End studies' project, Master of Automotive Engineering

Analysis of Liner's Bore Distortion from Finite Elements Method Calculations.

DAF Trucks

Damien Gode (Czech Technical University)
Summer 2018

Acknowledgments

This part has been embedded within this report for purposes of thanking every person having advised and assisted me all along this internship.

First of all, I would like to express my most truthful gratitude to DAF Trucks for having welcomed me inside its structure, but more precisely to the initiator of this project, Mr. Paul Steuten, for having given me the opportunity to expand my knowledge on bore distortion analysis, but also for having guided me throughout the development of the project.

In a second time, my acknowledgments go to the Czech Technical University specially for having enabled me to achieve the 5th semester of the Master of Automotive Engineering in its establishments, and for having offered me the chance to carry out such a project as well. The gratitude of the University would be incomplete without mentioning Mr. Radek Tichanek, professor having accompanied me all along the placement.

Finally, I would like to conclude this section by thanking ENSTA Bretagne, my original engineering school within which I have received the most part of my expertise in mechanical engineering.

Abstract

The environmental restrictions, which become more and more demanding, force companies such as DAF Trucks to meticulously assess their engines through truly accurate 3D model, for purposes of optimizing as much as possible the performances of their vehicles. In that respect, DAF Trucks, during the examination of their engines, pays the greatest attention to the liners' inner surface deformations, what might have consequences in terms of oil consumption in extreme cases. These defects mostly provoked by assembly forces, thermal loads and gas pressures may effectively induce a leakage of oil into the combustion chamber, what will, at the end, lead to an increase of hydrocarbon emission.

To alleviate this issue, it is necessary to execute a bore distortion analysis in order to evaluate the level of deformation and that way be sure that the ability of the piston ring to distort will be enough to avoid any oil leakage.

This project proposed by DAF Trucks aimed to embed an additional module within Abaqus, whose the main ambition was to carry out the bore distortion analysis, which was usually outsourced by external companies, in order to afford DAF to have its own method to validate the bore distortion, and this for its entire set of engines.

Introduction

For several years the various companies of the automotive field increasingly rely on the use of 3D models to create and design their engines before producing them. In this way, DAF Trucks, not derogating from this statement, bases the development of its engines on a set of detailed 3D computations which aims to return all the pieces of information required to analyse the performances of the simulated engine. These performances may be about mechanical outputs, such as the torque or the power coming out of the engine, or else about fuel consumption, or finally about pollutant emissions, which, nowadays, become a gradually significant criterion in engine design.

For purposes of completing the computational approach of the DAF's Research and Development department, DAF proposed the possibility to work on the embedding of an auxiliary module within Abaqus, a Finite Element Analysis software, in order to provide an exhaustive overview on the bore distortion analysis. Actually, DAF Trucks was asking for a program displaying a set of useful pieces of information to visualize the geometrical distortion of the liners, mainly induced by both assembly forces and thermal loads. This assessment which was ordinarily carried out by external companies, constitutes a genuine step in the validation of an engine design, notably due to its direct impact on the oil consumption, and consequently the emission of hydrocarbons.

This labour settles in the frame of an end studies' project within a Master of Automotive Engineering attended to the Czech Technical University. This internship requested by the university tends to offer an ultimate opportunity to accumulate a professional experience before entering in the real labour world, and also to have a preview of the engineer's job. The choice of carrying out such an internship within DAF Trucks was founded on two prospects; firstly to obtain an international experience, what is inseparable with the current profession of engineer, the globalization having forced companies to expand to the world market; and secondly to focus the study not only on cars as usual, but this time on trucks which are a conveyance largely used all around the world notably in the carriage of goods, but also submitted to really demanding design constraints, which make this sector truly attractive.

The supervisor of this project was expecting from the student, who would be in charge of this work, to have some notions of programming, a perfect knowledge on engine operation and also a certain awareness about the Fourier Transform. As I have always planned to become an automotive engineer specialized in engine design, I have invariably made choices, throughout my learning path, going in that direction. That is why, I immediately perceived the DAF proposal as a perfect opportunity to extend my knowledge on the liners' bore distortion analysis, and consequently move towards my career ambitions. Moreover, as computer science becomes progressively important nowadays, it is requested from engineers, even the ones completely focused on mechanics field, to have some basic concepts of programming. That way, this mission, offered by DAF Trucks, was the perfect occasion to enhance the little knowledge about this topic I had acquired until there, during my academic years.

In that respect, this report aims to clearly introduce the entire development process of the program, from the definition of the requirement specifications arisen from the DAF's expectations, by going through the presentation of the company and the explanation of issues related to this analysis, to finish with the part about the elaboration of the program.

Table of Contents

Table of figures

1. Presentation of the company

DAF Trucks is a Dutch manufacturer specialized in the production of Trucks of any kind, founded in 1928 and reattached to the PACCAR group in 1996, which was already disposing of a few Truck companies in North America such as Kenworth or Peterbilt. In this way, by the acquisition of DAF Trucks, the PACCAR group was able to extend its influence within the world market related to its entrance in the European market. Actually, DAF Trucks is one of the most prominent truck manufacturers in the European-Union, the company sharing 15.3% of the heavy segment (trucks heavier than 16 tons) and 10.5% of the light one during the year 2017, according to features, which represents a total production of about 60 900 trucks. To manufacture their trucks, DAF Trucks disposes of several plants spread through Europe, in countries such as the Netherlands, the United-Kingdom, Belgium, for instance. However, DAF Trucks does not restrain itself to the European market and truly wants to expatriate their trucks all around the world. That is why, in 2013, DAF Trucks opened a new site in Brazil at Ponta Grossa in order to conquer the South American market. Hence, by dint of these production plants, DAF Trucks is able to produce and assemble all the components composing its three models of truck, the LF, the CF and the XF, respectively for light, medium and heavy-duty applications. These trucks may involve the implementation of four different types of engine, two of them, the smallest ones, the PX5 and PX7, being manufactured by an external company and the two others, the MX11 and MX13 being produced by DAF Trucks itself.

Figure 1 : Photo of the new XF and CF trucks, Trucks of the year 2018

The success of DAF Trucks lies in the quality and the performance provided by its trucks. Actually, these trucks are renowned for their comfort, their low fuel consumption, and their reliability. Due to that, it sounds pretty obvious DAF's trucks are often brought to the fore on the international stage by means of prizes such as the "Truck of the year" reward, an award discerned by a set of truck magazines. Thus, once again, the legitimacy of DAF Trucks as one of the main truck manufacturers has increased since the acquisition of the 2018 "Truck of the year" prize for both CF and XF trucks. Nevertheless, despite all these rewards, DAF Trucks keeps focusing on the perpetual improvement of its trucks, notably in terms of environmental performances, in order to always satisfy the various requirements defined by the EURO 6 decree, and the future ones. For this

purpose, DAF Trucks initiates within its Research and Development departments, a multitude of projects having as ambition either to enhance the existing or to design a new way to proceed.

Finally, DAF Trucks relies on the high quality of its products, which is the outcome of a demanding and advanced Research and Development work, to satisfy the expectations of the customers and that way, gradually become the leader of the truck market.

2. Context of study

The operation of a thermal engine is based on the vertical back-and-forth motion of pistons inside cylinders. Then this movement is ensured by a surface interaction between the piston rings and the inner surface of the liner, the liner being a low friction component press-fitted within the cylinder block. This surface interaction will be altered by various loads applied to the liner and this throughout the life of the engine. Thereupon, it sounds quite obvious that these alterations will directly affect the proper operation of the piston motion, and consequently the operation of the engine. The purpose of this project is about verifying that these deformations remain under the tolerated limits in order to avoid any kind of misbehaviour.

2.1. Nature of distortions

First of all, it is essential to emphasize that the entire engine block will expand or contract along with the temperature variations induced by the engine operation. Nevertheless, these displacements do present no interests for this project, and hence they will be ignored. Actually, the only displacements which truly find an interest within this project are the radial deformations of the liner, notably due to their direct impact on the surface interaction between the liner and the piston rings. These deformations may be provoked by different types of load gathered in five specific cases. The first one represents the machining of the engine block, which will induce mechanical stresses within the liner. The second one consists of assembling the engine block and the cylinder head by screwing the several bolts, bringing about mechanical deformations. The third is about the thermal distortions caused by the temperature inside the engine when it is operating. The penultimate possibility susceptible to produce such distortions lies in the application of a mechanical pressure to the liner's wall by the gas load. The last one concerns the mechanical wear induced by a long-term operation of the engine. Before going forward in the description of the context, it is relevant to talk about the Out-of-roundness parameter (OOR), a variable often used in the bore distortion analysis, which simply references to the radial deformation.

2.2. Piston rings role

The motion of the piston through the cylinder is only possible if it exits a gap between the circumferential surface of the piston and the inner wall of the liner. However, to afford the combustion to occur properly, it was required to seal the combustion chamber, and that way avoid any mixture leakage or oil intrusion within the chamber. This sealing is often executed, in the recent automotive industry, by a triplet of piston rings located on the piston's lateral surface. Moreover, for purposes of enabling as much as possible the piston motion, these piston rings must be made from low friction material, easily deformable. Thus, the ability of the rings to deform will afford to compensate the alterations occurring throughout the long-term operation of the engine, and consequently still ensure the sealing of the combustion chamber. Unfortunately, as any kind of material, the deformation properties of the piston rings have a physical limit, which will induce the introduction of a critical value for the out-of-roundness.

2.3. Misconduct issues

As described in the previous sections, the radial deformation of the liner is an inevitable phenomenon, which does not present a great risk for the engine operation as long as the out-of-roundness remains under the critical limits. However, in case of excess of these limits, some disturbances may appear, which will affect, not only the engine performances but also the satisfaction of the customers. The purpose of this section is about shortly presenting the main misbehaviours related to a liner deformed in a too significant way.

2.3.1. Oil consumption

The oil is a lubricant component introduced within the engine in order to ease the motion of all mechanical parts, what obviously includes the piston. Indeed, the oil, initially located in the oil pan, is propagating by projection on the liners' wall while the engine is running. Henceforth, the piston rings will, on one hand, spread the oil all around the liner's inner surface to reduce as much as possible the friction, but also on the other hand, will prevent the oil from entering in the combustion chamber. However, too important OORs would provoke the creation of gaps which the oil would have the opportunity to flow through, into the combustion chamber. Even if this problem does not have a direct impact on the engine efficiency in terms of performances, it remains concerning in an environmental point of view, but also in a customer satisfaction point of view. Actually, as the lubricant oil is mainly comprised of carbon molecules, its potential burn within the combustion chamber would produce the appearance of hydrocarbon in the exhaust gas, which poses a problem because hydrocarbons are chemical molecules really polluting. Moreover, the combustion of oil would automatically induce a reduction of its quantity inside the engine, what would force the customer to refill oil levels more often than the statements made by DAF, and hence what would taint the brand image of the company. For all these reasons, DAF pays a special attention to the oil consumption, either by initiating projects, directly or indirectly connected to this matter, such as the one described in this report, or by assessing it through experimental measurements.

2.3.2. Rotation of piston rings

The piston rings are mounted on the piston head in such a way that it exits a degree of freedom, the rotation along the axis carrying the piston motion, which is needed to ensure the proper spread of the oil on the liner's wall. It is relevant to underline that the dynamic rotation of the piston rings has a straight effect on the wear of the liner's inner surface, and consequently on the lifetime of the entire engine. Actually, a too rapid rotation would engender an excessive wear of the liner, and on the opposite no rotation would be a source of non-uniform wear. This variation of rotational speed may be the result of the roundness modification, notably because it seems obvious that a shrinkage of the liner's radius, locally speaking, would induce an increase of the surface pressure between the piston rings and the liner's wall, what would provoke a reduction of rotational speed; conversely an expansion of the liner's radius, once again locally speaking, would bring about a diminution of surface pressure, and hence a rise of the rotational speed. Therefore, the change of the liner's roundness directly influences the wear of the liner's wall, and thus the lifetime of the engine.

2.3.3. Blowby

The blowby designates the passage of the exhaust gas from the combustion chamber to the crankcase. This phenomenon may be the outcome of many factors, such as the thermal expansion and shrinkage, the vibrations, or also the out-of-roundness. Indeed, it is quite natural to guess that the variation of diameter may ease the leakage of the exhaust gas towards the crankcase. Therefore the radial deformations of the liner may provoke an increase of the blowby effect, what may intensify the wear of the liner, and consequently reduce the lifetime of the entire engine.

2.4. Finite Elements Mechanics analysis

One of the methods implemented for purposes of assessing the bore distortions relies on experimental measurements carried out through a test rig. Indeed, by using a set of sensors, more precisely a set of Micro-epsilon transducers made from invar and with a low coefficient of thermal expansion, moreover located at key points all around the piston head's lateral surface, it is possible to measure the value of distortions. However, this process remains truly complex and demanding in time and resources. That is why, it is more common to measure the bore distortion from a Finite Elements Mechanics (FEM) model, which will afford to compute the outcome quite accurately. This method is based on a succession of calculations whose the ambition is to compute one by one all the inputs required for the elaboration of the complete computation. The first FEM analysis aims to compute the value of the coolant's heat transfer coefficient from a Computational Fluid Dynamics (CFD) method. Once this parameter known, the next step of calculation is the thermal analysis step which consists of determining the temperature within the engine. This computation may only be achieved if the thermal conductivity of each component, the heat transfer coefficient of each component's surfaces, and the boundary conditions in terms of temperature are specified. The last calculation needed to be done before launching the final one, is about evaluating the mechanical part of bore distortion, meaning the bore distortion only induced by the assembly forces excluding thermal effect. Once again, this model requires some input parameters such as Young's modulus or assembly bolt forces for instance. From there, by combining the outcome of the previous FEM calculations, it is possible to establish the final FEM model, and that way to obtain the genuine value of bore distortion, including both thermal and mechanical aspect. In order to complete the pieces of information given through this section, it is relevant to indicate that all the models depict a three-cylinder engine whereas real DAF engines are six cylinders engines because it is a design choice to save both memory space and computation time.

3. Requirements specification

As for any kind of project, the clear definition of objectives constitutes the foundation of the work assignment and affords to guide the outcome towards the most sustainable solution. That way, by means of discussions with the supervisor of this project, it was attainable to edit an explicit requirements specification, gathering the ambitions that the program had to satisfy.

3.1. Work base

The purpose of this project is about offering a method to analyse the liners' bore distortion of the two main engines produced by DAF Trucks, in order to complete the post-conception confirmation process. Usually, the bore distortion analysis is devolved to an external company (such as AVL for instance) in charge of achieving the FEM calculations, and then, in a second time, approved or not by DAF. But through this project, DAF shows its desire to possess its own computational routine to assess the liners' bore distortion for different engine states. Nevertheless, it is truly significant to declare that this project did not have to start from scratches, due to the existence of a draft carried out by the supervisor of this project, made from Excel. That way, the method which needed to be implemented was perfectly known. Actually, the main principle behind the bore distortion analysis relies on the application of Fourier transform (cf. the section 4. Fourier analysis) and this on a certain type of data provided by the outcomes of FEM calculations executed through Abaqus (a Computer Aided Engineering software). Therefore, the mission behind such a project was to embed within Abaqus an additional module aiming to complete the bore distortion analysis, including the Fourier transform. The ability to interact with a Python console through Abaqus, Python being the programming language implemented to code this software, has led to the choice of building the future program with Python for purposes of facilitating as much as possible the integration of the module. To complete this section, it may be relevant to underline that the elaboration of the program will be based on files concerning the MX11 and MX13 DAF's engines, which are both already produced by DAF. Hence, DAF already disposes of reports corroborating that the out of roundness remains under the critical limits for these engines. Then by relying on these reports, it was possible to calibrate the program until obtaining the same results.

3.2. Project ambitions

For purposes of ensuring the relevance of the bore distortion analysis, it is necessary to split the study into several perspectives. Indeed, to consider the future program as an effective analysis tool, it will have to afford the user to have an access to some fragments of information, and this way give a global overview of the state of deformation. These elements of study may be distributed into three categories.

3.2.1. Liner shape overview

The first aspect of the liners' bore distortion analysis consists of delivering a direct vision of the deformation in order to show how the liner warps under different cases of load. That way, it was decided, after few discussions with the supervisor of this project but also by looking at what was done in the report written by AVL, to display, on one hand, the cross-section profile of the liner, and on the other hand the way the liner deforms along its height.

3.2.2. Fourier analysis

The implementation of the Fourier transform within the bore distortion analysis affords to obtain further information about the way the liner deforms. In short, from the Fourier transform, it will be possible to set up some indirect view of the liner deformation. It is important to note that further information about the Fourier analysis will be depicted in details within the section 4 of this report.

3.2.3. Temperature aspect

As the liners' bore distortion strongly depends on the thermal state of the engine, more precisely on the temperature inside the liners, it seems natural to embed within the script a functionality displaying the temperature of the liners' walls. That way, it will be possible to connect the deformation of the liner with the local temperature and consequently corroborate the impact of the temperature on distortions.

4. Fourier analysis

Despite the good accuracy of Abaqus, the simple exploitation of the values coming out of this software is not enough to have a complete view on liners' bore distortion. Consequently, it is necessary, in order to dispose of an exhaustive analysis method, to execute a signal treatment process based on the implementation of the Fourier transform.

4.1. Fourier theory

The Fourier transform is a mathematical approach decomposing an input signal, most of the time it is about temporal signals, into a sum of basic terms defined from a complex exponential (sometimes expressed from the Euler's formula as the sum of a real cosine and a complex sine). It is important to add that if the nature of the input signal is temporal, then the output of the Fourier transform of such a signal would be frequency. Nevertheless, this method may obviously be applied to any kind of signal not only temporal ones, and hence may be set up within this project. Furthermore, this mathematical method relies on two types of transformation, one taking a discrete signal as input and the other one a continuous signal, and for both of them it exists a direct formula, computing the Fourier terms from the input signal, and a reverse formula, naturally achieving the opposite. As the values extracted from Abaqus will be related to a whole of distinct nodes, it may be postulated that only the discrete expression of the Fourier transform proves an interest within the project. The both formulas, direct and reverse, behind such a method are defined as below:

$$\begin{cases} X_k = \frac{1}{N}\sum_{n=0}^{N-1} x_n . \exp[\,-\mathrm{j}\,.\left(k.\frac{2\pi n}{N}\right)] \;\; direct \\ x_n = \sum_{k=0}^{N-1} X_k . \exp[\,\mathrm{j}\,.\left(k.\frac{2\pi n}{N}\right)] \;\; reverse \end{cases}$$

From the explanation detailed above these formulas, it is possible to deduce on one hand that x_n gathers the N values of the input signal and on the other hand that X_k represents the output of the Fourier transform. More precisely, X_k is a list of complex from which it is feasible to assess the values of both Fourier coefficients and the dephasing angles, respectively by taking the module and by computing the arctangent of the imaginary part over the real part. However, these entities are not merely mathematical coefficients deprived of any physical sense. Indeed, as specified earlier within this section, the main purpose of the Fourier transform is to decompose a signal into a sum of elementary components, nevertheless it may appear either the involvement of these components are not identical or they require to be lightly shifted to finally result to the initial signal. That way, the Fourier coefficients are in charge of weighting the commitment of the various exponential terms and the dephasing angles are responsible of shifting it at the proper position.

4.2. Application to the bore distortion analysis

As mentioned in the previous paragraph, the Fourier transform is usually applied to temporal signals, however, it appears that in the frame of this project, meaning the bore distortion analysis, this method is the most used in order to obtain a preview of the radial deformations.

4.2.1. Input of the method

First and foremost, it is important to say that this study only focuses on one specific group of surfaces, the liners' inner surfaces, and consequently only the nodes of these surfaces will be concerned by the analysis. Moreover, after several readings of former reports about this topic and also after having discussed with the supervisor of this project, it was decided to draw from what was commonly done in the bore distortion analysis, and to display the results for distinct profiles, and this all along the liner's height. In this way, according to the two previous remarks, the idea behind the execution of the Fourier transform within this project was about gathering the nodes located at the same height from the liner's top, under the form of one single entity, for purposes of obtaining the deformed shape of the liner's section at this height (cf. the figure right below).

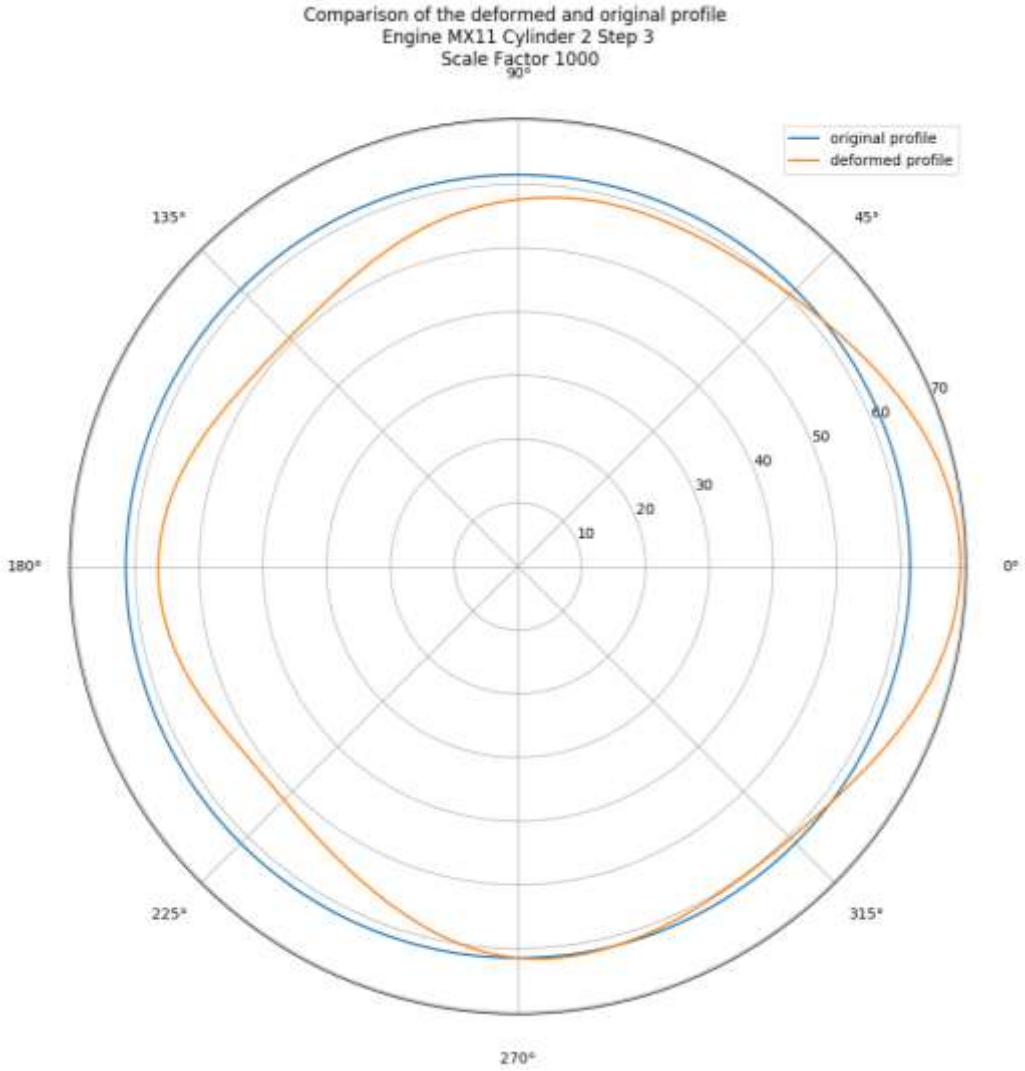


Figure 2 : Output of the final program illustrating a deformed section

Finally, the Fourier transform implemented within this project will have to compute both Fourier coefficients and dephasing angles, and this from the deformed liners' profiles, such as the one illustrated in the figure above, which means from the data lists bringing about these graphs, in others terms the lists containing the radial displacements of the section's nodes.

4.2.2. Change in the formulas

The adaptation of the Fourier transform to the bore distortion analysis involves the necessity to modify the mathematical expressions. From there, and in order to facilitate the future explanations, the decision of presenting the changes in the formulas before detailing the physical sense of this method has been taken. Moreover, it might appear useful, at the sight of the project expectations, to only focus on the direct Fourier transform, but it has been decided to also implement the reverse method, initially for the sake of outcome validation. Actually, the direct formula does not require any modification to be embedded within the bore distortion analysis, which is not the case of the reverse one. Nevertheless, the adjustment does not represent a big deal and only consists of rewriting the complex X_k under its exponential form, which is $X_k = F_c[k]e^{j\varphi_k}$, with F_c the Fourier coefficient and φ_k the dephasing angle. The substitution within the reverse Fourier transform expression affords to reach this formula:

$$x_n = \sum_{k=0}^{N-1} F_c[k].\exp[\,\mathrm{j}\,.(k.\theta_n + \varphi_k)]$$

To conclude with this section, two remarks have to be pointed out. The first one is about the appearance of the θ_n entity, which has been introduced to replace the term $\frac{2\pi n}{N}$ for the sake of simplification. The second remark is related to the function exponential, and to its potential change. Indeed, this function exponential may be substituted, according to the nature of the signal, either by a cosine when the signal is purely real or by a sine when the signal is complex. In this case, as the radial deformations are considered as mainly real, the exponential is the previous mathematical expression may be replaced by a cosine.

4.2.3. Meaning of the results

As it is mentioned in the first part of this section, the main interest of the Fourier transform is about decomposing a given signal into a sum of simpler signals, which are called harmonics. That way, in the frame of the bore distortion analysis, the Fourier transform is implemented in order to break down a deformed profile, such as the one depicted in the figure 2, into a sum of nine harmonics from the zeroth order to the eighth one (all the others being too small to really have an impact on the origins of deformations), each of them describing a specific mode of deformation. Actually, each harmonic, indexed by the variable k in the preceding mathematical formulas, corresponds to a certain pattern of distortion described in the figure 3 .

That way, by observing the figure, it is possible to notice that the zeroth order coincides with a potential radial expansion of the liner's section, the first one depicts the prospective eccentricity of the profile, whereas all the other harmonics, from the second to the eighth, have a more significant incidence on the geometry of the liner. Effectively, contrary to the two first orders, all the other ones straightforwardly modify the shape of the profile as if it was subjected to centripetal forces, whose the number is directly related to the order of the harmonic. That way, a cylinder surrounded by six fixation screws will present a sixth order more significant than a cylinder only surrounded by four. Therefore the application of the Fourier transform to the lists containing the radial displacements computed from Abaqus, will afford to obtain both Fourier coefficients and dephasing angles for each order. It is important to underline that the Fourier coefficients represent the maximal distance,

radially speaking, either positive or negative, between the deformed profile and the initial one, whereas the dephasing angle corresponds to a potential rotation of the deformed shape around its center.

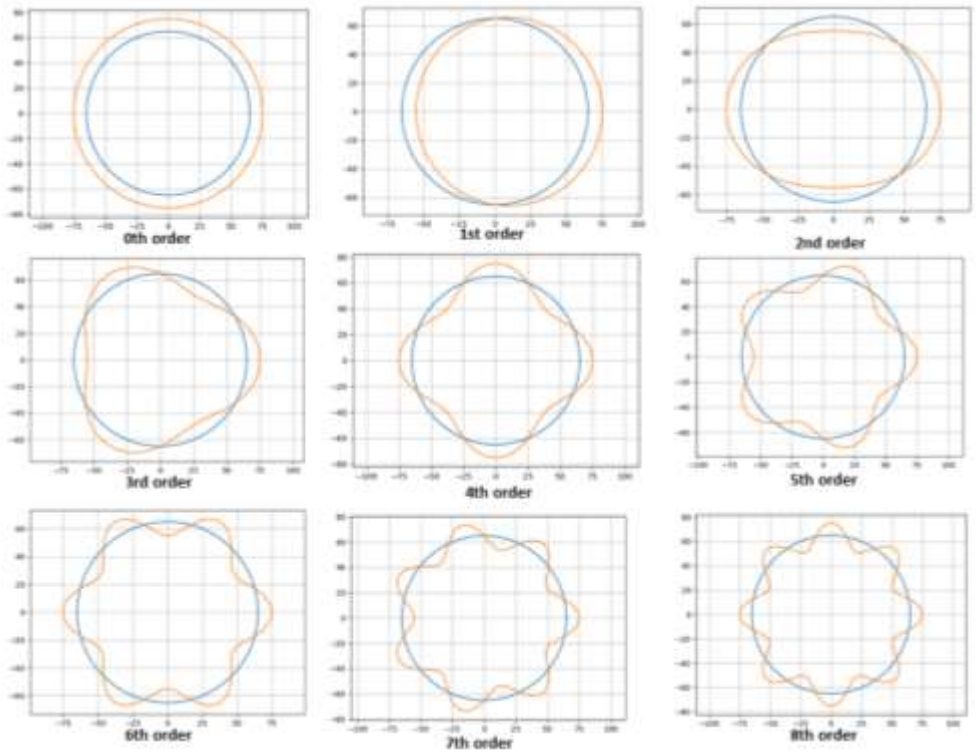


Figure 3 : Illustration of the different patterns of distortion

In that respect, as the forces applied to the liner will have a more or less significant impact on the different harmonics, the Fourier transform executed in the frame of the bore distortion analysis affords to obtain a quick preview on the most predominant modes of deformation, and that way inform the user about the real nature of the deformation.

4.3. Definition of limits

As for any type of analysis, it is not possible to make a judgement on results simply by observing the output computed by a method. Therefore, it is necessary to define a set of reference values in order to place the outcomes with respect to these limits, and that way to conclude with the analysis aspect of the project. In the case of bore distortion analysis, it exists two types of limit, the borderline limit and the critical limit. The critical limit, as its name suggests, represents the most extreme limit which must not be exceeded under any circumstances otherwise it would lead to engine's misbehaviour. The borderline limit, for its part, constitutes more a kind of design ambition; in other words it is wanted to remain under this limit as much as possible but in case of excess, the integrity of the engine is not put at risk while the values stay lower than the critical limit. The elaboration of these limits is based on the principle of oil consumption (detailed in the section 2.3.1.), and consequently on the physical properties of the piston rings. Actually, according to the report "Analysis of Distortions of Cylinders and Conformability of Position Rings" written by Dunaevsky, it is feasible to obtain these limits, at least the critical ones, from the following formula:

$$A_\omega = \frac{1.52.Q.r^3}{h.t^3.E.(\omega^2 - 1)}$$

To ensure the proper understanding of the previous expression, it is required to enumerate the name of the several entities involved in the definition of the bore distortion limit A_ω. That way, the variable Q stands for the piston ring tension, the r for the piston ring radius, the t for the piston ring thickness, E for its Young's modulus, h for its height and finally ω stands for the distortion order, such as the ones depicted in the previous section.

5. Definition of the operating mode

Once the context around this project was clarified, that meant after having pointed out the expectations behind the development of the program, but also after having completed all the bibliography research, notably the one about the Fourier transform, it was necessary to focus on a theoretical approach to execute the bore distortion analysis.

5.1. Selection of the input data

The reading of the previous section about the Fourier analysis easily affords to bring to the fore the type of information needed to be extracted from Abaqus. Actually, as the Fourier transform will be applied to the radial displacements of the nodes, it sounds obvious to work with the equivalent variable within Abaqus, which is the U1 variable when the user coordinates system is defined as a cylindrical one, centred on a liner. However, the Fourier transform may be correctly implemented only if the nodes are gathered by height, as explained in the paragraph 4.1. Hence, it is essential to find a way to refer to the nodes' coordinates in order to sort them. Finally, as the bore distortion analysis also consists of returning the liners' walls temperature, it was also required to retrieve the variable NT11 corresponding to the nodes' temperature in Abaqus.

5.2. Use of an external Python interpreter

At this moment of the project, it is relevant to remind that the main ambition of the program was to offer a visual overview of the bore distortion analysis through a set of graphs displaying the various entities defined in the requirements specification section. Consequently, after having identified the variables to be exploited, it was required to figure out a method to return the data under a graphic shape, and this from the Python console made available by Abaqus, as initially planned. The consideration of this issue has led to a couple of conclusions. On one hand, it turns out that Abaqus disposes of a few functionalities enabling to display some graphs, however these methods are truly specific and hence cannot satisfy the freedom demanded by the expectations of the project. Before explaining the matter of the second conclusion, it is significant to declare that, within Python, it exists a few libraries (a kind of program extension affording Python to carry out some additional tasks), whose the mission is to generate charts, the most famous is called *matplotlib*. In that respect, the second remark lies in the fact that the Python console provided by Abaqus is deprived of such a library, which makes impossible the generation of graph through it. From there, three options had emerged. The first one consisted of modifying the internal structure of Abaqus in order to embed a new functionality displaying the type of graph wished. This solution was immediately forgotten notably due to its complexity. The second solution, a little bit easier, was about installing a *matplotlib* library within the Abaqus' Python console. Unfortunately, after several attempts and after a discussion with the assistance service of SIMULIA, which is the group having implemented Abaqus, it appeared that such an update was not possible, for sake of software protection. Therefore, the last option, which will be the retained one, lied in calling an external Python interpreter in order to carry out all the applications of the program.

5.3. Retained method

As the choice of using an external Python interpreter was inevitable, it was necessary to figure out a solution to extract the pieces of information, depicted in the paragraph 5.1, from Abaqus to the external Python interpreter, in order to ensure the proper operation of the program. Fortunately, as Abaqus is a well-developed CAE software, it is possible, by using some functionalities available within its interface, to save some data inside text files. Therefore by means of this process, it is feasible to generate, in a dedicated workspace, a set of files containing all the pieces of data needed to complete the bore distortion analysis, and this for any configuration of study. In this way, as a Python console offers the opportunity to read and exploit such files, it has been decided to base the operation of this module on a two stages process consisting of firstly generating the text files from Abaqus, for purposes of, in a second time, importing and exploiting them into the Python interpreter.

5.4. Levels of design

Even if the program might be considered as fully operational once the script would dispose of a function for each ambition described in the section 3 of this report, it appeared that in its conception it was feasible to define several design levels. Indeed throughout the project, as soon as the coarse script was edited, it was decided to focus on a way to refine as much as possible the final outcome. That way, the idea to do so, was to clearly distinguish small improvement steps, and then try to roll them out one by one, until being unable to do better, in order to obtain the most optimized version. Before detailing more accurately the different levels, it is relevant to specify that most of these grades are based on observations appeared during the development of the program, however, they are included within this part, for sake of clarity.

5.4.1. First design level

From the various statements outlined within this section, it is admissible to declare that the first conception level merely consisted of writing a Python script achieving all the expectations raised by the project and this in a proper way. Hence, at this design stage, it was possible to display all the pieces of information about the bore distortion analysis by simply calling a few methods through the Python console (in other terms by typing the name of the method with all its parameters).

5.4.2. Second design level

The first upgrade which has been considered was to set up a graphical interface in order to ease all the interactions with the program, including the definition of the studied configuration but also the calling of the various functions. Incidentally, it is significant to mention that what is called “configuration” merely represents a set of information which gathers the name of the engine to be analysed, but also the cylinder, the loading case (usually called “step”) and potentially a specific cylinder section (defined from a height from its top). From there, the idea was to define this configuration by using an indexing system, with which every entity (cylinder, step, or height) is connected to an index (positive integer). Moreover, as depicted above in this paragraph, this interface must afford a quick access to the various methods conceived earlier in the program development. Therefore the retained solution to achieve such a request was to implement a series of buttons referring to each of the applications.

5.4.3. Third design level

This improvement is about the generation of the reports made from Abaqus. Actually, so far, the creation of these files had required the intervention of the user. Nevertheless, as the process to obtain the files for every potential configuration is long and laborious, it appeared quite natural that the next design amelioration lied in the automation of this procedure. Therefore the idea was to find a way to generate these files automatically from a Python script, which would be run within Abaqus.

5.4.4. Fourth design level

After having created a script enabling to generate the different data reports, it was natural, once again, to think about the implementation of a graphical interface providing a direct interaction with all the functions set up within the generation script, and this through Abaqus.

5.4.5. Fifth design level

The ultimate design level consisted of finding a way to bind the two distinct interfaces in order to have one entity program. Indeed, the idea was to figure out a method to launch the program interface from the generation files interface popped up within Abaqus, by simply clicking on a button, and moreover without manually opening the external Python interpreter.

6. Program conception

After having presented the theoretical approach of the program, it is now essential to look at the technical aspect induced by such a work. This section has been introduced within this report for purposes of clearly describing and justifying every design step and choice which has been made throughout the conception process. Hence, the following content will afford to better understand the objectives and the achievements of the different scripts involved in the program, and this without explaining the Python code line by line. Finally, the section will be organized according to these Python scripts, for the sake of clarity. Moreover, it is crucial to add that the Python script will be attached to this report in the appendix section.

6.1. BoreDistortion_Analysis_V7

This script constitutes the most important part of the entire program, because it gathers all the bore distortion analysis, which represents a huge portion of the demanded task. Henceforth, it is truly relevant to carefully dissect this Python code in order to explain in detail all the considerations behind the development of this script.

6.1.1. Script evolution

First of all, it is relevant to underline that the script which will be detailed in the following sections constitutes the most optimized version. However, it seems essential to introduce all the intermediary stages having led to this outcome. Indeed, even if the method, which has been defined from the project requirements, remains significantly the same from a version to another, the difference between two versions lies in the inclusion of problems appeared during the development of the previous version. That way by resolving these problems, related to automatization issues or to files layout issues for instance, it was possible to reach the most suitable version of the script.

6.1.1.1. First version

The first variant consisted of returning all the points defined in the requirements specification for one specific configuration, but in a coarse way. Indeed, in this version, there were no functions, the computations were executed consecutively; moreover everything was done for one data file, imported at the beginning of the script.

6.1.1.2. Second version

From that point, the next natural upgrade was to adapt the previous code to any configuration in order to expand the analysis to the whole of files. This ambition implied to organize the script by implementing various functions whose the main purpose was to return the different methods for a specific configuration defined from a set of parameters given as input, such as the name of the engine, the considered cylinder and the studied step, for instance.

6.1.1.3. Third version

Even if the second version of the program was perfectly working, a third variant had to be introduced in order to take into account the temperature analysis described in the paragraph 3.2.3. Furthermore, it is important to specify that, at this time, the temperature analysis was embedded to

the *BoreDistortion_Analysis* script, and it was only during the development of the sixth version that this part of the project was disconnected from the original file and then implemented in an independent script, notably for purposes of clarifying and distinguishing each different part of the project.

6.1.1.4. Fourth and fifth version

The third draft of the program was originally supposed to be final one. So from that statement, the next thing to do, before starting thinking about the graphical interfaces, was to validate the method, notably the Fourier transform. Actually, the first Fourier transform implemented was the one provided by a Python library and had to be checked once the entire script achieved. But it appeared, by comparing the results coming out of the program and those showed in a report written by AVL (obviously concerning the same model), that the Fourier transform within the program was obsolete. After several modifying attempts and two new versions, the good Fourier transform has been properly implemented in the program.

6.1.1.5. Sixth version

First of all, it is the ability of Abaqus to refer to its functionalities by means of Python command lines which initiated the elaboration of an automatic way to edit the reports. Effectively, the original ambition was to write a new Python script whose the mission would have been to produce all the required files once run within Abaqus' Python console. At this moment of the project, the last functioning edition of the program based its operation on a set of text files, generated from the Abaqus' interface, which gathered, on the same document, both the coordinates and the adequate parameter (radial displacement or temperature) of all the nodes located on the liners' inner surfaces. Unfortunately, it turned out that this method is one of the rare ones which does not have an equivalent in terms of Python command line. Then it was necessary to find a new process to edit the data reports. Nevertheless, the retrieval of nodes' coordinates being indispensable to the achievement of bore distortion analysis and the previous method being the only way to obtain these coordinates, the automatic generation of files has been reduced to a semi-automatic one. Actually, this new generative routine is based on a first files edition, manually accomplished, in order to create a few documents containing the coordinates of nodes for the different engine liners. It is relevant to specify that these coordinates are definitive, and do not require to be re-edited at each files generation, except in case of new engine or new mesh. Then it was feasible to build a Python script which will automatically generate the set of reports containing the missing information, in other terms the values of radial displacements or the temperature of nodes. Therefore, as the structure of the new reports differed from the previous ones, it was mandatory to develop a new variant of the program in order to adjust to these modifications.

Furthermore, for purposes of dealing with different models, a new parameter appeared during the development of the sixth version, whose the main interest is to select the folder where the files are stowed in.

6.1.1.6. Seventh version

The last version of the program did not involve a large amount of modifications and simply consisted of figuring out a way to compare the various results, obtained from the previous functions,

for two distinct engines, or at least two different models, in order to make a judgement about potential improvements regarding changes within the Abaqus models. Indeed, the analysis carried out by the program may only be complete if this last one affords to have a clear view on outcomes coming from two different models, for purposes of progressively reaching the most sustainable solution.

6.1.2. Data processing

6.1.2.1. Coordinates retrieval

The type of the files lightly mentioned in the previous sections is not directly exploitable by Python, notably due to the presence of undesired terms (symbols, words, etc.) and also useless data. Hence, in order to be able to use the data within the program, it was required to firstly remove every parasitic expression and then extract the values which constitute a genuine interest for this project. These points have been carried out through the functions “extract_coordinates” and “extract_def_data” in the script *BoreDistortion_Analysis_V7,* one working on coordinates and the other on radial displacements. The first technique implemented to achieve this task relies on the use of the ”replace” function, changing any disturbing terms by an “empty string”; the second one is based on a browsing and locating method (implemented by “for” loops and some “if” conditions) which consists of going through the file, then stowing the position of key terms into an index variable and finally reorganizing the data by extraction around these index.

6.1.2.2. Values distribution

It exists, as it is slightly described in the paragraph 6.1.1.5, two types of input files, one gathering the X, Y, Z coordinates of the considered liner, the other one grouping together either the radial displacements of the nodes or their temperature (in this case these files are about radial displacements). From that comment, it was necessary for both types of file, first and foremost to retrieve these pieces of information and secondly to hand out these values towards several lists specifically made to contain only one sort of data, meaning one list containing the X coordinates, another one the Y coordinates and so on. This stage has been introduced thanks to the functions “split_coordinates” and “extract_radial_def” by the use of a browsing and distributing method. Indeed, by observing the files produced by Abaqus, it was really easy to deduce the sequence employed by the report to organize the values. Then, by means of a “while” loop it was possible to primarily go through the entire file term by term and in a later step, by knowing the index of the picked out element and hence knowing its nature, to send the value in the appropriate list.

6.1.2.3. Data sorting

At the beginning of the script development, the files were supposed to be sorted and organized according to the height from the top of the liner to the bottom (or the opposite, what was expected was a logical classification). But the reality was completely different. Indeed, the values were following the nodes numbers which did not correspond with any convenient arrangement (the values were not gathered by height and they were spread into the document). From that conclusion, it was fundamental to manually sort these values according to their Z coordinates, in order to ease the achievement of the future functions. However, that was not the only issue which had to be solved. In fact, it appeared during the implementation of this method, that the Z coordinate of certain nodes,

which were supposed to be on the level, could differ a few (for instance a node may be located at an altitude of -16.1202 mm and another one at -16.1199 mm). Taking into account all these remarks, three functions have been created ("split_height", "gather_height", "sort_height") for purposes of respectively scattering the nodes according to their height, then getting together the nodes whose their height was approximately identical, and finally reorganizing all the lists with respect to the Z coordinates, from the top to the bottom.

6.1.3. Data applications

Once the coarse data, obtained from Abaqus, was refined by the methods described previously (that means the creation of the reorganized lists), it was possible to implement the functions having as objective to return all the requirements defined during the conception of the project plan.

6.1.3.1. Definition of paths

First of all, in order to ensure a proper understanding of this section it is important to explain what a "path" is. In this project, a path is considered as a succession of nodes along the inner periphery of the liner at a given height from the liner's top. Moreover because one of the project ambitions is to return the radial displacement along these paths, it was mandatory to find a way to produce these elements. So finally, by drawing from the definition, the idea was to characterize each node of the path by an angle going from 0 to 2π radians (or by a distance along the perimeter which implies to multiply the angle by the radius).

From this statement and from the coordinates, especially X and Y coordinates, it was feasible to design a function which returns an angle. Indeed, by definition of the tangent function, it is possible to express the angle as the arctangent of the ratio ordinate over abscissa (cf. the function "Angle"). Lastly, a function "create_angle_list" has been created in order to simply apply the previous method to the whole X and Y coordinates and that way to obtain a complete and sorted angle list.

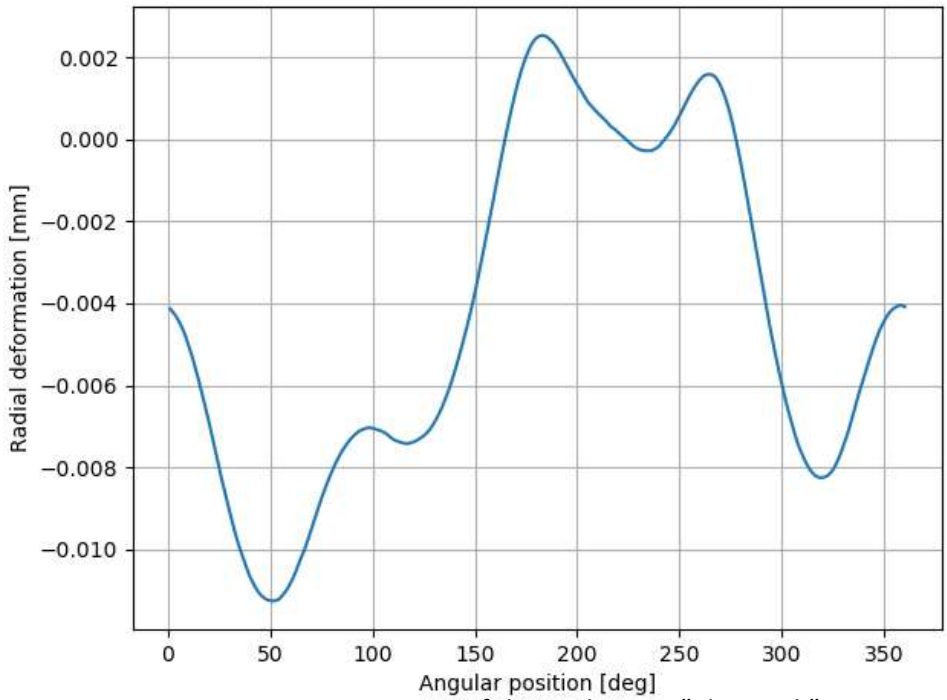


Figure 4 : Outcome of the application "plot_path"

Therefore, for purposes of satisfying the objective behind this part, which was about plotting the radial displacement of nodes along a path defined at a certain height, the function “plot_path” was established. The operation of this function is quite simple. Actually, this function will simply create both angle and radial displacement lists for the configuration specified through the various parameters. Then with respect to the height entered as input, the function will extract all the values related to this height, for finally return them in the form of a graph.

6.1.3.2. Generation of deformed profile

This part of the project did not raise too much issues notably due to its similarity with the function working on the radial distortions along a path. However, the main modification with respect to the previous method was the change of the graph’s coordinates system. Indeed, this time the radial distortions are no longer plotted along a path in a Cartesian coordinates system but around an axis in a cylindrical coordinates system. Hence, this new graphical representation induced to embed some points to the last method. That way, in addition to the creation of both angle and radial displacement lists for the specified configuration, a new list is created in which the real liner’s radius is computed. This entity represents the sum of the liner’s initial radius with the multiplication of a scale factor and the radial displacement of the considered point. It is essential to underline that the scale factor has been introduced for purposes of making the deformations more visual, deformations which are a few tens of micrometres big. All these elements may be illustrated under a Python form in the function “plot_def_profile”.

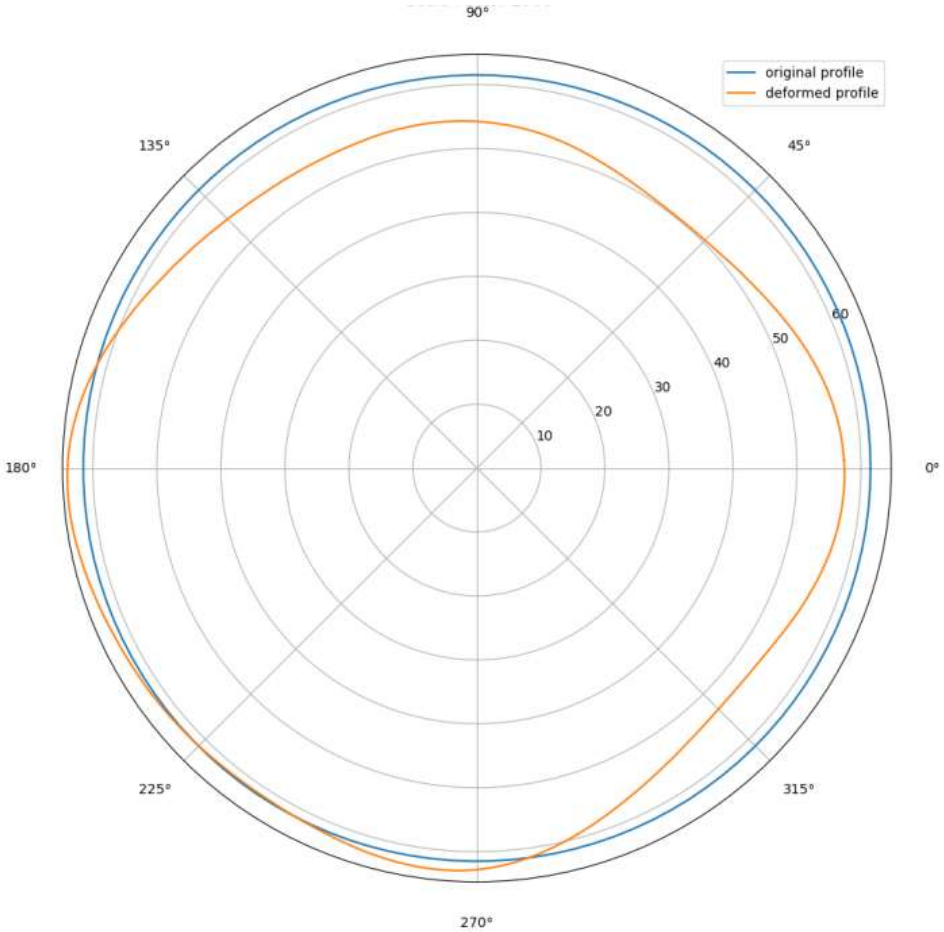


Figure 5 : Outcome of the application "plot_def_profile"

To conclude with this section, it is required to state that the function “plot_def_profile” relies on another function called “plot_initial_profile”, which tends, as its name indicates, to plot the undisturbed liner’s profile, in order to have a quick graphical comparison between the loaded liner and the intact one (which is provided by the superposition of the two graphs on the same figure).

6.1.3.3. Build of Fourier transform

The construction of a method computing the Fourier coefficients, based on the theory developed in the section 4, constituted the milestone of this project. Indeed, many requirements defined in the preamble of this assignment demanded a good computation of these coefficients. At this time of the work, several options could be executed, two options to be more accurate. The first one implied the use of a Fast Fourier Transform, which is an already implemented Fourier transform available in a Python library. The other one counted on the manual build of the Fourier Transform by using a sum of exponentials. Therefore from that point, the ambition was to firstly select one of these possibilities and then to check the accuracy of the results. To do so, a report written by AVL with regards to a complete study of the MX11 engine has been provided. In this report, an entire section has been dedicated to the bore distortion analysis, carried out from a certain model, which has been also supplied with the report. Thus the validation process consisted of writing two functions, each of them employing a different method in terms of Fourier Transform, to secondly apply them to a set of data files corresponding to the cases studied by AVL. That way, it was possible to compare the several samples of Fourier coefficients returned by the different functions with those showed in the report. At the sight of the results, it was feasible to take note of two things; on one hand the results returned by the two functions did not converge towards the same values, and on the other hand, these values were completely away from the ones described in the report. From these observations, it was decided, on one side, to focus only on the hand-built function, meaning without the Fast Fourier Transform which had the disadvantage not to be adaptable, and on the other side, to define a validation method to align the outcomes of this function towards the correct values (which were supposed to be the ones illustrated in the AVL report).

The validation attempt has been based on the idea to consider the problem in an opposite direction. In other terms, this time the Fourier coefficients were not viewed as an expected output but as a problem input. Actually, the plan consisted of rebuilding a deformed circle (equivalent to a deformed liner’s profile at a certain height) from the Fourier coefficients, to, in a second time, apply the Fourier Transform method to the data list related to the freshly rebuilt profile, to finally conclude with a comparison between the Fourier coefficients introduced as input and the ones returned by the computational function.

The first stage of this plan required to implement a method employing the reverse Fourier Transform for purposes of determining the values of radial displacements from Fourier coefficients and dephasing angles. According to the section 4, it exists two expressions to carry out such a thing:

$$x_n = \sum_{k=0}^{N-1} F_c[k].\cos(k.\theta_n + \varphi[k])$$

Or

$$x_n = \sum_{k=0}^{N-1} F_c[k].\exp[\,\mathrm{j}\,.(k.\theta_n + \varphi[k])]$$

In these mathematical formulas, F_c and φ represent respectively the Fourier coefficients and the dephasing angles whereas θ_n constitutes a list of angles arbitrarily created in order to simulate the distribution of the nodes along the periphery of the liner. Of course, the entire verification process depended on the proper elaboration of this reverse Fourier transform, which naturally implied to approve the construction of this method. In that respect, the calibration of the function was made by relying on the theory depicted in the section 4 of this report. Actually, by knowing the shape of the returned profile when only one Fourier coefficient is set to a non-null value and all the others to 0, what is described on the figure 3, it was possible to converge both methods to the correct form.

Once these two methods fully approved, it was necessary to resume the Fourier Transform developed at the beginning of this section in order to apply it to the radial displacement lists freshly computed. Finally, the comparison between the coefficients entered as inputs and the ones obtained by computations has afforded to reach two working configurations. Indeed, it appeared that if the method used to compute the radial displacements relies on a sum of exponentials then the respective Fourier Transform must work with a sum of exponentials without any multiplying factor; whereas if the sum of cosines is chosen, then the Fourier Transform must also use a sum of exponentials but this time with a factor 2 in front of the sum. The addition of a factor 2 within the second method may be explained by the lack of sine terms in the sum. Indeed, originally the Fourier Transform, forward or backward, is created from complex exponential, gathering by definition a real cosine and a sine attached to the imaginary part. However in this case, as the signal is purely real, the imaginary terms (coefficients in front of the sine) are truly close to zero and that way can be neglected. That is why, the second method, using only the sum of cosine, converges towards the good graphical solution as well. Nevertheless, as the sum of cosine constitutes only half of the information, the method requires a factor 2 in order to compensate this lack of data.

As the two possibilities are correct, it has been retained to use the method using the sum of cosine, notably because this expression is often employed in bore distortion computation.

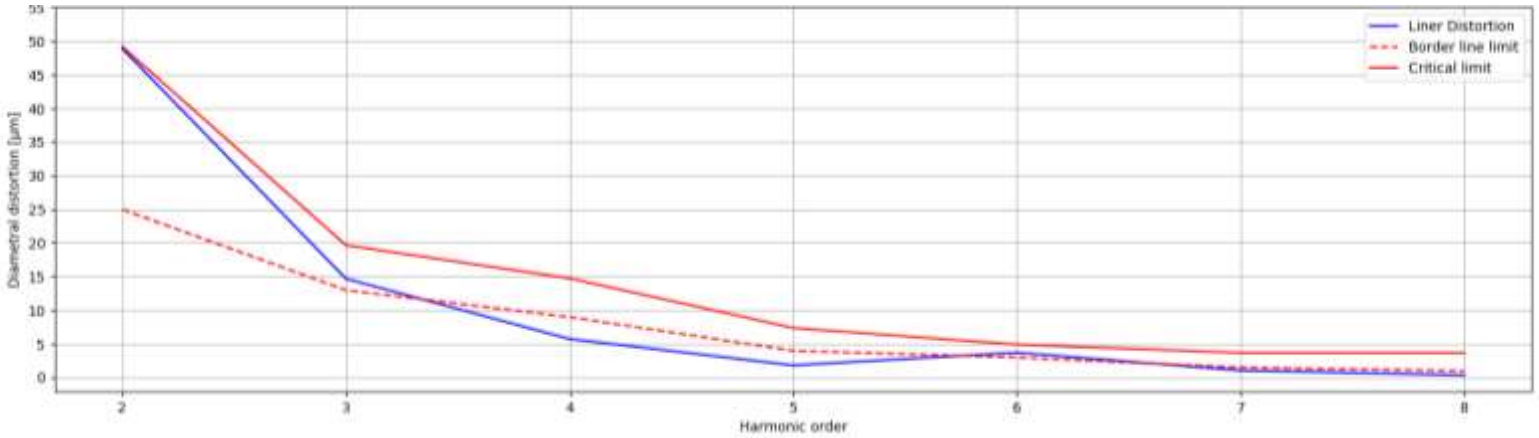
Just before ending with the section about the elaboration of a Fourier Transform, it is important to underline that despite the truthfulness of the selected method, which according to the check process returned the good results, it still remained a divergence between the outcomes depicted by the AVL report and those coming from the program. Indeed, the genuine results, the ones from AVL, seemed to be approximately twice bigger than the computed outcomes. As the Fourier Transform was no longer a potential reason to explain this divergence, it was mandatory to keep working on the rest of the project.

6.1.3.4. Application of Fourier transform

In accordance with the requirements specification defined at the preamble of the project, two outputs of the program are demanding the application of Fourier Transform.

The first one, achieved by the function called “plot_distortion_graph”, ensues from the ambition to compare the diametrical distortion values, from the second order harmonic to the eighth, with

some boundary limits, a borderline limit and a critical one. That way, the main purpose of this method is to graphically check if the values always remain inferior to the borderline limit or at least inferior to the critical limit which defines the maximal tolerated values. The computation of these diametrical distortions will be carried out for a specific configuration, defined by an engine, a cylinder, a step, a height and a folder. It is also significant to underline that initially the unit of these elements was the millimetre but it was decided to convert them into micrometres, for the sake of clarity. However, the problem raised during the development of this method was the definition of the diametrical distortions. Actually, at the beginning, this element was considered to be a Fourier coefficient, but it appeared, as slightly described in the previous section, the results obtained were lower than the targeted ones, more or less twice lower. After the rereading of all the documents retained during the bibliographical research, a potential justification has made its appearance. Indeed, within a SAE report, a new parameter, called U_{max} , is mentioned and is defined as two times the Fourier coefficient. From that discovery, it was decided to ask the people who wrote the AVL report if that was the reason of the results divergence. Finally, they confirmed this hypothesis and explained that the Fourier coefficient describes the radial distortion whereas the graph is about the diametrical distortion, so two times the Fourier coefficients. Once this consideration taken into account, both of the outcomes finally converge towards approximately the same value.



	order 2	order 3	order 4	order 5	order 6	order 7	order 8
Critical limit [µm]	49.2	19.68	14.76	7.38	4.92	3.69	3.69
Border line limit [µm]	25.0	13.0	9.0	4.0	3.0	1.5	1.0
Diametral distortion [µm]	48.8924545502	14.6978877386	5.739331030488	1.827914173028	3.710127907	1.060458997831	0.3754813934479

Figure 6 : Outcome of the application "plot_distortion_graph"

The second method essentially draws from the structure implemented in the "plot_distortion_graph" function with only the form of the output as difference. In fact, unlike the previous function which has to be told about the considered height before computing its Fourier coefficients, this application will carry out the computations for the entire liner. That way, this method aims to return the diametrical distortions from the second order harmonic to the eighth and this for every height composing the liner. The mechanism behind this function is simply based on the successive computations of Fourier coefficients which will be in a second time distributed into several lists containing only the values for one specific harmonic order. To conclude with this function, it is relevant to indicate that this method affords a global view of the diametrical distortion analysis (for one engine, one cylinder and one step), unfortunately at the expense of a calculation time quite significant.

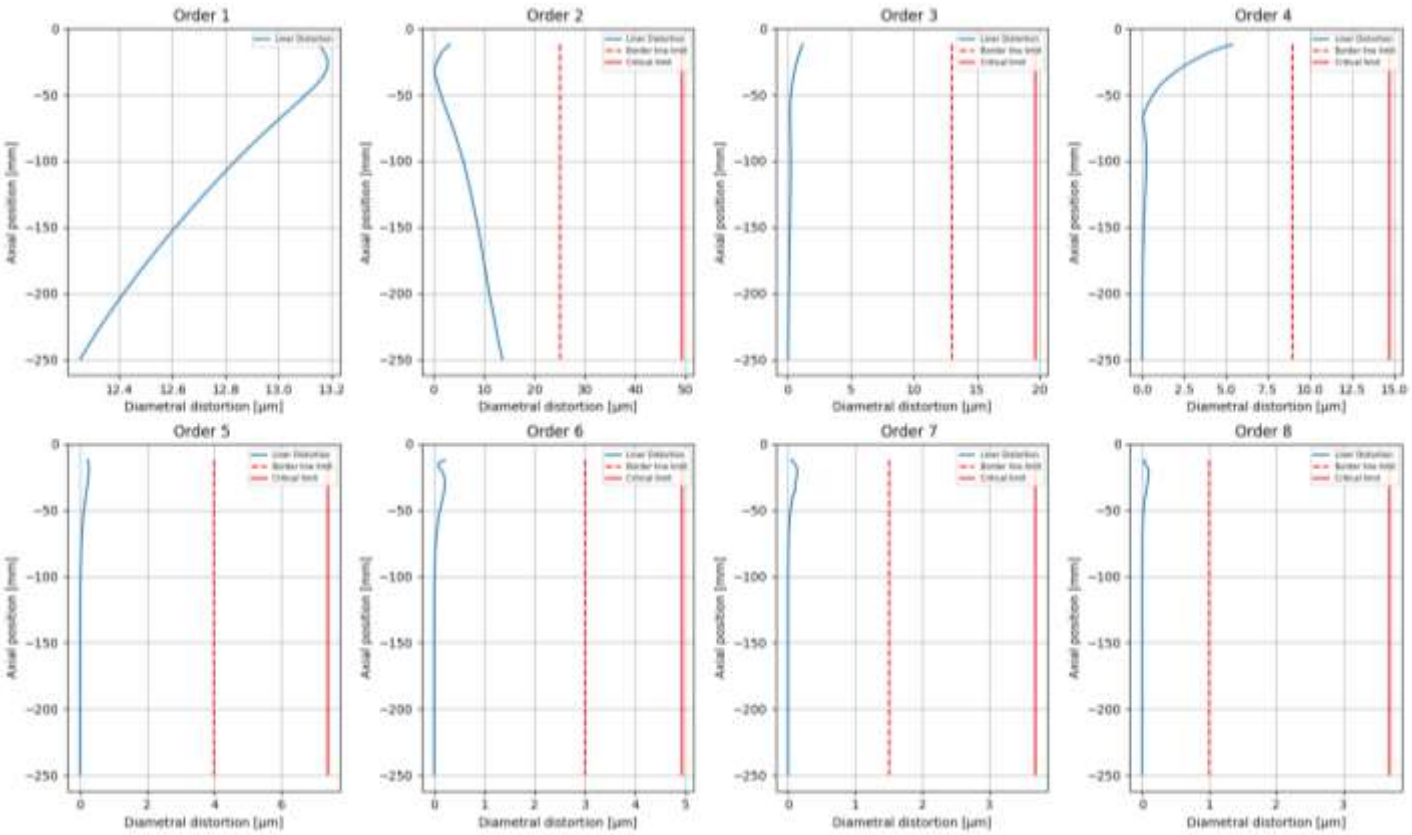


Figure 7 : Outcome of the application "plot_distortion_by_order"

Thus, the best way to carry out a complete bore distortion analysis could be to firstly display the diametrical distortions for the entire liner, thanks to the function “plot_distortion_by_order” and by doing so to spot the most critical cases, in order to, in a second phase, analyse these situations more precisely with the function “plot_distortion_graph”.

6.1.3.5. Creation of liner’s deformed shape

As the function built in the section 6.1.3.2 already affords to have an overview of the liner deformations, the last aspect the script had to deal with, was about finding a new way to describe the shape of the loaded liner. The retained approach lied in considering the radial deformations not along a circular path but this time along the Z-axis, which corresponds to the virtual cut as illustrated in the figure below.

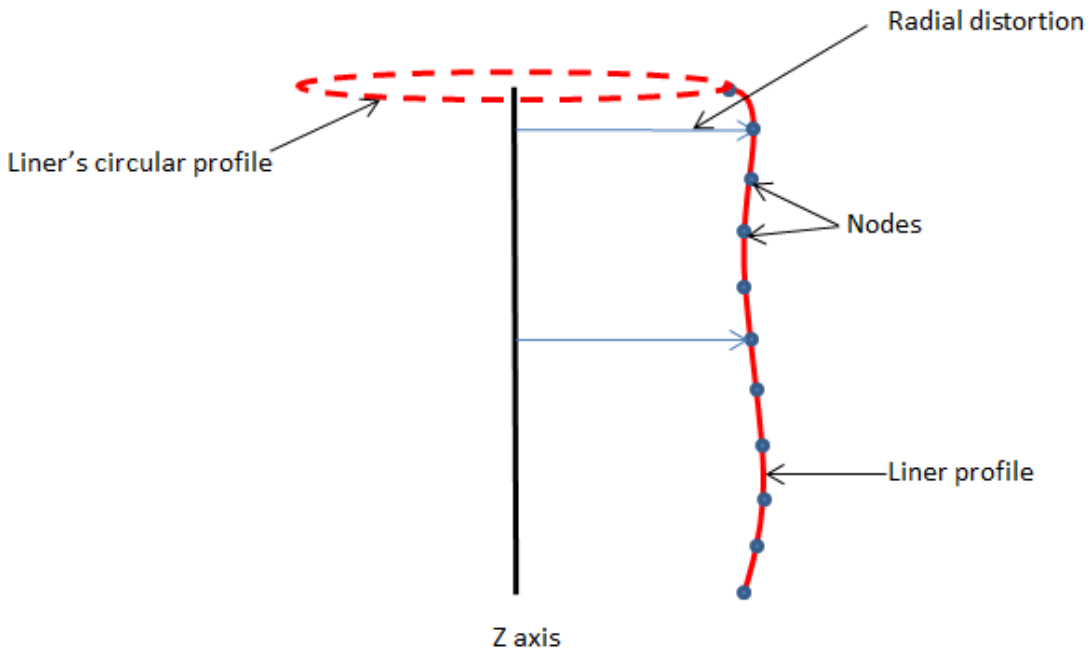


Figure 8 : Summary schema of the radial distortion along the Z axis

Nevertheless, the achievement of such an application required a preliminary stage. Indeed, as it would have been really laborious to return every deformed profile located all around the Z axis, it was mandatory to edit a method aiming to select only a few cases. The process retained to do so relied on the method previously developed which afforded to generate an angle list from the X and Y coordinates. That way, by creating this list, and hence by knowing the angular position of each node, it was possible to identify all the nodes located at a certain angle. Finally, it was agreed to focus the study on all angles from 0 degrees to 325 degrees with a 45 degrees increment between each of them. This selection process is carried out by the function called "select_angle" within the script *BoreDistortion_Analysis_V7.*

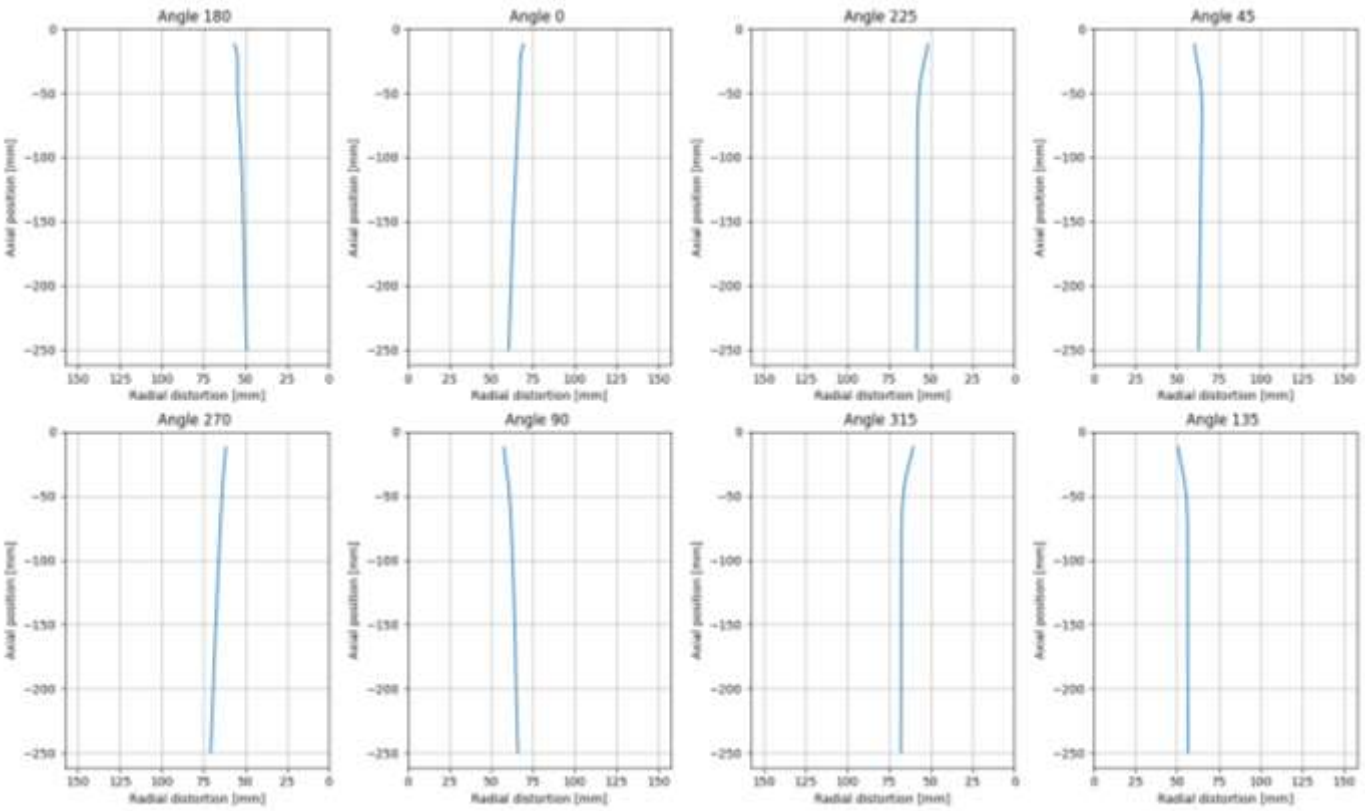


Figure 9 : Outcome of the application "plot_cut_profile"

Once this auxiliary function created, it was possible to concentrate on the elaboration of the application itself, called “plot_cut_profile”. Indeed, thanks to the previous method, the radial displacements of the nodes defined by the sample angles (angles described in the previous paragraph) may be extracted and subsequently distributed in different lists associated with a specific angle value. Obviously, this calculation routine must be executed for every height of the liner in order to obtain the radial distortions all along the Z axis. From there, it is only needed to generate the various charts resulting from these data lists.

6.2. Temperature_Analysis

Originally embedded within the *BoreDistortion_Analysis* script, it was decided during the elaboration of the sixth program version to work on the temperature analysis through an independent file, for sake of clarity. Ergo, this aspect of the project has been achieved in a script, structurally similar to the previous one but yet with a few divergences at some points.

6.2.1. Data processing

As for the bore distortion analysis program, before going forward in the genuine application, it is required to work on the data itself. Once again, as for the previous method, the data needed to follow the same three stages process consisting of values extraction, values distribution and finally values sorting, before being fully exploitable by the coming application. However, the values extraction stage of the temperature analysis slightly differed from the former script. Actually, the thermal study expected in this project must not only focus on the temperature of liner’s nodes but also on the temperature of the fire ring’s nodes. Therefore this inclusion of new nodes within the files edited by Abaqus required to modify the method extracting the values from these files, notably due to a change of length but also due to the presence of new parasitic expressions.

6.2.2. Application

The only application of this thermal section, called “plot_wall_temperature_complete”, consists of displaying the temperature of liner’s wall throughout the Z axis. Such a request raised issues already encountered in the part related to the method plotting the radial displacements along the Z-axis, and hence was treated the exact same way than the previous function. Firstly, as mentioned in the section 6.1.3.5, working along the Z-axis requires to retain only a few cases in order to save time and memory space, a problem which can be overcome by using the function “select_angle” created earlier in the project. Then, as in the application “plot_cut_profile”, some pieces of information will be extracted thanks to the angle selection method, obviously this time it is about temperature values. Finally, this process will be once again repeated for each of the liner and fire ring heights in order to produce all the data lists, and hence to return the expected graphs.

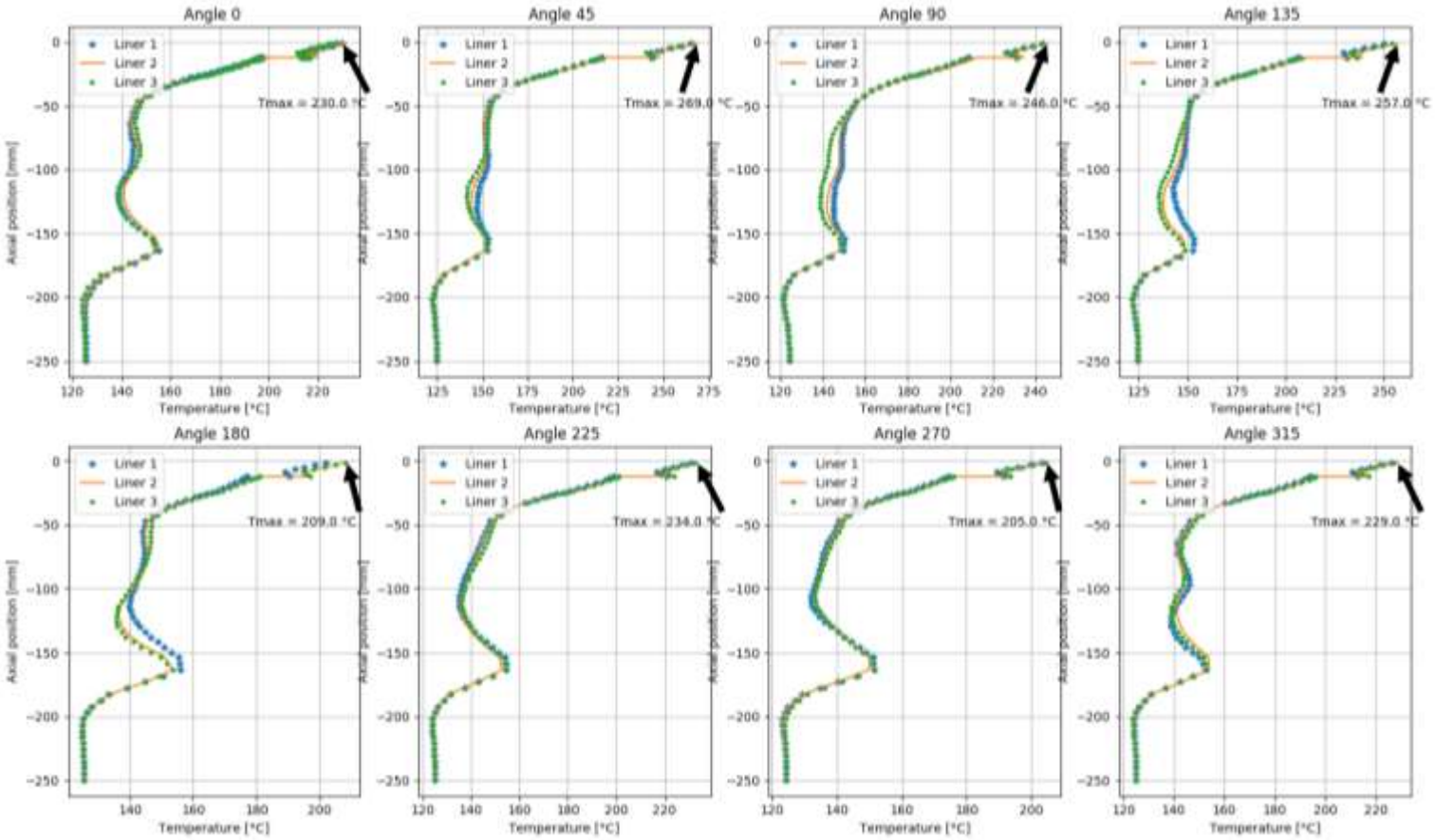


Figure 10 : Outcome of the application "plot_wall_temperature_complete"

6.3. Files generation program

The achievement of the main applications, carried out throughout the two previous scripts, has afforded to focus on the third level ambition of this project which consisted of generating the files in a way as automatic as possible. Indeed, the files generation, which was until that moment still manual, had to be rebuilt in such a way that the future user would have the least handlings to do. Before going forward in the explanation, it is significant to say that two files are involved in this section, one generating the files for the temperature analysis and the other for the radial distortion analysis.

6.3.1. Operational section

The development of this script was possible essentially thanks to the “Macro” functionality of Abaqus. Actually, this process affords to translate any action done from the Abaqus interface into its equivalent Python code. By relying on this method, it was feasible to manually carry out the entire process, leading to the generation of the files and this through the Abaqus window, and hence in a second time to obtain the analogous Python code. The process mentioned above consists of, first of all, isolating one of the liners in the viewport, afterwards, creating a cylindrical coordinates system for the selected liner (either located at the Fire ring shoulder for radial distortion analysis, or at the liner’s top for the temperature analysis). From that point, a display group containing all the nodes of the liner’s inner surface has to be established. Then this display group has to be isolated within the viewport, due to the fact that the FieldOutput report method, which will be used to generate the

files, works on the active view (that implies if the display group is not isolated, the report will be written for the entire liner). Finally, as slightly explained previously, a report concerning either the radial displacement along the first axis of the early defined coordinates system or concerning the temperature is edited. The repeat of this process for every liner affords to obtain all the Python code required to implement the files generation functions. Thus, after having removed the useless pieces of information (related to viewport manipulation for instance) and also after having modified some details within the code lines, it was possible to produce four functions, two about the radial distortion analysis for both MX11 and MX13 engines, and two others about temperature for the same engines.

6.3.2. Various upgrades

Despite the proper operation of original functions described in the previous section, some ideas of potential improvements appeared after a few thoughts. So in order to optimize as much as possible the program, it was decided to execute these suggestions.

6.3.2.1. Mode of generation

Throughout the check routine of the previous code, it was observed that the files generation for the entire set of configurations (in other terms for every step) was taking some time, about a few tens of seconds. Moreover, some of these steps do not offer a great interest within the analysis (for instance the step “Cold engine” for the thermal analysis to only mention that one) or may be redundant with respect to others. In that way, it was chosen to implement a manual way of generating the files which will afford the user to select only the cylinders and steps he wants to look into. However, it is quite relevant to underline that the files once generated are saved in a permanent way, there is no need to create them every time, that is why it could be better to generate all the files once and for all, despite the pointlessness of some. Nevertheless, the point behind this adaptation lied in giving the opportunity to the user to generate the files as he wants to.

6.3.2.2. Model folder handling

It was known at this moment that the program should be able to work for two engines described from several Abaqus models. Such a statement induced to find a way to dissociate the files with respect to the model used to generate them, in order to avoid any misguided removal for instance. Two methods were identified as a solution to this problem. The first one was about distinguishing the files from their name, by adding a reference in it and this in the main generator function. But this solution would have induced the addition to several “if” conditions within the function’s body. The second solution, the one retained, consisted of creating a folder, within the workspace, for each model, and hence saving the files inside the good folder. This solution has been implemented notably because it was the simplest one, but also the cleanest in terms of files storage.

6.4. Graphical interfaces

The main purpose of graphical interfaces is about providing a clear and direct access to the several applications gathered within the program, by simply clicking on a button instead of entering a complex call line in the Python interpreter. In that way, it was necessary to develop a graphical interface for each Python script demanding an interaction with the user.

6.4.1. Program interface

The first interface to be achieved is bound to the main body of the program, and because of that must afford to fully exploit all the applications executed within the equivalent Python script, and this in the simplest possible way. From there, the mission was about handling various functionalities made available by a Python library in charge of producing graphical interfaces, called “Tkinter”, in order to provide the most convenient interface in terms of settings and access to applications. In that respect, it has been decided to select both engine and folder name via a drop-down list, these entities having the main advantage to only show the existing cases. As the other parameters constituting the rest of the configuration, which means the selection of cylinder, step and height, are based on an indexing system (positive integer), it appeared quite natural to refer to them by using integer inputs which are basically blank boxes inside which the user may enter the number he wants. Finally, once the setting part achieved, it has been chosen to merely access the different applications implemented in both *BoreDistortion_Analysis* and *Temperature_Analysis* scripts, through a set of buttons.

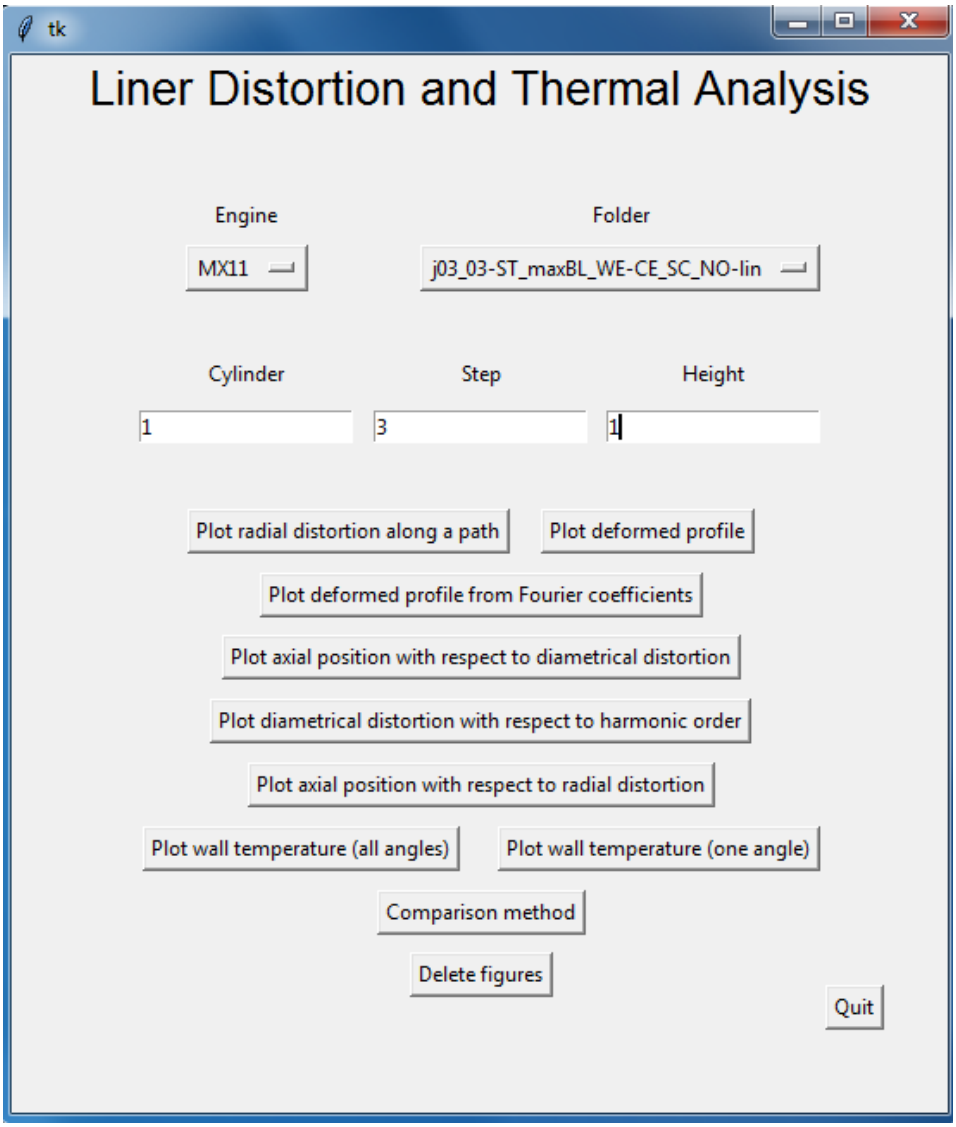


Figure 11 : Preview of the program's main interface

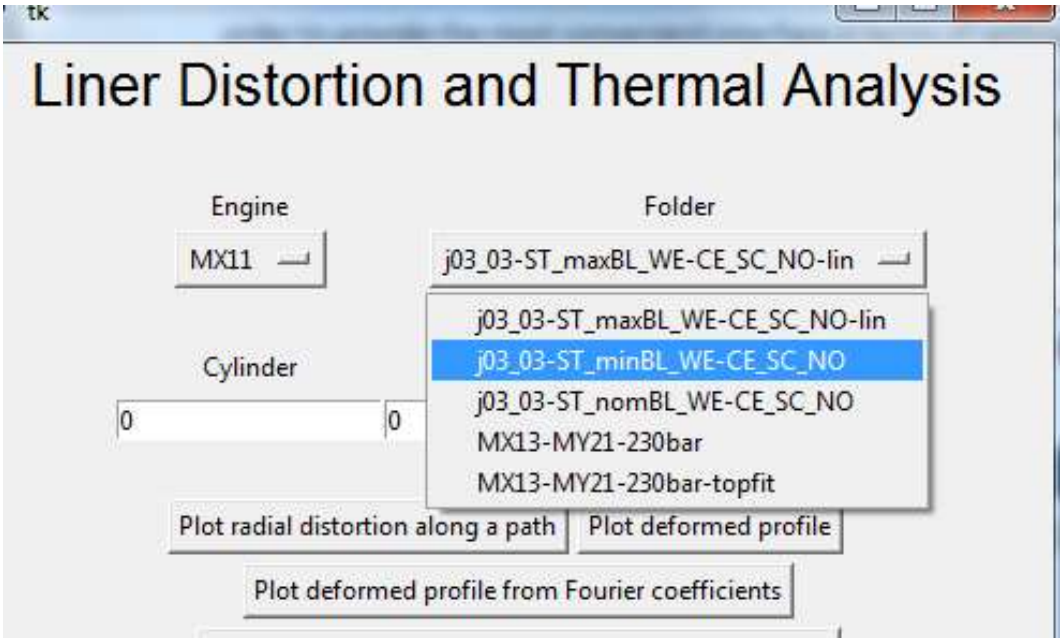


Figure 12 : Preview of the folder's drop-down list

6.4.2. Files generation interface

Despite the fact that this interface had to be developed with the ambition to be run from Abaqus, this last one does not differ from the previous one in terms of structure. Actually, once again the idea behind this interface was about accessing all the methods implemented within the files generation scripts by means of drop-down lists and buttons. In that way, through these entities, it will be possible to generate either the distortion files or the temperature files for one specific model, but also to create an explicative note (cf. the section 6.5) or finally to launch the external Python interpreter.

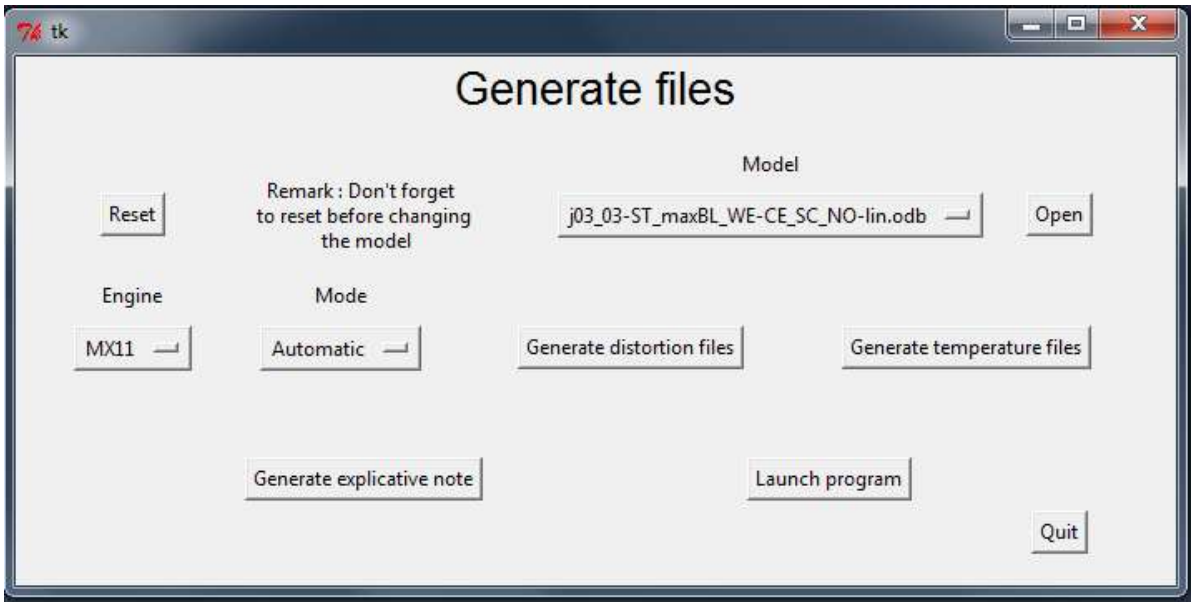


Figure 13 : Preview of the files generation interface

6.4.3. Comparative analysis interface

The graphical interface connected to the comparative analysis of two engines, depicted in the paragraph 6.1.1.6, does not derogate from the rule of offering a clear and mere access to the different applications which must be implemented within this section. In fact, this graphical interface does not differ from the one done for the main program. Indeed, the idea was to reproduce the same functionalities, but this time with two set of entities (drop-down lists, integer inputs, etc.) to set up the configurations of both engines (folder, engine, cylinder, step, and height). Moreover, it is also significant to underline that some applications, initially available within the *Program_interface* interface, are not offered by this comparative interface because they are considered as non-relevant in the frame of the comparative analysis.

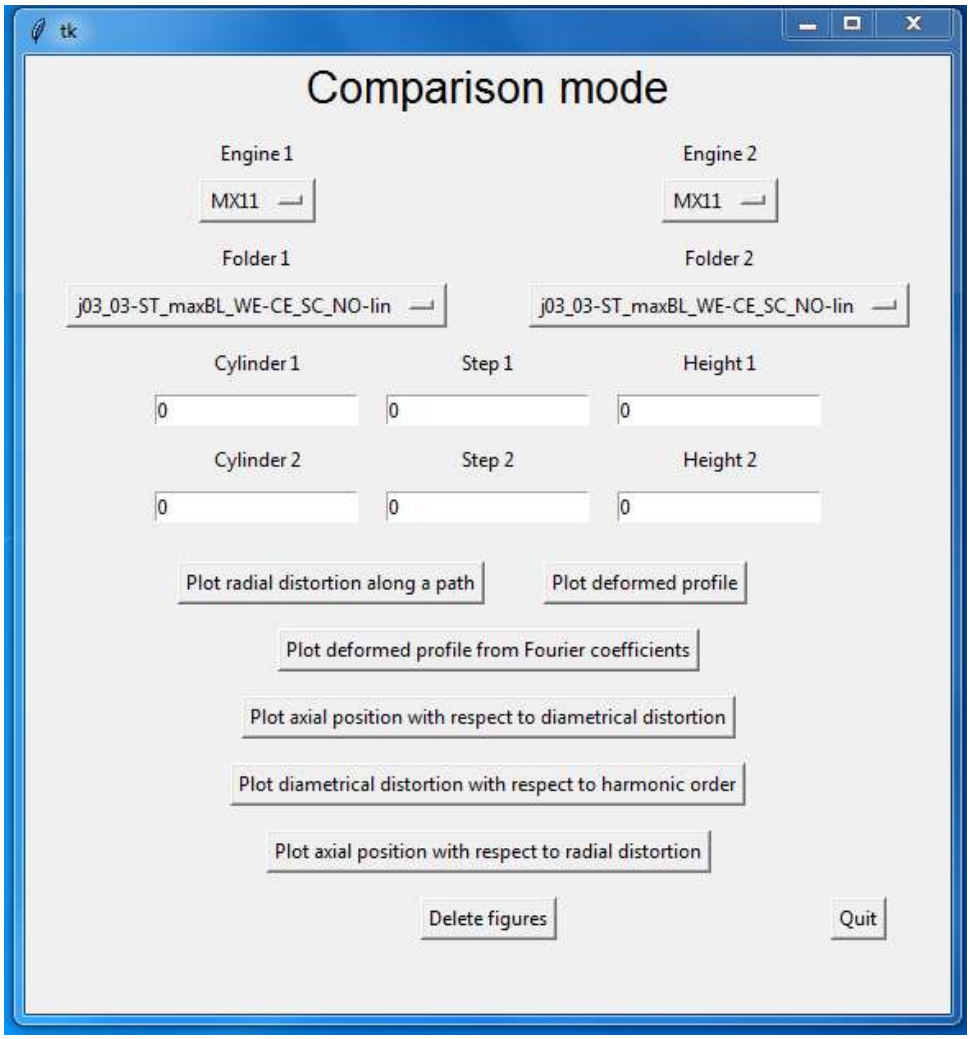


Figure 14 : Preview of the comparative window

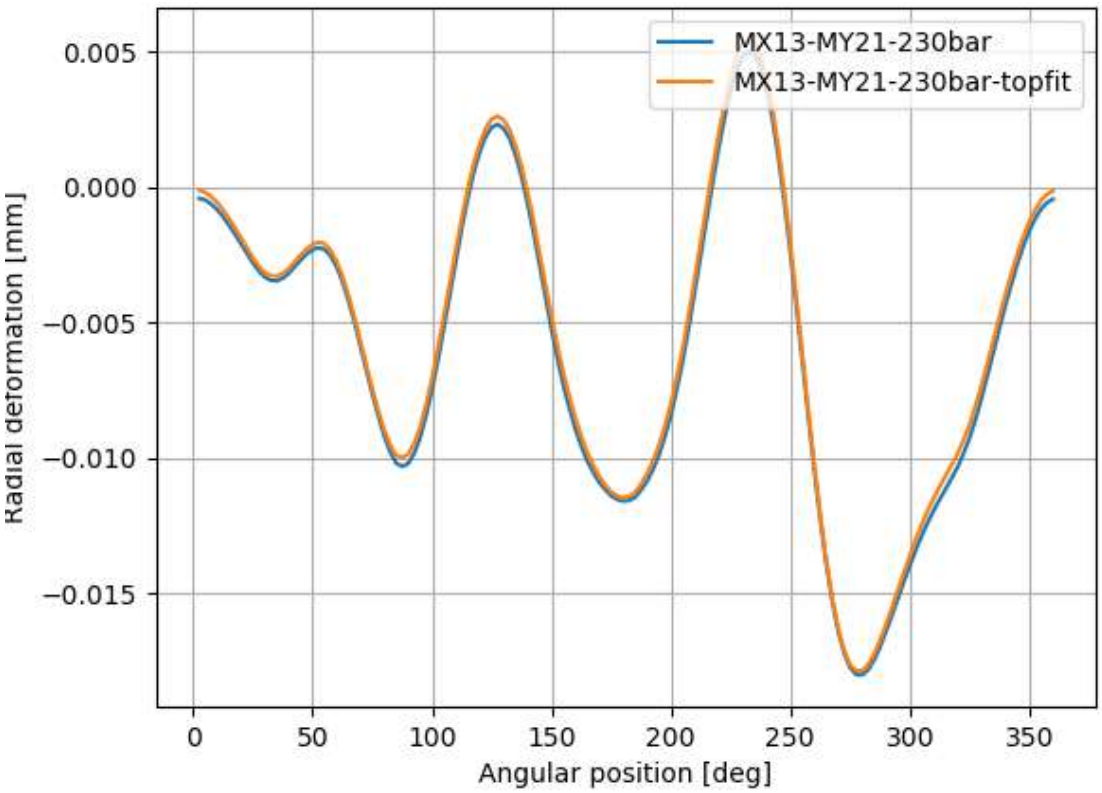


Figure 15 : Outcome of the plot path comparative application

6.5. Explicative note

The selection of the data set which the several applications have to work with is based on an indexing system, whether for the liner, the step or the height. Even if the indexing is not a big deal and remains truly simple, it may be possible that the user struggles to get through, at least during the first manipulations. Therefore, in order to ensure as much as possible the proper operation of the program, it was decided to implement an explicative note whose the main purpose is to enumerate all the steps but also all the heights and then assign to them a corresponding index. To conclude with this section, it may be relevant to specify that the different steps come from Abaqus models whereas the heights are extracted from the files gathering the coordinates of the liners' nodes, that is why some of the functions defined in the *BoreDistortion_Analysis* script are also present in the scrip *Explanation_note*.

6.6. Link between Pythons

As described in the section 5.2, it was mandatory to execute the program through two different Python interpreters, the one embedded within Abaqus and an external one. Hence, the final ambition of the project was about bringing together these two subsections into one single entity. The idea was then to create a bond between the two interfaces by calling one interface from the other. Fortunately, it exists a method already implemented in Python (called "subprocess.Popen") which affords to open a file from a specific software. However, for purposes of applying this method, it was necessary to update the Python environment. Indeed, it is important to underline that all Python interpreters are working with a specific environment, place gathering all the documents related to an

interpreter (tools, scripts, etc.), which is different from an interpreter to another. In that respect, the external Python console cannot be operational if the last active environment is the one in connection with the Abaqus' Python console. Consequently, after having redefined the working environment, it was feasible to apply the method "subprocess.Popen" in order to launch the program interface from the files generation interface. Nevertheless, even if the technique presented above is enough in the case of two interpreters using the same version of Python, a problem may occur when the versions differ (which turns out to be true in this project). In fact, the issue lies in the difference of the TCL library (library called for the interface generation) between Python 2 (Abaqus) and Python 3 (Canopy, interpreter used for the project). More precisely, the problem is due to the fact that when the subprocess method is used to open the second Python, there is a transfer of information between the two Pythons. That way, the TCL library imported in Abaqus will remain within the external interpreter, which will pose a problem when Python 3 will be attempting to import the library. Hence, in order to ensure the good operation of the program, this TCL library has to be reloaded before running the program interface. Taking into account all these remarks, a Python script called *Bridge_file* was executed in order to connect the two parts composing the program.

Conclusion

After several months of labour it is possible to draw conclusions about the outcomes of this end of studies' project. In that respect, a truthful and unbiased review of the final program affords to establish a contrasted balance-sheet of the additional module. Actually, it appears that the various applications provided by the main section of the program are perfectly operational and graphically return all the pieces of information needed to precisely analyse the bore distortion. Nevertheless, the fundamental defect of the program lies in the generation of files, more specifically in the automatic generation of files embedded within the module. As a matter of fact, even if the creation of the files may still be done manually, which represents an effective but quite long process, the self-generated method implemented in the corresponding graphical interface, yet working and fast, tends to freeze Abaqus notably because of the amount of reports which has to be edited from it. In the end, this complication make the automatic generation quite laborious needing sometimes to restart Abaqus several times to complete the production of the data documents.

After having brought to light the essential drawback of the program in the previous paragraph, it sounds pretty obvious that the main improvement of the module lies in fluidizing the automatic files' generation by preventing Abaqus from crashing. However, a quite thorough investigation has led to the following conclusion that, either this problem cannot be solved or it requires really extensive knowledge in programming.

Despite the inconvenience depicted above, the program globally performs without any other problems, and consequently will afford DAF Trucks to execute a correct bore distortion analysis to their set of engines. In this way, the program designed within this project successfully satisfies all the requirements expected by the supervisor of this project.

Bibliography

Name : *Analytical and Empirical for Optimization of Cylinder Liner Bore Distortion*
Author : Franz Maassen, Franz Koch, Markus Schwaderlapp, Timo Ortjohann, Jurgen Dohmen
Year of publication : 2001

Name : *Measurement of Bore Distortion in a Firing Engine*
Author : Lawrence E. Bird, Robert M. Gartside
Year of publication : 2002

Name : *Application to Engine Development of Friction Analysis by Piston Secondary Motion Simulation in Consideration of Cylinder Block Bore Distortion*
Author : Kenji Sato, Kinya Fujii, Makoto Ito, Shinsuke Koda
Year of publication : 2006

Name : *The GOETZE Cylinder Distortion Measurement System and the Possibilities of Reducing Cylinder Distortions*
Author : Klaus Loenne, Ron Ziemba
Year of publication : 1988

Name : *Effect of Cylinder Bore Out-of-Roundness on Piston Ring Rotation and Engine Oil Consumption*
Author : Eric W. Schneider, Daniel H. Blossfeld, Donald C. Lechman, Robert F. Hill, Richard F. Reising, John E. Brevick
Year of publication : 1993

Name : *Analysis of Cylinder Bore Distortion During Engine Operation*
Author : Shizuo Abe, Makoto Suzuki
Year of publication : 1995

Name : *DAF MX11 MY2021*
Author : Tamas Szabo, Balint Policza
Year of publication : 2017

Name : *Analysis of Distortions of Cylinders and Conformability of Position Rings*
Author : Dunaevsky
Year of publication : 1990

Appendix

BoreDistortion_Analysis_V7 script

```
import math as m
import os
import numpy as np
import time

dirpath = "C:/Users/A-Damien.Gode/Documents/Distortion
program/Analysis"

os.chdir(dirpath)

# In order to properly run the following program, it is needed to put
in the working folder the 6 coordinates documents

# As the data extracted from Abaqus are under .txt format, it is
required to reorganize these files in order to make them exploitable by
Python
# The following function is made to delete every parasitizing elements
in these files
def extract_coordinates(engine,cylinder):
    os.chdir(dirpath)
    if engine == 'MX11':
        doc=open('MX11_cyl%s_Coordinates.txt'%cylinder,'r')
        doc=doc.read()
        doc=doc.replace("\n","")
        doc=doc.replace("LIN-1","")
        doc=doc.split()
        Coor_table=[]
        index=0
        index1=0
        for i in range(len(doc)):
            if doc[i]=='Part':
                index=i
            if doc[i]=='-----------------------------------------------
-------------------------------------------------------------------
---':
                index1=i
        Coor_table=doc[index1+1:index]
        return(Coor_table)
    if engine == 'MX13':
        doc=open('MX13_cyl%s_Coordinates.txt'%cylinder,'r')
        doc=doc.read()
        doc=doc.replace("\n","")
        doc=doc.replace("LINER-1","")
        doc=doc.split()
        Coor_table=[]
        index=0
        index1=0
        for i in range(len(doc)):
            if doc[i]=='Part':
                index=i
```

```
            if doc[i]=='-------------------------------------------------
-------------------------------------------------------------------------
---':
                index1=i
        Coor_table=doc[index1+1:index]
        return(Coor_table)

# The purpose of this function is to split these values and to stow
them in respective list
def split_coordinates(engine,cylinder):
    Coor_table=extract_coordinates(engine,cylinder)
    X_coor=[]
    Y_coor=[]
    Z_coor=[]
    count=1
    X_coor.append(float(Coor_table[1]))
    Y_coor.append(float(Coor_table[2]))
    Z_coor.append(float(Coor_table[3]))
    while 1+7*count<len(Coor_table):
        X_coor.append(float(Coor_table[1+7*count]))
        Y_coor.append(float(Coor_table[2+7*count]))
        Z_coor.append(float(Coor_table[3+7*count]))
        count+=1
    return(X_coor,Y_coor,Z_coor)

# This time as the radial deformations are not gathered under the same
file, it was required to implement two new functions to extract these
deformations
def extract_def_data(engine,cylinder,step,folder_name):
    os.chdir(dirpath+"/%s"%folder_name)
    doc=open('%s_cyl%s_step%s.txt'%(engine,cylinder,step),'r')
    doc=doc.read()
    doc=doc.split()
    radial_def_table=[]
    index=0
    for i in range(len(doc)):
        if doc[i]=='--------------------------------':
            index=i
    radial_def_table=doc[index+1:len(doc)]
    return(radial_def_table)

def extract_radial_def(engine,cylinder,step,folder_name):
    data_table=extract_def_data(engine,cylinder,step,folder_name)
    radial_def=[]
    count=1
    radial_def.append(float(data_table[1]))
    while 2*count<len(data_table):
        radial_def.append(float(data_table[2*count+1]))
        count+=1
    return(radial_def)


# This function is done to return the diameter of engine
def diameter(engine):
    if engine=='MX13':
        diameter=130
    if engine=='MX11':
        diameter=123
```

```
    return(diameter)

# This function is done to return the coordinates of the cylinder's
center
def center(X_coor,Y_coor):
    center=0
    max_X=max(X_coor)
    min_X=min(X_coor)
    max_Y=max(Y_coor)
    min_Y=min(Y_coor)
    center=((max_X+min_X)/2,(max_Y+min_Y)/2)
    return(center)

# simple function to test if a value is in a list
def belong(List,val):
    for i in range(len(List)):
        if List[i]==val:
            return(True)
    return(False)

# For the proper operation of the following functions and in order to
plot all the graphs, it is mandatory to regroup the nodes by their
height from the top
def split_height(engine,cylinder,step):
    Z_coor=split_coordinates(engine,cylinder)[2]
    height=[]
    already=[]
    ind_list=[]
    index=0
    while index<len(Z_coor):
        value=Z_coor[index]
        if belong(already,value)==True:
            index+=1
        else:
            ind_list=[]
            for i in range(len(Z_coor)):
                if Z_coor[i]==value:
                    ind_list.append(i)
            height.append((value,ind_list))
            already.append(value)
            index+=1
    return(height)

# As some nodes, supposed to be on the same height, have not exactly
the same Z coordinate (0.1 micrometers difference in average), it was
needed to gather them under a same value
# That way it was possible to get the same number of nodes for each
height
def gather_height(engine,cylinder,step):
    height=split_height(engine,cylinder,step)
    comp_val=0
    index=0
    already=[]
    H=[]
    index_list=[]
    while index<len(height):
        comp_val=height[index][0]
        if belong(already,round(comp_val,1))==True:
```

```
                index+=1
            else:
                index_list=[]
                for i in range (index,len(height)):
                    if round(height[i][0],1)==round(comp_val,1):
                        index_list+=height[i][1]
                H.append((comp_val,index_list))
                already.append(round(comp_val,1))
                index+=1
        return(H)

# Then once the previous step done, it was required to reorder all the
list according to the height, from the top to the bottom
def sort_height(engine,cylinder,step,folder_name):
    height=gather_height(engine,cylinder,step)
    X_coor=split_coordinates(engine,cylinder)[0]
    Y_coor=split_coordinates(engine,cylinder)[1]
    rad_def=extract_radial_def(engine,cylinder,step,folder_name)
    New_X=[]
    New_Y=[]
    New_Z=[]
    New_rad_disp=[]
    height.sort(reverse=True)
    for i in range(len(height)):
        for j in range(len(height[i][1])):
            New_X.append(X_coor[height[i][1][j]])
            New_Y.append(Y_coor[height[i][1][j]])
            New_Z.append(height[i][0])
            New_rad_disp.append(rad_def[height[i][1][j]])
    return(New_X,New_Y,New_Z,New_rad_disp)

# Function computing an angle from X,Y coordinates
def Angle(X_coor,Y_coor,center):
    if X_coor-center[0]>0 and Y_coor-center[1]>=0:
        theta=m.atan((Y_coor-center[1])/(X_coor-center[0]))
    if X_coor-center[0]<0 and Y_coor-center[1]>=0:
        theta= m.pi-abs(m.atan((Y_coor-center[1])/(X_coor-center[0])))
    if X_coor-center[0]<0 and Y_coor-center[1]<=0:
        theta= m.pi+abs(m.atan((Y_coor-center[1])/(X_coor-center[0])))
    if X_coor-center[0]>0 and Y_coor-center[1]<=0:
        theta= 2*m.pi-abs(m.atan((Y_coor-center[1])/(X_coor-
center[0])))
    if X_coor-center[0]==0 and Y_coor-center[1]>=0:
        theta=m.pi/2
    if X_coor-center[0]==0 and Y_coor-center[1]<=0:
        theta=3*m.pi/2
    return (theta)

def create_angle_list(engine,cylinder,step,folder_name):
    X_coor=sort_height(engine,cylinder,step,folder_name)[0]
    Y_coor=sort_height(engine,cylinder,step,folder_name)[1]
    angle_list=[]
    Center_pt=center(X_coor,Y_coor)
    for i in range(len(X_coor)):
        angle_list.append(Angle(X_coor[i],Y_coor[i],Center_pt))
    return(angle_list)
```

```
# As the scale factor method, defined below, is based on the sign
change, it was necessary to implement a function spotting the sign
change
def sign_change(Data_list):
    gap_tab=[]
    N=len(Data_list)
    for i in range(N-1):
        if Data_list[i+1]*Data_list[i]<=0:
            gap_tab.append(abs(Data_list[i+1]-Data_list[i]))
    if Data_list[-1]*Data_list[0]<=0:
        gap_tab.append(abs(Data_list[-1]-Data_list[0]))
    return(gap_tab)

# As deformations are measured in micrometers, it is relevant to set a
scale factor in order to make the difference of shape clearer
def scale_factor(engine,cylinder,step,folder_name):
    rad_disp=sort_height(engine,cylinder,step,folder_name)[3]
    H=gather_height(engine,cylinder,step)
    n=len(H)
    gap_list=[]
    radius=diameter(engine)/2
    boundary1=0
    boundary2=0
    for i in range(n):
        boundary2+=len(H[i][1])
        temp=rad_disp[boundary1:boundary2]
        gap_list+=sign_change(temp)
        boundary1=boundary2
    max_gap=max(gap_list)
    scale_fact=int(radius/2/max_gap)
    if scale_fact>1000:
        scale_fact=1000
    return(scale_fact)

import matplotlib.pyplot as plt
import matplotlib
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter

# plot the profile of the initial state of the cylinder
def plot_initial_profile(engine,cylinder,step,index):
    radius=diameter(engine)/2
    H=gather_height(engine,cylinder,step)
    n=len(H[0][1])
    theoretical_theta=np.linspace(0,2*m.pi,n)
    list_radius=[radius]*n
    if index==0:
        plt.figure()
        ax=plt.subplot(111,projection = 'polar')
        plt.plot(theoretical_theta,list_radius)
        ax.grid(True)
        plt.show()
    if index==1:
        return(theoretical_theta,list_radius)

# As angles are in disorder, it was important to sort them by ascending
order
```

```
def reorder(list_1,list_2):
    tmp=[]
    for i in range(len(list_1)):
        tmp.append((list_1[i],i))
    tmp.sort()
    new_l1=[]
    new_l2=[]
    for i in range(len(tmp)):
        new_l1.append(tmp[i][0])
        new_l2.append(list_2[tmp[i][1]])
    return(new_l1,new_l2)


# Plot the deformed profile of a cylinder at a specific height
def plot_def_profile(engine,cylinder,step,height,folder_name,index):
    rad_def=sort_height(engine,cylinder,step,folder_name)[3]
    angle_list=create_angle_list(engine,cylinder,step,folder_name)
    H=gather_height(engine,cylinder,step)
    H.sort(reverse=True)
    radius=diameter(engine)/2
    scale_fact=scale_factor(engine,cylinder,step,folder_name)
    dR_by_height=[]
    theta_by_height=[]
    loc=0
    for s in range(height-1):
        n=len(H[s][1])
        loc+=n
    N=len(H[height-1][1])
    for i in range(loc,loc+N):
        dR_by_height.append(rad_def[i]*scale_fact+radius)
        theta_by_height.append(angle_list[i])
    theta_by_height,dR_by_height=reorder(theta_by_height,dR_by_height)
    theta_by_height.append(theta_by_height[0])
    dR_by_height.append(dR_by_height[0])
    if index==0:
        plot_initial_profile(engine,cylinder,step,index)
        fig=plt.subplot(111, projection='polar')
        plt.plot(theta_by_height,dR_by_height)
        plt.text(m.pi/6,120,"z = %s mm "%H[height-1][0],fontsize=15.0)
        fig.grid(True)
        plt.title("Comparison of the deformed and original profile \n
Engine %s Cylinder %s Step %s \n Scale Factor
%s"%(engine,cylinder,step,scale_fact))
        plt.legend(('original profile','deformed profile'),loc=0)
        plt.show()
    if index==1:
        return(theta_by_height,dR_by_height,H,scale_fact)

# Plot the deformation path of a cylinder at a specific height
def plot_path(engine,cylinder,step,height,folder_name,ind):
    rad_def=sort_height(engine,cylinder,step,folder_name)[3]
    angle_list=create_angle_list(engine,cylinder,step,folder_name)
    H=gather_height(engine,cylinder,step)
    H.sort(reverse=True)
    def_R=[]
    path=[]
    loc=0
    for s in range(height-1):
```

```
            n=len(H[s][1])
            loc+=n
        N=len(H[height-1][1])
        for i in range(loc,loc+N):
            def_R.append(rad_def[i])
            path.append(angle_list[i]*180/m.pi)
        path, def_R = reorder(path,def_R)
        if ind==0:
            plt.figure()
            plt.title("Plot of the radial deformation along a path \n
Engine %s Cylinder %s Step %s \n z = %s
mm"%(engine,cylinder,step,H[height-1][0]))
            plt.xlabel("Angular position [deg]")
            plt.ylabel("Radial deformation [mm]")
            plt.plot(path,def_R)
            plt.grid()
            plt.show()
        if ind==1:
            return(path,def_R,H)

    # The following function is used to compute the radial deformation from
    the Fourier coefficients
    def computed_dR(theta,Fc,Fa,starting_order):
        def_r=[]
        N=len(theta)
        for i in range(N):
            S=0
            for k in range(starting_order,9):
                S+=Fc[k]*m.cos((k*theta[i]+Fa[k]))
            def_r.append(S)
        return(def_r)

    # Fast Fourier Transform manually defined
    def MFFT(engine,cylinder,step,height,folder_name):

    theta=plot_def_profile(engine,cylinder,step,height,folder_name,1)[0]
        def_r=plot_path(engine,cylinder,step,height,folder_name,1)[1]
        N=len(def_r)
        X_k=[]
        X_k.append(sum(def_r)/N)
        for k in range(1,9):
            temp_sum=0
            for n in range(N):
                temp_sum+=def_r[n]*np.exp(-1j*k*theta[n])
            X_k.append(2*temp_sum/N)
        return(X_k)

    # Plotting of the deformed profile from Fourier Coefficients (without
    the zeroth and first order) in order to see if this profile obtained by
    this method looks like the one plotted by the function above
    def plot_def_profile_by_Fourier(engine,cylinder,step,height,
    folder_name,index):

    theta=plot_def_profile(engine,cylinder,step,height,folder_name,1)[0]
        H=gather_height(engine,cylinder,step)
        H.sort(reverse=True)
        Xk=MFFT(engine,cylinder,step,height,folder_name)
        Fc=np.abs(Xk)
```

```
    radius=diameter(engine)/2
    Fa=np.angle(Xk)
    dR=computed_dR(theta,Fc,Fa,2)
    scale_fact=1000
    defR=[]
    for i in range(len(dR)):
        defR.append(radius+dR[i]*scale_fact)
    if index == 0:
        plot_initial_profile(engine,cylinder,step,0)
        fig=plt.subplot(111, projection='polar')
        plt.plot(theta,defR)
        plt.text(m.pi/6,120,"z = %s mm "%H[height-1][0],fontsize=15.0)
        fig.grid(True)
        plt.title("Deformed profile plotted from Fourier coefficients
without zeroth and first order \n Engine %s Cylinder %s Step %s \n
Scale Factor %s"%(engine,cylinder,step,scale_fact))
        plt.legend(('original profile','deformed profile'),loc=0)
        plt.show()
    if index == 1:
        return(theta,defR,H)

# Plotting of the path from Fourier Coefficients in order to see if
this path obtained by this method looks like the one plotted by the
function above
def plot_path_by_Fourier(engine,cylinder,step,height,
folder_name,index):
    H=gather_height(engine,cylinder,step)
    H.sort(reverse=True)
    path=plot_path(engine,cylinder,step,height,folder_name,1)[0]

theta=plot_def_profile(engine,cylinder,step,height,folder_name,1)[0]
    del theta[-1]
    Xk=MFFT(engine,cylinder,step,height,folder_name)
    Fc=np.abs(Xk)
    Fa=np.angle(Xk)
    dR=computed_dR(theta,Fc,Fa,0)
    if index == 0:
        plt.figure()
        plt.title("Plot of the radial deformation along a path from
Fourier coefficients \n Engine %s Cylinder %s Step %s \n z = %s
mm"%(engine,cylinder,step,H[height-1][0]))
        plt.xlabel("Distance along the path [mm]")
        plt.ylabel("Radial deformation [mm]")
        plt.plot(path,dR)
        plt.grid()
        plt.show()
    if index == 1:
        return(path,dR)

# This function plots the distortions according to the different
Fourier's orders
def plot_distortion_by_order(engine,cylinder,step,folder_name,index):
    H=gather_height(engine,cylinder,step)
    H.sort(reverse=True)
    n=len(H)
    height=[]
    first_order=[]
    second_order=[]
```

```
    third_order=[]
    fourth_order=[]
    fifth_order=[]
    sixth_order=[]
    seventh_order=[]
    eighth_order=[]
    for i in range(n):
        tmp=np.abs(MFFT(engine,cylinder,step,i+1,folder_name))*2000
        first_order.append(tmp[1])
        second_order.append(tmp[2])
        third_order.append(tmp[3])
        fourth_order.append(tmp[4])
        fifth_order.append(tmp[5])
        sixth_order.append(tmp[6])
        seventh_order.append(tmp[7])
        eighth_order.append(tmp[8])
        height.append(H[i][0])
    BorderlineLim=[25,13,9,4,3,1.5,1]
    CriticalLim=[49.2,19.68,14.76,7.38,4.92,3.69,3.69]
    CriticalLim_list=[]
    BorderlineLim_list=[]
    for i in range(len(CriticalLim)):
         CriticalLim_list.append([CriticalLim[i]]*n)
         BorderlineLim_list.append([BorderlineLim[i]]*n)
    if index == 0:
        plt.figure()
        plt.suptitle("Plot of liner distortion \n Engine %s Cylinder %s
Step %s"%(engine,cylinder,step))
        plt.subplot(241)
        plt.subplots_adjust(hspace=0.2,wspace=0.3)
        plt.title("Order %s "%1)
        plt.plot(first_order,height)
        plt.grid()
        plt.xlabel("Diametral distortion [µm]")
        plt.ylabel("Axial position [mm]")
        plt.legend(('Liner Distortion','Border line limit','Critical
limit'),loc=1,fontsize=6)
        plt.subplot(242)
        plt.title("Order %s "%2)
        plt.plot(second_order,height)
        plt.plot(BorderlineLim_list[0],height,'r--')
        plt.plot(CriticalLim_list[0],height,'r-')
        plt.grid()
        plt.legend(('Liner Distortion','Border line limit','Critical
limit'),loc=1,fontsize=6)
        plt.xlabel("Diametral distortion [µm]")
        plt.ylabel("Axial position [mm]")
        plt.subplot(243)
        plt.title("Order %s "%3)
        plt.plot(third_order,height)
        plt.plot(BorderlineLim_list[1],height,'r--')
        plt.plot(CriticalLim_list[1],height,'r-')
        plt.grid()
        plt.xlabel("Diametral distortion [µm]")
        plt.ylabel("Axial position [mm]")
        plt.legend(('Liner Distortion','Border line limit','Critical
limit'),loc=1,fontsize=6)
        plt.subplot(244)
```

```
        plt.title("Order %s "%4)
        plt.plot(fourth_order,height)
        plt.plot(BorderlineLim_list[2],height,'r--')
        plt.plot(CriticalLim_list[2],height,'r-')
        plt.grid()
        plt.xlabel("Diametral distortion [µm]")
        plt.ylabel("Axial position [mm]")
        plt.legend(('Liner Distortion','Border line limit','Critical
limit'),loc=1,fontsize=6)
        plt.subplot(245)
        plt.title("Order %s "%5)
        plt.plot(fifth_order,height)
        plt.plot(BorderlineLim_list[3],height,'r--')
        plt.plot(CriticalLim_list[3],height,'r-')
        plt.grid()
        plt.xlabel("Diametral distortion [µm]")
        plt.ylabel("Axial position [mm]")
        plt.legend(('Liner Distortion','Border line limit','Critical
limit'),loc=1,fontsize=6)
        plt.subplot(246)
        plt.title("Order %s "%6)
        plt.plot(sixth_order,height)
        plt.plot(BorderlineLim_list[4],height,'r--')
        plt.plot(CriticalLim_list[4],height,'r-')
        plt.grid()
        plt.xlabel("Diametral distortion [µm]")
        plt.ylabel("Axial position [mm]")
        plt.legend(('Liner Distortion','Border line limit','Critical
limit'),loc=1,fontsize=6)
        plt.subplot(247)
        plt.title("Order %s "%7)
        plt.xlabel("Diametral distortion [µm]")
        plt.ylabel("Axial position [mm]")
        plt.plot(seventh_order,height)
        plt.plot(BorderlineLim_list[5],height,'r--')
        plt.plot(CriticalLim_list[5],height,'r-')
        plt.grid()
        plt.legend(('Liner Distortion','Border line limit','Critical
limit'),loc=1,fontsize=6)
        plt.subplot(248)
        plt.title("Order %s "%8)
        plt.plot(eighth_order,height)
        plt.plot(BorderlineLim_list[6],height,'r--')
        plt.plot(CriticalLim_list[6],height,'r-')
        plt.grid()
        plt.xlabel("Diametral distortion [µm]")
        plt.ylabel("Axial position [mm]")
        plt.legend(('Liner Distortion','Border line limit','Critical
limit'),loc=1,fontsize=6)
        plt.show()
    if index == 1:
      return(height,first_order,second_order,third_order,fourth_order,fi
fth_order,sixth_order,seventh_order,eighth_order,BorderlineLim_list,Cri
ticalLim_list)

# Accessory function in order to get a visible validation by color
def color_set(list_1,list_2,list_3):
    color_list=['w']*len(list_1)*2
```

```
    for i in range(len(list_3)):
        if list_3[i]<=list_2[i]:
            color_list.append('g')
        if list_3[i]>=list_1[i]:
            color_list.append('r')
        if list_2[i]<list_3[i]<list_1[i]:
            color_list.append('orange')
    color_list=np.array(color_list).reshape(3,7)
    return(color_list)

# This function plots the distortion of each order for a givem cylinder
and step
def plot_distortion_graph(engine,cylinder,step,height,
folder_name,index):
    H=gather_height(engine,cylinder,step)
    H.sort(reverse=True)
      Fourier_coef=np.abs(MFFT(engine,cylinder,step,height,folder_name))
[2:9]*2000
    absc=[2,3,4,5,6,7,8]
    BorderlineLim=[25,13,9,4,3,1.5,1]
    CriticalLim=[49.2,19.68,14.76,7.38,4.92,3.69,3.69]
    if index==0:
        plt.figure()
        plt.subplot(211)
        plt.title("Graph of the diametral distortion depending on
harmonic order \n Engine %s Cylinder %s Step %s \n z = %s
mm"%(engine,cylinder,step,H[height-1][0]))
        plt.plot(absc,Fourier_coef,'b')
        plt.plot(absc,BorderlineLim,'r--')
        plt.plot(absc,CriticalLim,'r-')
        plt.grid()
        plt.legend(('Liner Distortion','Border line limit','Critical
limit'),loc=1,fontsize=10)
        plt.xlabel("Harmonic order")
        plt.ylabel("Diametral distortion [µm]")
        plt.yticks(np.arange(0, 60, 5.0))
        plt.xticks([2,3,4,5,6,7,8],["2","3","4","5","6","7","8"])
        plt.subplot(212)
        collabel=["order %s"%s for s in range(2,9)]
        rowlabel=["Critical limit [µm]","Border line limit
[µm]","Diametral distortion [µm]"]

      val_table=np.array([CriticalLim,BorderlineLim,Fourier_coef]).resha
      pe(3,7)

      tab=plt.table(cellText=val_table,colLabels=collabel,rowLabels=rowl
      abel,cellColours=color_set(CriticalLim,BorderlineLim,Fourier_coef)
      ,loc='center left')
       tab.auto_set_font_size(False)
       tab.set_fontsize(12)
       plt.axis('off')
       plt.show()
    if index==1:
        return(absc,Fourier_coef,BorderlineLim,CriticalLim,H)

# This function is used to select the index of the nodes having a
specific angle
def select_angle(angle_list,value):
```

```
    index_list=[]
    if value==0:
        for i in range(len(angle_list)):
            if abs(angle_list[i])<0.05:
                index_list.append(i)
    else:
        for i in range(len(angle_list)):
            if round(angle_list[i],2)==round(value,2):
                index_list.append(i)
    return(index_list)

# Plot of the profile cut of the cylinder for a specific angle
def plot_cut_profile(engine,cylinder,step,folder_name,index):
    radius=diameter(engine)/2
    angle_list=create_angle_list(engine,cylinder,step,folder_name)
    rad_disp=sort_height(engine,cylinder,step,folder_name)[3]
    absc_max=max(abs(max(rad_disp)),abs(min(rad_disp)))
    scale_fact=scale_factor(engine,cylinder,step,folder_name)
    absc_max=radius+scale_fact*absc_max

     angle_value=[0,m.pi/4,m.pi/2,3*m.pi/4,m.pi,5*m.pi/4,3*m.pi/2,7*m.p
     i/4]
    cut_profiles=[]
    height=sort_height(engine,cylinder,step,folder_name)[2]
    H=[]
    for i in range(len(angle_value)):
        tmp=select_angle(angle_list,angle_value[i])
        TEMP=[]
        Tem=[]
        for j in range(len(tmp)):
            TEMP.append(rad_disp[tmp[j]]*scale_fact+radius)
            Tem.append(height[tmp[j]])
        cut_profiles.append(TEMP)
        H.append(Tem)
    if index == 0:
        plt.figure()
        plt.suptitle("Plot axial position with respect to radial
distortion \n Engine %s Cylinder %s Step %s \n Scale factor
%s"%(engine,cylinder,step,scale_fact))
        plt.subplots_adjust(wspace=0.3)
        plt.subplot(241)
        plt.title("Angle %s "%180)
        plt.plot(cut_profiles[4],H[4])
        plt.grid()
        plt.xlabel("Radial distortion [mm]")
        plt.xlim(2*absc_max+10,0)
        plt.ylabel("Axial position [mm]")
        plt.subplot(242)
        plt.title("Angle %s "%0)
        plt.plot(cut_profiles[0],H[0])
        plt.grid()
        plt.xlabel("Radial distortion [mm]")
        plt.xlim(0,2*absc_max+10)
        plt.ylabel("Axial position [mm]")
        plt.subplot(243)
        plt.title("Angle %s "%225)
        plt.plot(cut_profiles[5],H[5])
        plt.grid()
```

```
        plt.xlabel("Radial distortion [mm]")
        plt.xlim(2*absc_max+10,0)
        plt.ylabel("Axial position [mm]")
        plt.subplot(244)
        plt.grid()
        plt.title("Angle %s "%45)
        plt.plot(cut_profiles[1],H[1])
        plt.xlabel("Radial distortion [mm]")
        plt.xlim(0,2*absc_max+10)
        plt.ylabel("Axial position [mm]")
        plt.subplot(245)
        plt.title("Angle %s "%270)
        plt.plot(cut_profiles[6],H[6])
        plt.grid()
        plt.xlabel("Radial distortion [mm]")
        plt.xlim(2*absc_max+10,0)
        plt.ylabel("Axial position [mm]")
        plt.subplot(246)
        plt.title("Angle %s "%90)
        plt.plot(cut_profiles[2],H[2])
        plt.grid()
        plt.xlabel("Radial distortion [mm]")
        plt.xlim(0,2*absc_max+10)
        plt.ylabel("Axial position [mm]")
        plt.subplot(247)
        plt.title("Angle %s "%315)
        plt.plot(cut_profiles[7],H[7])
        plt.grid()
        plt.xlabel("Radial distortion [mm]")
        plt.xlim(2*absc_max+10,0)
        plt.ylabel("Axial position [mm]")
        plt.subplot(248)
        plt.title("Angle %s "%135)
        plt.plot(cut_profiles[3],H[3])
        plt.grid()
        plt.xlabel("Radial distortion [mm]")
        plt.xlim(0,2*absc_max+10)
        plt.ylabel("Axial position [mm]")
        plt.show()
    if index == 1:
        return(H,cut_profiles,absc_max)

def COMP_plot_def_profile(engine,cylinder,step,height,folder_name,
index):
    rad_def=sort_height(engine,cylinder,step,folder_name)[3]
    angle_list=create_angle_list(engine,cylinder,step,folder_name)
    H=gather_height(engine,cylinder,step)
    H.sort(reverse=True)
    radius=diameter(engine)/2
    scale_fact=100
    dR_by_height=[]
    theta_by_height=[]
    loc=0
    for s in range(height-1):
        n=len(H[s][1])
        loc+=n
    N=len(H[height-1][1])
    for i in range(loc,loc+N):
```

```
            dR_by_height.append(rad_def[i]*scale_fact+radius)
            theta_by_height.append(angle_list[i])
        theta_by_height,dR_by_height=reorder(theta_by_height,dR_by_height)
        theta_by_height.append(theta_by_height[0])
        dR_by_height.append(dR_by_height[0])
        if index==0:
            plot_initial_profile(engine,cylinder,step,index)
            fig=plt.subplot(111, projection='polar')
            plt.plot(theta_by_height,dR_by_height)
            plt.text(m.pi/6,120,"z = %s mm "%H[height-1][0],fontsize=15.0)
            fig.grid(True)
            plt.title("Comparison of the deformed and original profile \n
Engine %s Cylinder %s Step %s \n Scale Factor
%s"%(engine,cylinder,step,scale_fact))
            plt.legend(('original profile','deformed profile'),loc=0)
            plt.show()
        if index==1:
            return(theta_by_height,dR_by_height,H,scale_fact)

def COMP_plot_cut_profile(engine,cylinder,step,folder_name,index):
    radius=diameter(engine)/2
    angle_list=create_angle_list(engine,cylinder,step,folder_name)
    rad_disp=sort_height(engine,cylinder,step,folder_name)[3]
    absc_max=max(abs(max(rad_disp)),abs(min(rad_disp)))
    scale_fact=100
    absc_max=radius+scale_fact*absc_max

      angle_value=[0,m.pi/4,m.pi/2,3*m.pi/4,m.pi,5*m.pi/4,3*m.pi/2,7*m.p
      i/4]
    cut_profiles=[]
    height=sort_height(engine,cylinder,step,folder_name)[2]
    H=[]
    for i in range(len(angle_value)):
        tmp=select_angle(angle_list,angle_value[i])
        TEMP=[]
        Tem=[]
        for j in range(len(tmp)):
            TEMP.append(rad_disp[tmp[j]]*scale_fact+radius)
            Tem.append(height[tmp[j]])
        cut_profiles.append(TEMP)
        H.append(Tem)
    if index == 0:
        plt.figure()
        plt.suptitle("Plot axial position with respect to radial
distortion \n Engine %s Cylinder %s Step %s \n Scale factor
%s"%(engine,cylinder,step,scale_fact))
        plt.subplots_adjust(wspace=0.3)
        plt.subplot(241)
        plt.title("Angle %s "%180)
        plt.plot(cut_profiles[4],H[4])
        plt.grid()
        plt.xlabel("Radial distortion [mm]")
        plt.xlim(2*absc_max+10,0)
        plt.ylabel("Axial position [mm]")
        plt.subplot(242)
        plt.title("Angle %s "%0)
        plt.plot(cut_profiles[0],H[0])
        plt.grid()
```

```
        plt.xlabel("Radial distortion [mm]")
        plt.xlim(0,2*absc_max+10)
        plt.ylabel("Axial position [mm]")
        plt.subplot(243)
        plt.title("Angle %s "%225)
        plt.plot(cut_profiles[5],H[5])
        plt.grid()
        plt.xlabel("Radial distortion [mm]")
        plt.xlim(2*absc_max+10,0)
        plt.ylabel("Axial position [mm]")
        plt.subplot(244)
        plt.grid()
        plt.title("Angle %s "%45)
        plt.plot(cut_profiles[1],H[1])
        plt.xlabel("Radial distortion [mm]")
        plt.xlim(0,2*absc_max+10)
        plt.ylabel("Axial position [mm]")
        plt.subplot(245)
        plt.title("Angle %s "%270)
        plt.plot(cut_profiles[6],H[6])
        plt.grid()
        plt.xlabel("Radial distortion [mm]")
        plt.xlim(2*absc_max+10,0)
        plt.ylabel("Axial position [mm]")
        plt.subplot(246)
        plt.title("Angle %s "%90)
        plt.plot(cut_profiles[2],H[2])
        plt.grid()
        plt.xlabel("Radial distortion [mm]")
        plt.xlim(0,2*absc_max+10)
        plt.ylabel("Axial position [mm]")
        plt.subplot(247)
        plt.title("Angle %s "%315)
        plt.plot(cut_profiles[7],H[7])
        plt.grid()
        plt.xlabel("Radial distortion [mm]")
        plt.xlim(2*absc_max+10,0)
        plt.ylabel("Axial position [mm]")
        plt.subplot(248)
        plt.title("Angle %s "%135)
        plt.plot(cut_profiles[3],H[3])
        plt.grid()
        plt.xlabel("Radial distortion [mm]")
        plt.xlim(0,2*absc_max+10)
        plt.ylabel("Axial position [mm]")
        plt.show()
    if index == 1:
        return(H,cut_profiles,absc_max)
```

Temperature_Analysis script

```
import math as m
import os
import matplotlib.pyplot as plt
import numpy as np
import matplotlib
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter

# All  functions, implemented in BoreDistortion_Analysis_V6 have been
recoded and modified to deal with temperature files, which are bigger
# In order to reduce as much as possible the computing time each
function has been re written for temperatures instead of doing one
function dealing with both deformation and temperature files

plt.close('all')

dirpath = "C:/Users/A-Damien.Gode/Documents/Distortion
program/Analysis"

os.chdir(dirpath)

def center(X_coor,Y_coor):
    center=0
    max_X=max(X_coor)
    min_X=min(X_coor)
    max_Y=max(Y_coor)
    min_Y=min(Y_coor)
    center=((max_X+min_X)/2,(max_Y+min_Y)/2)
    return(center)

def T_extract_coordinates(engine,cylinder):
    os.chdir(dirpath)
    if engine == 'MX11':
        doc=open('TEMP_MX11_cyl%s_Coordinates.txt'%cylinder,'r')
        doc=doc.read()
        doc=doc.replace("\n","")
        doc=doc.replace("LIN-1","")
        doc=doc.replace("FR-1","")
        doc=doc.split()
        Coor_table=[]
        index=0
        index1=0
        for i in range(len(doc)):
            if doc[i]=='Part':
                index=i
            if doc[i]=='-----------------------------------------------
-------------------------------------------------------------------
---':
                index1=i
        Coor_table=doc[index1+1:index]
        return(Coor_table)
    if engine == 'MX13':
        doc=open('TEMP_MX13_cyl%s_Coordinates.txt'%cylinder,'r')
```

```
        doc=doc.read()
        doc=doc.replace("\n","")
        doc=doc.replace("LINER-1","")
        doc=doc.replace("FIRE-RING-1","")
        doc=doc.split()
        Coor_table=[]
        index=0
        index1=0
        for i in range(len(doc)):
            if doc[i]=='Part':
                index=i
            if doc[i]=='-------------------------------------------------
------------------------------------------------------------------------
---':
                index1=i
        Coor_table=doc[index1+1:index]
        return(Coor_table)

def T_split_coordinates(engine,cylinder):
    Coor_table=T_extract_coordinates(engine,cylinder)
    X_coor=[]
    Y_coor=[]
    Z_coor=[]
    count=1
    X_coor.append(float(Coor_table[1]))
    Y_coor.append(float(Coor_table[2]))
    Z_coor.append(float(Coor_table[3]))
    while 1+7*count<len(Coor_table):
        X_coor.append(float(Coor_table[1+7*count]))
        Y_coor.append(float(Coor_table[2+7*count]))
        Z_coor.append(float(Coor_table[3+7*count]))
        count+=1
    return(X_coor,Y_coor,Z_coor)

def T_extract_temp_data(engine,cylinder,step,folder_name):
    os.chdir(dirpath+"/%s"%folder_name)
    if engine == 'MX11':
        doc=open('TEMP_MX11_cyl%s_step%s.txt'%(cylinder,step),'r')
        doc=doc.read()
        doc=doc.split()
        Temp_table_1=[]
        Temp_table_2=[]
        Temp_table=[]
        index=0
        index1=0
        for i in range(len(doc)):
            if doc[i]=='FR-1':
                index=i
            if doc[i]=='LIN-1':
                index1=i
        Temp_table_1=doc[index+7:index1-7]
        Temp_table_2=doc[index1+7:len(doc)]
        Temp_table=Temp_table_1+Temp_table_2
        return(Temp_table)
    if engine == 'MX13':
        doc=open('TEMP_MX13_cyl%s_step%s.txt'%(cylinder,step),'r')
        doc=doc.read()
        doc=doc.split()
```

```
        Temp_table_1=[]
        Temp_table_2=[]
        Temp_table=[]
        index=0
        index1=0
        for i in range(len(doc)):
            if doc[i]=='FIRE-RING-1':
                index=i
            if doc[i]=='LINER-1':
                index1=i
        Temp_table_1=doc[index+7:index1-7]
        Temp_table_2=doc[index1+7:len(doc)]
        Temp_table=Temp_table_1+Temp_table_2
        return(Temp_table)

def T_extract_temp(engine,cylinder,step,folder_name):
    data_table=T_extract_temp_data(engine,cylinder,step,folder_name)
    Temp_table=[]
    count=1
    Temp_table.append(float(data_table[1]))
    while 2*count<len(data_table):
        Temp_table.append(float(data_table[2*count+1]))
        count+=1
    return(Temp_table)

def belong(List,val):
    for i in range(len(List)):
        if List[i]==val:
            return(True)
    return(False)

def T_split_height(engine,cylinder,step):
    TEMP_Z=T_split_coordinates(engine,cylinder)[2]
    TEMP_height=[]
    TEMP_already=[]
    TEMP_ind_list=[]
    TEMP_index=0
    while TEMP_index<len(TEMP_Z):
        TEMP_value=TEMP_Z[TEMP_index]
        if belong(TEMP_already,TEMP_value)==True:
            TEMP_index+=1
        else:
            TEMP_ind_list=[]
            for i in range(len(TEMP_Z)):
                if TEMP_Z[i]==TEMP_value:
                    TEMP_ind_list.append(i)
            TEMP_height.append((TEMP_value,TEMP_ind_list))
            TEMP_already.append(TEMP_value)
            TEMP_index+=1
    return(TEMP_height)

def T_gather_height(engine,cylinder,step):
    TEMP_height=T_split_height(engine,cylinder,step)
    TEMP_comp_val=0
    TEMP_index=0
    TEMP_already=[]
    TEMP_H=[]
    TEMP_index_list=[]
```

```
    while TEMP_index<len(TEMP_height):
        TEMP_comp_val=TEMP_height[TEMP_index][0]
        if belong(TEMP_already,round(TEMP_comp_val,1))==True:
            TEMP_index+=1
        else:
            TEMP_index_list=[]
            for i in range (TEMP_index,len(TEMP_height)):
                if round(TEMP_height[i][0],1)==round(TEMP_comp_val,1):
                    TEMP_index_list+=TEMP_height[i][1]
            TEMP_H.append((TEMP_comp_val,TEMP_index_list))
            TEMP_already.append(round(TEMP_comp_val,1))
            TEMP_index+=1
    return(TEMP_H)

def T_sort_height(engine,cylinder,step,folder_name):
    TEMP_height=T_gather_height(engine,cylinder,step)
    TEMP_height.sort(reverse=True)
    Coor_X=T_split_coordinates(engine,cylinder)[0]
    Coor_Y=T_split_coordinates(engine,cylinder)[1]
    Temperature_tab=T_extract_temp(engine,cylinder,step,folder_name)
    TEMP_X=[]
    TEMP_Y=[]
    TEMP_Z=[]
    New_Temperature=[]
    for i in range(len(TEMP_height)):
        for j in range(len(TEMP_height[i][1])):
            TEMP_X.append(Coor_X[TEMP_height[i][1][j]])
            TEMP_Y.append(Coor_Y[TEMP_height[i][1][j]])
            TEMP_Z.append(TEMP_height[i][0])

New_Temperature.append(Temperature_tab[TEMP_height[i][1][j]])
    return(TEMP_X,TEMP_Y,TEMP_Z,New_Temperature)

def Angle(X_coor,Y_coor,center):
    if X_coor-center[0]>0 and Y_coor-center[1]>=0:
        theta=m.atan((Y_coor-center[1])/(X_coor-center[0]))
    if X_coor-center[0]<0 and Y_coor-center[1]>=0:
        theta= m.pi-abs(m.atan((Y_coor-center[1])/(X_coor-center[0])))
    if X_coor-center[0]<0 and Y_coor-center[1]<=0:
        theta= m.pi+abs(m.atan((Y_coor-center[1])/(X_coor-center[0])))
    if X_coor-center[0]>0 and Y_coor-center[1]<=0:
        theta= 2*m.pi-abs(m.atan((Y_coor-center[1])/(X_coor-
center[0])))
    if X_coor-center[0]==0 and Y_coor-center[1]>=0:
        theta=m.pi/2
    if X_coor-center[0]==0 and Y_coor-center[1]<=0:
        theta=3*m.pi/2
    return (theta)

def T_create_angle_list(engine,cylinder,step,folder_name):
    TEMP_X_pos=T_sort_height(engine,cylinder,step,folder_name)[0]
    TEMP_Y_pos=T_sort_height(engine,cylinder,step,folder_name)[1]
    TEMP_angle_list=[]
    TEMP_center=center(TEMP_X_pos,TEMP_Y_pos)
    for i in range(len(TEMP_X_pos)):
        TEMP_angle_list.append(Angle(TEMP_X_pos[i],TEMP_Y_pos[i],
TEMP_center))
    return(TEMP_angle_list)
```

```
def select_angle(engine,angle_list,value):
    index_list=[]
    if value==0:
        for i in range(len(angle_list)):
            if engine=='MX11' and abs(angle_list[i])<0.1:
                index_list.append(i)
            if engine=='MX13' and abs(angle_list[i])<0.05:
                index_list.append(i)
    else:
        for i in range(len(angle_list)):
            if round(angle_list[i],2)==round(value,2):
                index_list.append(i)
    return(index_list)

def plot_wall_temperature_single(engine,step,folder_name,angle_value):
    H_3cylinder=[]
    TEMP_3cylinder=[]
    for c in range(1,4):
        angle_list=T_create_angle_list(engine,c,step,folder_name)
        temperature_tab=T_sort_height(engine,c,step,folder_name)[3]
        TEMP_profiles=[]
        height=T_sort_height(engine,c,step,folder_name)[2]
        H=[]
        tmp=select_angle(engine,angle_list,angle_value*m.pi/180)
        TEMP=[]
        Tem=[]
        for j in range(len(tmp)):
            TEMP.append(temperature_tab[tmp[j]])
            Tem.append(height[tmp[j]])
        TEMP_profiles.append(TEMP)
        H.append(Tem)
        H_3cylinder.append(H)
        TEMP_3cylinder.append(TEMP_profiles)
    plt.figure()
    plt.suptitle("Plot wall temperature \n Engine %s Step
%s"%(engine,step))
    plt.title("Angle %s "%angle_value)
    plt.plot(TEMP_3cylinder[0][0],H_3cylinder[0][0],'*')
    plt.plot(TEMP_3cylinder[1][0],H_3cylinder[1][0],'-')
    plt.plot(TEMP_3cylinder[2][0],H_3cylinder[2][0],'.')
    plt.grid()
    plt.legend(('Liner 1','Liner 2','Liner 3'),loc=2,fontsize=10)
    plt.xlabel("Temperature [°C]")
    plt.ylabel("Axial position [mm]")
    if round(max(TEMP_3cylinder[0][0][0],
TEMP_3cylinder[1][0][0],TEMP_3cylinder[2][0][0]),0) < 50:
        plt.annotate('Tmax = %s °C
'%round(max(TEMP_3cylinder[0][0][0],TEMP_3cylinder[1][0][0],TEMP_3cylin
der[2][0][0]),0),
xy=(max(TEMP_3cylinder[0][0][0],TEMP_3cylinder[1][0][0],TEMP_3cylinder[
2][0][0]), H_3cylinder[0][0][0]),
xytext=(max(TEMP_3cylinder[0][0][0],TEMP_3cylinder[1][0][0],TEMP_3cylin
der[2][0][0])*1.025, H_3cylinder[0][0][0]-50),
                arrowprops=dict(facecolor='black', shrink=0.05),
                )
    else:
```

```
        plt.annotate('Tmax = %s °C
'%round(max(TEMP_3cylinder[0][0][0],TEMP_3cylinder[1][0][0],TEMP_3cylin
der[2][0][0]),0),
xy=(max(TEMP_3cylinder[0][0][0],TEMP_3cylinder[1][0][0],TEMP_3cylinder[
2][0][0]), H_3cylinder[0][0][0]),
xytext=(max(TEMP_3cylinder[0][0][0],TEMP_3cylinder[1][0][0],TEMP_3cylin
der[2][0][0])*0.85, H_3cylinder[0][0][0]-50),
                arrowprops=dict(facecolor='black', shrink=0.05),
                )


def plot_wall_temperature_complete(engine,step,folder_name):
    H_3cylinder=[]
    TEMP_3cylinder=[]
    for c in range(1,4):
        angle_list=T_create_angle_list(engine,c,step,folder_name)
        temperature_tab=T_sort_height(engine,c,step,folder_name)[3]
        angle_value=[0,m.pi/4,m.pi/2,3*m.pi/4,m.pi,5*m.pi/4,
3*m.pi/2,7*m.pi/4]
        TEMP_profiles=[]
        height=T_sort_height(engine,c,step,folder_name)[2]
        H=[]
        for i in range(len(angle_value)):
            tmp=select_angle(engine,angle_list,angle_value[i])
            TEMP=[]
            Tem=[]
            for j in range(len(tmp)):
                TEMP.append(temperature_tab[tmp[j]])
                Tem.append(height[tmp[j]])
            TEMP_profiles.append(TEMP)
            H.append(Tem)
        H_3cylinder.append(H)
        TEMP_3cylinder.append(TEMP_profiles)
    plt.figure()
    plt.suptitle("Plot wall temperature \n Engine %s Step
%s"%(engine,step))
    plt.subplot(241)
    plt.title("Angle %s "%0)
    plt.plot(TEMP_3cylinder[0][0],H_3cylinder[0][0],'*')
    plt.plot(TEMP_3cylinder[1][0],H_3cylinder[1][0],'-')
    plt.plot(TEMP_3cylinder[2][0],H_3cylinder[2][0],'.')
    plt.grid()
    plt.legend(('Liner 1','Liner 2','Liner 3'),loc=2,fontsize=10)
    plt.xlabel("Temperature [°C]")
    plt.ylabel("Axial position [mm]")
    if round(max(TEMP_3cylinder[0][0][0],TEMP_3cylinder[1][0][0],
TEMP_3cylinder[2][0][0]),0) < 50:
        plt.annotate('Tmax = %s °C
'%round(max(TEMP_3cylinder[0][0][0],TEMP_3cylinder[1][0][0],TEMP_3cylin
der[2][0][0]),0),
xy=(max(TEMP_3cylinder[0][0][0],TEMP_3cylinder[1][0][0],TEMP_3cylinder[
2][0][0]), H_3cylinder[0][0][0]),
xytext=(max(TEMP_3cylinder[0][0][0],TEMP_3cylinder[1][0][0],TEMP_3cylin
der[2][0][0])*1.025, H_3cylinder[0][0][0]-50),
                arrowprops=dict(facecolor='black', shrink=0.05),
                )
    else:
```

```
        plt.annotate('Tmax = %s °C
'%round(max(TEMP_3cylinder[0][0][0],TEMP_3cylinder[1][0][0],TEMP_3cylin
der[2][0][0]),0),
xy=(max(TEMP_3cylinder[0][0][0],TEMP_3cylinder[1][0][0],TEMP_3cylinder[
2][0][0]), H_3cylinder[0][0][0]),
xytext=(max(TEMP_3cylinder[0][0][0],TEMP_3cylinder[1][0][0],TEMP_3cylin
der[2][0][0])*0.85, H_3cylinder[0][0][0]-50),
                arrowprops=dict(facecolor='black', shrink=0.05),
                )
    plt.subplot(242)
    plt.title("Angle %s "%45)
    plt.plot(TEMP_3cylinder[0][1],H_3cylinder[0][1],'*')
    plt.plot(TEMP_3cylinder[1][1],H_3cylinder[1][1],'-')
    plt.plot(TEMP_3cylinder[2][1],H_3cylinder[2][1],'.')
    plt.grid()
    plt.legend(('Liner 1','Liner 2','Liner 3'),loc=2,fontsize=10)
    plt.xlabel("Temperature [°C]")
    plt.ylabel("Axial position [mm]")
    if round(max(TEMP_3cylinder[0][1][0],TEMP_3cylinder[1][1][0],
TEMP_3cylinder[2][1][0]),0) < 50:
        plt.annotate('Tmax = %s °C
'%round(max(TEMP_3cylinder[0][1][0],TEMP_3cylinder[1][1][0],TEMP_3cylin
der[2][1][0]),0),
xy=(max(TEMP_3cylinder[0][1][0],TEMP_3cylinder[1][1][0],TEMP_3cylinder[
2][1][0]), H_3cylinder[0][1][0]),
xytext=(max(TEMP_3cylinder[0][1][0],TEMP_3cylinder[1][1][0],TEMP_3cylin
der[2][1][0])*1.025, H_3cylinder[0][1][0]-50),
            arrowprops=dict(facecolor='black', shrink=0.05),
            )
    else:
        plt.annotate('Tmax = %s °C
'%round(max(TEMP_3cylinder[0][1][0],TEMP_3cylinder[1][1][0],TEMP_3cylin
der[2][1][0]),0),
xy=(max(TEMP_3cylinder[0][1][0],TEMP_3cylinder[1][1][0],TEMP_3cylinder[
2][1][0]), H_3cylinder[0][1][0]),
xytext=(max(TEMP_3cylinder[0][1][0],TEMP_3cylinder[1][1][0],TEMP_3cylin
der[2][1][0])*0.85, H_3cylinder[0][1][0]-50),
            arrowprops=dict(facecolor='black', shrink=0.05),
            )
    plt.subplot(243)
    plt.title("Angle %s "%90)
    plt.plot(TEMP_3cylinder[0][2],H_3cylinder[0][2],'*')
    plt.plot(TEMP_3cylinder[1][2],H_3cylinder[1][2],'-')
    plt.plot(TEMP_3cylinder[2][2],H_3cylinder[2][2],'.')
    plt.grid()
    plt.legend(('Liner 1','Liner 2','Liner 3'),loc=2,fontsize=10)
    plt.xlabel("Temperature [°C]")
    plt.ylabel("Axial position [mm]")
    if round(max(TEMP_3cylinder[0][2][0],TEMP_3cylinder[1][2][0],
TEMP_3cylinder[2][2][0]),0) < 50:
        plt.annotate('Tmax = %s °C
'%round(max(TEMP_3cylinder[0][2][0],TEMP_3cylinder[1][2][0],TEMP_3cylin
der[2][2][0]),0),
xy=(max(TEMP_3cylinder[0][2][0],TEMP_3cylinder[1][2][0],TEMP_3cylinder[
2][2][0]), H_3cylinder[0][2][0]),
xytext=(max(TEMP_3cylinder[0][2][0],TEMP_3cylinder[1][2][0],TEMP_3cylin
der[2][2][0])*1.025, H_3cylinder[0][2][0]-50),
            arrowprops=dict(facecolor='black', shrink=0.05),
```

```
            )
    else:
        plt.annotate('Tmax = %s °C
'%round(max(TEMP_3cylinder[0][2][0],TEMP_3cylinder[1][2][0],TEMP_3cylin
der[2][2][0]),0),
xy=(max(TEMP_3cylinder[0][2][0],TEMP_3cylinder[1][2][0],TEMP_3cylinder[
2][2][0]), H_3cylinder[0][2][0]),
xytext=(max(TEMP_3cylinder[0][2][0],TEMP_3cylinder[1][2][0],TEMP_3cylin
der[2][2][0])*0.85, H_3cylinder[0][2][0]-50),
            arrowprops=dict(facecolor='black', shrink=0.05),
            )
    plt.subplot(244)
    plt.title("Angle %s "%135)
    plt.plot(TEMP_3cylinder[0][3],H_3cylinder[0][3],'*')
    plt.plot(TEMP_3cylinder[1][3],H_3cylinder[1][3],'-')
    plt.plot(TEMP_3cylinder[2][3],H_3cylinder[2][3],'.')
    plt.grid()
    plt.legend(('Liner 1','Liner 2','Liner 3'),loc=2,fontsize=10)
    plt.xlabel("Temperature [°C]")
    plt.ylabel("Axial position [mm]")
    if round(max(TEMP_3cylinder[0][3][0],TEMP_3cylinder[1][3][0],
TEMP_3cylinder[2][3][0]),0) < 50:
        plt.annotate('Tmax = %s °C
'%round(max(TEMP_3cylinder[0][3][0],TEMP_3cylinder[1][3][0],TEMP_3cylin
der[2][3][0]),0),
xy=(max(TEMP_3cylinder[0][3][0],TEMP_3cylinder[1][3][0],TEMP_3cylinder[
2][3][0]), H_3cylinder[0][3][0]),
xytext=(max(TEMP_3cylinder[0][3][0],TEMP_3cylinder[1][3][0],TEMP_3cylin
der[2][3][0])*1.025, H_3cylinder[0][3][0]-50),
            arrowprops=dict(facecolor='black', shrink=0.05),
            )
    else:
        plt.annotate('Tmax = %s °C
'%round(max(TEMP_3cylinder[0][3][0],TEMP_3cylinder[1][3][0],TEMP_3cylin
der[2][3][0]),0),
xy=(max(TEMP_3cylinder[0][3][0],TEMP_3cylinder[1][3][0],TEMP_3cylinder[
2][3][0]), H_3cylinder[0][3][0]),
xytext=(max(TEMP_3cylinder[0][3][0],TEMP_3cylinder[1][3][0],TEMP_3cylin
der[2][3][0])*0.85, H_3cylinder[0][3][0]-50),
            arrowprops=dict(facecolor='black', shrink=0.05),
            )
    plt.subplot(245)
    plt.title("Angle %s "%180)
    plt.plot(TEMP_3cylinder[0][4],H_3cylinder[0][4],'*')
    plt.plot(TEMP_3cylinder[1][4],H_3cylinder[1][4],'-')
    plt.plot(TEMP_3cylinder[2][4],H_3cylinder[2][4],'.')
    plt.grid()
    plt.legend(('Liner 1','Liner 2','Liner 3'),loc=2,fontsize=10)
    plt.xlabel("Temperature [°C]")
    plt.ylabel("Axial position [mm]")
    if round(max(TEMP_3cylinder[0][4][0],TEMP_3cylinder[1][4][0],
TEMP_3cylinder[2][4][0]),0) < 50:
        plt.annotate('Tmax = %s °C
'%round(max(TEMP_3cylinder[0][4][0],TEMP_3cylinder[1][4][0],TEMP_3cylin
der[2][4][0]),0),
xy=(max(TEMP_3cylinder[0][4][0],TEMP_3cylinder[1][4][0],TEMP_3cylinder[
2][4][0]), H_3cylinder[0][4][0]),
```

```
xytext=(max(TEMP_3cylinder[0][4][0],TEMP_3cylinder[1][4][0],TEMP_3cylin
der[2][4][0])*1.025, H_3cylinder[0][4][0]-50),
            arrowprops=dict(facecolor='black', shrink=0.05),
            )
    else:
        plt.annotate('Tmax = %s °C
'%round(max(TEMP_3cylinder[0][4][0],TEMP_3cylinder[1][4][0],TEMP_3cylin
der[2][4][0]),0),
xy=(max(TEMP_3cylinder[0][4][0],TEMP_3cylinder[1][4][0],TEMP_3cylinder[
2][4][0]), H_3cylinder[0][4][0]),
xytext=(max(TEMP_3cylinder[0][4][0],TEMP_3cylinder[1][4][0],TEMP_3cylin
der[2][4][0])*0.85, H_3cylinder[0][4][0]-50),
            arrowprops=dict(facecolor='black', shrink=0.05),
            )
    plt.subplot(246)
    plt.title("Angle %s "%225)
    plt.plot(TEMP_3cylinder[0][5],H_3cylinder[0][5],'*')
    plt.plot(TEMP_3cylinder[1][5],H_3cylinder[1][5],'-')
    plt.plot(TEMP_3cylinder[2][5],H_3cylinder[2][5],'.')
    plt.grid()
    plt.legend(('Liner 1','Liner 2','Liner 3'),loc=2,fontsize=10)
    plt.xlabel("Temperature [°C]")
    plt.ylabel("Axial position [mm]")
    if round(max(TEMP_3cylinder[0][5][0],TEMP_3cylinder[1][5][0],
TEMP_3cylinder[2][5][0]),0) < 50:
        plt.annotate('Tmax = %s °C
'%round(max(TEMP_3cylinder[0][5][0],TEMP_3cylinder[1][5][0],TEMP_3cylin
der[2][5][0]),0),
xy=(max(TEMP_3cylinder[0][5][0],TEMP_3cylinder[1][5][0],TEMP_3cylinder[
2][5][0]), H_3cylinder[0][5][0]),
xytext=(max(TEMP_3cylinder[0][5][0],TEMP_3cylinder[1][5][0],TEMP_3cylin
der[2][5][0])*1.025, H_3cylinder[0][5][0]-50),
            arrowprops=dict(facecolor='black', shrink=0.05),
            )
    else:
        plt.annotate('Tmax = %s °C
'%round(max(TEMP_3cylinder[0][5][0],TEMP_3cylinder[1][5][0],TEMP_3cylin
der[2][5][0]),0),
xy=(max(TEMP_3cylinder[0][5][0],TEMP_3cylinder[1][5][0],TEMP_3cylinder[
2][5][0]), H_3cylinder[0][5][0]),
xytext=(max(TEMP_3cylinder[0][5][0],TEMP_3cylinder[1][5][0],TEMP_3cylin
der[2][5][0])*0.85, H_3cylinder[0][5][0]-50),
            arrowprops=dict(facecolor='black', shrink=0.05),
            )
    plt.subplot(247)
    plt.title("Angle %s "%270)
    plt.plot(TEMP_3cylinder[0][6],H_3cylinder[0][6],'*')
    plt.plot(TEMP_3cylinder[1][6],H_3cylinder[1][6],'-')
    plt.plot(TEMP_3cylinder[2][6],H_3cylinder[2][6],'.')
    plt.grid()
    plt.legend(('Liner 1','Liner 2','Liner 3'),loc=2,fontsize=10)
    plt.xlabel("Temperature [°C]")
    plt.ylabel("Axial position [mm]")
    if round(max(TEMP_3cylinder[0][6][0],TEMP_3cylinder[1][6][0],
TEMP_3cylinder[2][6][0]),0) < 50:
        plt.annotate('Tmax = %s °C
'%round(max(TEMP_3cylinder[0][6][0],TEMP_3cylinder[1][6][0],TEMP_3cylin
der[2][6][0]),0),
```

```
xy=(max(TEMP_3cylinder[0][6][0],TEMP_3cylinder[1][6][0],TEMP_3cylinder[
2][6][0]), H_3cylinder[0][6][0]),
xytext=(max(TEMP_3cylinder[0][6][0],TEMP_3cylinder[1][6][0],TEMP_3cylin
der[2][6][0])*1.025, H_3cylinder[0][6][0]-50),
            arrowprops=dict(facecolor='black', shrink=0.05),
            )
    else:
        plt.annotate('Tmax = %s °C
'%round(max(TEMP_3cylinder[0][6][0],TEMP_3cylinder[1][6][0],TEMP_3cylin
der[2][6][0]),0),
xy=(max(TEMP_3cylinder[0][6][0],TEMP_3cylinder[1][6][0],TEMP_3cylinder[
2][6][0]), H_3cylinder[0][6][0]),
xytext=(max(TEMP_3cylinder[0][6][0],TEMP_3cylinder[1][6][0],TEMP_3cylin
der[2][6][0])*0.85, H_3cylinder[0][6][0]-50),
            arrowprops=dict(facecolor='black', shrink=0.05),
            )
    plt.subplot(248)
    plt.title("Angle %s "%315)
    plt.plot(TEMP_3cylinder[0][7],H_3cylinder[0][7],'*')
    plt.plot(TEMP_3cylinder[1][7],H_3cylinder[1][7],'-')
    plt.plot(TEMP_3cylinder[2][7],H_3cylinder[2][7],'.')
    plt.grid()
    plt.legend(('Liner 1','Liner 2','Liner 3'),loc=2,fontsize=10)
    plt.xlabel("Temperature [°C]")
    plt.ylabel("Axial position [mm]")
    if round(max(TEMP_3cylinder[0][7][0],TEMP_3cylinder[1][7][0],
TEMP_3cylinder[2][7][0]),0) < 50:
        plt.annotate('Tmax = %s °C
'%round(max(TEMP_3cylinder[0][7][0],TEMP_3cylinder[1][7][0],TEMP_3cylin
der[2][7][0]),0),
xy=(max(TEMP_3cylinder[0][7][0],TEMP_3cylinder[1][7][0],TEMP_3cylinder[
2][7][0]), H_3cylinder[0][7][0]),
xytext=(max(TEMP_3cylinder[0][7][0],TEMP_3cylinder[1][7][0],TEMP_3cylin
der[2][7][0])*1.025, H_3cylinder[0][7][0]-50),
            arrowprops=dict(facecolor='black', shrink=0.05),
            )
    else:
        plt.annotate('Tmax = %s °C
'%round(max(TEMP_3cylinder[0][7][0],TEMP_3cylinder[1][7][0],TEMP_3cylin
der[2][7][0]),0),
xy=(max(TEMP_3cylinder[0][7][0],TEMP_3cylinder[1][7][0],TEMP_3cylinder[
2][7][0]), H_3cylinder[0][7][0]),
xytext=(max(TEMP_3cylinder[0][7][0],TEMP_3cylinder[1][7][0],TEMP_3cylin
der[2][7][0])*0.85, H_3cylinder[0][7][0]-50),
            arrowprops=dict(facecolor='black', shrink=0.05),
            )
    plt.show()
```

Distortion_generate_files script

```
import os
import numpy as np
from collections import OrderedDict
from abaqus import *
from abaqusConstants import *
import __main__
from abaqus import getInput

dirpath = "C:\Users\A-Damien.Gode\Documents\Distortion
program\Analysis"

os.chdir(dirpath)

def create_folder():
    os.chdir(dirpath)
    odb = session.odbs[session.odbData.keys()[0]]
    tmp=odb.name.split()[-1]
    name=''
    index=0
    for i in range(len(tmp)):
        if tmp[i] == '/':
            index=i
    name=tmp[index+1:len(tmp)]
    name = name[0:len(name)-4]
    if os.path.exists(name) == False:
        os.makedirs(name)
    return(name)

def belong(List,val):
    for i in range(len(List)):
        if List[i]==val:
            return(True)
    return(False)

def cylinder_selection(manual=False):
    cylinder_list=[]
    if manual == False:
        cylinder_list=[1,2,3]
        return(cylinder_list)
    if manual == True:
        already=[]
        N=int(getInput("How many cylinders do you consider?"))
        for i in range(N):
            cyl=int(getInput("Which cylinder?"))
            while belong(already,cyl) == True:
                cyl=int(getInput("Already selected! Choose a new one
:"))
            already.append(cyl)
            cylinder_list.append(cyl)
        return(cylinder_list)

def step_index():
    index=[]
    odb = session.odbs[session.odbData.keys()[0]]
```

```
    N=len(odb.steps)
    for i in range(N):
        index.append((odb.steps.keys()[i],i))
    index = OrderedDict(index)
    return(index)

def step_selection(manual=False):
    step_list=[]
    odb = session.odbs[session.odbData.keys()[0]]
    N=len(odb.steps)
    if manual == False:
        step_list=list(np.linspace(0,N-1,N))
        for i in range(N):
            step_list[i]=int(step_list[i])
        return(step_list)
    if manual == True:
        already=[]
        n=int(getInput("How many steps do you consider?"))
        for i in range(n):
            step=int(getInput("Which step? "))
            while belong(already,step) == True:
                step=int(getInput("Already selected! Choose a new one
:"))
            already.append(step-1)
            step_list.append(step-1)
        return(step_list)

def MX11_files(manual=False):
    import visualization
    import xyPlot
    import displayGroupOdbToolset as dgo
    os.chdir(dirpath+"/%s"%create_folder())
    cylinder_list = cylinder_selection(manual)
    step_list = step_selection(manual)
    leaf = dgo.LeafFromPartInstance(partInstanceName=('LIN-1', ))
    session.viewports['Viewport:
1'].odbDisplay.displayGroup.replace(leaf=leaf)
    odb = session.odbs[session.odbData.keys()[0]]
    scratchOdb = session.ScratchOdb(odb)
    if belong(cylinder_list,1) == True:
        node1 = odb.rootAssembly.instances['LIN-1'].nodes[87368]
        node2 = odb.rootAssembly.instances['LIN-1'].nodes[86376]
        node3 = odb.rootAssembly.instances['LIN-1'].nodes[86306]
        scratchOdb.rootAssembly.DatumCsysByThreeCircNodes(name='CSYS-
1',
        coordSysType=CYLINDRICAL, node1Arc=node1, node2Arc=node2,
        node3Arc=node3)
        dtm = session.scratchOdbs[session.odbData.keys()[0]]
.rootAssembly.datumCsyses['CSYS-1']
        session.viewports['Viewport:
1'].odbDisplay.basicOptions.setValues(
        transformationType=USER_SPECIFIED, datumCsys=dtm)
        session.viewerOptions.setValues(deformedVariableCaching=False,
        primaryVariableCaching=False, cutVariableCaching=False)
        for i in range(len(step_list)):
            session.linkedViewportCommands.setValues
(_highlightLinkedViewports=True)
            leaf = dgo.LeafFromOdbNodePick(nodePick=(('LIN-1', 9252, (
```

```
            '[#0:1267 #ffff0000 #ffffffff:160 #7ffff #0:7 #fffffff8
#ffffffff:60',
            ' #1ff #0:2 #fffffc00 #ffffffff:4 #1fffff #0:1106
#ffff0000',
            ' #ffffffff:8 #3ffff #0:10 #7f00000 #0:5 #18 #0:2',
            ' #fffff800 #ffffffff:5 #7fff #0:2 #100 #0:5 #fffffff8',
            ' #ffffffff #3ff #0:4 #ffff8000 #ffffffff:2 #1ffffff #0:9',
            ' #fffffc00 #ffffffff:10 #7ffffff #0:8 #c7ff0000
#ffffffff:2 #1ffffff',
            ' #0:2 #3000 #0:2 #38000000 #0 #4010842 #80210401',
            ' #84208020 #ffffff00 #ffffffff:7 #3fffff #0 #ffde0000
#ffffffff',
            ' #ff #0:2 #fffffffe #ffffffff:6 #fff #0:3 #f8000000',
            ' #ffffffff:7 #ffff ]', )), ), )
            session.viewports['Viewport:
1'].odbDisplay.displayGroup.replace(leaf=leaf)
            dg = session.DisplayGroup(leaf=leaf, name='DisplayGroup-2')
            dg1= session.displayGroups['DisplayGroup-2']
            session.viewports['Viewport:
1'].odbDisplay.setValues(visibleDisplayGroups=(
            dg1, ))
            session.viewports['Viewport:
1'].odbDisplay.setFrame(step=step_list[i], frame=1)
            odb = session.odbs[session.odbData.keys()[0]]
            session.fieldReportOptions.setValues(separateTables=ON,
printTotal=OFF,
            printMinMax=OFF)

session.writeFieldReport(fileName='MX11_cyl1_step%s.txt'%(step_list[i]+
1), append=OFF,
            sortItem='Node Label', odb=odb, step=step_list[i], frame=1,
outputPosition=NODAL,
            variable=(('U', NODAL, ((COMPONENT, 'U1'), )), ))

    if belong(cylinder_list,2) == True:
        node1 = odb.rootAssembly.instances['LIN-1'].nodes[77505]
        node2 = odb.rootAssembly.instances['LIN-1'].nodes[77255]
        node3 = odb.rootAssembly.instances['LIN-1'].nodes[77175]
        scratchOdb.rootAssembly.DatumCsysByThreeCircNodes(name='CSYS-
2',
        coordSysType=CYLINDRICAL, node1Arc=node1, node2Arc=node2,
        node3Arc=node3)
        dtm = session.scratchOdbs[session.odbData.keys()[0]]
.rootAssembly.datumCsyses['CSYS-2']
        session.viewports['Viewport:
1'].odbDisplay.basicOptions.setValues(
        transformationType=USER_SPECIFIED, datumCsys=dtm)
        session.viewerOptions.setValues(deformedVariableCaching=False,
        primaryVariableCaching=False, cutVariableCaching=False)
        for i in range(len(step_list)):
            session.linkedViewportCommands.setValues
(_highlightLinkedViewports=True)
            leaf = dgo.LeafFromOdbNodePick(nodePick=(('LIN-1', 9252, (
            '[#0:279 #fff00000 #ffffffff:153 #fff #0:6 #f0000000
#ffffffff:73',
            ' #ffffff #0:1895 #fffffffe #ffffffff:3 #3ffff #0:4
#fffffffc',
            ' #ffffffff:3 #1ffff #0:4 #ffffffff:52 #ffff ]', )), ), )
```

```
            session.viewports['Viewport:
1'].odbDisplay.displayGroup.replace(leaf=leaf)
            dg = session.DisplayGroup(leaf=leaf, name='DisplayGroup-2')
            dg1= session.displayGroups['DisplayGroup-2']
            session.viewports['Viewport:
1'].odbDisplay.setValues(visibleDisplayGroups=(
            dg1, ))
            session.viewports['Viewport:
1'].odbDisplay.setFrame(step=step_list[i], frame=1)
            odb = session.odbs[session.odbData.keys()[0]]
            session.fieldReportOptions.setValues(separateTables=ON,
printTotal=OFF,
            printMinMax=OFF)
            session.writeFieldReport(fileName='MX11_cyl2_step%s.txt'%
(step_list[i]+1), append=OFF,
            sortItem='Node Label', odb=odb, step=step_list[i], frame=1,
outputPosition=NODAL,
            variable=(('U', NODAL, ((COMPONENT, 'U1'), )), ))

    if belong(cylinder_list,3) == True:
        node1 = odb.rootAssembly.instances['LIN-1'].nodes[79570]
        node2 = odb.rootAssembly.instances['LIN-1'].nodes[80194]
        node3 = odb.rootAssembly.instances['LIN-1'].nodes[79752]
        scratchOdb.rootAssembly.DatumCsysByThreeCircNodes(name='CSYS-
3',
        coordSysType=CYLINDRICAL, node1Arc=node1, node2Arc=node2,
        node3Arc=node3)
        dtm = session.scratchOdbs[session.odbData.keys()[0]]
.rootAssembly.datumCsyses['CSYS-3']
        session.viewports['Viewport:
1'].odbDisplay.basicOptions.setValues(
        transformationType=USER_SPECIFIED, datumCsys=dtm)
        session.viewerOptions.setValues(deformedVariableCaching=False,
        primaryVariableCaching=False, cutVariableCaching=False)
        for i in range(len(step_list)):
            session.linkedViewportCommands.setValues
(_highlightLinkedViewports=True)
            leaf = dgo.LeafFromOdbNodePick(nodePick=(('LIN-1', 9252, (
            '[#0:1690 #ff000000 #ffffffff:5 #7 #0:2 #fffffff0
#ffffffff:60',
            ' #3ff #0:6 #fc000000 #ffffffff:160 #1fffffff #0:555
#80000000',
            ' #ffffffff:4 #3fff #0:2 #ffffff80 #3dffffff #0:9 #6',
            ' #0 #ffc00000 #ffffffff:2 #1ffff #0:28 #2000000 #0:5',
            ' #6000000 #0:5 #3f8 #0:10 #fffff000 #ffffffff:51 #ffff ]',
)), ), )
            session.viewports['Viewport:
1'].odbDisplay.displayGroup.replace(leaf=leaf)
            dg = session.DisplayGroup(leaf=leaf, name='DisplayGroup-2')
            dg1= session.displayGroups['DisplayGroup-2']
            session.viewports['Viewport:
1'].odbDisplay.setValues(visibleDisplayGroups=(
            dg1, ))
            session.viewports['Viewport:
1'].odbDisplay.setFrame(step=step_list[i], frame=1)
            odb = session.odbs[session.odbData.keys()[0]]
            session.fieldReportOptions.setValues(separateTables=ON,
printTotal=OFF,
```

```
            printMinMax=OFF)
            session.writeFieldReport(fileName='MX11_cyl3_step%s.txt'%
(step_list[i]+1), append=OFF,
            sortItem='Node Label', odb=odb, step=step_list[i], frame=1,
outputPosition=NODAL,
            variable=(('U', NODAL, ((COMPONENT, 'U1'), )), ))
           session.viewports['Viewport: 1'].odbDisplay.setFrame
(step=0, frame=1)
    print("Files generation successfully done")

def MX13_files(manual = False):
    import visualization
    import xyPlot
    import displayGroupOdbToolset as dgo
    os.chdir(dirpath+"/%s"%create_folder())
    cylinder_list = cylinder_selection(manual)
    step_list = step_selection(manual)
    leaf = dgo.LeafFromPartInstance(partInstanceName=('LINER-1', ))
    session.viewports['Viewport:
1'].odbDisplay.displayGroup.replace(leaf=leaf)
    odb = session.odbs[session.odbData.keys()[0]]
    scratchOdb = session.ScratchOdb(odb)
    if belong(cylinder_list,1) == True:
        node1 = odb.rootAssembly.instances['LINER-1'].nodes[14433]
        node2 = odb.rootAssembly.instances['LINER-1'].nodes[14345]
        node3 = odb.rootAssembly.instances['LINER-1'].nodes[14389]
        scratchOdb.rootAssembly.DatumCsysByThreeCircNodes(name='CSYS-
1',
        coordSysType=CYLINDRICAL, node1Arc=node1, node2Arc=node2,
        node3Arc=node3)
        dtm = session.scratchOdbs[session.odbData.keys()[0]]
.rootAssembly.datumCsyses['CSYS-1']
        session.viewports['Viewport:
1'].odbDisplay.basicOptions.setValues(
        transformationType=USER_SPECIFIED, datumCsys=dtm)
        session.viewerOptions.setValues(deformedVariableCaching=False,
        primaryVariableCaching=False, cutVariableCaching=False)
        for i in range(len(step_list)):
            session.linkedViewportCommands.setValues
(_highlightLinkedViewports=True)
            leaf = dgo.LeafFromOdbNodePick(nodePick=(('LINER-1', 8208,
(
            '[#0:315 #fffffffe #ffffffff #3ff #0 #fffc0000 #ffffffff',
            ' #1ffffff #0:125 #ff800000 #ffffffff:4 #7f #0:126
#fffffc00',
            ' #ffffffff:246 #3ffffff ]', )), ), )
            session.viewports['Viewport:
1'].odbDisplay.displayGroup.replace(leaf=leaf)
            dg = session.DisplayGroup(leaf=leaf, name='DisplayGroup-2')
            dg1= session.displayGroups['DisplayGroup-2']
            session.viewports['Viewport:
1'].odbDisplay.setValues(visibleDisplayGroups=(
            dg1, ))
            session.viewports['Viewport:
1'].odbDisplay.setFrame(step=step_list[i], frame=1)
            odb = session.odbs[session.odbData.keys()[0]]
            session.fieldReportOptions.setValues(separateTables=ON,
printTotal=OFF,
```

```
            printMinMax=OFF)
            session.writeFieldReport(fileName='MX13_cyl1_step%s.txt'%
(step_list[i]+1), append=OFF,
            sortItem='Node Label', odb=odb, step=step_list[i], frame=1,
outputPosition=NODAL,
            variable=(('U', NODAL, ((COMPONENT, 'U1'), )), ))

    if belong(cylinder_list,2) == True:
        node1 = odb.rootAssembly.instances['LINER-1'].nodes[69383]
        node2 = odb.rootAssembly.instances['LINER-1'].nodes[69297]
        node3 = odb.rootAssembly.instances['LINER-1'].nodes[69339]
        scratchOdb.rootAssembly.DatumCsysByThreeCircNodes(name='CSYS-
2',
        coordSysType=CYLINDRICAL, node1Arc=node1, node2Arc=node2,
        node3Arc=node3)
        dtm =
session.scratchOdbs[session.odbData.keys()[0]].rootAssembly.datumCsyses
['CSYS-2']
        session.viewports['Viewport:
1'].odbDisplay.basicOptions.setValues(
        transformationType=USER_SPECIFIED, datumCsys=dtm)
        session.viewerOptions.setValues(deformedVariableCaching=False,
        primaryVariableCaching=False, cutVariableCaching=False)
        for i in range(len(step_list)):
            session.linkedViewportCommands.setValues
(_highlightLinkedViewports=True)
            leaf = dgo.LeafFromOdbNodePick(nodePick=(('LINER-1', 8208,
(
            '[#0:2032 #ffffffe0 #ffffffff #3fff #0 #ffc00000
#ffffffff',
            ' #1fffffff #0:125 #f8000000 #ffffffff:4 #7ff #0:126
#ffffc000',
            ' #ffffffff:246 #3fffffff ]', )), ), )
            session.viewports['Viewport:
1'].odbDisplay.displayGroup.replace(leaf=leaf)
            dg = session.DisplayGroup(leaf=leaf, name='DisplayGroup-2')
            dg1= session.displayGroups['DisplayGroup-2']
            session.viewports['Viewport:
1'].odbDisplay.setValues(visibleDisplayGroups=(
            dg1, ))
            session.viewports['Viewport:
1'].odbDisplay.setFrame(step=step_list[i], frame=1)
            odb = session.odbs[session.odbData.keys()[0]]
            session.fieldReportOptions.setValues(separateTables=ON,
printTotal=OFF,
            printMinMax=OFF)
            session.writeFieldReport(fileName='MX13_cyl2_step%s.txt'%
(step_list[i]+1), append=OFF,
            sortItem='Node Label', odb=odb, step=step_list[i], frame=1,
outputPosition=NODAL,
            variable=(('U', NODAL, ((COMPONENT, 'U1'), )), ))

    if belong(cylinder_list,3) == True:
        node1 = odb.rootAssembly.instances['LINER-1'].nodes[124327]
        node2 = odb.rootAssembly.instances['LINER-1'].nodes[124249]
        node3 = odb.rootAssembly.instances['LINER-1'].nodes[124283]
        scratchOdb.rootAssembly.DatumCsysByThreeCircNodes(name='CSYS-
3',
```

```
        coordSysType=CYLINDRICAL, node1Arc=node1, node2Arc=node2,
        node3Arc=node3)
        dtm = session.scratchOdbs[session.odbData.keys()[0]]
.rootAssembly.datumCsyses['CSYS-3']
        session.viewports['Viewport:
1'].odbDisplay.basicOptions.setValues(
        transformationType=USER_SPECIFIED, datumCsys=dtm)
        session.viewerOptions.setValues(deformedVariableCaching=False,
        primaryVariableCaching=False, cutVariableCaching=False)
        for i in range(len(step_list)):
            session.linkedViewportCommands.setValues
(_highlightLinkedViewports=True)
            leaf = dgo.LeafFromOdbNodePick(nodePick=(('LINER-1', 8208,
(
            '[#0:3749 #fffffe00 #ffffffff #3ffff #0 #fc000000
#ffffffff:2',
            ' #1 #0:124 #80000000 #ffffffff:4 #7fff #0:126 #fffc0000',
            ' #ffffffff:247 #3 ]', )), ), )
            session.viewports['Viewport:
1'].odbDisplay.displayGroup.replace(leaf=leaf)
            dg = session.DisplayGroup(leaf=leaf, name='DisplayGroup-2')
            dg1= session.displayGroups['DisplayGroup-2']
            session.viewports['Viewport:
1'].odbDisplay.setValues(visibleDisplayGroups=(
            dg1, ))
            session.viewports['Viewport:
1'].odbDisplay.setFrame(step=step_list[i], frame=1)
            odb = session.odbs[session.odbData.keys()[0]]
            session.fieldReportOptions.setValues(separateTables=ON,
printTotal=OFF,
            printMinMax=OFF)
            session.writeFieldReport(fileName='MX13_cyl3_step%s.txt'
%(step_list[i]+1), append=OFF,
            sortItem='Node Label', odb=odb, step=step_list[i], frame=1,
outputPosition=NODAL,
            variable=(('U', NODAL, ((COMPONENT, 'U1'), )), ))
    session.viewports['Viewport: 1'].odbDisplay.setFrame(step=0,
frame=1)
    print("Files generation successfully done")
```

Temperature_generate_files script

```
import os
import numpy as np
from collections import OrderedDict
from abaqus import *
from abaqusConstants import *
import __main__
from abaqus import getInput

dirpath = "C:/Users/A-Damien.Gode/Documents/Distortion
program/Analysis"

os.chdir(dirpath)

def create_folder():
    os.chdir(dirpath)
    odb = session.odbs[session.odbData.keys()[0]]
    tmp=odb.name.split()[-1]
    name=''
    index=0
    for i in range(len(tmp)):
        if tmp[i] == '/':
            index=i
    name=tmp[index+1:len(tmp)]
    name = name[0:len(name)-4]
    if os.path.exists(name) == False:
        os.makedirs(name)
    return(name)

def belong(List,val):
    for i in range(len(List)):
        if List[i]==val:
            return(True)
    return(False)

def cylinder_selection(manual=False):
    cylinder_list=[]
    if manual == False:
        cylinder_list=[1,2,3]
        return(cylinder_list)
    if manual == True:
        already=[]
        N=int(getInput("How many cylinders do you consider?"))
        for i in range(N):
            cyl=int(getInput("Which cylinder?"))
            while belong(already,cyl) == True:
                cyl=int(getInput("Already selected! Choose a new one
:"))
            already.append(cyl)
            cylinder_list.append(cyl)
        return(cylinder_list)

def step_index():
    index=[]
    odb = session.odbs[session.odbData.keys()[0]]
```

```
    N=len(odb.steps)
    for i in range(N):
        index.append((odb.steps.keys()[i],i))
    index = OrderedDict(index)
    return(index)

def step_selection(manual=False):
    step_list=[]
    odb = session.odbs[session.odbData.keys()[0]]
    N=len(odb.steps)
    if manual == False:
        step_list=list(np.linspace(0,N-1,N))
        for i in range(N):
            step_list[i]=int(step_list[i])
        return(step_list)
    if manual == True:
        already=[]
        n=int(getInput("How many steps do you consider?"))
        for i in range(n):
            step=int(getInput("Which step?"))
            while belong(already,step) == True:
                step=int(getInput("Already selected! Choose a new one
:"))
            already.append(step-1)
            step_list.append(step-1)
        return(step_list)

def MX11_Temperature_files(manual=False):
    import visualization
    import xyPlot
    import displayGroupOdbToolset as dgo
    os.chdir(dirpath+"/%s"%create_folder())
    cylinder_list = cylinder_selection(manual)
    step_list = step_selection(manual)
    leaf = dgo.LeafFromPartInstance(partInstanceName=('LIN-1', ))
    session.viewports['Viewport:
1'].odbDisplay.displayGroup.replace(leaf=leaf)
    odb = session.odbs[session.odbData.keys()[0]]
    scratchOdb = session.ScratchOdb(odb)

    if belong(cylinder_list,1) == True:
        session.viewerOptions.setValues(deformedVariableCaching=False,
        primaryVariableCaching=False, cutVariableCaching=False)
        for i in range(len(step_list)):
            session.linkedViewportCommands.setValues
(_highlightLinkedViewports=True)
            leaf = dgo.LeafFromOdbNodePick(nodePick=(('FR-1', 1584, (
            '[#0:96 #fffffffe #ffffffff #3fffff #0:4 #ffffffe0
#ffffffff:3',
            ' #1fffff #0:2 #ffc00000 #ffffffff #1ffff #0:154
#ffffffff:6',
            ' #0:143 #f0000000 #ffffffff:6 #7fffffff #0:11 #e0000000
#ffffffff:2',
            ' #fffffff #0:5 #7fc00000 #0:14 #f0000000 #ffffffff:6 #3f',
            ' #0 #fffffffe #ffffffff:16 #3ffffff #0:105 #18 ]', )),
('LIN-1', 9252,
            ('[#0:1267 #ffff0000 #ffffffff:160 #7ffff #0:7 #fffffff8
#ffffffff:60',
```

```
            ' #1ff #0:2 #fffffc00 #ffffffff:4 #1fffff #0:1106
#ffff0000',
            ' #ffffffff:8 #3ffff #0:10 #7f00000 #0:5 #18 #0:2',
            ' #fffff800 #ffffffff:5 #7fff #0:2 #100 #0:5 #fffffff8',
            ' #ffffffff #3ff #0:4 #ffff8000 #ffffffff:2 #1ffffff #0:9',
            ' #fffffc00 #ffffffff:10 #7ffffff #0:8 #c7ff0000
#ffffffff:2 #1ffffff',
            ' #0:2 #3000 #0:2 #38000000 #0 #4010842 #80210401',
            ' #84208020 #ffffff00 #ffffffff:7 #3fffff #0 #ffde0000
#ffffffff',
            ' #ff #0:2 #fffffffe #ffffffff:6 #fff #0:3 #f8000000',
            ' #ffffffff:7 #ffff ]', )), ), )
            session.viewports['Viewport:
1'].odbDisplay.displayGroup.replace(leaf=leaf)
            dg = session.DisplayGroup(leaf=leaf, name='DisplayGroup-2')
            dg1= session.displayGroups['DisplayGroup-2']
            session.viewports['Viewport:
1'].odbDisplay.setValues(visibleDisplayGroups=(
            dg1, ))
            session.viewports['Viewport:
1'].odbDisplay.setFrame(step=step_list[i], frame=1)
            odb = session.odbs[session.odbData.keys()[0]]
            session.fieldReportOptions.setValues(separateTables=ON,
printTotal=OFF,
            printMinMax=OFF)
            session.writeFieldReport(fileName=
'TEMP_MX11_cyl1_step%s.txt'%(step_list[i]+1), append=OFF,
            sortItem='Node Label', odb=odb, step=step_list[i], frame=1,
outputPosition=NODAL,
            variable=(('NT11', NODAL), ))

    if belong(cylinder_list,2) == True:
        session.viewerOptions.setValues(deformedVariableCaching=False,
        primaryVariableCaching=False, cutVariableCaching=False)
        for i in range(len(step_list)):
            session.linkedViewportCommands.setValues
(_highlightLinkedViewports=True)
            leaf = dgo.LeafFromOdbNodePick(nodePick=(('FR-1', 1584, (
            '[#0:60 #fffffffe #ffffffff:3 #3ffff #0:4 #fffffffc
#ffffffff:3',
            ' #1ffff #0:187 #ffffffff:6 #0:46 #ffff0000 #ffffffff:33
#7fffffff',
            ' #0:142 #80000000 ]', )), ('LIN-1', 9252, (
            '[#0:279 #fff00000 #ffffffff:153 #fff #0:6 #f0000000
#ffffffff:73',
            ' #ffffff #0:1895 #fffffffe #ffffffff:3 #3ffff #0:4
#fffffffc',
            ' #ffffffff:3 #1ffff #0:4 #ffffffff:52 #ffff ]', )), ), )
            session.viewports['Viewport:
1'].odbDisplay.displayGroup.replace(leaf=leaf)
            dg = session.DisplayGroup(leaf=leaf, name='DisplayGroup-2')
            dg1= session.displayGroups['DisplayGroup-2']
            session.viewports['Viewport:
1'].odbDisplay.setValues(visibleDisplayGroups=(
            dg1, ))
            session.viewports['Viewport:
1'].odbDisplay.setFrame(step=step_list[i], frame=1)
            odb = session.odbs[session.odbData.keys()[0]]
```

```
            session.fieldReportOptions.setValues(separateTables=ON,
printTotal=OFF,
            printMinMax=OFF)

session.writeFieldReport(fileName='TEMP_MX11_cyl2_step%s.txt'
%(step_list[i]+1), append=OFF,
            sortItem='Node Label', odb=odb, step=step_list[i], frame=1,
outputPosition=NODAL,
            variable=(('NT11', NODAL), ))

    if belong(cylinder_list,3) == True:
        session.viewerOptions.setValues(deformedVariableCaching=False,
        primaryVariableCaching=False, cutVariableCaching=False)
        for i in range(len(step_list)):
            session.linkedViewportCommands.setValues
(_highlightLinkedViewports=True)
            leaf = dgo.LeafFromOdbNodePick(nodePick=(('FR-1', 1584, (
            '[#0:199 #ffff8000 #ffffffff #3ff #0:2 #fffff800
#ffffffff:3',
            ' #7ffffff #0:4 #fffffc00 #ffffffff #7fffffff #0:57
#ffffffff:6',
            ' #0:91 #fffe0000 #ffffffff:2 #ffff #0:11 #ffffc000
#ffffffff:30',
            ' #fffffff #0:278 #70000 ]', )), ('LIN-1', 9252, (
            '[#0:1690 #ff000000 #ffffffff:5 #7 #0:2 #fffffff0
#ffffffff:60',
            ' #3ff #0:6 #fc000000 #ffffffff:160 #1fffffff #0:555
#80000000',
            ' #ffffffff:4 #3fff #0:2 #ffffff80 #3dffffff #0:9 #6',
            ' #0 #ffc00000 #ffffffff:2 #1ffff #0:28 #2000000 #0:5',
            ' #6000000 #0:5 #3f8 #0:10 #fffff000 #ffffffff:51 #ffff ]',
)), ), )
            session.viewports['Viewport:
1'].odbDisplay.displayGroup.replace(leaf=leaf)
            dg = session.DisplayGroup(leaf=leaf, name='DisplayGroup-2')
            dg1= session.displayGroups['DisplayGroup-2']
            session.viewports['Viewport:
1'].odbDisplay.setValues(visibleDisplayGroups=(
            dg1, ))
            session.viewports['Viewport:
1'].odbDisplay.setFrame(step=step_list[i], frame=1)
            odb = session.odbs[session.odbData.keys()[0]]
            session.fieldReportOptions.setValues(separateTables=ON,
printTotal=OFF,
            printMinMax=OFF)

session.writeFieldReport(fileName='TEMP_MX11_cyl3_step%s.txt'
%(step_list[i]+1), append=OFF,
            sortItem='Node Label', odb=odb, step=step_list[i], frame=1,
outputPosition=NODAL,
            variable=(('NT11', NODAL), ))
    session.viewports['Viewport: 1'].odbDisplay.setFrame(step=0,
frame=1)
    print("Files generation successfully done")

def MX13_Temperature_files(manual=False):
    import visualization
    import xyPlot
```

```
    import displayGroupOdbToolset as dgo
    os.chdir(dirpath+"/%s"%create_folder())
    cylinder_list = cylinder_selection(manual)
    step_list = step_selection(manual)
    leaf = dgo.LeafFromPartInstance(partInstanceName=('LINER-1', ))
    session.viewports['Viewport:
1'].odbDisplay.displayGroup.replace(leaf=leaf)
    odb = session.odbs[session.odbData.keys()[0]]
    scratchOdb = session.ScratchOdb(odb)
    if belong(cylinder_list,1) == True:
        session.viewerOptions.setValues(deformedVariableCaching=False,
        primaryVariableCaching=False, cutVariableCaching=False)
        for i in range(len(step_list)):
            session.linkedViewportCommands.setValues
(_highlightLinkedViewports=True)
            leaf = dgo.LeafFromOdbNodePick(nodePick=(('FIRE-RING-1',
1296, (
            '[#0:45 #ffffffff:31 #ffff #0:17 #ffff0000 #ffffffff:8
#ffff ]', )), (
            'LINER-1', 8208, (
            '[#0:315 #fffffffe #ffffffff #3ff #0 #fffc0000 #ffffffff',
            ' #1ffffff #0:125 #ff800000 #ffffffff:4 #7f #0:126
#fffffc00',
            ' #ffffffff:246 #3ffffff ]', )), ), )
            session.viewports['Viewport:
1'].odbDisplay.displayGroup.replace(leaf=leaf)
            dg = session.DisplayGroup(leaf=leaf, name='DisplayGroup-2')
            dg1= session.displayGroups['DisplayGroup-2']
            session.viewports['Viewport:
1'].odbDisplay.setValues(visibleDisplayGroups=(
            dg1, ))
            session.viewports['Viewport:
1'].odbDisplay.setFrame(step=step_list[i], frame=1)
            odb = session.odbs[session.odbData.keys()[0]]
            session.fieldReportOptions.setValues(separateTables=ON,
printTotal=OFF,
            printMinMax=OFF)

session.writeFieldReport(fileName='TEMP_MX13_cyl1_step%s.txt'
%(step_list[i]+1), append=OFF,
            sortItem='Node Label', odb=odb, step=step_list[i], frame=1,
outputPosition=NODAL,
            variable=(('NT11', NODAL), ))

    if belong(cylinder_list,2) == True:
        session.viewerOptions.setValues(deformedVariableCaching=False,
        primaryVariableCaching=False, cutVariableCaching=False)
        for i in range(len(step_list)):
            session.linkedViewportCommands.setValues
(_highlightLinkedViewports=True)
            leaf = dgo.LeafFromOdbNodePick(nodePick=(('FIRE-RING-1',
1296, (
            '[#0:124 #f8000000 #ffffffff:8 #7ffffff #0:7 #ff800000
#ffffffff:31',
            ' #7f ]', )), ('LINER-1', 8208, (
            '[#0:2032 #fffffe0 #ffffffff #3fff #0 #ffc00000
#ffffffff',
```

```
            ' #1fffffff #0:125 #f8000000 #ffffffff:4 #7ff #0:126
#ffffc000',
            ' #ffffffff:246 #3fffffff ]', )), ), )
            session.viewports['Viewport:
1'].odbDisplay.displayGroup.replace(leaf=leaf)
            dg = session.DisplayGroup(leaf=leaf, name='DisplayGroup-2')
            dg1= session.displayGroups['DisplayGroup-2']
            session.viewports['Viewport:
1'].odbDisplay.setValues(visibleDisplayGroups=(
            dg1, ))
            session.viewports['Viewport:
1'].odbDisplay.setFrame(step=step_list[i], frame=1)
            odb = session.odbs[session.odbData.keys()[0]]
            session.fieldReportOptions.setValues(separateTables=ON,
printTotal=OFF,
            printMinMax=OFF)

session.writeFieldReport(fileName='TEMP_MX13_cyl2_step%s.txt'
%(step_list[i]+1), append=OFF,
            sortItem='Node Label', odb=odb, step=step_list[i], frame=1,
outputPosition=NODAL,
            variable=(('NT11', NODAL), ))

    if belong(cylinder_list,3) == True:
        session.viewerOptions.setValues(deformedVariableCaching=False,
        primaryVariableCaching=False, cutVariableCaching=False)
        for i in range(len(step_list)):
            session.linkedViewportCommands.setValues
(_highlightLinkedViewports=True)
            leaf = dgo.LeafFromOdbNodePick(nodePick=(('FIRE-RING-1',
1296, (
            '[#0:196 #f8000000 #ffffffff:8 #7ffffff #0:7 #ff800000
#ffffffff:31',
            ' #7f ]', )), ('LINER-1', 8208, (
            '[#0:3749 #fffffe00 #ffffffff #3ffff #0 #fc000000
#ffffffff:2',
            ' #1 #0:124 #80000000 #ffffffff:4 #7fff #0:126 #fffc0000',
            ' #ffffffff:247 #3 ]', )), ), )
            session.viewports['Viewport:
1'].odbDisplay.displayGroup.replace(leaf=leaf)
            dg = session.DisplayGroup(leaf=leaf, name='DisplayGroup-2')
            dg1= session.displayGroups['DisplayGroup-2']
            session.viewports['Viewport:
1'].odbDisplay.setValues(visibleDisplayGroups=(
            dg1, ))
            session.viewports['Viewport:
1'].odbDisplay.setFrame(step=step_list[i], frame=1)
            odb = session.odbs[session.odbData.keys()[0]]
            session.fieldReportOptions.setValues(separateTables=ON,
printTotal=OFF,
            printMinMax=OFF)

session.writeFieldReport(fileName='TEMP_MX13_cyl3_step%s.txt'
%(step_list[i]+1), append=OFF,
            sortItem='Node Label', odb=odb, step=step_list[i], frame=1,
outputPosition=NODAL,
            variable=(('NT11', NODAL), ))
```

```
    session.viewports['Viewport: 1'].odbDisplay.setFrame(step=0,
frame=1)
    print("Files generation successfully done")
```

Program_interface script

```
import os
import fnmatch

dirpath = r"C:\Users\A-Damien.Gode\Documents\Distortion
program\Analysis"

import tkinter as t

os.chdir(dirpath)

from BoreDistortion_Analysis_V7 import *
from Temperature_Analysis import *
from Comparison_interface import *

files=os.listdir(dirpath)

pattern = '*.*'
folder=[]
for i in range(len(files)):
    if fnmatch.fnmatch(files[i],pattern) == False and files[i] !=
"__pycache__":
        folder.append(files[i])

check_dict = {}
check_dict["j03_03-ST_maxBL_WE-CE_SC_NO-lin"] = "MX11"
check_dict["j03_03-ST_minBL_WE-CE_SC_NO"] = "MX11"
check_dict["j03_03-ST_nomBL_WE-CE_SC_NO"] = "MX11"
check_dict["MX13-MY21-230bar"] = "MX13"
check_dict["MX13-MY21-230bar-topfit"] = "MX13"

height_check_dict = {}
height_check_dict["MX11"] = 65
height_check_dict["MX13"] = 57

step_check_dict = {}
step_check_dict["MX11"] = 14
step_check_dict["MX13"] = 12

def program():

    def SetValues():
        return(int(CylinderBox.get()),int(StepBox.get())
,int(HeightBox.get()))

    def apply_plot_path():
        if Engine.get() != check_dict[Folder.get()]:
            ErrorTextBox("Error : Wrong engine \n Please set the engine
button")
            return
        if (SetValues()[0]<=0 or SetValues()[0]>3):
            ErrorTextBox("Error : Wrong cylinder value \n Must be
between 1 and 3")
            return
```

```
        if (Engine.get() == "MX11" and (SetValues()[1]<=0 or
SetValues()[1]>step_check_dict["MX11"])):
            ErrorTextBox("Error : Wrong step value \n Look at the
explanation note to get more details")
            return
        if (Engine.get() == "MX13" and (SetValues()[1]<=0 or
SetValues()[1]>step_check_dict["MX13"])):
            ErrorTextBox("Error : Wrong step value \n Look at the
explanation note to get more details")
            return
        if (Engine.get() == "MX11" and (SetValues()[2]<=0 or
SetValues()[2]>height_check_dict["MX11"])):
            ErrorTextBox("Error : Wrong height value \n Look at the
explanation note to get more details")
            return
        if (Engine.get() == "MX13" and (SetValues()[2]<=0 or
SetValues()[2]>height_check_dict["MX13"])):
            ErrorTextBox("Error : Wrong height value \n Look at the
explanation note to get more details")
            return
        return(plot_path(Engine.get(),SetValues()[0],SetValues()[1]
,SetValues()[2],Folder.get(),0))

    def apply_plot_def_profile():
        if Engine.get() != check_dict[Folder.get()]:
            ErrorTextBox("Error : Wrong engine \n Please set the engine
button")
            return
        if (SetValues()[0]<=0 or SetValues()[0]>3):
            ErrorTextBox("Error : Wrong cylinder value \n Must be
between 1 and 3")
            return
        if (Engine.get() == "MX11" and (SetValues()[1]<=0 or
SetValues()[1]>step_check_dict["MX11"])):
            ErrorTextBox("Error : Wrong step value \n Look at the
explanation note to get more details")
            return
        if (Engine.get() == "MX13" and (SetValues()[1]<=0 or
SetValues()[1]>step_check_dict["MX13"])):
            ErrorTextBox("Error : Wrong step value \n Look at the
explanation note to get more details")
            return
        if (Engine.get() == "MX11" and (SetValues()[2]<=0 or
SetValues()[2]>height_check_dict["MX11"])):
            ErrorTextBox("Error : Wrong height value \n Look at the
explanation note to get more details")
            return
        if (Engine.get() == "MX13" and (SetValues()[2]<=0 or
SetValues()[2]>height_check_dict["MX13"])):
            ErrorTextBox("Error : Wrong height value \n Look at the
explanation note to get more details")
            return
        return(plot_def_profile(Engine.get(),SetValues()[0]
,SetValues()[1],SetValues()[2],Folder.get(),0))

    def apply_plot_def_profile_by_Fourier():
        if Engine.get() != check_dict[Folder.get()]:
```

```
            ErrorTextBox("Error : Wrong engine \n Please set the engine
button")
            return
        if (SetValues()[0]<=0 or SetValues()[0]>3):
            ErrorTextBox("Error : Wrong cylinder value \n Must be
between 1 and 3")
            return
        if (Engine.get() == "MX11" and (SetValues()[1]<=0 or
SetValues()[1]>step_check_dict["MX11"])):
            ErrorTextBox("Error : Wrong step value \n Look at the
explanation note to get more details")
            return
        if (Engine.get() == "MX13" and (SetValues()[1]<=0 or
SetValues()[1]>step_check_dict["MX13"])):
            ErrorTextBox("Error : Wrong step value \n Look at the
explanation note to get more details")
            return
        if (Engine.get() == "MX11" and (SetValues()[2]<=0 or
SetValues()[2]>height_check_dict["MX11"])):
            ErrorTextBox("Error : Wrong height value \n Look at the
explanation note to get more details")
            return
        if (Engine.get() == "MX13" and (SetValues()[2]<=0 or
SetValues()[2]>height_check_dict["MX13"])):
            ErrorTextBox("Error : Wrong height value \n Look at the
explanation note to get more details")
            return
        return(plot_def_profile_by_Fourier(Engine.get(),
SetValues()[0],SetValues()[1],SetValues()[2],Folder.get(),0))

    def apply_plot_distortion_by_order():
        if Engine.get() != check_dict[Folder.get()]:
            ErrorTextBox("Error : Wrong engine \n Please set the engine
button")
            return
        if (SetValues()[0]<=0 or SetValues()[0]>3):
            ErrorTextBox("Error : Wrong cylinder value \n Must be
between 1 and 3")
            return
        if (Engine.get() == "MX11" and (SetValues()[1]<=0 or
SetValues()[1]>step_check_dict["MX11"])):
            ErrorTextBox("Error : Wrong step value \n Look at the
explanation note to get more details")
            return
        if (Engine.get() == "MX13" and (SetValues()[1]<=0 or
SetValues()[1]>step_check_dict["MX13"])):
            ErrorTextBox("Error : Wrong step value \n Look at the
explanation note to get more details")
            return
        return(plot_distortion_by_order(Engine.get(),SetValues()[0]
,SetValues()[1],Folder.get(),0))

    def apply_plot_distortion_graph():
        if Engine.get() != check_dict[Folder.get()]:
            ErrorTextBox("Error : Wrong engine \n Please set the engine
button")
            return
        if (SetValues()[0]<=0 or SetValues()[0]>3):
```

```
            ErrorTextBox("Error : Wrong cylinder value \n Must be
between 1 and 3")
            return
        if (Engine.get() == "MX11" and (SetValues()[1]<=0 or
SetValues()[1]>step_check_dict["MX11"])):
            ErrorTextBox("Error : Wrong step value \n Look at the
explanation note to get more details")
            return
        if (Engine.get() == "MX13" and (SetValues()[1]<=0 or
SetValues()[1]>step_check_dict["MX13"])):
            ErrorTextBox("Error : Wrong step value \n Look at the
explanation note to get more details")
            return
        if (Engine.get() == "MX11" and (SetValues()[2]<=0 or
SetValues()[2]>height_check_dict["MX11"])):
            ErrorTextBox("Error : Wrong height value \n Look at the
explanation note to get more details")
            return
        if (Engine.get() == "MX13" and (SetValues()[2]<=0 or
SetValues()[2]>height_check_dict["MX13"])):
            ErrorTextBox("Error : Wrong height value \n Look at the
explanation note to get more details")
            return
        return(plot_distortion_graph(Engine.get(),SetValues()[0]
,SetValues()[1],SetValues()[2],Folder.get(),0))

    def apply_plot_cut_profile():
        if Engine.get() != check_dict[Folder.get()]:
            ErrorTextBox("Error : Wrong engine \n Please set the engine
button")
            return
        if (SetValues()[0]<=0 or SetValues()[0]>3):
            ErrorTextBox("Error : Wrong cylinder value \n Must be
between 1 and 3")
            return
        if (Engine.get() == "MX11" and (SetValues()[1]<=0 or
SetValues()[1]>step_check_dict["MX11"])):
            ErrorTextBox("Error : Wrong step value \n Look at the
explanation note to get more details")
            return
        if (Engine.get() == "MX13" and (SetValues()[1]<=0 or
SetValues()[1]>step_check_dict["MX13"])):
            ErrorTextBox("Error : Wrong step value \n Look at the
explanation note to get more details")
            return
        return(plot_cut_profile(Engine.get(),SetValues()[0]
,SetValues()[1],Folder.get(),0))

    def apply_plot_wall_temperature_complete():
        if Engine.get() != check_dict[Folder.get()]:
            ErrorTextBox("Error : Wrong engine \n Please set the engine
button")
            return
        if (Engine.get() == "MX11" and (SetValues()[1]<=0 or
SetValues()[1]>step_check_dict["MX11"])):
            ErrorTextBox("Error : Wrong step value \n Look at the
explanation note to get more details")
            return
```

```
        if (Engine.get() == "MX13" and (SetValues()[1]<=0 or
SetValues()[1]>step_check_dict["MX13"])):
            ErrorTextBox("Error : Wrong step value \n Look at the
explanation note to get more details")
            return
        return(plot_wall_temperature_complete(Engine.get()
,SetValues()[1],Folder.get()))

    def AngleSelectionBox():

        def SetAngle():
            ang = Angle.get()
            return(plot_wall_temperature_single(Engine.get(),
SetValues()[1],Folder.get(),ang))

        if Engine.get() != check_dict[Folder.get()]:
            ErrorTextBox("Error : Wrong engine \n Please set the engine
button")
            return
        if (Engine.get() == "MX11" and (SetValues()[1]<=0 or
SetValues()[1]>step_check_dict["MX11"])):
            ErrorTextBox("Error : Wrong step value \n Look at the
explanation note to get more details")
            return
        if (Engine.get() == "MX13" and (SetValues()[1]<=0 or
SetValues()[1]>step_check_dict["MX13"])):
            ErrorTextBox("Error : Wrong step value \n Look at the
explanation note to get more details")
            return
        ValWindow=t.Tk()
        ValWindow.geometry("200x100")
        Angle_leg=t.Label(ValWindow,text="Select Angle")
        Angle_leg.pack()
        Angle_leg.place(relx=0.5, rely=0.2, anchor='center')
        Angle=t.IntVar(ValWindow)
        Angle.set(0)
        Angle_choice=t.OptionMenu(ValWindow,Angle,
0,45,90,135,180,225,270,315)
        Angle_choice.pack()
        Angle_choice.place(relx=0.5, rely=0.45, anchor='center')
        OKButton=t.Button(ValWindow,text="OK",command = SetAngle)
        OKButton.pack()
        OKButton.place(relx=0.3, rely=0.8, anchor='center')
        QuitButton=t.Button(ValWindow,text="Quit",command =
ValWindow.destroy)
        QuitButton.pack()
        QuitButton.place(relx=0.7, rely=0.8, anchor='center')

    def Delete_figures():
        plt.close("all")

    def ErrorTextBox(message):
        N = len(message)
        Ewindow=t.Tk()
        Ewindow.geometry('%sx100'%((N+20)*4))
        Message=t.Label(Ewindow,text=message)
        Message.pack()
        Message.place(relx=0.5, rely=0.4, anchor='center')
```

```
        OKButton=t.Button(Ewindow,text="OK",command = Ewindow.destroy)
        OKButton.pack()
        OKButton.place(relx=0.5, rely=0.8, anchor='center')


def comparison_method():
    comparison_window(window)

    window = t.Tk()
    window.geometry('500x500')
    label = t.Label(window, text="Liner Distortion and Thermal
Analysis")
    label.config(font=("Courrier",20))
    label.pack()

    Engine_leg=t.Label(window,text="Engine")
    Engine_leg.pack()
    Engine_leg.place(relx=0.25, rely=0.15, anchor='center')

    Engine=t.StringVar(window)
    Engine.set("MX11")
    Engine_choice=t.OptionMenu(window,Engine,"MX11","MX13")
    Engine_choice.pack()
    Engine_choice.place(relx=0.25, rely=0.2, anchor='center')

    Folder_leg=t.Label(window,text="Folder")
    Folder_leg.pack()
    Folder_leg.place(relx=0.65, rely=0.15, anchor='center')

    Folder=t.StringVar(window)
    Folder.set("%s"%folder[0])
    Folder_choice=t.OptionMenu(window,Folder,*folder)
    Folder_choice.pack()
    Folder_choice.place(relx=0.65, rely=0.2, anchor='center')

    Cylinder_leg=t.Label(window,text="Cylinder")
    Cylinder_leg.pack()
    Cylinder_leg.place(relx=0.25, rely=0.3, anchor='center')

    CylinderVar=t.IntVar(window)
    CylinderBox=t.Entry(window,textvariable=CylinderVar)
    CylinderBox.pack()
    CylinderBox.place(relx=0.25, rely=0.35, anchor='center')

    Step_leg=t.Label(window,text="Step")
    Step_leg.pack()
    Step_leg.place(relx=0.5, rely=0.3, anchor='center')

    StepVar=t.IntVar(window)
    StepBox=t.Entry(window,textvariable=StepVar)
    StepBox.pack()
    StepBox.place(relx=0.5, rely=0.35, anchor='center')

    Height_leg=t.Label(window,text="Height")
    Height_leg.pack()
    Height_leg.place(relx=0.75, rely=0.3, anchor='center')

    HeightVar=t.IntVar(window)
```

```
    HeightBox=t.Entry(window,textvariable=HeightVar)
    HeightBox.pack()
    HeightBox.place(relx=0.75, rely=0.35, anchor='center')

    button=t.Button(window,text="Quit",command = window.destroy)
    button.pack()
    button.place(relx=0.9, rely=0.9, anchor='center')

    button1=t.Button(window,text="Plot radial distortion along a
path",command = apply_plot_path)
    button1.pack()
    button1.place(relx=0.36, rely=0.45, anchor='center')

    button2=t.Button(window,text="Plot deformed profile",command =
apply_plot_def_profile)
    button2.pack()
    button2.place(relx=0.68, rely=0.45, anchor='center')

    button3=t.Button(window,text="Plot deformed profile from Fourier
coefficients",command = apply_plot_def_profile_by_Fourier)
    button3.pack()
    button3.place(relx=0.5, rely=0.51, anchor='center')

    button4=t.Button(window,text="Plot axial position with respect to
diametrical distortion",command = apply_plot_distortion_by_order)
    button4.pack()
    button4.place(relx=0.5, rely=0.57, anchor='center')

    button5=t.Button(window,text="Plot diametrical distortion with
respect to harmonic order",command = apply_plot_distortion_graph)
    button5.pack()
    button5.place(relx=0.5, rely=0.63, anchor='center')

    button6=t.Button(window,text="Plot axial position with respect to
radial distortion",command = apply_plot_cut_profile)
    button6.pack()
    button6.place(relx=0.5, rely=0.69, anchor='center')

    button7=t.Button(window,text="Plot wall temperature (all
angles)",command = apply_plot_wall_temperature_complete)
    button7.pack()
    button7.place(relx=0.31, rely=0.75, anchor='center')

    button8=t.Button(window,text="Plot wall temperature (one
angle)",command = AngleSelectionBox)
    button8.pack()
    button8.place(relx=0.69, rely=0.75, anchor='center')

    button9=t.Button(window,text="Delete figures",command =
Delete_figures)
    button9.pack()
    button9.place(relx=0.5, rely=0.87, anchor='center')

    button10=t.Button(window,text="Comparison method",command =
comparison_method)
    button10.pack()
    button10.place(relx=0.5,rely=0.81,anchor='center')
```

```
        window.update()
    #   window.mainloop()

program()
```

GenerateFiles_interface script

```
import os
import fnmatch
import Tkinter as T
import abaqus

dirpath = "C:\Users\A-Damien.Gode\Documents\Distortion
program\Analysis"
Modelpath = "C:/Users/A-Damien.Gode/Documents/Distortion program/Odb
files"

os.chdir(dirpath)

from Explicative_Note import *
from distortion_generate_files import *
from Temperature_generate_files import *
from bridge_file import *

os.chdir(Modelpath)

files=os.listdir(Modelpath)
pattern = '*.odb'
Modelfolder=[]
for i in range(len(files)):
    if fnmatch.fnmatch(files[i],pattern) == True:
        Modelfolder.append(files[i])

os.chdir(dirpath)

check_dict = {}
check_dict["j03_03-ST_maxBL_WE-CE_SC_NO-lin.odb"] = "MX11"
check_dict["j03_03-ST_minBL_WE-CE_SC_NO.odb"] = "MX11"
check_dict["j03_03-ST_nomBL_WE-CE_SC_NO.odb"] = "MX11"
check_dict["MX13-MY21-230bar.odb"] = "MX13"
check_dict["MX13-MY21-230bar-topfit.odb"] = "MX13"

window = T.Tk()
window.geometry("600x250")
label = T.Label(window, text="Generate files")
label.config(font=("Courrier",20))
label.pack()

Engine_leg=T.Label(window,text="Engine")
Engine_leg.pack()
Engine_leg.place(relx=0.1, rely=0.45, anchor='center')

Engine=T.StringVar(window)
Engine.set("MX11")
Engine_choice=T.OptionMenu(window,Engine,"MX11","MX13")
Engine_choice.pack()
Engine_choice.place(relx=0.1, rely=0.55, anchor='center')

Mode_leg=T.Label(window,text="Mode")
Mode_leg.pack()
Mode_leg.place(relx=0.28, rely=0.45, anchor='center')
```

```
Mode=T.StringVar(window)
Mode.set("Automatic")
Mode_choice=T.OptionMenu(window,Mode,"Manual","Automatic")
Mode_choice.pack()
Mode_choice.place(relx=0.28, rely=0.55, anchor='center')

Remark = T.Label(window, text = "Remark : Don't forget \n to reset
before changing \n the model")
Remark.pack()
Remark.place(relx=0.3, rely=0.3, anchor='center')

name=""
index=0

if  session.odbData.items() == []:
    name = "No model"
else:
    odb = session.odbs[session.odbData.keys()[0]]
    tmp=odb.name.split()[-1]
    for i in range(len(tmp)):
        if tmp[i] == '/':
            index=i
    name=tmp[index+1:len(tmp)]

Model_leg=T.Label(window,text="Model")
Model_leg.pack()
Model_leg.place(relx=0.65, rely=0.2, anchor='center')

Model=T.StringVar(window)
Model.set(name)
Model_choice=T.OptionMenu(window,Model,*Modelfolder)
Model_choice.pack()
Model_choice.place(relx=0.65, rely=0.3, anchor='center')


def reset():
    if  session.odbData.items() == []:
        ErrorTextBox("Error : There is no model open in session")
    import visualization
    import xyPlot
    import displayGroupOdbToolset as dgo
    odb = session.odbs[session.odbData.keys()[0]]
    odb.close()

def OpenFile():
    if Model.get() == "No model":
        ErrorTextBox("Error : You must select a model")
    else:
        import visualization
        import xyPlot
        import displayGroupOdbToolset as dgo
        o1 = session.openOdb(name= Modelpath + "/%s"%Model.get())
        session.viewports['Viewport: 1'].setValues(displayedObject=o1)
        session.viewports['Viewport: 1'].odbDisplay.setFrame(step=0,
frame=1)
        odb = session.odbs[Modelpath + "\%s"%Model.get()]
```

```
def gen_dist_files():
    if Model.get() == "No model":
        ErrorTextBox("Error : You must open a model")
    if Engine.get() != check_dict[Model.get()]:
        ErrorTextBox("Error : Wrong engine \n Please set the engine
button")
        return
    if Engine.get() == 'MX11' and Mode.get() == "Automatic":
        return(MX11_files(False))
    if Engine.get() == 'MX11' and Mode.get() == "Manual":
        return(MX11_files(True))
    if Engine.get() == 'MX13' and Mode.get() == "Automatic":
        return(MX13_files(False))
    if Engine.get() == 'MX13' and Mode.get() == "Manual":
        return(MX13_files(True))

def gen_temp_files():
    if Model.get() == "No model":
        ErrorTextBox("Error : You must open a model")
    if Engine.get() != check_dict[Model.get()]:
        ErrorTextBox("Error : Wrong engine \n Please set the engine
button")
        return
    if Engine.get() == 'MX11' and Mode.get() == "Automatic":
        return(MX11_Temperature_files(False))
    if Engine.get() == 'MX11' and Mode.get() == "Manual":
        return(MX11_Temperature_files(True))
    if Engine.get() == 'MX13' and Mode.get() == "Automatic":
        return(MX13_Temperature_files(False))
    if Engine.get() == 'MX13' and Mode.get() == "Manual":
        return(MX13_Temperature_files(True))

def gen_expl_note():
    if Model.get() == "No model":
        ErrorTextBox("Error : You must open a model")
    return(create_explanation_note(Engine.get()))

def launch_program():
    window.destroy()
    ENV = os.environ
    ENV['TCL_LIBRARY'] = r"C:\Users\A-
Damien.Gode\AppData\Local\Enthought\Canopy\edm\envs\User\tcl\tcl8.6"
    return(bridge())

def ErrorTextBox(message):
    N = len(message)
    Ewindow=T.Tk()
    Ewindow.geometry('%sx100'%((N+20)*4))
    Message=T.Label(Ewindow,text=message)
    Message.pack()
    Message.place(relx=0.5, rely=0.4, anchor='center')
    OKButton=T.Button(Ewindow,text="OK",command = Ewindow.destroy)
    OKButton.pack()
    OKButton.place(relx=0.5, rely=0.8, anchor='center')

ResetButton=T.Button(window,text="Reset",command = reset)
ResetButton.pack()
ResetButton.place(relx=0.1, rely=0.3, anchor='center')
```

```
OpenButton=T.Button(window,text="Open",command = OpenFile)
OpenButton.pack()
OpenButton.place(relx=0.9, rely=0.3, anchor='center')

Qbutton=T.Button(window,text="Quit",command = window.destroy)
Qbutton.pack()
Qbutton.place(relx=0.9, rely=0.9, anchor='center')

Button=T.Button(window,text="Generate distortion files",command =
gen_dist_files)
Button.pack()
Button.place(relx=0.53, rely=0.55, anchor='center')

Button1=T.Button(window,text="Generate temperature files",command =
gen_temp_files)
Button1.pack()
Button1.place(relx=0.82, rely=0.55, anchor='center')

Button2=T.Button(window,text="Generate explicative note",command =
gen_expl_note)
Button2.pack()
Button2.place(relx=0.3, rely=0.8, anchor='center')

Button3=T.Button(window,text="Launch program",command = launch_program)
Button3.pack()
Button3.place(relx=0.7, rely=0.8, anchor='center')

window.update()
window.mainloop()
```

Comparison_interface script

```
import os
import fnmatch

dirpath = r"C:\Users\A-Damien.Gode\Documents\Distortion
program\Analysis"

import tkinter as t
import matplotlib.pyplot as plt

os.chdir(dirpath)

from BoreDistortion_Analysis_V7 import *
from Program_interface import *

files=os.listdir(dirpath)


pattern = '*.*'
folder=[]
for i in range(len(files)):
    if fnmatch.fnmatch(files[i],pattern) == False and files[i] !=
"__pycache__":
        folder.append(files[i])

check_dict = {}
check_dict["j03_03-ST_maxBL_WE-CE_SC_NO-lin"] = "MX11"
check_dict["j03_03-ST_minBL_WE-CE_SC_NO"] = "MX11"
check_dict["j03_03-ST_nomBL_WE-CE_SC_NO"] = "MX11"
check_dict["MX13-MY21-230bar"] = "MX13"
check_dict["MX13-MY21-230bar-topfit"] = "MX13"

height_check_dict = {}
height_check_dict["MX11"] = 65
height_check_dict["MX13"] = 57

step_check_dict = {}
step_check_dict["MX11"] = 14
step_check_dict["MX13"] = 12

def comparison_window(active_window):
    active_window.destroy()
    window = t.Tk()
    window.geometry('500x500')
    label = t.Label(window, text="Comparison mode")
    label.config(font=("Courrier",20))
    label.pack()

    Engine1_leg=t.Label(window,text="Engine 1")
    Engine1_leg.pack()
    Engine1_leg.place(relx=0.25, rely=0.1, anchor='center')

    Engine1=t.StringVar(window)
    Engine1.set("MX11")
    Engine1_choice=t.OptionMenu(window,Engine1,"MX11","MX13")
    Engine1_choice.pack()
    Engine1_choice.place(relx=0.25, rely=0.15, anchor='center')

    Engine2_leg=t.Label(window,text="Engine 2")
```

```
Engine2_leg.pack()
Engine2_leg.place(relx=0.75, rely=0.1, anchor='center')

Engine2=t.StringVar(window)
Engine2.set("MX11")
Engine2_choice=t.OptionMenu(window,Engine2,"MX11","MX13")
Engine2_choice.pack()
Engine2_choice.place(relx=0.75, rely=0.15, anchor='center')

Folder1_leg=t.Label(window,text="Folder 1")
Folder1_leg.pack()
Folder1_leg.place(relx=0.25, rely=0.21, anchor='center')

Folder1=t.StringVar(window)
Folder1.set("%s"%folder[0])
Folder1_choice=t.OptionMenu(window,Folder1,*folder)
Folder1_choice.pack()
Folder1_choice.place(relx=0.25, rely=0.26, anchor='center')

Folder2_leg=t.Label(window,text="Folder 2")
Folder2_leg.pack()
Folder2_leg.place(relx=0.75, rely=0.21, anchor='center')

Folder2=t.StringVar(window)
Folder2.set("%s"%folder[0])
Folder2_choice=t.OptionMenu(window,Folder2,*folder)
Folder2_choice.pack()
Folder2_choice.place(relx=0.75, rely=0.26, anchor='center')

Cylinder_leg1=t.Label(window,text="Cylinder 1")
Cylinder_leg1.pack()
Cylinder_leg1.place(relx=0.25, rely=0.32, anchor='center')

CylinderVar1=t.IntVar(window)
CylinderBox1=t.Entry(window,textvariable=CylinderVar1)
CylinderBox1.pack()
CylinderBox1.place(relx=0.25, rely=0.37, anchor='center')

Step_leg1=t.Label(window,text="Step 1")
Step_leg1.pack()
Step_leg1.place(relx=0.5, rely=0.32, anchor='center')

StepVar1=t.IntVar(window)
StepBox1=t.Entry(window,textvariable=StepVar1)
StepBox1.pack()
StepBox1.place(relx=0.5, rely=0.37, anchor='center')

Height_leg1=t.Label(window,text="Height 1")
Height_leg1.pack()
Height_leg1.place(relx=0.75, rely=0.32, anchor='center')

HeightVar1=t.IntVar(window)
HeightBox1=t.Entry(window,textvariable=HeightVar1)
HeightBox1.pack()
HeightBox1.place(relx=0.75, rely=0.37, anchor='center')

Cylinder_leg2=t.Label(window,text="Cylinder 2")
Cylinder_leg2.pack()
```

```
    Cylinder_leg2.place(relx=0.25, rely=0.42, anchor='center')

    CylinderVar2=t.IntVar(window)
    CylinderBox2=t.Entry(window,textvariable=CylinderVar2)
    CylinderBox2.pack()
    CylinderBox2.place(relx=0.25, rely=0.47, anchor='center')

    Step_leg2=t.Label(window,text="Step 2")
    Step_leg2.pack()
    Step_leg2.place(relx=0.5, rely=0.42, anchor='center')

    StepVar2=t.IntVar(window)
    StepBox2=t.Entry(window,textvariable=StepVar2)
    StepBox2.pack()
    StepBox2.place(relx=0.5, rely=0.47, anchor='center')

    Height_leg2=t.Label(window,text="Height 2")
    Height_leg2.pack()
    Height_leg2.place(relx=0.75, rely=0.42, anchor='center')

    HeightVar2=t.IntVar(window)
    HeightBox2=t.Entry(window,textvariable=HeightVar2)
    HeightBox2.pack()
    HeightBox2.place(relx=0.75, rely=0.47, anchor='center')

    def SetValues1():
        return(int(CylinderBox1.get()),int(StepBox1.get()),
int(HeightBox1.get()))

    def SetValues2():
        return(int(CylinderBox2.get()),int(StepBox2.get()),
int(HeightBox2.get()))

    def apply_plot_path():
        if Folder1.get() == Folder2.get():
            ErrorTextBox("Error : Identical model")
            return
        if Engine1.get() != check_dict[Folder1.get()]:
            ErrorTextBox("Error : Wrong engine \n Please set the engine
button")
            return
        if Engine2.get() != check_dict[Folder2.get()]:
            ErrorTextBox("Error : Wrong engine \n Please set the engine
button")
            return
        if (SetValues1()[0]<=0 or SetValues1()[0]>3):
            ErrorTextBox("Error : Wrong cylinder value \n Must be
between 1 and 3")
            return
        if (SetValues2()[0]<=0 or SetValues2()[0]>3):
            ErrorTextBox("Error : Wrong cylinder value \n Must be
between 1 and 3")
            return
        if (Engine1.get() == "MX11" and (SetValues1()[1]<=0 or
SetValues1()[1]>step_check_dict["MX11"])):
            ErrorTextBox("Error : Wrong step value \n Look at the
explanation note to get more details")
            return
```

```
        if (Engine1.get() == "MX13" and (SetValues1()[1]<=0 or
SetValues1()[1]>step_check_dict["MX13"])):
            ErrorTextBox("Error : Wrong step value \n Look at the
explanation note to get more details")
            return
        if (Engine1.get() == "MX11" and (SetValues1()[2]<=0 or
SetValues1()[2]>height_check_dict["MX11"])):
            ErrorTextBox("Error : Wrong height value \n Look at the
explanation note to get more details")
            return
        if (Engine1.get() == "MX13" and (SetValues1()[2]<=0 or
SetValues1()[2]>height_check_dict["MX13"])):
            ErrorTextBox("Error : Wrong height value \n Look at the
explanation note to get more details")
            return
        if (Engine2.get() == "MX11" and (SetValues2()[1]<=0 or
SetValues2()[1]>step_check_dict["MX11"])):
            ErrorTextBox("Error : Wrong step value \n Look at the
explanation note to get more details")
            return
        if (Engine2.get() == "MX13" and (SetValues2()[1]<=0 or
SetValues2()[1]>step_check_dict["MX13"])):
            ErrorTextBox("Error : Wrong step value \n Look at the
explanation note to get more details")
            return
        if (Engine2.get() == "MX11" and (SetValues2()[2]<=0 or
SetValues2()[2]>height_check_dict["MX11"])):
            ErrorTextBox("Error : Wrong height value \n Look at the
explanation note to get more details")
            return
        if (Engine2.get() == "MX13" and (SetValues2()[2]<=0 or
SetValues2()[2]>height_check_dict["MX13"])):
            ErrorTextBox("Error : Wrong height value \n Look at the
explanation note to get more details")
            return
        path1,def_R1,H1=plot_path(Engine1.get(),SetValues1()[0],
SetValues1()[1],SetValues1()[2],Folder1.get(),1)
        path2,def_R2,H2=plot_path(Engine2.get(),SetValues2()[0],
SetValues2()[1],SetValues2()[2],Folder2.get(),1)
        plt.figure()
        plt.title("Comparison of the radial deformation along a path
between \n Engine %s Cylinder %s Step %s z = %s mm \n Engine %s
Cylinder %s Step %s z = %s mm
"%(Engine1.get(),SetValues1()[0],SetValues1()[1],H1[SetValues1()[2]-
1][0],Engine2.get(),SetValues2()[0],SetValues2()[1],H2[SetValues2()[2]-
1][0]))
        plt.plot(path1,def_R1)
        plt.plot(path2,def_R2)
        plt.grid()
        plt.xlabel("Angular position [deg]")
        plt.ylabel("Radial deformation [mm]")
        plt.legend(('%s'%Folder1.get(),'%s'%Folder2.get()),
loc=1,fontsize=10)
        plt.show()

    def apply_plot_def_profile():
        if Folder1.get() == Folder2.get():
            ErrorTextBox("Error : Identical model")
```

```
            return
        if Engine1.get() != check_dict[Folder1.get()]:
            ErrorTextBox("Error : Wrong engine \n Please set the engine
button")
            return
        if Engine2.get() != check_dict[Folder2.get()]:
            ErrorTextBox("Error : Wrong engine \n Please set the engine
button")
            return
        if (SetValues1()[0]<=0 or SetValues1()[0]>3):
            ErrorTextBox("Error : Wrong cylinder value \n Must be
between 1 and 3")
            return
        if (SetValues2()[0]<=0 or SetValues2()[0]>3):
            ErrorTextBox("Error : Wrong cylinder value \n Must be
between 1 and 3")
            return
        if (Engine1.get() == "MX11" and (SetValues1()[1]<=0 or
SetValues1()[1]>step_check_dict["MX11"])):
            ErrorTextBox("Error : Wrong step value \n Look at the
explanation note to get more details")
            return
        if (Engine1.get() == "MX13" and (SetValues1()[1]<=0 or
SetValues1()[1]>step_check_dict["MX13"])):
            ErrorTextBox("Error : Wrong step value \n Look at the
explanation note to get more details")
            return
        if (Engine1.get() == "MX11" and (SetValues1()[2]<=0 or
SetValues1()[2]>height_check_dict["MX11"])):
            ErrorTextBox("Error : Wrong height value \n Look at the
explanation note to get more details")
            return
        if (Engine1.get() == "MX13" and (SetValues1()[2]<=0 or
SetValues1()[2]>height_check_dict["MX13"])):
            ErrorTextBox("Error : Wrong height value \n Look at the
explanation note to get more details")
            return
        if (Engine2.get() == "MX11" and (SetValues2()[1]<=0 or
SetValues2()[1]>step_check_dict["MX11"])):
            ErrorTextBox("Error : Wrong step value \n Look at the
explanation note to get more details")
            return
        if (Engine2.get() == "MX13" and (SetValues2()[1]<=0 or
SetValues2()[1]>step_check_dict["MX13"])):
            ErrorTextBox("Error : Wrong step value \n Look at the
explanation note to get more details")
            return
        if (Engine2.get() == "MX11" and (SetValues2()[2]<=0 or
SetValues2()[2]>height_check_dict["MX11"])):
            ErrorTextBox("Error : Wrong height value \n Look at the
explanation note to get more details")
            return
        if (Engine2.get() == "MX13" and (SetValues2()[2]<=0 or
SetValues2()[2]>height_check_dict["MX13"])):
            ErrorTextBox("Error : Wrong height value \n Look at the
explanation note to get more details")
            return
        scale_fact=100
```

```
theta1,dR1,H1=COMP_plot_def_profile(Engine1.get(),SetValues1()[0]
,SetValues1()[1],SetValues1()[2],Folder1.get(),1)[0:3]

theta2,dR2,H2=COMP_plot_def_profile(Engine2.get(),SetValues2()[0]
,SetValues2()[1],SetValues2()[2],Folder2.get(),1)[0:3]
        deltaR1=0
        count1=0
        deltaR2=0
        count2=0
        radius1=diameter(Engine1.get())/2
        radius2=diameter(Engine2.get())/2
        iniR1=[radius1]*len(dR1)
        iniR2=[radius2]*len(dR2)
        for i in range(len(dR1)):
            if abs(dR1[i]-iniR1[i])>deltaR1:
                deltaR1=abs(dR1[i]-iniR1[i])
                count1=i
        for i in range(len(dR2)):
            if abs(dR2[i]-iniR2[i])>deltaR2:
                deltaR2=abs(dR2[i]-iniR2[i])
                count2=i
        plt.figure()
        plt.subplot(111,projection = 'polar')
        line1,=plt.plot(theta1,iniR1)
        line2,=plt.plot(theta2,iniR2)
        line3,=plt.plot(theta1,dR1)
        line4,=plt.plot(theta2,dR2)
        plt.text(m.pi/6,120," Engine 1, z = %s mm \n Engine 2, z = %s
mm"%(H1[SetValues1()[2]-1][0],H2[SetValues2()[2]-1][0]),fontsize=15.0)
        if 0<=theta1[count1]<=m.pi/2:
            plt.annotate(" %s maximal gap = %s
µm"%(Folder1.get(),deltaR1*10),xy=(theta1[count1],dR1[count1]),xytext=(
theta1[count1],dR1[count1]+10),arrowprops=dict(facecolor='black',
shrink=0.05))
        if m.pi/2<theta1[count1]<3*m.pi/2:
            plt.annotate(" %s maximal gap = %s
µm"%(Folder1.get(),deltaR1*10),xy=(theta1[count1],dR1[count1]),xytext=(
theta1[count1],dR1[count1]+30),arrowprops=dict(facecolor='black',
shrink=0.05))
        if 3*m.pi/2<=theta1[count1]<=2*m.pi:
            plt.annotate(" %s maximal gap = %s
µm"%(Folder1.get(),deltaR1*10),xy=(theta1[count1],dR1[count1]),xytext=(
theta1[count1],dR1[count1]+10),arrowprops=dict(facecolor='black',
shrink=0.05))
        if 0<=theta2[count2]<=m.pi/2:
            plt.annotate(" %s maximal gap = %s
µm"%(Folder2.get(),deltaR2*10),xy=(theta2[count2],dR2[count2]),xytext=(
theta2[count2],dR2[count2]+15),arrowprops=dict(facecolor='black',
shrink=0.05))
        if m.pi/2<theta2[count2]<3*m.pi/2:
            plt.annotate(" %s maximal gap = %s
µm"%(Folder2.get(),deltaR2*10),xy=(theta2[count2],dR2[count2]),xytext=(
theta2[count2],dR2[count2]+35),arrowprops=dict(facecolor='black',
shrink=0.05))
        if 3*m.pi/2<=theta2[count2]<=2*m.pi:
            plt.annotate(" %s maximal gap = %s
µm"%(Folder2.get(),deltaR2*10),xy=(theta2[count2],dR2[count2]),xytext=(
```

```
theta2[count2],dR2[count2]+15),arrowprops=dict(facecolor='black',
shrink=0.05))
        plt.grid(True)
        plt.title("Comparison of the deformed and original profile
between  \n Engine %s Cylinder %s Step %s \n Engine %s Cylinder %s Step
%s \n Scale Factor
%s"%(Engine1.get(),SetValues1()[0],SetValues1()[1],Engine2.get(),SetVal
ues2()[0],SetValues2()[1],scale_fact))

plt.figlegend(handles=[line1,line2,line3,line4],labels=('Original
profile %s'%Engine1.get(),'Original profile
%s'%Engine2.get(),'%s'%Folder1.get(),'%s'%Folder2.get()),loc='upper
right',fontsize=12)
        plt.show()

    def apply_plot_def_profile_by_Fourier():
        if Folder1.get() == Folder2.get():
            ErrorTextBox("Error : Identical model")
            return
        if Engine1.get() != check_dict[Folder1.get()]:
            ErrorTextBox("Error : Wrong engine \n Please set the engine
button")
            return
        if Engine2.get() != check_dict[Folder2.get()]:
            ErrorTextBox("Error : Wrong engine \n Please set the engine
button")
            return
        if (SetValues1()[0]<=0 or SetValues1()[0]>3):
            ErrorTextBox("Error : Wrong cylinder value \n Must be
between 1 and 3")
            return
        if (SetValues2()[0]<=0 or SetValues2()[0]>3):
            ErrorTextBox("Error : Wrong cylinder value \n Must be
between 1 and 3")
            return
        if (Engine1.get() == "MX11" and (SetValues1()[1]<=0 or
SetValues1()[1]>step_check_dict["MX11"])):
            ErrorTextBox("Error : Wrong step value \n Look at the
explanation note to get more details")
            return
        if (Engine1.get() == "MX13" and (SetValues1()[1]<=0 or
SetValues1()[1]>step_check_dict["MX13"])):
            ErrorTextBox("Error : Wrong step value \n Look at the
explanation note to get more details")
            return
        if (Engine1.get() == "MX11" and (SetValues1()[2]<=0 or
SetValues1()[2]>height_check_dict["MX11"])):
            ErrorTextBox("Error : Wrong height value \n Look at the
explanation note to get more details")
            return
        if (Engine1.get() == "MX13" and (SetValues1()[2]<=0 or
SetValues1()[2]>height_check_dict["MX13"])):
            ErrorTextBox("Error : Wrong height value \n Look at the
explanation note to get more details")
            return
        if (Engine2.get() == "MX11" and (SetValues2()[1]<=0 or
SetValues2()[1]>step_check_dict["MX11"])):
```

```
            ErrorTextBox("Error : Wrong step value \n Look at the
explanation note to get more details")
            return
        if (Engine2.get() == "MX13" and (SetValues2()[1]<=0 or
SetValues2()[1]>step_check_dict["MX13"])):
            ErrorTextBox("Error : Wrong step value \n Look at the
explanation note to get more details")
            return
        if (Engine2.get() == "MX11" and (SetValues2()[2]<=0 or
SetValues2()[2]>height_check_dict["MX11"])):
            ErrorTextBox("Error : Wrong height value \n Look at the
explanation note to get more details")
            return
        if (Engine2.get() == "MX13" and (SetValues2()[2]<=0 or
SetValues2()[2]>height_check_dict["MX13"])):
            ErrorTextBox("Error : Wrong height value \n Look at the
explanation note to get more details")
            return
        theta1,defR1,H1=plot_def_profile_by_Fourier(Engine1.get()
,SetValues1()[0],SetValues1()[1],SetValues1()[2],Folder1.get(),1)
        theta2,defR2,H2=plot_def_profile_by_Fourier(Engine2.get()
,SetValues2()[0],SetValues2()[1],SetValues2()[2],Folder2.get(),1)
        deltaR1=0
        count1=0
        deltaR2=0
        count2=0
        radius1=diameter(Engine1.get())/2
        radius2=diameter(Engine2.get())/2
        iniR1=[radius1]*len(defR1)
        iniR2=[radius2]*len(defR2)
        for i in range(len(defR1)):
            if abs(defR1[i]-iniR1[i])>deltaR1:
                deltaR1=abs(defR1[i]-iniR1[i])
                count1=i
        for i in range(len(defR2)):
            if abs(defR2[i]-iniR2[i])>deltaR2:
                deltaR2=abs(defR2[i]-iniR2[i])
                count2=i
        plt.figure()
        plt.subplot(111,projection = 'polar')
        line1,=plt.plot(theta1,iniR1)
        line2,=plt.plot(theta2,iniR2)
        line3,=plt.plot(theta1,defR1)
        line4,=plt.plot(theta2,defR2)
        plt.text(m.pi/6,100," Engine 1, z = %s mm \n Engine 2, z = %s
mm"%(H1[SetValues1()[2]-1][0],H2[SetValues2()[2]-1][0]),fontsize=15.0)
        if 0<=theta1[count1]<=m.pi/2:
            plt.annotate(" %s maximal gap = %s
µm"%(Folder1.get(),deltaR1),xy=(theta1[count1],defR1[count1]),xytext=(t
heta1[count1],defR1[count1]+10),arrowprops=dict(facecolor='black',
shrink=0.05))
        if m.pi/2<theta1[count1]<3*m.pi/2:
            plt.annotate(" %s maximal gap = %s
µm"%(Folder1.get(),deltaR1),xy=(theta1[count1],defR1[count1]),xytext=(t
heta1[count1],defR1[count1]+30),arrowprops=dict(facecolor='black',
shrink=0.05))
        if 3*m.pi/2<=theta1[count1]<=2*m.pi:
```

```
            plt.annotate(" %s maximal gap = %s
µm"%(Folder1.get(),deltaR1),xy=(theta1[count1],defR1[count1]),xytext=(t
heta1[count1],defR1[count1]+10),arrowprops=dict(facecolor='black',
shrink=0.05))
        if 0<=theta2[count2]<=m.pi/2:
            plt.annotate(" %s maximal gap = %s
µm"%(Folder2.get(),deltaR2),xy=(theta2[count2],defR2[count2]),xytext=(t
heta2[count2],defR2[count2]+15),arrowprops=dict(facecolor='black',
shrink=0.05))
        if m.pi/2<theta2[count2]<3*m.pi/2:
            plt.annotate(" %s maximal gap = %s
µm"%(Folder2.get(),deltaR2),xy=(theta2[count2],defR2[count2]),xytext=(t
heta2[count1],defR2[count2]+35),arrowprops=dict(facecolor='black',
shrink=0.05))
        if 3*m.pi/2<=theta2[count2]<=2*m.pi:
            plt.annotate(" %s maximal gap = %s
µm"%(Folder2.get(),deltaR2),xy=(theta2[count2],defR2[count2]),xytext=(t
heta2[count2],defR2[count2]+15),arrowprops=dict(facecolor='black',
shrink=0.05))
        plt.grid(True)
        plt.title(" Comparison of deformed profile plotted from Fourier
coefficients without zeroth and first order between \n Engine %s
Cylinder %s Step %s \n Engine %s Cylinder %s Step %s \n Scale Factor
%s"%(Engine1.get(),SetValues1()[0],SetValues1()[1],Engine2.get(),SetVal
ues2()[0],SetValues2()[1],1000))

plt.figlegend(handles=[line1,line2,line3,line4],labels=('Original
profile %s'%Engine1.get(),'Original profile
%s'%Engine2.get(),'%s'%Folder1.get(),'%s'%Folder2.get()),loc='upper
right',fontsize=12)
        plt.show()


    def apply_plot_distortion_by_order():
        if Folder1.get() == Folder2.get():
            ErrorTextBox("Error : Identical model")
            return
        if Engine1.get() != check_dict[Folder1.get()]:
            ErrorTextBox("Error : Wrong engine \n Please set the engine
button")
            return
        if Engine2.get() != check_dict[Folder2.get()]:
            ErrorTextBox("Error : Wrong engine \n Please set the engine
button")
            return
        if (SetValues1()[0]<=0 or SetValues1()[0]>3):
            ErrorTextBox("Error : Wrong cylinder value \n Must be
between 1 and 3")
            return
        if (SetValues2()[0]<=0 or SetValues2()[0]>3):
            ErrorTextBox("Error : Wrong cylinder value \n Must be
between 1 and 3")
            return
        if (Engine1.get() == "MX11" and (SetValues1()[1]<=0 or
SetValues1()[1]>step_check_dict["MX11"])):
            ErrorTextBox("Error : Wrong step value \n Look at the
explanation note to get more details")
            return
```

```
        if (Engine1.get() == "MX13" and (SetValues1()[1]<=0 or
SetValues1()[1]>step_check_dict["MX13"])):
            ErrorTextBox("Error : Wrong step value \n Look at the
explanation note to get more details")
            return
        if (Engine2.get() == "MX11" and (SetValues2()[1]<=0 or
SetValues2()[1]>step_check_dict["MX11"])):
            ErrorTextBox("Error : Wrong step value \n Look at the
explanation note to get more details")
            return
        if (Engine2.get() == "MX13" and (SetValues2()[1]<=0 or
SetValues2()[1]>step_check_dict["MX13"])):
            ErrorTextBox("Error : Wrong step value \n Look at the
explanation note to get more details")
            return
        height1,first_order1,second_order1,third_order1,
fourth_order1,fifth_order1,sixth_order1,seventh_order1,eighth_order1,Bo
rderlineLim_list,CriticalLim_list=plot_distortion_by_order(Engine1.get(
),SetValues1()[0],SetValues1()[1],Folder1.get(),1)
        height2,first_order2,second_order2,third_order2,
fourth_order2,fifth_order2,sixth_order2,seventh_order2,eighth_order2=pl
ot_distortion_by_order(Engine2.get(),SetValues2()[0],SetValues2()[1],Fo
lder2.get(),1)[0:9]
        plt.figure()
        plt.suptitle("Comparison of liner distortion between \n Engine
%s Cylinder %s Step %s \n Engine %s Cylinder %s Step
%s"%(Engine1.get(),SetValues1()[0],SetValues1()[1],Engine2.get(),SetVal
ues2()[0],SetValues2()[1]))
        plt.subplot(241)
        plt.subplots_adjust(hspace=0.2,wspace=0.3)
        plt.title("Order %s "%1)
        plt.plot(first_order1,height1)
        plt.plot(first_order2,height2)
        plt.grid()
        plt.xlabel("Diametral distortion [µm]")
        plt.ylabel("Axial position [mm]")
        plt.subplot(242)
        plt.title("Order %s "%2)
        line1,=plt.plot(second_order1,height1)
        line2,=plt.plot(second_order2,height2)
        line3,=plt.plot(BorderlineLim_list[0],height1,'r--')
        line4,=plt.plot(CriticalLim_list[0],height1,'r-')
        plt.figlegend(handles = [line1,line2,line3,line4]
,labels=('%s'%Folder1.get(),'%s'%Folder2.get(),'Border line
limit','Critical limit'),loc='upper right',fontsize=12)
        plt.grid()
        plt.xlabel("Diametral distortion [µm]")
        plt.ylabel("Axial position [mm]")
        plt.subplot(243)
        plt.title("Order %s "%3)
        plt.plot(third_order1,height1)
        plt.plot(third_order2,height2)
        plt.plot(BorderlineLim_list[1],height1,'r--')
        plt.plot(CriticalLim_list[1],height1,'r-')
        plt.grid()
        plt.xlabel("Diametral distortion [µm]")
        plt.ylabel("Axial position [mm]")
        plt.subplot(244)
```

```
            plt.title("Order %s "%4)
            plt.plot(fourth_order1,height1)
            plt.plot(fourth_order2,height2)
            plt.plot(BorderlineLim_list[2],height1,'r--')
            plt.plot(CriticalLim_list[2],height1,'r-')
            plt.grid()
            plt.xlabel("Diametral distortion [µm]")
            plt.ylabel("Axial position [mm]")
            plt.subplot(245)
            plt.title("Order %s "%5)
            plt.plot(fifth_order1,height1)
            plt.plot(fifth_order2,height2)
            plt.plot(BorderlineLim_list[3],height1,'r--')
            plt.plot(CriticalLim_list[3],height1,'r-')
            plt.grid()
            plt.xlabel("Diametral distortion [µm]")
            plt.ylabel("Axial position [mm]")
            plt.subplot(246)
            plt.title("Order %s "%6)
            plt.plot(sixth_order1,height1)
            plt.plot(sixth_order2,height2)
            plt.plot(BorderlineLim_list[4],height1,'r--')
            plt.plot(CriticalLim_list[4],height1,'r-')
            plt.grid()
            plt.xlabel("Diametral distortion [µm]")
            plt.ylabel("Axial position [mm]")
            plt.subplot(247)
            plt.title("Order %s "%7)
            plt.xlabel("Diametral distortion [µm]")
            plt.ylabel("Axial position [mm]")
            plt.plot(seventh_order1,height1)
            plt.plot(seventh_order2,height2)
            plt.plot(BorderlineLim_list[5],height1,'r--')
            plt.plot(CriticalLim_list[5],height1,'r-')
            plt.grid()
            plt.subplot(248)
            plt.title("Order %s "%8)
            plt.plot(eighth_order1,height1)
            plt.plot(eighth_order2,height2)
            plt.plot(BorderlineLim_list[6],height1,'r--')
            plt.plot(CriticalLim_list[6],height1,'r-')
            plt.grid()
            plt.xlabel("Diametral distortion [µm]")
            plt.ylabel("Axial position [mm]")
            plt.show()


    def apply_plot_distortion_graph():
        if Folder1.get() == Folder2.get():
            ErrorTextBox("Error : Identical model")
            return
        if Engine1.get() != check_dict[Folder1.get()]:
            ErrorTextBox("Error : Wrong engine \n Please set the engine
button")
            return
        if Engine2.get() != check_dict[Folder2.get()]:
            ErrorTextBox("Error : Wrong engine \n Please set the engine
button")
```

```
            return
        if (SetValues1()[0]<=0 or SetValues1()[0]>3):
            ErrorTextBox("Error : Wrong cylinder value \n Must be
between 1 and 3")
            return
        if (SetValues2()[0]<=0 or SetValues2()[0]>3):
            ErrorTextBox("Error : Wrong cylinder value \n Must be
between 1 and 3")
            return
        if (Engine1.get() == "MX11" and (SetValues1()[1]<=0 or
SetValues1()[1]>step_check_dict["MX11"])):
            ErrorTextBox("Error : Wrong step value \n Look at the
explanation note to get more details")
            return
        if (Engine1.get() == "MX13" and (SetValues1()[1]<=0 or
SetValues1()[1]>step_check_dict["MX13"])):
            ErrorTextBox("Error : Wrong step value \n Look at the
explanation note to get more details")
            return
        if (Engine1.get() == "MX11" and (SetValues1()[2]<=0 or
SetValues1()[2]>height_check_dict["MX11"])):
            ErrorTextBox("Error : Wrong height value \n Look at the
explanation note to get more details")
            return
        if (Engine1.get() == "MX13" and (SetValues1()[2]<=0 or
SetValues1()[2]>height_check_dict["MX13"])):
            ErrorTextBox("Error : Wrong height value \n Look at the
explanation note to get more details")
            return
        if (Engine2.get() == "MX11" and (SetValues2()[1]<=0 or
SetValues2()[1]>step_check_dict["MX11"])):
            ErrorTextBox("Error : Wrong step value \n Look at the
explanation note to get more details")
            return
        if (Engine2.get() == "MX13" and (SetValues2()[1]<=0 or
SetValues2()[1]>step_check_dict["MX13"])):
            ErrorTextBox("Error : Wrong step value \n Look at the
explanation note to get more details")
            return
        if (Engine2.get() == "MX11" and (SetValues2()[2]<=0 or
SetValues2()[2]>height_check_dict["MX11"])):
            ErrorTextBox("Error : Wrong height value \n Look at the
explanation note to get more details")
            return
        if (Engine2.get() == "MX13" and (SetValues2()[2]<=0 or
SetValues2()[2]>height_check_dict["MX13"])):
            ErrorTextBox("Error : Wrong height value \n Look at the
explanation note to get more details")
            return
      absc,Fourier_coef1,BorderlineLim,CriticalLim,H1 =
      plot_distortion_graph(Engine1.get(),
SetValues1()[0],SetValues1()[1],SetValues1()[2],Folder1.get(),1)
        absc2,Fourier_coef2,BorderlineLim2,CriticalLim2,H2 =
      plot_distortion_graph(Engine2.get(),SetValues2()[0],SetValues2()[1
      ],SetValues2()[2],Folder2.get(),1)

color_cell=(np.concatenate((color_set(CriticalLim,BorderlineLim,Fourier
_coef1)[0][:],color_set(CriticalLim,BorderlineLim,Fourier_coef1)[1][:],
```

```
color_set(CriticalLim,BorderlineLim,Fourier_coef1)[2][:],color_set(Crit
icalLim,BorderlineLim,Fourier_coef2)[2][:]))).reshape(4,7)
        Fourier_coef1 = np.around(Fourier_coef1,decimals=4)
        Fourier_coef2 = np.around(Fourier_coef2,decimals=4)
        plt.figure()
        plt.subplot(211)
        plt.title("Comparison of the diametral distortion depending on
harmonic order between \n Engine %s Cylinder %s Step %s z = %s mm \n
Engine %s Cylinder %s Step %s z = %s
mm"%(Engine1.get(),SetValues1()[0],SetValues1()[1],H1[SetValues1()[2]-
1][0],Engine2.get(),SetValues2()[0],SetValues2()[1],H2[SetValues2()[2]-
1][0]))
        plt.plot(absc,Fourier_coef1,'b')
        plt.plot(absc,Fourier_coef2,'orange')
        plt.plot(absc,BorderlineLim,'r--')
        plt.plot(absc,CriticalLim,'r-')
        plt.grid()
        plt.legend(('%s'%Folder1.get(),'%s'%Folder2.get(),'Border line
limit','Critical limit'),loc=1,fontsize=10)
        plt.xlabel("Harmonic order")
        plt.ylabel("Diametral distortion [µm]")
        plt.yticks(np.arange(0, 60, 5.0))
        plt.xticks([2,3,4,5,6,7,8],["2","3","4","5","6","7","8"])
        plt.subplot(212)
        collabel=["order %s"%s for s in range(2,9)]
        rowlabel=["Critical limit [µm]","Border line limit
[µm]","%s"%Folder1.get(),"%s"%Folder2.get()]

val_table=np.array([CriticalLim,BorderlineLim,Fourier_coef1,Fourier_coe
f2]).reshape(4,7)

tab=plt.table(cellText=val_table,colLabels=collabel,rowLabels=rowlabel,
cellColours=color_cell,loc='center left')
        tab.auto_set_font_size(False)
        tab.set_fontsize(10)
        plt.axis('off')
        plt.show()

    def apply_plot_cut_profile():
        if Folder1.get() == Folder2.get():
            ErrorTextBox("Error : Identical model")
            return
        if Engine1.get() != check_dict[Folder1.get()]:
            ErrorTextBox("Error : Wrong engine \n Please set the engine
button")
            return
        if Engine2.get() != check_dict[Folder2.get()]:
            ErrorTextBox("Error : Wrong engine \n Please set the engine
button")
            return
        if (SetValues1()[0]<=0 or SetValues1()[0]>3):
            ErrorTextBox("Error : Wrong cylinder value \n Must be
between 1 and 3")
            return
        if (SetValues2()[0]<=0 or SetValues2()[0]>3):
            ErrorTextBox("Error : Wrong cylinder value \n Must be
between 1 and 3")
            return
```

```
        if (Engine1.get() == "MX11" and (SetValues1()[1]<=0 or
SetValues1()[1]>step_check_dict["MX11"])):
            ErrorTextBox("Error : Wrong step value \n Look at the
explanation note to get more details")
            return
        if (Engine1.get() == "MX13" and (SetValues1()[1]<=0 or
SetValues1()[1]>step_check_dict["MX13"])):
            ErrorTextBox("Error : Wrong step value \n Look at the
explanation note to get more details")
            return
        if (Engine2.get() == "MX11" and (SetValues2()[1]<=0 or
SetValues2()[1]>step_check_dict["MX11"])):
            ErrorTextBox("Error : Wrong step value \n Look at the
explanation note to get more details")
            return
        if (Engine2.get() == "MX13" and (SetValues2()[1]<=0 or
SetValues2()[1]>step_check_dict["MX13"])):
            ErrorTextBox("Error : Wrong step value \n Look at the
explanation note to get more details")
            return
        H1,cut_profiles1,absc_max1=COMP_plot_cut_profile
(Engine1.get(),SetValues1()[0],SetValues1()[1],Folder1.get(),1)
        H2,cut_profiles2,absc_max2=COMP_plot_cut_profile
(Engine2.get(),SetValues2()[0],SetValues2()[1],Folder2.get(),1)
        absc_max = max(absc_max1,absc_max2)
        plt.figure()
        plt.suptitle("Comparison axial position with respect to radial
distortion between \n Engine %s Cylinder %s Step %s \n Engine %s
Cylinder %s Step %s \n Scale factor
%s"%(Engine1.get(),SetValues1()[0],SetValues1()[1],Engine2.get(),SetVal
ues2()[0],SetValues2()[1],100))
        plt.subplots_adjust(wspace=0.3)
        plt.subplot(241)
        plt.title("Angle %s "%180)
        line1,=plt.plot(cut_profiles1[4],H1[4])
        line2,=plt.plot(cut_profiles2[4],H2[4])
        plt.figlegend(handles = [line1,line2]
,labels=('%s'%Folder1.get(),'%s'%Folder2.get()),loc='upper
right',fontsize=12)
        plt.grid()
        plt.xlabel("Radial distortion [mm]")
        plt.xlim(2*absc_max+10,0)
        plt.ylabel("Axial position [mm]")
        plt.subplot(242)
        plt.title("Angle %s "%0)
        plt.plot(cut_profiles1[0],H1[0])
        plt.plot(cut_profiles2[0],H2[0])
        plt.grid()
        plt.xlabel("Radial distortion [mm]")
        plt.xlim(0,2*absc_max+10)
        plt.ylabel("Axial position [mm]")
        plt.subplot(243)
        plt.title("Angle %s "%225)
        plt.plot(cut_profiles1[5],H1[5])
        plt.plot(cut_profiles2[5],H2[5])
        plt.grid()
        plt.xlabel("Radial distortion [mm]")
        plt.xlim(2*absc_max+10,0)
```

```
        plt.ylabel("Axial position [mm]")
        plt.subplot(244)
        plt.grid()
        plt.title("Angle %s "%45)
        plt.plot(cut_profiles1[1],H1[1])
        plt.plot(cut_profiles2[1],H2[1])
        plt.xlabel("Radial distortion [mm]")
        plt.xlim(0,2*absc_max+10)
        plt.ylabel("Axial position [mm]")
        plt.subplot(245)
        plt.title("Angle %s "%270)
        plt.plot(cut_profiles1[6],H1[6])
        plt.plot(cut_profiles2[6],H2[6])
        plt.grid()
        plt.xlabel("Radial distortion [mm]")
        plt.xlim(2*absc_max+10,0)
        plt.ylabel("Axial position [mm]")
        plt.subplot(246)
        plt.title("Angle %s "%90)
        plt.plot(cut_profiles1[2],H1[2])
        plt.plot(cut_profiles2[2],H2[2])
        plt.grid()
        plt.xlabel("Radial distortion [mm]")
        plt.xlim(0,2*absc_max+10)
        plt.ylabel("Axial position [mm]")
        plt.subplot(247)
        plt.title("Angle %s "%315)
        plt.plot(cut_profiles1[7],H1[7])
        plt.plot(cut_profiles2[7],H2[7])
        plt.grid()
        plt.xlabel("Radial distortion [mm]")
        plt.xlim(2*absc_max+10,0)
        plt.ylabel("Axial position [mm]")
        plt.subplot(248)
        plt.title("Angle %s "%135)
        plt.plot(cut_profiles1[3],H1[3])
        plt.plot(cut_profiles2[3],H2[3])
        plt.grid()
        plt.xlabel("Radial distortion [mm]")
        plt.xlim(0,2*absc_max+10)
        plt.ylabel("Axial position [mm]")
        plt.show()

    def Delete_figures():
        plt.close("all")

    def ErrorTextBox(message):
        N = len(message)
        Ewindow=t.Tk()
        Ewindow.geometry('%sx100'%((N+20)*4))
        Message=t.Label(Ewindow,text=message)
        Message.pack()
        Message.place(relx=0.5, rely=0.4, anchor='center')
        OKButton=t.Button(Ewindow,text="OK",command = Ewindow.destroy)
        OKButton.pack()
        OKButton.place(relx=0.5, rely=0.8, anchor='center')

    def End_method():
```

```
        window.destroy()
        program()

    button=t.Button(window,text="Quit",command = End_method)
    button.pack()
    button.place(relx=0.9, rely=0.9, anchor='center')

    button1=t.Button(window,text="Plot radial distortion along a
path",command = apply_plot_path)
    button1.pack()
    button1.place(relx=0.33, rely=0.55, anchor='center')

    button2=t.Button(window,text="Plot deformed profile",command =
apply_plot_def_profile)
    button2.pack()
    button2.place(relx=0.67, rely=0.55, anchor='center')

    button3=t.Button(window,text="Plot deformed profile from Fourier
coefficients",command = apply_plot_def_profile_by_Fourier)
    button3.pack()
    button3.place(relx=0.5, rely=0.62, anchor='center')

    button4=t.Button(window,text="Plot axial position with respect to
diametrical distortion",command = apply_plot_distortion_by_order)
    button4.pack()
    button4.place(relx=0.5, rely=0.69, anchor='center')

    button5=t.Button(window,text="Plot diametrical distortion with
respect to harmonic order",command = apply_plot_distortion_graph)
    button5.pack()
    button5.place(relx=0.5, rely=0.76, anchor='center')

    button6=t.Button(window,text="Plot axial position with respect to
radial distortion",command = apply_plot_cut_profile)
    button6.pack()
    button6.place(relx=0.5, rely=0.83, anchor='center')

    button7=t.Button(window,text="Delete figures",command =
Delete_figures)
    button7.pack()
    button7.place(relx=0.5, rely=0.9, anchor='center')

    window.update()
 #   window.mainloop()
```

Brigde_file script

```
import os
import subprocess
import copy
```

```
def bridge():
    pyt_path = r"C:\Users\A-
Damien.Gode\AppData\Local\Enthought\Canopy\App\Canopy.exe"
    pyt_script = r"C:\Users\A-Damien.Gode\Documents\Distortion
program\Analysis\Program_interface.py"

    ENV = copy.deepcopy(os.environ)

    del ENV['PYTHONPATH']

    sp =
subprocess.Popen([pyt_path,pyt_script],stderr=subprocess.PIPE,stdout=su
bprocess.PIPE,shell=True,env=ENV)
    sp.communicate()
```

Explicative_Note script

```
import os
from abaqus import *
from abaqusConstants import *
import __main__
```

```
dirpath = "C:/Users/A-Damien.Gode/Documents/Distortion
program/Analysis"

os.chdir(dirpath)

def extract_coordinates(engine,cylinder):
    os.chdir(dirpath)
    if engine == 'MX11':
        doc=open('MX11_cyl%s_Coordinates.txt'%cylinder,'r')
        doc=doc.read()
        doc=doc.replace("\n","")
        doc=doc.replace("LIN-1","")
        doc=doc.split()
        Coor_table=[]
        index=0
        index1=0
        for i in range(len(doc)):
            if doc[i]=='Part':
                index=i
            if doc[i]=='----------------------------------------------
----------------------------------------------------------------------
---':
                index1=i
        Coor_table=doc[index1+1:index]
        return(Coor_table)
    if engine == 'MX13':
        doc=open('MX13_cyl%s_Coordinates.txt'%cylinder,'r')
        doc=doc.read()
        doc=doc.replace("\n","")
        doc=doc.replace("LINER-1","")
        doc=doc.split()
        Coor_table=[]
        index=0
        index1=0
        for i in range(len(doc)):
            if doc[i]=='Part':
                index=i
            if doc[i]=='----------------------------------------------
----------------------------------------------------------------------
---':
                index1=i
        Coor_table=doc[index1+1:index]
        return(Coor_table)

# The purpose of this function is to split these values and to stow
them in respective list
def split_coordinates(engine,cylinder):
    Coor_table=extract_coordinates(engine,cylinder)
    X_coor=[]
    Y_coor=[]
    Z_coor=[]
    count=1
    X_coor.append(float(Coor_table[1]))
    Y_coor.append(float(Coor_table[2]))
    Z_coor.append(float(Coor_table[3]))
    while 1+7*count<len(Coor_table):
        X_coor.append(float(Coor_table[1+7*count]))
```

```
        Y_coor.append(float(Coor_table[2+7*count]))
        Z_coor.append(float(Coor_table[3+7*count]))
        count+=1
    return(X_coor,Y_coor,Z_coor)

# This time as the radial deformations are not gathered under the same
file, it was required to implement two new functions to extract these
deformations
def extract_def_data(engine,cylinder,step,folder_name):
    os.chdir(dirpath+"/%s"%folder_name)
    doc=open('%s_cyl%s_step%s.txt'%(engine,cylinder,step),'r')
    doc=doc.read()
    doc=doc.split()
    radial_def_table=[]
    index=0
    for i in range(len(doc)):
        if doc[i]=='--------------------------------':
            index=i
    radial_def_table=doc[index+1:len(doc)]
    return(radial_def_table)

def extract_radial_def(engine,cylinder,step,folder_name):
    data_table=extract_def_data(engine,cylinder,step,folder_name)
    radial_def=[]
    count=1
    radial_def.append(float(data_table[1]))
    while 2*count<len(data_table):
        radial_def.append(float(data_table[2*count+1]))
        count+=1
    return(radial_def)


# This function is done to return the diameter of engine
def diameter(engine):
    if engine=='MX13':
        diameter=130
    if engine=='MX11':
        diameter=123
    return(diameter)

# This function is done to return the coordinates of the cylinder's
center
def center(X_coor,Y_coor):
    center=0
    max_X=max(X_coor)
    min_X=min(X_coor)
    max_Y=max(Y_coor)
    min_Y=min(Y_coor)
    center=((max_X+min_X)/2,(max_Y+min_Y)/2)
    return(center)

# simple function to test if a value is in a list
def belong(List,val):
    for i in range(len(List)):
        if List[i]==val:
            return(True)
    return(False)
```

```
# For the proper operation of the following functions and in order to
plot all the graphs, it is mandatory to regroup the nodes by their
height from the top
def split_height(engine,cylinder,step):
    Z_coor=split_coordinates(engine,cylinder)[2]
    height=[]
    already=[]
    ind_list=[]
    index=0
    while index<len(Z_coor):
        value=Z_coor[index]
        if belong(already,value)==True:
            index+=1
        else:
            ind_list=[]
            for i in range(len(Z_coor)):
                if Z_coor[i]==value:
                    ind_list.append(i)
            height.append((value,ind_list))
            already.append(value)
            index+=1
    return(height)

# As some nodes, supposed to be on the same height, have not exactly
the same Z coordinate (0.1 micrometers difference in average), it was
needed to gather them under a same value
# That way it was possible to get the same number of nodes for each
height
def gather_height(engine,cylinder,step):
    height=split_height(engine,cylinder,step)
    comp_val=0
    index=0
    already=[]
    H=[]
    index_list=[]
    while index<len(height):
        comp_val=height[index][0]
        if belong(already,round(comp_val,1))==True:
            index+=1
        else:
            index_list=[]
            for i in range (index,len(height)):
                if round(height[i][0],1)==round(comp_val,1):
                    index_list+=height[i][1]
            H.append((comp_val,index_list))
            already.append(round(comp_val,1))
            index+=1
    return(H)

def create_explanation_note(engine):
    if engine == 'MX11':
        expl_note=open('MX11_explanation_note.txt','w')
        height=gather_height('MX11',1,1)
        height.sort(reverse=True)
        odb = session.odbs[session.odbData.keys()[0]]
        tmp=odb.name.split()
        name=''
        index=0
```

```
        for i in range(len(tmp)):
            if tmp[i] == '/':
                index=0
        name=tmp[index+1:len(tmp)][0]
        expl_note.write("\n\n\n\n")
        expl_note.write("FOR THE ENGINE %s"%name)
        expl_note.write("\n\n\n\n")
        expl_note.write("Step indexing explanation")
        expl_note.write("\n\n")
        N=len(odb.steps)
        for i in range(N):
            expl_note.write("\n\nThe step indexed by %s is %s
"%(i+1,odb.steps.keys()[i]))
        expl_note.write("\n\n\n\n")
        expl_note.write("Height indexing explanation")
        expl_note.write("\n\n")
        for i in range(len(height)):
            expl_note.write("\n\n the height indexed by %s, for the
engine MX11, is %s mm from the liner top" %(i+1,height[i][0]))
    if engine == 'MX13':
        expl_note=open('MX13_explanation_note.txt','w')
        height=gather_height('MX13',1,1)
        height.sort(reverse=True)
        odb = session.odbs[session.odbData.keys()[0]]
        tmp=odb.name.split()
        name=''
        index=0
        for i in range(len(tmp)):
            if tmp[i] == '/':
                index=0
        name=tmp[index+1:len(tmp)][0]
        expl_note.write("\n\n\n\n")
        expl_note.write("FOR THE ENGINE %s"%name)
        expl_note.write("\n\n\n\n")
        expl_note.write("Step indexing explanation")
        expl_note.write("\n\n")
        N=len(odb.steps)
        for i in range(N):
            expl_note.write("\n\nThe step indexed by %s is %s
"%(i+1,odb.steps.keys()[i]))
        expl_note.write("\n\n\n\n")
        expl_note.write("Height indexing explanation")
        expl_note.write("\n\n")
        for i in range(len(height)):
            expl_note.write(" \n\n the height indexed by %s, for the
engine MX13, is %s mm from the liner top"  %(i+1,height[i][0]))
        expl_note.close()
        return(expl_note)
```