UNIVERSIDADE DE SÃO PAULO – USP

ESCOLA DE ENGENHARIA DE SÃO CARLOS – EESC

DEPARTAMENTO DE ENGENHARIA MECÂNICA – SEM

ÉCOLE CENTRALE DE NANTES

MATHEUS RAPOSO DE ALMEIDA

EVALUATION OF HEAT TRANSFER PHENOMENA IN A TURBO CHARGER USING ARTIFICIAL NEURAL NETWORKS

São Carlos

2018

MATHEUS RAPOSO DE ALMEIDA

EVALUATION OF HEAT TRANSFER PHENOMENA IN A TURBO CHARGER USING ARTIFICIAL NEURAL NETWORKS

Undergraduate thesis presented to the
course of Mechanical Engineering of
the University of São Paulo, as part of
the title of Mechanical Engineer

Advisor: Prof. Dr. Lúben Cabezas Gómez,
Prof. Dr. David Chalet,
Prof. Dr. Georges Salameh

São Carlos

2018

R219e Raposo de Almeida, Matheus

Evaluation of heat transfer phenomena in a turbo charger using artificial neural networks / Matheus Raposo de Almeida; advisor Lúben Cabezas Gómez. São Carlos, 2018.

Undergraduate thesis (Mechanical Engineering) – Escola de Engenharia de São Carlos da Universidade de São Paulo, 2018.

1. Artificial Neural Networks. 2. Turbo Chargers. 3. Heat Transfer. 4. Thermal modelling. I. Title.

FOLHA DE AVALIAÇÃO

Candidato: Matheus Raposo de Almeida

Título: Evaluation of heat transfer phenomena in a turbo charger using artificial neural networks.

Trabalho de Conclusão de Curso apresentado à
Escola de Engenharia de São Carlos da
Universidade de São Paulo
Curso de Engenharia ____________

BANCA EXAMINADORA

Professor Luben Cabezas Gómez
(Orientador)

Nota atribuída: 10 (Dez) [illegible]
(assinatura)

Professor Gherhardt Ribatski

Nota atribuída: 10 (DEZ) [illegible]
(assinatura)

Tiago Augusto Moreira

Nota atribuída: 10 (DEZ) [illegible]
(assinatura)

Média: 10 (Dez)

Resultado: Aprovado

Data: 07/12/2018.

Este trabalho tem condições de ser hospedado no Portal Digital da Biblioteca da EESC

[✓ SIM] NÃO ☐ Visto do orientador [illegible]

ACKNOWLEDGEMENTS

To the professors from the Ecole Centrale de Nantes, Dr. Georges Salameh and Prof. David Chalet, and my undergraduate thesis tutor, Prof. Dr. Lúben Cabezas Gómez, from the Thermal and Fluid Engineering Nucleus at the University of São Paulo, for their availability and their contributions to my reasoning all along this project.

To my colleague, Léo Le Roy from the Ecole Centrale de Nantes, for the long discussions and work on the physical model of this project.

To Prof. Dr. Roseli Aparecida Francelin Romero, responsible for the 'Neural Networks' graduate course at the Institute of Mathematical Sciences and Computing of the University of São Paulo.

To the constant support of friends and family, for the discussions and insights along all these years.

To the University of São Paulo, for not only providing precious teaching, but also for the opportunity to experience a Double Degree abroad. In that matter, a special thank you to the Ecole Centrale de Nantes, where the motivation for this project started.

ABSTRACT

RAPOSO DE ALMEIDA, M. (2018) **Evaluation of heat transfer phenomena in a turbo charger using artificial neural networks.** Undergraduate Thesis – Departamento de Engenharia Mecânica – Escola de Engenharia de São Carlos, Universidade de São Paulo.

Due to the current tendency to downsize motors, supercharging has become inevitable to maintain engine performance at high levels. The turbo charger not only offers more power to the engine, but it also responds to the environmental norms in perpetual evolution. An often-overlooked phenomenon taking place in a turbo charger are potential heat losses, which occur due to the high inlet turbine temperatures. These heat exchanges influence the turbo charger's efficiency considerably, and ultimately degrades the engine's performance. It is thus of innate interest of the manufacturer to evaluate and quantify these heat exchanges, in order to optimize turbo chargers. The challenges are numerous: complex geometry, difficulty evaluating convection coefficients and difficulty measuring parameters. To overcome these barriers, and the problem being of high complexity and with noisy and/or fuzzy experimental data, the application of neural networks is suggested. A Multi-Layer Perceptron is trained on a database and real turbo charger efficiencies are approximated with an easy-to-use computer application.

Keywords: *Artificial Neural Networks (ANNs); Turbo charger; Heat Transfer; Thermal modelling.*

TABLE OF FIGURES

LIST OF SYMBOLS

Symbol	Description	Unity
D	Diameter	m
L	Length	m
e	Width	m
P	Perimeter	m
A	Surface	m
h	Convective exchange coefficient	$W.m^{-2}.K^{-1}$
λ	Thermal conductivity	$W.m^{-1}.K^{-1}$
Nu	Nusselt number	Dimensionless
Gr	Grashof number	Dimensionless
Pr	Prandt number	Dimensionless
Ra	Raleigh number	Dimensionless
β	Gas dilatation coefficient	K^{-1}
ΔT	Difference between wall temperature and cabin	K
μ	Dynamic viscosity	Pa.s
η	Efficiency	Dimensionless
P	Power	W
T	Temperature	K
Q	Heat flow	W
Dm	Mass flow	$Kg.s^{-1}$
σ	Stefan-Boltzmann constant	$W.m^{-2}K^{-4}$
ε	Emissivity	Dimensionless
C1 up to C6	Temperature law coefficients	K
Alpha1 up to Alpha3	Geometrical coefficients – components' thermal properties	$1.m^{-2}$
Delta	Geometrical term + constant for the temperature evolution along the central body	K
Q_{T_PT}	Forced convection Turbine – Turbine Plateau	W
Q_{ext_T}	Natural convection + Radiation Turbine - Exterior	W
Q_{nc_T}	Natural convection Turbine - Exterior	W
Q_{RT}	Radiation Turbine - Exterior	W
$Q_{cond\_PT\_par\_diff}$	Conduction Turbine Plateau – Central body	W
$Q_{cond\_PT\_cote\_PT}$	Conduction Turbine Plateau - Central body	W
$Q_{cond\_PT\_cote\_cc}$	Conduction Turbine Plateau - Central body	W
Q_{ext_cc}	Natural convection Central body - Exterior	W
Q_H	Forced convection Central body - oil	W
Q_{C_PC}	Forced convection Compressor Plateau - Compressor	W
Q_{ext_PC}	Natural convection Compressor Plateau - Exterior	W
Q_{nc_C}	Natural convection Compressor - Exterior	W
Q_{RC}	Radiation Compressor - Exterior	W
$Q_{cond\_PC\_par\_diff}$	Conduction Central body – Compressor Plateau	W
$Q_{cond\_PC\_cote\_cc}$	Conduction Central body – Compressor Plateau	W

Symbol	Description	Unity
X_{pj}	Neuron j's input for the p pattern	Variable
η(eta)	Learning speed	Dimensionless
W_{ij}	Weight connecting current layer's neuron j and following layer's neuron i	Variable
v_{pj}	hidden layer's output at the neuron j for the pattern p.	Variable
t_{pj}	Desired output at neuron j for the pattern p	Variable
Ep	Mean Squared Error	Dimensionless
α (alpha)	Momentum term	Dimensionless
n	Number of patterns	Dimensionless
Tol1	Mean Squared Error tolerance	Dimensionless
Alph	Parameter correction rate	Dimensionless

Sub-Index	Description	Unity
1	Input	--/--
2	Output	--/--
is	Isentropic	--/--
C	Compressor	--/--
T	Turbine	--/--
cc	Central Body	--/--
PT	Turbine Plateau	--/--
PC	Compressor Plateau	--/--
e	Input	--/--
s	Output	--/--
int	Interior	--/--
h	Oil	--/--
Plto	Plateau (Compressor or Turbine)	--/--
RT	Radiation Turbine	--/--
RC	Radiation Compressor	--/--
nc	Natural Convection	--/--
f	Cast iron / Friction	--/--
a	Aluminum	--/--

TABLE OF CONTENTS

1. Introduction

Due to the current tendency to downsize motors, supercharging has become inevitable to maintain engine performance at high levels. The turbo charger not only offers more power to the engine, but it also responds to the environmental norms in perpetual evolution.

A turbo charger is a mechanical device used in internal combustion engines to tune up performance, by allowing the engine to intake compressed air, to inject more fuel and ultimately provide more torque to the axle.

The conception of a turbo charger requires a good apprehension of the physical phenomena which take place within it. The study of heat transfer is therefore important due to the presence of high-temperature exhaust gas entering the turbine and will lead to a better comprehension of the global functioning of this device.

To predict performance, turbo charger manufacturers offer the so-called "design charts", curves which form the efficiency field for the turbo charger under adiabatic conditions for several operating conditions. One such design chart is visible on Figure 1.

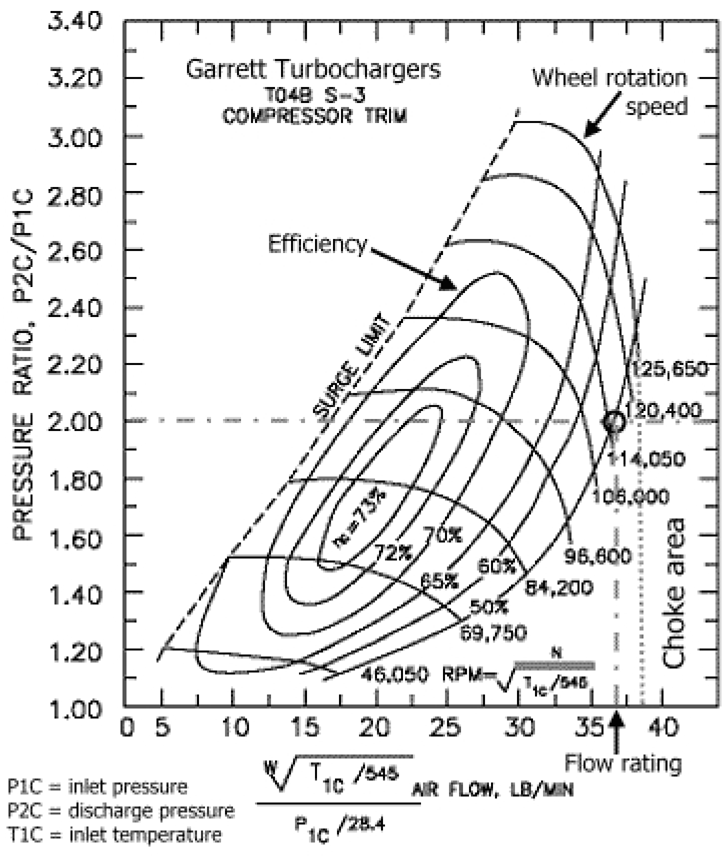


Figure 1: turbo charger design chart. Efficiency ellipses, wheel rotation speed parabolas, corrected air flow in the horizontal and compression ratio in the vertical. [26]

In fact, under the hypothesis of an adiabatic behavior, the absence of heat transfer results in the work provided to the compressor being equal to the work given to the fluid. Unfortunately, this hypothesis proves itself wrong under the usual operation conditions, as the exhaust gas entering the turbine volute easily reaches 600°C against the ambient-temperature gas flow which penetrates the compressor. Therefore, a logic formation of temperature gradients along the turbo charger is presumed (Figure 2).

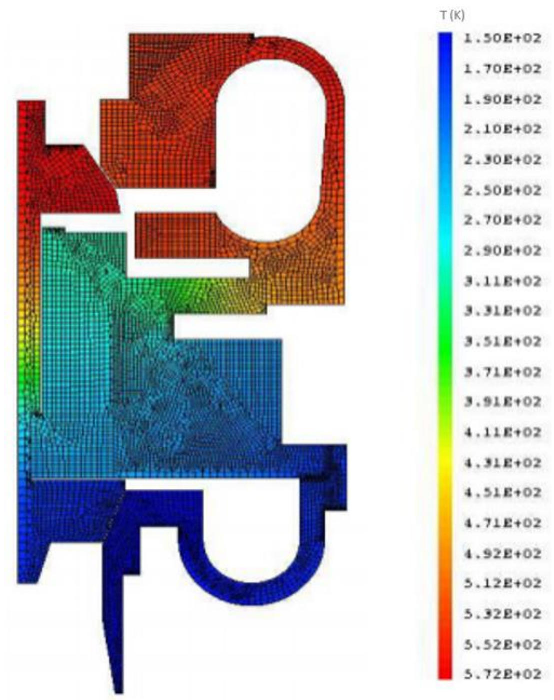


Figure 2: Visualization of the temperature distribution along a turbo charger using the finite element analysis. (M. Cormerais [2])

Due to the high complexity and non-linearity of the turbo charger heat exchange mechanism, the application of a machine learning algorithm is justifiable. In fact, a so-called “neural network”, if appropriately trained, may be able to predict the performance parameters for any turbo charger under any condition.

According to Simon S. Haykin’s book “Neural Networks: A Comprehensive Foundation” [15], an “ANN is a massively parallel-distributed processor, made up of interconnected simple processing units, which has a natural propensity to store experiential information and to make it available for use. It resembles the brain in two respects: i) knowledge is acquired by the network from its environment through a learning process. ii) interneuron strengths, known as synaptic weights, are used to store the knowledge.”

Artificial Neural Networks are inserted in the much wider category of Machine Learning, which dates to the rudimentary 200-year-old linear regression method. It only evolved into neural networks in 1958, with Frank Rosenblatt’s Perceptron algorithm. The psychologist developed a simplified algorithm to model the way brain neurons operate (Figure 3). Figure 4 contains a simplified model of an artificial neuron. The Perceptron, alongside the foundational work of the psychologist Donald Hebb, allowed machines to actually “learn”, taking inspiration from Hebb’s Rule:

"When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased." [D. Hebb, 1949] [22]

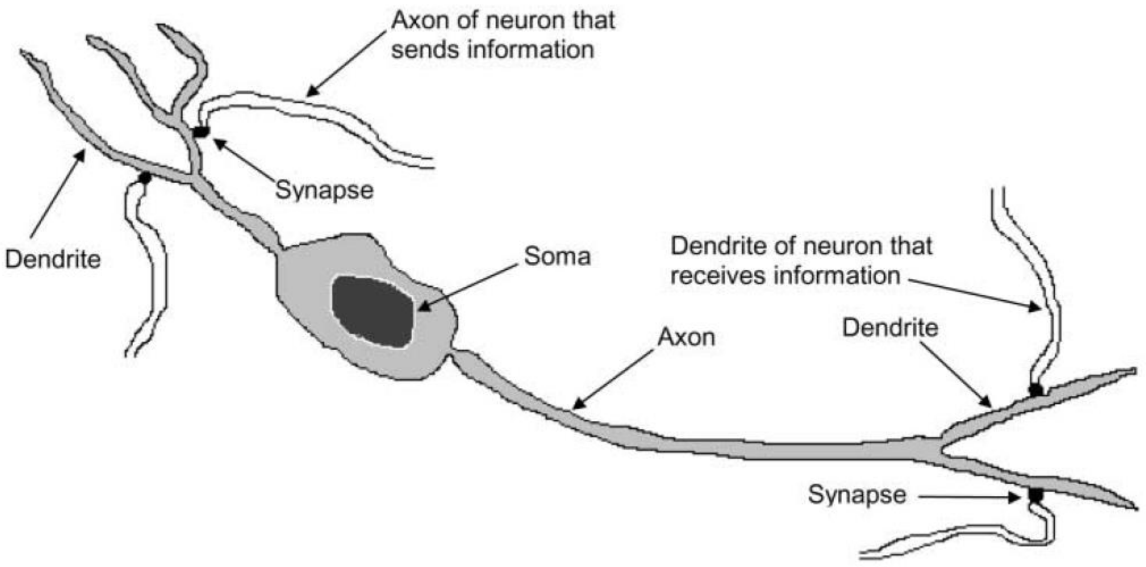

Axon of neuron that sends information
Synapse
Dendrite
Soma
Dendrite of neuron that receives information
Axon
Dendrite
Synapse

Figure 3: a simplified model of a biological neuron. [25]

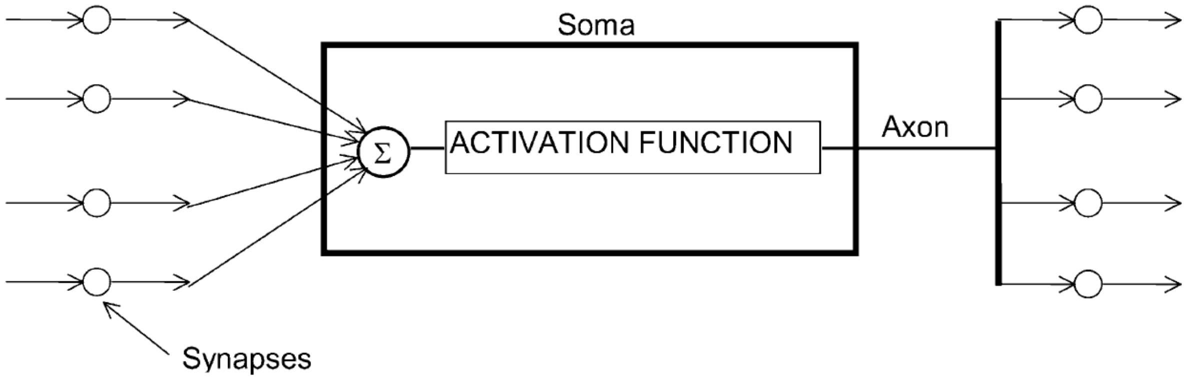


Figure 4: a simplified model of an artificial neuron. [25]

ANNs have since been extensively developed, and the now so-called deep learning methods occupy the most successful technology companies worldwide.

1.1 Objectives

This work's main objective is to develop a simple-to-use tool to predict turbo charger real operating efficiencies and heat losses. For this purpose, a theoretical model based on physical models will be developed to the extent of the model's efficiency and applicability. As for the domains for which the physical model is inconsistent, namely on the evaluation of heat convection coefficients, an artificial neural network will be trained to 'learn' to evaluate the desired outputs from experimental data.

The model is intended to be generalized, but the experimental data available comes from tests performed on a single turbo charger. The code's structure shall nevertheless allow for customization and application on various turbo charger geometries and experimental datasets.

1.2 Structure

The structure of the present work is divided as to present in the most linear way the physical modelling developed, in the following order:

- Chapter 2: basics of turbo chargers and artificial neural networks are provided.

- Chapter 3: bibliographical evidence of the state-of-the-art is presented, along with theoretical references, to guide the thought process and set the basis of knowledge for the model.

- Chapter 4: geometrical characteristics of the turbo charger used for experiments are given.

- Chapter 5: the physical model developed is detailed for each part of the turbo charger.

- Chapter 6: the choice of the heat exchange coefficients is explained.

- Chapter 7: efficiencies calculated are defined.

- Chapter 8: results are presented for the initial model and an optimization is suggested.

- Chapter 9: application of the artificial neural network. The results are presented and commented, comparing its performance and application to the original model and to other applications of neural networks in mechanical engineering.

2. Turbo charger and ANN basics

2.1 Turbo Chargers

The turbo charger's principle of functioning has practically remained unchanged since its invention by the swiss engineer Alfred Büchi in 1905. The turbo charger is made up of a compressor and a turbine connected through a rotating axis (Figure 5). The axis is supported by ball bearings, lubricated and cooled off by flowing oil. The turbine is propelled by the combustion chamber's hot exhaust gases, which in turn rotate the compressor, allowing the motor to intake more air and inject more fuel into each stroke.

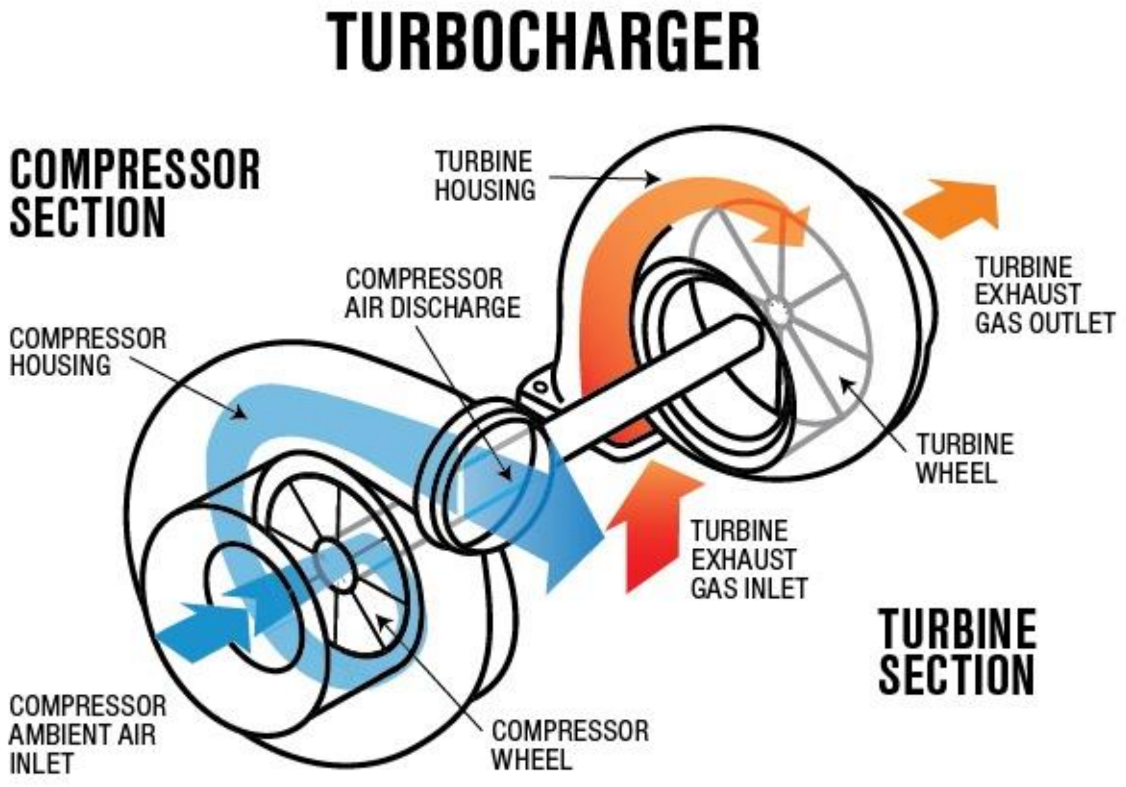


Figure 5: simplified turbo charger composition. (Eagle Ridge) [21]

The adiabatic efficiency of a compressor is usually defined as:

$$\eta_{is} = \frac{\dot{W_{\iota s}}}{\dot{W}} \tag{2.1}$$

The hypothesis of adiabatic functioning allows the reformulation of the efficiency as follows:

$$\mathrm{W} = \Delta H \quad \text{so:} \quad \eta_{is} = \frac{T_{2is} - T_1}{T_2 - T_1} \tag{2.2}$$

Considering the non-adiabatic phenomena leads to reformulating the compressor efficiency, considering the heat lost by the compressor, coming from the turbine:

$$\eta_{is,C} = \frac{\dot{W_{\iota s}}}{\Delta H - Q}, \qquad (2.3)$$

where Q is the heat lost by the compressor

P. Chessé et al. [6] highlight the influence of the inlet turbine temperature to the compressor efficiency by taking similar operational points. Figure 6 puts in perspective different inlet turbine temperatures and the corresponding compression ratios by mass flow. As a rule, the compressor has more difficulty applying its work onto the air flow at high intake. The different efficiency curves in function of the inlet turbine temperature may be withdrawn from Figure 7. Their method is explained more in detail under the "3. Bibliographic Study: characterization of the non-adiabatic performances of a turbo charger" section of this report.

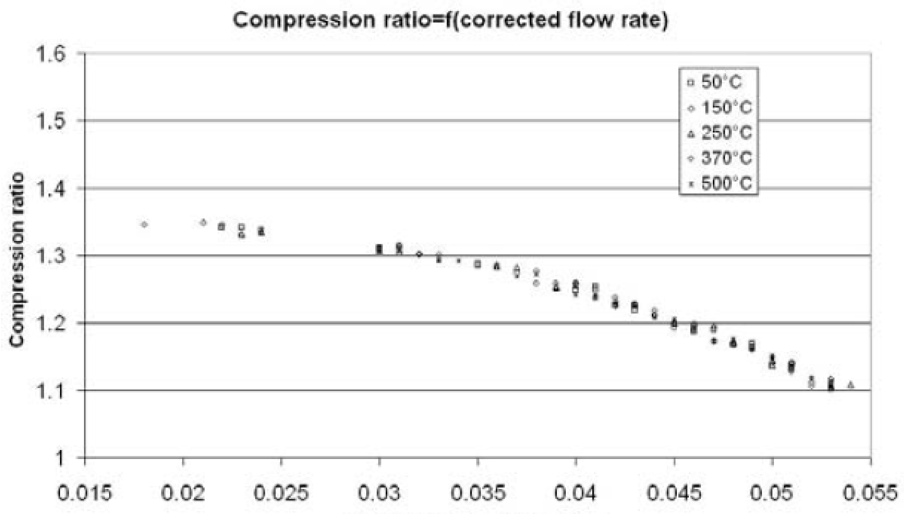


Figure 6: Evolution of the compression rate as a function of the corrected mass flow for several inlet turbine temperatures.

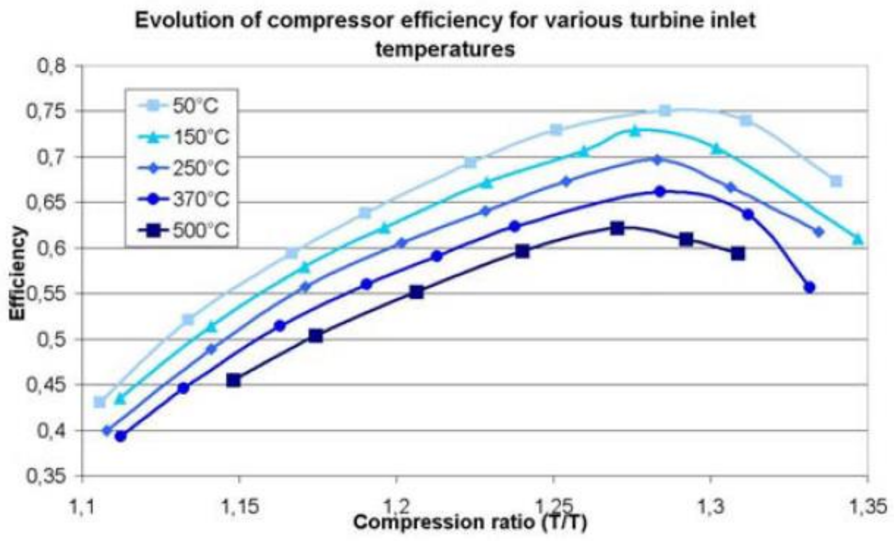


Figure 7: compressor efficiency as a function of the compression rate for several inlet turbine temperatures.

Using the same logic, the overestimation of the performances of the turbine under the adiabatic hypothesis may be acknowledged:

$$\eta_{is,T} = \frac{\Delta H - Q}{\dot{W}_{is,T}}, \tag{2.4}$$

where Q is the heat lost by the turbine, with negative sign

The heat exchanges which take place within a turbo charger are simplified on Figure 8.

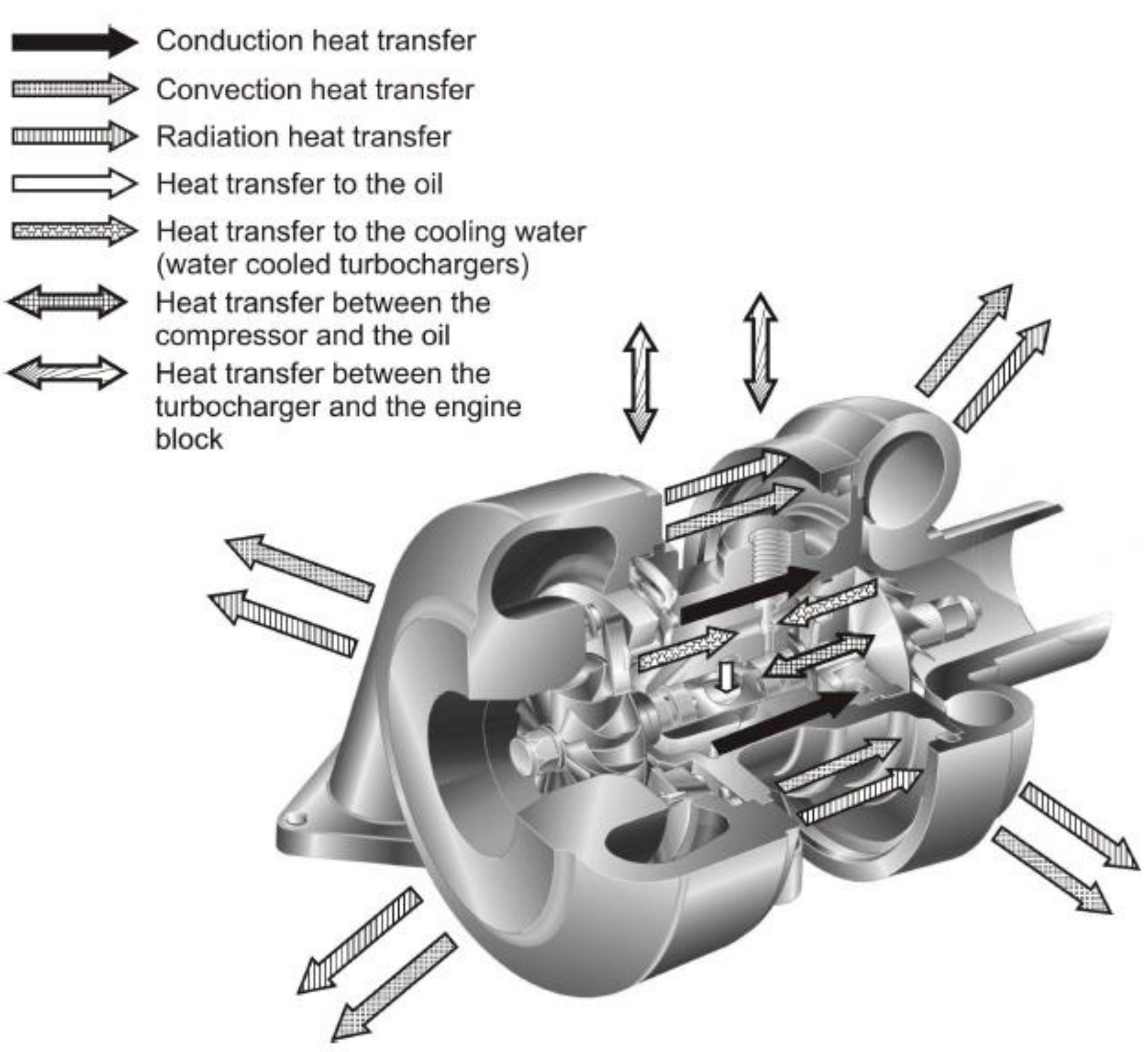

Conduction heat transfer
Convection heat transfer
Radiation heat transfer
Heat transfer to the oil
Heat transfer to the cooling water (water cooled turbochargers)
Heat transfer between the compressor and the oil
Heat transfer between the turbocharger and the engine block

Figure 8: mechanisms of heat transfer in/from the turbo charger. (Shaaban, p.12) [8]

The gaps between real operation and designer charts' predictions result from the errors in the forecasting of the performances of the turbo charger due to heat loss. The work presented here will try to humbly quantify the heat transfer not considered by the designers to refine the performance prediction for this device.

2.2 Artificial Neural Networks (ANNs)

Different network architectures are used to solve various problems with the ANN methodology, of two main categories: classification and approximation. The latter will be the one developed in this project. To solve both these problem types, the Multi-Layer Perceptron (MLP) network is of simplest application among ANNs and mainly addresses the solution of nonlinear behavioral problems, such as the one approached here.

The MLP is a feed-forward ANN, which means it takes an input and calculates an output, going through successive calculation layers. This algorithm can map a set of input data to a set of appropriate outputs.

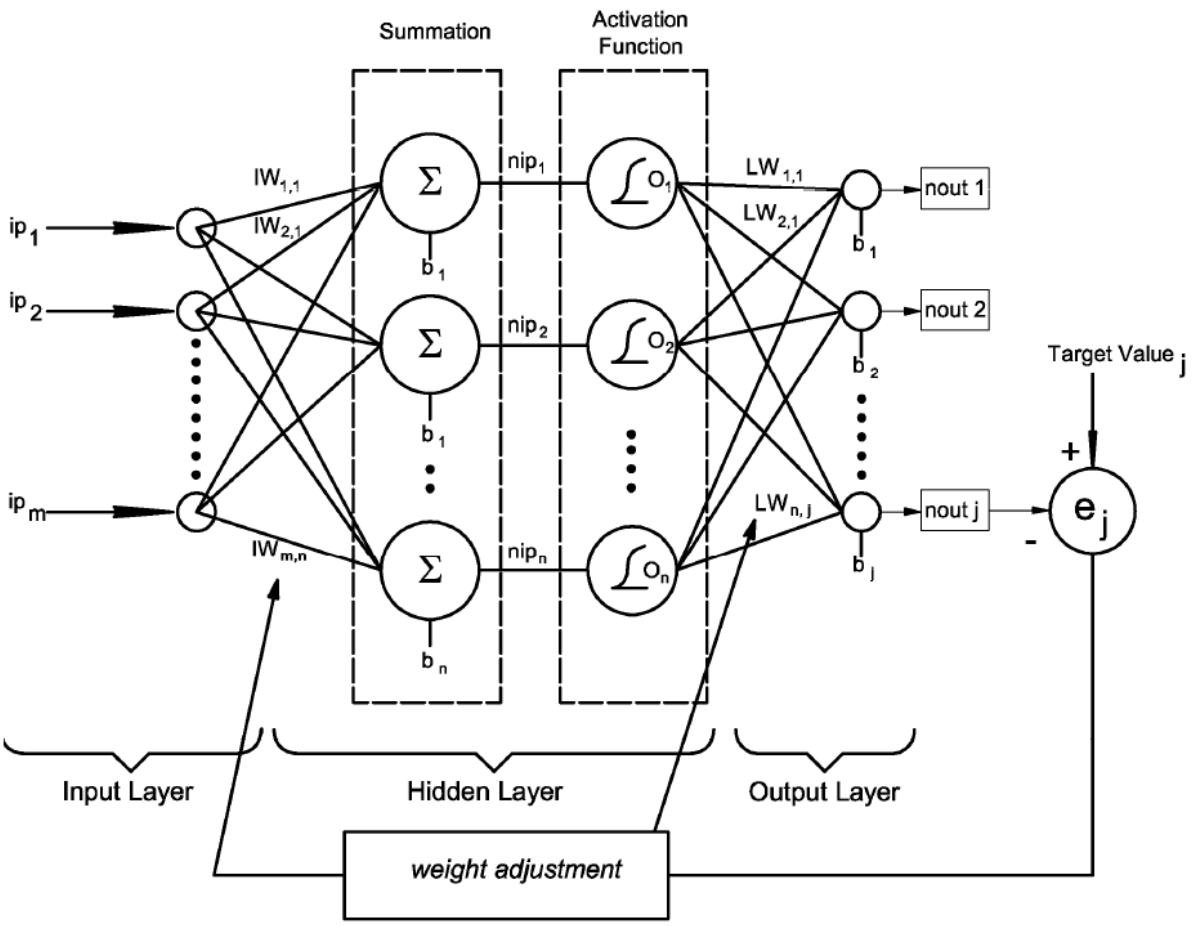


Figure 9: architecture of an MLP ANN. [14]

The MLP structure is mainly composed of three layers: input, hidden and output (Figure 9). The hidden layer is where the inputs are processed within the weights with the predefined nonlinear activation function (usually a sigmoid, but special for the turbo charger application, as explained on section “9. Correction of the convection coefficients using an artificial neural network”. Every example or occurrence fed into the network is called a pattern and noted “p”.

It is said that a neural network has completed an "epoch" when it has updated every weight element for every instance from the input.

The MLP algorithm is built as follows:

i) Initialization: the weights are initialized to random values near zero.

ii) Training [16,17,18,23]:

Repeat:

Take a new pattern p of x_i inputs at random, with the corresponding desired output.

- Present the pair (x_i, t_i) to the network.
- Calculate the hidden layer's output and then the final output (feed forward calculation step), then activate it with the activation function.
- Calculate the mean squared error (MSE).
- Update weights, starting from the output layer.

Until the MSE is inferior to a tolerance (tol1) or the number of maximum epochs is reached.

iii) the network is used to approximate or classify a pattern.

The mean squared error is defined as:

$$E_p = \frac{1}{2}\sum_j (t_{pj} - y_{pj})^2, \tag{2.5}$$

where t_{pj} and y_{pj} represent the expected and calculated outputs for the 'p' pattern at the j-th neuron

The weight update is usually defined as:

$$w_{ij}(k+1) = w_{ij}(k) - \eta \left.\frac{\partial E_p}{\partial w_{ij}}\right|_{w(k)}, \tag{2.6}$$

where w_{ij} is the weight between the current layer's i-th neuron and the next layer's j-th neuron, η is the learning rate and E_p is the mean squared error.

For training, any standard numerical optimization algorithm can be used to optimize the performance function. Beale stated [18], "there are a few key ones that have shown excellent performance for neural network training in which these optimization methods use either the gradient of the network performance with respect to the network weights or the Jacobian of the network errors with respect to the weights. The gradient and the Jacobian are calculated using a technique called the back-propagation algorithm, which involves performing computations backward through the network." When the error reaches a previously determined tolerance value, the training process is stopped [19].

The gradient descent method is used for this work and it is evaluated according to the following equations:

For the output layer:

$$\frac{\partial E_p}{\partial w_{ji}} = \delta_{pj}.y_{pi} \text{ where: } \delta_{pj} = -(t_{pj} - y_{pj}).y_{pj}.(1 - y_{pj}) \tag{2.7}$$

For the hidden layer:

$$\frac{\partial E_p}{\partial w_{ji}} = \sum_k (\delta_{pk} w_{kj}).y_{pj}.(1 - y_{pj}).y_{pi} \tag{2.8}$$

To accelerate training, a momentum term (α) may be applied to the error term:

$$w_{ij}(k+1) = w_{ij}(k) - \eta \sum_{t=0}^{n} \alpha^{n-t} \left.\frac{\partial E_p}{\partial w_{ji}}\right|_{w(k)} \tag{2.9}$$

This reinforces the weight update in the desired direction. The parameter alpha must be inferior to the unity and positive to ensure training convergence.

A common issue regarding network training is gradient vanish near local minima. To avoid this problem, the learning rate "η" and the momentum term "α" are decreased to a minimum value, low enough to stabilize learning and avoid divergence. A decay rate, noted "alph" here, is applied to these parameters after every epoch of training.

Both on hidden and output layers, activation functions are applied to optimize training. A commonly used activation function is the SoftMax:

$$softmax(x) = \frac{e^{x-\max(x)}}{\sum e^{x-\max(x)}} \tag{2.10}$$

This function returns a vector containing elements positive and inferior to the unity, with sum equal to the unity. The originally large values become closer to the unity and the smaller ones are reduced to close to zero.

The activation function is the most relevant challenge this work tries to surpass. As a custom activation function is needed, no pre-programmed and encrypted machine learning libraries may be used, such as TensorFlow and Keras on Python or Matlab Neural Network library, so the algorithm must be entirely written from scratch. A custom activation function will be responsible for taking the convection coefficients, applying them to the physical model of the turbo charger, and returning its efficiencies.

According to Öztemel [17] "the information that is produced during this process is measured and stored within these adjusted weights and it's hard to reveal and interpret this information." During these processes, the ANN learns the underlying function/physics of the system, while the results of the ANN learning are adjusted weights that could be used to accurately approximate the underlying function/physics of the system [20]. After the learning step, the network is tested with a different data set than that which was actualized before, and the performance of the network is analyzed [17].

Training is usually performed on a subset of the input data, called the training set. The percentage of the data commonly used if 75% for training and 25% for validation. The validation step takes the trained weights and evaluates the network's outputs, comparing them to expected ones, which gives a performance parameter.

When the weights are trained, the network may be used to predict output for an individual input.

3. Bibliographic study: characterization of the non-adiabatic performances of a turbo charger

Several works have been carried out with the goal to analyze the non-adiabatic performances of turbo chargers. This section will present and explain the advantages and disadvantages of some of the proposed methods, enabling the choice of one model over the others. This model's hypothesis and conditions will be further explained at the end of the section.

3.1 Proposed methods

A mechanical method was proposed by P. Podevin et al. [9] by measuring the torque on the central axis with a torque meter. This model, which aims to characterize the performances of turbo chargers based on heat loss, will not make use of such method, because a torque meter is not available.

G. Tanda et al. [10] suggests evaluating heat transfer by using infrared cameras, which would allow obtaining temperature gradients, as well as surface temperatures. This method, although interesting, is not accessible for this project's budget.

In order to quantify heat transfer, Bohn et al. [7] developed an approach using a meshing of the turbo charger. The turbo charger is divided into two parts: a solid part, on which Fourier's equation is solved and a fluid part, on which Navier Stokes' equations are solved. The temperature limiting conditions were set using images taken with an infrared camera and measures from thermocouples installed on the equipment's surface.

This method uses a rather fine mesh (40 million points), of great complexity, therefore it has not been retained for this project, as the goal here is to develop a simple and practical method to evaluate heat transfer in any turbo charger.

P. Chessé et al. [6] presented a method of correlation based on experimental results anticipating the influence parameters on heat transfer. Based on these tests run on cabin-temperature inlet flow, the isentropic efficiencies were calculated for the functioning points given and reused to calculate the heat transfer for the tests on higher temperature inlet flow. Once the correlations were obtained, the model proves itself rather efficient, although it does not consider heat transfer at a theoretical level.

Despite the convincing results, this project shall not use correlations of such type, too specific, given the goal to develop a replicable model for other turbo chargers.

Even without retaining the methods employed by Chessé et al. [6] and Bohn et al. [7] to determine heat transfer, a description of the heat exchanges along the turbo charger (Figure 10) will be used, and it is presented in the following paragraphs.

The turbine is the source for all the mechanical power and heat, from which a part is dissipated to the outside and another part feeds and heats up the compressor. The different components will exchange heat with the exterior in the form of radiation and convection. The power balance for each component will be detailed in the section "5. Physical Model".

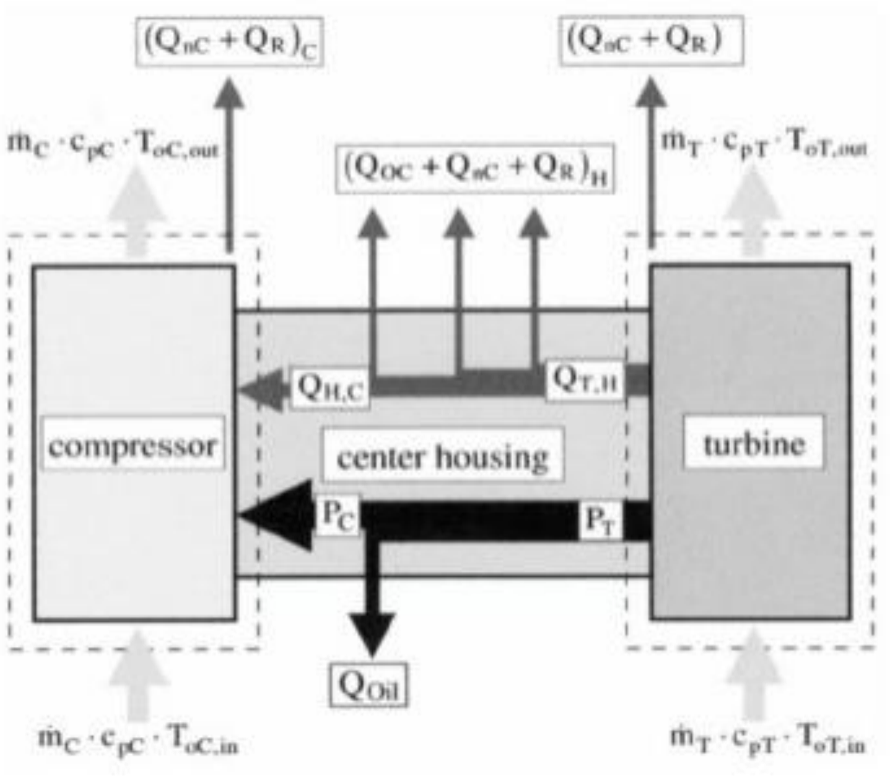

$(Q_{nC}+Q_R)_C$
$(Q_{oC}+Q_R)$
$\dot{m}_C \cdot c_{pC} \cdot T_{oC,out}$
$(Q_{OC}+Q_{nC}+Q_R)_H$
$\dot{m}_T \cdot c_{pT} \cdot T_{oT,out}$
$Q_{H,C}$
$Q_{T,H}$
compressor
center housing
turbine
P_C
P_T
Q_{Oil}
$\dot{m}_C \cdot c_{pC} \cdot T_{oC,in}$
$\dot{m}_T \cdot c_{pT} \cdot T_{oT,in}$

Figure 10: model presented by P. Chessé et al. [6], Bohn et al. [7].

M. Cormerais [2] suggests a model of resistances for the transfer emanating from the turbine. The first resistance is the turbine plateau, where the heat splits towards the oil and the central body. It then reaches the compressor plateau's resistance and is delivered to the compressor. The resistance diagram is represented on Figure 11.

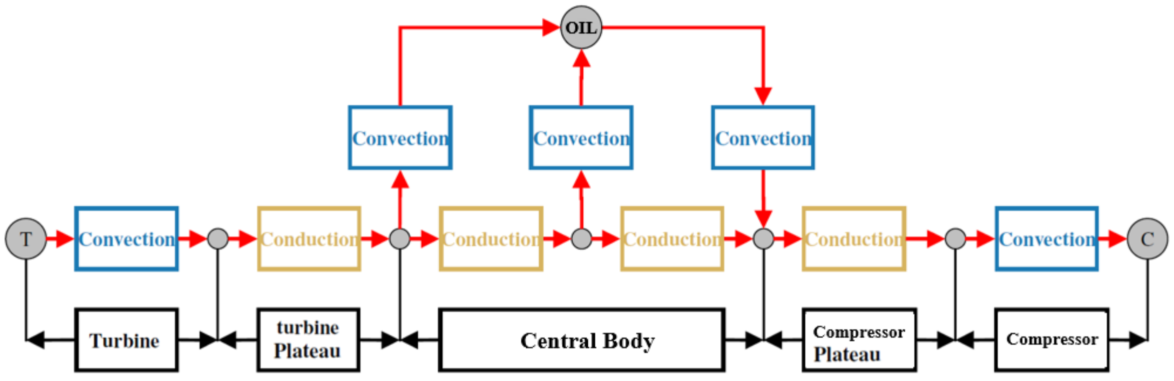


Figure 11: resistance model suggested by M. Cormerais [2].

Instead of using the resistance method as suggested, only the division setting will be used on this project. That is, the turbo charger will be decomposed into 5 parts: turbine, turbine plateau, central body, compressor plateau and compressor, with limiting conditions between each one of them, making it possible to determine the 1D temperature evolution function along the turbo charger axis. The equations to determine this evolution function will also be borrowed from M. Cormerais [2] and detailed under the section "5. Physical Model".

3.2 Applications of Artificial Neural Networks (ANNs) in Mechanical Engineering

Kalogirou SA et al. [25] gathers and implements several applications of artificial neural networks on energy systems, ranging from the starting-up of the solar steam generating plant at an optimal energy-saving time to a long-term performance prediction of solar domestic water heating systems. For the computational resources available at the time of the article (2001), the results are rather remarkable.

In 2013, O. Özener et al. [14] have applied an ANN to predict engine emissions and performance parameters of a turbo charger diesel engine. In their methodology, they use a Multi-Layer Perceptron network with three layers, achieving reliable training results. Table 1 contains the parameters modelled and the results obtained with their work.

Output	***Training***		***Validation***		***Testing***		***ALL***		***NN****
	R	***MSE***	***R***	***MSE***	***R***	***MSE***	***R***	***MSE***	
CO_2	0.9999	1.6E-17	0.9959	2.3E-1	0.99836	9.6E-1	0.9979	6.0E-1	13
CO	0.9992	3.8E-3	0.9938	3.3E-2	0.99442	1.2E-2	0.9969	1.2E-2	5
NO_X	0.9999	1.0E-2	0.9898	5.6E-2	0.99896	5.9E-2	0.9994	8.5E-2	14
FSN	0.9999	4.4E-6	0.9668	4.7E-3	0.96424	4.9E-3	0.9982	2.0E-3	16
THC	0.9463	1.6E-1	0.7307	3.4E-1	0.73600	3.2E-1	0.8494	2.4E-1	13
Torque	0.9999	4.1E-2	0.9996	1.2E-2	0.99957	1.9E-2	0.9998	8.4E-1	14
Power	0.9999	4.5E-3	0.9999	2.4E-2	0.99984	6.7E-2	0.9993	2.7E-2	7
BSFC	0.9999	1.9E-5	0.9990	1.3E-2	0.99757	3.6E-2	0.9992	1.3E-1	11

Table 1: Artificial neural network application on diesel engines, to predict performance parameters. 'R' stands for regression, NN for the number of neurons at hidden layer and 'MSE' is the mean squared error. O. Özener et al. [14]

No applications of neural networks to turbo charger performance prediction have been recorded to this date.

4. Geometrical Characteristics of the turbo charger

This model will focus on the turbo charger K9KGEN5, analyzed on G. Salameh's doctorate thesis [1]. Figures 12, 13 and 14 contain the geometrical characteristics employed. In order to better represent the turbo charger, some additional measures were taken/estimated and are shown on Tables 2, 3 and 4.

Measure	Value in mm	Description
Ds_C	37	Compressor's volute's outlet diameter
D_PC	111	Compressor's plateau's external diameter
Le_C	25	Compressor's inlet pipe length
Ls_C	102,44	Compressor's outlet pipe length, from the volute's spiral
Dext_C	50	Compressor's inlet pipe external diameter
De_int_C	36	Compressor's inlet pipe internal diameter

Table 2: measures taken using a caliper for the K9KGEN5 turbo charger's compressor. [1]

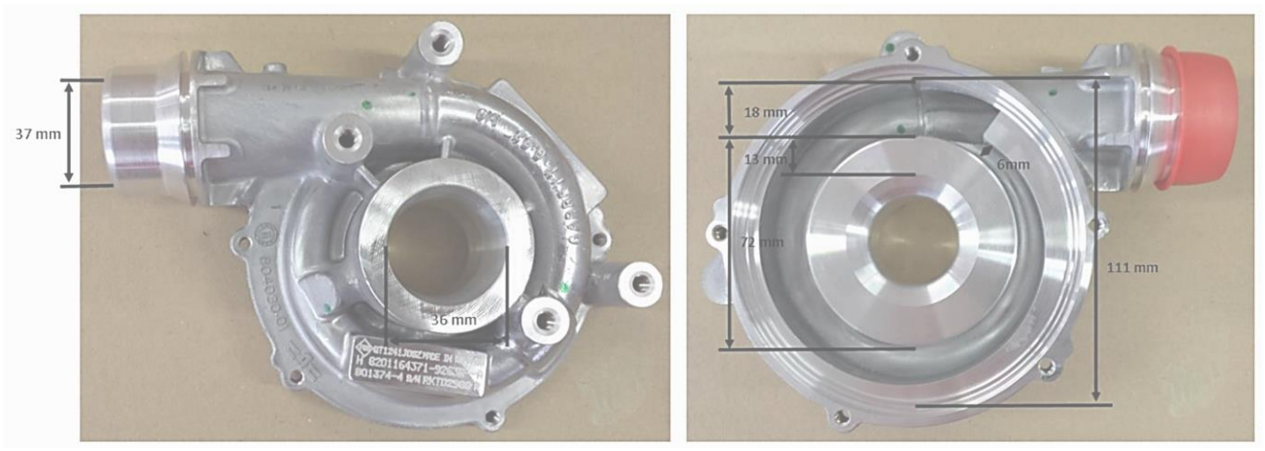


Figure 12: compressor component for the K9KGEN5 turbo charger.

Measure	Value in mm	Description
D_cc	40	Central Body external diameter
L_cc	41,1	Central body length (without the plateaus)
e_Plto	10,1	Plateaus' thickness
D_axe	10	Central axis diameter
D_h	11	Central body internal diameter

Table 3: measures taken with a caliper on the central body of the K9KGEN5 turbo charger. [1]

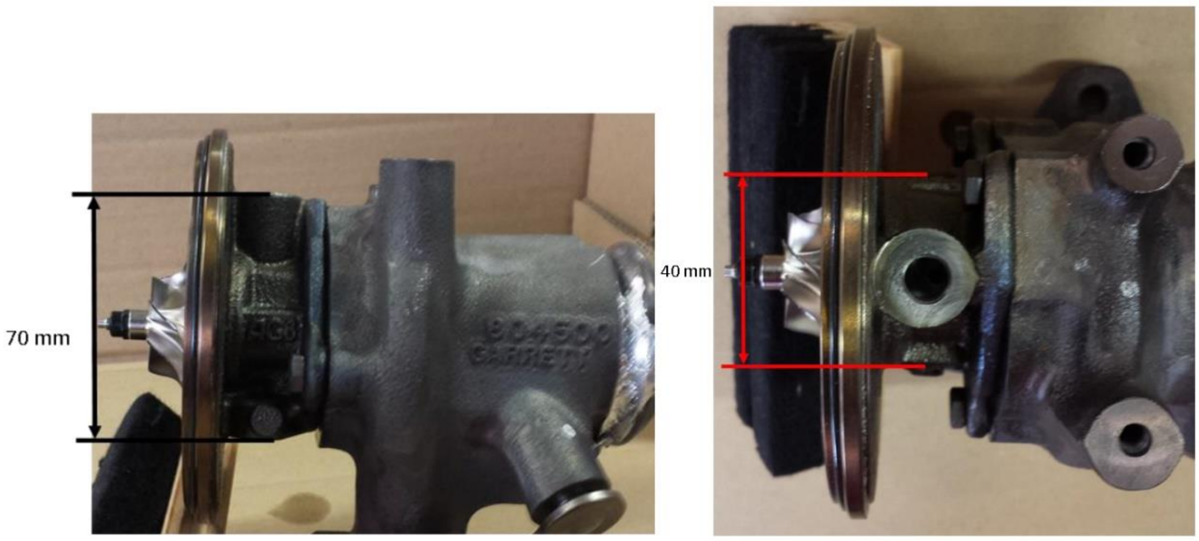


Figure 13: turbine component from the K9KGEN5 turbo charger. [1]

Measure	Value in mm	Description
De_T	35	Turbine volute's inlet pipe diameter
L_entrée_volute	45	Turbine volute's inlet pipe length
L_sortie_volute	60,6	Turbine's outlet pipe length
L_sortie_volute_totale	130	Turbine volute total length (from plateau to outlet)
D_moyen_ext_T	58,9	Mean diameter of the geometry between the turbine plateau and the outlet pipe
D_PT	40	Turbine plateau diameter
Ds_PT	46	Turbine plateau external diameter

Table 4: measures taken with a caliper from the K9KGEN5 turbo charger's turbine. [1]

From the given values and measures, it has been possible to calculate the geometrical parameters of interest to the model: the perimeters (Table 5) and the surfaces (Table 6).

Measure	Value in mm	Description
P_h	12,6	Perimeter of the oil flow section
P_ext_PT	125,7	Turbine plateau external perimeter
P_ext_cc	125,7	Central body external perimeter
P_ext_PC	348,7	Compressor plateau external perimeter

Table 5: perimeters of interested calculated for the K9KGEN5 turbo charger. [1]

Measure	Value in mm^2	Description
A_cc	1244,1	Central body cross section
A_h	12,6	Oil flow cross section
A_ext_T	29003,2	Turbine volute external surface
A_PT	1256,6	Turbine plateau cross section
A_ext_cc	5164,8	Central body total external surface
A_PC	9676,9	Compressor plateau cross section
A_ext_C	21610,5	Compressor volute total external surface
A_ext_PT	1269,2	Turbine plateau total external surface
A_ext_PC	3522,0	Compressor plateau total external surface

Table 6: surfaces of interest calculated for the K9KGEN5 turbo charger. [1]

The turbine volute total external surface was assimilated to a union of two intersecting cylinders: one at the air inlet and the other at the outlet. On the first, a constant mean diameter was estimated, based on the different diameter measures taken along the volute.

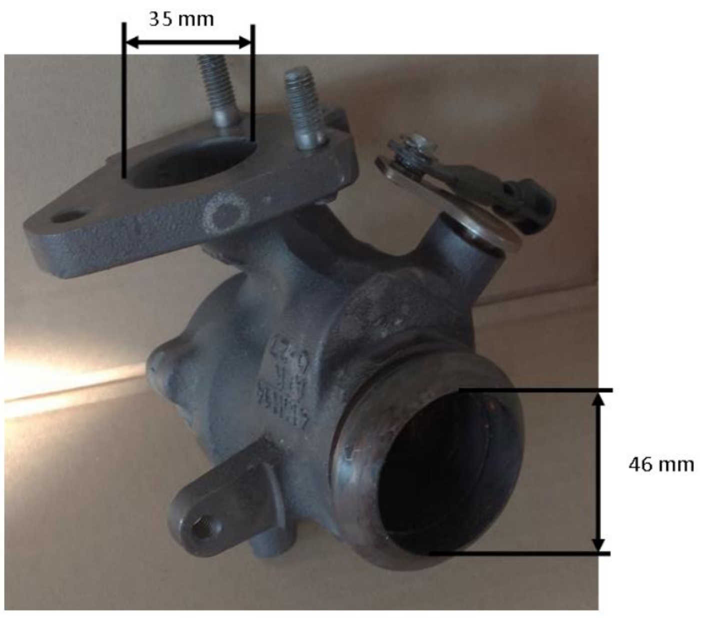


Figure 14: elementary measures of the K9KGEN5 turbo charger's fixed-geometry turbine. [1]

These values are filled out in an MS Excel sheet named 'Geometrical_Data.xlsx' and imported by the Python program to run the ANN, which makes possible the generalization of the model. To apply it to another turbo charger, just update this worksheet and provide new experimental data.

5. Physical Model

The model presented here has the goal to quantify the heat transfer taking place during the use of a turbo charger, by means of experimental data. On Table 7, the experimental input data is detailed.

Data Title	Description
T_Ce	Compressor inlet gas temperature
T_Cs	Compressor outlet gas temperature
T_Te	Turbine inlet gas temperature
T_Ts	Turbine outlet gas temperature
T_h	Lubrication oil inlet temperature
Tp_ext_C_volute	Compressor external wall mean temperature
Tp_ext_T_volute	Turbine external wall mean temperature
Dm_T	Turbine gas flow
Dm_C	Compressor gas flow
Dm_h	Oil flow
Régime	Axis rotating speed
Text	Cabin temperature

Table 7: model's input data.

This input data is given for 3 different experimental settings according to G. Salameh's [1] Doctorate's thesis on the turbo charger presented on "4. Geometrical Characteristics of the turbo charger":

- Cold test
- Test with inlet turbine temperature at 300°C
- Test with inlet turbine temperature at 580°C

A model previously employed by P. Chessé et al [6] is used to describe the different energy exchanges along the turbo charger and is presented on Figure 10. Based on this model, a simplified diagram was developed to express the thermal exchanges between the turbo charger's different components. Figure 15 contains this diagram, as well as the notation which will be used for the rest of the project.

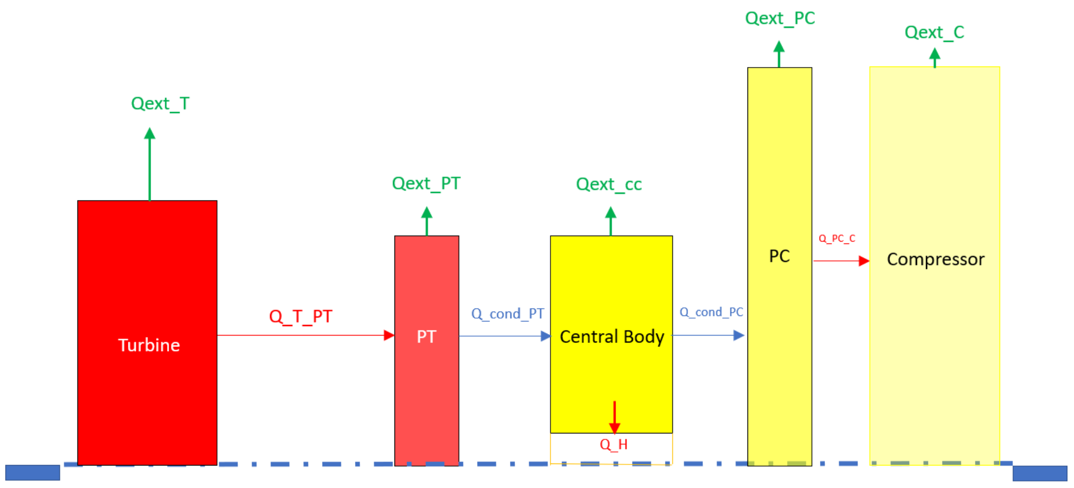


Figure 15: diagram of the modelled heat exchanges.

The different hypotheses are presented hereafter, as well as the modelling and the power balance for each part of the turbo charger. The evaluation of the physical data required (convection, radiation and conduction coefficients) will be detailed in the following section.

NOTE: For practical reasons, all power variables are considered positive disrespecting thermodynamics' conventions. The same applies to the model's Python code.

5.1 Interfaces Turbine / Turbine Plateau/ Central Body

By expanding the exhaust flow, the turbine transmits a mechanical power to the axis. It is the turbo charger's hot spot. Therefore, it transfers heat to the other parts of the turbo charger and towards the outside and will never be considered receiving heat on this model. The turbine plateau will be attached to the central body block and will not be considered as a part of the turbine block. The turbine block will be considered at constant temperature, the mean of the input and output temperatures. The turbine plateau will follow a temperature 1D evolution function calculated from the power balance on the elementary cutaway presented.

Figure 16 shows a small enough element on the length of the turbine plateau, and its corresponding heat exchanges. All its heat is conducted from the turbine, a small portion is dissipated on its walls, and some of the original heat is conducted towards the central body.

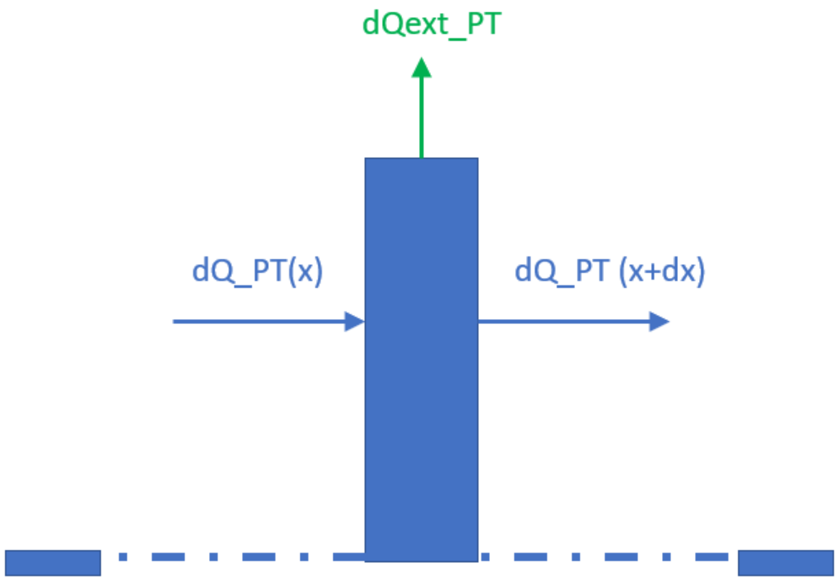


Figure 16: diagram of the elementary cutaway of the turbine plateau. The heat exchanges in blue represent conductions and the ones in green a natural convection.

The heat transfer mechanism from the turbine up to the turbine plateau is represented on Figure 17. The heat coming from the turbine's mean flow is partly dissipated on the volute external walls and partly transferred through forced convection to the turbine plateau.

On the turbine plateau, the heat initially from a forced convection is the conducted to the other end, part of it being dissipated on the external walls.

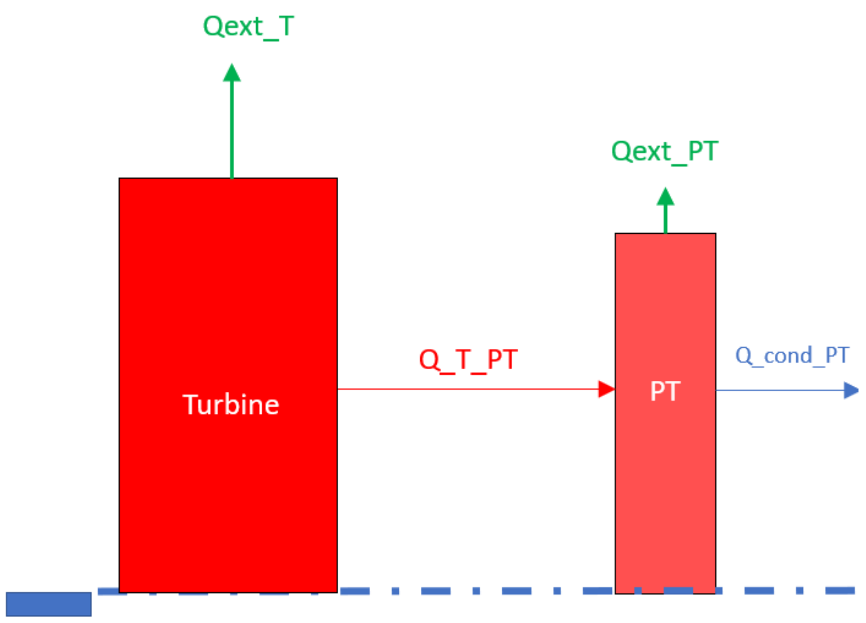


Figure 17: representative diagram of the interface Turbine / Turbine Plateau. The heat in blue represents a conduction, and the ones in green natural convections, being those in red forced convections.

The total turbine power is the sum of the useful power (P_{arbre_T}) and the heat dissipated:

$$P_T = Dm_T.\ Cp_T.\ \Delta T_T = P_{arbre\_T} + Q_{T\_PT} + Q_{ext\_T} \tag{5.1}$$

where the heat dissipated on the volute walls is radiative and naturally convective:

$$Q_{ext\_T} = Q_{nc,T} + Q_{RT} \tag{5.2}$$

$$Q_{nc,T} = h_{ext\_T}.\ A_{ext\_T}.\ (T_{p\_ext\_T\_volute} - T_{ext}) \tag{5.3}$$

$$Q_{RT} = \varepsilon.\sigma.A_{ext\_T}.\ \left(\left(T_{p\_ext\_T\_volute}\right)^4 - (T_{ext})^4\right) \tag{5.4}$$

The heat transfered at the interface with the turbine plateau is defined as a forced convection between the mean turbine flow temperature and the temperature at the extremity of the turbine plateau:

$$Q_{T\_PT} = h_{T\_PT}.A_{PT}.(T_{moy\_T} - T_{PT}\ (0)) \tag{5.5}$$

According to M.Cormerais's doctorate thesis [2], the temperature for each part of the turbo charger may be defined as the solution of a second order differential equation. Three equations are thus defined, for the turbine plateau, the central body and the compressor plateau. The first of these is defined as:

$$T_{PT} = C_1.e^{\sqrt{\alpha_1}\ .\ x} + C_2.e^{-\sqrt{\alpha_1}\ .\ x} + T_{ext} \tag{5.6}$$

where $\alpha_1 = \frac{h_{ext\_T}.P_{ext\_PT}}{\lambda.A_{PT}}$ (5.7)

and C_1 and C_2 are constants to be defined.

5.1.1 Limiting Conditions

A balance of the heat flow source is considered.

All the heat coming from the convection of the mean turbine flow crosses the volute walls and the turbine plateau (Q_{T_TOT}), turning partly into radiation on the turbine plateau surface (Q_{RT}) partly into conduction towards the turbine plateau (Q_{cond_PT}) and partly into natural convection on the external wall of the turbine plateau:

$$Q_{T\_TOT} = Q_{nc\_T} + Q_{RT} + Q_{cond\_PT} \tag{5.8}$$

$$\begin{aligned} h_{T\_PT} \,.\, A_{TOT}.\,(T_{moy_T} - T_{PT})\Big|_{XPT=0} = \\ h_{ext\_T}.\,A_{ext\_T}.\,(T_{p\_ext\_T\_volute} - T_{ext}) \\ +\varepsilon.\sigma.A_{ext\_T}.\left(\left(T_{p\_ext\_T\_volute}\right)^4 - (T_{ext})^4\right) \\ -\lambda \,.\, A_{T\_PT} \,.\frac{\partial T_{PT}}{\partial x}\Big|_{XPT=0} \end{aligned} \tag{5.9}$$

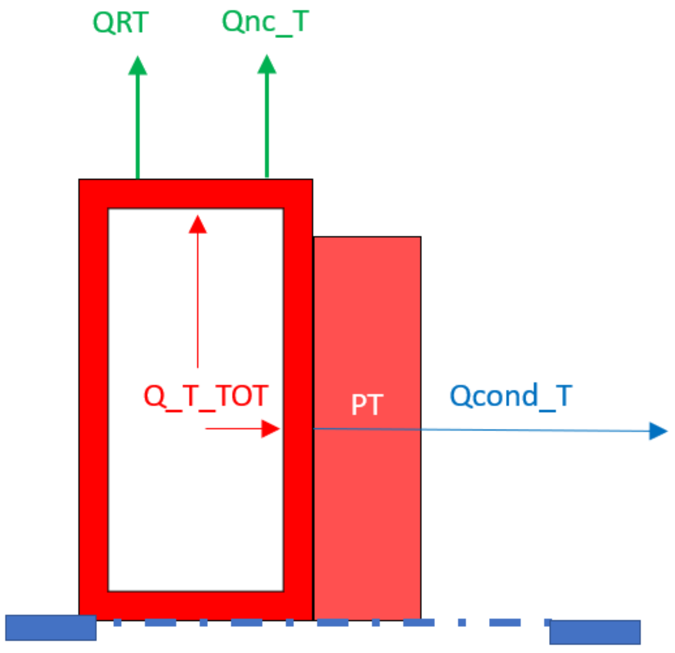


Figure 18: heat transfer diagram considering the limiting condition on the interface turbine/turbine plateau.

The conductive heat transfer at the end of the turbine plateau is entirely converted into a conductive heat flow in the central body (Figure 18):

$$-\lambda \,.\, A_{T\_PT} \,.\, \frac{\partial T_{PT}}{\partial x}\Big|_{XPT=e\_Plto} = -\lambda \,.\, A_{CC} \,.\, \frac{\partial T_{CC}}{\partial x}\Big|_{XCC=0} \tag{5.10}$$

The temperature continuity must be assured at the interface between the turbine plateau and the central body:

$$T_{PT}\ |_{\text{XPT = e\_Plto}} = T_{CC}\ |_{\text{XCC = 0}} \quad (5.11)$$

5.1.2 Hypothesis

$-\ Q_{T\_PT}$: for the heat transfer from the turbine towards the central body, a conductive exchange will be considered with the turbine plateau (the part in total contact with the central body). For this calculation, the mean gas temperature inside the turbine will be used (evolution function previously described), as well as a convection coefficient and an exchange surface (A_{PT}), which was evaluated under the section “4. Geometrical Characteristics of the turbo charger”.
$-\ (Q_{nC}, Q_R)_C$: as for the convective and radiative exchanges, the external surface of the volute and its mean temperature shall be considered. For this purpose, G. Salameh [1] measured two-point temperatures on this external wall during his experiments. An emissivity value as well as a convection coefficient will need to be evaluated.

5.2 Central body and interface central body / compressor plateau

The turbine and compressor plateaus compose the central body block, but the oil as well as the axis will be excluded. The temperature gradient being primarily important in this part of the turbo charger, the temperature will follow a one-dimensional law along the axis and will be determined by a balance on the elementary cutaway of the central body.

Figure 19 shows a cutaway element on the length of the central body, and its corresponding heat exchanges. All its heat is conducted from the turbine plateau, a small portion is dissipated on its walls, a fair amount is dissipated by the contact with the oil, and finally some of the original heat is conducted towards the compressor plateau.

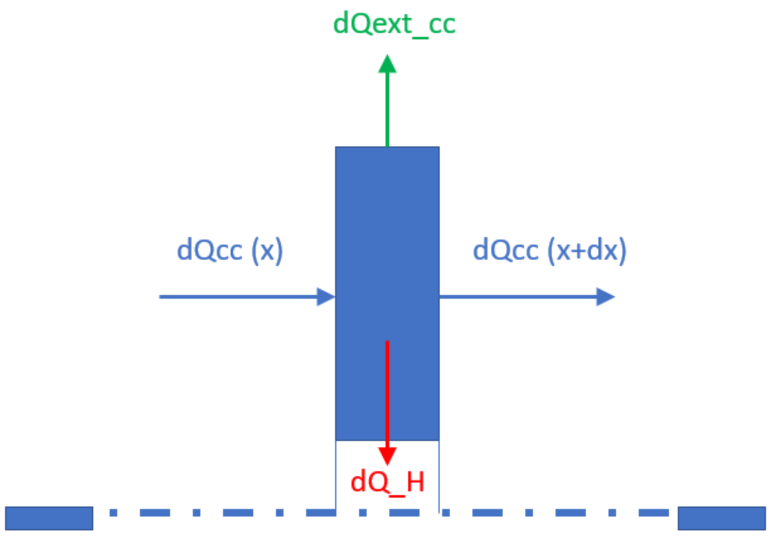


Figure 19: diagram representing the elementary cutaway of the turbine plateau. The heat exchanges in blue represent conduction, the exchange in green a natural convection and the one in red a forced convection.

The heat transfer mechanism from the central body to the compressor is represented on Figure 20. The heat coming from the turbine plateau is partly dissipated on the central body and partly conducted through to the compressor plateau. Here again some of the incoming heat gets lost on the external walls and a part is transferred through to the compressor mean flow.

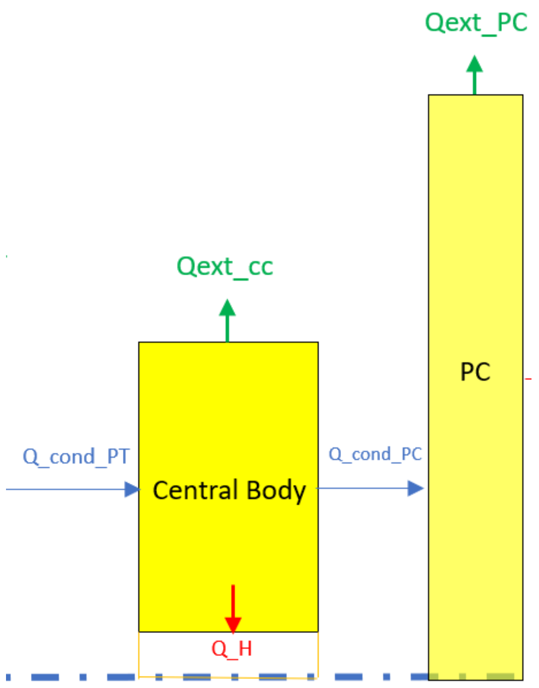


Figure 20: diagram representing the central body. The heat exchanges in blue represent conductions, the one in green a natural convection and the one in red a forced convection.

The heat transferred by the contact with the oil may be expressed as the sum of the elementary contribution of the forced convection with the flow perimeter (P_h):

$$Q_H = \int_0^{Lcc} h_h . P_h . (T_{CC}(x) - T_h) . dx \qquad (5.12)$$

An analogous sum is used to evaluate the heat dissipated on the central body's external walls (Q_{ext_cc}).

The second solution proposed by M.Cormerais [2] regards the central body's temperature evolution law and is expressed as:

$$T_{CC} = C_3 . e^{\sqrt{\alpha_2} . x} + C_4 . e^{-\sqrt{\alpha_2} . x} + \Delta \qquad (5.13)$$

where $\alpha_2 = \frac{h_h . P_h + h_{ext\_cc} . P_{ext\_cc}}{\lambda . A_{CC}}$ (5.14)

and $\Delta = \frac{h_h . P_h . T_h + h_{ext\_cc} . P_{ext\_cc} . T_{ext}}{h_h . P_h + h_{ext\_cc} . P_{ext\_cc}}$ (5.15)

5.2.1 Limiting conditions

The conductive heat flow which gets to the end of the central body becomes partly heat lost by natural convection at the compressor plateau and partly conductive heat at the end of the plateau. This modification on M. Cormerais' [2] model is necessary to assure the flow coherence in the presented model.

$$- \lambda . A_{CC}. \frac{\partial T_{CC}}{\partial x}\Big|_{XCC = e\_Plto} = Q_{ext,\,PC} - \lambda . A_{PC}. \frac{\partial T_{PC}}{\partial x}\Big|_{XPC = e\_Plto} \qquad (5.16)$$

Originally, the condition was to impose a continuity of conductive heat flow at the interface central body / compressor plateau. This hypothesis seemed unrealistic, given that the cross-section variation is of order 6 and the heat flow does not transform entirely into conductive flow instantly at the compressor plateau.

To impose a temperature continuity at the interface central body / compressor, it is necessary that:

$$T_{CC}\,|_{XCC = L\_cc} = T_{PC}\,|_{XPC = 0} \qquad (5.17)$$

5.3 Interface Compressor Plateau / Compressor

On this model, the compressor receives heat from the plateau and loses heat towards the outside through natural convection and radiation on its walls. The heat exchanges on the compressor will be considerably lower than those observed on the turbine.

Figure 21 shows a cutaway element on the length of the compressor plateau, and its corresponding heat exchanges. All its heat is conducted from the central body, a small portion is dissipated on its walls, and finally some of the original heat is conducted towards the compressor plateau.

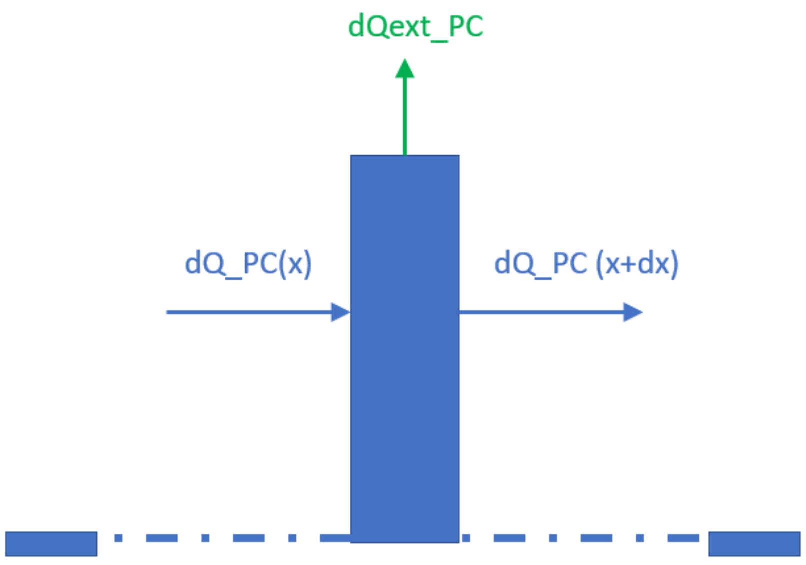


Figure 21: diagram representing the elementary cutaway of the compressor plateau. The heat exchanges in blue represent conductions, the one in green a natural convection.

The heat transfer mechanism from the compressor plateau to the compressor is represented on Figure 22. The heat coming from the central body is partly dissipated on the compressor plateau and partly conducted through to the compressor, where it exchanges with the mean compressor flow. Here again some of the incoming heat gets lost on the external walls and a part is transferred through to the compressor mean flow.

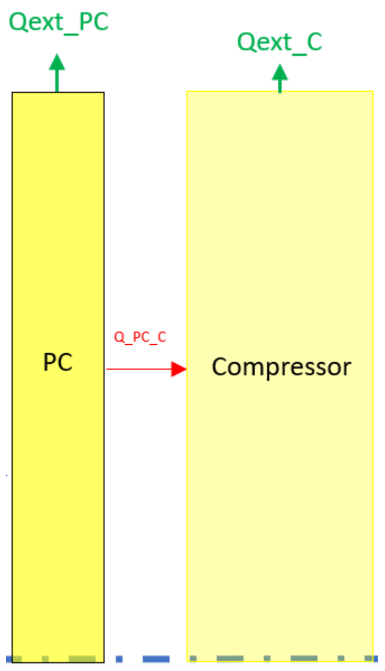


Figure 22: diagram representing the compressor. The heat in green represents a natural convection and the one in red a forced convection.

The total compressor power considers the heat ultimately received from the turbine (Q_{C_PC}) and the heat lost to the outside ($Q_{ext,\ C}$):

$$P_C = Dm_C.\ Cp_C.\ \Delta T_C = P_{arbre\_C} + Q_{C\_PC} - Q_{ext,\ C} \tag{5.18}$$

$$P_{arbre\_C} = P_C - Q_{C\_PC} + Q_{ext,\ C}$$

where each of the heat exchanges is defined as:

$$Q_{ext,\ C} = Q_{nc,\ C} + Q_{RC} \tag{5.19}$$

$$Q_{nc,\ C} = h_{ext\_T}.A_{ext\_T}.(T_{p\_ext\_T\_volute} - T_{ext}) \tag{5.20}$$

$$Q_{RC} = \varepsilon\_a.\sigma.\ A_{ext\_C}.\ \left(\left(T_{p\_ext\_C\_volute}\right)^4 - (T_{ext})^4\right) \tag{5.21}$$

The forced convection is defined between the mean compressor flow and the temperature at the extremity of the compressor plateau:

$$Q_{C\_PC} = h_{C\_PC}.A_{PC}.\ (T_{PC}(e\_Plto) - T_{moy\_C}) \tag{5.22}$$

The third and last of M.Cormerais's [2] equations is the temperature evolution law on the compressor plateau, defined as:

$$T_{PC} = C_5.e^{\sqrt{\alpha_3}\,.\,x} + C_6.e^{-\sqrt{\alpha_3}\,.\,x} + T_{ext} \tag{5.23}$$

where $\alpha_3 = \frac{h_{ext\_C}.P_{ext\_PC}}{\lambda.A_{PC}}$ (5.24)

5.3.1 Limiting condition

All the heat coming from the conduction at the end of the central body crosses the volute walls ($Q_{cond\_fin\_PC}$) and turns partly into radiation on the compressor volute surface (Q_{RC}), partly into convection towards the mean compressor flow (Q_{C_TOT}) and partly into natural convection on the wall of the compressor volute (Q_{nc_C}):

$$Q_{cond\_fin\_PC} = Q_{nc\_C} + Q_{RC} + Q_{C\_TOT} \tag{5.25}$$

$$\begin{aligned} - \lambda \,.\, A_{C\_PC}\,.\frac{\partial T_{PC}}{\partial x}\bigg|_{XPC\,=\,e\_Plto} = \\ h_{ext\_C}.\,A_{ext\_C}.\,(T_{p\_ext\_C\_volute} - T_{ext}) \\ + \varepsilon\_a.\sigma.A_{ext\_C}.\,\left(\left(T_{p\_ext\_C\_volute}\right)^4 - (T_{ext})^4\right) \\ + h_{C\_PC}\,.\,A_{PC}.\,(T_{PC} - T_{moy\_C})\big|_{XPC\,=\,e\_Plto} \end{aligned} \tag{5.26}$$

This limiting condition, illustrated on Figure 23, is in accordance with the operation on heat, which is the goal of this project. A reevaluation would be necessary for applications in other situations.

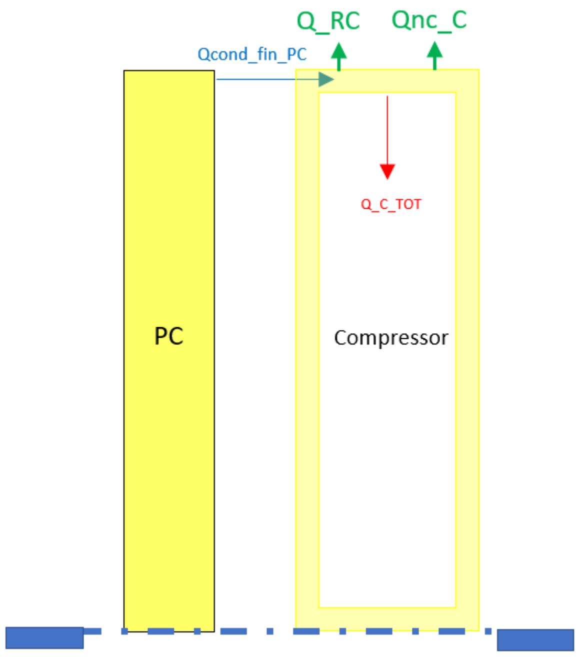


Figure 23: diagram of the heat transfers considering the new limiting condition at the interface compressor plateau / compressor.

6. Coefficient choice using correlations

On a first approach, heat convection coefficients will be taken from correlations. These shall later be revised and "learned" from an artificial neural network.

6.1 Forced convection coefficients

According to A. Romagnoli et al. [11], the convective coefficients on the compressor and turbine plateaus may be calculated using the following empirical relations:

$$\text{h\_T\_PT (or h\_C\_PC)} = \frac{2\lambda}{D_{T,C}} Re^{0.8} Pr^{0.4} \tag{6.1}$$

6.2 Natural convection coefficient

The natural convection coefficient for a turbo charger may be calculated through the expression:

$$h = \frac{\lambda\, Nu}{D_{T,C}}, \tag{6.2}$$

where Nu (Nusselt Number) is calculated according to [4], by:

$$Nu = \left\{ 0.60 + \frac{0.387\, Ra^{1/6}}{\left[1+(\frac{0.559}{Pr})^{9/16}\right]^{8/27}} \right\}^2, \tag{6.3}$$

Gr (Grashof number), Ra (Raleigh number) and Pr (Prandt number) are calculated by:

$$Gr = \frac{D^3 \rho^2 g \Delta T \beta}{\mu^2}, \tag{6.4}$$

where:

g – gravity
β – dilatation coefficient (equal to $1/T_{moyenne}$)
ΔT – difference between the wall temperature and the cabin temperature
μ - dynamic viscosity

$$Pr = \frac{\mu\, C_p}{\lambda} \tag{6.5}$$

$$Ra = Gr\, Pr \tag{6.6}$$

The natural convection coefficient on the central body is calculated as a mean of the coefficients of the turbine and the compressor. The turbine's coefficient is used for the turbine plateau and the compressor's for the compressor plateau.

In order to consider radiation on the surfaces of the turbine plateau and of the central body, the radiative transfer has been evaluated considering a linear temperature evolution law. This law is then integrated over the parts' lengths:

$$\varepsilon.\sigma.p_{ext}.\left((T_p)^4 - (T_{ext})^4\right) dx \tag{6.7}$$

This has provided an estimate of the radiated power, leading to a fair adjustment of the convective coefficient.

Comments:

- Given the high radiated power, this correction has not been applied to the 580°C case.
- The compressor plateau is too cold to justify such correction.

6.3 Forced convection coefficient for the oil

Based on S. Shaaban's [8] method, the importance of the oil in the heat transfer along the turbo charger is better understood.

According to his works on the GTI749V55 turbo charger, the heat dissipated into the oil may be expressed as:

$$\frac{Q_{B,fri}}{\dot{W}_C} = 491627.1\ \exp(-7.87 Re_{oil,u}^{-0.12} Sr^{0.14}) \tag{6.8}$$

This ratio reaches up to 3, which proves the importance of the oil in the heat transfer.

He also develops a correlation law to determine the convective coefficient. Unfortunately, this law returns values which are absurd for the K9KGEN5 turbo charger, so the model is not general enough.

The estimate of the h_huile has been based on M. Fénot's [12] work. He suggests several methods to calculate the convective coefficient between two coaxial cylinders, where the inner one is rotating. The Aoki et al. method has been chosen for simplicity.

7. Efficiencies

The efficiencies observed to validate the model include:

- The global mechanical efficiency:

$$\eta_{m\acute{e}ca} = \frac{P_{arbre\_C}}{P_{arbre\_T}} \tag{7.1}$$

- The total isentropic turbine efficiency:

$$\eta_T = \frac{P_{arbre\_T}}{P_{is\_T}} \tag{7.2}$$

- And the total isentropic compressor efficiency:

$$\eta_C = \frac{P_{is\_C}}{P_{arbre\_C}} \tag{7.3}$$

8. Results – Model Validation

On the bibliography, some examples of expected mechanical (Figure 24), compressor (Figure 26) and turbine (Figure 25) efficiencies are found and may be used as reference.

Figure 26 is particularly interesting, as it shows the correction that is intended by this model. Nevertheless, the results obtained will be presented as a function of the compressor rate.

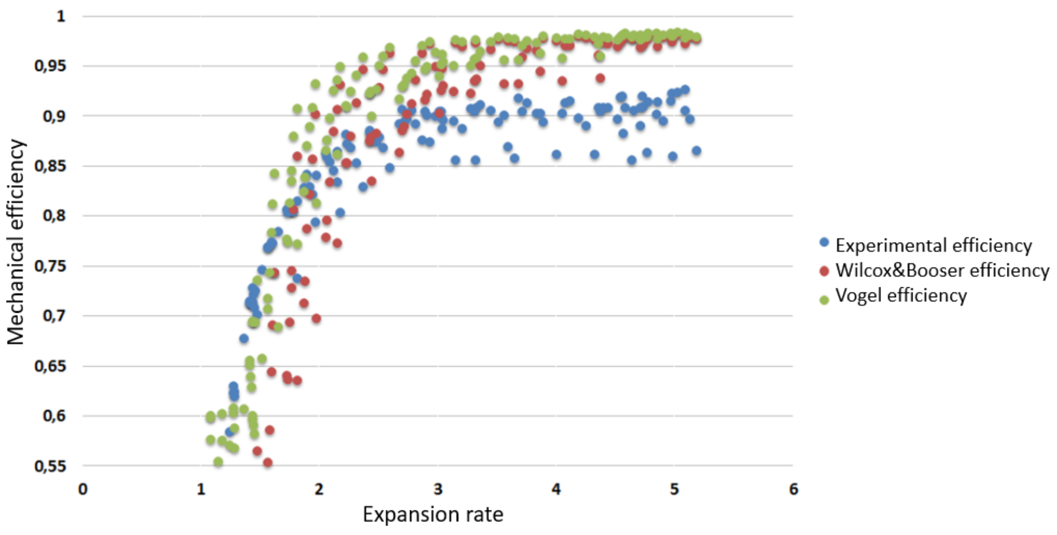


Figure 24: expected mechanical efficiency of the turbo charger. [13]

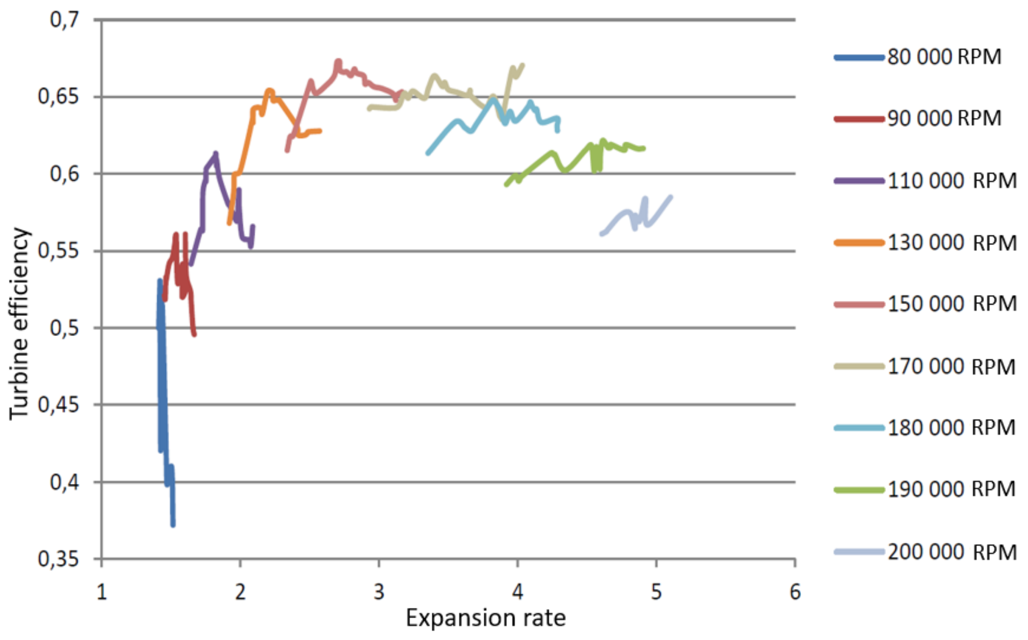


Figure 25: expected turbine efficiencies. [1]

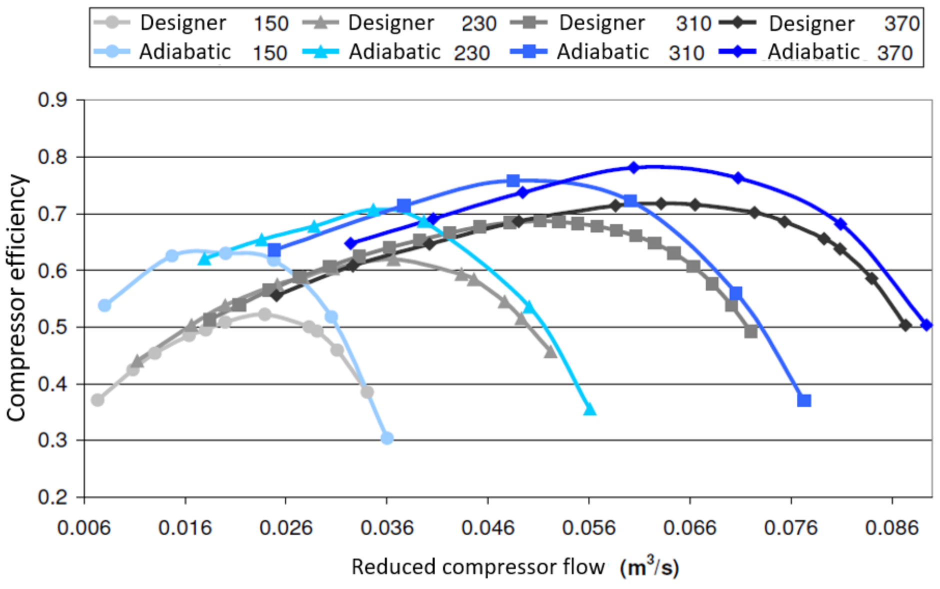


Figure 26: expected compressor efficiencies. [2]

The approach is then to observe the efficiency curves obtained from cold tests (adiabatic) to have an estimation of the minimum and maximum values of efficiency attainable. As the non-adiabatic efficiency curves follow the same profile, the cold test curves are used as a parameter.

The model was developed in Python language under the Pyzo platform. Using the « numpy », « xlrd », « matloblib » and « os » libraries, the compressor and turbine efficiencies were plotted by rotation range. For the compressor, these efficiencies are expressed as a function of the compression rate. As for the turbine, as a function of the expansion rate. The code will be available on the section « Appendix: Python Code ».

The results obtained from the original model as it is set are synthesized on Figures 27 to 30. All exchanges are expressed in Watt.

8.1 Adiabatic Case

The adiabatic case will serve as a reference, as it is a cold test. Thus, there is theoretically no heat transfer to be considered. Nevertheless, this case represents a limit of usage which leads to aberrant measures at some points.

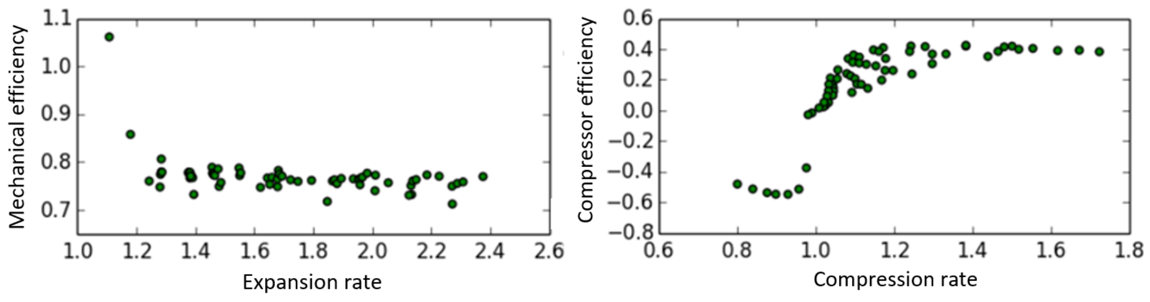


Figure 27: mechanical / compressor efficiencies for the adiabatic case.

The mechanical efficiency (Figure 27 – left) is globally stable around 0.75. This is explained by a week rotation range (6000 to 15000 rpm). With the low power on the axis, a superior mechanical efficiency is expected for cases where the inlet turbine temperature is higher.

The compressor efficiency (Figure 27 – right) is not exploitable, given that the axis power is too weak. The communicated heat transfers are logically rather weak. This model seems coherent with these data points.

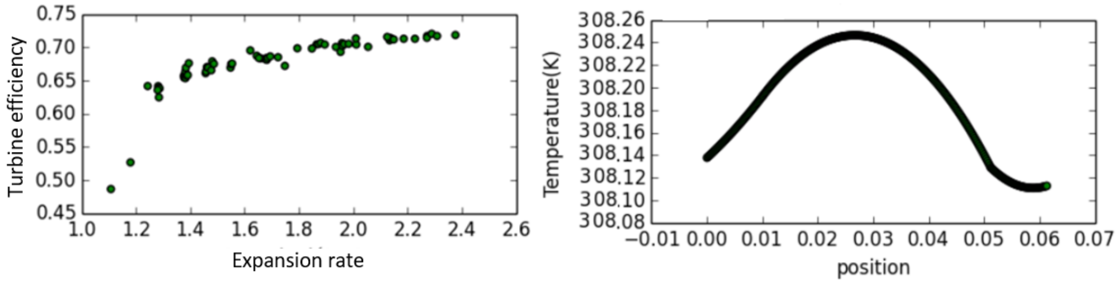


Figure 28: turbine efficiency and temperature evolution function (last data input) for the adiabatic case.

The turbine efficiency (Figure 28 – left) resembles the mechanical efficiency profile curve, increasing up to 0.70 approximately. The temperature evolution profile (Figure 28 – right) is an evidence of the adiabatic functioning, as the temperature remains practically constant along the turbo charger.

8.2 Inlet Turbine temperature at 300°C and 580°C

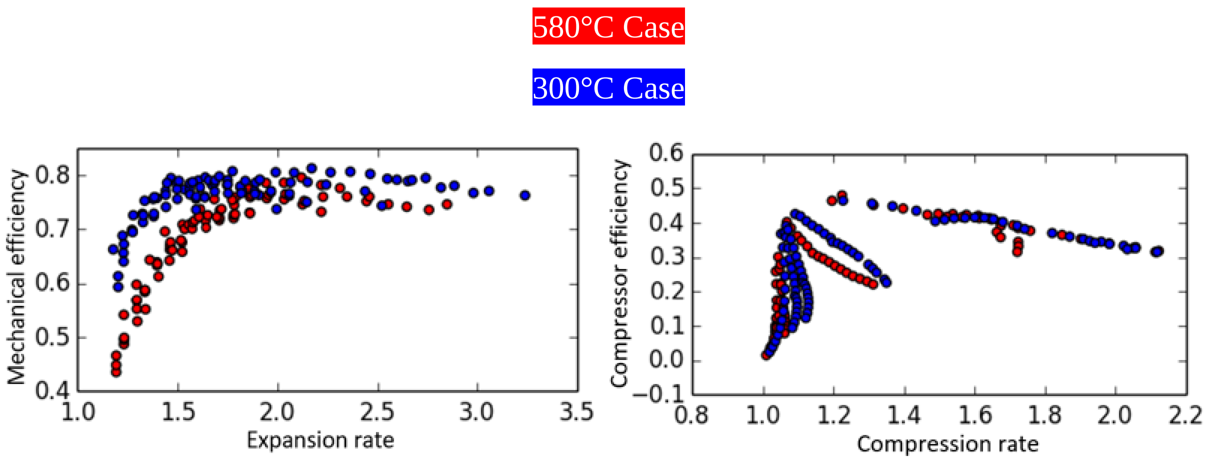


Figure 29: mechanical / compressor efficiencies for the inlet turbine temperatures at 300°C and 580°C.

The mechanical efficiency profile curve (Figure 29 – left) is as expected, with a maximum reached at high expansion rate. On the other hand, the part relative to the friction being weaker for a comparatively lower mechanical power on the axis, a higher efficiency would be expected compared to the adiabatic case, which is not the case here. Therefore, some heat exchanges must have been underestimated.

The compressor efficiency (Figure 29 – right) is not satisfying, as it presents values too low compared to those expected.

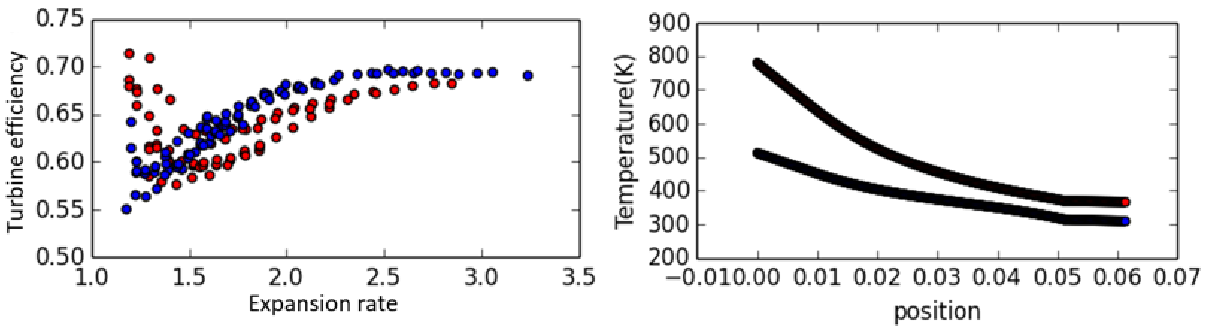


Figure 30: turbine efficiency / temperature evolution function for the inlet turbine temperatures at 300°C and 580°C.

The turbine efficiency (Figure 30 – left) is largely underestimated at low rpm, where the transfers are the most important. In fact, the exchanges with the neighboring parts to the turbine should be considered to correct this efficiency. Too much mechanical power has been attributed to the axis, while this power is dissipated as heat to the neighboring parts. The correction (in red) given by the following relation is not explored well enough:

$$P_{arbre\_T} = Dm_T.\ Cp_T.\ \Delta T_T - Q_{T\_PT} - Q_{ext\_T} \quad (48)$$

To provide coherent efficiency values, the temperature evolution profile (Figure 30 – right) should have been steeper near the turbine and flat near the compressor at a higher end-of-plateau temperature, so more heat would have been transferred from the turbine to the compressor and the efficiencies would be greater.

9. Correction of the convection coefficients using an artificial neural network

Despite the existence of several python libraries and functions to train and apply neural networks, this project has required the implementation of the basic mechanisms of machine learning. That is because the network structure is quite particular to this application, and the encrypted TensorFlow and Keras libraries do not allow for such customization.

The adopted approach was then to build an MLP ANN [24] to target the adiabatic efficiency values calculated based on the very entries. By trying to find a weight matrix which approaches theses efficiency values, the efficiencies are guaranteed to have the correct profile. Nevertheless, it is known that the network will and should have a residual error, which represents the gap between the adiabatic and real operating conditions of the turbo charger.

The architecture takes as input the experimental data obtained for the device. Then, a technique called Principal Component Analysis (PCA) [23] determines the most determinant parameters to be considered as inputs. It does so by analyzing the variance contribution of each input attribute and neglecting the ones with the least impact. The algorithm involves following the steps consecutively:

a) Normalize the input by column with variance equals to one and centered on zero.
b) Calculate the covariance matrix of the normalized input
c) Calculate the eigenvalues of the covariance matrix, which are guaranteed to represent the variance of each input attribute in descending order.
d) Neglect the input parameters with the least variance impact according to a pre-defined criterion.

By considering fewer input parameters, the network has a higher chance of converging more quickly.

The MLP architecture then takes as many input parameters as the PCA technique has determined. The hidden layer is fixed with only four neurons to speed up training. The output layer will contain the convection coefficients in a pre-determined order, which will be activated by a function called ‘turbo’ on the Python code. This function is the physical model which calculates the heat transfers and the resulting efficiencies, allowing the network to compare them to desired ones. It takes the most consistent theoretical model, the one which calculates the oil convective heat, as a reference and the ratios between correlation-calculated coefficients as a starting training point. As it may be seen on the results, the training happens rather quickly due to this appropriate starting position.

All parameters related to the neural network are written in an Excel worksheet, and may be changed at any time before training. The Python script has been converted to an Executable (.exe) file using Pyinstaller, so to be launched it just needs a double click on the icon. This strategy allows for a more friendly-user interface, as all inputs are taken from Excel files and the script is run on the operational system’s command prompt.

The MLP parameters are shown on Table 8. They may be edited on the Excel sheet named “MLP_Parameters.xlsx” at any time before training is launched.

Update Parameter ratio (alph)	0.99
Initial Momentum Parameter (alpha)	0.3
Initial Learning rate (eta)	0.3
Minimum Learning rate (min_eta)	0.00001
Minimum Momentum Parameter (min_alpha)	0.00001
Training Data Proportion (proporcao_dados)	0.75
Maximum number of epochs (max_cycles)	75
Learning stopping tolerance	0.01
Number of neurons in the hidden layer	4
Number of instances on the 300°C case	51
Number os instances on the 580°C case	43
Number of attributes	16

Table 8: parameters used for training the MLP network.

Given the very distant case scenarios, separate trainings have been performed for the 300°C and the 580°C inlet turbine temperatures. If the experimental data contained several turbine inlet temperatures, the network could have been trained only once and generalized for all operating points.

9.1 Training Results

The network has been run on a 64-bit Windows 10 laptop with 8 Gb of RAM and an Intel Core i5-7200U CPU 2.50 GHz processor, performing at 1.053 s/epoch. Figures 32, 33 and 34 show the best results obtained from training.

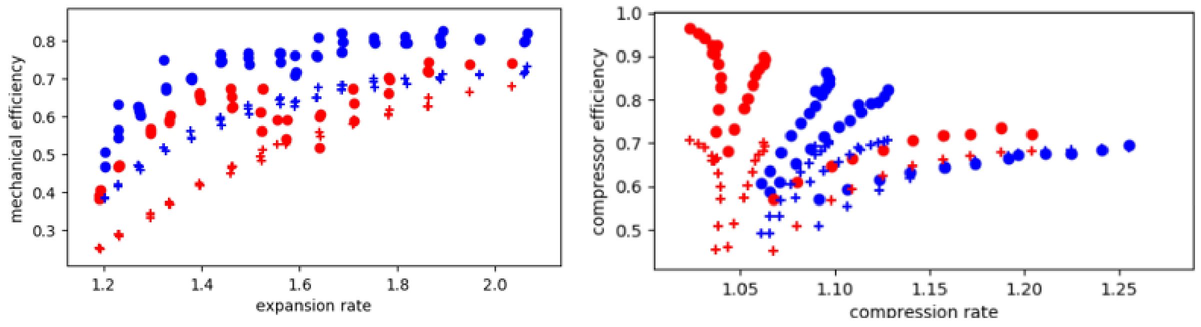


Figure 31: mechanical / compressor efficiencies for the inlet turbine temperatures at 300°C (blue) and 580°C (red). The curves marked in with '+' signs correspond to the target values for the efficiencies, whereas the dotted ones are the predicted values given by the neural network.

The mechanical efficiency (Figure 31 – left) approaches the target results considerably from above (average 0.1 higher than isentropic efficiencies). This is coherent with reality, as the turbine will dissipate more heat and the compressor will receive more heat in reality than in the isentropic case, due to heat transfer.

The same applies to the compressor efficiency (Figure 31 – right). The neural network manages to adapt to the curve profiles, but making efficiencies a little higher than isentropic ones, therefore closer to real operational situations.

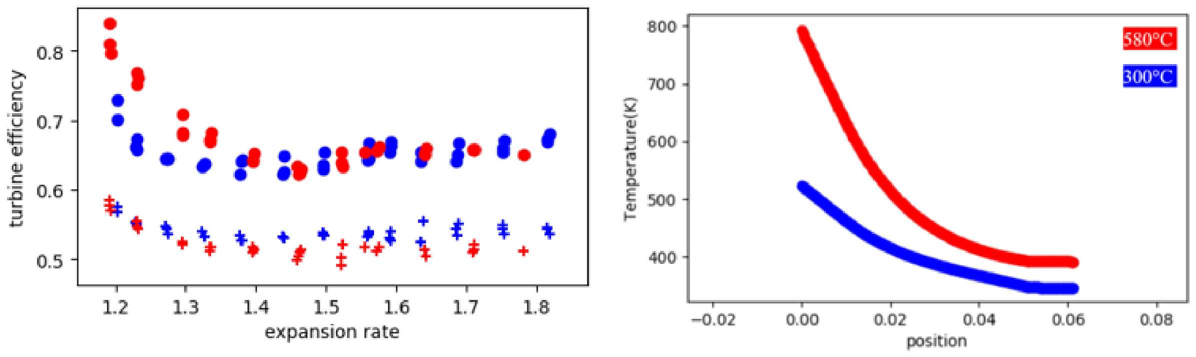


Figure 32: turbine efficiency / temperature evolution function for the inlet turbine temperatures at 300°C (blue) and 580°C (red). The curves marked in with '+' signs correspond to the target values for the efficiencies, whereas the dotted ones are the predicted values given by the neural network.

The turbine efficiency (Figure 32 – left) is responsible for the largest gap with the target efficiencies, which is also coherent, as the turbine is where most of the heat will be dissipated.

The temperature profile curves (Figure 32 – right) are as expected, being steeper near the turbine and flattening out near the compressor.

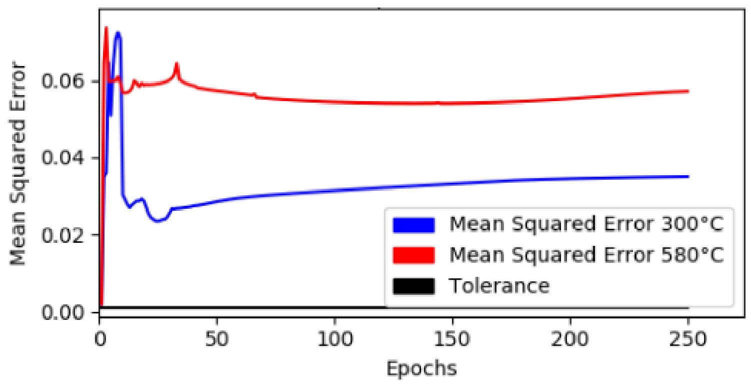


Figure 33: learning curves for the inlet turbine temperatures at 300°C (blue) and 580°C (red).

Figure 33 shows how the mean squared error may start very high and be unstable at the beginning of training but eventually the network converges to a limit value. After 250 epochs, the network had not converged entirely yet, but it shows a tendency to converge in the next few epochs, as the learning rate would be near its minimum value.

The errors obtained for these parameters resemble some of those presented by O. Özener et al. [14] in his work on diesel engines (section “3.2 Applications of Artificial Neural Networks (ANNs) in Mechanical Engineering”). He obtained mean squared errors of 0.085 for NO_x and 0.027 for power in his training.

	RPM	h_T_PT	h_ext_T	h_ext_PT	h_C_PC	h_ext_C	h_ext_cc	h_huile
300°C	87199	226 (430)	52 (6)	10 (6)	274 (269)	39 (3)	25 (5)	4356
580°C	112197	232(544)	69 (7)	29 (7)	255 (331)	9 (4)	10 (5)	4941

Table 8: rounded-up heat convection coefficients approximated by a neural network trained upon the input data from experimental tests on a turbo charger at 300°C and 580°C inlet turbine temperatures. The values in parenthesis are correlation-calculated heat coefficients.

Table 8 shows the calculated coefficients for an rpm taken at random from the dataset. It makes clear that the correlations overestimate the forced convections from the turbine and to the compressor, whereas the natural convections to the outside are underestimated. This could have been one of the sources of error in the original model.

The neural network has an overall satisfying performance with little running time on a simple personal computer, which indicates the interest for its application.

10. Distribution of Heat Transfer along the Turbo charger

The following charts allow for a better understanding of the phenomena taking place in the turbo charger modelled during this project. As expected, at high rpm, the gap between adiabatic and corrected power is less important than at low rpm.

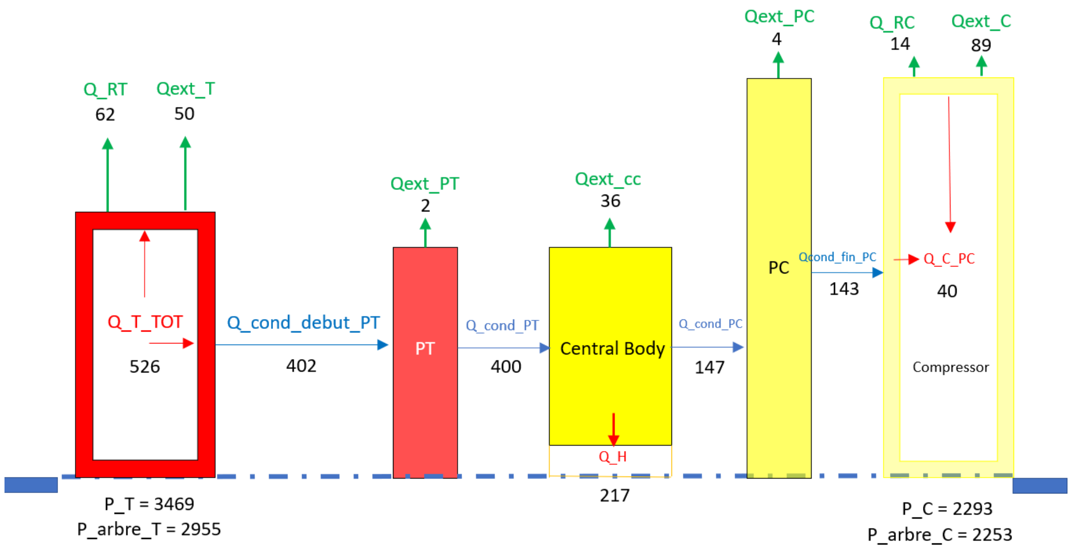


Figure 34: Heat Transfer distribution (in Watt) at 151236 rpm for the inlet turbine temperature at 300°C on the K9KGEN5 turbo charger.

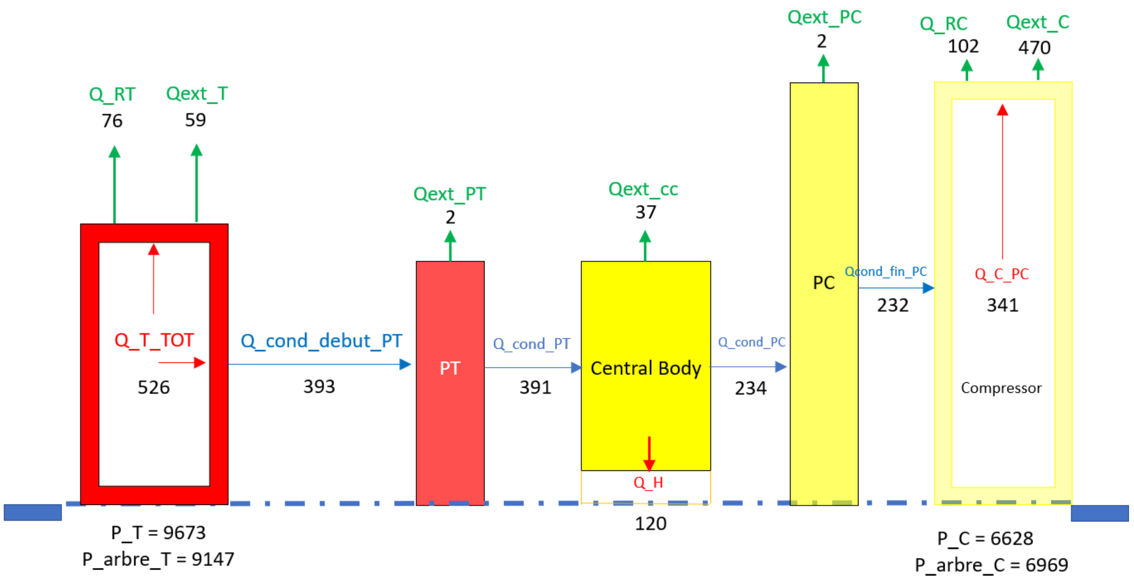


Figure 35: Heat Transfer distribution (in Watt) at 240670 rpm for the inlet turbine temperature at 300°C on the K9KGEN5 turbo charger.

Some reserve must be applied when analyzing the values of the radiated and convective power on the compressor, as these values become higher than those at the turbine at high rpm (Figure

35). This may be explained by the imprecisions on the determination of surfaces and by the volute wall temperature measurement, from which the experimental data points' locations are not specified.

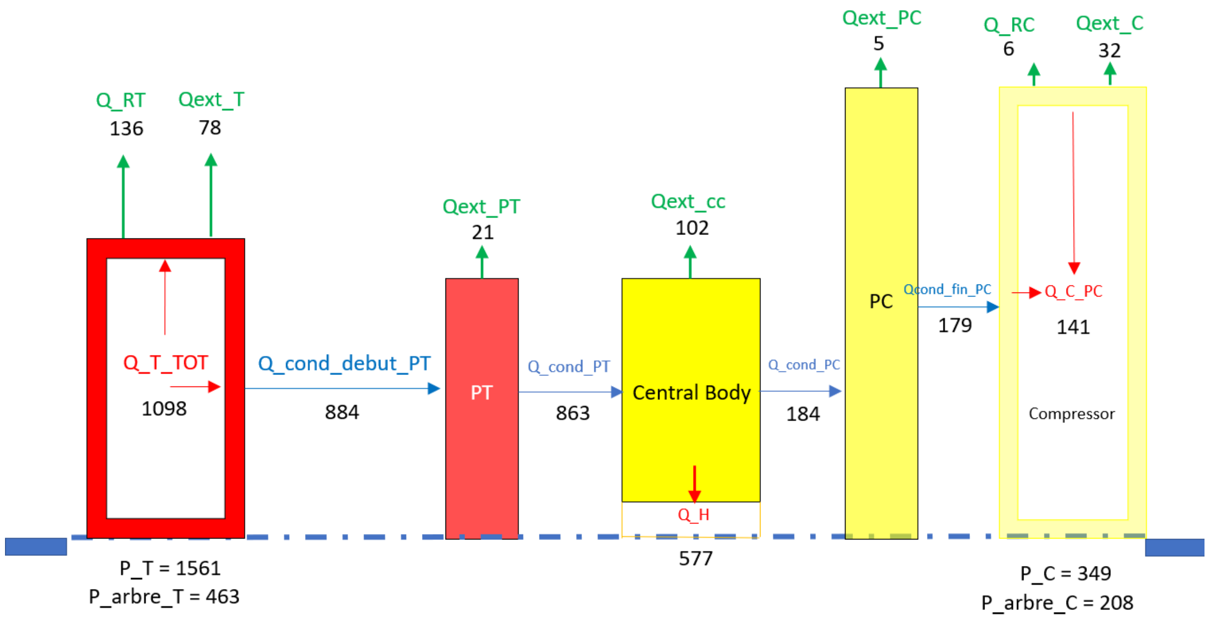


Figure 36: Heat Transfer distribution (in Watt) at 240670 rpm for the inlet turbine temperature at 300°C on the K9KGEN5 turbo charger.

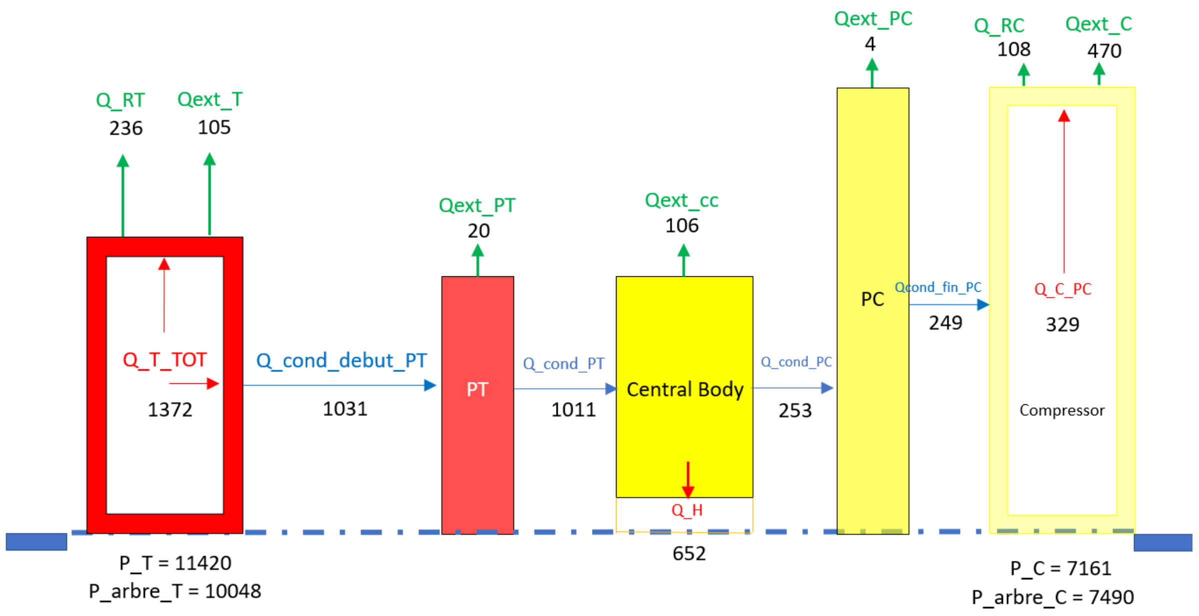


Figure 37: Heat Transfer distribution (in Watt) at 245023 rpm for the inlet turbine temperature at 580°C on the K9KGEN5 turbo charger.

It is also notable that the heat flow towards the mean compressor flow may be either negative (Figure 36) or positive (Figure 37). It is thus difficult to assure whether the model is faulty at this aspect, as it is also possible that the gas flow gets heated up and then dissipated heat back to the compressor walls. Nevertheless, it remains surprising that it evacuates heat even for the 580°C case.

Only a part of the turbine radiation is considered to heat up the compressor. Without experimental means, it has not been possible to add this aspect to the transfers on this model. It would have been interesting to note its effect regarding the conclusion taken from this project.

11. Conclusion and Future Work

For such a complex problem, it is best to obtain a simple model for some specific operating points rather than a very complex model, which need too many details over its parameters. The correlation model being rather complex and specific on the one side, the neural network simplifies the work by applying a regression of the input data to the desired outputs.

The presented model, although simple and zero-dimensional, is successful at correcting the efficiency curves for the non-adiabatic cases. An improvement may be made in the experimental database, which could contain other inlet turbine temperatures and a larger volume of data points. Unfortunately, due to financial and temporal resources, no additional experimental tests could have been performed for the present project, which would have enabled the recognition of further patterns for the calculated powers.

The complex geometry, the complex physical phenomena and the experimental uncertainty have challenged this model's accuracy, but the neural network seems to have elegantly overcome these obstacles, being paralleled with the current state-of-the-art.

Finally, this model, although limited in its results and precision, constitutes a good first approach to evaluating heat transfer in a turbo charger. Further advancements may be achieved by adding more experimental data to the network input.

With enough computational resources and an adaptation to the script, the network may be run in real-time for performance tuning applications.

REFERENCES

[1] G. Salameh, "Caractérisation expérimentale d'une turbine de suralimentation automobile et modélisation de ses courbes caractéristiques de fonctionnement", PhD Thesis, Ecole Centrale de Nantes, 2016.

[2] M. Cormerais, "Caractérisation expérimentale et modélisation des transferts thermiques au sein d'un turbocompresseur automobile. Application à la simulation du comportement transitoire d'un moteur Diesel à forte puissance spécifique", PhD Thesis, Ecole Centrale de Nantes, 2007.

[3] Romagnoli, A. et Martinez-Botas, R., "Heat transfer analysis in a turbo charger turbine: an experimental and computational evaluation", Imperial College London, 2010.

[4] A. Rattner, J. Bohren, "Heat and Mass Correlations", November 2016.

[5] Website, Wikipedia: Air, https://fr.wikipedia.org/wiki/Air

[6] M. Cormerais, J.F. Hetet, P. Chessé, A. Maiboom, 2006, "Heat Transfers Characterizations in a Turbo charger: Experiments and Correlations", ASME Internal Combustion Engine Division 2006 Spring Technical Conference, May 8-10, 2006, Aachen, Germany.

[7] D. Bohn, T. Heuer, K. Kusterer, 2005, "Conjugate Flow and Heat Transfer Investigation of a Turbo Charger", Journal of Engineering for Gas Turbines and Power, ASME, July 2005, Vol. 27, Aachen, Germany.

[8] S. Shaaban, 2004, "Experimental investigation and extended simulation of turbo charger non-adiabatic performance", Hannover University, 14.12.2004, Hannover, Germany.

[9] P. Podevin, M. Toussaint, G. Richard, G. Farinole, "Performances of Turbo charger at Low Speed", 2002, Conservatoire National des Arts et Métiers, Cielplne Maszyny Przeplyowe.

[10] G. Tanda, S. Marelli, G. Marmorato, M. Capobianco, "An experimental investigation of internal heat transfer in an automotive turbo charger compressor", DIME, University of Genoa, Italy.

[11] A. Romagnoli, R. Martinez-Botas, "Heat Transfer Analysis in a Turbo charger Turbine: An Experimental and Computational Evaluation", Imperial College London, London, United Kingdom.

[12] M. Fénot, Y. Bertin, E. Dorignac, G. Lalizel, "International Journal of Thermal Science", ENSMA – Université de Poitiers, Elsevier, February 2011.

[13] H. Belkhou, L.A. Carbajal Carrasco, P. Rengade, "Caractérisation et modélisation des frottements mécaniques dans un turbocompresseur", March 2016, Ecole Centrale de Nantes.

[14] O. Özener, L. Yüksek, M. Özkan, "Artificial Neural Network Approach to Prediction engine-out emissions and performance parameters of a turbo charged diesel engine", Yildiz Technical University, Istanbul, Turkey, 2013.

[15] Haykin, S., "Neural Networks: A Comprehensive Foundation", Prentice Hall, 1998.

[16] Karonis, D., Lois, E., Zannikos, F., Alexandridis, A., Sarimveis, H., "A Neural Network Approach for the Correlation of Exhaust Emissions from a Diesel Engine with Diesel Fuel Properties", Energy & Fuels, 17 (2003) pp. 1259-1265.

[17] Öztemel, E., "Artificial Neural Networks", Papatya Publishing, 2003.

[18] Beale, H.M., Hagan, T.M., Demuth, H.B., "Matlab Neural Network Toolbox User Guide", in, The MathWorks Inc., 2010.

[19] Oğuz, H., Sarıtas, I., Baydan, H.E., "Prediction of diesel engine performance using biofuels with artificial neural network, Expert Systems with Applications", 37 (2010) pp. 6579-6586.

[20] He, Y., Rutland, C.J., Application of artificial neural networks in engine modelling, International Journal of Engine Research, 5 (2004) pp. 281-296.

[21] Eagle Ridge, Chevrolet Buick GMC, https://www.eagleridgegm.com/what-is-a-turbo charger-and-how-does-it-work/, access on 22/11/2018.

[22] D. O. Hebb, "The Organization of Behavior: A Neurophysiological Theory", 1949.

[23] R.A.F. Romero, "Aula 12 : PCA e Rede PCA", SCC-ICMC-USP,2017.

[24] R.A.F. Romero, "Aula 3 : Multi-Layer Perceptron (MLP)", SCC-ICMC-USP,2018.

[25] Kalogirou AS. "Applications of artificial neural networks in energy systems", Energy Convers Manage 1999;40:1073-87.

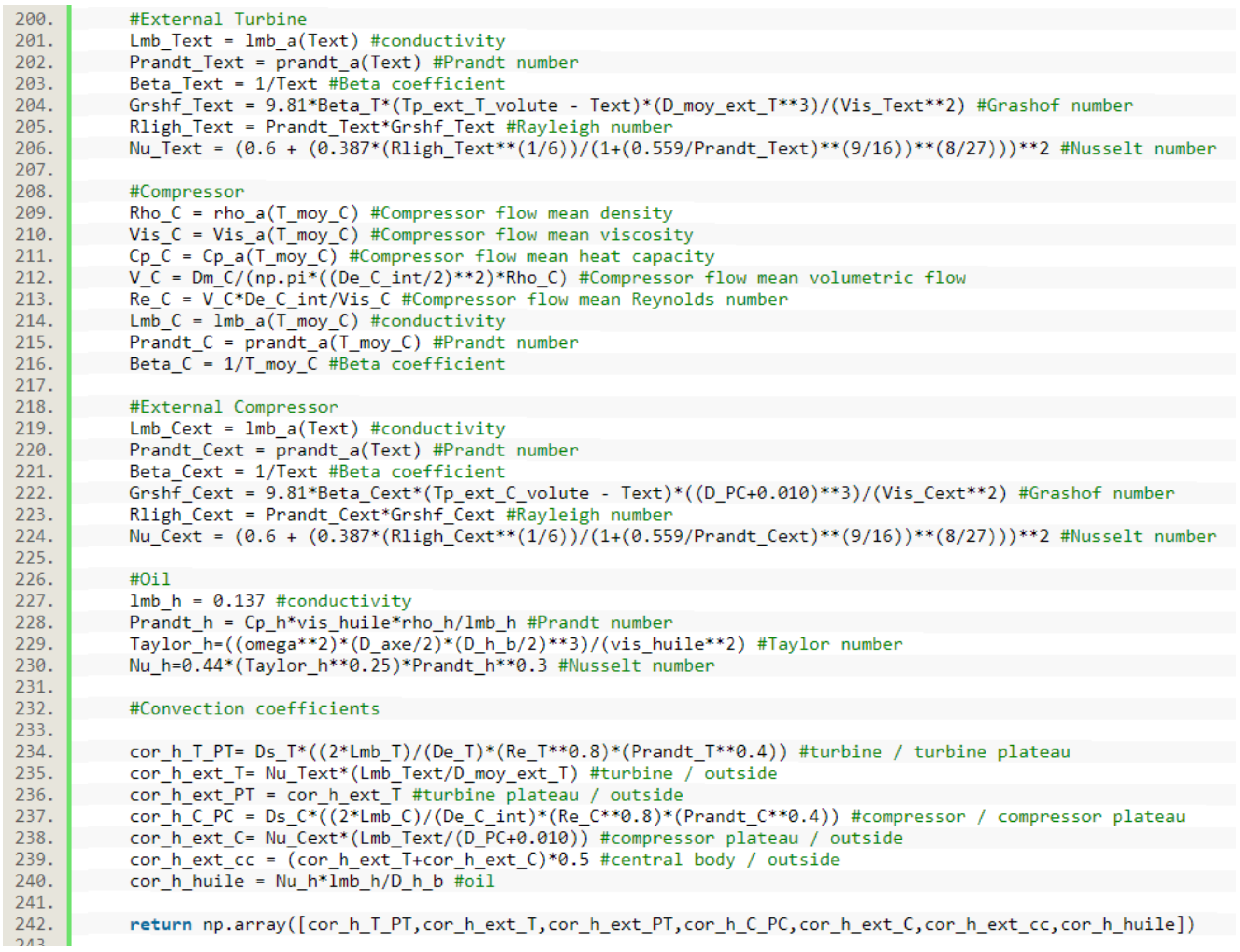
[26] Automotive Articles, http://www.automotivearticles.com/printer_Turbo_Selection.shtml.

APPENDIX: PYTHON CODE

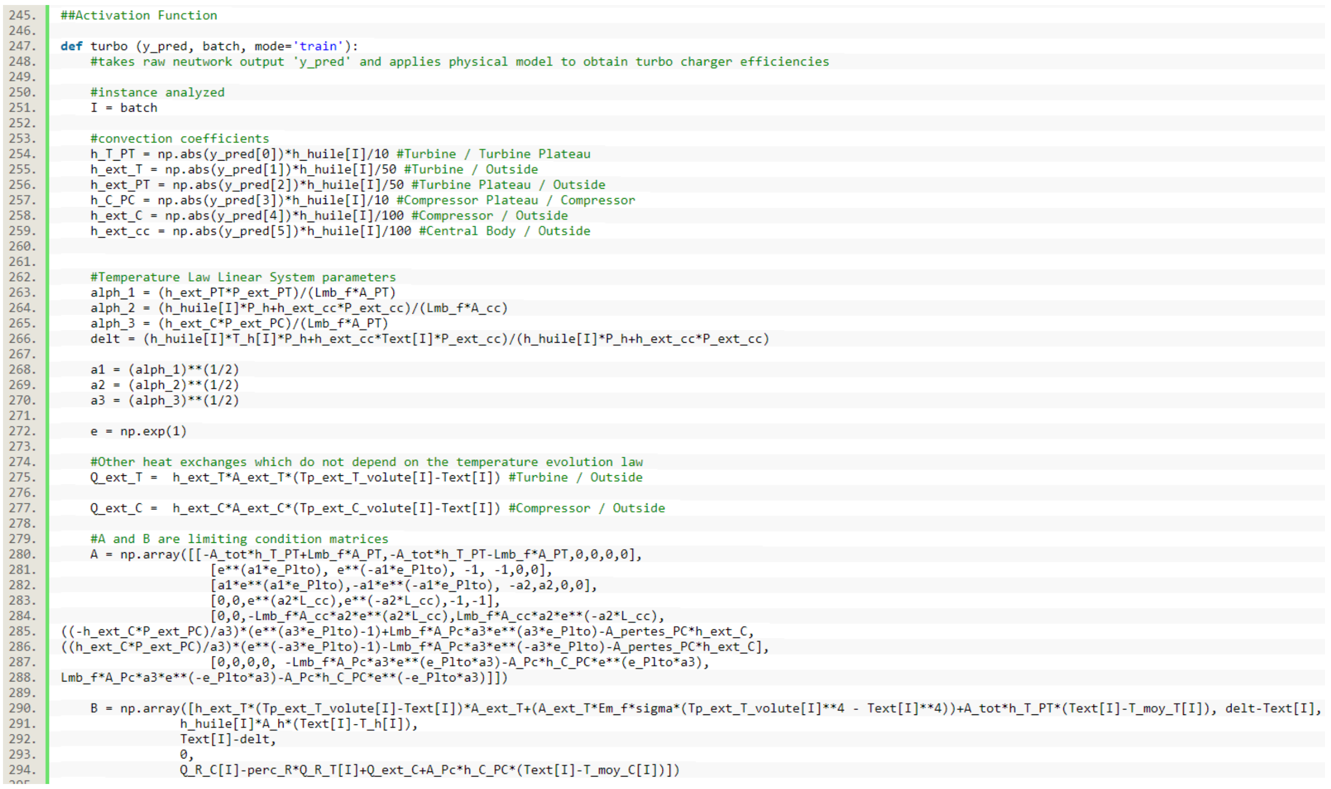
```python
##Application of Artificial Neural Networks to the Determination of Heat Exchange in a Turbocharger
"""
Authors: Matheus Raposo de Almeida (nº USP: 8956050)
         Léo Le Roy

Travail de Fin d'Etudes/ Trabalho de Conclusão de Curso/Undergraduate Thesis

Prof. Dr. Luben Cabezas Gómez
Prof. Dr. David Chalet
Prof. Dr. Georges Salameh

"""

import numpy as np #calculations
import random as rd #weight initialization
import matplotlib.pyplot as plt #plots
import matplotlib.patches as mpatches #plot content
import time #counting the time
import xlrd #open sheet files
import os #opening files

start_time = time.time()

## ---> Dimensions of the turbo charger

def open_xlrd(f):
    #opens an Excel file and returns the first sheet
    workbook = xlrd.open_workbook(f)
    sheet = workbook.sheet_by_index(0)
    return sheet

#opens the Excel file containing the turbo charger geometrical data
geometry_sheet = open_xlrd('Geometrical_Data.xlsx')

#Turbine
De_T = geometry_sheet.cell_value(1,1)     #inlet turbine volute diameter
D_PT = geometry_sheet.cell_value(2,1) #plateau external diameter
Le_T = geometry_sheet.cell_value(3,1) #inlet turbine volute length
Ls_T = geometry_sheet.cell_value(4,1) #outlet turbine volute length
D_moy_ext_T = geometry_sheet.cell_value(5,1) #mean external turbine diameter
e_Plto = geometry_sheet.cell_value(6,1) #plateau thickness
Ds_T= geometry_sheet.cell_value(7,1) #outlet section diameter

#Central Body
L_cc=geometry_sheet.cell_value(9,1) #length
D_axe=geometry_sheet.cell_value(10,1) #central axis diameter
D_cc=geometry_sheet.cell_value(11,1) #central body mean diameter
D_h=geometry_sheet.cell_value(12,1) #mean oil flow diameter
```

```
#Compressor
D_PC = geometry_sheet.cell_value(14,1) #compressor plateau diameter
Ls_C = geometry_sheet.cell_value(15,1) #compressor outlet length
Le_C= geometry_sheet.cell_value(16,1) #compressor inlet length
Ds_C=geometry_sheet.cell_value(17,1) #compressor outlet diameter
De_C = geometry_sheet.cell_value(18,1) #compressor inlet diameter
De_C_int = geometry_sheet.cell_value(19,1) #compressor internal inlet diameter

P_ext_PT = np.pi*D_PT #external turbine plateau perimeter
A_ext_T=np.pi*(D_moy_ext_T*Ls_T+De_T*Le_T) #external turbine volute surface

A_PT = np.pi*(D_PT*0.5)**2 #turbine plateau cross-section

A_tot= (A_ext_T+A_PT) #total heat transfer turbine surface

D_h_b = D_h-D_axe #effective oil flow diameter

P_ext_cc=np.pi*D_cc #external central body perimeter
P_h = np.pi*D_h #mean oil flow perimeter
A_h=np.pi*((D_h*0.5)**2) #mean oil flow surface
A_cc=A_PT-A_h #central body cross-section

P_ext_PC=np.pi*D_PC #external compressor plateau perimeter
A_Pc=np.pi*((D_PC*0.5)**2) #compressor plateau surface
A_ext_C= (0.05776+np.pi*De_C*Le_C+Ls_C*np.pi*Ds_C)*3 #external compressor surface
A_pertes_PC = A_Pc - A_PT #compressor plateau heat loss surface

#inlet and outlet turbocharger cross-sections
section_e_c = np.pi/4*(De_C_int**2) #inlet compressor cross-section
section_s_c = np.pi/4*(Ds_C**2) #outlet compressor cross-section
section_e_t = np.pi/4*(De_T**2) #inlet turbine cross-section
section_s_t = np.pi/4*(Ds_T**2) #outlet turbine cross-section


##Fluid and Material Physical Properties


Rec_f = geometry_sheet.cell_value(21,1) #cast iron receptivity
Em_f = geometry_sheet.cell_value(22,1) #cast iron emissivity
Em_a = geometry_sheet.cell_value(23,1) #aluminum emissivity
Lmb_f = geometry_sheet.cell_value(24,1) #cast iron conductivity
vis_huile = geometry_sheet.cell_value(25,1) #oil viscosity
rho_h = geometry_sheet.cell_value(26,1) #oil density
Cp_h = geometry_sheet.cell_value(27,1) #oil heat capacity

sigma = 5.67037*10**-8 #radiative heat transfer constant (Boltzmann)

#reference parameters
Tref = 288.33 #reference temperature
Pref = 1013.207 #reference pressure in mB
```

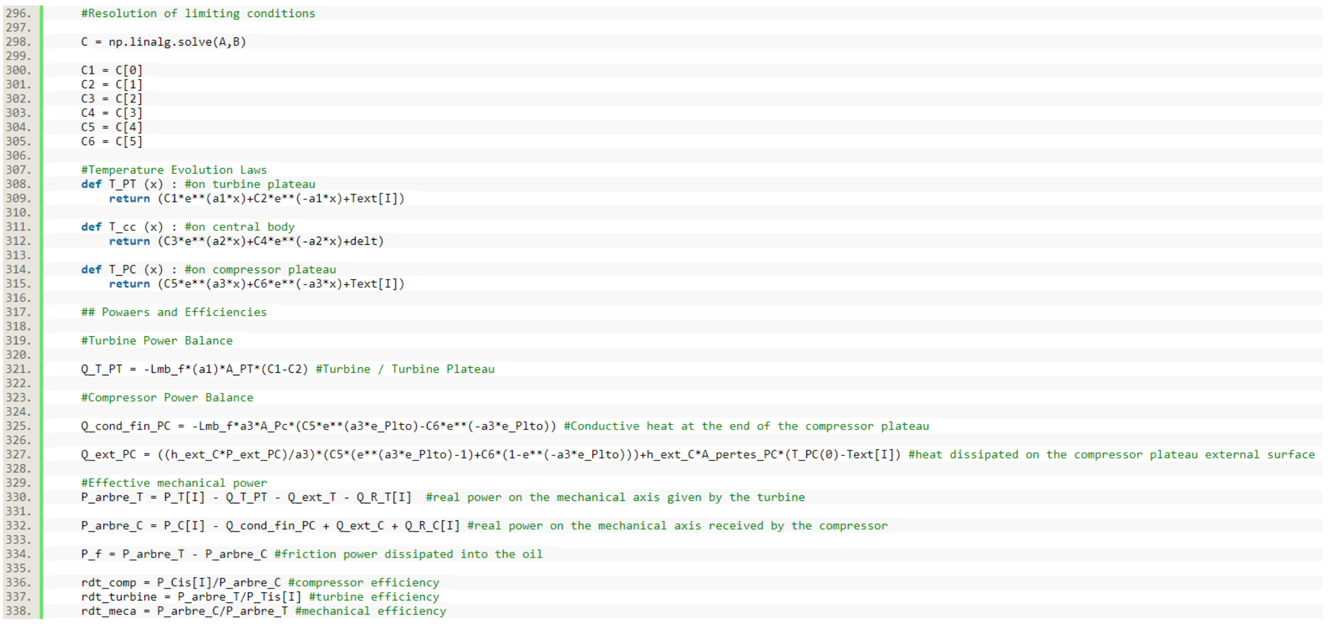
```
#Percentage of loss due to radiation
perc_R = 0.2

## ---> Defining physical functions

## Physical laws polynomial approximation from tables (as a function of temperature in K)
def rho_a (T) :  #density
    return(-2*(10**-9)*T**3 + 6*(10**-6)*T**2 - 0.0063*T + 2.5695)

def Cp_a (T) :   #heat capacity
    return (-2*(10**-7)*T**3 + 6*(10**-4)*T**2 - 0.2135*T + 1024.5)

def Vis_a (T) :  #viscosity
    return (-1*(10**-14)*T**3 +9 *(10**-11)*T**2 + 4*(10**-8)*T -7*10**-6)

def independent_powers():
    #calculates the heat exchanges and powers which are independant from the temperature evolution law

    # --> Flow Physical Characteristics

    #Turbine
    Cp_T = Cp_a(T_moy_T) #Turbine flow mean heat capacity

    #Preliminary oil convection coefficient
    v_h = Dm_h/(rho_h*np.pi*0.25*(D_h**2 - D_axe**2)) #oil volumetric flow
    Re_h = v_h*D_h_b/vis_huile #oil Reynolds number

    #Compressor
    Cp_C = Cp_a(T_moy_C) #Compressor flow mean heat capacity

    #Expansion and Compression rates
    taux_de_comp_c = Ps_C/Pe_C #compressor compression rate
    taux_de_detente_t = Pe_T/Ps_T #turbine expansion rate

    Rho_C_e = rho_a ( Te_C ) #inlet compressor flow density
    Rho_C_s = rho_a ( Ts_C ) #outlet compressor flow density
    Rho_T_e = rho_a ( Te_T ) #inlet turbine flow density
    Rho_T_s = rho_a ( Ts_T ) #outlet turbine flow density

    Cp_C_e = Cp_a(Te_C) #inlet compressor flow heat capacity
    Cp_C_s = Cp_a(Ts_C) #outlet compressor flow heat capacity
    Cp_T_e = Cp_a(Te_T) #inlet turbine flow heat capacity
    Cp_T_s = Cp_a(Ts_T) #outlet turbine flow heat capacity

    #Stop parameters
    Te_Ci = Te_C + ((Dm_C/Rho_C_e/section_e_c)**2)/2/Cp_C_e #inlet compressor flow stop temperature
    Ts_Ci = Ts_C + ((Dm_C/Rho_C_s/section_s_c)**2)/2/Cp_C_s #outlet compressor flow stop temperature
    Te_Ti = Te_T + ((Dm_T/Rho_T_e/section_e_t)**2)/2/Cp_T_e #inlet turbine flow stop temperature
    Ts_Ti = Ts_T + ((Dm_T/Rho_T_s/section_s_t)**2)/2/Cp_T_s #outlet turbine flow stop temperature
    Ts_Ci_is = Te_Ci * (taux_de_comp_c**(0.4/1.4)) #outlet compressor flow isentropic stop temperature
    Ts_Ti_is = Te_Ti * (taux_de_detente_t**(-0.4/1.4)) #outlet turbine flow isentropic stop temperature

    #Heat exchanges which do not need the temperature evolution law along the turbo charger
    P_T = Dm_T*Cp_T*(Te_Ti-Ts_Ti) #turbine power
    P_Tis = Dm_T*Cp_T*(Te_Ti-Ts_Ti_is) #turbine isentropic power
    Q_R_T = Em_f*sigma*A_ext_T*((Tp_ext_T_volute)**4 - Text**4) #turbine radiated heat

    P_C = Dm_C*Cp_C*(Ts_Ci - Te_Ci) #compressor power
    P_Cis = Dm_C*Cp_C*(Ts_Ci_is - Te_Ci) #compressor isentropic power
    Q_R_C = Em_a*sigma*A_ext_C*((Tp_ext_C_volute)**4 - Text**4) #compressor radiated heat

    return [P_T,P_Tis,Q_R_T,P_C,P_Cis,Q_R_C,taux_de_comp_c,taux_de_detente_t]

##Convection Coefficients through Correlations

def correlation():
    #calculates heat coefficients based on correlations

    u_axe = regime*np.pi*D_axe/60  #axis linear rotating speed
    omega = u_axe/D_axe #axis angular rotating speed

    #dimensionless flow coefficients

    def lmb_a (T) : #conductivity
        return(8*(10**-12)*T**3 - 4*(10**-8)*T**2 + (10**-4)*T + 5*10**-4)

    def prandt_a (T) : #Prandt Number
        return (-2*(10**-10)*T**3 + 6*(10**-7)*T**2 -5*(10**-4)*T + 0.8147)

    #Cabin air parameters
    Rho_Text = rho_a(Text) #Cabin air density
    Vis_Text = Vis_a(Text) #Cabin air viscosity
    Cp_Text = Cp_a(Text) #Cabin air heat capacity

    #Cabin air parameters
    Rho_Cext = rho_a(Text) #Cabin air density
    Vis_Cext = Vis_a(Text) #Cabin air viscosity
    Cp_Cext = Cp_a(Text) #Cabin air heat capacity

    #Turbine
    Rho_T = rho_a(T_moy_T) #Turbine flow mean density
    Vis_T = Vis_a(T_moy_T) #Turbine flow mean visosity
    V_T = Dm_T/(np.pi*((De_T/2)**2)*Rho_T) #Turbine flow mean volumetric flow
    Re_T = V_T*De_T/Vis_T #Turbine flow mean Reynolds number
    Lmb_T = lmb_a(T_moy_T) #conductivity
    Prandt_T = prandt_a(T_moy_T) #Prandt number
    Beta_T = 1/T_moy_T #Beta coefficient
```



```
    #External Turbine
    Lmb_Text = lmb_a(Text) #conductivity
    Prandt_Text = prandt_a(Text) #Prandt number
    Beta_Text = 1/Text #Beta coefficient
    Grshf_Text = 9.81*Beta_T*(Tp_ext_T_volute - Text)*(D_moy_ext_T**3)/(Vis_Text**2) #Grashof number
    Rligh_Text = Prandt_Text*Grshf_Text #Rayleigh number
    Nu_Text = (0.6 + (0.387*(Rligh_Text**(1/6))/(1+(0.559/Prandt_Text)**(9/16))**(8/27)))**2 #Nusselt number

    #Compressor
    Rho_C = rho_a(T_moy_C) #Compressor flow mean density
    Vis_C = Vis_a(T_moy_C) #Compressor flow mean viscosity
    Cp_C = Cp_a(T_moy_C) #Compressor flow mean heat capacity
    V_C = Dm_C/(np.pi*((De_C_int/2)**2)*Rho_C) #Compressor flow mean volumetric flow
    Re_C = V_C*De_C_int/Vis_C #Compressor flow mean Reynolds number
    Lmb_C = lmb_a(T_moy_C) #conductivity
    Prandt_C = prandt_a(T_moy_C) #Prandt number
    Beta_C = 1/T_moy_C #Beta coefficient

    #External Compressor
    Lmb_Cext = lmb_a(Text) #conductivity
    Prandt_Cext = prandt_a(Text) #Prandt number
    Beta_Cext = 1/Text #Beta coefficient
    Grshf_Cext = 9.81*Beta_Cext*(Tp_ext_C_volute - Text)*((D_PC+0.010)**3)/(Vis_Cext**2) #Grashof number
    Rligh_Cext = Prandt_Cext*Grshf_Cext #Rayleigh number
    Nu_Cext = (0.6 + (0.387*(Rligh_Cext**(1/6))/(1+(0.559/Prandt_Cext)**(9/16))**(8/27)))**2 #Nusselt number

    #Oil
    lmb_h = 0.137 #conductivity
    Prandt_h = Cp_h*vis_huile*rho_h/lmb_h #Prandt number
    Taylor_h=((omega**2)*(D_axe/2)*(D_h_b/2)**3)/(vis_huile**2) #Taylor number
    Nu_h=0.44*(Taylor_h**0.25)*Prandt_h**0.3 #Nusselt number

    #Convection coefficients

    cor_h_T_PT= Ds_T*((2*Lmb_T)/(De_T)*(Re_T**0.8)*(Prandt_T**0.4)) #turbine / turbine plateau
    cor_h_ext_T= Nu_Text*(Lmb_Text/D_moy_ext_T) #turbine / outside
    cor_h_ext_PT = cor_h_ext_T #turbine plateau / outside
    cor_h_C_PC = Ds_C*((2*Lmb_C)/(De_C_int)*(Re_C**0.8)*(Prandt_C**0.4)) #compressor / compressor plateau
    cor_h_ext_C= Nu_Cext*(Lmb_Text/(D_PC+0.010)) #compressor plateau / outside
    cor_h_ext_cc = (cor_h_ext_T+cor_h_ext_C)*0.5 #central body / outside
    cor_h_huile = Nu_h*lmb_h/D_h_b #oil

    return np.array([cor_h_T_PT,cor_h_ext_T,cor_h_ext_PT,cor_h_C_PC,cor_h_ext_C,cor_h_ext_cc,cor_h_huile])
```



```python
##Activation Function

def turbo (y_pred, batch, mode='train'):
    #takes raw neutwork output 'y_pred' and applies physical model to obtain turbo charger efficiencies

    #instance analyzed
    I = batch

    #convection coefficients
    h_T_PT = np.abs(y_pred[0])*h_huile[I]/10 #Turbine / Turbine Plateau
    h_ext_T = np.abs(y_pred[1])*h_huile[I]/50 #Turbine / Outside
    h_ext_PT = np.abs(y_pred[2])*h_huile[I]/50 #Turbine Plateau / Outside
    h_C_PC = np.abs(y_pred[3])*h_huile[I]/10 #Compressor Plateau / Compressor
    h_ext_C = np.abs(y_pred[4])*h_huile[I]/100 #Compressor / Outside
    h_ext_cc = np.abs(y_pred[5])*h_huile[I]/100 #Central Body / Outside


    #Temperature Law Linear System parameters
    alph_1 = (h_ext_PT*P_ext_PT)/(Lmb_f*A_PT)
    alph_2 = (h_huile[I]*P_h+h_ext_cc*P_ext_cc)/(Lmb_f*A_cc)
    alph_3 = (h_ext_C*P_ext_PC)/(Lmb_f*A_PT)
    delt = (h_huile[I]*T_h[I]*P_h+h_ext_cc*Text[I]*P_ext_cc)/(h_huile[I]*P_h+h_ext_cc*P_ext_cc)

    a1 = (alph_1)**(1/2)
    a2 = (alph_2)**(1/2)
    a3 = (alph_3)**(1/2)

    e = np.exp(1)

    #Other heat exchanges which do not depend on the temperature evolution law
    Q_ext_T =  h_ext_T*A_ext_T*(Tp_ext_T_volute[I]-Text[I]) #Turbine / Outside

    Q_ext_C =  h_ext_C*A_ext_C*(Tp_ext_C_volute[I]-Text[I]) #Compressor / Outside

    #A and B are limiting condition matrices
    A = np.array([[-A_tot*h_T_PT+Lmb_f*A_PT,-A_tot*h_T_PT-Lmb_f*A_PT,0,0,0,0],
                    [e**(a1*e_Plto), e**(-a1*e_Plto), -1, -1,0,0],
                    [a1*e**(a1*e_Plto),-a1*e**(-a1*e_Plto), -a2,a2,0,0],
                    [0,0,e**(a2*L_cc),e**(-a2*L_cc),-1,-1],
                    [0,0,-Lmb_f*A_cc*a2*e**(a2*L_cc),Lmb_f*A_cc*a2*e**(-a2*L_cc),
((-h_ext_C*P_ext_PC)/a3)*(e**(a3*e_Plto)-1)+Lmb_f*A_Pc*a3*e**(a3*e_Plto)-A_pertes_PC*h_ext_C,
((h_ext_C*P_ext_PC)/a3)*(e**(-a3*e_Plto)-1)-Lmb_f*A_Pc*a3*e**(-a3*e_Plto)-A_pertes_PC*h_ext_C],
                    [0,0,0,0, -Lmb_f*A_Pc*a3*e**(e_Plto*a3)-A_Pc*h_C_PC*e**(e_Plto*a3),
Lmb_f*A_Pc*a3*e**(-e_Plto*a3)-A_Pc*h_C_PC*e**(-e_Plto*a3)]])

    B = np.array([h_ext_T*(Tp_ext_T_volute[I]-Text[I])*A_ext_T+(A_ext_T*Em_f*sigma*(Tp_ext_T_volute[I]**4 - Text[I]**4))+A_tot*h_T_PT*(Text[I]-T_moy_T[I]), delt-Text[I],
                h_huile[I]*A_h*(Text[I]-T_h[I]),
                Text[I]-delt,
                0,
                Q_R_C[I]-perc_R*Q_R_T[I]+Q_ext_C+A_Pc*h_C_PC*(Text[I]-T_moy_C[I])])
```



```python
    #Resolution of limiting conditions

    C = np.linalg.solve(A,B)

    C1 = C[0]
    C2 = C[1]
    C3 = C[2]
    C4 = C[3]
    C5 = C[4]
    C6 = C[5]

    #Temperature Evolution Laws
    def T_PT (x) : #on turbine plateau
        return (C1*e**(a1*x)+C2*e**(-a1*x)+Text[I])

    def T_cc (x) : #on central body
        return (C3*e**(a2*x)+C4*e**(-a2*x)+delt)

    def T_PC (x) : #on compressor plateau
        return (C5*e**(a3*x)+C6*e**(-a3*x)+Text[I])

    ## Powaers and Efficiencies

    #Turbine Power Balance

    Q_T_PT = -Lmb_f*(a1)*A_PT*(C1-C2) #Turbine / Turbine Plateau

    #Compressor Power Balance

    Q_cond_fin_PC = -Lmb_f*a3*A_Pc*(C5*e**(a3*e_Plto)-C6*e**(-a3*e_Plto)) #Conductive heat at the end of the compressor plateau

    Q_ext_PC = ((h_ext_C*P_ext_PC)/a3)*(C5*(e**(a3*e_Plto)-1)+C6*(1-e**(-a3*e_Plto)))+h_ext_C*A_pertes_PC*(T_PC(0)-Text[I]) #heat dissipated on the compressor plateau external surface

    #Effective mechanical power
    P_arbre_T = P_T[I] - Q_T_PT - Q_ext_T - Q_R_T[I]  #real power on the mechanical axis given by the turbine

    P_arbre_C = P_C[I] - Q_cond_fin_PC + Q_ext_C + Q_R_C[I] #real power on the mechanical axis received by the compressor

    P_f = P_arbre_T - P_arbre_C #friction power dissipated into the oil

    rdt_comp = P_Cis[I]/P_arbre_C #compressor efficiency
    rdt_turbine = P_arbre_T/P_Tis[I] #turbine efficiency
    rdt_meca = P_arbre_C/P_arbre_T #mechanical efficiency
    if mode=='train': #training mode
        y_pred = np.array([rdt_comp, rdt_comp, rdt_turbine, rdt_turbine, rdt_meca, rdt_meca])
        return y_pred
    elif mode=='plot': #plot mode
        #partition of the turbine plateau + central body + compressor plateau assembly
        x1=np.linspace(0,0.0101,50)
        x2=np.linspace(0.0101,0.0512,200)
        x3=np.linspace(0.0512,0.0613,50)

        y1=T_PT(x1)
        y2=T_cc(x2-0.0101)
        y3=T_PC(x3-0.0512)

        y_plot=np.concatenate((y1,y2,y3),axis=0)
        return y_plot
    elif mode=='predict': #prediction mode
        y_pred = np.array([rdt_comp, rdt_turbine, rdt_meca])
        return [y_pred,[P_arbre_T,Q_ext_T,Q_T_PT,P_f,Q_ext_PC,Q_cond_fin_PC,Q_ext_C,P_arbre_C]]

## ---> Defining ANN functions

def normalize(matrix_to_normalize):
    #normalizes input data to mean = 0 and variance = 1 by column
    row = matrix_to_normalize.shape[0]
    col = matrix_to_normalize.shape[1]
    normalized_matrix_by_column = np.zeros((row,col))
    for j in range(col):
        mean = np.mean(matrix_to_normalize[:,j])
        std = np.std(matrix_to_normalize[:,j])
        for i in range(row):
            normalized_matrix_by_column[i,j] = (matrix_to_normalize[i,j]-mean)/std
    return normalized_matrix_by_column
```

```
def extract_data(sheet):
    #Extracts turbo charger performance data from Excel sheet

    Te_C = np.array([sheet.cell_value(r+1,0)+273 for r in range(sheet.nrows-1)]) #inlet compressor temperature in K
    Ts_C = np.array([sheet.cell_value(r+1,1)+273 for r in range(sheet.nrows-1)]) #outlet compressor temperature in K
    Te_T = np.array([sheet.cell_value(r+1,2)+273 for r in range(sheet.nrows-1)]) #inlet turbine temperature in K
    Ts_T = np.array([sheet.cell_value(r+1,3)+273 for r in range(sheet.nrows-1)]) #outlet turbine temperature in K
    T_h = np.array([sheet.cell_value(r+1,4)+273 for r in range(sheet.nrows-1)]) #oil temperature in K
    Text = np.array([sheet.cell_value(r+1,5)+273 for r in range(sheet.nrows-1)]) #cabin temperature in K
    Pe_C = np.array([sheet.cell_value(r+1,6) for r in range(sheet.nrows-1)]) #inlet compressor pressure in relative mB
    Ps_C = np.array([sheet.cell_value(r+1,7) for r in range(sheet.nrows-1)]) #outlet compressor pressure in relative mB
    Pe_T = np.array([sheet.cell_value(r+1,8) for r in range(sheet.nrows-1)]) #inlet turbine pressure in relative mB
    Ps_T = np.array([sheet.cell_value(r+1,9) for r in range(sheet.nrows-1)]) #outlet turbine pressure in relative mB
    Dm_C = np.array([sheet.cell_value(r+1,10) for r in range(sheet.nrows-1)])/3600 #compressor mass flow in kg/s
    Dm_T = np.array([sheet.cell_value(r+1,11) for r in range(sheet.nrows-1)])/3600 #turbine mass flow in kg/s
    regime = np.array([sheet.cell_value(r+1,12) for r in range(sheet.nrows-1)]) #turbo charger rpm
    Pext = np.array([sheet.cell_value(r+1,13) for r in range(sheet.nrows-1)]) #cabin pressure in mB
    Tp_ext_T_volute = np.array([(sheet.cell_value(r+1,15)+sheet.cell_value(r+1,20))*0.5+273 for r in range(sheet.nrows-1)]) #turbine volute external wall temperature in K
    Tp_ext_C_volute = np.array([(sheet.cell_value(r+1,16)+sheet.cell_value(r+1,22))*0.5+273 for r in range(sheet.nrows-1)]) #compressor volute external wall temperature in K
    Dm_h = np.array([sheet.cell_value(r+1,17) for r in range(sheet.nrows-1)])/3600.0*rho_h/1000 #oil mass flow in kg/h
    EV_T = np.array([sheet.cell_value(r+1,18) for r in range(sheet.nrows-1)]) #Percentage of opening of the turbine inlet valve
    EV_C = np.array([sheet.cell_value(r+1,19) for r in range(sheet.nrows-1)]) #Percentage of opening of the compressor inlet valve

    #Inlet Treatment
    T_moy_C = (Te_C+Ts_C)/2. #mean compressor flow temperature
    T_moy_T = (Te_T+Ts_T)/2. #mean turbine flow temperature

    Pe_C = Pe_C + Pext #inlet compressor pressure in bar
    Ps_C = Ps_C + Pext #outlet compressor pressure in bar
    Pe_T = Pe_T + Pext #inlet turbine pressure in bar
    Ps_T = Ps_T + Pext #outlet turbine pressure in bar

    #Corrected mass flow
    correction = Pref/(Tref**(1/2))

    Dm_C_cor = ((Dm_C)*((Te_C**(1/2)))/Pe_C)*correction #corrected compressor mass flow
    Dm_T_cor = ((Dm_T)*((Te_T**(1/2)))/Pe_T)*correction #corrected turbine mass flow

    #Network input
    x_train = np.array(np.transpose([Te_C, Ts_C, Te_T, Ts_T, T_h, Pe_C, Ps_C, Pe_T, Ps_T, Dm_C_cor, Dm_T_cor, regime, Tp_ext_T_volute, Tp_ext_C_volute, EV_T, EV_C])) #input parameters
    x = normalize(x_train) #normalized input parameters

    rdt_C = np.array([sheet.cell_value(r+1,20) for r in range(sheet.nrows-1)]) #compressor efficiency
    rdt_T = np.array([sheet.cell_value(r+1,21) for r in range(sheet.nrows-1)]) #turbine efficiency
    rdt_meca = np.array([sheet.cell_value(r+1,22) for r in range(sheet.nrows-1)]) #mechanical efficiency

    #Network desired output
    t = np.array(np.transpose([rdt_C, rdt_C, rdt_T, rdt_T, rdt_meca, rdt_meca]))

    return [x,t,Te_C,Ts_C,Te_T,Ts_T,T_h,Text,Pe_C,Ps_C,Pe_T,Ps_T,Dm_C,Dm_T,regime,Pext,Tp_ext_T_volute,Tp_ext_C_volute,Dm_h,EV_T,EV_C,T_moy_T,T_moy_C,x_train]
```

```
def softmax(entrada):
    #activation function applied to the hidden layer output
    #higher values become closer to 1 and lower values become closer to zero
    e_x = np.exp(entrada - np.max(entrada))
    return e_x / e_x.sum()

def PCA(x,nb_attributes):
    #adapts the input by neglecting low-impact parameters

    x_norm = normalize(x)

    #Covariance matrix
    C = np.cov(np.transpose(x_norm))

    #Covariance matrix eigenvalues
    eigen_values = np.linalg.eig(C)[0]
    eigen_vectors = np.linalg.eig(C)[1]

    i = 0
    percentage = 0
    sum = np.sum(eigen_values)

    #Number of attribute to be neglected
    while i<nb_attributes and percentage<0.90:
        percentage += eigen_values[i]/sum
        i = i+1
    neglect = i

    #Input without neglected attributes
    x_PCA = x_norm[:,:-neglect]

    #New number of input attributes
    nb_attributes -= neglect

    return [x_PCA, nb_attributes]
```

```
def prepare_training_validation(x_PCA,t):
    #Divides input data into training and validation sets

    x_full = x_PCA

    x = x_full[0:nb_class_train,:]
    x_val = x_full[nb_class_train:nb_class,:]

    t_full = t

    t = t_full[0:nb_class_train,:]
    t_val = t_full[nb_class_train:nb_class,:]

    x_full_shuffle = x_full
    t_full_shuffle = t_full

    return [x,x_val,t,t_val,nb_class_train,nb_class_val,t_full_shuffle,x_full_shuffle]

def initialize_weight(n1,n2):
    #initialized trainig weights
    rand = 0
    w = np.ones((np.int(n1),np.int(n2)))
    for i in range(0,np.int(n2)):
        for j in range(0,np.int(n1)):
            rand = rd.uniform(-1,+1)
            w[j,i] = rand
    return w

def initialize_error(n1,n2):
    #initializes training errors
    e = np.zeros((n1,n2))
    return e
```

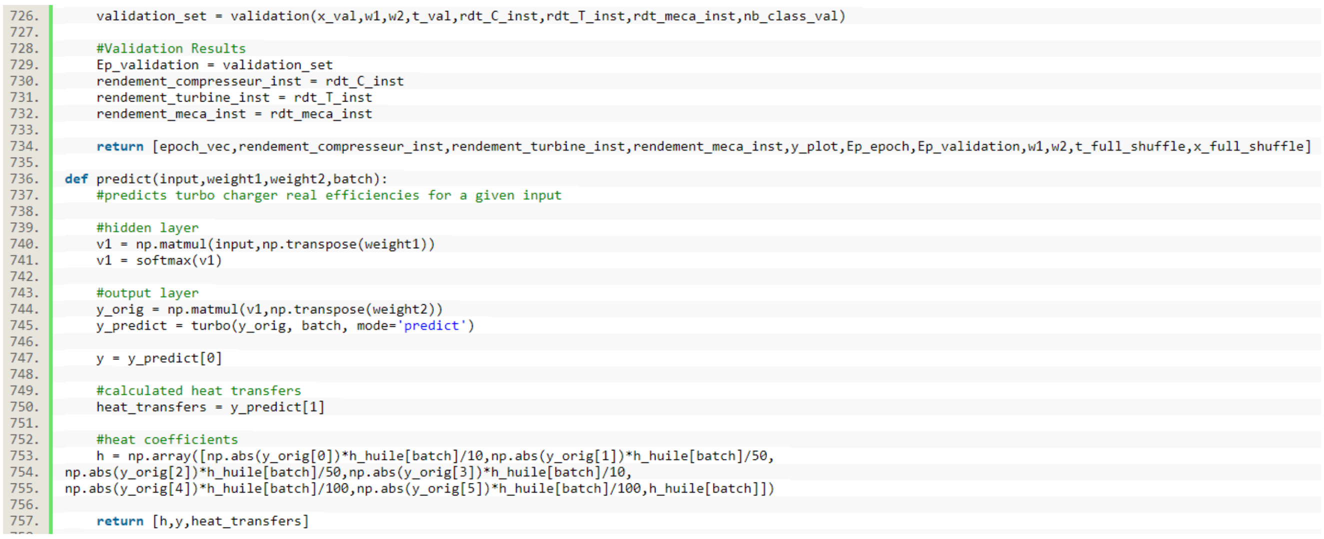
```
def gradiente(var_index,y_pred, batch):
    #turbo function gradient calculated through centered difference quotient
    #Gradient = ( f(x+dx) - f(x-dx) )/ 2dx

    inf = 1e-7
    input = np.abs(y_pred[var_index])
    #f(x+dx)
    upper_input = input + inf

    input_length = len(y_pred)
    y_pred_prim = np.zeros((input_length))

    for i in range(input_length):
        if i==var_index:
            y_pred_prim[i] = upper_input
        else:
            y_pred_prim[i] = y_pred[i]
    upper = turbo(y_pred_prim, batch)

    #f(x-dx)
    lower_input = input - inf
    for i in range(input_length):
        if i==var_index:
            y_pred_prim[i] = lower_input
        else:
            y_pred_prim[i] = y_pred[i]
    lower = turbo(y_pred_prim, batch)

    #( f(x+dx) - f(x-dx) )/ 2dx
    return (upper[var_index]-lower[var_index])/(2*inf)

def update_error_last(e,n1,n2,expected,output,output_orig,batch,output1,alpha,classe,delt):
    #updates the output layer error matrix
    e_old = e
    e_out = np.zeros((np.int(n2),np.int(n1)))
    for i in range(0,np.int(n1)):
        for j in range(0,np.int(n2)):
            delt[j] = -(expected[classe,j]-output[j])*output[j]*(1-output[j])
            e_out[j,i] = delt[j]*output1[i]*gradiente(j, output_orig, batch)
    #momentum term
    e_out += alpha*e_old
    return [e_out,delt]
```

```python
def update_error(e,n1,n2,expected,output,output_orig,batch,output1,output2,alpha,classe,delt,last,n3,w_final):
    #Updates hidden layer error matrix
    e_old = e
    e_out = np.zeros((np.int(n2),np.int(n1)))
    for i in range(0,np.int(n1)):
        for j in range(0,np.int(n2)):
            soma = 0
            for k in range(0,np.int(n3)):
                soma += delt[k]*w_final[k,j]
            e_out[j,i] = soma*output2[j]*output1[i]*(1-output2[j])
    e_out += alpha*e_old
    return e_out

def update_weight(w,n1,n2,eta,e):
    #Updates training weights
    w_old = w
    w = np.zeros((np.int(n2),np.int(n1)))
    for i in range(0,np.int(n1)):
        for j in range(0,np.int(n2)):
            w[j,i] = w_old[j,i] - eta*e[j,i]
    return w

def update_parameter(par,mode):
    #updates parameters alpha and eta
    if mode=='alpha':
        return max(par*alph,min_alpha)
    else:
        return max(alph*par,min_eta)

def mean_sqr_error(MSE,expected,output,n_last,classe):
    #Mean Squared Error
    for i in range(np.int(n_last)):
        MSE = MSE + 0.5*(expected[classe,i]-output[i])**2
    return MSE
```

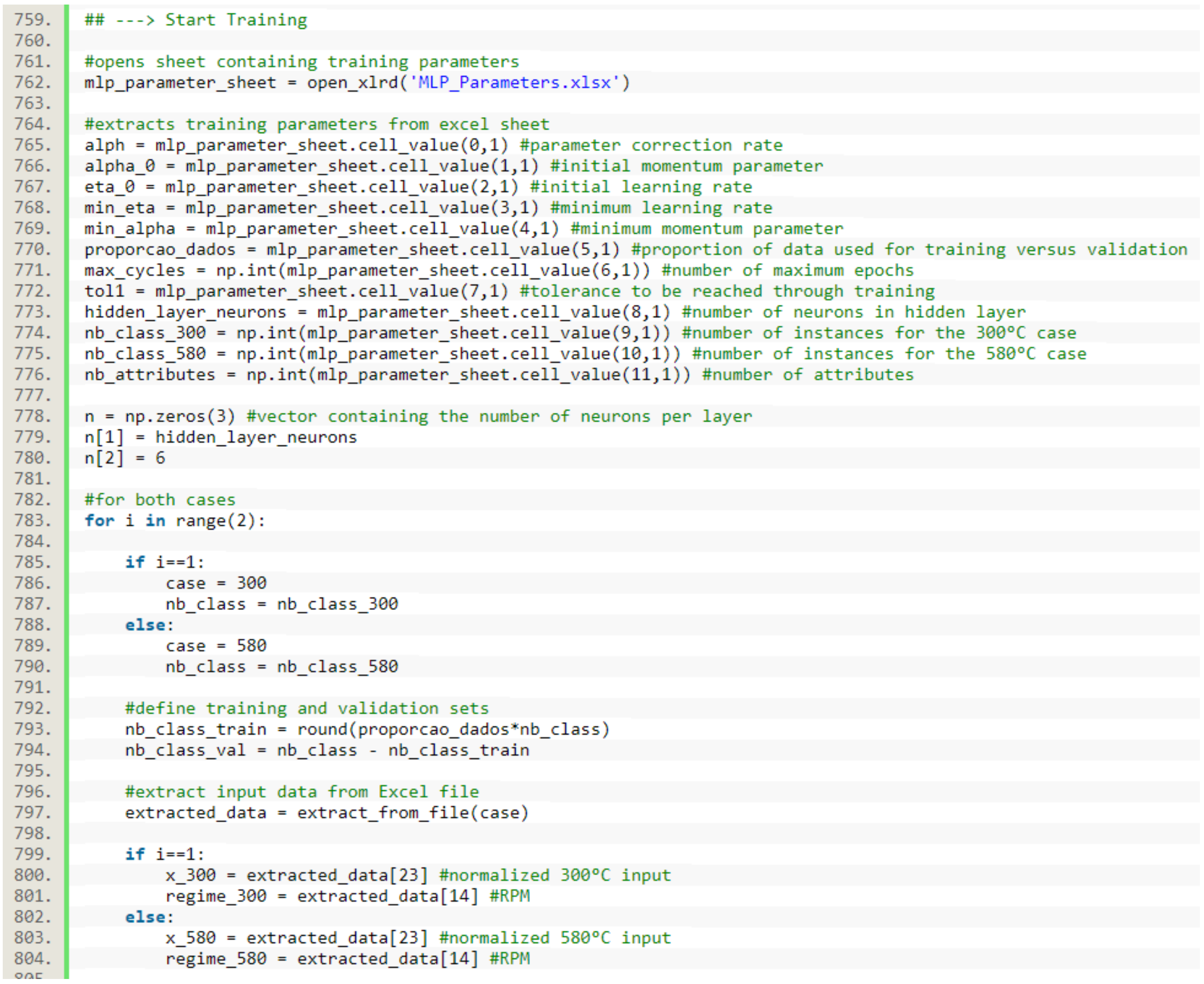
```
def train(input,weight1,weight2,expected,delt,error2,error3, nb_class_train):
    #trains MLP model

    epoch = 1 #initial epoch
    epoch_vec = epoch #epoch vector, for visualizing
    Ep = 0 #initial mean squared error

    Ep = tol1*2 #initial mean squared error

    Ep_epoch = Ep #mean squared error by epoch, for visualizing

    eta = eta_0 #initial learning rate
    alpha = alpha_0 #initial momentum term

    while(Ep>tol1 and epoch < max_cycles):

        rdt_C_inst = [] #compressor efficiency by epoch
        rdt_turbine_inst = [] #turbine efficiency by epoch
        rdt_meca_inst = [] #mechanical efficiency by epoch

        Ep_old = Ep_epoch #mean squared error by epoch, for visualizing
        Ep = 0 #initial mean squared error for each epoch

        for classe in range(0,nb_class_train):

            batch = classe #instance analyzed

            #hidden layer
            v1 = np.matmul(input[classe,:],np.transpose(weight1))
            v1 = softmax(v1)

            #output layer
            y_orig = np.matmul(v1,np.transpose(weight2))
            y = turbo(y_orig, batch)

            #output layer error and weight update
            error3 = update_error_last(error3,n[1],n[2],expected,y,y_orig,batch,v1,alpha,classe,delt)[0]
            delta = update_error_last(error3,n[1],n[2],expected,y,y_orig,batch,v1,alpha,classe,delt)[1]
            weight2 = update_weight(weight2,n[1],n[2],eta,error3)

            #hidden layer error and weight update
            error2 = update_error(error2,n[0],n[1],expected,y,y_orig,batch,input[classe],v1,alpha,classe,delta,False,n[2],weight2)
            weight1 = update_weight(weight1,n[0],n[1],eta,error2)

            rdt_C_inst = np.append(rdt_C_inst,y[0])  #compressor efficiency by epoch
            rdt_turbine_inst = np.append(rdt_turbine_inst,y[3])  #turbine efficiency by epoch
            rdt_meca_inst = np.append(rdt_meca_inst,y[5])  #mechanical efficiency by epoch

            Ep = mean_sqr_error(Ep,expected,y,n[2],classe) #mean squared error by epoch, for visualizing
        eta = update_parameter(eta,'eta') #update learning rate
        alpha = update_parameter(alpha,'alpha') #update momentum parameter

        #epoch vector, for visualizing
        epoch_old = epoch_vec
        epoch = epoch+1
        epoch_vec = np.append(epoch_old, epoch)

        Ep = Ep/(nb_class_train*2) #mean squared error for current epoch by training instance

        Ep_epoch = np.append(Ep_old, Ep) #mean squared error by epoch, for visualizing

        print('Epoch',epoch,'/',max_cycles)
        print('[=>............................] - loss:',Ep)
        print(y_orig)

    y_plot = turbo(y_orig, batch,'plot') #Plotting data

    return [Ep_epoch,weight1,weight2,rdt_C_inst,rdt_turbine_inst,rdt_meca_inst,epoch_vec,y_plot]

##Validation

def validation(x_val,w1,w2,t,rendement_compresseur_inst, rendement_turbine_inst, rendement_meca_inst,nb_class_val):
    #evaluates trained model's performance on validation set

    Ep = 0

    for classe in range(0,nb_class_val):

        #hidden layer
        v1 = np.matmul(x_val[classe,:],np.transpose(w1))
        v1 = softmax(v1)

        #instance analyzed
        batch = classe

        #output layer
        y_orig = np.matmul(v1,np.transpose(w2))
        y = turbo(y_orig, batch)

        Ep = mean_sqr_error(Ep,t,y,n[2],classe) #mean squared error by epoch, for visualizing

    Ep = Ep/(nb_class_val*2) #mean squared error for validation set

    return Ep
```

```
## Multi-Layer Perceptron

def extract_from_file(case):
    #extracts data from excel file data sheet

    if case==300:
        sheet = open_xlrd('Dados_Brutos_300.xlsx')
    elif case==580:
        sheet = open_xlrd('Dados_Brutos_580.xlsx')
    extracted_data = extract_data(sheet)

    return extracted_data
```

```
def MLP(input,output,nb_attributes):
    #Multi-Layer Perceptron Model

    #Principal Component Analysis
    pca = PCA(input,nb_attributes)
    x_PCA = pca[0]
    nb_attributes = pca[1]

    n[0] = nb_attributes

    #Weight and Error Initialization
    w1 = initialize_weight(n[1],n[0])
    w2 = initialize_weight(n[2],n[1])

    e2 = initialize_error(np.int(n[1]),np.int(n[0]))
    e3 = initialize_error(np.int(n[2]),np.int(n[1]))

    delta_0 = np.zeros(np.int(n[2]))

    #Prepare for Training
    training_set = prepare_training_validation(x_PCA,output)
    x = training_set[0]
    x_val = training_set[1]
    t = training_set[2]
    t_val = training_set[3]
    nb_class_train = training_set[4]
    nb_class_val = training_set[5]
    t_full_shuffle = training_set[6]
    x_full_shuffle = training_set[7]

    #Training
    train_set = train(x,w1,w2,t,delta_0,e2,e3,nb_class_train)

    Ep_epoch = train_set[0]

    #Trained Weights
    w1 = train_set[1]
    w2 = train_set[2]

    #Trained efficiencies (Network Output)
    rdt_C_inst = train_set[3]
    rdt_T_inst = train_set[4]
    rdt_meca_inst = train_set[5]
    epoch_vec = train_set[6]
    y_plot = train_set[7]
```



```
    validation_set = validation(x_val,w1,w2,t_val,rdt_C_inst,rdt_T_inst,rdt_meca_inst,nb_class_val)

    #Validation Results
    Ep_validation = validation_set
    rendement_compresseur_inst = rdt_C_inst
    rendement_turbine_inst = rdt_T_inst
    rendement_meca_inst = rdt_meca_inst

    return [epoch_vec,rendement_compresseur_inst,rendement_turbine_inst,rendement_meca_inst,y_plot,Ep_epoch,Ep_validation,w1,w2,t_full_shuffle,x_full_shuffle]

def predict(input,weight1,weight2,batch):
    #predicts turbo charger real efficiencies for a given input

    #hidden layer
    v1 = np.matmul(input,np.transpose(weight1))
    v1 = softmax(v1)

    #output layer
    y_orig = np.matmul(v1,np.transpose(weight2))
    y_predict = turbo(y_orig, batch, mode='predict')

    y = y_predict[0]

    #calculated heat transfers
    heat_transfers = y_predict[1]

    #heat coefficients
    h = np.array([np.abs(y_orig[0])*h_huile[batch]/10,np.abs(y_orig[1])*h_huile[batch]/50,
np.abs(y_orig[2])*h_huile[batch]/50,np.abs(y_orig[3])*h_huile[batch]/10,
np.abs(y_orig[4])*h_huile[batch]/100,np.abs(y_orig[5])*h_huile[batch]/100,h_huile[batch]])

    return [h,y,heat_transfers]
```



```
## ---> Start Training

#opens sheet containing training parameters
mlp_parameter_sheet = open_xlrd('MLP_Parameters.xlsx')

#extracts training parameters from excel sheet
alph = mlp_parameter_sheet.cell_value(0,1) #parameter correction rate
alpha_0 = mlp_parameter_sheet.cell_value(1,1) #initial momentum parameter
eta_0 = mlp_parameter_sheet.cell_value(2,1) #initial learning rate
min_eta = mlp_parameter_sheet.cell_value(3,1) #minimum learning rate
min_alpha = mlp_parameter_sheet.cell_value(4,1) #minimum momentum parameter
proporcao_dados = mlp_parameter_sheet.cell_value(5,1) #proportion of data used for training versus validation
max_cycles = np.int(mlp_parameter_sheet.cell_value(6,1)) #number of maximum epochs
tol1 = mlp_parameter_sheet.cell_value(7,1) #tolerance to be reached through training
hidden_layer_neurons = mlp_parameter_sheet.cell_value(8,1) #number of neurons in hidden layer
nb_class_300 = np.int(mlp_parameter_sheet.cell_value(9,1)) #number of instances for the 300°C case
nb_class_580 = np.int(mlp_parameter_sheet.cell_value(10,1)) #number of instances for the 580°C case
nb_attributes = np.int(mlp_parameter_sheet.cell_value(11,1)) #number of attributes

n = np.zeros(3) #vector containing the number of neurons per layer
n[1] = hidden_layer_neurons
n[2] = 6

#for both cases
for i in range(2):

    if i==1:
        case = 300
        nb_class = nb_class_300
    else:
        case = 580
        nb_class = nb_class_580

    #define training and validation sets
    nb_class_train = round(proporcao_dados*nb_class)
    nb_class_val = nb_class - nb_class_train

    #extract input data from Excel file
    extracted_data = extract_from_file(case)

    if i==1:
        x_300 = extracted_data[23] #normalized 300°C input
        regime_300 = extracted_data[14] #RPM
    else:
        x_580 = extracted_data[23] #normalized 580°C input
        regime_580 = extracted_data[14] #RPM
    x_norm = extracted_data[0] #normalized input
    t = extracted_data[1] #desired output
    Te_C = extracted_data[2] #input compressor temperature
    Ts_C = extracted_data[3] #output compressor temperature
    Te_T = extracted_data[4] #input turbine temperature
    Ts_T = extracted_data[5] #output turbine temperature
    T_h = extracted_data[6] #input oil temperature
    Text = extracted_data[7] #cabin temperature
    Pe_C = extracted_data[8] #input compressor pressure
    Ps_C = extracted_data[9] #output compressor pressure
    Pe_T = extracted_data[10] #input turbine pressure
    Ps_T = extracted_data[11] #output turbine pressure
    Dm_C = extracted_data[12] #compressor mass flow
    Dm_T = extracted_data[13] #turbine mass flow
    regime = extracted_data[14] #RPM
    Pext = extracted_data[15] #cabin pressure
    Tp_ext_T_volute = extracted_data[16] #external turbine volute wall temperature
    Tp_ext_C_volute = extracted_data[17] #external compressor volute wall temperature
    Dm_h = extracted_data[18] #oil mass flow
    EV_T = extracted_data[19] #turbine valve opening
    EV_C = extracted_data[20] #compressor valve opening
    T_moy_T = extracted_data[21] #mean turbine flow temperature
    T_moy_C = extracted_data[22] #mean compressor flow temperature

    #correlation coefficients
    correlation_coefs = correlation()

    cor_h_T_PT = correlation_coefs[0] #Turbine / Turbine Plateau
    cor_h_ext_T =correlation_coefs[1] #Turbine / Outside
    cor_h_ext_PT =correlation_coefs[2] #Turbine Plateau / Outside
    cor_h_C_PC =correlation_coefs[3] #Compressor / Compressor Plateau
    cor_h_ext_C =correlation_coefs[4] #Compressor / Outside
    cor_h_ext_cc =correlation_coefs[5] #Central Body / Outside
    h_huile = correlation_coefs[6] #Oil
    if i==1:
        cor_h_T_PT_300 = cor_h_T_PT #Turbine / Turbine Plateau
        cor_h_ext_T_300 = cor_h_ext_T #Turbine / Outside
        cor_h_ext_PT_300 = cor_h_ext_PT #Turbine Plateau / Outside
        cor_h_C_PC_300 = cor_h_C_PC #Compressor / Compressor Plateau
        cor_h_ext_C_300 = cor_h_ext_C #Compressor / Outside
        cor_h_ext_cc_300 = cor_h_ext_cc #Central Body / Outside
        h_huile_300 = h_huile #Oil
    else:
        cor_h_T_PT_580 = cor_h_T_PT #Turbine / Turbine Platea
        cor_h_ext_T_580 = cor_h_ext_T #Turbine / Outside
        cor_h_ext_PT_580 = cor_h_ext_PT #Turbine Plateau / Outside
        cor_h_C_PC_580 = cor_h_C_PC #Compressor / Compressor Plateau
        cor_h_ext_C_580 = cor_h_ext_C #Compressor / Outside
        cor_h_ext_cc_580 = cor_h_ext_cc #Central Body / Outside
        h_huile_580 = h_huile #Oil

    powers = independent_powers()

    if i==1:
        powers_300 = powers
    else:
        powers_580 = powers

    P_T = powers[0] #Turbine power
    P_Tis = powers[1] #Turbine isentropic power
    Q_R_T = powers[2] #Turbine radiated power
    P_C = powers[3] #Compressor power
    P_Cis = powers[4] #Compressor isentropic power
    Q_R_C =powers[5] #Compressor radiated power

    if i==1:
        taux_de_comp_c_300 = powers[6][0:nb_class_train] #Compressor Compression rate
        taux_de_detente_t_300 = powers[7][0:nb_class_train] #Turbine Compression rate
    else:
        taux_de_comp_c_580 = powers[6][0:nb_class_train] #Compressor Compression rate
        taux_de_detente_t_580 = powers[7][0:nb_class_train] #Turbine Compression rate

    trained_set = MLP(x_norm,t,nb_attributes)

    epoch_vec = trained_set[0]
    if i==1:
        rendement_compresseur_inst_300 = trained_set[1] #compressor efficiency by epoch
        rendement_turbine_inst_300 = trained_set[2] #turbine efficiency by epoch
        rendement_meca_inst_300 = trained_set[3] #mechanical efficiency by epoch
        y_plot_300 = trained_set[4]
        Ep_epoch_300 = trained_set[5] #mean squared error by training epoch
        Ep_validation_300 = trained_set[6] #mean squared error by validation epoch
        weight1_300 = trained_set[7] #trained weights from input to hidden layer
        weight2_300 = trained_set[8] #trained weights for hidden layer to output
        t_full_shuffle_300 = trained_set[9][0:nb_class_train]
        x_full_shuffle_300 = trained_set[10]
    else:
        rendement_compresseur_inst_580 = trained_set[1] #compressor efficiency by epoch
        rendement_turbine_inst_580 = trained_set[2] #turbine efficiency by epoch
        rendement_meca_inst_580 = trained_set[3] #mechanical efficiency by epoch
        y_plot_580 = trained_set[4]
        Ep_epoch_580 = trained_set[5] #mean squared error by training epoch
        Ep_validation_580 = trained_set[6] #mean squared error by validation epoch
        weight1_580 = trained_set[7] #trained weights from input to hidden layer
        weight2_580 = trained_set[8] #trained weights for hidden layer to output
        t_full_shuffle_580 = trained_set[9][0:nb_class_train]
        x_full_shuffle_580 = trained_set[10]

#Predict Random case

#300°C
occurence_300 = rd.randint(0,nb_class_300-1)
prediction_300 = predict(x_300[occurence_300][:-3],weight1_300,weight2_300,occurence_300)

predicted_convection_coefs_300 = prediction_300[0]
predicted_efficiencies_300 = 100*prediction_300[1]
heat_transfers_300 = prediction_300[2]

#580°C
occurence_580 = rd.randint(0,nb_class_580-1)
prediction_580 = predict(x_580[occurence_580][:-4],weight1_580,weight2_580,occurence_580)

predicted_convection_coefs_580 = prediction_580[0]
predicted_efficiencies_580 = 100*prediction_580[1]
heat_transfers_580 = prediction_580[2]
```

```
## Plots

tol1_vec = tol1*np.ones(max_cycles)

#Modelled vs Expected 300°C and 580°C
plt.figure(1)

#Mechanical Efficiencies x Expansion Rate
plt.subplot(331)
plt.scatter (taux_de_detente_t_300,rendement_meca_inst_300,c='blue',label='300°C')
plt.scatter (taux_de_detente_t_580,rendement_meca_inst_580,c='red',label='580°C')
plt.scatter (taux_de_detente_t_300,t_full_shuffle_300[:,4],c='blue',label='300°C',marker='+')
plt.scatter (taux_de_detente_t_580,t_full_shuffle_580[:,4],c='red',label='580°C',marker='+')
plt.xlabel('expansion rate')
plt.ylabel('mechanical efficiency')

#Compressor Efficiencies x Compression Rate
plt.subplot(332)
plt.xlabel('compression rate')
plt.ylabel('compressor efficiency')
plt.scatter (taux_de_comp_c_300,rendement_compresseur_inst_300,c='blue',label='300°C')
plt.scatter (taux_de_comp_c_580,rendement_compresseur_inst_580,c='red',label='580°C')
plt.scatter (taux_de_comp_c_300,t_full_shuffle_300[:,0],c='blue',label='300°C',marker='+')
plt.scatter (taux_de_comp_c_580,t_full_shuffle_580[:,0],c='red',label='580°C',marker='+')

#Turbine Efficiencies x Expansion Rate
plt.subplot(333)
plt.scatter (taux_de_detente_t_300, rendement_turbine_inst_300, c='blue',label='300°C')
plt.scatter (taux_de_detente_t_580, rendement_turbine_inst_580, c='red',label='580°C')
plt.scatter (taux_de_detente_t_300, t_full_shuffle_300[:,2], c='blue',label='300°C',marker='+')
plt.scatter (taux_de_detente_t_580, t_full_shuffle_580[:,2], c='red',label='580°C',marker='+')
plt.xlabel('expansion rate')
plt.ylabel('turbine efficiency')

#Temperature evolution law for the last input instance
plt.subplot (334)
x1=np.linspace(0,0.0101,50)
x2=np.linspace(0.0101,0.0512,200)
x3=np.linspace(0.0512,0.0613,50)
x_plot=np.concatenate((x1,x2,x3),axis=0)
plt.scatter(x_plot,y_plot_300,c='blue',label='300°C')
plt.scatter(x_plot,y_plot_580,c='red',label='580°C')
plt.xlabel('position')
plt.ylabel('Temperature(K)')
```

```
#Mean Squared Error by Epoch
plt.subplot (335)

plt.plot(epoch_vec,tol1_vec,c='black')
plt.xlabel('Epochs')

blue_patch = mpatches.Patch(color='blue', label='Mean Squared Error 300°C')
red_patch = mpatches.Patch(color='red', label='Mean Squared Error 580°C')
black_patch = mpatches.Patch(color='black', label='Tolerance')
plt.legend(handles=[blue_patch, red_patch, black_patch])

plt.plot(epoch_vec, Ep_epoch_300, c = 'blue')
plt.ylabel('Mean Squared Error')
print("Mean Squared Error on the output validation layer for input turbine temperature at 300°C:")
print(Ep_validation_300)

plt.plot(epoch_vec, Ep_epoch_580, c = 'red')
plt.ylabel('Mean Squared Error')
print("Mean Squared Error on the output validation layer for input turbine temperature at 580°C:")
print(Ep_validation_580)

#Modelled vs Expected 300°C
plt.figure(2)

#Mechanical Efficiencies x Expansion Rate
plt.subplot(341)
plt.scatter (taux_de_detente_t_300,rendement_meca_inst_300,c='blue',label='300°C')
plt.scatter (taux_de_detente_t_300,t_full_shuffle_300[:,4],c='blue',label='300°C',marker='+')
plt.xlabel('expansion rate')
plt.ylabel('mechanical efficiency')

#Compressor Efficiencies x Compression Rate
plt.subplot(342)
plt.xlabel('compression rate')
plt.ylabel('compressor efficiency')
plt.scatter (taux_de_comp_c_300,rendement_compresseur_inst_300,c='blue',label='300°C')
plt.scatter (taux_de_comp_c_300,t_full_shuffle_300[:,0],c='blue',label='300°C',marker='+')

#Turbine Efficiencies x Expansion Rate
plt.subplot(343)
plt.scatter (taux_de_detente_t_300, rendement_turbine_inst_300, c='blue',label='300°C')
plt.scatter (taux_de_detente_t_300, t_full_shuffle_300[:,2], c='blue',label='300°C',marker='+')
plt.xlabel('expansion rate')
plt.ylabel('turbine efficiency')
```

```
#Temperature evolution law for the last input instance
plt.subplot (345)
x1=np.linspace(0,0.0101,50)
x2=np.linspace(0.0101,0.0512,200)
x3=np.linspace(0.0512,0.0613,50)
x_plot=np.concatenate((x1,x2,x3),axis=0)
plt.scatter(x_plot,y_plot_300,c='blue',label='300°C')
plt.xlabel('position')
plt.ylabel('Temperature(K)')

#Mean Squared Error by Epoch
plt.subplot (346)
plt.plot(epoch_vec,tol1_vec,c='black')
plt.xlabel('Epochs')
blue_patch = mpatches.Patch(color='blue', label='Mean Squared Error 300°C')
black_patch = mpatches.Patch(color='black', label='Tolerance')
plt.legend(handles=[blue_patch, black_patch])
plt.plot(epoch_vec, Ep_epoch_300, c = 'blue')
plt.ylabel('Mean Squared Error')

#Modelled vs Expected 580°C
plt.figure(3)

#Mechanical Efficiencies x Expansion Rate
plt.subplot(351)
plt.scatter (taux_de_detente_t_580,rendement_meca_inst_580,c='red',label='580°C')
plt.scatter (taux_de_detente_t_580,t_full_shuffle_580[:,4],c='red',label='580°C',marker='+')
plt.xlabel('expansion rate')
plt.ylabel('mechanical efficiency')

#Compressor Efficiencies x Compression Rate
plt.subplot(352)
plt.xlabel('compression rate')
plt.ylabel('compressor efficiency')
plt.scatter (taux_de_comp_c_580,rendement_compresseur_inst_580,c='red',label='580°C')
plt.scatter (taux_de_comp_c_580,t_full_shuffle_580[:,0],c='red',label='580°C',marker='+')

#Turbine Efficiencies x Compression Rate
plt.subplot(353)
plt.scatter (taux_de_detente_t_580, rendement_turbine_inst_580, c='red',label='580°C')
plt.scatter (taux_de_detente_t_580, t_full_shuffle_580[:,2], c='red',label='580°C',marker='+')
plt.xlabel('expansion rate')
plt.ylabel('turbine efficiency')
```

```
#Temperature evolution law for the last input instance
plt.subplot (354)
x1=np.linspace(0,0.0101,50)
x2=np.linspace(0.0101,0.0512,200)
x3=np.linspace(0.0512,0.0613,50)
x_plot=np.concatenate((x1,x2,x3),axis=0)
plt.scatter(x_plot,y_plot_580,c='red',label='580°C')
plt.xlabel('position')
plt.ylabel('Temperature(K)')

#Mean Squared Error by Epoch
plt.subplot (355)
plt.plot(epoch_vec,tol1_vec,c='black')
plt.xlabel('Epochs')
red_patch = mpatches.Patch(color='red', label='Mean Squared Error 580°C')
black_patch = mpatches.Patch(color='black', label='Tolerance')
plt.legend(handles=[red_patch, black_patch])
plt.plot(epoch_vec, Ep_epoch_580, c = 'red')
plt.ylabel('Mean Squared Error')

#Modelled 300°C vc Modelled 580°C
plt.figure(4)

#Mechanical Efficiencies x Expansion Rate
plt.subplot(361)
plt.scatter (taux_de_detente_t_300,rendement_meca_inst_300,c='blue',label='300°C')
plt.scatter (taux_de_detente_t_580,rendement_meca_inst_580,c='red',label='580°C')
plt.xlabel('expansion rate')
plt.ylabel('mechanical efficiency')

#Compressor Efficiencies x Compression Rate
plt.subplot(362)
plt.xlabel('compression rate')
plt.ylabel('compressor efficiency')
plt.scatter (taux_de_comp_c_300,rendement_compresseur_inst_300,c='blue',label='300°C')
plt.scatter (taux_de_comp_c_580,rendement_compresseur_inst_580,c='red',label='580°C')

#Turbine Efficiencies x Compression Rate
plt.subplot(363)
plt.scatter (taux_de_detente_t_300, rendement_turbine_inst_300, c='blue',label='300°C')
plt.scatter (taux_de_detente_t_580, rendement_turbine_inst_580, c='red',label='580°C')
plt.xlabel('expansion rate')
plt.ylabel('turbine efficiency')
```

```
#Temperature evolution law for the last input instance
plt.subplot (364)
x1=np.linspace(0,0.0101,50)
x2=np.linspace(0.0101,0.0512,200)
x3=np.linspace(0.0512,0.0613,50)
x_plot=np.concatenate((x1,x2,x3),axis=0)
plt.scatter(x_plot,y_plot_300,c='blue',label='300°C')
plt.scatter(x_plot,y_plot_580,c='red',label='580°C')
plt.xlabel('position')
plt.ylabel('Temperature(K)')

#Mean Squared Error by Epoch
plt.subplot (335)
plt.plot(epoch_vec,tol1_vec,c='black')
plt.xlabel('Epochs')
blue_patch = mpatches.Patch(color='blue', label='Mean Squared Error 300°C')
red_patch = mpatches.Patch(color='red', label='Mean Squared Error 580°C')
black_patch = mpatches.Patch(color='black', label='Tolerance')
plt.legend(handles=[blue_patch, red_patch, black_patch])
plt.plot(epoch_vec, Ep_epoch_300, c = 'blue')
plt.ylabel('Mean Squared Error')
plt.plot(epoch_vec, Ep_epoch_580, c = 'red')
plt.ylabel('Mean Squared Error')

#Convection Coefficients (Correlation and Model), Efficiencies and Heat Exchanges
fig,axs = plt.subplots (4,1)

#Correlation Convection Coefficients
row = ['300°C','580°C']
columns = ['RPM','cor_h_T_PT','cor_h_ext_T','cor_h_ext_PT','cor_h_C_PC','cor_h_ext_C','cor_h_ext_cc','cor_h_huile']

cell_text = [[np.int(regime_300[occurence_300]),np.int(cor_h_T_PT_300[occurence_300]),
np.int(cor_h_ext_T_300[occurence_300]),np.int(cor_h_ext_PT_300[occurence_300]),
np.int(cor_h_C_PC_300[occurence_300]),np.int(cor_h_ext_C_300[occurence_300]),
np.int(cor_h_ext_cc_300[occurence_300]),np.int(h_huile_300[occurence_300])],[np.int(regime_580[occurence_580]),np.int(cor_h_T_PT_580[occurence_580]),
np.int(cor_h_ext_T_580[occurence_580]),np.int(cor_h_ext_PT_580[occurence_580]),
np.int(cor_h_C_PC_580[occurence_580]),np.int(cor_h_ext_C_580[occurence_580]),
np.int(cor_h_ext_cc_580[occurence_580]),np.int(h_huile_580[occurence_580])]]

axs[0].axis('tight')
axs[0].axis('off')

result_table = axs[0].table(cellText=cell_text,
                            rowLabels=row,
                            colLabels=columns,
                            loc='bottom')

result_table.auto_set_font_size(False)
result_table.set_fontsize(10)
```

```
#Modelled Convection Coefficients
row = ['300°C','580°C']
columns = ['RPM','h_T_PT','h_ext_T','h_ext_PT','h_C_PC','h_ext_C','h_ext_cc','h_huile']

cell_text = [[np.int(regime_300[occurence_300]),np.int(predicted_convection_coefs_300[0]),
np.int(predicted_convection_coefs_300[1]),np.int(predicted_convection_coefs_300[2]),
np.int(predicted_convection_coefs_300[3]),np.int(predicted_convection_coefs_300[4]),
np.int(predicted_convection_coefs_300[5]),np.int(predicted_convection_coefs_300[6])],[np.int(regime_580[occurence_580]),np.int(predicted_convection_coefs_580[0]),
np.int(predicted_convection_coefs_580[1]),np.int(predicted_convection_coefs_580[2]),
np.int(predicted_convection_coefs_580[3]),np.int(predicted_convection_coefs_580[4]),
np.int(predicted_convection_coefs_580[5]),np.int(predicted_convection_coefs_580[6])]]

axs[1].axis('tight')
axs[1].axis('off')

result_table = axs[1].table(cellText=cell_text,
                            rowLabels=row,
                            colLabels=columns,
                            loc='bottom')

result_table.auto_set_font_size(False)
result_table.set_fontsize(10)

#Efficiencies
row = ['300°C','580°C']
columns = ['Compressor','Turbine','Mechanical']

cell_text = [[np.int(predicted_efficiencies_300[0]),np.int(predicted_efficiencies_300[1]),np.int(predicted_efficiencies_300[2])],
[np.int(predicted_efficiencies_580[0]),np.int(predicted_efficiencies_580[1]),np.int(predicted_efficiencies_580[2])]]

axs[2].axis('tight')
axs[2].axis('off')

result_table = axs[2].table(cellText=cell_text,
                            rowLabels=row,
                            colLabels=columns,
                            loc='bottom')

result_table.auto_set_font_size(False)
result_table.set_fontsize(10)
```

```python
#Heat Exchanges
row = ['300°C','580°C']
columns = ['P_T','P_Tis','Q_R_T','P_arbre_T','Q_ext_T','Q_T_PT','P_f','Q_ext_PC','Q_cond_fin_PC','Q_ext_C','P_arbre_C','Q_R_C','P_Cis','P_C']

cell_text = [[np.int(powers_300[0][occurence_300]),np.int(powers_300[1][occurence_300]),np.int(powers_300[2][occurence_300]),np.int(heat_transfers_300[0]),
np.int(heat_transfers_300[1]),
np.int(heat_transfers_300[2]),np.int(heat_transfers_300[3]),np.int(heat_transfers_300[4]),
np.int(heat_transfers_300[5]),np.int(heat_transfers_300[6]),np.int(heat_transfers_300[7]),
np.int(powers_300[5][occurence_300]),np.int(powers_300[4][occurence_300]),np.int(powers_300[3][occurence_300])],[np.int(powers_580[0][occurence_580]),
np.int(powers_580[1][occurence_580]),np.int(powers_580[2][occurence_580]),np.int(heat_transfers_580[0]),np.int(heat_transfers_580[1]),
np.int(heat_transfers_580[2]),np.int(heat_transfers_580[3]),np.int(heat_transfers_580[4]),
np.int(heat_transfers_580[5]),np.int(heat_transfers_580[6]),np.int(heat_transfers_580[7]),
np.int(powers_580[5][occurence_580]),np.int(powers_580[4][occurence_580]),np.int(powers_580[3][occurence_580])]]

axs[3].axis('tight')
axs[3].axis('off')

result_table = axs[3].table(cellText=cell_text,
                      rowLabels=row,
                      colLabels=columns,
                      loc='bottom')

result_table.auto_set_font_size(False)
result_table.set_fontsize(10)

plt.show()

print("--- %s seconds ---" % (time.time() - start_time))
```