Chapter 1: INTRODUCTION

1.1 Case Hardening

Case hardening, is a heat treatment process that produces a surface that is hard and resistant to wear while maintaining the strength and toughness of the core. The significant feature about carburization is that it increases wear resistance and strength by diffusing carbon into the steel specimens surface creating a case while retaining hardness in the substantially lesser core. Machined low carbon steels usually adopt the process of carburization. The surface carburization followed by quenching and low-temperature tempering provides the high hardness value on the surface, excellent ductility, and toughness of the core.

Case-hardening by carburizing is achieved using three distinct processes which are as follows:

1) Enrichment of the steel surface component by carbon (carburizing)
2) The hardening process of the components (by quenching in a liquid)
3) Reduction of brittleness through tempering.

1.2 Carburization

The carbon potential at a given process temperature determines the rate of carburizing process and the surface layer parameters (the depth, carbon content, and hardness of martensite) and therefore acts as the main parameter pf the carburizing process. The temperature and the time of the carburizing process must be known to obtain the surface layer with the specified hardness and depth.

There are several existing processes by which carburizing can be done. Figure 1.1 gives a brief insight into the numerous existing carbon-containing agents used. The requirements for carburizing a steel component are a carbon donor and high temperatures. The earliest instance of carburizing donors in the history of carburization was through charcoal fires. They were mainly used for producing the heat but they also delivered a layer of carbon.

With an increasing understanding of the importance of carbon transfer, the first industrial process for carburizing was developed in the 19th century, which utilized carburizing powder. According to this process, the parts were placed in an enclosure brimming with carburizing powder. The enclosure comprised of anthracite coal, or coke, charcoal, and was put into the furnace. The carburization took place in temperatures between 870°C and 930°C. The carburizing effect was observed by the reaction of carbon with the atmospheric air, the products of which were carbon dioxide. The carbon dioxide turned into carbon monoxide which on further contact with the surface breaks up to produce carbon which then diffuses on the surface of the steel part. But using this carburizing system requires high manual power and limits the carburizing depth accounting for powder transformation and expansion during the process. Due to the above limitations, powder carburizing, also known as pack carburizing became obsolete.

At the beginning of the 20th century, the next milestone in the carburizing process was achieved through the development of salt bath carburizing technology. In salt bath carburization, either the full part or some regions of the part are partially dipped into a liquid salt melt. The salt melt comprises of cyanides (e.g., sodium cyanide, NaCN) along with an activator (e.g., barium chloride, BaCl). The activator assists in the dissociation of the cyanide and produce carbon which becomes the bi-product. The free carbon diffuses into the part's surface. This carburizing system is hazardous to the environment which is a major disadvantage. Nevertheless, for special surface contour applications, salt bath technology is still a plausible choice.

During the second half of the 20th century, there was an emergence of controlled gas carburizing technology and continues to be one of the most successful carburizing systems even today, especially for mass production in the automotive industry. The possibilities of changing carbon potential during a heat treatment cycle with controlled gas carburization resulting in the creation of intricate carbon profiles in the parts were formulated. The gas atmospheres used for the carburization process have carbon monoxide, hydrogen, and nitrogen which are produced in different ways. The significant difference between controlled gas carburizing and 'normal' gaseous carburizing is that it uses hydrocarbon gases which act as carbon donors instead of carbon monoxide. As a consequence, no transfer of oxygen occurs during carburizing, thus hindering internal oxidation, in the regions lying close to the surface of the carburized case. As the low-pressure carburizing technology is carried out by high-pressure gas quenching in place of oil-quenching, the process becomes very clean and environmentally friendly.

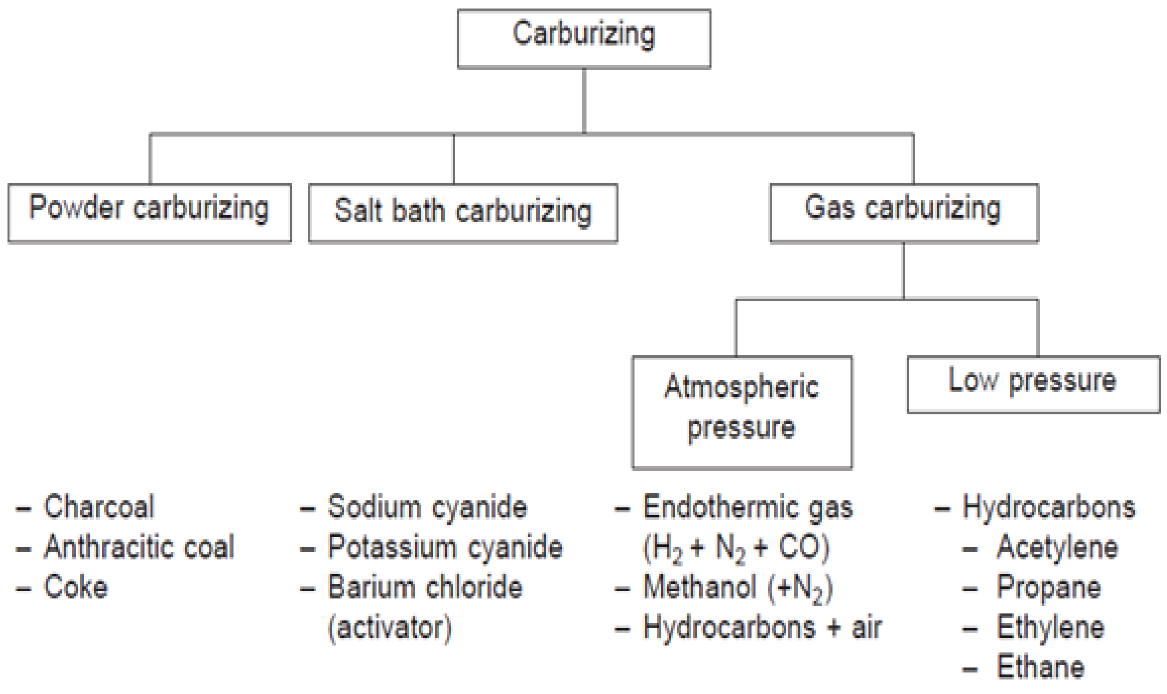


Figure 1.1: Different types of Carburizing techniques and the various carbon donors used

1.3 Gaseous Carburization

Gaseous carburizing can be described as a thermochemical process that changes the chemical composition of the material in the vicinity of the part surface by enriching it with carbon. This process is undertaken in a carbon-rich atmosphere inside a carburizing furnace to achieve the required case depth. It is often adopted as a surface hardening procedure in batch furnaces. Depending upon the hardness and the required case depth for a particular application, the quality of the carburized parts is determined. Hence, to ensure reliability and prolonged service life of the carburized parts, it is vital to understand the mechanism of carbon mass transfer to accurately predict carbon concentration profiles and the case depth during the heat treatment processes. Despite the widespread application of the carburized product, industries still rely on process control methods such as empirical analysis and trial and error methods both of which are time-consuming and expensive and are insufficient to solve challenges faced in the current scenario.

The mass transfer coefficient is a complex function of the atmospheric gas composition, carburizing potential, surface carbon content and temperature. Another factor that determines the rate of carbon transport is the coefficient of carbon diffusion in austenite. It is strongly influenced by the carburizing temperature and carbon concentration in steel. The carburizing potential in the furnace depends upon the atmospheric gas composition. Precise carbon potential calculations require complete measurements of the gaseous constituents such as H_2O, CH_4, CO, CO_2 in the furnace, and representation of sampling locations where the constituents are analyzed. Since the surface carbon concentration and the flux of carbon atoms from atmosphere to the steel surface is a function of time, maintaining a uniform atmospheric carbon potential during single-stage carburization requires constant attention and adjustments in that particular region until the criteria have been met. An increase in the carburizing temperature results in the increase of the rate of mass transfer in steel and the furnace atmosphere. It also promotes excessive austenite grain growth and deteriorates the furnace condition. It is to be noted that the effect of time on the case depth is interdependent with the carburizing temperature.

1.3.1 Carbon Transfer Mechanism During the process of Gas Carburization

During the process of gas carburization, the mass transfer mechanism is a complex phenomenon that is carried out in three different stages:

1) carbon transport from the ambient to the steel surface

2) chemical reactions on the surface

3) diffusion of absorbed carbon atoms towards the bulk of the steel down the chemical potential gradient.

The total carbon mass transfer from ambient conditions to the steel is thus found by the limiting process, which kinetically becomes the rate-controlling stage of carburization. Figure 1.2 represents the mechanisms of the carbon transfer during carburizing and also the primary control parameters, namely the coefficient of carbon diffusion in steel (D) at austenitizing temperatures and the mass transfer coefficient (β) defining carbon atoms flux (J) from the ambient to the steel surface.

Considering the kinetics of the carburizing process, the maximum carburization rate is obtained when carbon transfer from the ambient conditions is equal to or greater than the carbon diffusion rate in the solid-state. However, in practice, the non-equilibrium carbon transfer from the ambient to the solid boundary including surface reaction is often reported to be the rate-limiting factor especially at the start of the carburizing process.

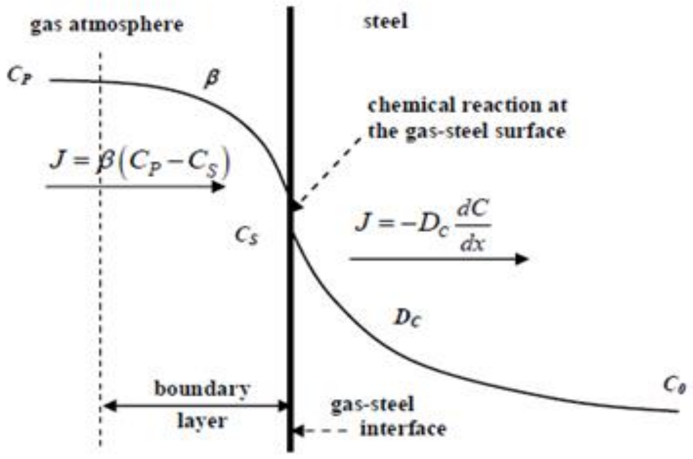


Figure 1.2: Representation of carbon mass transfer during gas carburization

1.3.2 Generation of carburizing atmospheres

The required carburizing atmosphere for gas carburization can be created in different ways. The three commonly used industrial systems for the production of the carburizing atmosphere are endothermic gas, a mixture of nitrogen gas and liquid methanol (CH_3OH), a mixture of hydrocarbon gas (propane, natural gas or others) and air. In the current study, the focus is directed towards the endothermic gas carburization.

1.3.2.1 Endothermic gas carburization

Endothermic gas is a mixture of approximately 20 vol-% CO, 40 vol-% N_2, and 40 vol-% H_2. It is produced in a distinct endothermic gas generator (Hotchkiss, 1942, Hollmann, 1961; Wünning, 1964). Inside the generator, the hydrocarbon gas reacts with the air at a temperature of about 1000°C. In case of natural gas, which comprises mainly of methane, the net reaction in the generator can be approximated to:

$CH_4 + 2.5\,(0.2O_2 + 0.8N_2) \longrightarrow CO + 2H_2 + 2N_2$ ……………………………….……….…….(1.1)

where ($0.2O_2 + 0.8N_2$) is the air content with an oxygen content just over 20 vol.% and nitrogen content just below 80 vol.%. The gas mixture produced is further cooled down to a temperature ≤60°C quickly at the gas exit of the generator to avoid the decomposition of the CO into CO_2 and carbon (soot) as per the Boudouard reaction:

$2CO \longrightarrow [C] + CO_2$ …………………………………………………………………….…..(1.2)

Figure 1.3 shows the principle of an endothermic gas generator. After proper cooling, the endothermic gas can be used for any kind of heat treatment such as tempering, annealing, hardening, and for carburizing processes which is one of the major advantages of endothermic gases. The CO content in the furnace is independent of the gassing rate and will nearly remain constant during the entire heat treatment cycle. It is also possible to create endothermic gas with mild changes in the composition by using other hydrocarbon-rich gases. For example, the endothermic gas generator running on air and propane delivers an endothermic gas according to the following reaction:

$$C_3H_8 + 7.5(0.2O_2 + 0.8N_2) \longrightarrow 3CO + 4H_2 + 6N_2 \ldots\ldots (1.3)$$

with a composition of approximately 44 vol.% N_2, 32.2 vol.% H_2 and 23.8 vol.% CO. Carburizing atmospheres can be created in a separate gas generator as stated above, but the production of the carburizing atmosphere directly inside the carburizing furnace is also possible, provided, that the furnace temperature is sufficiently high (above 850°C). There are two ways of forming such in-situ atmospheres: one such method is by feeding the hydrocarbon gas mixed with an oxidizing gas such as air directly in the furnace, and the other is by feeding liquid methanol in the absence or presence of nitrogen gas into the furnace.

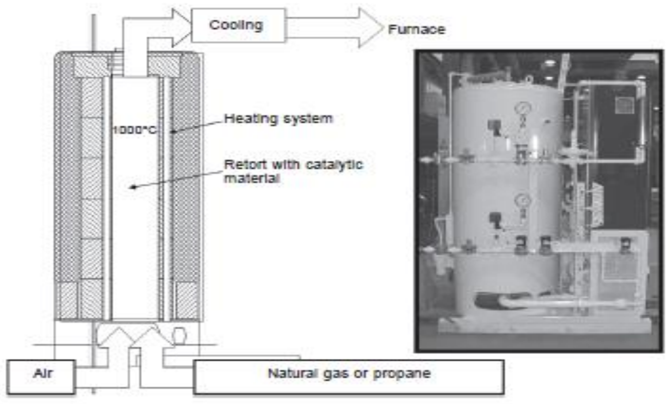


Figure 1.3: Principle of an endothermic gas generator.

1.4 Sansera Engineering

The process parameters for case carburization that was performed on gears were provided by Sansera Engineering for our validation. Sansera is an engineering-led integrated manufacturer of complex and high-quality precision components for the aerospace and automotive sectors, mainly supplying original equipment manufacturers ("OEMs") in India and abroad. They manufacture and distribute a wide range of machined and precision forged components that are critical for transmissions, engines, and other systems for a two-wheeler, passenger vehicle, and the light and heavy commercial vehicle segments in the automotive sector. Sansera Engineering also supplies components to the aerospace sector and off-road vehicles.

1.5 Machine Learning

Most generally, a machine learning algorithm can be thought of as a black box. It takes inputs and provides outputs. The "black box" is in fact a representation of a mathematical model. The machine learning algorithm follows a kind of trial-and-error method to determine the model that estimates the outputs, given inputs.

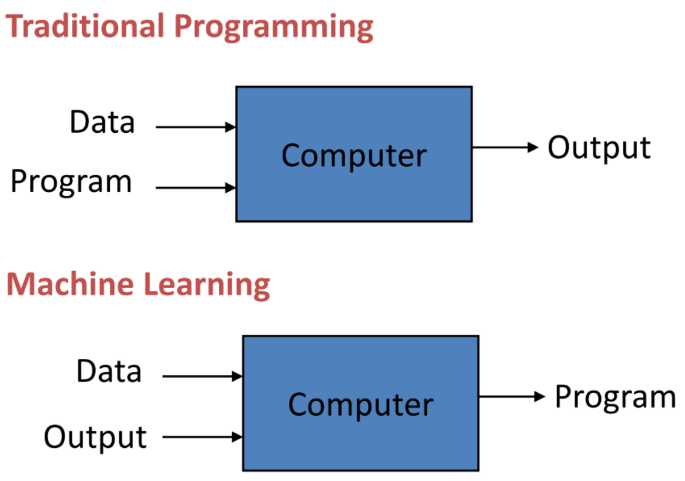


Figure 1.4: Difference between traditional programming and machine learning

Once we have a model, we must train it. Training is the method by which the model learns how to make sense of the input data. Types of Machine learning are as follows:

1.Supervised

It is called supervised as we provide the algorithm not only with the inputs but also with the targets (desired outputs). Based on that information the algorithm learns how to produce outputs as close to the targets as possible. Common methods:

- Regression: Regression outputs are continuous numbers.
- Classification: Classification outputs are labels from some sort of class.

2.Unsupervised

In unsupervised machine learning, the researcher feeds the model with inputs, but not with targets. Instead the user asks it to find some sort of dependence or underlying logic in the data provided. Common methods:

- Clustering

3.Reinforced

In reinforcement ML, the goal of the algorithm is to maximize its reward. It is inspired by human behavior and the way people change their actions according to incentives, such as getting a reward or avoiding punishment. Common methods:

- Decision process
- Reward system

The basic idea behind training an algorithm includes four ingredients: objective function, data, model and optimization algorithm. They are ingredients, instead of steps, as the process is iterative. The simplest possible model is a linear model. Despite appearing unrealistically simple, in the deep learning context, it is the basis of more complicated models.

Chapter 2: LITERATURE REVIEW

During the initial phase of our project progress, we began to understand the existing research work performed on case carburization. Various numerical models and thermo-chemical analysis has been done with regards to this topic since the carburization method became a widely used heat transfer process. Various experimental analysis and subsequent modeling using large data sets were observed. Some of the research work that provided us incentives and direction for the development of our case carburization model are mentioned below.

Maisuradze and Kuklina.[1] in their paper on the numerical solution of differential diffusion equation for a steel carburizing process, modeled a 1D axisymmetric diffusion model by applying third Boundary condition and finite element method. Parameters like temperature and time at which carbon is diffused into the component were varied and the variation of carbon content percentage with distance from the surface was obtained.

Cavaliere *et.al.*[2]; modeled thermo-chemical diffusion process of nitriding and carburizing. The diffusion model, which was based on Fick's laws, was applied to steels to explain the growth kinetics of layers. The analytical model was then employed to perform finite element calculations. Therefore, it was possible to calculate the carbon and nitrogen concentration in cross-sections of the cylindrical samples.

Lee *et.al.*[3]; in their paper developed a FEM subroutine code for Carburization and thermal distortion and used it for simulating a carburizing process of heat-treatable steel components and compared with commercial code for a cylindrical sample. The advantage of the CTDT in this research was the flexibility of editing and updating the models and updating the database for mechanical and thermal properties while giving desirable outcomes.

Eck *et.al.*[4]; focused on the porosity on solid-solid phase transformation and the influence of local density of a surface densified gear on the carburizing process during the heat treatment process and of the local density of a surface densified gear on the carburizing process. The model found a link between surface densification of PM parts and the calculation of the carburization process.

Zajusz *et.al.*[5]; focused on the fundamental understanding of the carbon transfer into steel. The influence of processing parameters on the carbon content and distribution in the steel is studied. Introducing the same carbon content into steel demands a much longer boost time when the process is carried out at lower pressures. When longer boosts come into play, the gradient of carbon concentration near the surface is higher than in the case of short boosts. Less cycles and longer diffusion times are essential to obtain lower carbon concentration on the surface.

From the work of Deshpande et al.[6], we see that three surrogate models are made starting with the fundamental model of linear regression and then transitioning to complex models like genetic programming and artificial neural networks. In their case, Artificial Neural Networks offer the best predictions both in terms of accuracy and processing time. Depending on the data set that we get from the industry, we will be using any of the above models or a more suitable model as there are a plethora of machine learning models freely available. The significant reduction in time using these surrogate models is a strong compelling reason for using these lean surrogate models for process optimization etc.

Chapter 3: MATHEMATICAL MODELLING

Initially to get familiar with the development of the model, the paper written by Maisuradze, et.al.[1] was taken as a reference. The analysis adopted in the paper was reproduced and a 1D case carburization model using the Finite Element approach was developed.

This chapter provides the mathematical analysis which is the primary foundation in the development of the numerical finite element model using Fick's governing diffusion equations for one, two- and three-dimensional geometries.

3.1 Mathematical modelling for 1 Dimensional geometry

The 1-Dimensional finite element model for case carburization was developed using the governing equation based on Fick's second law of diffusion is given by:

$$D_c \frac{\partial^2 C}{\partial x^2} = \frac{\partial C}{\partial t} \quad \text{……………………………………………..……………(3.1)}$$

The boundary condition is given by the mass transfer as follows:

$$D_c \frac{\partial C}{\partial x} = \beta(c_p - c_s) \quad \text{……………………………………………….(3.2)}$$

Where, D - Carbon diffusion Coefficient in the austenite, in m^2/s

D is calculated using the Arrhenius type equation,

$$D = Do*e\text{^}(-Q/RT)\ m^2/s \text{………………………………………………(3.3)}$$

Q - Activation energy in J/mol

R - Universal gas constant, J/mol K

T - Carburization temperature or furnace temperature in Kelvin

c - carbon content, mass %, cp- furnace carbon percentage = φ, s - surface

β - coefficient of carbon mass transfer from the furnace atmosphere to steel part surface, m/s

x - distance from the surface, m

The carbon mass transport mechanism is explained earlier for gas carburization.

3.1.1 Finite Element Model Development

For evaluating the carbon profile using a 2nd order time-dependent differential, (3.2) and boundary condition of the third kind, (3.3), 'Time Discretization-Finite Difference Method (FDM)' is used for transient system analysis. This method involves discretizing the differential operator involving the time-dependent and space terms. It provides a numerical approximation of the transient term using the Finite Difference Method (FDM) and Taylor series. The equation is given by:

$$([C] + \theta\Delta t[K])\{c\}^{n+1} = ([C] - (1-\theta)\Delta t[K])\{c\}^{n} + \Delta t(\theta\{f\}^{n+1} + (1-\theta)\{f\}^{n}) \quad \ldots\ldots(3.3)$$

Equation (3.3) gives the nodal values of carbon mass percentage at (n+1) time levels. These values are calculated using n time level values. By varying the parameter Θ, different transient schemes can be constructed. In our study a fully implicit scheme is used which evaluates using forward difference method, hence $\Theta = 1$.

Here,

[C] is the capacitance matrix,

[K] is the stiffness matrix,

c is the mass percentage of carbon, %

Δt is the time intervals in seconds,

{f} is the force vector.

The element matrices, [C], [K], {f} for the transient equations are evaluated using Galerkin's weighted residual formulation. The carbon percentage is approximated over space as follows:

$$c = \sum_{i=1}^{n} N_i c_i \quad \ldots\ldots(3.4)$$

where, Ni are the shape functions, n is the number of nodes in a domain, and ci is the time-dependent nodal carbon percentages.

In matrix form,

$$[C] = \int_{\Omega} [N]^T [N] d\Omega \quad \ldots\ldots(3.5)$$

$$[K] = \int_{\Omega} [B]^T [D][B] d\Omega + \int_{\Gamma} \beta [N]^T [N] d\Gamma \quad \ldots\ldots(3.6)$$

$$\{f\} = \int_{\Gamma} \beta\varphi [N]^T d\Gamma \quad \ldots\ldots(3.7)$$

Where, [B] is the gradient matrix, [D] represents the carbon diffusion matrix over space. Ω represents integral over volume and Γ represents integral over boundary surface.

3.1.2 1-Dimensional linear element matrices

For the one-dimension geometry, a linear element with two nodes was used for evaluation of the matrix integrals,

Stiffness matrix (1st part),

$$[K_D]^e = \begin{bmatrix} D/l & -D/l \\ -D/l & D/l \end{bmatrix} \quad \ldots\ldots(3.8)$$

Where, l is the length of each linear element.

Capacitance matrix,

$$[C]^e = \begin{bmatrix} 2l/6 & l/6 \\ l/6 & 2l/6 \end{bmatrix} \quad \ldots\ldots(3.9)$$

The boundary condition is applied on node 1 and the corresponding force vector and the stiffness matrix is given below:

Force vector,

$$\{F_s\} = \begin{bmatrix} \beta\varphi \\ 0 \end{bmatrix} \qquad (3.10)$$

Stiffness matrix, 2nd half,

$$[K_M] = \begin{bmatrix} \beta & 0 \\ 0 & 0 \end{bmatrix} \qquad (3.11)$$

3.2 Mathematical modelling for 2-Dimensional geometry

The 2-Dimensional finite element model for case carburization was developed using the governing equation based on Fick's second law of diffusion is given by:

$$\mathrm{D_x}\frac{\partial^2 \mathrm{C}}{\partial \mathrm{x}} + \mathrm{D_y}\frac{\partial^2 \mathrm{C}}{\partial \mathrm{y}} = \frac{\partial \mathrm{C}}{\partial \mathrm{t}} \qquad (3.12)$$

The boundary condition is given by the mass transfer as follows:

$$D_x\frac{\partial C}{\partial x} = \beta(c_p - c_s) \qquad (3.13)$$

Where, Dx = Dy = D is Carbon diffusion Coefficient in the austenite, in m^2/s

D is calculated using the Arrhenius type equation,

D = Do*e^(-Q/RT) m^2/s..(3.14)

Linear triangular elements with 3 nodes are used for finite element modeling of 2-Dimensional geometry. The stiffness, capacitance matrices, force vector are calculated using similar Galerkin's formulation as seen in the case of 1D geometry and are given by:

The element matrices obtained are:

Stiffness matrix(1),

$$[K_D]^e = \frac{D_x}{4A}\begin{bmatrix} b_i^{\,2} & b_ib_j & b_ib_k \\ b_jb_i & b_j^{\,2} & b_jb_k \\ b_kb_i & b_kb_j & b_k^{\,2} \end{bmatrix} + \frac{D_y}{4A}\begin{bmatrix} c_i^{\,2} & c_ic_j & c_ic_k \\ c_jc_i & c_j^{\,2} & c_jc_k \\ c_kc_i & c_kc_j & c_k^{\,2} \end{bmatrix} \qquad (3.15)$$

Where A is the area of each element

Capacitance matrix,

$$[C]^e = \frac{A}{12}\begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix} \qquad (3.16)$$

The boundary condition is applied on edge nodes and the corresponding force vector and stiffness matrix are given below:

Force vector,

$$\{F_s\}^e = \frac{SL_{ij}}{2}\begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \qquad (3.17)$$

Boundary Stiffness matrix,

$$[K_M] = \frac{ML_{ij}}{6}\begin{bmatrix} 2 & 1 & 0 \\ 1 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(3.18)$$

Where, bi, bj, bk and ci, cj, ck are constants evaluated by the triangular element geometry, Dx = Dy = D and M = β and S = βφ.

3.3 Mathematical modelling for 3-Dimensional geometry

The 2-Dimensional finite element model for case carburization was developed using the governing equation based on Fick's second law of diffusion is given by:

$$\mathrm{D_x}\frac{\partial^2 \mathrm{C}}{\partial \mathrm{x}} + \mathrm{D_y}\frac{\partial^2 \mathrm{C}}{\partial \mathrm{y}} + \mathrm{D_z}\frac{\partial^2 \mathrm{C}}{\partial \mathrm{z}} = \frac{\partial \mathrm{C}}{\partial \mathrm{t}} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(3.19)$$

The boundary condition is given by the mass transfer as follows:

$$D_x\frac{\partial C}{\partial x} = \beta(c_p - c_s) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(3.20)$$

Where, Dx = Dy = Dz = D is Carbon diffusion Coefficient in the austenite, in m^2/s

D is calculated using the Arrhenius type equation,

D = Do*e^(-Q/RT) m^2/s……………………………………………(3.21)

Tetrahedral elements with 4 nodes are used for finite element modeling of 3-Dimensional geometry. The stiffness, capacitance matrices, force vector are calculated using similar Galerkin's formulation as seen in the case of 1D and 2D geometry and are given by:

Stiffness matrix (1st part),

$$[K_D]^e = \int_{\Omega}[B]^T[D][B]d\Omega \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(3.22)$$

Capacitance matrix,

$$[C]^e = \frac{V^e}{20}\begin{bmatrix} 2 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 \\ 1 & 1 & 2 & 1 \\ 1 & 1 & 1 & 2 \end{bmatrix} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(3.23)$$

Boundary Stiffness matrix,

$$[K_M]^e = \frac{MA_{ijk}}{12}\begin{bmatrix} 2 & 1 & 1 & 0 \\ 1 & 2 & 1 & 0 \\ 1 & 1 & 2 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(3.24)$$

Force Vector,

$$\{F_s\}^e = \frac{SA_{ijk}}{3}\begin{bmatrix}1\\1\\1\\0\end{bmatrix} \qquad (3.25)$$

Where, M = β and S = βφ.

Chapter 4. METHODOLOGY

This chapter discusses the methodology followed for the developed case carburization model using diffusion models for 1D, 2D, and 3D geometry and the development of hardness prediction models for various general steels using empirical relations and linear regression.

4.1 Finite element modelling

The methodology adopted for developing the mathematical model explained in the previous chapter is broadly shown in figure 4.1.

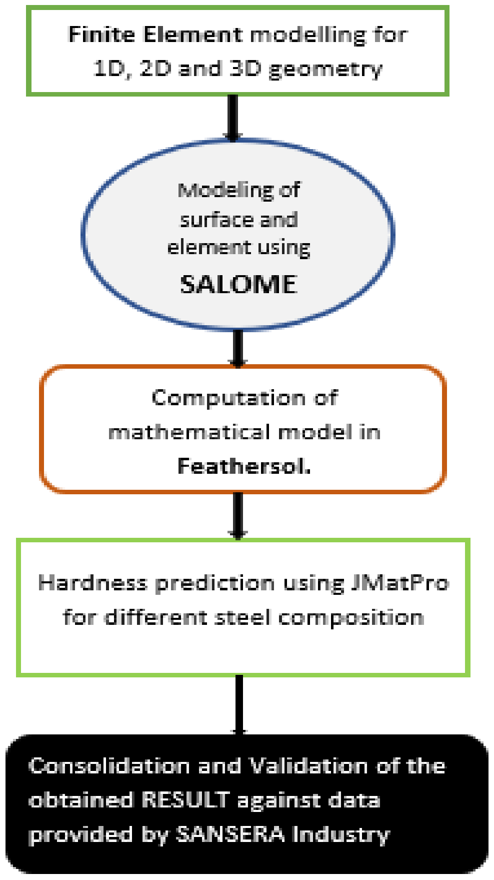

Finite Element modelling for 1D, 2D and 3D geometry
Modeling of surface and element using SALOME
Computation of mathematical model in Feathersol.
Hardness prediction using JMatPro for different steel composition
Consolidation and Validation of the obtained RESULT against data provided by SANSERA Industry

Figure 4.1: Methodology flow chart

4.1.1 One-Dimensional finite element modeling

For developing the mathematical diffusion model of the carburization of the steel parts for one dimensional case as mentioned in section 3.1, 1d linear rod element with 2 nodes was modelled and matrices were evaluated for the same. The One-dimensional formulation assumes that the diffusion of the carbon into the steel occurs in one direction which is normal concerning the saturated surface. When the finite element method is applied, the surface layer of the steel part is divided into several N nodes and (N-1) elements with width $\Delta x = l/(N-1)$, where l is the length of the rod. Figure 4.2 depicts the calculation mesh utilized for the finite element method.

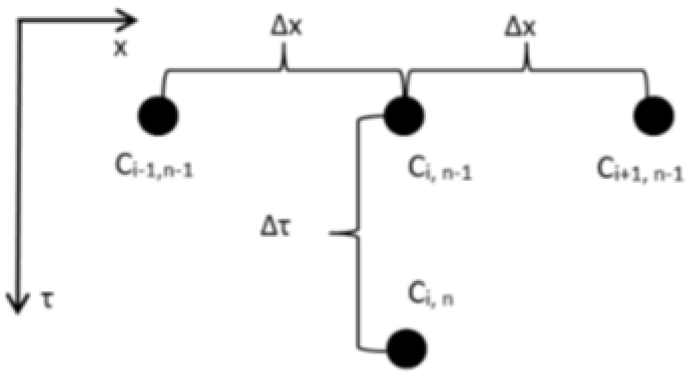


Figure 4.2: 1D linear calculation mesh

For the linear elements, the matrices were evaluated using Python ide. For the deployment of the code, various inbuilt python libraries were called into the Jupyter console. Some of them are:

1. Numpy

It is used mostly for scientific calculations of equations involving arrays and matrices. It is the foundation on which most of the libraries are built-in Python.

2. Pandas

Pandas provide data frames that are easily interpretable when there is a huge dataset for analysis.

3. Matplotlib

Matplotlib is used for analyzing the data pictorially and is the most commonly used library for numerical plotting.

4. Seaborn

It is built on top of matplotlib and is a very good library for statistical plotting and works well with different types of data.

5.Scipy

It is used for performing efficient numerical routines for better optimization and numerical analysis

6. Scikit-learn or sklearn

This library contains a bunch of advanced algorithms to be used in machine learning for better prediction.

7. Statsmodel

This Python library has functions and classes to perform statistical tests and statistical analysis.

With the help of these libraries the local matrices were evaluated. Another feature of Python called the slicing operator was used to calculate global matrices. In the initial stages of our model build-up and verification, reference [1] was used as a baseline. The input parameters for the calculation of the case depth such as the process temperature, T, K; the carbon potential of the furnace atmosphere, φ, mass. %; the carbon mass transfer coefficient, β, m/s were set following [1]

The values were set as follows:

Diffusivity, $Do = 0.11 * 10^{-4} m^2/s$,

Carburizing temperature, T = 940°C+273K,

Carburization time, Δt = 12h

Universal gas constant, R = 8.31J/mol-K

Activation energy, Q = 132000J/mol

$\beta = 1 * 10^{-7} m/s$

φ = 0.75%

Running the python code, with the above-mentioned parameters gave the carbon percentage profile as the result. The calculated carbon distribution in the carburized layer was further used for the estimation of the hardness profile using JMatPro software for steel grade 19NiCrMo4. This was done by obtaining an empirical relation between Vickers hardness number (Hv) and the carbon mass percentage, figure 4.4, providing varying carbon mass percentages along with other element compositions of the steel grade as input. The result was obtained using a quenching process at a starting temperature of 860°C and a cooling rate of 10K/s.

[1], provided the theoretical and experimental values of hardness. The experimental process was carried out in the industrial electrical pit furnace equipped with the carbon potential control system. The carburization temperature was 940 °C, the target depth of the surface layer was 1.2 – 1.6 mm. The carbon potential of the furnace was set as 0.70…0.75 mass. % C. The steel parts were subjected to the following heat treatment after the carburizing: tempering (650 °C, 8 h), oil quenching (820 °C, 2 h), and tempering (200 ° C, 5 h). The hardness profiles in the carburized layer were obtained for several batches of the steel parts. The Vickers hardness was measured through HVS-1000A hardness tester. The theoretical process made use of the dependence of the martensite hardness on the carbon content for the specified steel grade and hardness profile was evaluated for case hardened carburized 19NiCrMo4 steel parts. Case depth (hardness depth) was defined for the surface area segment where the microstructure consisted of at least 50% martensite.

The carbon and hardness profile obtained using our FE model and subsequently JMatPro were plotted against the plots from reference [1] and it was observed that the results obtained were fairly accurate as seen in Figures 4.3 and 4.5. Thus, the estimation of the carbon distribution in the carburized layer for any process time was possible. The time required to obtain the specified surface layer depth also could be determined.

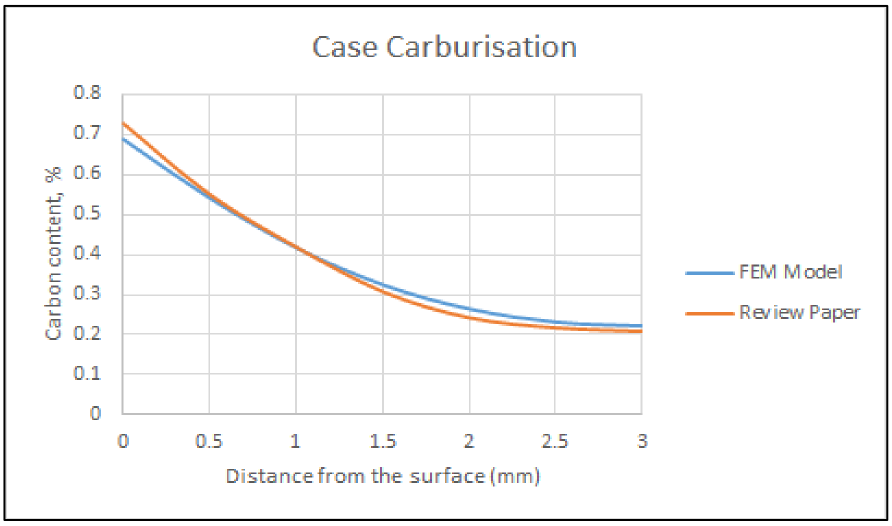


Figure 4.3: Comparison of carbon profiles obtained.

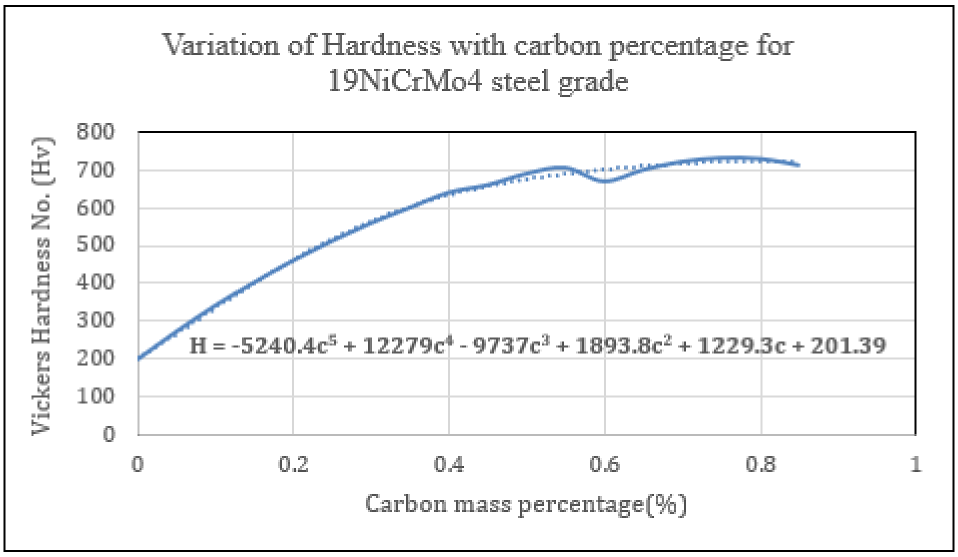


Figure 4.4: Hardness profile along with the carbon percentage for 19NiCrMo4

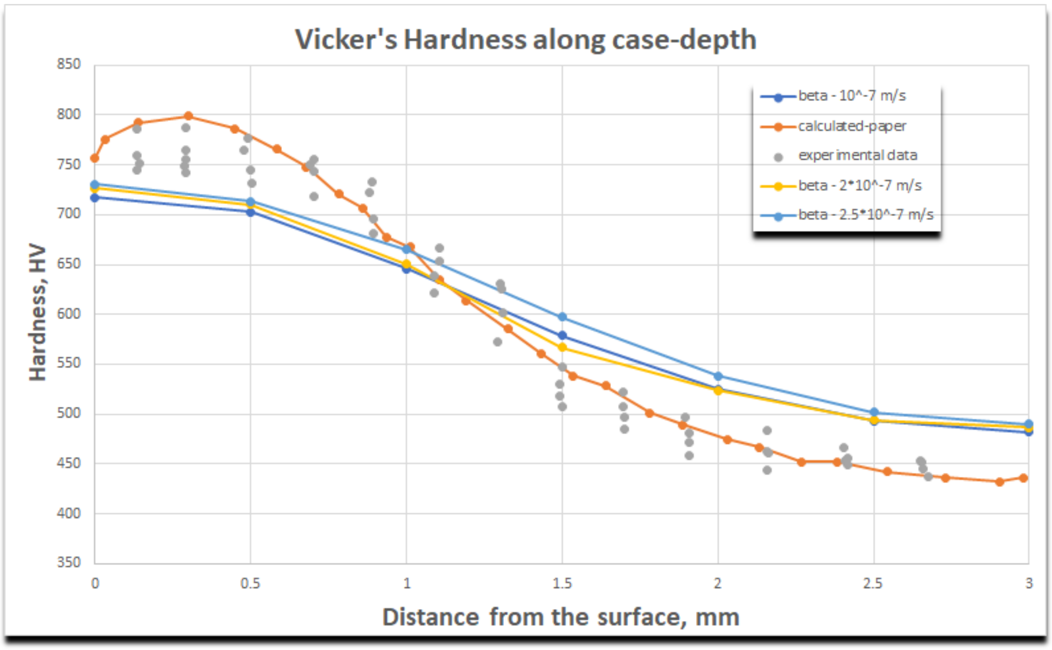


Figure 4.5: Hardness Profile comparison

Upon primary verification of the one-dimensional model, the second stage involved verification concerning the data provided in the HT instruction sheet by Sansera Engineering, Appendix B. The experimental data for endo-gas case carburization of two different batches were provided. Each batch was accompanied by three trials. Instead of single-stage carburization as seen in the previous case, the carburization was divided into 3 sub heat transfer layers, i.e, the boost stage, the diffusion stage, and the equalizing stage. Here the concept of boosting was elaborated on. In this method, high carbon percentages are supplied for shorter durations. This helps in decreasing the time taken for obtaining the required surface carbon percentage and case depth which helps in increasing the efficiency of the heat transfer process, especially during mass productions. To accommodate for multiple stages of carburization, interdependent loops were created in python.

4.1.2 Two-Dimensional finite element modeling

After the verification of the finite element model for one-dimensional geometry, the following step involved incorporating geometries whose contours required two spaces or coordinates for its definition, hence the development of a two-dimensional finite element diffusion model. The mathematical model for two-dimensional geometry using linear triangular element (Figure 4.6) mesh hypothesis as described in section 3.2 is utilized to develop the FE model. The procedure followed for developing the 2D model is illustrated in Figure 4.7.

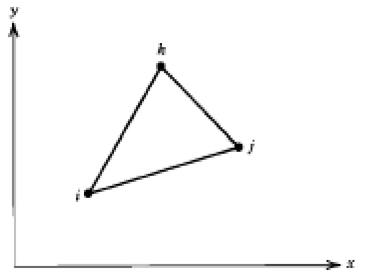


Figure 4.6: Linear triangular element

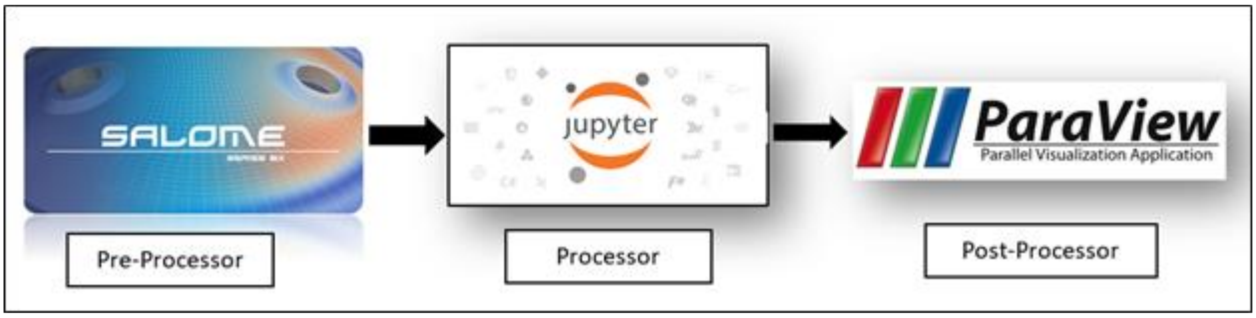


Figure 4.7: Stages of two-dimensional model development

The first stage is the geometrical modeling or designing stage. A rectangular geometry is meshed using linear triangular elements using the opensource designing software SALOME. This stage is the pre-processing stage. The nodal coordinates, element dictionaries, boundary conditions, and applications are extracted from the SALOME platform using .unv format. The mesh generated is as shown in Figure 4.8. The aspect ratio lies for most elements in the range of 1.2- 1.4.

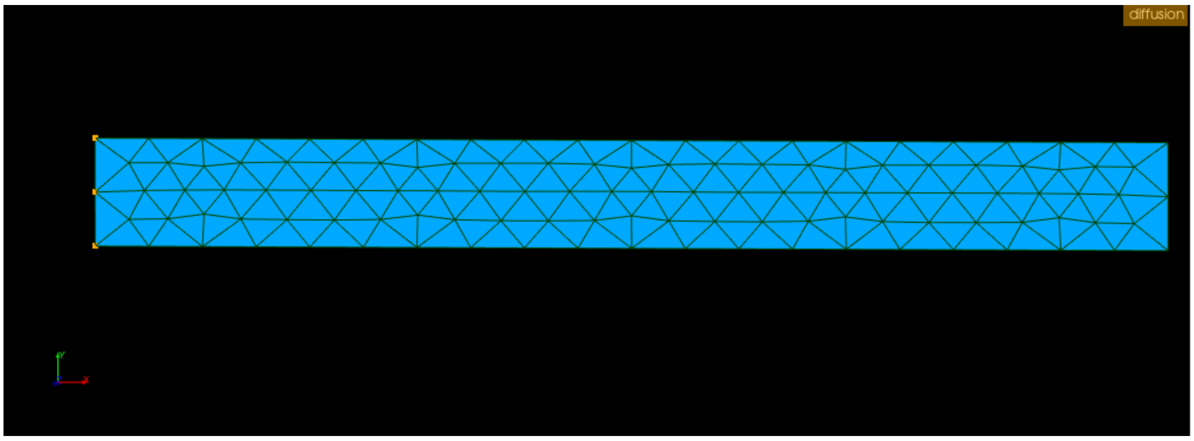


Figure 4.8: Two-Dimensional linear triangular mesh.

The second stage or formally called the processing stage is carried out in the jupyter console. The steps followed are as shown in figure 4.9. The python code utilizes the inbuilt libraries in a similar way as in the case of one-dimensional modeling. The .UNV file is read into the code and the transient analysis is progressed. As followed in the case of one-dimensional geometry, input parameters from reference [1] are taken and the first stage of validation of two-dimensional geometry is concerning results obtained in a one-dimensional finite element model. After the evaluation, the carbon percentage variation profile is obtained and JMatPro is run for general steel grade to obtain the hardness profile. For visualization of variation of carbon percentage in two dimensions, an output file .vtk is generated to be processed in the post-processing stage.

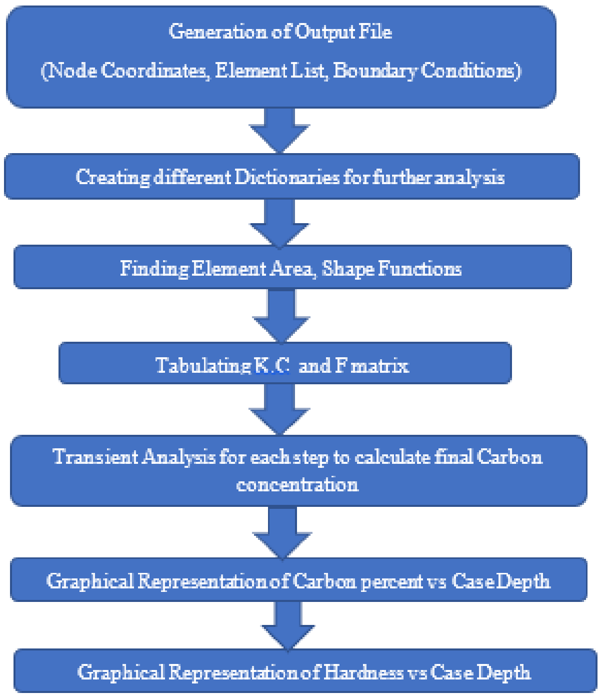

Generation of Output File
(Node Coordinates, Element List, Boundary Conditions)
Creating different Dictionaries for further analysis
Finding Element Area, Shape Functions
Tabulating K.C and F matrix
Transient Analysis for each step to calculate final Carbon concentration
Graphical Representation of Carbon percent vs Case Depth
Graphical Representation of Hardness vs Case Depth

Figure 4.9: Two-dimensional processing methodology

The post-processing is done in Paraview. The .vtk file obtained in the previous stage is given as an input to this stage. The two-dimensional distribution of carbon mass percentage after diffusion is as shown in figure 4.10. It can be observed that the diffusion is maximum at the surface and decreases exponentially with distance away from the surface. The carbon profile obtained was in precise comparison with what was obtained for the one-dimensional model.

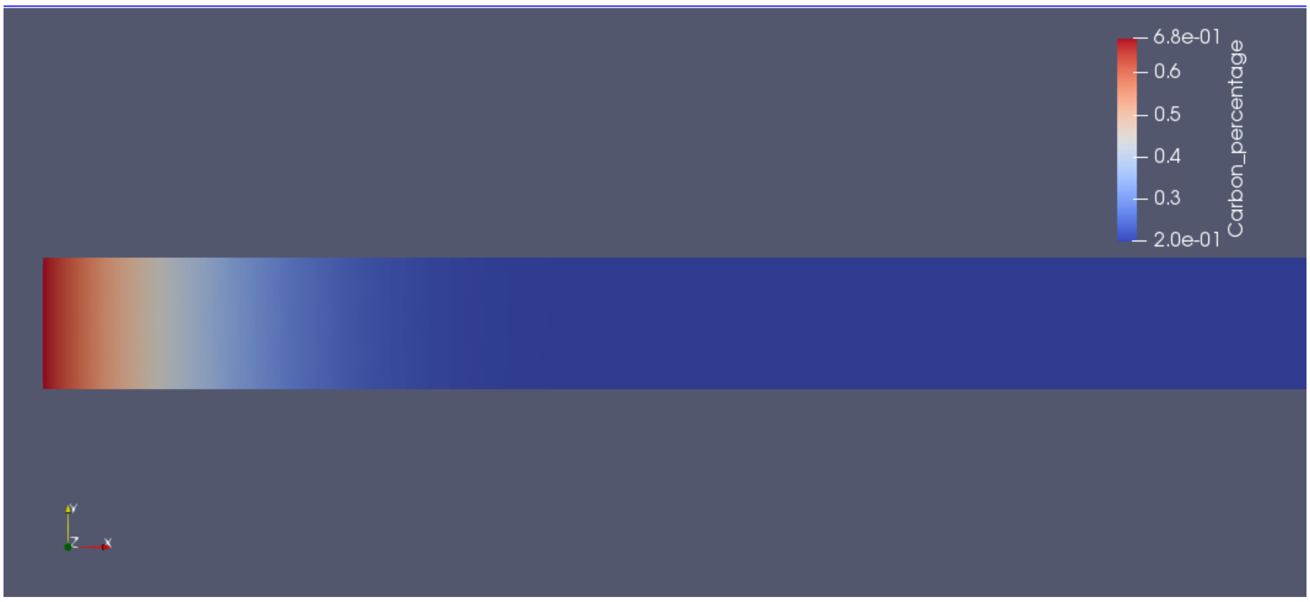


Figure 4.10: Carbon mass percentage visualization for 2D geometry

4.1.3 Three-Dimensional finite element modeling

Section 3.3, provides insight on the mathematical modeling of a three-dimensional geometry using the tetrahedral element, figure 4.11. The requirement of three-dimensional finite element modeling arises when three space coordinates are required to define volume contours. Case carburization is a heat treatment vastly used in the production of gear parts that have non-uniform geometries, varying contours, in such cases three-dimensional model is best suited.

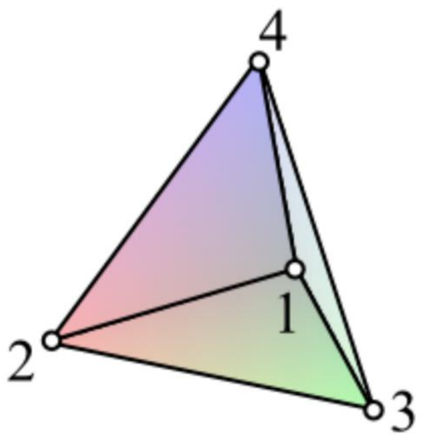


Figure 4.11: Three-dimensional Tetrahedral element

The methodology adopted for three-dimensional geometry is similar to that of two-dimensional geometry. The three processing stages are repeated, the three-dimensional tetrahedral mesh is as shown in figure 4.12 and the corresponding carbon mass percentage variation is shown in figure 4.13.

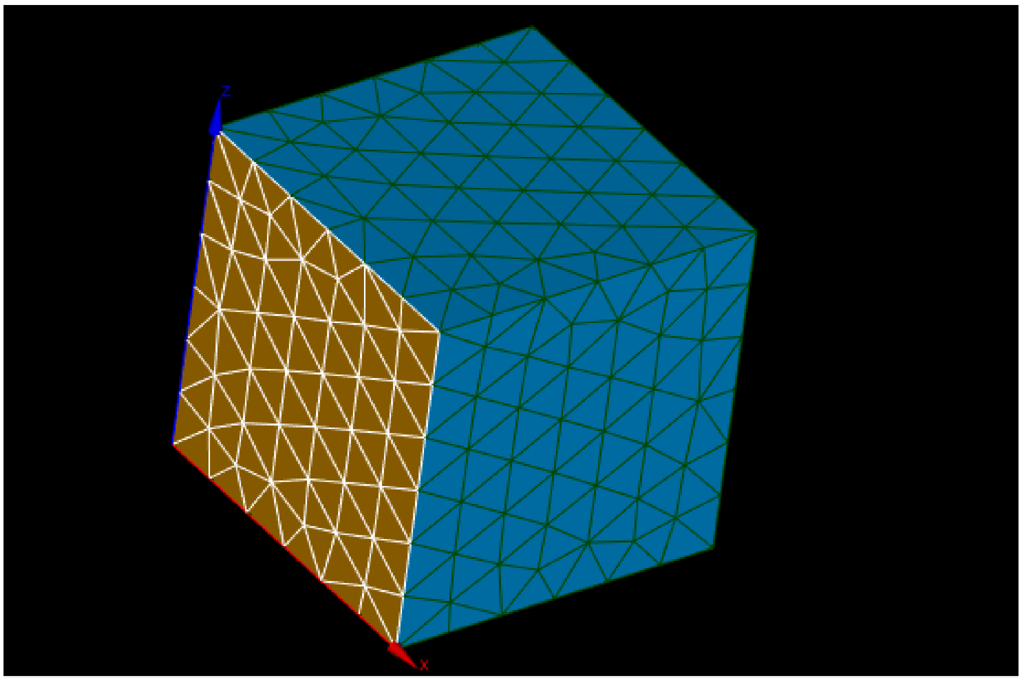


Figure 4.12: Three-dimensional tetrahedral mesh

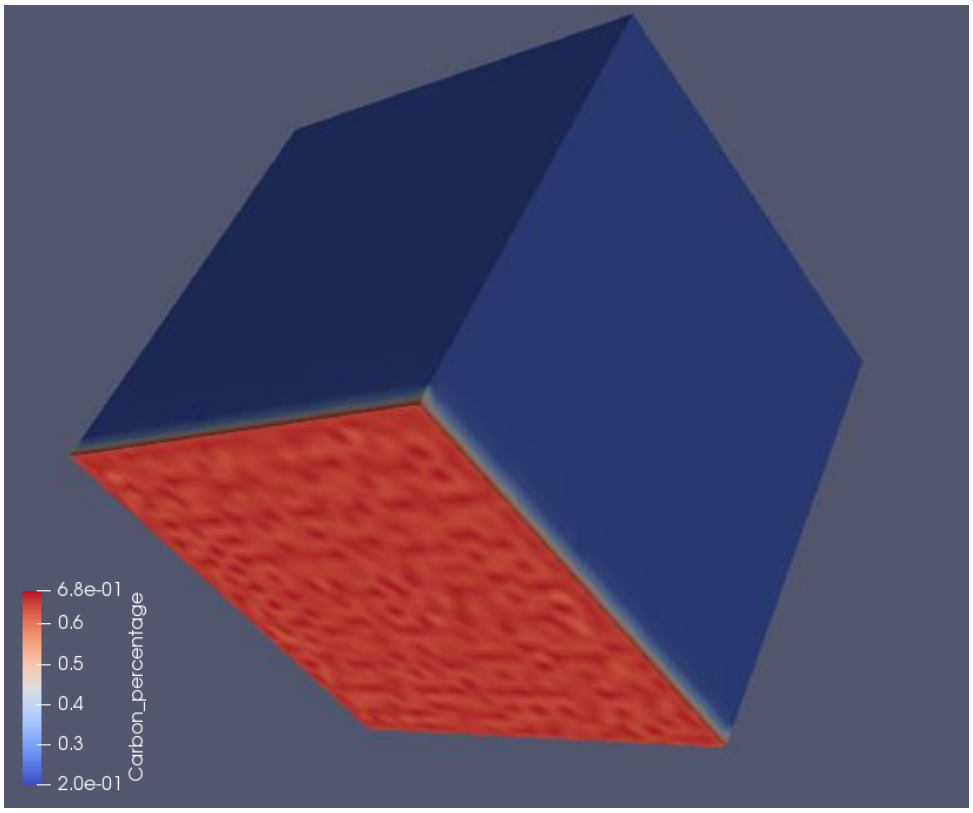


Figure 4.13: Carbon mass percentage visualization for 3D geometry

4.2 Hardness Prediction for SCM 420 steel grade

The experimental data provided by Sansera Engineering for prediction of case depth utilizes the hardness profile. The hardness calculation for a steel grade is dependent on its microstructure, its composition, the different phases present after quenching, tempering. As can be observed in Appendix B, the method used for carburizing the SCM 420 steel grade parts adopted by Sansera is Endo-gas carburization followed by oil quenching for 9-12 minutes. For obtaining the hardness profile directly from JMatPro version 7.0 required repetitive input of every carbon percentage obtained for each node, which is an exhaustive process. To simplify this approach, the hardness calculations were studied as a function of carbon percentage at a given temperature and cooling rate and an empirical relationship for established each cooling rate. For the process parameters provided by Sansera Engineering, the cooling rate was estimated to lie in the range of 7.5-15K/s. The empirical relationship formula established along with the mean root square error for cooling rates 7.5, 10, 12.5, 15 K/s is as shown in figure 4.13-4.16 respectively.

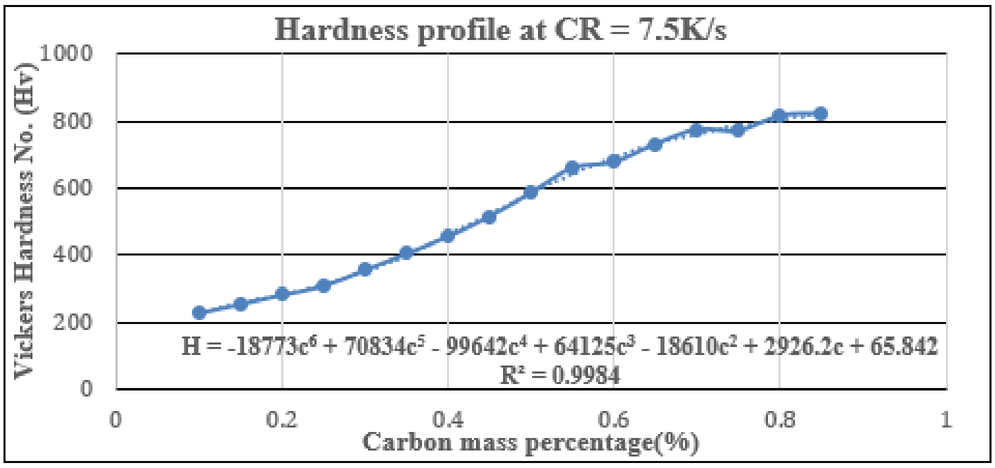


Figure 4.14: Hardness profile at a cooling rate of 7.5K/s

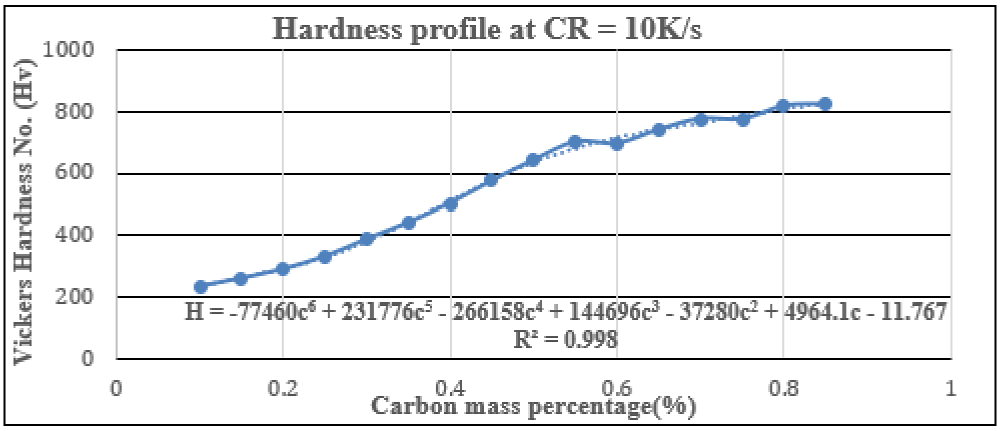


Figure 4.15: Hardness profile at a cooling rate of 10K/s

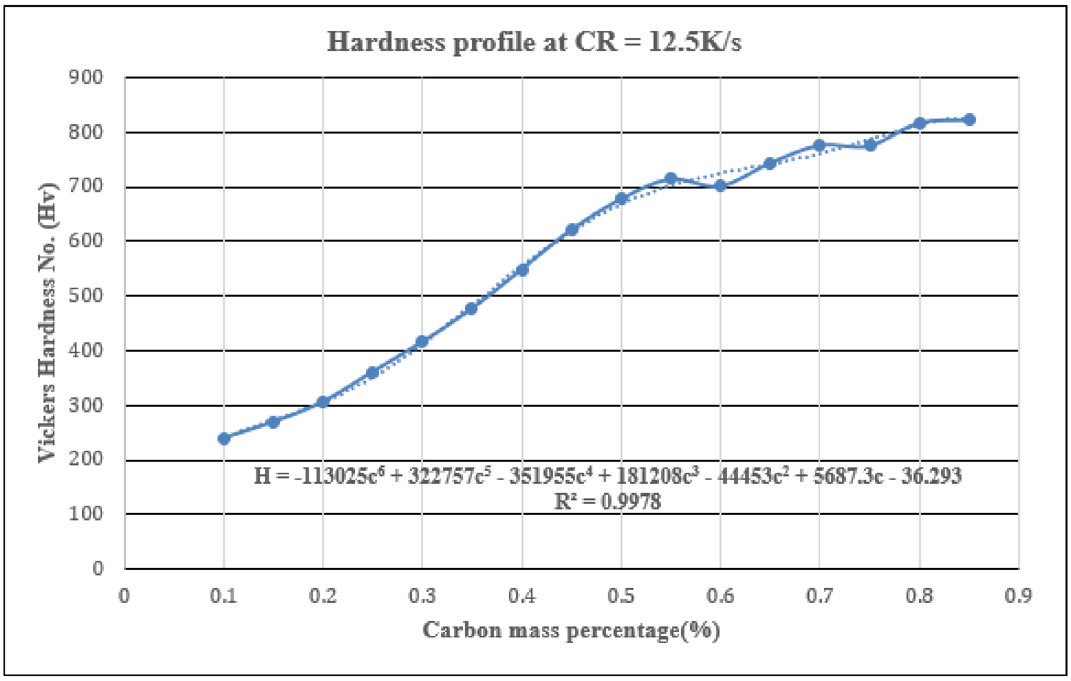


Figure 4.16: Hardness profile at a cooling rate of 12.5K/s

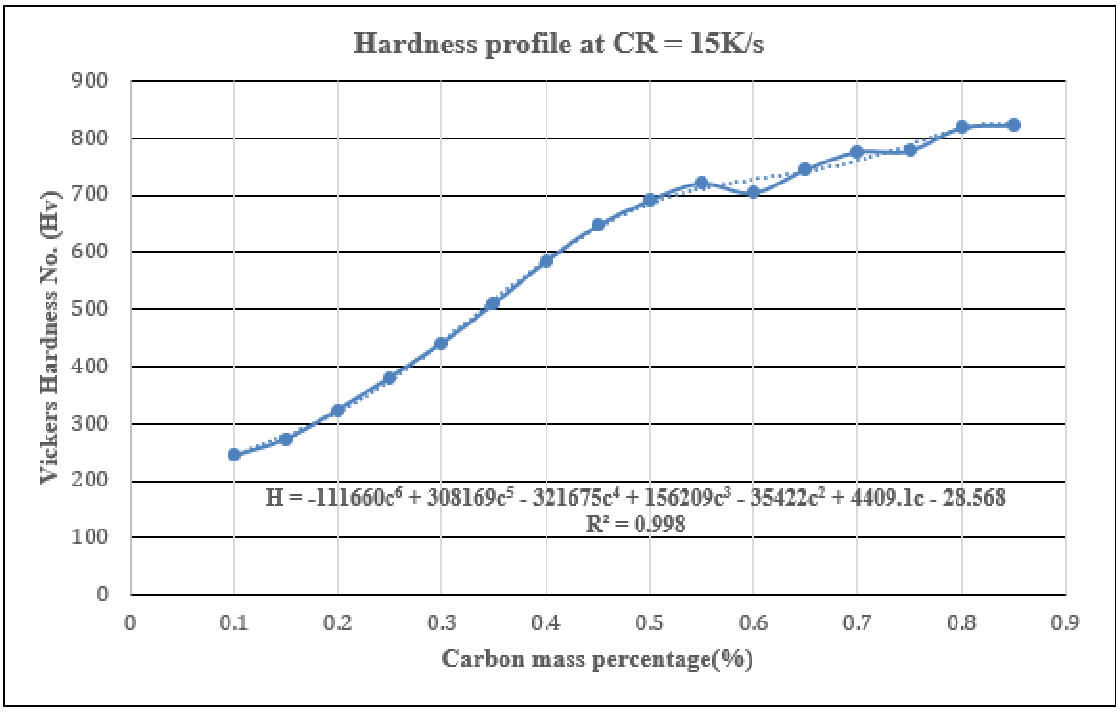


Figure 4.17: Hardness profile at a cooling rate of 15K/s

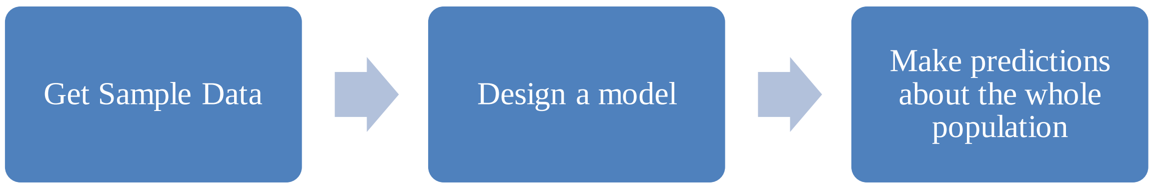
The empirical formula methodology is a sufficient relationship for predicting hardness for particular general steel. To expand our FE model to accommodate various other steel grades, a general hardness model using linear regressions techniques was designed in python and will be discussed in the upcoming sections.

4.3 Application of Machine Learning

In the field of data science, linear regression is the most widely used method for prediction. It serves as a base and starting point for advance deep learning techniques.

"A linear regression is defined as a linear approximation of a casual relationship between two or more variables."

Regression models are highly valuable, as they are one of the most common ways to make inferences and predictions. Apart from this, regression analysis is also employed to determine and assess factors that affect a certain outcome in a meaningful way. Like many other statistical techniques, regression models help us make predictions about the population based on sample data. The most frequently used regression models are linear regression, logistic regression, multiple linear regression, polynomial regression, stepwise regression, ridge regression, lasso regression, and ElasticNet regression.



4.3.1 Geometrical Representation of Linear Regression

$$y_i = bo + b_1x_i$$

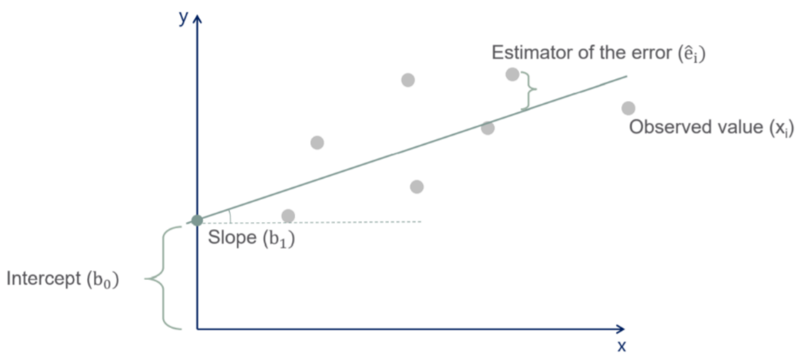


Figure 4.18: Linear Regression representation on the x-y plane

4.3.2 Multiple Regression

Since we have more than one input parameter to predict an output parameter, we choose multiple linear regression. Good models require multiple regressions to address the higher complexity of problems.

The equation for multiple regression:

$y = b_o + b_1x_1 + b_2x_2 + \dots\dots\dots + b_kx_k$

y = Inferred value

b_0 = Intercept

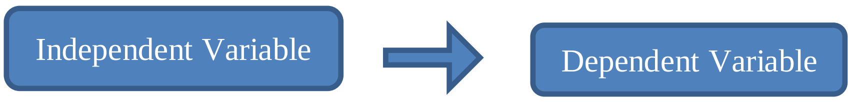
$x_1, x_2\dots., x_k$ = Independent variables

b1,b2,….,bk = Corresponding Coefficients

While linear regression focusses on finding the best fit line, multiple linear regression focuses on finding the best fitting model.

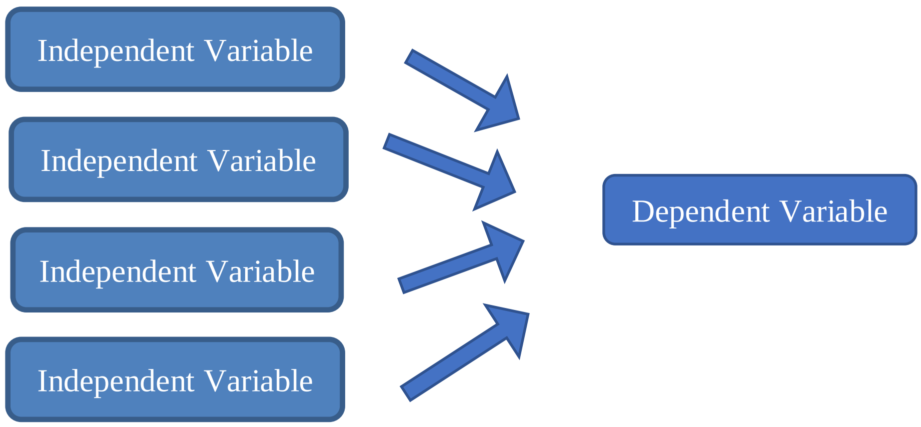
Simple Linear Regression

One-to-one



Multiple Linear Regression

Many-to-one



Metrics to determine the best model:

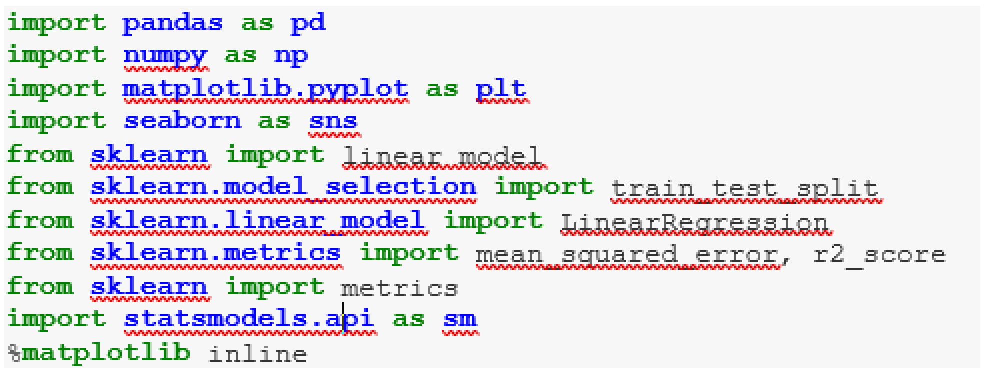
1) Root Mean Square Error (RMSE): When the square root of the average of squared differences are taken between the prediction and the actual observation, it is known as root mean root mean square error. RMSE = $\sqrt{\left(\frac{1}{n}\right)\sum_{k=0}^{n}(\widehat{yk}-yk)\text{^}2}$

2) R-Squared: It is defined as the percentage of the dependent variable variation that a linear model explains.

4.3.2.1 Application of Multiple Linear Regression

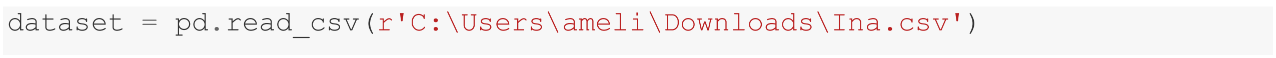
We aim to predict the hardness of the geared surface and for that we start by taking the input parameters as the composition of various elements of SCM420H and the cooling rates.

Step 1: We start by incorporating the various libraries and modules used for performing this machine learning technique.



```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import linear_model
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn import metrics
import statsmodels.api as sm
%matplotlib inline
```

Step 2: Load the data from a .csv in the same folder



```
dataset = pd.read_csv(r'C:\Users\ameli\Downloads\Ina.csv')
```

Step 3: Exploring the data

A great step in the data exploration is to display the probability distribution function (PDF) of the variable to be predicted. This makes it very easy to spot anomalies, such as outliers.

```
sns.distplot(dataset['Hardness']);
```

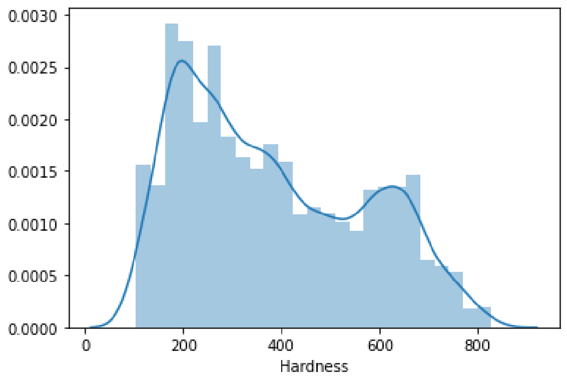


Figure 4.19: Probability distribution function of hardness

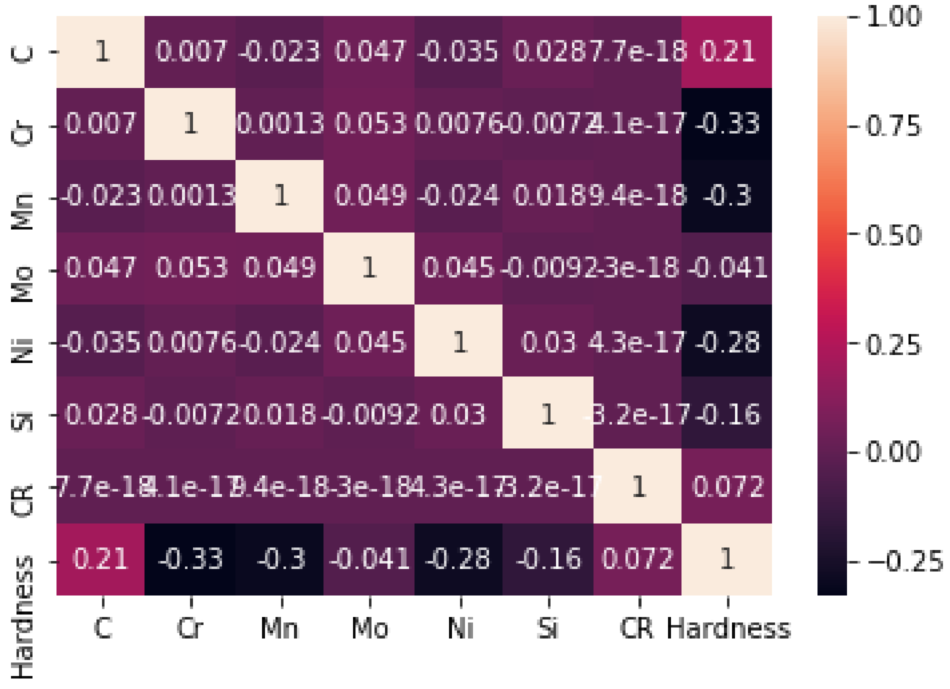
Step 4: We try to find how the variables are correlated to each other. The correlation of a variable with itself is 1. For this reason, all the diagonal values are 1.00. We should remove one of two features that have a correlation value higher than 0.9.

```
dataset.corr()
```

	C	Cr	Mn	Mo	Ni	Si	CR	Hardness
C	1.00	0.01	-0.02	0.05	-0.04	0.03	0.00	0.21
Cr	0.01	1.00	0.00	0.05	0.01	-0.01	0.00	-0.33
Mn	-0.02	0.00	1.00	0.05	-0.02	0.02	0.00	-0.30
Mo	0.05	0.05	0.05	1.00	0.04	-0.01	-0.00	-0.04
Ni	-0.04	0.01	-0.02	0.04	1.00	0.03	0.00	-0.28
Si	0.03	-0.01	0.02	-0.01	0.03	1.00	-0.00	-0.16
CR	0.00	0.00	0.00	-0.00	0.00	-0.00	1.00	0.07
Hardness	0.21	-0.33	-0.30	-0.04	-0.28	-0.16	0.07	1.00

Another way of representing the correlation between the elements is with the usage of heatmaps.

```
sns.heatmap(dataset.corr(),annot=True)
```



Step 5: Exploring the descriptive statistics of the variables

Descriptive statistics are very useful for the initial exploration of the variables. By default, only descriptives for the numerical variables are shown. To include the categorical ones, we should specify this with an argument.

```
dataset.describe()
```

	C	Cr	Mn	Mo	Ni	Si	CR	Hardness
count	9968.00	9968.00	9968.00	9968.00	9968.00	9968.00	9968.00	9968.00
mean	0.44	3.62	1.39	2.61	5.11	1.30	13.93	381.45
std	0.29	4.48	1.22	16.41	4.06	1.24	14.81	184.12
min	0.10	0.00	0.10	0.00	0.00	0.00	1.00	104.02
25%	0.10	0.00	0.10	0.00	0.00	0.00	4.00	220.01
50%	0.40	1.00	1.00	2.00	5.00	1.00	7.50	342.80
75%	0.80	10.00	3.00	4.00	10.00	3.00	20.00	537.51
max	0.80	10.00	3.00	438.00	10.00	3.00	50.00	827.95

Step 6: Preprocessing

Analyzing how each of the parameters varies with the other parameters

```
sns.pairplot(dataset)
```

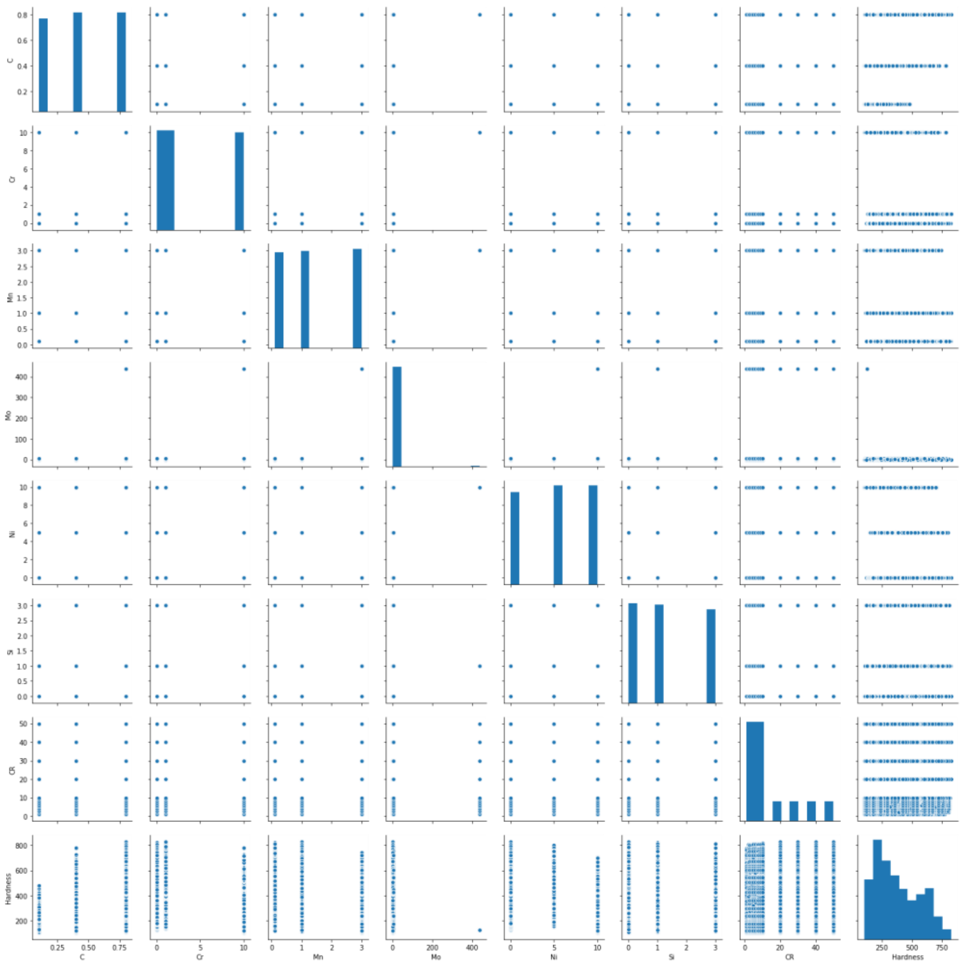


Figure 4.20: Pairplot of all the parameters with each other

Step 7: The data is checked for the type of values it contains for further evaluation.

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9968 entries, 0 to 9967
Data columns (total 8 columns):
C           9968 non-null float64
Cr          9968 non-null int64
Mn          9968 non-null float64
Mo          9968 non-null int64
Ni          9968 non-null int64
Si          9968 non-null int64
CR          9968 non-null int64
Hardness    9968 non-null float64
dtypes: float64(3), int64(5)
memory usage: 623.1 KB
```

Step 8: The size of the dataset is checked for a rough estimation of data points.

```
dataset.shape
```

```
(9968, 8)
```

Step 9: The data is now checked for any missing values. True implies that the data point is missing, while False implies that the data point is not missing.

```
dataset.isnull().any()
```

```
C           False
Cr          False
Mn          False
Mo          False
Ni          False
Si          False
CR          False
Hardness    False
dtype: bool
```

Step 10: The input and the target variables are declared

```
X = dataset[['C', 'Cr', 'Mn', 'Mo', 'Ni', 'Si', 'CR']].values
y = dataset['Hardness'].values
```

Step 11: Linear regression model – Train, test, split

We have already imported the module for performing train-test-split at the very beginning of sklearn.model_selection. The variables are split with an 80-20 split.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ra
ndom_state=0)
```

Training the Simple Linear Regression model on the Training set

Step 12: A linear regression object is created

```
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

Step 13: The weights i.e. the coefficients of the regression model are found.

```
print('Coefficients: \n', regressor.coef_)
Coefficients:
 [ 1.29354820e+02 -1.37089328e+01 -4.54878582e+01 -7.14078597e-02
 -1.25140108e+01 -2.31617250e+01  7.80698983e-01]
```

Step 14: The output of the regression model is calculated. These values are stored in y_pred.

```
y_pred = regressor.predict(X_train)
```

Step 15: Comparison of the data

The simplest way to compare the targets (y_train) and the predictions (y_pred) is to plot them on a scatter plot. The closer the points to the 45-degree line, the better the prediction

```
plt.scatter(y_train,y_pred)
```

The axes are also named.

```
plt.xlabel('Y Test')plt.ylabel('Predicted Y')
```

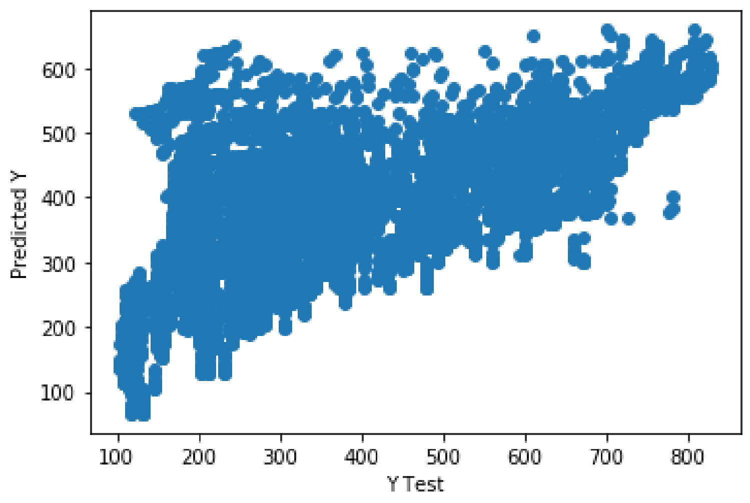


Figure 4.21: Predicted ‘y’ versus Test ‘y’

Step 16: Further analysis of our output

Another useful check of our model is a residual plot. We can plot the Probability Distribution Function of the residuals and check for anomalies.

```
sns.distplot(y_train - y_pred)
plt.title("Residuals PDF", size=18)
```

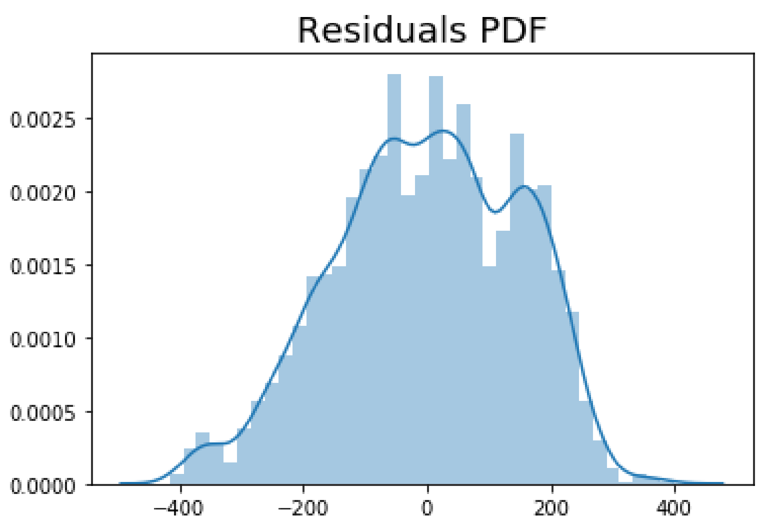


Figure 4.22: Probability distribution function of the residuals

Step 17: Finding how much validity does our model hold

```
regressor.score(X_train,y_train)
```

```
0.35005184159115665
```

Step 18: The bias(intercept) of the regression is obtained.

```
regressor.intercept_
```

```
520.6400251720138
```

Step 19: The weights(coefficients) of the regression are also obtained.

```
regressor.coef_
```

```
array([ 1.29354820e+02, -1.37089328e+01, -4.54878582e+01, -7.14078597e-02,-
1.25140108e+01, -2.31617250e+01,  7.80698983e-01])
```

Step 20: The Mean Absolute Error, Mean Squared Error and Root Mean Squared Error are calcula ted which form the metrics for further analysis

```
print('Mean Absolute Error:', metrics.mean_absolute_error(y_train, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_train, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_trai
n, y_pred)))
```

```
Mean Absolute Error: 122.03285443345882
Mean Squared Error: 21960.58154512275
Root Mean Squared Error: 148.1910305825651
```

Testing the Simple Linear Regression model on the Testing set

Step 21: Once we have trained and fine-tuned our model, we can proceed to testing it. Testing is done on a dataset that the algorithm has never seen. Our test inputs are 'x_test', while the outputs are 'y_test'. We SHOULD NOT TRAIN THE MODEL ON THEM; we just feed them and find the predictions. If the predictions are far off, we will know that our model overfitted.

```
y_pred_test = regressor.predict(X_test)
```

Step 22: A scatter plot is created with the test targets and the test predictions.

```
plt.scatter(y_test,y_pred_test)
plt.xlabel('Y Test')
plt.ylabel('Predicted Y')
```

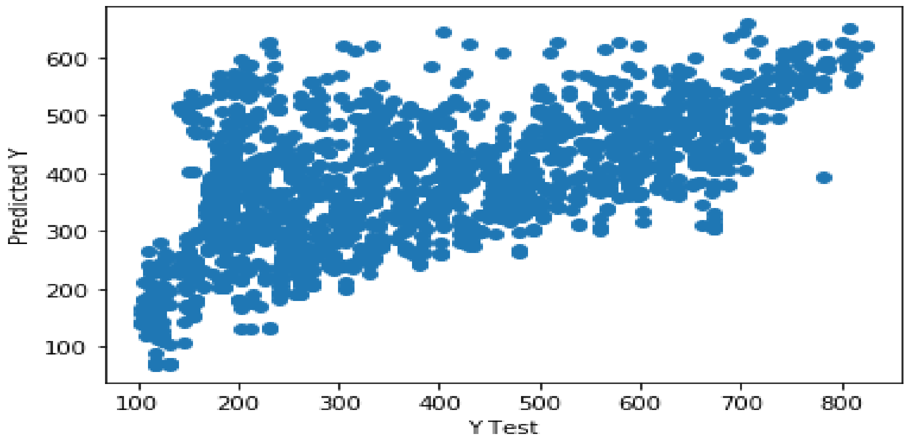
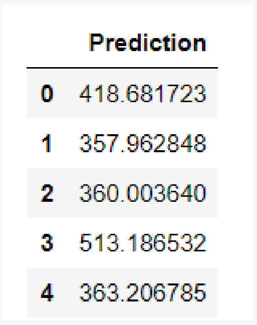


Figure 4.23: Predicted ‘y’ versus Test ‘y’

Step 21: Finally, the predictions are checked manually.

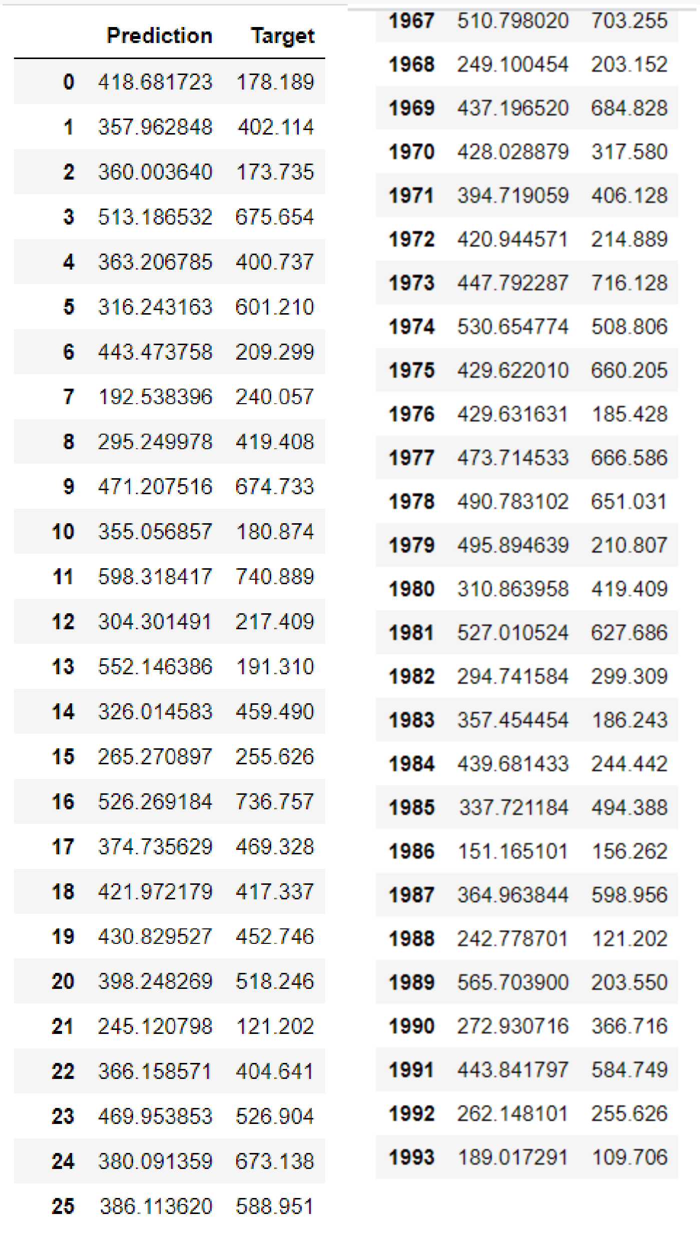
```
df_pf = pd.DataFrame(y_pred_test, columns=['Prediction'])
df_pf.head()
```



	Prediction
0	418.681723
1	357.962848
2	360.003640
3	513.186532
4	363.206785

Step 22: The test targets are also included in the dataframe so that they can be manually compared.

```
df_pf['Target'] = y_test
```



	Prediction	Target
0	418.681723	178.189
1	357.962848	402.114
2	360.003640	173.735
3	513.186532	675.654
4	363.206785	400.737
5	316.243163	601.210
6	443.473758	209.299
7	192.538396	240.057
8	295.249978	419.408
9	471.207516	674.733
10	355.056857	180.874
11	598.318417	740.889
12	304.301491	217.409
13	552.146386	191.310
14	326.014583	459.490
15	265.270897	255.626
16	526.269184	736.757
17	374.735629	469.328
18	421.972179	417.337
19	430.829527	452.746
20	398.248269	518.246
21	245.120798	121.202
22	366.158571	404.641
23	469.953853	526.904
24	380.091359	673.138
25	386.113620	588.951
1967	510.798020	703.255
1968	249.100454	203.152
1969	437.196520	684.828
1970	428.028879	317.580
1971	394.719059	406.128
1972	420.944571	214.889
1973	447.792287	716.128
1974	530.654774	508.806
1975	429.622010	660.205
1976	429.631631	185.428
1977	473.714533	666.586
1978	490.783102	651.031
1979	495.894639	210.807
1980	310.863958	419.409
1981	527.010524	627.686
1982	294.741584	299.309
1983	357.454454	186.243
1984	439.681433	244.442
1985	337.721184	494.388
1986	151.165101	156.262
1987	364.963844	598.956
1988	242.778701	121.202
1989	565.703900	203.550
1990	272.930716	366.716
1991	443.841797	584.749
1992	262.148101	255.626
1993	189.017291	109.706

Step 23: Additionally, the difference between the targets and the predictions can also be calculated.

```
df_pf['Residual'] = df_pf['Target'] - df_pf['Prediction']
```

Step 24: Finally, it makes sense to see how far off we are from the result percentage-wise. We take the absolute difference in %, so that the data frame can be easily ordered.

```
df_pf['Difference%'] = np.absolute(df_pf['Residual']/df_pf['Target']*100)
df_pf
```

	Prediction	Target	Residual	Difference%
0	418.681723	178.189	-240.492723	134.964966
1	357.962848	402.114	44.151152	10.979760
2	360.003640	173.735	-186.268640	107.214228
3	513.186532	675.654	162.467468	24.045957
4	363.206785	400.737	37.530215	9.365298
5	316.243163	601.210	284.966837	47.398885
6	443.473758	209.299	-234.174758	111.885273
7	192.538396	240.057	47.518604	19.794717
8	295.249978	419.408	124.158022	29.603160
9	471.207516	674.733	203.525484	30.163855
10	355.056857	180.874	-174.182857	96.300661
11	598.318417	740.889	142.570583	19.243177
12	304.301491	217.409	-86.892491	39.967293
13	552.146386	191.310	-360.836386	188.613447
14	326.014583	459.490	133.475417	29.048601
15	265.270897	255.626	-9.644897	3.773050
16	526.269184	736.757	210.487816	28.569503
17	374.735629	469.328	94.592371	20.154854
18	421.972179	417.337	-4.635179	1.110656
19	430.829527	452.746	21.916473	4.840788
20	398.248269	518.246	119.997731	23.154589
21	245.120798	121.202	-123.918798	102.241545
22	366.158571	404.641	38.482429	9.510264
23	469.953853	526.904	56.950147	10.808448
24	380.091359	673.138	293.046641	43.534408
25	386.113620	588.951	202.837380	34.440451
26	234.523137	239.826	5.302863	2.211129
27	370.348222	554.275	183.926778	33.183308
28	565.422158	697.045	131.622842	18.882976
29	88.818291	117.109	28.290709	24.157587
...	...	...	...	...
1964	472.702259	154.988	-317.714259	204.992812
1965	227.630152	175.022	-52.608152	30.058022
1966	474.495145	327.417	-147.078145	44.920742
1967	510.798020	703.255	192.456980	27.366600
1968	249.100454	203.152	-45.948454	22.617771
1969	437.196520	684.828	247.631480	36.159660
1970	428.028879	317.580	-110.448879	34.778286
1971	394.719059	406.128	11.408941	2.809198
1972	420.944571	214.889	-206.055571	95.889306
1973	447.792287	716.128	268.335713	37.470356
1974	530.654774	508.806	-21.848774	4.294127
1975	429.622010	660.205	230.582990	34.925968
1976	429.631631	185.428	-244.203631	131.697279
1977	473.714533	666.586	192.871467	28.934221
1978	490.783102	651.031	160.247898	24.614480
1979	495.894639	210.807	-285.087639	135.236325
1980	310.863958	419.409	108.545042	25.880475
1981	527.010524	627.686	100.675476	16.039146
1982	294.741584	299.309	4.567416	1.525987
1983	357.454454	186.243	-171.211454	91.929068
1984	439.681433	244.442	-195.239433	79.871476

Step 25: Additional insights on the output are given by exploring the descriptives.

```
df_pf.describe()
```

	Prediction	Target	Residual	Difference%
count	1994.000000	1994.000000	1994.000000	1994.000000
mean	384.035526	383.081407	-0.954119	40.928531
std	110.750296	185.318172	149.535156	42.198455
min	68.520118	104.017000	-396.276289	0.015739
25%	305.552974	220.498000	-102.558242	14.763888
50%	386.226280	346.282000	5.796760	27.105909
75%	466.776360	547.331000	119.682603	47.823966
max	658.324413	823.672000	389.579008	265.740558

Step 26: Sometimes it is useful to check these outputs manually. To see all rows, we use the relevant pandas syntax.

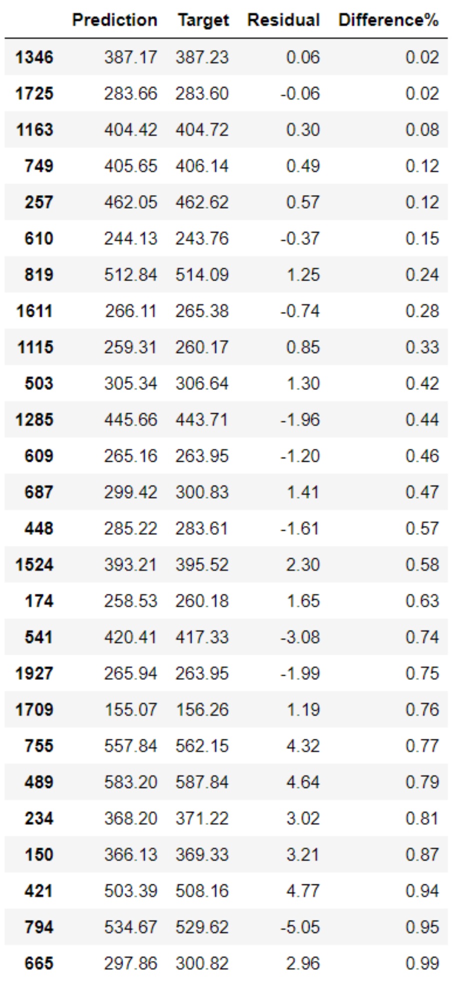
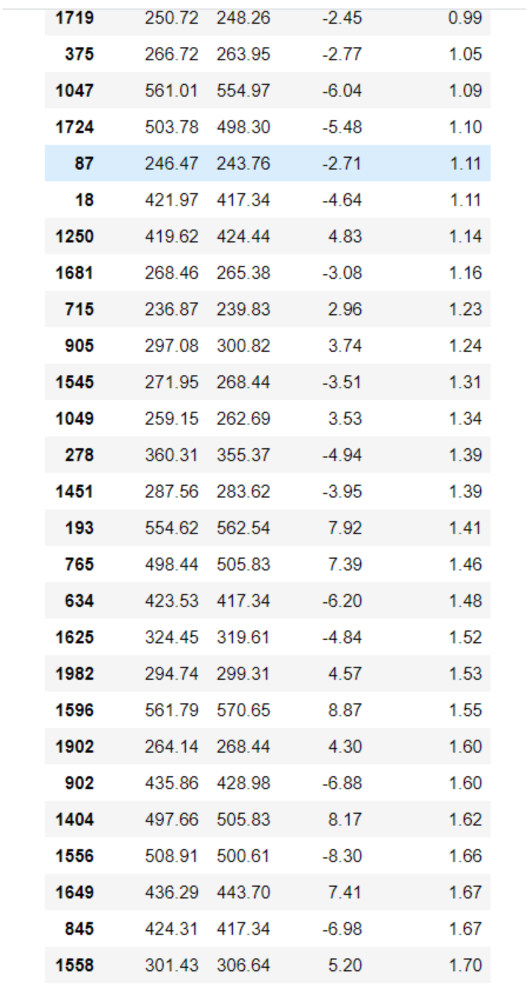
```
pd.options.display.max_rows = 999
```

Moreover, to make the dataset clear, we can display the result with only 2 digits after the dot.

```
pd.set_option('display.float_format', lambda x: '%.2f' % x)
```

Finally, difference is sorted in % and the model is made ready for better prediction of values.

```
df_pf.sort_values(by=['Difference%'])
```

	Prediction	Target	Residual	Difference%
1346	387.17	387.23	0.06	0.02
1725	283.66	283.60	-0.06	0.02
1163	404.42	404.72	0.30	0.08
749	405.65	406.14	0.49	0.12
257	462.05	462.62	0.57	0.12
610	244.13	243.76	-0.37	0.15
819	512.84	514.09	1.25	0.24
1611	266.11	265.38	-0.74	0.28
1115	259.31	260.17	0.85	0.33
503	305.34	306.64	1.30	0.42
1285	445.66	443.71	-1.96	0.44
609	265.16	263.95	-1.20	0.46
687	299.42	300.83	1.41	0.47
448	285.22	283.61	-1.61	0.57
1524	393.21	395.52	2.30	0.58
174	258.53	260.18	1.65	0.63
541	420.41	417.33	-3.08	0.74
1927	265.94	263.95	-1.99	0.75
1709	155.07	156.26	1.19	0.76
755	557.84	562.15	4.32	0.77
489	583.20	587.84	4.64	0.79
234	368.20	371.22	3.02	0.81
150	366.13	369.33	3.21	0.87
421	503.39	508.16	4.77	0.94
794	534.67	529.62	-5.05	0.95
665	297.86	300.82	2.96	0.99
1719	250.72	248.26	-2.45	0.99
375	266.72	263.95	-2.77	1.05
1047	561.01	554.97	-6.04	1.09
1724	503.78	498.30	-5.48	1.10
87	246.47	243.76	-2.71	1.11
18	421.97	417.34	-4.64	1.11
1250	419.62	424.44	4.83	1.14
1681	268.46	265.38	-3.08	1.16
715	236.87	239.83	2.96	1.23
905	297.08	300.82	3.74	1.24
1545	271.95	268.44	-3.51	1.31
1049	259.15	262.69	3.53	1.34
278	360.31	355.37	-4.94	1.39
1451	287.56	283.62	-3.95	1.39
193	554.62	562.54	7.92	1.41
765	498.44	505.83	7.39	1.46
634	423.53	417.34	-6.20	1.48
1625	324.45	319.61	-4.84	1.52
1982	294.74	299.31	4.57	1.53
1596	561.79	570.65	8.87	1.55
1902	264.14	268.44	4.30	1.60
902	435.86	428.98	-6.88	1.60
1404	497.66	505.83	8.17	1.62
1556	508.91	500.61	-8.30	1.66
1649	436.29	443.70	7.41	1.67
845	424.31	417.34	-6.98	1.67
1558	301.43	306.64	5.20	1.70

Step 27: The Mean Absolute Error, Mean Squared Error and Root Mean Squared Error are calcula
ted on the test data.

```
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred_te
st))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred_test
))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test
, y_pred_test)))
```

```
Mean Absolute Error: 122.16239900616169
Mean Squared Error: 22350.459146838184
Root Mean Squared Error: 149.50069948611673
```

Step 28: Finding how much validity does our model hold

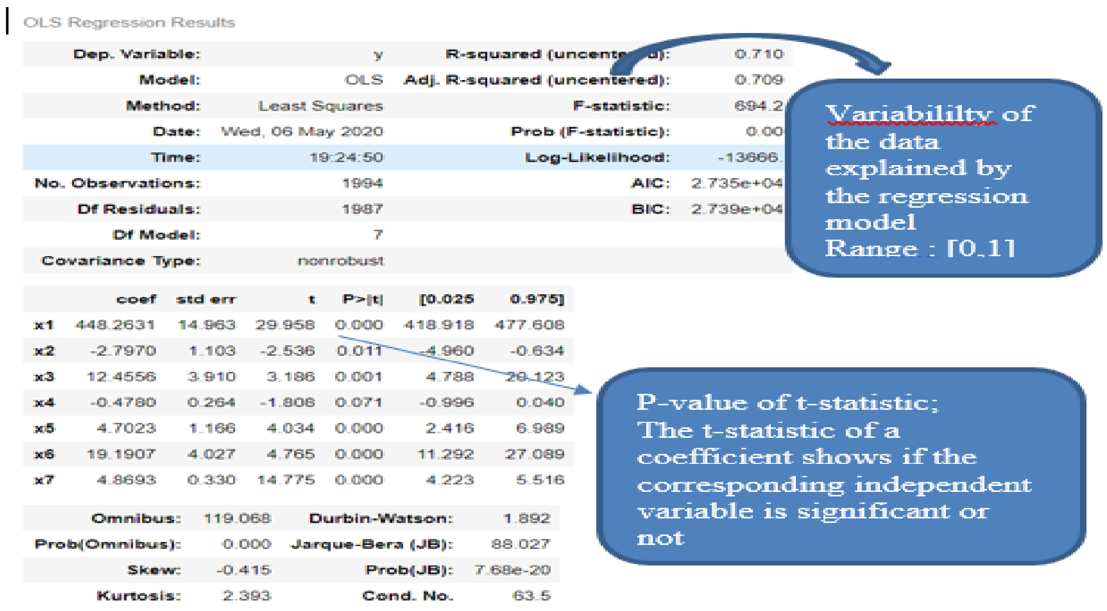
```
regressor.score(X_test,y_test)
0.34886912207507337
```

Step 29: After performing multiple linear regression on test and train data, it is imperative to know if any parameters are not a significant predictor of hardness. To find that, we use statsmodel api which has already been imported at the very beginning. We now implement linear regression to our data using StatsModel.

```
model = sm.OLS(y_test,X_test)
results = model.fit()
results_summary = results.summary()
```

```
results_summary
```

We can clearly see that the p-value of x4 which is Mo is greater than 0.05. Therefore, we can conclude that Mo is not a significant predictor of Mo as compared to the other input parameters.


Variabililty of the data explained by the regression model Range : [0,1]
P-value of t-statistic; The t-statistic of a coefficient shows if the corresponding independent variable is significant or not

OLS Regression Results

Dep. Variable:	y	R-squared (uncentered):	0.710
Model:	OLS	Adj. R-squared (uncentered):	0.709
Method:	Least Squares	F-statistic:	694.2
Date:	Wed, 06 May 2020	Prob (F-statistic):	0.00
Time:	19:24:50	Log-Likelihood:	-13666.
No. Observations:	1994	AIC:	2.735e+04
Df Residuals:	1987	BIC:	2.739e+04
Df Model:	7		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
x1	448.2631	14.963	29.958	0.000	418.918	477.608
x2	-2.7970	1.103	-2.536	0.011	-4.960	-0.634
x3	12.4556	3.910	3.186	0.001	4.788	20.123
x4	-0.4780	0.264	-1.808	0.071	-0.996	0.040
x5	4.7023	1.166	4.034	0.000	2.416	6.989
x6	19.1907	4.027	4.765	0.000	11.292	27.089
x7	4.8693	0.330	14.775	0.000	4.223	5.516

Omnibus:	119.068	Durbin-Watson:	1.892
Prob(Omnibus):	0.000	Jarque-Bera (JB):	88.027
Skew:	-0.415	Prob(JB):	7.68e-20
Kurtosis:	2.393	Cond. No.	63.5

Figure 4.24: Summary results of the regression performed

Step 30: The residual plot is plotted again to check for any anomalies.

```
sns.distplot((y_test-y_pred_test),bins=50);
plt.title("Residuals PDF", size=18)
```

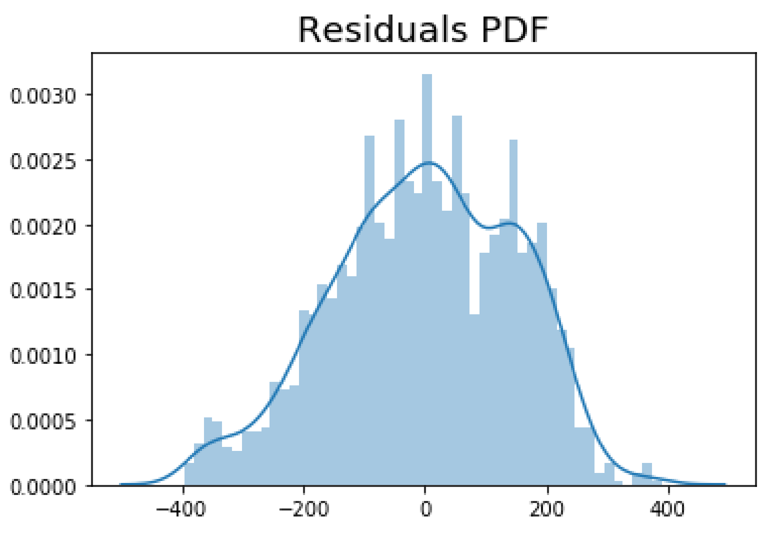
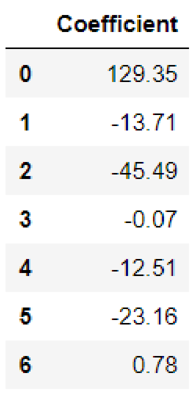


Figure 4.25: Probability Distribution Function of the Residuals

Step 31: The coefficients of each of the input parameters is finally calculated.

```
x_df = pd.DataFrame(X)
coefficients = pd.DataFrame(regressor.coef_,x_df.columns)
coefficients.columns = ['Coefficient']
coefficients
```



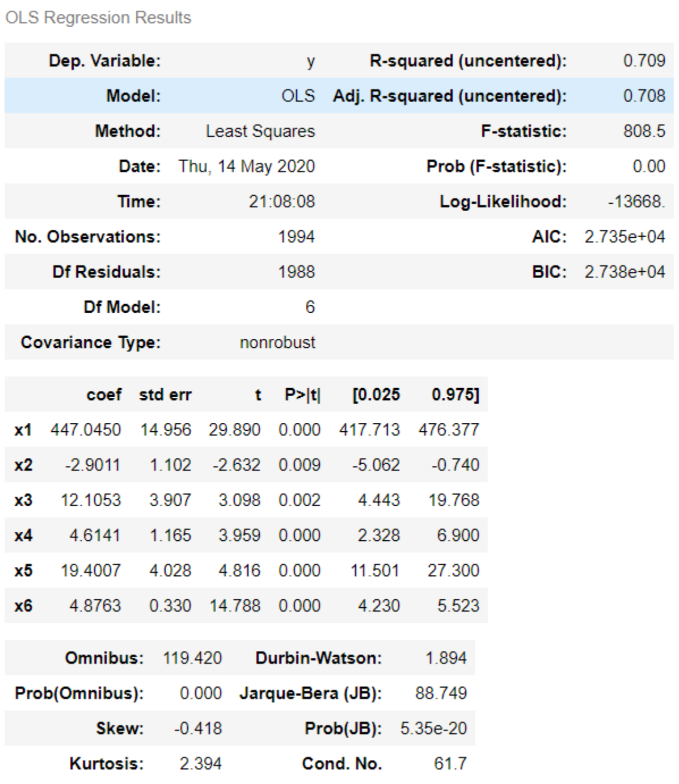
	Coefficient
0	129.35
1	-13.71
2	-45.49
3	-0.07
4	-12.51
5	-23.16
6	0.78

Step 32: The y-intercept of the linear equation is calculated.

```
print ('intercept:', regressor.intercept_)
intercept: 520.6400251720138
```

4.3.2.2 Multiple Linear Regression without 'Mo'

From the previous section, it can be concluded that Mo is not a significant predictor of Mo as compared to the other input parameters. It has a p value greater than 0.05. Therefore, linear regression is now performed similarly excluding Mo from the list of input parameters to predict hardness. We get the following outputs:



OLS Regression Results

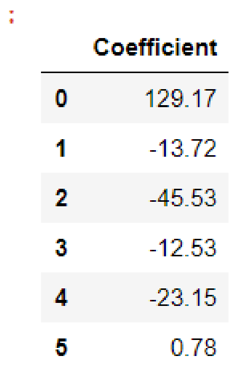
Dep. Variable:	y	**R-squared (uncentered):**	0.709
Model:	OLS	**Adj. R-squared (uncentered):**	0.708
Method:	Least Squares	**F-statistic:**	808.5
Date:	Thu, 14 May 2020	**Prob (F-statistic):**	0.00
Time:	21:08:08	**Log-Likelihood:**	-13668.
No. Observations:	1994	**AIC:**	2.735e+04
Df Residuals:	1988	**BIC:**	2.738e+04
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
x1	447.0450	14.956	29.890	0.000	417.713	476.377
x2	-2.9011	1.102	-2.632	0.009	-5.062	-0.740
x3	12.1053	3.907	3.098	0.002	4.443	19.768
x4	4.6141	1.165	3.959	0.000	2.328	6.900
x5	19.4007	4.028	4.816	0.000	11.501	27.300
x6	4.8763	0.330	14.788	0.000	4.230	5.523

Omnibus:	119.420	**Durbin-Watson:**	1.894
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	88.749
Skew:	-0.418	**Prob(JB):**	5.35e-20
Kurtosis:	2.394	**Cond. No.**	61.7

Figure 4.26: Summary results of the regression performed

```
Mean Absolute Error: 122.19974115652252
Mean Squared Error: 22354.070636188582
Root Mean Squared Error: 149.51277750141819
```



```
:
   Coefficient
0       129.17
1       -13.72
2       -45.53
3       -12.53
4       -23.15
5         0.78
intercept: 520.6906284849083
```

4.3.2.3 Normalisation of the Data

While performing linear regression, an assumption is considered to get better results, which is the normalization of the output parameter. From the subplots, it can be easily determined that 'Hardness' is not normally distributed. A good transformation, in that case, is a log transformation or a square root transformation.

1)Log transformation

C	Cr	Mn	Ni	Si	CR	Hardness	Hardness_log
0.1	0	0.1	0	0	1	122.001	2.08636339
0.1	0	0.1	0	0	2	130.595	2.11592655
0.1	0	0.1	0	0	3	135.403	2.131628287
0.1	0	0.1	0	0	4	139.376	2.144187996
0.1	0	0.1	0	0	5	142.358	2.153381878
0.1	0	0.1	0	0	6	144.673	2.160387487
0.1	0	0.1	0	0	7	146.875	2.16694788
0.1	0	0.1	0	0	8	148.84	2.172719661
0.1	0	0.1	0	0	9	150.578	2.177761524

Table 4.1: Input parameters with Log of hardness

2)Square root transformation

C	Cr	Mn	Ni	Si	CR	Hardness	Hardness_sqrt
0.1	0	0.1	0	0	1	122.001	11.04540628
0.1	0	0.1	0	0	2	130.595	11.42781694
0.1	0	0.1	0	0	3	135.403	11.63627947
0.1	0	0.1	0	0	4	139.376	11.80576131
0.1	0	0.1	0	0	5	142.358	11.93138718
0.1	0	0.1	0	0	6	144.673	12.02800898
0.1	0	0.1	0	0	7	146.875	12.11919964
0.1	0	0.1	0	0	8	148.84	12.2
0.1	0	0.1	0	0	9	150.578	12.27102278

Table 4.2: Input parameters with Square root of hardness

Mo is dropped as it is not a significant predictor in hardness.

4.3.2.4 Multiple Linear Regression with Log Transformation

Linear regression is performed similarly as described before. There are two noticeable changes now:

1)Mo is not taken into considerable

2)Instead of Hardness, log (Hardness) is predicted.

```
sns.distplot(dataset['Hardness_log']);
```

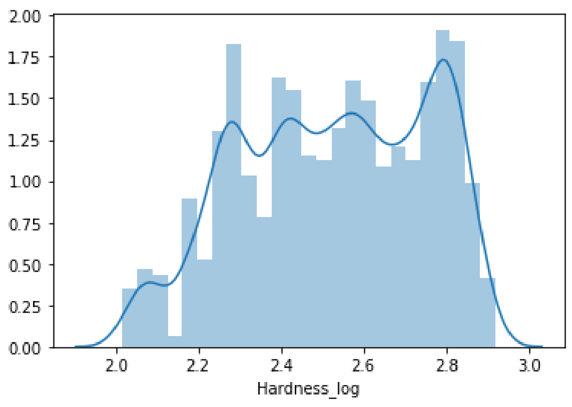
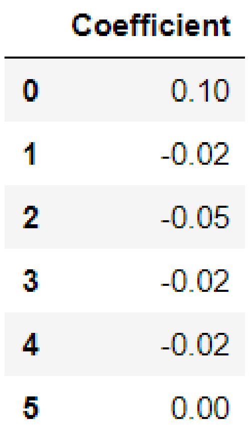
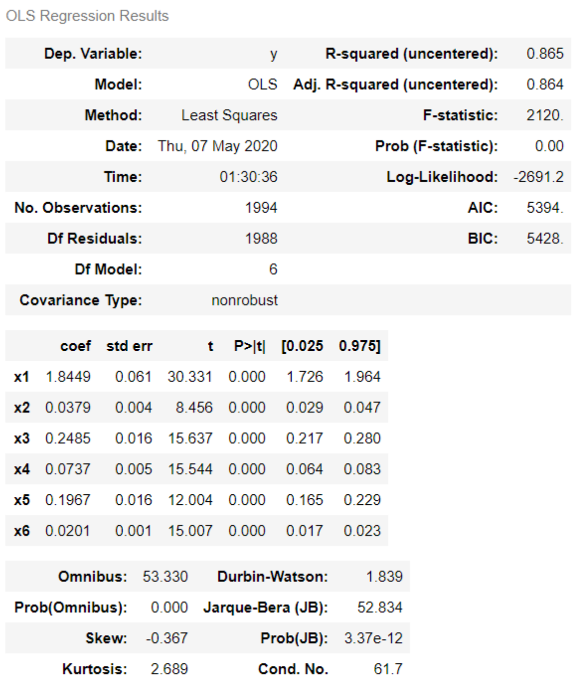


Figure 4.27: Probability Distribution Function of Hardness Log

```
Mean Absolute Error: 0.15040587815708395
Mean Squared Error: 0.03290492517205965
Root Mean Squared Error: 0.181397147640363    intercept: 2.732259026308569
```



	Coefficient
0	0.10
1	-0.02
2	-0.05
3	-0.02
4	-0.02
5	0.00



OLS Regression Results

Dep. Variable:	y	R-squared (uncentered):	0.865
Model:	OLS	Adj. R-squared (uncentered):	0.864
Method:	Least Squares	F-statistic:	2120.
Date:	Thu, 07 May 2020	Prob (F-statistic):	0.00
Time:	01:30:36	Log-Likelihood:	-2691.2
No. Observations:	1994	AIC:	5394.
Df Residuals:	1988	BIC:	5428.
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
x1	1.8449	0.061	30.331	0.000	1.726	1.964
x2	0.0379	0.004	8.456	0.000	0.029	0.047
x3	0.2485	0.016	15.637	0.000	0.217	0.280
x4	0.0737	0.005	15.544	0.000	0.064	0.083
x5	0.1967	0.016	12.004	0.000	0.165	0.229
x6	0.0201	0.001	15.007	0.000	0.017	0.023

Omnibus:	53.330	Durbin-Watson:	1.839
Prob(Omnibus):	0.000	Jarque-Bera (JB):	52.834
Skew:	-0.367	Prob(JB):	3.37e-12
Kurtosis:	2.689	Cond. No.	61.7

Figure 4.28: Summary Results from the Regression performed

4.3.2.5 Multiple Linear Regression with Square Root Transformation

Linear regression is performed similarly as described before. There are two noticeable changes now:

1)Mo is not taken into considerable

2)Instead of Hardness, Square Root (Hardness) is predicted.

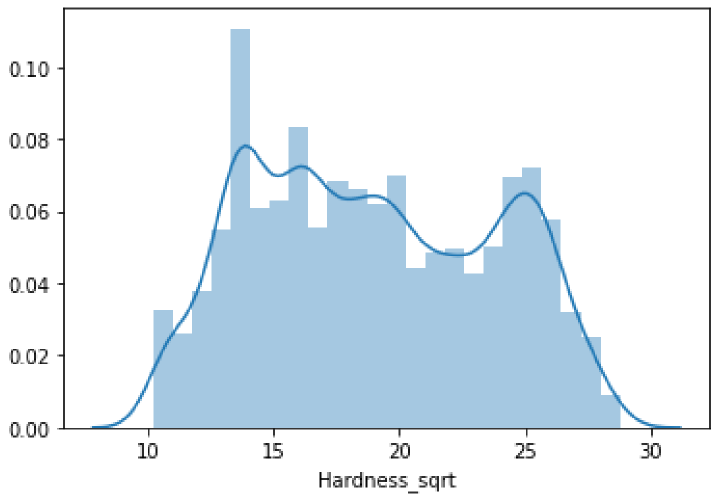
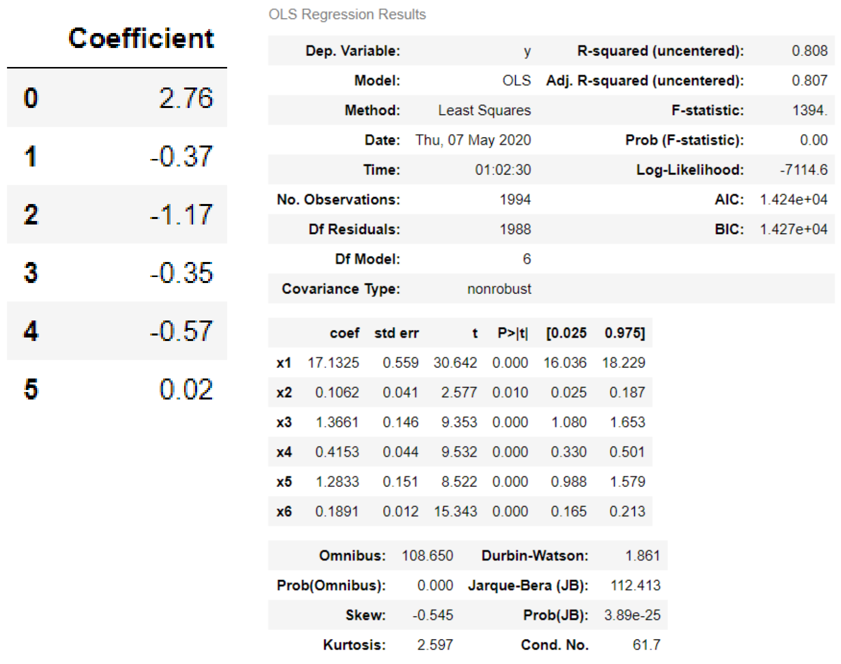


Figure 4.29: Probability Distribution Function of sqrt (Hardness)

```
Mean Absolute Error: 3.1749142257238674
Mean Squared Error: 14.768668218074364
Root Mean Squared Error: 3.843002500399182
```



	Coefficient
0	2.76
1	-0.37
2	-1.17
3	-0.35
4	-0.57
5	0.02

OLS Regression Results

Dep. Variable:	y	R-squared (uncentered):	0.808
Model:	OLS	Adj. R-squared (uncentered):	0.807
Method:	Least Squares	F-statistic:	1394.
Date:	Thu, 07 May 2020	Prob (F-statistic):	0.00
Time:	01:02:30	Log-Likelihood:	-7114.6
No. Observations:	1994	AIC:	1.424e+04
Df Residuals:	1988	BIC:	1.427e+04
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
x1	17.1325	0.559	30.642	0.000	16.036	18.229
x2	0.1062	0.041	2.577	0.010	0.025	0.187
x3	1.3661	0.146	9.353	0.000	1.080	1.653
x4	0.4153	0.044	9.532	0.000	0.330	0.501
x5	1.2833	0.151	8.522	0.000	0.988	1.579
x6	0.1891	0.012	15.343	0.000	0.165	0.213

Omnibus:	108.650	Durbin-Watson:	1.861
Prob(Omnibus):	0.000	Jarque-Bera (JB):	112.413
Skew:	-0.545	Prob(JB):	3.89e-25
Kurtosis:	2.597	Cond. No.	61.7

Figure 4.30: Summary Results from the Regression performed

```
intercept: 22.92316535067743
```

Inference

There are two metrics for the analysis of our model, namely Root Mean Square Error and R-squared. For choosing the model which gives the best algorithm for the prediction of hardness, we take the model with the highest R-squared and lowest Root Mean Square Error.

	Root Mean Square Error	R-squared
Multiple Linear Regression	149.50069	0.71
Multiple Linear Regression without Mo	149.51277	0.709
Multiple Linear Regression without Mo and with log transformation	0.18139	0.865
Multiple Linear Regression without Mo and with Square root transformation	3.84325	0.808

Table 4.3: Comparative study of different techniques used

As is evident from the above table, Multiple Linear regression with log transformation is the best model that has been obtained so far as it has the least value of Root Mean Square Error and the highest value of R-squared.

Linear Equation to predict hardness:

$$\boxed{\text{Log(Hardness)} : y = 2.73226 + 0.10C - 0.02Cr - 0.05Mn - 0.02Ni - 0.02Si}$$

For any new values of C, Cr, Mn, Ni, and Si the above algorithm predicts log(hardness) with a root mean square error of 0.18 and an accuracy of 86.5%.

Chapter 5: RESULTS AND DISCUSSION

As mentioned in the previous section, the second layer of verification involved comparison between the theoretical hardness values predicted using the process parameters as mentioned in Appendix B provided by Sansera Engineering and the experimental data of Vicker's hardness obtained from endo-gas case hardening of 2 steel pages made of SCM 420 grade steel. For, each batch code, 4-5 trial values were provided and the case depth was calculated using the hardness profile against the distance from the surface.

6.1 Verification of the one-dimensional model

6.1.1 Part no. YMCR1007

The endo-gas process was divided into three stages, i.e. the boost stage, diffusion stage, and the equalizing stage and the input parameters for the respective stages are:

Boost stage: Carburizing temperature, T1= 940°C,

Carburizing time, t1 = 450min

Furnace atmosphere carbon potential, cp or φ1 = 1.1%

Diffusion stage: Carburizing temperature, T2 = 940°C,

Carburizing time, t2 = 240min

Furnace atmosphere carbon potential, cp or φ2 = 0.75%

Equalizing stage: Carburizing temperature, T2 = 850°C,

Carburizing time, t2 = 30min

Furnace atmosphere carbon potential, cp or φ2 = 0.6%

The surface carbon content, cs = co = 0.2%. The code for 1D, Appendix A was run and the variation of carbon percentage along the surface was obtained and is as shown below.

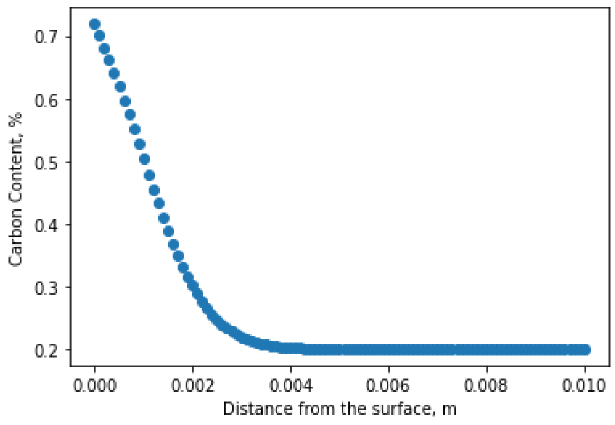


Figure 5.1: Carbon profile for YMCR1007 part number using 1D FEM model

Using the carbon profile and the empirical hardness equations obtained for SCM 420 steel, figure 5.1 the hardness profile for different cooling rates was obtained as shown in figure 6.2. The case depth obtained at a given cut-off of 550Hv was 1.2mm, 1.3mm, 1.4mm, 1.5mm, and for the cut-off of 700Hv were 0.6mm,0.7mm,0.8mm,0.9mm for cooling rates of 7.5, 10, 12.5, 15K/s respectively.

The error percentage between the theoretical and the experimental hardness values were calculated for different experimental samples and at different cooling rates. The error percentages are represented in table 5.1(a) and 5.1(b) at hardness cut-off of 550Hv and 700Hv respectively. It was observed that for the cooling rate of 12.5K/s the experimental and theoretical hardness values had the least variation and smallest error percentages for both cutoffs, and the plots were seen to be coinciding as can be observed in figure 5.3.

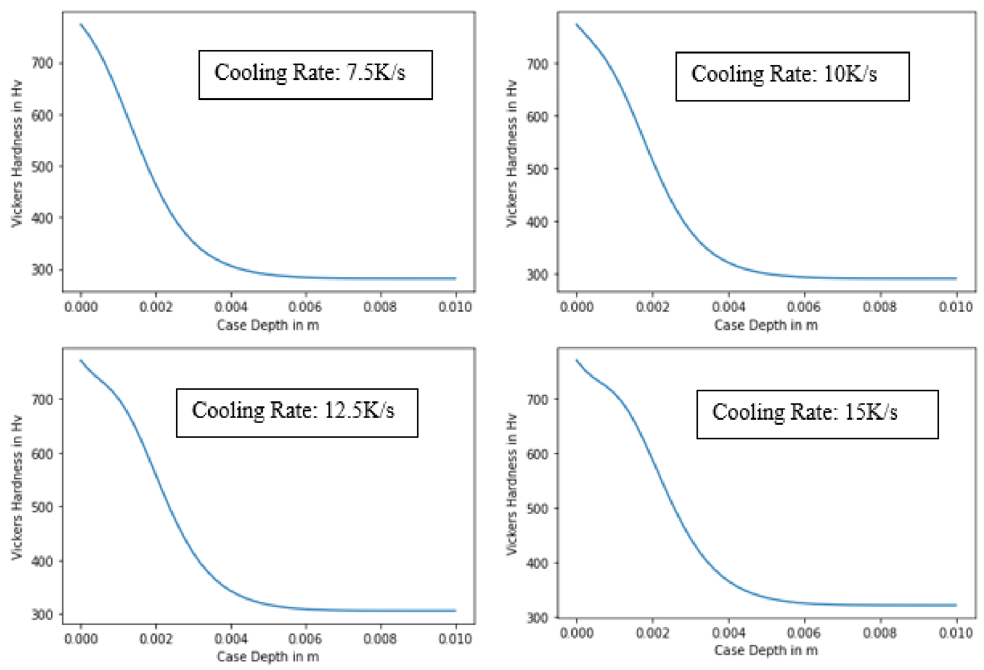


Figure 5.2: Hardness profiles of YCMR 1007-part number at different cooling rates

Cut-off of 550Hv		Case depth (mm)	Sample 1	Sample 2	Sample 3	Sample 4
			1.39	1.4	1.44	1.37
Cooling rate	7.5K/s	1.2	13.67	14.29	16.67	12.41
	10K/s	1.3	6.47	7.14	9.72	5.11
	12.5K/s	1.4	0.72	0.00	2.78	2.19
	15K/s	1.5	7.91	7.14	4.00	9.49

Table 5.1(a): Case depth error percentage of part no. YCMR1007 at a cutoff of 550Hv

Cut-off of 700Hv		Case-depth (mm)	Sample 1	Sample 2	Sample 3	Sample 4
			0.78	0.74	0.77	0.78
Cooling rate	7.5K/s	0.6	23.08	18.92	22.08	23.08
	10K/s	0.7	10.26	5.41	9.09	10.26
	12.5K/s	0.8	2.56	8.11	3.90	2.56
	15K/s	0.9	15.38	21.62	14.44	15.38

Table 5.1(b): Case depth error percentage of part no. YCMR1007 at a cutoff of 700Hv

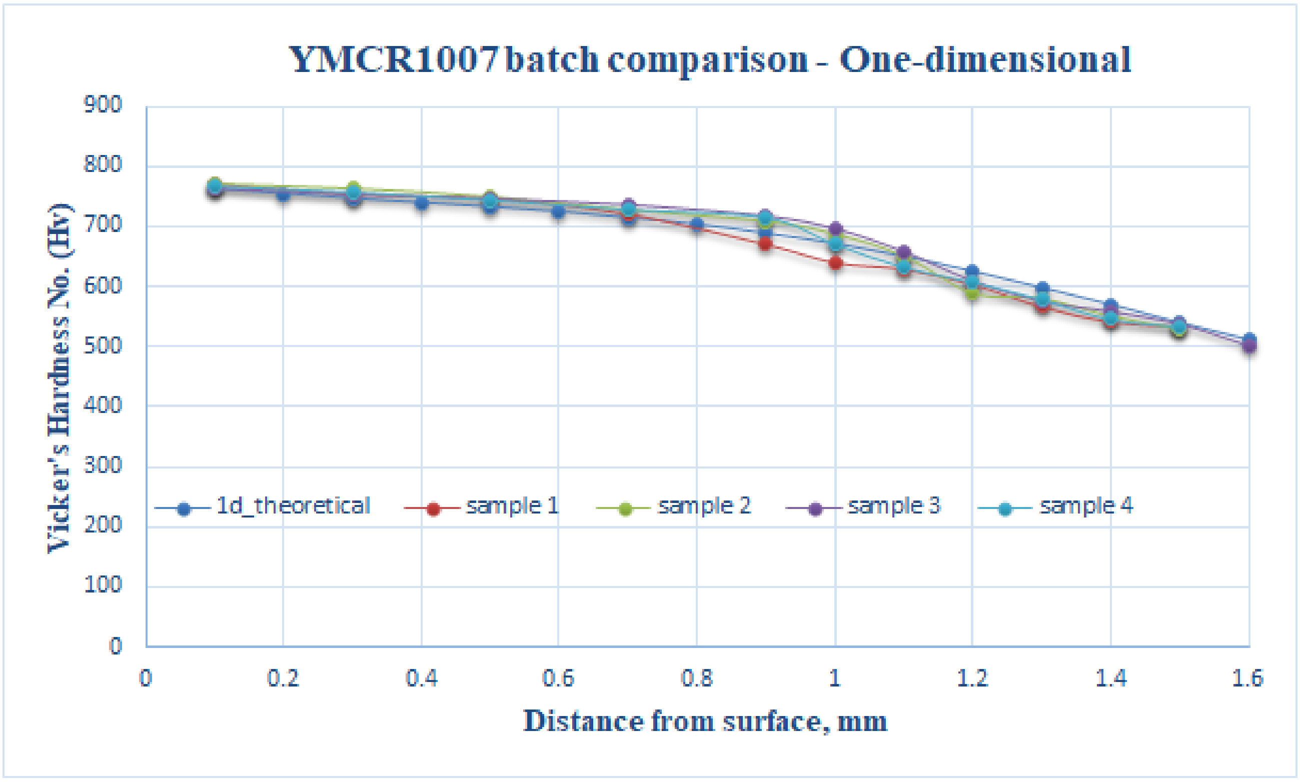


Figure 5.3: Theoretical and experimental hardness plot for part no. YMCR1007

5.1.2 Part no. BACR1070

The endo-gas process is divided into three stages, i.e. the boost stage, diffusion stage, and the equalizing stage as seen in the previous case and the input parameters for the respective stages are:

Boost stage: Carburizing temperature, T1= 940°C,

Carburizing time, t1 = 330min

Furnace atmosphere carbon potential, cp or φ1 = 1.1%

Diffusion stage: Carburizing temperature, T2 = 940°C,

Carburizing time, t2 = 165min

Furnace atmosphere carbon potential, cp or φ2 = 0.75%

Equalizing stage: Carburizing temperature, T2 = 850°C,

Carburizing time, t2 = 30min

Furnace atmosphere carbon potential, cp or φ2 = 0.75%

The surface carbon content, cs = co = 0.2%. The code for 1D, Appendix A was run and the variation of carbon percentage along the surface was obtained and is as shown in figure 5.4. The theoretical hardness profile at a cooling rate of 12.5K/s and the experimental hardness values for 5 samples are as shown in figure 5.5.

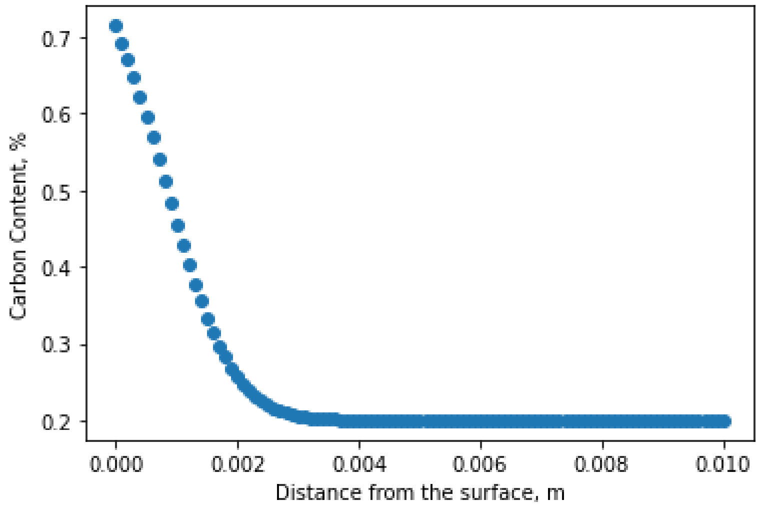


Figure 5.4: Carbon profile for BACR1070 part number using 1D FEM model

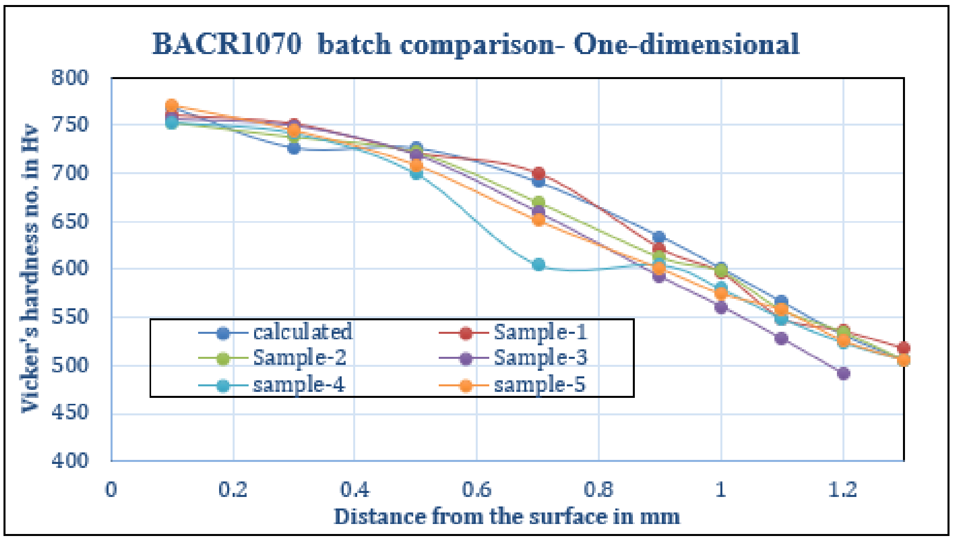


Figure 5.5: Theoretical and experimental hardness plot for part no. BACR1070

5.2 Verification of two-dimensional and three-dimensional models

The secondary verification of two- and three-dimensional geometry follows suit the process adopted for verification of the one-dimensional model. The input process parameters were taken from the HT instruction sheet provided by Sansera Engineering. For both part nos. BACR1070 and YMCR1007, it was observed that for the 2d and 3d geometries modeled in the previous section, the carbon profile obtained again the distance from the surface coincided with each other, as can be observed in figure 5.6 for part no. BACR1070.

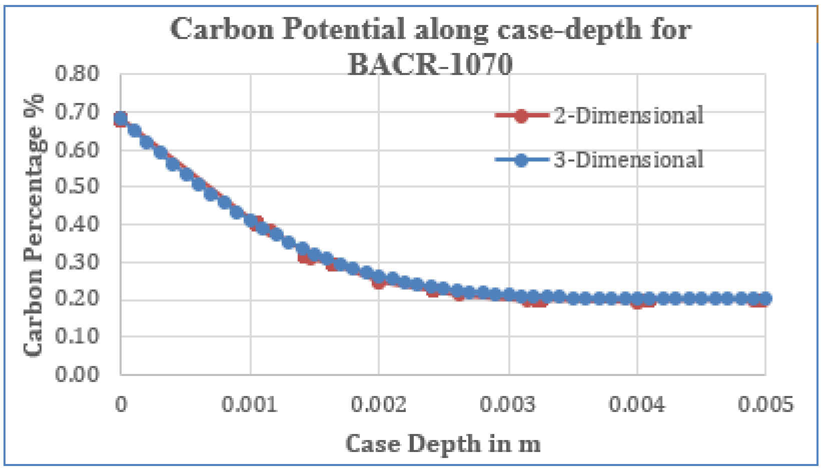


Figure 5.6: Carbon profile of BACR1070 using 2D and 3D model

The corresponding plots of experimental and theoretical hardness using empirical relation evaluated for cooling rate of 12.5K/s are as shown in Figure 5.7.

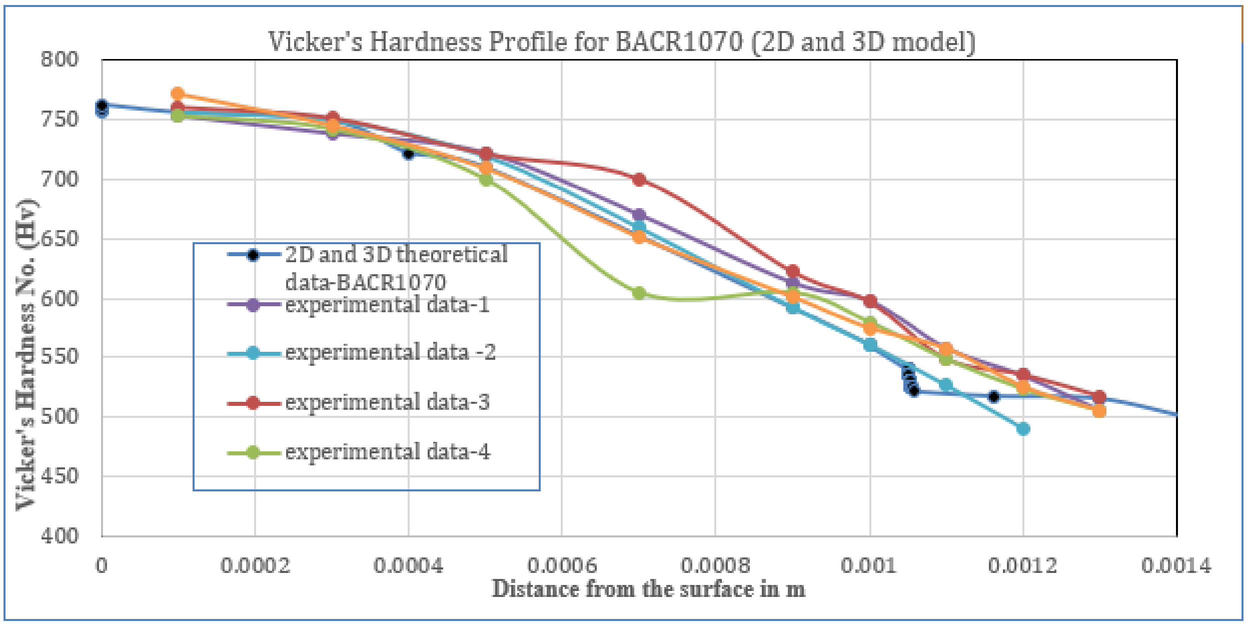


Figure 5.7: Vicker's Hardness Profile for BACR1070 (2D and 3D modelling)

The case depth obtained at a cutoff of 550Hv for BACR1070 part no was 1.102mm which was well within the specified case depth of 1.0-1.2mm. The error percentage corresponding to 3 experimental samples is shown in table 5.2.

CASE DEPTH (Part No: BACR)	*Sample 1*	*Sample 2*	*Sample 3*
Theoretical	1.102	1.102	1.102
Experimental	1.1	1.13	1.14
Error	0.002	0.028	0.038
Error %	1.81	2.54	3.45

Table 5.2: Variation of 2,3D theoretical values with experimental data for part no. BACR1070

Similarly, for part no. YMCR1007 the ECD (effective case-depth) obtained at a cutoff Vicker's hardness no. 550Hv was 1.358mm and the case depth, MCD obtained at a cutoff of 700Hv is 0.758. The corresponding error percentages between theoretical values and experimental data for three samples are as shown in table 5.3 and 5.4.

CASE DEPTH (Part No: YMCR (550HV))	*Sample 1*	*Sample 2*	*Sample 3*
Theoretical	1.358	1.358	1.358
Experimental	1.37	1.39	1.4
Error	0.012	0.032	0.042
Error %	0.88	2.36	3.09

Table 5.3: ECD variation of part no. YMCR1070 (2D and 3D model)

CASE DEPTH (Part No: YMCR (700HV))	*Sample 1*	*Sample 2*	*Sample 3*
Theoretical	0.758	0.758	0.758
Experimental	0.78	0.77	0.74
Error	0.022	0.012	0.018
Error %	2.9	1.58	2.37

Table 5.4: MCD variation of part no. YMCR1070 (2D and 3D model)

The case depth (ECD and MCD) obtained in all 9 cases, i.e., 1D, 2D, 3D modelled values for two-part nos. lay between the ranges specified in the HT instruction sheets and agreed with the experimental data provided by Sansera Engineering for endo-gas case carburization.

Chapter 6: CONCLUSION AND SCOPE FOR FUTURE WORK

Finite element numerical diffusion models are developed using Fick’s diffusion laws for one, two, three-dimensional geometries. The models predict the carbon profile, hardness, and in turn case depth of the specimen under various input parameters. The Python model developed can be used for evaluation of the carbon and hardness profiles for various complex geometrical models up to three-dimensional spaces including but not limited to gears, connecting rods, and various other system parts that undergo continuous wear cycle.

6.1 Conclusion:

The following conclusions are drawn whilst developing 1D, 2D, and 3D model in Python for the respective geometries and finding the best algorithm to predict hardness of the carburized surface:

1. The theoretical predicted results obtained are in agreement with the experimental data provided by Sansera Engineering for endo-gas case carburization. The diffusion model is developed and tested for endo-gas carburization.
2. The predicted output provides a quantitative understanding of how various parameters such as coefficient of mass transfer from atmosphere to the part surface, diffusivity coefficient, the carburizing temperature affect the hardness of a surface and in turn its case depth after a heat treatment process is carried out.
3. The advantage of boost stages in reducing the carburizing time and increasing performance efficiency is discussed.
4. Hardness calculation after quenching can be evaluated formulating empirical relations with the steel composition and cooling rates as input variables.
5. An algorithm for the prediction of hardness is developed using multiple linear aggression followed by log transformation.

6.2 Scope of future work

Moving ahead, the scope of this project for future development includes:

1. Improvements in the hardness prediction model by analyzing the effects of various additional parameters such as cooling rates, quenching speed, microstructure changes, phase transformation after carburizing and quenching, expanding the model to various general steels which can be achieved by deploying austenite decomposition models based on Johnson Mehl Avrami equations and integrating it with linear regression techniques. The advantage with linear regression is that it can be performed in a limited set of parameters

but to improve the accuracy of the target (parameter to be predicted), we need to perform certain advanced techniques such as random forest, decision trees, support vector machines, etc. which needs a huge dataset.

2. With the integration of deep learning techniques, such as artificial neural networks, optimum selection of input parameters such as furnace temperature, carbon percentage of furnace atmosphere for the boost, diffusion stages for different carburizing processes can be determined to obtain the required case depth for various applications in aerospace, automobile industries. Python libraries such as Keras and TensorFlow provide various functionalities to perform such complex tasks and give us better results.

REFERENCES

[1] Maisuradze, M.V. and Kuklina, A.A., 2018. Numerical solution of the differential diffusion equation for a steel carburizing process. In *Solid State Phenomena* (Vol. 284, pp. 1230-1234). Trans Tech Publications Ltd.

[2] Cavaliere, P., Zavarise, G. and Perillo, M., 2009. Modeling of the carburizing and nitriding processes. *Computational Materials Science*, *46*(1), pp.26-35.

[3] Lee, S.J., Matlock, D.K. and Van Tyne, C.J., 2013. Comparison of two finite element simulation codes used to model the carburizing of steel. *Computational Materials Science*, *68*, pp.47-54.

[4] Eck, S., Ishmurzin, A., Wlanis, T., Ebner, R., Planitzer, F. and Hatzenbichler, T., 2012. A finite element model for carburisation of surface densified PM components. *International Journal of Computational Materials Science and Surface Engineering*, *5*(1), pp.16-30.

[5] Zajusz, M., Tkacz-Śmiech, K. and Danielewski, M., 2014. Modeling of vacuum pulse carburizing of steel. *Surface and Coatings Technology*, *258*, pp.646-651.

[6] Deshpande, P.D., Gautham, B.P., Gupta, U. and Khan, D., 2014, December. Modeling the steel case carburizing quenching process using statistical and machine learning techniques. In *2014 9th International Conference on Industrial and Information Systems (ICIIS)* (pp. 1-6). IEEE.

[7] Fundamentals of the Finite Element Method for Heat and Mass Transfer Second Edition, Wiley, 2016, P. Nithiarasu, R. W. Lewis,K. N. Seetharamu

APPENDIX A

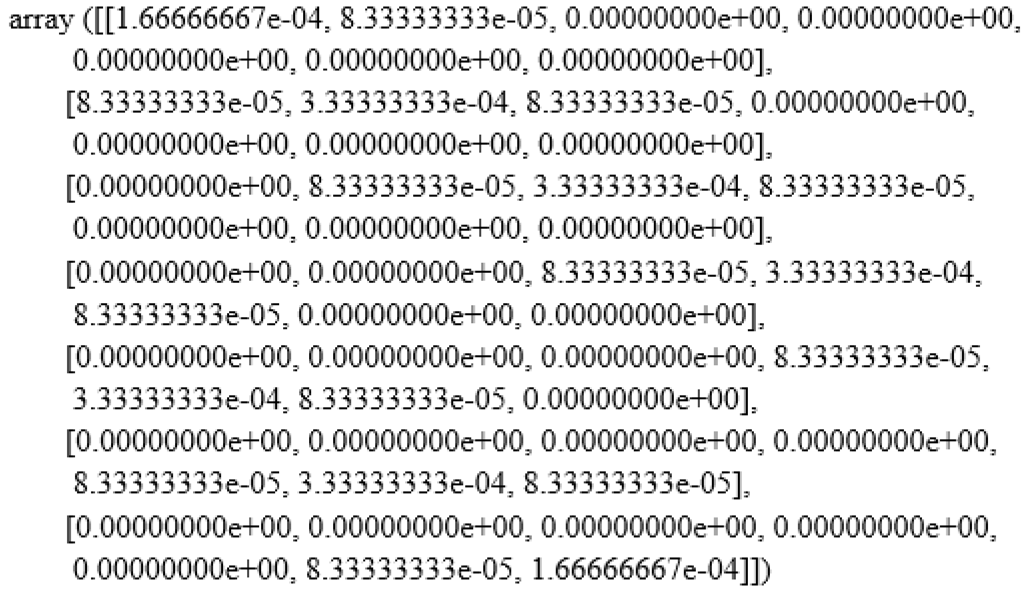
The following codes were written on Python IDE(Integrated Development Environment), which in our case is JupyterNotebook.
Please note that '#' denotes comments in Python and is used at multiple stages to add useful information for the user or to prevent printing output.

A.1 Code development for One-Dimensional Geometry:

```
#Importing python libraries
import numpy as np
import math
import matplotlib.pyplot as plt
import plotly. graph_objects as go

#Input Parameters
length = 0.003                          #length of the rod
Do = 0.11 * math.pow(10,-4)
beta = 2 * (math.pow(10,-7))            #Coefficient of carbon mass transfer, m/s
psi = 0.75                              #Carbon potential of surface atmosphere, %
Q = 132000                              #Activation energy
R = 8.31                                #Universal gas constant
T = 940+273                             #Furnace temperature in Kelvin
D = Do * math.exp(-Q/(R*T))             #Arrhenius eqn (carbon diffusion coefficient)
N = 7                                   #No. of nodes
E = N-1                                 #No. of elements
#Considering a 1D linear system with 7(N) nodes, 6(N-1) elements
l = length/E                            #length of each element

#Evaluation of force vector, {F}
F = np.matrix(np.zeros((N,1)))
F[0] = beta * psi
F
Output:
matrix([[1.5e-07],
        [0.0e+00],
        [0.0e+00],
        [0.0e+00],
        [0.0e+00],
        [0.0e+00],
        [0.0e+00]])


#Evaluation of capacitance matrix, [C]
C = np.zeros((N,N))
C_1 = np.matrix([[2*(l/6),1*(l/6)],[1*(l/6),2*(l/6)]])
n1 = 0
n2 = 2
```

```
for i in range(0,N-1):
    C[n1:n2,n1:n2] += C_1
    n1 += 1
    n2 += 1
C
```

Output:

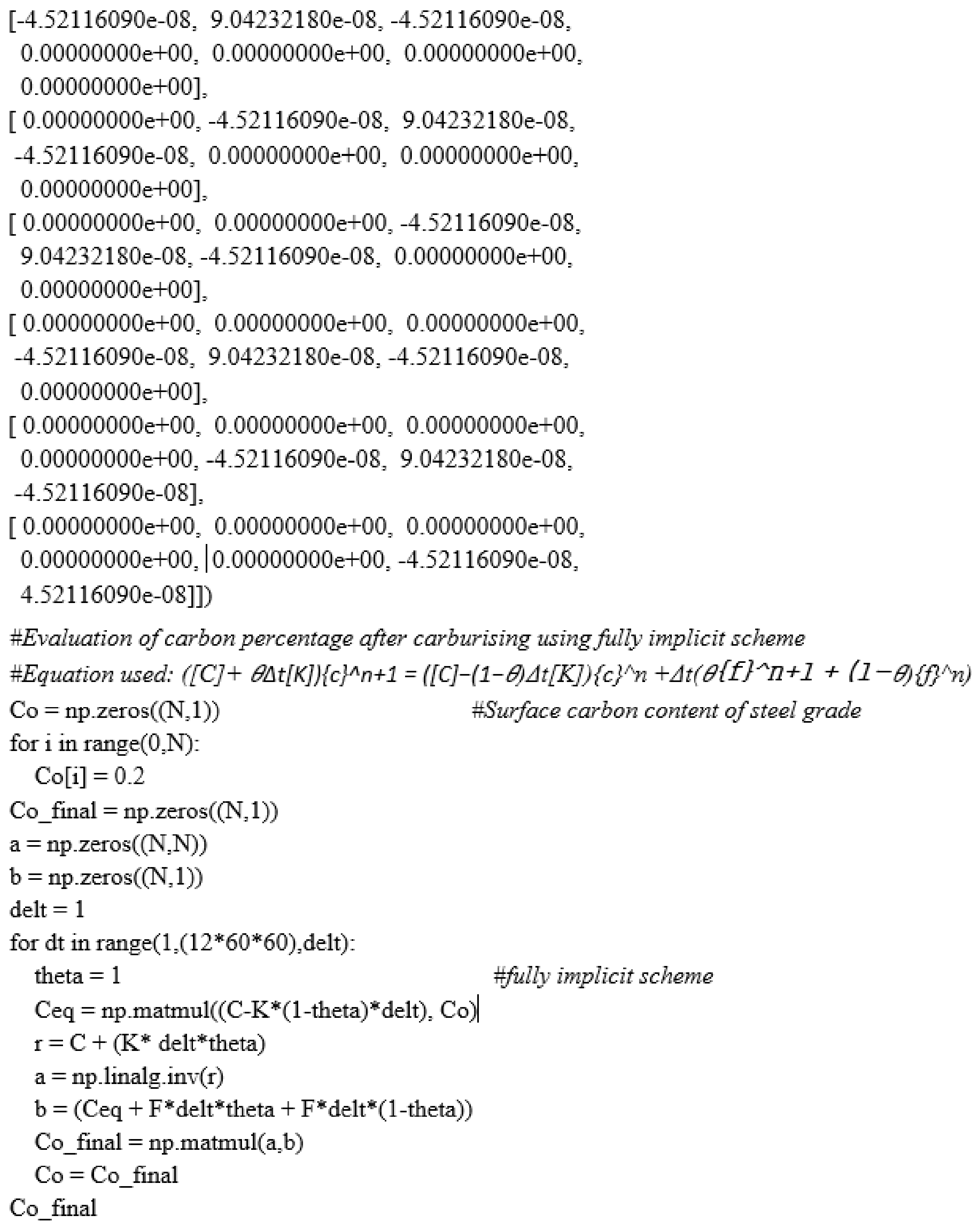


```
array ([[1.66666667e-04, 8.33333333e-05, 0.00000000e+00, 0.00000000e+00,
         0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
        [8.33333333e-05, 3.33333333e-04, 8.33333333e-05, 0.00000000e+00,
         0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
        [0.00000000e+00, 8.33333333e-05, 3.33333333e-04, 8.33333333e-05,
         0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
        [0.00000000e+00, 0.00000000e+00, 8.33333333e-05, 3.33333333e-04,
         8.33333333e-05, 0.00000000e+00, 0.00000000e+00],
        [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 8.33333333e-05,
         3.33333333e-04, 8.33333333e-05, 0.00000000e+00],
        [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
         8.33333333e-05, 3.33333333e-04, 8.33333333e-05],
        [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
         0.00000000e+00, 8.33333333e-05, 1.66666667e-04]])
```

```
#Evaluation of stiffness matrix [K]
K = np.zeros((N,N))
KD = np.matrix([[D/l,-D/l],[-D/l,D/l]])
a1 = 0
a2 = 2
for i in range(0,N-1):
    K[a1:a2,a1:a2] += KD
    a1 += 1
    a2 += 1
KM = np.matrix([[beta,0],[0,0]])
K[:2,:2]+=KM
K
```

Output:

```
array([[ 2.45211609e-07, -4.52116090e-08,  0.00000000e+00,
         0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
         0.00000000e+00],
```



```
[-4.52116090e-08,  9.04232180e-08, -4.52116090e-08,
  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
  0.00000000e+00],
[ 0.00000000e+00, -4.52116090e-08,  9.04232180e-08,
 -4.52116090e-08,  0.00000000e+00,  0.00000000e+00,
  0.00000000e+00],
[ 0.00000000e+00,  0.00000000e+00, -4.52116090e-08,
  9.04232180e-08, -4.52116090e-08,  0.00000000e+00,
  0.00000000e+00],
[ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
 -4.52116090e-08,  9.04232180e-08, -4.52116090e-08,
  0.00000000e+00],
[ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
  0.00000000e+00, -4.52116090e-08,  9.04232180e-08,
 -4.52116090e-08],
[ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
  0.00000000e+00,  0.00000000e+00, -4.52116090e-08,
  4.52116090e-08]])
```

```
#Evaluation of carbon percentage after carburising using fully implicit scheme
#Equation used: ([C]+ θΔt[K]){c}^n+1 = ([C]−(1−θ)Δt[K]){c}^n +Δt(θ{f}^n+1 + (1−θ){f}^n)
Co = np.zeros((N,1))                          #Surface carbon content of steel grade
for i in range(0,N):
   Co[i] = 0.2
Co_final = np.zeros((N,1))
a = np.zeros((N,N))
b = np.zeros((N,1))
delt = 1
for dt in range(1,(12*60*60),delt):
   theta = 1                                  #fully implicit scheme
   Ceq = np.matmul((C-K*(1-theta)*delt), Co)
   r = C + (K* delt*theta)
   a = np.linalg.inv(r)
   b = (Ceq + F*delt*theta + F*delt*(1-theta))
   Co_final = np.matmul(a,b)
   Co = Co_final
Co_final
```

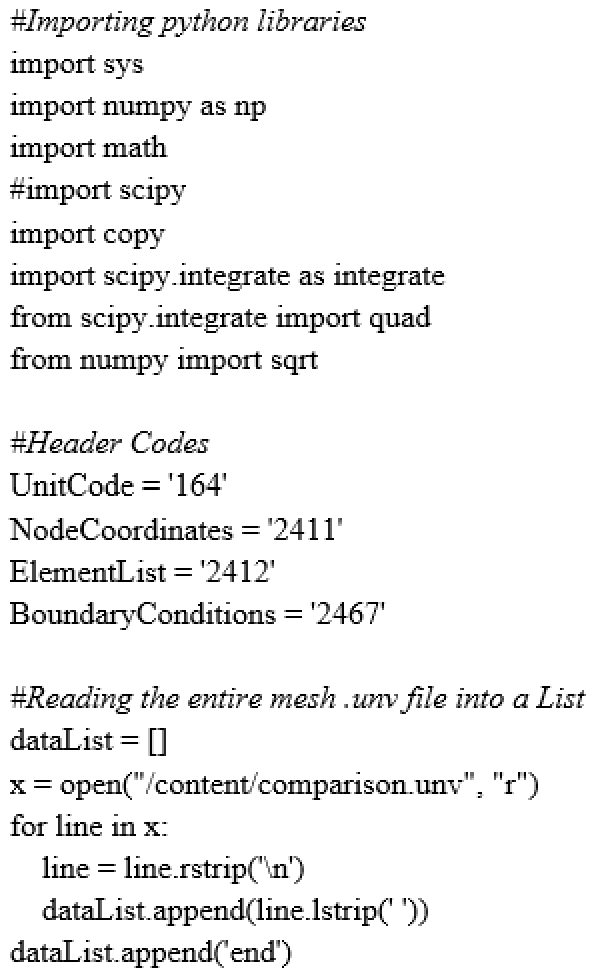
Output:

```
matrix([[0.71491497],
        [0.56483505],
        [0.43554523],
        [0.33728185],
        [0.27219501],
        [0.23659863],
        [0.22547307]])
```

#Plotting the final carbon percentage wrt the distance from the surface in mm.

```
%matplotlib inline
x = np.zeros((N,1))
for j in range(N):
  x[j] = 0 + j*l
fig = plt.figure()
plt.xlabel('Distance from the surface, m')
plt.ylabel('Carbon Content, %')
plt.ylim(0.1,0.8,0.1)
plt.plot(x,Co_final)
```

A.2 Code development for Two-Dimensional Geometry



```
#Importing python libraries
import sys
import numpy as np
import math
#import scipy
import copy
import scipy.integrate as integrate
from scipy.integrate import quad
from numpy import sqrt

#Header Codes
UnitCode = '164'
NodeCoordinates = '2411'
ElementList = '2412'
BoundaryConditions = '2467'

#Reading the entire mesh .unv file into a List
dataList = []
x = open("/content/comparison.unv", "r")
for line in x:
    line = line.rstrip('\n')
    dataList.append(line.lstrip(' '))
dataList.append('end')
```

```
x.close()
#print("Data of mesh.unv = ", dataList)
```

```
#Boundary condition
diffusion_index_start = dataList.index('diffusion')
diffusion_index_end = len(dataList) - 2
diffusion_data = dataList[diffusion_index_start + 1: diffusion_index_end]
second_column = list(map(lambda x: x[1], map(lambda x: x.split(), diffusion_data)))
fifth_column = list(map(lambda x: x[5] if len(x) > 5 else None, map(lambda x: x.split(),
diffusion_data)))
diffusion_data = second_column + fifth_column
diffusion_data.remove(None)
diffusion_data_final = list(map(int,diffusion_data))
print("Final Diffusion data = ",diffusion_data_final)

#Input parameters
Q = 132000                                    #Activation energy
R = 8.31                                      #Universal gas constant
T = 940+273                                   #Furnace temperature in Kelvin
Do = 0.11 * math.pow(10, -4)
psi = 0.75
CarbonDiffusionCoefficient_D = Do * math.exp(-Q / (R * T))   #Arrhenius-type equation
beta = (math.pow(10, -7))   #Coefficient of carbon mass transfer from the furnace atmosphere

#Function call
def check(text,linenumber):
   found = False
   if text in dataList[linenumber]:
      found = True
   return found

def findBlockStart(dataList,HeaderCode):
   linenumber=0
   for linenumber in range(len(dataList)):
      if check('-1',linenumber)== True:
         if check(HeaderCode,linenumber+1)==True:
            return(linenumber+2)

#Code block to create NODE DICTIONARY
NodeDictionary  = {}
startLine = findBlockStart(dataList,NodeCoordinates)

line = copy.deepcopy(startLine)
coordinateLine = line+1

while dataList[line]!=('-1'):
   NodeDictionary.update({int((dataList[line].split())[0]):[float(x) for x in
dataList[coordinateLine].split()]})
   line+=2
   coordinateLine+=2
```

```
#print(NodeDictionary)

#Code block to create Element Dictionary
EdgeElementDictionary = {}
ElementDictionary = {}
elementStart = findBlockStart(dataList,ElementList)
line = copy.deepcopy(elementStart)
i=1
while dataList[line]!=('-1'):
   if [int(x) for x in dataList[line].split()][5] == 2:
      EdgeElementDictionary.update({int((dataList[line].split())[0]):[int(x) for x in
dataList[line+2].split()]})
      line+=3
   else:
      ElementDictionary.update({i:[int(x) for x in dataList[line+1].split()]})
      line+=2
      i+=1
#print(EdgeElementDictionary)
#print(ElementDictionary)
#Type of Analysis
analysis = '2D'

#C, K and F Matrix with assembly
C = np.zeros((len(NodeDictionary),len(NodeDictionary)))
K = np.zeros((len(NodeDictionary),len(NodeDictionary)))
F = np.zeros((len(NodeDictionary),1))
nele = len(ElementDictionary)

#for loop evaluation for each element
for x in range(nele):
   #element area
   elemarea =   abs((np.linalg.det(np.array([[1,NodeDictionary[ElementDictionary[x+1][0]][0],
                    NodeDictionary[ElementDictionary[x+1][0]][1]],
                   [1,NodeDictionary[ElementDictionary[x+1][1]][0],
NodeDictionary[ElementDictionary[x+1][1]][1]],
                         [1,NodeDictionary[ElementDictionary[x+1][2]][0],
                          NodeDictionary[ElementDictionary[x+1][2]][1]]])))/2)

  #shape function constants a, b and c
    ai =
NodeDictionary[ElementDictionary[x+1][1]][0]*NodeDictionary[ElementDictionary[x+1][2]][1]
-
NodeDictionary[ElementDictionary[x+1][2]][0]*NodeDictionary[ElementDictionary[x+1][1]][1]
    bi = NodeDictionary[ElementDictionary[x+1][1]][1] -
NodeDictionary[ElementDictionary[x+1][2]][1]
    ci = NodeDictionary[ElementDictionary[x+1][2]][0] -
NodeDictionary[ElementDictionary[x+1][1]][0]
    aj =
NodeDictionary[ElementDictionary[x+1][2]][0]*NodeDictionary[ElementDictionary[x+1][0]][1]
-
NodeDictionary[ElementDictionary[x+1][0]][0]*NodeDictionary[ElementDictionary[x+1][2]][1]
    bj = NodeDictionary[ElementDictionary[x+1][2]][1]-
NodeDictionary[ElementDictionary[x+1][0]][1]
    cj = NodeDictionary[ElementDictionary[x+1][0]][0] -
NodeDictionary[ElementDictionary[x+1][2]][0]
    ak =
NodeDictionary[ElementDictionary[x+1][0]][0]*NodeDictionary[ElementDictionary[x+1][1]][1]
-
NodeDictionary[ElementDictionary[x+1][1]][0]*NodeDictionary[ElementDictionary[x+1][0]][1]
    bk = NodeDictionary[ElementDictionary[x+1][0]][1] -
NodeDictionary[ElementDictionary[x+1][1]][1]
    ck = NodeDictionary[ElementDictionary[x+1][1]][0] -
NodeDictionary[ElementDictionary[x+1][0]][0]

#B, D and BTranspose Matrix and multiplication
B = (1/(2*elemarea))*np.matrix([[bi,bj,bk], [ci,cj,ck]])
D = np.array([[CarbonDiffusionCoefficient_D,0],[0,CarbonDiffusionCoefficient_D]])
BTranspose = np.array(B).transpose()
Bmult = np.matmul(np.matmul(BTranspose,D),B)

# Calculating the first half of the K matrix
    K1 = np.zeros(Bmult.shape)
    for i in range(list(Bmult.shape)[0]):
        for j in range(list(Bmult.shape)[1]):
            K1[i,j]= list(quad(lambda x:(Bmult.item(i,j)),0,elemarea))[0]
```

```
 # calculating the second half of the K matrix
 K2 = np.zeros(Bmult.shape)
 lij = 0
 for i in range(list(Bmult.shape)[0]):
     for j in range(list(Bmult.shape)[1]):
         if ElementDictionary[x+1][i] in diffusion_data_final:
             if ElementDictionary[x+1][j] in diffusion_data_final:
                 if i==j:
                     K2[i,j]= 2
                 else:
                     K2[i,j]= 1
                     #Boundary length of the element
                     lij = np.sqrt((NodeDictionary[ElementDictionary[x+1][i]][0]-
                             NodeDictionary[ElementDictionary[x+1][j]][0])**2+
                             (NodeDictionary[ElementDictionary[x+1][i]][1]-
                             NodeDictionary[ElementDictionary[x+1][j]][1])**2)
 Ke = K1+K2*beta*lij/6

 #K matrix assembly
 for i in range(list(Ke.shape)[0]): # calculating the first half of the integral
     for j in range(list(Ke.shape)[1]):
         K[int(ElementDictionary[(x+1)][i])-1,int(ElementDictionary[(x+1)][j])-1]+=Ke[i,j]

#C matrix
ctemp = np.zeros((3,3))
for i in range(3):
   for j in range(3):
      if i==j:
            ctemp[i,j]= 2*elemarea/12
      else:
            ctemp[i,j]= 1*elemarea/12

#C matrix assembly
for l in range(3):
   for m in range(3):
      C[int(ElementDictionary[(x+1)][l])-1,int(ElementDictionary[(x+1)][m])-1]+=ctemp[l,m]
#print(C)

#Load vector F
Fe = np.zeros((3,1))
p1 = np.zeros((3,1))
F2p1 = [0,0]
```

```
      Lij = 0
       m=0
       for i in range(3):
          if ElementDictionary[x+1][i] in diffusion_data_final:
             F2p1[m]=ElementDictionary[x+1][i]
             p1[i][0]+=1
             m+=1
       if (F2p1[0]!=0)&(F2p1[1]!=0):
          Lij = np.sqrt((NodeDictionary[F2p1[0]][0]-
                                NodeDictionary[F2p1[1]][0])**2+
                                (NodeDictionary[F2p1[0]][1]-
                                NodeDictionary[F2p1[1]][1])**2)
          Fe=(beta*psi*Lij/2)*p1

       #Assembly of F Matrix
       for i in range(list(Fe.shape)[0]):
          F[int(ElementDictionary[(x+1)][i])-1,0]+=Fe[i,0]
    #print(F)
    #print(K)

#Evaluation of carbon potential
Co = np.zeros((len(NodeDictionary),1))
for i in range(len(NodeDictionary)):
   Co[i,0] += 0.2
N = len(NodeDictionary)
a = np.zeros((N,N))
b = np.zeros((N,1))
delt = 3600
for dt in range(1,(12*60*60),delt):
   theta = 1
   Ceq = np.matmul((C-K*(1-theta)*delt), Co)
   r = C + (K* delt*theta)
   a = np.linalg.inv(r)
   b = (Ceq + F*delt*theta + F*delt*(1-theta))
   Co_final = np.matmul(a,b)
   Co = Co_final
import matplotlib.pyplot as plt
x = np.zeros((len(NodeDictionary),1))
for i in range(len(NodeDictionary)):
   x[i] = NodeDictionary[i+1][0]
plt.plot(x,Co_final,'o')
plt.xlabel('Case Depth in m')
plt.ylabel('Carbon Percentage %')
plt.show()
```

A.3 Code development for Three-Dimensional Geometry

```
#Importing python libraries
import sys
import numpy as np
import math
import scipy
import copy
import scipy.integrate as integrate
from scipy.integrate import quad
import collections
from numpy import sqrt

#Header Codes
UnitCode = '164'
NodeCoordinates = '2411'
ElementList = '2412'
BoundaryConditions = '2467'

#Reading the entire mesh .unv file into a List
dataList = []
x = open("/content/mesh.unv", "r")
for line in x:
   line = line.rstrip('\n')
   dataList.append(line.lstrip(' '))
dataList.append('end')
x.close()

#Boundary condition
diffusion_index_start = dataList.index('diffusion')
print(len(dataList))
diffusion_index_end = len(dataList) - 2
diffusion_data = dataList[diffusion_index_start + 1: diffusion_index_end]
second_column = list(map(lambda x: x[1], map(lambda x: x.split(), diffusion_data)))
fifth_column = list(map(lambda x: x[5] if len(x) > 5 else None, map(lambda x: x.split(),
diffusion_data)))
diffusion_data = second_column + fifth_column
diffusion_data.remove(None)
diffusion_data_final = list(map(int, diffusion_data))
```

```
#Assigning input parameters
Q = 132000           #Activation energy \n",
R = 8.31             #Universal gas constant\n",
T = 940 + 273        #Furnance temperature in Kelvin\n",
Do = 0.11 * math.pow(10, -4)
psi = 0.75
CarbonDiffusionCoefficient_D = Do * math.exp(-Q / (R * T))  #Arrhenius-type equation - carbon
diffusion coefficient
beta = 1*(math.pow(10, -5))  #Coefficient of carbon mass transfer from the furnace atmosphere to
the steel part surface\n"

#Function call
def check(text, linenumber):
   found = False
   if text in dataList[linenumber]:
      found = True
   return found

def findBlockStart(dataList, HeaderCode):
   linenumber = 0
   for linenumber in range(len(dataList)):
      if check('-1', linenumber) == True:
         if check(HeaderCode, linenumber + 1) == True:
            return (linenumber + 2)

#Code block to create NODE DICTIONARY
NodeDictionary  = {}
startLine = findBlockStart(dataList,NodeCoordinates)
line = copy.deepcopy(startLine)
coordinateLine = line+1
while dataList[line]!=('-1'):
   NodeDictionary.update({int((dataList[line].split())[0]):[float(x) for x in
dataList[coordinateLine].split()]})
   line+=2
   coordinateLine+=2

#Code block to create Element Dictionary
EdgeElementDictionary = {}
ElementDictionary = {}
BoundaryElementDictionary = {}
elementStart = findBlockStart(dataList, ElementList)
line = copy.deepcopy(elementStart)
i = 1
j = 1
```

```
while dataList[line] != ('-1'):
   if [int(x) for x in dataList[line].split()][5] == 2:
      EdgeElementDictionary.update({int((dataList[line].split())[0]): [int(x) for x in dataList[line +
2].split()]})
      line += 3
   elif [int(x) for x in dataList[line].split()][5] == 3:
      BoundaryElementDictionary.update({(i+len(EdgeElementDictionary)): [int(x) for x in
dataList[line + 1].split()]})
      line += 2
      i += 1
   else:
      ElementDictionary.update({j: [int(x) for x in dataList[line + 1].split()]})
      line += 2
      j += 1

#Type of Analysis
analysis = '3D'

#C, K and F Matrix with assembly
C = np.zeros((len(NodeDictionary),len(NodeDictionary)))
K = np.zeros((len(NodeDictionary),len(NodeDictionary)))
F = np.zeros((len(NodeDictionary),1))
nele = len(ElementDictionary)
# for loop for each element
for x in range(nele):
   # element volume
   elevolume =
abs((np.linalg.det(np.array([[1, NodeDictionary[ElementDictionary[x + 1][0]][0],
                              NodeDictionary[ElementDictionary[x + 1][0]][1],
                              NodeDictionary[ElementDictionary[x + 1][0]][2]],
                             [1, NodeDictionary[ElementDictionary[x + 1][1]][0],
                              NodeDictionary[ElementDictionary[x + 1][1]][1],
                              NodeDictionary[ElementDictionary[x + 1][1]][2]],
                             [1, NodeDictionary[ElementDictionary[x + 1][2]][0],
                              NodeDictionary[ElementDictionary[x + 1][2]][1],
                              NodeDictionary[ElementDictionary[x + 1][2]][2]],
                             [1, NodeDictionary[ElementDictionary[x + 1][3]][0],
                              NodeDictionary[ElementDictionary[x + 1][3]][1],
                              NodeDictionary[ElementDictionary[x + 1][3]][2]]]))))
```

```
#shape function constants a,b,c and d
 ai = np.linalg.det(([[NodeDictionary[ElementDictionary[x + 1][1]][0],
                   NodeDictionary[ElementDictionary[x + 1][1]][1],
                   NodeDictionary[ElementDictionary[x + 1][1]][2]],
                  [NodeDictionary[ElementDictionary[x + 1][2]][0],
                   NodeDictionary[ElementDictionary[x + 1][2]][1],
                   NodeDictionary[ElementDictionary[x + 1][2]][2]],
                  [NodeDictionary[ElementDictionary[x + 1][3]][0],
                   NodeDictionary[ElementDictionary[x + 1][3]][1],
                   NodeDictionary[ElementDictionary[x + 1][3]][2]]]))
 aj = -1*np.linalg.det(([[NodeDictionary[ElementDictionary[x + 1][0]][0],
                   NodeDictionary[ElementDictionary[x + 1][0]][1],
                   NodeDictionary[ElementDictionary[x + 1][0]][2]],
                  [NodeDictionary[ElementDictionary[x + 1][2]][0],
                   NodeDictionary[ElementDictionary[x + 1][2]][1],
                   NodeDictionary[ElementDictionary[x + 1][2]][2]],
                  [NodeDictionary[ElementDictionary[x + 1][3]][0],
                   NodeDictionary[ElementDictionary[x + 1][3]][1],
                   NodeDictionary[ElementDictionary[x + 1][3]][2]]]))

 ak = np.linalg.det(([[NodeDictionary[ElementDictionary[x + 1][0]][0],
                    NodeDictionary[ElementDictionary[x + 1][0]][1],
                    NodeDictionary[ElementDictionary[x + 1][0]][2]],
                   [NodeDictionary[ElementDictionary[x + 1][1]][0],
                    NodeDictionary[ElementDictionary[x + 1][1]][1],
                    NodeDictionary[ElementDictionary[x + 1][1]][2]],
                   [NodeDictionary[ElementDictionary[x + 1][3]][0],
                    NodeDictionary[ElementDictionary[x + 1][3]][1],
                    NodeDictionary[ElementDictionary[x + 1][3]][2]]]))
   am = -1*np.linalg.det(([[NodeDictionary[ElementDictionary[x + 1][0]][0],
                    NodeDictionary[ElementDictionary[x + 1][0]][1],
                    NodeDictionary[ElementDictionary[x + 1][0]][2]],
                   [NodeDictionary[ElementDictionary[x + 1][1]][0],
                    NodeDictionary[ElementDictionary[x + 1][1]][1],
                    NodeDictionary[ElementDictionary[x + 1][1]][2]],
                   [NodeDictionary[ElementDictionary[x + 1][2]][0],
                    NodeDictionary[ElementDictionary[x + 1][2]][1],
                    NodeDictionary[ElementDictionary[x + 1][2]][2]]]))
   bi = -1*(((NodeDictionary[ElementDictionary[x+1][1]][1]-
NodeDictionary[ElementDictionary[x+1][3]][1])*(NodeDictionary[ElementDictionary[x+1][2]][2
]-NodeDictionary[ElementDictionary[x+1][3]][2]))
-((NodeDictionary[ElementDictionary[x+1][2]][1]-
NodeDictionary[ElementDictionary[x+1][3]][1])*(NodeDictionary[ElementDictionary[x+1][1]][2
]-NodeDictionary[ElementDictionary[x+1][3]][2])))
   bj = -1*(((NodeDictionary[ElementDictionary[x+1][2]][1]-
NodeDictionary[ElementDictionary[x+1][3]][1])*(NodeDictionary[ElementDictionary[x+1][0]][2
]-NodeDictionary[ElementDictionary[x+1][3]][2]))
       -((NodeDictionary[ElementDictionary[x+1][0]][1]-
NodeDictionary[ElementDictionary[x+1][3]][1])*(NodeDictionary[ElementDictionary[x+1][2]][2
]-NodeDictionary[ElementDictionary[x+1][3]][2])))
   bk = -1*(((NodeDictionary[ElementDictionary[x+1][0]][1]-
NodeDictionary[ElementDictionary[x+1][3]][1])*(NodeDictionary[ElementDictionary[x+1][1]][2
]-NodeDictionary[ElementDictionary[x+1][3]][2]))
       -((NodeDictionary[ElementDictionary[x+1][1]][1]-
NodeDictionary[ElementDictionary[x+1][3]][1])*(NodeDictionary[ElementDictionary[x+1][0]][2
]-NodeDictionary[ElementDictionary[x+1][3]][2])))
   bm = -(bi+bj+bk)
   ci = -1*(((NodeDictionary[ElementDictionary[x+1][2]][0]-
NodeDictionary[ElementDictionary[x+1][3]][0])*(NodeDictionary[ElementDictionary[x+1][1]][2
]-NodeDictionary[ElementDictionary[x+1][3]][2]))
       -((NodeDictionary[ElementDictionary[x+1][1]][0]-
NodeDictionary[ElementDictionary[x+1][3]][0])*(NodeDictionary[ElementDictionary[x+1][2]][2
]-NodeDictionary[ElementDictionary[x+1][3]][2])))
cj = -1*(((NodeDictionary[ElementDictionary[x+1][0]][0]-
NodeDictionary[ElementDictionary[x+1][3]][0])*(NodeDictionary[ElementDictionary[x+1][2]][2
]-NodeDictionary[ElementDictionary[x+1][3]][2]))
       -((NodeDictionary[ElementDictionary[x+1][2]][0]-
NodeDictionary[ElementDictionary[x+1][3]][0])*(NodeDictionary[ElementDictionary[x+1][1]][2
]-NodeDictionary[ElementDictionary[x+1][3]][2])))
   ck = -1*(((NodeDictionary[ElementDictionary[x+1][1]][0]-
NodeDictionary[ElementDictionary[x+1][3]][0])*(NodeDictionary[ElementDictionary[x+1][0]][2
]-NodeDictionary[ElementDictionary[x+1][3]][2]))
       -((NodeDictionary[ElementDictionary[x+1][0]][0]-
NodeDictionary[ElementDictionary[x+1][3]][0])*(NodeDictionary[ElementDictionary[x+1][1]][2
]-NodeDictionary[ElementDictionary[x+1][3]][2])))
   cm = -(ci+cj+ck)
   di = -1*(((NodeDictionary[ElementDictionary[x+1][1]][0]-
NodeDictionary[ElementDictionary[x+1][3]][0])*(NodeDictionary[ElementDictionary[x+1][2]][1
]-NodeDictionary[ElementDictionary[x+1][3]][1]))
       -((NodeDictionary[ElementDictionary[x+1][2]][0]-
NodeDictionary[ElementDictionary[x+1][3]][0])*(NodeDictionary[ElementDictionary[x+1][1]][1
]-NodeDictionary[ElementDictionary[x+1][3]][1])))
```

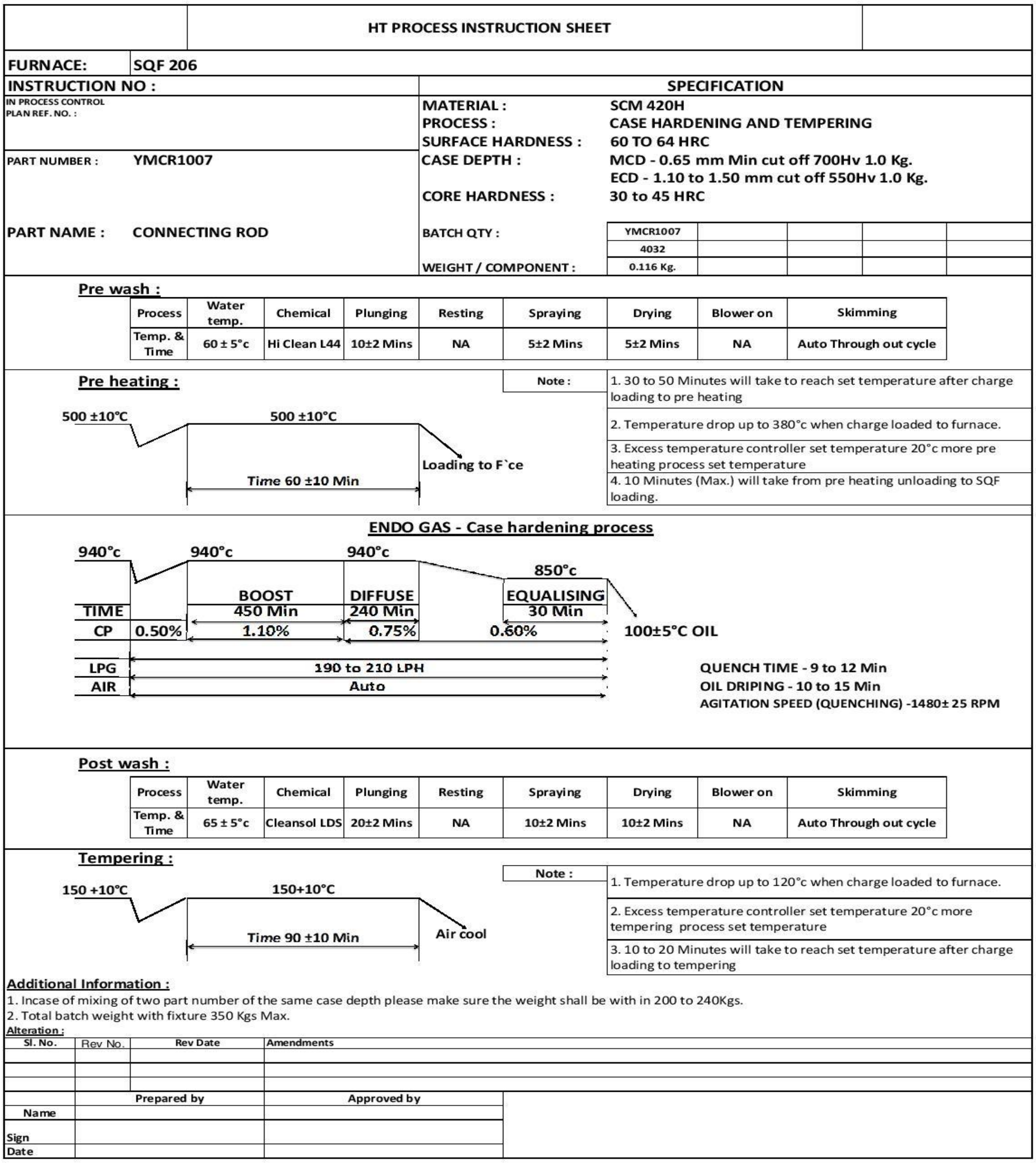
```
    dj = -1*(((NodeDictionary[ElementDictionary[x+1][2]][0]-
NodeDictionary[ElementDictionary[x+1][3]][0])*(NodeDictionary[ElementDictionary[x+1][0]][1
]-NodeDictionary[ElementDictionary[x+1][3]][1]))
          -((NodeDictionary[ElementDictionary[x+1][0]][0]-
NodeDictionary[ElementDictionary[x+1][3]][0])*(NodeDictionary[ElementDictionary[x+1][2]][1
]-NodeDictionary[ElementDictionary[x+1][3]][1])))
    dk = -1*(((NodeDictionary[ElementDictionary[x+1][0]][0]-
NodeDictionary[ElementDictionary[x+1][3]][0])*(NodeDictionary[ElementDictionary[x+1][1]][1
]-NodeDictionary[ElementDictionary[x+1][3]][1]))
          -((NodeDictionary[ElementDictionary[x+1][1]][0]-
NodeDictionary[ElementDictionary[x+1][3]][0])*(NodeDictionary[ElementDictionary[x+1][0]][1
]-NodeDictionary[ElementDictionary[x+1][3]][1])))
    dm = -(di+dj+dk)

    # B, D and BTranspose Matrix and multiplication
    B = (1 / (6 * elevolume)) * np.matrix([[bi, bj, bk, bm], [ci, cj, ck, cm], [di, dj, dk, dm]])
    D = np.array([[CarbonDiffusionCoefficient_D, 0, 0], [0,CarbonDiffusionCoefficient_D, 0],
                 [0, 0, CarbonDiffusionCoefficient_D]])
    BTranspose = np.array(B).transpose()
    Bmult = np.matmul(np.matmul(BTranspose, D), B)

   #Calculating the first half of the K matrix
   K1 = Bmult*elevolume
   #Calculating the second half of the K matrix
   K2 = np.zeros(Bmult.shape)
   Aijk = 0
   for i in range(list(Bmult.shape)[0]):
      for j in range(list(Bmult.shape)[1]):
        for k in range(list(Bmult.shape)[1]):
          if ElementDictionary[x+1][i] in diffusion_data_final:
             if ElementDictionary[x+1][j] in diffusion_data_final:
               if i==j:
                 K2[i,j]= 2
               else:
                 K2[i,j]= 1
               if ElementDictionary[x+1][k] in diffusion_data_final:
                 if (i!=j)&(i!=k)&(j!=k):
                   #Boundary length of the element
                   y = x
                   Aijk =
float(abs((np.linalg.det(np.array([[1,NodeDictionary[ElementDictionary[y+1][i]][1],
                       NodeDictionary[ElementDictionary[y+1][i]][0]],
                      [1,NodeDictionary[ElementDictionary[y+1][j]][1],
                    NodeDictionary[ElementDictionary[y+1][j]][0]],
                    [1,NodeDictionary[ElementDictionary[y+1][k]][1],
                     NodeDictionary[ElementDictionary[y+1][k]][0]]])))/2))
 Ke = K1+K2*beta*Aijk/12

 #K matrix assembly
 for i in range(list(Ke.shape)[0]): #Calculating the first half of the integral
    for j in range(list(Ke.shape)[1]):
       K[int(ElementDictionary[(x+1)][i])-1,int(ElementDictionary[(x+1)][j])-1]+=Ke[i,j]

 #C matrix
 ctemp = np.zeros((4,4))
 for i in range(4):
    for j in range(4):
       if i==j:
          ctemp[i,j]= 2*elevolume/20
       else:
          ctemp[i,j]= 1*elevolume/20

 #C matrix assembly
 for l in range(4):
    for m in range(4):
       C[int(ElementDictionary[(x+1)][l])-1,int(ElementDictionary[(x+1)][m])-1]+=ctemp[l,m]

#Load vector F
Fe = np.zeros((4,1))
p1 = np.zeros((4,1))
F2p1 = [0,0,0]
Aijk = 0
m=0
for i in range(4):
   if ElementDictionary[x+1][i] in diffusion_data_final:
      F2p1[m]=ElementDictionary[x+1][i]
      p1[i][0]+=1
      m+=1
if (F2p1[0]!=0)&(F2p1[1]!=0)&(F2p1[2]!=0):
   Aijk = float(abs((np.linalg.det(np.array([[1,NodeDictionary[F2p1[0]][1],
                   NodeDictionary[F2p1[0]][0]],
                  [1,NodeDictionary[F2p1[1]][1],
                   NodeDictionary[F2p1[1]][0]],
                  [1,NodeDictionary[F2p1[2]][1],
                   NodeDictionary[F2p1[2]][0]]])))/2))
```

```
        Fe=(beta*psi*Aijk/3)*p1

    #Assembly of F Matrix
    for i in range(list(Fe.shape)[0]):
        F[int(ElementDictionary[(x+1)][i])-1,0]+=Fe[i,0]

#Carbon potential calculation
Co = np.zeros((len(NodeDictionary),1))
for i in range(len(NodeDictionary)):
    Co[i,0] += 0.2
a = np.zeros((N,N))
b = np.zeros((N,1))
delt = 1
for dt in range(0,(12*60*60),delt):
    theta = 1
    Ceq = np.matmul((C-K*(1-theta)*delt), Co)
    r = C + (K* delt*theta)
    a = np.linalg.inv(r)
    b = (Ceq + F*delt*theta + F*delt*(1-theta))
    Co_final = np.matmul(a,b)
    Co = Co_final

import matplotlib.pyplot as plt
x = np.zeros((len(NodeDictionary),1))
for i in range(len(NodeDictionary)):
    x[i] = NodeDictionary[i+1][2]
plt.plot(x,Co_final,'o')
plt.show()

#Creating post process mesh file for GiD
f=open("trialpost.vtk","w+")
f.write("# vtk DataFile Version 2.0\n")
f.write("3D scalar data\n")
f.write("ASCII\n")
f.write("DATASET UNSTRUCTURED_GRID\n")
f.write("POINTS "+str(len(NodeDictionary))+" float\n")
for i in range(1,len(NodeDictionary)+1):
    f.write(str(NodeDictionary[i][0])+" "+str(NodeDictionary[i][1])+"
"+str(NodeDictionary[i][2])+"\n")
f.write("CELLS "+str(len(ElementDictionary))+" "+str(len(ElementDictionary)*5)+"\n")
for i in range(1,len(ElementDictionary)+1):
    f.write("4 "+str(int(ElementDictionary[i][0])-1)+" "+str(int(ElementDictionary[i][1])-1)+"
"+str(int(ElementDictionary[i][2])-1)+" "+str(int(ElementDictionary[i][3])-1)+" "+"\n")
```

```
f.write("CELL_TYPES "+str(len(ElementDictionary))+"\n")
for i in range(1,len(ElementDictionary)+1):
    f.write("10\n")
f.write("POINT_DATA "+str(len(NodeDictionary))+"\n")
f.write("SCALARS Carbon_percentage float 1\n")
f.write("LOOKUP_TABLE default\n")
for j in range(0,len(NodeDictionary)):
    f.write(str(float(Co_final[j]))+" ")
f.close()
```

APPENDIX B

Attached below are the HT Process Instruction sheet for two-part numbers with different input parameters provided by Sansera Engineering. These were used as input specifications for verification of the finite element model.



HT PROCESS INSTRUCTION SHEET

FURNACE:	SQF 206
INSTRUCTION NO :	
IN PROCESS CONTROL PLAN REF. NO. :	
PART NUMBER :	YMCR1007
PART NAME :	CONNECTING ROD

SPECIFICATION

MATERIAL :	SCM 420H
PROCESS :	CASE HARDENING AND TEMPERING
SURFACE HARDNESS :	60 TO 64 HRC
CASE DEPTH :	MCD - 0.65 mm Min cut off 700Hv 1.0 Kg. ECD - 1.10 to 1.50 mm cut off 550Hv 1.0 Kg.
CORE HARDNESS :	30 to 45 HRC
BATCH QTY :	YMCR1007
	4032
WEIGHT / COMPONENT :	0.116 Kg.

Pre wash :

Process	Water temp.	Chemical	Plunging	Resting	Spraying	Drying	Blower on	Skimming
Temp. & Time	60 ± 5°c	Hi Clean L44	10±2 Mins	NA	5±2 Mins	5±2 Mins	NA	Auto Through out cycle

Pre heating :

500 ±10°C

500 ±10°C

Time 60 ±10 Min

Loading to F`ce

Note :
1. 30 to 50 Minutes will take to reach set temperature after charge loading to pre heating
2. Temperature drop up to 380°c when charge loaded to furnace.
3. Excess temperature controller set temperature 20°c more pre heating process set temperature
4. 10 Minutes (Max.) will take from pre heating unloading to SQF loading.

ENDO GAS - Case hardening process

	940°c	940°c	940°c		850°c	
		BOOST	DIFFUSE		EQUALISING	
TIME		450 Min	240 Min		30 Min	
CP	0.50%	1.10%	0.75%	0.60%		100±5°C OIL
LPG	190 to 210 LPH					
AIR	Auto					

QUENCH TIME - 9 to 12 Min
OIL DRIPING - 10 to 15 Min
AGITATION SPEED (QUENCHING) -1480± 25 RPM

Post wash :

Process	Water temp.	Chemical	Plunging	Resting	Spraying	Drying	Blower on	Skimming
Temp. & Time	65 ± 5°c	Cleansol LDS	20±2 Mins	NA	10±2 Mins	10±2 Mins	NA	Auto Through out cycle

Tempering :

150 +10°C

150+10°C

Time 90 ±10 Min

Air cool

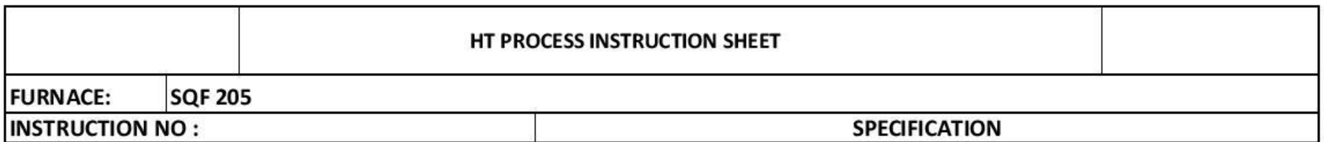
Note :
1. Temperature drop up to 120°c when charge loaded to furnace.
2. Excess temperature controller set temperature 20°c more tempering process set temperature
3. 10 to 20 Minutes will take to reach set temperature after charge loading to tempering

Additional Information :
1. Incase of mixing of two part number of the same case depth please make sure the weight shall be with in 200 to 240Kgs.
2. Total batch weight with fixture 350 Kgs Max.

Alteration :

Sl. No.	Rev No.	Rev Date	Amendments

	Prepared by	Approved by
Name		
Sign		
Date		



HT PROCESS INSTRUCTION SHEET

FURNACE:	SQF 205
INSTRUCTION NO :	

SPECIFICATION