turbine.set_outlet_pressure(1.013e5,"Pa")
1617 my_turbine.set_mechanical_efficiency(0.985)
1618 my_turbine.calc()
1619
1620 my_gt = GasTurbine()
1621 my_gt.turbine = my_turbine
1622 my_gt.compressor = my_compressor
1623 my_gt.combustor = my_combustor
1624 my_gt.generator_efficiency = 0.986
1625 my_gt.mechanical_efficiency = 1
1626 my_gt.calc()
1627
1628 #Off - Design Calculation
1629
1630 #Reference case - design case for Gas Turbine, calculated above
1631
1632 #New inlet_mass_flow
1633
1634 #R = 8.314 J/molK
1635
1636
1637 air = fluid("srk")
1638 air.addComponent("nitrogen", 0.7981)
1639 air.addComponent("oxygen", 0.2019)
1640 air.addComponent("argon", 0.0)
1641 air.addComponent("methane", 0.0)
1642 air.addComponent("H2O", 0.0)
1643 air.setMixingRule(2)
1644
1645 #New air flow rate
1646 air_stream_off_design = Stream() #create object string
1647 air_stream_off_design.set_temperature(288.15,"K") # -> change this T!
1648 air_stream_off_design.set_pressure(1.013e5, "Pa")
```

```
1649
1650 reference_flow_rate = my_compressor.get_inlet_stream().get_flow_rate("kg/hr")
1651 reference_T = my_compressor.get_inlet_stream().get_temperature("K")
1652 off_design_inlet_T = air_stream_off_design.get_temperature("K")
1653 off_design_flow_rate = reference_flow_rate * sqrt(reference_T/
         off_design_inlet_T) #new mass flow rate of air, according to off design
         formula
1654 air_stream_off_design.set_flow_rate(off_design_flow_rate,"kg/hr")
1655 air_stream_off_design.set_fluid(air)
1656 air_stream_off_design.calc()
1657
1658 #Compressor Off Design
1659
1660 off_design_compressor = Compressor()
1661 off_design_compressor.set_inlet_stream(air_stream_off_design)
1662 off_design_compressor.set_polytropic_efficiency(my_compressor.
         get_polytropic_efficiency())
1663
1664 MW = air_stream_off_design.get_fluid().getMolarMass()
1665 R = 8.314 / MW #J/kgK
1666 Cp = my_compressor.get_inlet_stream().get_fluid().getCp("J/kgK")
1667 P1 = my_compressor.get_inlet_stream().get_pressure("Pa")
1668 T0 = 288.15 #K
1669 np = my_compressor.get_polytropic_efficiency()
1670 m1 = air_stream_off_design.get_flow_rate("kg/hr")
1671 LHV = my_combustor.get_LHV()
1672 reference_delta_P = (my_combustor.get_deltaP()/100)
1673 reference_P3 = my_combustor.get_outlet_stream().get_pressure("Pa")
1674 reference_T3 = my_combustor.get_outlet_temperature("K")
1675 reference_m3 = my_combustor.get_outlet_stream().get_flow_rate("kg/hr")
1676 reference_MW = my_combustor.get_outlet_stream().get_fluid().getMolarMass()
1677 reference_fuel_rate = my_combustor.get_fuel_inlet_stream().get_flow_rate("kg/hr
         ")
1678 T1 = air_stream_off_design.get_temperature("K")
1679 P2_new = 0
1680 mfuel = my_combustor.get_fuel_inlet_stream().get_flow_rate("kg/hr")
1681
1682 while (True):
1683
1684     iteration = 0
1685     max_iterations = 1000
1686
1687     #Update P2, T3 and MW - Recycle
1688     if P2_new > 0 and T3_new > 0:
1689         P2_guess = P2_new
1690         T3_guess = T3_new
1691     else:
1692         P2_guess = my_compressor.get_outlet_stream().get_pressure("Pa")
1693         T3_guess = my_combustor.get_outlet_stream().get_temperature("K")
1694
1695     tolerance_T = 5
1696     tolerance = 0.1
1697
1698     while (True):
1699
1700         T2 = T1 * ((P2_guess / P1 ) ** (R / (Cp * np)))
1701
1702         #Calculate air fluid properties at T2 and P2_guess
```

```
1703        air = fluid("srk")
1704        air.addComponent("nitrogen", 0.7981)
1705        air.addComponent("oxygen", 0.2019)
1706        air.addComponent("argon", 0.0)
1707        air.addComponent("methane", 0.0)
1708        air.addComponent("H2O", 0.0)
1709        air.setTemperature(T2, 'K')
1710        air.setPressure(P2_guess/1e5,'bara')
1711        air.setTotalFlowRate(off_design_flow_rate,'kg/hr')
1712        air.setMixingRule(2)
1713
1714        TPflash(air)
1715        air.initProperties()
1716        enthalpy_air = air.getEnthalpy('J')
1717        #enthalpy_air = off_design_compressor.get_inlet_stream().get_fluid().
    getEnthalpy()
1718        enthalpy_fuel = LHV * mfuel
1719
1720        #Total enthalpy
1721        enthalpy = enthalpy_air + enthalpy_fuel/3600
1722
1723        #combustion_fluid = off_design_compressor.get_inlet_stream().get_fluid
    ().clone()
1724        PHflash(air, enthalpy)
1725        T3_new = air.getTemperature('K')
1726
1727        #Calculating P2 based on new  values of T3 and mfuel
1728
1729        off_design_m3 = mfuel + off_design_flow_rate
1730        P3 = reference_P3 * (off_design_m3 / reference_m3) * sqrt(T3_new/
    reference_T3)
1731        delta_P = reference_delta_P * (off_design_m3 / reference_m3) **1.8 * ((
    T3_new * reference_P3)/(reference_T3 * P3))**0.8
1732        P2 = P3*(1 + delta_P)
1733        P2_new = P2
1734        iteration = iteration + 1
1735        diff_guess = abs(P2_guess - P2_new)
1736
1737
1738        if diff_guess <= tolerance and abs(T3_new - T3_guess) < tolerance_T:
1739            break
1740
1741        P2_guess = P2_new
1742        T3_guess = T3_new
1743        off_design_compressor.set_outlet_pressure(P2_new, "Pa")
1744        off_design_compressor.calc()
1745        #Off design combustor
1746
1747        methane_stream = Stream()
1748        methane_stream.set_temperature(air_stream_off_design.get_temperature("K
    "),"K")
1749        methane_stream.set_pressure(off_design_compressor.get_outlet_stream().
    get_pressure("Pa"), "Pa")
1750        methane_stream.set_flow_rate(mfuel,"kg/hr")
1751        methane_stream.set_fluid(methane_fluid) #assign an air stream with a
    fluid (read the fluid from neqsim)
1752        methane_stream.calc()
1753
```

```
1754         off_design_combustor = Combustor()
1755         off_design_combustor.set_inlet_stream(off_design_compressor.
     get_outlet_stream())
1756         off_design_combustor.set_fuel_inlet_stream(methane_stream) #assign
     methane stream as fuel inlet stream
1757         off_design_combustor.set_deltaP(delta_P*100)
1758         off_design_combustor.calc()
1759
1760         #Off design turbine
1761
1762         off_design_turbine = Turbine()
1763         off_design_turbine.set_inlet_stream(off_design_combustor.
     get_outlet_stream())
1764         off_design_turbine.set_polytropic_efficiency(my_turbine.
     get_polytropic_efficiency())
1765         off_design_turbine.set_deltaP(my_turbine.get_deltaP("Pa"),"Pa")
1766         off_design_turbine.set_outlet_pressure(1.013e5, "Pa")
1767         off_design_turbine.set_mechanical_efficiency(my_turbine.
     get_mechanical_efficiency())
1768         off_design_turbine.calc()
1769
1770         #Off design overall gas turbine
1771
1772         off_design_gt = GasTurbine()
1773         off_design_gt.compressor = off_design_compressor
1774         off_design_gt.combustor = off_design_combustor
1775         off_design_gt.turbine = off_design_turbine
1776         off_design_gt.mechanical_efficiency = 0.985
1777         off_design_gt.generator_efficiency = 0.986
1778         off_design_gt.calc()
1779
1780     #Convergence Criteria
1781     if diff_guess <= tolerance:
1782         break
1783
1784 #HYSYS Comparison - same procedure done for 0, 15 and 30C
1785
1786 import numpy as np
1787 import matplotlib.pyplot as plt
1788 import pandas as pd
1789
1790 fuel_flow_rates = np.arange(3, 7.25, 0.25) #kg/s - list of fuel flow rates to
     simulate
1791 outlet_compressor_T = []
1792 outlet_comp_Cp=[]
1793 outlet_comp_kappa = []
1794 power_compressor = []
1795 outlet_combustor_Cp = []
1796 inlet_turbine_T = []
1797 inlet_turbine_Cp = []
1798 outlet_turbine_T = []
1799 power_turbine = []
1800 efficiencies = []
1801 power_outputs = []
1802
1803
1804 for mfuel in fuel_flow_rates:
1805
```

```
1806        air = fluid("srk")
1807        air.addComponent("nitrogen", 0.7981)
1808        air.addComponent("oxygen", 0.2019)
1809        air.addComponent("argon", 0.0)
1810        air.addComponent("methane", 0.0)
1811        air.addComponent("H2O", 0.0)
1812        air.setMixingRule(2)
1813
1814        #New air flow rate
1815
1816        air_stream_off_design = Stream() #create object string
1817        air_stream_off_design.set_temperature(273.15,"K") # -> change this T!
1818        air_stream_off_design.set_pressure(1.013e5, "Pa")
1819
1820        reference_flow_rate = my_compressor.get_inlet_stream().get_flow_rate("kg/hr
            ")
1821        reference_T = my_compressor.get_inlet_stream().get_temperature("K")
1822        off_design_inlet_T = air_stream_off_design.get_temperature("K")
1823        off_design_flow_rate = reference_flow_rate * (reference_T/
            off_design_inlet_T) #new mass flow rate of air, according to off design
            formula
1824        air_stream_off_design.set_flow_rate(off_design_flow_rate,"kg/hr")
1825        air_stream_off_design.set_fluid(air)
1826        air_stream_off_design.calc()
1827
1828        #Compressor Off Design
1829
1830        off_design_compressor = Compressor()
1831        off_design_compressor.set_inlet_stream(air_stream_off_design)
1832        off_design_compressor.set_isentropic_efficiency(my_compressor.
            get_isentropic_efficiency())
1833
1834        MW = air_stream_off_design.get_fluid().getMolarMass()
1835        R = 8.314 / MW #J/kgK
1836        Cp = my_compressor.get_inlet_stream().get_fluid().getCp("J/kgK")
1837        P1 = my_compressor.get_inlet_stream().get_pressure("Pa")
1838        T0 = 288.15 #K
1839        np = my_compressor.get_polytropic_efficiency()
1840        m1 = air_stream_off_design.get_flow_rate("kg/hr")
1841        LHV = my_combustor.get_LHV()
1842        reference_delta_P = (my_combustor.get_deltaP()/100)
1843        reference_P3 = my_combustor.get_outlet_stream().get_pressure("Pa")
1844        reference_T3 = my_combustor.get_outlet_temperature("K")
1845        reference_m3 = my_combustor.get_outlet_stream().get_flow_rate("kg/hr")
1846        reference_MW = my_combustor.get_outlet_stream().get_fluid().getMolarMass()
1847        reference_fuel_rate = my_combustor.get_fuel_inlet_stream().get_flow_rate("
            kg/hr")
1848        T1 = air_stream_off_design.get_temperature("K")
1849        P2_new = 0
1850
1851
1852        while (True):
1853
1854            iteration = 0
1855            max_iterations = 1000
1856
1857            #Update P2, T3 and MW - Recycle
1858            if P2_new > 0 and T3_new > 0:
```

```
1859            P2_guess = P2_new
1860            T3_guess = T3_new
1861        else:
1862            P2_guess = my_compressor.get_outlet_stream().get_pressure("Pa")
1863            T3_guess = my_combustor.get_outlet_stream().get_temperature("K")
1864            tolerance_T = 10
1865            tolerance = 0.1
1866
1867        while (True):
1868
1869                T2 = T1 * ((P2_guess / P1 ) ** (R / (Cp * np)))
1870
1871                #Calculate air fluid properties at T2 and P2_guess
1872                air = fluid("srk")
1873                air.addComponent("nitrogen", 0.7981)
1874                air.addComponent("oxygen", 0.2019)
1875                air.addComponent("argon", 0.0)
1876                air.addComponent("methane", 0.0)
1877                air.addComponent("H2O", 0.0)
1878                air.setTemperature(T2, 'K')
1879                air.setPressure(P2_guess/1e5,'bara')
1880                air.setTotalFlowRate(off_design_flow_rate,'kg/hr')
1881                air.setMixingRule(2)
1882
1883                TPflash(air)
1884                air.initProperties()
1885                enthalpy_air = air.getEnthalpy('J')
1886                enthalpy_fuel = LHV * mfuel
1887
1888                #Total enthalpy
1889                enthalpy = enthalpy_air + enthalpy_fuel
1890
1891                #combustion_fluid = off_design_compressor.get_inlet_stream().
        get_fluid().clone()
1892                PHflash(air, enthalpy)
1893                T3_new = air.getTemperature('K')
1894
1895                #Calculating P2 based on new  values of T3 and mfuel
1896
1897                off_design_m3 = mfuel*3600 + off_design_flow_rate
1898                P3 = reference_P3 * (off_design_m3 / reference_m3) * sqrt(
        T3_new/ reference_T3)
1899                delta_P = reference_delta_P * (off_design_m3 / reference_m3)
        **1.8 * ((T3_new * reference_P3)/(reference_T3 * P3))**0.8
1900                P2 = P3*(1 + delta_P)
1901                P2_new = P2
1902                iteration = iteration + 1
1903                diff_guess = abs(P2_guess - P2_new)
1904
1905                if diff_guess <= tolerance and abs(T3_new - T3_guess) <
        tolerance_T:
1906                    break
1907
1908                P2_guess = P2_new
1909                T3_guess = T3_new
1910                off_design_compressor.set_outlet_pressure(P2_new, "Pa")
1911                off_design_compressor.calc()
1912
```

```
1913                #Off design combustor
1914
1915                methane_stream = Stream()
1916                methane_stream.set_temperature(288.15,"K")
1917                methane_stream.set_pressure(off_design_compressor.
     get_outlet_stream().get_pressure("Pa"), "Pa")
1918                methane_stream.set_flow_rate(mfuel,"kg/s")
1919                methane_stream.set_fluid(methane_fluid) #assign an air stream
     with a fluid (read the fluid from neqsim)
1920                methane_stream.calc()
1921
1922                off_design_combustor = Combustor()
1923                off_design_combustor.set_inlet_stream(off_design_compressor.
     get_outlet_stream())
1924                off_design_combustor.set_fuel_inlet_stream(methane_stream) #
     assign methane stream as fuel inlet stream
1925                off_design_combustor.set_deltaP(delta_P*100)
1926                off_design_combustor.calc()
1927
1928                #Off design turbine
1929
1930                off_design_turbine = Turbine()
1931                off_design_turbine.set_inlet_stream(off_design_combustor.
     get_outlet_stream())
1932                off_design_turbine.set_isentropic_efficiency(my_turbine.
     get_isentropic_efficiency())
1933                off_design_turbine.set_deltaP(my_turbine.get_deltaP("Pa"),"Pa")
1934                off_design_turbine.set_atmospheric_pressure(1.013e5, "Pa")
1935                off_design_turbine.set_mechanical_efficiency(my_turbine.
     get_mechanical_efficiency())
1936                off_design_turbine.calc()
1937
1938                #Off design overall gas turbine
1939
1940                off_design_gt = GasTurbine()
1941                off_design_gt.compressor = off_design_compressor
1942                off_design_gt.combustor = off_design_combustor
1943                off_design_gt.turbine = off_design_turbine
1944                off_design_gt.mechanical_efficiency = 0.985
1945                off_design_gt.generator_efficiency = 0.986
1946                off_design_gt.calc()
1947
1948                outlet_comp_kappa.append(off_design_compressor.
     get_outlet_stream().get_fluid().getKappa())
1949                outlet_comp_Cp.append(off_design_compressor.get_outlet_stream()
     .get_fluid().getCp("J/kgK"))
1950                outlet_compressor_T.append(off_design_gt.compressor.
     get_outlet_stream().get_temperature("K"))
1951                power_compressor.append(off_design_gt.compressor.get_work("MW")
     )
1952                outlet_turbine_T.append(off_design_turbine.get_outlet_stream().
     get_temperature("K"))
1953                power_turbine.append(off_design_gt.turbine.get_work("MW"))
1954                efficiencies.append(off_design_gt.get_gt_efficiency())
1955                power_outputs.append(off_design_gt.get_gt_work("MW"))
1956                inlet_turbine_Cp.append(off_design_turbine.get_inlet_stream().
     get_fluid().getCp("J/kgK"))
1957        #Convergence Criteria
```

```
1958        if diff_guess <= tolerance:
1959            break
1960
1961
1962 df = pd.read_excel(r'D:\Dados PC\Documents\Master thesis\Comparison_model.xlsx'
     , sheet_name='Off_design_OC')
1963 plt.plot(df['Power_output_MW'], df['Efficiency_'], label = 'HYSYS model', color
      = 'r')
1964 plt.plot(power_outputs,efficiencies, label = 'Python code', color = 'b' )
1965 plt.grid()
1966 plt.xlabel("Gas Turbine Power Output (MW)")
1967 plt.ylabel("Gas Turbine Efficiency (%)")
1968 plt.legend()
1969 plt.plot()
1970 #plt.savefig('D:\Dados PC\Documents\Master thesis\Images\OC_comparison_exhaustT
     .pdf', dpi = 300, bbox_inches = 'tight')
1971
1972 #Weather API connection for retrieving historical data
1973
1974 %pip install meteostat
1975
1976 #Historical Weather Data Connection
1977
1978 from datetime import datetime
1979 import matplotlib.pyplot as plt
1980 from meteostat import Point, Hourly
1981
1982 #Set time period
1983 start = datetime(2023,1,1,12)
1984 end = datetime(2023,5,1,12)
1985
1986 #Create point for Heidrun Platform location
1987 Heidrun_platform = Point(65.33, 2.33,16.0)
1988
1989 #Get Hourly data
1990 data = Hourly(Heidrun_platform, start, end)
1991
1992 data = data.fetch()
1993 # data.plot(y= ['tavg','tmin', 'tmax'])
1994 # plt.show()
1995
1996 data.plot(y=['temp'])
1997 plt.show()
1998
1999 #Extract average temperature values and store them in a list
2000
2001 tavg_list = []
2002 for row in data.itertuples():
2003     tavg_list.append(row.temp)
2004
2005 #Validation of model Thermoflow GTPRO - Design conditions
2006 #Siemens SGT-A35 RB with three shafts, PR of 21.7 according to Thermoflow
2007
2008 air = fluid("srk")
2009 air.addComponent("nitrogen", 0.76)
2010 air.addComponent("oxygen", 0.141)
2011 air.addComponent("argon", 0.009)
2012 air.addComponent("methane", 0.0)
```

```
2013 air.addComponent("ethane", 0.0)
2014 air.addComponent("propane", 0.0)
2015 air.addComponent("i-butane",0.0)
2016 air.addComponent("n-butane", 0.0)
2017 air.addComponent("i-pentane", 0.0)
2018 air.addComponent("n-pentane", 0.0)
2019 air.addComponent("n-hexane", 0.0)
2020 air.addComponent("CO2", 0.034)
2021 air.setMixingRule(2)
2022
2023 inlet_air = Stream()
2024 inlet_air.set_fluid(air)
2025 inlet_air.set_flow_rate(94.12, "kg/s")
2026 inlet_air.set_pressure(1.01, "bara")
2027 inlet_air.set_temperature(15, "C")
2028 inlet_air.calc()
2029
2030 Compressor_ipc_thermoflow = Compressor()
2031 Compressor_ipc_thermoflow.set_inlet_stream(inlet_air)
2032 Compressor_ipc_thermoflow.set_pressure_ratio(5.243)
2033 Compressor_ipc_thermoflow.set_isentropic_efficiency(0.92)
2034 Compressor_ipc_thermoflow.calc()
2035
2036 Compressor_hpc_thermoflow = Compressor()
2037 Compressor_hpc_thermoflow.set_inlet_stream(Compressor_ipc_thermoflow.
         get_outlet_stream())
2038 Compressor_hpc_thermoflow.set_pressure_ratio(4.19)
2039 Compressor_hpc_thermoflow.set_isentropic_efficiency(0.89)
2040 Compressor_hpc_thermoflow.calc()
2041
2042 methane_fluid = fluid("srk")
2043 methane_fluid.addComponent("nitrogen", 0.003)
2044 methane_fluid.addComponent("oxygen", 0.0)
2045 methane_fluid.addComponent("argon", 0.0)
2046 methane_fluid.addComponent("methane", 0.816)
2047 methane_fluid.addComponent("ethane", 0.089)
2048 methane_fluid.addComponent("propane", 0.042)
2049 methane_fluid.addComponent("i-butane",0.009)
2050 methane_fluid.addComponent("n-butane", 0.014)
2051 methane_fluid.addComponent("n-pentane", 0.003)
2052 methane_fluid.addComponent("n-hexane", 0.003)
2053 methane_fluid.addComponent("CO2", 0.019)
2054 methane_fluid.setMixingRule(2)
2055
2056 fuel = Stream()
2057 fuel.set_fluid(methane_fluid)
2058 fuel.set_flow_rate(1.71, "kg/s")
2059 fuel.set_pressure(30.78, "bara")
2060 fuel.set_temperature(267, "C")
2061 fuel.calc()
2062
2063 combustor_thermoflow = Combustor()
2064 combustor_thermoflow.set_inlet_stream(Compressor_hpc_thermoflow.
         get_outlet_stream())
2065 combustor_thermoflow.set_fuel_inlet_stream(fuel)
2066 combustor_thermoflow.set_deltaP(0)
2067 combustor_thermoflow.calc()
2068
```

```
2069 turbine_hp_thermoflow = Turbine()
2070 turbine_hp_thermoflow.set_inlet_stream(combustor_thermoflow.get_outlet_stream()
     )
2071 turbine_hp_thermoflow.set_outlet_pressure(4.747,"bara")
2072 turbine_hp_thermoflow.set_isentropic_efficiency(0.85)
2073 turbine_hp_thermoflow.set_mechanical_efficiency(1)
2074 turbine_hp_thermoflow.calc()
2075
2076 turbine_lp_thermoflow = Turbine()
2077 turbine_lp_thermoflow.set_inlet_stream(turbine_hp_thermoflow.get_outlet_stream
     ())
2078 turbine_lp_thermoflow.set_outlet_pressure(1.026, "bara")
2079 turbine_lp_thermoflow.set_isentropic_efficiency(0.8)
2080 turbine_lp_thermoflow.set_mechanical_efficiency(1)
2081 turbine_lp_thermoflow.calc()
2082
2083 gt_Thermoflow = GasTurbine()
2084 gt_Thermoflow.set_compressor(Compressor_hpc_thermoflow)
2085 gt_Thermoflow.set_combustor(combustor_thermoflow)
2086 gt_Thermoflow.set_turbine(turbine_hp_thermoflow)
2087 gt_Thermoflow.generator_efficiency = 0.9801
2088 gt_Thermoflow.mechanical_efficiency = 0.9902
2089 gt_Thermoflow.calc()
2090
2091 #Validation with Thermoflow, assuming one compressor only
2092 #Design conditions
2093
2094 air = fluid("srk")
2095 air.addComponent("nitrogen", 0.76)
2096 air.addComponent("oxygen", 0.141)
2097 air.addComponent("argon", 0.009)
2098 air.addComponent("methane", 0.0)
2099 air.addComponent("ethane", 0.0)
2100 air.addComponent("propane", 0.0)
2101 air.addComponent("i-butane",0.0)
2102 air.addComponent("n-butane", 0.0)
2103 air.addComponent("i-pentane", 0.0)
2104 air.addComponent("n-pentane", 0.0)
2105 air.addComponent("n-hexane", 0.0)
2106 air.addComponent("CO2", 0.034)
2107 air.setMixingRule(2)
2108
2109 inlet_air = Stream()
2110 inlet_air.set_fluid(air)
2111 inlet_air.set_flow_rate(94.12, "kg/s")
2112 inlet_air.set_pressure(1.01, "bara")
2113 inlet_air.set_temperature(15, "C")
2114 inlet_air.calc()
2115
2116 Compressor_thermoflow_ref = Compressor()
2117 Compressor_thermoflow_ref.set_inlet_stream(inlet_air)
2118 Compressor_thermoflow_ref.set_pressure_ratio(21.7)
2119 Compressor_thermoflow_ref.set_isentropic_efficiency(0.84)
2120 Compressor_thermoflow_ref.calc()
2121
2122 methane_fluid = fluid("srk")
2123 methane_fluid.addComponent("nitrogen", 0.003)
2124 methane_fluid.addComponent("oxygen", 0.0)
```

```
2125 methane_fluid.addComponent("argon", 0.0)
2126 methane_fluid.addComponent("methane", 0.816)
2127 methane_fluid.addComponent("ethane", 0.089)
2128 methane_fluid.addComponent("propane", 0.042)
2129 methane_fluid.addComponent("i-butane",0.009)
2130 methane_fluid.addComponent("n-butane", 0.014)
2131 methane_fluid.addComponent("n-pentane", 0.003)
2132 methane_fluid.addComponent("n-hexane", 0.003)
2133 methane_fluid.addComponent("CO2", 0.019)
2134 methane_fluid.setMixingRule(2)
2135
2136 fuel = Stream()
2137 fuel.set_fluid(methane_fluid)
2138 fuel.set_flow_rate(1.71, "kg/s")
2139 fuel.set_pressure(30.78, "bara")
2140 fuel.set_temperature(267, "C")
2141 fuel.calc()
2142
2143 combustor_thermoflow_ref = Combustor()
2144 combustor_thermoflow_ref.set_inlet_stream(Compressor_thermoflow_ref.
         get_outlet_stream())
2145 combustor_thermoflow_ref.set_fuel_inlet_stream(fuel)
2146 combustor_thermoflow_ref.set_deltaP(0)
2147 combustor_thermoflow_ref.calc()
2148
2149 turbine_thermoflow_ref = Turbine()
2150 turbine_thermoflow_ref.set_inlet_stream(combustor_thermoflow_ref.
         get_outlet_stream())
2151 turbine_thermoflow_ref.set_outlet_pressure(1.026, "bara")
2152 turbine_thermoflow_ref.set_isentropic_efficiency(0.87)
2153 turbine_thermoflow_ref.set_mechanical_efficiency(1)
2154 turbine_thermoflow_ref.calc()
2155
2156
2157 gt_Thermoflow = GasTurbine()
2158 gt_Thermoflow.set_compressor(Compressor_thermoflow_ref)
2159 gt_Thermoflow.set_combustor(combustor_thermoflow_ref)
2160 gt_Thermoflow.set_turbine(turbine_thermoflow_ref)
2161 gt_Thermoflow.generator_efficiency = 0.9801
2162 gt_Thermoflow.mechanical_efficiency = 0.9902
2163 gt_Thermoflow.calc()
2164
2165 #Off-design calculation with Thermoflow
2166
2167 #Assuming one compressor, and one turbine
2168
2169 #Reference case - design case for Gas Turbine, calculated above
2170
2171
2172 #R = 8.314 J/molK
2173
2174 air = fluid("srk")
2175 air.addComponent("nitrogen", 0.76)
2176 air.addComponent("oxygen", 0.141)
2177 air.addComponent("argon", 0.009)
2178 air.addComponent("methane", 0.0)
2179 air.addComponent("ethane", 0.0)
2180 air.addComponent("propane", 0.0)
```

```
2181 air.addComponent("i-butane",0.0)
2182 air.addComponent("n-butane", 0.0)
2183 air.addComponent("i-pentane", 0.0)
2184 air.addComponent("n-pentane", 0.0)
2185 air.addComponent("n-hexane", 0.0)
2186 air.addComponent("CO2", 0.034)
2187 air.setMixingRule(2)
2188
2189 #New air flow rate
2190 air_stream_off_design = Stream() #create object string
2191 air_stream_off_design.set_temperature(273.15,"K") # -> change this T!
2192 air_stream_off_design.set_pressure(1.010, "bara")
2193
2194 reference_flow_rate = Compressor_thermoflow_ref.get_inlet_stream().
         get_flow_rate("kg/hr")
2195 reference_T = Compressor_thermoflow_ref.get_inlet_stream().get_temperature("K")
2196 off_design_inlet_T = air_stream_off_design.get_temperature("K")
2197 off_design_flow_rate = reference_flow_rate * sqrt(reference_T/
         off_design_inlet_T) #new mass flow rate of air, according to off design
         formula
2198 air_stream_off_design.set_flow_rate(off_design_flow_rate,"kg/hr")
2199 air_stream_off_design.set_fluid(air)
2200 air_stream_off_design.calc()
2201
2202 #Compressor Off Design
2203
2204 off_design_compressor = Compressor()
2205 off_design_compressor.set_inlet_stream(air_stream_off_design)
2206 off_design_compressor.set_polytropic_efficiency(Compressor_thermoflow_ref.
         get_polytropic_efficiency())
2207
2208 MW = air_stream_off_design.get_fluid().getMolarMass()
2209 R = 8.314 / MW #J/kgK
2210 Cp = Compressor_thermoflow_ref.get_inlet_stream().get_fluid().getCp("J/kgK")
2211 P1 = Compressor_thermoflow_ref.get_inlet_stream().get_pressure("Pa")
2212 T0 = 288.15 #K
2213 np = Compressor_thermoflow_ref.get_polytropic_efficiency()
2214 m1 = air_stream_off_design.get_flow_rate("kg/hr")
2215 LHV = combustor_thermoflow_ref.get_LHV()
2216 reference_delta_P = (combustor_thermoflow_ref.get_deltaP()/100)
2217 reference_P3 = combustor_thermoflow_ref.get_outlet_stream().get_pressure("Pa")
2218 reference_T3 = combustor_thermoflow_ref.get_outlet_temperature("K")
2219 reference_m3 = combustor_thermoflow_ref.get_outlet_stream().get_flow_rate("kg/
         hr")
2220 reference_MW = combustor_thermoflow_ref.get_outlet_stream().get_fluid().
         getMolarMass()
2221 reference_fuel_rate = combustor_thermoflow_ref.get_fuel_inlet_stream().
         get_flow_rate("kg/hr")
2222 T1 = air_stream_off_design.get_temperature("K")
2223 P2_new = 0
2224 mfuel = combustor_thermoflow_ref.get_fuel_inlet_stream().get_flow_rate("kg/hr")
2225
2226
2227 print(np, "np")
2228
2229 methane_fluid = fluid("srk")
2230 methane_fluid.addComponent("nitrogen", 0.00)
2231 methane_fluid.addComponent("oxygen", 0.0)
```

```
2232 methane_fluid.addComponent("argon", 0.0)
2233 methane_fluid.addComponent("methane", 0.816)
2234 methane_fluid.addComponent("ethane", 0.089)
2235 methane_fluid.addComponent("propane", 0.042)
2236 methane_fluid.addComponent("i-butane",0.009)
2237 methane_fluid.addComponent("n-butane", 0.014)
2238 methane_fluid.addComponent("n-pentane", 0.003)
2239 methane_fluid.addComponent("n-hexane", 0.003)
2240 methane_fluid.addComponent("CO2", 0)
2241 methane_fluid.setMixingRule(2)
2242
2243 fuel = Stream()
2244 fuel.set_fluid(methane_fluid)
2245 fuel.set_flow_rate(1.71, "kg/s")
2246 fuel.set_pressure(30.78, "bara")
2247 fuel.set_temperature(267, "C")
2248 fuel.calc()
2249
2250 while (True):
2251
2252     iteration = 0
2253     max_iterations = 1000
2254
2255     #Update P2, T3 and MW - Recycle
2256     if P2_new > 0 and T3_new > 0:
2257         P2_guess = P2_new
2258         T3_guess = T3_new
2259     else:
2260         P2_guess = Compressor_thermoflow_ref.get_outlet_stream().get_pressure("
     Pa")
2261         T3_guess = combustor_thermoflow_ref.get_outlet_stream().get_temperature
     ("K")
2262
2263     tolerance_T = 5
2264     tolerance = 0.1
2265
2266     while (True):
2267
2268         T2 = T1 * ((P2_guess / P1 ) ** (R / (Cp * np)))
2269
2270         #Calculate air fluid properties at T2 and P2_guess
2271         air = fluid("srk")
2272         air.addComponent("nitrogen", 0.76)
2273         air.addComponent("oxygen", 0.141)
2274         air.addComponent("argon", 0.009)
2275         air.addComponent("methane", 0.0)
2276         air.addComponent("ethane", 0.0)
2277         air.addComponent("propane", 0.0)
2278         air.addComponent("i-butane",0.0)
2279         air.addComponent("n-butane", 0.0)
2280         air.addComponent("i-pentane", 0.0)
2281         air.addComponent("n-pentane", 0.0)
2282         air.addComponent("n-hexane", 0.0)
2283         air.addComponent("CO2", 0.034)
2284         air.setMixingRule(2)
2285         air.setTemperature(T2, 'K')
2286         air.setPressure(P2_guess/1e5,'bara')
2287         air.setTotalFlowRate(off_design_flow_rate,'kg/hr')
```

```
2288        air.setMixingRule(2)
2289
2290        TPflash(air)
2291        air.initProperties()
2292        enthalpy_air = air.getEnthalpy('J')
2293        enthalpy_fuel = LHV * mfuel/3600
2294
2295        #Total enthalpy
2296        enthalpy = enthalpy_air + enthalpy_fuel
2297        PHflash(air, enthalpy)
2298        T3_new = air.getTemperature('K')
2299
2300        #Calculating P2 based on new  values of T3 and mfuel
2301
2302        off_design_m3 = mfuel + off_design_flow_rate
2303        P3 = reference_P3 * (off_design_m3 / reference_m3) * sqrt(T3_new/
    reference_T3)
2304        delta_P = reference_delta_P * (off_design_m3 / reference_m3) **1.8 * ((
    T3_new * reference_P3)/(reference_T3 * P3))**0.8
2305        P2 = P3*(1 + delta_P)
2306        P2_new = P2
2307        iteration = iteration + 1
2308        diff_guess = abs(P2_guess - P2_new)
2309
2310        if diff_guess <= tolerance and abs(T3_new - T3_guess) < tolerance_T:
2311            break
2312
2313        P2_guess = P2_new
2314        T3_guess = T3_new
2315        off_design_compressor.set_outlet_pressure(P2_new/1e5, "bara")
2316        off_design_compressor.calc()
2317
2318        #Off design combustor
2319
2320        methane_stream = Stream()
2321        methane_stream.set_temperature(fuel.get_temperature("K"),"K")
2322        methane_stream.set_pressure(fuel.get_pressure("Pa"), "Pa")
2323        methane_stream.set_flow_rate(mfuel/3600,"kg/s")
2324        methane_stream.set_fluid(methane_fluid) #assign an air stream with a
    fluid (read the fluid from neqsim)
2325        methane_stream.calc()
2326
2327        off_design_combustor = Combustor()
2328        off_design_combustor.set_inlet_stream(off_design_compressor.
    get_outlet_stream())
2329        off_design_combustor.set_fuel_inlet_stream(methane_stream) #assign
    methane stream as fuel inlet stream
2330        off_design_combustor.set_deltaP(0)
2331        off_design_combustor.calc()
2332
2333        #Off design turbine
2334
2335        off_design_turbine = Turbine()
2336        off_design_turbine.set_inlet_stream(off_design_combustor.
    get_outlet_stream())
2337        off_design_turbine.set_polytropic_efficiency(turbine_thermoflow_ref.
    get_polytropic_efficiency())
2338        off_design_turbine.set_outlet_pressure(1.029, "bara")
```

```
2339         off_design_turbine.set_mechanical_efficiency(turbine_thermoflow_ref.
         get_mechanical_efficiency())
2340         off_design_turbine.calc()
2341
2342         #Off design overall gas turbine
2343
2344         off_design_gt = GasTurbine()
2345         off_design_gt.compressor = off_design_compressor
2346         off_design_gt.combustor = off_design_combustor
2347         off_design_gt.turbine = off_design_turbine
2348         off_design_gt.mechanical_efficiency = 0.9902
2349         off_design_gt.generator_efficiency = 0.9801
2350         off_design_gt.calc()
2351
2352     #Convergence Criteria
2353     if diff_guess <= tolerance:
2354         break
2355
2356 #Comparison of Thermoflow with Python code
2357
2358
2359 import numpy as np
2360 import matplotlib.pyplot as plt
2361 import pandas as pd
2362
2363
2364 data = pd.read_excel('D:\Dados PC\Documents\Master thesis\Comparison_model.xlsx
     ', sheet_name='Thermoflow_graphs')
2365
2366 fuel_flow_rates = data['GT_fuel_flow'].tolist()  #kg/s - list of fuel flow
     rates to simulate
2367 ambient_temperatures = data['Compressor_inlet_T'].tolist() #C
2368
2369 for i in range(len(ambient_temperatures)):
2370      ambient_temperatures[i] = ambient_temperatures[i] +273.15
2371
2372 inlet_t=[]
2373 air_flow=[]
2374 outlet_compressor_T = []
2375 outlet_comp_Cp=[]
2376 outlet_comp_kappa = []
2377 power_compressor = []
2378 outlet_combustor_Cp = []
2379 inlet_turbine_T = []
2380 inlet_turbine_kappa = []
2381 outlet_turbine_T = []
2382 power_turbine = []
2383 efficiencies = []
2384 power_outputs = []
2385 gross_power_output=[]
2386 gross_efficiency=[]
2387
2388 for mfuel, ambient_t in zip(fuel_flow_rates,ambient_temperatures):
2389
2390     air = fluid("srk")
2391     air.addComponent("nitrogen", 0.75808)
2392     air.addComponent("oxygen", 0.14100)
2393     air.addComponent("argon", 0.00913)
```

```
2394    air.addComponent("methane", 0.0)
2395    air.addComponent("ethane", 0.0)
2396    air.addComponent("propane", 0.0)
2397    air.addComponent("i-butane",0.0)
2398    air.addComponent("n-butane", 0.0)
2399    air.addComponent("i-pentane", 0.0)
2400    air.addComponent("n-pentane", 0.0)
2401    air.addComponent("n-hexane", 0.0)
2402    air.addComponent("CO2", 0.03355)
2403    air.setMixingRule(2)
2404
2405    #New air flow rate
2406
2407    air_stream_off_design = Stream() #create object string
2408    air_stream_off_design.set_temperature(ambient_t,"K") # -> change this T!
2409    air_stream_off_design.set_pressure(1.010e5, "Pa")
2410
2411    reference_flow_rate = Compressor_thermoflow_ref.get_inlet_stream().
        get_flow_rate("kg/hr")
2412    reference_T = Compressor_thermoflow_ref.get_inlet_stream().get_temperature(
        "K")
2413    off_design_inlet_T = air_stream_off_design.get_temperature("K")
2414    off_design_flow_rate = reference_flow_rate * sqrt(reference_T/
        off_design_inlet_T) #new mass flow rate of air, according to off design
        formula
2415    air_stream_off_design.set_flow_rate(off_design_flow_rate,"kg/hr")
2416    air_stream_off_design.set_fluid(air)
2417    air_stream_off_design.calc()
2418
2419    #Compressor Off Design
2420
2421    off_design_compressor = Compressor()
2422    off_design_compressor.set_inlet_stream(air_stream_off_design)
2423    off_design_compressor.set_polytropic_efficiency(Compressor_thermoflow_ref.
        get_polytropic_efficiency())
2424
2425    MW = air_stream_off_design.get_fluid().getMolarMass()
2426    R = 8.314 / MW #J/kgK
2427    Cp = Compressor_thermoflow_ref.get_inlet_stream().get_fluid().getCp("J/kgK"
        )
2428    P1 = Compressor_thermoflow_ref.get_inlet_stream().get_pressure("Pa")
2429    T0 = 288.15 #K
2430    np = turbine_thermoflow_ref.get_polytropic_efficiency()
2431    m1 = air_stream_off_design.get_flow_rate("kg/hr")
2432    LHV = combustor_thermoflow_ref.get_LHV()
2433    reference_delta_P = (combustor_thermoflow_ref.get_deltaP()/100)
2434    reference_P3 = combustor_thermoflow_ref.get_outlet_stream().get_pressure("
        Pa")
2435    reference_T3 = combustor_thermoflow_ref.get_outlet_temperature("K")
2436    reference_m3 = combustor_thermoflow_ref.get_outlet_stream().get_flow_rate("
        kg/hr")
2437    reference_MW = combustor_thermoflow_ref.get_outlet_stream().get_fluid().
        getMolarMass()
2438    reference_fuel_rate = combustor_thermoflow_ref.get_fuel_inlet_stream().
        get_flow_rate("kg/hr")
2439    T1 = air_stream_off_design.get_temperature("K")
2440    P2_new = 0
2441
```

```
2442
2443     while (True):
2444
2445         iteration = 0
2446         max_iterations = 1000
2447
2448         #Update P2, T3 and MW - Recycle
2449         if P2_new > 0 and T3_new > 0:
2450             P2_guess = P2_new
2451             T3_guess = T3_new
2452         else:
2453             P2_guess = Compressor_thermoflow_ref.get_outlet_stream().
    get_pressure("Pa")
2454             T3_guess = combustor_thermoflow_ref.get_outlet_stream().
    get_temperature("K")
2455
2456         tolerance_T = 5
2457         tolerance = 0.1
2458
2459         while (True):
2460
2461             T2 = T1 * ((P2_guess / P1 ) ** (R / (Cp * np)))
2462
2463             #Calculate air fluid properties at T2 and P2_guess
2464             air = fluid("srk")
2465             air.addComponent("nitrogen", 0.76)
2466             air.addComponent("oxygen", 0.141)
2467             air.addComponent("argon", 0.009)
2468             air.addComponent("methane", 0.0)
2469             air.addComponent("ethane", 0.0)
2470             air.addComponent("propane", 0.0)
2471             air.addComponent("i-butane",0.0)
2472             air.addComponent("n-butane", 0.0)
2473             air.addComponent("i-pentane", 0.0)
2474             air.addComponent("n-pentane", 0.0)
2475             air.addComponent("n-hexane", 0.0)
2476             air.addComponent("CO2", 0.034)
2477             air.setMixingRule(2)
2478             air.setTemperature(T2, 'K')
2479             air.setPressure(P2_guess/1e5,'bara')
2480             air.setTotalFlowRate(off_design_flow_rate,'kg/hr')
2481             air.setMixingRule(2)
2482
2483             TPflash(air)
2484             air.initProperties()
2485             enthalpy_air = air.getEnthalpy('J')
2486             enthalpy_fuel = LHV * mfuel
2487
2488             #Total enthalpy
2489             enthalpy = enthalpy_air + enthalpy_fuel
2490             PHflash(air, enthalpy)
2491             T3_new = air.getTemperature('K')
2492
2493             #Calculating P2 based on new  values of T3 and mfuel
2494
2495             off_design_m3 = mfuel + off_design_flow_rate
2496             P3 = reference_P3 * (off_design_m3 / reference_m3) * sqrt(T3_new/
    reference_T3)
```

```
2497            delta_P = reference_delta_P * (off_design_m3 / reference_m3) **1.8
    * ((T3_new * reference_P3)/(reference_T3 * P3))**0.8
2498            P2 = P3*(1 + delta_P)
2499            P2_new = P2
2500            iteration = iteration + 1
2501            diff_guess = abs(P2_guess - P2_new)
2502
2503            if diff_guess <= tolerance and abs(T3_new - T3_guess) < tolerance_T
    :
2504                break
2505
2506            P2_guess = P2_new
2507            T3_guess = T3_new
2508            off_design_compressor.set_outlet_pressure(P2_new/1e5, "bara")
2509            off_design_compressor.calc()
2510
2511            #Off design combustor
2512
2513            methane_stream = Stream()
2514            methane_stream.set_temperature(fuel.get_temperature("K"),"K")
2515            methane_stream.set_pressure(fuel.get_pressure("Pa"), "Pa")
2516            methane_stream.set_flow_rate(mfuel,"kg/s")
2517            methane_stream.set_fluid(methane_fluid) #assign an air stream with
    a fluid (read the fluid from neqsim)
2518            methane_stream.calc()
2519
2520            off_design_combustor = Combustor()
2521            off_design_combustor.set_inlet_stream(off_design_compressor.
    get_outlet_stream())
2522            off_design_combustor.set_fuel_inlet_stream(methane_stream) #assign
    methane stream as fuel inlet stream
2523            off_design_combustor.set_deltaP(0)
2524            off_design_combustor.calc()
2525
2526            #Off design turbine
2527
2528            off_design_turbine = Turbine()
2529            off_design_turbine.set_inlet_stream(off_design_combustor.
    get_outlet_stream())
2530            off_design_turbine.set_polytropic_efficiency(turbine_thermoflow_ref
    .get_polytropic_efficiency())
2531            off_design_turbine.set_outlet_pressure(turbine_thermoflow_ref.
    get_outlet_pressure("Pa"), "Pa")
2532            off_design_turbine.set_outlet_pressure(1.0296e5, "Pa")
2533            off_design_turbine.set_mechanical_efficiency(turbine_thermoflow_ref
    .get_mechanical_efficiency())
2534            off_design_turbine.calc()
2535
2536            #Off design overall gas turbine
2537
2538            off_design_gt = GasTurbine()
2539            off_design_gt.compressor = off_design_compressor
2540            off_design_gt.combustor = off_design_combustor
2541            off_design_gt.turbine = off_design_turbine
2542            off_design_gt.mechanical_efficiency = 0.9902
2543            off_design_gt.generator_efficiency = 0.9801
2544            off_design_gt.calc()
2545
```

```
2546            outlet_comp_kappa.append(off_design_compressor.get_outlet_stream().
    get_fluid().getKappa())
2547            outlet_comp_Cp.append(off_design_compressor.get_outlet_stream().
    get_fluid().getCp("J/kgK"))
2548            outlet_compressor_T.append(off_design_gt.compressor.
    get_outlet_stream().get_temperature("K"))
2549            power_compressor.append(off_design_gt.compressor.get_work("MW"))
2550            power_turbine.append(off_design_gt.turbine.get_work("MW"))
2551            efficiencies.append(off_design_gt.get_gt_efficiency())
2552            power_outputs.append(off_design_gt.get_gt_work("MW"))
2553            outlet_turbine_T.append(off_design_turbine.get_outlet_stream().
    get_temperature("K")-273.15)
2554            gross_power_output.append(off_design_gt.get_gross_power_output("MW"
    ))
2555            gross_efficiency.append(off_design_gt.get_gross_efficiency())
2556            air_flow.append(off_design_compressor.get_inlet_stream().
    get_flow_rate("kg/hr")/1e3)
2557            inlet_t.append(off_design_compressor.get_inlet_stream().
    get_temperature("K")-273.15)
2558
2559            #Convergence Criteria
2560        if diff_guess <= tolerance:
2561            break
2562
2563
2564 # print(mfuel)
2565 # print(ambient_t)
2566 df = pd.read_excel('D:\Dados PC\Documents\Master thesis\Comparison_model.xlsx',
     sheet_name='Thermoflow_graphs')
2567 plt.plot(df['GT_gross_power_MW'], df['GT_eff'],  label = 'Thermoflow data',
    color = 'r')
2568 plt.plot(power_outputs, efficiencies,  label = 'Python code', color = 'g' )
2569 plt.grid()
2570 plt.xlabel("Gas Turbine Power Output - MW")
2571 plt.ylabel("Gas Turbine Efficiency - %")
2572 plt.legend()
2573 plt.plot()
2574
2575 #plt.savefig('D:\Dados PC\Documents\Master thesis\Images\Thermoflow_airflow.pdf
    ', dpi = 300, bbox_inches = 'tight')
2576
2577 %pip install meteostat
2578
2579 #Historical Weather Data - Validation of the model
2580
2581 from datetime import datetime
2582 import matplotlib.pyplot as plt
2583 from meteostat import Point, Hourly
2584
2585 #Set time period
2586 start = datetime(2023,1,1,12)
2587 end = datetime(2023,5,1,12)
2588
2589 #Create point for Heidrun
2590 Heidrun_platform = Point(65.33, 2.33,16.0)
2591
2592 #Get daily data
2593
```

```
2594 #data = Daily(Heidrun_platform, start, end)
2595 data = Hourly(Heidrun_platform, start, end)
2596
2597 data = data.fetch()
2598 # data.plot(y= ['tavg','tmin', 'tmax'])
2599 # plt.show()
2600
2601 data.plot(y=['temp'])
2602 plt.xlabel('Time')
2603 plt.ylabel('Temperature (C)')
2604 plt.grid()
2605 plt.plot()
2606
2607 #Extract tavg values and store them in a list
2608
2609 tavg_list = []
2610 day_list = []
2611 month_list = []
2612 timestamps = []
2613 for row in data.itertuples():
2614     timestamps.append(row.Index)
2615
2616 print(timestamps)
2617
2618 for row in data.itertuples():
2619     tavg_list.append(row.temp)
2620     day_list.append(row.Index.day)
2621     month_list.append(row.Index.month)
2622
2623
2624 ################################################################
2625
2626 import statsmodels.api as sm
2627 import numpy as np
2628 import pandas as pd
2629 import matplotlib.pyplot as plt
2630
2631 # Linear regression relating fuel intake with temperature
2632
2633 fuel_flow_rates = [1.82, 1.82, 1.82, 1.82, 1.81, 1.80, 1.80, 1.79, 1.78, 1.77,
    1.76, 1.75, 1.74, 1.73, 1.72, 1.71, 1.70, 1.69, 1.68, 1.67, 1.67] #kg/s
2634
2635 ambient_temperatures = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ,11 ,12 ,13 ,14 ,15
     ,16 ,17 ,18 ,19 ,20] #C
2636
2637 # Fit the model
2638 X = sm.add_constant(ambient_temperatures)
2639 model = sm.OLS(fuel_flow_rates, X)
2640 results = model.fit()
2641
2642 # Print the results
2643 print(results.summary())
2644
2645 # Get the fitted parameters
2646 intercept = results.params[0]
2647 slope = results.params[1]
2648
2649 # Plot the data and the fitted line
```

```
2650 plt.scatter(ambient_temperatures, fuel_flow_rates)
2651 plt.show()
2652
2653 import numpy as np
2654 import matplotlib.pyplot as plt
2655 import pandas as pd
2656
2657 ambient_temperature_list = tavg_list
2658
2659 time_period = timestamps
2660 efficiencies = []
2661 power_outputs = []
2662 fuel_flow_rates=[]
2663 inlet_t=[]
2664
2665 for i in range(len(ambient_temperature_list)):
2666     t = ambient_temperature_list[i]
2667     fuel_flow = 1.8403 - 0.0086 * t
2668     fuel_flow_rates.append(fuel_flow)
2669
2670 fuel_inlet = fuel_flow_rates
2671
2672 for t, mfuel in zip (ambient_temperature_list, fuel_inlet):
2673
2674     air = fluid("srk")
2675     air.addComponent("nitrogen", 0.75808)
2676     air.addComponent("oxygen", 0.14100)
2677     air.addComponent("argon", 0.00913)
2678     air.addComponent("methane", 0.0)
2679     air.addComponent("ethane", 0.0)
2680     air.addComponent("propane", 0.0)
2681     air.addComponent("i-butane",0.0)
2682     air.addComponent("n-butane", 0.0)
2683     air.addComponent("i-pentane", 0.0)
2684     air.addComponent("n-pentane", 0.0)
2685     air.addComponent("n-hexane", 0.0)
2686     air.addComponent("CO2", 0.03355)
2687     air.setMixingRule(2)
2688
2689     #New air flow rate
2690
2691     air_stream_off_design = Stream() #create object string
2692     air_stream_off_design.set_temperature(t,"C") # -> change this T!
2693     air_stream_off_design.set_pressure(1.010e5, "Pa")
2694
2695     reference_flow_rate = Compressor_thermoflow_ref.get_inlet_stream().
         get_flow_rate("kg/hr")
2696     reference_T = Compressor_thermoflow_ref.get_inlet_stream().get_temperature(
         "K")
2697     off_design_inlet_T = air_stream_off_design.get_temperature("K")
2698     off_design_flow_rate = reference_flow_rate * sqrt(reference_T/
         off_design_inlet_T) #new mass flow rate of air, according to off design
         formula
2699     air_stream_off_design.set_flow_rate(off_design_flow_rate,"kg/hr")
2700     air_stream_off_design.set_fluid(air)
2701     air_stream_off_design.calc()
2702
2703     #Compressor Off Design
```

```
2704
2705     off_design_compressor = Compressor()
2706     off_design_compressor.set_inlet_stream(air_stream_off_design)
2707     off_design_compressor.set_polytropic_efficiency(Compressor_thermoflow_ref.
    get_polytropic_efficiency())
2708
2709     MW = air_stream_off_design.get_fluid().getMolarMass()
2710     R = 8.314 / MW #J/kgK
2711     Cp = Compressor_thermoflow_ref.get_inlet_stream().get_fluid().getCp("J/kgK"
    )
2712     P1 = Compressor_thermoflow_ref.get_inlet_stream().get_pressure("Pa")
2713     T0 = 288.15 #K
2714     np = turbine_thermoflow_ref.get_polytropic_efficiency()
2715     m1 = air_stream_off_design.get_flow_rate("kg/hr")
2716     LHV = combustor_thermoflow_ref.get_LHV()
2717     reference_delta_P = (combustor_thermoflow_ref.get_deltaP()/100)
2718     reference_P3 = combustor_thermoflow_ref.get_outlet_stream().get_pressure("
    Pa")
2719     reference_T3 = combustor_thermoflow_ref.get_outlet_temperature("K")
2720     reference_m3 = combustor_thermoflow_ref.get_outlet_stream().get_flow_rate("
    kg/hr")/3600
2721     reference_MW = combustor_thermoflow_ref.get_outlet_stream().get_fluid().
    getMolarMass()
2722     reference_fuel_rate = combustor_thermoflow_ref.get_fuel_inlet_stream().
    get_flow_rate("kg/hr")
2723     T1 = air_stream_off_design.get_temperature("K")
2724     P2_new = 0
2725
2726     while (True):
2727
2728         iteration = 0
2729         max_iterations = 1000
2730
2731         #Update P2, T3 and MW - Recycle
2732         if P2_new > 0 and T3_new > 0:
2733             P2_guess = P2_new
2734             T3_guess = T3_new
2735         else:
2736             P2_guess = Compressor_thermoflow_ref.get_outlet_stream().
    get_pressure("Pa")
2737             T3_guess = combustor_thermoflow_ref.get_outlet_stream().
    get_temperature("K")
2738
2739         tolerance_T = 5
2740         tolerance = 0.1
2741
2742         while (True):
2743
2744             T2 = T1 * ((P2_guess / P1 ) ** (R / (Cp * np)))
2745
2746             #Calculate air fluid properties at T2 and P2_guess
2747             air = fluid("srk")
2748             air.addComponent("nitrogen", 0.76)
2749             air.addComponent("oxygen", 0.141)
2750             air.addComponent("argon", 0.009)
2751             air.addComponent("methane", 0.0)
2752             air.addComponent("ethane", 0.0)
2753             air.addComponent("propane", 0.0)
```

```
2754            air.addComponent("i-butane",0.0)
2755            air.addComponent("n-butane", 0.0)
2756            air.addComponent("i-pentane", 0.0)
2757            air.addComponent("n-pentane", 0.0)
2758            air.addComponent("n-hexane", 0.0)
2759            air.addComponent("CO2", 0.034)
2760            air.setMixingRule(2)
2761            air.setTemperature(T2, 'K')
2762            air.setPressure(P2_guess/1e5,'bara')
2763            air.setTotalFlowRate(off_design_flow_rate,'kg/hr')
2764            air.setMixingRule(2)
2765
2766            TPflash(air)
2767            air.initProperties()
2768            enthalpy_air = air.getEnthalpy('J')
2769            #enthalpy_air = off_design_compressor.get_inlet_stream().get_fluid
    ().getEnthalpy()
2770            enthalpy_fuel = LHV * mfuel
2771
2772            #Total enthalpy
2773            enthalpy = enthalpy_air + enthalpy_fuel
2774
2775            #combustion_fluid = off_design_compressor.get_inlet_stream().
    get_fluid().clone()
2776            PHflash(air, enthalpy)
2777            T3_new = air.getTemperature('K')
2778
2779            #Calculating P2 based on new  values of T3 and mfuel
2780
2781            off_design_m3 = mfuel + off_design_flow_rate/3600
2782            P3 = reference_P3 * (off_design_m3 / reference_m3) * sqrt(T3_new/
    reference_T3)
2783            delta_P = reference_delta_P * (off_design_m3 / reference_m3) **1.8
    * ((T3_new * reference_P3)/(reference_T3 * P3))**0.8
2784            P2 = P3*(1 + delta_P)
2785            P2_new = P2
2786            iteration = iteration + 1
2787            diff_guess = abs(P2_guess - P2_new)
2788
2789            if diff_guess <= tolerance and abs(T3_new - T3_guess) < tolerance_T
    :
2790                break
2791
2792            P2_guess = P2_new
2793            T3_guess = T3_new
2794            off_design_compressor.set_outlet_pressure(P2_new/1e5, "bara")
2795            off_design_compressor.calc()
2796
2797            #Off design combustor
2798
2799            methane_stream = Stream()
2800            methane_stream.set_temperature(fuel.get_temperature("K"),"K")
2801            methane_stream.set_pressure(fuel.get_pressure("Pa"), "Pa")
2802            methane_stream.set_flow_rate(mfuel,"kg/s")
2803            methane_stream.set_fluid(methane_fluid) #assign an air stream with
    a fluid (read the fluid from neqsim)
2804            methane_stream.calc()
2805
```

```
2806            off_design_combustor = Combustor()
2807            off_design_combustor.set_inlet_stream(off_design_compressor.
        get_outlet_stream())
2808            off_design_combustor.set_fuel_inlet_stream(methane_stream) #assign
        methane stream as fuel inlet stream
2809            off_design_combustor.set_deltaP(0)
2810            off_design_combustor.calc()
2811
2812            #Off design turbine
2813            off_design_turbine = Turbine()
2814            off_design_turbine.set_inlet_stream(off_design_combustor.
        get_outlet_stream())
2815            off_design_turbine.set_polytropic_efficiency(turbine_thermoflow_ref
        .get_polytropic_efficiency())
2816            off_design_turbine.set_outlet_pressure(turbine_thermoflow_ref.
        get_outlet_pressure("Pa"), "Pa")
2817            off_design_turbine.set_outlet_pressure(1.0296e5, "Pa")
2818            off_design_turbine.set_mechanical_efficiency(turbine_thermoflow_ref
        .get_mechanical_efficiency())
2819            off_design_turbine.calc()
2820
2821            #Off design overall gas turbine
2822
2823            off_design_gt = GasTurbine()
2824            off_design_gt.compressor = off_design_compressor
2825            off_design_gt.combustor = off_design_combustor
2826            off_design_gt.turbine = off_design_turbine
2827            off_design_gt.mechanical_efficiency = 0.9902
2828            off_design_gt.generator_efficiency = 0.9801
2829            off_design_gt.calc()
2830
2831            efficiencies.append(off_design_gt.get_gt_efficiency())
2832            power_outputs.append(off_design_gt.get_gt_work("MW"))
2833            inlet_t.append(off_design_compressor.get_inlet_stream().
        get_temperature("K")-273.15)
2834
2835            #Convergence Criteria
2836        if diff_guess <= tolerance:
2837            break
2838
2839
2840 power = power_outputs
2841 temperature = tavg_list
2842
2843 # plt.show()
2844 plt.figure(figsize=(10, 6))
2845 plt.plot(time_period[:len(power_outputs)], power_outputs[:len(time_period)],
        color='r')  # Plotting time vs. power
2846 plt.xlabel("Time")
2847 plt.ylabel("Gas Turbine Power Output - MW")
2848 plt.grid()
2849
2850 plt.xticks(rotation=45)  # Rotate the x-axis labels by 45 degrees
2851
2852
2853 #
        ################################################################################
```

```
2854
2855 #Digital Twin Model
2856
2857 #Weather Forecast Connection
2858
2859 import requests
2860 from datetime import datetime, timedelta
2861
2862 # API URL
2863 url = "https://api.met.no/weatherapi/locationforecast/2.0/compact?altitude=30&
         lat=65.33&lon=7.32"
2864
2865 # Define User-Agent header
2866 headers = {
2867     'User-Agent': 'vmkaplan/PowerGeneration github.com/vmkaplan/PowerGeneration
         ',
2868     'Contact': 'victoriakaplan@hotmail.com'
2869 }
2870
2871 # Specify the time range for which you want to retrieve weather data
2872 now = datetime.now()
2873 start_time = now
2874 end_time = start_time + timedelta(days=12)
2875
2876 # Make GET request to API
2877 response = requests.get(url, headers=headers)
2878
2879 # Extract JSON response from the response body
2880 data = response.json()
2881
2882 # Extract time and temperature values
2883 timeseries = data['properties']['timeseries']
2884 current_time = now.strftime('%Y-%m-%dT%H:%M:%SZ')
2885
2886 time_period = []
2887 forecast_temperature = []
2888 time_plot = []
2889
2890 for ts in timeseries:
2891     time = ts['time']
2892     temperature = ts['data']['instant']['details']['air_temperature']
2893
2894     # Check if the time is equal to or greater than the current time
2895     if time >= current_time:
2896
2897         time_object = datetime.strptime(time, '%Y-%m-%dT%H:%M:%SZ')
2898         day = time_object.day
2899         month = time_object.month
2900         year = time_object.year
2901
2902         time_plot.append(f'{day}-{month}-{year}')
2903         time_period.append(time)
2904         forecast_temperature.append(temperature)
2905
2906 temperature = forecast_temperature
2907 time = time_period
2908
2909 #Digital twin for power production forecast
```

```
2910
2911 import numpy as np
2912 import matplotlib.pyplot as plt
2913 import pandas as pd
2914
2915 ambient_temperature_list = temperature
2916
2917 efficiencies = []
2918 power_outputs = []
2919 fuel_flow_rates=[]
2920 inlet_t=[]
2921
2922 for i in range(len(ambient_temperature_list)):
2923     t = ambient_temperature_list[i]
2924     fuel_flow = 1.8403 - 0.0086 * t
2925     fuel_flow_rates.append(fuel_flow)
2926
2927 fuel_inlet = fuel_flow_rates
2928
2929 for t, mfuel in zip (ambient_temperature_list, fuel_inlet):
2930
2931     air = fluid("srk")
2932     air.addComponent("nitrogen", 0.75808)
2933     air.addComponent("oxygen", 0.14100)
2934     air.addComponent("argon", 0.00913)
2935     air.addComponent("methane", 0.0)
2936     air.addComponent("ethane", 0.0)
2937     air.addComponent("propane", 0.0)
2938     air.addComponent("i-butane",0.0)
2939     air.addComponent("n-butane", 0.0)
2940     air.addComponent("i-pentane", 0.0)
2941     air.addComponent("n-pentane", 0.0)
2942     air.addComponent("n-hexane", 0.0)
2943     air.addComponent("CO2", 0.03355)
2944     air.setMixingRule(2)
2945
2946     #New air flow rate
2947
2948     air_stream_off_design = Stream() #create object string
2949     air_stream_off_design.set_temperature(t,"C") # -> change this T!
2950     air_stream_off_design.set_pressure(1.010e5, "Pa")
2951
2952     reference_flow_rate = Compressor_thermoflow_ref.get_inlet_stream().
         get_flow_rate("kg/hr")
2953     reference_T = Compressor_thermoflow_ref.get_inlet_stream().get_temperature(
         "K")
2954     off_design_inlet_T = air_stream_off_design.get_temperature("K")
2955     off_design_flow_rate = reference_flow_rate * sqrt(reference_T/
         off_design_inlet_T) #new mass flow rate of air, according to off design
         formula
2956     air_stream_off_design.set_flow_rate(off_design_flow_rate,"kg/hr")
2957     air_stream_off_design.set_fluid(air)
2958     air_stream_off_design.calc()
2959
2960     #Compressor Off Design
2961
2962     off_design_compressor = Compressor()
2963     off_design_compressor.set_inlet_stream(air_stream_off_design)
```

```
2964        off_design_compressor.set_polytropic_efficiency(Compressor_thermoflow_ref.
        get_polytropic_efficiency())
2965
2966        MW = air_stream_off_design.get_fluid().getMolarMass()
2967        R = 8.314 / MW #J/kgK
2968        Cp = Compressor_thermoflow_ref.get_inlet_stream().get_fluid().getCp("J/kgK"
        )
2969        P1 = Compressor_thermoflow_ref.get_inlet_stream().get_pressure("Pa")
2970        T0 = 288.15 #K
2971        np = turbine_thermoflow_ref.get_polytropic_efficiency()
2972        m1 = air_stream_off_design.get_flow_rate("kg/hr")
2973        LHV = combustor_thermoflow_ref.get_LHV()
2974        reference_delta_P = (combustor_thermoflow_ref.get_deltaP()/100)
2975        reference_P3 = combustor_thermoflow_ref.get_outlet_stream().get_pressure("
        Pa")
2976        reference_T3 = combustor_thermoflow_ref.get_outlet_temperature("K")
2977        reference_m3 = combustor_thermoflow_ref.get_outlet_stream().get_flow_rate("
        kg/hr")/3600
2978        reference_MW = combustor_thermoflow_ref.get_outlet_stream().get_fluid().
        getMolarMass()
2979        reference_fuel_rate = combustor_thermoflow_ref.get_fuel_inlet_stream().
        get_flow_rate("kg/hr")
2980        T1 = air_stream_off_design.get_temperature("K")
2981        P2_new = 0
2982
2983        while (True):
2984
2985            iteration = 0
2986            max_iterations = 1000
2987
2988            #Update P2, T3 and MW - Recycle
2989            if P2_new > 0 and T3_new > 0:
2990                P2_guess = P2_new
2991                T3_guess = T3_new
2992            else:
2993                P2_guess = Compressor_thermoflow_ref.get_outlet_stream().
        get_pressure("Pa")
2994                T3_guess = combustor_thermoflow_ref.get_outlet_stream().
        get_temperature("K")
2995
2996            tolerance_T = 5
2997            tolerance = 0.1
2998
2999            while (True):
3000
3001                T2 = T1 * ((P2_guess / P1 ) ** (R / (Cp * np)))
3002
3003                #Calculate air fluid properties at T2 and P2_guess
3004                air = fluid("srk")
3005                air.addComponent("nitrogen", 0.76)
3006                air.addComponent("oxygen", 0.141)
3007                air.addComponent("argon", 0.009)
3008                air.addComponent("methane", 0.0)
3009                air.addComponent("ethane", 0.0)
3010                air.addComponent("propane", 0.0)
3011                air.addComponent("i-butane",0.0)
3012                air.addComponent("n-butane", 0.0)
3013                air.addComponent("i-pentane", 0.0)
```

```
3014            air.addComponent("n-pentane", 0.0)
3015            air.addComponent("n-hexane", 0.0)
3016            air.addComponent("CO2", 0.034)
3017            air.setMixingRule(2)
3018            air.setTemperature(T2, 'K')
3019            air.setPressure(P2_guess/1e5,'bara')
3020            air.setTotalFlowRate(off_design_flow_rate,'kg/hr')
3021            air.setMixingRule(2)
3022
3023            TPflash(air)
3024            air.initProperties()
3025            enthalpy_air = air.getEnthalpy('J')
3026            #enthalpy_air = off_design_compressor.get_inlet_stream().get_fluid
     ().getEnthalpy()
3027            enthalpy_fuel = LHV * mfuel
3028
3029            #Total enthalpy
3030            enthalpy = enthalpy_air + enthalpy_fuel
3031
3032            #combustion_fluid = off_design_compressor.get_inlet_stream().
     get_fluid().clone()
3033            PHflash(air, enthalpy)
3034            T3_new = air.getTemperature('K')
3035
3036            #Calculating P2 based on new  values of T3 and mfuel
3037
3038            off_design_m3 = mfuel + off_design_flow_rate/3600
3039            P3 = reference_P3 * (off_design_m3 / reference_m3) * sqrt(T3_new/
     reference_T3)
3040            delta_P = reference_delta_P * (off_design_m3 / reference_m3) **1.8
     * ((T3_new * reference_P3)/(reference_T3 * P3))**0.8
3041            P2 = P3*(1 + delta_P)
3042            P2_new = P2
3043            iteration = iteration + 1
3044            diff_guess = abs(P2_guess - P2_new)
3045
3046            if diff_guess <= tolerance and abs(T3_new - T3_guess) < tolerance_T
     :
3047                break
3048
3049            P2_guess = P2_new
3050            T3_guess = T3_new
3051            off_design_compressor.set_outlet_pressure(P2_new/1e5, "bara")
3052            off_design_compressor.calc()
3053
3054            #Off design combustor
3055
3056            methane_stream = Stream()
3057            methane_stream.set_temperature(fuel.get_temperature("K"),"K")
3058            methane_stream.set_pressure(fuel.get_pressure("Pa"), "Pa")
3059            methane_stream.set_flow_rate(mfuel,"kg/s")
3060            methane_stream.set_fluid(methane_fluid) #assign an air stream with
     a fluid (read the fluid from neqsim)
3061            methane_stream.calc()
3062
3063            off_design_combustor = Combustor()
3064            off_design_combustor.set_inlet_stream(off_design_compressor.
     get_outlet_stream())
```

```
3065            off_design_combustor.set_fuel_inlet_stream(methane_stream) #assign
        methane stream as fuel inlet stream
3066            off_design_combustor.set_deltaP(0)
3067            off_design_combustor.calc()
3068
3069            #Off design turbine
3070            off_design_turbine = Turbine()
3071            off_design_turbine.set_inlet_stream(off_design_combustor.
        get_outlet_stream())
3072            off_design_turbine.set_polytropic_efficiency(turbine_thermoflow_ref
        .get_polytropic_efficiency())
3073            off_design_turbine.set_outlet_pressure(turbine_thermoflow_ref.
        get_outlet_pressure("Pa"), "Pa")
3074            off_design_turbine.set_outlet_pressure(1.0296e5, "Pa")
3075            off_design_turbine.set_mechanical_efficiency(turbine_thermoflow_ref
        .get_mechanical_efficiency())
3076            off_design_turbine.calc()
3077
3078            #Off design overall gas turbine
3079
3080            off_design_gt = GasTurbine()
3081            off_design_gt.compressor = off_design_compressor
3082            off_design_gt.combustor = off_design_combustor
3083            off_design_gt.turbine = off_design_turbine
3084            off_design_gt.mechanical_efficiency = 0.9902
3085            off_design_gt.generator_efficiency = 0.9801
3086            off_design_gt.calc()
3087
3088            efficiencies.append(off_design_gt.get_gt_efficiency())
3089            power_outputs.append(off_design_gt.get_gt_work("MW"))
3090            inlet_t.append(off_design_compressor.get_inlet_stream().
        get_temperature("K")-273.15)
3091
3092            #Convergence Criteria
3093        if diff_guess <= tolerance:
3094            break
3095
3096 power = power_outputs
3097 temperature = forecast_temperature
3098 time = time_period
3099 time_plot = time_plot
3100 plt.figure(figsize=(12, 6))
3101 plt.plot(time_plot, power_outputs[:len(time)], color='g')  # Plotting time vs.
        power
3102 plt.xlabel("Time", fontsize = '12')
3103 plt.ylabel("Power Output Forecast - MW", fontsize = '12')
3104 plt.grid()
3105
3106 plt.xticks(rotation=45)
3107
3108 plt.savefig('Power_forecastAPI.pdf', format = 'pdf')
3109 plt.show()
```

GasTurb Outputs

Date: 03jul23
Time: 17:07

Boosted Turboshaft
Alt= 0m ISA 60% Relative Humidity

Station	W kg/s	T K	P kPa	WRstd kg/s
amb		288,15	101,325	
1	94,120	288,15	101,325	
2	94,120	288,15	98,585	96,926
24	94,120	475,65	516,883	23,752
25	94,120	475,65	516,883	23,752
3	94,120	741,98	2165,742	7,080
31	94,120	741,98	2165,742	
4	95,840	1428,32	2165,742	10,002
41	95,840	1428,32	2165,742	10,002
43	95,840	1048,28	440,509	
44	95,840	1048,28	440,509	
45	95,840	1048,28	440,509	42,127
49	95,840	779,15	105,134	
5	95,840	779,15	105,134	152,174
6	95,840	779,15	105,134	
8	95,840	779,15	105,134	152,174
Bleed	0,000	741,98	2165,747	

Ps0-P2= 2,740 Ps8-Ps0= 1,245

Efficiencies:	isentr	polytr	RNI	P/P
Booster	0,9200	0,9360	0,973	5,243
Compressor	0,8529	0,8778	2,807	4,190
Burner	1,0000			1,000
HP Turbine	0,8500	0,8236	3,291	4,916
LP Turbine	0,8574	0,8335	0,954	4,190
Generator	0,9801			

HP Spool mech Eff 1,0000 Nom Spd 34000 rpm
PT Spool mech Eff 1,0000 Nom Spd 10000 rpm

hum [%]	war0	FHV	Fuel
60,0	0,00653	46,700	Generic

Parameter		Value	Unit
PWSD	=	29958,0	kW
PSFC	=	0,2067	kg/(kW*h)
V0	=	0,00	m/s
P25/P24	=	1,00000	
P3/P2	=	21,97	
FN res	=	10,08	kN
Heat Rate	=	9654,5	kJ/(kW*h)
WF	=	1,72037	kg/s
Loading	=	100,00	%
s NOx	=	0,6356	
Therm Eff	=	0,37288	
P45/P44	=	1,00000	
P6/P5	=	1,00000	
A8	=	1,98227	m^2
P8/Pamb	=	1,03759	
WBld/W2	=	0,00000	
P2/P1	=	0,97296	
Ps8	=	102,570	kPa

driven by HPT

Parameter		Value	Unit
WCHN/W25	=	0,00000	
WCHR/W25	=	0,00000	
e444 th	=	0,85000	
eta t-s	=	0,83955	
PW gen	=	29361,8	kW
TRQ	=	100,00	%
WCLN/W25	=	0,00000	
WCLR/W25	=	0,00000	

Input Data File:
C:\Users\Usuário\AppData\Roaming\GasTurb14\DemoData\Last_1_PRP.CYB (modified)

Date: 03jul23
Time: 17:15

Single Spool Turboshaft
Alt= 0m ISA 60% Relative Humidity

Station	W kg/s	T K	P kPa	WRstd kg/s
amb		288,15	101,325	
1	94,143	288,15	101,325	
2	94,143	288,15	101,325	94,324
3	94,143	751,27	2198,753	7,019
31	94,143	751,27	2198,753	
4	95,864	1436,26	2198,753	9,881
41	95,864	1436,26	2198,753	9,881
49	95,864	779,15	104,365	
5	95,864	779,15	104,365	153,327
6	95,864	779,15	104,365	
8	95,864	779,15	104,365	153,327
Coolg	0,000	751,27	2198,748	
Bleed	0,000	751,27	2198,748	

Ps0-P2= 0,000 Ps8-Ps0= 0,000

Efficiencies:	isentr	polytr	RNI	P/P
Compressor	0,8413	0,8921	1,000	21,700
Burner	1,0000			1,000
Turbine	0,8785	0,8300	3,320	21,068
Generator	1,0000			

Spool mech Eff 1,0000 Nom Spd 17000 rpm

hum [%]	war0	FHV	Fuel
60,0	0,00637	46,700	Generic

Parameter		Value	Unit
PWSD	=	29958,2	kW
PSFC	=	0,2068	kg/(kW*h)
Heat Rate	=	9657,1	kJ/(kW*h)
Therm Eff	=	0,3728	
WF	=	1,72086	kg/s
s NOx	=	0,67096	
incidence	=	0,00	°
XM8	=	0,2102	
A8	=	1,8327	m^2
P8/Ps8	=	1,03000	
Wcl L/W2	=	0,00000	
WBld/W2	=	0,00000	
P2/P1	=	1,00000	
Ps8	=	101,325	kPa
WCLN/W2	=	0,00000	
WCLR/W2	=	0,00000	
Loading	=	100,00	%
e45 th	=	0,87848	
PW gen	=	29958,2	kW
P6/P5	=	1,0000	

Input Data File:
C:\Users\Usuário\AppData\Roaming\GasTurb14\DemoData\singlespool.C1S (modified)

Thermoflow Outputs

Plant Summary

GT PRO 30.0 Usuário						
383 06-07-2023 13:56:22 file=D:\Dados PC\Documents\Thermoflow\MYFILES30\GTPRO2_sgta35_2.GTP						
Program revision date: March 3, 2022						
Plant Configuration: Simple Cycle Gas Turbine(s)						
One SIE SGT-A30 Engine (Curve Fit OEM Data Model #421), GT PRO Type 0, Subtype 0						
Steam Property Formulation: IFC-67						
Site ambient conditions: 1,013 bar, 15 C, 0,01% RH (3,227 C WB)						

SYSTEM SUMMARY						
	Power Output kW		LHV Heat Rate kJ/kWh		Elect. Eff. LHV%	
	@ gen. term.	net	@ gen. term.	net	@ gen. term.	net
Gas Turbine(s)	29075		9912		36,32	
Steam Turbine(s)	0					
Plant Total	29075	27199	9912	10595	36,32	33,98

GAS TURBINE PERFORMANCE - SIE SGT-A30 (Curve Fit OEM Data Model #421)					
	Gross power	Gross LHV	Gross LHV Heat Rate	Exh. flow	Exh. temp.
	output, kW	efficiency, %	kJ/kWh	t/h	C
per unit	29075	36,32	9912	345	506
Total	29075			345	

Number of gas turbine unit(s) =	1	
Gas turbine load [%] =	100	%
Fuel chemical HHV (77F/25C) per gas turbine =	88028	kW
Fuel chemical LHV (77F/25C) per gas turbine =	80049	kW

Plant Summary

ESTIMATED PLANT AUXILIARIES (kW)		
GT fuel compressor(s)*	1427,1	kW
GT supercharging fan(s)*	0	kW
GT electric chiller(s)*	0	kW
GT chiller/heater water pump(s)	0	kW
HRSG feedpump(s)*	0	kW
Condensate pump(s)*	0	kW
HRSG forced circulation pump(s)	0	kW
LTE recirculation pump(s)	0	kW
Cooling water pump(s)	0	kW
Air cooled condenser fans	0	kW
Cooling tower fans	0	kW
Dilution/Fresh air fan(s)*	0	kW
Aux. from PEACE running motor/load list	229,8	kW
Miscellaneous gas turbine auxiliaries	58,8	kW
Miscellaneous steam turbine auxiliaries	0	kW
Miscellaneous plant auxiliaries	14,54	kW
Constant plant auxiliary load	0	kW
Gasification plant, ASU*	0	kW
Gasification plant, fuel preparation	0	kW
Gasification plant, AGR*	0	kW
Gasification plant, other/misc	0	kW
Desalination plant auxiliaries	0	kW
Program estimated overall plant auxiliaries	1730,2	kW
Actual (user input) overall plant auxiliaries	1730,2	kW
Transformer losses	145,4	kW
Total auxiliaries & transformer losses	**1875,5**	**kW**
* Heat balance related auxiliaries		

Plant Summary

PLANT HEAT BALANCE		
Energy In	**89720**	**kW**
Ambient air sensible	1423,5	kW
Ambient air latent	0,2462	kW
Fuel enthalpy @ supply	88296	kW
External gas addition to combustor	0	kW
Steam and water	0	kW
Makeup and process return	0	kW
Energy Out	**90172**	**kW**
Net power output	27199	kW
Stack gas sensible	52146	kW
Stack gas latent	8173	kW
GT mechanical loss	298,8	kW
GT gear box loss	443,8	kW
GT generator loss	589,2	kW
GT miscellaneous losses	440,1	kW
GT ancillary heat rejected	219,5	kW
GT process air bleed	0	kW
Fuel compressor mech/elec loss	214,1	kW
Supercharging fan mech/elec loss	0	kW
Condenser	0	kW
Process steam	0	kW
Process water	0	kW
Blowdown/leakages	0	kW
Heat radiated from steam cycle	0	kW
ST/generator mech/elec/gear loss	0	kW
Non-heat balance related auxiliaries	303,1	kW
Transformer loss	145,4	kW
Energy In - Energy Out	-452,2	kW
GT heat balance error (arising from GT definitions)	-452,2	kW
Zero enthalpy: dry gases & liquid water @ 32 F (273.15 K)		

Macro Outputs

GT Cycle	Unit	Base Case	Case 1	Case 2	Case 3	Case 4	Case 5
Computation Result Messages		Messages	Messages	Messages	Messages	Messages	Messages
1. GT gross power	kW	31386	31386	31384	31341	31216	31092
2. GT gross LHV eff	%	36,89	36,89	36,86	36,83	36,8	36,76
3. GT gross heat rate	kJ/kWh	9758	9758	9766	9774	9783	9793
4. Compressor inlet mass flow	t/h	359,6	359,6	358,7	357,7	356,4	355
5. Compressor inlet temperature	C	0	0	1	2	3	4
6. Turbine inlet mass flow	t/h	N/A	N/A	N/A	N/A	N/A	N/A
7. Turbine inlet temperature	C	N/A	N/A	N/A	N/A	N/A	N/A
8. Turbine exhaust mass flow	t/h	366,2	366,2	365,3	364,3	362,9	361,5
9. Turbine exhaust temperature	C	488,8	488,8	491,2	493,3	494,7	496,1
10. GT fuel HHV chemical energy input (77F/25C)	kW	93557	93557	93619	93569	93288	93007
11. GT fuel LHV chemical energy input (77F/25C)	kW	85076	85076	85133	85087	84832	84576
12. Exhaust gas molecular weight		28,68	28,68	28,68	28,68	28,68	28,68
13. Exhaust gas N2 mole percentage	%	75,93	75,93	75,92	75,92	75,92	75,92
14. Exhaust gas O2 mole percentage	%	14,44	14,44	14,42	14,41	14,4	14,4
15. Exhaust gas CO2 mole percentage	%	3,279	3,279	3,289	3,296	3,299	3,301
16. Exhaust gas SO2 mole percentage	%	0	0	0	0	0	0
17. Exhaust gas H2O mole percentage	%	5,435	5,435	5,451	5,463	5,467	5,471
18. Exhaust gas Ar mole percentage	%	0,9144	0,9144	0,9143	0,9143	0,9142	0,9142
19. GT fuel flow	t/h	6,553	6,553	6,557	6,554	6,534	6,514
20. Combustor steam injection flow (per GT)	t/h	N/A	N/A	N/A	N/A	N/A	N/A
21. Combustor water injection flow (per GT)	t/h	N/A	N/A	N/A	N/A	N/A	N/A
22. Inlet filter pressure loss	millibar	2,457	2,457	2,464	2,47	2,473	2,475
23. Total exhaust pressure loss	millibar	13,72	13,72	13,7	13,66	13,58	13,5
24. Fogging water mass flow	t/h	N/A	N/A	N/A	N/A	N/A	N/A
25. Number of chillers in plant		0	0	0	0	0	0
26. Nameplate capacity at standard conditions	kW	N/A	N/A	N/A	N/A	N/A	N/A
27. Nameplate COP at standard conditions		N/A	N/A	N/A	N/A	N/A	N/A
28. Chiller load at current heat balance conditions	kW	N/A	N/A	N/A	N/A	N/A	N/A
29. COP at current heat balance conditions		N/A	N/A	N/A	N/A	N/A	N/A
30. Chilled water temperature	C	N/A	N/A	N/A	N/A	N/A	N/A
31. Chilled water range	C	N/A	N/A	N/A	N/A	N/A	N/A
32. Chilled water mass flow @ coil	t/h	0	0	0	0	0	0
33. Desired air temperature drop	C	N/A	N/A	N/A	N/A	N/A	N/A
34. Actual air temperature drop	C	N/A	N/A	N/A	N/A	N/A	N/A
35. Coolant temperature	C	N/A	N/A	N/A	N/A	N/A	N/A
36. Chilled water temperature leaving chiller	C	N/A	N/A	N/A	N/A	N/A	N/A
37. Chilled water mass flow leaving chiller (per GT)	t/h	N/A	N/A	N/A	N/A	N/A	N/A
38. Net chilled water flow to storage tank (plant total)	t/h	0	0	0	0	0	0

Macro Outputs

GT Cycle	Unit	Base Case	Case 6	Case 7	Case 8	Case 9	Case 10
Computation Result Messages		Messages	Messages	Messages	Messages	Messages	Messages
1. GT gross power	kW	31386	30968	30844	30721	30530	30320
2. GT gross LHV eff	%	36,89	36,73	36,69	36,66	36,62	36,57
3. GT gross heat rate	kJ/kWh	9758	9802	9812	9821	9832	9843
4. Compressor inlet mass flow	t/h	359,6	353,7	352,3	351	349,4	347,9
5. Compressor inlet temperature	C	0	5	6	7	8	9
6. Turbine inlet mass flow	t/h	N/A	N/A	N/A	N/A	N/A	N/A
7. Turbine inlet temperature	C	N/A	N/A	N/A	N/A	N/A	N/A
8. Turbine exhaust mass flow	t/h	366,2	360,2	358,8	357,4	355,9	354,2
9. Turbine exhaust temperature	C	488,8	497,4	498,8	500,2	501	501,7
10. GT fuel HHV chemical energy input (77F/25C)	kW	93557	92725	92444	92162	91690	91164
11. GT fuel LHV chemical energy input (77F/25C)	kW	85076	84320	84064	83808	83379	82900
12. Exhaust gas molecular weight		28,68	28,68	28,68	28,68	28,68	28,68
13. Exhaust gas N2 mole percentage	%	75,93	75,91	75,91	75,91	75,91	75,91
14. Exhaust gas O2 mole percentage	%	14,44	14,39	14,39	14,38	14,39	14,4
15. Exhaust gas CO2 mole percentage	%	3,279	3,304	3,306	3,309	3,306	3,302
16. Exhaust gas SO2 mole percentage	%	0	0	0	0	0	0
17. Exhaust gas H2O mole percentage	%	5,435	5,476	5,48	5,484	5,48	5,473
18. Exhaust gas Ar mole percentage	%	0,9144	0,9142	0,9142	0,9142	0,9142	0,9142
19. GT fuel flow	t/h	6,553	6,495	6,475	6,455	6,422	6,385
20. Combustor steam injection flow (per GT)	t/h	N/A	N/A	N/A	N/A	N/A	N/A
21. Combustor water injection flow (per GT)	t/h	N/A	N/A	N/A	N/A	N/A	N/A
22. Inlet filter pressure loss	millibar	2,457	2,478	2,481	2,483	2,485	2,485
23. Total exhaust pressure loss	millibar	13,72	13,43	13,35	13,27	13,17	13,06
24. Fogging water mass flow	t/h	N/A	N/A	N/A	N/A	N/A	N/A
25. Number of chillers in plant		0	0	0	0	0	0
26. Nameplate capacity at standard conditions	kW	N/A	N/A	N/A	N/A	N/A	N/A
27. Nameplate COP at standard conditions		N/A	N/A	N/A	N/A	N/A	N/A
28. Chiller load at current heat balance conditions	kW	N/A	N/A	N/A	N/A	N/A	N/A
29. COP at current heat balance conditions		N/A	N/A	N/A	N/A	N/A	N/A
30. Chilled water temperature	C	N/A	N/A	N/A	N/A	N/A	N/A
31. Chilled water range	C	N/A	N/A	N/A	N/A	N/A	N/A
32. Chilled water mass flow @ coil	t/h	0	0	0	0	0	0
33. Desired air temperature drop	C	N/A	N/A	N/A	N/A	N/A	N/A
34. Actual air temperature drop	C	N/A	N/A	N/A	N/A	N/A	N/A
35. Coolant temperature	C	N/A	N/A	N/A	N/A	N/A	N/A
36. Chilled water temperature leaving chiller	C	N/A	N/A	N/A	N/A	N/A	N/A
37. Chilled water mass flow leaving chiller (per GT)	t/h	N/A	N/A	N/A	N/A	N/A	N/A
38. Net chilled water flow to storage tank (plant total)	t/h	0	0	0	0	0	0

Macro Outputs

GT Cycle	Unit	Base Case	Case 11	Case 12	Case 13	Case 14	Case 15
Computation Result Messages		Messages	Messages	Messages	Messages	Messages	Messages
1. GT gross power	kW	31386	30111	29901	29692	29483	29279
2. GT gross LHV eff	%	36,89	36,53	36,49	36,45	36,41	36,37
3. GT gross heat rate	kJ/kWh	9758	9854	9865	9876	9888	9900
4. Compressor inlet mass flow	t/h	359,6	346,3	344,7	343,1	341,6	340,2
5. Compressor inlet temperature	C	0	10	11	12	13	14
6. Turbine inlet mass flow	t/h	N/A	N/A	N/A	N/A	N/A	N/A
7. Turbine inlet temperature	C	N/A	N/A	N/A	N/A	N/A	N/A
8. Turbine exhaust mass flow	t/h	366,2	352,6	351	349,4	347,8	346,4
9. Turbine exhaust temperature	C	488,8	502,3	502,9	503,6	504,3	505,1
10. GT fuel HHV chemical energy input (77F/25C)	kW	93557	90636	90106	89575	89048	88539
11. GT fuel LHV chemical energy input (77F/25C)	kW	85076	82420	81939	81456	80977	80513
12. Exhaust gas molecular weight		28,68	28,68	28,68	28,68	28,68	28,68
13. Exhaust gas N2 mole percentage	%	75,93	75,92	75,92	75,92	75,93	75,93
14. Exhaust gas O2 mole percentage	%	14,44	14,4	14,41	14,42	14,43	14,44
15. Exhaust gas CO2 mole percentage	%	3,279	3,298	3,294	3,29	3,286	3,28
16. Exhaust gas SO2 mole percentage	%	0	0	0	0	0	0
17. Exhaust gas H2O mole percentage	%	5,435	5,467	5,46	5,453	5,445	5,436
18. Exhaust gas Ar mole percentage	%	0,9144	0,9142	0,9143	0,9143	0,9143	0,9144
19. GT fuel flow	t/h	6,553	6,348	6,311	6,274	6,237	6,202
20. Combustor steam injection flow (per GT)	t/h	N/A	N/A	N/A	N/A	N/A	N/A
21. Combustor water injection flow (per GT)	t/h	N/A	N/A	N/A	N/A	N/A	N/A
22. Inlet filter pressure loss	millibar	2,457	2,486	2,486	2,486	2,487	2,489
23. Total exhaust pressure loss	millibar	13,72	12,95	12,85	12,74	12,63	12,54
24. Fogging water mass flow	t/h	N/A	N/A	N/A	N/A	N/A	N/A
25. Number of chillers in plant		0	0	0	0	0	0
26. Nameplate capacity at standard conditions	kW	N/A	N/A	N/A	N/A	N/A	N/A
27. Nameplate COP at standard conditions		N/A	N/A	N/A	N/A	N/A	N/A
28. Chiller load at current heat balance conditions	kW	N/A	N/A	N/A	N/A	N/A	N/A
29. COP at current heat balance conditions		N/A	N/A	N/A	N/A	N/A	N/A
30. Chilled water temperature	C	N/A	N/A	N/A	N/A	N/A	N/A
31. Chilled water range	C	N/A	N/A	N/A	N/A	N/A	N/A
32. Chilled water mass flow @ coil	t/h	0	0	0	0	0	0
33. Desired air temperature drop	C	N/A	N/A	N/A	N/A	N/A	N/A
34. Actual air temperature drop	C	N/A	N/A	N/A	N/A	N/A	N/A
35. Coolant temperature	C	N/A	N/A	N/A	N/A	N/A	N/A
36. Chilled water temperature leaving chiller	C	N/A	N/A	N/A	N/A	N/A	N/A
37. Chilled water mass flow leaving chiller (per GT)	t/h	N/A	N/A	N/A	N/A	N/A	N/A
38. Net chilled water flow to storage tank (plant total)	t/h	0	0	0	0	0	0

Macro Outputs

GT Cycle	Unit	Base Case	Case 16	Case 17	Case 18	Case 19	Case 20
Computation Result Messages		Messages	Messages	Messages	Messages	Messages	Messages
1. GT gross power	kW	31386	29075	28878	28680	28483	28291
2. GT gross LHV eff	%	36,89	36,32	36,28	36,24	36,2	36,15
3. GT gross heat rate	kJ/kWh	9758	9912	9923	9935	9946	9958
4. Compressor inlet mass flow	t/h	359,6	338,8	337,3	335,7	334,1	332,6
5. Compressor inlet temperature	C	0	15	16	17	18	19
6. Turbine inlet mass flow	t/h	N/A	N/A	N/A	N/A	N/A	N/A
7. Turbine inlet temperature	C	N/A	N/A	N/A	N/A	N/A	N/A
8. Turbine exhaust mass flow	t/h	366,2	345	343,4	341,8	340,2	338,6
9. Turbine exhaust temperature	C	488,8	506	506,5	507,1	507,6	508,3
10. GT fuel HHV chemical energy input (77F/25C)	kW	93557	88028	87533	87035	86537	86057
11. GT fuel LHV chemical energy input (77F/25C)	kW	85076	80049	79598	79146	78693	78256
12. Exhaust gas molecular weight		28,68	28,68	28,68	28,68	28,68	28,68
13. Exhaust gas N2 mole percentage	%	75,93	75,93	75,93	75,94	75,94	75,94
14. Exhaust gas O2 mole percentage	%	14,44	14,45	14,46	14,46	14,47	14,48
15. Exhaust gas CO2 mole percentage	%	3,279	3,275	3,272	3,268	3,265	3,262
16. Exhaust gas SO2 mole percentage	%	0	0	0	0	0	0
17. Exhaust gas H2O mole percentage	%	5,435	5,427	5,422	5,417	5,411	5,405
18. Exhaust gas Ar mole percentage	%	0,9144	0,9144	0,9144	0,9145	0,9145	0,9145
19. GT fuel flow	t/h	6,553	6,166	6,131	6,096	6,061	6,028
20. Combustor steam injection flow (per GT)	t/h	N/A	N/A	N/A	N/A	N/A	N/A
21. Combustor water injection flow (per GT)	t/h	N/A	N/A	N/A	N/A	N/A	N/A
22. Inlet filter pressure loss	millibar	2,457	2,49	2,491	2,491	2,491	2,491
23. Total exhaust pressure loss	millibar	13,72	12,45	12,35	12,24	12,14	12,04
24. Fogging water mass flow	t/h	N/A	N/A	N/A	N/A	N/A	N/A
25. Number of chillers in plant		0	0	0	0	0	0
26. Nameplate capacity at standard conditions	kW	N/A	N/A	N/A	N/A	N/A	N/A
27. Nameplate COP at standard conditions		N/A	N/A	N/A	N/A	N/A	N/A
28. Chiller load at current heat balance conditions	kW	N/A	N/A	N/A	N/A	N/A	N/A
29. COP at current heat balance conditions		N/A	N/A	N/A	N/A	N/A	N/A
30. Chilled water temperature	C	N/A	N/A	N/A	N/A	N/A	N/A
31. Chilled water range	C	N/A	N/A	N/A	N/A	N/A	N/A
32. Chilled water mass flow @ coil	t/h	0	0	0	0	0	0
33. Desired air temperature drop	C	N/A	N/A	N/A	N/A	N/A	N/A
34. Actual air temperature drop	C	N/A	N/A	N/A	N/A	N/A	N/A
35. Coolant temperature	C	N/A	N/A	N/A	N/A	N/A	N/A
36. Chilled water temperature leaving chiller	C	N/A	N/A	N/A	N/A	N/A	N/A
37. Chilled water mass flow leaving chiller (per GT)	t/h	N/A	N/A	N/A	N/A	N/A	N/A
38. Net chilled water flow to storage tank (plant total)	t/h	0	0	0	0	0	0

Macro Outputs

GT Cycle	Unit	Base Case	Case 21
Computation Result Messages		Messages	Messages
1. GT gross power	kW	31386	28101
2. GT gross LHV eff	%	36,89	36,11
3. GT gross heat rate	kJ/kWh	9758	9970
4. Compressor inlet mass flow	t/h	359,6	331,2
5. Compressor inlet temperature	C	0	20
6. Turbine inlet mass flow	t/h	N/A	N/A
7. Turbine inlet temperature	C	N/A	N/A
8. Turbine exhaust mass flow	t/h	366,2	337,2
9. Turbine exhaust temperature	C	488,8	508,9
10. GT fuel HHV chemical energy input (77F/25C)	kW	93557	85585
11. GT fuel LHV chemical energy input (77F/25C)	kW	85076	77827
12. Exhaust gas molecular weight		28,68	28,68
13. Exhaust gas N2 mole percentage	%	75,93	75,94
14. Exhaust gas O2 mole percentage	%	14,44	14,48
15. Exhaust gas CO2 mole percentage	%	3,279	3,258
16. Exhaust gas SO2 mole percentage	%	0	0
17. Exhaust gas H2O mole percentage	%	5,435	5,399
18. Exhaust gas Ar mole percentage	%	0,9144	0,9146
19. GT fuel flow	t/h	6,553	5,995
20. Combustor steam injection flow (per GT)	t/h	N/A	N/A
21. Combustor water injection flow (per GT)	t/h	N/A	N/A
22. Inlet filter pressure loss	millibar	2,457	2,492
23. Total exhaust pressure loss	millibar	13,72	11,94
24. Fogging water mass flow	t/h	N/A	N/A
25. Number of chillers in plant		0	0
26. Nameplate capacity at standard conditions	kW	N/A	N/A
27. Nameplate COP at standard conditions		N/A	N/A
28. Chiller load at current heat balance conditions	kW	N/A	N/A
29. COP at current heat balance conditions		N/A	N/A
30. Chilled water temperature	C	N/A	N/A
31. Chilled water range	C	N/A	N/A
32. Chilled water mass flow @ coil	t/h	0	0
33. Desired air temperature drop	C	N/A	N/A
34. Actual air temperature drop	C	N/A	N/A
35. Coolant temperature	C	N/A	N/A
36. Chilled water temperature leaving chiller	C	N/A	N/A
37. Chilled water mass flow leaving chiller (per GT)	t/h	N/A	N/A
38. Net chilled water flow to storage tank (plant total)	t/h	0	0