Integration of machine learning with phase field modeling to study equilibrium precipitate shape

A Thesis

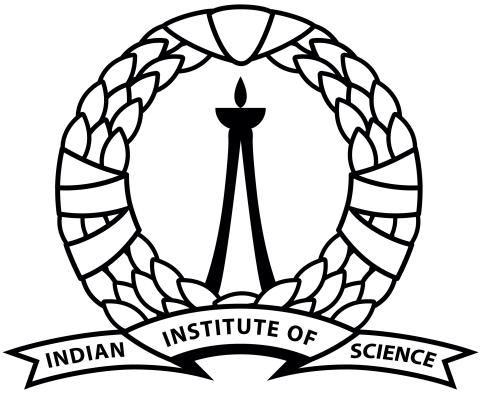
Submitted for the Degree of

Master of Technology

in the **Faculty of Engineering**

by

Ajay Sagar



Department of Materials Engineering
Indian Institute of Science
Bangalore – 560 012 (INDIA)

JUNE 2021

DEDICATED TO

my Parents and Teachers

Signature of the Author: .....

Ajay Sagar

Dept. of Materials Engineering

Indian Institute of Science, Bangalore

Signature of the Thesis Supervisor: .....

Dr. Abhik Choudhury

Associate Professor

Dept. of Materials Engineering

Indian Institute of Science, Bangalore

Acknowledgements

My past two years at IISc was nothing but an adventurous ride. I lived every moment here and through this acknowledgements chapter. I got the opportunity to thank everyone who have played a part in making this happen. First and foremost, I would like to express my sincere gratitude to Prof. Abhik Choudhury for the invaluable advice, continuous support, and patience.

I would like to thank Sumeet, swapnil and Jitin for their valuable guidance. Who provided me all the helps and tools that is required to choose the right direction and for successful completion of my thesis.I would also like to thank all my friends at the department with whom I enjoyed my past two year through various activities like Cultural fests, Holi and Diwali celebration and, institute open days.

In addition, I would like to thank my parents for their wise counsel and sympathetic ear.Who were always there for me. Finally, there are my friends, who were of great support in deliberating over our problems and findings, as well as providing happy distraction to rest my mind outside of my research.

Abstract

The phase-field method has become an important and highly versatile technique for simulating microstructure evolution at the mesoscale. Still, some simulations can take up to 2-3 days or more to run. This research aims to integrate machine learning techniques to make an ML model based on phase simulation data to predict equilibrium precipitate shape. This thesis contains four chapters.

In the first chapter, we build a linear regression model to predict equilibrium precipitate shape for a given Az and E, i.e., Anisotropy in elastic and surface energy. the second chapter discusses the variation in precipitate morphology with inter-particle distance.

Artificial Neural Network algorithms have long been used for modelling, and they provide automated knowledge extraction and high inference accuracy. In the third chapter, we train such a model to predict the shape of equilibrium precipitate for given Az and eigenstrains.

The last chapter discusses the tool PyMKS (Python-Based Materials Knowledge Systems), which is an open-source materials data science framework that can be used to produce high-fidelity, reduced-order (i.e., low computational cost) and, process–structure-property (PSP) linkages for a broad range of material systems having a rich hierarchy of internal structures spanning multiple length scales. This section about the alternate approach for study equilibrium precipitate shape.

Contents

List of Figures

List of Tables

Chapter 1

Phase-Field Modeling to study the effects of Anisotropy on equilibrium Precipitate Morphology

1.1 Background

The phase-field method has become an efficient and extremely versatile simulation approach for modelling microstructure evolution at the smaller scale such as mesoscale.A distinct advantage of phase field technique is that there is no need to explicitly track various complex evolution of interfaces, such as grain boundaries. The microstructure is described by a system of continuous variables, where the microstructure interfaces have a finite width over which the variables transition between values and using the phase-field for simulation the study of microstructural evolution in a wide variety of material processes,such as solidification, solid-state phase transformations, precipitate growth and coarsening,martensitic transformations and grain growth, domain evolution in ferro-electric and ferro-magnetic materials becomes easy.[3][4]

The purpose of this chapter is to provide a broad overview on the different aspects of coarsening of coherent precipitates.The chapter begins with a brief discussion on the different anisotropies

(like A_Z, elastic anisotropy and ϵ, interfacial energy anisotropy) in materials which can affect the precipitate morphologies.

1.2 Research Work

The sole aim of this chapter is to find the equilibrium precipitate shapes of coherent precipitates and more importantly the understanding of equilibrium morphology of precipitates as a function of elastic anisotropy A_Z and interfacial energy anisotropy ϵ. And building a machine learning model to predict equilibrium precipitate shape for given elastic anisotropy and interfacial energy anisotropy

1.3 Outline

In the following section, we firstly describe our physical model followed by implementation in OpenFOAM and study the effect of anisotropy in elastic and surface energy.Lastly we build an Machine Learning model based on data obtained from OpenFOAM and perform verification and validation of the model

1.4 Physical Model

In the phase-field method, the individual phases (i.e.,precipitates and matrix) and their crystallographic variants are described by a set of order parameters for this case ϕ is our order parameter. The interface between these two phases has a gradient in one of the order parameters that varies from 0 to 1 at the corresponding interface. Solid-state phase transformations mostly involves a difference of the lattice parameter between the matrix and the precipitates. which gives rise to a misfit strains/stresses for a coherent interface. which in turn contributes

to the system energy in terms of an elastic contribution and it scales with the volume of the precipitate. Similarly, the interfacial energy which is the other component of the energy in the system, varies with the interfacial area. In this context, the equilibrium shape of the precipitate is the one which minimizes the sum total of the contributions from both the elastic energy and interfacial components, which given the scaling of the two energy components is a function of the size of the precipitate. In this section, we formulate a phase-field model, where the functional consists of both the elastic and the interfacial energy contributions. Since the equilibrium precipitate shape depends on the size of the precipitate, we formulate a model which minimizes the system energy while preserving the volume of the precipitate, and thereby allows the computation of the equilibrium shape of precipitates. This allows the determination of the precipitate shapes as a function of the different precipitate sizes as has been done previously using sharp-interface methods. This constrained minimization is achieved through the technique of volume preservation which is also described elsewhere[2] that is essentially the coupling of the Allen–Cahn type equation with a correction term using a Lagrange parameter that ensures the conservation of the precipitate volume during evolution.

In the following, we discuss the details of the phase-field model. We begin with the free energy functional that reads

$$\mathcal{F}(\phi) = \int_v \left[\gamma W a^2(n)|\nabla\phi|^2 + \omega(\phi)\right] dV + \int_v f_{el}(u,\phi)\, dV + \lambda_\beta \int_v h(\phi)\, dV \tag{1.1}$$

where V is the volume of system. ϕ is the phase field order parameter,$\phi = 1$ is precipitate phase and $\phi = 0$ is matrix phase. The first term on the right hand side of equation represents the interfacial energy which is sum total of gradient energy and potential contributions. γ controls the interfacial energy in the system. W influences the width of the diffuse interface between precipitate and matrix phases. a(n) represents interfacial energy anisotropy between

$\nabla\phi$ matrix/precipitate phase. It is a function of interface normal given by, $n = \frac{\nabla\phi}{|\nabla\phi|}$. The second term in first integral is double obstacle potential given as,

$$\omega(\phi) = A\phi^2(1-\phi)^2 \tag{1.2}$$

Where ϕ is order parameter and $A = \frac{9\gamma}{W}$

The second integral is elastic energy contribution to the free energy density of the system which is a function of the order parameter ϕ that is also used to interpolate between the phase properties and the misfit. $h(\phi)$ represents the volume of precipitate,where $h(\phi) = \phi^2(3-2\phi)$ is an interpolation function which varies from 0 to 1. λ_β is the Lagrange parameter that is added for volume conservation of the precipitate. The evolution equation of ϕ as given by Allen-Cahn is

$$\tau W\frac{\partial\phi}{\partial t} = \frac{\delta\mathcal{F}}{\partial\phi} \tag{1.3}$$

and elaborates as

$$\tau W\frac{\partial\phi}{\partial t} = 2\gamma W\nabla\cdot\left[a(n)\left(\frac{\partial a(n)}{\partial\nabla\phi}|\nabla\phi|^2 + a(n)\nabla\phi\right)\right] - \frac{d\omega(\phi)}{d\phi} - \frac{f_{el}(u,\phi)}{\partial\phi} - \lambda_\beta h'(\phi) \tag{1.4}$$

where τ is the interface relaxation constant, which in the present modeling context is chosen as the smallest value that allows for a stable explicit temporal evolution using a simple finite difference implementation of the forward Euler-scheme. Note, 0 denotes differentiation of the function with respect to its argument. In order to complete the energetic description, it is important to elaborate the elastic energy density $f_{el}(u,\phi)$ in terms of the physical properties of the matrix and the precipitate phases that are the stiffness matrices, as well as the misfit

1.5 Implementation in OpenFOAM

1.5.1 Implementation

Allen Cahn dynamics evolution for ϕ is given by following equation

$$\tau W \frac{\partial \phi}{\partial t} = 2\gamma W \nabla \cdot \left[a(n) \left(\frac{\partial a(n)}{\partial \nabla \phi} |\nabla \phi|^2 + a(n) \nabla \phi \right) \right] - \frac{d\omega(\phi)}{d\phi} - \frac{f_{el}(u, \phi)}{\partial \phi} - \lambda_\beta h'(\phi) \tag{1.5}$$

Where

$$\frac{d\omega(\phi)}{d\phi} = 2A\phi(1-\phi)(1-2\phi)$$

and

$$h'(\phi) = 6\phi(1-\phi)$$

For OpenFOAM implementation code , refer [appendix A]

1.5.2 Effect of anisotropy in elastic energy (A_Z) and interfacial energy (ϵ)

To study the effect of anisotropy in elastic energy, we altered the magnitude of Az from 0.3 to 3.0 and keep the other parameter a constant value And to study the effect anisotropy in interfacial energy (ϵ) we increase the value of ϵ from 0.01 to 0.04, while keeping the values of other parameter constant. We study these effect on precipitate morphology separately by changing only Az and ϵ at a time

1.5.2.1 Effect of anisotropy in elastic energy

Here, we investigated three prominent cases i.e. A_Z = 0:3; A_Z = 1:0 and A_Z = 3:0, where the effect of the change in magnitude of Az on the equilibrium morphologies is discussed. All these simulations are performed with condition, that the precipitate sizes is kept below the bifurcation point.

for the case, When Az >1, (in this case $A_Z = 3.0$), keeping other parameter constant. The equilibrium precipitate morphology (thick green line) shown in figure[1.1], where there is cubic anisotropy in the elastic energy which drives the precipitate to acquire a square like shape with rounded corners where the square faces are aligned normal to the elastically soft directions<10 >as Az > 1. and result in the cube shaped precipitate shape

For the case, where we keep Az = 1.0 and, keeping other parameter constant. The equilibrium precipitate takes a circular shape (thick orange line) due to isotropic elastic energy with no anisotropy(Az =1) in interfacial energy. In this case the elastic modulus in each direction is same. Hence, no net driving force And as a result the equilibrium precipitate tends to acquire

perfect circular shape

For last the case, When Az <1, (in this case $A_Z = 0.3$),keeping other parameter constant. The resultant equilibrium precipitate form a shape which prefers to align its face normal to <11 >directions which is elastically softer as the Az <1. This is shown in figure[1.1] (Thick blue line), which shows the precipitate morphologies where the anisotropy in elastic energy drive the equilibrium precipitate shape like a diamond and has its faces normal to <11 >directions. As we deceases the values of Az from 1 the precipitate acquire more and more diamond structure i.e., the precipitate's faces normal to <11 >directions become more and more sharper as decreasing Az.

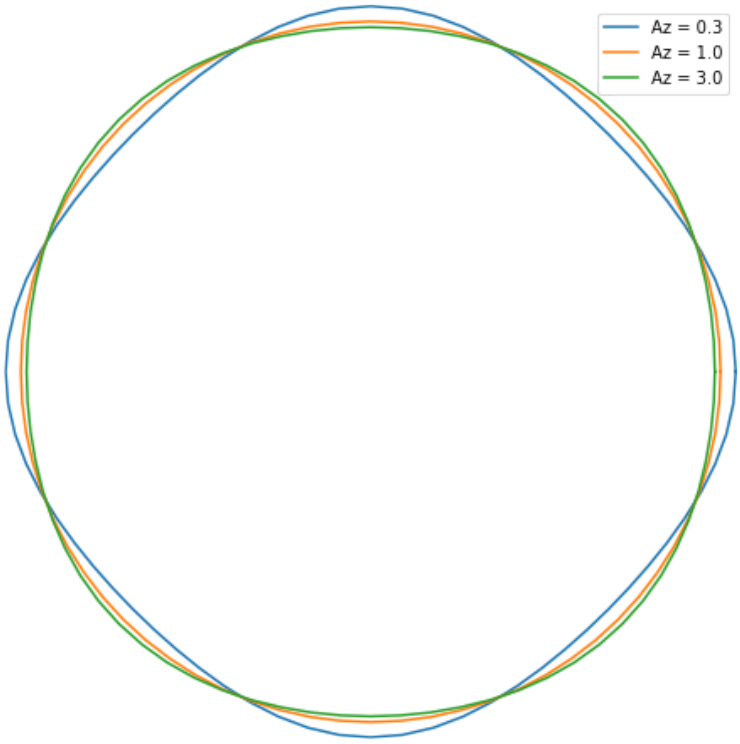


Figure 1.1: Comparison of Equilibrium precipitate morphology for different cases where Az=0.3 , A =1.0 and Az= 3.0

1.5.2.2 Effect of anisotropy in interfacial energy

lastly, Here, we vary the magnitude of ϵ from 0.0 to 0.06, while holding the other parameters constant. The equilibrium morphology of the precipitate initially (For $\epsilon = 0.0$, Thick blue line) acquires a circular shape as there is no net driving force acting in any of the direction. Further, with increase in the strength of ϵ, the precipitate faces start to orient towards <11 >directions. This is due to a significant increase in the strength of anisotropy in interfacial energy. Thus, an increase in the strength of anisotropy in interfacial energy imparts a driving force for alignment of the precipitate faces normal to the <11 >directions rather than <10 >directions, giving rise to a diamond like shape as shown in Fig. 1.2.

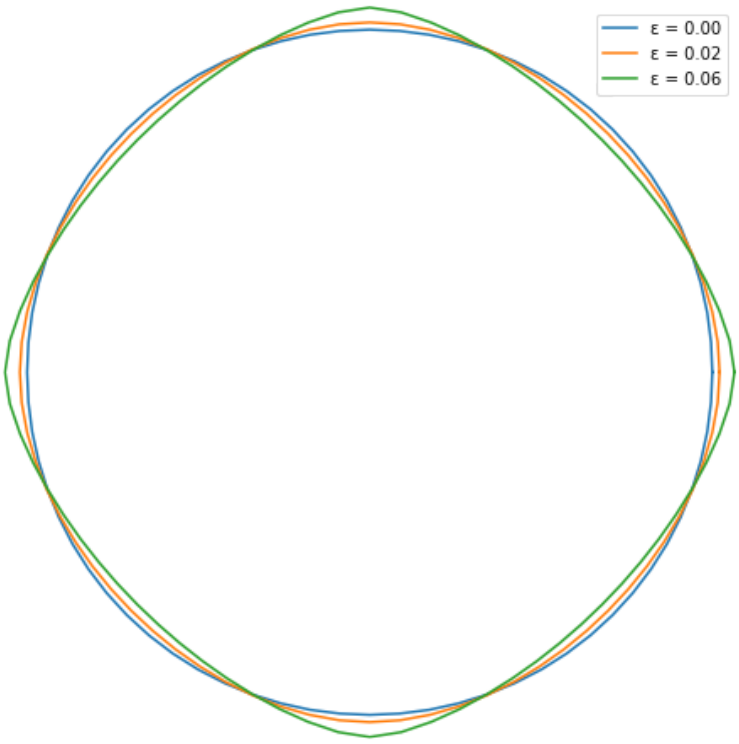


Figure 1.2: Comparison of Equilibrium precipitate morphology for different cases where $\epsilon = 0.00$, $\epsilon = 0.02$ and $\epsilon = 0.06$

1.6 Machine Learning

In this section, we discuss the Machine learning techniques used to train the data obtain from OpenFOAM and build an ML- model to predict equilibrium precipitate shape based on given input.

In this Project, the main aim to do machine learning was to reduce the time taken for obtaining a result, as the taken for simulations in OpenFOAM is around 24-48 hrs. but with the help of a trained ML model it be obtain instantaneously.

1.6.1 Introduction to Machine Learning

Types of Machine Learning Machine learning can be sub-categorized into three types:

1) Supervised Learning :

It is the most basic type of machine learning.In supervised Learning, the machine learning algorithm is trained on given labeled data. Here, the Machine Learning algorithm is given a training data-set to work with and this training data-set is a smaller part of the large data set and we use this data to build a ML model and once the ML model gets trained on data, it can start making a prediction or decision when fresh data is given to it.

Common types of Supervised Learning:

- Regression - linear , polynomial
- Classification
- decision tree
- Random forest

2) Unsupervised Learning :

The unsupervised machine learning have a advantage of being able to work without labeled data. This model learns through observation and finds structures within the data. And once the model is given a new data set, it automatically finds relationships and patterns in the data set by creating clusters in it. Types of unsupervised learning:

- Clustering
- Association Analysis
- Hidden Markov Model

3) Reinforcement Learning :

Reinforcement learning directly takes inspiration from how the human beings learn from data in their lives. It features an algorithm that improves upon itself and learns from new situations using a trial-and-error method. Favorable outputs are encouraged or 'reinforced', and non-favorable outputs are discouraged or 'punished'.

1.6.2 Linear Regression

Most commonly used techniques of Supervised Machine Learning algorithms is linear regression.Simple linear regression models the relationship between the magnitude of one variable to that of a second. Here, A single/Multiple input data variable(s) is used to predict one (or more) output data variables, assuming that the input data variable(s) is not correlated with each other. It is represented as :[6]

$$Y = b_0 + b_1 X + e$$

Where, Y is dependent data variable output and X is an Non-dependent data variable whereas b_0 and b_1 are the linear regression coefficients. The Value of b_0 and b_1 depends on the

best fitted line where b_1 is the slop of line and b_0 is the intercept with e as an explicit error term e.A linear regression model that fits the data well when is set up such that changes in X lead to changes in Y.However, by itself, the regression equation does not prove the direction of causation. The figure [1.3] shows that how a linear regression model fits a data on a line where Y_i is the original data outputs and the fitted values, (also referred to as the predicted values), which are obtained from best fitted line are generally denoted by $\hat{Y}_i$. e can be calculated by subtracting the predicted values from the original data :

$$e = (y_i - \hat{y}_i)$$

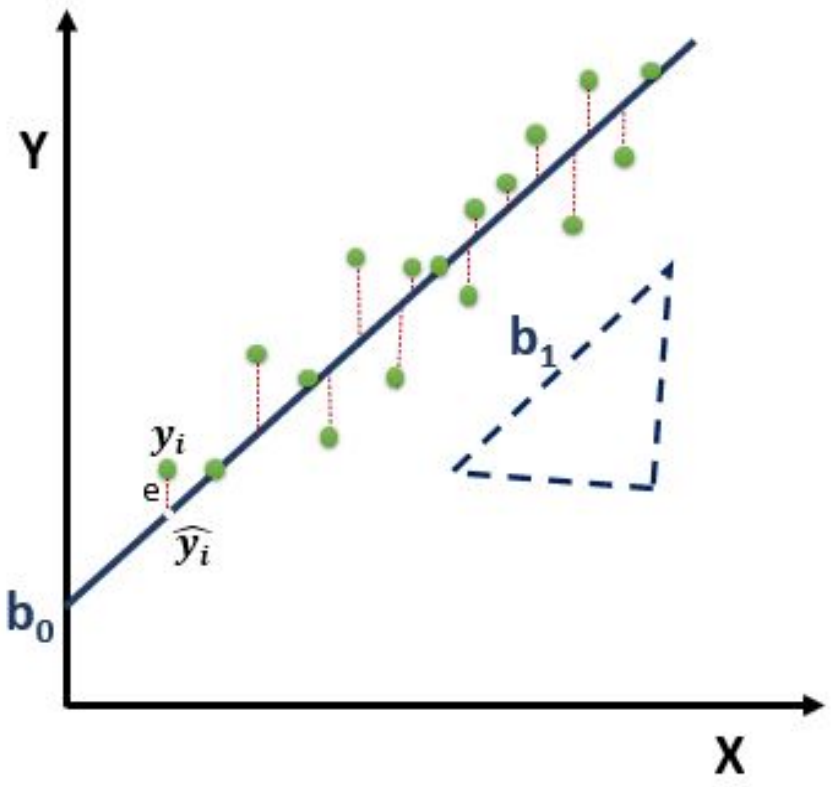


Figure 1.3: Slope and intercept for the regression fit to the regression equation

1.6.3 Data Preparation

This step is concerned with transforming the raw data that was collected into a form that can be used in modeling. The data collected from OpenFOAM contained coordinates of points for equilibrium precipitate shape at $\phi = 0.5$ and then shape transformed to radial distances at every 5° angle as shown in figure below :

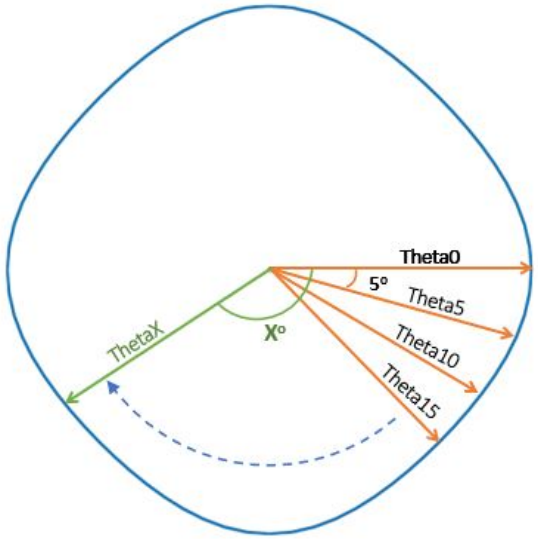


Figure 1.4: Calculation of radial distances at every 5° angle for equilibrium precipitate shape for Az = 0.3

Now, we tabulated our data as shown in table[1.1]

	Az	Theta0	Theta5	Theta10	...	Theta355	Theta360
0	0.3	34.886311	34.888082	34.902252	...	34.888024	34.886311
1	0.4	35.066715	35.053381	35.031217	...	35.053307	35.066715
2	0.5	35.249462	35.22613	35.16027	...	35.226043	35.249462
...	...	...	...	...	...	...	...
19	2.5	35.434116	35.399546	35.288556	...	35.399491	35.434116
20	3.0	35.623962	35.571879	35.418418	...	35.571824	35.623962

Table 1.1: Data Preparation for Regression Model

1.6.4 Training and Testing Data

Here, the we split the data into training and testing sets.Setting a variable X equal to the numerical features of the data [Az] (for Case when studying the effect of Az) or [ϵ] (for Case when studying the effect of ϵ) and a variable y equal to the radial distance.

`model_selection.train_test_split` from sklearn is a great tool for splitting data in desired ratio for training and testing purpose. Setting `|test_size=0.3|`which divides total data into training and testing data in the ratio of 70% and 30% respectively [5].For training the data, we are use Linear Regression model, using this we create a model named lm and then fit our training data in this model and predictions were done afterwards as

Firstly, we declare input variable data As X and output data as y which we are going to predict later using input variable X.

```
X= data[Az] ( Case when studying Az )
X= data[ϵ] ( Case when studying ϵ )
y =data[ [ Theta0, Theta5, Theta10, ... , Theta350, Theta355, Theta360 ] ]
```

Splitting data for training and testing purpose.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

Using Linear regression model to fit training data.

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

Prediction of Redial distance using testing data.

```
predictions = lm.predict(X_test)
```

1.6.5 Evaluation of Model

Model evaluation aims to estimate the generalization accuracy of a model on future (unseen/out-of-sample) data. This evaluation can be easily done by MAE , MSE ,RMSE and most commonly used R-Square SCORE.

MAE (Mean Absolute Error) is the average of the absolute difference between the predicted values and observed value. The MAE is a linear score which means that all the individual differences are weighted equally in the average.

$$MAE = (\frac{1}{n}) \sum_{i=1}^{n} |y_i - x_i|$$

MSE (Mean Square Error) is defined as Mean or Average of the square of the difference between actual and estimated values.

$$MSE = (\frac{1}{n}) \sum_{i=1}^{n} (y_i - x_i)^2$$

RMSE (Root Mean Square Error) It represents the sample standard deviation of the differences between predicted values and observed values (called residuals). Mathematically, it is calculated using this formula:[6]

$$RMSE = \sqrt{(\frac{1}{n}) \sum_{i=1}^{n} (y_i - x_i)^2}$$

R2 SCORE/R-Square is a statistical measure that represents the goodness of fit of a regression model. The ideal value for r-square is 1. The closer the value of R-square to 1, the better is the model fitted.R-square is a comparison of residual sum of squares SS_{res} with total sum

of squaresSS_{tot}. Total sum of squares is calculated by summation of squares of perpendicular distance between data points and the average line. And the residual sum of squares is calculated by the summation of squares of perpendicular distance between data points and the best fitted line.[6]

$$SS_{res} = SUM(y_i - y_{avg})^2$$

$$SS_{tot} = SUM(y_i - \hat{y}_i)^2$$

$$R2 - SCORE = 1 - \frac{SS_{res}}{SS_{tot}}$$

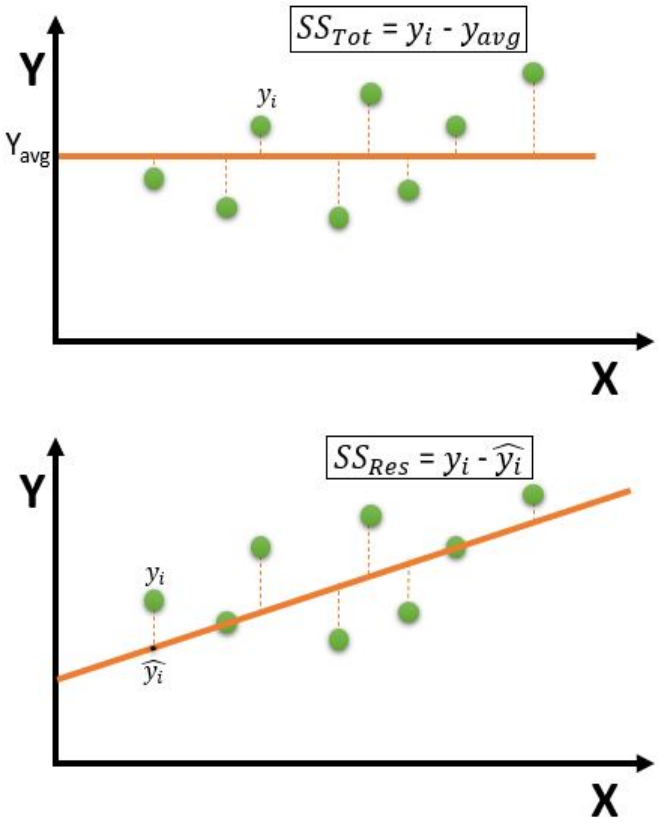


Figure 1.5: Comparison of Equilibrium precipitate morphology obtained form Machine Learning model and Phase-Field Model

Model Evaluation :

Here we evaluated our trained model on test data set. the below plot[1.6] shows the correlation between the data output from trained model and test data set.The plot shows a strong correlation, concluding good prediction capability of our linear regression model. Which can be easily interpreted from the R2 SCORE value given below. Accuracy of model increase as the R2 SCORE tends to 1.

MAE: 0.0152

MSE: 0.0003

RMSE: 0.0184

R2 SCORE: 0.9729

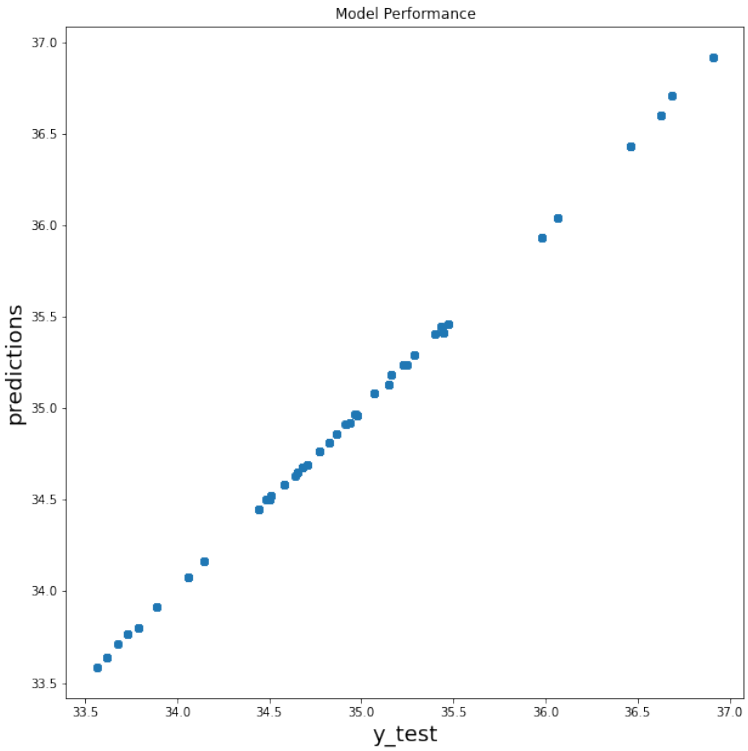


Figure 1.6: Evaluation of Machine Learning Model

1.6.6 Model Verification

In machine learning, model verification can be stated as the process where a trained model is evaluated with a fresh data set. Here we have taken a random case where Az=0.45 . The below figure [1.7] shows the comparison between the equilibrium precipitate shapes obtained from OpenFOAM and ML model.

On observing figure [1.7], the equilibrium precipitate shape obtained from OpenFOAM and the predicted equilibrium precipitate shape from our regression model perfectly overlap to each other. i.e., equilibrium precipitate shape was found to be same in both cases. Which shows a very good predictably of our Linear Regression model.

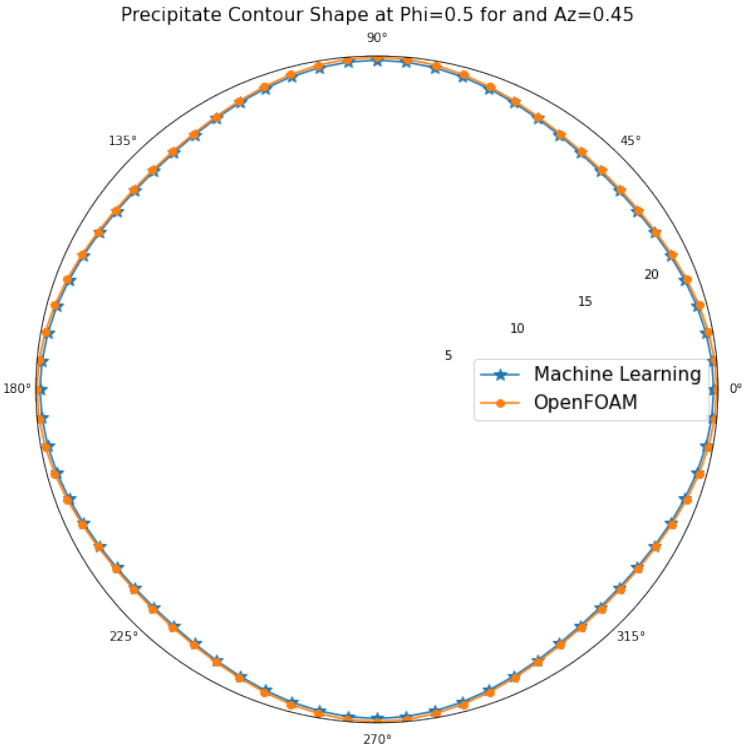


Figure 1.7: Comparison of Equilibrium precipitate morphology obtained form Machine Learning model and OpenFOAM(Phase-Field Model)

Chapter 2

Phase-Field Modeling to study the effects of inter-particle distance on Precipitate Morphology

2.1 Background

The phase-field method has become an efficient and extremely versatile simulation approach for modelling microstructure evolution at the smaller scale such as mesoscale.A distinct advantage of phase field technique is that there is no need to explicitly track various complex evolution of interfaces, such as grain boundaries. The microstructure is described by a system of continuous variables, where the microstructure interfaces have a finite width over which the variables transition between values and using the phase-field for simulation the study of microstructural evolution in a wide variety of material processes,such as solidification, solid-state phase transformations, precipitate growth and coarsening,martensitic transformations and grain growth, domain evolution in ferro-electric and ferro-magnetic materials becomes easy.The mechanical properties of precipitation-hardened alloys depend on the precipitate size, shape and their arrangement in the matrix[7] and this section deals about such parameter i.e., inter-particle which

affects the precipitate morphology.

The purpose of this chapter is to provide a broad overview on the different aspects of coarsening of coherent precipitates. study the effect of change in morphology by changing the inter-particle distance.

2.2 Research Work

The sole aim of this chapter is to study the effect of inter-particle distance on equilibrium precipitate shapes

2.3 Outline

In the following section, we firstly describe our physical model followed by implementation in OpenFOAM and then try to study the effect of inter-particle distance on equilibrium precipitate shapes

2.4 Implementation in OpenFOAM

As in this section we are studying the effect of inter-particle distance. The inter-particle distance can be imposed by changing the mesh size of system and using the periodic boundary condition ensures the interaction between neighbouring particles. The figure [2.1] shows the case where the mesh size was 120x120, as the result the aspect ratio of the precipitate is found to be equal unlike the case when mesh size was 120x240, here we observe some elongation in the direction where the inter-particle distance was less due to the elastic stress field as shown in figure [2.2].

For implementation code in OpenFOAM, refer section [1.5] from previous chapter

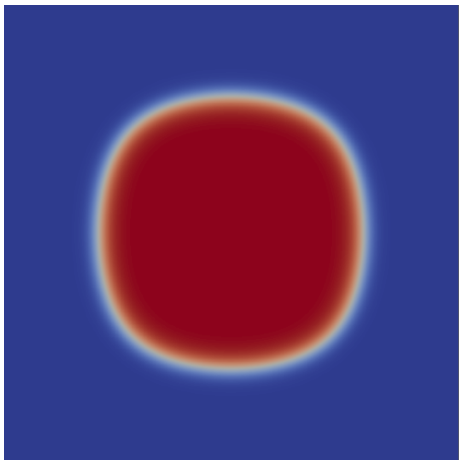


Figure 2.1: Equilibrium precipitate shape when particle are at equal distance

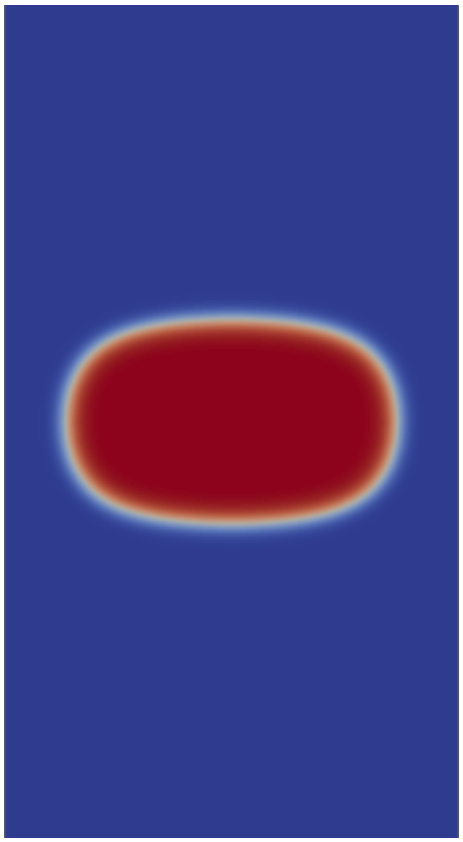


Figure 2.2: Equilibrium precipitate shape when inter-particle distance is very low and ratio is 1:2

2.4.1 Effect of inter-particle distance on precipitate morphologies

During solid-state phase transformations, there can be a difference in the lattice parameter between the precipitate and the matrix. And these difference in the lattice parameter gives rise to misfit strains/stresses around precipitate interface and thus generating elastic stress field. When the particles are too close to each other their elastic stress field interfere with each other, resulting change in morphology.

The figure [2.3] shows that, when the inter-particle distance was less the precipitate gets elongate in the direction where inter-particle distance was less as the elastic field interaction is very high at shorter distance. And, on increase the inter-particle distance the precipitate acquires the shape where aspect ratio is more closer to 1. And after a limit we don't see the effect of inter-particle spacing as elastic field interaction is not strong enough.

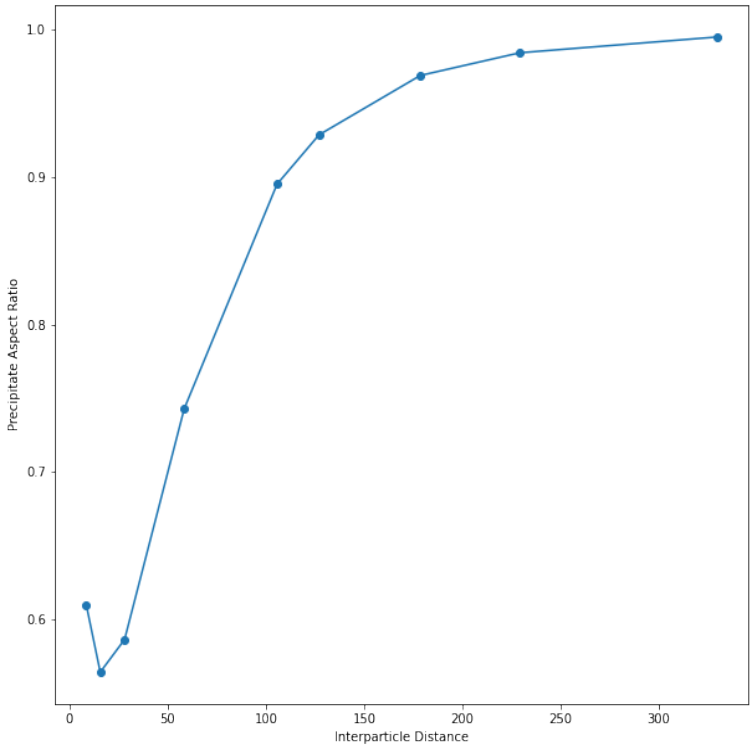


Figure 2.3: precipitate morphology with respect to inter-particle distance on equilibrium

2.5 Machine Learning - Linear regression Model

Here, we perform the similar procedure as earlier (refer section 1.6.4) and as for this case as we are training the inter-particle distance with precipitate morphology. So , we declare input variable data (X) consisting inter-particle distance in x and y direction respectively i.e., X = [IPD_x , IPD_y] as shown in figure [2.4]..

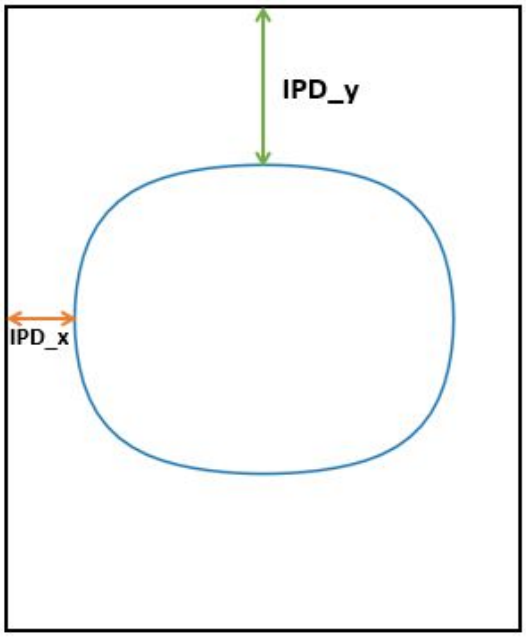


Figure 2.4: Calculation of inter-particle distance in x and y direction

And output data (y) consisting all the redial distance at every 5° degree from center of precipitate, which is going to be predicted later using input variable X.

	IPD_x	IPD_y	Theta0	Theta5	Theta10	....	Theta350	Theta355	Theta360
1	15.73	78.80	47.1311	46.9611	46.4788	....	46.4787	46.9647	47.1311
2	28.52	105.85	45.9459	45.6557	45.5184	....	45.5437	45.8486	45.9459
....	....	....	....	....	....	...	....	....	....
28	279.44	350.02	35.2137	35.2809	35.4758	....	35.2545	35.4594	35.2137
29	329.56	409.92	35.2789	35.3463	35.5407	....	35.5156	35.21315	35.2789

Table 2.1: Data Preparation for training model

Hence, the training parameters are declared as :

X= data[IPD_x , IPD_y]

y =data[[Theta0, Theta5, Theta10, ... , Theta350, Theta355, Theta360]]

We fit the data as given in table [2.1]. in an Linear Regression model in a similar way as discussed earlier in previous chapter [1.6.4].

Model Evaluation

The figure 2.5(a) shows the correlation between the theoretical inter-particle distance (i.e.,

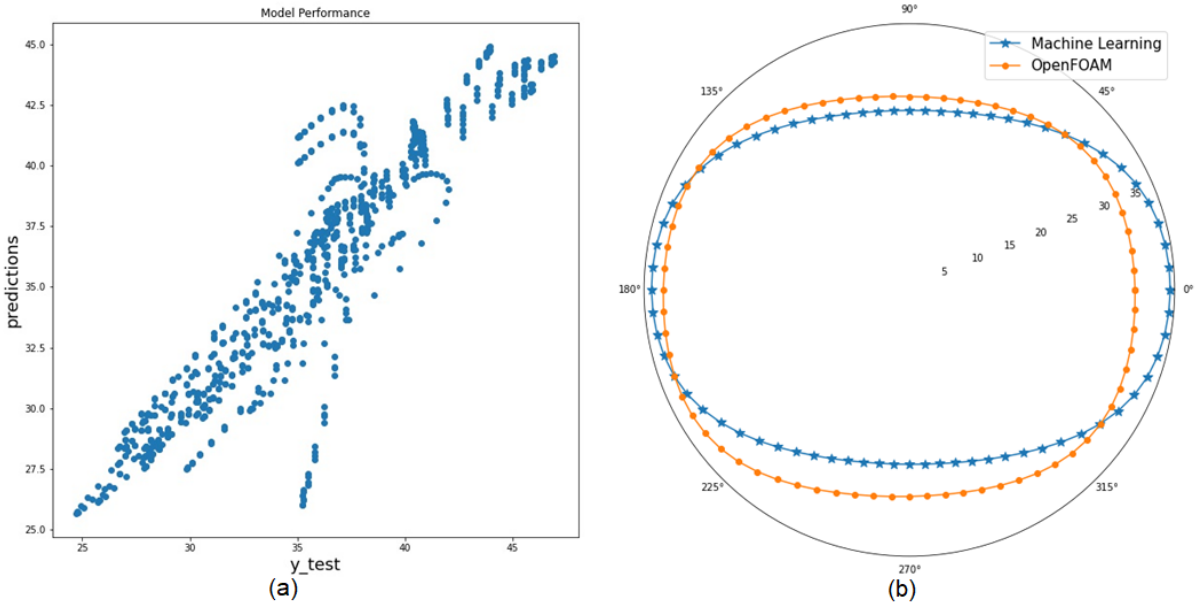


Figure 2.5: Model evaluation for inter-particle distance

y_test) and the predicted inter-particle distance and as the points are found to be much scattered instead of straight line which shows the poor predictability of this model, which can be also seen in the figure 2.5(b) showing the comparison between precipitate shape predicted from ML model and precipitate obtain from OpenFOAM. For this model RMSE = 2.47 and the R2 SCORE = 0.693 and this is happening because of the poor predictability of Linear regression for non-linear relation.

2.6 Neural Networks (NNs)

2.6.1 Biological Neurons

The human brain's biological neurons or Perceptrons are functional units of our nervous system and helps in transmit electrical signals to long distance, which consists of [10]

1) Dendrites, which are there to take inputs from other neurons or the sensory organs .

2) Soma, which is some kind of processing unit it takes all the signals from the neuron and does some processing on top of it .

3) Synapse, these are the strength of the interaction between two neurons. Means help in transferring signals from one neuron to another.

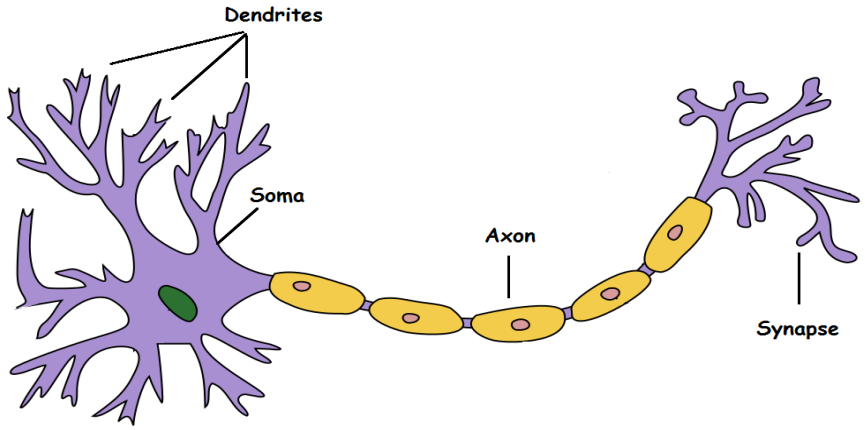


Figure 2.6: Illustration of biological neurons. [Source : Creative Commons - Wikipedia]

2.6.2 Artificial Neuron

Artificial Neural Networks are inspired by the human brain's biological neural network, which consists of [10]

1) Input layer (x_i), It is a collection of array values that can be used to pass input values to a neuron or Perceptron using this layer. The input layer in ANNs is similar to a dendrite in human biological neurons.

2) Weights (w_i) **and Bias** (b), Weights are the values that are multiplied to the respective input values to control the signal (or the strength of the correlation) between two neurons. In other words, a weight corresponding to an input decides how much influence will it have on the output. Next, we add a bias value to our weighted sum to get the final value for prediction by our neuron.

3) Activation Function, It decides whether or not the corresponding neuron is fired and determines the output values that the neuron should generate. In simple words, the activation function can be considered as a mathematical function that can normalise the given inputs.[10]

4) Output Layer ($\hat{y}_i$), It is the final layer that gives the final output of a neuron, and it can then be passed to other neurons in the network or can also be considered as the final output value.

$$\hat{y}_i = \sum_{i=1}^{n} x_i w_i + b$$

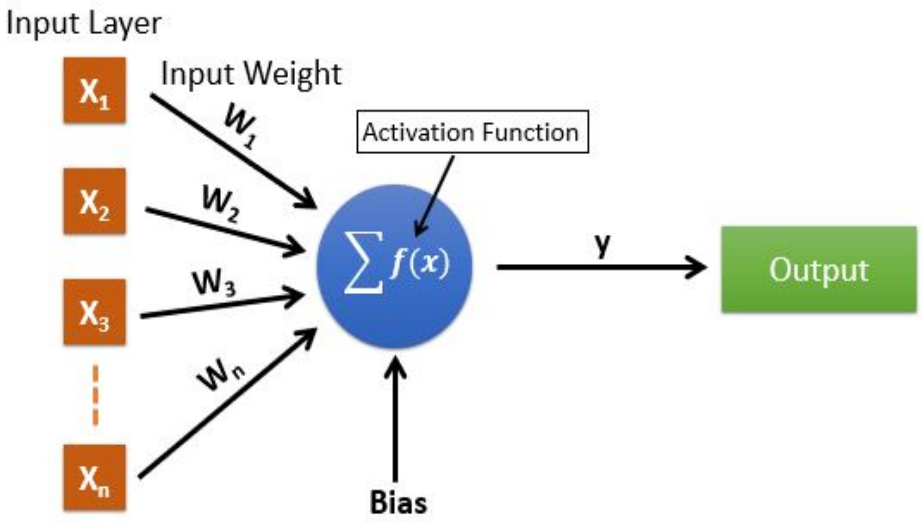


Figure 2.7: Basic artificial neuron diagram

2.6.3 Machine Learning - Artificial neural network (ANN) Model

As discuss in earlier section, the linear regression model are not good for fitting non-linear data. So, in this section we are going to perform a very common neural network method i.e., artificial neural network algorithm.

Firstly, the data id prepared similarly as for linear regression model given in table [2.1]. Where X will passing through input layer and will be trained based on the value of y . X and y are declared as :

```
X= data[IPD_x , IPD_y]

y =data[ [ Theta0, Theta5, Theta10, ... , Theta350, Theta355, Theta360 ] ]
```

Then passing the whole dataset through an ANN model containing 4 hidden dense layer. Dense layer means every neuron is connected with every other neuron from next layer. These four dense hidden layers have 50, 100, 100 and 80 nodes respectively and output layer consist of 73 output nodes consisting of 73 redial distance values and used relu (rectified linear activation function) activation function.

```
model = Sequential()


model.add(Dense(50,activation = 'relu'))
model.add(Dense(100,activation = 'relu'))
model.add(Dense(100,activation = 'relu'))
model.add(Dense(80,activation = 'relu'))


model.add(Dense(73))


model.compile(optimizer = 'rmsprop' , loss='mse')
```

The Model is compile using rmsprop optimizer and trying to minimize mean square error on every pass as shown in given below figure [2.8]

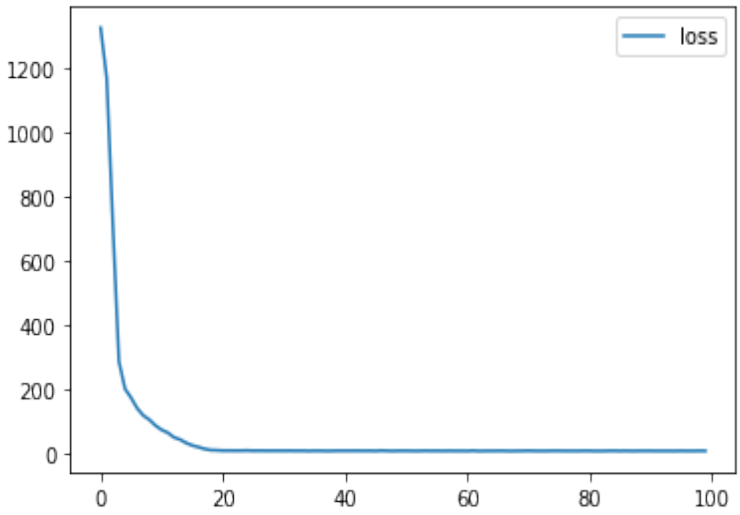


Figure 2.8: Change in error after every pass

Model validation and verification

In this step, We pass fresh data to our trained ANN model and compare the results with OpenFOAM output.

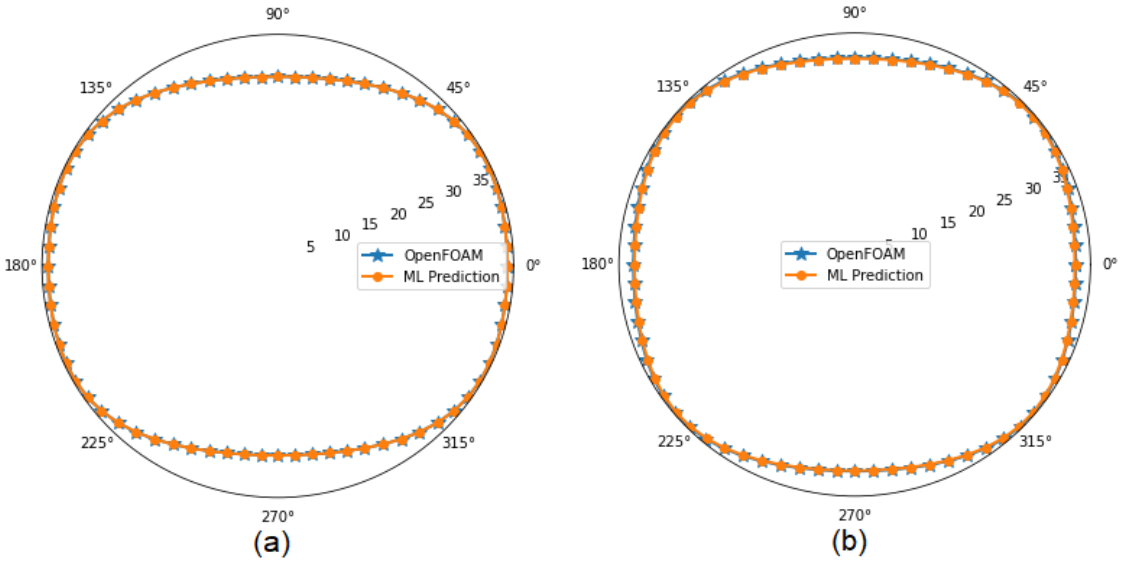


Figure 2.9: Model verification for ANN Model

The model performance can be evaluated be by:-

```
model.evaluate(X_test,y_test,verbose=0)
```

And this evaluate the model performance by calculating root mean square by using test data. The Root Mean Square of this comes out equal to 0.3360

The figure [2.9] show the comparison between the precipitate morphology predicted from by trained ANN model and the precipitate obtained from OpenFOAM. The figure [2.9(a)] show the case when input inter-particle distances are low as X = [65.5419 , 103.3368]. The predicted precipitate shape from model have bit elongated in the direction in which the inter-particle distance is low i.e., x direction. And adding to it, the shape predicted from the ANN model is overlapping to the precipitate obtained from OpenFOAM implying this model is good for solving such problem.

Figure [2.9(b)] shows the case where X = [127.3205 , 212.073200], As this can be observe that the shape accrues more square as we increase the inter-particle distance. And similar to first case the shape predicted from the ANN model is overlapping to the precipitate obtained from OpenFOAM.

Chapter 3

Prediction of precipitate morphologies for given elastic anisotropy with misfit using Neural Network

3.1 Background

Neural networks are the sub-field of machine learning. The name and structure of neural networks are inspired by our human brain and mimicking the way our biological neurons signal to one another. The fundamental difference between the Neural networks method and classical computer programming is that the Neural networks gather their knowledge by detecting the patterns and relationships in the data and learn (or trained) through trying all possible connection, not from programming unlike classical computer programmers.

Neural networks are formed from hundreds of single units, i.e. artificial neurons or processing elements, connected with coefficients (weights), constituting the neural structure and organising layers. The ability of neural computations comes from connecting neurons in a network. Therefore, the better the neurons are connected in networks, the better is the prediction as output.

Also, the phase-field method are effective methods for simulation approach for modelling microstructure evolution. Using the phase-field model for simulation the study of microstructural evolution in a wide variety of material processes, such as solidification, solid-state phase transformations, precipitate growth and coarsening,martensitic transformations and grain growth etc.

The goal of this paper is to provide a broad overview on the different aspects of coarsening of coherent precipitates.The chapter begins with a brief discussion on the different parameters (like Az and misfit) in materials which can affect the precipitate morphologies. And using these data building a Artificial neural network model.

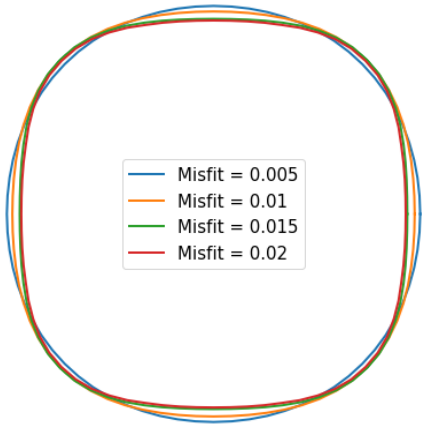
3.2 Research Work

The sole aim of this chapter is to study the effect of elastic anisotropy (A_Z) along with misfit strain (ϵ^*) and build ANN model based on data.

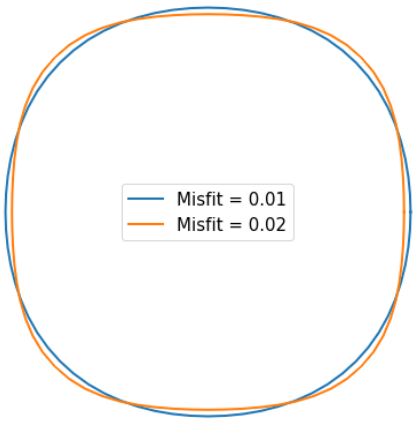
3.3 Outline

The sole aim of this chapter is to find the equilibrium precipitate shapes of coherent precipitates and more importantly the understanding of equilibrium morphology of precipitates as a function of elastic anisotropy A_Z along with misfit strain (ϵ^*). And building a artificial neural network to predict equilibrium precipitate shape for given elastic anisotropy and Misfit

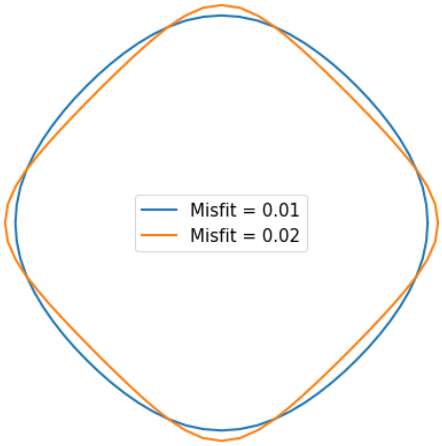
3.4 Effect of elastic anisotropy along with misfit on precipitate morphologies



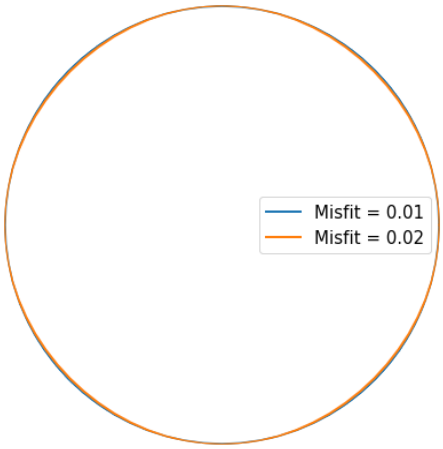
(a) Effect of misfit



(b) Az=3.0



(c) Az=0.4



(d) Az=1.0

Figure 3.1: Equilibrium shapes of precipitate as a function of Az with Misfit

3.5 Artificial Neural Networks (ANNs)

Neural Networks, also called multi-level perceptron. As the single perceptron cannot predict complex problems, we need to create a multi-level perceptron, which consists of multiple levels of an interconnected network of perceptrons. In this section, we will implement such a neural network, a feed-forward Artificial Neural Network.

3.5.1 Data Preparation

This step is concerned with transforming the raw data collected into a form used in modelling. For example, the data collected from OpenFOAM contained coordinates of points for equilibrium precipitate shape at ϕ= 0.5 and then shape transformed to radial distances at every 5°angle as shown in the figure [1.4]. Here, we prepared a input array X = [[A_Z, ϵ]] and in this section we have considered the elastic stress with dilatational misfit i.e. $\epsilon_{xx} = \epsilon_{yy}$ and denoted this misfit value as ϵ. The output array consist of all redial distances (at every 5^o angle) calculated from center of precipitate, as elaborated in section [1.6.3] and figure [1.4].Now, we tabulated our data as shown in table[3.1]

	Az	Misfit	Theta0	Theta5	Theta10	....	Theta350	Theta355	Theta360
1	0.4	0.005	20.1203	20.112	20.07019	....	20.07016	20.11627	20.1203
2	0.4	0.01	20.7068	20.65592	20.51841	....	20.51846	20.65626	20.7068
3	0.4	0.02	20.4651	20.38799	20.15938	....	20.15927	20.39639	20.4651
4	0.4	0.03	19.8427	19.8294	19.77606	....	19.7762	19.83001	19.8427
5	0.4	0.04	19.9415	19.91442	19.82519	....	19.82518	19.91428	19.9415
....	....	....	....	....	....	...	....	....	....
127	3	0.01	19.9631	19.95941	19.95456	....	19.95457	19.95943	19.9631
128	3	0.005	19.914	19.91325	19.91564	....	19.91562	19.91315	19.914
129	3	0.03	18.3923	18.45385	18.63757	....	18.63757	18.45384	18.3923
130	3	0.018	19.1092	19.15261	19.27874	....	19.27875	19.15163	19.1092
131	3	0.04	17.9818	18.04621	18.24037	....	18.24037	18.04623	17.9818

Table 3.1: Data Preparation for ANN model

3.5.2 Training and Testing Data

Here, the we split the data into training and testing sets.Setting a variable X equal to the numerical features of the data [A_Z, ϵ] and a variable y equal to the radial distance at every 5°angle. such as $y = [[theta0, theta5, theta10, theta15.....theta360]]$

`model_selection.train_test_split` from sklearn library is a great tool for splitting data in desired ratio for training and testing purpose.Here, we are setting `|test_size=0.25|`which divides total data into training and testing data in the ratio of 75% and 25% respectively . And we pass the input and output variable i.e., X and y respectively for training neural network model.

Firstly, we declare input variable data As X and output data as y which we are going to predict later using input variable X.

```
X= data[ A_Z, ϵ ]
y =data[ [ Theta0, Theta5, Theta10, ... , Theta350, Theta355, Theta360 ] ]
```

Splitting data for training and testing purpose.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)
```

Normalising data to a range of [0 , 1] using minmax scaler. which takes minimum value as 0 and maximum value as 1 and scale other data in between.

```
scaler = MinMaxScaler()
scaler.fit(X)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

In next step we build a model by adding 4 Dense Layers and then compile it by defining our solver.

```
model = Sequential()

model.add(Dense(50,activation = 'relu'))
model.add(Dense(100,activation = 'relu'))
model.add(Dense(100,activation = 'relu'))
model.add(Dense(80,activation = 'relu'))

model.add(Dense(73))

model.compile(optimizer = 'rmsprop' , loss='mse')
```

Dense layer means every neuron is connected with every other neuron from next layer. These four dense hidden layers have 50, 100, 100 and 80 nodes respectively and output layer consist of 73 output nodes consisting of 73 redial distance values and used relu activation function. The Model is compile using rmsprop optimizer and trying to minimize mean square error on every pass.

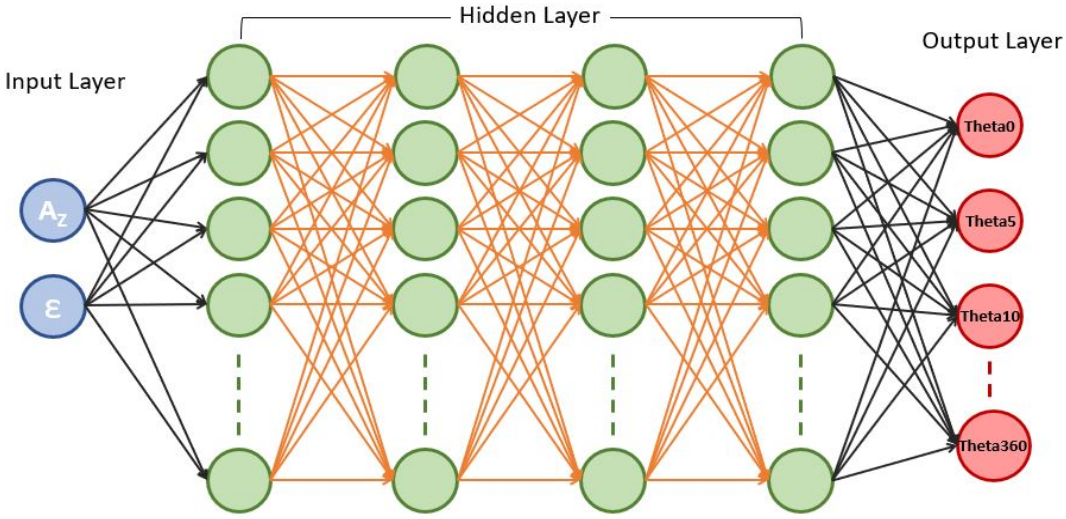


Figure 3.2: Basic artificial neuron diagram

ReLU Activation Function

The rectified linear activation function or ReLU for short, is the most used activation function and for the most cases it is the default activation function for ANNs. As it is shown in figure [3.3], the ReLU function is half rectified (from bottom). $f(z)$ is zero when z is less than zero (i.e., -ve value) and $f(z)$ is equal to z when z is greater than or equal to zero.

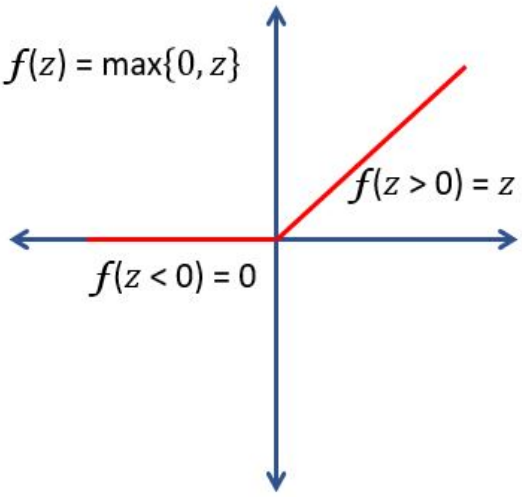


Figure 3.3: ReLU Activation Function

Other popular activation function :

· **Sigmoid :**

$$f(z) = \frac{1}{1 + e^{-z}}$$

· **Tanh :**

$$f(z) = \frac{(e^{z} - e^{-z})}{(e^{z} + e^{-z})}$$

Using the above model , we fit training data by splitting the training data in batch of 25 datasets every time. with epochs =5000 means we are passing the whole data 5000 to the neural network.

```
model.fit(x=X_train,y=y_train,epochs=5000,verbose=1, batch_size=25)
```

From the below figure [3.4], for the first pass model shows the Mean Square Error (mse) of 173 and on every pass it is optimising the error and also as we can observe the error till 2000 passes is decreasing continuously and become constant after that. The Mean Square Error (mse) after 5000 passes shows an error of 0.0057.

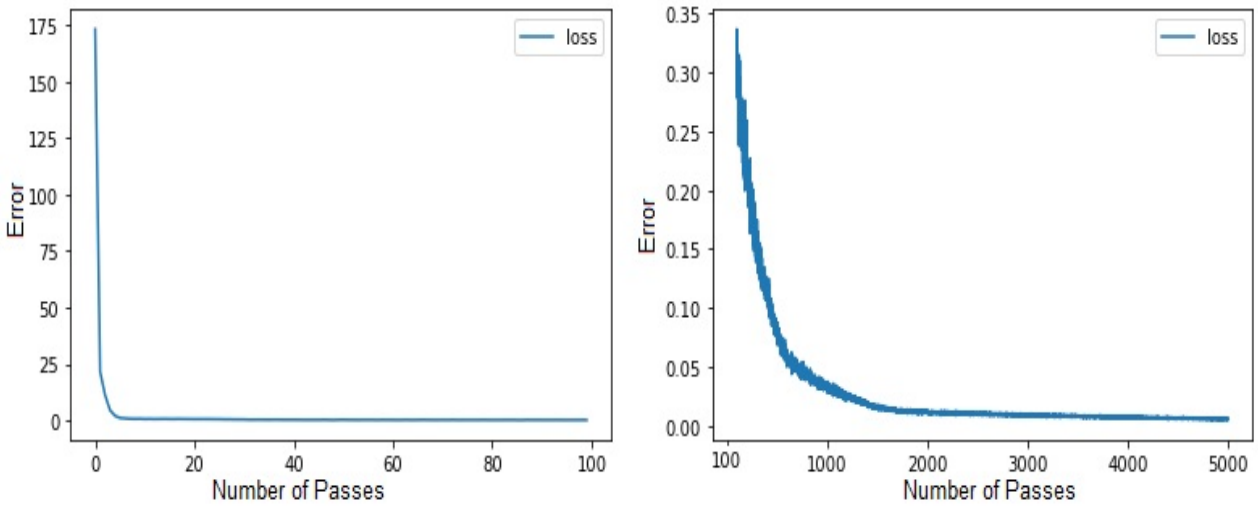


Figure 3.4: Variation in mse after every pass

3.5.3 Model validation and verification

Model evaluation is an essential step to determine whether the trained model is performing good or not. Let's recall the section [3.6.3.2], where we separated test data from main data to provide unseen data to our trained model to determine its fitting. The sklern library have build in function to evaluate model performance by passing unseen data-set as :

```
model.evaluate(X_test,y_test,verbose=0)
```

The Root Mean Square of this comes out equal to 0.00616.

Model Verification

The below figure [3.5] shows that the compression of equilibrium precipitate resulting from OpenFOAM and Artificial neural network model for cases where (a) Az = 1.9 with misfit =

0.028 and (b) Az = 0.45 with misfit = 0.018. The equilibrium precipitate shape from both the cases perfectly overlap to each other expressing the good predictably of our model.

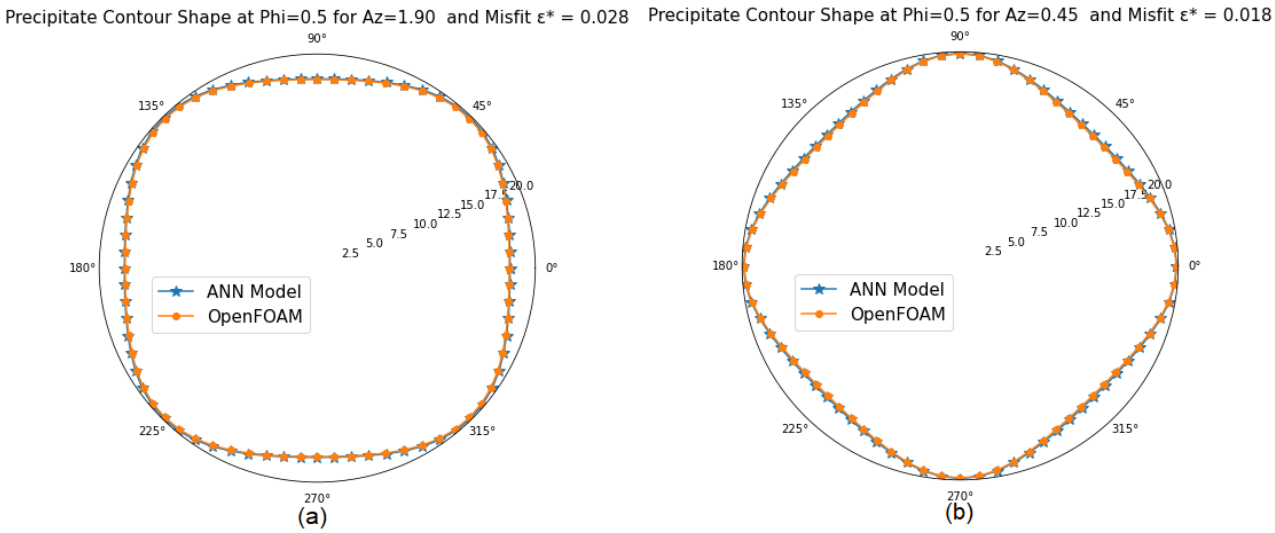


Figure 3.5: ANN Model Verification

3.5.4 The Limitations of Machine Learning

As in previous section, the ANN model was trained for predicting equilibrium shape for given two input variable i.e., X = [Az , ϵ], which is done by training 131 quality data. Next, the aim was to add more input parameter which affects the precipitate morphology but on addition of one more input variable leads to increase the need of more than 500 structured data which is 4 to 5 times of the previous data. and on adding more and more input variable leads to creation of thousands of data and this is a very laborious and time taken process. And presence of missing data always leads to under-fit the model.

As discussed, each independent variable which affect the precipitate morphology needs to be specially trained and eventually require large amounts of hand-crafted, structured training data. Hence we move onto a different approach to predict precipitate morphology as discussed in next chapter [4].

3.6 Prediction of eigenstrain (Misfit) value for a given precipitate Shape using ANN Model

Earlier we have created a model by training Az and ϵ^* i.e., elastic anisotropy and eigenstrain i.e., misfit respectively with redial distances of equilibrium precipitate shape. And using model created earlier we can easily predict the equilibrium precipitate shape for a given value of Az and ϵ^*.

Now for this project, the aim is to create a python program which uses the earlier trained ML model to predict value of the misfit at constant Az for a given equilibrium precipitate shape.

3.6.1 Numerical Approach

We initially take two guess (X_1, X_2) for ϵ i.e, Eigenstrain (misfit) and using the earlier machine learning model, we try to predict precipitate shape corresponding to those initial guess and report the error(e_1, e_2) obtained from guess and input precipitate shape such as.

$$e_1 = (y_{x_1} - y_{input})$$

$$e_2 = (y_{x_2} - y_{input})$$

3rd guess is generated using the line obtained from point $(X_1, e_1), (X_2, e_2)$ (where y-axis is error and x-axis is surface anisotropy) , by finding intercept point corresponding to error =0.

$$\text{error} = m \times \text{3rd guess} + C$$

$$\text{Where , } m = \frac{e_2 - e_1}{X_2 - X_1}$$

And again from these 3 points (X_3, X_2, X_1), it choose 2 points which have least error and again using those 2 points it extrapolate 4th point. and so on until the difference in values

reaches a tolerance of 10^{-6} as shown in figure [3.6]

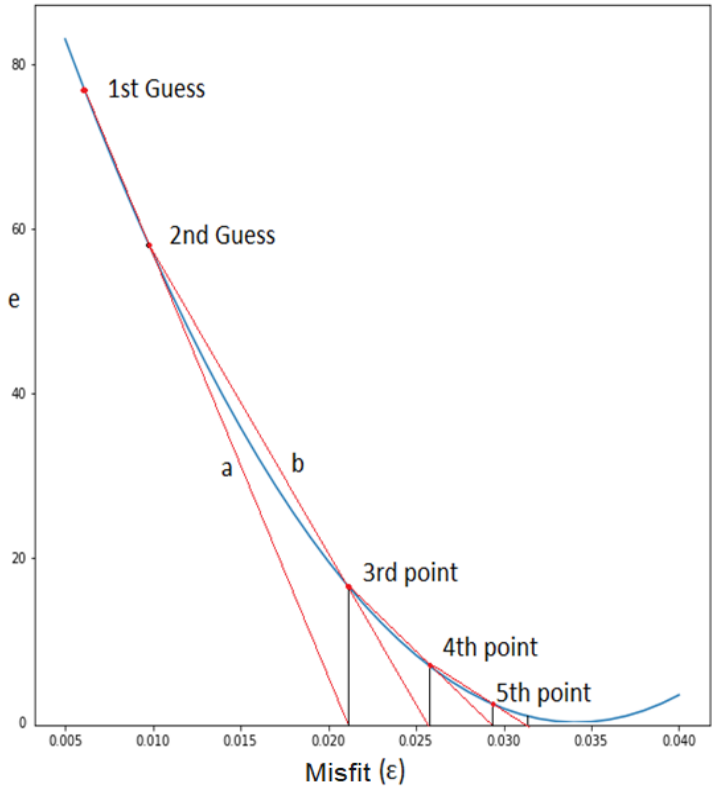


Figure 3.6: Plot between Error and Misfit value

3.6.2 Implementation Code

```
def get_E(x,X,Y):
    dferror = pd.DataFrame({'X' : X , 'Y' : Y})
    dferror = dferror.sort_values(by = ['X'], ignore_index=True)
    x_points = dferror['X']
    y_points = dferror['Y']
    tck = interpolate.splrep(x_points , y_points , k=1)
    return interpolate.splev(x, tck)

def square_error(X,Y):
    SE =np.sum(((X - Y)**2))
    return SE
```

```
guess1 = input("First guess: ")
guess2 = input("Second guess: ")
diff =1.0
while diff > 0.00001 :

    left = square_error(lm.predict([[float(guess1)]]),df2[0][3:])
    right = square_error(lm.predict([[float(guess2)]]),df2[0][3:])
    vdata = get_E(0.075,[right , left],[ guess2 ,guess1 ]) #error taken = 0.075,
    which is approximate error in ML model
    guess1 = guess2
    guess2 = vdata
    diff = abs(float(guess2) - float(guess1))

print('Final Ans : ' + str(guess2))
Perror=(abs(0.033-guess2)/0.033)*100
print('% Error = ' + str(Perror))
```

3.6.3 Validation

For validation purpose, we taken a random input of equilibrium precipitate where value of misfit strain $\epsilon = 0.033$. And initialise the program by taking initial guesses 0.01 and0.017 (these values can be anything, More closer your guess more faster will be the result).After successful run, the program converses ϵ to a value of 0.03327 with a square error =0.82

Chapter 4

Alternate approach to study equilibrium precipitate shape using PyMKS

4.1 Background

The Materials Knowledge Systems in Python project (PyMKS) is an open-source materials data science framework that can be used to produce high-fidelity, reduced-order (i.e., low computational cost) and, process–structure-property (PSP) linkages for a broad range of material systems having a rich hierarchy of internal structures spanning multiple length scales.[8] [9] The PyMKS framework supports an emergent community at the intersection of materials science and engineering, manufacturing, machine learning, and data science. The framework explicitly addresses the customized analytics needed to account for the stochastic nature of the complex internal structure of materials at multiple length scales to extract high-value knowledge databases that allow high-fidelity explorations of substantial materials design spaces. The formulations are broadly classified as either homogenization (i.e., determination of the effective material response at the more prominent length scale given the material structure information

at the lower length scale) or localization (i.e., spatially resolving an imposed quantity at the more prominent length scale into a field defined at the lower length scale).
The purpose of this chapter is to explore the different tools available on PyMKS and using these tools finding a different approach to compute equilibrium precipitate shape during coarsening.

4.2 Research Work

The goal of this chapter is to find the equilibrium precipitate shapes of coherent precipitates using different approach. Using PyMKS tool to find equilibrium precipitate shape instead of OpenFOAM.

4.3 Outline

In the following section, we are using PyMKS Machine Learning tool to train phi (order parameter) field and driving force corresponding to the phi field. And from the trained model, generating influence coefficients and finally using those influence coefficients we numerically compute the final equilibrium precipitate shape.

4.4 PyMKS (Python-Based Materials Knowledge Systems)

PyMKS is an object-oriented numerical implementation of the MKS theory developed. The PyMKS framework is the set of machine learning tools for constructing process-structure-property linkages models for materials science applications. It provides a high-level, computationally high efficient framework to implement data pipelines for classification, cataloguing, and quantifying materials structures for PSP relationships as shown in figure [4.1].

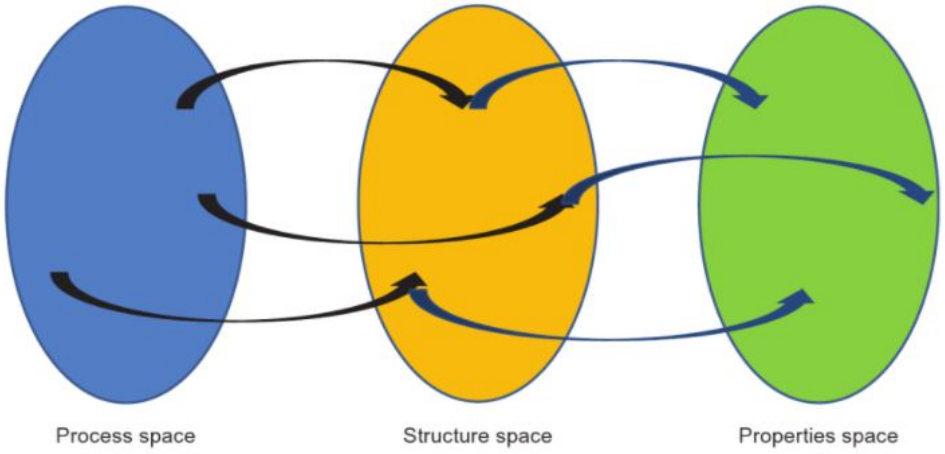


Figure 4.1: PSP Linkages in PyMKS [11]

The two principle objects that provided by PyMKS are the TwoPointCorrelation transformer and the LocalizationRegressor which provide the homogenization (scaling up) and localization (scaling down) functionality. The objects provided by PyMKS all work as either transformers or regressors in a Scikit-Learn pipeline and use both Numpy and Dask arrays for out-of-memory, distributed or parallel computations.

2-Point Spatial Correlations

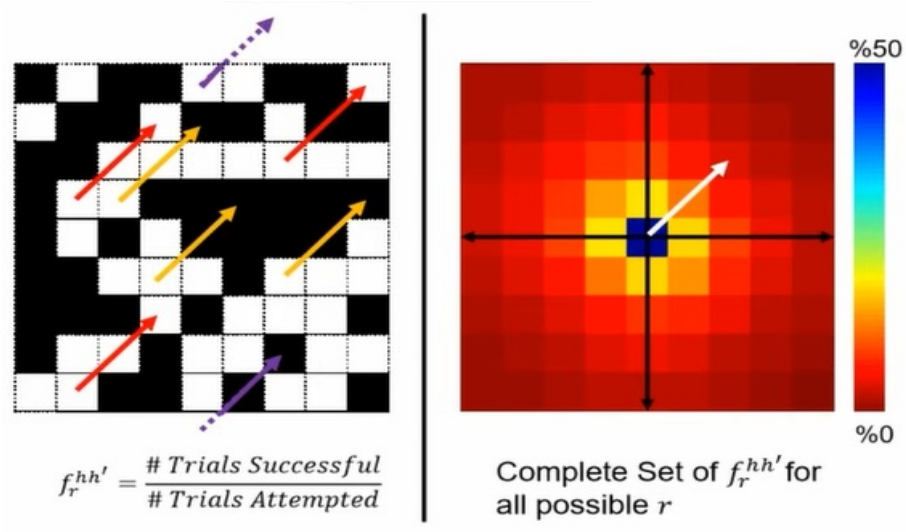


Figure 4.2: 2-Point Spatial Correlations [11]

2-point spatial correlations, it contain information about the fractions of local states and the information about how the different local states are distributed in the microstructure.

2-point statistics can be thought of as the probability of having a vector placed randomly in the microstructure and having one end of the vector be on one specified local state and the other end on another specified local state as shown in figure [4.2]. This vector could have any length or orientation that the discrete microstructure allows. The equation for 2-point statistics can found below.[11]

$$f[r|l,l'] = \frac{1}{S}\sum_{s} m[s,l]m[s+r,l']$$

In this equation $f[rl,l']$ is the conditional probability of finding the local states l and l' at a distance and orientation away from each other defined by the vector r.$m[s,l]$ is the microstructure function (the digital representation of the microstructure), S is the total number of spatial bins in the microstructure and s refers to a specific spatial bin.

4.4.1 Model Creation

In this chapter, the aim is to build such relation between the phi field (where phi = 0 is for matrix phase and phi = 1 for precipitate phase) and the driving force field generated due to misfit. We used PyMKS two basic principle objects TwoPointCorrelation transformer and the LocalizationRegressor. The first step is to create a linkage between the phi field which will be containing the information about precipitate shape and the corresponding driving force. The calibrated influence coefficient can be save to any readable file for the next process. The second step is to use the influence coefficient to predict the corresponding driving force for any given phi field. And further by using the numerical methods to compute the equilibrium precipitate shape.

During coarsening (solid-state phase transformations), there can be a difference in the lattice parameter between the precipitate and the matrix. And these difference in the lattice parameter gives rise to misfit strains/stresses for a coherent interface and these generate elastic stress in the system. So in this project we trying to relate i.e., training our phi field to the elastic driving field. as given in figure [4.3].

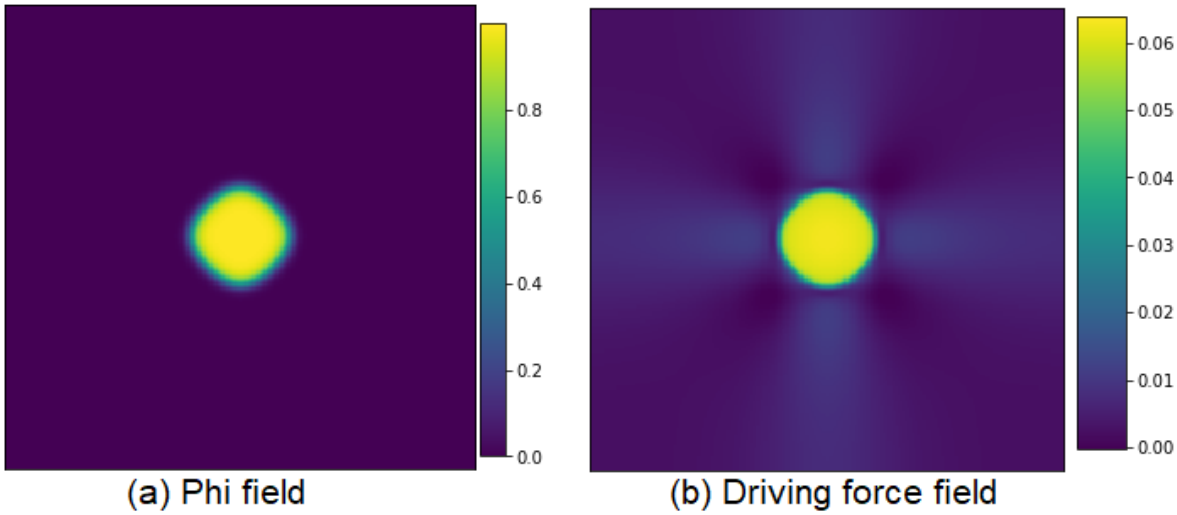


Figure 4.3: Phi and driving force field

As we have discussed earlier, We created a regression model in a Scikit-Learn pipeline pipeline model and using LocalizationRegressor(). Here, the PyMKS provides the TwoPoint-Correlation transformer and the LocalizationRegressor() which provide the homogenization and localization functionality.

Here we have only 2 discrete phases i.e, matrix and precipitate, and we will use the Primitive-Transformer from pymks. We only have two phases denoted by 0 and 1, therefore we have two local states and our domain is 0 to 1. next, we train the phi field with driving force field as shown in figure [4.3]

```
#Creating regression model using pipeline tool from Sci-Kit Learn
model = Pipeline(steps=[
    ('reshape', ReshapeTransformer(shape=X.shape)),
    ('discretize', PrimitiveTransformer( min_=0, max_=1.0)),
    ('regressor', LocalizationRegressor())
])

#Setting n_state
model.set_params(discretize__n_state=2)

#Spliiting dataset for traning and testing purpose
x_train, x_test, y_train, y_test = train_test_split(
    X.reshape(X.shape[0], -1),
    Y.reshape(Y.shape[0], -1),
    test_size=0.15
)

#Model Training
model.fit(x_train, y_train);
```

After successfully creating a trained PyMKS model. It allows to predict driving force for any precipitate state or shape given i.e., phi field. Next for our model validation, we took a random precipitate shape and pass it through the trained model created in previous section. And As an observation it was found that the driving force obtained from the PyMKS Model and the OpenFOAM are approx same with an error of order 10^{-2}.

As the figure [4.4] shows the driving force obtained from the OpenFOAM and the PyMKS

Model and figure [4.5] show the error in the prediction of driving force using PyMKS compare to the field obtained from OpenFOAM.

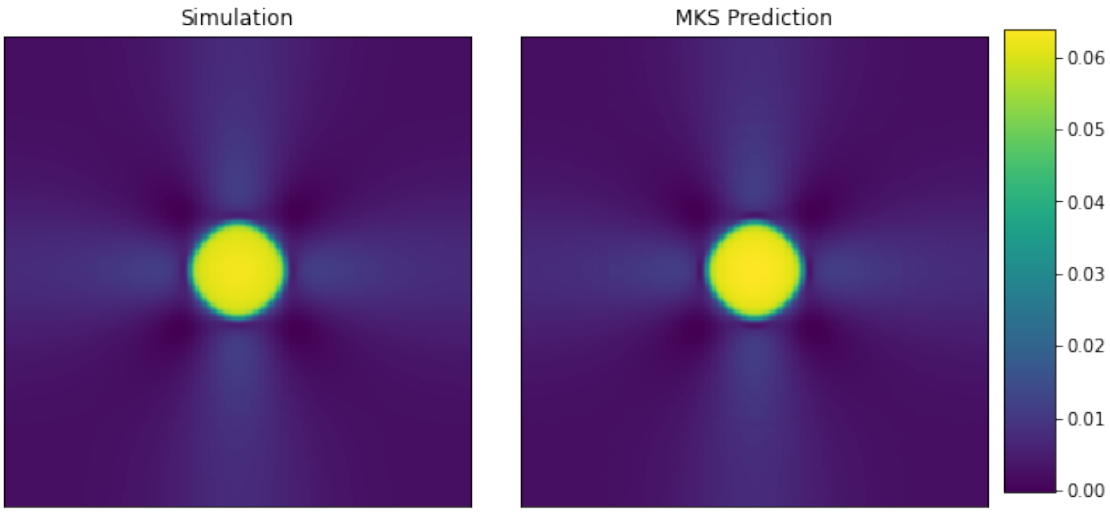


Figure 4.4: Driving force field obtained from OpenFOAM and PyMKS

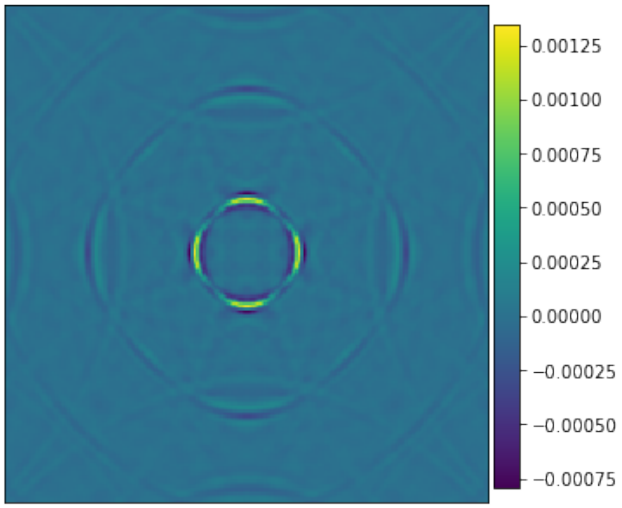


Figure 4.5: Error in driving force field obtained from OpenFOAM and PyMKS

4.4.2 Influence Coefficients

Now that we have the phi field and corresponding driving force field, we will calibrate the influence coefficients by the instance of LocalizationRegressor(). As we are going to calibrate the influence coefficients with phi field, we can create an model with n_states equal to 2, and use it to create an instance of LocalizationRegressor model .Next, the phi field and driving force field will then be passed to the fit method.

```
#Calibrating influence coefficients
model.set_params(discretize__n_state=2)


#Model Training
model.fit(x_train, y_train);
```

The calibrated influence coefficients were save in an txt file for using it later for prediction of driving force.

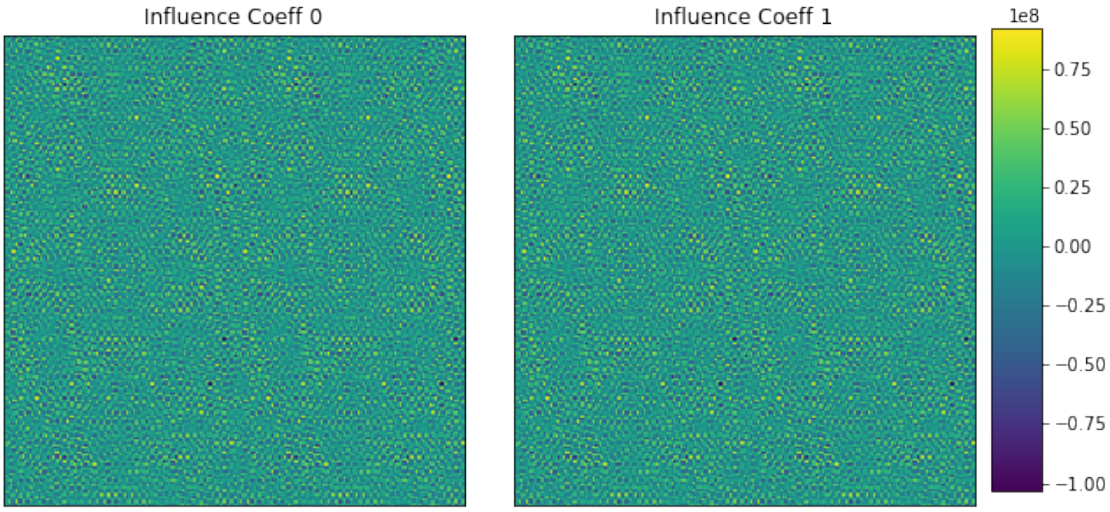


Figure 4.6: Influence Coefficients

4.4.3 Implementation Code

Let's recall the Allen Cahn dynamics evolution equation for ϕ as discuss in section [1.4] in equation (1.4)

$$\tau W \frac{\partial \phi}{\partial t} = 2\gamma W \nabla \cdot \left[a(n) \left(\frac{\partial a(n)}{\partial \nabla \phi} |\nabla \phi|^2 + a(n) \nabla \phi \right) \right] - \frac{d\omega(\phi)}{d\phi} - \frac{f_{el}(u,\phi)}{\partial \phi} - \lambda_\beta h'(\phi)$$

Here, the elastic driving force $dF = \frac{f_{el}(u,\phi)}{\partial \phi}$ can be calculated from the above calibrated influence coefficients as [9]-

$$dF[i,j] = \sum_{p=0}^{Nx} \sum_{q=0}^{Ny} \text{coeff0}[p,q]\phi[i-p, j-q] + \text{coeff1}[p,q](1 - \phi[i-p, j-q]) \tag{1}$$

Where ϕ is he input field and coeff0 and coeff1 are the influence coefficient. and he we solve the above equation by discretizing using finite difference method. The implemented code the is given below.

```
#include<stdio.h>
#include<math.h>
#define Nx 200
#define Ny 200
#define dt 0.1
#define dx 1.0
#define epsilon 4.0
#define gamma 0.15
#define tau 0.28
#define A ((9.0*gamma)/epsilon)
#define L (1.0/(tau*epsilon))
#define kappa (gamma*epsilon)
#define radius 20
double phi[Nx][Ny];
double hphi[Nx][Ny];
```

```
16 double gphi[Nx][Ny];
17 double dF[Nx][Ny];
18 double correction;
19 double sum_func(double phi[Nx][Ny]);
20 void hfunc(double phi[Nx][Ny] , double hphi[Nx][Ny]);
21 void gfunc(double phi[Nx][Ny] , double gphi[Nx][Ny]);
22 void update_phi(double phi[Nx][Ny] , double gphi[Nx][Ny] ,double dF[Nx][Ny]);
23 void correct_phi(double phi[Nx][Ny] , double correction);
24
25 int main()
26 {
27   FILE *fp;
28   long x, y;
29   long t_x, t_y;
30   int i;
31   long k_x, k_y;
32   int timestep = 1500;
33   double coeff1[Nx][Ny];
34   double coeff2[Nx][Ny];
35   double sum_old=0.0;
36   double sum_new=0.0;
37   double deltaphi;
38   double hphi_sum;
39    fp  = fopen("coeff_1.txt","r");
40     for (x=0; x<Nx; x++) {
41      for(y=0;y<Ny;y++) {
42       fscanf(fp, "%le  ",&coeff1[x][y]);
43     }
44    }
45   fclose(fp);
46    fp  = fopen("coeff_2.txt","r");
```

```
47    for (x=0; x<Nx; x++) {
48      for(y=0;y<Ny;y++) {
49        fscanf(fp, "%le  ",&coeff2[x][y]);
50      }
51    }
52   fclose(fp);
53   for (x=0; x < Nx; x++) {
54     for (y=0; y < Ny; y++) {
55       if((x-Nx/2)*(x-Nx/2) + (y-Ny/2)*(y-Ny/2) < radius*radius) {
56
57         phi[x][y] = 1.0;
58       } else {
59         phi[x][y] = 0.0;
60       }
61     }
62   }
63 sum_old = sum_func(phi);
64 for(i=0;i<timestep;i++){
65
66     gfunc(phi,gphi);
67     for(x=0;x<Nx;x++){
68       for(y=0;y<Nx;y++){
69
70         long x_index = (Nx/2 + x)%Nx;
71         long y_index = (Ny/2 + y)%Ny;
72         dF[x_index][y_index] = 0.0;
73
74         for(t_x=0;t_x<Nx;t_x++){
75           for(t_y=0;t_y<Nx;t_y++){
76
77             k_x = ((x-t_x)+Nx)%(Nx);
```

```
78              k_y = ((y-t_y)+Nx)%(Nx);
79              dF[x_index][y_index] += coeff1[t_x][t_y]*(1-phi[k_x][k_y]) + coeff2[
    t_x][t_y]*(phi[k_x][k_y]);
80            }
81           }
82        }
83     }
84     update_phi(phi,gphi,dF);
85     sum_new = sum_func(phi);
86     deltaphi =  sum_new - sum_old ;
87     hfunc(phi,hphi);
88     hphi_sum= sum_func(hphi);
89     correction = deltaphi/hphi_sum;
90     //correcting Phi
91     for(x=0; x<Nx; x++) {
92       for (y=0; y < Ny; y++) {
93          phi[x][y] -= hphi[x][y]*correction;
94       }
95     }
96     sum_new = sum_func(phi);
97     sum_old = sum_new;
98 }//time loop ends here
99
100 //saving phi value to txt
101 fp = fopen("phi_final.txt","w");
102  for(x=0;x<Nx;x++){
103   for(y=0;y<Ny;y++){
104     fprintf(fp,"%lf\n",  phi[x][y]);
105   }
106  }
107  fclose(fp);
```

```
108
109 return 0;
110 }//closing main loop
111 double sum_func(double phi[Nx][Ny] ) {
112   int x,y;
113    double sum= 0;
114    for(x=0; x<Nx; x++) {
115     for (y=0; y < Ny; y++) {
116       sum += phi[x][y];
117     }
118   }
119   return sum;
120 }
121
122 void hfunc(double phi[Nx][Ny] , double hphi[Nx][Ny]) {
123   int x,y;
124    for(x=0; x<Nx; x++) {
125     for (y=0; y < Ny; y++) {
126       hphi[x][y] = 6.0*phi[x][y]*(1.0-phi[x][y]);
127     }
128   }
129 }
130
131 void gfunc(double phi[Nx][Ny] , double gphi[Nx][Ny]) {
132   int x,y;
133    for(x=0; x<Nx; x++) {
134     for (y=0; y < Ny; y++) {
135       gphi[x][y] = 2.0*A*phi[x][y]*(1.0 - phi[x][y])*(1.0 - 2.0*phi[x][y]);
136     }
137   }
138 }
```

```
139 void update_phi(double phi[Nx][Ny] ,double gphi[Nx][Ny] ,double dF[Nx][Ny]) {//
140   long x, y, i;
141     double phi_new[Nx][Ny];
142 //Fixed Boundary Condition
143 for(i=0; i<Nx; i++) {
144         phi[0][i] = 0.0;
145         phi[Nx-1][i] = 0.0;
146         phi[i][0] = 0.0;
147         phi[i][Nx-1] = 0.0;
148     }
149     for(x=1; x<Nx-1; x++) {
150      for (y=1; y < Ny-1; y++) {
151         phi_new[x][y] = phi[x][y] +( ((2*L*kappa*dt)*(phi[x][y+1] + phi[x][y-1]
    + phi[x-1][y] + phi[x+1][y] - 4*phi[x][y] ))/(dx*dx) ) -  L*dt*gphi[x][y] -
    1.0*L*dt*dF[x][y]*3.0*phi[x][y]*(1.0-phi[x][y]);
152     }
153   }
154    for(x=0; x<Nx; x++) {
155      for (y=0; y < Ny; y++) {
156         phi[x][y] = phi_new[x][y];
157     }
158   }
159 }
160 void correct_phi(double phi[Nx][Ny] ,  double correction) {
161   long x, y;
162    for(x=0; x<Nx; x++) {
163     for (y=0; y < Ny; y++) {
164       phi[x][y] -= 6.0*phi[x][y]*(1.0-phi[x][y])*correction;
165     }
166   }
167 }
```

4.4.4 Model Verification

As in the previous section, we use PyMKS to calibrated influence coefficients and then using those calibrated influence coefficients to predict the value of dF i.e., driving force . And evolving the ϕ field by Allen Cahn equation as given in equation [4.1].

The figure below [4.7] show that the comparison of equilibrium precipitate shape obtain from numerically solving the above equation[4.1] and using the predicted dF (shown by Orange line) and shape obtained from OpenFOAM (shown by blue line). The figure shows the equilibrium precipitate shape for Az = 0.45 and misfit = 0.01. The equilibrium precipitate shape obtained from both methods are very similar with insignificant error at the corner of the precipitate arises due to error during predicting dF value.

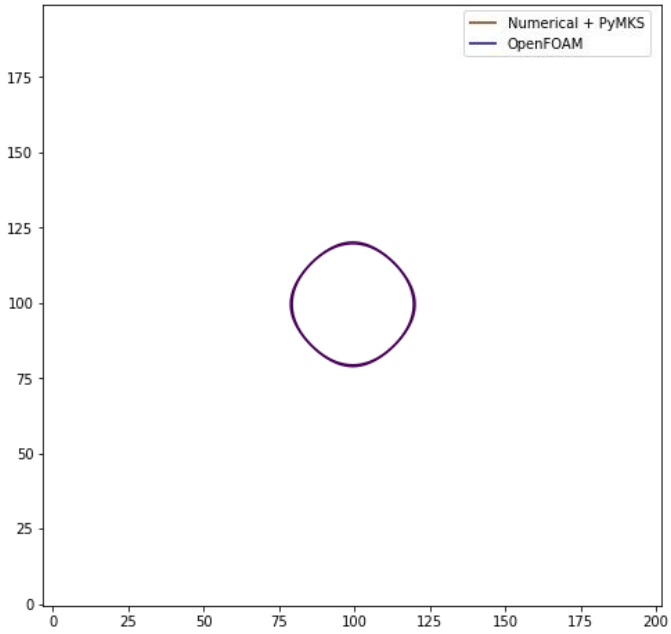


Figure 4.7: Comparison of precipitate morphology obtained from OpenFOAM and PyMKS

4.5 Comparison of PyMKS with other Methods based on execution time

In earlier section, we observed the precipitate shape obtained from PyMKS and OpenFOAM were same which can be seen in figure [4.7] and therefore it can be concluded that this method (numerical + PyMKS) is efficient to determine the equilibrium precipitate shape for any given parameter. Next, we determine the time taken to reach equilibrium for every process as given below -

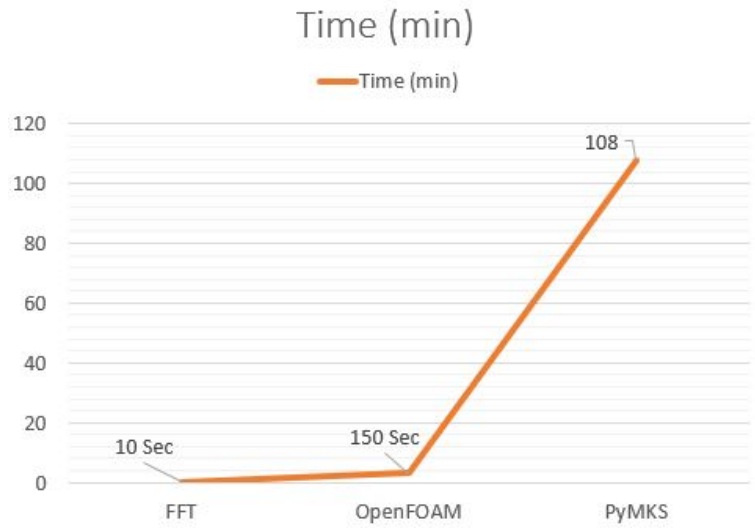


Figure 4.8: Time taken to achieve equilibrium for FFT, OpenFOAM and PyMKS

The figure 4.8 shows that the time taken for the FFT code is significantly less, i.e., 10 sec, and OpenFOAM takes 150 sec. In contrast, the PyMKS takes 108 minutes, which is much greater than the other methods, and this arises because of the high complexity of the program. And this concludes that numerical methods are still the best way to solve PDE problems in comparison to other machine learning methods.

On the other hand, despite long processing time, this method can be helpful for the case when replacing the term, which is much higher complex to compute by numerical method.

Chapter 5

Conclusions

In this thesis we have the studied the effect various parameter which affect the precipitate morphology during coarsening and build the machine learning model for the same.

Chapter 1 discuss about the effect of anisotropy in elastic energy and surface energy. We observe that for the for case where Az < 1 precipitate acquires a diamond shape, Az > 1 precipitate acquires the cubic shape. And on increasing surface anisotropy, precipitate become more and more diamond shape. For the machine learning part, we used linear regression model to fit data. The ML model have very good R2-Score $= 0.97$ shows that model have good fitted. The precipitate obtained from machine learning and OpenFOAM perfectly overlap to each other shows good predictability of our model.

Chapter 2 , In this chapter we studied the effect of inter-particle distance on precipitate morphology. Here, we observe that the precipitate gets elongated the direction where inter-particle distance was very less because of the interaction of elastic field and as we increase the inter-particle distance precipitate acquires a shape having aspect ratio is more closer to one. And after a limit we don't see the effect of inter-particle spacing as elastic field interaction

is not strong enough. The linear regression model fails to predict the precipitate shape because of the non-linearity of data. And hence we build ANN model and the ANN model shows a good fitting and the precipitate obtained from ANN model and OpenFOAM found to be same

Chapter 3, This chapter discuss about the effect of elastic anisotropy with misfit. and lastly we build a Artificial Neural Network model to predict the shape for a given elastic anisotropy and misfit value. As we already discussed the effect of Az in earlier section. On increase misfit, the effect of elastic stress increase means shape acquires more and more cubic shape for Az < 1 and acquires more and more diamond shape for Az > 1. The ANN model build in this section also have good predictability with RMSE $= 0.051$. And for every test case predicted shape perfectly overlap to the shape obtain from OpenFOAM. And adding more independent variable which affect the precipitate morphology needs to be specially trained and eventually require large amounts of hand-crafted, structured training data.Hence we move onto a different approach to predict precipitate morphology.

Chapter 4 , Here we implemented a different approach to study the equilibrium precipitate shape. Using PyMKS tool to predict driving force for a given ϕ field unlike earlier method where we were training independent parameter with precipitate shape then numerically solving the allen cahn evolution equation by finite difference method to find equilibrium precipitate shape equilibrium precipitate shape obtained from both methods are very similar but the time taken to achieve equilibrium was much higher comparison to other methods (FFT and OpenFOAM) because of high complexity of program computing predicting the dF value. However, This method can still be useful for when replacing the term which is much higher complex to compute by numerical method. Lastly, this concludes that the numerical methods are still the best way to solve PDE problems in comparison to other machine learning methods which requires more of the data as we increase number of independent input variable.

Appendices

Appendix A

Implementation in OpenFOAM

```
#include "fvCFD.H"
#include "Switch.H"

int main(int argc, char *argv[])
{
    #include "postProcess.H"
    #include "setRootCase.H"
    #include "createTime.H"
    #include "createMesh.H"
    #include "createControls.H"
    #include "createFields.H"


    Info<< "\nCalculating displacement field\n" << endl;

    while (runTime.loop())
    {
        Info<< "Iteration: " << runTime.value() << nl << endl;

        #include "readSolidDisplacementFoamControls.H"
```

```
21
22      int iCorr = 0;
23      scalar initialResidual = 0;
24
25      forAll(phi, cellI)
26          {
27              phiOld[cellI]=phi[cellI];
28          }
29
30          gradD = fvc::grad(D);
31          #include "phiEqn.H"
32          do
33          {
34          {
35              fvVectorMatrix DEqn
36              (
37                  fvm::d2dt2(D)
38               ==
39                  sig1*fvm::laplacian(2*(mu1*phi*phi*(3-2*phi) + mu2*(1-phi)
    *(1-phi)*(1+2*phi))
40      + lambda1*phi*phi*(3-2*phi) + lambda2*(1-phi)*(1-phi)*(1+2*phi), D, "
    laplacian(DD,D)")
41                  + (sig1/sig2)*divSigmaExp
42    - (sig1)*fvc::div((2*mu1*phi*phi*(3-2*phi) + 2*mu2*(1-phi)*(1-phi)*(1+2*
    phi))*phi*phi*(3-2*phi)*cEigenStrain
43                   +(lambda1*phi*phi*(3-2*phi) + (1-phi)*(1-phi)*(1+2*phi)*
    lambda2)*I*tr(phi*phi*(3-2*phi)*cEigenStrain))
44              );
45
46              initialResidual = DEqn.solve().max().initialResidual();
47
```

```
48                  if (!compactNormalStress)
49                  {
50                      divSigmaExp = fvc::div(DEqn.flux());
51                  }
52              }
53
54              {
55
56                  gradD = fvc::grad(D);
57
58                    strain=((gradD-phi*phi*(3-2*phi)*cEigenStrain)&&symmTensor
    (1,0,0,0,0,0))*symmTensor(1,0,0,0,0,0)
59      +((gradD-phi*phi*(3-2*phi)*cEigenStrain)&&symmTensor(0,0,0,1,0,0))*
    symmTensor(0,0,0,1,0,0)
60      +((gradD-phi*phi*(3-2*phi)*cEigenStrain)&&symmTensor(0,0,0,0,0,1))*
    symmTensor(0,0,0,0,0,1);
61
62
63                  sigmaD = (mu1*phi*phi*(3-2*phi) + mu2*(1-phi)*(1-phi)*(1+2*phi))
    *twoSymm(gradD)
64      + (lambda1*phi*phi*(3-2*phi)     + lambda2*(1-phi)*(1-phi)*(1+2*phi))*(I*tr(
    gradD))
65                  + (mu1_*phi*phi*(3-2*phi) + mu2_*(1-phi)*(1-phi)*(1+2*phi))*
    strain;
66
67                  if (compactNormalStress)
68                  {
69                      divSigmaExp = sig2*fvc::div
70                      (
71                          sigmaD - (2*mu1*phi*phi*(3-2*phi) + 2*mu2*(1-phi)*(1-phi
    )*(1+2*phi)
```

```
72      +lambda1*phi*phi*(3-2*phi) + (1-phi)*(1-phi)*(1+2*phi)*lambda2)*gradD,
73                          "div(sigmaD)"
74                      );
75                  }
76                  else
77                  {
78                      divSigmaExp += sig2*fvc::div(sigmaD);
79                  }
80              }
81
82          } while (initialResidual > convergenceTolerance && ++iCorr < nCorr);
83
84
85          #include "calculateStress.H"
86
87          Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
88              << "  ClockTime = " << runTime.elapsedClockTime() << " s"
89              << nl << endl;
90      dimensionedScalar totalEnergy = 0.0;
91      dimensionedScalar elasticEnergy = 0.0;
92      dimensionedScalar surfaceEnergy = 0.0;
93      volScalarField consta(0.5*(Sigma && (symm(gradD)-phi*phi*(3-2*phi)*
    cEigenStrain)));//symm(fvc::grad(D));
94      volVectorField gradT(fvc::grad(phi));
95      volScalarField constb(2.0*Gamma*Epsilon*(magSqr(gradT)));
96      forAll(consta, cellI) {
97          elasticEnergy += 0.5*consta[cellI];
98          surfaceEnergy += 2.0*Gamma*Epsilon*(magSqr(gradT[cellI]))*4.0;
99  totalEnergy    += elasticEnergy + surfaceEnergy;
100      }
101     dimensionedScalar surfaceEnergyGsum = gSum(constb());
```

```
102     dimensionedScalar elasticEnergyGsum = gSum(consta());
103     dimensionedScalar totalEnergyGsum   = surfaceEnergyGsum + elasticEnergyGsum;
104
105  Info<< "elasticEnergyGsum: " << (elasticEnergyGsum) << endl;
106  Info<< "surfaceEnergyGsum: " << (surfaceEnergyGsum) << endl;
107  Info<< "totalEnergyGsum: " << (totalEnergyGsum) << endl;
108
109  min((0.5*(deltaSigmaD && (symm(fvc::grad(D))-T*T*(3-2*T)*cEigenStrain))-(Sigma
       && cEigenStrain))()).value() << ' '
110
111 //for total energy calculation//
112 Info<< "Min/max phi:" << min(phi()).value() << ' '
113     << max(phi()).value() << endl;
114     }
115
116     Info<< "End\n" << endl;
117
118     return 0;
119 }
```

Taken From [1]

Bibliography

[1] Bhalchandra Bhadak, R Sankarasubramanian, and Abhik Choudhury. Phase-field modeling of equilibrium precipitate shapes under the in uence of coherency stresses. Metallurgical and Materials Transactions A, 49(11):5705 5726. [60]

[2] Harald Garcke, Britta Nestler, Bjorn Stinner, and Frank Wendler. Allen cahn systems with volume constraints. Mathematical Models and Methods in Applied Sciences, 18(08):1347-1381, 2008.[3]

[3] P. Wu, X. Ma, J. Zhang, and L. Chen, "Phase-field model of multiferroic composites: Domain structures of ferroelectric particles embedded in a ferromagnetic matrix," Philosophical Magazine, vol. 90, no. 1-4, pp. 125–140, 2010. [1]

[4] N. Moelans, B. Blanpain, and P. Wollants, "An introduction to phase-field modeling of microstructure evolution," Calphad, vol. 32, no. 2, pp. 268–294, 2008[1]

[5] Machine learning and the physical sciences* Giuseppe Carleo, Ignacio Cirac, Kyle Cranmer, Laurent Daudet, Maria Schuld, Naftali Tishby, Leslie Vogt-Maranto, and Lenka Zdeborová Rev. Mod. Phys. 91, 045002[13]

[6] Peter Bruce, Andrew Bruce, Practical Statistics for Data Scientists.[10][14] [15]

[7] Abhay Maheshwari and Alan J Ardell. Elastic interactions and their effect on gammaprime

precipitate shapes in aged dilute ni-al alloys. Scripta Metallurgica et Materialia;(United States), 26, 1992.[18]

[8] David B Brough, Daniel Wheeler & Surya R. Kalidindi. Materials Knowledge Systems in Python—a Data Science Framework for Accelerated Development of Hierarchical Materials. [41]

[9] David B. Brough, Daniel Wheeler, James A. Warren, and Surya R. Kalidindia. Microstructure-based knowledge systems for capturing process-structure evolution linkages. [41][49]

[10] Leonardo Vanneschi, Mauro Castelli, in Encyclopedia of Bioinformatics and Computational Biology, 2019 [24][25]

[11] Surya R. Kalidindi Hierarchical Materials Informatics: Novel Analytics for Materials Data [44]