Inter-dendritic segregation in Austenitic stainless steels

Thesis submitted in partial fulfilment of the requirements for the degree of

Bachelor of Science (research)

in

Materials Science and Engineering

by

Rohith K.M.S.

Department of Materials Engineering
Indian Institute of Science

under the supervision of

Abhik N. Choudhury

Department of Materials Engineering
Indian Institute of Science

Certificate

Mr. Rohith K.M.S. has worked under my supervision between January 2020 to March 2020. I have gone through his work on inter-dendritic segregation in austenitic stainless steels presented in this thesis and have found it to be satisfactory.

Dr. Abhik N. Choudhury

Department of Materials Engineering

Indian Institute of Science

Declaration

I, Mr. Rohith K.M.S., have participated in a project to investigate inter-dendritic segregation in Austenitic stainless steels under the supervision of Dr. Abhik N. Choudhury during January-March 2020. The contents of the thesis are written entirely by me and no unfair means were resorted to at any point in the project.

Rohith K.M.S.

Undergraduate Programme (Materials Science and Engineering)

Indian Institute of Science

Acknowledgements

I am extremely grateful to my supervisor Dr. Abhik N. Choudhury for his constant guidance and unwavering support without which this work would not have been possible. Thank you sir for giving me the freedom to choose my working hours and pace, which made the whole process a lot more manageable among the million different things to do in college. Thanks for not spoon-feeding me and trusting my capability to figure things out on my own even when I didn't trust myself. I needed to be thrown in the deep end of the metaphorical research swimming pool, so I could learn to do things on my own. I am grateful for the opportunity to be a TA for your course, it was a valuable experience.

I thank Dr. Karthikeyan Subramanian for his constant support throughout my time in the Materials Engineering department. Thank you for the guidance when I was stuck at different cross-roads both academic and philosophical in nature. I also thank Dr. Abhishek Singh for his guidance and support, and Dr. T.A. Abinandan for that one time he actually took the time to listen to my existensial rambling about how the future seemed scary and unpredictable.

I thank my labmate Sumeet for taking the time to answer the incessant questions I had even in the midst of his busy schedule. Your input was more valuable than you'll ever know. I thank my senior Bikramjit for helping me with OpenFOAM and saving me the time and effort of going through obscure forums in search of solutions to the many problems I faced while writing a phase field solver. I thank Swapnil for the stimulating discussions after Abhik sir's solidification class.

I thank all my amazing friends here at IISc. Especially Pratyusha, my rock. You are the reason I got out of this roller coaster ride we call college with my sanity intact. I thank Nehal, Jana and Akash for all the amazing experiences we've shared together across various trips to colorful places. I thank Julian, Gaurang and Raj for the best mess table conversations where we just geek out for hours. You guys are the reason I have such a deep appreciation for all fields of science. I am extremely grateful to my friend and labmate Kartik Sharma, whose decision to stay back on campus had a monumental impact on my thesis, because he gave me remote access to my lab computer.

Lastly, I am indebted to this amazing institution. IISc has given me access to education and people of a quality that is a privelage only a few people in the world get to have. I thank the KVPY scholarship for financially supporting my stay here.

Abstract

Additive manufacturing has been hailed as the "Third industrial revolution", offering near net-shape production of parts with complex geometries from a digital 3D model of a part , thus eliminating the need for specialised equipment for each part. This decreases the influence of the economics of scale, facilitating the production of a small number of customised parts on demand. In the additive manufacturing of austenitic (γ) stainless steels - one of the most widely used class of corrosion-resistant alloys - an important challenge is to control the fraction of the δ-ferrite phase in the microstructure. The δ-ferrite fraction is determined by the extent of the austenite to ferrite ($\gamma \rightarrow \delta$) post solidification phase transformation, which is in turn determined by the micro segregation between the dendrites formed during solidification. The primary elements in common stainless steel grades like 316L are Iron, Chromium and Nickel. The aim of this work is to investigate the inter-dendritic segregation in the Iron-Chromium-Nickel ternary system through mesoscale simulations using the phase field method. However, the work had to be restricted to the Fe-Cr binary system on account of a lack of time. Free energies of different phases were approximated by parabolic forms and a grand potential formulation for multicomponent systems was used to derive evolution equations for the phase and composition fields. Phase field codes were written to simulate the microstructural evolution of solidification of delta-ferrite in the Fe-Cr binary system in 1D and 2D, and different simulation parameters were tuned when the simulations did not lead to the formation of dendrites.

Contents

List of figures

List of tables

1

Introduction

1.1 Background

1.1.1 Additive Manufacturing

Traditional manufacturing methods are heavily influenced by the economics of scale. Making a single custom-designed part or a product is often prohibitively expensive, whereas making a million identical parts can susbtantially reduce the cost per part, making mass production profitable. Additive Manufacturing (AM) is an emerging manufacturing technique that promises to break this dependence on scale and repeatability, paving the way for highly customisable need-based production of different kinds of parts from a single machine. This reduces the need for manufacturers to maintain an inventory of spare parts, as they can simply be printed when the need arises. AM is a paradigm shift in manufacturing, so much so that The Economist called it the "Third industrial revolution" - the digitization of manufacturing. [15] AM also enables rapid prototyping and the production of intricate parts with complex geometries, offering manufacturers the ability to quickly test complex designs. Huang et. al. have analysed the societal impact of Additive Manufacturing technology and summarised the key positive impacts as follows [7]:

Customised healthcare products: Implants, scaffold for tissue engineering, etc.. tailored to an individual's physiology.

Reduced environmental impact: Less material wastage, lack of need of coolants, more efficient in water usage.

Simplified supply chain: Significantly reduces the need for warehousing, transportation and packaging. Facilitates on-demand production.

Due to all these benefits, there has been a recent surge in interest in Additive Manufacturing. Along with the development of new AM methods and improving existing ones, there is also a lot of research effort going into selecting materials and optimizing process parameters in order to meet the property requirements of a manufactured part.

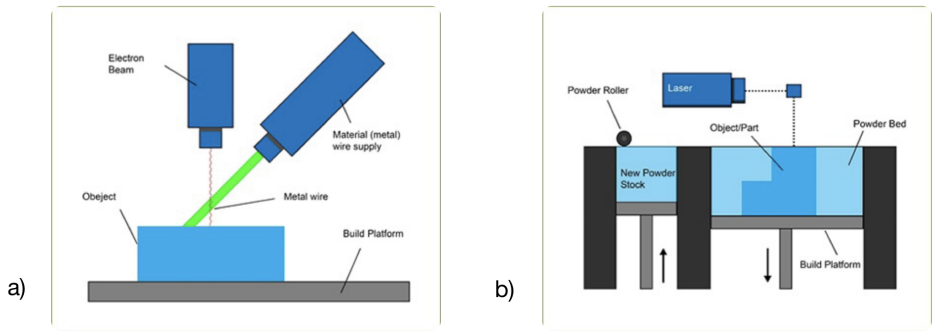


Figure 1.1: Schematic of the two types of Additive Manufacturing processes for metals (a) Directed Energy Deposition and (b) Powder Bed Fusion. Images taken from [1]

1.1.2 Metal Additive Manufacturing Techniques

While the AM of polymers has found widespread adoption even among amateur designers (3D printers and filaments are available for affordable prices), the AM of metals still has a long way to go. This is primarily because of the high fusion temperatures of metals requiring powerful lasers/electron beams and the high cost of the raw materials (metal powders).

Currently used AM techniques for metals can be broadly divided into two types:

Directed Energy Deposition (DED): Also called Laser Engineered Net Shaping (LENS) and Directed Metal Deposition (DMD), it is a process where a high power Laser/Electron beam is focused on the region where the material is to be deposited and the material is fed in as a wire or powder through a separate nozzle or through a nozzle built into the laser/electron beam head. Schematic is shown in Figure 1.1 (a).

Powder Bed Fusion (PBF): Includes Selective Laser Melting (SLM), Direct metal laser sintering (DMLS) and their electron beam counterparts. In this process, a thin layer of metal powder is deposited onto the substrate after which a laser/electron beam selectively melts the region which is to be fused. Schematic is shown in Figure 1.1 (b).

1.1.3 Stainless steels in Additive Manufacturing

Steels are the most widely used class of alloys today. They offer an unrivalled variety of achieveable microstructural features and have the benefit of many decades of dedicated research into their structure-property-process relationships that have resulted in their adoption into an extremely wide variety of applications. When the application requires corrosion resistance along with good toughness and strength, stainless steels are often a good choice. Based on their microstructure, they can be broadly classified into Austenitic (fcc), Ferritic (bcc), Martensitic and Austenitic/Ferritic (Duplex) stainless steels. Due to their comparably moderate price and good processablilty, austenetic stainless steel grades like 316L are commonly studied for their Additive manufacturing potential. Figure 1.2 shows an Ashby plot of different classes of steel manufactured through Laser PBF, DED and conventional processing routes.

The scope of this work is restricted to Austenitic stainless steel grades like 316L and 304L. This is shown as the blue region in Figure 1.2. The composition ranges of the aforementioned grades are such that, upon solidification, they can give rise to either Ferrite (δ) or Austenite (γ) as the primary solidification phase. In their review of AM of Stainless steels, Bajaj et. al. have noted

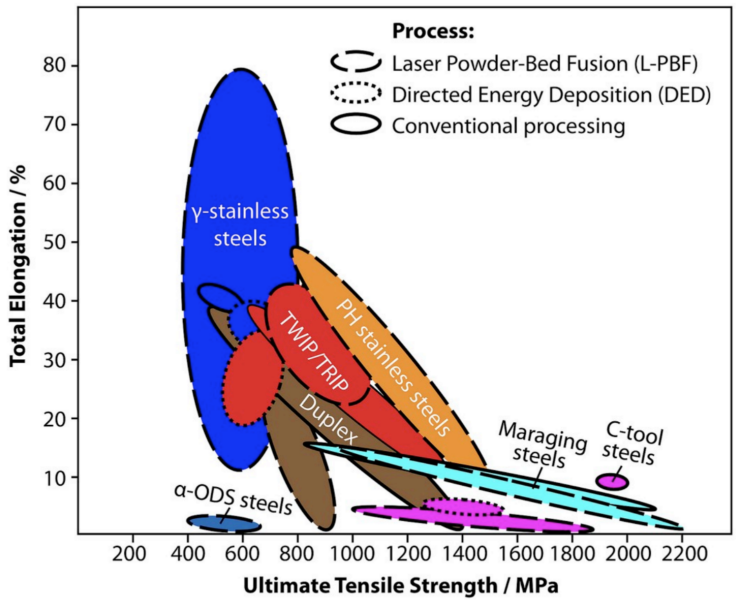


Figure 1.2: Ashby plot of Ductility vs. Ultimate Tensile Strength of different steel families. The scope of this work is restricted to Austenitic(γ) stainless steels, shown here in blue. Figure taken from [4]

that Directed Energy Deposition processing gives rise to microstructures containing films of δ-ferrite along the borders of solidification cells, whereas the microstructures produced in Laser Powder Bed Fusion (L-PBF) are fully austenitic. [4]

1.1.3.1 Importance of ferrite control

The presence of δ-ferrite can be beneficial or detrimental depending on the application. The ASM Handbook on Irons and steels [3] clearly outlines the benefits and detriments. The presence of δ-ferrite can improve weldability by making the part less suceptible to microfissuring or hot cracking during welding. It can also improve resistance to Stress-Corrosion Cracking, when used in specific environments. It can be detrimental in high-temperature applications, where an intermetallic σ or χ phase can form at the δ phase boundaries, which can lead to embrittlement if the ferrite forms a continuous network. The presence of ferrite also reduces toughness. Thus, it is important to control the volume fraction and distribution of ferrite in the microstructure in order to keep it in the suitable range for the application.

1.1.3.2 Role of microsegregation

The addition of Chromium in sufficient quantities ($> 11\%$) imparts passivity to steels. However, since Chromium is a δ-stabilizer, γ-stabilizing elements like Nickel or Manganese must be added to the alloy in order to get fully-austenitic steels. This is especially the case in low carbon (C is an austenite stabilizer) alloys like 316L and 304L. The δ and γ stabilisers are listed below [3]:

Ferrite (δ) *stabilising elements:* Cr, Mo, Si and Nb

Austenite (γ) *stabilising elements:* Ni, C, Mn, N

As mentioned before, 316L and 304L stainless steels are in the composition range where the primary solidification phase can be δ or γ. Depending on the primary solidification phase,

Chromium can segregate into the liquid or the solid phase. In primary Ferrite solidification, the microsegregation leads to the depletion of Cr in the liquid as the fraction solidified increases. This makes the core of the δ-ferrite dendrites richer in Cr than the outer regions. Upon further cooling, the Cr-depleted outer regions undergo a post-solidification phase transformation into the γ-phase, leaving behind Cr-rich dendrite "skeletons" [14]. This inter-dendritic segregation is an important factor in determining the fraction of the δ-ferrite in the final microstructure.

The inter-dendritic segregation can be used to estimate the fraction of δ-ferrite in the final microstructure, so that the process parameters can be tuned to give the required amount of ferrite. It can also be used to estimate any further heat treatment that needs to be given to the part to bring it to the required δ-fraction. This is the primary motivation behind this work.

1.2 Aim and scope of this work

It is very difficult to measure inter-dendritic segregation during solidification experimentally, because often we only have access to the final microstructure. Hence, we have to rely on simulations to get the time-resolved evolution of inter-dendritic segregation. For such problems involving the movement of a boundary between homogenous phases, one of the most common simulation techniques is the Phase Field Technique.

The aim of this work was to study inter-dendritic segregation in the Ternary Fe-Cr-Ni system through phase field simulations of solidification at the thermal gradients and interface velocities seen during L-PBF of Austenitic stainless steel. The Fe-Cr binary system was modelled first under isothermal (zero temperature gradient) conditions, with a view to expand to the more complicated Fe-Cr-Ni ternary system and also under conditions seen in AM. However, the work had to be stopped prematurely to COVID-19, and only isothermal solidification in the Fe-Cr system could be done, and is presented in this thesis.

1.3 Organisation of this thesis

This introduction forms the first chapter of this thesis. Chapter 2 reviews the background literature needed to understand this work - some stainless steel metallurgy and the grand potential formulation of the phase field method - and establishes its aim and scope. Chapter 3 deals with the Parabolic Approximation to the free energy expressions in the Fe-Cr binary system and the derivation of phase field evolution equations for parabolic free energies. Chapter 4 deals with the methods used in the 1D isothermal solidification simulations, presents their results and discussion. Chapter 5 deals with the methods used in the 2D isothermal solidification simulations, presents their results and discussion. Chapter 6 discusses the work planned for the next 1.5 months after the COVID-19 lockdown and all the differnt problems faced during the course of this project. Chapter 7 concludes this thesis and gives the direction for future work.

The appendix A contains Python codes written for 1D solidification simulations presented in this thesis. Appendix B contains an introduction to OpenFOAM, the file structure used in OpenFOAM cases and the OpenFOAM codes for the 2D simulations presented in this thesis.

2

Literature Review

2.1 Stainless steel welding microstructures

Metal additive manufacturing techniques like PBF and DED can be seen as sophisticated variants of Laser/Electron Beam welding. The material is subjected to similar power densities, thermal gradients and welding speeds so the microstructural evolution is also expected to be similar. Austenitic stainless steel welding microstructures have been extensively studied, so it is helpful to review the well-established aspects of this area. Sindo Kou's book on welding metallurgy offers a good discussion on this topic [14].

2.1.1 Fe-Cr-Ni pseudo-binary Phase diagram

Pseudo-binary phase diagrams are constructed by taking isoplethal (fixed concentration of one constituent element) sections of the ternary phase diagram. They are helpful in understanding ternary phase diagrams through a representation similar to a conventional binary phase diagram (composition-temperature plot). Fe-Cu-Ni pseudo-binary phase diagrams are shown in Figure 2.1. The three phase triangle is a region where the three phases Austenite (γ), Ferrite (δ) and the liquid (L) are in equilibrium. This feature would not exist in a binary phase diagram, because Gibb's phase rule confines the three-phase equilibrium to a point on the phase diagram of binary systems.

2.1.2 Ferrite morphology

The primary solidification mode and the subsequent post-solidification phase transformations determine the final morphology of ferrite in the microstructure, which can be interdendritic (Figure 2.2a), vermicular (Figure 2.2b) or lathy (Figure 2.2c).

2.1.2.1 Primary Solidification Modes

An alloy with composition that lies on the left of the three-phase triangle apex (marked in Figure 2.2d as 1) solidifies with ferrite as the primary phase, whereas an Ni-rich alloy (right of three-phase triangle apex) solidifies with austenite as primary phase. Ferrite is shown in Figure

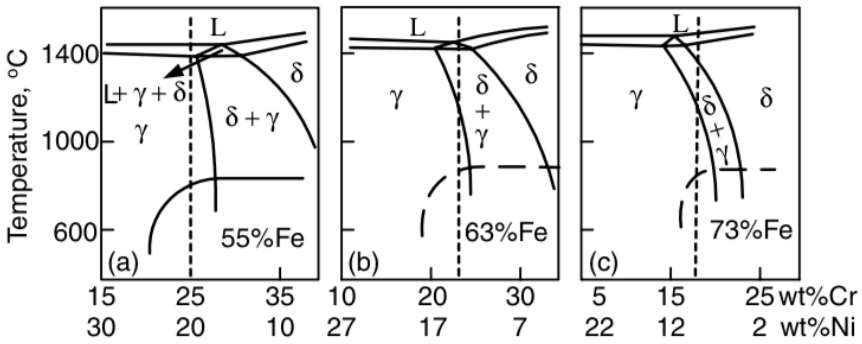


Figure 2.1: Fe-Cr-Ni pseudo-binary phase diagrams at (a) 55 wt % Fe; (b) 63 wt% Fe; (c) 73 wt % Fe. The area marked $L+\delta+\gamma$ is the three phase triangle, where the liquid (L), ferrite δ and austinite (γ) phases are in equilibrium. Figure taken from [14]. The dotted lines represent an intermetallic (σ) phase boundary.

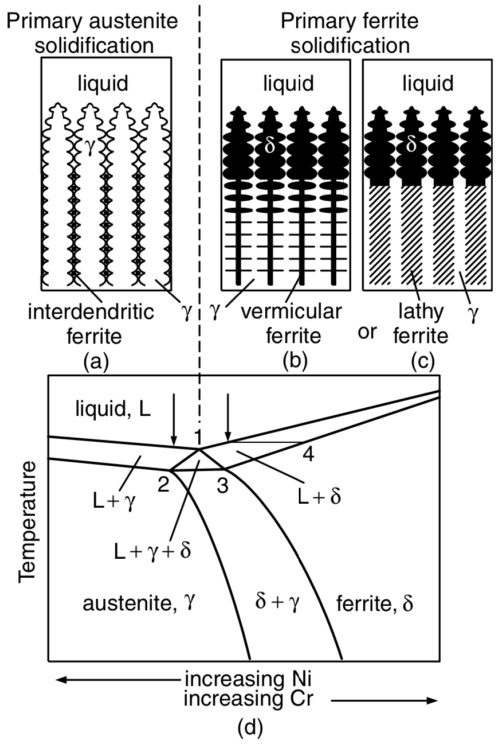


Figure 2.2: Schematic diagram of the different ferrite morphologies formed during solidification of Fe-Cr-Ni welds: (a) interdendritic ferrite; (b) vermicular ferrite; (c) lathy ferrite. A pseudo-binary phase diagram for approximately 70% Fe is shown in (d). The apex of the three-phase triangle is marked by 1. The two arrows to the left and right of 1 correspond to starting compositions leading to primary δ and primary γ solidification respectively. 4 marks the composition of initial ferrite solidified.

Cr-Ni ratio	Primary solidification mode	Ferrite content on increasing cooling rate
Low	Primary γ-Austenite	Decreases
High	Primary δ-Ferrite	Increases

Table 2.1: Effects of cooling rate on alloys with high and low Cr-Ni ratios. [14]

2.2 in black whereas austenite is shown in white.

For an Ni-rich alloy with composition shown by the arrow in Figure 2.2d to the left of the three-phase triangle apex, γ-austenite dendrites grow into the liquid, the interdendritic region between the primary arms enriched in Cr due to segregation. If the three-phase triangle is reached during solidification, ferrite phase is formed at the interdendritic region. This is called interdendritic ferrite (dark region in Figure 2.2a).

2.1.2.2 Post-solidification phase transformation

For a Cr-rich alloy with composition shown by the arrow in Figure 2.2d to the right of the three-phase triangle apex, the δ-ferrite dendrites grow into the liquid, the interdendritic region between the primary arms depleted in Cr due to segregation. This leads to the δ-ferrite dendrites cores having a larger concentration of Cr than the outer regions. The initial composition of these δ-ferrite dendrites cores is marked in 2.2 as point 4. These outer regions transform into austenite upon cooling into the $\delta+\gamma$ region, leaving behind Cr-rich cores of the dendrites of δ-ferrite. This morphology is called vermicular ferrite (dark region in Figure 2.2b).

There is a third possible morphology called lacy or lathy ferrite, which occurs if certain crystallographic conditions are satisfied. During the initial phase of primary δ-ferrite solidification, austenite first grows epitaxially on the unmelted austenite grains followed by nucleation of δ-ferrite. The crystallographic relationship between the ferrite nuclei and the asutenite susbtrate primarily determines the formation of vermicular or lacy ferrite, along with the relationship between the heat flow direction and preferential growth directions of ferrite and austenite. Inoue et. al. have studied the formation mechanisms of vermicular and lacy ferrite [8]. The necessary conditions for the formation of lacy ferrite are:

> The ferrite and the austenite should be oriented with respect to each other such that they satisfy the so-called *Kurdjmov-Sachs* orientation crystallographic relationships namely, $(\bar{1}10)_\delta//(\bar{1}11)_\gamma$ and $[\bar{1}\bar{1}1]_\delta//[\bar{1}\bar{1}0]_\gamma$
>
> The preferential growth direction of both austenite and ferrite i.e. $< 100 >$ should be aligned with the heat flow direction.

2.1.3 Effect of Cooling rate

In additive manufacturing, the cooling rates are comparable to laser and electron beam welding. At such high cooling rates, the final microstructure depends quite strongly on the cooling rate. Table 2.1 outlines the effect of cooling rate on alloys with high and low Cr-Ni ratios. Alloys with low Cr-Ni ratios solidify with γ-Austenite as their primary phase which leads to the formation of interdendritic δ-ferrite, as discussed in 2.1.2.1. Increasing the cooling rate decreases the solute redistribution during solidification. As a result, interdendritic weld metal is less depleted in Cr, which leads to lesser δ-ferrite formation. Alloys with high Cr-Ni ratios solidify with δ-Ferrite as their primary phase. Austenite is then formed by the post solidification transformation discussed in 2.1.2.2. Increasing the cooling rate decreases the time available for this $\delta \rightarrow \gamma$ transformation, thus decreasing the γ-austenite content and increasing δ-ferrite content.

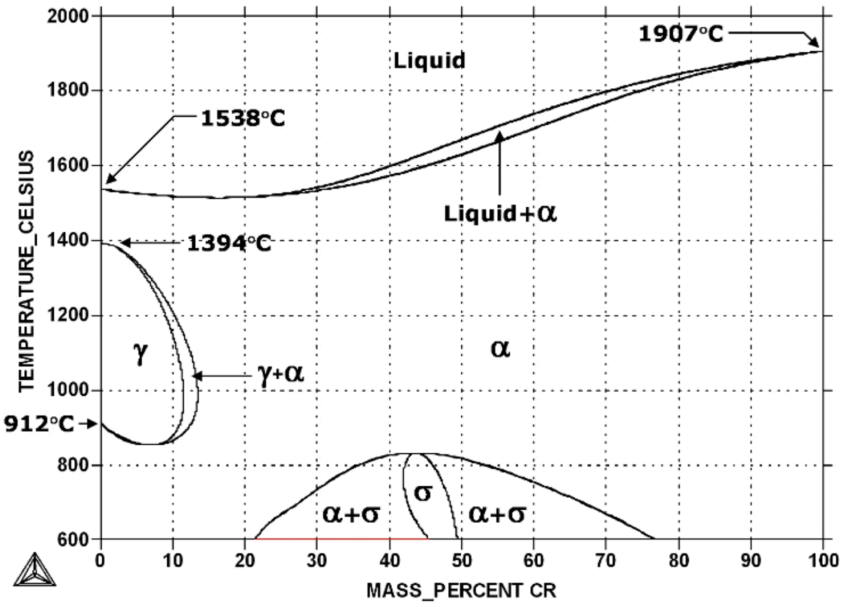


Figure 2.3: Fe-Cr phase diagram. The ferrite phase is denoted by α, the austenite phase is denoted by γ and σ is an intermetallic phase. Figure taken from [16].

2.2 Thermodynamics of Fe-Cr binary system

After reviewing the established literature on microstructures of stainless steel welds, the next area to focus on was the background needed to model microstructural evolution. As mentioned in 1.2 it was decided to model solidification in the Fe-Cr binary system first. In order to do this, it was essential to obtain good models for the free energy of the phases involved. Andersson and Sundman have evaluated the thermodynamic properties of the Cr-Fe system and constructed free energy equations for different phases using the CALPHAD (CALculation of PHAse Diagrams) technique [2]. The following section is mostly a summary of the parts of their paper which are relevant to this work.

2.2.1 Fe-Cr binary phase diagram

The Fe-Cr phase diagram taken from the CALPHAD website [16] is shown in figure 2.3. The bcc ferrite phase is denoted in this phase diagram by α. The fcc austenite phase is denoted as usual by γ, and σ corresponds to an intermetallic phase. The primary solidification phase is ferrite, as expected from the discussion in 2.1.2.1. The austenite phase is thermodynamically stable in the "γ-loop" region of the phase diagram. Upon cooling into this region, the ferrite undergoes a solid state transformation as discussed in 2.1.2.2.

2.2.2 Free energy functions

The molar Gibbs free energy equations of the phases have the following terms:

$$G_m = G^o + G^{xs} + G^{mo} \tag{2.1}$$

Here, G^o is the contribution from the ideal solution model given in 2.2, G^{xs} is the excess energy

Coefficient	Expression (SI units)
L^0_δ	$+20500 - 9.68T$
L^0_L	$-14550 + 6.65T$
L^0_γ	$-10833 - 7.477T$
L^1_γ	$+1410$

Table 2.2: Redlich-Kister coefficients for different phases. γ is the fcc austenite phase, δ is the bcc ferrite phase, and L is the liquid phase. Values taken from [2]

term, which is expressed as a *Redlich-Kister* polynomial as shown in , and G^{mo} is the magnetic contribution, which only appears in the expression for the bcc (ferrite) phase. Each of these terms are discussed in some detail in the subsections that follow.

2.2.2.1 Ideal solution term

The first term in 2.1 G^o corresponds to the ideal solution model for binary alloys (where the enthalpy of mixing ΔH_{mix} is zero). This is given by:

$$G^o = X_{Cr}G^o_{Cr} + X_{Fe}G^o_{Fe} + RT(X_{Cr}lnX_{Cr} + X_{Fe}lnX_{Fe}) \tag{2.2}$$

Here, X_{Cr} and X_{Fe} are the mole fractions of Chromium and Iron respectively. G^o_{Fe} and G^o_{Cr} are the standard molar free energies of Iron and Chromium respectively. The expressions for G^o_{Fe} and G^o_{Cr} for different phases are given in [2]. R is the gas constant and T is the temperature.

2.2.2.2 Excess energy term

The second term in 2.1 G^{xs} is the excess energy term. Andersson and Sundman have used the Redlich-Kister binary excess model for the excess Gibbs free energy [2]. The general form of the excess free energy for a multicomponent system is as follows:

$$G^{xs} = \sum_{i=1}^{n-1} \sum_{j=1+1}^{n} X_i X_j L_{ij} \tag{2.3}$$

Here, i and j are indices that denote one component in an n component system. And X_i is the mole fraction of component i. This is basically a sum of binary interactions between each possible pair of components in the system. In the Redlich-Kister model, the binary interaction parameter L_{ij} is given by a Redlich-Kister (RK) polynomial (or rather, power series expansion):

$$L_{ij} = \sum_{\nu=0}^{k} (X_i - X_j)^\nu L^\nu_{ij} \tag{2.4}$$

For binary systems, there is only one i-j pair and so the indices can be dropped. Combining equations 2.3 and 2.4 and dropping the indices, for the Fe-Cr system we get:

$$G^{xs} = X_{Cr}X_{Fe} \sum_{\nu=0}^{k} (X_{Cr} - X_{Fe})^\nu L^\nu \tag{2.5}$$

The RK coefficients L^ν are listed in table 2.2 for the different phases. A good discussion on the theory behind using RK binary excess model is given in [10].

2.2.2.3 Magnetic contribution term

The third term in equation 2.1 is the magnetic contribution term G^{mo}. This appears only for the bcc δ-ferrite phase. It is given in equation 2.6. The average magnetic moment term β is given in equation 2.7. $f(\tau)$ is a function of the temperature scaled by the Curie temperature $\tau = T/T_c$ and is given in 2.8. The Curie temperature in Kelvin is given by 2.9.

$$G^{mo} = RTln(\beta + 1)f(\tau) \tag{2.6}$$

$$\beta = 2.22X_{Fe} - 0.008X_{Cr} - X_{Cr}X_{Fe}0.85 \tag{2.7}$$

$$f(\tau) = \begin{cases} -0.90530\tau^{-1} + 1 - 0.153\tau^{3} - 6.8 \times 10^{-1}\tau^{9} - 1.53 \times 10^{-3}\tau^{15} & \tau < 1 \\ -6.417 \times 10^{-2}\tau^{-5} - 2.037 \times 10^{-3}\tau^{-15} - 4.278 \times 10^{-4}\tau^{-25} & \tau > 1 \end{cases} \tag{2.8}$$

$$T_c = 1043X_{Fe} - 311.5X_{Cr} + X_{Cr}X_{Fe}(1650 + 550(X_{Cr} - X_{Fe})) \tag{2.9}$$

2.3 Phase Field Technique

The Phase Field Method is a technique for solving so called free-boundary problems, i.e. problems which deal with transport of quantities like mass, heat and momentum across an interface between two homogeneous phases. The interface can freely move as long as it satisfies certain boundary conditions, and hence can take up complicated shapes such as in dendrites. Conventional methods of solving such problems explicitly track the interface and apply boundary conditions at the cells in the computational grid that the interface lies on. Models that follow this approach are called *sharp-interface models*. A major disadvantage of such models is that the explicit tracking of the interface is difficult and computationally intensive. Moreover, such models are not feasible for problems where the interface has a complex geometry, such as in dendrites. Hence these methods are not suitable for our purpose. The Phase Field method takes a different approach. Here, the interface is not sharp but spatially smeared or "diffuse". Hence, it belongs to a class called *diffuse-interface models*. The two approaches are contrasted schematically in figure 2.4. As we will see in the following subsections, this approach allows us to solve such problems without explicitly tracking the interface which gives it a great advantage over the sharp interface models.

In the following subsections, we first formulate the phase field method for solidification of a pure substance, in order to gain a general understanding of the phase field method. Then, we move to our system of interest, the more complicated binary alloy solidification. In this work, we have used the *grand potential formulation* developed by Choudhury and Nestler [5]. In the following discussion, we first formulate a phase field model for solidification of a pure substance from the free energy functional. Then, we discuss the rationale behind using a Grand Potential Functional for alloy solidification, and formulate the evolution equations for the phase field and chemical potential field.

2.3.1 Order Parameters and Phase Fields

In order to compute the evolution of the phases, it is necessary to first distinguish them. A parameter that is used to differentiate between the different phases in the problem is called an *order parameter*. This can be chosen from any of the various intensive quantities that differ between the two phases. Order parameters can be *conserved* or *non-conserved*. Composition

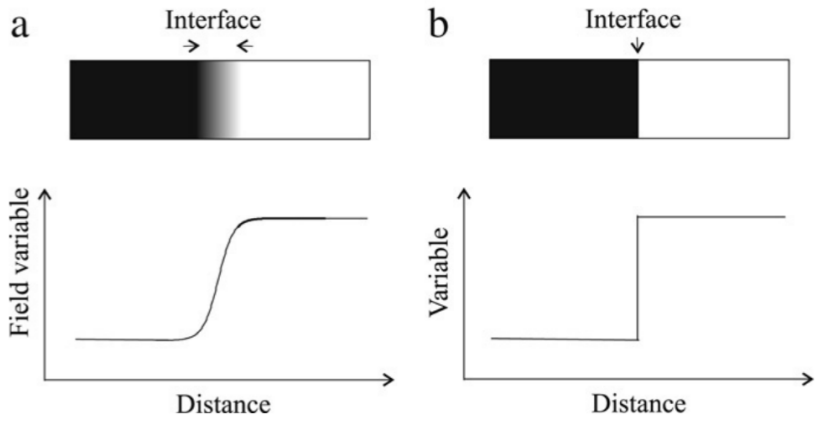


Figure 2.4: Schematics showing the difference between (a) Diffuse interface and (b) Sharp interface models. Figure taken from [11]

is an example for a conserved order parameter, since composition is different between different phases and the number of atoms of each species is a conserved quantity.

In the phase field approach, the order parameter (denoted ϕ) varies continuously over a finite length between two phases. That is to say, the interface has a finite width and is not atomistically sharp. The interface width is chosen depending on the length scale of the phenomena to be modeled. The order parameter (also called the phase field variable) changes smoothly across the interface length as shown in figure 2.4a. The phase field variables are continuous functions in spatial and temporal coordinates. The temporal evolution of the different variables in the system (including the phase field variable) can then be described through the use of partial differential equations of the different fields. The boundary conditions are self-consistently accounted for in these equations, thus removing the need to track the interface explicitly.

2.3.2 Pure Substance Solidification: Free Energy formulation

2.3.2.1 Free Energy Functional

To formulate the evolution of the different fields mathematically, we make use of *functionals*. For our purposes, a functional can be thought of as a function of functions, i.e. a function that takes another function as an argument and returns a value. For solidification of a pure substance, the commonly used choice of fields to describe the system are the phase field ϕ and temperature T. The system is described by a free energy functional F given in 2.10. Let us take the solid as $\phi = 1$ and liquid as $\phi = 0$.

$$F = \int_V f(\phi, \nabla\phi, T) = \int_V f_{int}(\phi, \nabla\phi) + h_s(\phi) f_s(T) + [1 - h_s(\phi)] f_l(T) \tag{2.10}$$

Here, f is the free energy density. f_s and f_l are the free energy densities in the solid and liquid phase respectively, and $h_s(\phi)$ is a function used to interpolate the free energies across the interface. h_s satisfies the conditions $h_s(0) = 0$, $h_s(1) = 1$ and $h'_s(0) = h'_s(1) = 0$. The function we use is given in 2.11.

$$h_s(\phi) = \phi^2(3 - 2\phi) \tag{2.11}$$

f_{int} corresponds to the interfacial contribution to the free energy density, given in 2.12. σ and H are constants with dimensions energy per unit length and energy per unit volume respectively. f_{dw} is a double-well potential with minima at $\phi = 0, 1$.

$$f_{int}(\phi, \nabla\phi) = \frac{1}{2}\sigma(\nabla\phi)^2 + H f_{dw}(\phi) \tag{2.12}$$

The interface width W is given by 2.13, so these constants can be chosen as per our choice of interface width and the surface energy [13].

$$W = \sqrt{\frac{\sigma}{H}} \tag{2.13}$$

2.3.2.2 Allen-Cahn Dynamics

The Allen-Cahn equation is an evolution equation for non-conserved order parameters. It is applicable for materials phenomena describing the evolution of phase transformations, grain growth, etc... The essential idea is that the phase field evolves in a manner that minimises the free energy.

$$\frac{\partial\phi}{\partial t} = -M\frac{\delta F}{\delta\phi} \tag{2.14}$$

Here ϕ is the phase field variable, M is the mobility of the interface, $\frac{\delta}{\delta\phi}$ is the variational derivative with respect to the phase field ϕ and F is the Helmholtz free energy functional described in 2.10.

The variational derivative of a functional $G = \int_{-\infty}^{\infty} g(\eta, \nabla\eta, \nabla^2\eta, ...)dx$ with respect to a field η is defined as

$$\frac{\delta G}{\delta\phi} = \left(\frac{\partial}{\partial\eta} + \nabla.\frac{\partial}{\partial\nabla\eta} + \nabla^2.\frac{\partial}{\partial\nabla^2\eta} + ...\right)g \tag{2.15}$$

Therefore, after applying the variational derivative in 2.14 it reads as

$$\frac{\partial\phi}{\partial t} = -M\left\{H f'_{dw}(\phi) - \sigma\nabla^2\phi + \frac{h'_s(\phi)}{2}[f_s(T) - f_l(T)]\right\} \tag{2.16}$$

This is the phase field evolution equation given by Allen-Cahn dynamics.

2.3.3 Binary Alloy Solidification: Grand Potential Formulation

For alloy solidification, the obvious choice of fields seems to be the phase field ϕ, the temperature T and the mole fraction X. This corresponds to a canonical ensemble and hence, the natural choice of functional to describe the system is the Helmholtz free energy functional F. However, Choudhury et.al. and Plapp [5, 13] have argued that a better starting point for formulating phase field models for binary alloy solidification is the *grand potential functional*, which has the phase field variable ϕ, temperature T and μ the fundamental fields. This corresponds to the grand canonical ensemble. The main reason behind this choice is that the free energy functional approach for alloy solidification leads to an intrinsic coupling between the interface and the bulk. This coupling makes simulation results dependant on the choice of interface thickness [13]. It

also prevents us from being able to choose an independent interface length. This is an important disadvantage, because the interface width needs to be chosen according to the length scale of the phenomena to be modeled. Otherwise resolving the two length scales would be computationally prohibitively expensive.

The Grand Potential functional is defined as given in [5].

$$\Omega[\phi, T] = \int_V \Psi(T, \mu, \phi) + \left(\epsilon a(\phi, \nabla\phi) + \frac{1}{\epsilon}\omega(\phi)\right) dV \tag{2.17}$$

Here, ϵ is a factor related to the length scale of the interface. Ψ is the grand potential density function, which is related to the free energy function by 2.19. ω is the double well potential term, similar to f_{dw} in 2.12. The double well potential is given by

$$\omega = W\phi^2(1-\phi)^2 \tag{2.18}$$

W is a parameter that controls the height of the double well potential. The grand potential density can be written for each phase (solid or liquid) as

$$\Psi_{s/l}(T, \mu, \phi) = f_{s/l}(c_{s/l}(\mu), T) - \mu c_{s/l}(\mu, T) \tag{2.19}$$

In the interface region, the grand potential density is computed by an interpolation function exactly as in 2.11

$$\Psi = \Psi_s h_s(\phi) + \Psi_l(1 - h_s(\phi)) \tag{2.20}$$

f is the free energy density. Note that to compute Ψ, we need the chemical potential to be an invertible function of the composition, i.e. we need to be able to compute $c(\mu)$. This is necessary if we want to switch to the chemical potential μ as the fundamental variable instead of the composition c. The form of the free energies given in 2.2.2 do not lead to chemical potentials that are reversible functions of composition. The next chapter discusses the solution to this problem i.e. approximating the free energies to a parabolic form.

$\epsilon a(\phi, \nabla\phi)$ in 2.17 is the interfacial energy term (analogous to $\frac{1}{2}\sigma(\nabla\phi)^2$ in 2.10). It also has a similar form.

$$a(\phi, \nabla\phi) = \sigma a_c^2(\hat{n})(\nabla\phi)^2 \tag{2.21}$$

Here, $a_c(\hat{n})$ is a function that accounts for anisotropy in the interfacial energy. $\hat{n}$ is the normal vector on the interface pointing towards the liquid side.

The Allen-Cahn evolution equation for the grand potential formulation is

$$\frac{\partial\phi}{\partial t} = -M_\phi \frac{\delta\Omega}{\delta\phi} \tag{2.22}$$

The only difference is that the variational derivative is taken on the grand potential functional instead of the free energy functional. Computing the variational derivative of 2.19 and substituting

in 2.22, we get the evolution equation for the phase field [5].

$$\tau\epsilon\frac{\partial\phi}{\partial t} = \epsilon\left(\nabla.\frac{\partial a(\phi,\nabla\phi)}{\partial\nabla\phi} - \frac{\partial a(\phi,\nabla\phi)}{\partial\phi}\right) - \frac{1}{\epsilon}\frac{\partial\omega(\phi)}{\partial\phi} - \frac{\partial\Psi(T,\mu,\phi)}{\partial\phi} \tag{2.23}$$

Here, τ is the relaxation constant of the interface. It must be chosen according to the following expression

$$\tau = \epsilon\frac{[c_s(\mu_{eq},T) - c_l(\mu_{eq},T)]^2}{D_l\frac{\partial c_l}{\partial\mu}}(M+F) \tag{2.24}$$

Here, M and F are constants that depend on the interpolation function for free energy density $h_s(\phi)$. For the function we have chosen 2.11 these values are $M = 0.063828$ and $F = 0.158741$ [5].

Choudhury et.al also give the evolution equation for the chemical potential field [5]

$$\left(\frac{\partial c_s(\mu,T)}{\partial\mu}h_s(\phi) + \frac{\partial c_l(\mu,T)}{\partial\mu}(1-h_s(\phi))\right)\frac{\partial\mu}{\partial t} = -[c_s(\mu,T) - c_l(\mu,T)]\frac{\partial h_s(\phi)}{\partial t} + \nabla.\left[\left(D_s g_s(\phi)\frac{\partial c_s(\mu,T)}{\partial\mu} + D_l(1-g_s(\phi))\frac{\partial c_l(\mu,T)}{\partial\mu}\right)\nabla\mu\right] \tag{2.25}$$

Here, $g_s(\phi)$ is an interpolation function for the mobility of atoms in the interphase region, similar to the interpolation function for grand potential densities $h_s(\phi)$ given in 2.20. D_s and D_l are the diffusivities in the solid and liquid respectively. c_s and c_l are the equilibrium compositions of the solid and liquid respectively.

Computing the microstructural evolution thus involves solving the two coupled evolution equations 2.25 and 2.23 for the phase field and chemical potential field. The next chapter 3 discusses the parabolic approximation to free energies in the Fe-Cr system and the derivation of the final form of the evolution equations for ϕ and μ that we will use in our codes.

2.4 Anisotropy

The anisotropy in the solid-liquid interface energy is an important factor that influences dendritic growth. Hence, it is important to include anisotropy in the 2D simulations. As discussed before, this is done through the term $a_c(\hat{n})$ in equation 2.21. $a_c(\hat{n})$ has a form corresponding to fourfold anisotropy

$$a_c(\hat{n}) = 1 \pm \zeta\left[3 - 4(\sum_{i=1}^{d} n_i^4)\right] \tag{2.26}$$

Where $\hat{n}$ is the nomal vector and n_i its component along the i^{th} basis vector. d is the number of dimensions in the problem, i.e. for 2D simulations $d = 2$. ζ is the anisotropy parameter. Liu et. al have calculated the anisotropy term for solid-liquid interface in bcc iron using molecular dynamics simulations, which we have used here [9].

Substituting the form of the anisotropy function given in 2.26 into 2.23, we get the following equations for *phi* evolution.

$$\tau\epsilon\frac{\partial\phi}{\partial t} = 2\epsilon\sigma\nabla^2(a_c^2(\hat{n})\phi) + \epsilon\nabla.(\frac{\partial a}{\partial\vec{\nabla}\phi}) - \frac{2W}{\epsilon}\sigma\phi(1-\phi) - 6(\Psi_s - \Psi_l)\phi(1-\phi) \tag{2.27}$$

Here, the vector derivative $\frac{\partial a}{\partial\vec{\nabla}\phi}$ is given by the equation 2.28.

$$\frac{\partial a}{\partial\vec{\nabla}\phi} = 2\sigma a_c|\vec{q}|\left[\sum_{i=1}^{d}\left(\frac{q_i^3}{|q|^6} - \frac{q_1^4+q_2^4+q_3^4}{|q|^6}\right)\hat{q}\right] \tag{2.28}$$

$\vec{q}$ is the gradient of the phase field variable given in 2.29. $\hat{q}$ is the unit vector along the direction of the gradient and q_i is the i^{th} component of the vector $\vec{q}$.

$$\vec{q} = \vec{\nabla}\phi \tag{2.29}$$

3

Parabolic Approximation

Free energies for different phases in the Fe-Cr system as a function of composition and temperature were given by Andersson and Sundman [2]. The various parameters are tabulated in 2.2.2. However, these functions give a form of the chemical potential μ that does not have an analytical inverse i.e. there is no analytical expression $c(\mu)$ that is an inverse of the function $\mu(c)$. As mentioned in 2.3.3, in order to switch to μ as the fundamental field instead of c, we need $\mu(c)$ to be an invertible (implying monotonic, since the chemical potential corresponds to the y-intercepts of the G-X curve) function of the composition c. Thus, we need to recast the equations in 2.2.2 in a form where the chemical potential is monotonic in c.

A straightforward way to do this is to approximate each free energy function by a parabola $p(c, T)$ of the form given in 3.1. A parabolic free energy function is strictly monotonic in chemical potential μ. The expression for μ is shown in 3.2 for the parabolic free energy $p(c, T)$. This is clearly monotonic and taking an inverse gives us the simple expression 3.3.

$$p(c, T) = a_2(T)c^2 + a_1(T)c + a_0(T) \tag{3.1}$$

$$\mu(c, T) = \frac{\partial p}{\partial c} = 2a_2(T)c + a_1(T) \tag{3.2}$$

$$c(\mu, T) = \frac{\mu - a_1(T)}{2a_2(T)} \tag{3.3}$$

The parabolic approximation has another advantage. The expressions 3.3 and 3.2 do not contain any logarithmic terms unlike the expressions in 2.2.2. This speeds up the calculation significantly, as evaluation of logarithms is computationally much more expensive than simple operations like multiplication. Since μ and c need to be evaluated at every step, this leads to a significant decrease in computational time.

In this chapter, we will discuss how the coefficient functions $a_0(T), a_1(T)$ and $a_2(T)$ were evaluated. Then, we derive analytical expressions for the evolution equations given in 2.3.3 by substituting the parabolic forms of free energy.

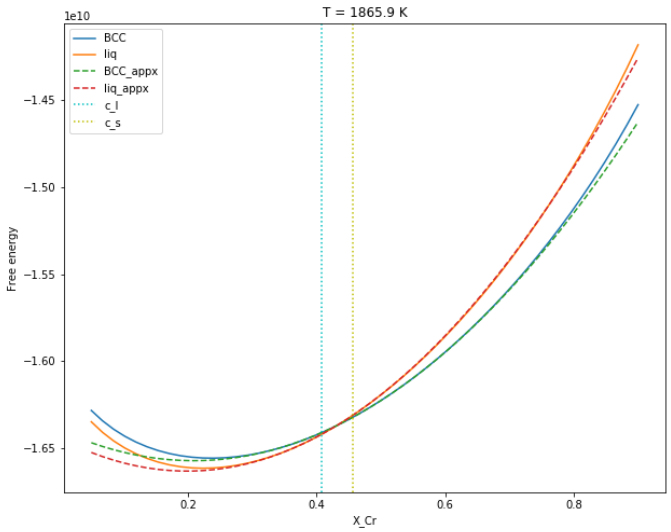


FIGURE 3.1: Parabolic approximation to the δ-ferrite and liquid phase free energies. The equilibrium compositions c_s and c_l are shown in the figure as dotted vertical lines in cyan and yellow. Figure was generated by the python code given in A.2.

3.1 EVALUATION OF COEFFICIENT FUNCTIONS

The parabolic approximation needs to satisfy certain conditions to be considered a good approximation. Here, c_s and c_l denote the equilibrium compositions of the solid and liquid phase respectively at a temperature T. The original free energy function is denoted by $f(c,T)$.

The approximations to solid and liquid free energies $p_s(c,T)$ and $p_l(c,T)$ must have the same *value* 3.4 , *slope* 3.5 and *curvature* 3.6 as the original functions $f_s(c,T)$ and $f_l(c,T)$ at the equilibrium compositions c_s and c_l respectively. This ensures that the phase diagrams produced by the two functions are identical and have the same chemical potential at equilibrium compositions.

$$p_{s,l}(c_{s/l},T) = f_{s,l}(c_{s/l},T) \tag{3.4}$$

$$\left.\frac{\partial p_{s,l}(c,T)}{\partial c}\right|_{c_{s/l}} = \left.\frac{\partial f_{s,l}(c,T)}{\partial c}\right|_{c_{s/l}} \tag{3.5}$$

$$\left.\frac{\partial^2 p_{s,l}(c,T)}{\partial c^2}\right|_{c_{s/l}} = \left.\frac{\partial^2 f_{s,l}(c,T)}{\partial c^2}\right|_{c_{s/l}} \tag{3.6}$$

The python code for the parabolic approximation is given in the first appendix A.2. Figure 3.1 shows the parabolic approximation generated by this code.

The values for a_0,a_1 and a_2 at various temperatures between $1790-2080K$ (the range of solidus and liquid) were obtained by applying these conditions at each temperatures. Then, a 7th order polynomial in T was fit to these values using SciDavis software to obtain expressions for $a_0(T)$,

$a_1(T)$ and $a_2(T)$. It should be noted that while such a high order polynomial fit can interpolate well in the given temperature range, we cannot use the expressions to extrapolate beyond the temperature range due to overfitting. Hence, our simulations need to be within this temperature range.

3.2 Evolution equations

The parabolic approximation leads to simple expressions of the evolution equations for the phase field ϕ and chemical potential field μ given in 2.3.3. Substituting the expression in 3.1 into 2.23, we get the following evolution equation for the phase field ϕ.

$$\tau\epsilon\frac{d\phi}{dt} = \epsilon\sigma\nabla^2\phi - \frac{1}{\epsilon}2W\sigma\phi(1-\phi) - (\Psi_s - \Psi_l)6\phi(1-\phi) \tag{3.7}$$

Similarly, substituting the expression in 3.1 into 2.25, we get the following evolution equation for the chemical potential field μ.

$$\left[\frac{h_s(\phi)}{2a_2^s} + \frac{[1-h_s(\phi)]}{2a_2^l}\right]\frac{d\mu}{dt} = \frac{D_l}{2a_2^l}[1-g_s(\phi)]\nabla^2\mu - 6(c_s - c_l)\phi(1-\phi)\frac{d\phi}{dt} \tag{3.8}$$

Here, a_2^s and a_2^l are the leading coefficients of the parabolic approximation to the solid and liquid free energies respectively. c_s and c_l are the equilibrium solid and liquid compositions. D_l is the diffusivity in the liquid. The diffusivity in solid D_s is assumed to be zero. These are the expressions used in the 1D isothermal solidification simulations.

4

1D Isothermal solidification

The next step after obtaining the parabolic free energy equations was to run a 1D simulation of solidification under isothermal conditions in the Fe-Cr system. This essentially involves to solving the evolution equations for the phase field 3.7 and chemical potential field 3.8. Since this is a 1D simulation, we don't consider the anisotropy effects discussed in 2.4. Instead, we take the anisotropy function a_c to be equal to unity.

$$a_c(\hat{n}) = 1 \tag{4.1}$$

In this chapter the methods used in the 1D simulation are discussed, followed by the results of the simulations. The 1D simulations were done mainly to verify the working of the code and whether the parabolic free energy functions behave as expected, before moving on to the more resource intensive 2D simulations of dendrites. Since inter-dendritic segregation is the focus of this work, we cannot expect useful results just from 1D simulations because we cannot have dendrites in a 1D simulation.

4.1 Methods

The 1D isothermal solidificetion simulation code was written in Python using Jupyter notebooks. Appendix A.3 contains these notebooks. In this section we discuss the details of the 1D simulation, namely the discretization schemes used, boundary conditions and simulation parameters like timestep, simulation box size, etc...

4.1.1 Discretization Schemes

The evolution equations are solved explicitly using a finite difference method. The gradient is computed by a central-difference scheme.

$$\nabla_j f(x) = \frac{f(x_{j+1}) - f(x_{j-1})}{2\Delta x} \tag{4.2}$$

Here, $\nabla_j f(x)$ is the gradient of the function $f(x)$ computed at the j^{th} node in the simulation box. $f(x_j)$ is the value of the function computed at the j^{th} node. Δx is the cell size. The laplacian is also computed by a central difference scheme.

$$\nabla_j^2 f(x) = \frac{f(x_{j+1}) - 2f(x_j) + f(x_{j-1})}{(\Delta x)^2} \tag{4.3}$$

Where $\nabla_j^2 f(x)$ is the laplacian calculated at the j^{th} node. At each timestep, the ϕ and μ fields for the next timestep are calculated as follows.

$$\phi(t + \Delta t) = \phi(t) + \frac{\partial \phi}{\partial t}\Delta t \tag{4.4}$$

$$\mu(t + \Delta t) = \mu + \frac{\partial \mu}{\partial t}\Delta t \tag{4.5}$$

Where $\frac{\partial \phi}{\partial t}$ and $\frac{\partial \mu}{\partial t}$ are given by the evolution equations 3.7 and 3.8 respectively. The spatial derivative terms in those equations are calculated by the expressions in 4.2 and 4.3.

4.1.2 Boundary Conditions

The simulation uses a zero-flux boundary condition (gradients at the boundary points are zero) for both the phase field ϕ and the chemical potential field μ, also called as the *Neumann boundary conditions*

4.1.3 Simulation parameters

For explicit methods, the timestep needs to be small in order to ensure stability. Different timesteps were tried and the largest timestep giving a stable solution was selected, $\Delta t = 0.5\mu s$. The Δx was selected according to the length scale of the problem as $1\mu m$. The simulation box was $300\mu m$ in length, corresponding to 300 cells.

4.2 Results

1D simulations were carried out with a starting composition of $X_{Cr}^l = 0.35$, which corresponds to the equilibrium liquid composition at $T_{eq} = 1831.8K$ (The equilibrium solid composition at that temperature being $X_{Cr}^s 0.388$. The simulation temperature was given by $T = T_{eq} - \Delta T$, and simulations were carried out for different values of ΔT. Figure 4.1 shows the composition and phase field profiles of one such simulation ($\Delta T = 20K$)at different timesteps. The code for this simulation can be found in A.3.

The interface velocities were also calculated for the different ΔT, and are tabulated in 4.1.

4.3 Discussion

Through the 1D simulations we were able to find out the optimal simulation parameters like Δt and Δx, and verify that our simulations are stable and give meaningful results. These optimised simulation parameters can then be used in 2D simulations. The interface velocity is expected to increase as the undercooling (and hence the driving force) increases, which is exactly what we observe in 4.1. The value of the interface velocity can be used to estimate the simulation

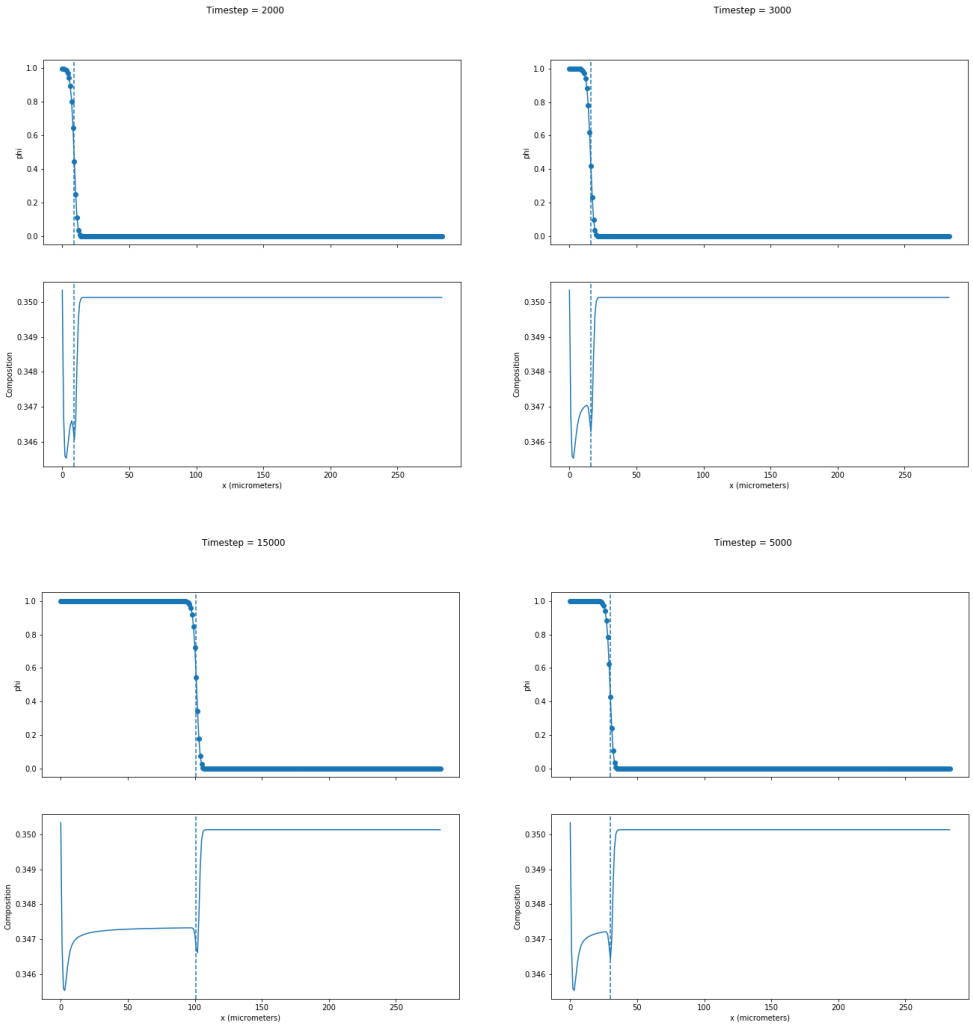


Figure 4.1: Phase field simulation results showing phase field and composition field at different timesteps. (Increasing clockwise from top left.) Simulation done for $\Delta T = 20K$, with starting composition $c_l = 0.35$. The solid fraction is increasing with time, and the solidified phase is richer in Cr as solidification progresses. The dotted vertical line corresponds to the interface position.

ΔT **(K)**	**velocity (mm/s)**
15	5.15
20	6.775
25	9
30	10.7
40	14
50	17.4

Table 4.1: Interface velocities for different values of ΔT. Starting composition is $X^l_{Cr} = 0.35$

cell size and simulation time for our 2D simulations. This helps us save time and computational resources by avoiding unnecessarily large simulation box sizes.

5

2D ISOTHERMAL SOLIDIFICATION

The optimal values of the simulation parameters were found using the 1D simulations, as well as estimates of the interface velocity at different ΔT. This helped to set up the 2D simulations. The initial 2D simulations were done for isothermal conditions, following which simulations were to be done for non-isothermal conditions as seen in additive manufacturing processes like powder bed fusion. However, as mentioned before the work had to be stopped before this stage and only the isothermal simulations are presented here. In this chapter, we discuss the methods used in the simulation - the software, simulation parameters, etc. - and the results of some preliminary simulations.

5.1 Methods

For 1D simulations it was sufficient to write a Python code. The problem was computationally inexpensive, so the simulations could be run on a normal desktop computer in a matter of a few minutes. The 2D simulations are computationally more expensive (by at least three orders of magnitude for 300×300 cells compared to just 300 cells for the 1D simulation) so they had to be run parallelly on computing clusters. Furthermore, python being an interpreted language runs much slower than the same code written in C/C++. To solve these problems, it was necessary to rewrite the phase field simulation code in a faster programming language and to parallellize the code. Hence, the phase field simulations were run on OpenFOAM, a general purpose Computational Fluid Dynamics software written in C++. OpenFOAM has built-in support for parallelization, partial differential equations and building custom solvers. An introduction to OpenFOAM is given in the appendix B.1.

5.1.1 Discretization Schemes

The evolution equations for the phase field 2.27 and 3.8 have to be discretised here as well. The evolution equation for ϕ includes the anisotropy terms discussed in 2.4. OpenFOAM has various discretization schemes built-in for different derivatives, so it is easy to select from among them [12]. OpenFOAM uses the finite volume method (FVM) to solve partial differential equations, in contrast to the finite difference method employed in the 1D simulations. Here, the simulation space is discretised into non-overlapping volumes of unequal shapes. Thus, the derivatives cannot

be computed as in the finite difference methods 4.2 and 4.3. In FVM, the derivatives at the centres of a volume element are calculated by integrating the derivative (gradient, divergence or laplacian) through the entire volume. These volume integrals are then converted into surface integrals using the *Gauss theorem* 5.1 [6].

$$\int_V \nabla \star \phi dV = \oint_S \phi \star d\boldsymbol{S} \tag{5.1}$$

Here, ϕ is any tensor field. $\star$ represents any tensor product and $\nabla \star \phi$ is the corresponding derivative: divergence $\nabla.\phi$, gradient $\nabla\phi$ or curl $\nabla \times \phi$. $\boldsymbol{S}$ is the surface area vector.

5.1.1.1 GRADIENT

The gradient terms are discretized using the Gauss theorem 5.1 as follows

$$\int_V \nabla\phi dV = \oint_S \phi d\boldsymbol{S} = \sum_f \boldsymbol{S}_f \phi_f \tag{5.2}$$

Here,ϕ is a scalar field, f is an index that runs over the different faces of the volume element and $\boldsymbol{S_f}$ is the surface normal at the point where the centres of two adjacent volume elements meet their common face f. ϕ_f is the value of ϕ at this point, which is found by interpolating the values between the two cells. For this simulation, we have used linear interpolation for gradient terms.

5.1.1.2 DIVERGENCE

The divergence terms are discretised using the Gauss theorem 5.1 as follows

$$\int_V \nabla.\boldsymbol{\phi} dV = \oint_S \boldsymbol{\phi}.d\boldsymbol{S} = \sum_f \boldsymbol{S_f}.\boldsymbol{\phi_f} \tag{5.3}$$

Here $\boldsymbol{\phi}$ is a vector field. Other terms have the same meaning as in 5.2. We use linear interpolation to find ϕ_f for divergence terms.

5.1.1.3 LAPLACIAN

The laplacian terms are discretized using the Gauss theorem 5.1 as follows

$$\int_V \nabla.(\Gamma\nabla\phi) dV = \oint_S d\boldsymbol{S}.(\Gamma\nabla\phi) = \sum_f \Gamma_f \boldsymbol{S_f}.(\nabla\phi)_f \tag{5.4}$$

Γ is a placeholder term for any expression. Γ_f is its value at face f. If P is the centre of the cell where the laplacian is calculated and N is the centre of the neighboring cell sharing face f, the term $\boldsymbol{S_f}.(\nabla\phi)_f$ is calculated as follows (assuming the line connecting P and N $\mathbf{d}$) is orthogonal to the surface $\boldsymbol{S_f}$)

$$\boldsymbol{S_f}.(\nabla\phi)_f = |S_f| \frac{\phi_N - \phi_P}{|\mathbf{d}|} \tag{5.5}$$

Here ϕ_N and ϕ_P are the values of ϕ at the points N and P respectively. Linear interpolation in all cases.

Equation	Implicit Discretization Scheme	Explicit Discretization Scheme
Phase field (ϕ) evolution 2.27	$\frac{\partial\phi}{\partial t}$,$\nabla^2(a_c^2\phi)$	$\nabla.\frac{\partial a}{\partial\nabla\phi}$
Chemical Potential (μ) evolution 3.8	$\nabla^2\mu$, $\frac{\partial\mu}{\partial t}$	$\frac{\partial\phi}{\partial t}$

TABLE 5.1: Time discretization schemes for different terms in equations 2.27 and 3.8

5.1.1.4 TIME

The different time derivatives in 2.27 and 3.8 are discretised using implicit/explicit Euler schemes. To give clarity on exactly which derivatives are implicit and explicit, a code snipped from the `equations.H` file in B.2 has been reproduced here. This file contains the evolution equations to be solved 2.27 and 3.8. The derivatives beginning with `fvm::` are implicitly calculated and terms beginning with `fvc::` are explicitly calculated.

LISTING 5.1: Code snippet containing the main evolution equations.

```
...

fvScalarMatrix phiEqn
(
        omega*epsilon*fvm::ddt(phi)
        ==
        2.0*epsilon*sigma*fvm::laplacian(ac*ac,phi)
        + epsilon*fvc::div(da_dgradPhi)
        - 2*W/epsilon*sigma*phi*(1-phi)
        - (drivingForce)*6*phi*(1-phi)
        + 6*dimf*noise_mag*phi*phi*(1-phi)*(1-phi)*randNumber
);

...

fvScalarMatrix muEqn
(
        (0.5*h_phi/a2_s + 0.5*(1-h_phi)/a2_l)/dimf*fvm::ddt(mu)
        ==
        (0.5*D_l*(1-g_phi)/a2_l/dimf)*fvm::laplacian(mu)
        - 6*(c_s - c_l)*phi*(1-phi)*fvc::ddt(phi)
);

...
```

From the above code snippet, we see the discretization schemes used for different derivatives. This is tabulated in the table 5.1.

5.1.2 SOLVERS

Discretization is essentially a way of converting the partial differential equations to a system of linear equations. To solve these linear equations, we need to choose a solver. OpenFOAM has

a wide variety of built-in solvers. As seen in 5.1.1.4, the discretized evolution equations contain both implicit and explicit terms. Hence, we need solvers that can handle both implicit and explicit terms. The solver for each variable is declared in the `fvSolutions` file as mentioned in B.1. For both ϕ adn μ, we use the iterative Gauss-Siedel solver.

5.1.3 Simulation Parameters

The optimized timestep and grid cell size from the 1D simulations were used - $dt = 0.5\mu s$ and $dx = 1\mu m$. Simulations were run on a variety of grid sizes ranging from 300×300 to 1200×1200. The code was parallelized to run on 4 processors parallelly. All 2D simulations were done with a starting composition of $X_{Cr} = 0.4$, which corresponds to the equilibrium liquid composition at around $1860.4K$. The temperature field was uniform with a value of $1805K$, which corresponds to a ΔT of around $55K$. From 2.1, we expect the interface velocity to be of the order of 20mm/s (It is 17.4mm/s for $X_{Cr} = 0.35$). Infact, a 1D simulation was run under these conditions and the interface velocity was found to be 22mm/s. This helped decide the number of timesteps for simulations of different grid sizes.

5.1.3.1 Interface fluctuations

Dendrites evolve because of the instability in the solid-liquid interface. Due to this instability, property fluctuations in the solid-liquid interface grow, leading to dendrites. In simulations, the fluctuations must be added artificially. This is done through a "noise" term added to the right side of the phi evolution equation 2.27. The noise term N is given in 5.6

$$N = 6N_m\phi^2(1-\phi)^2 r \tag{5.6}$$

Here, N_m is the noise magnitude denoted as the variable `noise_mag` in the code snippet in 5.1.1.4. r is a random number sampled from a uniform distribution between 0 and 1. $6\phi^2(1-\phi)^2$ has a similar form to the double well potential 2.18 and vanishes at $\phi = 0, 1$. This ensures that the fluctuations are restricted to the interface regions of the simulation. $N_m = 0.003$ was taken as the noise magnitude for the simulations presented here.

5.1.4 Boundary conditions

Neumann boundary conditions (zero gradient at boundary) were used for the phase field ϕ, composition field c and the chemical potential μ.

5.2 Results

One of the initial simulations is shown in the figure 5.1. The initial solid phase is in the shape of a quadrant of a circle in the bottom left corner. The effects of surface anisotropy can be seen from the morphology of the solidified phase - interface velocity is faster along [100] directions, leading to a higher curvature. However, dendrites are not formed. The simulations were repeated for larger grid sizes and done with the initial solid phase as a circular region in the centre as shown in figure 5.2. These did not lead to dendrites either. Composition was plotted across the interface for the simulation with the solid phase in the center.

Simulations were run for an increased value of the noise magnitude N_m given in 5.6. However, these simulations resulted in unstable solutions that gave unphysical values of ϕ that were greater than 1.

5.3 Discussion

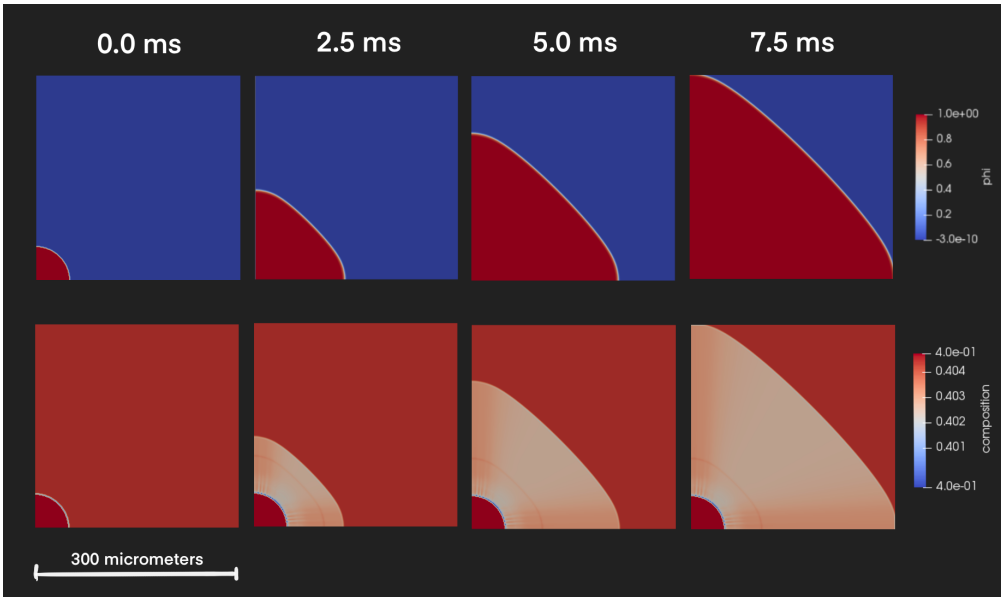


Figure 5.1: Colourmaps showing the values of phase field ϕ (top) and composition field c (bottom) at different times in the 2D isothermal solidification simulation from the corner. The simulation box size is $300\mu m \times 300\mu m$. The effect of surface energy anisotropy can be seen from the variation in curvature of the interface. However, dendrites are not formed. The scale bars for the heatmap are shown on the right.

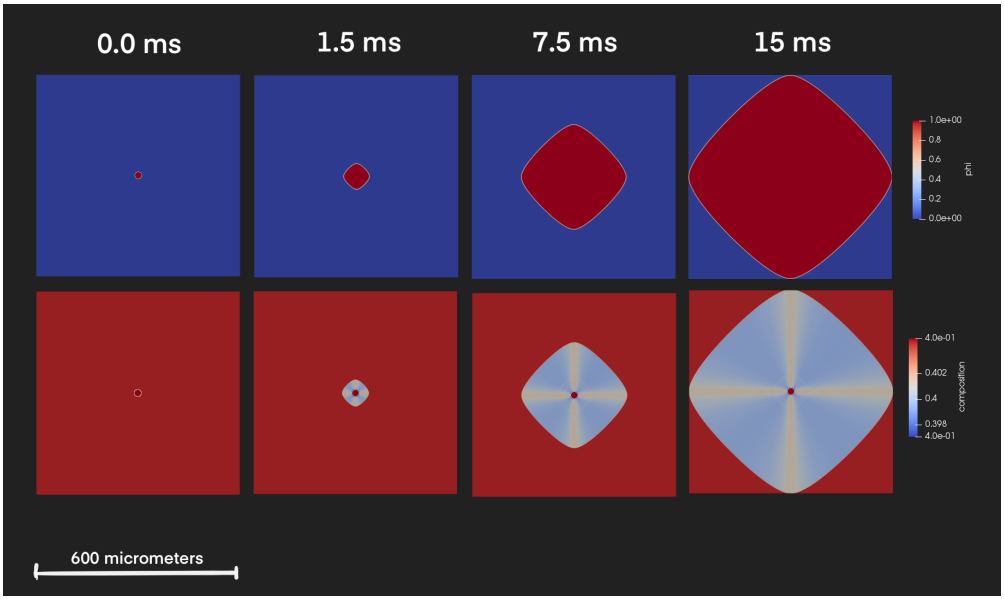


Figure 5.2: Phase field ϕ (top) and composition field c (bottom) at different times in the 2D isothermal solidification simulation from the centre. The simulation box size is $600\mu m \times 600\mu m$. The composition across the <100> (y-axis in this figure) and <110> interfaces are plotted in5.3

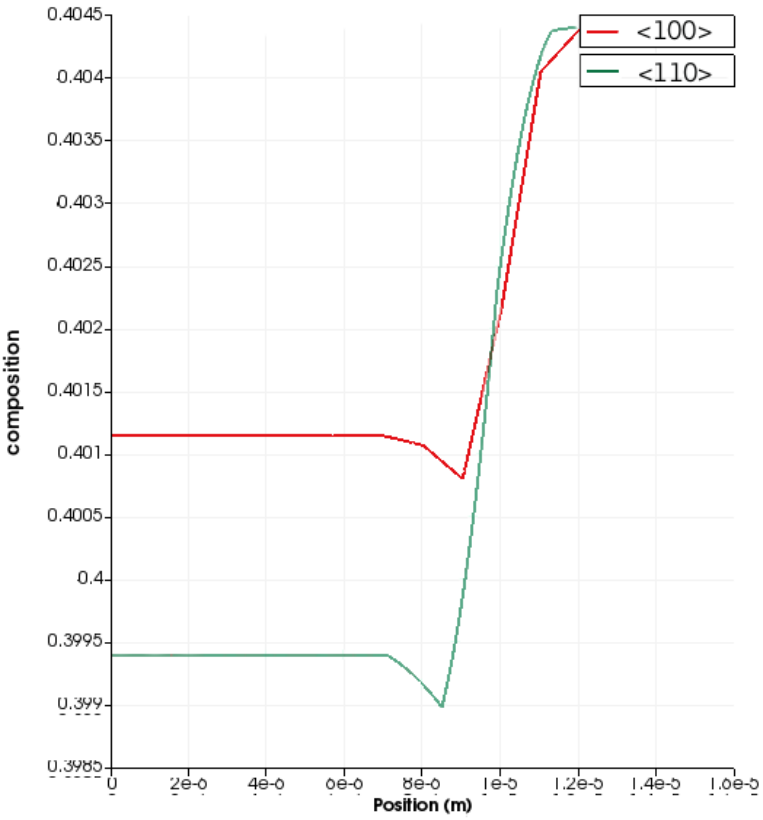


Figure 5.3: Composition plots across the <100> and <110> interfaces in the simulation given in 5.2 at $t = 7.5ms$.

When the initial simulation shown in 5.1 and 5.2 did not give dendrites, it was thought that the dendrites simply need more time to evolve. Hence, simulations were done on larger grids for more timesteps, but longer simulations still failed to evolve dendrites. The next step was to increase the magnitude of the fluctuations at the interface to make it more unstable. However, only one other higher value of the noise magnitude N_m was tried and that simulation gave runaway solutions that numerically exploded, giving values of ϕ and c greater than 1. There is possibly an optimal value of N_m that would've made the interface unstable enough to form dendrites without causing runaway solutions, but the work had to be stopped before the noise magnitude could be optimised.

We still gain some useful information from these simulations, especially the concentration profiles across the interface. Figure 5.3 shows the composition plotted across the $< 100 >$ and $< 110 >$ interfaces. The liquid ahead of the $< 110 >$ interface is richer in Chromium than at the $< 100 >$ interface. This can also be seen in the colourmap 5.2. This observation can be explained by the Gibbs-Thompson effect. For phase diagrams with a positive liquidus slope, a positive curvature of the solid-liquid interface favors the segregation of the solute atoms into the liquid, increasing the composition of the liquid just ahead of the interface.

6

Work Planned and Problems Faced

This chapter discusses the work that was planned for the next 2 months, had the project not been brought to an abrupt halt by a global pandemic. It also discusses the work that was planned but couldn't be done due to the optimistic attitude one has while planning that fails to predict the messiness of reality. This project was started with an aim of simulating inter-dendritic segregation in the Fe-Cr-Ni ternary system, at compositions similar to commonly used Austenitic stainless steel grades like 316L. The work was initially planned in the following different stages:

1. Find good models for the free energy of the phases in the Fe-Cr binary system
2. Derive phase field and composition field evolution equations as per the Grand Potential formulation
3. Simulate solidification in this binary system and calculate the inter-dendritic segregation
4. Find good models for the free energies of different phases in the Fe-Cr-Ni ternary system
5. Modify the phase field and composition field evolution equations to simulate the Fe-Cr-Ni ternary system
6. Calculate inter-dendritic segregation

The first step was straightforward, because there is no dearth of information on thermodynamics of ferrous alloys. The first roadblock was faced in the second step of the grand plan, i.e. deriving the evolution equations according to the grand potential formulation. As seen in 2.3.3, the chemical potential needs to be a monotonic function of the concentration to ensure that it is invertible. The chemical potential calculated from the free energy expressions in [2] were not invertible in c. Quite some time was spent in trying to find an analytical inverse before realising the expression was non-invertible. This led to the parabolic approximation. This was supposed to be an easy step, but a small typo in the Andersson et.al. paper in one of the free energy expressions took a good couple of days to discover.

The 1D simulations were pretty straightforward, but porting from Python to OpenFOAM took more effort than anticipated. OpenFOAM has its own conventions and file structure geared

towards Computational Fluid Dynamics simulations, so figuring out how to write a phase field solver involved learning the finite volume method and the inner workings of OpenFOAM. No complaints here though, this was really exciting.

It took two months by the time a working phase field solver was written and the first 2D simulations were started. Then came the biggest hurdle - dendrites did not evolve! This was the first time progress seemed to plateau, and it felt like it was going nowhere.An entire week was spent tuning different simulation parameters to get dendrites. This is when the COVID situation became bad, and the work had to be suspended. The lab was shut down, so I did not have remote access to my lab computer, which had all the code I wrote. I was very lucky that Kartik decided to stay back in campus because he was able to facilitate remote access to the lab computer, right in time for me to start writing this thesis. I am grateful for the way things turned out, things could've been much worse.

7

Conclusions and Future Work

Most of the groundwork has been done for carrying out phase field simulations of microstructural evolution in Austenitic stainless steels. A working, parallelized phase field code was written for the Iron-Chromium system that uses parabolic free energy functions. However, the parameters in the code still need tuning to capture dendritic growth during solidification of δ-ferrite in Fe-Cr alloys. After this is done, a few more steps remain to calculate interdendritic growth during additive manufacturing of austenitic stainless steels, as we set out to do at the beginning of this work. These steps are listed in the following section, along with the future direction this work could take.

7.1 Future Work

The initial work planned is given in chapter 6. The work had to be stopped at step 3 (simulate solidification in the Fe-Cr system and calculate inter-dendritic segregation) as listed in that chapter. The following subsections explore in some detail the work remaining to finish the initial plan and also some future directions for this research after the initial plan.

7.1.1 Parameter Tuning for Dendritic Growth in Fe-Cr binary

The immediate next step if one were to pick up where this work left off, is the tuning of the simulation parameters in order to facilitate evolution of dendrites in the simulation such as:

Magnitude of interface fluctuations (N_m): A possible reason for dendrites not growing is the interface fluctuations having a low magnitude. However, too high a value can lead to unstable solutions. Hence, an optimal value must be found.

Interface width: Another possible reason could be that the interface width does not match the length scale in the problem. Hence, different interface widths need to be tried.

Simulation geometry: We have run simulations with a curved solid-liquid interface. This causes a curvature-dependant lowering of the melting point of the solid and changes the equilibrium compisitions at the interface. We could remove these effects by using a flat interface as the starting point for the simulations.

7.1.2 Interdendritic segregation in Fe-Cr-Ni ternary system

After calculating inter-dendritic segregation for solidification in the Fe-Cr binary system by tuning simulation parameters, the next step would be to do it for the ternary Fe-Cr-Ni system. As in the binary system, this would involve finding suitable expressions for free energies of different phases from CALPHAD literature, parabolic approximation of those free energies, and deriving multicomponent evolution equations through the grand potential formulation. We also need to extend the range of application of the parabolic free energies from $1790 - 2080K$, since this is a very narrow range of temperatures compared to what is seen in actual AM processes.

7.1.3 Input from Additive Manufacturing Process Models

After developing a working phase field model for the ternary system and running isothermal solidificaiton simulations, the next step is to change the temperature fields as seen in additive manufacturing process simulations, using continuum-scale techniques such as the Finite Element Method (FEM). These were already being done in the lab.

Another (albeit a bit ambitious) direction this work could take is the multi-scale integration of the phase-field and FEM simulations, with one providing input for the simulation parameters of the other. That is, the concentration dependent material paremeters in the continuum simulations can be better modeled if one knows the segregation at the mesoscale in the system and the temperature fields calculated using these parameters can then be fed back into the continuum scale model to get better results for segregation, etc...

7.2 Learning Outcomes and Concluding Remarks

Perhaps the most important learning outcome of this project was the fact that research is a whole different ball game than coursework. Plans fail for various different reasons and failure is an integral part of the research workflow. Persevering in spite of repeated failure requires mental fortitude which is a requisite trait in every researcher - I learnt this the hard way.

Through this project I was able to learn and appreciate the implementation of the Finite Volume Method in commercial-grade software like OpenFOAM. Learning about the handling of the geometric mesh data and the integration schemes was an illuminating experience. The computational methods taught in computational modelling courses used finite differences for spatial discretization, so FVM was a paradigm-shift in numerically solving partial differential equations.

In computational modelling courses, examples and assignments on alloy systems usually involve simple expressions for the free energies and consequently the phase field and composition field evolution equations are simple. The complicated expressions of the free energies and inclusion of anisotropy led to added complexities and these had to be resolved through the parabolic approximation, adding anisotropy terms to the surface energy, etc... Thus, I was exposed to the various complications that arise when simulating complex alloy systems that differ from ideal behaviour.

This project introduced me to the fascinating field of Additive Manufacturing and I was able to truly appreciate the impact AM will have on the world and the different hurdles that remain in the way of realising its true potential. We are in the midst of a revolution in manufacturing and supply chain logistics, and I am extremely glad to be a part of it, however small.

A

Python codes

The first appendix contains codes written in python. The codes were written as Jupyter notebooks, so the entire notebooks have been converted to pdf via LaTeX and attached to this thesis.

A.1 Iron-Chromium Phase Diagram calculation

Given below is the python code written to verify the free energy functions given by Andersson and Sundman [2]. The phase diagram was calculated from the free energy functions and plotted to see if they match the phase diagram given in 2.3. The solidus and liquidus were plotted, along with the γ-loop. These regions of the phase diagram were found to match. The solidus and liquidus compositions in the temperature range $1750 - 2200K$ were saved in the file `eq_data.csv` in comma-seperated values format. This file is used in later codes.

Fe-Cr_PhaseDiagam

June 26, 2020

0.1 Iron-Chromium Phase diagram calculation

This is done to verify the free energy functions given in the Andersson-Sundman paper

The molar free energies are given in the format:

$$G_m = X_{Cr}G^o_{Cr} + X_{Fe}G^o_{Fe} + RT(X_{Cr}ln(X_{Cr}) + X_{Fe}ln(X_{Fe})) + G^{xs}_m + G_M$$

Where:

G^{xs}_m is the excess free energy given as a Redlich-Kister polynomial

$$G^{xs}_m = X_{Cr}X_{Fe}\sum_{\nu=0} L^{\nu}_{Cr,Fe}(X_{Cr} - X_{Fe})^{\nu}$$

And G_M is the magnetic contribution to the free energy given by

$$G_M = RTln(1+\beta)f(\tau)$$

Where $\tau = \frac{T}{T_c}$

This contribution is considered only for the bcc phase. For the fcc phase, this is negligible.

[1]:
```
import numpy as np
import scipy.optimize as opt
import matplotlib.pyplot as plt
```

[2]:
```
# Define standard gibbs energies of components

R = 8.31448

def G0_Cr_bcc(T):

    # -439.0 to account for difference in standard enthalpies

    return (T<2180)*(-439.0 -8851.93 + 157.48*T -26.908*T*np.log(T) + 1.89435E-3
↪* T**2 -1.47721E-6*T**3 + 139250/T) +\
    (T>=2180)*(-34864+344.18*T-50*T*np.log(T)-2.88526E32/T**9)
```

```
# T < 1811
def G0_Fe_bcc(T):

    return (T<1811)*(1224.83 + 124.134*T - 23.5143*T*np.log(T) -4.3975E-3 * T**2
 ↪- 5.89269E-8*T**3 + 77358.5/T) +\
    (T>=1811)*(-25384.451+299.31255*T -46*T*np.log(T) +2.2960305E31/T**9)

def G0_Fe_fcc(T):

    return (T<1811)*(-237.57 + 132.416*T - 24.6643*T*np.log(T) -3.75752E-3*T**2
 ↪-5.89269E-8*T**3 + 77358.5/T)+\
    (T>=1811)*(-27098.266+300.25256*T - 46*T*np.log(T) + 2.78854E31/T**9)


def G0_Cr_fcc(T):

    return G0_Cr_bcc(T) + 7284+0.163*T

def G0_Fe_liq(T):

    return (T<1811)*(G0_Fe_bcc(T) + 12040.17 - 6.55843*T - 3.6751551E-21*T**7) +\
    (T>=1811)*(-10839.7+291.302*T-46*T*np.log(T))

def G0_Cr_liq(T):

    return (T<2180)*(G0_Cr_bcc(T) + 24335.93 - 11.42*T + 2.37615E-21*T**7) +\
    (T>=2180)*(-16459+335.618*T-50*T*np.log(T))


# Excess terms

def Gxs_bcc(x,T):

    return x*(1-x)*(20500-9.68*T)

def Gxs_fcc(x,T):

    return x*(1-x)*(10833-7.477*T+1410*(2*x-1))

def Gxs_liq(x,T):

    return x*(1-x)*(-14550+6.65*T)

# Magnetic Term

def GM(x,T):
```

```
    Tc = 1043*(1-x) - 311.5*x + x*(1-x)*(1650+550*(2*x-1))
    t = T/Tc
    b = 2.22*(1-x) - 0.008*x -x*(1-x)*0.85

    # different equations for t>1 and t<1

    f = (t>1)*(-6.417E-2/t**5 - 2.037E-3/t**15 -4.278E-4/t**25) +\
    (t<=1)*(-0.90530/t +1.0 + 0.153*t**3 - 6.8E-3*t**9 - 1.53E-3*t**15)


    return R*T*np.log(b+1)*f
```

[3]:
```
# Calculate Free energies

def G_bcc(x,T):

    return x*G0_Cr_bcc(T) + (1-x)*G0_Fe_bcc(T) + R*T*(x*np.log(x)+(1-x)*np.
 ↪log(1-x))+ Gxs_bcc(x,T) + GM(x,T)

def G_fcc(x,T):

    return x*G0_Cr_fcc(T) + (1-x)*G0_Fe_fcc(T) + R*T*(x*np.log(x)+(1-x)*np.
 ↪log(1-x))+ Gxs_fcc(x,T)

def G_bcc_nomag(x,T):

    return x*G0_Cr_bcc(T) + (1-x)*G0_Fe_bcc(T) + R*T*(x*np.log(x)+(1-x)*np.
 ↪log(1-x))+ Gxs_bcc(x,T)

def G_liq(x,T):

    return x*G0_Cr_liq(T) + (1-x)*G0_Fe_liq(T) + R*T*(x*np.log(x)+(1-x)*np.
 ↪log(1-x)) + Gxs_liq(x,T)
```

[5]:
```
from ipywidgets import interact, interactive, fixed, interact_manual
import ipywidgets as widgets
```

[6]:
```
@interact(T=(800,2400,50))
def plotfn(T):
    fig,ax = plt.subplots(figsize=(8,6))

    ax.plot(xs,G_bcc(xs,T),label='BCC')
    ax.plot(xs,G_bcc_nomag(xs,T),label='BCC_nomag',linestyle='--')
    ax.plot(xs,G_fcc(xs,T),label='FCC')
    ax.plot(xs,G_liq(xs,T),label='liq')
    ax.legend()
```

```
interactive(children=(IntSlider(value=1600, description='T', max=2400, min=800, step=50), Output
```

Alright, the free energies seem to make sense. Onto plot the phase diagrams, for confirmation.

[7]:
```
# Expressions for derivatives

def dGdx_bcc(x,T):

    Tc = 1043*(1-x) - 311.5*x + x*(1-x)*(1650+550*(2*x-1))
    t = T/Tc

    b = 2.22*(1-x) - 0.008*x -x*(1-x)*0.85
    f = (t>1)*(-6.417E-2/t**5 - 2.037E-3/t**15 -4.278E-4/t**25) +\
    (t<=1)*(-0.90530/t +1.0 + 0.153*t**3 - 6.8E-3*t**9 - 1.53E-3*t**15)

    dbdx = -2.228 + (2*x-1)*0.85
    dGMdx = R*T/(b+1)*f*dbdx

    return G0_Cr_bcc(T) - G0_Fe_bcc(T) + R*T*(np.log(x/(1-x))) +␣
 ↪(1-2*x)*(20500-9.68*T) + dGMdx

def dGdx_fcc(x,T):

    return G0_Cr_fcc(T) - G0_Fe_fcc(T) + R*T*(np.log(x/(1-x))) +\
    (1-2*x)*(10833-7.477*T+1410*(2*x-1)) + x*(1-x)*2*1410
```

0.2 Gamma loop, (1600-1200K)

[8]:
```
def opt_fn(x_vec,T):

    x_fcc = x_vec[0]
    x_bcc = x_vec[1]

    f1 = dGdx_fcc(x_fcc,T) - dGdx_bcc(x_bcc,T)
    f2 = G_fcc(x_fcc,T) - G_bcc(x_bcc,T) - dGdx_fcc(x_fcc,T)*x_fcc +␣
 ↪dGdx_bcc(x_bcc,T)*x_bcc

    return f1,f2
```

[9]:
```
opt.fsolve(opt_fn, [0.12,0.13],1300)
```

[9]:
```
array([0.12109292, 0.1464804 ])
```

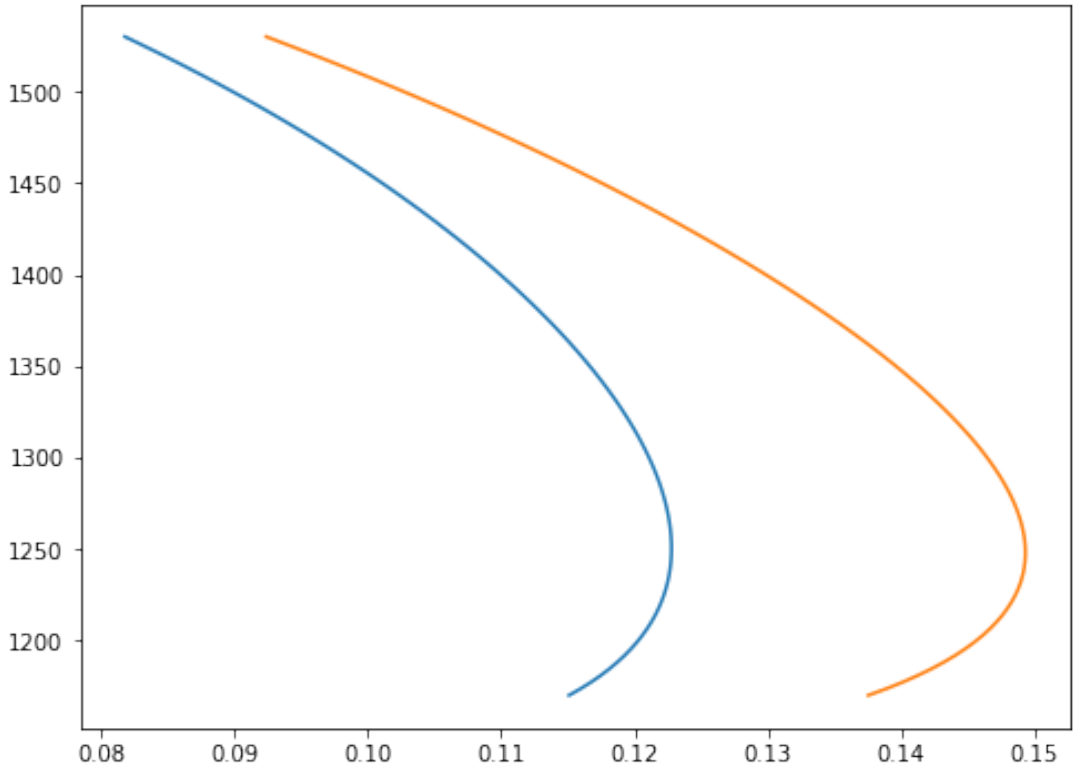
[10]:
```
Ts = np.linspace(1170,1530,100)
results=[]
```

```
for T in Ts:

    soln = opt.fsolve(opt_fn, [0.12,0.13],T)
    results.append(soln)
```

[11]:
```
results = np.array(results)
```

[12]:
```
fig,ax = plt.subplots(figsize = (8,6))

ax.plot(results[:,0],Ts)
ax.plot(results[:,1],Ts)
```

[12]: [<matplotlib.lines.Line2D at 0x7f859d67fc90>]



0.3 Solidus/Liquidus (1750-2200K)

[13]:
```
def dGdx_liq(x,T):

    return G0_Cr_liq(T) - G0_Fe_liq(T) + R*T*(np.log(x/(1-x))) +\
    (1-2*x)*(-14450+6.65*T)
```

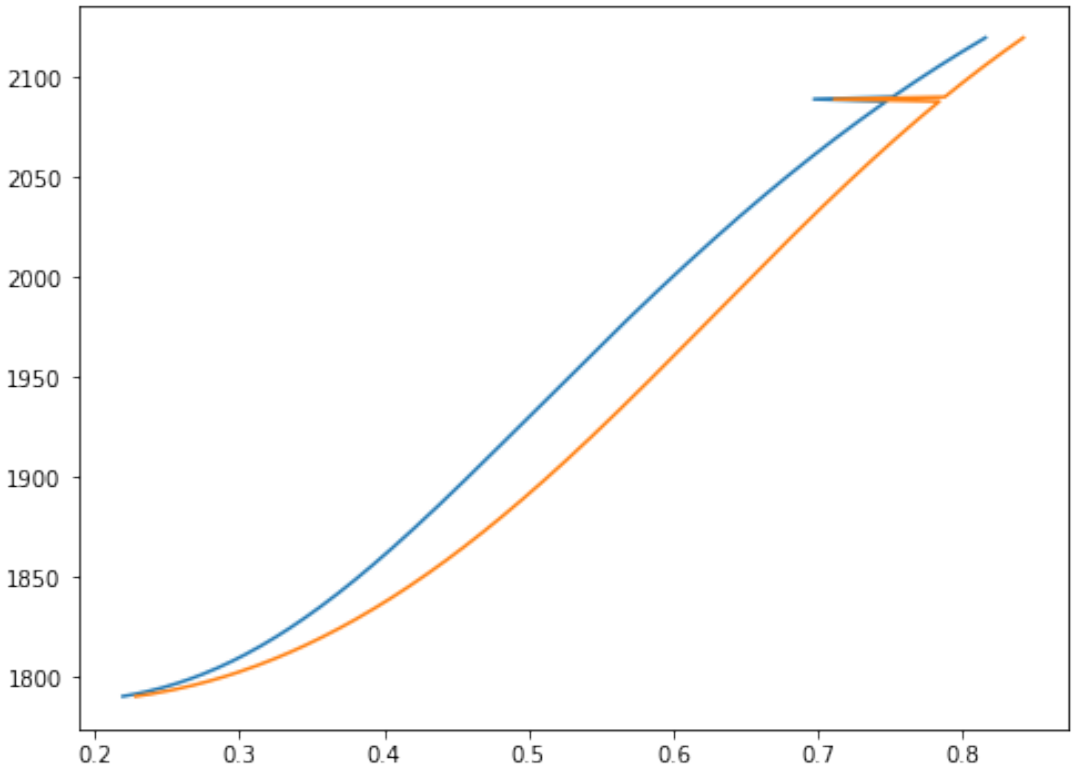
[14]:
```
def opt_fn_liq(x_vec,T):

    x_liq = x_vec[0]
    x_bcc = x_vec[1]

    f1 = dGdx_liq(x_liq,T) - dGdx_bcc(x_bcc,T)
    f2 = G_liq(x_liq,T) - G_bcc(x_bcc,T) - dGdx_liq(x_liq,T)*x_liq +␣
 ↪dGdx_bcc(x_bcc,T)*x_bcc

    return f1,f2
```

[15]:
```
Ts = np.linspace(1790,2120,301)
results=[]

for T in Ts:

    soln = opt.fsolve(opt_fn_liq, [0.38+(T-1825)/225*0.27,0.4+(T-1825)/255*0.
 ↪3],T)
    results.append(soln)
```

[16]:
```
results = np.array(results)
```

[17]:
```
fig,ax = plt.subplots(figsize = (8,6))

ax.plot(results[:,0],Ts)
ax.plot(results[:,1],Ts)
```

[17]:
```
[<matplotlib.lines.Line2D at 0x7f859d5f2d90>]
```



1 Exporting values

[18]:
```
eq_data = np.hstack((np.array([Ts]).transpose(),results))
np.savetxt('eq_data.csv',eq_data,delimiter=',')
```

A.2 Parabolic Approximation to Free energies

Given below is the python code for the parabolic approximation of the free energies given in 2.2.2 in the $1750 - 2200K$ temperature range. The values for solidus and liquidus compositions are obtained from the file `eq_data.csv` The coefficients a_0, a_1 and a_2 (denoted as a, b and c in the code) for different temperatures are saved in the files `bcc_consts.csv` and `liq_consts.csv` for the δ-ferrite and liquid phases respectively. These constants were then fitted to a 7th order polynomial as discussed in 3.

Parabolic_approximation

June 18, 2020

1 Parabolic approximation to free energies

We gotts find $a(T), b(T), c(T)$ (for each phase) such that

$$p(X_{Cr}, T) = a(T)X_{Cr}^2 + b(T)X_{Cr} + c(T)$$

is a good approximation to the free energy density $f = \frac{G}{V_m}$.

Here, "good" approximation means satisfying the following conditions:

1. $$f(x_{s/l}, T) = p(x_{s/l}, T)$$

   .

2. $$\left.\frac{\partial f(x,T)}{\partial x}\right|_{x_{s/l}} = \left.\frac{\partial p(x,T)}{\partial x}\right|_{x_{s/l}}$$

   .

3. $$\left.\frac{\partial^2 f(x,T)}{\partial x^2}\right|_{x_{s/l}} = \left.\frac{\partial^2 p(x,T)}{\partial x^2}\right|_{x_{s/l}}$$

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
#import scipy.optimize as opt

from ipywidgets import interact, interactive, fixed, interact_manual
import ipywidgets as widgets
```

[2]:
```
# Define standard gibbs energies of components

R = 8.31448
V_m = 7.09e-6 #m^3/mol

def G0_Cr_bcc(T):

    # -439.0 to account for difference in standard enthalpies
```

```
    return (T<2180)*(-439.0 -8851.93 + 157.48*T -26.908*T*np.log(T) + 1.89435E-3*T**2 -1.47721E-6*T**3 + 139250/T) +\
    (T>=2180)*(-34864+344.18*T-50*T*np.log(T)-2.88526E32/T**9)

def G0_Fe_bcc(T):

    return (T<1811)*(1224.83 + 124.134*T - 23.5143*T*np.log(T) -4.3975E-3 * T**2 - 5.89269E-8*T**3 + 77358.5/T) +\
    (T>=1811)*(-25384.451+299.31255*T -46*T*np.log(T) +2.2960305E31/T**9)


def G0_Fe_liq(T):

    return (T<1811)*(G0_Fe_bcc(T) + 12040.17 - 6.55843*T - 3.6751551E-21*T**7) +\
    (T>=1811)*(-10839.7+291.302*T-46*T*np.log(T))

def G0_Cr_liq(T):

    return (T<2180)*(G0_Cr_bcc(T) + 24335.93 - 11.42*T + 2.37615E-21*T**7) +\
    (T>=2180)*(-16459+335.618*T-50*T*np.log(T))


# Excess terms

def Gxs_bcc(x,T):

    return x*(1-x)*(20500-9.68*T)

def Gxs_liq(x,T):

    return x*(1-x)*(-14550+6.65*T)

# Magnetic Term

def GM(x,T):

    Tc = 1043*(1-x) - 311.5*x + x*(1-x)*(1650+550*(2*x-1))
    t = T/Tc
    b = 2.22*(1-x) - 0.008*x -x*(1-x)*0.85

    # different equations for t>1 and t<1

    f = (t>1)*(-6.417E-2/t**5 - 2.037E-3/t**15 -4.278E-4/t**25) +\
    (t<=1)*(-0.90530/t +1.0 + 0.153*t**3 - 6.8E-3*t**9 - 1.53E-3*t**15)


    return R*T*np.log(b+1)*f
```

[3]:
```
# Free energy densities (in J/cm^3)

def f_bcc(x,T):

    return (x*G0_Cr_bcc(T) + (1-x)*G0_Fe_bcc(T) + R*T*(x*np.log(x)+(1-x)*np.
 ↪log(1-x))+ Gxs_bcc(x,T) + GM(x,T))/V_m

def f_liq(x,T):

    return (x*G0_Cr_liq(T) + (1-x)*G0_Fe_liq(T) + R*T*(x*np.log(x)+(1-x)*np.
 ↪log(1-x)) + Gxs_liq(x,T))/V_m
```

[4]:
```
# First derivatives (J/cm^4)

def dfdx_bcc(x,T):

    Tc = 1043*(1-x) - 311.5*x + x*(1-x)*(1650+550*(2*x-1))
    t = T/Tc

    b = 2.22*(1-x) - 0.008*x -x*(1-x)*0.85
    f = (t>1)*(-6.417E-2/t**5 - 2.037E-3/t**15 -4.278E-4/t**25) +\
    (t<=1)*(-0.90530/t +1.0 + 0.153*t**3 - 6.8E-3*t**9 - 1.53E-3*t**15)

    dbdx = -2.228 + (2*x-1)*0.85
    dGMdx = R*T/(b+1)*f*dbdx

    return (G0_Cr_bcc(T) - G0_Fe_bcc(T) + R*T*(np.log(x/(1-x))) +␣
 ↪(1-2*x)*(20500-9.68*T) + dGMdx)/V_m

def dfdx_liq(x,T):

    return (G0_Cr_liq(T) - G0_Fe_liq(T) + R*T*(np.log(x/(1-x))) +\
    (1-2*x)*(-14450+6.65*T))/V_m
```

[5]:
```
# Second derivatives (J/cm^4)

def d2fdx2_bcc(x,T):

    Tc = 1043*(1-x) - 311.5*x + x*(1-x)*(1650+550*(2*x-1))
    t = T/Tc

    b = 2.22*(1-x) - 0.008*x -x*(1-x)*0.85
    f = (t>1)*(-6.417E-2/t**5 - 2.037E-3/t**15 -4.278E-4/t**25) +\
    (t<=1)*(-0.90530/t +1.0 + 0.153*t**3 - 6.8E-3*t**9 - 1.53E-3*t**15)

    dbdx = -2.228 + (2*x-1)*0.85
    d2bdx2 = 2*0.85
```

```
    dGMdx = R*T/(b+1)*f*dbdx
    d2GMdx2 = -R*T/(b+1)**2 * f * dbdx**2 + R*T/(b+1)*f*d2bdx2

    return (R*T/(x*(1-x)) -2*(20500-9.68*T) + d2GMdx2)/V_m

def d2fdx2_liq(x,T):

    return (R*T/(x*(1-x)) -2*(-14450+6.65*T))/V_m
```

1.1 Solving for parabolic constants

$$p(X_{Cr}, T) = a(T)X_{Cr}^2 + b(T)X_{Cr} + c(T)$$

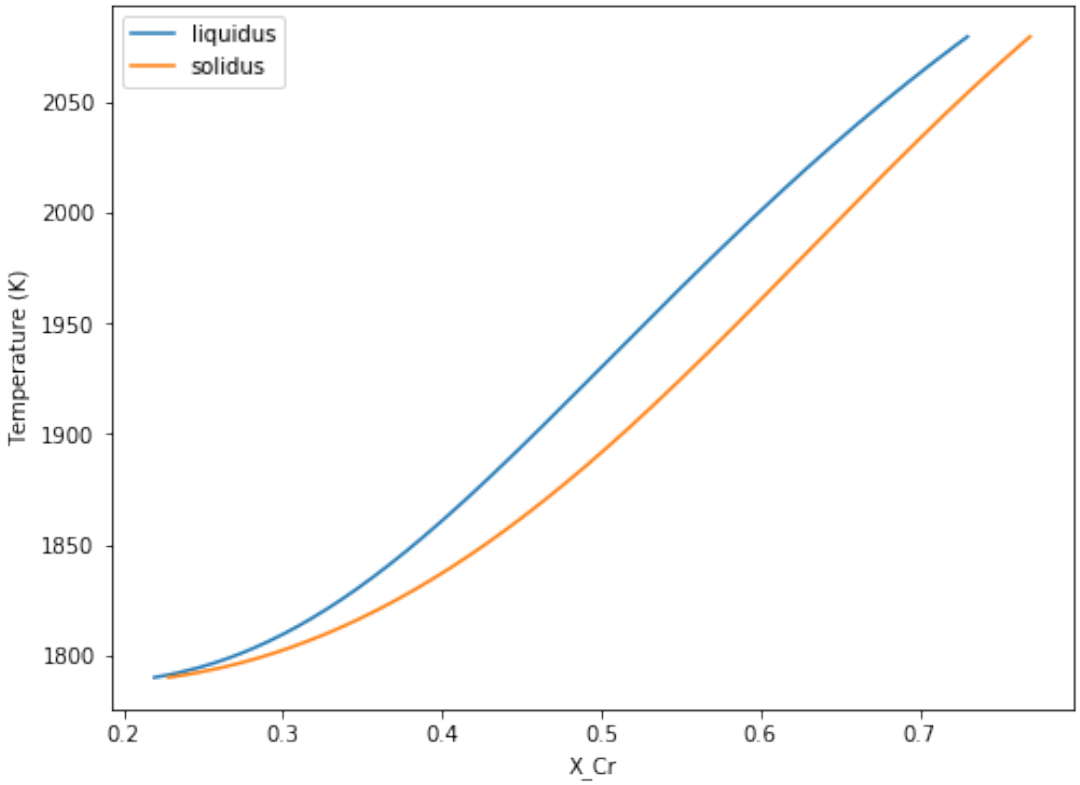
Let's take $x = X_{Cr}$

$$\frac{\partial p}{\partial x} = 2a(T)x + b(T)$$

$$\frac{\partial^2 p}{\partial x^2} = 2a(T)$$

[6]:
```
# load data and plot phase diagram

data = np.genfromtxt('eq_data.csv',delimiter=',')
data = data[data[:,0]<2080]

fig,ax = plt.subplots(figsize=(8,6))

ax.set_xlabel("X_Cr")
ax.set_ylabel("Temperature (K)")
ax.plot(data[:,1],data[:,0],label='liquidus')
ax.plot(data[:,2],data[:,0],label='solidus')
ax.legend()
```

[6]:
```
<matplotlib.legend.Legend at 0x7f6cb495a9d0>
```

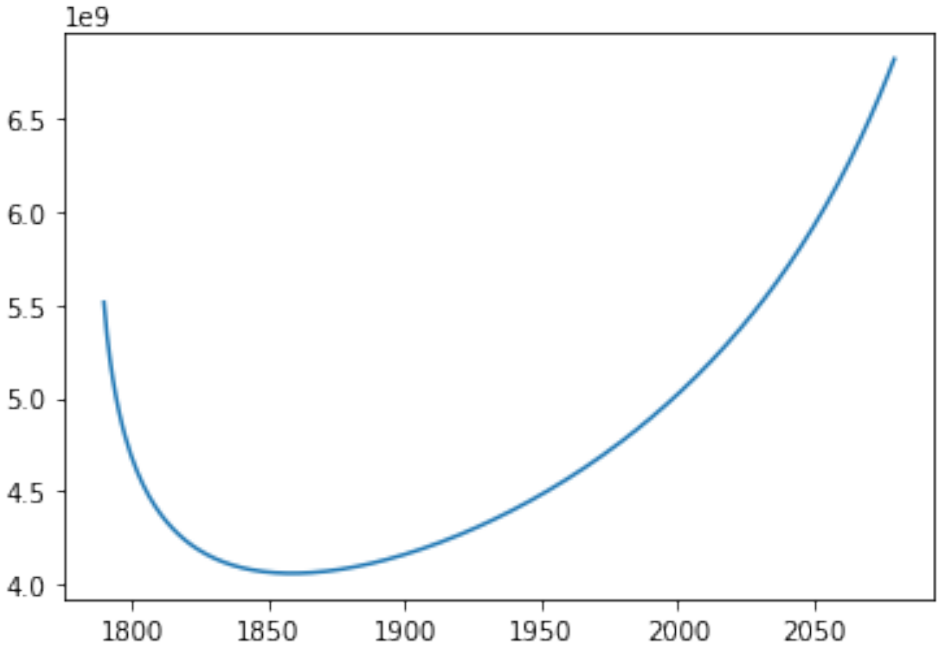


[7]:
```
bcc_consts = []
liq_consts = []

for pt in data:

    T = pt[0]
    x_l = pt[1]
    x_s = pt[2]

    #bcc
    a_bcc = d2fdx2_bcc(x_s,T)/2
    b_bcc = dfdx_bcc(x_s,T) - 2*a_bcc*x_s
    c_bcc = f_bcc(x_s,T) - a_bcc*x_s**2 - b_bcc*x_s

    #liquid
    a_liq = d2fdx2_liq(x_l,T)/2
    b_liq = dfdx_liq(x_l,T) - 2*a_liq*x_l
    c_liq = f_liq(x_l,T) - a_liq*x_l**2 - b_liq*x_l

    bcc_consts.append([T,a_bcc,b_bcc,c_bcc])
    liq_consts.append([T,a_liq,b_liq,c_liq])
```

[8]:
```
bcc_consts = np.array(bcc_consts)
liq_consts = np.array(liq_consts)
```

[9]:
```
plt.plot(bcc_consts[:,0],bcc_consts[:,1])
```

[9]:
```
[<matplotlib.lines.Line2D at 0x7f6cb3f498d0>]
```

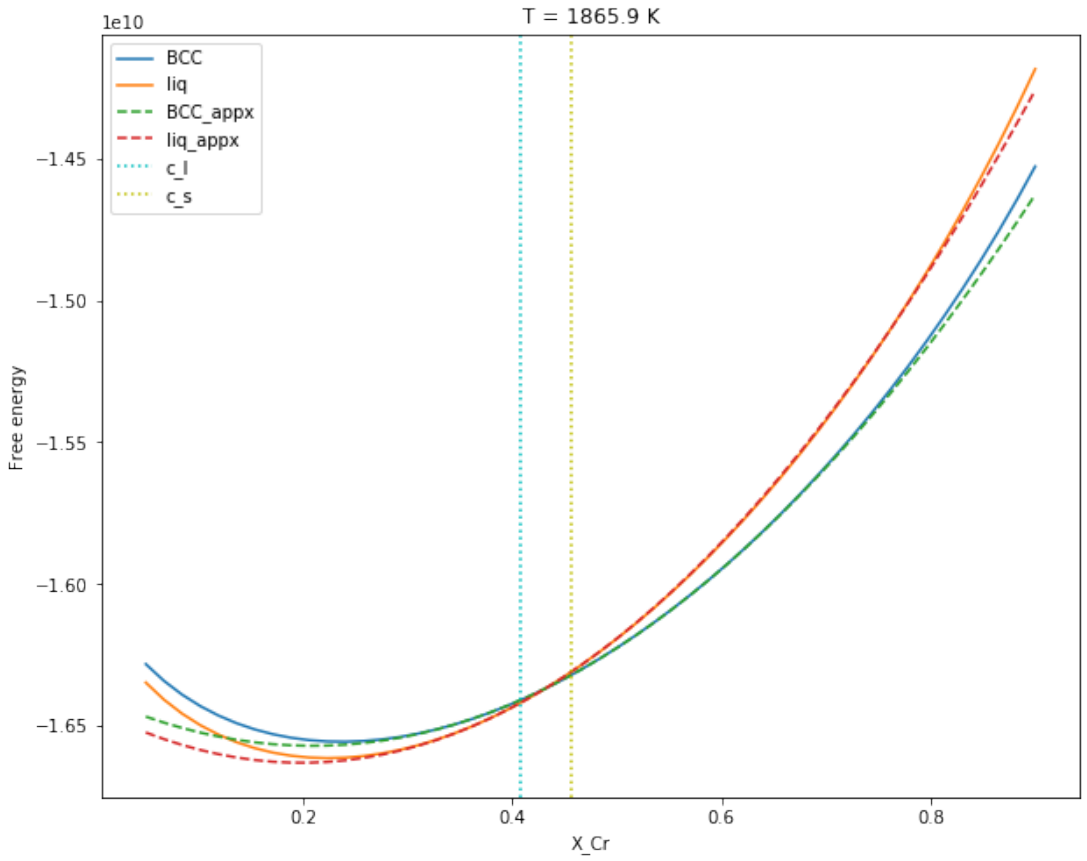


[10]:
```
# Interactive plot of G-X curves and parabolic approximation at different␣
 ↪temperatures
@interact(i=(0,100))
def plotfn(i):

    xs = np.linspace(0.05,0.9)
    T = bcc_consts[i][0]

    x_l = data[i][1]
    x_s = data[i][2]

    fig,ax = plt.subplots(figsize=(10,8))

    ax.plot(xs,f_bcc(xs,T),label='BCC')
    ax.plot(xs,f_liq(xs,T),label='liq')
    ax.plot(xs,np.polyval(bcc_consts[i][1:],xs),label='BCC_appx',linestyle='--')
    ax.plot(xs,np.polyval(liq_consts[i][1:],xs),label='liq_appx',linestyle='--')
```

```
    ax.set_xlabel("X_Cr")
    ax.set_ylabel("Free energy")
    ax.set_title("T = " + str(T) +" K")
    ax.axvline(x_l,label='c_l',linestyle='dotted',color='c')
    ax.axvline(x_s,label='c_s',linestyle='dotted',color='y')
    ax.legend()
```

```
interactive(children=(IntSlider(value=50, description='i'), Output()), _dom_classes=('widget-int
```

[11]:
```
# Plot G-X curves and parabolic approximation

plotfn(69)
fig.savefig("ParabolicAppx")
```



[12]:
```
np.savetxt('bcc_consts.csv',bcc_consts,delimiter=',')
np.savetxt('liq_consts.csv',liq_consts,delimiter=',')
```

1.2 Polynomial fits to the constants

The constants were fitted to a 7th order polynomial in T using SciDavis. **Note: This will only work for the given temperature range (1790-2080K). Polynomials extrapolate badly**

A.3 1D Isothermal Solidification

Given below is the python code for the phase field simulation of 1D isothermal solidification in the Fe-Cr binary system. The free energies are calculated from the 7th order polynomials in T fitted to the coefficients of the parabolas. The code also calculates the interface velocity.

1D_Isothermal_Solidification

June 22, 2020

0.1 Isothermal alloy solidification - (Fe-Cr)

Here, we simulate Fe-Cr alloy solidification using phase field method. We use the grand potential formulation to solve for the composition fields. Free energies taken from Fe-Cr_PhaseDiag.ipynb (parabolic approximations)

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

Free energy densities are of the form

$$f_{s/l} = a(T)x^2 + b(T)x + c(T) J/cm^3$$

Where $x = X_{Cr}$. Look to Parabolic_appx_2.ipynb for more details

[2]:
```
def a_bcc(T):

    a0  = 1.48573160876362e+17
    a1  = -533959921453932
    a2  = 822132803086.902
    a3  = -702980512.848244
    a4  = 360524.747722819
    a5  = -110.896060621478
    a6  = 0.0189435512365575
    a7  = -1.38632597908866e-06

    return a0+a1*T+a2*T**2+a3*T**3+a4*T**4+a5*T**5+a6*T**6+a7*T**7


def b_bcc(T):

    a0  = -5.74988758596428e+16
    a1  = 205888139888278
    a2  = -315807244126.921
    a3  = 268986846.012062
    a4  = -137396.890145104
    a5  = 42.0876190128756
    a6  = -0.00715869712372715
```

```
    a7  = 5.21561342179072e-07

    return a0+a1*T+a2*T**2+a3*T**3+a4*T**4+a5*T**5+a6*T**6+a7*T**7

def c_bcc(T):

    a0  = 2.24280665461153e+15
    a1  = -7672646510499.51
    a2  = 11198716987.0004
    a3  = -9033989.94168094
    a4  = 4346.56671776632
    a5  = -1.24600097323119
    a6  = 0.000196784418722745
    a7  = -1.31853064948386e-08

    return a0+a1*T+a2*T**2+a3*T**3+a4*T**4+a5*T**5+a6*T**6+a7*T**7


def a_liq(T):

    a0  = 1.34402214639946e+17
    a1  = -483041743694295
    a2  = 743752241655.028
    a3  = -635975330.764811
    a4  = 326169.32777634
    a5  = -100.331108793173
    a6  = 0.0171392878798394
    a7  = -1.25432210980197e-06

    return a0+a1*T+a2*T**2+a3*T**3+a4*T**4+a5*T**5+a6*T**6+a7*T**7

def b_liq(T):

    a0  = -5.02636054089711e+16
    a1  = 180002914938405
    a2  = -276136627305.307
    a3  = 235226732.682717
    a4  = -120167.335429361
    a5  = 36.8144381507289
    a6  = -0.00626257159618173
    a7  = 4.56330855404118e-07

    return a0+a1*T+a2*T**2+a3*T**3+a4*T**4+a5*T**5+a6*T**6+a7*T**7

def c_liq(T):

    a0  = 2.12467464521575e+15
```

```
    a1  = -7310151586822.15
    a2  = 10738897653.4874
    a3  = -8727402.7907034
    a4  = 4235.13227776304
    a5  = -1.22627109914018
    a6  = 0.00019598329226659
    a7  = -1.33213290371344e-08

    return a0+a1*T+a2*T**2+a3*T**3+a4*T**4+a5*T**5+a6*T**6+a7*T**7
```

[3]:
```
R = 8.314   #J/mol/K
T = 1810    #K
V_m = 7.09e-6 #m^3/mol

D_l = 3.2e-9 #m^2/s
sigma = 3.2 #J/m^2
```

[4]:
```
# Constants

W = 1
interface_width = 1e-5    #m
ep = interface_width/2.5
dx = interface_width/10

dt = 1e-6 #s
```

[5]:
```
# Calculate Relaxation coefficient

M = 0.063828
F = 0.158741


# Equilibrium Temperture
T_eq = 1.8318E+03

# Equilibrium compositions of solid and liquid
c_liq_eq = 3.50136318050824E-01
c_sol_eq = 3.8804745109386E-01



tau = ep*(c_liq_eq - c_sol_eq)**2/(D_l/a_liq(T_eq))*(M+F)
print(tau)
```

```
2007018416.4521654
```

[6]:
```
# Now, for the approximate free energies
# Units: J/m^3

def f_s(x,T):

    a = a_bcc(T)
    b = b_bcc(T)
    c = c_bcc(T)

    return a*x**2 + b*x + c

def f_l(x,T):

    a = a_liq(T)
    b = b_liq(T)
    c = c_liq(T)

    return a*x**2 + b*x + c

print(f_s(0.3,1800))
print(f_l(0.3,1800))
```

```
-15642127716.58
-15637604483.325
```

The chemical potential($\mu = \mu_{Cr} - \mu_{Fe}$) is chosen as the independent variable (as opposed to the concentration) So, the natural choice of functional to descrube the system is the Grand Potential funcational:

$$\Omega[\phi, T] = \int_V \Psi(T, \mu, \phi) + \left(\epsilon a(\nabla\phi) + \frac{1}{\epsilon} w(\phi)\right) dV$$

Here, Ψ is the grand potential density function given by (for one phase)

$$\Psi^{(s/l)}(\mu, T) = \frac{1}{V_m}\left[F_m^{s/l}(\mu, T) - \mu c^{s/l}(\mu, T)\right]$$

F_m being the molar Helmholtz free energy. We have assumed that the chemical potential function is monotonic, and hence reversible in c, allowing us to compute $c(\mu)$. At the diffuse interface, the Ψ is computed as follows:

$$\Psi = \Psi^s h_s(\phi) + \Psi^l(1 - h_s(\phi))$$

Using an interpolation function $h_s(\phi)$ such that $h_s(\phi) + h_s(1-\phi) = 1$ which has value 1 at $\phi = 1$ (solid) and 0 at $\phi = 0$ (liquid). Further, it has the property that $\frac{\partial h_s}{\partial \phi} = 0$ at both $\phi = 0$ and 1. We take that function to be:

$$h_s(\phi) = \phi^2(3 - 2\phi)$$

[7]:
```
# define interpolation functions

# to interpolate chemical potentials
def h(phi):

    return phi**2*(3-2*phi)

# to interpolate mobilities
def g(phi):

    return phi**2*(3-2*phi)
```

The terms in the brackets in the equation for Ω correspond to the interfacial terms: ϵ is the surface energy. $w(\phi)$ is a double well potential given by

$$w(\phi) = W\sigma\phi^2(1 - \phi)^2$$

$a(\nabla\phi)$ is the anisotropy function, given by

$$a(\nabla\phi) = \sigma a_c^2(\hat{n})(\nabla\phi)^2$$

In this simulation, we take $a_c^2(\hat{n}) = 1$ that is, surface energy is isotropic.

[8]:
```
def w(phi):

    return sigma*W*phi**2*(1-phi)**2


def an(dphi):

    return sigma*dphi**2
```

We're changing the independant variable from c to μ, so it makes sense to write functions converting them

[9]:
```
def mu_s(c,T):

    return 2*a_bcc(T)*c + b_bcc(T)

def mu_l(c,T):

    return 2*a_liq(T)*c + b_liq(T)
```

[10]:
```
def c_s(mu,T):

    return (mu - b_bcc(T))/(2*a_bcc(T))

def c_l(mu,T):

    return (mu - b_liq(T))/(2*a_liq(T))
```

[11]:
```
def c(mu,T,phi):

    return h(phi)*c_s(mu,T) + (1-h(phi))*c_l(mu,T)

# Grand potential densities

def gp_s(mu,T,phi):

    C = c(mu,T,phi)

    return f_s(C,T) - mu*C/V_m

def gp_l(mu,T,phi):

    C = c(mu,T,phi)

    return f_l(C,T) - mu*C/V_m

# interpolated grand potential

def gp(mu,T,phi):

    c = c(mu,T,phi)

    return h(phi)*gp_s(mu,T,phi) + (1-h(phi))*gp_l(mu,T,phi)
```

[12]:
```
# Testing

print(mu_s(c_sol_eq,T_eq))
print(mu_l(c_liq_eq,T_eq))

print(gp_s(mu_s(c_sol_eq,T_eq),T_eq,1))
print(gp_l(mu_l(c_liq_eq,T_eq),T_eq,1))
print(c_s(mu_s(c_sol_eq,T_eq),1812))
print(c_l(942.6142621057734,1812))
```

```
1402699558.2124934
1404786645.5457425
-76788050945084.2
```

```
-76952578449374.52
0.3875406665833451
0.21835829215218072
```

0.2 Evolution equations

The evolution equations for the phase field and the chemical potetial field are, respectively

$$w\epsilon\frac{d\phi}{dt} = \epsilon\sigma\nabla^2\phi - \frac{1}{\epsilon}2W\sigma\phi(1-\phi) - (\Psi_s - \Psi_l)6\phi(1-\phi)$$

$$\left[\frac{h_s(\phi)}{2a_s} + \frac{[1-h_s(\phi)]}{2a_l}\right]\frac{d\mu}{dt} = \frac{D_l}{2a_l}[1-g_s(\phi)]\nabla^2\mu - 6(c_s - c_l)\phi(1-\phi)\frac{d\phi}{dt}$$

Where a_s and a_l are leading constants of the parabolic approximations to solid and liquid free energies, respectively. D_l is the diffusivity in liquid. D_s is assumed to be zero.

[13]:
```python
# Define some useful functions

def grad(phi):

    grad = np.zeros(len(phi))
    for i in range(len(phi)-2):

        grad[i+1] = (phi[i+2] - phi[i])/(2*dx)

    return grad

def lap(phi):

    lap = np.zeros(len(phi))
    for i in range(len(phi)-2):

        lap[i+1] = (phi[i+2] - 2*phi[i+1] + phi[i])/dx**2

    return lap

def apply_bc_phi(phi):

    #neumann bc
    phi[1] = phi[2]
    phi[0] = phi[1]
    phi[-3] = phi[-2]
    phi[-2] = phi[-1]

    return phi
```

```
def apply_bc_mu(mu):

    # neumann bc
    mu[1] = mu[2]
    mu[0] = mu[1]
    mu[-3] = mu[-2]
    mu[-2] = mu[-1]

    return mu
```

1 1D simulation

[14]:
```
# simulation parameters

#undercooling
delT = 20 #K
T = T_eq - delT
```

[15]:
```
# 1D simulation

phi_t = []
mu_t = []

# Grid pts
mx = 300

phi = np.zeros(mx)
phi[:15] = 1

cs = np.ones(mx)*c_liq_eq
cs[:15] = c_sol_eq

mus = np.zeros(mx)
mus[:15] = mu_s(cs[:15],T)
mus[15:] = mu_l(cs[15:],T)

timesteps = 51000

phi_t.append(np.copy(phi))
mu_t.append(np.copy(mus))

for t in range(timesteps):

    dphi_dt = (ep*sigma*lap(phi) - 2*W*sigma*phi*(1-phi)/ep - (gp_s(mus,T,phi) -␣
 ↪gp_l(mus,T,phi))*6*phi*(1-phi))/(tau*ep)
```

```
    phi += dphi_dt*dt
    phi_t.append(np.copy(phi))

    dmu_dt = (0.5*D_l/a_liq(T)*(1-g(phi))*lap(mus) - 6*(c_s(mus,T) -␣
↪c_l(mus,T))*phi*(1-phi)*dphi_dt)\
    /(0.5*h(phi)/a_bcc(T) + 0.5*(1-h(phi))/a_liq(T))

    mus += dmu_dt*dt
    mu_t.append(np.copy(mus))

    apply_bc_phi(phi)
    apply_bc_mu(mus)
```

[16]:
```
from ipywidgets import interact, interactive, fixed, interact_manual
import ipywidgets as widgets
```

[17]:
```
%matplotlib inline
@interact(i=(0,9999,100))
def plotfn(i):

    fig,ax = plt.subplots(figsize=(10,8))

    ax.plot(phi_t[i],label='Phi',marker='o')
    ax.set_xlabel("x" + str(dx))
    ax.legend()
```

```
interactive(children=(IntSlider(value=4900, description='i', max=9999, step=100), Output()), _do
```

[18]:
```
%matplotlib inline
@interact(i=(0,9999,100))
def plotfn(i):

    fig,ax = plt.subplots(figsize=(10,8))

    ax.plot(c(mu_t[i],T,phi_t[i]),label='C')
    ax.legend()
```

```
interactive(children=(IntSlider(value=4900, description='i', max=9999, step=100), Output()), _do
```

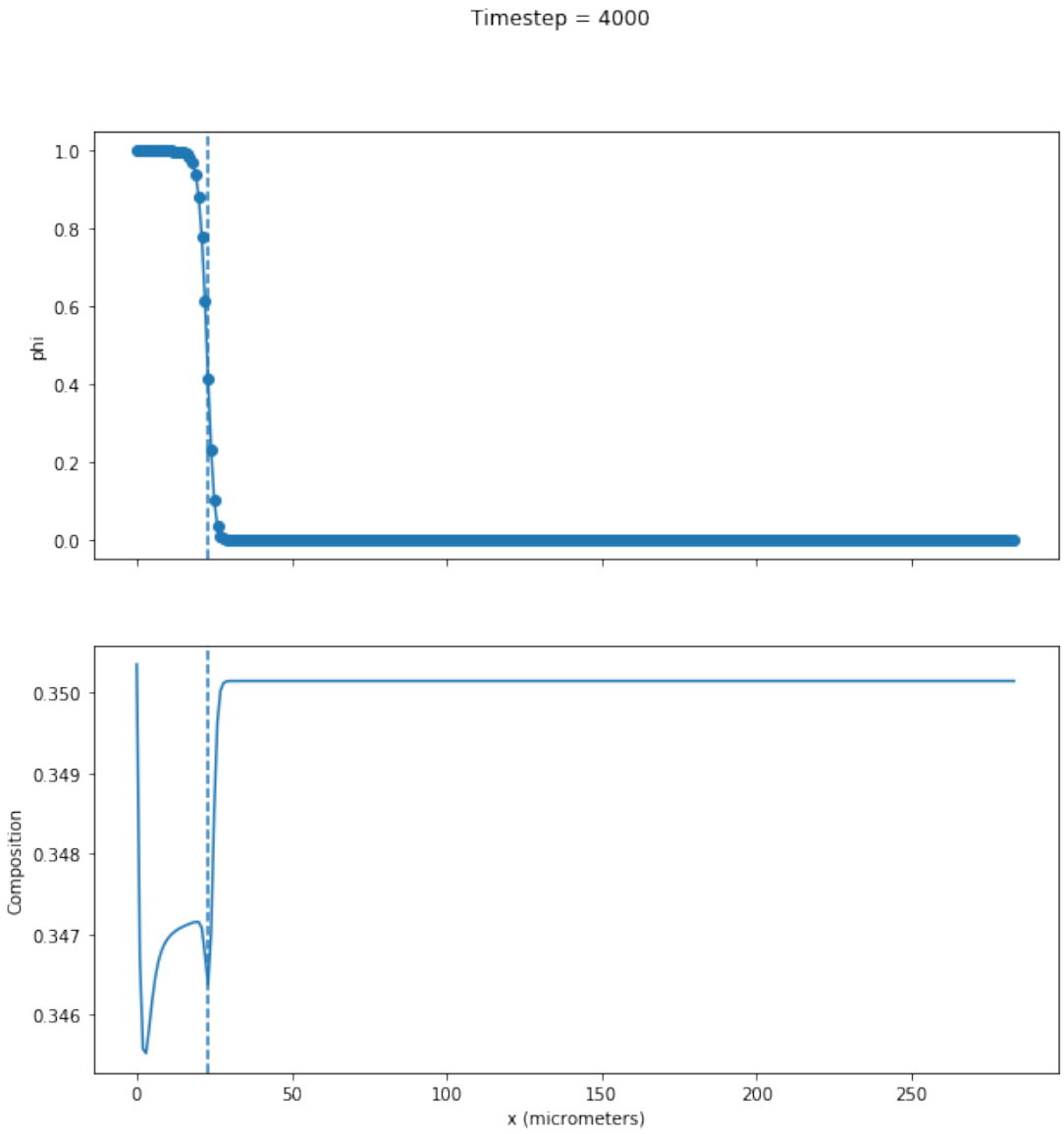
[19]:
```
print(mu_s(3.34696064e-01,1805))
print(mu_l(3.06752518e-01,1805))
```

```
951991912.6114345
947359987.0627813
```

[20]:
```
# find interface index

def find_interface(phi):

    idx = (np.abs(phi-0.5)).argmin()
    return idx

def calc_velocity():

    return (find_interface(phi_t[42000]) - find_interface(phi_t[2000]))/40*dx/
 ↪dt #mm/s

print(calc_velocity())

print(c(mu_t[-1],T,phi_t[-1])[50],c(mu_t[-1],T,phi_t[-1])[18])
```

```
6.775000000000001
0.3472477167507555 0.3455795882491013
```

[29]:
```
# Generate time profiles

def plot_both_t(ts):

    fig,axs = plt.subplots(2,sharex=True,figsize = (10,10))
    fig.suptitle('Timestep = ' + str(ts))

    intrfc = find_interface(phi_t[ts])

    #phi

    axs[0].plot(phi_t[ts][16:],label='Phi',marker='o')
    axs[0].set_ylabel('phi')
    axs[0].axvline(intrfc-16,linestyle='--')

    #mu
    axs[1].plot(c(mu_t[ts],T,phi_t[ts])[16:],label='Composition')
    axs[1].set_ylabel('Composition')
    axs[1].axvline(intrfc-16,linestyle='--')
    axs[1].set_xlabel('x (micrometers)')

    fig.savefig(str(ts))

plot_both_t(4000)
```


Timestep = 4000
phi
1.0
0.8
0.6
0.4
0.2
0.0
Composition
0.350
0.349
0.348
0.347
0.346
0
50
100
150
200
250
x (micrometers)

[ ]:

B

OpenFOAM codes

B.1 Introduction to OpenFOAM

OpenFOAM stands for **Open**-source **F**ield **O**peration **A**nd **M**anipulation. It is a general purpose open source software written in C++ used primarily for Computational Fluid Dynamics (CFD) simulations. OpenFOAM has a allows users to write custom solvers for specific problems, which extends its applications much beyond just fluid dynamics simulations.

For this work, a phase field solver was created for OpenFOAM. This solver was then used to run different simulations in seperate directories. The solver code is given in B.2.

B.1.1 Case directory structure

A typical OpenFOAM case directory is shown in B.1. The important files and folders are explained below.

The `polyMesh` folder contains the information about the simulation grid geometry.

The `system` folder contains various files to decide the simulation parameters, discretization schemes, equation solvers, etc...

The `controlDict` file contains simulation parameters like number of timesteps, write interval, etc...

The `fvSchemes` file contains the discretization schemes for the different partial derivatives in the equation like divergence, laplacian, gradient of the different variables.

The `fvSolution` file contains the solvers to be used for the different variables in the problem like ϕ and μ

The different `time directories` contain the information about the different fields at a particular time. We need to initialise with a `0` directory before the start of the simulation.

B.2 Phase Field Solver

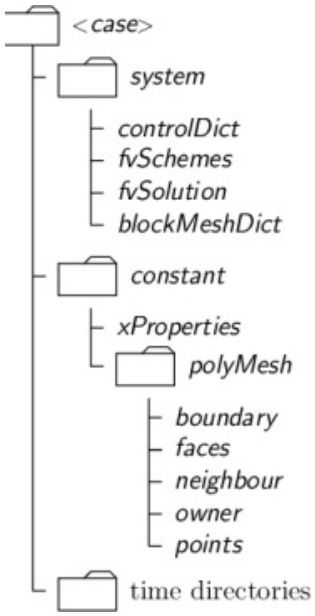


Figure B.1: A typical OpenFOAM case directory structure. Figure taken from the OpenFOAM User Guide.

Header File	Function
`createFields.H`	Initialises the different fields used in the simulation like temperature, composition, etc...
`freeEnergies.H`	Contains the parabolic free energies given in 3 and their interpolation functions.
`equations.H`	Contains the expressions for the main evolution equations 2.27 and 3.8
`anisotropy.H`	Contains the expressions for the anisotropy equations given in 2.4.

Table B.1: The role of different files in the phase field solver that was written

A solver named PFsolver was written for OpenFOAM and integrated into the OpenFOAM solvers directory by compiling with `make`. The main solver file `PFsolver.C` is given below, followed by the different header files `createFields.H`, `equations.H`, `freeEnergies.H` and `anisotropy.H`. The role of each different header file is given in the table B.1. The contents of these files are given below.

B.2.1 PFsolver.C

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Copyright (C) 2011-2018 OpenFOAM Foundation
     \\/     M anipulation  |
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.
```

```
    OpenFOAM is free software: you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM.  If not, see <http://www.gnu.org/licenses/>.

Application
    PFsolver

Description
    Solves the Allen–Cahn evolution equations in the grand potential
    formulation for chemical potential and order parameter fields
\*---------------------------------------------------------------------------*/

#include "fvCFD.H"
#include "fvOptions.H"
#include "simpleControl.H"

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

int main(int argc, char *argv[])
{
    #include "setRootCaseLists.H"

    #include "createTime.H"
    #include "createMesh.H"

    simpleControl simple(mesh);

    #include "createFields.H"

    // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

    Info<< "\nCalculating␣evolution␣of␣phi␣and␣mu\n" << endl;

    while (simple.loop(runTime))
    {
        Info<< "Time␣=␣" << runTime.timeName() << nl << endl;

        while (simple.correctNonOrthogonal())
        {
                #include "equations.H"
        }

        #include "write.H"

        Info<< "ExecutionTime␣=␣" << runTime.elapsedCpuTime() <<
```

```
        "␣s"
            << "␣␣ClockTime␣=␣" << runTime.elapsedClockTime() <<
            "␣s"
            << nl << endl;
    }

    Info<< "End\n" << endl;

    return 0;
}


// ************************************************************************* //
```

B.2.2 createFields.H

```
   Info<< "Reading␣field␣T\n" << endl;

volScalarField T
(
    IOobject
    (
        "T",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);


volScalarField phi
(
    IOobject
    (
        "phi",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);

volScalarField mu
(
    IOobject
    (
        "mu",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
```

```
    mesh
);

volScalarField composition
(
    IOobject
    (
        "composition",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);


// Reading constants

Info<< "Reading transportProperties\n" << endl;

IOdictionary transportProperties
(
    IOobject
    (
        "transportProperties",
        runTime.constant(),
        mesh,
        IOobject::MUST_READ_IF_MODIFIED,
        IOobject::NO_WRITE
    )
);

// Surface Energy

Info<< "Reading Surface Energy sigma\n" << endl;

dimensionedScalar sigma
(
    transportProperties.lookup("sigma")
);

// Interface Width

Info<< "Reading Interface width epsilon\n" << endl;

dimensionedScalar epsilon
(
    transportProperties.lookup("epsilon")
);

// Parameter W

Info<< "Reading parameter W\n" << endl;
```

```
dimensionedScalar W
(
    transportProperties.lookup("W")
);

// Relaxation parameter

Info<< "Reading␣Relaxation␣Parameter␣omega\n" << endl;

dimensionedScalar omega
(
    transportProperties.lookup("omega")
);

// Diffusivity in liquid

dimensionedScalar D_l
(
    transportProperties.lookup("D_l")
);

// Anisotropy strength

dimensionedScalar zeta
(
    transportProperties.lookup("zeta")
);

// Noise magnitude

dimensionedScalar noise_mag
(
    transportProperties.lookup("noise_mag")
);


/* Dimensional corrections for T */

dimensionedScalar dimT ( "dimT", dimensionSet(0,0,0,1,0,0,0), 1);
dimensionedScalar dimT2 ( "dimT2", dimensionSet(0,0,0,2,0,0,0), 1);
dimensionedScalar dimT3 ( "dimT3", dimensionSet(0,0,0,3,0,0,0), 1);
dimensionedScalar dimT4 ( "dimT4", dimensionSet(0,0,0,4,0,0,0), 1);
dimensionedScalar dimT5 ( "dimT5", dimensionSet(0,0,0,5,0,0,0), 1);
dimensionedScalar dimT6 ( "dimT6", dimensionSet(0,0,0,6,0,0,0), 1);
dimensionedScalar dimT7 ( "dimT7", dimensionSet(0,0,0,7,0,0,0), 1);

// Dimension of energy density
dimensionedScalar dimf ("dimf", dimensionSet(1,-1,-2,0,0,0,0),1);

// Dimension of x
dimensionedScalar dimx("dimx", dimensionSet(0,1,0,0,0,0,0),1);

// Some other fields

volScalarField ac = phi*0.0;
```

```
volVectorField dac_dq = phi*dimx*vector(0,0,0);
volVectorField da_dgradPhi = phi*dimf*vector(0,0,0);

/*******************************/
//#include "createFvOptions.H"
```

B.2.3 EQUATIONS.H

```
// Include file for constants related to free energies
// Include file for interpolations (composition, defs of h and g)

Info << "Free energy eqns" << endl;
#include "freeEnergies.H"

Info << "Anisotropy equations" << endl;
#include "anisotropy.H"

// Generate random number (for fluctuations)

Random obj(1);
const scalar randNumber(obj.scalar01());

// Evolution Equations

//phi

Info << "Phi Evo Eqn" << endl;

fvScalarMatrix phiEqn
(
        omega*epsilon*fvm::ddt(phi)
        ==
        2.0*epsilon*sigma*fvm::laplacian(ac*ac,phi)
        + epsilon*fvc::div(da_dgradPhi)
        - 2*W/epsilon*sigma*phi*(1-phi)
        - (drivingForce)*6*phi*(1-phi)
        + 6*dimf*noise_mag*phi*phi*(1-phi)*(1-phi)*randNumber
);

Info << "Min/max phi = "
<< min(phi).value() << " // "
<< max(phi).value() << endl;
Info << "Driving Force"
<< min(drivingForce).value() << " // "
<< max(drivingForce).value() << endl;
Info << "Compositions"
<< min(composition).value() << "// "
<< max(composition).value() << endl;


phiEqn.solve();

Info <<"Mu evo eqn" << endl;

// mu
```

```
fvScalarMatrix muEqn
(
        (0.5*h_phi/a2_s + 0.5*(1-h_phi)/a2_l)/dimf*fvm::ddt(mu)
        ==
        (0.5*D_l*(1-g_phi)/a2_l/dimf)*fvm::laplacian(mu)
       - 6*(c_s - c_l)*phi*(1-phi)*fvc::ddt(phi)
);

muEqn.solve();
```

B.2.4 ANISOTROPY.H

```
volVectorField q = fvc::grad(phi);

Info << "min/max mag(q): " << min(mag(q)).value()
<<" // "
<< max(mag(q)).value() <<endl;

ac  = 1.0 - zeta*(3.0 - 4.0*(pow(q.component(0),4) +
                             pow(q.component(1),4) +

                             pow(q.component(2),4))/
                             (1E-20/pow(dimx,4) + pow(mag(q),4)));

Info <<"ac done" << endl;

dac_dq = 16.0*zeta*(

(pow(q.component(0),3)/(1E-20/pow(dimx,4) + pow(mag(q),4)) -
q.component(0)*
(pow(q.component(0),4) + pow(q.component(1),4) +
pow(q.component(2),4))/(1E-20/pow(dimx,6) + pow(mag(q),6)))*vector(1,0,0)

+(pow(q.component(1),3)/(1E-20/pow(dimx,4) + pow(mag(q),4)) -
q.component(1)*
(pow(q.component(0),4) + pow(q.component(1),4) +
pow(q.component(2),4))/(1E-20/pow(dimx,6) + pow(mag(q),6)))*vector(0,1,0)

+(pow(q.component(2),3)/(1E-20/pow(dimx,4) + pow(mag(q),4)) -
q.component(2)*
(pow(q.component(0),4) + pow(q.component(1),4) +
pow(q.component(2),4))/(1E-20/pow(dimx,6) + pow(mag(q),6)))*vector(0,0,1)

);

da_dgradPhi = 2*sigma*ac*pow(mag(q),2)*dac_dq;


Info << "min/max da_dgradPhi " <<
min(da_dgradPhi).value() << " // "
<< max(da_dgradPhi).value() <<endl;
```

Bibliography

[1] Loughborough University Additive Manufacturing Research Group. URL: https://www.lboro.ac.uk/research/amrg/about/the7categoriesofadditivemanufacturing.

[2] Jan-Olof Andersson and Bo Sundman. "Thermodynamic properties of the Cr-Fe system". In: *Calphad* 11.1 (1987), pp. 83–92. ISSN: 0364-5916. DOI: https://doi.org/10.1016/0364-5916(87)90021-6. URL: http://www.sciencedirect.com/science/article/pii/0364591687900216.

[3] *ASM Handbook, Volume 1: Properties and Selection: Irons, Steels, and High-Performance Alloys*. Vol. 1. ASM International, 1990. ISBN: 978-0-87170-377-4.

[4] P. Bajaj et al. "Steels in additive manufacturing: A review of their microstructure and properties". In: *Materials Science and Engineering A* 772 (Jan. 2020). ISSN: 09215093. DOI: 10.1016/j.msea.2019.138633.

[5] Abhik Choudhury and Britta Nestler. "Grand-potential formulation for multicomponent phase transformations combined with thin-interface asymptotics of the double-obstacle potential". In: *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics* 85.2 (2012). ISSN: 15393755. DOI: 10.1103/PhysRevE.85.021602.

[6] L. Mangani F. Moukalled and M. Darwiush. *The Finite Volume Method in Computational Fluid Dynamics: An Avanced Introduction with OpenFOAM and Matlab*. Springer, Cham, 2016. DOI: https://doi.org/10.1007/978-3-319-16874-6.

[7] Samuel H. Huang et al. *Additive manufacturing and its societal impact: A literature review*. July 2013. DOI: 10.1007/s00170-012-4558-5.

[8] H Inoue et al. *Formation mechanism of vermicular and lacy ferrite in austenitic stainless steel weld metals*. Tech. rep.

[9] J. Liu, R. L. Davidchack, and H. B. Dong. "Molecular dynamics calculation of solid-liquid interfacial free energy and its anisotropy during iron solidification". In: *Computational Materials Science* 74 (2013), pp. 92–100. ISSN: 09270256. DOI: 10.1016/j.commatsci.2013.03.018.

[10] Hans Lukas, Suzana G. Fries, and Bo Sundman. *Computational Thermodynamics: The Calphad Method*. Cambridge University Press, 2007. DOI: 10.1017/CBO9780511804137.

[11] Nele Moelans, Bart Blanpain, and Patrick Wollants. "An introduction to phase-field modeling of microstructure evolution". In: *Calphad: Computer Coupling of Phase Diagrams and Thermochemistry* 32.2 (2008), pp. 268–294. ISSN: 03645916. DOI: 10.1016/j.calphad.2007.11.003.

[12] *OpenFOAM documentation*. URL: www.openfoam.com.

[13] Mathis Plapp. "Unified derivation of phase-field models for alloy solidification from a grand-potential functional". In: *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics* 84.3 (Sept. 2011). ISSN: 15393755. DOI: 10.1103/PhysRevE.84.031601.

[14] "Post-Solidification Phase Transformations". In: *Welding Metallurgy*. John Wiley and Sons, Ltd, 2003. Chap. 9, pp. 216–242. ISBN: 9780471434023. DOI: 10.1002/0471434027.ch9. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/0471434027.ch9.

[15] Brett Ryder. *The Third Industrial Revolution*. 2012. URL: https://www.economist.com/leaders/2012/04/21/the-third-industrial-revolution.

[16] CALPHAD website. URL: http://www.calphad.com/iron-chromium.html.