MATLAB extended project

Interim assessment

On Tuesday 11th February 2020 (Friday 15th February 2020), between 14:00 and 16:00, you will meet with me for 10 minutes (room B331 in the TYC/CDT corridor on the ground floor of the RSM) for a short discussion to ensure the project is progressing well. This carries no credit, but is compulsory. I suggest we meet in the order given in the spreadsheet:

Tuesday 11th February 2020

1. Jiarui Zhang
2. Wonjun Choi
3. Jakub Lala
4. Yoonseo Lim
5. Zhongqi Ma
6. Saajan Shah
7. Bruno Emerick
8. Yinghe Zhu
9. Zixu Zhao
10. Marcus Auty-Jacklin

Friday 14th February 2020

1. Jiacheng Ruan
2. Junting Zhang
3. Fengyi Li
4. Han Wong
5. Jiwen Yu
6. Ziyu Zhao
7. Samir Choudhury-Asghar
8. Craig Lough
9. Gwyn Viriyaprapaikit
10. Zeyu Wang

Final report

The final report should not exceed **8 pages** (this does not include the front cover or the Python code listing). The pages should be A4, and the font 11 pt. This project is worth 48 points. To ensure you do not lose marks unnecessarily, **please follow the report template exactly**.

Background

Materials are most stable when at equilibrium, and this is achieved by making the Gibbs Free Energy (G) as small as possible. This free energy is given by G=H-TS where H is the enthalpy, T is the temperature and S is the entropy. From this equation you can see that making S bigger reduces the value of G, and the larger T is the greater the reduction. In other words, at higher temperatures it is favourable for materials to increase their entropy. Thus, as entropy is a measure of disorder, materials tend to become less ordered at higher temperatures.

One important contribution to entropy in alloys is the arrangement of the alloying element in the host metal. From the above argument we would expect the alloying element to distribute itself randomly (to maximize the entropy) but possibly becoming ordered at lower temperatures if this produces a lowering of energy.

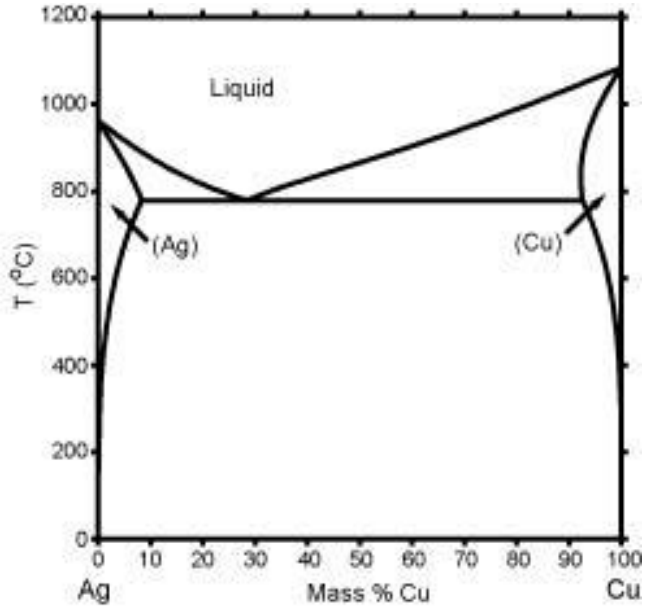

Figure 1: From http://www.metallurgy.nist.gov/phase/solder/agcu.html

This can be seen in binary eutectic systems. Consider the copper-silver system presented in Figure 1. At the Cu rich end (right hand side of the figure) below the eutectic temperature (779^{o} C) we find that we can dissolve only a few percent of Ag to form a random (disordered) alloy. If we keep the temperature fixed and add more Ag the system phase separates into nearly pure Ag and Cu regions, which is an ordered state. As we raise the temperature we find the higher the temperature the more Ag can dissolve in the Cu; this is because the lowering of the free energy by the increased entropy of mixing exceeds the increase in energy because of the size mismatch between the atoms. If we go still higher in temperature, we get solid Cu (with some Ag) plus a liquid solution of Cu and Ag. The liquid solution is disordered in two ways: the atoms are no longer constrained to occupy lattice sites; the two types of atom mix randomly. Finally, if the temperature is raised even further we get a pure liquid solution. We thus see a systematic loss of order as the temperature is raised.

In this project you will carry out a Monte Carlo computer simulation of a substitutional alloy in two dimensions using a program that you will need to write yourself. You can use **MATLAB** or **Python**. You will analyze the data to understand the effects of changing interaction energy, temperature and composition.

Model of alloy

We will model the alloy as a two dimensional system, which allows for fast simulations, and makes analyzing the results easy (we can visualize 2D images readily). We treat our system as a regular array of sites, each one of which can hold either an A atom (an alloying atom -- coloured red in the diagram below) or a B atom (a host atom – coloured blue in the diagram below). The system is a square, and each atom has 4 nearest neighbours. We assign an energy to each bond depending on the type of atoms. If the two atoms forming a bond are both type A, then the energy of the bond is E_{AA}. Similarly, if they are both type B the energy is E_{BB}. Finally, if the atoms are of different types, then the energy is E_{AB}. The total energy of your system is the sum of these bond energies.

From your lectures on the regular solution model you probably recall that we only need one value for the bonding energy, equivalent to $w= E_{AA}+E_{BB}-2E_{AB}$. Thus, for this project set $E_{AA}=E_{BB}=0$ eV for all the simulations, and consider three values for E_{AB}, which are -0.1 eV, 0.0 eV and 0.1 eV. If your temperature is in units of kelvin, then you will probably find it useful to use the Boltzmann constant in units of eV/K, which is $k_B = 8.617332\text{x}10^{-5}$ eV/K.

Periodic boundaries

We have to decide what to do about the edges of the box. The most realistic way to proceed is to effectively remove the boundaries by using periodic boundary conditions. That is, we assume that our central box is surrounded on all sides by images of itself. These images are represented by the shaded boxes surrounding the central box in Figure 2. Note that this can have the effect of making atoms on opposite sides of the box (e.g. atoms labeled a and b) neighbours. Thus, when computing the total energy, we have to consider atoms outside the central box. The expression for the total energy is

$$E_{tot} = \frac{1}{2}\sum_{i(cell)}\sum_{j(neighbour of i)} E_{ij}$$

where E_{ij} equals E_{AA}, E_{BB} or E_{AB} depending on the atom types *i* and *j*. Note that atom *i* must always be inside the central cell, while atom *j* can be inside or in one of the neighbouring cells.

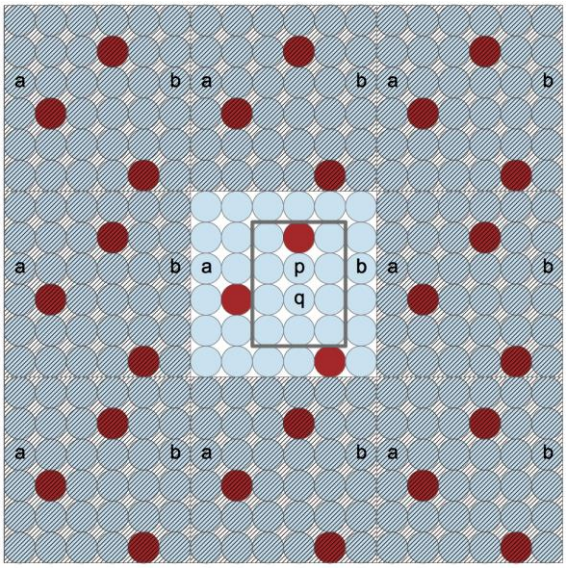

Figure 2: The structure of the simulation cell, in 2D, including periodic boundaries

Metropolis algorithm

Your program needs to do the following.

1. Set up an initial random arrangement of atoms in your central cell
2. Repeat the following steps multiple times until the system reaches equilibrium.
    a. Pick a pair of nearest neighbours at random (e.g. the atoms labeled p and q in the diagram above), and try to swap them. Let the change in energy of the system following the swap be $\Delta E=E_{after}-E_{before}$. Note that you can compute this change in energy by only considering the two atoms being swapped, and the other atoms they form bonds with (see the box around atoms p and q in the diagram). This helps make the program much more efficient.
    b. If $\Delta E \leq 0$ (the move lowers the energy) then keep the atoms in their swapped position.
    c. If $\Delta E > 0$ (the move raises the energy) then you need to perform one more test: if $\exp(-\Delta E/k_B T) > R$, where R is a random number between 0 and 1, then again leave the atoms in their swapped position, otherwise put them back.
3. Plot the final configuration.
4. Provide a measure of the randomness (see below)

Analyzing the data

To make comparing structures easier, we would like to quantify the ordering. To do this we will compute the distribution of the number of unlike neighbours for each site. That is, suppose one site has an atom of type A on it, and three of its neighbours are type B and one type A, then that site will have three unlike neighbours. We now compute the number of sites with zero unlike neighbours, the number with one, the number with two, etc. The maximum number of unlike neighbours is four. Using this distribution function, it is possible to distinguish between the types of ordering encountered by comparing with what you would get for a random distribution. If each site has Z neighbouring sites, and the fraction of alloying atoms is f, then for a random alloy, the probability of having n different neighbours is given by the binomial distribution, $P_n = {}^{Z}c_n[ff^{Z-n}(1-f)^n + (1-f)f^n(1-f)^{Z-n}]$, where ${}^{z}c_n = \frac{Z!}{n!(Z-n)!}$ is the number of ways to choose n items from a collection of Z items.

We will also want a single number that gives a measure of the overall ordering. I suggest using the mean number of unlike neighbours: $\acute{n} = \frac{\sum_n nN_n}{\sum_n N_n}$ where N_n is the number of sites with n unlike neighbours.

What you need to do

1. Write your Python program to perform the simulations in 2D. A skeleton code for the 2D case is provided at the end of this document that you can edit.
2. Test it carefully to make sure it works.
3. Run the program for a range of values of
    a. Energies E_{AB} (remember $E_{AA} = E_{BB} = 0$). Use values -0.1 eV, 0.0 eV and 0.1 eV.
    b. Compositions
    c. Temperatures. Start at room temperature (about 300 K) and go up from there.
4. For each case you will need to make sure the simulation has reached equilibrium. The easiest way to do this is to monitor the energy of the system and continue until it remains nearly constant. It should decrease over time initially.

5. Using the final configuration for each simulation, analyze your results using the method described above, as well as plots of the configuration, and present the data in a way that is straightforward to interpret. Here we are looking for general trends.
6. For a given alloy concentration and bond energy, we would like to find the temperature at which there is a change of phase. We can spot this by a sudden change in the average order $\acute{n}$ at the transition temperature. Once the simulation has reached equilibrium, run the simulation for more steps, and compute the average value of $\acute{n}$ over those additional steps. Now plot $\acute{n}$ against temperature to find the transition temperature for each of the alloy concentrations and bonding energies.
7. We will repeat the previous calculations but using the heat capacity instead of the order parameter. You could gather both sets of numbers at the same time if you like. The heat capacity is given by

$$C_v = \frac{< E^2 > - < E >^2}{k_B T^2}$$

   where <E> is the average energy, and <E^2> is the average of the square of the energy. We use heat capacity as this is a 2nd order phase transition with large fluctuations in the energy at the transition.

8. Compare the estimates of the transition temperatures.
9. How do the variations of order parameter and heat capacity depend on the size of your simulation cell?
10. Write up your results, **using the template**, and discuss what you have found.

Skeleton program

```
import numpy as np
import math as m
import matplotlib.pyplot as plt
from scipy.stats import binom

##########################################################################################
# GLOBAL VALUES
##########################################################################################
cellA = 0 # Matrix atom
cellB = 1 # Alloy atom

def orderRandom(Z, f):
    """
    ORDERRANDOM produces a distribtion function of the order parameter for a
    random alloy. The order parameter is just the number of AB bonds around a
    site. The distribution is computed from the binomial distribution.

       Input arguments
           Z  The number of neighbouring sites per site
           f  The fraction of alloy atoms

       Output arguments
           N       List of possible number of neighbours
           P       The probability distribution of the order parameter
    """
    #
    # Initialise the probability distribution
    N = np.linspace(0,Z,Z+1)
```

```
    P = np.linspace(0,0,Z+1)
    #
    # Run over allowed number of AB bonds around each site and compute the
    # probability distribution
    ADD CODE HERE
    return N, P

def order2D(config):
    """
    ORDER2D produces a distribtion function of the order
    parameter. The order parameter is just the number of AB bonds
    around a site.

    Input arguments
    config  The configuration

    Output arguments
    N       List of possible number of neighbours
    P       The probability distribution of the order parameter
    """
    ADD CODE HERE
    return N, P

def swapInfo(ixa, iya, dab, nBox, config, Ematrix):
    """
    SWAPINFO Returns the position of the neighbour and the energy change following a
swap

    Input arguments
       ixa         X coordinate of first atom
       iya         Y coordinate of first atom
       dab         Direction of second atom relative to first. There are four
                   possible directions, so this takes values between 1 and
                   4. Together with ixa and ixb, this allows the position
                   of the second atom to be computed. This calculation is
                   done by getNeighbour
       config      The configuration of alloy atoms
       nBox        System size
       Ematrix     The 2x2 matrix of bond energies

    Output arguments
       ixb         X coordinate of second atom
       iyb         Y coordinate of second atom
       dE          Energy change following swap
    """
    ADD CODE HERE
    return ixb, iyb, dE

def getNeighbour (nBox, ix1, iy1, d12):
    """
    GETNEIGHBOUR returns the position of a neighbouring atom

    Input arguments
       nBox    The size of the simulation box
       ix1     X coordinate of first atom
       iy1     Y coordinate of first atom
       d12     Direction of second atom relative to first

    Output arguments
```

```
        ix2     X coordinate of second atom
        iy2     Y coordinate of second atom
    """
    ADD CODE HERE
    #
    # Return the new coordinates
    return ix2, iy2

def alloy2D(nBox, fAlloy, nSweeps, nEquil, T, Eam, job):
    """
    ALLOY2D Performs Metropolis Monte Carlo of a lattice gas model of an alloy
    A random alloy is represented as a 2 dimensional lattice gas in which
    alloying atoms can exchange position with matrix atoms using the
    Metropolis alogorithm. The purpose is to show how alloys become more
    random as the temperature increases.

    Input arguments
    nBox    The size of the 2-D grid
    fAlloy  The fraction of sites occupied by alloying atoms
    nSweeps The total number of Monte Carlo moves
    nEquil  The number of Monte Carlo moves used to equilibrate the system
    T       The temperature (K)
    Eam     Alloy-matrix interaction energy (eV)
    job     Name or number given to this simulation. Useful for creating file names

    Output arguments
    nBar    The average number of unlike neighbours
    Ebar    The average energy
    C       The heat capacity
    """
    ADD CODE HERE
    #
    # Plot the configuration
    # Put extra zeros around border so pcolor works properly.
    config_plot = np.zeros((nBox+1, nBox+1))
    config_plot[0:nBox, 0:nBox] = config
    plt.figure(0)
    plt.pcolor(config_plot)
    plt.savefig(job+'-config.png')
    plt.close(0)
    #
    # Plot the energy
    plt.figure(1)
    plt.plot (Etable[0:nTable+1])
    plt.title ("Energy")
    plt.xlabel("Time step / 1000")
    plt.ylabel("Energy")
    plt.savefig(job+'-energy.png')
    plt.close(1)
    #
    # Plot the final neighbour distribution
    N, P = order2D(config)
    N0, P0 = orderRandom(4, fAlloy)
    plt.figure(2)
    bar_width = 0.35
    plt.bar(N , P, bar_width, label="Simulation")
    plt.bar(N0+bar_width, P0, bar_width, label="Random")
    plt.title ("Distribution of unlike neighbours")
    plt.xlabel("Number of unlike neighbours")
```

```
    plt.ylabel("Probability")
    plt.legend()
    plt.savefig(job+'-order.png')
    plt.close(2)
    #
    # Display the plots (GUI only)
    # plt.show()
    #
    # Print statistics
    nBar = np.dot(N,P)
    Ebar = Ebar/nStats
    E2bar = E2bar/nStats
    C = (E2bar - Ebar*Ebar)/(kT*kT)
    print('')
    print('Heat capacity                                = {0:7.3f}'.format(C),' kB')
    print('The average number of unlike neighbours is = {0:7.3f}'.format(nBar))
    #
    # Return the statistics
    return nBar, Ebar, C

##########################################
# MAIN FUNCTION
##########################################
#
# This invokes the operations in the required order
def main():
    #
    # Define the simulation parameters
    EDIT THESE VALUES AS NEEDED
    nBox    = 10
    nEquil  = 50000
    nSweeps = 100000
    fAlloy_list = [0.1, 0.2, 0.3, 0.4, 0.5]
    T_list = [300, 1000, 2000]
    Eam_list = [-0.1, 0.0, 0.1]
    #
    # Open file to save the statistics
    file = open ("stats.csv", "w")
    file.write('Job number, Alloy fraction, Temperature (K), Unlike bond energy (eV),
Average number of unlike neighbours, Average energy (eV), Heat capacity (kB)\n')
    #
    # Loop over values
    count = 0
    for fAlloy in fAlloy_list:
        for T in T_list:
            for Eam in Eam_list:
                count = count + 1
                job = '{:04d}'.format(count)
                #
                # Echo the parameters back to the user
                print ("")
                print ("Simulation ", job)
                print ("----------------")
                print ("Cell size                      = ", nBox)
                print ("Alloy fraction                 = ", fAlloy)
                print ("Total number of moves          = ", nSweeps)
                print ("Number of equilibration moves = ", nEquil)
                print ("Temperature                    = ", T, "K")
                print ("Bond energy                    = ", Eam, "eV")
```

```
                #
                # Run the simulation
                nBar, Ebar, C = alloy2D(nBox, fAlloy, nSweeps, nEquil, T, Eam, job)
                #
                # Write out the statistics
                file.write('{0:4d}, {1:6.4f}, {2:8.2f}, {3:5.2f}, {4:6.4f}, {5:14.7g},
{6:14.7g}\n'.format(count, fAlloy, T, Eam, nBar, Ebar, C))
    #
    # Close the file
    file.close()
    #
    # Sign off
    print('')
    print ("Simulations completed.")
#
# Ensure main is invoked
if __name__== "__main__":
    main()
```