Maria Giosuè

Project and Design of a Digital Twin for Gas Turbines for Power Generation in the Offshore Field

Master's thesis in Gas Technology
Supervisor: Even Solbraa
October 2023

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

NTNU
Norwegian University of
Science and Technology

ABSTRACT

In the global energy market, there is an increasing pressure on industries to digitise and develop efficient methods of simulating and controlling plants and components, one of the many reasons being the prediction that energy consumption will increase by 50% by 2050.
To this end, one of the most cutting-edge technologies to emerge in recent years is the Digital Twin. This thesis describes the modelling and development project of a digital twin for gas turbines owned by Equinor, used to produce the energy required to operate offshore plants.
The aim of the project is therefore to create a digital twin in Python to simulate and predict the behaviour of two gas turbines for offshore power generation, with the aim of implementing this model for predictive maintenance purposes. In the first phase of the project, the code for the computational model of the components of a gas turbine is developed using object-oriented programming in Python, designing specific methods and functions for the detailed description of the thermodynamic behaviour of the compressor, combustion chamber, turbine and air and fuel flows, using the NeqSim library for the determination of fluid properties. Subsequently, the computational model is validated through a comparative analysis with the Aspen HYSYS® software, developing a single-shaft gas turbine case study, under different operating conditions.
Following the validation of the prediction and calculation methods for a generic gas turbine, the digital twin modeling is carried out under design conditions for the two real turbines under study: GE LM2500 and GE LM6000, through the use of Thermoflow©, which provides the main inputs to build the model in Python and the outputs to validate the digital twin itself, and GasTurb© to determine the polytropic and isoentropic efficiencies of the components in design conditions. Next, a model for the off-design behaviour at varying ambient temperature of the GE LM2500 turbine is developed, using the off-design model outlined in Thermoflow© as a reference.
Finally, since one of the most important and significant applications of digital twins is predictive maintenance, a brief analysis of the indicator for detecting the degradation of gas turbine components is presented, with an initial and illustrative comparison of the results with field data, provided by Equinor.
The results show that the digital twin developed in Python under design conditions for both turbines, and under off-design conditions for the GE LM2500 turbine, produce outputs that deviate from Thermoflow© by less than 1%, while also providing detailed data for individual components, such as temperatures and power required and generated at each turbine stage, fluid composition, etc.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

- **CBM** Class Based Maintenance
- **CFD** Computational Fluid Dynamics
- **CM** Condition Monitoring
- **D** Design
- **FDI** Fault Detection and Identification
- **GE** General Electric
- **GPA** Gas Path Analysis
- **GT** Gas Turbine
- **HP** High Pressure
- **HPC** High Pressure Compressor
- **HPT** High Pressure Turbine
- **IGV** Inlet Guide Vane
- **LCV** Lower Calorific Value
- **LHV** Lower Heating Value
- **LP** Low Pressure
- **LPC** Low Pressure Compressor
- **LPT** Low Pressure Turbine
- **MAPE** Mean Absolute Percentage Error
- **OD** Off Design
- **RMSE** Root Mean Sqared Error
- **TET** Turbine Exhaust Temperature
- **TIT** Turbine Inlet Temperature

- **VIGV** Variable Inlet Guide Vane
- **NGV** Noozle Guide Vane

CHAPTER

ONE

INTRODUCTION

1.1 Motivation

There is an increasing pressure on global energy industries to develop efficient operating and plant control methods to reduce the impact of fluctuating conditions such as variable weather, fuel changes or component degradation [1].
The need for digitisation of the energy and capital intensive sector can be found in several reasons, including the increasing energy consumption, which is projected to increment by 50% before 2050 [2]. For these purposes, one of the most cutting-edge technologies that has been developing more and more in recent years, is the Digital Twin.

Digital twin is defined by the CIRP Encyclopedia of Production Engineering [3] as a "digital representation of a machine, device, service, object, asset or product–service system that tracks the characteristics, properties, conditions, and behaviors of the system by means of models, information, and data". In particular, a Digital Twin for power plants in the energy industry can be defined as a combined physics based and analytical methods used to model the individual components and the plant as a whole: these models can be applied to new and existing plants to simulate and predict component and system behaviour under different operating conditions. The output of Digital Twin, combined with other prediction and control tools, can thus improve the power plant performance, reliability, availability, maintainability and flexible operation, and be an important tool to support real time decisions [1].

In particular, one of the most significant uses of the digital twin is in relation to maintenance and particularly predictive maintenance with the advent of Industry 4.0. Predictive maintenance can maximise the reliability and the machine in-service time by monitoring the actual condition, and predicting the future behaviour; in this sense, condition monitoring (CM) has played an increasingly significant role in supporting predictive maintenance by estimating the current and future condition of the monitored machine [4].

Gas turbines are an element in the energy sector, as they are essential for power generation in various sectors, including industry; however, long-term service can easily result in performance faults, such as fouling, erosion, corrosion, abrasion and damage, leading to economic losses and potential safety hazard [5]. The implementation of a digital twin for the purpose of predictive maintenance and condition monitoring for gas turbines would therefore lead to an improvement in operating conditions as well as a reduction in safety and economic risk, reasons that paved the way for the development of the project described in this work.

1.2 Project description

In this section, we delve into the core of this research endeavor, offering a detailed narrative that encompasses the research question, methodology and the overarching relevance of this study. The first part of this paper deals with an overview of the theoretical background, which is fundamental for the development of the model: an initial overview of the thermodynamics of gas turbines is followed by a detailed description of the two real turbines under study and a general analysis of the state of the art in the application of Condition Monitoring in the energy sector and, in particular, in the implementation of Digital Twin models for gas turbines.

Next, an explanation of the methods and models implemented for the design of the Digital Twin follows, starting with a detailed description of the class-based code in Python for defining the calculation functions for the main components of the turbine: compressor, turbine, combustion chamber and streams. In order to evaluate the quality of the code designed in Python, a case study of a single-shaft gas turbine is implemented, and the results in terms of temperature, power, efficiency, and mole fractions of components are compared with the same case study developed in Aspen HYSYS®, a process simulation software known in the energy industry, and used for optimization in design and operations.

After designing and validating the source code for thermodynamic modelling of a generic gas turbine, the digital twin for the two real turbines under study is carried out: the GE (*General Electric*) LM2500 and the GE LM6000, through the use of Thermoflow© and GasTurb© software. For the digital twin at design conditions, some data from GT PRO, software included in Thermoflow© which provides design points calculations for combined plants and gas turbines, are used as input values, while others are used to check the output from the Python code. In particular, the data taken as input are: inlet air flow rate, pressure drops at the intake and at the exhaust, fuel flow rate, air losses, nominal compressor pressure ratio and ambient temperature; on the other hand, the data taken as reference values for comparison are: exhaust gas temperature, power, heat rate and gas turbine efficiency.

To complete the modelling of the design point, it is furthermore necessary to use GasTurb© for the identification of the isentropic and polytropic efficiencies of the gas turbine components (compressor and turbine), as these are crucial elements not only for the design of the model itself, but also for the evaluation of the degradation for condition monitoring and predictive maintenance purposes: to do this, an iterative procedure is implemented which, by identifying the values of exhaust gas temperature, power, etc. as targets, finds out the values of polytropic and isentropic efficiencies as variables. These values are used as inputs in the Python model together with the inputs from Thermoflow©.

Through the simulation of the off-design model by changing the ambient temperatures in GT MASTER, a further extension of the Thermoflow© software, a digital off-design model is engineered for the gas turbine GE LM2500, since predicting the behaviour of the gas turbine under operating conditions different than design conditions is important for the design of a digital twin as close to reality as possible. As it is done for the design model, the off-design model of the GE LM2500 gas turbine takes as inputs some values from Thermoflow©, including air flow rate, fuel flow rate, pressure losses at the intake and at the exhaust, air losses through the compressor, while some others are used to compare the results with Python, such as power, gas turbine efficiency, heat rate and exhaust gas temperature. Finally, a brief and illustrative analysis of the field data provided by Equinor is presented in order to sketch an initial predictive maintenance test by using indices to identify component degradation.

CHAPTER

TWO

THEORETICAL BACKGROUND

The main objective of this chapter is to illustrate the thermodynamic models and theoretical principles underlying the construction of the Digital Twin. The first part of the chapter will discuss the thermodynamic (and partly mechanical) models describing the behaviour of gas turbines. The second part of the chapter will provide a description of the reference turbines under study: General Electric LM6000 and General Electric LM2500. Finally, a discussion will be given on condition monitoring in the context of gas turbines and how it could be applied in the specific case study, the main use for which the Digital Twin is designed.

2.1 Gas turbine thermodynamics

Before getting to the heart of the modeling of the Digital Twin, it is important to discuss the thermodynamic principles underlying a gas turbine, which can be a very complex system. The thermodynamic cycle describing a gas turbine in ideal conditions is the ideal Brayton cycle, depicted in Figure 2.1.1. The four operations describing the Brayton cycle are:

- Operation 1-2: isentropic compression of air from a lower pressure P_1 to an upper pressure P_2, with a rise of the temperature from T_1 to T_2.
- Operation 2-3: heat flowing into the system, leading to an increase of the volume from V_2 to V_3 and an increase of the temperature from T_2 to T_3, whilst the pressure remains constant.
- Operation 3-4: isentropic expansion of air from P_3 to P_4, leading to a decrease of the temperature from T_3 to T_4.
- Operation 4-1 heat flowing out of the system, leading to a reduction in temperature and volume, from V_4 to V_1 and from T_4 to T_1.

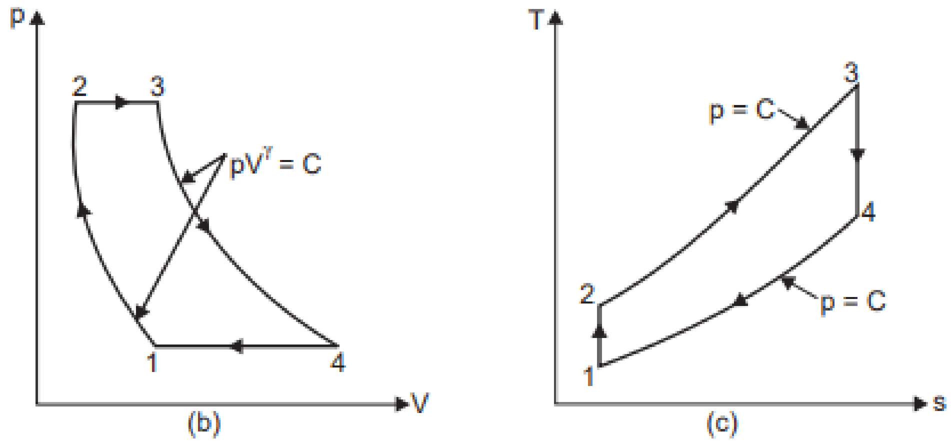


Figure 2.1.1: Ideal Brayton cycle: p-V diagram (left) and T-s diagram (right)

Compression and expansion processes are reversible and adiabatic, i.e. isentropic [6], which means that:

- Reversible: it does not leave traces of energy in the surroundings, so that it can be carried out with infinitesimally small changes and restored to the original states.
- Adiabatic: there is no heat transfer between the system and the surroundings.
- Isentropic: there is no change in entropy, which implies that is reversible and adiabatic.

The equations used for describing the isentropic process are [7]:

$$\frac{T_2}{T_1} = \left(\frac{p_2}{p_1}\right)^{\frac{\gamma-1}{\gamma}} \tag{2.1}$$

and similarly:

$$\frac{T_3}{T_4} = \left(\frac{p_3}{p_4}\right)^{\frac{\gamma-1}{\gamma}} \tag{2.2}$$

where γ is equal to:

$$\gamma = \frac{C_p}{C_v} \tag{2.3}$$

Referring to the steady flow energy equation [6]:

$$Q = (h_2 - h_1) + \frac{1}{2}\left(C_2^2 - C_1^2\right) + W \tag{2.4}$$

neglecting the change in kinetic energy and assuming constant specific heat capacity, the heat and work transfer for unit mass flow can be calculated as follows:

$$W_{12} = -\left(h_2 - h_1\right) = -c_p\left(T_2 - T_1\right) \tag{2.5}$$

$$Q_{23} = (h_3 - h_2) = c_p\,(T_3 - T_2) \tag{2.6}$$

$$W_{34} = (h_3 - h_4) = c_p\,(T_3 - T_4) \tag{2.7}$$

A real gas turbine, however, is an open cycle system composed of a rotatory compressor and a turbine mounted on the same shaft. Air flows into the compressor, through which is compressed to the combustion chamber, to be combined with the fuel leading to a combustion process. The hot gases at high temperature and high pressure (which means high enthalpy and high kinetic energy due to the velocity of the particles) then get into the turbine, in order to expand and transform the thermal energy into mechanical energy that drives the shaft. The exhaust is then released to the atmosphere. A representation of the simple cycle gas turbine is Figure 2.1.2

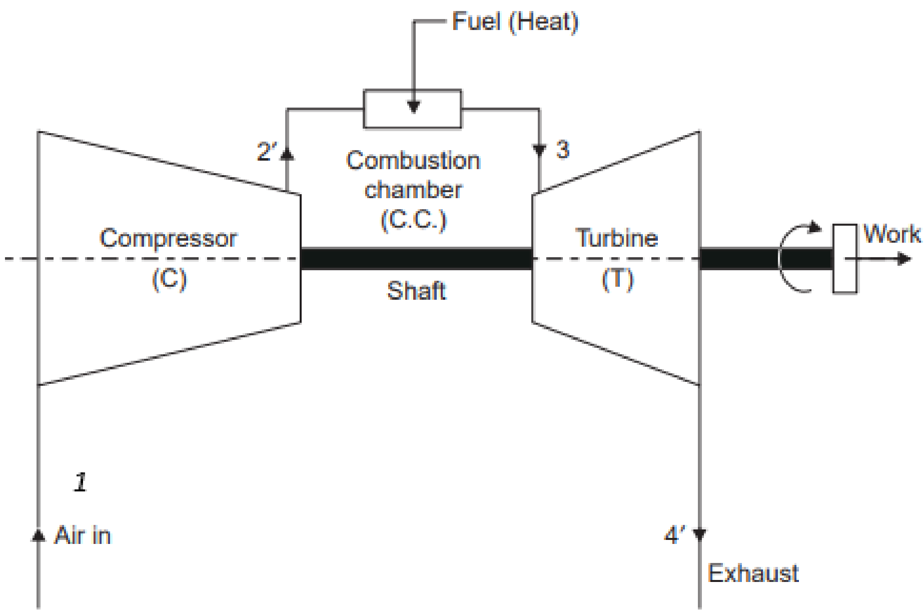


Figure 2.1.2: Simple cycle gas turbine representation: the four states are labeled 1-4 where the prime over states 2 and 4 indicates real rather than ideal conditions as also displayed in Figure 2.1.3

In particular, the real Brayton cycle is the one represented in Figure 2.1.3, and as it can be seen the compression and expansion processes can not be considered reversible and adiabatic due to a multiple reasons, one of which the leakage into the surroundings of heat, friction losses, or pressure drops, that lead to the irreversibility of the system.

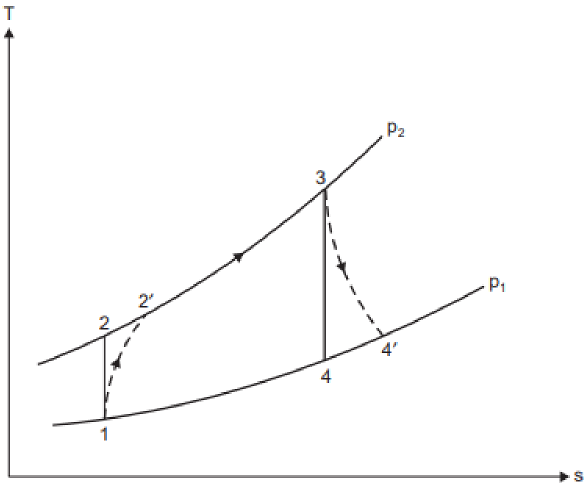


Figure 2.1.3: Brayton real cycle T-s diagram, states numbered in accordance with Figure 2.1.2

In real compression and expansion processes, an important factor to be noted is the increase of system entropy: the entropy is a measure of the degree of disorder in a system, and according to the second law of thermodynamics the total entropy of a system tend to increase or at least to remain constant over time. The real values for the temperatures are so T_2' and T_4'.

Since the isentropic efficiency can be defined as the ratio of the ideal work to the actual work for the compressor and the ratio of the real work to the actual work for the turbine, and referring to 2.5 and to 2.7, the efficiency for the compressor can be formulated as follows:

$$\eta_{is,C} = \frac{(T_2 - T_1)}{(T_2' - T_1)} \tag{2.8}$$

and for the turbine:

$$\eta_{is,T} = \frac{(T_3 - T_4')}{(T_3 - T_4)} \tag{2.9}$$

Furthermore, in order to calculate the real temperatures in the model, having the isentropic efficiency, the followed equations are used:

$$T_2' = \frac{(T_2 - T_1)}{\eta_{ise,C}} + T_1 \tag{2.10}$$

$$T_4' = -\eta_{ise,T}\,(T_3 - T_4) + T_3 \tag{2.11}$$

The real work for compressor and turbine, thus, can be calculated based on the equation 2.5 and the equation 2.7:

$$W_{real,C} = -c_p \dot{m}_{air}\,(T_2' - T_1) \tag{2.12}$$

$$W_{real,T} = c_p\,(\dot{m}_{air} + \dot{m}_{fuel})\,(T_3 - T_4') \tag{2.13}$$

It is crucial to note that in real gas compression and expansion, the specific heat c_p varies with the change of the temperature, and also in an open cycle the specific heat of the gases in the combustion chamber and in the turbine is different from that in the compressor, because of the fuel and because a chemical reaction takes place [7]. In order to design a model that is as close to reality as possible, that simulates the behaviour of a real gas rather than an ideal gas, the variation of specific heat as temperature changes id taken into account in the modeling phase in Python, and reference is made to sections 3.1.2 and 3.1.4.

Regarding the calculation for the heat of combustion, since a chemical reaction between air and fuel is involved, the difference in enthalpy is decided to be used, as shown in the equation 2.6: the change of the specific heat according to the change of temperature, in fact, is more hard to detect due to the combustion process, and furthermore the difference in enthalpy involves the change in enthalpy between the reactants and the products, taking into account the energy released or absorbed, providing an overall energy change of the system, way more precise than the change in temperature with a constant c_p.

2.1.1 Relation between polytropic and isentropic efficiency

Polytropic efficiency is a measure of how a process is efficient compared to an ideal polytropic process, and it is an important concept of efficiency often used when modeling a gas turbine. In particular, it is referred as small or infinitesimal stage of efficiency, and it is exclusive of the pressure-ratio effect [8]. The polytropic efficiency for a compressor can be expressed as it follows [8]:

$$\eta_{p,C} = \frac{\left[1 + \frac{dP_2}{dP_1}\right]^{\frac{\gamma-1}{\gamma}} - 1}{\left[1 + \frac{dP_2}{dP_1}\right]^{\frac{n-1}{n}} - 1} \tag{2.14}$$

and expanding numerator and denominator using a Taylor expansion series, considering $dP_2/dP_1 << 1$:

$$\left[1 + \frac{dP_2}{dP_1}\right]^{\frac{\gamma-1}{\gamma}} = \left[1 + \frac{\gamma - 1}{\gamma}\right]\left(\frac{dP_2}{dP_1}\right) + \left[1 + \frac{\gamma - 1}{\gamma}\right]\left(\frac{dP_2}{dP_1}\right)^2 + ... \tag{2.15}$$

$$\left[1 + \frac{dP_2}{dP_1}\right]^{\frac{n-1}{n}} = \left[1 + \frac{n - 1}{n}\right]\left(\frac{dP_2}{dP_1}\right) + \left[1 + \frac{n - 1}{n}\right]\left(\frac{dP_2}{dP_1}\right)^2 + ... \tag{2.16}$$

the polytropic efficiency for the compressor results as:

$$\eta_{p,C} = \frac{\frac{\gamma-1}{\gamma}}{\frac{n-1}{n}} \tag{2.17}$$

As it can be seen, the polytropic efficiency for a compressor is the limiting value for the isentropic efficiency, as the increase of pressure approaches to zero [8].
The relation between polytropic and isentropic efficiency in the compressor and in the turbine is defined in the equations 2.18 and 2.19 [9]:

$$\eta_{is,C} = \frac{\left(\frac{P_2}{P_1}\right)^{\left(\frac{\gamma-1}{\gamma}\right)} - 1}{\left(\frac{P_2}{P_1}\right)^{\left(\frac{\gamma-1}{\gamma\eta_{p,C}}\right)} - 1} \tag{2.18}$$

$$\eta_{is,T} = \frac{\left(1 - \frac{P_4}{P_3}\right)^{\left(\frac{\gamma-1}{\gamma}\eta_{p,T}\right)}}{\left(1 - \frac{P_4}{P_3}\right)^{\left(\frac{\gamma-1}{\gamma}\right)}} \tag{2.19}$$

In particular, the Figure 2.1.4 represents what is explained above: varying the pressure ratio of the compressor from a value of 2 to a value of 20 and keeping constant the polytropic efficiency, applying the equation 2.18 it can be seen how the isentropic efficiency decreases: for a compressor, thus, the polytropic efficiency is the upper value for the isentropic efficiency, as the pressure ratio is reduced, q.e.d.

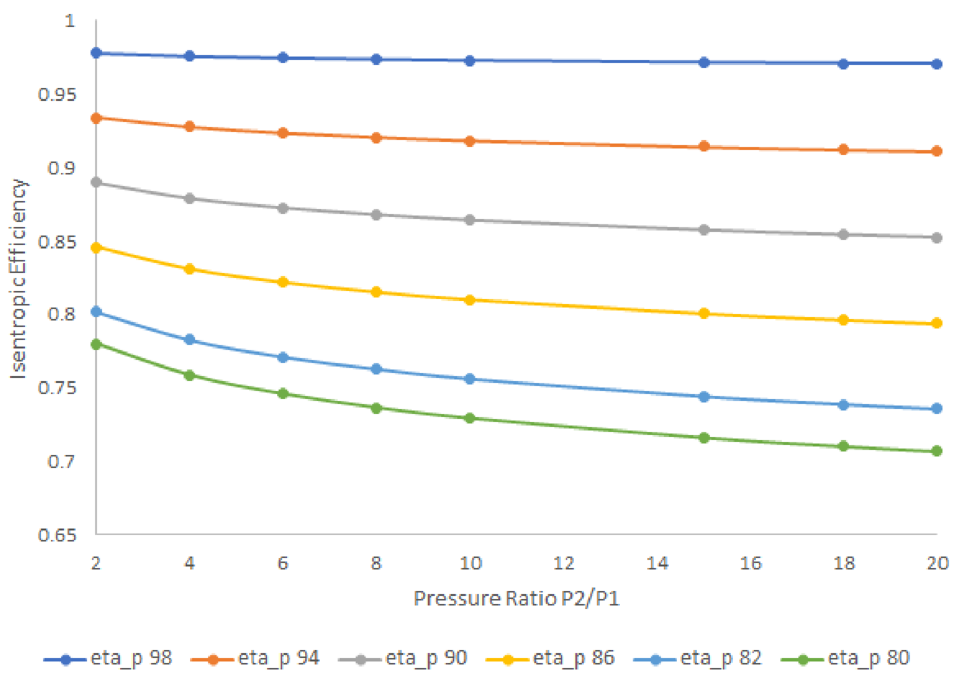


Figure 2.1.4: Relation between isentropic and polytropic efficiency in a compressor by changing the pressure ratio

Similar considerations can be done for the relation between isentropic and polytropic efficiency in a turbine: the polytropic efficiency, in this case, is the lower limit for the isentropic efficiency. Therefore, from the 2.18 and 2.19 equations relating isentropic and polytropic efficiencies for compressor and turbine, the equations for calculating the temperature out of the compressor and out of the turbine are [9]:

$$\frac{T_2}{T_1} = \left(\frac{P_2}{P_1}\right)^{\frac{\gamma-1}{\gamma\eta_{p,C}}} \tag{2.20}$$

$$\frac{T_4}{T_3} = \left(\frac{P_4}{P_3}\right)^{\left(\frac{\gamma-1}{\gamma}\eta_{p,T}\right)} \tag{2.21}$$

2.1.2 Off-design model

The design point for a gas turbine is the operating point for which the pressure ratio, air and fuel mass flow, and component efficiencies are modeled in order to achieve the desired power and maximum efficiency. When the operating conditions change, however, the performance of the turbine also changes, such as the efficiency of the components, or the power generated: therefore, the *off-design* model for a single shaft gas turbine has to be modeled. For example, the *part-load performance* refers to the fuel flow rate decrease in order to decrease the change of the power generated. Important changes in gas turbine performance, furthermore, can be detected with the change of the ambient conditions, especially the ambient temperature.

In general, the changes in pressure ratio, efficiency and flow rate are to be studied while building an off-design model, and these variations can be detected in compressor and turbine maps[6]. First of all, a brief digression about compressor and turbine characteristic maps must be done. In order to fully understand the meaning of the compressor map components, it is needed to first refer to the Figure 2.1.5, from [6].

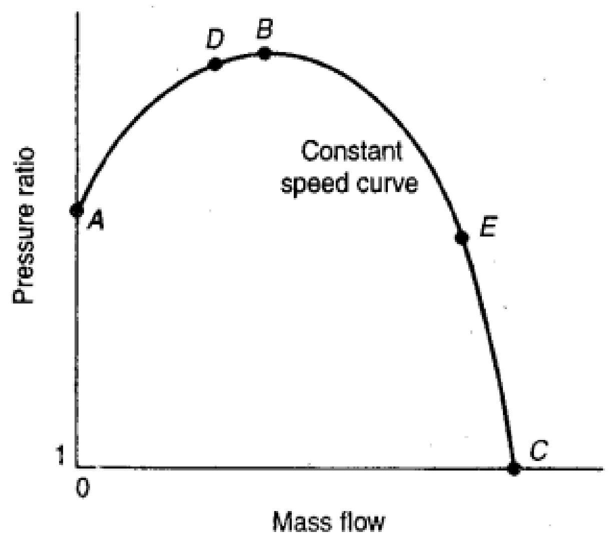


Figure 2.1.5: Constant speed line in a compressor map

In the x-axis there is the flow rate of air, while in the y-axis there is the pressure ratio across the compressor. The curve is a curve of constant speed of the shaft, and the different points marked show different operating points achievable by assuming that the compressor valve is opened slowly (and thus varying the flow rate of air that can be blown into the compressor).

At A the valve is completely closed, and consequently the air flow rate is zero: the pressure ratio obtained is given by the compression of some amount of air being trapped inside the vanes. Slowly opening the vanes, point B is reached, which represents the maximum efficiency achievable as well as the maximum pressure ratio. By going to increase the input air flow rate, this will have a negative effect on the compression ratio and efficiency, which will suddenly drop: at point C, in fact, the valve is fully opened, and the loss of efficiency indicates the loss of power that is spent to overcome the frictional forces with the air.

Another consideration may be done regarding the phenomena of the *surging*, that may happen between A and B, so that operating between these two points, even if it brings an high value of efficiency, it is not possible. At point D, a small decrease in the air flow rate of air can cause a drop in the pressure, leading to a violent aerodynamic pulsation through the all engine. In fact, when a pressure drop occurs, the air inside the compressor will tend to reverse the flowing direction, due to the pressure gradient, causing the effect mentioned above.

There is another important point that must be considered as well as the *surging* point: the *chocking* point. By going from B to C, the mass flow increases while the pressure ratio decreases; the density thus decrease, bringing an increase of the radial component of velocity. This leads to an increase of the resultant velocity hence, of the incidence angle, until the point E is reached, at which no mass flow rate can occur.

After this digression, the more complete compressor map can be shown, for an axial compressor in figure 2.1.6. Joining the surge lines of the several constant rotational speed curves, leads to the draw of the compressor surge line; the several point at the edge of the constant rotational speed curves represent the chocking condition.

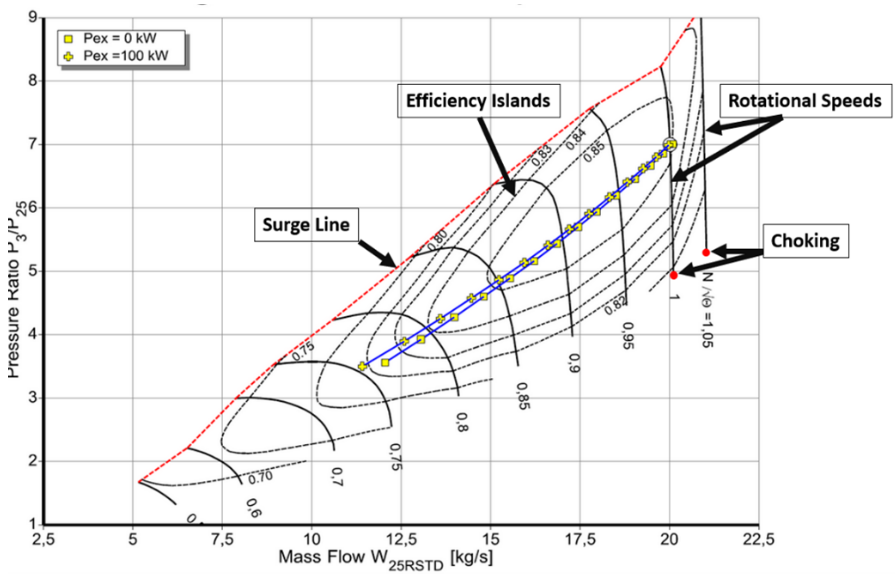


Figure 2.1.6: Compressor characteristic map for an high pressure compressor [10]

The curves of constant rotational speed represent the performance boundaries of the compressor at different rotational speeds, while the *efficiency contours* represent constant efficiency zone. The turbine characteristic map is shown in Figure 2.1.7, but it is not as essential as the compressor map, since no significant variation in the corrected flow rate will occur.

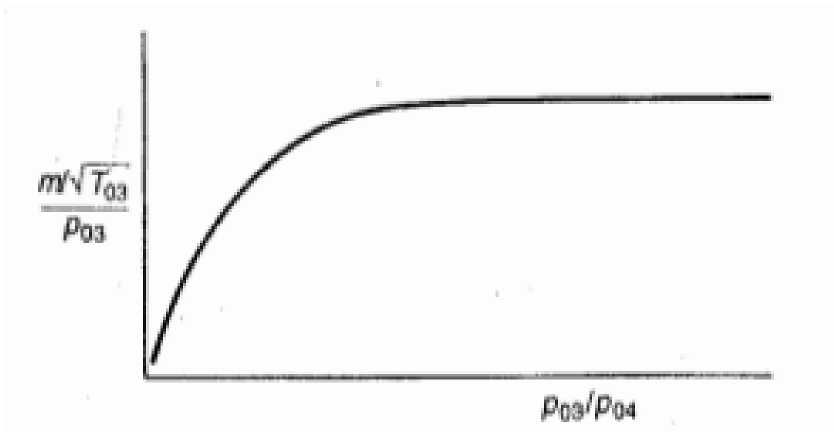


Figure 2.1.7: Turbine characteristic map [6]

Coming back to the analysis of the off-design model for a single shaft gas turbine, some simplifications can be done: in particular, no pressure losses at the intake and at the exhaust are considered, and the pressure losses at the combustion chamber are considered as a fixed percentage value of the pressure [6]. Furthermore, some assumption are made [9]:

- The analysis is made considering a curve of constant rotational speed;

- The curve of constant rotational speed is supposed to be vertical (which is very likely for an axial compressor): this can lead to the decouple of the relation between pressure ratio and flow rate of air. In fact, as it can be seen in the compressor map, if the constant speed curve is vertical, no matter the pressure ratio, the corrected flow rate remains constant, at that speed.

- The turbine is supposed to act as a choked noozle.

- Compressor and turbine polytropic efficiencies are constant.

If it is considered that the turbine is chocked and there is no dimensional change in the corrected flow rate of the turbine, so the compressor as well operates in choked conditions. This, together with the assumption of vertical constant rotational speed, leads to the formulation of the adimensional *corrected* or *reduced* flow rate[9]:

$$\frac{\dot{m}_1\sqrt{T_1}}{P_1} = constant \tag{2.22}$$

Due to this constant relation, in order to find the new parameters in off-design condition, a comparison with known conditions (such as design condition) is just to be done. The equation for the turbine is:

$$\frac{\dot{m}_3\sqrt{T_3}}{P_3} = constant \tag{2.23}$$

However, this equation is used when the working fluid is air and there is no significative change in the molecular weight. In cases where the fluid compositions change quite a lot, it is better to include the molecular weight, as follows [9]:

$$\frac{\dot{m}_3}{P_3}\sqrt{\frac{T_3}{MW_3}} = constant \tag{2.24}$$

Pressure drops across the air filter, the combustor and at the exhaust can be considered. The general equation for pressure drop for turbolent flow in tubes is as follows[6]:

$$\Delta p = f\left(\frac{\rho C^2}{2}\right) \tag{2.25}$$

where f is the frictional factor, equal to:

$$f = \frac{0.0791}{Re^{0.25}} \tag{2.26}$$

Assuming that the frictional losses are constant, the pressure drop across the intake, the combustion chamber or the exhaust can be generally written as[9]:

$$\Delta p = \dot{m}^2\frac{T}{P}MW \tag{2.27}$$

An iterative procedure to calculate the off-design condition based on these equations is explained as follows.
First of all, from equation 2.22, relating the design condition and the off-design condition based on the new ambient temperature and/or new ambient pressure,

the new mass flow rate of air is equal to (where *ref* is for *reference*, which is the design condition):

$$\dot{m}_1 = \dot{m}_{1,ref}\sqrt{\frac{T_{1,ref}}{T_1}}\frac{P_1}{P_{1,ref}} \tag{2.28}$$

The air filter pressure drop is calculated as [9]:

$$\frac{\Delta p_{af}}{\Delta p_{af,ref}} = \left(\frac{\dot{m}_1}{\dot{m}_{1,ref}}\right)^{1.8}\left(\frac{T_1 P_{1,ref}}{T_{1,ref} P_1}\right)^{0.8} \tag{2.29}$$

The pressure at the intake becomes thus:

$$P_1 = P_0 - \Delta p \tag{2.30}$$

where P_0 is the ambient pressure of air before entering the system. Next, a pressure value P_2 is guessed to initialize the iterative calculation: generally this value corresponds to the pressure P_2 in the design conditions. The value for T_2 can be calculated from equation 2.20. As it is done for the air filter in equation 2.29 the combustor pressure drop is calculated as:

$$\frac{\Delta p_c}{\Delta p_{c,ref}} = \left(\frac{\dot{m}_3}{\dot{m}_{3,ref}}\right)^{1.8}\left(\frac{T_3 P_{3,ref}}{T_{3,ref} P_3}\right)^{0.8} \tag{2.31}$$

From the equation established for the turbine in choked conditions 2.24, relating reference and off-design condition, the new inlet for the turbine is:

$$\frac{P_3}{P_{3,ref}} = \frac{\dot{m}_3}{\dot{m}_{3,ref}}\sqrt{\frac{T_3}{T_{3,ref}}\frac{MW_{3,ref}}{MW_3}} \tag{2.32}$$

Considering the pressure drop calculated in 2.31 and the P_3 calculated in 2.32:

$$P_3 = P_2 - \Delta p_c \tag{2.33}$$

At this point, the value guessed for P_2 has to be checked: if it satisfies the equation 2.33, the iteration is ended, if not, it is restarted with a new value of P_2.

2.1.2.1 Variable inlet guide vanes (VIGV)

The off-design model that has just been studied is a highly simplified model, which does not take into account the analysis and modeling of the mechanics of the variable inlet guide vanes, important components for the compressor that have the function of regulating the incoming air flow, constituting an important part of the construction of a complete off-design model.

While changes in ambient conditions "naturally" influence the intake air flow rate, the VIGV provide an active control mechanism to further optimize and stabilize the flow rate in different operating conditions. In design conditions, the variable inlet guide vanes can be considered 100% open, referring to a specific compressor map. But when the VIGVs are at a lower than maximum opening percentage, operating conditions change: all modern turbines, in particular, are equipped with at least one stage of VIGV, resulting in up to a 70 percent reduction in air flow rate. [11].

The equations that describe the phenomenon must be explained, providing numerical example from [11]. When VIGVs are fully closed, the corrected flow rate can be expressed as follows, dependent on the rotation speed and related to the corrected flow rate at 100% open VGIV:

$$\frac{\mu_C}{\mu_O} = (1.55 - 0.85v) \tag{2.34}$$

where μ indicates the corrected flow rate, and the subscript O means the corrected flow rate at 100% open VGIV. When the VGIV is between the 0 and the 100% of the full opening, the corrected flow rate is expressed as function of the opening angle:

$$\frac{\mu_C}{\mu_O} = C_{IGV} \tag{2.35}$$

$$C_{IGV} = 1 + (K - 1)\left(1 - \frac{Y_{IGV}}{100}\right) \tag{2.36}$$

Where Y_{IGV} is the fraction of opening setting in percentage, and K gives the fraction of the 100% open VIGV flow at a given speed v.
In Figure 2.1.8 it can be seen, thus, how the corrected flow rate changes by changing the opening of the VIGV, and so the pressure ratio, defined as $\pi*$.

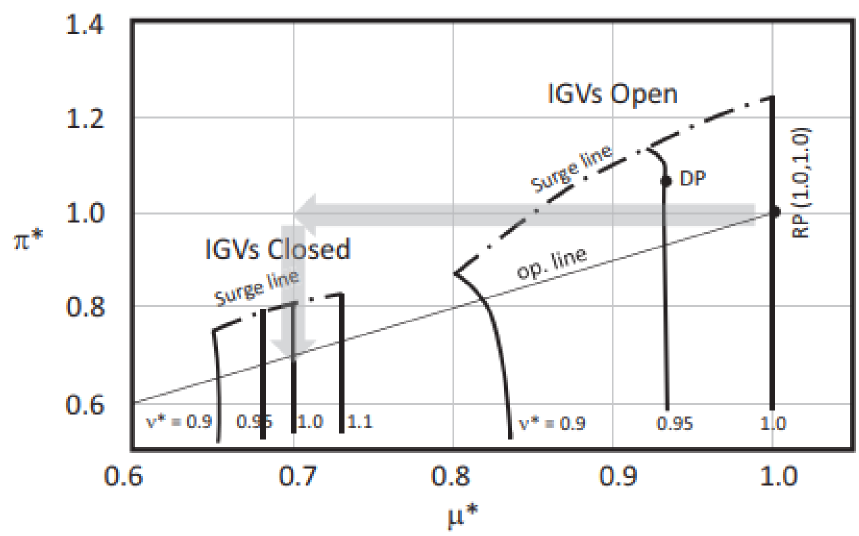


Figure 2.1.8: Change of corrected flow rate in a compressor map, by changing the opening of VGIV from [11]

Considering now $v = 0.9$ and the VGIV at 60% open:

- The full closed flow scaler K is obtained from equation 2.34, and it is equal to 0.785.
- From equation 2.36, C_{IGV} is equal to 0.914.

- Lastly, the flow scaler at 60% open and at $v = 0.9$ from equation 2.35 is 0.914.

In conclusion, for an operating point $\mu = M$ from the performance map for 100 percent open VIGV, and $\mu = 0.914M$ when VGIV are 60 percent open.
Tracing back to the change in ambient temperature, therefore, when the ambient temperature is reduced from the design conditions, the density increases, and therefore the angle of the VGIVs increases to allow more airflow to pass through, which leads to an increase in the compression ratio and a reduction in isentropic efficiency.
In Figure 2.1.9 it can be seen that, at 100% v, an increase in angle change results in an increase in air flow rate, and a coefficient $C_{CIGV} > 1$, as the opening of VGIVs is greater than the opening at 100% in design conditions.

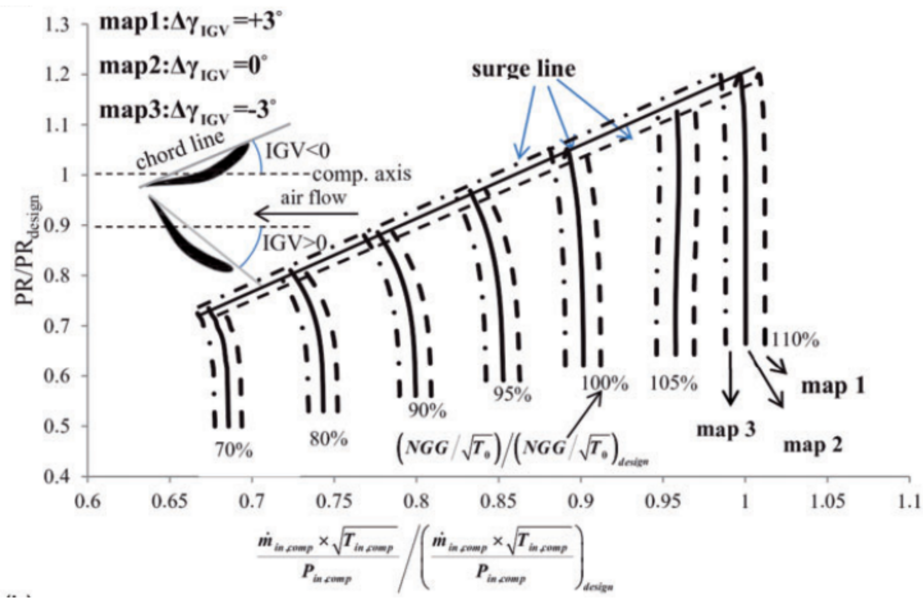


Figure 2.1.9: Change in conditions in a compressor map by changing the VGIV angle [12]

In the Figure 2.1.10 [11] there is a representation of VGIVs, driven by hydraulic actuators, and related to variable guide stator vanes (VGSVs), in this case in three stages, placed on the stator of the compressor, which are in turn responsible for modifying the air flow to maintain the direction of flow and speed as desired.

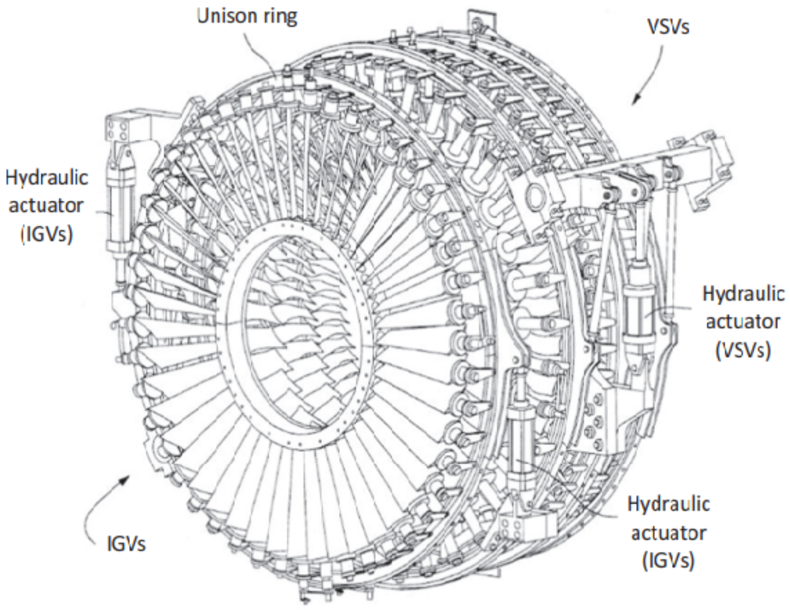


Figure 2.1.10: Graphical representation of VGIV in an axial flow compressor

2.2 GE LM2500 and GE LM6000 gas turbines

The purpose of the project is to create a digital twin for electric power generation in the offshore field: in particular, two turbines are being considered: the aeroderivatives GE (General Electric) LM2500 and GE LM6000. Before getting into the details of the description of the two turbines, a few words must be spent on the concept of 'aeroderivative' gas turbine.

An aircraft gas turbine engine consists of a *net thrust* output, while a landbased gas turbine consists of a *mechanical shaft* output. Specifically, the thrust is described by Newton's second and third laws: for the second law, the acceleration of a body is directly proportional and in the same direction as the force applied on the body itself, and inversely proportional to the mass; for the third law, when a body exerts a force on a second body, the second body simultaneously exerts a force equal in magnitude and opposite in direction to that of the first body. An aircraft thus must produce thrust to overcome the drag of the aircraft, which results from the interaction between the aircraft itself and the air molecules around it. The low that describes the thrust in an aircraft is, from Newton's law [13]:

$$F = \dot{m}\,(V_2 - V_1) \tag{2.37}$$

where F is the thrust, $\dot{m}$ the mass flow of air, V_2 the outlet velocity while V_1 the inlet velocity. Therefore, if an aeroderivative turbine is placed on the ground, the thrust it will produce in output is the static thrust, considering the aircraft speed to be zero, from equation 2.37 [11]:

$$F = \dot{m}V_j \tag{2.38}$$

where V_j is the velocity at the jet nozzle exit.

After having understood the physical principle behind an aeroderivative, some advantages are highlighted: first of all, aeroderivative gas turbines are designed to have a small footprint and low weight, using special materials for high efficiency, and with a very short start-up time, in fact they have the same characteristic as turbojets of fast engine response, especially under changing conditions (and higher pressure ratios as well). The higher efficiency leads to significative fuel cost savings, and the capability to be shut down and switch up quickly allows fast transient and short downtimes for maintenance [14].

Some modifications are made when changing from an aircraft turbine to a derivative one, such as removing the fan and then modifying the low-pressure compressor and low-pressure turbine, as shown in Figure 2.2.1, in which the fan is removed, a power turbine is added and the LP compressor is modified.

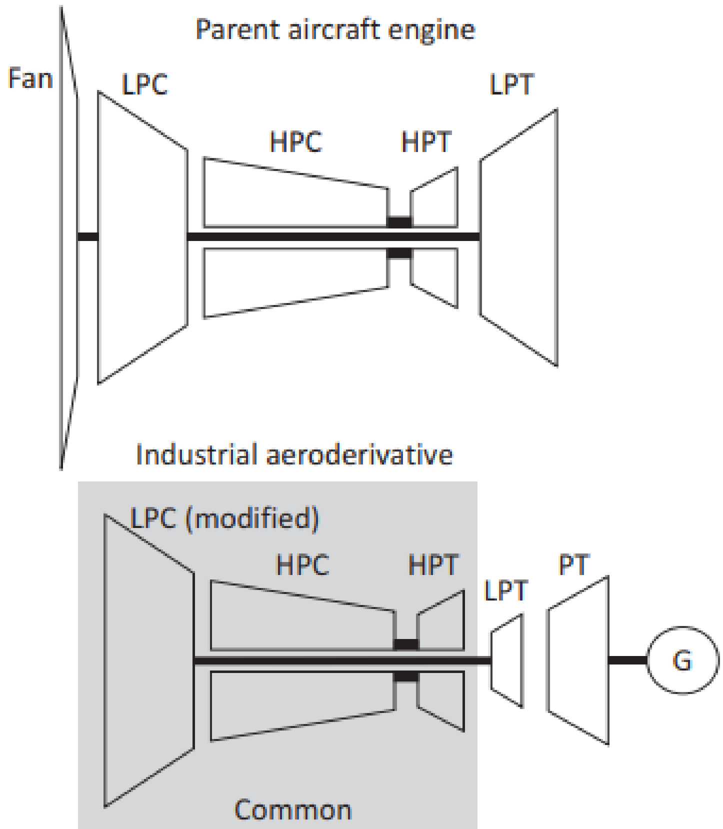


Figure 2.2.1: Modifications of an aircraft turbine for an industrial aeroderivative turbine [11]

General Electric is one of the most important companies in the market for aeroderivative gas turbines, and the turbines object study, GE LM2500 and GE LM6000, are manufactured by this company. A first picture of the two turbines from General Electric website can be seen in pictures 2.2.2 and 2.2.3.

Figure 2.2.2: Picture of GE LM2500 gas turbine from General Electric[15]

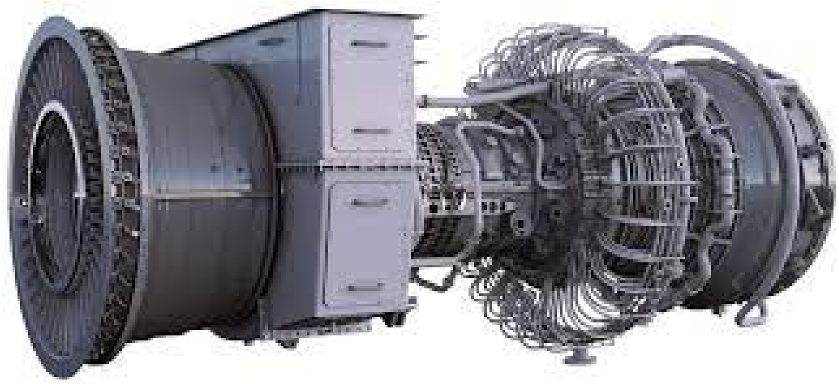


Figure 2.2.3: Picture of GE LM6000 gas turbine from General Electric[15]

A more detailed description of the two turbines will be given in the following section.

2.2.1 Mechanical description

The GE LM2500 gas turbine consist of 16 stages compressor, a combustion chamber and an aerodynamically coupled power turbine.
In particular, a 2-stages high pressure turbine and a 6-stages low pressure (or power) turbine can be distinguished: the HPT drives the compression shaft, while the LPT is coupled to another shaft together with the generator, and provides power. The compression pressure ratio in nominal condition is set to be 18:1 [16].
A representation of GE LM2500 can be seen in Figure 2.2.4.

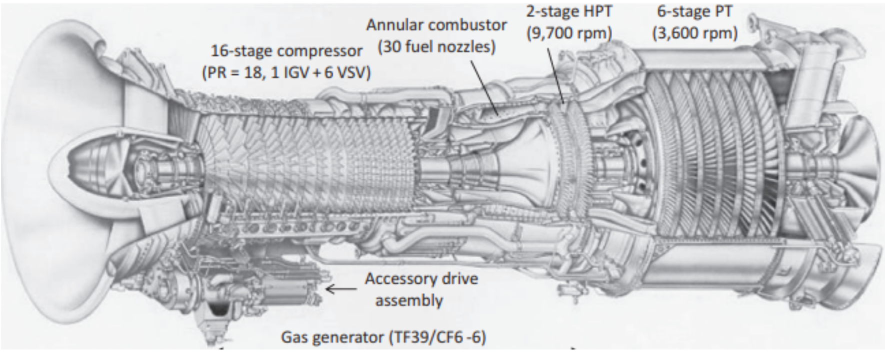


Figure 2.2.4: Representation of the inner structure of GE LM2500 [11]

The GE LM6000 gas turbine was generated from the GE CF6 jet engine, and has been manufactured by General Electric since 1991. The turbine configuration is 2-shaft, made of 2 compressors (one low-pressure LPC and one high-pressure HPC), a combustion chamber, and 2 turbines (one high-pressure HPT and one low-pressure LPT). The nominal compression ratio is 29.1:1, and in particular the LPC consists of 5 stages while the HPC consists of 14 stages [17].

The special feature of the LM6000 turbine is the fact that it is direct-driven, which makes it similar to its parent aircraft. "Direct-driven" means that the low pressure rotor directly drives the turbine without any intermediary components, like a gearbox: in fact, the low pressure rotor it is directly connected to the load it is driving, and in this way more power is provided, with an higher efficiency. In other cases there is the need of a transmission, or a change-speed gearbox that can modify the rotational speed of an input source to an output source shaft: this leads to transmission losses, and it explains why in the peculiar configuration of the LM6000 an higher efficiency can be achieved. In Figure 2.2.5 an illustration of a gearbox is represented, showing how the change of speed works between two different shafts.

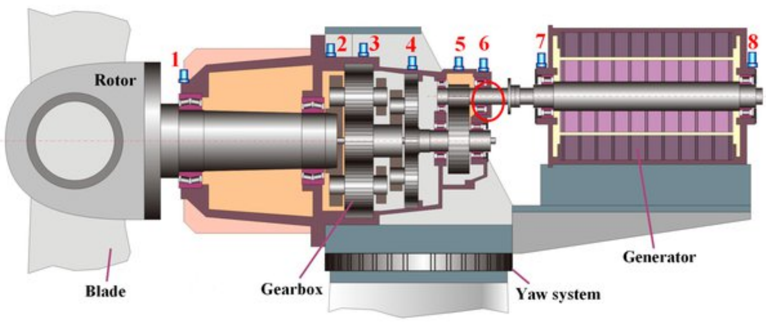


Figure 2.2.5: Illustration of the operation of a gearbox in a gas turbine for changing shaft speed from an input source to an output source.

In conclusion, the GE LM6000 turbine maintained a strong similarity with the corresponding aircraft, resulting in a different design than a classic aeroderivative, which typically adds a power turbine coupled to the generator. Figure 2.2.6 shows the difference between the configuration of the GE LM6000 turbine and the configuration of a normal aeroderivative turbine [16].

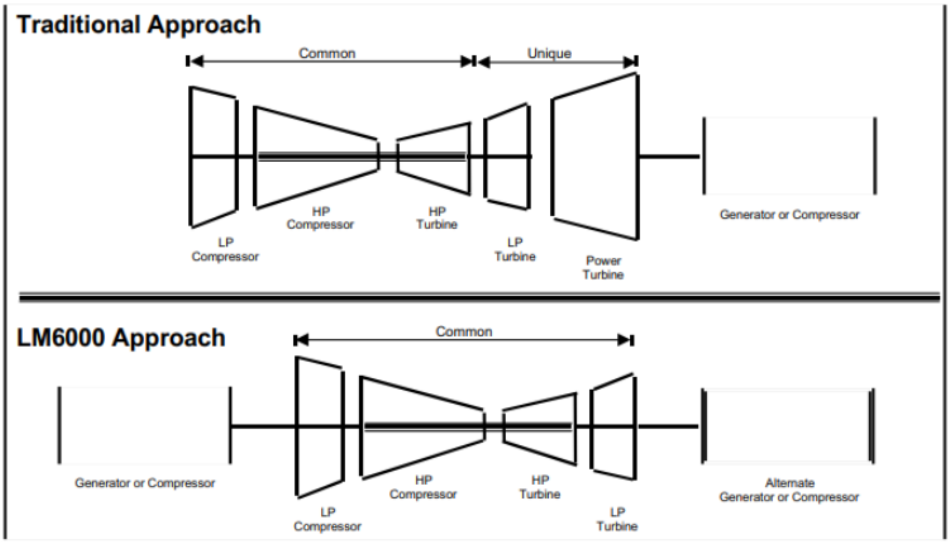


Figure 2.2.6: Different configuration concept of GE LM6000 gas turbine compared to other aeroderivatives

In Figure 2.2.7 there is a representation of GE LM6000 with LPC and HPC, and LPT and HPT pointed out.

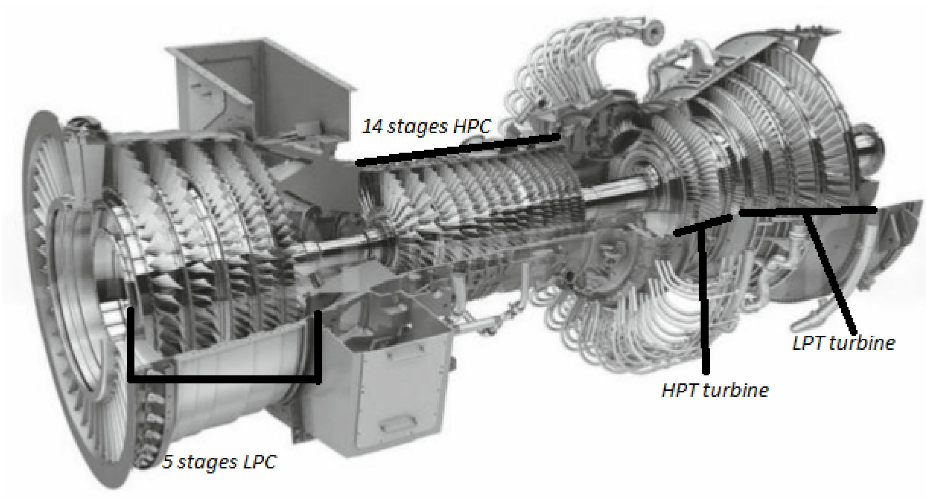


Figure 2.2.7: GE LM6000 representation, with compressor and turbine stages shown

2.2.2 Optimum multistage compression ratio

As explained in the previous section, compressors (and also turbines) consist of several stages, and it is important to understand what is meant by a stage and what it implies computationally. A compressor stage consists of an impeller, also known as rotor, which is the rotating component; the stationary inlet passages (the inlet guide vanes) and the stationary discharge passages (diffuser); and the seals [18].

As explained in 2.1.2.1, the inlet guide vanes are located at the entrance of the compressor stage, and they have the function of directing and guiding the incoming air onto the rotating impeller at the correct angle. The diffuser, instead, is located at the outlet of the compressor stage, and it is aimed to slow down the high-speed air in order to convert its kinetic energy into pressure energy. Seals are components in gas turbines that help to maintain proper airflow and prevent leakage of gases between the different sections of the engine.

In a compressor, thus, the air enters the impeller and it achieves high-speed (kinetic energy) as the impeller rotates, going onto the diffuser afterwards to convert kinetic energy in pressure energy. A schematic representation of the sequence of IGVs, impeller and stator for multiple stage axial compressor is shown in picture 2.2.8.

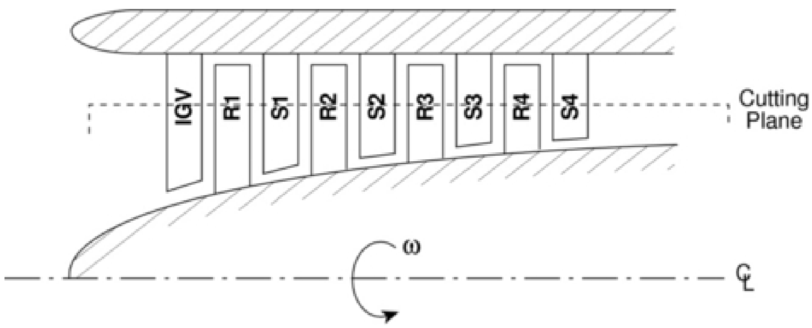


Figure 2.2.8: Sequence of IGVs, impeller and diffuser in a multistage axial compressor

In a turbine, the sequence of rotor and stator is obviously inverted: the stator is located first, acting as a nozzle to increase the velocity of a gas by converting pressure energy in kinetic energy. The rotor, afterwards, converts the kinetic energy to power by causing a rotation of the shaft [9]. A row of stator and rotor is depicted in Figure 2.2.9.

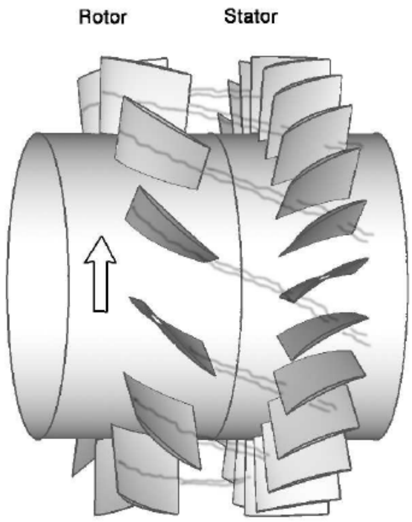


Figure 2.2.9: Rotor-stator configuration [19]

Analyzing the compressor stage, in particular, each stage, at a certain flow rate and shaft speed, will produce a certain amount of energy (head) and have a certain stage efficiency [18]. For the purpose of modeling the digital twin, a model calculation of the compression ratio at each stage is presented, assuming a constant isentropic efficiency for every stage [20].

The purpose of having stages made of rotors and stators lies in the principle of cooling: for minimum power consumption, in fact, gases should ideally be cooled while being compressed, since the power consumption increases as the compression implies hotter gases, becoming economically not sustainable [21].
Since this is not possible, large compressions are divided in stages in the way explained above: by reducing the air velocity through the diffuser at each stage, the air loses heat, which is a saving benefit. Furthermore, the iso-pressure ratio is defined as:

$$r_t = P_{out}/P_{in} \tag{2.39}$$

Thus, considering n stages and constant isentropic efficiency η, the optimal pressure ratio for each stage is formulated as follows [20]:

$$r = r_t^{1/n} \tag{2.40}$$

For example, the pressure ratio in a stage between pressure i and pressure $i+1$ is:

$$r_{i,i+1} = \frac{P_{i+1}}{P_i} \tag{2.41}$$

which leads to:

$$\prod_{i=1}^{n} r_{i,i+1} = \frac{P_2}{P_1}\frac{P_3}{P_2}...\frac{P_{n+1}}{P_n} = r_t \tag{2.42}$$

In the Figure 2.2.10 is depicted in a T-s diagram the compression process divided in stages [20].

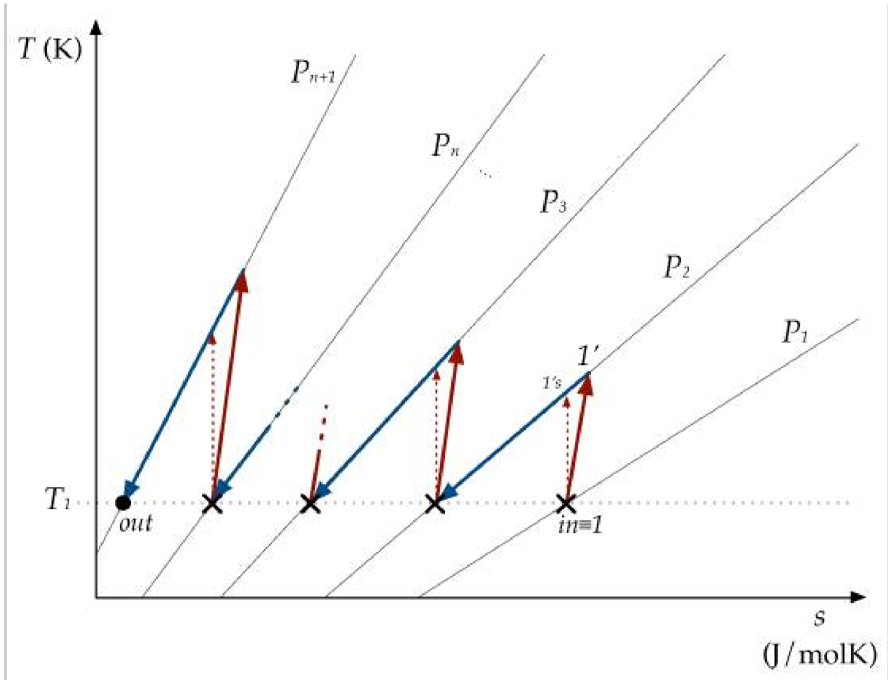


Figure 2.2.10: Compression process divided in stages in a T-s diagram

2.3 Condition monitoring

The pursuit of high reliability, availability, and efficiency in gas turbines has governed the evolution of engine maintenance methods [22], and currently the maintenance cost of a gas turbine is expected to be one of the most impacting costs during the life cycle of the engine. It is suggested that a more cost-efficient way of operating gas turbines could be achieved by enhanced engine condition monitoring and predictive maintenance, implemented together with a digital twin: digitising control of gas turbine behaviour helps to detect, identify and assess components degradation, which in turn affects the maintenance of gas turbine assets in a positive way [23], in terms of performance from both a thermodynamic and an economic perspective [24].

With the aim of presenting an initial implementation approach of digital twin for predictive maintenance purposes, therefore, this section will present an overview of Condition Monitoring methods applied to the energy field and more specifically, to gas turbines, proposing analytical approaches to detect components degradation.

The condition-based maintenance (*CBM*), which is a maintenance strategy that monitors the actual condition of an asset, is an effective method for enhancing the machinery maintenance strategy and shifting from classical "fail and fix" practices to a "predict and prevent" methodology [22]. Generally, Condition Monitoring relies on two different processes: diagnostic approach and prognostic approach [22].

Diagnostics is the process of determining the status of the equipment and components, using information from a technology such as a digital twin, which compares the expected condition with the actual condition. The final purposes of a diagnostic approach are fault detection, fault isolation and fault identification. As explained above, the most significant tool for implementing this process is the digital twin, since, once connected with real-time field data, it can give a real-time comparison between the expected and predicted condition and the actual behaviour, leading to fault identification if a mismatch occurs.

Prognostics, instead, is the ability to forecast the evolution of the engine behaviour and deterioration, with a long term purpose on forecasting the impending failures and estimating the remaining useful life of the engine. In order to implement a prognostic condition monitoring analysis not only a predictive tool for engine behaviour is required, such as a digital twin, but also a consistent data history that reinforces and validates, together with analytical demonstration, the correlation between thermodynamic component behaviour and physical component degradation.

Condition Monitoring related to gas turbines has several developments, depending on the cause of the deterioration, which can fall into two main categories. First, a case of a mechanical nature can occur, such as loose of components, lack of lubrication, unbalance, etc. The second cause is aerodynamic or performance related, which can include fouling, erosion, corrosion, improper combustion, etc. The most known approach in the case of this kind of deterioration is a performance based health monitoring, also known as gas path analysis (*GPA*): the development of the digital twin in this project, in particular, is suitable for this type of analysis, as it models the corresponding "health" status of the real turbine predicted, represented by the engine health parameters, such as compressor and turbine isentropic or polytropic efficiencies [25].

The concept behind the GPA is that physical faults yield to deviation in one or more of the engine health variables or independent parameters, called gas path measurements; these variations, in turn cause deviation in the measured variables, such as pressure and temperature. In a nutshell, component health parameters are not directly measurable, but they are thermodynamically correlated with the measurable parameters [22]; variation in the component health parameters are identified in the literature as $\Delta\vec{x}$, while deviation in the measurable variables as $\Delta\vec{z}$.

In picture 2.3.1, a schematic representation of the implementation of GPA for Condition Monitoring of gas turbines: values of measurable variables come from the sensors, which, when compared with the output of the digital twin, lead to deviations in the expected outputs; these deviations are converted into health deviations of components health parameters through the *FDI*, fault detection and isolation.

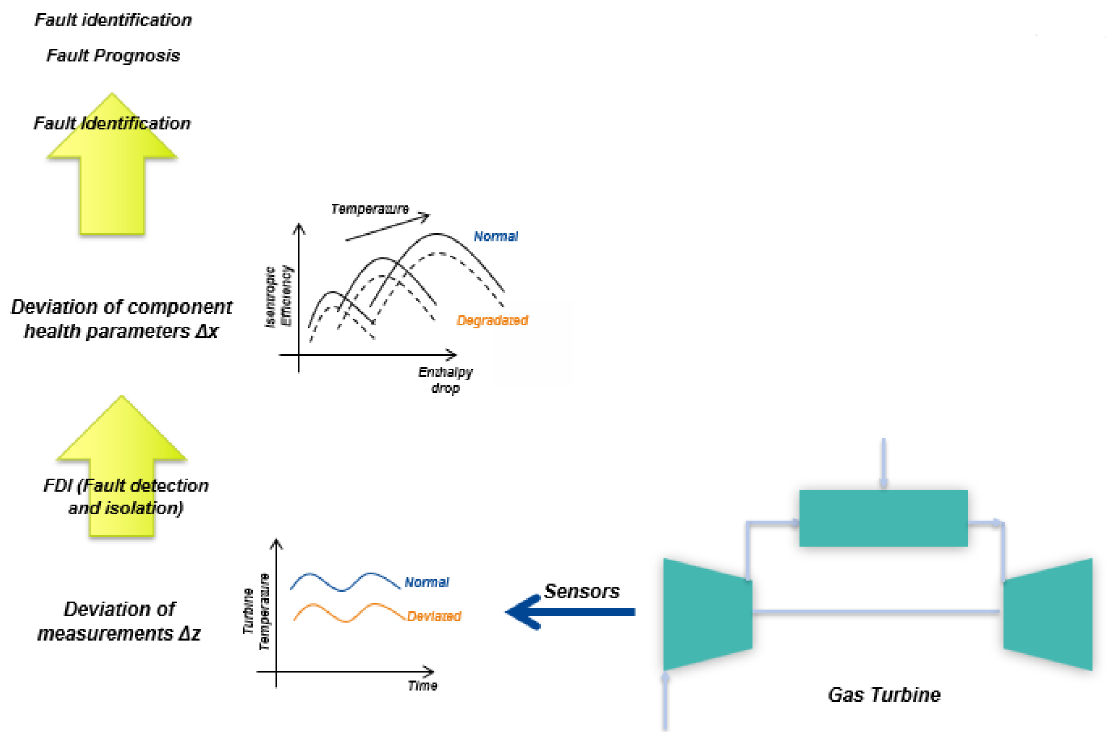


Figure 2.3.1: Condition Monitoring scheme for gas turbine Gas Path Analysis, structure idea from [22]

Fault detection is the procedure for determining whether an abnormal situation has occurred in the monitored system and fault identification is the procedure for estimating information relevant to the fault upon its detection [26].
The point of the FDI is therefore to continuously process the performance of the engine according to the set point of the controller u and the ambient conditions, in order to assess the variation of component health variables z_r (from the sensors) from their clean and health condition z_m (from the Digital Twin simulation), and convert this measured deviation into a deviation in degradation parameters.

In particular, there are several methods to detect the component degradation, and the method that could best suit the digital twin developed in this project is a model-based method. Model-based (or physics-based) methods, in fact, establish mathematical models to describe the physics and the thermodynamics of the components and the systems under study: they are limited to the cases where failure mechanisms can be quantified, and they cannot be easily used for complex systems whose internal parameters are inaccessible for direct measurements by sensors [27], but at the same time they have a strong physical and analytical basis of turbine behaviour, as opposed to data-driven models, which make use of deep learning and other learning techniques that are often unclear to the user.

The following stage is the identification of physical faults: this phase is particularly critical and tricky, as it is not trivial to identify the correlation between a variation in a thermodynamic parameter and the actual physical cause. In the literature, there are several causes of component degradation that can be associated with gas turbines, such as fouling, corrosion, erosion, etc., just as there are several hypotheses of correlation between degradation in terms of efficiency or other variables and physical cause, but in order to advance hypotheses, it is necessary to identify an analytical correlation and verify it with field data.

CHAPTER

THREE

MODELS AND METHODS

This chapter will explain in detail the methodology adopted for the design and modeling of the digital twin, starting with the model in Python of the main classes describing the gas turbine (stream, compressor, combustion chamber and tinder) and continuing with the validation of the model through comparison with Aspen HYSYS®.

Next, the model in Thermoflow© is implemented for the design and off-design simulation of the GE LM2500 and GE LM6000 turbines, and the iterative model in GasTurb© for the identification of the poltiropic and isentropic efficiencies of the components; a simplified off-design model for the GE LM2500 turbine is also presented. Finally, an initial approach to analyse component degradation is presented using a performance indicator that considers the deviation of the actual temperature at the compressor outlet from the temperature calculated by the model.

3.1 Object-oriented code in Python

The first approach to design a digital twin that simulates the behavior of a gas turbine is to create a model that describes its main thermodynamic laws. To fulfill this goal, object-oriented programming in Python is used.

Object-oriented programming allows the development of code that is suitable for reuse and adaptable to multiple contexts, having the objects and methods within the class as the main core; the classes that build the gas turbine model are: stream class, compressor class, combustion chamber class, and expander class. In Figure 3.1.1 a schematic representation of the classes, showing the input values required by each class and the calculated output values.

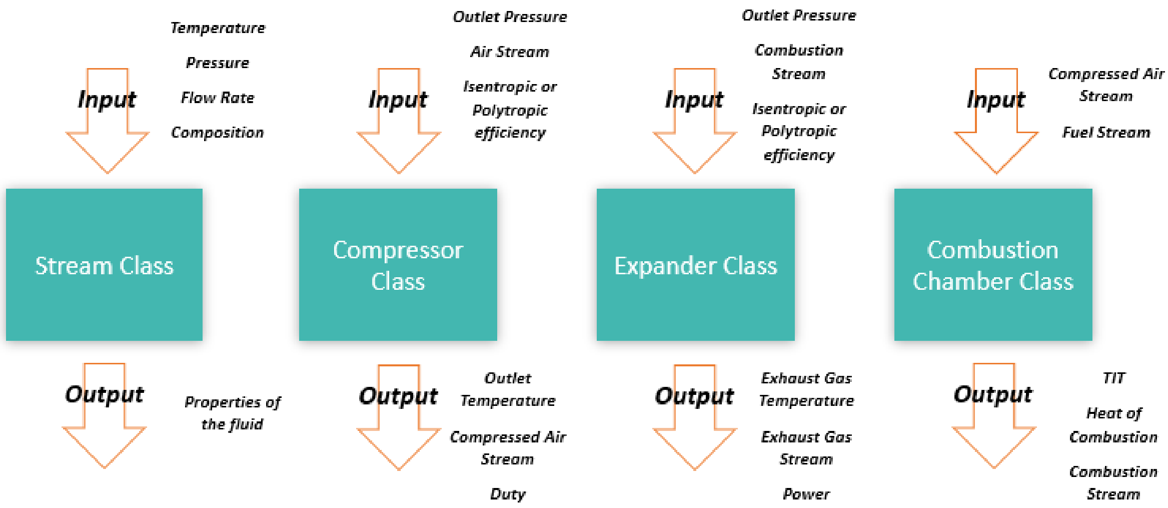


Figure 3.1.1: Python classes scheme with the variable inputs required for each class and the calculated outputs

3.1.1 Stream Class

In this class streams are modeled, whether they are of air, or fuel. To describe a stream, in particular, it is necessary to specify the flow rate, the temperature, the pressure, and the composition. Neqsim, a library for calculating fluid behavior and properties, is used to model the fluid the stream is made of [28].

For a better usability, each method of *set* properties, has a corresponding *get* method. For example, is it possible to specify the temperature in degrees Celsius, but also to display it in degrees Kelvin. Below an example for the temperature is shown:

```python
def set_temperature(self, temperature, units):
    if units == 'K':
        self.temperature = temperature
    if units == 'C':
        self.temperature = temperature + 273.15
```

```python
def get_temperature(self, units):
    if units == 'K':
        return (self.temperature)
    elif units == 'C':
        return (self.temperature - 273.15)
```

In the same way, also mass flow rate and pressure are initialised.

Pressure:

```
def set_pressure(self, pressure):
    self.pressure = pressure
```

```
def get_pressure(self, units):
    if units == 'bara':
        return (self.pressure)
```

Mass flow rate:

```
def set_flow_rate(self, flow_rate, units):
    if units == 'kg/hr':
        self.flow_rate = flow_rate
    elif units == 'kg/sec':
        self.flow_rate = flow_rate * 3600
    else:
        print("ERROR no units found")
```

```
def get_flow_rate(self, units):
    if units == 'kg/hr':
        return self.flow_rate
    elif units == 'kg/sec':
        return self.flow_rate / 3600
    else:
        print(f"ERROR no units found for flow rate units:
        {units} in {self}")
```

Two important methods within the stream class, in particular, are the *calculate()* and *get_LCV()* methods. The *calculate()* method, in particular, is used to initialise the fluid with the Neqsim library in order to calculate fluid properties (such as enthalpy, c_p, c_v, density, etc.). As can be seen in the code shown below, the method takes as input the already set values of pressure, temperature and flow rate, and gives as output the fluid properties.

```
def calculate(self):
    self.fluid.setTemperature(self.temperature, "K")
    self.fluid.setPressure(self.pressure, 'bara')
    self.fluid.setTotalFlowRate(self.flow_rate, 'kg/hr')
    TPflash(self.fluid)
    self.fluid.initProperties()
```

The *get_LCV()* method, on the other hand, is used to calculate the lower calorific value (LCV), as well as the amount of heat released for a unit of fuel completely burnt, and whose combustion products can escape.

```python
def get_LCV(self):
    iso6976 = ISO6976(self.fluid)
    iso6976.setReferenceType('mass')
    iso6976.setVolRefT(15.0)
    iso6976.setEnergyRefT(25.0)
    iso6976.calculate()
    return iso6976.getValue("InferiorCalorificValue") * 1e3
```

This method is of crucial relevance in the modeling phase of the combustion process, since it is needed to calculate the enthalpy of the fuel, but also in the final calculation of the gas turbine, since it is a parameter in the gas turbine efficiency equation and heat rate equation.

3.1.2 Compressor Class

In the compressor class, the inputs required by the model are: stream, output pressure, isentropic or polytropic efficiency. The code returns as output the work and temperature after the compression process. Specifically, the temperature is calculated using two different methodologies. If the polytropic efficiency is not given as input, first the ideal temperature is calculated, which would be if the process were adiabatic and reversible, referring to the equation 2.1 in 2.1:

```python
def calc_ideal_outlet_temp(self):
    base = (self.get_p_out('bara') /
    self.stream.get_pressure('bara'))
    kappa = self.stream.fluid.getGamma2()
    exp = ((1 - kappa) / kappa)
    x = pow(base, exp)
    self.t_ideal_out = self.stream.get_temperature('K') * x
    return (self.t_ideal_out)
```

Next, the real temperature is calculated using the relationship between the ideal work and the real work, through the isentropic efficiency, and referring to equation 2.10 in 2.1:

```python
elif self.pol_efficiency == None:
    self.calc_ideal_outlet_temp()
    delta = self.t_ideal_out - self.stream.get_temperature('K')
    self.t_out = delta / self.ise_efficiency +
    self.stream.get_temperature('K')
```

If the polytropic efficiency is given as input, the output temperature is calculated directly:

```python
if self.pol_efficiency != None:
    base = (self.get_p_out('bara') /
    self.stream.get_pressure('bara'))
    kappa = self.stream.fluid.getGamma2()
```

```
    exp = ((kappa - 1) / (kappa * self.pol_efficiency))
    x = pow(base, exp)
    self.t_out = self.stream.get_temperature('K') * x
```

To have a more compact and more usable code, a calculation function is introduced that automatically returns the output temperature after the compression in case of either isentropic or polytropic efficiency, without the need to call the temperature calculation functions individually:

```
def calc(self):
    if self.p_out is not None:
        if self.pol_efficiency is not None:
            self.calc_outlet_temperature()
        elif self.ise_efficiency is not None:
            self.calc_outlet_temperature()
```

Since the c_p and the c_v change by changing the temperature and the pressure, in order to capture as accurately as possible the variation of these parameters, the compression process has been divided into steps. In fact, the model divides the compression process into n steps, a value defined as input by the user.
The total pressure P_2 is thus divided in n pressure values, every of each is equal to P_2/n. At each iteration i, therefore, the output pressure is defined as the pressure P_{i-1} plus the pressure of the single iteration, equal to P_2/n, and based on this value, the temperature and work at iteration i are calculated.

As a result, the stream c_p and c_v values vary more significantly as pressure and temperature change than the condition in which compression occurs in a single step, and temperature and work values at the final iteration n are more accurate. The method is shown below:

```
def compression_by_steps(self, steps):
    total_p = self.p_out
    iteration = 0
    number_of_steps = steps
    pressure_of_step = (self.p_out - self.stream.get_pressure('bara')) / number_of_steps
    self.p_out = self.stream.get_pressure('bara') # at the beginning the self.p is set as the single step
    temperature_step_before = self.ambient_temperature # inizialize the temperature
    # of the step before at the temperature of the stream
    self.work = 0

    while iteration < number_of_steps:
        if self.p_out < total_p:
            self.p_out = self.p_out + pressure_of_step
            self.calc()
            new_iteration_stream = self.get_outlet_stream()
```

```
            self.stream = new_iteration_stream
            delta_T = self.get_outlet_temperature('K') - temperature_step_before
            temperature_step_before = self.t_out # the new temperature will be used to calculate the work for the step
            flow_rate = self.stream.get_flow_rate('kg/sec') # does not change
            cp = self.stream.fluid.getPhase(0).getCp('kJ/kgK')
            work_new_stage = (flow_rate * delta_T) * cp
            self.work = self.work + work_new_stage
        iteration = iteration + 1
```

Sensitivity analysis that relates the number of iterations and the temperature and work values compared to the baseline defined by the value in Aspen Hysys will be presented in Chapter 4.

Referring to the fact that the compression ratio is divided into compression stages, as explained in 2.2.2, a method for calculating the single-stage compression ratio is implemented for the compressor, used especially to develop the model in detail for the digital twin of the two real turbines. This function takes as input the total compression ratio for the gas turbine, and the stages of each compressor, and returns as output the value of the compression ratio of the single stage and the compression ratio of the single compressor.

```
def calculation_pressure_ratio(stages, pressure_input, p_ratio_stage):
    n = 0
    p1 = pressure_input
    list = []
    while n < stages:
        p2 = p_ratio_stage * p1
        p_ratio = p2/p1
        list.append(p_ratio)
        p1 = p2
        n = n + 1

    pressure_ratio = 1
    for i in list:
        pressure_ratio *= i

    pressure_ratio = pressure_ratio * pressure_input

    return(pressure_ratio)
```

This is important because in gas turbine data sheets, the compression ratio is described as the compression ratio for the turbine as a whole, while the compression ratios for each individual compressor are quite different, and it is important to outline them as accurately as possible in order to analyse the behaviour of the individual component.

3.1.3 Combustor Class

In the class describing the combustion chamber, the main core is the model of the chemical reactions between different hydrocarbons and air, for calculating the chemical compositions of the reaction products and the TIT. Specifically, the input values of the class are the air stream and the fuel stream, while in output are obtained the turbine inlet temperature and the heat of combustion.

The first step to get the TIT, is to calculate the enthalpy of the reactants, by summing the enthalpy of the air and the enthalpy of the fuel.
In particular, the enthalpy of the air is obtained from the method *get_Enthalpy()* from Neqsim, while the enthalpy of the fuel is given by the product between the LCV value and the flow rate in kg/sec:

```
def calc_enthalpy(self):
    enthalpy_air = self.air.fluid.getEnthalpy()
    enthalpy_fuel = self.fuel.get_LCV() * self.fuel.get_flow_rate('kg/sec') # joul / kg * kg / sec = joul / sec = watt
    self.enthalpy = enthalpy_air + enthalpy_fuel
    return (self.enthalpy)
```

In order to calculate the reaction temperature, in fact, a function of Neqsim *PHflash()* is used, which requires as input an enthalpy (in this case, the enthalpy of the reactants) and a fluid with a certain composition, and calculates the output temperature: in this sense, the type of fluid given as input becomes critical to understand the importance of modeling chemical reaction as accurately as possible. In fact, if only air (mainly composed of nitrogen and oxygen) were considered as the fluid, the specific heat capacity c_p would be quite different from the c_p of the fluid produced by the combustion reaction (consisting of nitrogen as inert gas, water vapor, carbon dioxide, and a percentage of unreacted oxygen).

A chemical reaction calculation method is therefore implemented, so that the chemical compositions of the combustion products could be obtained and entered into the *PHflash()*function in order to obtain a more correct value for the TIT.
For validation of the results, refer to Chapter 4 where an analysis between the TIT values with and without chemical reactions is conducted, comparing these values with those of the simulation model in Aspen HYSYS®.

To simplify the reading of the fluids implemented in the code, two dictionaries are created, one for air and one for fuel, which as keys have the component names, and as values the mole fraction of each component.
Below, the development of the dictionary for air and for the fuel:

```
number_of_components_air =
self.air.fluid.getNumberOfComponents()
names_air = [self.air.fluid.getComponent(i).getName()
             for i in range(number_of_components_air)]
molar_fractions_air = [self.air.fluid.getComponent(i).getx()
                       for i in range(number_of_components_air)]
air_dictionary = {}

for i in range(number_of_components_air):
    air_dictionary[names_air[i]] = molar_fractions_air[i]
```

```
number_of_components_fuel =
self.fuel.fluid.getNumberOfComponents()
names_fuel = [self.fuel.fluid.getComponent(i).getName()
              for i in range(number_of_components_fuel)]
molar_fractions_fuel = [self.fuel.fluid.getComponent(i).getx()
                        for i in
                        range(number_of_components_fuel)]

fuel_dictionary = {}

for i in range(number_of_components_fuel):
    fuel_dictionary[names_fuel[i]] = molar_fractions_fuel[i]
```

After that, the number of moles of oxygen and nitrogen in the air is calculated, based on the mass flow rate of the stream (since the mass fraction is calculated based on the flow rate in kg/sec, the number of moles will be defined in moles/sec):

```
molar_mass_mix = air_dictionary['oxygen'] * 31.998 +
air_dictionary[
    'nitrogen'] * 28.013
weight_fractionO2 = (air_dictionary['oxygen'] * 31.9989) /
molar_mass_mix
weight_fractionN2 = (air_dictionary['nitrogen'] * 28.013) /
molar_mass_mix
massO2 = weight_fractionO2 * self.air.get_flow_rate('kg/sec')
massN2 = weight_fractionN2 * self.air.get_flow_rate('kg/sec')
molN2 = (massN2) * (1 / (28.013 / 1000))
molO2 = (massO2) * (1 / (31.998 / 1000))
```

Then, for each hydrocarbon in the fuel, the number of moles is calculated following the same procedure implemented for the calculation of the number of moles of oxygen and nitrogen in the air. The number of moles produced of CO2 and H20 is then calculated for the moles of hydrocarbon burnt, as well as the moles of unreacted oxygen (for simplification the fuel is always considered as the limitant reactant, due to the fact that in gas turbines the mass flow rate of air is always greater than the mass flow rate of fuel, and so is the amount of oxygen, that always allows the total combustion of the fuel).

Nitrogen, being an inert gas that does not participate in combustion, is considered to be totally present in the value of total moles participating in the reaction for each hydrocarbon, while oxygen, participates for each reaction with an amount equal to the value of total oxygen from which is subtracted the oxygen that has already reacted with the previous hydrocarbons: at every iteration, in fact, the total amount of oxygen available for the combustion is reduced based on the amount of oxygen reacted to burnt the moles of the hydrocarbon considered before. The code for methane is given as an example.

```
methane =
(self.fuel.fluid.getPhase(0).getComponent('methane').getMolarMass())
* 1000
weight_fraction_methane = (fuel_dictionary['methane'] * methane)
/ molar_mass_mix_ng
mass_methane = weight_fraction_methane *
self.fuel.get_flow_rate('kg/sec')
mol_methane = mass_methane * (
        1 / self.fuel.fluid.getPhase(0).getComponent('methane')
        .getMolarMass())
```

```
molCO2 = mol_methane * (1 / 1)
molH2O = mol_methane * (2 / 1)
molO2_not_reacted_methane = molO2 - mol_methane * 2
total_moles = molCO2 + molH2O + molO2_not_reacted_methane +
molN2
O2_methane = (molO2_not_reacted_methane / total_moles) *
fuel_dictionary['methane']
CO2_methane = (molCO2 / total_moles)
H2O_methane = (molH2O / total_moles)
molO2 = molO2 - mol_methane * 2
```

In the model, however, the majority of hydrocarbons the natural gas is made of, such as, in addition to methane, ethane, propane, n-butane, i-butane, i-pentane, n-pentane and n-hexane, are considered. After the moles of water vapour and carbon dioxide are calculated for every component of the fuel, the new composition can be determined, by summing all the the molar fraction of every products together, so that the combustion fluid is created:

```
to_turbine = fluid('srk')
to_turbine.addComponent('oxygen', mfO2)
to_turbine.addComponent('nitrogen', mfN2)
to_turbine.addComponent('CO2', mfCO2)
to_turbine.addComponent('H2O', mfH2O)
```

3.1.4 Expander Class

The expander class is modeled in a similar way to the compressor class.
The stream, a polytropic or isentropic efficiency, and the pressure at which the gas has to be expanded are required as inputs. As it is done for the compressor, also in the case of the turbine the expansion process is divided in steps, in order to detect the smallest changes of the c_p and c_v as temperature and pressure changes:

```
def expansion_by_steps(self, steps):
    starting_p = self.P3
    p_end = self.p_out
    total_p = starting_p - p_end
    iteration = 0
    number_of_steps = steps
    pressure_of_step = total_p / number_of_steps
    self.p_out = starting_p
    self.work = 0
    temperature_step_before = self.TIT

    while iteration < number_of_steps:
        if self.p_out >= p_end:
            self.p_out = self.p_out - pressure_of_step
            self.calc()
```

```
            new_iteration_stream = self.get_outlet_stream()
            self.stream = new_iteration_stream
            delta_T = temperature_step_before -
            self.get_outlet_temperature('K')
            temperature_step_before = self.t_out
            flow_rate = self.stream.get_flow_rate('kg/sec')
            cp = self.stream.fluid.getPhase(0).getCp('kJ/kgK')
            work_iteration = (flow_rate * delta_T) * cp
            self.work = self.work + work_iteration
        iteration = iteration + 1
```

Since the model is designed to be used as a basis for building a Digital Twin for systems more complex than a simple single-shaft gas turbine with a compressor, a combustion chamber and a turbine, a function has been included to calculate the expansion pressure of a turbine not directly coupled to the generator, but coupled to one or more compressors.

In particular, this function allows the turbine output pressure to be calculated based on the work required to run the compressor(s) to which it is coupled. As will be seen later in the design and modeling of the turbines under study (GE LM6000 and GE LM2500) for example, the high-pressure turbine is not connected to the generator, but has the only function of producing mechanical work to rotate the compressor shafts. This makes the turbine to not expand up to ambient pressure, but up to a pressure value lower enough to produce a work equal to the work required by the compressor; and since expansion stages are not often available, the function shown below allows this unknown pressure value to be calculated, having the work of compression as input:

```
def calc_p_out_iterations(self, work, units):
    resetting_stream = self.stream
    self.T4 = self.TIT - (work /
    (self.stream.get_flow_rate('kg/sec') *

                              self.stream.fluid.getCp('kJ/kgK')))
    self.defined_work = work
    temperature_ratio = self.T4 / self.TIT
    k = self.stream.fluid.getGamma2()
    exponent = (k / (k - 1)) * (1 / self.pol_efficiency)

    self.p_out = self.P3 * ((temperature_ratio) ** exponent)
    self.expansion_by_steps(100)
```

As in the case of expansion and compression processes, however, the value of c_p and c_v vary as pressure and temperature change, and with the application of the formula alone, the pressure value obtained corresponds to a work value that does not perfectly coincide with the work of the compressor. Therefore, an iterative algorithm is implemented, which reduces the pressure by a minimum value close to zero at each iteration, until the resulting work is greater by a maximum of 10 kW (tolerance value set) than the value of work of the compressor:

```python
# iterating on the pressure:
iteration = 100
i = 0
pressure_list = []
work_list = []
tolerance = 10 # kw
while i < iteration:
    self.work = 0
    self.stream = resetting_stream
    self.p_out = self.p_out - 0.001
    pressure_list.append(self.p_out)
    self.expansion_by_steps(100)
    work_list.append(self.work)
    i = i + 1

closest_value = None
min_difference = float('inf')
corresponding_pressure = None

for i in range(len(work_list)):
    difference = abs(work_list[i]- work)
    if abs(work_list[i]- work) < tolerance:
        if work_list[i] > work:
            if difference < min_difference:
                min_difference = difference
                corresponding_pressure = pressure_list[i]

                print('The work that can satisy ',work,' is ',work_list[i])
                print('The pressure that corresponds to that work is ',corresponding_pressure)

                self.p_out = corresponding_pressure
                self.work = 0
                self.stream = resetting_stream
                self.expansion_by_steps(100)
```

A numerical analysis will be shown in the Chapter 4.

3.2 Single shaft gas turbine

To analyse the calculation methods developed in Python, a case study of a single-shaft gas turbine (consisting of a compressor, a combustion chamber and a turbine) is developed in design conditions (a single operating point referred to a single ambient temperature) and in off-design conditions (more operating points referred to different ambient temperatures), also considering different type of fuels. The same model is designed in Aspen HYSYS®, in order to compare the results, and validate the model in Python.

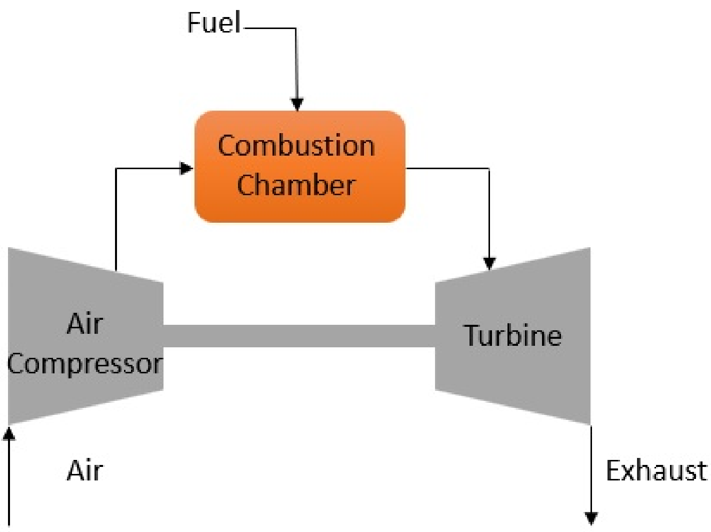


Figure 3.2.1: Simplified illustration of the single-shaft case study turbine

In particular, the data assumed for the stream of air are:

- mass flow rate: 50 kg/sec
- temperature: 20°C,considered as design temperature
- pressure: 1 bara
- composition in molar fraction: 0.8 for N_2 and 0.2 for 0_2

The data assumed for the compressor and the turbine are:

- compressor outlet pressure: 10 bara
- compressor polytropic efficiency: 87%
- turbine outlet pressure: 1 bara
- turbine polytropic efficiency: 90%

For simplification, in this case study no pressure losses at the intake or at the exhaust are considered, since the main objective of the example is to estimate the accuracy of the thermodynamic model.

In the first case study, only methane at 20°C and at 10 bara is considered. Then analysis on the TIT value considering different fuels but the same gas turbine design are done, and below a table with the compositions (expressed in molar fraction) of the case study fuels is shown:

Fuel	CH_4	C_2H_6	C_3H_8	C_4H_{10}	C_5H_{12}*	C_5H_{12}**
Methane	1.0	0.0	0.0	0.0	0.0	0.0
Ethane	0.0	1.0	0.0	0.0	0.0	0.0
Propane	0.0	0.0	1.0	0.0	0.0	0.0
I-butane	0.0	0.0	0.0	1.0	0.0	0.0
Fuel1	0.8	0.2	0.0	0.0	0.0	0.0
Fuel2	0.0	0.8	0.2	0.0	0.0	0.0
Fuel3	0.5	0.5	0.0	0.0	0.0	0.0
Fuel4	0.8	0.0	0.2	0.0	0.0	0.0
Fuel5	0.6	0.2	0.2	0.1	0.0	0.0
Fuel6	0.5	0.2	0.1	0.1	0.1	0.0
Fuel7	0.4	0.2	0.1	0.1	0.1	0.1

Table 3.2.1: Case study fuel compositions in molar fractions; *C_5H_{12} refers to iso-pentane; **C_5H_{12} instead, refers to normal-pentane

Below, it is shown as an example, how the code for modelling the turbine components is developed, starting from the air fluid:

```
component_names = ["oxygen", "nitrogen", "methane", "ethane", "propane",
                   "i-butane", "n-butane", "i-pentane", "n-pentane", "n-hexane", "H2O", "CO2"]
air_composition = [0.2, 0.8, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
```

```
air = fluid("srk")
for component in component_names:
    air.addComponent(component, air_composition[component_names.index(component)])
```

The air stream is therefore designed by entering the values for fluid, pressure, temperature and flow rate:

```
air_stream = Stream()
air_stream.set_fluid(air)
air_stream.set_temperature(20, 'C')
air_stream.set_pressure(1)
air_stream.set_flow_rate(50, 'kg/sec')
air_stream.calculate()
```

Compressor, combustion chamber and turbine are built the with a similar procedure. For the compressor:

```
compressor = Compressor()
compressor.set_losses(0)
compressor.set_stream(air_stream)
compressor.set_p_out(10)
compressor.set_pol_efficiency(0.87)
compressor.calc_isentropic_efficiency()
compressor.compression_by_steps(100)
```

For the combustion chamber:

```
combustor1 = Combustor()
combustor1.set_stream_air(compressor.outlet_stream)
combustor1.set_stream_fuel(methane_stream)
combustor1.calc_enthalpy()
combustor1.calc_TIT_reaction()
combustor1.calc_enthalpy()
combustor1.calc_TIT()
```

As it can be seen, the stream inserted in every component is the outlet stream from the component before. Another consideration for the combustion chamber is that the temperature is calculated both considering the chemical reaction, implementing the *calc_ TIT_ reaction/()*, and not considering the chemical reaction, impelemnting the *calc_ TIT()*.
For the turbine:

```
turbine = Expander()
turbine.set_losses(0)
turbine.set_stream(combustor1.outlet_stream)
turbine.set_p_out(1)
turbine.set_pol_efficiency(0.9)
turbine.calc_isentropic_efficiency()
turbine.expansion_by_steps(100)
turbine.get_outlet_stream()
```

3.2.0.1 Off design model

The off-design model developed calculates the change of the operating conditions as the ambient temperature changes, the explanation for which is described in

2.1.2. As the ambient temperature changes, the new air flow rate is first calculated, relative to the corrected mass flow rate; the model then implements a while cycle that iterates the calculation of the compressor outlet pressure that can satisfy the equation 2.33, through equations 2.31 and 2.32. Once the new P_2 and P_3, net of the compressor pressure loss, have been found, the new turbine parameters are calculated. For the code, refer to Appendix, in .2.

3.2.1 Aspen HYSYS® model

As mentioned in the previous section, the same case study is built in Aspen HYSYS®, whose values are taken as a baseline to validate the results of the model in Python [29]. For the design conditions described, in particular, the model looks like this:

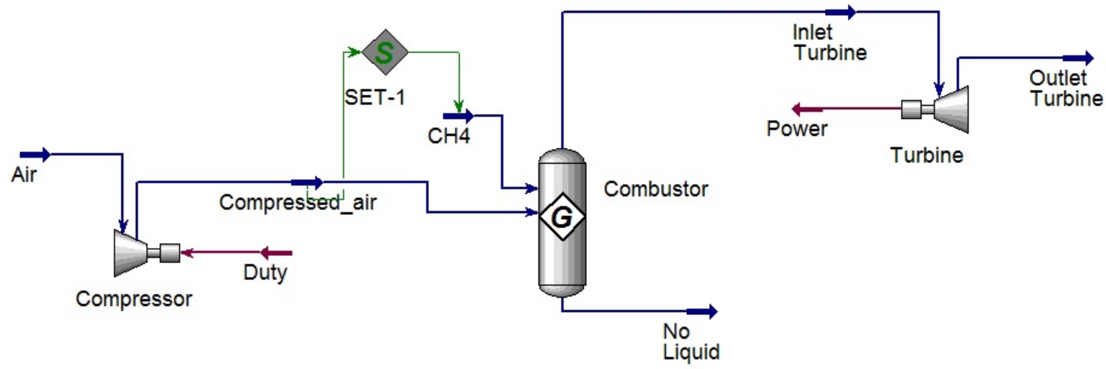


Figure 3.2.2: Illustration of the design model in Aspen HYSYS®

The procedure for modelling the gas turbine in Aspen HYSYS® will be briefly explained below. Firstly, in the *Properties* section, the components and the fluid type are selected. As shown in the pictures below, the list of components includes all the various hydrocarbons typical of natural gas in addition to nitrogen and oxygen.

Component	Type	Group
Methane	Pure Component	
CO2	Pure Component	
Oxygen	Pure Component	
Nitrogen	Pure Component	
H2O	Pure Component	
Ethane	Pure Component	
Propane	Pure Component	
i-Butane	Pure Component	
n-Butane	Pure Component	
i-Pentane	Pure Component	
n-Pentane	Pure Component	
n-Hexane	Pure Component	

Figure 3.2.3: Selection of components

The last step to complete the implementation of the properties includes the setting of the chemical reactions for modeling the combustion chamber.
In the picture below is shown as example what the reaction between methane and oxygen looks like:

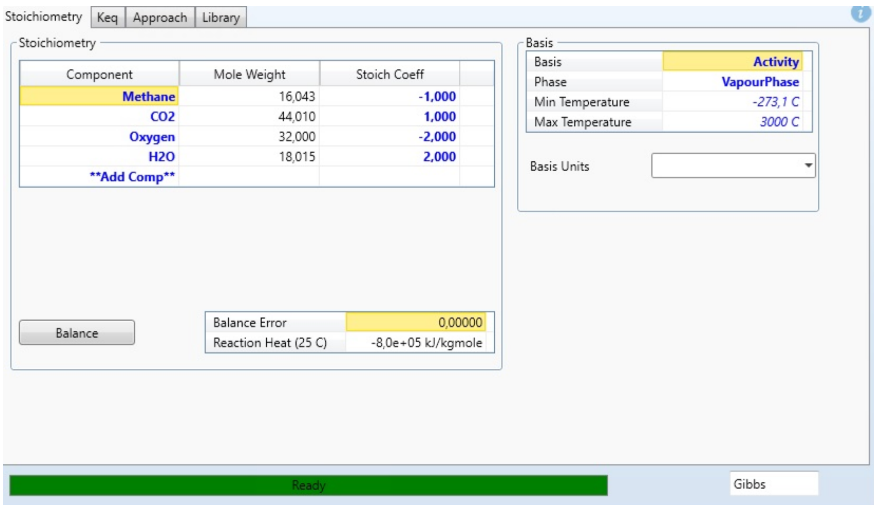


Figure 3.2.4: Example of setting a chemical reaction: methane

Afterwards, the modelling phase of the gas turbine components is performed in the section *Simulation*, by entering the same values as assumed when designing the model in Python. In particular, some considerations may be made regarding the Gibbs reactor, used to model the combustion chamber.

The Gibbs reactor calculates the outlet composition, and it is based on the principle that the free Gibbs energy is at a minimum equilibrium [30].
Gibbs free energy is a thermodynamic value, representing the chemical potential that is minimised when a system reaches equilibrium at constant pressure and temperature. In particular, it is based on the principle that every natural system tends to reach the lowest possible energy level: the quantitative measure representing this phenomenon is indeed the change in Gibbs free energy, which, when negative, indicates a favoured process, which releases energy. On the other side, when the change of Gibbs free energy is positive, it indicates a state of non-equilibrium of the system, which requires work and energy to favour the reaction, and reach the minimum possible energy level [31].

Another consideration can be made regarding the *SET* function, which is a function that adjusts the pressure of the methane to the pressure of the air leaving the compressor: without this function, the fuel would reduce the air pressure if it were at ambient temperature (for simplification, any fuel compressor is neglected).

3.2.1.1 Off design model in Aspen HYSYS®

The off-design model in Aspen HYSYS® is constructed following the procedure described in 2.1.2; as the model built for the design, the off-design simulation in HYSYS® is used for checking the behaviour of the Python code by getting further away from the design operating point. In particular, two simulation schemes are implemented, one referring to design conditions and one referring to off-design conditions. In Figure 3.2.5 the design simulation is shown, which is similar to the one used for developing the design point; in Figure 3.2.6 the off-design simulation is shown, together with the calculation spreadsheet.

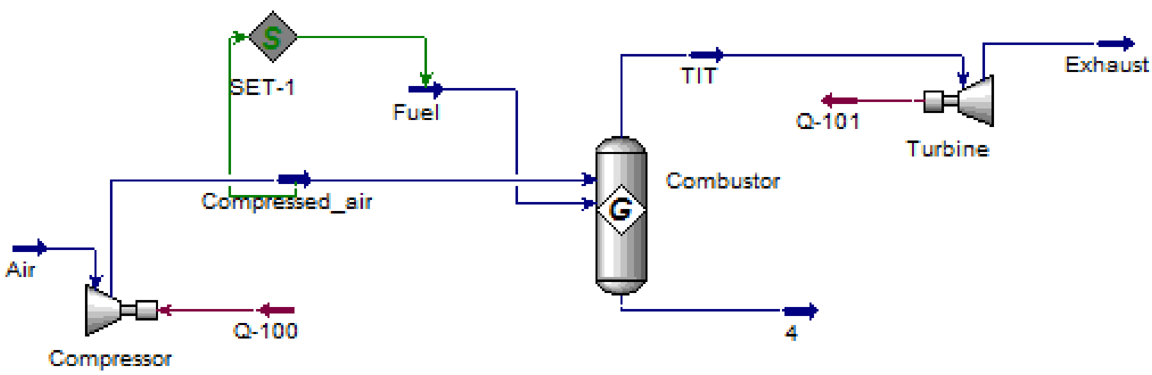


Figure 3.2.5: Design simulation model in Aspen HYSYS®, built as a baseline for the off-design calculations.

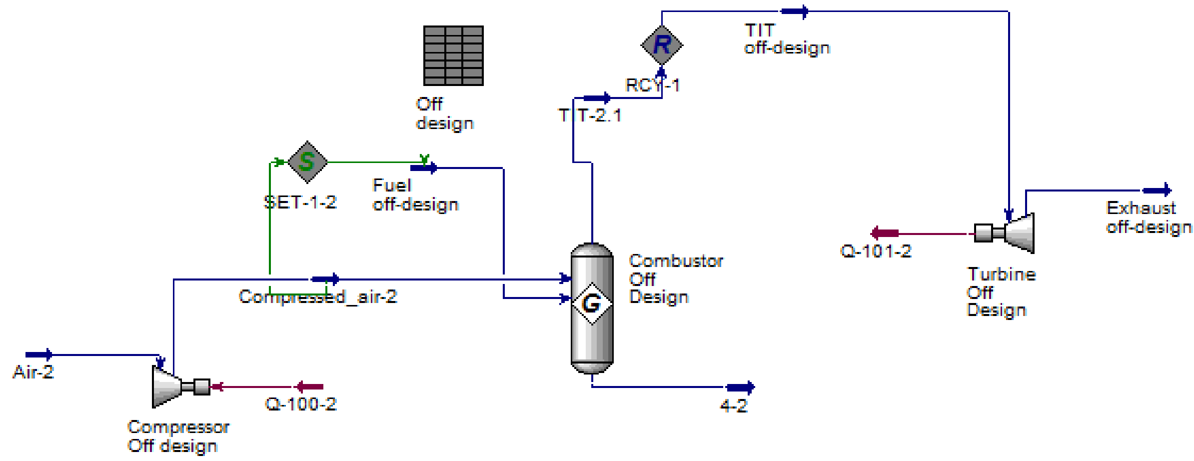


Figure 3.2.6: Off-design simulation model in Aspen HYSYS®, connected through the calculation spreadsheet to the design simulation model

By linking a spreadsheet to the model and implementing 'case study' calculations, the ambient temperature changes, affecting the calculations of the new pressure values, and consequently the calculations of the new parameters of the gas turbine. Figure 3.2.7 shows as an example the variables exported from the calculation spreadsheet to the new off-design model: as can be seen, the new air flow rate, dependent on the ambient temperature, is calculated according to the corrected flow rate formula; the pressure values P_2 and P_3 are calculated and exported as well.

Exported Variables

Cell	Object	Variable Description	
B20	Air-2	Mass Flow	
B30	TIT-2.1	Pressure	
B31	Compressed_air-2	Pressure	

Figure 3.2.7: Exported variables from the spreadsheet to the off-design simulation model

3.3 Model for GE LM6000 and GE LM2500

The design of the class-based code describing the behaviour of a generic gas turbine is functional to the design of digital models for real plants, in order to simulate the behaviour under design and off-design conditions of reference turbines in as much detail as possible.

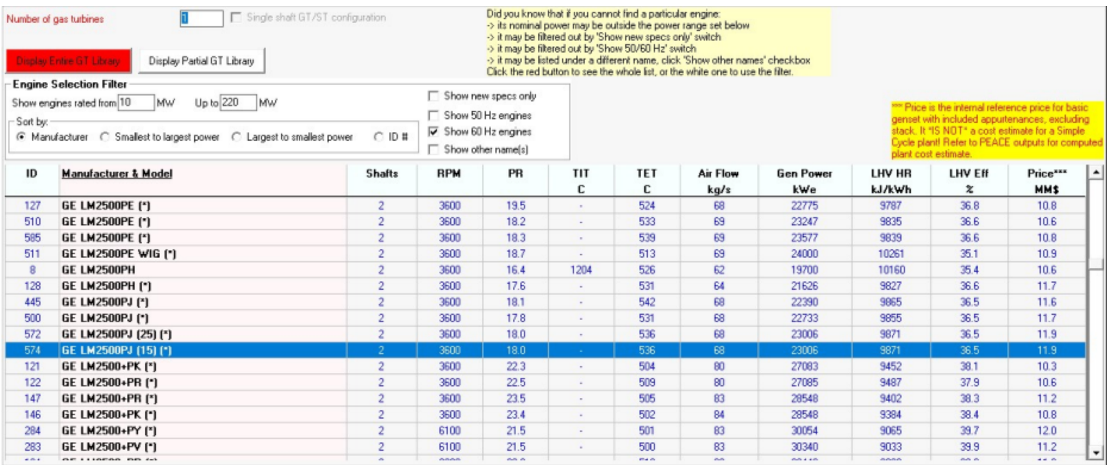
In the specific case study, the digital twin design concerns the GE LM2500 and GE LM600 turbines, described in 2.2.1. The modelling procedure of the turbines

is carried out in the following way: firstly, the corresponding design and off-design model is built in Thermoflow©, in order to obtain significant parameters and values, such as exhaust gas temperature, or power output.
Next, an iterative procedure is implemented in GasTurb© to identify the polytropic and isentropic efficiencies under design conditions of the different components, a fundamental step to have an optimal baseline parameter for condition monitoring purposes. Finally, the model is designed in Python, and validated through a comparison with Thermoflow©.

3.3.1 Thermoflow© model

Thermoflow© is the leading developer of thermal engineering software for the power and co-generation industries [32]. It is a software aimed to create a plant configuration and technical parameters that suit the criteria inserted as input. The software is structured in several programs, and the one in particular used for design modelling is GT PRO.

First of all, the gas turbine model must be selected, as shown in Figures 3.3.1 and 3.3.2: as it can be seen, the selection of the specific gas turbine implies certain standard input parameters: heat rate, exhaust flow rate, gross power output, gas turbine efficiency, pressure ratio, etc. Those parameters will be adjusted based on the given input.



ID	Manufacturer & Model	Shafts	RPM	PR	TIT C	TET C	Air Flow kg/s	Gen Power kWe	LHV HR kJ/kWh	LHV Eff %	Price*** MM$
127	GE LM2500PE (*)	2	3600	19.5	-	524	68	22775	9787	36.8	10.8
510	GE LM2500PE (*)	2	3600	18.2	-	533	69	23247	9835	36.6	10.6
585	GE LM2500PE (*)	2	3600	18.3	-	539	69	23577	9839	36.6	10.8
511	GE LM2500PE WIG (*)	2	3600	18.7	-	513	69	24000	10261	35.1	10.9
8	GE LM2500PH	2	3600	16.4	1204	526	62	19700	10160	35.4	10.6
128	GE LM2500PH (*)	2	3600	17.6	-	531	64	21626	9827	36.6	11.7
445	GE LM2500PJ (*)	2	3600	18.1	-	542	68	22390	9865	36.5	11.6
500	GE LM2500PJ (*)	2	3600	17.8	-	531	68	22733	9855	36.5	11.7
572	GE LM2500PJ (25) (*)	2	3600	18.0	-	536	68	23006	9871	36.5	11.9
574	GE LM2500PJ (15) (*)	2	3600	18.0	-	536	68	23006	9871	36.5	11.9
121	GE LM2500+PK (*)	2	3600	22.3	-	504	80	27083	9452	38.1	10.3
122	GE LM2500+PR (*)	2	3600	22.5	-	509	80	27085	9487	37.9	10.6
147	GE LM2500+PR (*)	2	3600	23.5	-	505	83	28548	9402	38.3	11.2
146	GE LM2500+PK (*)	2	3600	23.4	-	502	84	28548	9384	38.4	10.8
284	GE LM2500+PY (*)	2	6100	21.5	-	501	83	30054	9065	39.7	12.0
283	GE LM2500+PV (*)	2	6100	21.5	-	500	83	30340	9033	39.9	11.2

Figure 3.3.1: Selection of GE LM2500 gas turbine model in GT PRO

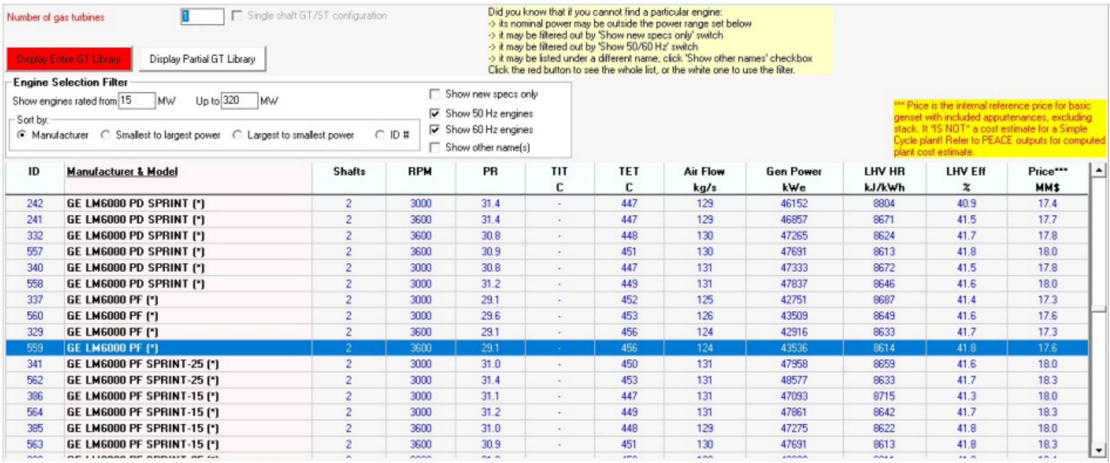


Figure 3.3.2: Selection of GE LM6000 gas turbine model in GT PRO

Afterwards, the plant criteria are defined; the most important parameter to choose are the design ambient temperature, the ambient pressure and the type of plant: since no co-generation is considered, the plant is gas turbine only. For the purpose of example, in Figure 3.3.3 is the representation of the choice of the plant criteria for GE LM2500.

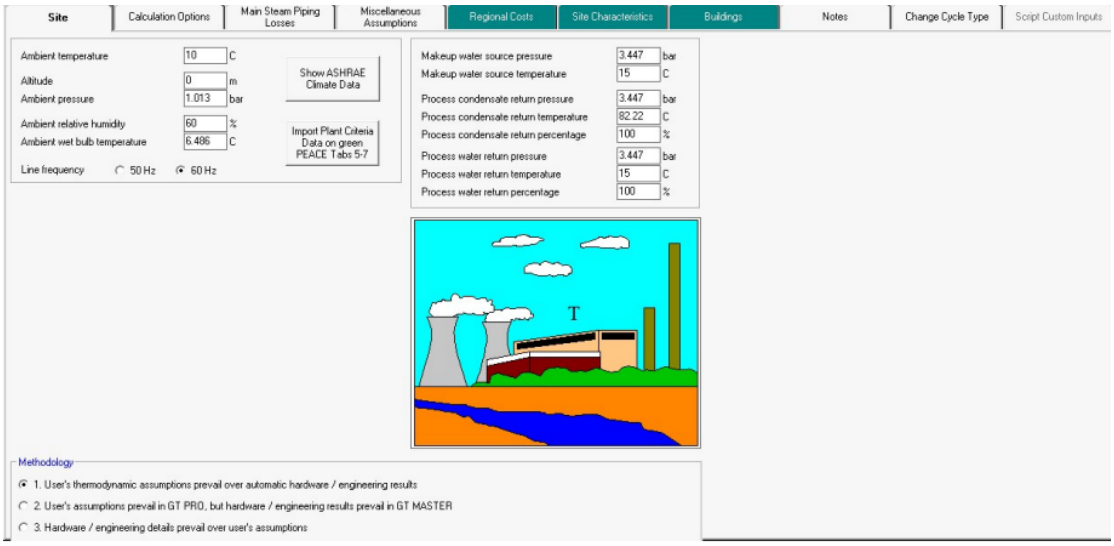


Figure 3.3.3: Selection of plant criteria for GE LM2500 in GT PRO

Then inputs must be defined, such as pressure losses at intake and exhaust, and fuel conditions (composition and pressure). In Figure 3.3.4 is shown as example the input definition interface for the GE LM2500 turbine.

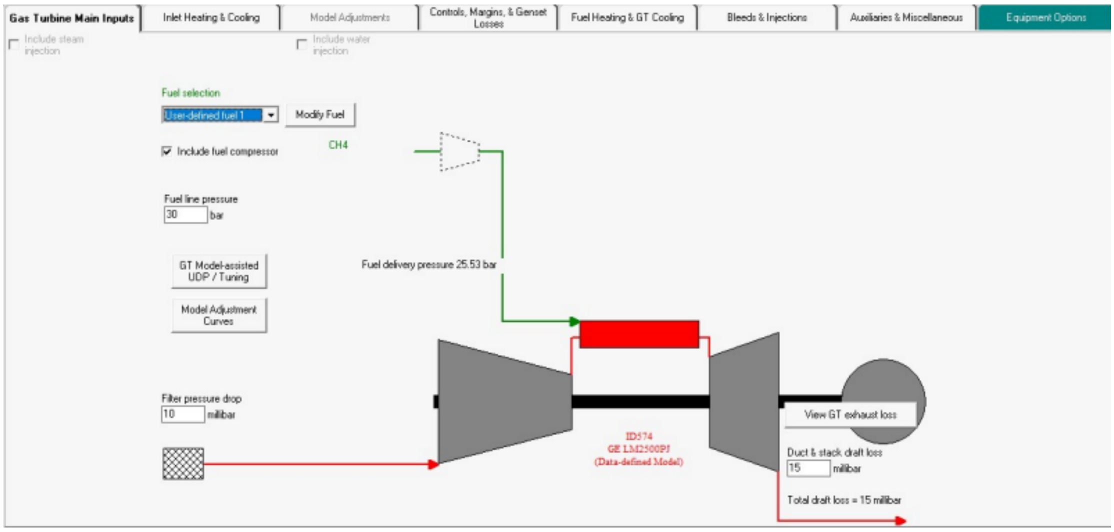


Figure 3.3.4: Definition of input parameters(losses and fuel) for GE LM2500 in GT PRO

For the choice of fuel composition, in particular, a generic natural gas fuel is created for the two turbines, since, for the purposes of model design, reference is made to the fact that part of the extracted natural gas is used at Equinor to operate the turbines. In table 3.3.1 compositions expressed in volume percentage are shown, the same for the two turbines.

Fuel	**Chemical Formula**	**Volume %**
Nitrogen	$N2$	0.83
Carbon Dioxide	CO_2	0.41
Methane	CH_4	93.74
Ethane	C_2H_6	3.73
Propane	C_3H_8	0.57
n-Butane	C_4H_{10}	0.16
n-Pentane	C_5H_{12}	0.23
Isobutane	C_4H_{10}	0.33
Total		100

Table 3.3.1: Fuel composition in volume percentage in Thermoflow©

In tables 3.3.2 and 3.3.3 values taken as input for the model in Python are shown, respectively for the gas turbine GE LM2500 and gas turbine GE LM6000.

$\dot{m}_{intake}$ kg/s	$losses_{intake}$ $mbar$	$losses_{exhaust}$ $mbar$	$\dot{m}_{fuel}$ kg/s	$leakage$ kg/s	η_{mech} $\%$	η_{gen} $\%$
66	10	15	1.26	0.58	98.95	97.53

Table 3.3.2: Model input data taken from Thermoflow© for GE LM2500 gas turbine

$\dot{m}_{intake}$ kg/s	$losses_{intake}$ $mbar$	$losses_{exhaust}$ $mbar$	$\dot{m}_{fuel}$ kg/s	$leakage$ kg/s	η_{mech} $\%$	η_{gen} $\%$
129	10	12.45	2.23	1.52	99.22	98.21

Table 3.3.3: Model input data taken from Thermoflow© for GE LM6000 gas turbine

The composition of the air is also taken as input to the model by Thermoflow©, and is considered the same for the two turbines. The values are shown in the table 3.3.4.

N_2 $\%$	O_2 $\%$	CO_2 $\%$	H_2O $\%$	Ar $\%$
74.54	13.51	3.33	7.7	0.897

Table 3.3.4: Air composition data in volume percentage taken from Thermoflow©

Data from Thermoflow© are also taken to verify the quality of calculation of the model in Python, which are listed for the two turbines in the tables 3.3.5 and 3.3.6.

TET $°C$	$Power$ kW	$HeatRate$ kJ/kWh	η_{mech} $\%$
543	21958	10033	35.88

Table 3.3.5: Model output data taken from Thermoflow© for GE LM2500 gas turbine to be compared with the Python model

TET **°C**	$Power$ kW	$HeatRate$ kJ/kWh	η_{mech} %
453	45199	8610	41.8

Table 3.3.6: Model output data taken from Thermoflow© for GE LM6000 gas turbine to be compared with the Python model

However, in order to be able to proceed with the design of the model in Python, important and fundamental parameters must be defined: the polytropic and/or isentropic efficiencies of the components. These parameters are unknown outside the manufacturer, and cannot even be defined using Thermoflow©, as compressor maps are required.

These parameters are not negligible, indeed, they must be identified as correctly and reliably as possible not only because the design modeling is impossible without the definition of the efficiencies, but also because, for the purpose of condition monitoring and predictive maintenance, they constitute the baseline of the non-degraded condition with which to compare the actual field data. In the following section, an iterative procedure in GasTurb© for identifying efficiencies is presented.

3.3.2 GasTurb© model for efficiency

GasTurb© is a gas turbine performance calculation and optimization program. It simulates most of the gas turbine configurations in use for propulsion or for power generation [33]. It is made of different software configurations, and for the purposes of the finding of components efficiency, a cycle design is implemented: in the gas turbine design process, in fact, many alternative thermodynamic cycles are evaluated, whose the cycle reference point (or design point) is chosen.

First, the correct geometric and mechanical configuration is selected for the turbines, shown in the figures below. In Figure 3.3.5 the white shaft is the one the two compressors and the high pressure turbine are coupled to, while the black shaft is the power shaft, to which the power turbine, and so the generator, are coupled.

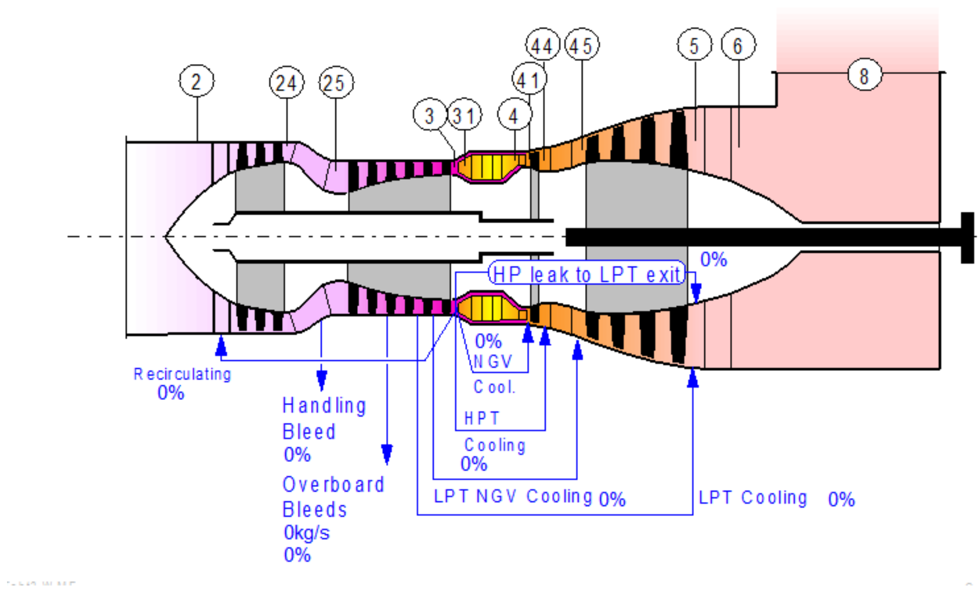


Figure 3.3.5: GE LM2500 turbine selected configuration in GasTurb©

In Figure 3.3.6 instead, is the inner-shaft configuration of the gas turbine GE LM6000, with the black shaft being the power shaft (to which LPC and LPT are coupled) and the white shaft being the HP inner shaft.

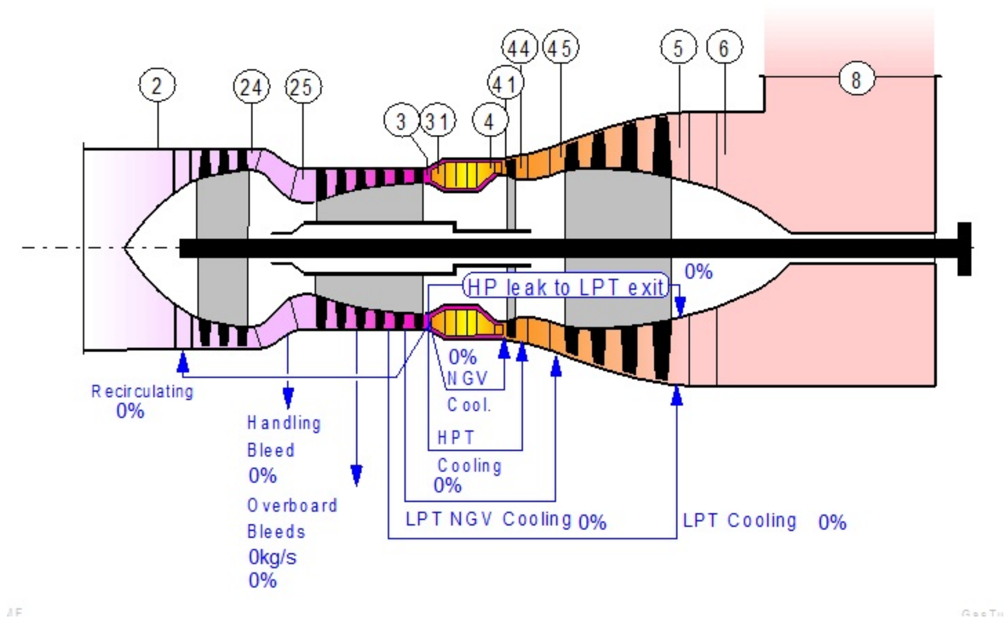


Figure 3.3.6: GE LM6000 turbine selected configuration in GasTurb©

In order to define the cycle design point, the model in GasTurb© requires the following data as input, taken from Thermoflow©:

- compression ratio;
- pressure losses at the intake;

- pressure losses at the exhaust;
- LHV of the fuel;
- ambient temperature and pressure

Therefore, the application has to be selected: since the purpose of the turbines is the power generation, the application is *turboshaft* instead of *turbopropeller*. The last step to do is the initialisation of the efficiency values for turbines and compressors: following the values for efficiencies given in [11], compressors are initialised with a polytropic efficiency of 0.9 and turbines with a polytropic efficiency of 0.85: these values are only used to start the calculation iterations. As an example, Figure 3.3.7 shows the GasTurb© interface for the modeling of the GE LM6000 turbine.

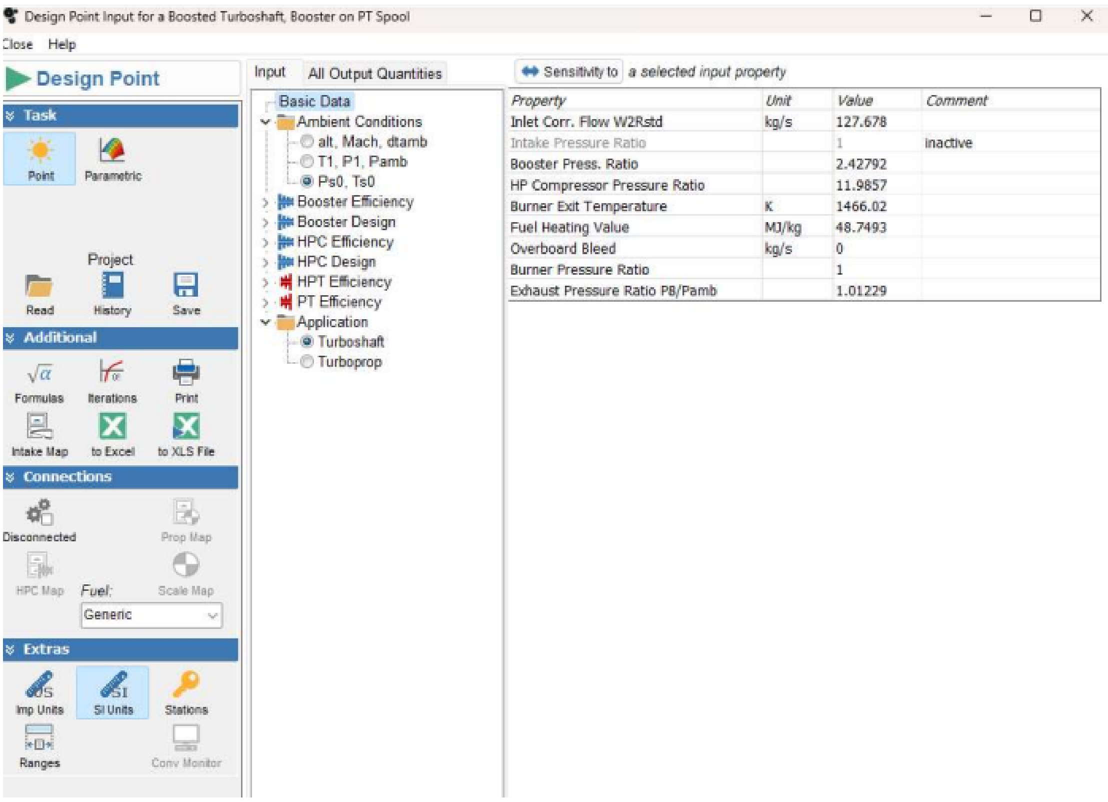


Figure 3.3.7: Input parameter interface for GE LM6000

In order to identify the design point, and thus the values of the component efficiencies, once the main input data have been modelled, an iterative procedure is implemented, which relates target values (such as the power output) to variable values (the values of the efficiencies, which are, in fact, variable, as they are to be identified). In GasTurb©, in particular, the correlation between target and variable must be one-to-one, so the procedure of assigning to a given target the corresponding variable efficiency is not trivial, and is done iteratively.

The target values are taken from Thermoflow©, and are the parameters and are the parameters describing the turbine, and are listed below:

- power output;
- exhaust gas temperature;
- heat rate;
- gas turbine efficiency

Variables, on the other hand:

- LP efficiency;
- HP efficiency;
- LP turbine;
- HP turbine

The first correlation identified is the correlation between exhaust gas temperature and low-pressure turbine efficiency, as these are directly and thermodynamically related parameters. It is then seen that the power correlated with the efficiency of the high-pressure compressor completes the iterations positively. As for the other two efficiencies, they are identified by GasTurb© without a specific correlation, as none is found to lead the iteration to be completed correctly.

In Figures 3.3.8 is brought as example the iteration procedure for the gas turbine GE LM6000, that shows the relation between target and variables. As it can be seen, also fuel flow rate (correlated with the temperature at the outlet of the combustion chamber) and the exhaust flow rate are considered in the iterations, since they are parameters to be defined.

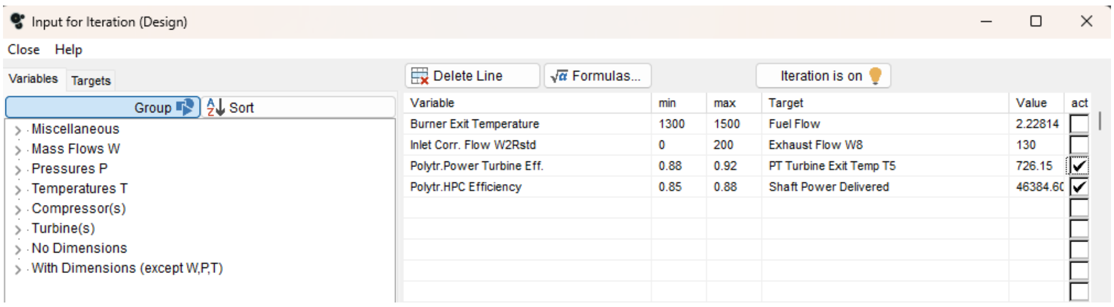


Figure 3.3.8: GasTurb© iteration procedure interface for GE LM6000

The interfaces of the results obtained by GasTurb© are shown below for the GE LM2500 turbine 3.3.9 and the GE LM6000 turbine 3.3.10: the values taken as input for the model are the polytropic and isentropic efficiencies of the components.

```
Boosted Turboshaft
Alt=     0m   ISA + 5 C    60% Relative Humidity


          W          T          P          WRstd
Station  kg/s        K         kPa         kg/s        PWSD      =    22752.0 kW
 amb                293.15    101.325
  1      65.510     293.15    101.325                  PSFC      =     0.1949 kg/(kW*h)
  2      65.510     293.15    100.325     66.913       V0        =       0.00 m/s
 24      65.510     412.64    296.576     26.855       P25/P24   =    1.00000
 25      65.510     412.64    296.576     26.855       P3/P2     =      18.00
  3      65.510     720.57   1805.844      5.828       FN res    =       5.53 kN
 31      65.510     720.57   1805.844                  Heat Rate=     9501.3 kJ/(kW*h)
  4      66.742    1456.87   1805.844      8.442       WF        =    1.23177 kg/s
 41      66.742    1456.87   1805.844      8.442       Loading   =     100.00 %
 43      66.742    1102.39    444.352                  s NOx     =     0.5271
 44      66.742    1102.39    444.352                  Therm Eff=    0.37890
 45      66.742    1102.39    444.352     29.844       P45/P44   =    1.00000
 49      66.742     812.05    102.825                  P6/P5     =    1.00000
  5      66.742     812.05    102.825    110.691
  6      66.742     812.05    102.825                  A8        =    1.85468 m²
  8      66.742     812.05    102.825    110.691       P8/Pamb   =    1.01480
Bleed     0.000     720.57   1805.841                  WBld/W2   =    0.00000
--------------------------------------------           P2/P1     =    0.99013
Ps0-P2=  1.000     Ps8-Ps0=    0.000                   Ps8       =    101.325 kPa
Efficiencies:    isentr  polytr    RNI      P/P
 Booster         0.8838  0.8999  0.970    2.956        driven by HPT
 Compressor      0.8617  0.8905  1.908    6.089        WCHN/W25 =    0.00000
 Burner          1.0000                   1.000        WCHR/W25 =    0.00000
 HP Turbine      0.8703  0.8500  2.683    4.064        e444 th   =    0.87028
 LP Turbine      0.8710  0.8486  0.908    4.321        eta t-s   =    0.86380
 Generator       1.0000                                PW gen    =    22752.0 kW
--------------------------------------------           TRQ       =     100.00 %
HP Spool mech Eff 1.0000  Nom Spd  34000 rpm           WCLN/W25 =    0.00000
PT Spool mech Eff 1.0000  Nom Spd  10000 rpm           WCLR/W25 =    0.00000
--------------------------------------------
hum [%]     war0        FHV      Fuel
  60.0    0.00886     48.749    Generic
```

Figure 3.3.9: GasTurb© output interface for GE LM2500

```
Boosted Turboshaft
Alt=     0m   ISA -5 C    60% Relative Humidity


             W          T          P         WRstd
Station   kg/s          K        kPa         kg/s        PWSD       =   46380.9 kW
 amb                 283.15    101.325
  1     127.352     283.15    101.325                    PSFC       =    0.1717 kg/(kW*h)
  2     127.352     283.15    100.325     127.678        V0         =      0.00 m/s
 24     127.352     377.26    243.581      60.701        P25/P24    =   1.00000
 25     127.352     377.26    243.581      60.701        P3/P2      =     29.10
  3     127.352     788.67   2919.484       7.323        FN res     =      9.27 kN
 31     127.352     788.67   2919.484                    Heat Rate=     8369.6 kJ/(kW*h)
  4     129.564    1466.02   2919.484      10.156        WF         =   2.21193 kg/s
 41     129.564    1466.02   2919.484      10.156        Loading    =    100.00 %
 43     129.564    1119.08    761.839                    s NOx      =    0.9140
 44     129.564    1119.08    761.839                    Therm Eff=    0.43013
 45     129.564    1119.08    761.839      34.005        P45/P44    =   1.00000
 49     129.564     729.51    102.570                    P6/P5      =   1.00000
  5     129.564     729.51    102.570     203.922
  6     129.564     729.51    102.570                    A8         =   3.73906 m²
  8     129.564     729.51    102.570     203.922        P8/Pamb    =   1.01229
Bleed     0.000     788.67   2919.483                    WBld/W2    =   0.00000
--------------------------------------------              P2/P1      =   0.99013
Ps0-P2=  1.000     Ps8-Ps0=    0.000                     Ps8        =   101.325 kPa
Efficiencies:    isentr   polytr     RNI     P/P
 Booster         0.8643   0.8800   1.011   2.428         driven by PT
 Compressor      0.8973   0.9250   1.744  11.986         WCHN/W25 =    0.00000
 Burner          1.0000                    1.000         WCHR/W25 =    0.00000
 HP Turbine      0.8756   0.8568   4.307   3.832         e444 th    =   0.87556
 LP Turbine      0.8860   0.8568   1.530   7.428         eta t-s    =   0.88186
 Generator       1.0000                                  PW gen     =   46380.9 kW
--------------------------------------------              TRQ        =    100.00 %
HP Spool mech Eff 1.0000   Nom Spd  60000 rpm            WCLN/W25 =    0.00000
LP Spool mech Eff 1.0000   Nom Spd  34000 rpm            WCLR/W25 =    0.00000
--------------------------------------------
hum [%]     war0       FHV      Fuel
  60.0    0.00462    48.749    Generic
```

Figure 3.3.10: GasTurb© output interface for GE LM6000

For a clearer view of the results, the efficiencies are shown in the tables 3.3.7 and 3.3.8.

	LPC	HPC	HPT	LPT
$\eta_{isentropic}$	88.38	86.17	87.03	87.1
$\eta_{polytropic}$	89.99	89.05	85	84.86

Table 3.3.7: Efficiency for GE LM2500 gas turbine

	LPC	HPC	HPT	LPT
$\eta_{isentropic}$	86.43	89.73	87.56	88.6
$\eta_{polytropic}$	88	92.5	85.68	85.68

Table 3.3.8: Efficiency for GE LM6000 gas turbine

3.3.3 GE LM2500 turbine off-design

The simplifications adopted for the off-design model of the case study seems to be approximate and not adoptable to describe a complex off-design model such as that of the aeroderivative turbines under study, in particular the gas turbine LM2500, the one taken into account in this section. In the simplified case, in fact, the corrected flow rate is considered constant in varying operating conditions, as described by the formula 2.22: in truth, as illustrated in the section 2.1.2.1, the relationship between corrected flow rate in design conditions and in off-design conditions may consider a quantity directly proportional to the variation of the IGVs angle at the compressor intake.

In Figure 3.3.11 is it possible to precisely see how as the ambient temperature decreases, there is an increasing difference between the air flow rate returned by Thermoflow© and the air flow rate calculated considering a constant corrected flow rate, according to the formula $\dot{m_1} = \dot{m_{1,ref}}\sqrt{\frac{T_{1,ref}}{T_1}\frac{P_1}{P_{1,ref}}}$.

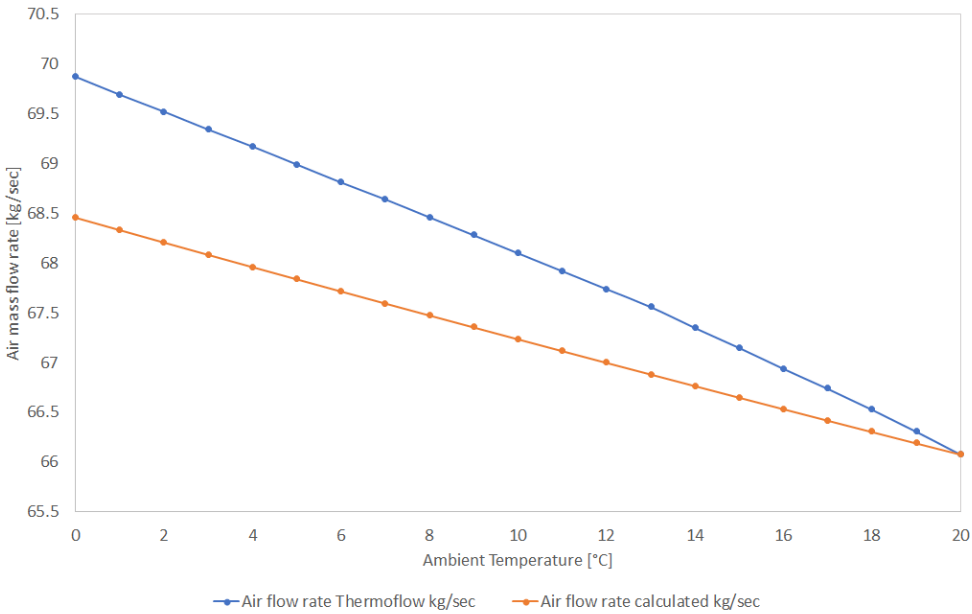


Figure 3.3.11: Comparison of the air flow rate between Thermoflow©w and Python for the GE LM2500, not considering the effect of the IGVs

But the actual question that arises is whether there is an actual correlation between the variation in ambient temperature and the variation in the opening of IGVs, especially in full-load. In the literature there some research that relates the ambient temperature and the opening of the IGVs: for example, in [34] it is explained how, when the IGV opening is constant and the inlet air temperature is lower than the optimal value, as the inlet air temperature decreases, the GT efficiency decreases. In fact, the change in opening angle of the IGVs is an important parameter for regulating and optimising the performance of the gas turbine. However, the relationship between the variation of ambient temperature and the variation of the angle of the IGVs is known and studied in the case of part-load performance, and it is unclear whether the variation of the IGVs can find application sense in the case of full load, as the ambient temperature changes. In addition, no specific information is available from the manufacturer (General Electric) on how off-design tests are carried out for gas turbines.

Although there is no more precise information on how the off-design performance actually occurs as the ambient temperature changes, in light of the relationship between the air flow rate in Thermoflow© and the air flow rate calculated considering the constant corrected flow rate, it is assumed that, for the GE LM2500 gas turbine, the off-design model takes into account a possible opening of the IGVs as the ambient temperature decreases compared to the design temperature.

With this in mind, an initial simplified off-design model for the LM2500 turbine is roughly developed, considering an off-design model of a turbine with similar mechanics, described in [12]. This model considers how air flow, compression ratio and efficiency of the compressor/turbine component varies not only with varying ambient conditions but also with varying vanes angle.

In 2.1.2.1 section, the coefficient $CIGV$, directly proportional to the opening of the IGVs, is defined as the ratio between the corrected flow rate in off-design and the corrected flow rate in design:

$$CIGV = \frac{\mu_{OD}}{\mu_D} \tag{3.1}$$

where the subscrit OD stands for *off-design*, while the subscript D stands for *design*.

As it is demonstrated in [11], the coefficient $CIGV$ that relates the value of the corrected flow rate in off-design and the corrected flow rate in design is almost the same as the coefficient $CIGV$ implemented for the change of the compression ratio: therefore, by deriving this coefficient from the ratio between the corrected flow rate in off-design (obtained from the values in Thermoflow©) and the corrected flow rate in design, the new values of the compression ratio are obtained as the ambient temperature varies.

As ambient temperature decreases and air flow increases, also a reduction in compressor and turbine efficiency occurs. The turbine, in fact, is built on the same concept, except for the fact that instead of having IGVs, NGVs are modeled (noozle guide vanes), whose opening is modified to allow more or less flow of hot gases to pass through: since a full load case is considered, the fuel flow rate and air flow rate increase as the ambient temperature decreases, so, for simplification, for the NGVs is considered a similar behaviour and opening as the IGVs.

Therefore, the opening of the NGVs reduces the turbine stage efficiency, due to the increasing aerodynamic losses in the turbine, as the opening of the IGVs reduces the compressor stage efficiency.
In the Figures 3.3.12 and 3.3.13 is it possible to see, in fact, the reduction in compressor and turbine efficiency as the opening angle of the IGVs changes.

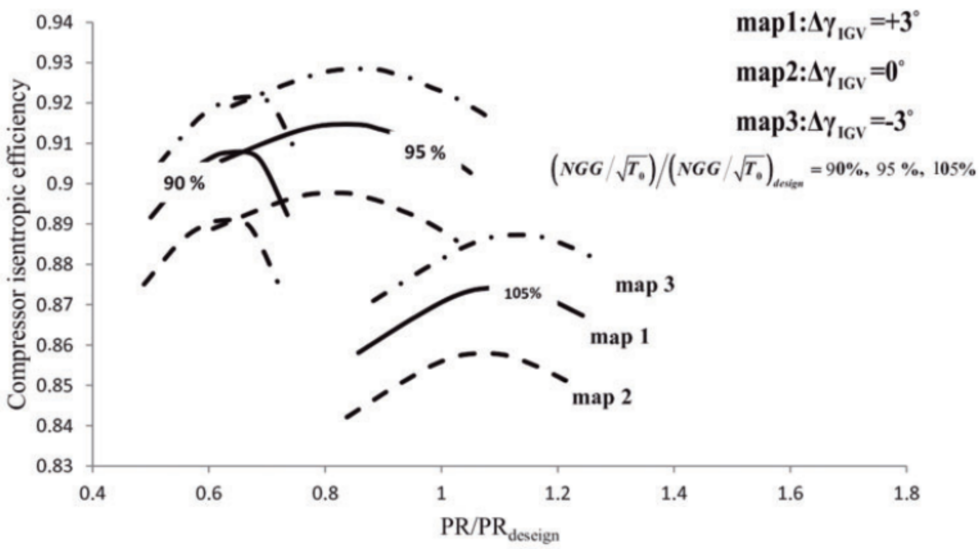


Figure 3.3.12: Change in compressor efficiency by changing the angle of the IGVs [12]

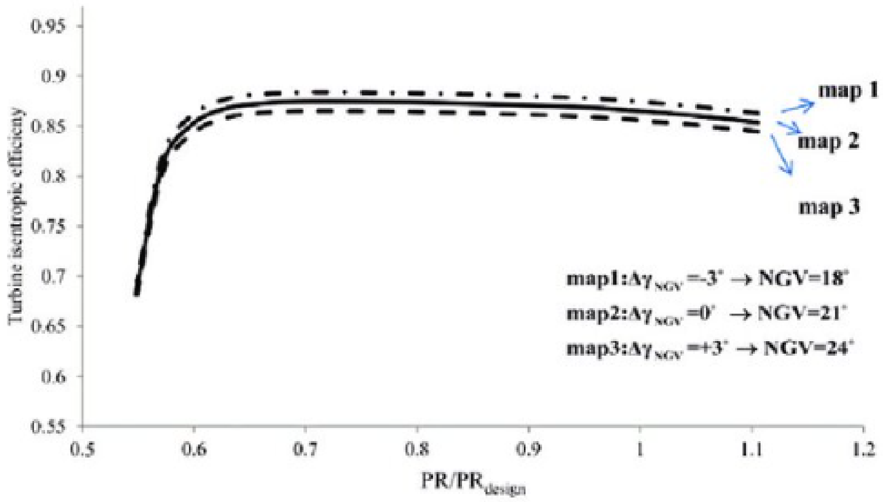


Figure 3.3.13: Change in turbine efficiency by changing the angle of the IGVs [12]

Following this logic, therefore, the first draft of the off-design model for the GE LM2500 gas turbine is implemented, taking the following data as input from Thermoflow©:

Input Values from Thermoflow©
Air Flow Rate
Fuel Flow Rate
Intake Pressure Drops
Exhaust Pressure Drops
Air Leakage

Table 3.3.9: Values taken from Thermoflow© as inputs to the model in Python for GE LM2500 gas turbine

3.4 Performance Indicators

As highlighted in the section 2.3, one of the most significant uses of the digital twin lies in the implementation of predictive maintenance and condition monitoring strategies. In order to implement a mathematical and analytical model for identifying component degradation according to the model-based, (or physics-based) strategy, the approach outlined in [27] is followed, in which the authors define performance indicators (or degradation indices), relating the gas turbine current operating conditions, the design conditions and the expected operating point according to the actual ambient parameters.

In this sense, the digital twin would provide the information related to the design operating point and the expected operating point under specific ambient conditions, while the sensors would provide information related to the actual operating conditions. As can be guessed, developing such an index would require detailed and specific information from both the sensors collecting the actual data and the accurate predictions of the digital twin: regarding the field data, the most accurate information provided refers to the GE LM6000 turbine, while the model developed even in off-design conditions is of the GE LM2500 turbine, whose data from the field is not adequately significant to be applied. For these reasons, a simplified performance indicator model applicable to the GE LM6000 turbine low-pressure compressor is proposed.

The idea behind the model is to identify a possible variation in the polytropic efficiency of the low-pressure compressor compared to the nominal conditions identified during the design of the model in design.
The equation 2.20 is the equation referred to in this simplified model, which relates the temperature and pressure at the inlet and outlet of the compressor, also shown below for ease of reading:

$$\frac{T_2}{T_1} = \left(\frac{P_2}{P_1}\right)^{\frac{\gamma-1}{\gamma\eta_{p,C}}} \tag{3.2}$$

The ambient pressure and the ambient temperature are taken from the field data to determine the real properties of the air at the compressor inlet.

The compressor inlet temperature and the compressor outlet pressure are also taken from the field, while the reference polytropic efficiency is the polytropic efficiency in design conditions: taking the design polytropic efficiency as a reference and considering it constant as the ambient temperature changes, is obviously a simplification, since, as also demonstrated for the GE LM2500 turbine, in operating conditions other than design, the efficiency may vary.

The aim is to calculate the difference between the calculated temperature by the digital twin at the compressor outlet and compare it with the temperature from the field data, considering the outlet temperature at the compressor calculated by the digital twin in hypothetical 'clean' conditions.
The performance indicator is thus developed this way, it is shown in 3.5 and it is normalised by the temperature calculated by the model. Since it is dependent on the temperature scale, the temperature is converted as temperature difference from a reference value, i.e. ambient temperature T_{amb}:

$$\Delta T_{2,field} = T_{2,field} - T_{amb} \tag{3.3}$$

$$\Delta T_{2,model} = T_{2,model} - T_{amb} \tag{3.4}$$

$$\frac{\Delta T_{2,field} - \Delta T_{2,model}}{\Delta T_{2,model}} \tag{3.5}$$

Taking the principles of thermodynamics into account, if this index has a positive value, then it means an higher temperature than the nominal one calculated by the model at the same compression ratio, which consequently points out a reduction of the compressor polytripic efficiency.

A first hypothesis, which would go back to the component degradation topic, could be that the compressor is using more power to compress air at the predefined pressure than it would if the polytropic efficiency were equal to the nominal one. Naturally, as also mentioned above, this is only a hypothesis, as the change in efficiency could also be traced back to a change in ambient conditions.
The results will be discussed in chapter 4.

CHAPTER
FOUR

RESULTS AND DISCUSSION

In the first part of this section, the results of the Python and Aspen HYSYS® model comparison for the single-shaft gas turbine case study in design and off-design conditions will be shown. The purpose of the comparison is to validate the computational quality of the code methods in Python, and the results are analysed in terms of temperature, power, efficiency, fluid molar composition, etc. Next, the results of the Python model in design for the GE LM2500 and GE LM6000 turbines will be shown, compared to the output in Thermoflow©. The off-design results for the GE LM2500 turbine model will also be highlighted. Finally, the results of the analysis of real data from the field for the implementation of the performance indicator for condition monitoring in predictive maintenance will be presented.

4.1 Comparison with Aspen HYSYS®

As illustrated in section 3.2.1, two comparative models are designed and built in Python and Aspen HYSYS®, in order to validate the methods of the class-based model in Python and the accuracy of the calculation of thermodynamic functions.

4.1.1 Sensitivity analysis for power and temperature

Paragraph 3.1.2 illustrates the calculating method for the temperature and the other thermodynamic properties of the stream after the compression and expansion processes, using a step iterative procedure, which takes into account the smallest variations of the specific heat.

A sensitivity analysis is performed to show how the temperature and power of compressor and turbine change by changing the number of iterations, keeping the temperature and power values from the simulation in Aspen HYSYS® as the ideal baseline. For the temperatures, the T is referred to the difference between the temperature considered and the ambient temperature. In Figures 4.1.1 and 4.1.2 is shown the sensitivity analysis for the temperature out of the compressor and the compressor duty.

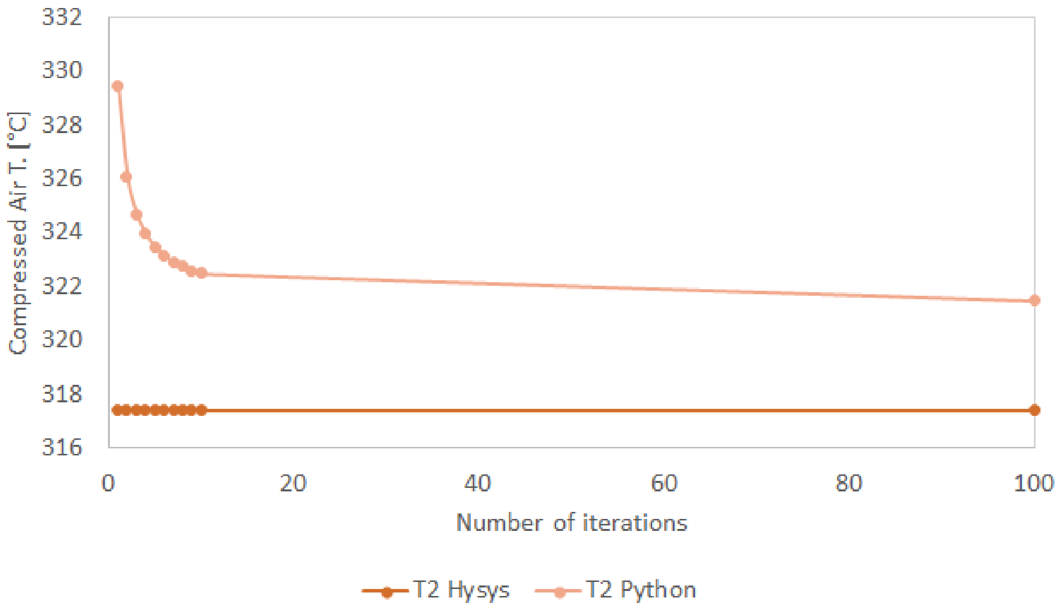


Figure 4.1.1: Sensitivity Analysis of the $T2$ by changing the number of iterations compared to the baseline simulation

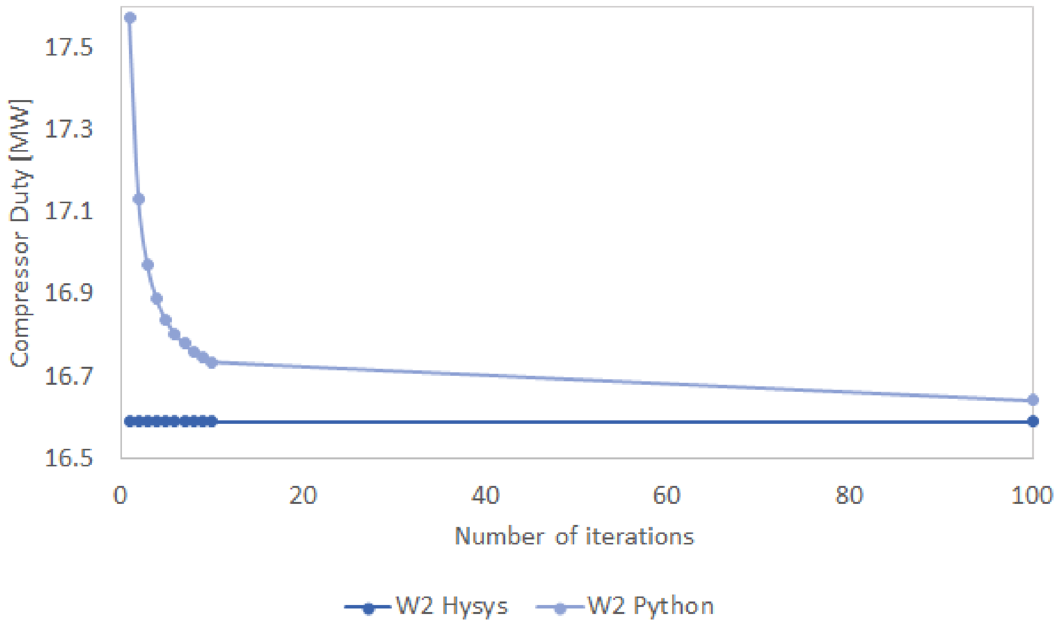


Figure 4.1.2: Sensitivity Analysis of the $W2$ by changing the number of iterations compared to the baseline simulation

In order to better show the improvement of the results, a histogram of the variation in the error percentage is shown in Figure 4.1.3. The percentage error is calculated as follows for temperature and duty, where W is for *work* in a generic way.

$$\frac{|(T_{2,Python} - T_{ambient}) - (T_{2,HYSYS} - T_{ambient})|}{(T_{2,Hysys} - T_{ambient})} \tag{4.1}$$

$$\frac{|W_{2,Python} - W_{2,HYSYS}|}{W_{2,Hysys}} \tag{4.2}$$

Even though the temperature in Python does not exactly converge to the temperature in HYSYS®, the percentage error is under 1.5%, with difference of about 4 degrees, leading to the conclusion of a good result.

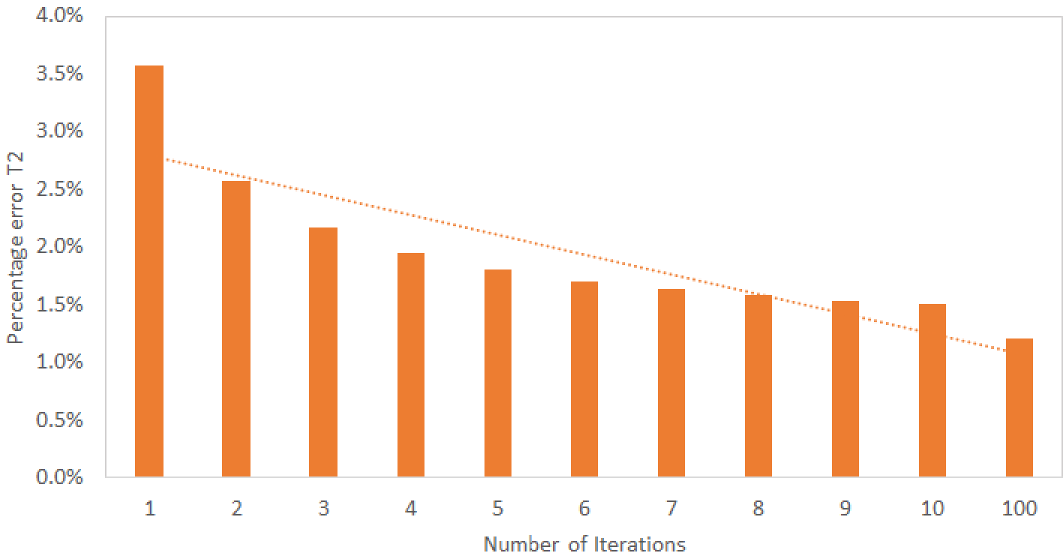


Figure 4.1.3: Percentage error of $T2$ by changing the number of iterations

The duty, on the other hand, is more is more positively affected by the number of iterations, and therefore by the change of the specific heat, as can be seen by the greater slope of the trend line in the percentage error histogram. Leading to at almost 0% error, the work calculation also produces good results.

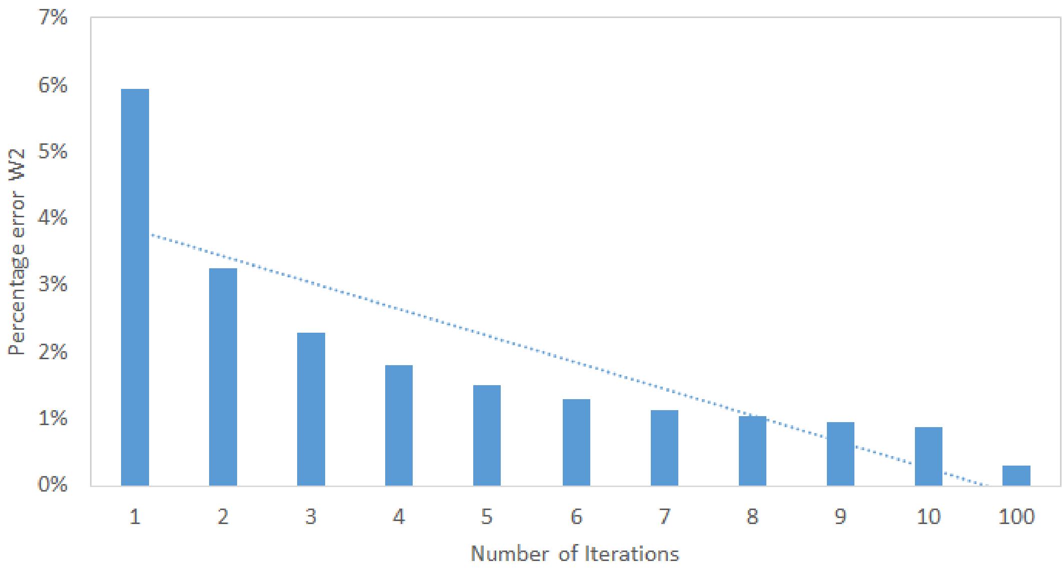


Figure 4.1.4: Percentage error of $W2$ by changing the number of iterations

As it is done for the compressor, a similar sensitivity analysis is conducted for the turbine. In Figures 4.1.5 and 4.1.6 the sensitivity analysis by changing the number of iterations is shown, respectively for temperature and power.

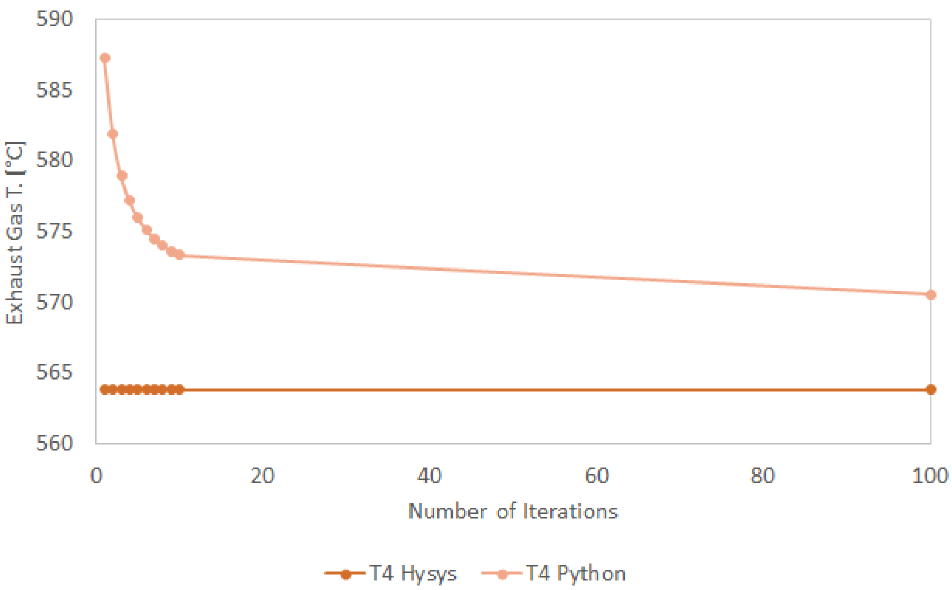


Figure 4.1.5: Sensitivity Analysis of the $T4$ by changing the number of iterations compared to the baseline simulation

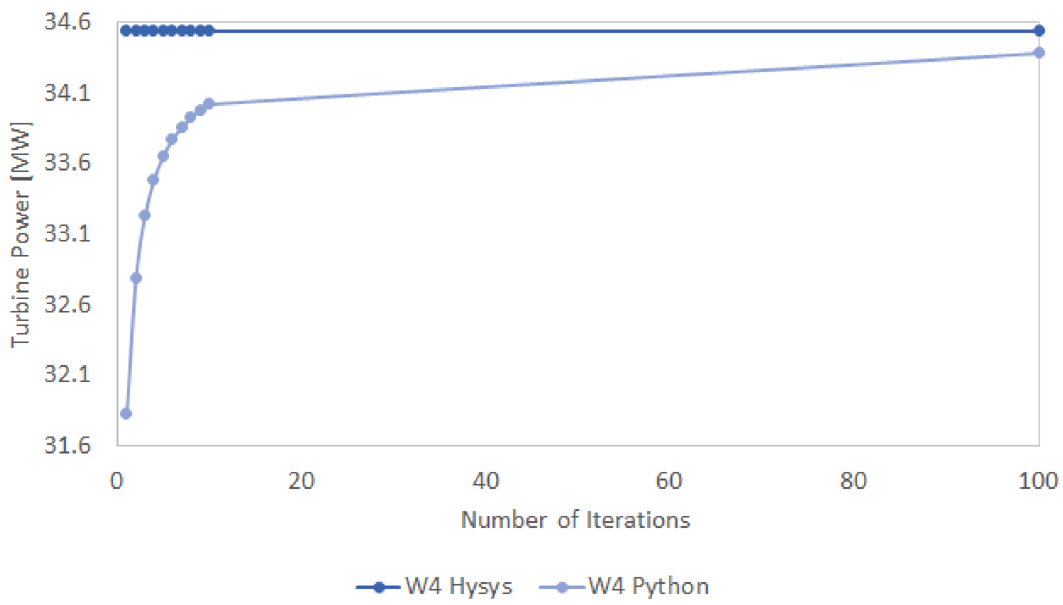


Figure 4.1.6: Sensitivity Analysis of $W4$ by changing the number of iterations compared to the simulation

The trend of the turbine power compared to the baseline defined with Aspen HYSYS® is perfectly consistent with what is expected: as the number of iterations is reduced, the calculation of the temperature is more approximate, and the resulting value of the temperature is higher than it should be, so the resulting power is lower.

In Figures 4.1.7 and 4.1.8 the percentage error histograms, for temperature and power, calculated in the same way as the compressor (refer to eq. 4.1 and 4.2. As it is detected for the compressor, the power is more positively affected by the change of the number of iterations, since the slope of the trend line is greater. Overall, the percentage errors, less than 1.5% for the temperature and almost 0% for the power, lead to a validation of the iterative procedure for calculating temperature and power for the turbine as well.

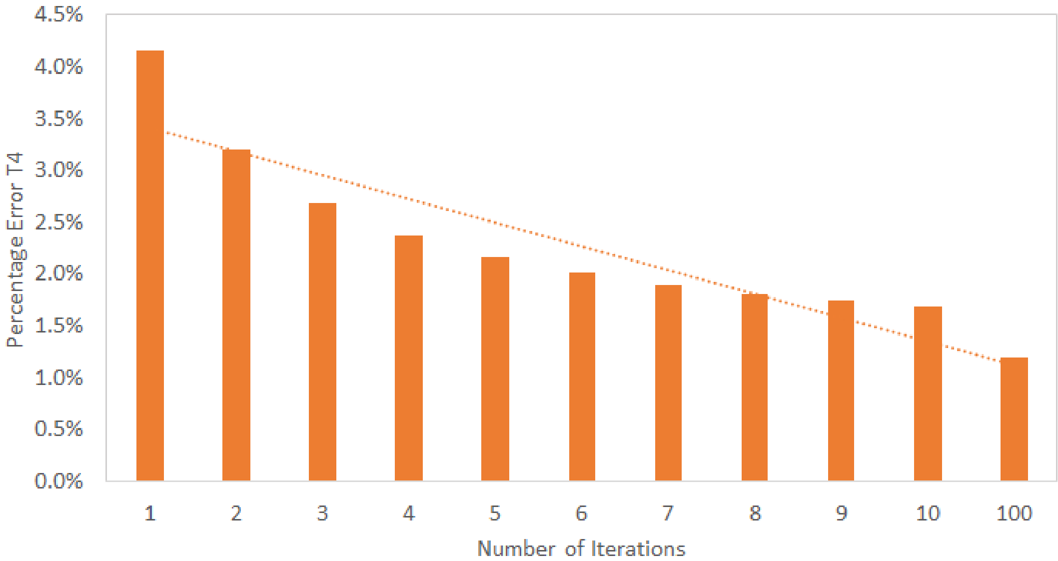


Figure 4.1.7: Percentage error of $T4$ by changing the number of iterations

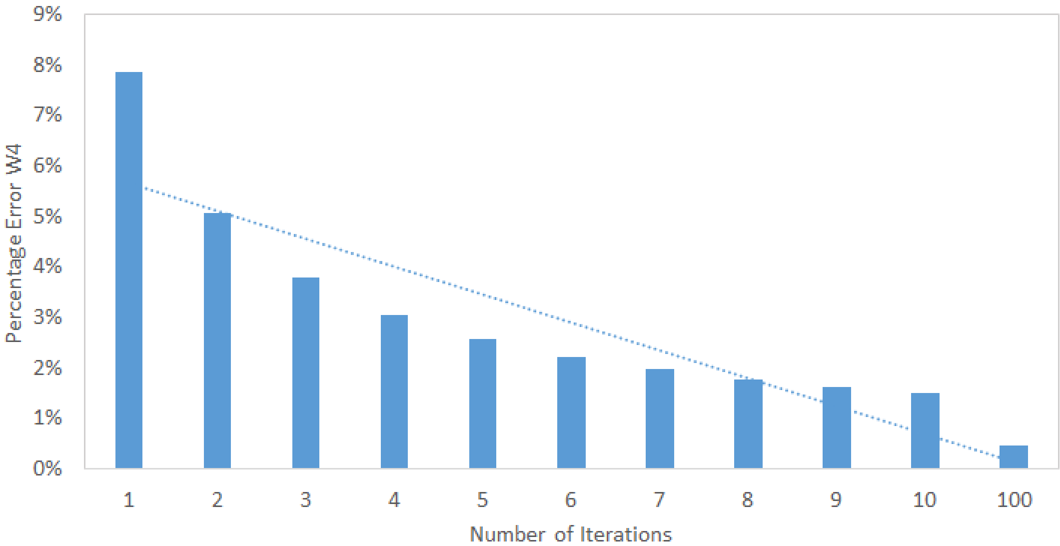


Figure 4.1.8: Percentage error of $W4$ by changing the number of iterations

4.1.2 Case study: design

Section 3.2 illustrates the reference case study for comparing the calculation results between Python and Aspen HYSYS® of the single-shaft gas turbine. Below is the table illustrating the results obtained, and the percentage error:

Variable	Unit	HYSYS®	Python	Error	% Error
Compressed Air T	°C	337.4	341.4	4.0	1.21
Compressor Duty	kW	16590	16641.5	51.5	0.31
Compressor η_{is}	%	83	82.37	0.63	
TIT (reactions)	°C	1135	1136.1	1.18	0.10
TIT (no reactions)	°C	1135	1221.4	86.4	7.08
Exhaust Gas T	°C	583.8	590.5	6.77	1.15
Turbine Power	kW	34540	34381.9	158.04	0.46
Turbine η_{is}	%	92	92.18	0.18	
LHV methane	kJ/kg	50030	50028	1.95	0.0
GT Net Power	kW	17950	17740.3	209.6	1.17
GT efficiency	%	35.88	35.46	0.42	

Table 4.1.1: Case study design model: results

The first consideration taken into account is that the temperature out of the combustion chamber (*TIT* is for *turbine inlet temperature*) can be calculated in two ways: either by considering the chemical reactions between the fuel and the air and, consequently, considering the combustion fluid made of the products of the reaction, which also affects the specific heat; or by not considering the chemical reactions and simplifying the fluid out of the combustion chamber as just air, not made of the products of the reaction (and, consequently, a reduced specific heat).

As it can be seen from the results in the table 4.1.1, the use of the chemical reactions calculation method is justified by the improvement of the temperature inlet turbine, in terms of reduction in the deviation from the value in the simulation baseline. For a more detailed discussion about the TIT and the combustion process, refers to the 4.1.3. Overall, with the exception of the TIT calculated without considering chemical reactions, the results from the design analysis can be considered good, settling for each variable below 1.5%. For a more impactful insight into the data, refer to the Figure 4.1.9, representing the percentage error for each variable being studied.

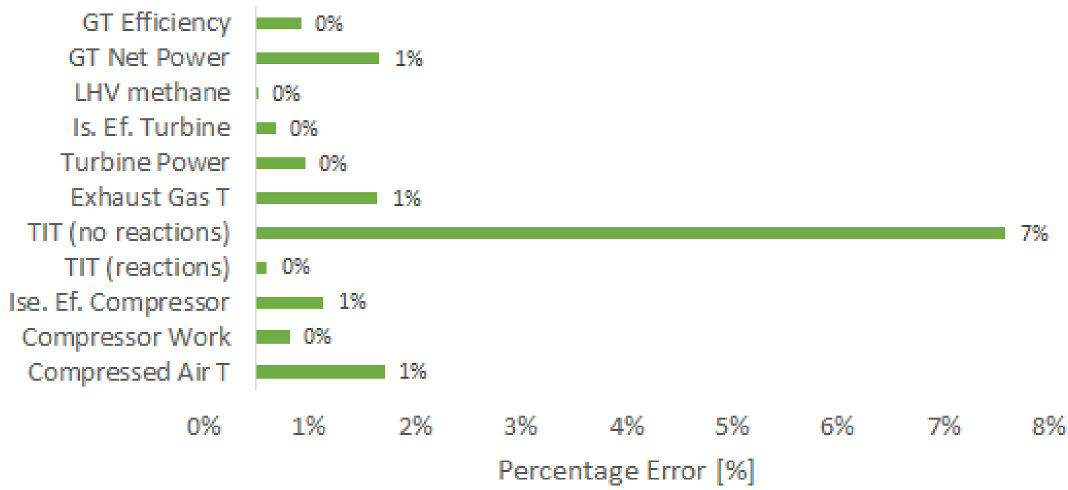


Figure 4.1.9: Percentage deviation of the variables being studied in the case study design model

The mole fractions of the fluid made of combustion products, compared with the HYSYS® mole fractions, are also shown in the table 4.1.2, proving the calculation efficiency of the combustion chamber chemical reaction method.

	N_2	O_2	CO_2	H_2O
HYSYS®	0.7723	0.1237	0.0347	0.0693
Python	0.7784	0.1204	0.0337	0.0675
Difference	0.0061	0.0033	0.0010	0.0018
%Error	0.79	2.68	2.78	2.64

Table 4.1.2: Combustion fluid compositions in mole fraction

4.1.3 Temperature inlet turbine analysis

This section is concerned with combustion analysis, giving examples of calculation results with different fuels, and comparing temperature values calculated using the chemical and non-chemical reaction methods.

For the development of this analysis, the compressor, turbine and stream are modelled with the same values as defined for the design conditions (refer to 3.2), while the different fuels entering the combustion chamber are tabled in 3.2.1, with their respective compositions in molar fraction. In Figure 4.1.10 there is the graph representing the TIT values for each different fuel used: the comparison is made between the temperature in HYSYS®, taken as reference, the temperature calculated using the chemical reactions method, and the temperature calculated without implementing chemical reactions (and whose fluid therefore has the same composition as the incoming air).

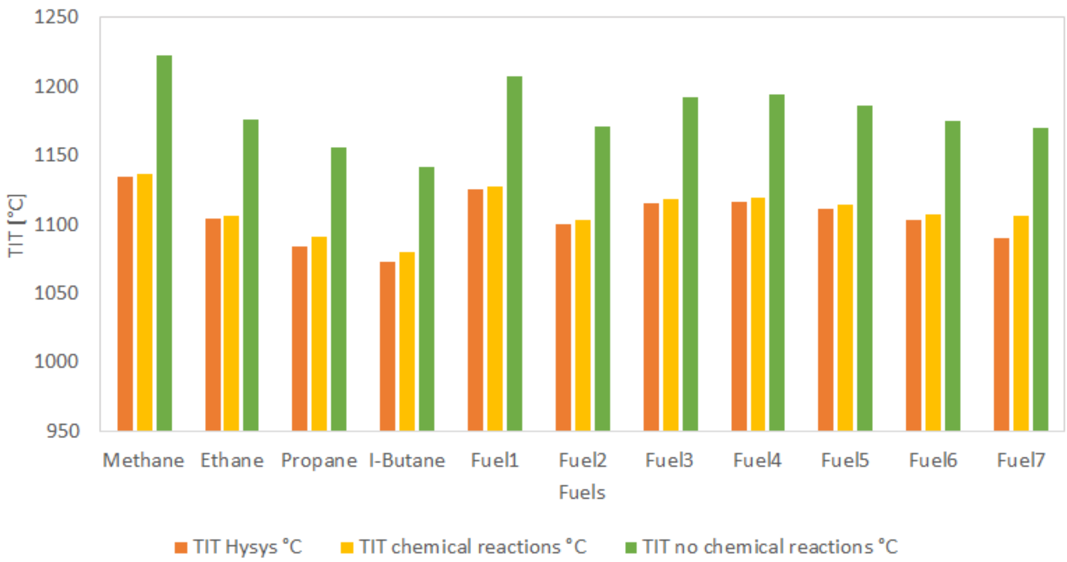


Figure 4.1.10: TIT values for different fuels listed in 3.2.1: the comparison is between the value in HYSYS®, in Python with the chemical reaction method and in Python without the chemical reaction method

The values are also shown in terms of absolute temperature difference, in 4.1.11. The absolute difference is calculated between the TIT in HYSYS® and the TIT calculated with the chemical reaction method, and between the TIT in HYSYS® and the TIT calculated without the chemical reaction method. In particular, it is shown that the difference between the TIT in HYSYS® and the TIT obtained without the chemical reactions is visually (and quantitatively) greater than the difference between the TIT in HYSYS® and the TIT obtained by the use of the chemical reactions method.

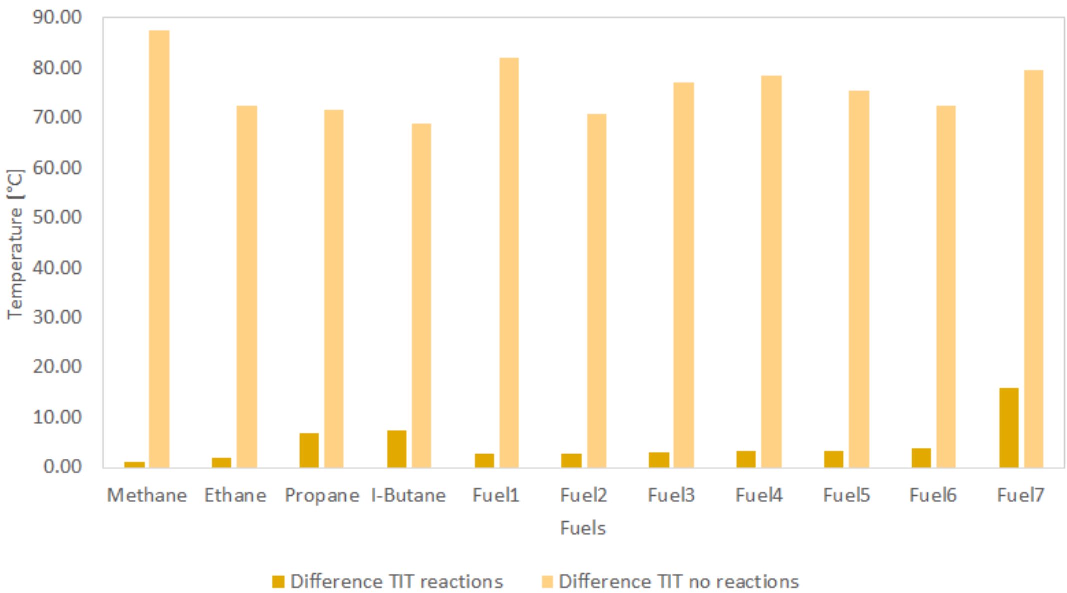


Figure 4.1.11: Comparison between the absolute difference of the TIT in HYSYS® and the TIT with the chemical reactions and the absolute difference of the TIT in HYSYS® and the TIT without the chemical reactions

To give a better idea in terms of the percentage difference of the two temperature calculation methods, the percentage error of the absolute deviation between the difference between the temperature out of the combustion chamber and the ambient temperature is calculated, both for TIT in HYSYS®, for TIT calculated without the chemical reactions, and for TIT calculated with the chemical reactions.

In the equations below, the difference between the TIT and the ambient temperature. In particular, the subscript r stands for *reactions*, and the temperature is referred to the TIT considering the chemical reactions, while the subscript nr stands for *no reactions*, and the temperature is referred to the TIT that does not consider the chemical reactions.

$$\epsilon_{Hysys} = TIT_{Hysys} - T_{ambient} \tag{4.3}$$

$$\epsilon_{r} = TIT_{r} - T_{ambient} \tag{4.4}$$

$$\epsilon_{nr} = TIT_{nr} - T_{ambient} \tag{4.5}$$

The percentage error calculated for the two methods is therefore expressed as follows in 4.6 for the TIT considering the chemical reactions and in 4.7 for the TIT not considering the chemical reactions:

$$e_{r} = \frac{\left|\epsilon_{Hysys} - \epsilon_{r}\right|}{\epsilon_{Hysys}} \tag{4.6}$$

$$e_{nr} = \frac{\left|\epsilon_{Hysys} - \epsilon_{nr}\right|}{\epsilon_{Hysys}} \tag{4.7}$$

As can be seen from the Figure 4.1.12, the percentage error calculated with the method described confirms in quantitative percentage terms the results analysed above: the percentage error of the TIT calculated with the method of chemical reactions settles at a value close to 0%, in comparison with the 6/7% error of the TIT calculated without the chemical reactions.

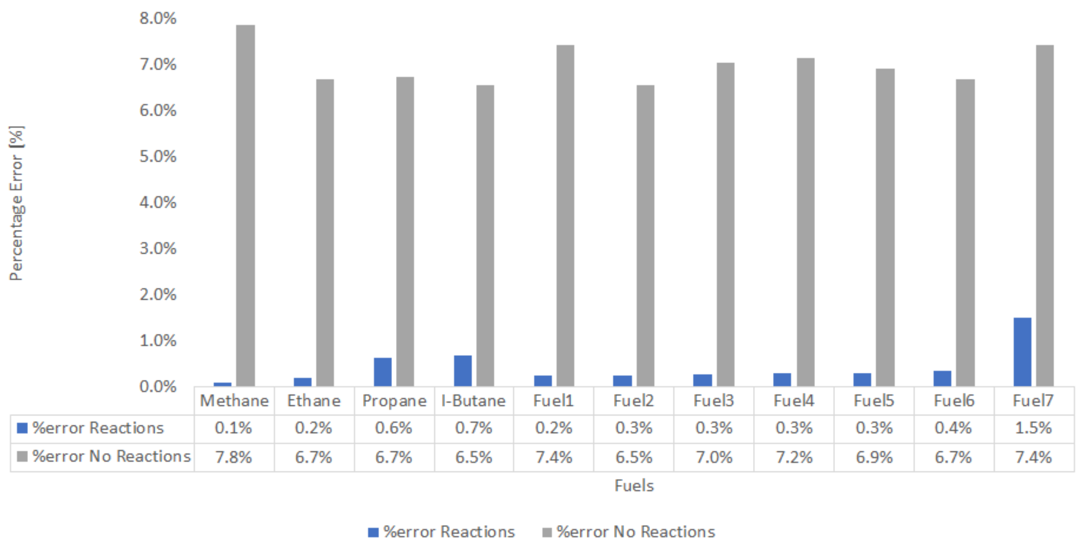


Figure 4.1.12: Percentage error of the absolute deviation between the difference between the temperature out of the combustion chamber and the ambient temperature: TIT with chemical reactions compared to the TIT without the chemical reactions

As can be seen from the trend in fuel type number 7 (and referring to the mole fraction compositions in 3.2.1, it consists of methane, ethane, propane, i-butane, i-pentane and n-pentane), increasing the number of components in the fuel, the convergence of the chemical reaction method to the values in HYSYS® is slightly reduced, while is still significantly better than the method without chemical reactions.

Another analysis takes into account the change of the mass flow rate of the air and it evaluates how the TIT is affected.

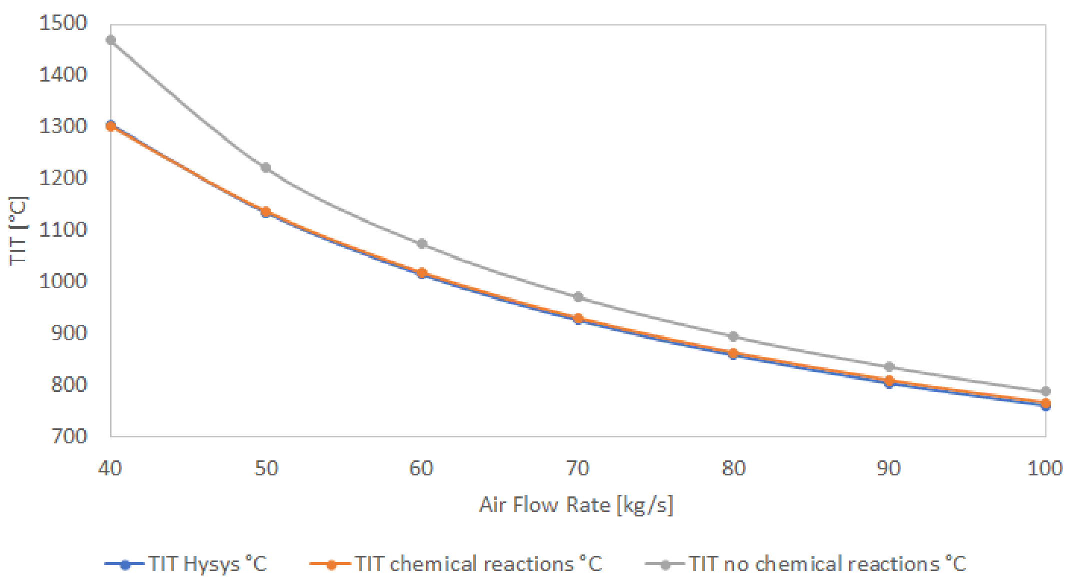


Figure 4.1.13: Change of TIT by changing the mass flow rate of air

As it can be seen in the picture 4.1.13, the trend is as might be expected. The temperature calculated with the chemical reaction methods is always comparable to the temperature calculated by HYSYS®, since the specific heat c_p of the combustion fluid is quite similar to the c_p calculated in HYSYS®, displayed in Figure 4.1.14.

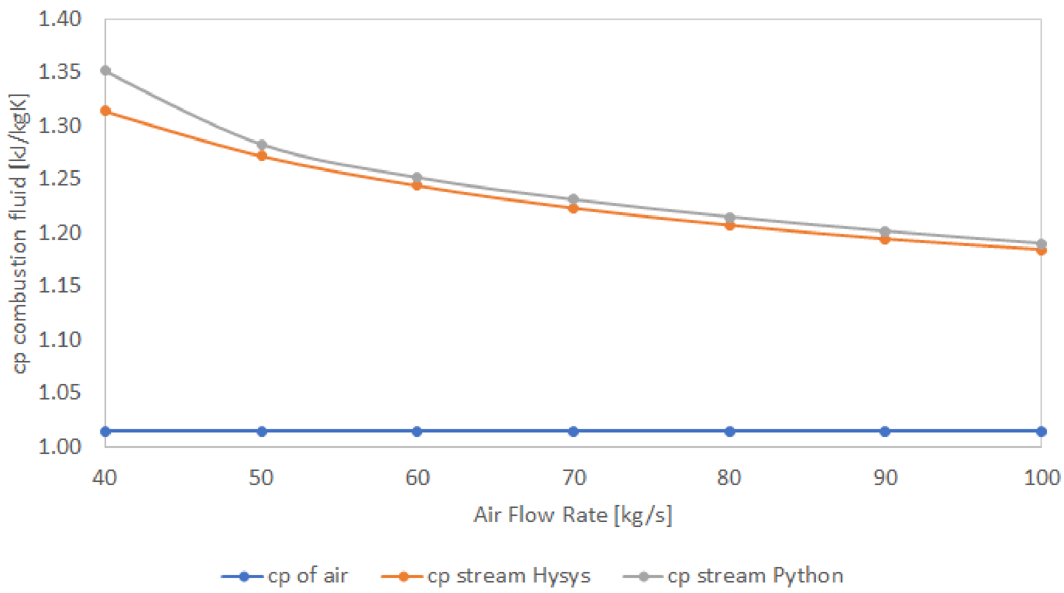


Figure 4.1.14: Change of c_p of combustion fluid by changing the mass flow rate of air

The temperature calculated without considering the chemical reactions - and so considering the specific heat of the combustion fluid as the specific heat of air - is instead improving as the mass flow rate of air increases: not changing the mass flow of the fuel, in fact, as the mass flow rate of the air increases, the c_p tends to be closer to the c_p of air, leading to a smaller error in calculating the reaction temperature.

Furthermore, an aspect to consider is the always higher value of the TIT calculated without considering the chemical reaction than the temperature that should actually be. When considering the combustion process, in fact, a release of energy is considered, which leads to an increase of the heat capacity of the fluid: this increase represents the increase of the energy that is required to raise the temperature out of the combustion chamber.

4.1.4 Case study: off-design

In this section the results of the off-design model will be highlighted, referring to the section 3.2.1.1, in which the model is described.
Firstly, a comparison of the compressed air temperature and the exhaust gas temperature and between Python and HYSYS® is shown, by changing the operating point with different ambient temperatures.

The Python and HYSYS® temperature trends for the compressor outlet temperature and exhaust gas temperature, respectively, are shown in the Figures 4.1.15 and 4.1.16; the percentage error (calculated on the absolute difference of the deviation between the temperature in HYSYS® and the ambient temperature and the deviation between the temperature in Python and the ambient temperature) is shown in the Figures 4.1.17 and 4.1.18.

Going into the details of the analysis, it seems that the model performs better at lower ambient temperatures, as can be seen from the decreasing trend of the percentage error as the ambient temperature decreases; in any case, the worst-case percentage error is about 1.3%, leading to the conclusion that the model has a very good calculation prediction compared to HYSYS®.

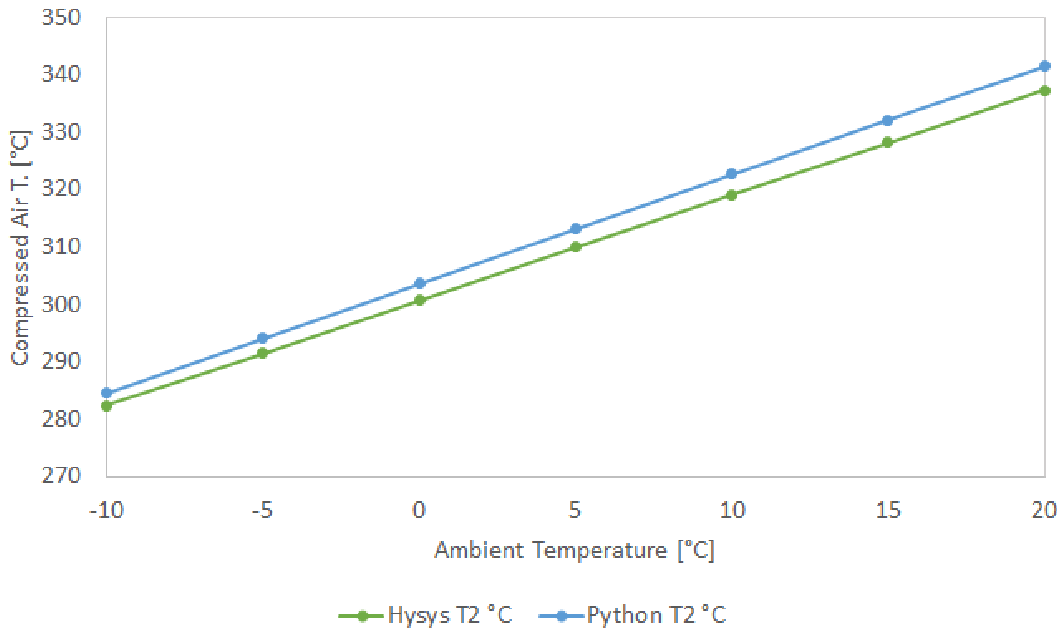


Figure 4.1.15: Trend of the T_2 in off design condition by changing the ambient temperature

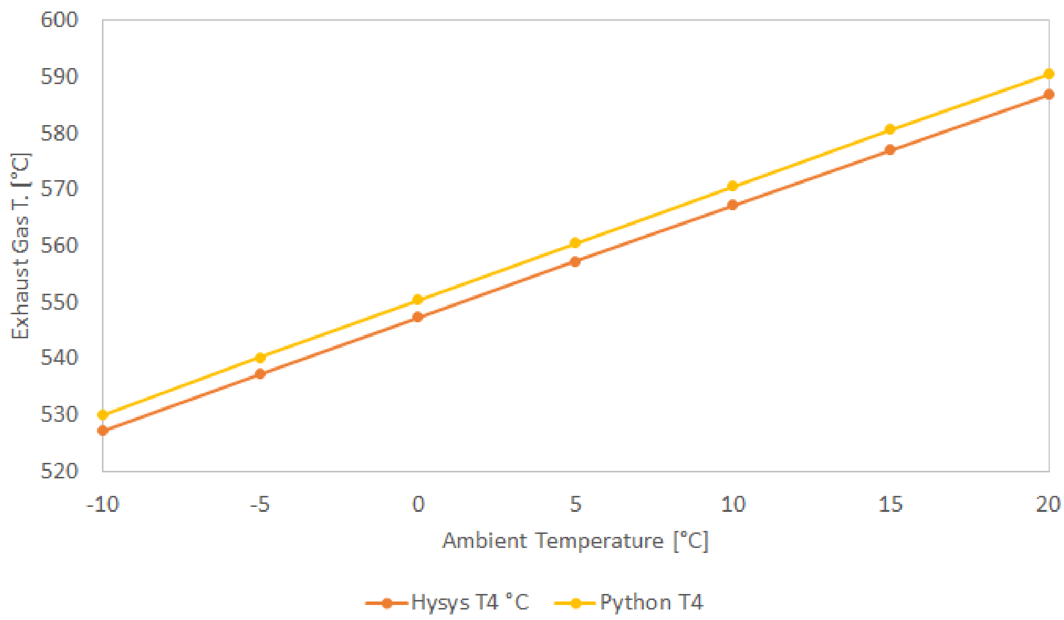


Figure 4.1.16: Trend of the T_4 in off design condition by changing the ambient temperature

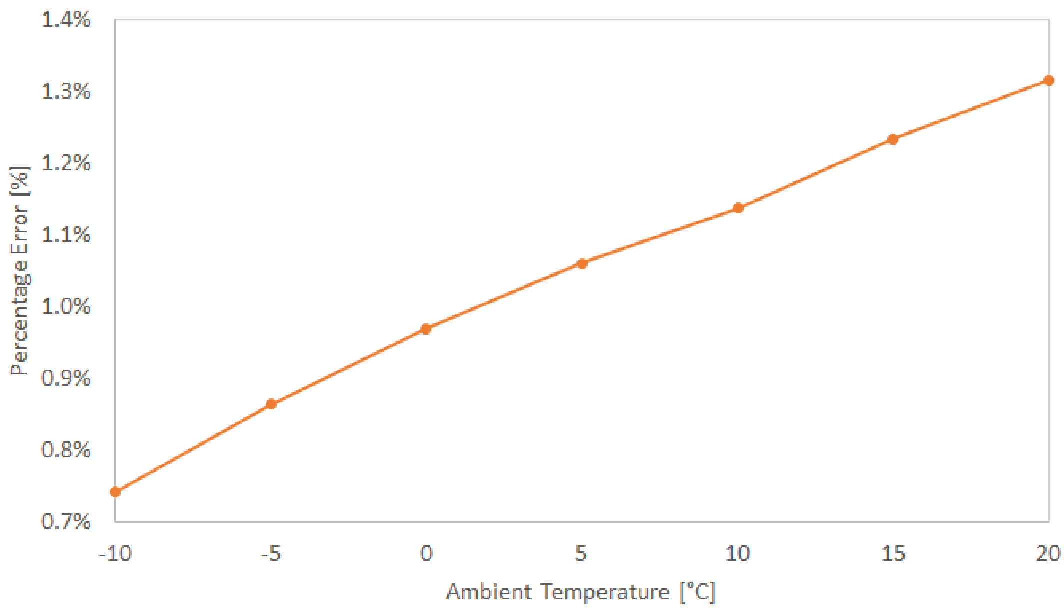


Figure 4.1.17: Percentage error of the absolute difference of the deviation between the compressed temperature in HYSYS® and the ambient temperature and the deviation between the compressed temperature in Python and the ambient temperature

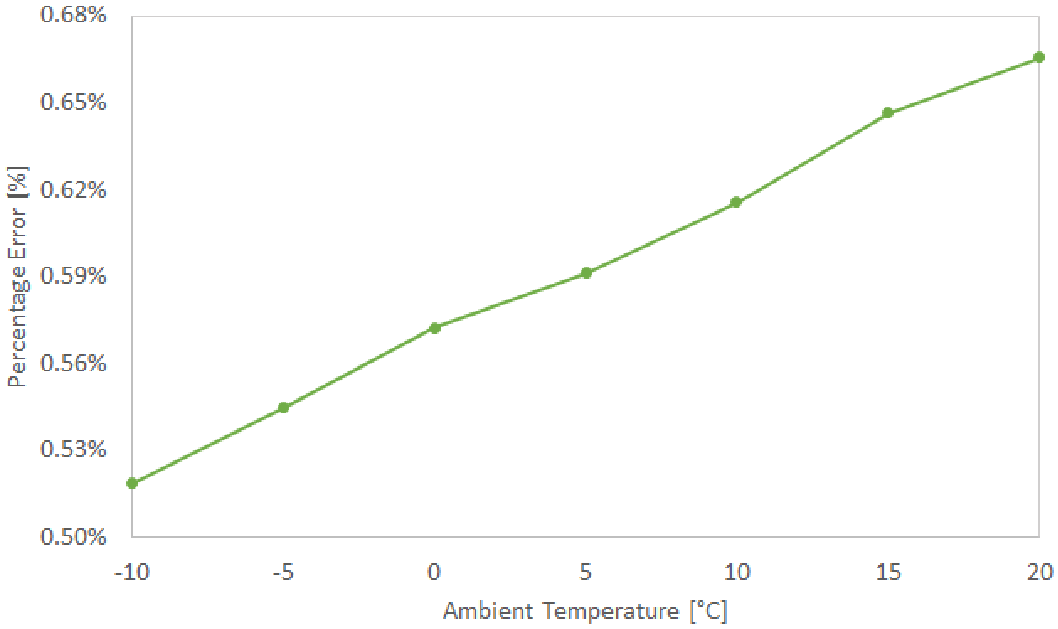


Figure 4.1.18: Percentage error of the absolute difference of the deviation between the exhaust gas temperature in HYSYS® and the ambient temperature and the deviation between the exhaust gas temperature in Python and the ambient temperature

Figure 4.1.19 instead shows the trends of the temperature leaving the combustion chamber: as can be seen, the temperature in HYSYS® and the temperature in Python calculated using the chemical reactions method are comparable, while the temperature calculated without the chemical reactions is significantly deviating from the value taken as a reference in HYSYS®.

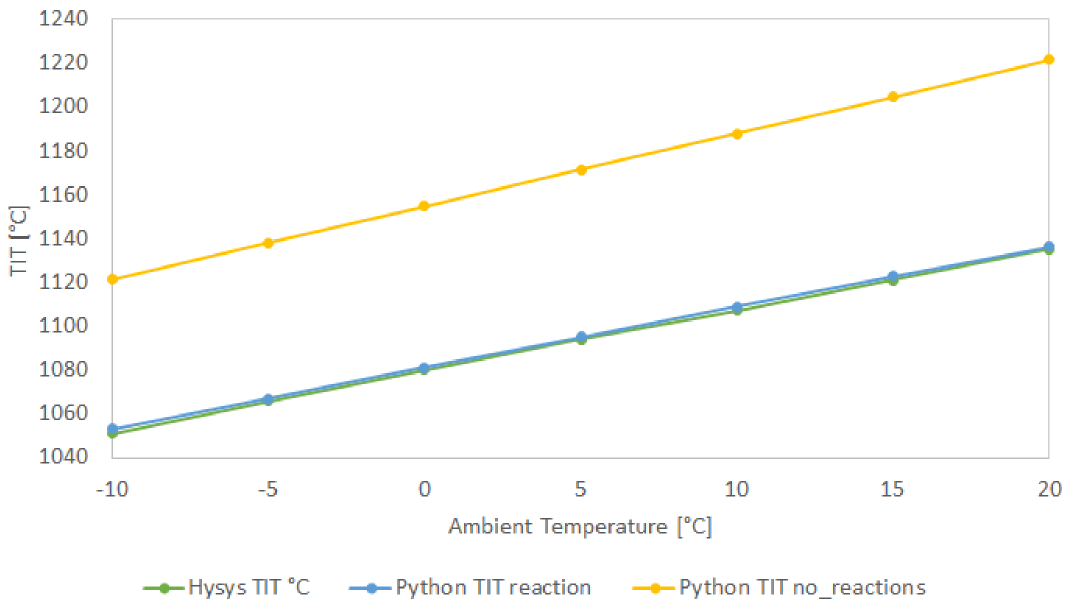
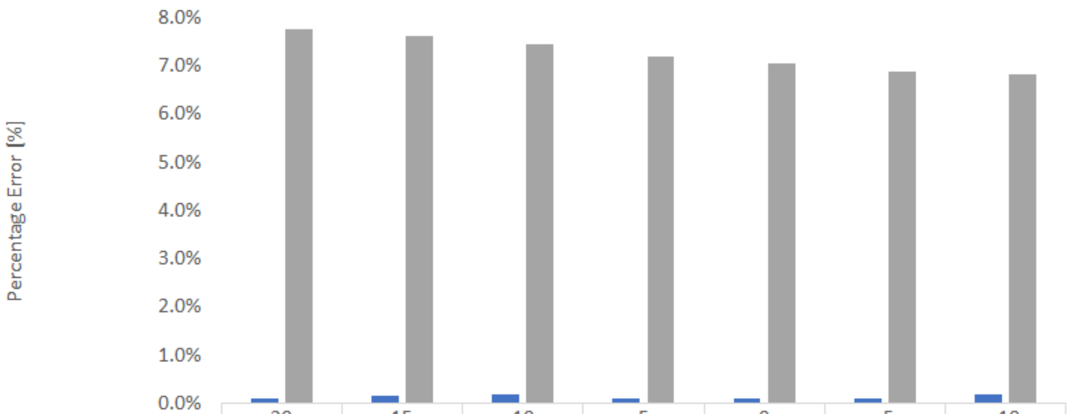


Figure 4.1.19: Trend of TIT in off design condition by changing the ambient temperature

To give a quantitative percentage idea about the comparison of the TIT calculation methods, the percentage error of the absolute deviation between the difference between the temperature out of the combustion chamber and the ambient temperature is calculated, as it is done for the design.

As can be seen from the picture 4.1.20, the results are consistent with what is also shown for the design model: with the chemical reaction calculation method, the TIT values deviate very little from the values obtained from HYSYS®, as the ambient temperature changes.



	20	15	10	5	0	-5	-10
%error Reactions	0.1%	0.1%	0.2%	0.1%	0.1%	0.1%	0.2%
%error No Reactions	7.8%	7.6%	7.4%	7.2%	7.0%	6.9%	6.8%

Figure 4.1.20: Percentage error of the absolute deviation between the difference between the temperature out of the combustion chamber and the ambient temperature for different operating points: TIT with chemical reactions compared to the TIT without the chemical reactions

The same analysis is made for comparing the trend by changing the ambient temperature of the compressor duty and the turbine power. Results are shown in Figures 4.1.21 and 4.1.22. In particular, for both the compressor and the turbine (although the trend is less linear), the percentage error between the value in HYSYS® and the value in Python seems to decrease as the ambient temperature decreases. In any case, the error always settles at about 0%, leading to a good result for the power computational method in off-design.

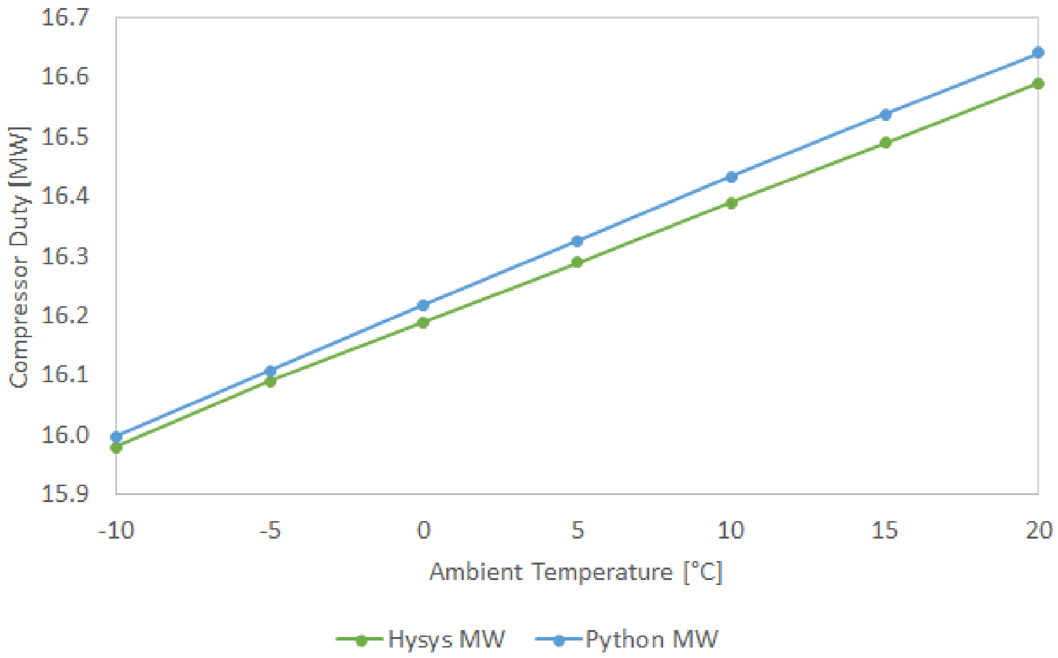


Figure 4.1.21: Trend of the compressor duty by changing the ambient temperature

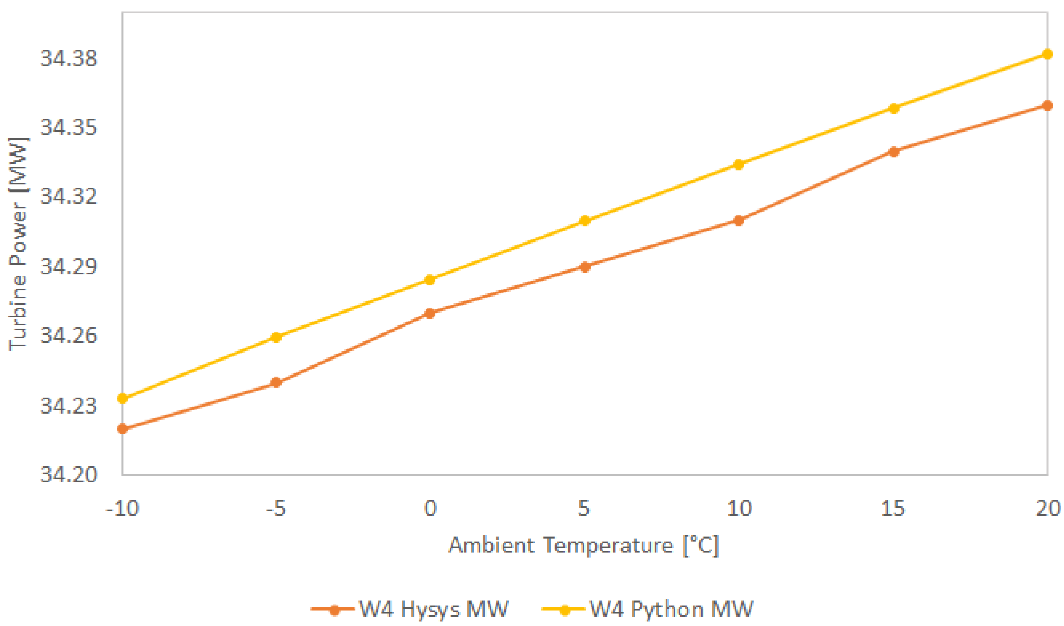


Figure 4.1.22: Trend of the turbine power by changing the ambient temperature

Finally, a comparison of the gas turbine net power trend (calculated as the power generated by the turbine minus the work required by the compressor) is shown in Figure 4.1.23, and the percentage error trend is shown in Figure 4.1.24: as can be seen, the error also seems to decrease for power, as well as for temperature, as the ambient temperature decreases. In any case, the maximum value of the percentage error is below 0.2%, leading to the conclusion of a good calculation result in comparison with HYSYS®.

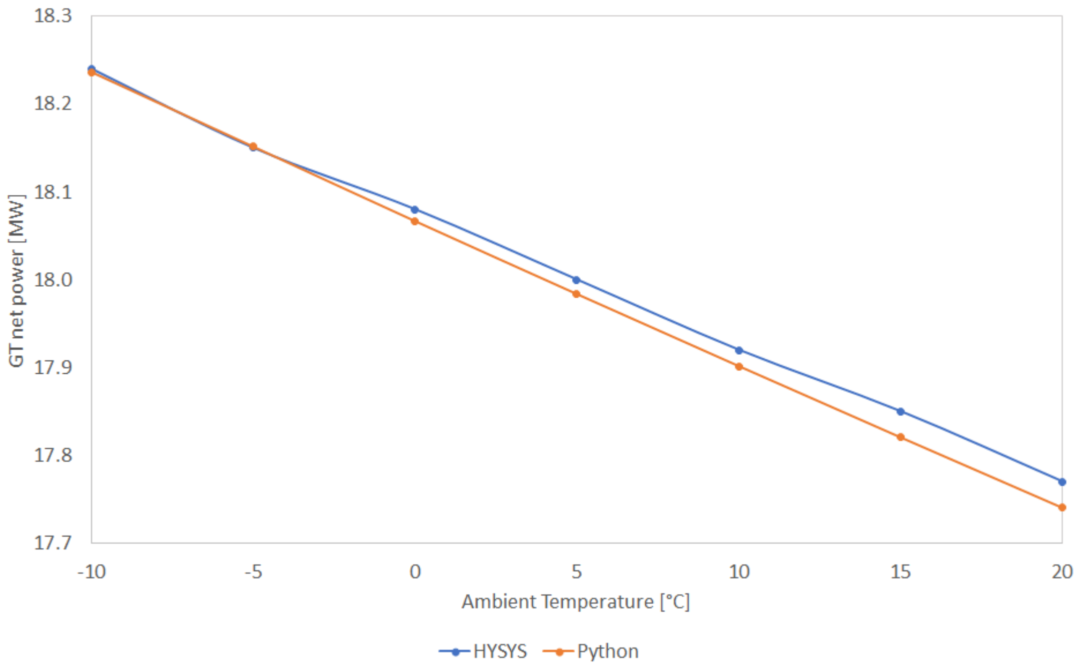


Figure 4.1.23: Comparison of the overall net gas turbine power generated between HYSYS® and Python in off-design conditions, as the difference between the turbine power and the compressor work

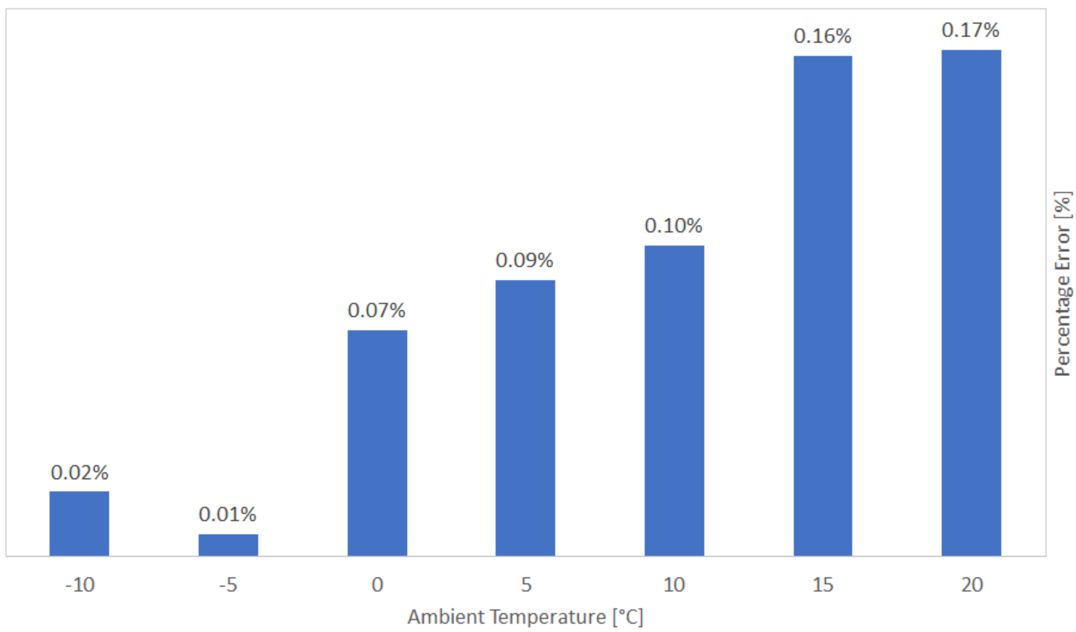


Figure 4.1.24: Percentage deviation of the net gas turbine power generated calculated in Python, compared to HYSYS®

4.2 Comparison with Thermoflow©

This section will show the results from the comparison between the model designed in Python for the GE LM2500 and GE LM6000 turbines and the values found in Thermoflow©. Firstly, the discussion will cover the analysis of the design conditions.

4.2.1 Design model: results

Taking into account the turbine design procedure described in section , and the efficiency values identified with GasTurb© described in section , tables 4.2.1 and 4.2.2 show the results in terms of absolute difference and percentage difference between Python and Thermoflow© for the two turbines.

Variable	**Unit**	**GT PRO**	**Python**	**Dev.**	**% Dev.**
Gross Power Output	kW	21958	22227.15	269.15	1.23
Exhaust Temperature	K	816.15	816.67	0.52	0.06
Heat Rate	kJ/kWh	1033	9958.86	74.14	0.74
GT efficiency	%	35.88	36.15	0.27	

Table 4.2.1: GE LM2500 turbine design conditions: comparative results with GT PRO, Thermoflow©. *dev* is for *deviation*

Variable	**Unit**	**GT PRO**	**Python**	**Dev.**	**% Dev.**
Gross Power Output	kW	45199	44908.34	290.66	0.64
Exhaust Temperature	K	726.15	727.05	0.90	0.12
Heat Rate	kJ/kWh	8610	8706.95	96.95	1.13
GT efficiency	%	41.80	41.35	0.45	

Table 4.2.2: GE LM6000 turbine design conditions: comparative results with GT PRO, Thermoflow©. *dev* is for *deviation*

4.2.2 GE LM2500 off-design model: results

This section shows the results of the development of the off-design model for the GE LM2500 turbine, as explained in 3.3.3. In order to model the off-design conditions for the gas turbine GE LM2500, air flow rate values are taken from Thermoflow©, as the change in the angle of the IGVs is not known.

If the operating condition for temperature values different than those plotted in Thermoflow© had to be identified, a linear regression function is furthermore implemented to identify the mass flow rate value for all the other ambient temperature values: this is, of course, a simplification, adopted because calculating the corrected flow rate dependent on the opening of the vanes assumes parameters that are not available, and a dedicated in-depth study.
In Figure 4.2.1, the linear regression plot can be seen.

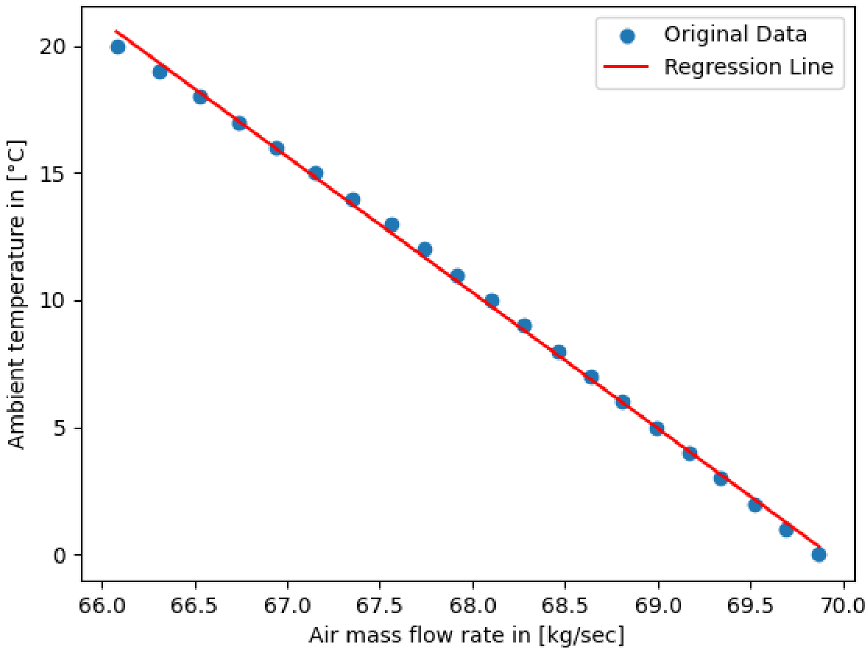


Figure 4.2.1: Air flow rate linear regression plot by changing the ambient temperature for GE LM2500, $RMSE = 0.2$

To evaluate the quality of the approximation by linear regression, the RMSE (root mean squared error) is calculated, as the square root of the mean of the sum of the squared deviations between the values predicted with the linear regression model and the Thermoflow© values, for ambient temperature from 0 to 20 degrees Celsius:

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}} \tag{4.8}$$

The value of RMSE close to zero, leads to the conclusion that the calculation of the flow rate as a linear regression is acceptable.

As explained in the 3.3.3 section, the off-design model for the GE LM2500 turbine is constructed in a simplified way according to the identified procedure. In particular, the Figure 4.2.2 shows the multiplicative coefficient $CIGV$ given in the equation 3.1: as can be guessed, this coefficient, being the ratio between the corrected flow rate in off-design conditions and the corrected flow rate in design conditions, is greater than one for all values below the design temperature of 20 °C: this is consistent with the development of the compressor map as the ambient temperature changes (and thus as the angle of the IGVs,) as explained in Figure 2.1.9.

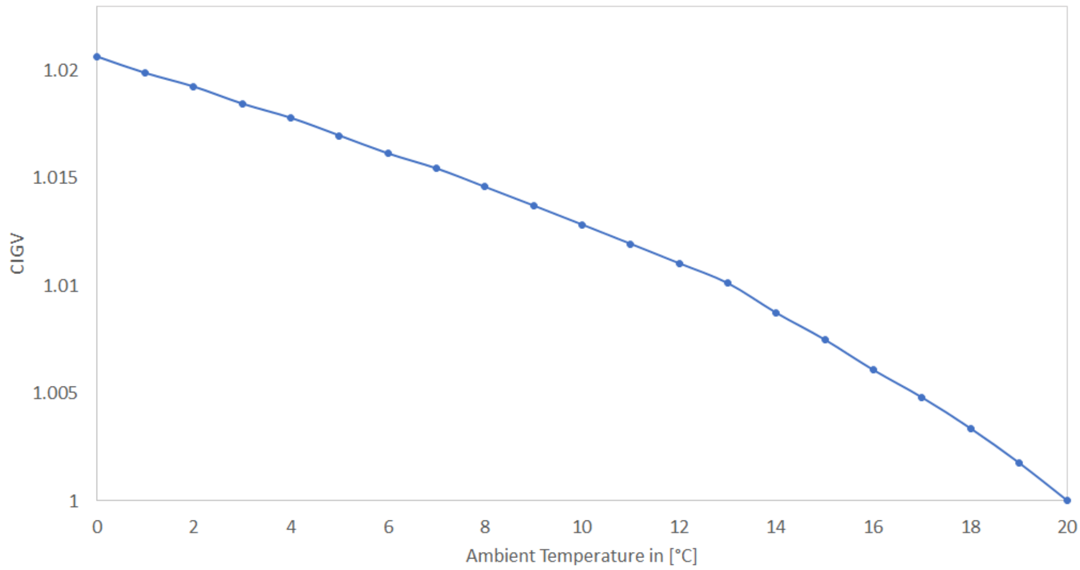


Figure 4.2.2: Ratio between corrected flow rate in off-design and corrected flow rate in design, by changing the ambient temperature, with 20°C being the design temperature for GE LM2500

Considering a constant shaft speed, it is also possible, from the values of the air flow rate in Thermoflow©, to approximately identify the value of the opening angle of the IGVs in relation to the angle in design condition (considered as 100% opened), by referring to the equations 2.34 and 2.36.

In Figure 4.2.3, the trend of the angle opening (delta gamma) as the ambient temperature changes is shown: the angle opening even at ambient temperatures much lower than the design temperature is not significantly great, but this still seems to have an impact on the performance of the turbine, especially on the efficiency of the components.

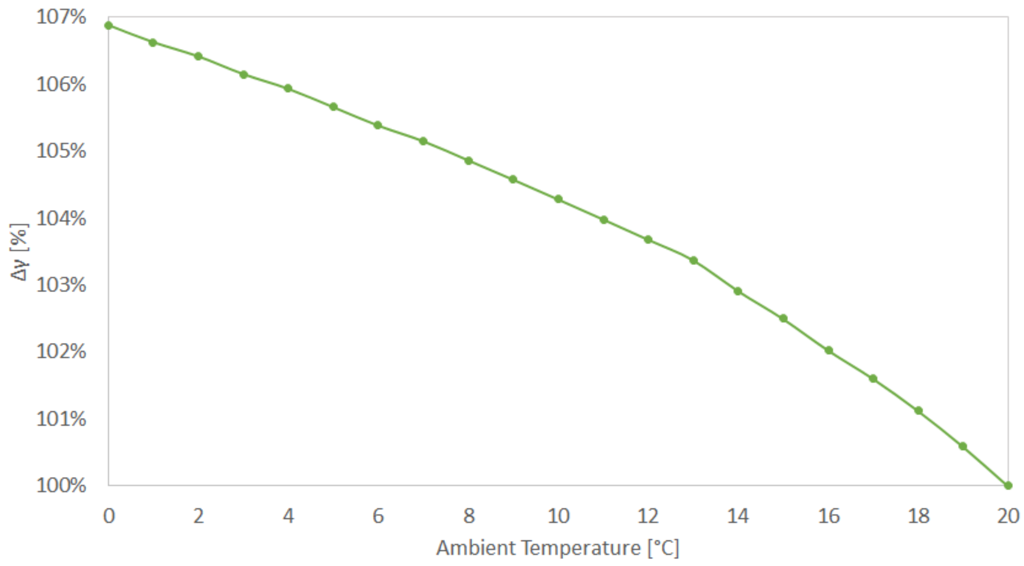


Figure 4.2.3: Opening angle of the IGVs as the ambient temperature changes, taking the opening angle in design conditions as a reference

Thus, according to the identified methodology, the off-design model is implemented for the GE LM2500 turbine at different ambient temperatures, and the values are compared with those in Thermoflow©. In Figure 4.2.4 the comparison between the net power trend in Python and the net power trend in Thermoflow© is shown, and in Figure 4.2.5 the comparison between the exhaust gas temperature trend in Thermoflow© and the exhaust gas temperature trend in Python.

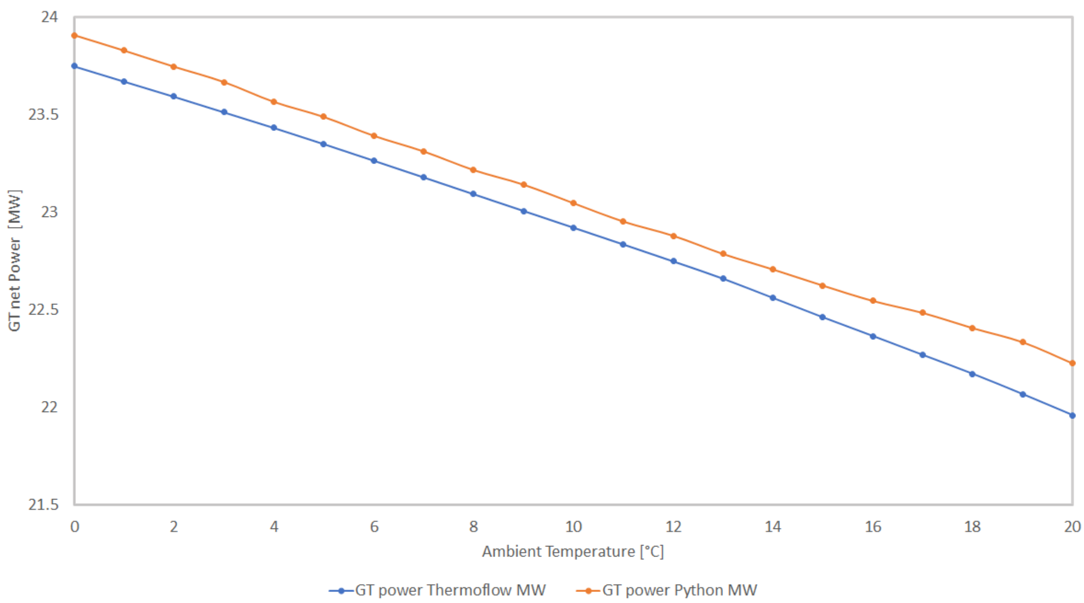


Figure 4.2.4: GT power calculated with Python for GE LM2500 gas turbine in off-design conditions, compared with Thermoflow© results

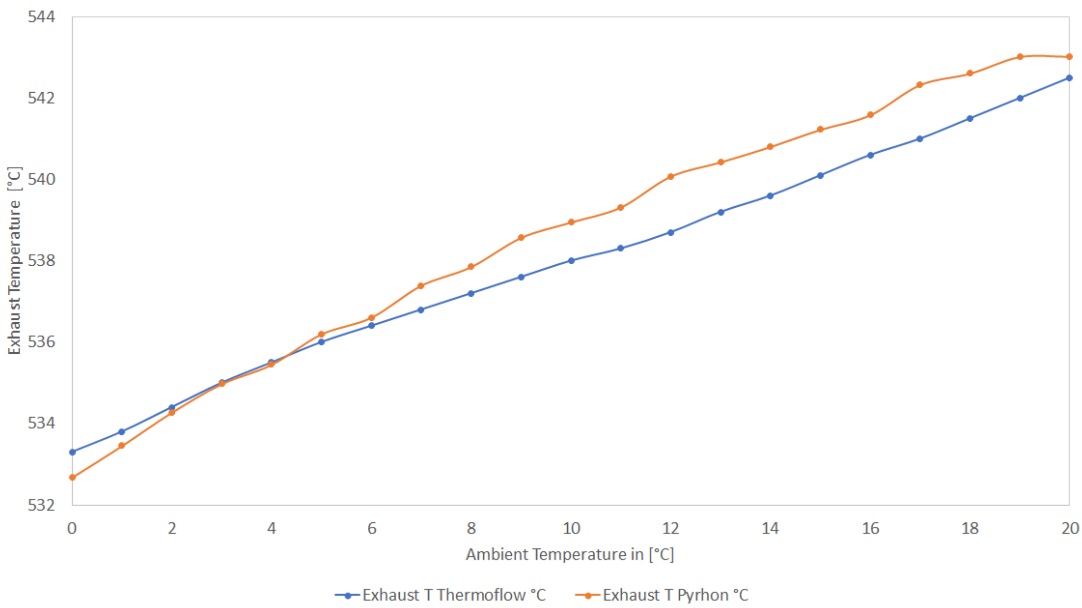


Figure 4.2.5: Exhaust gas temperature calculated with Python for GE LM2500 gas turbine in off-design conditions, compared with Thermoflow© results

Percentage deviation trend are also shown: the power net percentage in Figure 4.2.6 trend shows that the deviation seems to increase as the ambient temperature increases, and stabilise at 19°C ambient temperature. The error is, however, minimal and settles at a maximum percentage deviation of 1.2%.

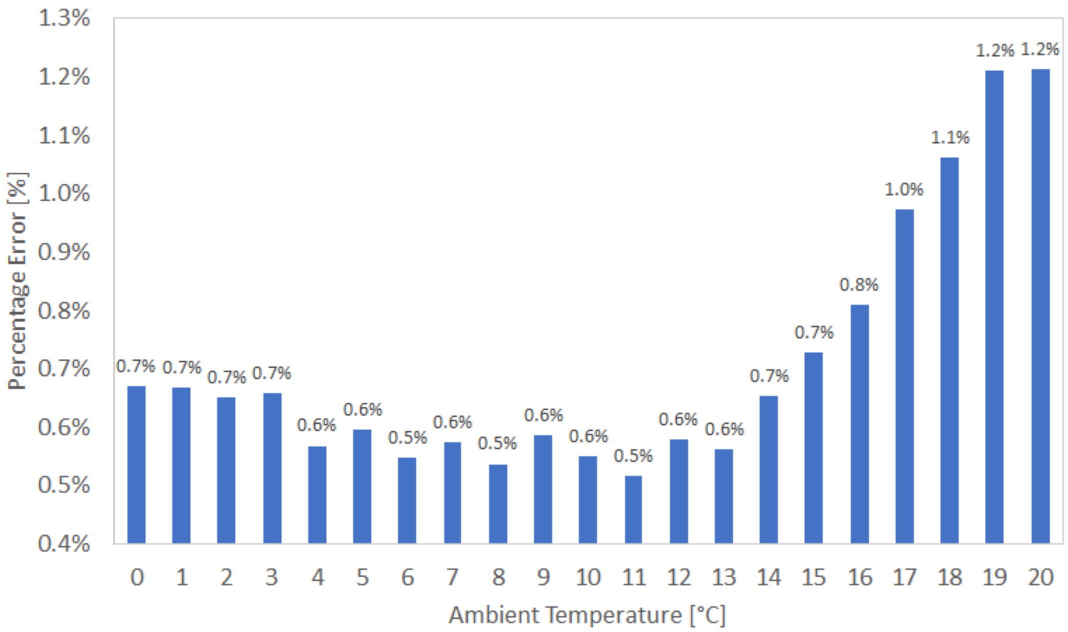


Figure 4.2.6: GT net power percentage error between Python and Thermoflow©

The percentage deviation for the exhaust gas temperature in Figure 4.2.7 is calculated with the same principle explained also in the analysis of the case study: first is calculated the difference between the exhaust gas temperature (for both Python and Thermoflow©) and the ambient temperature, and with these values the percentage deviation is calculated.

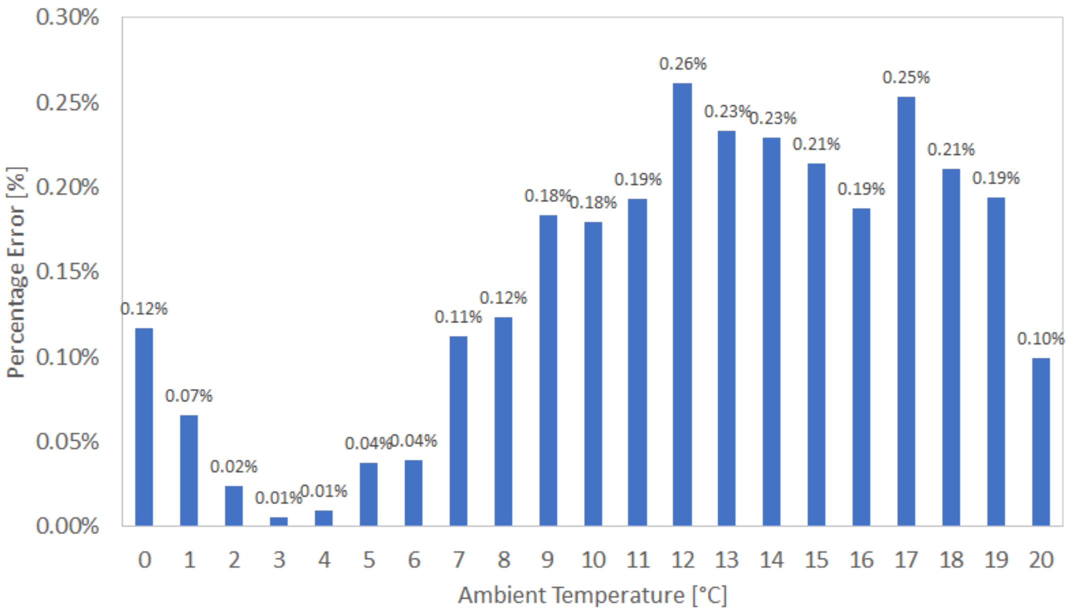


Figure 4.2.7: Exhaust gas temperature percentage error between Python and Thermoflow$^{T©}$

The trend in percentage error seems more jagged and less linear than that of power, but in any case settles at very low values below 0.3%. There could be lots of possible explanations of the deviations of the Python model from the results in Thermoflow$^{©}$, and one of them could be that the method for calculating the turbine efficiency reduction takes into account the fact that the opening of NGVs is comparable with the opening of IGVs, a simplification that should be discussed and analyzed in detail in future work.

The table 4.2.3 also shows the percentage MAPE values calculated for exhaust gas temperature and GT net power, to give an overall view of the goodness of the model. In addition to the MAPE for temperature and power, the MAPE values of the heat rate are also shown, even though this is a power-dependent value.

Variable	**MAPE%**
GT net Power	0.71
Exhaust Temperature	0.14
Heat Rate	0.23

Table 4.2.3: MAPE percentage values for exhaust gas temperature, GT net power and heat rate

4.3 Performance Indicator

This section will present the results obtained from the analysis of the field data the field in relation to the performance index identified in the section 3.4, as a first example analysis of the behaviour of the components with regard to condition monitoring and degradation.

As already highlighted in the method explanation, for modelling the turbine in the digital twin, ambient condition data data (ambient pressure and ambient temperature), and the pressure out of the compressor are taken from the field.
The output temperature at the compressor, on the other hand, is calculated and compared with the actual data to identify any deviation from the prediction.

In order to use the data, it is necessary to first clean the data of values that are extremely far from the average, which could be due to a sensor error or a turbine shutdown. In Figures 4.3.1 and 4.3.2, by way of example, the compressor raw data before the 'cleaning' are shown.

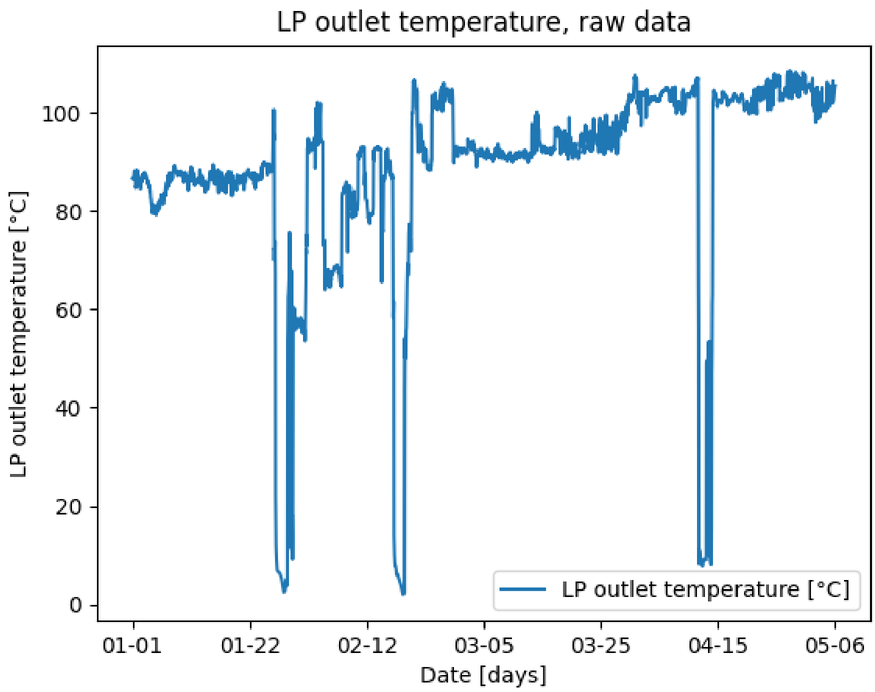


Figure 4.3.1: LP compressor outlet temperature raw data from the field

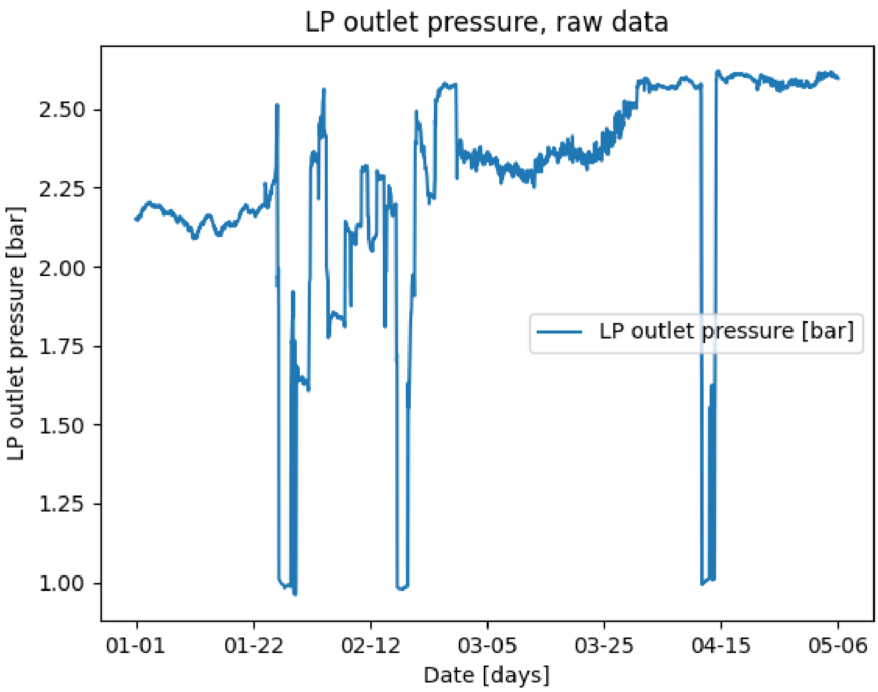


Figure 4.3.2: LP compressor outlet pressure raw data from the field

Values that differ by more than a certain threshold are first removed, and then a moving average with a window of 10 values is applied to remove sensor noise, a choice also applied by [27]. In Figures 4.3.3 and 4.3.4 compressor temperature and pressure data after the removal of the distant values are shown.

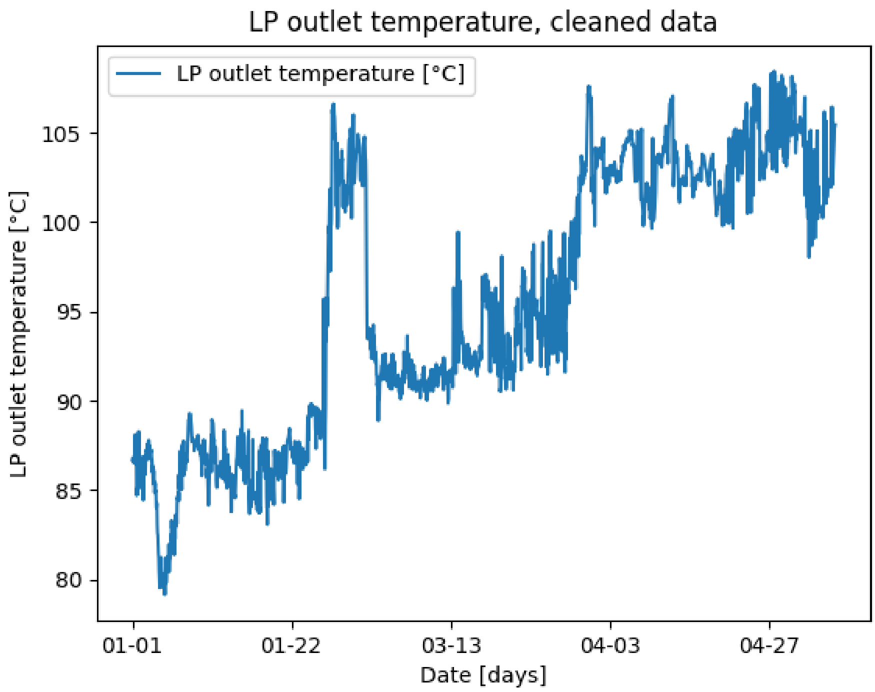


Figure 4.3.3: LP compressor outlet temperature data from the field, after the removal of the outliers

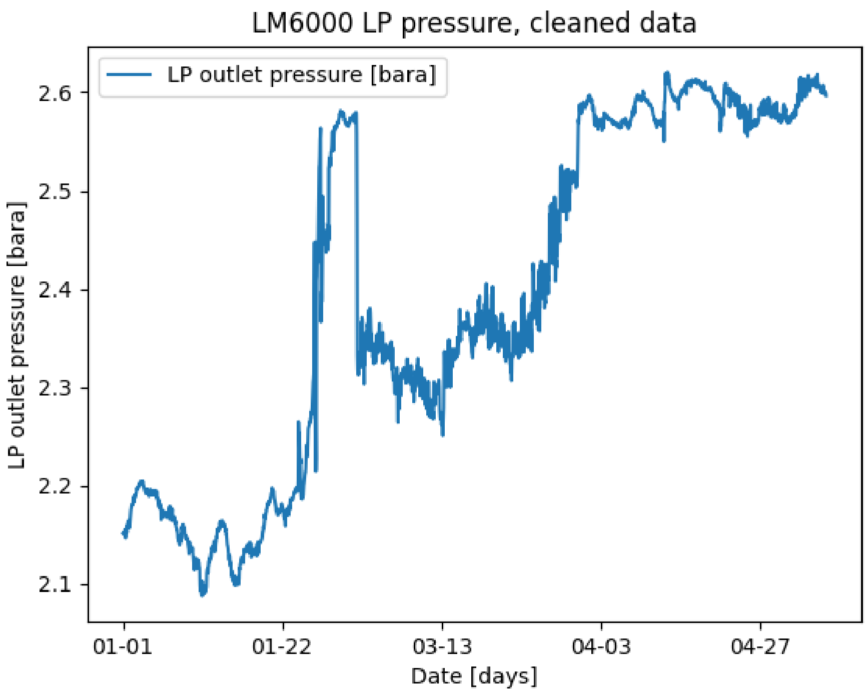


Figure 4.3.4: LP compressor outlet pressure data from the field, after the removal of the outliers

The choice of a moving average with a window of 10 does not reduce the number of values drastically, and at the same time provides a trend whose development is more easily detectable. In Figures 4.3.5 and 4.3.6 compressor temperature and pressure data after the moving average are shown.

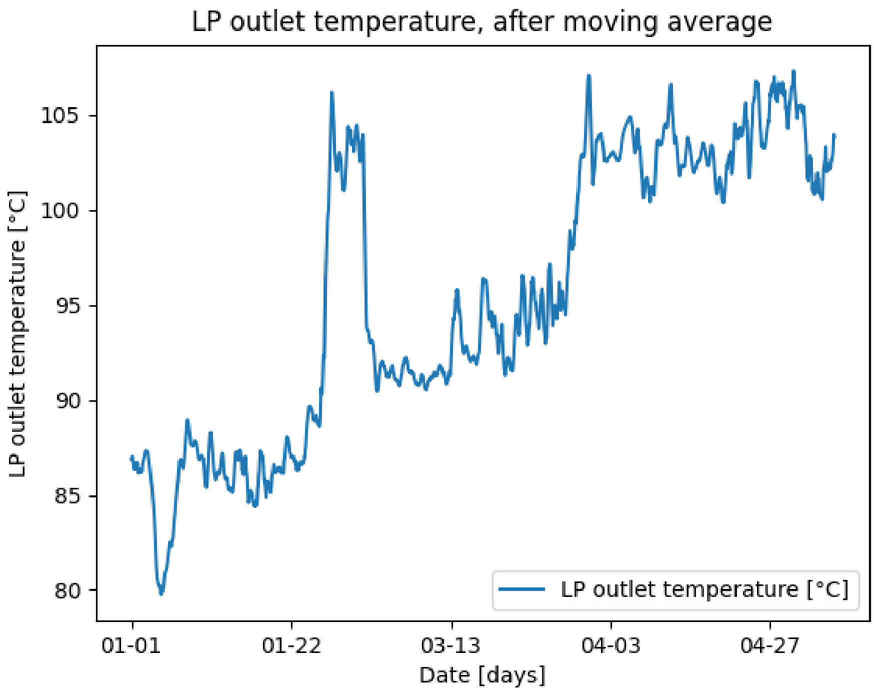


Figure 4.3.5: LP compressor outlet temperature data from the field, after the moving average with a window of 10 values to remove the sensor noise

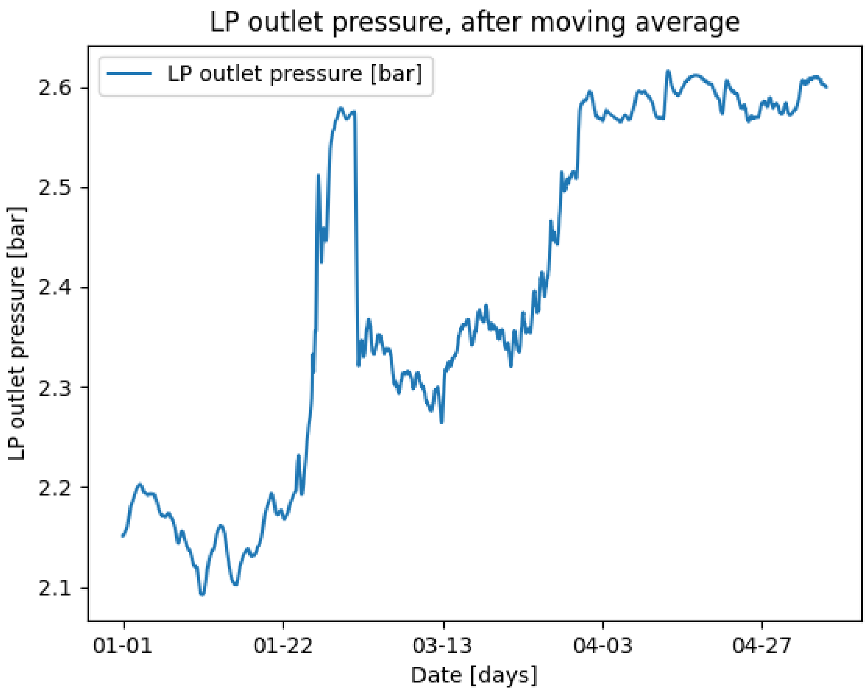


Figure 4.3.6: LP compressor outlet pressure data from the field, after the moving average with a window of 10 values to remove the sensor noise

As an example, therefore, Figure 4.3.7 shows the trend of the compressor outlet temperature data with a moving average of 3: as can be seen, the trend is less clear and more jagged, with only 7 more values (in the case of a moving average of 10, the number of data output is 2201, whereas with a moving average of 3 is 2208).

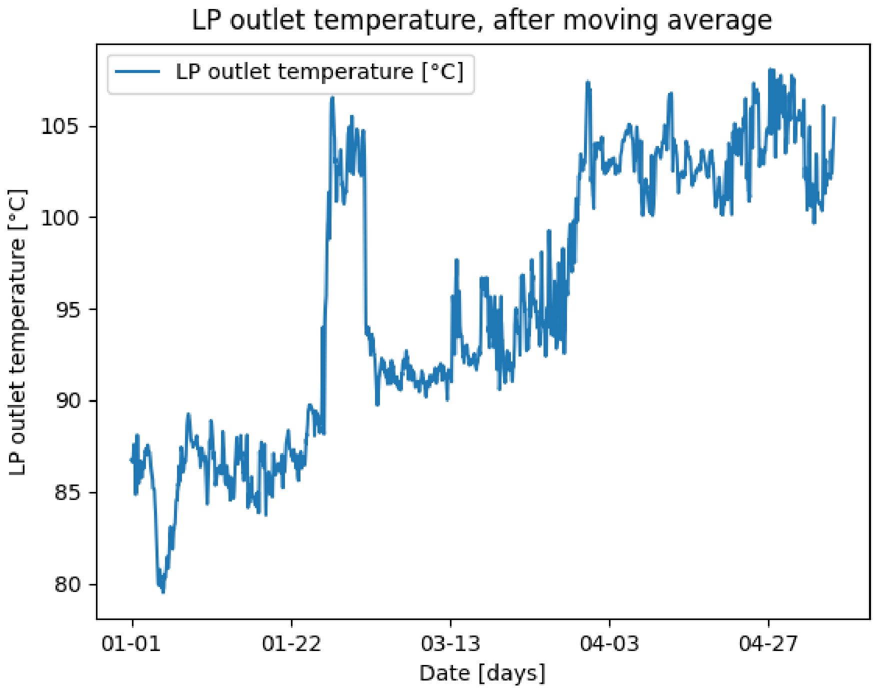


Figure 4.3.7: LP compressor outlet temperature data from the field, after the moving average with a window of 3 values

After the generation of usable data for the analysis, therefore, the model predicting the compressor behaviour is implemented, and the trend of the performance indicator is shown in Figure 4.3.8. This index is compared with a baseline of 0, which indicates the equality of the calculated temperature with the measured temperature.

When the performance indicator gets a positive value, it means an increase in the compressor outlet temperature compared to the temperature calculated by the model: this trend indicates a reduction in the compressor polytropic efficiency in comparison with the nominal polytropic efficiency in design conditions, identified in the iterative procedure in GasTurb©. A comparison between the temperature trend calculated by the model and the actual temperature trend is also shown in Figure 4.3.9.

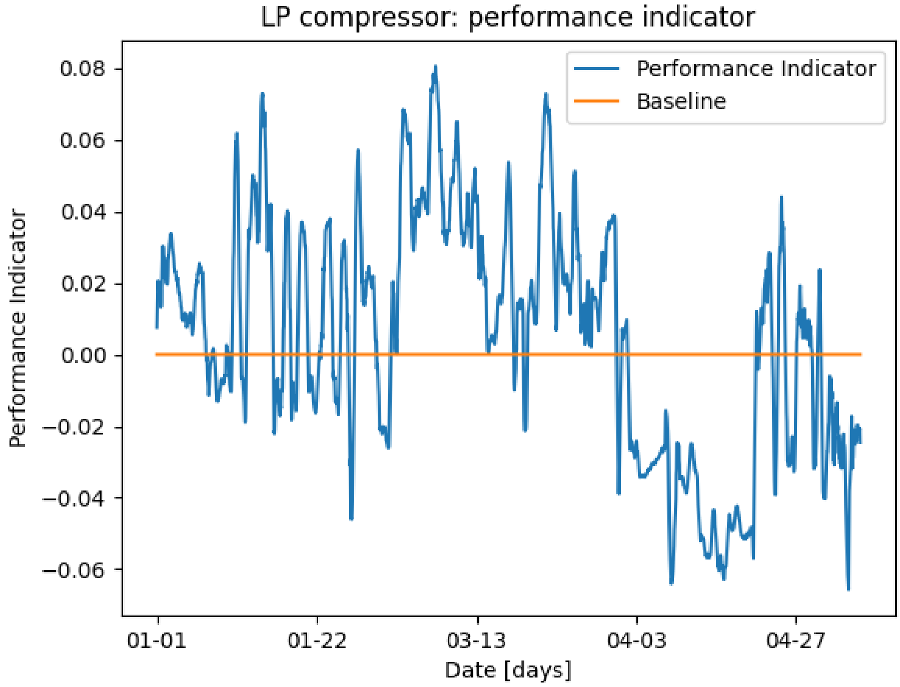


Figure 4.3.8: Performance Indicator for the LP compressor: it represents the normalised difference between the LP outlet temperature calculated by the Python model and the real outlet temperature taken from the field

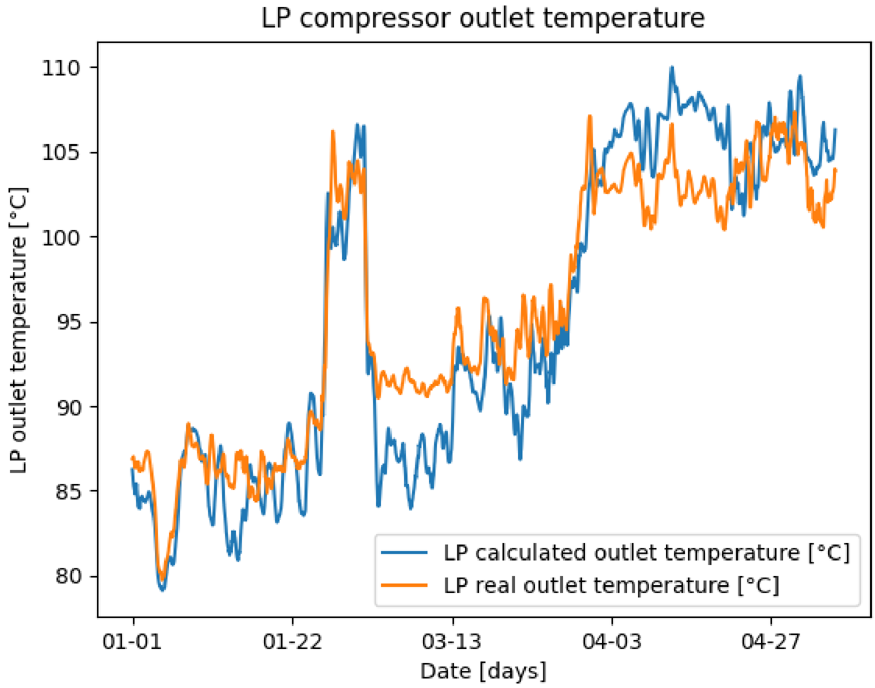


Figure 4.3.9: Comparison between the LP outlet temperature calculated by the model and the LP outlet temperature from the field data

As can be seen from the comparison from the Figure 4.3.8 showing the trend of the performance indicator and the Figure 4.3.9 showing the comparison between the calculated and actual temperature trends, the indicator assumes a positive value when the actual temperature assumes an higher value than the calculated temperature; when the actual temperature is lower than the calculated temperature, on the other hand, the index assumes a negative value.

The results show an ability of the model to capture the compressor outlet temperature trend, but attributing this variation from the model to actual component degradation is not trivial. In particular, the indicator is created in order to be implemented in a condition monitoring strategy applied to predictive maintenance for component degradation, but there are several considerations and limitations to be noted at this stage of the project.

Firstly, the data analysis is performed on the GE LM6000 turbine, for which, however, an ad-hoc off-design model is not designed, which is reflected in the lack of accurate prediction of the variation of the compressor polytropic/isentropic efficiency as operating conditions change.

The first question that occurs is whether the deviation in efficiency (reflected in the deviation of the temperature at the compressor outlet) is due to an actual degradation of the component, or whether it is instead linked to the change of the ambient operating conditions.
It is highlighted how, in the case of the GE LM2500 gas turbine, when the ambient temperature changes in off-design, the efficiency undergoes a reduction or an increase depending on the operating point.
If, on the other hand, the reduction in efficiency is not related to changes in operating conditions but instead to actual compressor degradation, it is not trivial to quantify and analytically define the degradation.

Firstly, it would be necessary to analyse the data over a time span of years, as also pointed out in [27], to identify an actual upward trend in the performance indicator (which, as it increases, would indicate an increase in degradation): the amount of data used for this analysis is limited to only 5 months, hence there are not enough information to detect an effective long-term trend. Furthermore, an analytically correlation between the thermodynamic cause, the reduction in efficiency, and an observable physical consequence should be found and demonstrated.

In the literature there are several faults that can occur in a gas turbine, and a common one related to the compressor is the fouling, which is caused by the dust, insects and pollen that, mixed with engine exhaust and oil vapors from both internal and external leaks, form a sticky mass that adheres to the blading and annulus areas of the compressor [35]. The fouling has been demonstrated to be often the cause of drop in airflow, pressure ratio, power, thermal efficiency and also compressor efficiency [36], but at the same time being able to attribute the reduction in efficiency to compressor fouling requires a more detailed analysis of the specific case, to be validated with field data: for example, the behaviour of the performance indicator could be analysed over time and check whether, at any drops, a compressor washing happened.

4.4 Model error

In the view of the results obtained, it is appropriate to make a brief overview and recap on the accuracy of the model and the deviations it might encounter from reality. The digital twin for the two gas turbines is developed in Python using a code that has itself been validated by comparison with simulation models developed in other software, which also have a certain deviation from the real data.

First of all, the source code for the development of the calculation methods is validated through a comparative analysis with Aspen HYSYS®: the maximum error detected, considering both the analysis under design and off-design conditions, is for the compressed air temperature approximately 1.2%, for the exhaust gas temperature 1.1% and for the combustion temperature in the implementation of the chemical reactions close to 0%. The maximum deviation measured for the net power generated by the gas turbine is instead about 1.2%.

It can therefore be said that the calculation model settles to a deviation from the HYSYS® parameters of always less than 1.5% in worst-case conditions: those listed, in fact, are the maximum error parameters detected, and as analysed by the results, the model behaves better at low ambient temperatures, assuming deviations from HYSYS® of 0%. However, the accuracy of HYSYS® have also to be considered. In this project, the deviations of this software results from reality are not analysed, as this data is not directly accessible to the user when developing a simulation, but it is important to bear in mind its existence.

The same applies to the validation of the digital twin for real turbines with Thermoflow©: the final model will therefore acquire a deviation equal to the deviation from HYSYS® for the calculation method and the deviation from Thermoflow© for the results of real turbines, with a maximum value of 1.2%.

Furthermore, it is important to consider the fact that the value of the polytropic and isentropic efficiencies are found through an iterative process in GasTurb©: although the model in Python is not directly validated through a comparison with GasTurb©, but only acquires the values of isentropic and polytropic efficiency, there is still the possibility of a deviation from reality.

CHAPTER

FIVE

CONCLUSIONS

The project presented in this work deals with the modeling and development of a digital twin for two gas turbines for energy production in the off-shore field, that finds its justification in the increasing pressure on global energy industries to develop efficient operating and plant control methods, through the digitisation of components, machinery and the industry as a whole. The project planned a development of the digital twin from the scratch, starting with the modeling of thermodynamic calculation methods for turbine components, through object-oriented programming in Python.

Through the development of the same case study in Python and in Aspen HYSYS®, a validation of the calculation efficiency of the classes in Python is carried out by comparing it with the results obtained from the simulation in HYSYS®, both for the development of the turbine in design operating conditions and for the analysis of the accuracy of the model in off-design operating conditions by changing the ambient temperature. The results show a good behaviour of the model in comparison with the parameters in HYSYS®, with a percentage deviation always less than 1.2%, found to be the maximum error value.

Next, the actual digital twin model for the two gas turbines GE LM2500 and GE LM6000 is developed in design conditions. As a reference, a Thermoflow© simulation model is built, whose results are in part taken as an input to the Python model, and in part used as a baseline reference values to validate the Python model itself; the values taken as an input are: compression ratio, air flow rate, fuel flow rate, intake and exhaust pressure losses and air leakage at the compressor. The values taken as reference for validating the Python model, instead, are: gas turbine net power, exhaust gas temperature, heat rate and gas turbine efficiency.

For the complete development of the digital twin, an iterative process is implemented in GasTurb© for the determination of the polytropic and isentropic efficiencies of the turbine components, as these values are not known outside the manufacturer: for the construction of the model in GasTurb©, the values resulting from the simulation in Thermoflow© were used. The model thus developed in design conditions in Python is shows for both the gas turbines a percentage deviation from Thermoflow©, of less than 1.2%.

Subsequently, a simplified approach for the off-design model is also developed for the GE LM2500 turbine by changing the ambient temperature, advancing the hypothesis that as the ambient temperature changes, a change in the opening angle of the IGVs occurs, justified by the deviation between the trend of the inlet air flow rate calculated considering the corrected flow rate constant and the air flow generated by Thermoflow©: this hypothesis considers also a change in the efficiencies of compressors and turbine.

As was done for the design model, the same inputs are taken from Thermoflow© to build the off-design model in Python, as well as the same outputs to validate the results, which settle at a percentage deviation close to 0%: in any case, the hypothesis of the variation of the angle of the IGVs is to be verified in future developments, considering a possible influence of the shaft speed.

Finally, a first simplified approach of development of a compressor performance indicator is presented, in order to show an example of implementation of condition monitoring for predictive maintenance purposes, using the GE LM6000 gas turbine field data: this indicator considers the variation of the compressor outlet temperature in comparison with temperature calculated by the model, in order to detect an eventual reduction in the efficiency of the component. This approach has to be further studied in deep, to understand if there could be an actual degradation or just a change in the operating conditions.

The digital twin, both in its ability to predict the thermodynamic behaviour of the components and in its ability to simulate the GE LM2500 and GE LM6000 gas turbines in design conditions and the GE LM2500 turbine in off-design conditions, produced results that differed from the reference simulation models by a very small percentage, less than 1% overall. However, it is important to note that this model is therefore sensitive to the errors and deviations of the simulation models on which it was validated, the quantitative determination of which in terms of deviation from reality could be the starting point for future work. Furthermore, for the construction of the digital twin for real turbines, the results of the model in Thermoflow© were not only used as a reference for validation, but also partly as input for the parameterisation of the model in Python: in the design and development of digital twins, it is common practice to use reference models as input for data definition and as a baseline for validation of results, as there is often a lack of reliable and consistent historical data on which to base the design of the model itself.

The digital twin developed, even if its design limitations make it somewhat dependent on the other models in Thermoflow©, HYSYS® and GasTurb©, nevertheless offers the possibility of being adapted and shaped to the needs of each reference turbine, making it possible not only to simulate the behaviour in specific conditions, but also to provide possible answers on the state of the components and their degradation, and, with the appropriate future developments, to predict possible interventions *ad hoc*, through real data and on time connection, as outlined in the final project objectives at the beginning of this work.

CHAPTER

SIX

FUTURE WORK

The work done in this project has opened up a number of possible future developments, to the point of dedicating a section to it. The two real turbines GE LM2500 and GE LM6000 are modelled through the development of simulations in Thermoflow© and GasTurb©, but the core of a gas turbine are the compressor and turbine maps. Therefore, in the first instance, the model can be thought of as being developed through a derivation of the compressor and turbine maps for the two turbines.

As outlined in the introduction to this work only the manufacturer is in possession of the compressor and turbine maps, as they are the result of considerably expensive experiments and simulations: there are thus several methods in the engineering literature that would allow the estimation of such maps, in particular of the compressor, such as a CFD model simulation. In [37] a 3D CDF analysis simulation method for compressor map is presented.

Building maps not only allows accurate identification of polytropic and isentropic efficiency values of components, but also allows verification of turbine behaviour in off-design conditions: another important input for a future work, in fact, considers an in-depth development of the off-design model for the GE LM6000 turbine and a possible validation or otherwise of the considerations made for the off-design of the GE LM2500 turbine.

The hypothesis put forward in this project of the variation of IGVs opening angle as the ambient temperature changes in off-design conditions in full-load could be investigated further, and it could be verified whether this is actually an implemented operating condition or whether the deviation between the air flow rate calculated according to the hypothesis of a constant corrected flow rate and the air flow rate in Thermoflow© can be justified by other hypotheses, such as the variation of the shaft rotation speed.

Further future work to be implemented also relates to the implementation of condition monitoring for predictive maintenance: firstly, it could be analysed on a time scale of years whether the identified compressor performance indicator shows an increasing trend, and if so, whether this is due to an actual degradation of the component or a change in operating conditions. To analyse the presence or absence of degradation a comparison between the indicator trend and maintenance actions implemented on the compressor and the gas turbine as a whole can be done.

Through further analysis of the field data, other performance indicators could also be identified, with the aim of analysing and identifying further variations in the behaviour of the turbine components: for example, performance indicators for generated power and exhaust gas temperature.

REFERENCES

[1] Ahmad K. Sleiti, Jayanta S. Kapat, and Ladislav Vesely. “Digital twin in energy industry: Proposed robust digital twin for power plant and other complex capital-intensive large engineering systems”. In: *Energy Reports* 8 (2022), pp. 3704–3726. ISSN: 2352-4847. DOI: 10.1016/j.egyr.2022.02.305.

[2] *International energy outlook 2019. Report 2020.* Tech. rep. EIA, 2022.

[3] R. Stark and T. Damerau. “Digital Twin”. In: (2019). DOI: 10.1007/978-3-642-35950-7_16870-1.

[4] H. Liu et al. “Digital Twin-Driven Machine Condition Monitoring: A Literature Review”. In: *Journal of Sensors* (2022). DOI: 10.1155/2022/6129995.

[5] H. Minghui 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: 10.1016/j.cja.2022.07.021.

[6] HIH Saravanamuttoo, H. Cohen, and GFG Rogers. *Gas Turbine Theory.* Pearson, 2013. ISBN: 978-81-7758-902-3.

[7] R.K. Rajput. *Engineering Thermodynamics.* LAXMI PUBLICATIONS (P) LTD, 2007. ISBN: 978-0-7637-8272-6.

[8] Meherwan P. Boyce. “3 - Compressor and Turbine Performance Characteristics”. In: *Gas Turbine Engineering Handbook (Fourth Edition)*. Ed. by M. P. Boyce. Fourth Edition. Oxford: Butterworth-Heinemann, 2012, pp. 139–176. ISBN: 978-0-12-383842-1. DOI: 10.1016/B978-0-12-383842-1.00003-2.

[9] L. O. Nord and O. Bolland. *Thermal Power Generation.* 2022.

[10] *How AI improves axial compressor maps generation.* URL: https://blog.softinway.com/how-ai-improves-axial-compressor-map-generation/.

[11] S. Gülen. *Gas Turbines for Electric Power Generation.* Cambridge: Cambridge University Press, 2019. DOI: 10.1017/9781108241625.

[12] S. Sanaye and S. Hosseini. “Off-design performance improvement of twin-shaft gas turbine by variable geometry turbine and compressor besides fuel control”. In: *Proceedings of the Institution of Mechanical Engineers, Part A: Journal of Power and Energy* (2020).

[13] C. E. Dole et al. *Flight Theory and Aerodynamics: a practical guide for operational safety.* John Wiley & Sons, Inc, New York, NY, 1981.

[14] D. Vyncke-Wilson. "Advantages of aeorderivative gas turbines: technical & operational consideration on equipment selection". In: 2013.

[15] *GE Aerospace*. URL: `https://www.ge.com/`.

[16] G. H. Badeer. "GE aeroderivative gas turbines-design and operating features". In: *GER-3695E, GE Power Systems, Evendale, OH* (2000).

[17] G. G. Ol'khovskii. "G.G. Aeroderivative GTUs for Power Generation (Overview)". In: *Therm. Eng. 68* (2021).

[18] "Chapter 3 - Compressors". In: *Forsthoffer's More Best Practices for Rotating Equipment*. Ed. by M. S. Forsthoffer. Butterworth-Heinemann, 2017, pp. 73–185. ISBN: 978-0-12-809277-4.

[19] M. Olausson. *Turbomachinery aeroacoustic calculations using nonlinear methods*. Chalmers Tekniska Hogskola (Sweden), 2011.

[20] I. López-Paniagua et al. "Step by Step Derivation of the Optimum Multistage Compression Ratio and an Application Case". In: *Entropy (Basel)* (2020). DOI: `10.3390/e22060678.PMID:33286450;PMCID:PMC7517211.`.

[21] G. Venkatarathnam. *Cryogenic Mixed Refrigerant Processes*. Springer Science & Business Media, 2010.

[22] M. Tahan et al. "Performance-based health monitoring, diagnostics and prognostics for condition-based maintenance of gas turbines: A review". In: *Applied Energy* 198 (2017), pp. 122–144. ISSN: 0306-2619. DOI: `10.1016/j.apenergy.2017.04.048`.

[23] F. Safiyullah et al. "Prediction on performance degradation and maintenance of centrifugal gas compressors using genetic programming". In: *Energy* 158 (2018), pp. 485–494. ISSN: 0360-5442. DOI: `10.1016/j.energy.2018.06.051`.

[24] D. W. Kang and T. S. Kim. "Model-based performance diagnostics of heavy-duty gas turbines using compressor map adaptation". In: *Applied Energy* 212 (2018), pp. 1345–1359. ISSN: 0306-2619. DOI: `10.1016/j.apenergy.2017.12.126`.

[25] Y.G. Li et al. "Application of Gas Path Analysis to Compressor Diagnosis of an Industrial Gas Turbine Using Field Data". In: June 2014. DOI: `10.1115/GT2014-25330`.

[26] H. Jeong et al. "Fault detection and identification method using observer-based residuals". In: *Reliability Engineering & System Safety* 184 (2019). Impact of Prognostics and Health Management in Systems Reliability and Maintenance Planning, pp. 27–40. ISSN: 0951-8320. DOI: `10.1016/j.ress.2018.02.007`.

[27] H. Hanachi et al. "A Physics-Based Modeling Approach for Performance Monitoring in Gas Turbine Engines". In: *IEEE Transactions on Reliability* 64.1 (2015), pp. 197–205. DOI: `10.1109/TR.2014.2368872`.

[28] *NeqSim*. 2023. URL: `https://equinor.github.io/neqsimhome/` (visited on 05/22/2023).

[29] AspenTech Inc. *HYSYS®*. Version v12.1. 2023. URL: `https://www.aspentech.com/en/products/engineering/aspen-hysys`.

[30] Aspen Technology. *Hysys 2004.2, Operation Guide*. 2004.

[31] *Minimise Gibbs Free Energy*. 2011. URL: http://apmonitor.com/wiki/index.php/Apps/GibbsFreeEnergy.

[32] Thermoflow Inc. *Thermoflow©*. Version 30. 2022. URL: info@thermoflow.com.

[33] GasTurb GmBH. *GasTurb©*. Version 14. 2023. URL: https://www.gasturb.com/.

[34] W. Zhu et al. "Improvement of part-load performance of gas turbine by adjusting compressor inlet air temperature and IGV opening". In: *Frontiers in Energy* 16.6, 1000 (2022), p. 1000. DOI: 10.1007/s11708-021-0746-z.

[35] G.F. Aker and H.I.H Saravanamuttoo. "Predicting Gas Turbine Performance Degradation Due to Compressor Fouling Using Computer Simulation Techniques". In: *Journal of Engineering for Gas Turbines and Power* 111.2 (Apr. 1989), pp. 343–350. ISSN: 0742-4795. DOI: 10.1115/1.3240259. eprint: https://asmedigitalcollection.asme.org/gasturbinespower/article-pdf/111/2/343/5750748/343_1.pdf.

[36] C.B Meher-Homji, A. Bromley, et al. "Gas Turbine Axial Compressor Fouling And Washing." In: *Proceedings of the 33rd turbomachinery symposium*. Texas A&M University. Turbomachinery Laboratories. 2004.

[37] C. Janke, D. Bestle, and B. Becker. "Compressor map computation based on 3D CFD analysis". In: *CEAS Aeronaut J 6* (2015). DOI: 10.1007/s13272-015-0159-y.

APPENDICES

.1 Classes

```python
import math
import matplotlib.pyplot as plt
import numpy as np
from neqsim.thermo.thermoTools import fluid, TPflash, PHflash
from neqsim.standards import ISO6976
import copy

class Stream():
    def __init__(self):
        self.fluid = None
        self.flow_rate = None
        self.temperature = None
        self.pressure = None

    def set_fluid(self, fluid):
        self.fluid = fluid

    def set_flow_rate(self, flow_rate, units):
        if units == 'kg/hr':
            self.flow_rate = flow_rate
        elif units == 'kg/sec':
            self.flow_rate = flow_rate * 3600
        else:
            print("ERROR no units found")

    def set_temperature(self, temperature, units):
        if units == 'K':
            self.temperature = temperature
        if units == 'C':
            self.temperature = temperature + 273.15

    def set_pressure(self, pressure):
        self.pressure = pressure
```

```

    def get_flow_rate(self, units):
        if units == 'kg/hr':
            return self.flow_rate
        elif units == 'kg/sec':
            return self.flow_rate / 3600
        else:
            print(f"ERROR no units found for flow rate units:
            {units} in {self}")

    def get_temperature(self, units):
        if units == 'K':
            return (self.temperature)
        elif units == 'C':
            return (self.temperature - 273.15)

    def get_pressure(self, units):
        if units == 'bara':
            return (self.pressure)

    def calculate(self):
        self.fluid.setTemperature(self.temperature, "K")
        self.fluid.setPressure(self.pressure, 'bara')
        self.fluid.setTotalFlowRate(self.flow_rate, 'kg/hr')
        TPflash(self.fluid)
        self.fluid.initProperties()

    def get_LCV(self):
        iso6976 = ISO6976(self.fluid)
        iso6976.setReferenceType('mass')
        iso6976.setVolRefT(15.0)
        iso6976.setEnergyRefT(25.0)
        iso6976.calculate()
        return iso6976.getValue("InferiorCalorificValue") * 1e3

class Compressor():
    def __init__(self):
        self.t_out = None
        self.t_ideal_out = None
        self.p_out = None
        self.work = None
        self.stream = None
        self.pol_efficiency = None
        self.ise_efficiency = None
        self.polytropic_head = None
        self.pressure_ratio = None
        self.ambient_temperature = None
```

```
        self.ambient_pressure = None
        self.k_ambient = None
        self.losses = None

    def set_stream(self, stream):
        self.stream = stream
        self.ambient_temperature = self.stream.get_temperature('K')
        self.k_ambient = self.stream.fluid.getGamma2()
        self.ambient_pressure = self.stream.get_pressure('bara')
        flow_rate = self.stream.get_flow_rate('kg/sec')
        self.stream.set_flow_rate(flow_rate - self.losses,'kg/sec')

    def set_losses(self,losses):
        self.losses = losses

    def set_pol_efficiency(self, pol_efficiency):
        self.pol_efficiency = pol_efficiency

    def set_isentropic_efficiency(self, ise_efficiency):
        self.ise_efficiency = ise_efficiency

    def calc_isentropic_efficiency(self):
        kappa = self.stream.fluid.getGamma2()
        num_exp = ((kappa - 1) / kappa)
        den_exp = ((kappa - 1) / (kappa * self.pol_efficiency))
        pressure_ratio = self.get_p_out('bara') / self.ambient_pressure
        self.ise_efficiency = (pow(pressure_ratio, num_exp) - 1) \
                              / (pow(pressure_ratio, den_exp) - 1)

    def get_isentropic_efficiency(self):
        return (self.ise_efficiency)

    def calc_polytropyc_efficiency(self):
        P2 = self.get_p_out('bara')
        P1 = self.ambient_pressure
        k = self.stream.fluid.getGamma2()
        i_p = self.ise_efficiency
        self.pol_efficiency = ((k - 1) * np.log(P2 / P1)) / (
                k * np.log((1 / i_p) * ((P2 / P1) ** ((k - 1) / k) - 1) + 1))

    def get_polytripic_efficiency(self):
```

```
        return (self.pol_efficiency)

    def set_p_out(self, p_out):
        self.p_out = p_out

    def get_p_out(self, units):
        if units == 'bara':
            return (self.p_out)

    def set_t_out(self, t_out, units):
        if units == 'C':
            self.t_out = t_out
        elif units == 'K':
            self.t_out = t_out + 273.15

    def calc_ideal_outlet_temp(self):
        base = (self.get_p_out('bara') /
        self.stream.get_pressure('bara'))
        kappa = self.stream.fluid.getGamma2()
        exp = ((1 - kappa) / kappa)
        x = pow(base, exp)
        self.t_ideal_out = self.stream.get_temperature('K') * x
        return (self.t_ideal_out)

    def calc_outlet_temperature(self):
        if self.pol_efficiency != None:
            base = (self.get_p_out('bara') /
            self.stream.get_pressure('bara'))
            kappa = self.stream.fluid.getGamma2()
            exp = ((kappa - 1) / (kappa * self.pol_efficiency))
            x = pow(base, exp)
            self.t_out = self.stream.get_temperature('K') * x
        elif self.pol_efficiency == None:
            self.calc_ideal_outlet_temp()
            delta = self.t_ideal_out -
            self.stream.get_temperature('K')
            self.t_out = delta / self.ise_efficiency \
                         + self.stream.get_temperature('K')

    def get_outlet_temperature(self, units):
        if units == 'K':
            return (self.t_out)
        if units == 'C':
            return (self.t_out - 273.15)

    def get_work(self, units):
        flow_rate = self.stream.get_flow_rate('kg/sec')
```

```
        delta_T = self.get_outlet_temperature('K') - self.ambient_temperature
        cp = self.stream.fluid.getPhase(0).getCp('kJ/kgK')
        self.work = (flow_rate * delta_T) * cp
        if units == 'W':
            return (self.work * 1e3)
        elif units == 'kW':
            return (self.work)
        elif units == 'MW':
            return (self.work / 1e3)

    def get_outlet_stream(self):
        self.outlet_stream = copy.copy(self.stream)
        self.outlet_stream.set_fluid(self.stream.fluid.clone())
        self.outlet_stream.set_pressure(self.p_out)
        self.outlet_stream.set_temperature(self.t_out, 'K')
        self.outlet_stream.calculate()
        return (self.outlet_stream)

    def calc(self):
        if self.p_out is not None:
            if self.pol_efficiency is not None:
                self.calc_outlet_temperature()
            elif self.ise_efficiency is not None:
                self.calc_outlet_temperature()

    def visualize_work(self, units):
        if units == 'kW':
            return(self.work)

    def compression_by_steps(self, steps):
        total_p = self.p_out
        iteration = 0
        number_of_steps = steps
        pressure_of_step = (self.p_out - self.stream.get_pressure('bara')) / number_of_steps
        self.p_out = self.stream.get_pressure('bara')
        temperature_step_before = self.ambient_temperature
        self.work = 0

        while iteration < number_of_steps:
            if self.p_out < total_p:
                self.p_out = self.p_out + pressure_of_step
                self.calc()
                new_iteration_stream = self.get_outlet_stream()
                self.stream = new_iteration_stream
```

```
                delta_T = self.get_outlet_temperature('K') - temperature_step_before
                temperature_step_before = self.t_out
                flow_rate = self.stream.get_flow_rate('kg/sec')
                cp = self.stream.fluid.getPhase(0).getCp('kJ/kgK')
                work_new_stage = (flow_rate * delta_T) * cp
                self.work = self.work + work_new_stage
            iteration = iteration + 1

    def solve_polytropic_efficiency(self):
        k = self.k_ambient
        P_out = self.p_out
        T_in = self.ambient_temperature
        P_in = self.ambient_pressure
        T_out = self.t_out
        a = 1/(math.log((T_out/T_in),(P_out/P_in)))
        x = (a*(k-1))/k
        return(x)

class Combustor():
    def __init__(self):
        self.air = None
        self.TIT = None
        self.fuel = None
        self.enthalpy = None
        self.cp = None
        self.Q = None
        self.outlet_stream = None
        self.reaction_fluid = None
        self.TIT_reaction = None
        self.p3 = None
        self.TIT_noPHflash = None
        self.T2 = None

    def set_stream_air(self, air):
        self.air = air
        self.T2 = self.air.get_temperature('K')

    def set_stream_fuel(self, fuel):
        self.fuel = fuel

    def calc_enthalpy(self):
        enthalpy_air = self.air.fluid.getEnthalpy()
        enthalpy_fuel = self.fuel.get_LCV() * self.fuel.get_flow_rate('kg/sec')
        self.enthalpy = enthalpy_air + enthalpy_fuel
```

```
        return (self.enthalpy)

    def set_enthalpy(self, enthalpy):
        enthalpy_fuel = self.fuel.get_LCV() *
        self.fuel.get_flow_rate('kg/sec')
        self.enthalpy = enthalpy + enthalpy_fuel

    def chemical_reaction(self):
        # AIR:
        number_of_components_air =
        self.air.fluid.getNumberOfComponents()
        names_air = [self.air.fluid.getComponent(i).getName()
                     for i in range(number_of_components_air)]
        molar_fractions_air =
        [self.air.fluid.getComponent(i).getx()
                               for i in
                               range(number_of_components_air)]
        air_dictionary = {}

        for i in range(number_of_components_air):
            air_dictionary[names_air[i]] =
            molar_fractions_air[i]

        molar_mass_mix = air_dictionary['oxygen'] * 31.998 +
        air_dictionary[
            'nitrogen'] * 28.013
        weight_fractionO2 = (air_dictionary['oxygen'] * 31.9989)
        / molar_mass_mix
        weight_fractionN2 = (air_dictionary['nitrogen'] *
        28.013) / molar_mass_mix
        massO2 = weight_fractionO2 *
        self.air.get_flow_rate('kg/sec')
        massN2 = weight_fractionN2 *
        self.air.get_flow_rate('kg/sec')
        molN2 = (massN2) * (1 / (28.013 / 1000))
        molO2 = (massO2) * (1 / (31.998 / 1000))

        # FUEL:
        number_of_components_fuel =
        self.fuel.fluid.getNumberOfComponents()
        names_fuel = [self.fuel.fluid.getComponent(i).getName()
                      for i in range(number_of_components_fuel)]
        molar_fractions_fuel =
        [self.fuel.fluid.getComponent(i).getx()
                                for i in
                                range(number_of_components_fuel)]

```

```
        fuel_dictionary = {}

        for i in range(number_of_components_fuel):
            fuel_dictionary[names_fuel[i]] = molar_fractions_fuel[i]

        molar_mass_mix_ng = self.fuel.fluid.getMolarMass() * 1000

        nitrogen = (self.fuel.fluid.getPhase(0).getComponent(
            'nitrogen').getMolarMass()) * 1000
        weight_fraction_nitrogen = (fuel_dictionary['nitrogen'] * nitrogen) / molar_mass_mix_ng
        mass_nitrogen = weight_fraction_nitrogen * self.fuel.get_flow_rate('kg/sec')
        mol_nitrogen = mass_nitrogen * (
                1 / self.fuel.fluid.getPhase(0).getComponent('nitrogen').getMolarMass())
        N2_nitrogen = mol_nitrogen * fuel_dictionary['nitrogen']

        co2 = (self.fuel.fluid.getPhase(0).getComponent('CO2').getMolarMass()) * 1000
        weight_fraction_co2 = (fuel_dictionary['CO2'] * co2) / molar_mass_mix_ng
        mass_co2 = weight_fraction_co2 * self.fuel.get_flow_rate('kg/sec')
        mol_co2 = mass_co2 * (1 / self.fuel.fluid.getPhase(0).getComponent('CO2')
                              .getMolarMass())
        CO2_co2 = mol_co2 * fuel_dictionary['CO2']

        ########################### METHANE #############################

        methane = (self.fuel.fluid.getPhase(0).getComponent('methane').getMolarMass()) * 1000
        weight_fraction_methane = (fuel_dictionary['methane'] * methane) / molar_mass_mix_ng
        mass_methane = weight_fraction_methane * self.fuel.get_flow_rate('kg/sec')
        mol_methane = mass_methane * (
                1 / self.fuel.fluid.getPhase(0).getComponent('methane')
                .getMolarMass())
        limCH4 = mol_methane * (2 / 1)
```

```
        limO2 = molO2 * (2 / 2)

        molCO2 = mol_methane * (1 / 1)
        molH20 = mol_methane * (2 / 1)
        molO2_not_reacted_methane = molO2 - mol_methane * 2
        total_moles = molCO2 + molH20 +
        molO2_not_reacted_methane + molN2
        O2_methane = (molO2_not_reacted_methane / total_moles) *
        fuel_dictionary['methane']
        CO2_methane = (molCO2 / total_moles)
        H20_methane = (molH20 / total_moles)
        molO2 = molO2 - mol_methane * 2

        ########################### ETHANE
        #############################

        ethane =
        (self.fuel.fluid.getPhase(0).getComponent('ethane').getMolarMass())
        * 1000
        weight_fraction_ethane = (fuel_dictionary['ethane'] *
        ethane) / molar_mass_mix_ng
        mass_ethane = weight_fraction_ethane *
        self.fuel.get_flow_rate('kg/sec')
        mol_ethane = mass_ethane * (
                1 /
                self.fuel.fluid.getPhase(0).getComponent('ethane')
                .getMolarMass())
        limC2H6 = mol_ethane * (3 / 1)
        limO2 = molO2 * (3 / 5 / 2)

        molCO2 = mol_ethane * (2 / 1)
        molH20 = mol_ethane * (3 / 1)
        molO2_not_reacted = molO2 - mol_ethane * (7 / 2)
        total_moles = molCO2 + molH20 + molO2_not_reacted +
        molN2
        O2_ethane = (molO2_not_reacted / total_moles) *
        fuel_dictionary['ethane']
        CO2_ethane = (molCO2 / total_moles)
        H20_ethane = (molH20 / total_moles)
        molO2 = molO2 - mol_ethane * (7 / 2)

        ########################### PROPANE
        #############################

        propane =
        (self.fuel.fluid.getPhase(0).getComponent('propane').getMolarMass())
        * 1000
```

```
        weight_fraction_propane = (fuel_dictionary['propane'] * propane) / molar_mass_mix_ng
        mass_propane = weight_fraction_propane * self.fuel.get_flow_rate('kg/sec')
        mol_propane = mass_propane * (
                1 / self.fuel.fluid.getPhase(0).getComponent('propane')
                .getMolarMass())
        limC3H8 = mol_propane * (4 / 1)
        limO2 = molO2 * (4 / 5)

        molCO2 = mol_propane * (3 / 1)
        molH2O = mol_propane * (4 / 1)
        molO2_not_reacted = molO2 - 5 * mol_propane
        total_moles = molCO2 + molH2O + molO2_not_reacted + molN2
        O2_propane = (molO2_not_reacted / total_moles) * fuel_dictionary['propane']
        CO2_propane = (molCO2 / total_moles)
        H2O_propane = (molH2O / total_moles)
        molO2 = molO2 - mol_propane * 5

        ########################### N - BUTANE ##############################

        n_butane = (self.fuel.fluid.getPhase(0).getComponent('n-butane').getMolarMass()) * 1000
        weight_fraction_nbutane = (fuel_dictionary['n-butane'] * n_butane) / molar_mass_mix_ng
        mass_nbutane = weight_fraction_nbutane * self.fuel.get_flow_rate('kg/sec')
        mol_nbutane = mass_nbutane * (
                1 / self.fuel.fluid.getPhase(0).getComponent('n-butane')
                .getMolarMass())
        limC4H10 = mol_nbutane * (5 / 1)
        limO2 = molO2 * (5 / 13 / 2)

        if fuel_dictionary['n-butane'] is not None and all(
                v is None for k, v in fuel_dictionary.items() if k != "n-butane"):
            molCO2 = mol_nbutane * (4.819 / 1)
            molH2O = mol_nbutane * (3 / 1)
            molO2_not_reacted = molO2 - mol_nbutane * 6.5
            total_moles = molCO2 + molH2O + molN2 * fuel_dictionary['n-butane'] \
```

```
                      + molO2_not_reacted
            O2_nbutane = (molO2_not_reacted / total_moles) *
            fuel_dictionary['n-butane']
            N2_nbutane = (molN2 / total_moles) *
            fuel_dictionary['n-butane']
            CO2_nbutane = (molCO2 / total_moles)
            H2O_nbutane = (molH2O / total_moles)
            molO2 = molO2 - mol_nbutane * 6.5
        else:
            molCO2 = mol_nbutane * (4.819 / 1)
            molH2O = mol_nbutane * (3 / 1)
            molO2_not_reacted = molO2 - mol_nbutane * 6.5
            total_moles = molCO2 + molH2O + molN2 *
            fuel_dictionary['n-butane'] \
                      + molO2_not_reacted
            O2_nbutane = (molO2_not_reacted / total_moles) *
            fuel_dictionary['n-butane']
            N2_nbutane = (molN2 / total_moles) *
            fuel_dictionary['n-butane']
            CO2_nbutane = (molCO2 / total_moles)
            H2O_nbutane = (molH2O / total_moles)
            molO2 = molO2 - mol_nbutane * 6.5

        ########################### I - BUTANE
        #############################

        i_butane =
        (self.fuel.fluid.getPhase(0).getComponent('i-butane').getMolarMass())
        * 1000
        weight_fraction_ibutane = (fuel_dictionary['i-butane'] *
        i_butane) / molar_mass_mix_ng
        mass_ibutane = weight_fraction_ibutane *
        self.fuel.get_flow_rate('kg/sec')
        mol_ibutane = mass_ibutane * (
                1 /
                self.fuel.fluid.getPhase(0).getComponent('i-butane')
                .getMolarMass())
        limC4H10 = mol_ibutane * (5 / 1)
        limO2 = molO2 * (5 / 13 / 2)

        molCO2 = mol_ibutane * (4 / 1)
        molH2O = mol_ibutane * (5 / 1)
        molO2_not_reacted = molO2 - mol_ibutane * 6.5
        total_moles = molCO2 + molH2O + molN2 +
        molO2_not_reacted
        O2_ibutane = (molO2_not_reacted / total_moles) *
        fuel_dictionary['i-butane']
```

```
        CO2_ibutane = (molCO2 / total_moles)
        H2O_ibutane = (molH2O / total_moles)
        molO2 = molO2 - mol_ibutane * 6.5

        ########################### I - PENTANE
        #############################

        i_pentane =
        (self.fuel.fluid.getPhase(0).getComponent('i-pentane').getMolarMass())
        * 1000
        weight_fraction_ipentane = (fuel_dictionary['i-pentane']
        * i_pentane) \
                                   / molar_mass_mix_ng
        mass_ipentane = weight_fraction_ipentane *
        self.fuel.get_flow_rate('kg/sec')
        mol_ipentane = mass_ipentane * (
                1 /
                self.fuel.fluid.getPhase(0).getComponent('i-pentane')
                .getMolarMass())
        limC5H12 = mol_ipentane * (6 / 1)
        limO2 = molO2 * (6 / 8)

        molCO2 = mol_ipentane * (5 / 1)
        molH2O = mol_ipentane * (6 / 1)
        molO2_not_reacted = molO2 - mol_ipentane * 8
        total_moles = molCO2 + molH2O + molN2 +
        molO2_not_reacted
        O2_ipentane = (molO2_not_reacted / total_moles) *
        fuel_dictionary['i-pentane']
        CO2_ipentane = (molCO2 / total_moles)
        H2O_ipentane = (molH2O / total_moles)
        molO2 = molO2 - mol_ipentane * 8

        ########################### N - PENTANE
        #############################

        n_pentane =
        (self.fuel.fluid.getPhase(0).getComponent('n-pentane').getMolarMass())
        * 1000
        weight_fraction_npentane = (fuel_dictionary['n-pentane']
        * n_pentane) / molar_mass_mix_ng
        mass_npentane = weight_fraction_npentane *
        self.fuel.get_flow_rate('kg/sec')
        mol_npentane = mass_npentane * (
                1 /
                self.fuel.fluid.getPhase(0).getComponent('n-pentane')
                .getMolarMass())
```

```
        limC5H12 = mol_npentane * (6 / 1)
        limO2 = molO2 * (6 / 8)

        molCO2 = mol_npentane * (5 / 1)
        molH2O = mol_npentane * (6 / 1)
        molO2_not_reacted = molO2 - mol_npentane * 8
        total_moles = molCO2 + molH2O + molN2  + molO2_not_reacted
        O2_npentane = (molO2_not_reacted / total_moles) * fuel_dictionary['n-pentane']
        CO2_npentane = (molCO2 / total_moles)
        H2O_npentane = (molH2O / total_moles)
        molO2 = molO2 - mol_npentane * 8

        ########################### N - HEXANE ##############################

        n_hexane = (self.fuel.fluid.getPhase(0).getComponent('n-hexane').getMolarMass()) * 1000
        weight_fraction_nhexane = (fuel_dictionary['n-hexane'] * n_hexane) / molar_mass_mix_ng
        mass_nhexane = weight_fraction_nhexane * self.fuel.get_flow_rate('kg/sec')
        mol_nhexane = mass_nhexane * (
                1 / self.fuel.fluid.getPhase(0).getComponent('n-hexane')
                .getMolarMass())
        limC6H14 = mol_nhexane * (7 / 1)
        limO2 = molO2 * (7 / 19 / 2)

        molCO2 = mol_nhexane * (6 / 1)
        molH2O = mol_nhexane * (7 / 1)
        molO2_not_reacted = molO2 - mol_nhexane
        total_moles = molCO2 + molH2O + molN2 + molO2_not_reacted
        O2_nhexane = (molO2_not_reacted / total_moles) * fuel_dictionary['n-hexane']
        CO2_nhexane = (molCO2 / total_moles)
        H2O_nhexane = (molH2O / total_moles)
        molO2 = molO2 - (19 / 2) * mol_nhexane

        mfO2 = O2_methane + O2_ethane + O2_propane + O2_ibutane + \
               O2_nbutane + O2_ipentane + O2_npentane + O2_nhexane

```

```python
        mfN2 = air_dictionary['nitrogen'] + fuel_dictionary['nitrogen']

        mfCO2 = CO2_methane + CO2_ethane + CO2_propane + CO2_ibutane + \
                CO2_nbutane + CO2_ipentane + CO2_npentane + CO2_nhexane + \
                fuel_dictionary['CO2']

        mfH2O = H2O_methane + H2O_ethane + H2O_propane + \
                H2O_ibutane + H2O_nbutane + H2O_ipentane + H2O_npentane + H2O_nhexane

        to_turbine = fluid('srk')
        to_turbine.addComponent('oxygen', mfO2)
        to_turbine.addComponent('nitrogen', mfN2)
        to_turbine.addComponent('CO2', mfCO2)
        to_turbine.addComponent('H2O', mfH2O)
        self.reaction_fluid = Stream()
        self.reaction_fluid.set_fluid(to_turbine)
        self.reaction_fluid.set_pressure(self.air.get_pressure('bara'))
        self.reaction_fluid.set_temperature(self.air.get_temperature('K'), 'K')
        self.reaction_fluid.set_flow_rate(self.air.get_flow_rate('kg/sec') +
                                          self.fuel.get_flow_rate('kg/sec'), 'kg/sec')
        self.reaction_fluid.calculate()

    def calc_TIT_reaction(self):
        self.chemical_reaction()
        combustion_fluid = self.reaction_fluid.fluid.clone()
        PHflash(combustion_fluid, self.enthalpy)
        self.TIT_reaction = combustion_fluid.getTemperature('k')

    def get_TIT_reaction(self, units):
        if units == 'K':
            return (self.TIT_reaction)
        elif units == 'C':
            return (self.TIT_reaction - 273.15)

    def calc_TIT(self):
        combustion_fluid = self.air.fluid.clone()
```

```
        PHflash(combustion_fluid, self.enthalpy)
        self.TIT = combustion_fluid.getTemperature('K')

    def get_TIT(self, units):
        if units == 'K':
            return (self.TIT)
        elif units == 'C':
            return (self.TIT - 273.15)

    def set_pressure(self, p3):
        self.p3 = p3

    def get_pressure(self):
        pressure_drop = 0.015
        self.p3 = self.air.get_pressure('bara') -
        self.air.get_pressure('bara') * pressure_drop
        return (self.p3)

    def cp_mix(self):
        total_flow = self.air.get_flow_rate('kg/sec') +
        self.fuel.get_flow_rate('kg/sec')
        w_air = self.air.get_flow_rate('kg/sec') / total_flow
        w_fuel = self.fuel.get_flow_rate('kg/sec') / total_flow
        cp_air = w_air *
        (self.air.fluid.getPhase(0).getCp('kJ/kgK'))
        cp_fuel = w_fuel *
        (self.fuel.fluid.getPhase(0).getCp('kJ/kgK'))
        self.cp = cp_air + cp_fuel

    def get_Q(self, units):
        self.cp_mix()
        total_flow = self.air.get_flow_rate('kg/sec') +
        self.fuel.get_flow_rate('kg/sec')
        delta_T = self.get_TIT_reaction('K') -
        self.air.get_temperature('K')
        self.Q =  (self.enthalpy -
        self.reaction_fluid.fluid.getEnthalpy())/1000
        if units == 'W':
            return (self.Q * 1e3)
        elif units == 'kW':
            return (self.Q)
        elif units == 'MW':
            return (self.Q / 1e3)

    def get_outlet_stream(self):
        self.outlet_stream = copy.copy(self.reaction_fluid)
```

```python
        self.outlet_stream.set_fluid(self.reaction_fluid.fluid.clone())
        self.outlet_stream.set_pressure(self.air.get_pressure('bara') - (self.air.get_pressure('bara') * 0.015))
        self.outlet_stream.set_temperature(self.TIT_reaction, 'K')
        self.outlet_stream.set_flow_rate(self.air.get_flow_rate('kg/sec') +
                                         self.fuel.get_flow_rate('kg/sec'), 'kg/sec')
        self.outlet_stream.calculate()
        return (self.outlet_stream)

class Expander():
    def __init__(self):
        self.t_out = None
        self.t_ideal_out = None
        self.p_out = None
        self.work = None
        self.stream = None
        self.pol_efficiency = None
        self.ise_efficiency = None
        self.outlet_stream = None
        self.TIT = None
        self.P3 = None
        self.T4 = None
        self.k_3 = None
        self.defined_work = None
        self.losses = None

    def set_losses(self,losses):
        self.losses = losses

    def set_stream(self, stream):
        self.stream = stream
        self.TIT = self.stream.get_temperature('K')
        self.P3 = self.stream.get_pressure('bara')
        self.k_3 = self.stream.fluid.getGamma2()
        flow_rate = self.stream.get_flow_rate('kg/sec')
        self.stream.set_flow_rate(flow_rate - self.losses,'kg/sec')

    def set_p_out(self, p_out):
        self.p_out = p_out
```

```

    def get_p_out(self, units):
        if units == 'bara':
            return (self.p_out)

    def set_pol_efficiency(self, pol_efficiency):
        self.pol_efficiency = pol_efficiency

    def set_isentropic_efficiency(self, ise_efficiency):
        self.ise_efficiency = ise_efficiency

    def calc_isentropic_efficiency(self):
        kappa = self.stream.fluid.getGamma2()
        num_exp = ((kappa - 1) / kappa) * self.pol_efficiency
        den_exp = (kappa - 1) / kappa
        pressure_ratio = self.get_p_out('bara') / self.P3
        self.ise_efficiency = (1 - pow(pressure_ratio, num_exp)) / \
                              (1 - pow(pressure_ratio, den_exp))

    def calc_polytropyc_efficiency(self):
        P4 = self.get_p_out('bara')
        P3 = self.P3
        k = self.stream.fluid.getGamma2()
        i_p = self.ise_efficiency
        self.pol_efficiency = ((k - 1) * np.log(P4 / P3)) / (
                k * np.log((1 / i_p) * ((P4 / P3) ** ((k - 1) / k) - 1) + 1))

    def get_isentropic_efficiency(self):
        return (self.ise_efficiency)

    def get_pol_efficiency(self):
        return (self.pol_efficiency)

    def calc_ideal_outlet_temp(self):
        base = (self.get_p_out('bara') / self.stream.get_pressure('bara'))
        kappa = self.stream.fluid.getPhase(0).getCp() / self.stream.fluid.getPhase(0).getCv()
        exp = ((1 - kappa) / kappa)
        x = pow(base, exp)
        self.t_ideal_out = self.stream.get_temperature('K') * x
        return (self.t_ideal_out)

    def calc_outlet_temperature(self):
        if self.pol_efficiency != None:
```

```
                base = (self.get_p_out('bara') /
                self.stream.get_pressure('bara'))
                kappa = self.stream.fluid.getGamma2()
                exp = ((kappa - 1) / kappa) * self.pol_efficiency
                x = pow(base, exp)
                self.t_out = self.stream.get_temperature('K') * x
            elif self.pol_efficiency == None:
                delta = self.stream.get_temperature('K') -
                self.t_ideal_out
                self.t_out = -self.ise_efficiency * (delta) +
                self.stream.get_temperature('K')

    def get_outlet_temperature(self, units):
        if units == 'K':
            return (self.t_out)
        if units == 'C':
            return (self.t_out - 273.15)

    def get_work(self, units):
        flow_rate = self.stream.get_flow_rate('kg/sec')
        delta_T = self.TIT - self.get_outlet_temperature('K')
        cp = self.stream.fluid.getPhase(0).getCp('kJ/kgK')
        self.work = (flow_rate * delta_T) * cp
        if units == 'W':
            return (self.work * 1e3)
        elif units == 'kW':
            return (self.work)
        elif units == 'MW':
            return (self.work / 1e3)

    def get_outlet_stream(self):
        self.outlet_stream = copy.copy(self.stream)
        self.outlet_stream.set_fluid(self.stream.fluid.clone())
        self.outlet_stream.set_pressure(self.p_out)
        self.outlet_stream.set_temperature(self.t_out, 'K')
        self.outlet_stream.calculate()
        return (self.outlet_stream)

    def calc(self):
        if self.p_out is not None:
            if self.pol_efficiency is not None:
                self.calc_outlet_temperature()
            elif self.ise_efficiency is not None:
                self.calc_ideal_outlet_temp()
                self.calc_outlet_temperature()

    def calc_p_out_iterations(self, work, units):
```

```
        resetting_stream = self.stream
        self.T4 = self.TIT - (work /
        (self.stream.get_flow_rate('kg/sec') *
                                        self.stream.fluid.getCp('kJ/kgK')))
        self.defined_work = work
        temperature_ratio = self.T4 / self.TIT
        k = self.stream.fluid.getGamma2()
        exponent = (k / (k - 1)) * (1 / self.pol_efficiency)

        self.p_out = self.P3 * ((temperature_ratio) ** exponent)
        self.expansion_by_steps(100)

        iteration = 100
        i = 0
        pressure_list = []
        work_list = []
        tolerance = 10
        while i < iteration:
            self.work = 0
            self.stream = resetting_stream
            self.p_out = self.p_out - 0.001
            pressure_list.append(self.p_out)
            self.expansion_by_steps(100)
            work_list.append(self.work)
            i = i + 1

        closest_value = None
        min_difference = float('inf')
        corresponding_pressure = None

        for i in range(len(work_list)):
            difference = abs(work_list[i]- work)
            if abs(work_list[i]- work) < tolerance:
                if work_list[i] > work:
                    if difference < min_difference:
                        min_difference = difference
                        corresponding_pressure =
                        pressure_list[i]

                        print('The work that can satisy ',work,'
                        is ',work_list[i])
                        print('The pressure that corresponds to
                        that work is ',corresponding_pressure)

                        self.p_out = corresponding_pressure
                        self.work = 0
```

```
                        self.stream = resetting_stream
                        self.expansion_by_steps(100)

    def expansion_by_steps(self, steps):
        starting_p = self.P3
        p_end = self.p_out
        total_p = starting_p - p_end
        iteration = 0
        number_of_steps = steps
        pressure_of_step = total_p / number_of_steps
        self.p_out = starting_p
        self.work = 0
        temperature_step_before = self.TIT

        while iteration < number_of_steps:
            if self.p_out >= p_end:
                self.p_out = self.p_out - pressure_of_step
                self.calc()
                new_iteration_stream = self.get_outlet_stream()
                self.stream = new_iteration_stream
                delta_T = temperature_step_before - self.get_outlet_temperature('K')
                temperature_step_before = self.t_out
                flow_rate = self.stream.get_flow_rate('kg/sec')
                cp = self.stream.fluid.getPhase(0).getCp('kJ/kgK')
                work_iteration = (flow_rate * delta_T) * cp
                self.work = self.work + work_iteration
            iteration = iteration + 1

    def visualize_work(self,units):
        if units == 'kW':
            return(self.work)

    def solve_polytropic_efficiency(self):
        k = self.k_3
        P_out = self.p_out
        T_in = self.TIT
        P_in = self.P3
        T_out = self.t_out

        a = math.log((T_out/T_in),(P_out/P_in))
        x = a*(k/(k-1))
        return(x)

    def set_t_out(self, t_out, units):
        if units == 'C':
```

```
            self.t_out = t_out
        elif units == 'K':
            self.t_out = t_out + 273.15
```

.2 Case study

```
import numpy as np
from matplotlib import pyplot as plt

from Classes_overleaf import Stream, Compressor, Combustor,
Expander
from neqsim.thermo.thermoTools import fluid, TPflash, PHflash
from neqsim.standards import ISO6976

# Design Model:

# List of components:
component_names = ["oxygen", "nitrogen", "methane", "ethane",
"propane",
                   "i-butane", "n-butane", "i-pentane",
                   "n-pentane", "n-hexane", "H2O", "CO2"]
air_composition = [0.2, 0.8, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0]
methane_composition = [0.0, 0.0, 1, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0]

# Creating the fluid air:
air = fluid("srk")
for component in component_names:
    air.addComponent(component,
    air_composition[component_names.index(component)])

# Creating the fluid fuel (methane):
methane = air.clone()
methane.setMolarComposition(methane_composition)

# Creating the stream of air:
air_stream = Stream()
air_stream.set_fluid(air)
air_stream.set_temperature(20, 'C')
air_stream.set_pressure(1)
air_stream.set_flow_rate(50, 'kg/sec')
air_stream.calculate()

# Setting the compressor using pol_efficiency
compressor = Compressor()
```

```
compressor.set_losses(0)
compressor.set_stream(air_stream)
compressor.set_p_out(10)
compressor.set_pol_efficiency(0.87)
compressor.calc_isentropic_efficiency()
compressor.compression_by_steps(100)

# Re-setting the stream after the compressor:
compressor.get_outlet_stream()
enthalpy_air = compressor.outlet_stream.fluid.getEnthalpy()

# Creating the fuel methane:
methane_stream = Stream()
methane_stream.set_fluid(methane)
methane_stream.set_temperature(20, 'C')
methane_stream.set_pressure(1)
methane_stream.set_flow_rate(1, 'kg/sec')
methane_stream.calculate()

# Defining the combustor:
combustor1 = Combustor()
combustor1.set_stream_air(compressor.outlet_stream)
combustor1.set_stream_fuel(methane_stream)
combustor1.calc_enthalpy()
combustor1.calc_TIT_reaction()
combustor1.calc_enthalpy()
combustor1.calc_TIT()

# Re-setting the stream after the combustor
combustor1.get_outlet_stream()

# TO TURBINE FLUID:
number_of_components = combustor1.get_outlet_stream().fluid.getNumberOfComponents()
names = [combustor1.get_outlet_stream().fluid.getComponent(i).getName()
         for i in range(number_of_components)]
molar_fractions = [combustor1.get_outlet_stream().fluid.getComponent(i).getx()
                   for i in range(number_of_components)]
print(names)
print(molar_fractions)

# Turbine:
turbine = Expander()
turbine.set_losses(0)
turbine.set_stream(combustor1.outlet_stream)
```

```python
turbine.set_p_out(1)
turbine.set_pol_efficiency(0.9)
turbine.calc_isentropic_efficiency()
turbine.expansion_by_steps(100)
turbine.get_outlet_stream()

# Gas Turbine
GT_work = turbine.get_work('kW') - compressor.get_work('kW')
GT_eff = GT_work / combustor1.get_Q('kW')

# Off-Design model:
print('OFF DESIGN WITH WHILE CICLE:')
component_names = ["oxygen", "nitrogen", "methane", "ethane", "propane",
                   "i-butane", "n-butane", "i-pentane", "n-pentane", "n-hexane", "H2O", "CO2"]
air_composition = [0.2, 0.8, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
methane_composition = [0.0, 0.0, 1, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]

# Creating the fluid air:
air_off_design = fluid("srk")
for component in component_names:
    air_off_design.addComponent(component, air_composition[component_names.index(component)])

# Creating the fluid fuel (methane):
methane_off_design = air_off_design.clone()
methane_off_design.setMolarComposition(methane_composition)

# CALCULATION OF REFERENCES:
ambient_temperature = []
compressed_temperature = []
exhaust_temperature = []
TIT_temperature = []
TIT_no_reaction = []
net_power = []
compressor_duty = []
turbine_duty = []
compressor_ise_efficiency = []
P2_list = []
P3_list = []

temperature_off_design_K = 298.15
i = 0
iteration = 7
```

```
while i < iteration:
    temperature_off_design_K = temperature_off_design_K - 5
    temperature_off_design_C = temperature_off_design_K - 273.15
    # Collecting ambient temperatures:
    ambient_temperature.append(temperature_off_design_C)
    ref_mass_flow_kg_s = air_stream.get_flow_rate("kg/sec")
    ref_temperature_K = air_stream.get_temperature("K")
    mass_flow_rate_off_design_kg_s = ref_mass_flow_kg_s * \
                                     ((ref_temperature_K / temperature_off_design_K)**(1/2))
    print("Mass flow of air bacome [kg/sec] :", mass_flow_rate_off_design_kg_s)
    T3_ref = combustor1.get_TIT_reaction('K')
    P3_ref = compressor.get_p_out('bara') - compressor.get_p_out('bara') * 0.015
    MW3_ref = combustor1.outlet_stream.fluid.getMolarMass()*1000
    k = air_stream.fluid.getGamma2()

    # Creating the stream air:
    air_stream_off_design = Stream()
    air_stream_off_design.set_fluid(air_off_design)
    air_stream_off_design.set_temperature(temperature_off_design_K, 'K')
    air_stream_off_design.set_pressure(1)
    air_stream_off_design.set_flow_rate(mass_flow_rate_off_design_kg_s, 'kg/sec')
    air_stream_off_design.calculate()

    compressor_off_design = Compressor()
    compressor_off_design.set_stream(air_stream_off_design)
    compressor_off_design.set_p_out(10)
    compressor_off_design.set_pol_efficiency(0.87)
    compressor_off_design.calc_isentropic_efficiency()
    compressor_off_design.compression_by_steps(100)

    # Creating the stream fuel(methane):
    methane_stream_off_design = Stream()
    methane_stream_off_design.set_fluid(methane_off_design)
    methane_stream_off_design.set_temperature(20, 'C')
    methane_stream_off_design.set_pressure(1)
    methane_stream_off_design.set_flow_rate(1, 'kg/sec')
    methane_stream_off_design.calculate()

    combustor1_off_design = Combustor()
    combustor1_off_design\
```

```
        .set_stream_air(compressor_off_design.get_outlet_stream())
    combustor1_off_design.set_stream_fuel(methane_stream_off_design)
    combustor1_off_design.calc_enthalpy()
    combustor1_off_design.calc_TIT_reaction()
    combustor1_off_design.get_outlet_stream()

    # Turbine:
    turbine_off_design = Expander()
    turbine_off_design\
        .set_stream(combustor1_off_design.get_outlet_stream())
    turbine_off_design.set_p_out(1)
    turbine_off_design.set_pol_efficiency(0.9)
    turbine.expansion_by_steps(100)

    P2 = compressor.get_p_out('bara') # initial guess for P2

    n = 0
    while True:
        n = n + 1
        P2_n = P2  # initial guess for P2
        P1 = air_stream.get_pressure('bara')
        i_p = compressor_off_design.get_isentropic_efficiency()
        p_c = ((k - 1) * np.log(P2_n / P1)) / (k * np.log((1 / i_p)
                                    * ((P2_n / P1) ** ((k - 1) / k) - 1) + 1))
        compressor_off_design.set_stream(air_stream_off_design)
        compressor_off_design.set_p_out(P2_n)
        compressor_off_design.set_pol_efficiency(p_c)
        compressor_off_design.compression_by_steps(100)

        compressor_off_design.get_outlet_stream()

        # RE-CALCULATION FOR COMBUSTOR 1:
        combustor1_off_design\
            .set_stream_air(compressor_off_design.get_outlet_stream())
        combustor1_off_design.set_stream_fuel(methane_stream_off_design)
        combustor1_off_design.calc_enthalpy()
        combustor1_off_design.calc_TIT_reaction()
        MW3_new = combustor1_off_design.outlet_stream.fluid.getMolarMass() * 1000

```

```
        m3 = mass_flow_rate_off_design_kg_s \
             + methane_stream_off_design.get_flow_rate('kg/sec')
        m3_ref = ref_mass_flow_kg_s \
                 +
                 methane_stream_off_design.get_flow_rate('kg/sec')
        T3 = combustor1_off_design.get_TIT_reaction('K')
        P3 = P3_ref * (m3 / m3_ref) * ((T3 /
        T3_ref)*(MW3_ref/MW3_new))** 0.5
        pressure_drop = (0.15 *
                         (m3 / m3_ref)**(1.8))* ((T3 / T3_ref) *
                         (P3_ref / P3)) ** 0.8

        P2 = P3 + pressure_drop  # new p2

        if (P2_n == P2):
            compressor_off_design.set_p_out(P2)

            compressor_off_design.set_stream(air_stream_off_design)
            compressor_off_design.set_pol_efficiency(p_c)
            compressor_off_design.compression_by_steps(100)
            to_combustor =
            compressor_off_design.get_outlet_stream()

            # RE-CALCULATION FOR COMBUSTOR 1:
            combustor1_off_design.set_stream_air(to_combustor)

            combustor1_off_design.set_stream_fuel(methane_stream_off_design)
            combustor1_off_design.calc_enthalpy()
            combustor1_off_design.calc_TIT_reaction()
            combustor1_off_design.calc_enthalpy()
            combustor1_off_design.calc_TIT()
            combustor1_off_design.get_outlet_stream()
            to_turbine =
            combustor1_off_design.get_outlet_stream()

            # CALCULATION FOR THE TURBINE:
            turbine_off_design.set_stream(to_turbine)
            turbine_off_design.set_p_out(1)
            turbine_off_design.set_pol_efficiency(0.9)
            turbine_off_design.expansion_by_steps(100)
            break

    compressed_temperature\

        .append(compressor_off_design.get_outlet_temperature('C'))
    TIT_temperature\
        .append(combustor1_off_design.get_TIT_reaction('C'))
```

```
    TIT_no_reaction.append(combustor1_off_design.get_TIT('C'))
    P2_list.append(compressor_off_design.get_p_out('bara'))
    P3_list.append(combustor1_off_design.outlet_stream.get_pressure('bara'))
    exhaust_temperature.append(turbine_off_design.get_outlet_temperature('C'))
    compressor_ise_efficiency\
        .append(compressor_off_design.get_isentropic_efficiency())
    compressor_duty.append(compressor_off_design.visualize_work('kW'))
    turbine_duty.append(turbine_off_design.visualize_work('kW'))
    i = i + 1

# Exporting the temperatures from Hysys
t4_hysys = []
t2_hysys = []
TIT_hysys = []
with open('Tambient_T4.csv') as f:
    next(f)
    for line in f:
        parts = line.split(';')
        parts1 = parts[1].split('"')
        t2_hysys.append(float(parts1[1]))
        parts2 = parts[2].split('"')
        t4_hysys.append(float(parts2[1]))
        parts3 = parts[3].split('"')
        TIT_hysys.append(float(parts3[1]))

# Data for T2 in Hysys
x1 = t2_hysys
y1 = ambient_temperature

# Data for T2 in Python
x2 = compressed_temperature
y2 = ambient_temperature

# Plot the Temperatures
plt.plot(x1, y1, label='T2 Hysys')
plt.plot(x2, y2, label='T2 Python')

# Add legend and axis labels
plt.legend()
plt.xlabel('Compressed Air Temperature')
plt.ylabel('Ambient Temperature')
plt.title('Comparison of T2 between Hysys and Python')
```

```python
# Display the plot
plt.show()

# Data for T4 in Hysys
x1 = t4_hysys
y1 = ambient_temperature

# Data for T4 in Python
x2 = exhaust_temperature
y2 = ambient_temperature

# Plot the Temperatures
plt.plot(x1, y1, label='T4_Hysys')
plt.plot(x2, y2, label='T4_Python')

# Add legend and axis labels
plt.legend()
plt.xlabel('Exhaust Gas Temperature')
plt.ylabel('Ambient Temperature')
plt.title('Comparison of T4 between Hysys and Python')

# Display the plot
plt.show()

# Data for TIT in Hysys
x1 = TIT_hysys
y1 = ambient_temperature

# Data for T2 in Python
x2 = TIT_temperature
y2 = ambient_temperature

# Plot the Temperatures
plt.plot(x1, y1, label='TIT_Hysys')
plt.plot(x2, y2, label='TIT_Python')

# Add legend and axis labels
plt.legend()
plt.xlabel('TIT')
plt.ylabel('Ambient Temperature')
plt.title('Comparison of TIT between Hysys and Python')

# Display the plot
plt.show()
```

.3 GE LM600: design

```python
from Classes_overleaf import Stream, Compressor, Combustor, Expander
from neqsim.thermo.thermoTools import fluid, TPflash, PHflash
from neqsim.standards import ISO6976

def calculation_pressure_ratio(stages, pressure_input, p_ratio_stage):
    n = 0
    p1 = pressure_input
    list = []
    while n < stages:
        p2 = p_ratio_stage * p1
        p_ratio = p2/p1
        list.append(p_ratio)
        p1 = p2
        n = n + 1

    pressure_ratio = 1
    for i in list:
        pressure_ratio *= i

    pressure_ratio = pressure_ratio * pressure_input

    return(pressure_ratio)

flow_rate_air = 128.9
flow_rate_fuel = 2.228
pressure_LP_compressor = 2.43581
pressure_HP_compressor = 29.19484
pressure_LP_turbine = 6.52188

polytropic_efficiency_gasturb = [0.88, 0.925, 0.8568, 0.8568]

iso_pressure_ratio = 29.1
stages_LP = 5
stages_HP = 14
total_stages = stages_LP + stages_HP
pressure_ratio_stage = (iso_pressure_ratio ** (1/total_stages))

pressure_LP = calculation_pressure_ratio(5,1,pressure_ratio_stage)
pressure_HP = calculation_pressure_ratio(14,pressure_LP,pressure_ratio_stage)

# List of components:
component_names = ["oxygen", "nitrogen", "argon","methane", "ethane", "propane",
```

```
                         "i-butane", "n-butane", "i-pentane",
                         "n-pentane", "n-hexane", "H2O", "CO2"]
air_composition = [0.20660, 0.77, 0.00927, 0.0, 0.0, 0.0,
                         0.0, 0.0, 0.0, 0.0, 0.0, 0.01384, 0.00030]
fuel_case22_composition = [0.0, 0.013, 0.0, 0.9516, 0.024,
                                   0.0036, 0.0017, 0.0009, 0.0011,
                                   0.000, 0.0,
                                   0.0000, 0.0042]

# Creating the fluid air:
air = fluid("srk")
for component in component_names:
     air.addComponent(component,
     air_composition[component_names.index(component)])

# Creating the fluid fuel case 22 for design:
fuel_22 = air.clone()
fuel_22.setMolarComposition(fuel_case22_composition)

# Creating the stream of air:
air_stream = Stream()
air_stream.set_fluid(air)
air_stream.set_temperature(10, 'C')
air_stream.set_pressure(1.01325 - (10/1000))
air_stream.set_flow_rate(flow_rate_air, 'kg/sec')
air_stream.calculate()

# Setting the compressor using pol_efficiency
compressorLP = Compressor()
compressorLP.set_losses(1.528)
compressorLP.set_stream(air_stream)
compressorLP.set_p_out(pressure_LP)
compressorLP.set_pol_efficiency(polytropic_efficiency_gasturb[0])
compressorLP.compression_by_steps(100)

# Re-setting the stream after the compressor:
compressorLP.get_outlet_stream()

# Creating the HP compressor:
compressorHP = Compressor()
compressorHP.set_losses(0)
compressorHP.set_stream(compressorLP.outlet_stream)
compressorHP.set_p_out(pressure_HP)
compressorHP.set_pol_efficiency(polytropic_efficiency_gasturb[1])
compressorHP.compression_by_steps(100)
compressorHP.calc_isentropic_efficiency()
ise_compressor = compressorHP.get_isentropic_efficiency()
```

```python

# Re-setting the stream after the compressor:
compressorHP.get_outlet_stream()

# Creating the stream fuel_22:
fuel22_stream = Stream()
fuel22_stream.set_fluid(fuel_22)
fuel22_stream.set_temperature(10, 'C')
fuel22_stream.set_pressure(40)
fuel22_stream.set_flow_rate(flow_rate_fuel, 'kg/sec')
fuel22_stream.calculate()
LHV = fuel22_stream.get_LCV()

# Defining the combustor:
combustor1 = Combustor()
combustor1.set_stream_air(compressorHP.outlet_stream)
combustor1.set_stream_fuel(fuel22_stream)
combustor1.calc_enthalpy()
combustor1.calc_TIT_reaction()

# Re-setting the stream after the combustor
combustor1.get_outlet_stream()

# Turbine HP: the one that drives the compressors
turbineHP = Expander()
turbineHP.set_losses(0)
turbineHP.set_stream(combustor1.outlet_stream)
turbineHP.set_pol_efficiency(polytropic_efficiency_gasturb[2])
turbineHP.calc_p_out_iterations(compressorHP.visualize_work('kW'),'kW')
turbineHP.get_outlet_stream()

# Turbine LP:
turbineLP = Expander()
turbineLP.set_losses(0)
turbineLP.set_stream(turbineHP.outlet_stream)
turbineLP.set_pol_efficiency(polytropic_efficiency_gasturb[3])
turbineLP.set_p_out(1.01325 + (12.45/1000))
turbineLP.expansion_by_steps(100)
turbineLP.get_outlet_stream()

# Gas Turbine
GT_work = (turbineLP.visualize_work('kW')
           -
           compressorLP.visualize_work('kW'))*((0.9821)*(0.9922))
GT_eff_gasturb = GT_work / (flow_rate_fuel*((LHV)/1000))
heat_rate_calculated = (flow_rate_fuel*(LHV*3600)/GT_work)/1000

```

```
abs_error_power = abs(GT_work - 45199)
abs_error_temperature =
abs(turbineLP.get_outlet_temperature('K') - 726.15)
abs_error_hr = abs(heat_rate_calculated - 8610)
abs_error_efficiency = abs(GT_eff_gasturb - 0.418)

print((abs_error_power/45199)*100)
print(abs_error_temperature)
print((abs_error_hr/8610)*100)
print(abs_error_efficiency*100)
```

.4 GE LM2500: design and off-design

```
import math

from Classes_overleaf import Stream, Compressor, Combustor,
Expander
from neqsim.thermo.thermoTools import fluid, TPflash, PHflash
from neqsim.standards import ISO6976
import matplotlib.pyplot as plt

iso_pressure_ratio = 18
stages_LP = 6
stages_HP = 10
total_stages = stages_LP + stages_HP
pressure_ratio_stage = (iso_pressure_ratio ** (1/total_stages))

def calculation_pressure_ratio(stages, pressure_input,
p_ratio_stage):
    n = 0
    p1 = pressure_input
    list = []
    while n < stages:
        p2 = p_ratio_stage * p1
        p_ratio = p2/p1
        list.append(p_ratio)
        p1 = p2
        n = n + 1

    pressure_ratio = 1
    for i in list:
        pressure_ratio *= i

    pressure_ratio = pressure_ratio * pressure_input

    return(pressure_ratio)
```

```python

# Values from GasTurb: #
pressure_LP = calculation_pressure_ratio(6,1,pressure_ratio_stage)
pressure_HP = calculation_pressure_ratio(10,pressure_LP,pressure_ratio_stage)
mass_flow_air = 66.08
mass_flow_fuel = 1.26132

pratio = pressure_HP/pressure_LP
pol1 = 0.93 - 0.0053*math.log(pressure_LP)
pol2 = 0.93 - 0.0053*math.log(pratio)
print('POLI 1',pol1)
print('POLI 2',pol2)

efficiency_gas_turb = [0.899, 0.8905, 0.85, 0.8486]
# List of components:
component_names = ["oxygen", "nitrogen", "argon","methane", "ethane", "propane",
                   "i-butane", "n-butane", "i-pentane", "n-pentane", "n-hexane", "H2O", "CO2"]
air_composition = [0.20660, 0.77, 0.00927, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.01384, 0.00030]
fuel_case22_composition = [0.0, 0.013, 0.0, 0.950290,
                           0.023989, 0.003547, 0.001715, 0.000878, 0.000744, 0.001093, 0.0,
                           0.000032, 0.004193]

# Creating the fluid air:
air = fluid("srk")
for component in component_names:
    air.addComponent(component, air_composition[component_names.index(component)])

# Creating the fluid fuel case 22 for design:
fuel_22 = air.clone()
fuel_22.setMolarComposition(fuel_case22_composition)

# Creating the stream of air:
air_stream = Stream()
air_stream.set_fluid(air)
air_stream.set_temperature(20, 'C')
air_stream.set_pressure(1.01325 - (10 / 1000))
air_stream.set_flow_rate(mass_flow_air, 'kg/sec')
air_stream.calculate()

# Setting the compressor using pol_efficiency
```

```
compressorLP = Compressor()
compressorLP.set_losses(0.58)
compressorLP.set_stream(air_stream)
compressorLP.set_p_out(pressure_LP)
compressorLP.set_pol_efficiency(efficiency_gas_turb[0])
compressorLP.calc_isentropic_efficiency()
compressorLP.compression_by_steps(100)

# Re-setting the stream after the compressor:
compressorLP.get_outlet_stream()

# Setting the compressor using pol_efficiency
compressorHP = Compressor()
compressorHP.set_losses(0)
compressorHP.set_stream(compressorLP.outlet_stream)
compressorHP.set_p_out(pressure_HP)
compressorHP.set_pol_efficiency(efficiency_gas_turb[1])
compressorHP.compression_by_steps(100)
compressorHP.calc_isentropic_efficiency()
ise_compressor = compressorHP.get_isentropic_efficiency()

# Creating the stream fuel_22:
fuel22_stream = Stream()
fuel22_stream.set_fluid(fuel_22)
fuel22_stream.set_temperature(20, 'C')
fuel22_stream.set_pressure(30)
fuel22_stream.set_flow_rate(mass_flow_fuel, 'kg/sec')
fuel22_stream.calculate()
LHV = fuel22_stream.get_LCV()

# Defining the combustor:
combustor1 = Combustor()
combustor1.set_stream_air(compressorHP.outlet_stream)
combustor1.set_stream_fuel(fuel22_stream)
combustor1.calc_enthalpy()
combustor1.calc_TIT_reaction()

# Re-setting the stream after the combustor
combustor1.get_outlet_stream()

# Turbine HP:
turbineHP = Expander()
turbineHP.set_losses(0)
turbineHP.set_stream(combustor1.outlet_stream)
turbineHP.set_pol_efficiency(efficiency_gas_turb[2])
turbineHP.calc_p_out_iterations(compressorLP.visualize_work('kW')
```

```
                                +compressorHP.visualize_work('kW'),'kW')
turbineHP.get_outlet_stream()

# Turbine LP:
turbineLP = Expander()
turbineLP.set_losses(0)
turbineLP.set_stream(turbineHP.outlet_stream)
turbineLP.set_pol_efficiency(efficiency_gas_turb[3])
turbineLP.set_p_out(1.01325 + (15 / 1000))
turbineLP.expansion_by_steps(100)
turbineLP.get_outlet_stream()
turbineLP.calc_isentropic_efficiency()

number_of_components =
turbineLP.get_outlet_stream().fluid.getNumberOfComponents()
names =
[turbineLP.get_outlet_stream().fluid.getComponent(i).getName()
for i in range(number_of_components)]
molar_fractions =
[turbineLP.get_outlet_stream().fluid.getComponent(i).getx() for
i in range(number_of_components)]

molar_thermoflow = [0.1352, 0.7454, 0.03337, 0.07705]

# Gas Turbine
GT_work = turbineLP.visualize_work('kW')*(0.9753*0.9895)

GT_eff_thermoflow = GT_work / (mass_flow_fuel *((LHV)/1000))
heat_rate_calculated = (mass_flow_fuel*(LHV*3600)/GT_work)/1000

abs_error_power = abs(GT_work - 21958)
abs_error_temperature =
abs(turbineLP.get_outlet_temperature('K') - 816.15)
abs_error_hr = abs(heat_rate_calculated - 10033)
abs_error_efficiency = abs(GT_eff_thermoflow - 0.3588)

print((abs_error_power/21958)*100)
print(abs_error_temperature)
print((abs_error_hr/10033)*100)
print(abs_error_efficiency*100)

# Off-Design:
air_thermo = []
fuel_thermo = []
intake = []
exhaust = []
```

```
power = []
t_exhaust = []
heat_rate = []
efficiency = []
losses = []
ambient_temperature = []

# Result:
power_python = []
exhaust_python = []
heat_rate_python = []
efficiency_python = []
dp_intake_python = []

def data_extraction(index, namedata):
    with open('off_design_LM2500.csv') as f:
        lines = f.readlines()
        line_index = [0,1,2]
        if index == 'air':
            line_index = 0
        elif index == 'fuel':
            line_index = 1
        elif index == 'intake_drop':
            line_index = 2
        elif index == 'exhaust_drop':
            line_index = 3
        elif index == 'power':
            line_index = 4
        elif index == 't_exhaust':
            line_index = 5
        elif index == 'heat_rate':
            line_index = 6
        elif index == 'efficiency':
            line_index = 7
        elif index == 'losses':
            line_index = 8
        headers = lines[line_index].strip().split('\t')
        i = 20
        index = 0
        while index <= i:
            if index == 0:
                headers0 = headers[0]
                headers1 = headers0.split()
                headers2 = headers1[1]
                namedata.append(float(headers2))

            elif index != 0:
```

```
                namedata.append(float(headers[index]))
            index = index + 1

# Usage
data_extraction('air',air_thermo)
data_extraction('fuel',fuel_thermo)
data_extraction('intake_drop',intake)
data_extraction('exhaust_drop',exhaust)
data_extraction('power',power)
data_extraction('t_exhaust',t_exhaust)
data_extraction('heat_rate',heat_rate)
data_extraction('efficiency',efficiency)
data_extraction('losses',losses)
LHV_plot = []
# Design Input #
efficiency_gas_turb = [0.899, 0.8905, 0.85, 0.8486]
starting_temperature = 273.15
temperature_design = 293.15
mass_flow_design = 66.08
P2_design = 18

iteration = 20
i = 0
while i <= iteration:
    print(i)

    # Off-design Input
    mass_flow_off_design = air_thermo[i]
    fuel_rate_off_design = fuel_thermo[i]
    temperature_off_design = i + 273.15
    ambient_temperature.append(temperature_off_design)
    mass_flow_no_IGV = mass_flow_design *
    (temperature_design/temperature_off_design)**(1/2)
    # Coefficients off-design #
    c = mass_flow_off_design/mass_flow_no_IGV # 1 + c*delta/100
    c_efficiency = c - 1

    # New Pressure Off Design #
    iso_pressure_ratio = P2_design * c
    stages_LP = 6
    stages_HP = 10
    total_stages = stages_LP + stages_HP
    pressure_ratio_stage = (iso_pressure_ratio **
    (1/total_stages))
    pressure_LP =
    calculation_pressure_ratio(6,1,pressure_ratio_stage)
```

```
    pressure_HP =
    calculation_pressure_ratio(10,pressure_LP,pressure_ratio_stage)

    # New efficiency Off Design #
    #efficiency_off_design = efficiency_gas_turb[1]*(1 -
    c_efficiency)
    efficiency_off_design = ise_compressor*(1 - c_efficiency)
    efficiency_turbine_off_design = efficiency_gas_turb[3]*(1 -
    c_efficiency)

    # Setting the GT in Off-Desing conditions #
    air_stream_off_design = Stream()
    air_stream_off_design.set_fluid(air)
    air_stream_off_design.set_temperature(i, 'C')
    air_stream_off_design.set_pressure(1.01325 - (intake[i]/
    1000))
    air_stream_off_design.set_flow_rate(mass_flow_off_design,
    'kg/sec')
    air_stream_off_design.calculate()

    air_dp = Stream()
    air_dp.set_fluid(air)
    air_dp.set_temperature(i, 'C')
    air_dp.set_pressure(1.01325)
    air_dp.set_flow_rate(mass_flow_off_design, 'kg/sec')
    air_dp.calculate()

    compressorLP_off_design = Compressor()
    compressorLP_off_design.set_losses(losses[i])
    compressorLP_off_design.set_stream(air_stream_off_design)
    compressorLP_off_design.set_p_out(pressure_LP)

    compressorLP_off_design.set_pol_efficiency(efficiency_gas_turb[0])
    compressorLP_off_design.compression_by_steps(100)
    compressorLP_off_design.get_outlet_stream()

    compressorHP_off_design = Compressor()
    compressorHP_off_design.set_losses(0)

    compressorHP_off_design.set_stream(compressorLP_off_design.get_outlet_stream())
    compressorHP_off_design.set_p_out(pressure_HP)

    #compressorHP_off_design.set_pol_efficiency(efficiency_off_design)

    compressorHP_off_design.set_isentropic_efficiency(efficiency_off_design)
    compressorHP_off_design.calc_polytropyc_efficiency()
    poly = compressorHP_off_design.get_polytripic_efficiency()
```

```
    compressorHP_off_design.set_pol_efficiency(poly)
    compressorHP_off_design.compression_by_steps(100)
    compressorHP_off_design.get_outlet_stream()

    # Creating the stream fuel(methane):
    fuel22_stream_off_design = Stream()
    fuel22_stream_off_design.set_fluid(fuel_22)
    fuel22_stream_off_design.set_temperature(293.15, 'K')
    fuel22_stream_off_design.set_pressure(30)
    fuel22_stream_off_design.set_flow_rate(fuel_rate_off_design,
    'kg/sec')
    fuel22_stream_off_design.calculate()
    LHV = fuel22_stream_off_design.get_LCV()
    LHV_plot.append(LHV)

    combustor1_off_design = Combustor()

    combustor1_off_design.set_stream_air(compressorHP_off_design.get_outlet_st

    combustor1_off_design.set_stream_fuel(fuel22_stream_off_design)
    combustor1_off_design.calc_enthalpy()
    combustor1_off_design.calc_TIT_reaction()
    combustor1_off_design.get_outlet_stream()

    # Re-setting the stream after the combustor
    combustor1_off_design.get_outlet_stream()

    # Turbine HP:
    turbineHP_off_design = Expander()
    turbineHP_off_design.set_losses(0)

    turbineHP_off_design.set_stream(combustor1_off_design.outlet_stream)

    turbineHP_off_design.set_pol_efficiency(efficiency_gas_turb[2])

    turbineHP_off_design.calc_p_out_iterations(compressorLP_off_design.visuali

                                           +compressorHP_off_design.visual
    turbineHP_off_design.get_outlet_stream()

    # Turbine LP:
    turbineLP_off_design = Expander()
    turbineLP_off_design.set_losses(0)

    turbineLP_off_design.set_stream(turbineHP_off_design.outlet_stream)

    turbineLP_off_design.set_pol_efficiency(efficiency_turbine_off_design)
```

```
    turbineLP_off_design.set_p_out(1.01325 + (exhaust[i]/ 1000))
    turbineLP_off_design.expansion_by_steps(100)
    turbineLP_off_design.get_outlet_stream()

    # Gas Turbine
    GT_work = turbineLP_off_design.visualize_work('kW')*(0.9753*0.9895)
    power_python.append(GT_work)

    GT_eff_thermoflow = GT_work / (fuel_rate_off_design *((LHV)/1000))
    efficiency_python.append(GT_eff_thermoflow)
    heat_rate_calculated = (fuel_rate_off_design*(LHV*3600)/GT_work)/1000
    heat_rate_python.append(heat_rate_calculated)

    exhaust_python.append(turbineLP_off_design.get_outlet_temperature('C'))

    abs_error_power = abs(GT_work - power[i])
    abs_error_temperature = abs(turbineLP_off_design.get_outlet_temperature('K') -(t_exhaust[i]+273.15))
    abs_error_hr = abs(heat_rate_calculated - heat_rate[i])
    abs_error_efficiency = abs(GT_eff_thermoflow - (efficiency[i]/100))

    print((abs_error_power/power[i])*100)
    print(abs_error_temperature)
    print((abs_error_hr/heat_rate[i])*100)
    print(abs_error_efficiency*100)

    i = i + 1

print(power_python)
print(exhaust_python)
print(efficiency_python)
print(heat_rate_python)
```

.5 Performance Indicator analysis

```
import csv
from ClassesFORCASESTUDY import Stream, Compressor, Combustor, Expander
from neqsim.thermo.thermoTools import fluid, TPflash, PHflash
from neqsim.standards import ISO6976
import matplotlib.pyplot as plt
```

```

# Plot function, 2 variables: #
def plot(x, ya, yb, label, y, v1, v2, x_tick_positions,
x_tick_labels):
    # Real data:
    x1 = x # date range
    y1 = ya

    # Calculated
    x2 = x
    y2 = yb

    # Plot the Temperatures
    plt.plot(x1, y1, label=v1)
    plt.plot(x2, y2, label=v2)

    plt.xticks(x_tick_positions, x_tick_labels)

    # Add legend and axis labels
    plt.legend()
    plt.xlabel('Date [days]')
    plt.ylabel(y)
    plt.title(f"{label}")

    plt.show()
# Plot function: 1 variable: #
def plot_1(x, y, label, v1, x_tick_positions, x_tick_labels):

    x1 = x
    y1 = y

    plt.plot(x1, y1, label=v1)

    plt.xticks(x_tick_positions, x_tick_labels)

    # Add legend and axis labels
    plt.legend()
    plt.xlabel('Date [days]')
    plt.ylabel(v1)
    plt.title(f"{label}")

    plt.show()

date_list = []
ambient_temperature = []
ambient_pressure = []
pressure_LP_6000 = []
```

```python
temperature_LP_6000 = []
pressure_HP_6000 = []
temperature_HP_6000 = []

def data_extraction(index, namedata):
    with open('data_not_modified.csv') as f:
        reader = csv.reader(f)

        # Read the first line,
        first_line = next(reader)

        column_index = first_line.index(index)

        for row in reader:
            if index == 'time':
                namedata.append(row[column_index])
            else:
                namedata.append(float((row[column_index])))

data_extraction('time', date_list)
data_extraction('1219-27PT1053A', pressure_LP_6000)
data_extraction('1219-27TE1054A', temperature_LP_6000)
data_extraction('1219-27PT1062A', pressure_HP_6000)
data_extraction('1219-27TE1061A', temperature_HP_6000)
data_extraction('1219-27TT1193', ambient_temperature)
data_extraction('1219-27PT1051', ambient_pressure)

y = 3000
x_tick_labels = []
for i in range(len(date_list)):
    if i == 0 or i == 500 or i == 1000 or i == 1500 or i == 2000
    or i == 2500 or i == 2999:
        x_tick_labels.append(date_list[i].split()[0][5:10])
x_tick_positions = [0, 500, 1000, 1500, 2000, 2500, 2999]  #
Tick positions

plot_1(date_list[:y], temperature_LP_6000[:y],
       'LP outlet temperature, raw data','LP outlet temperature
       [°C]',
       x_tick_positions,x_tick_labels)
plot_1(date_list[:y], pressure_LP_6000[:y],
       'LP outlet pressure, raw data','LP outlet pressure
       [bar]',
       x_tick_positions,x_tick_labels)

# Cleaning the data from bad numbers #
def remove_distant_values(data_list, threshold):
```

```python
    cleaned_list = [data_list[0]]  # Start with the first value
    indices_to_remove = []  # Store indices of values to be removed

    for i in range(1, len(data_list)):
        if abs(data_list[i] - cleaned_list[-1]) <= threshold:
            cleaned_list.append(data_list[i])
        else:
            indices_to_remove.append(i)

    return cleaned_list, indices_to_remove

def list_days(data_list, threshold, date):
    cleaned_list = [data_list[0]]  # Start with the first value
    indices_to_remove = []  # Store indices of values to be removed
    x_tick_labesl_total = []
    for i in range(1, len(data_list)):
        if abs(data_list[i] - cleaned_list[-1]) <= threshold:
            cleaned_list.append(data_list[i])
            x_tick_labesl_total.append(date[i])
        else:
            indices_to_remove.append(i)

    return x_tick_labesl_total

threshold = 5
temperature_LP_cleaned, indices_to_remove = remove_distant_values(temperature_LP_6000[:y], threshold)
new_days = list_days(temperature_LP_6000[:y],threshold, date_list[:y])

number_cleaned = len(temperature_LP_cleaned)

# Remove corresponding values from another list
pressure_LP_cleaned = [pressure_LP_6000[i] for i in range(len(pressure_LP_6000[:y]))
                       if i not in indices_to_remove]
date_cleaned = [date_list[i] for i in range(len(date_list[:y]))
                if i not in indices_to_remove]
ambient_t_cleaned = [ambient_temperature[i] for i in range(len(ambient_temperature[:y]))
                     if i not in indices_to_remove]
ambient_p_cleaned = [ambient_pressure[i] for i in range(len(ambient_pressure[:y]))
                     if i not in indices_to_remove]
# Resetting x_tick_labels:
```

```
x_tick_labels = []

for i in range(len(date_cleaned)):
    if i == 0 or i == 500 or i == 1000 or i == 1500 or i ==
    2000:
        x_tick_labels.append(new_days[i].split()[0][5:10])

x_tick_positions = [0, 500, 1000, 1500, 2000]  # Tick positions
print(x_tick_labels)

plot_1(date_list[:number_cleaned], temperature_LP_cleaned, 'LP
outlet temperature, cleaned data',
       'LP outlet temperature [°C]',
       x_tick_positions, x_tick_labels)
plot_1(date_list[:number_cleaned], pressure_LP_cleaned, 'LM6000
LP pressure, cleaned data',
       'LP outlet pressure [bara]',
    x_tick_positions,x_tick_labels)

def moving_average(data, window_size):
    moving_averages = []

    for i in range(len(data) - window_size + 1):
        window = data[i:i + window_size]
        average = sum(window) / window_size
        moving_averages.append(average)

    return moving_averages
# Cleaning data: #
x = number_cleaned
window_size = 10
temperature_LP_cleaned =
moving_average(temperature_LP_cleaned[:x], window_size)
pressure_LP_cleaned = moving_average(pressure_LP_cleaned[:x],
window_size)
ambient_t_cleaned = moving_average(ambient_t_cleaned[:x],
window_size)
ambient_p_cleaned = moving_average(ambient_p_cleaned[:x],
window_size)

number = len(temperature_LP_cleaned)
print('Number with 3 = ',number)
plot_1(date_list[:number], temperature_LP_cleaned,
       'LP outlet temperature, after moving average','LP outlet
       temperature [°C]',
       x_tick_positions,x_tick_labels)
plot_1(date_list[:number], pressure_LP_cleaned,
```

```
        'LP outlet pressure, after moving average','LP outlet
        pressure [bar]',
        x_tick_positions,x_tick_labels)

def calculation_pressure_ratio(stages, pressure_input,
p_ratio_stage):
    n = 0
    p1 = pressure_input # inizializiation of the variable
    list = []
    # for the first compressor:
    while n < stages:
        p2 = p_ratio_stage * p1
        p_ratio = p2/p1
        list.append(p_ratio)
        p1 = p2
        n = n + 1

    pressure_ratio = 1
    for i in list:
        pressure_ratio *= i

    pressure_ratio = pressure_ratio * pressure_input

    return(pressure_ratio)

############################## THERMOFLOW
#################################
flow_rate_air = 128.9
flow_rate_fuel = 2.228

polytropic_efficiency_gasturb = [0.88, 0.925, 0.8568, 0.8568]

iso_pressure_ratio = 29.1
stages_LP = 5
stages_HP = 14
total_stages = stages_LP + stages_HP
pressure_ratio_stage = (iso_pressure_ratio ** (1/total_stages))

pressure_LP =
calculation_pressure_ratio(5,1,pressure_ratio_stage)
pressure_HP =
calculation_pressure_ratio(14,pressure_LP,pressure_ratio_stage)

# List of components:
component_names = ["oxygen", "nitrogen", "argon","methane",
"ethane", "propane",
```

```
                    "i-butane", "n-butane", "i-pentane",
                    "n-pentane", "n-hexane", "H2O", "CO2"]
air_composition = [0.20660, 0.77, 0.00927, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.01384, 0.00030]
fuel_case22_composition = [0.0, 0.013, 0.0, 0.9516, 0.024,
0.0036, 0.0017, 0.0009, 0.0011, 0.000, 0.0,
                           0.0000, 0.0042]

# Creating the fluid air:
air = fluid("srk")
for component in component_names:
    air.addComponent(component,
    air_composition[component_names.index(component)])
# Creating the fluid fuel case 22 for design:
fuel_22 = air.clone()
fuel_22.setMolarComposition(fuel_case22_composition)
# Iteration and plot #
date_range = date_list[:number]
t_calculated_LP = []
t_calculated_HP = []
index_LP_list = []
index_HP_list = []
baseline = []

x1 = date_list[:number]
y1 = ambient_t_cleaned[:number]

plt.plot(x1, y1, label='Ambient Temperature in [°C]')

# Add legend and axis labels
plt.legend()
plt.xlabel('Date [days]')
plt.ylabel('Ambient Temperature in [°C]')
plt.title('Ambient Temperature: plot')

plt.show()

for i in range(len(ambient_t_cleaned[:number])):
    # Creating the stream of air:
    air_stream = Stream()
    air_stream.set_fluid(air)
    air_stream.set_temperature(ambient_t_cleaned[i], 'C')
    air_stream.set_pressure(ambient_p_cleaned[i])
    air_stream.set_flow_rate(flow_rate_air, 'kg/sec')
    air_stream.calculate()

    # Setting the compressor using pol_efficiency
```

```
    compressorLP = Compressor()
    compressorLP.set_losses(1.528)
    compressorLP.set_stream(air_stream)
    compressorLP.set_p_out(pressure_LP_cleaned[i])
    compressorLP.set_pol_efficiency(polytropic_efficiency_gasturb[0])
    compressorLP.compression_by_steps(100)
    t1 = compressorLP.get_outlet_temperature('C')
    t_calculated_LP.append(t1)
    dt1 = temperature_LP_cleaned[i] - ambient_t_cleaned[i]
    dt2 = t1 - ambient_t_cleaned[i]
    index_LP = (dt1 - dt2)/dt1
    index_LP_list.append(index_LP)
    baseline.append(0)

plot(date_list[:number], index_LP_list[:number], baseline[:number],
     'LP compressor: performance indicator','Performance Indicator', 'Performance Indicator',
     'Baseline',x_tick_positions,x_tick_labels)
plot(date_list[:number], t_calculated_LP[:number],
     temperature_LP_cleaned[:number], 'LP compressor outlet temperature',
     'LP outlet temperature [°C]','LP calculated outlet temperature [°C]',
     'LP real outlet temperature [°C]',
     x_tick_positions,x_tick_labels)
moving_average_list = moving_average(index_LP_list[:number], 5)
number_x = len(moving_average_list)
plot_1(date_list[:number_x], moving_average_list[:number_x],
       'MOVING AVERAGE INDEX','MOVING AVERAGE LP',
       x_tick_positions,x_tick_labels)

# Export to CSV:

index = [index_LP_list[:number]]
ambient_t = [ambient_t_cleaned[:number]]
date = [date_list[:number]]

file_name = "index.csv"

# Scrivi i dati nel file CSV
with open(file_name, mode="w", newline="") as file:
    writer = csv.writer(file)
    writer.writerows(index)
    writer.writerows(ambient_t)
    writer.writerows(date)
```

```
# Index and ambient T
x = date_list[:number]
y1 = ambient_t_cleaned[:number]
y2 = index_LP_list[:number]

# First y axis:
fig, ax1 = plt.subplots()

color = 'tab:blue'
ax1.set_xlabel('Date [days]')
ax1.set_ylabel('Ambient Temperature [°C]', color=color)
ax1.plot(x, y1, color=color)
ax1.tick_params(axis='y', labelcolor=color)

# Second y axis:
ax2 = ax1.twinx()

color = 'tab:red'
ax2.set_ylabel('Performance Indicator', color=color)
ax2.plot(x, y2, color=color)
ax2.tick_params(axis='y', labelcolor=color)

plt.title('Performance Indicator and Ambient Temperature')
plt.show()
```