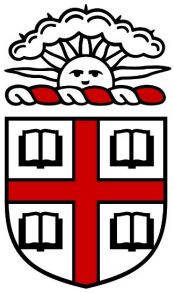
A Study of Phase Separation in Ternary Alloys

Course Project Proposal - ENGN 2912B

ARJUN SHARMA, AVPS ANCHALA, SAYAN SAMANTA



Brown University

Date of submission of proposal: November 20, 2016

Expected date of project completion: December 12, 2016

Address: Box D, 196 Hope Street
Providence, RI,
USA 02904

arjun_sharma@brown.edu
avps_anchala@brown.edu
sayan_samanta@brown.edu

Contents

1 Problem Statement

Several important engineeting alloys are multicomponent systems with multiple phases co-existing at equilibrium. Often during processing, these alloys undergo phase transformation which determine its morphology and microstructure which in turn determine its physical and mechanical properties. It is therefore essential to have a sound understanding of these phase transition and how the microstrucutre evolves with time under the driving force of free-energy reduction. With this objective in mind, the work described in this paper was carried out. In this study, an attempt shall be made to reproduce the results of a phase field simulation of a ternary alloy with varying interfacial energy parameter of the 3 components. The temperature regime in the simulation is chosen in such a fashion that the alloy at a given composition undergoes spinodal decomposition at that temperature. The objective of this project is to observe the temporal evolution of the microstructure of the alloy and the effect of the interfacial energy parameter on the morphology of individual phases. The microstructure of any alloy is an indication of the mechanical characteristics of the alloy, and thus a well-defined idea of the individual phase volumes and morphology would make it convenient for the production industry to tailor their heat treatment schedules to the desired microstruture. Such computer experiments are highly advantageous as it is a quick and cost-effective alternative to actual experiments which are very expensive and time consuming such a systematic study beyond a few discreet time points is non-feasible.

Hilliard in 1970[1] had studied both theoretically and experimentally the phase separation kinetics for binary alloys which undergo spinodal decomposition. While Meijering[2], Kikuchi[3] studied simple models of ternary phase diagrams, de Fontaine[4] analysed the spinodal decomposition kinetics during the early stages in ternary systems. The effect of alloy composition on the non-linear dynamics of phase evolution during spinodal decomposition was studied by Chen[5].

2 Technical Objectives

In this work, a ternary alloy having 3 componenets A, B, C have been chosen. The concentration of each component have been denoted as $c_i(\mathbf{r}, t)$ for i = A, B, C represent mole fraction of the ith component as a function of position $\mathbf{r}$ and time t. Since composition is a conserved quantity, we can claim that

$$c_A + c_B + c_C = 1 \tag{1}$$

We also assume that the bulk free energy of the system follows the regular solid solution model and therefore the free energy per molecule $f(c_A, c_B, c_C)$ can be expressed as

$$\frac{1}{k_B T} f(c_A, c_B, c_C) = \sum_{i \neq j} \chi_{ij} c_i c_j + \sum_i c_i \ln c_i \tag{2}$$

[1] Hilliard J E 1970 Phase transformations (ed.) H I Aaronson (Metals Park, Ohio: ASM)
[2] Meijering J L 1950 Philips Res. Rep. 5 333
[3] Kikuchi R 1997 Acta Metall. 25 195
[4] de Fontaine D 1972 J. Phys. Chem. Solids 33 297
[5] Chen L Q 1994 Acta Metall. Mater. 42 3503

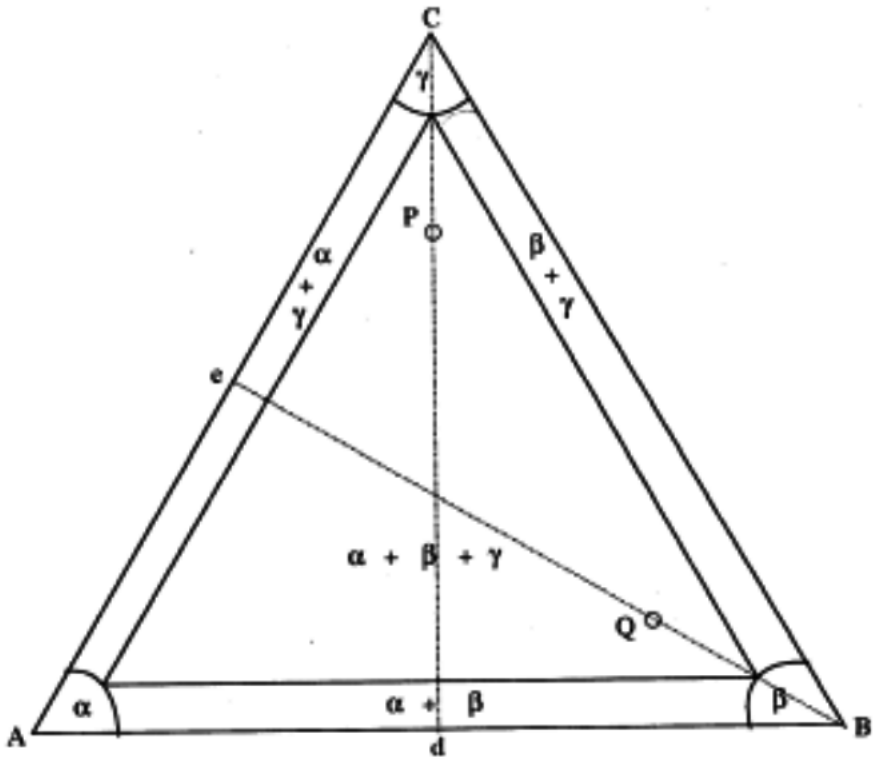


Figure 1: Isothermal section of the phase diagram at χ = 3.5

where χ_{ij} ($= \frac{\omega_{ij}}{k_B T},\ i,j = A,B,C; i \neq j\ k_B = Boltzmann\ Constant, T = Absolute\ Temperature$) is the effective interaction energy parameter between components i and j. It is to be noted that χ_{ij} is inversely proportional to temperature and thus although Cahn-Hilliard equations governs isothermal transformation, it is effectivly dependent on temperature through the interaction energy parameter. In this study we shall consider a symmetric alloy, therefore $\chi_{AB} = \chi_{BC} = \chi_{AC} = \chi$.

In this particular work, we consider $\chi = 3.5$, in which we observe a region of ternary miscibility gap where 3 phases α - A rich phase, β - B rich phase and γ - C rich phase coexist simultaneously. (refer figure 1). At high-temperature the alloy with average composition $A_{25}B_{25}C_{50}$ is homogenous and disordered. When the disordered alloy is quenced to low temperature (at which it phase separates), the alloy undergoes phase separation into 3 phase structure. The equilibrium composition of the α, β, γ phases are $(c_A, c_B, c_C) = (0.91, 0.045, 0.045), (0.045, 0.91, 0.045)$ and $(0.045, 0.045, 0.91)$ respectively.

Since total concentration is conserved, as given in equation 1, we need to solve only for 2 components (c_A and c_B) because then c_C will be given as $c_C = 1 - c_A - c_B$. Using the equation, $\dot{c_I} = M_I \nabla^2 \mu_I$, in conjunction with Gibbs-Duhem Equation we can derive the time-evolution

equations of c_A and c_B as

$$\begin{aligned}\frac{\partial c_A}{\partial t} &= M_{AA}\{\nabla^2\left(\frac{\partial f}{\partial c_A}\right) - 2K_{AA}\nabla^4 c_A - 2K_{AB}\nabla^4 c_B\} \\ &+ M_{AB}\{\nabla^2\left(\frac{\partial f}{\partial c_B}\right) - 2K_{AB}\nabla^4 c_A - 2K_{BB}\nabla^4 c_B\}\end{aligned} \tag{3}$$

$$\begin{aligned}\frac{\partial c_B}{\partial t} &= M_{BA}\{\nabla^2\left(\frac{\partial f}{\partial c_A}\right) - 2K_{AA}\nabla^4 c_A - 2K_{AB}\nabla^4 c_B\} \\ &+ M_{BB}\{\nabla^2\left(\frac{\partial f}{\partial c_B}\right) - 2K_{AB}\nabla^4 c_A - 2K_{BB}\nabla^4 c_B\}\end{aligned} \tag{4}$$

where M_{ij} - Effective Mobilities, $K_{AA} = K_A + K_C$; $K_{BB} = K_B + K_C$; $K_{AB} = K_{BA} = K_C$. For non-dimensionalizing the equations, the characteristic length used is, $l^* = \frac{(K_A^*)^{1/2}}{2k_BT}$ and characteristic time used is, $t^* = \frac{k_BT}{M_{AA}^* l^{*2}}$. Also from equation **??** we can derive that,

$$\frac{\partial f}{\partial c_A} = \chi(c_A - c_C) + \ln\left(\frac{c_A}{c_C}\right) \tag{5}$$

$$\frac{\partial f}{\partial c_B} = \chi(c_B - c_C) + \ln\left(\frac{c_B}{c_C}\right) \tag{6}$$

Superscript * indicates dimensional quantities. In order to simulate bulk properties through a fixed small simulation box size, we employ periodic boundary conditions, hence to solve the equation, we have used semi-implicit Fourier spectral method which was proposed by Chen & Shen[6] to solve equation 3 and 4. By using Fourier spectral method , we can convert the partial differential equation into a ordinary differential equation due to the identitites,

$$\begin{aligned}\mathcal{F}\{f'(x)\} &= \imath k\mathcal{F}\{f(x)\} \\ \mathcal{F}\{f''(x)\} &= -k^2\mathcal{F}\{f(x)\} \\ \mathcal{F}\{f''''(x)\} &= k^4\mathcal{F}\{f(x)\}\end{aligned}$$

The fourier transformed equation corresponding to equation 3 and 4 are given by,

$$\begin{aligned}\frac{\partial \tilde{c_A}(\mathbf{k},t)}{\partial t} &= M_{AA}\{-k^2\left(\frac{\partial f}{\partial c_A}\right)_k - 2K_{AA}k^4\tilde{c}_A^{t+1} - 2K_{AB}k^4\tilde{c}_B^{t+1}\} \\ &+ M_{AB}\{-k^2\left(\frac{\partial f}{\partial c_B}\right)_k - 2K_{AB}k^4\tilde{c}_A^{t+1} - 2K_{BB}k^4\tilde{c}_B^{t+1}\}\end{aligned} \tag{7}$$

$$\begin{aligned}\frac{\partial \tilde{c_B}(\mathbf{k},t)}{\partial t} &= M_{BA}\{-k^2\left(\frac{\partial f}{\partial c_A}\right)_k - 2K_{AA}k^4\tilde{c_A}^{t+1} - 2K_{AB}k^4\tilde{c_B}^{t+1}\} \\ &+ M_{BB}\{-k^2\left(\frac{\partial f}{\partial c_B}\right)_k - 2K_{AB}k^4\tilde{c_A}^{t+1} - 2K_{BB}k^4\tilde{c_B}^{t+1}\}\end{aligned} \tag{8}$$

[6]Chen L Q and Shen J 1998 Comput. Phys. Commun. 108 147

where $\mathbf{k}$ = (k_x, k_y) - reciprocal lattice vector, $k = |\mathbf{k}|$ $\tilde{c}_A(\mathbf{k}, t)$ and $\tilde{c}_B(\mathbf{k}, t)$ - respective composition in fourier space, $\left(\frac{\partial f}{\partial c_A}\right)_k$, $\left(\frac{\partial f}{\partial c_B}\right)_k$ - respective fourier transforms of partial free energy fields. As proposed by the semi-implicit method, the linear fourth order terms in equation 7 and 8 are treated implicitly and the non-linear terms are treated explicitly. The finite difference equation to be solved simultaneously to obtain $\tilde{c}_A(\mathbf{k}, t)$ and $\tilde{c}_B(\mathbf{k}, t)$ are given by,

$$\begin{aligned}\Big[1 + k^4\Delta t\{2M_{AA}K_{AA} + 2M_{AB}K_{AB}\}\Big]\tilde{c}_A^{t+1} + \Big[\Delta t k^4\{2M_{AA}K_{AB} + 2M_{AB}K_{BB}\}\Big]\tilde{c}_B^{t+1} \\ = \tilde{c}_A^t - k^2\Delta t\{M_{AA}\tilde{g}_A^t + M_{AB}\tilde{g}_B^t\}\end{aligned} \tag{9}$$

$$\begin{aligned}\Big[k^4\Delta t\{2M_{AB}K_{AA} + 2M_{BB}K_{AB}\}\Big]\tilde{c}_A^{t+1} + \Big[1 + \Delta t k^4\{2M_{AB}K_{AB} + 2M_{BB}K_{BB}\}\Big]\tilde{c}_B^{t+1} \\ = \tilde{c}_B^t - k^2\Delta t\{M_{AB}\tilde{g}_A^t + M_{BB}\tilde{g}_B^t\}\end{aligned} \tag{10}$$

The system of equation is solved using cramer's rule.[10]

3 C++ Libraries

In §2, the theory behind the simulation, the evolution equations and the free energy function has been described in detail. To perform the simulation in a computer, certain scientific libraries designed to perform numerical computation has to employed. Although the equations correspond to a continuous bulk system, it is impossible to conduct a simulation on the real-life scale of the system. Hence to mimic bulk condition, we employ a cartesian mesh of $512 \times 512 \times 512$ with periodic boundary conditions. The distance between two grid points in the mesh is 1.0.

In this particular study, we have chosen the following libraries:

3.1 Pseudo Random Number Generator

The C++ Standard Template Library contains the `<random>`, which generates pseudo random numbers by a combination of generators and distribution.

- Generators: Objects that generate uniformly distributed numbers.
- Distributions: Objects that transform sequences of numbers generated by a generator into sqeuences of numbers that follow a specific random variable distribution, such as uniform, normal or Binomial.

Since each random generator algorithm require a starting random value (known as seed), we use the system clock value.

```
unsigned tSeed = std::chrono::system_clock::now().time_since_epoch().count();
```

Following which we use the Mersene Twister 19937 generator (64 bit), which is one the most widely used engines primarily due to it's good precision and wide bit length.

[10]Cramer, Gabriel (1750). "Introduction a lAnalyse des lignes Courbes algebriques" (in French). Geneva: Europeana. pp. 656-659.

```
std::mt19937_64mersenne(tSeed);
```

Now to set the initial values of the composition of our system, we gnerate pseudo random numbers between [-0.5:+0.5] using the uniform real distribution described by the following probability density function

$$p(x|a,b) = \frac{1}{b-a}, a \leq x < b$$

using the function

```
std::uniform_real_distribution<> distr(-0.5,+0.5);
```

3.2 Complex Numbers

The complex datatypes are defined in the header `gsl_complex.h` while the functions and operators are defined in `gsl_complex_math.h`. The required functions and macros required for the the project are:

3.3 FFTW

FFTW (stands for Fastest fourier Transform in the West) is a comprehensive collection of routines which perform discrete fourier transform (DFT). Instead of using a fixed algorithm to calculate DFT, it adapts it to the hardware on which it is being run to maximize performance. Therefore the computation involves two phases:

1. The first step involves a 'planner' routine, which *learns*, the fastest way to compute the transform at the host machine.
2. The 'planner' introduces a data structure 'plan' which contains the information. Consequently, each time one data set has to be fourier transform, the plan is called and the transform is executed. In typical parallel computing cases, this one time overhead, is a relatively inexpensive when compared to the entire runtime.

3.3.1 FFTW Complex

Since FFTW has no provision for parallel real to real transform in its MPI infrastructure, we use the `fftw_complex` datatype which uses the `double` precision for all floating point numbers, and then later set the imaginary part to 0. We can also use the C++ standard `<complex>` header files since the native complex type is binary-compatible with the `fftw_complex`. This is true to the extent that a datatype of `complex<double> *x` can be directly passed to FFTW via `reinterpret_cast<fftw_complex>(x)`.

FFTW also provides methods for dynamically allocating memory to the datatypes using the `fftw_malloc` and then be deallocated using `fftw_free`. However, in C++ we would have to typecast the output to whichever datatype which we are allocating. The function signatures are:

```
fftw_complex *fftw_alloc_complex(size_t n);
void fftw_free(void *p);
```

3.3.2 Complex DFT's

Once we have the required datatypes, a plan in 0 or more dimensions has to be created, which is returned as an object `fftw_plan`. The function signature is as follows (for our case, it is a 3D plan)

```
  fftw_plan fftw_plan_dft_3d(int n0, int n1, int n2, fftw_complex *in,
fftw_complex *out, int sign, unsigned flags);
```

The requried arguments to the function are:

- n0, n1, n3 are the size of the transform dimension. They can be any postive integer.
- in and out point to the input and output array during the transform
- rank is the sign of the exponenent in the formula that defines the fourier transform (its direction), it can be `FFTW_FORWARD` or `FFTW_BACKWARD`
- flags denotes the planner routine which FFTW follows, in our case we shall employ the quickest plan (probably sub-optimal) namely `FFTW_ESTIMATE`. With this plan the input/output arrays are not overwritten during the plan.

Once a plan has been created, it can be executed multiple number of times on the specified input and output arrays, computing the actual transforms `fftw_execute(plan)`

```
  void fftw_execute(const fftw_plan plan)
```

3.4 MPI DFT's using FFTW

In addition to the concepts discussed above, FFTW's has a MPI interface to support multidimensional DFTs of real data but not of real to real. However we can manually set the imaginary part to zero in our calculation as a leeway through the problem. However we have to keep in mind the data distribution between the processes, a simple way to do that is to follow the guidelines in the FFTW manual which gives a detailed description has to which part of the array is in which processor and how much space to allocate. The general way of incorporating MPI in FFTW is:

```
  const ptrdiff_t n0, n1, n2;
  ptrdiff_t alloc_local, local_n0, local _0_start;
  fftw_plan plan;

  MPI_Init(&argc, &argv);
  fftw_mpi_init();
  alloc_local = fftw_mpi_local_size_3d(n0, n1, n2, MPI_COMM_WORLD,
&local_n0, &local_0_start);
  out = fftw_alloc_complex(alloc_local);

  //create plan
  plan = fftw_mpi_plan_dft_3d(n0,n1,n2, in, out, MPI_COMM_WORLD,
 FFTW_FORWARD, FFTW_ESTIMATE);
```

```
void fftw_mpi_execute_dft(fftw_plan p, fftw_complex *in, fftw_complex *out);

//rest of the code
```

Note the subtle changes required to incorporate MPI, first we have to call one of the MPI routine to determine the required allocation size and the portion of the array locally stored on a given process. Secondly, the MPI version of the 'planner' routine has an additional argument, which is global processor reference and has corresponding `execute` routine for the execution of the transform.

4 Data visualisation

At each timestep of the simulation, the code generates concentrations values for each component which ranges between [0,1]. Any particular phase has a combination of A, B and C with each phase α, β and γ being predominant in A, B and C respectively. For the purpose of visualisation, we combine these different concentration into one single function by assigning 'weights' to each component. For this project, we assigned $10 \rightarrow$ A, $20 \rightarrow$ B and $30 \rightarrow$ C. Therefore the overrall concentration parameter is given by:

$$F(c_a, c_b, c_c) = 10 \times c_a + 20 \times c_b + 30 \times c_c \tag{11}$$

with the range of values being [10,30]. This values for selected timesteps can be written to a file in a matrix format (each column of data is separated with newline, while each datapoint is separated by a space).

This data file can then be plotted with GNUplot with a Pm3D splot

```
set style data pm3d
```

style with the palette range set between black and white mapping the values 10 to 20.

```
set cbrange[10:30]
set palette defined (10 'black', 30 'gray')
```

Thereby translating the data to a colored surface map with white, gray and black denoting α, β and γ respectively. This can be then plotted with the splot command.

```
splot "filename.dat" u 1:2:3:4 with pm3d
```

5 Simulation Parameters

The simulations would be run on the Center for Computation and visualization (CCV), Brown University. The simulation parameters are:

1. Simulation box size: 512×512
2. Spatial step $\Delta x = \Delta y = \Delta z = 1.0$
3. Timestep $\Delta t = 0.05$

Throughout the simulation, we will have to use scaled mobilities of $M_{AA} = 1.0$, $M_{BB} = 1.0$ and $M_{AB} = M_{BA} = -0.5$

The compositions chosen for this study is denoted by points P and Q in figure 1. The simulations will be run for 3 cases,

I $K_A = K_B = K_C$ with initial overall composition at point P $(c_A, c_B, c_C) = (0.25, 0.25, 0.5)$

II (a) $K_A = K_C \neq K_B$ with initial overall composition at point P $(c_A, c_B, c_C) = (0.25, 0.25, 0.5)$

(b) $K_A = K_C \neq K_B$ with initial overall composition at point Q $(c_A, c_B, c_C) = (0.25, 0.5, 0.25)$

Each simulations is initiated by the given average composition homogenously distributed throughout the simulation box with a uniform random noise between $\pm$ 0.005 (for each component A, B and C). The values for the gradient energy coefficient K_A, K_B, K_C which correspond to different interfacial energy values $\sigma_{\alpha\beta}$, $\sigma_{\alpha\gamma}$ and $\sigma_{\beta\gamma}$ respectively are given in table 1
increasing the value of K would increase the interface width(more diffuse interace) in order to

Case	K_A	K_B	K_C	$\sigma_{\alpha\beta}$	$\sigma_{\alpha\gamma}$	$\sigma_{\beta\gamma}$
I	4.0	4.0	4.0	1.195	1.195	1.195
II	4.0	32.0	4.0	2.628	1.236	2.628

Table 1: Gradient energy parameters and sealed interfacial energies

decrease the contribution to free energy. However as it would entail adding more number of atoms of non-uniform composition at the interface, there is a balance between the two. However in this experiment since every other parameter is same, any interface in association with 'B-rich' (β) phase has higher energy[11].

6 Pseudo Code

1: Input $c_A, c_B, c_C, K_A, K_B, K_C$
2: Choose $\Delta x, \Delta y, \Delta z, \Delta t$
3: Choose grid size n_x, n_y, n_z
4: Declare variables g_A, g_B, c_A, c_B, c_C
5: Set initial homogenous composition at each point, (according to case I and case II), with a noise of $\pm$ 0.005
6: $\Delta k_x \longleftarrow {}^{2\pi}/_{(n_X \Delta x)}$
7: $\Delta k_y \longleftarrow {}^{2\pi}/_{(n_Y \Delta y)}$
8: $\Delta k_z \longleftarrow {}^{2\pi}/_{(n_Z \Delta z)}$
9: **for** index: 1 **to** timesteps **do**
10: $g_A \leftarrow f(c_A, c_B)$
11: $g_B \leftarrow h(c_A, c_B)$
12: // convert to fourier space
13: $\tilde{g}_A \leftarrow$ ForwardDFT(g_A)

[11] D. Raabe, F. Roters, F. Barlat, L.Q.Chen, "Continuum Scale Simulation of Engineering Materials"

```
    g̃_B ← ForwardDFT(g_B)
    c̃_A ⟵ ForwardDFT(c_A)
    c̃_B ⟵ ForwardDFT(c_B)
    // Periodic Boundary Conditions
    for i1: 1 to n_x do
        if i < n_x then
            k_x = i1Δk_x
        else
            k_x = (i1 − n_x)Δk_x
        end if
    end for
    for i2: 1 to n_y do
        if i2 < n_y then
            k_y = i2Δk_y
        else
            k_y = (i2 − n_y)Δk_y
        end if
    end for
    for i3: 1 to n_z do
        if i3 < n_z then
            k_z = i1Δk_z
        else
            k_z = (i1 − n_z)Δk_z
        end if
    end for
    k² ← k_x² + k_y² + k_z²
    SolveEq(k², g_A, g_B, c_A, c_B, Δt) // Solve using Cramer's Rule
    g_A ← BackwardDFT(g̃_A)
    g_B ← BackwardDFT(g̃_B)
    c_A ← BackwardDFT(g̃_A)
    c_B ← BackwardDFT(g̃_B)
    c_A ← c_A/(n_X × n_y × n_z)
    c_B ← c_B/(n_X × n_y × n_z)
    c_C ← 1 − c_A − c_B
end for
```

7 Project Schedule

The major milestones in this project and the associated deadlines (tentative) are set as follows:

- Week 1 (27th November): Set up the major routines independently among the groups for a serial code in 2D. The major subdomains are:
  - Read the input from the user in the specified format and raise errors if the input file is not in the proper format.

  - Generate pseudo random numbers based on the perturbations across the mean variable for each component.
  - Write the code for the 2D serial.
  - Reading the generated data to file.
  - Visualisation of the data in a 3D matrix plot.

- Week 2 (4th December): Observe the evolution equations in 2D and test with the results as given in the paper, once successfull, extend the code in 3D and incorporate parallelisation. This task has to be done in tandem with all the members through the common git repository.
- Week 3 (11th December): Improve any glitches in the code, improve the performance and the visualization of the data.