AIX-MARSEILLE UNIVERSITY
MASTER'S DEGREE OF PHYSICS

Modeling project

Surface Segregation in Ag-Cu nanoalloys using Monte Carlo techniques

Yasmina Moussaoui

2022-2023

Contents

Abstract

A **metal alloy** is a mixture of chemical elements, with each one of them being a metal. In an alloy, the concentration of the extern layers can differ from those of the intern ones. This is the so-called **"segregation phenomena"**, that has been the subject of various research for its possible scientific and industrial use.

To model this phenomena we use a tight-biding Ising model (**TBIM**), the hamiltonian contains a linear term, quasi-concentration independant, which proves to be very close to the difference in surface tensions between pure constituents and a quadratic one. This model relies on an description of the energy as a sum of effective pair interactions that are larger at the surface than in the bulk.

Key words : Monte carlo, Ag-Cu, Tight-binding Ising model, metal alloys, segregation, nanoparticle.

---

1 Introduction

We will compute the tight-binding Ising model and the effective pair interactions for binary systems using Monte carlo techniques. Our system is a nanoparticle with the shape of a truncated octahedron of 405 atoms. Using the Monte Carlo simulation with the Metropolis algorithm, we will exchange Cu or Ag type of atoms with a certain probability, following a Boltzman law, choosing randomly any site to exchange.

Using c++ for the implementation of the model, we will then vizualize the concentration of impurities for various mesh areas and for different steps of chemical potential using python.

It could be interesting to represent the mesh at certain point of chemical potential, using xmakemol, in order to witness the segregation phenomenon. In the case of the Ag-Cu system, the preferential impurities(Ag) placement in the surface layers.

2 The tight-binding ising model

The tight-binding Ising model (ref. G. Tréglia, B. Legrand, F. Ducastelle, Europhysics Letters 7, 575 (1988)) relies on the description of the part of the energy related to the chemical configuration as a sum of pair interactions. This part of the total energy of binary nanoalloy A_c, A_{1-c} systems , is obtained from the Hamiltonian :

$$\boxed{H = \frac{1}{2}\sum_{n,m\neq n} P_n^i P_m^j \epsilon_{nm}^{ij}} \qquad \text{i,j=A,B,C...} \tag{1}$$

where ϵ_{nm}^{ij} is the interaction energy between the atoms of type "i" and "j" placed in the sites "n" and "m".
P_n^i is the occupation factor of site "n" for an atom of type "i".

For a binary system AB, we can use : $P_n^A = 1 - P_n^B = P_n$. We can, then, simplify the expression (1) :

$$H = \frac{1}{2}\left(\sum_{n,m\neq n} P_n^A P_m^A \epsilon_{nm}^{AA} + \sum_{n,m\neq n} P_n^B P_m^B \epsilon_{nm}^{BB} + \sum_{n,m\neq n} P_n^A P_m^B \epsilon_{nm}^{AB} + \sum_{n,m\neq n} P_n^B P_m^A \epsilon_{nm}^{AB}\right) \tag{2}$$

Let $P_n^B = 1 - P_n^A$ and $P_n^A = P_n$, we then have :

$$H = \frac{1}{2}\left(\sum_{n,m\neq n} P_n P_m \epsilon_{nm}^{AA} + (1-P_n)(1-P_m)\epsilon_{nm}^{BB} + P_n(1-P_m)\epsilon_{nm}^{AB} + (1-P_n)P_m\epsilon_{nm}^{AB}\right)$$

$$H = \frac{1}{2}\left(\sum_{n,m\neq n} P_n P_m \epsilon_{nm}^{AA} + \epsilon_{nm}^{BB} - (P_n + P_m)\epsilon_{nm}^{BB} + P_n P_m \epsilon_{nm}^{BB} + (P_n + P_m)\epsilon_{nm}^{AB} - 2P_n P_m \epsilon_{nm}^{AB}\right)$$

$$H = \frac{1}{2}\left(\sum_{n,m\neq n} P_n P_m (\epsilon^{AA} + \epsilon^{BB} - 2\epsilon^{AB}) + (P_n + P_m)(\epsilon^{AB} - \epsilon^{BB}) + \epsilon^{BB}\right)$$

Let $V = \frac{1}{2}(\epsilon^{AA} + \epsilon^{BB} - 2\epsilon^{AB})$, the effective pair interaction. Let $\mathcal{T}_{nm} = \frac{1}{2}(\epsilon^{AA} - \epsilon^{BB})$, and $H_0 = \sum_{n,m\neq n} \epsilon^{BB}$.

$$\mathcal{T} - V = \frac{1}{2}\left(\epsilon^{AA} - \epsilon^{BB} - \epsilon^{AA} - \epsilon^{BB} + 2\epsilon^{AB}\right) = \epsilon^{AB} - \epsilon^{BB}$$

We, then, describe the ordering or segregation processes at the surface in terms of an effective Ising Hamiltonian :

$$\boxed{H = H_0 + \sum_{n,m\neq n} P_n(\mathcal{T}_{nm} - V_{nm}) + \sum_{n,m\neq n} P_n P_m V_{nm}} \tag{3}$$

We will call this approach the **tight-binding Ising model (TBIM)**. $\mathcal{T}$ and V both depend on the bulk concentration.

- For n and m, first neighbours, we have $\mathcal{T}_{nm} = Z^1$ and $V_{nm} = V^1$.
- $\sum_{n,m} P_n(Z^1 - V^1)$ is a linear term in P_n.
- $\sum_{n,m} P_n P_m V^1$ is a a quadratic term, where it is equal to V^1 when we have two first neighbours of the same type, and 0 if not.
- H_0 is a constant, therefore we do not take it into account in the Monte-Carlo simulation, since we are only considering differences of energy, that is determind with the **"diffenergy"** method within the namespace nano.

3 Implementation of the code

The objective is to build a program that modelises the tight-binding Ising model using Monte-Carlo techniques.

3.1 Important files and their use

- **init.dat :** contains the position in the mesh of our 405 studied atoms. In the last column is indicated the number of first neighbours, therefore knowing on which area the atoms are(bulk, outer layers,edges..)
- **properties/system.dat** : Potential for different type of systems : Ag-Cu, Ni-Ag, Cu-Ni
- **Properties/elements.dat** : Contains the cohesion energies and mesh parameters for differents types of atoms.
- **in.dat :** contains all the parameters of our simulation :
  - **500** npas (number of steps for a Monte carlo loop)
  - **20** npw (npas > 20*npw) (we wait npeq=20*npw steps for a statiscal equilibrium, before calculating the concentrations of impurities.
  - **500 :** temp (temperature)

- **0.4 :** dmu (we start with that chemical potential)
- **20 :** idmumax (the number of steps in cheical potential)
- **0.01 :** ddmu (the size of the step in chemical potential)
- **6 :** imax (x dimension of the box, not used here)
- **6 :** jmax (y dimension of the box, not used here)
- **6 :** kmax (z dimension of the box, not used here)
- **12682319 :** irand (our random seed)
- **Cu :** (base element)
- **Ag :** (impurity)

- **nano.cpp and the header nano.h :** contains the namespace nano, which contains the class **"maille"**, and the various methods used to implement this simulation.
- **tbim_main.cpp :** We call different methods and fucntions defined in nano.cpp
- **makefile** : Used to compile and execute the program, we just write "make" in the terminal.
- **concen_figs.py** : python script used to plot the concentration in impurities for various areas within the mesh.

3.2 Functions and Methods

3.2.1 Initialization

Firstly, we will initialize our system and its parameters by extracting them from our differents files. For that we defined those parameters in the head of the namespace, and we used the function **initialize(filename)**.

The **"Set_param(string line)"** method in the class Nanoparticule sets the coordinates and type of a nanoparticule within a mesh. This corresponds to a line in the **"init.dat"** file.

To set the mesh, we use the **maille** method, in the Maille class. Each lines corresponds to a nanoparticule in the mesh.

3.2.2 Monte-carlo

Our initialization is done. The **"DoMonteCarlo(maille)"** function does a Monte-Carlo technique for idmumax steps of chemical potential starting with the potential "dmu" defined in the **"init.dat"** file. The **"MonteCarlo(maille)"** function, is considering a particular step of chemical potential and does a complete Monte-Carlo of npas, it is therefore called for each step of dμ.

The function **"mc_exchange"** implements, at a random picked site, an exchange of Cu or Ag, with a certain probability, following a Boltzman law.

Therefore, after computing the energy difference between the new obtained mesh and the former one, we can either reject or accept an exchange following this Boltzman law :

- if the energy difference de$<$0, exchange accepted.
- f not, we compute the Boltzman weight and compare it with a random number p $\in$ [0..1]

If the boltzman weight is greater than p, we keep the exchange. If not, we come back to the previous mesh. This possible exchange is done for a random site witin the mesh and for a number of time equal to the number of atom in the mesh.

3.2.3 Concentrations

When we call **"DoMonteCarlo(maille)"**, we use the **"create_concen_files(dir)"** function to create in the directory "dir", 5 "concentrations.dat" files. Then,**"write_all"** will, at each step of chemical potential, write the concentration of impurities depending on the area of the mesh. Concentrations are determined trough the use of **"concen_neighbor"**, that we call at each step of the Monte-carlo loop and then the obtained results are divided by (npas-neq), to have an average concentration for a step of chemical potential. Let us remind that all our calulations are done after waiting neq steps for a statistical equilibrium.

3.2.4 Writting

At each step of chemical potential we will write the concentrations in our concentration files (defined previously). We will also write our updated mesh(.xyz files), with the contained nanoparticules, their positions and their types. For that we will call the function **"write_parameters"** inside the loop for chemical potential, within the **"DoMonteCarlo(maille)"** function .

4 Problems encountered and their solutions

4.1 Runtime

After everything was implemented, we started experimenting on ways to improve the efficency and the run time of the programm. Indeed, for a 100 steps in a Monte-Carlo loop, the code took around two hours for a step of chemical potential ? It was due to computing at each step the energy of our mesh of 405 nanoparticules using the **"energy(maille)"** function.

```
float energy(Maille& maille){
  float energy_total=0;
  for(int i=0;i<maille.getNumberOfAtoms();i++){
    if(maille.getParticleKind(i)==impurity){
      for(int j=0;j<maille.getNVois()[i];j++){
        int voisin_k=maille.getIVois()[i][j];
        if(maille.getParticleKind(voisin_k)==impurity){
          energy_atoms[i]=energy_atoms[i]+V;
        }
      }
      energy_atoms[i]=energy_atoms[i]+maille.getNVois()[i]*(Tau-V);
    }
    energy_total=energy_total+energy_atoms[i];
  }
  clear_vect(energy_atoms);
  return energy_total;
}
```

Figure 1: The **"energy"** function that caused an escalation of the running time.

Therefore, we compute the energy difference within the mesh, not the total one. Indeed, we will only focuse on nanoparticules that have been exchanged and their surrounding. Which saves a lot of calculation time, it now takes only a few minutes for 500 steps in a Monte-Carlo loop, instead of around 2 hours for 10 steps.

```
float diffenergy(Maille& maille,int site,vector<float>& ener_0){
  float energyMod=0;
  copy(energy_atoms.begin(),energy_atoms.end(),ener_0.begin());
  energy_atoms[site]=0;
  if(maille.getParticleKind(site)==impurity){
    for(int j=0;j<maille.getNVois()[site];j++){
      int voisin_k=maille.getIVois()[site][j];
      if(maille.getParticleKind(voisin_k)==impurity){
        energy_atoms[site]=energy_atoms[site]+V;
      }
    }
    energy_atoms[site]=energy_atoms[site]+maille.getNVois()[site]*(Tau-V);
  }
  energyMod+=energy_atoms[site]-ener_0[site];

  for(int i=0;i<maille.getNVois()[site];i++){
    int voisin_k=maille.getIVois()[site][i];
    energy_atoms[voisin_k]=0;
    if(maille.getParticleKind(voisin_k)==impurity){
      for(int k=0;k<maille.getNVois()[voisin_k];k++){
        int voisin_k_k=maille.getIVois()[voisin_k][k];
        if(maille.getParticleKind(voisin_k_k)==impurity){
          energy_atoms[voisin_k]+=V;
        }
      }
      energy_atoms[voisin_k]+=maille.getNVois()[voisin_k]*(Tau-V);
    }
    energyMod+=energy_atoms[voisin_k]-ener_0[voisin_k];
  }
  return energyMod;
}
```

Figure 2: The **"diffenergy"** function that computes the energy difference and therefore saves running time.

Speaking about running time for the concentration caculations. At each step of chemical potential we determined, again, the number of different types of atoms depending on their occupation areas within the mesh. We, therefore, added a loop over 405 atoms at each step of chemical potential. It saved running time by simply doing this one time and save it into a vector. It was done using the **"count_type_atoms"**, that is called only one time at the beginning of the **"DoMonteCarlo"** function.

4.2 Concentration inconsistencies

Another problem, was that the results for the concentrations were not consistent with the theory. It was due to computing the concentration on the last step of the Monte-Carlo, instead of an average concentration (the concentration on all steps, after npeq steps for statistical equilibrium, then dividing by (npas-npeq) for an average)

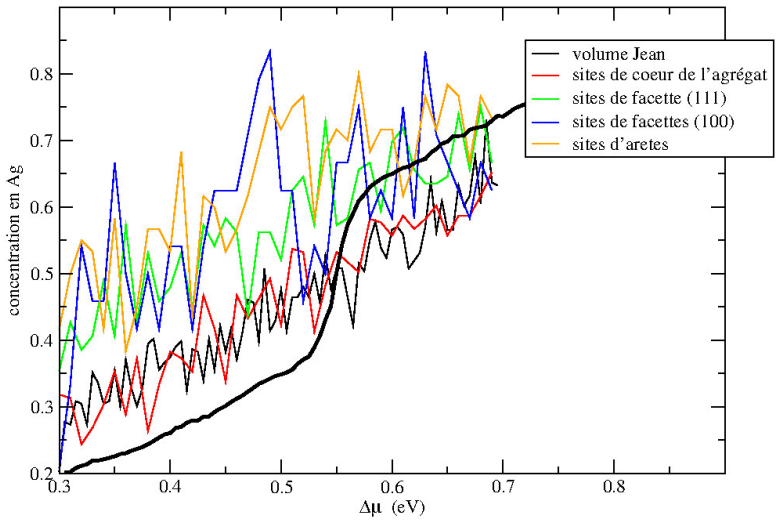

Figure 3: Inconsistent results of the concentrations, we should have obtained the drawn curve in black

5 Results

After solving those previous issues, we could then compute the programm and analyze the obtained results.

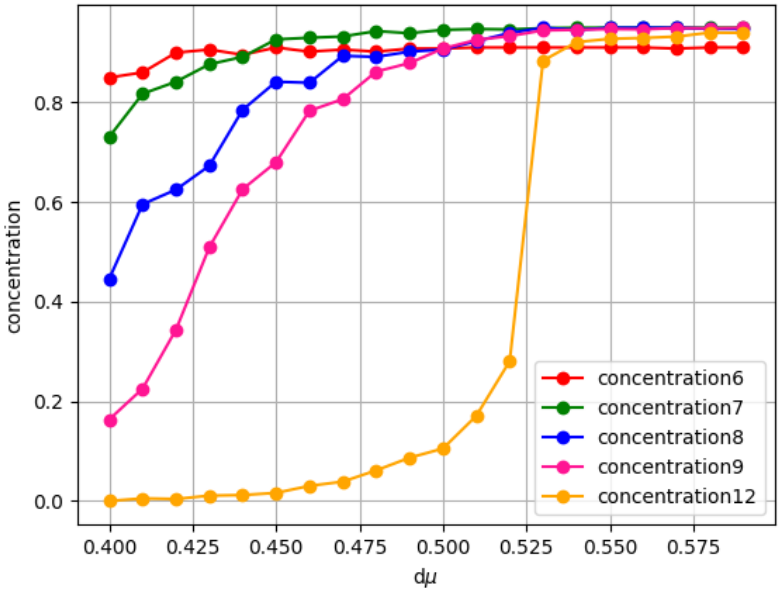

Figure 4: Concentrations in Ag for various areas within the mesh using 100 steps for Monte-Carlo, with a temperature of 500K

We can clearly see a jump in the contentration for the nanoparticules in the heart (12 neighbours) between 0.52 and 0.53. What happens ? At μ=0.4 J/mol, we can cleary see a

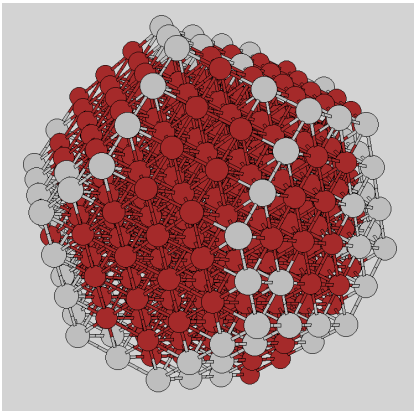

Figure 5: xmakemol vizualisation at T=500K and μ=0.4

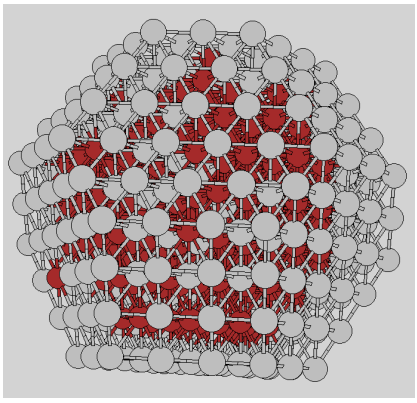

Figure 6: xmakemol vizualisation at T=500K and μ=0.52

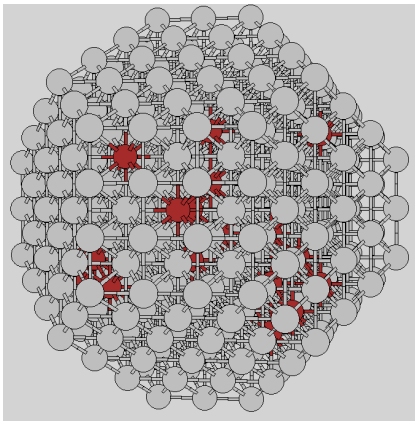

Figure 7: xmakemol vizualisation at T=500K and μ=0.53

preferential agregation on the surface and more specifically on the edges. As the chemical potential rises up, the surface gets filled with impurities and they start slowy gaining the bulk.

We can cleary see a jump at μ=0.53 J/mol, indeed at μ=à.52 J/mol, the bulk remained quite free of Ag, but it almost full of impurities in the next step. Jump vizualised in the (8) figure.

We runned our program again, but this time using 500 steps in Monte-carlo. We obtained similar results :

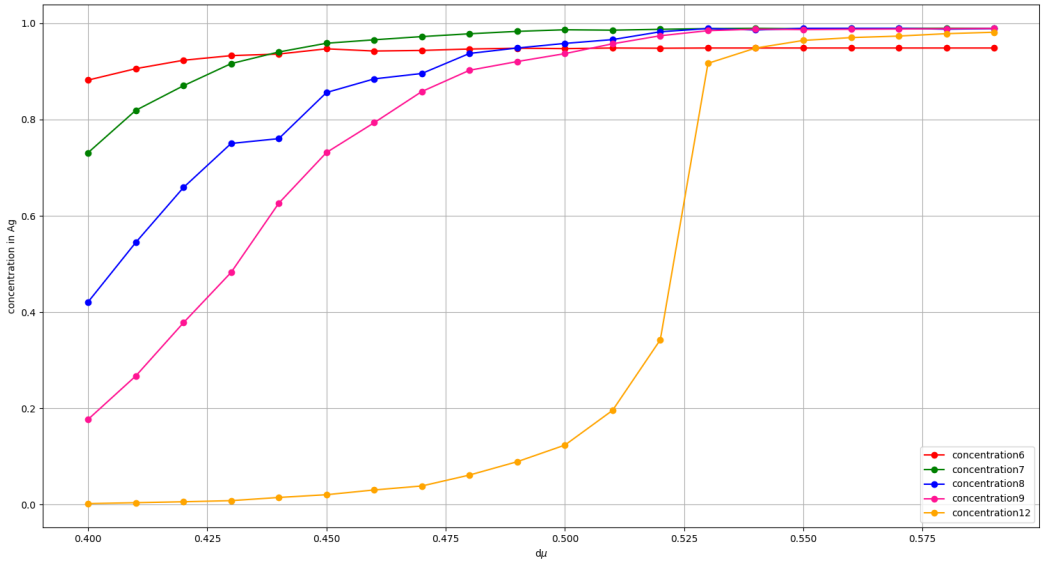

Figure 8: Concentrations in Ag for various areas within the mesh using 500 steps for Monte-Carlo, with a temperature of 500K

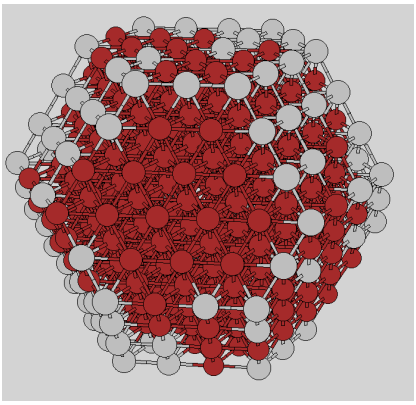

Figure 9: xmakemol vizualisation at T=500K and μ=0.4, 500 steps

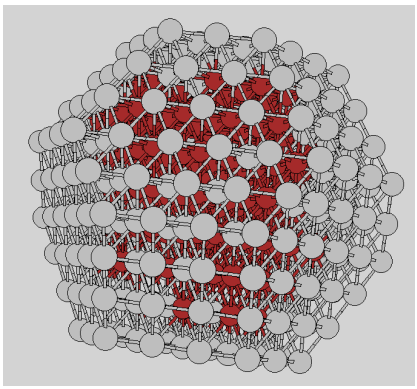

Figure 10: xmakemol vizualisation at T=500K and μ=0.52, 500 steps

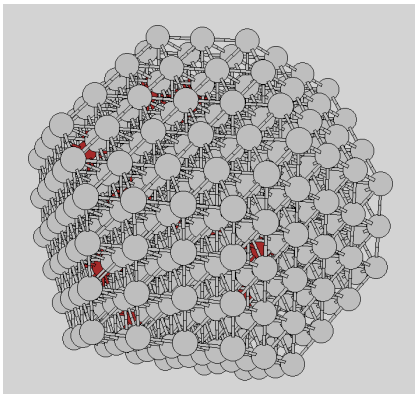

Figure 11: xmakemol vizualisation at T=500K and μ=0.53, 500 steps

We can cleary see a similar event of impurities agregation in the surface, that slowy fills up the bulk. We can still see a jump at μ=0.53.

6 Conclusion

In this project, the main goal was to simulate the segregation properties of metals nanoalloys. In order to do that we builded a "simple" tight-binding Ising model (TBIM), using Monte-Carlo techniques in c++. We created a namespace called "nano" that contains various classes used to improve the efficency of the code.

We were subject to various issues regarding eitheir the runtime or the consistency of the obtained results. We managed to improve the runtime by computing only the energy difference instead of the total energy.

We, then, vizualised the concentrations in impurities using python and noticed a preferential agregation in the surface of the system. It was verified using xmakemol. We also noticed a jump in the agregation of the bulk at μ=0.53, that was quite empty of impurities before that.

To conclude, this project allowed us to implement a very famous simulation technique (Metropolis) to simulate an Hamiltonian used to describe the effective pair_interactions of nanoparticules.

We verified computanionaly predicted behaviours.