Supplementary Material

Mechanisms of nucleation and defect growth in undercooled melt containing oxide clusters

Sepideh Kavousi and Mohsen Asle Zaeem*

Department of Mechanical Engineering, Colorado School of Mines, Golden, CO 80401, USA

1. Solid-Liquid Interface energy for Al

MD is a powerful approach for obtaining material properties that are costly and sometimes impossible to determine by current experimental techniques [1]. In this study, we used a well-developed technique named capillary fluctuation method (CFM) [2] for calculating the solid-liquid interface energy. CFM offers a precise description of the anisotropic nature of solid-liquid interface energy. In this method, the interface free energy is estimated based on the spectrum of interfacial fluctuations. Equation (S1) relates the interface stiffness to the amplitudes and modes of Fourier transform.

$$\gamma + d^2\gamma/d\theta^2 = \frac{k_B T}{bW\langle|A(k)|^2\rangle k^2} \quad \text{(S1)}$$

k_B, is the Boltzmann constant, $\langle|A(k)|^2\rangle$ is the mean squared amplitude of the Fourier modes and k is the mode wave number. γ is the CM interface free energy and $\gamma + d^2\gamma/d\theta^2$ is the interface stiffness, where θ is the angle between instantaneous local interface normal and the average orientation of the interface. Repeating the simulations for multiple orientations along the interface normal enables obtaining the stiffness for various orientations and fitting it to interface energy, shown in Equation (S2), is used to estimate the mean CM interface free energy and the corresponding anisotropy terms [3, 4]. Details of the CFM calculation are discussed in our previous works.

$$\gamma(\hat{n}) = \gamma_0\left[1 + \delta_1\left(3\sum_{i=1}^{3} n_i^4 - 3\right) + \delta_2\left(\sum_{i=1}^{3} n_i^6 + 30n_1^2n_2^2n_3^2\right)\right] \quad \text{(S2)}$$

$n_i(i = 1,2,3)$ are the interface normal components, γ_0 is the mean CM interface free energy, δ_1, and δ_2 are the anisotropy parameters. The slopes of the

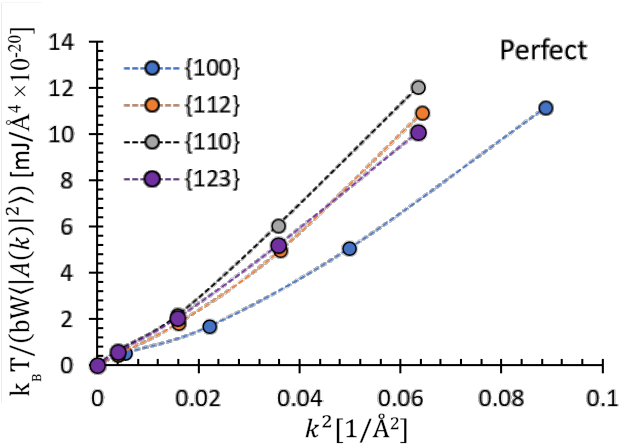

Figure S1: The variation of $k_BT/(bW\langle|A(k)|^2\rangle)$ versus k^2 for different orientations as obtained from MD simulation of Al

Fitting the stiffness obtained from Figure S1 to the orientation-dependent stiffness relationships gives γ_0 =173 mJ/m^2±2.3, δ_1=0.04±0.003, and δ_2=-0.02±0.002.

2. **Python Code for grain orientation and boundary transformation matrix:**

```
from ovito.data import *
import math
import numpy as numpy

def quaternions_to_colors(data_all):
    data_x = data_all[:,0]; data_y = data_all[:,1]; data_z = data_all[:,2]; data_w = data_all[:,3]
    data_size = numpy.sqrt(data_x**2 + data_y**2 + data_z**2)
    data_direction = numpy.full((data_all.shape[0], 3), (0.0, 0.0, 1.0))
    AA=data_all.shape[0]
    angle = numpy.zeros(AA)
    for ii in range(AA)
        numpy.true_divide(data_x[ii], data_size[ii], out=data_direction[ii,0],where=(data_size[ii]>1e-12)
        numpy.true_divide(data_y[ii], data_size[ii], out=data_direction[ii,1],where=(data_size[ii]>1e-12)
        numpy.true_divide(data_z[ii], data_size[ii], out=data_direction[ii,2],where=(data_size[ii]>1e-12)
        angle[ii] = 2.0 * numpy.arccos(data_w[ii], out=angle[ii], where=(data_size[ii]>1e-12))*180/3.14
    return angle

def axis(data_all):
    data_x = data_all[:,0]; data_y = data_all[:,1]; data_z = data_all[:,2]; data_w = data_all[:,3]
    data_size = numpy.sqrt(data_x**2 + data_y**2 + data_z**2)
    data_direction = numpy.full((data_all.shape[0], 3), (0.0, 0.0, 1.0))
    AA=data_all.shape[0]
    angle = numpy.zeros(AA)
    for ii in range(AA)
        numpy.true_divide(data_x[ii], data_size[ii], out=data_direction[ii,0],where=(data_size[ii]>1e-12)
        numpy.true_divide(data_y[ii], data_size[ii], out=data_direction[ii,1],where=(data_size[ii]>1e-12)
        numpy.true_divide(data_z[ii], data_size[ii], out=data_direction[ii,2],where=(data_size[ii]>1e-12)
        angle[ii] = 2.0 * numpy.arccos(data_w[ii], out=angle[ii], where=(data_size[ii]>1e-12))*180/3.14
     return data_direction

def matrix(data_all):
    data_x = data_all[:,0]; data_y = data_all[:,1]; data_z = data_all[:,2]; data_w = data_all[:,3]
    data_size = numpy.sqrt(data_x**2 + data_y**2 + data_z**2)
    data_direction = numpy.full((data_all.shape[0], 3), (0.0, 0.0, 1.0))
    AA=data_all.shape[0]
    for ii in range(AA)
        numpy.true_divide(data_x[ii], data_size[ii], out=data_direction[ii,0],where=(data_size[ii]>1e-12)
        numpy.true_divide(data_y[ii], data_size[ii], out=data_direction[ii,1],where=(data_size[ii]>1e-12)
```

```
        numpy.true_divide(data_z[ii], data_size[ii],
out=data_direction[ii,2],where=(data_size[ii]>1e-12)
    angle = numpy.zeros(AA)
    matrices = numpy.empty((AA,9))
    for ii in range (aa):
            matrices[ii,0] = 1.0-2.0*(data_y[ii]*data_y[ii] + data_z[ii]*data_z[ii])
            matrices[ii,1] = 2.0*(data_x[ii]*data_y[ii] + data_w[ii]*data_z[ii])
            matrices[ii,2] = 2.0*(data_x[ii]*data_z[ii] - data_w[ii]*data_y[ii])
            matrices[ii,3] = 2.0*(data_x[ii]*data_y[ii] - data_w[ii]*data_z[ii])
            matrices[ii,4] = 1.0-2.0*(data_x[ii]*data_x[ii] + data_z[ii]*data_z[ii])
            matrices[ii,5] = 2.0*(data_y[ii]*data_z[ii] + data_w[ii]*data_x[ii])
            matrices[ii,6] = 2.0*(data_x[ii]*data_z[ii] + data_w[ii]*data_y[ii])
            matrices[ii,7] = 2.0*(data_y[ii]*data_z[ii] - data_w[ii]*data_x[ii])
            matrices[ii,8] = 1.0-2.0*(data_x[ii]*data_x[ii] + data_y[ii]*data_y[ii])
    return matrices

def euler(data_all):
    data_x = data_all[:,0]; data_y = data_all[:,1]; data_z = data_all[:,2]; data_w = data_all[:,3]
    AA=data_all.shape[0]
    phi = numpy.zeros((AA,3))
    for ii in range (aa):
            phi[ii,0]=numpy.arctan2(2*(data_w[ii]*data_x[ii]+data_y[ii]*data_z[ii]),1-
2*(data_x[ii]*data_x[ii]+data_y[ii]*data_y[ii]))
            phi[ii,1]=numpy.arcsin(2*(data_w[ii]*data_y[ii]-data_z[ii]*data_x[ii]))
            phi[ii,2]=numpy.arctan(2*(data_w[ii]*data_z[ii]+data_y[ii]*data_x[ii])/(1-
2*(data_z[ii]*data_z[ii]+data_y[ii]*data_y[ii])))
    return phi

def modify(frame, data):
    orientations = data.particles['Orientation']
    data.particles_.create_property('charge', data=quaternions_to_colors(orientations))
    data.particles_.create_property('Color', data=axis(orientations))
    data.particles_.create_property('M',dtype=float, components=9, data=matrix(orientations))
    data.particles_.create_property('Color', data=euler(orientations))
```

3. Python code for determining the clusters

```
from ovito.io import *
from ovito.data import *
from ovito.modifiers import *
import numpy as np
for ii in range (0,201,40):
    node=import_file("solid-2-3-4-5/"+str(ii)+".xyz", columns =
        ["Particle Identifier", "Particle Type", "Position.X", "Position.Y", "Position.Z"])#+str(ii))
    data = node.source

#modify N,N1,latticeparam,d1,d2,Nx,Ny orderinterface
    def order_parameter_calc(frame, input, output):
        print("ii",ii)
        # Initialize neighbor finder object.
        # Visit the 14 nearest neighbors of each particle.
        N = 12;
```

```
            finder = NearestNeighborFinder(N, input)
            input.apply(CommonNeighborAnalysisModifier(cutoff=3.2), frame)
            ptype = input.particles['Structure Type']
            valuesz=np.zeros(input.particles.count,dtype=float)
            order_paramsx=input.particles_.create_property('OrderParameter',data=valuesz)
        # Loop over all input particles:
            for index in range(input.number_of_particles):
                d=0.0;i=1
            # Iterate over the neighbors of the current particle, starting with the closest:
                for neigh in finder.find(index):
                    if (ptype[neigh.index]==1 or ptype[neigh.index]==2):
                        d+=1
                    if d>6:
                        order_paramsx[index]=1
            export_file(input, "nuclei_"+str(ii)+".xyz", "xyz", columns = ["Particle Identifier", "Particle
    Type", "Position.X", "Position.Y", "Position.Z", "OrderParameter"])

            node.modifiers.append(PythonScriptModifier(function = order_parameter_calc))
            node.compute()
```

4. Python code for crystal structure

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
def bootstrap_replicate_1d(data,target):
    """Generate bootstrap replicate of 1D data."""
    bs_sample = np.random.choice(data, len(data))
#    print(len(data))
    n_target=0;n=0;
    for i in range(len(bs_sample)):
        n+=1
        if (bs_sample[i]==target):
            n_target+=1
#            print(n_target)

    return n_target/n
    #return bs_sample.groupby(["CNA"]).mean()
def draw_bs_reps(data,target, size=1):
    """Draw bootstrap replicates."""

    # Initialize array of replicates: bs_replicates
    bs_replicates = np.empty(size)

    # Generate replicates
    for i in range(size):
        bs_replicates[i] = bootstrap_replicate_1d(data["CNA"],target)

    return bs_replicates
```

```
time=[]
fcc_mean=[]
fcc_std=[]
other_mean=[]
other_std=[]
timestep=0.0005
frequency=500
size=200
n_step1=398
step=4

for i in range(0,n_step1+1,16):
    name="CNA."+str(i)
    data_array = np.loadtxt('data_1/'+name,skiprows=2,delimiter=" ")
#    print(data_array.shape)
    data = pd.DataFrame(data_array,columns=["number", "CNA"])
#    print(data.head())

    bs_replicates_fcc = draw_bs_reps(data,1.0,size)
    n_fcc_mean= np.mean(bs_replicates_fcc)
    n_fcc_std= np.std(bs_replicates_fcc)

    time.append(i*frequency*timestep)
    fcc_mean.append(n_fcc_mean)
    fcc_std.append(n_fcc_std)

    bs_replicates_other = draw_bs_reps(data,0.0,size)
    n_other_mean= np.mean(bs_replicates_other)
    n_other_std= np.std(bs_replicates_other)
    print(i,n_other_mean, n_other_std,n_fcc_mean, n_fcc_std)
    other_mean.append(n_other_mean)
    other_std.append(n_other_std)
laststep_1=np.floor(n_step1/step)*4
#for j in range(0,n_step2+1,160):
n_step2=1099
for j in range(0,n_step2+1,16):
    name="CNA."+str(j)
    data_array = np.loadtxt('data_2/'+name,skiprows=2,delimiter=" ")
#    print(data_array.shape)
    data = pd.DataFrame(data_array,columns=["number", "CNA"])
#    print(data.head())

    bs_replicates_fcc = draw_bs_reps(data,1.0,size)
    n_fcc_mean= np.mean(bs_replicates_fcc)
    n_fcc_std= np.std(bs_replicates_fcc)

    time.append((j+laststep_1)*frequency*timestep)
    fcc_mean.append(n_fcc_mean)
    fcc_std.append(n_fcc_std)

    bs_replicates_other = draw_bs_reps(data,0.0,size)
```

```
    n_other_mean= np.mean(bs_replicates_other)
    n_other_std= np.std(bs_replicates_other)
    print(laststep_1+j,n_other_mean, n_other_std,n_fcc_mean, n_fcc_std)
    other_mean.append(n_other_mean)
    other_std.append(n_other_std)
plt.plot(time, fcc_mean, color='orange',label="fcc")
plt.fill_between(time, np.array(fcc_mean)-np.array(fcc_std),
np.array(fcc_mean)+np.array(fcc_std),alpha=0.3, color="blue")

plt.plot(time, other_mean, color='blue',label="Amorphous")
plt.fill_between(time, np.array(other_mean)-np.array(other_std),
np.array(other_mean)+np.array(other_std),alpha=0.3, color="red")
plt.xlabel("time (ps)",size=15)
plt.xticks(fontsize=15)
plt.ylabel("phase fraction",size=15)
plt.yticks(fontsize=15)
plt.title("Al-TO",size=15)
plt.legend(fontsize=15)
plt.savefig("image/PF_TO.jpeg",bbox_inches='tight',dpi=1200) #facecolor='black',
```

5. Python code for twin boundary characterization

```
from ovito.io import *
from ovito.data import *
from ovito.modifiers import *
import numpy as np
import time

# The number of neighbors to take into account per atom:
num_neighbors = 14; A1_max=5; A2_max=0.02;
filename="solid_6_189.data.xyz"
node=import_file(str(filename), columns =
      ["Particle Identifier", "Position.X", "Position.Y", "Position.Z", "Structure Type", "Charge",
"Velocity.X", "Velocity.Y", "Velocity.Z"])#+str(ii))

gg=open("grains_data.dat","w")
def prop(frame,input,output):
    neigh_finder = NearestNeighborFinder(num_neighbors, input)
    position=input.particles['Position']
    input.apply(CommonNeighborAnalysisModifier(cutoff=3.2), frame)
    types = input.particles['Structure Type']
    iden=input.particles['Particle Identifier']
    angle=input.particles['Charge']
    axis=input.particles['Velocity']

    valuesx=np.zeros(input.particles.count,dtype=float)
    order_paramsx=input.particles_.create_property('OrderParameterx',data=valuesx)
    num_prop=0
    grain=[]
    for i in range(input.particles.count):#input.particles.count
#       print(i,types[i])
```

```
      if (types[i]==1 and order_paramsx[i] ==0): #
         print(i)
         for k in range(i):
            if (types[k]==1):
#                print("All",i,order_paramsx[i] ,k, order_paramsx[k])
               A1=abs(angle[k]-angle[i])
               A2=abs(abs(np.inner(axis[k],axis[i]))-1.0)
               if (order_paramsx[i]==0.0):
                  if (A1<A1_max and A2<A2_max):
                     order_paramsx[i]=order_paramsx[k]
#                      print("Replace",order_paramsx[i],order_paramsx[k])
#                      time.sleep(3)
               if (order_paramsx[i]!=0.0 and order_paramsx[i]!=order_paramsx[k]):
                   garbage=0
#                   if (A1<A1_max and A2<A2_max):
#                      print("ERROR first",i,order_paramsx[i] ,k, order_paramsx[k])
#                      time.sleep(3)
#                print("Final",i,order_paramsx[i] ,k, order_paramsx[k])
#                time.sleep(3)
         for neigh in neigh_finder.find(i):
            if (types[neigh.index]==1):

               A1=abs(angle[i]-angle[neigh.index])
               A2=abs(abs(np.inner(axis[i],axis[neigh.index]))-1.0)
               if (A1<A1_max and A2<A2_max):
                  if   ((order_paramsx[i]*order_paramsx[neigh.index]   !=0.0)   and   (order_paramsx[i]
!=order_paramsx[neigh.index])):
                      garage=0
#                      print("ERROR neigh",i,order_paramsx[i] ,neigh.index, order_paramsx[neigh.index])
#                      time.sleep(3)

                  elif ((order_paramsx[i] ==0.0) and (order_paramsx[neigh.index] ==0.0)):
                     num_prop=num_prop+1
                     order_paramsx[i]=num_prop
                     order_paramsx[neigh.index]=num_prop
                     grain.append(1)

                  elif (order_paramsx[i]!=0.0 and order_paramsx[neigh.index]==0.0  ):
                     order_paramsx[neigh.index]= order_paramsx[i]
                     aa=int(order_paramsx[i])-1
                     grain[aa]=grain[aa]+1

                  elif (order_paramsx[neigh.index]!=0.0 and order_paramsx[i]==0.0):
                     order_paramsx[i]= order_paramsx[neigh.index]
                     aa=int(order_paramsx[neigh.index])-1
                     grain[aa]=grain[aa]+1

   gg.write(str(grain))
   export_file(input, "o_solid_6_189.xyz", "xyz", columns = ["Particle Identifier", "Position.X",
"Position.Y", "Position.Z", "Structure Type", "Charge", "Velocity.X", "Velocity.Y", "Velocity.Z",
"OrderParameterx"])
```

```
    node.modifiers.append(prop)
    dd=node.compute()
```

6. Python code for thermodynamic modeling of TB

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.ticker as mticker
def V_het(h,γt,γsl,L,Tm,Ti,theta):
   return 1.0/6.0*3.14*(h*6)*L*(Ti)/Tm, 1.0/6.0*3.14*3*(2/np.tan(theta)-(1+np.cos(theta))/np.sin(theta)-
(1+np.cos(theta))/np.sin(theta)*(1/(np.tan(theta))**2))*h**2*L*(Ti)/Tm,
1.0/6.0*3.14*(1+(3/(np.tan(theta))**2))*h**3*L*(Ti)/Tm
def dV_het(h,γt,γsl,L,Tm,Ti,theta):
   return 1.0/6.0*3.14*(2*h*6)*L*(Ti)/Tm, 1.0/6.0*3.14*3*(2/np.tan(theta)-
(1+np.cos(theta))/np.sin(theta)-(1+np.cos(theta))/np.sin(theta)*(1/(np.tan(theta))**2))*h**2*L*(Ti)/Tm

def V_hom(h,γt,γsl,L,Tm,Ti):
   return 3.14*h*L*(Ti)/Tm,0.0,0.0
def dV_hom(h,γt,γsl,L,Tm,Ti):
   return 2*3.14*h*L*(Ti)/Tm,0.0


def S_5fold_hom(h,γt,γsl,L,Tm,Ti):
   return 0.0,5*h*γt,0.0
def dS_5fold_hom(h,γt,γsl,L,Tm,Ti):
   return 0.0,5*h*γt

def S_5fold_het(h,γt,γsl,L,Tm,Ti,theta):
   return 0.0,5*h*γt,5/2*h**2*γt/np.tan(3.14-theta)
def dS_5fold_het(h,γt,γsl,L,Tm,Ti,theta):
   return 0.0,5*h*γt


def S_lam(h,γt,γsl,L,Tm,Ti):
   return 3.14*γt,0.0,0.0
def dS_lam(h,γt,γsl,L,Tm,Ti):
   return 2*3.14*γt,0.0

def S_sl(h,γt,γsl,L,Tm,Ti):
   return 0.0,2*3.14*h*γsl,0.0
def dS_sl(h,γt,γsl,L,Tm,Ti):
   return 0.0,2*3.14*h*γsl


def G_hom_perfect(X,h,γt,γsl,L,Tm,dT):
   V_hom=3.14*X**2*h
   S_sl=2*3.14*X*h
   return V_hom*L*dT/Tm+S_sl*γsl
```

```
def G_hom_5fold(X,h,γt,γsl,L,Tm,dT):
    V_hom=3.14*X**2*h
    S_sl=2*3.14*X*h
    S_5fold=5*X*h
    return V_hom*L*dT/Tm+S_sl*γsl+S_5fold*γt
def G_hom_lam(X,h,γt,γsl,L,Tm,dT):
    V_hom=3.14*X**2*h
    S_sl=2*3.14*X*h
    S_lam=3.14*X**2
    return V_hom*L*dT/Tm+S_sl*γsl+S_lam*γt


def G_het_perfect(X,h,γt,γsl,L,Tm,dT,theta):
    V_het=1.0/6.0*3.14*(X**2*h*6+3*(2/np.tan(theta)-(1+np.cos(theta))/np.sin(theta)-
(1+np.cos(theta))/np.sin(theta)*(1/(np.tan(theta))**2))*X*h**2+(1+(3/(np.tan(theta))**2))*h**3)
    S_sl=2*3.14*X*h
    return V_het*L*dT/Tm+S_sl*γsl

def G_het_5fold(X,h,γt,γsl,L,Tm,dT,theta):
    V_het=1.0/6.0*3.14*(X**2*h*6+3*(2/np.tan(theta)-(1+np.cos(theta))/np.sin(theta)-
(1+np.cos(theta))/np.sin(theta)*(1/(np.tan(theta))**2))*X*h**2+(1+(3/(np.tan(theta))**2))*h**3)
    S_sl=2*3.14*X*h
    S_5fold=5*(X+h/2/np.tan(3.14-theta))*h
    return V_het*L*dT/Tm+S_sl*γsl+S_5fold*γt
def G_het_lam(X,h,γt,γsl,L,Tm,dT,theta):
    V_het=1.0/6.0*3.14*(X**2*h*6+3*(2/np.tan(theta)-(1+np.cos(theta))/np.sin(theta)-
(1+np.cos(theta))/np.sin(theta)*(1/(np.tan(theta))**2))*X*h**2+(1+(3/(np.tan(theta))**2))*h**3)
    S_sl=2*3.14*X*h
    S_lam=3.14*X**2
    return V_het*L*dT/Tm+S_sl*γsl+S_lam*γt


γt=       0.08; γsl=0.175 ;h=6.96E-10;   kB=1.38E-23;   L=9.47E+08;Tm=940;dT1=-140;dT2=-360;dT3=-
440;theta=157.5*3.14/180
T=np.arange(-10,-510,-10,dtype=float) #undercooling range
print(T)
r=np.arange(0.1e-9,7.01e-9,0.02e-9,dtype=float) #radius range
DG_hom_perfect_dT1=[];DG_hom_perfect_dT3=[]
DG_hom_lam_dT1=[];DG_hom_lam_dT3=[]
DG_hom_5fold_dT1=[];DG_hom_5fold_dT3=[]
for ri in r:
    DG_hom_perfect_dT1.append(G_hom_perfect(ri,h,γt,γsl,L,Tm,dT1))
    DG_hom_perfect_dT3.append(G_hom_perfect(ri,h,γt,γsl,L,Tm,dT3))

    DG_hom_lam_dT1.append(G_hom_lam(ri,h,γt,γsl,L,Tm,dT1))
    DG_hom_lam_dT3.append(G_hom_lam(ri,h,γt,γsl,L,Tm,dT3))

    DG_hom_5fold_dT1.append(G_hom_5fold(ri,h,γt,γsl,L,Tm,dT1))
    DG_hom_5fold_dT3.append(G_hom_5fold(ri,h,γt,γsl,L,Tm,dT3))

DG_het_perfect_dT1=[];DG_het_perfect_dT3=[]
```

```
DG_het_lam_dT1=[];DG_het_lam_dT3=[]
DG_het_5fold_dT1=[];DG_het_5fold_dT3=[]
for ri in r:
    DG_het_perfect_dT1.append(G_het_perfect(ri,h,γt,γsl,L,Tm,dT1,theta))
    DG_het_perfect_dT3.append(G_het_perfect(ri,h,γt,γsl,L,Tm,dT3,theta))

    DG_het_lam_dT1.append(G_het_lam(ri,h,γt,γsl,L,Tm,dT1,theta))
    DG_het_lam_dT3.append(G_het_lam(ri,h,γt,γsl,L,Tm,dT3,theta))

    DG_het_5fold_dT1.append(G_het_5fold(ri,h,γt,γsl,L,Tm,dT1,theta))
    DG_het_5fold_dT3.append(G_het_5fold(ri,h,γt,γsl,L,Tm,dT3,theta))

plt.clf()
plt.rcParams.update({'font.family':'Calibri'})
fig,axes=plt.subplots(2,2, sharey=True)
ticks=np.arange(0,7.0e-9,0.1e-9)

x_dT1=[0,7]
y_dT1=[52*kB*(Tm+dT1),52*kB*(Tm+dT1)]
axes[0,0].plot(x_dT1,y_dT1,label="52$k_B$T",color="r")
axes[0,0].plot(r*1e9,DG_hom_perfect_dT1,label="Perfect")
axes[0,0].plot(r*1e9,DG_hom_lam_dT1,label="Lamellar TB")
axes[0,0].plot(r*1e9,DG_hom_5fold_dT1,label="5-fold TB")
axes[0,0].set_yscale("log")
axes[0,0].set_ylim([1.0e-20,1.0e-17])
axes[0,0].set_xlim([0,7])
axes[0,0].yaxis.set_major_locator(mticker.LogLocator(numticks=999))
axes[0,0].yaxis.set_minor_locator(mticker.LogLocator(numticks=999, subs="auto"))
axes[0,0].set_xticks(ticks,minor=True)
#axes[0,0].legend(frameon=False,loc='best',bbox_to_anchor=(0.4, 0.29, 0.5, 0.5))
axes[0,0].set_ylabel("ΔG (J)",fontstyle="italic")
axes[0,0].set_title("Nucleation on Al ΔT=140 K",fontsize=10)



x_dT3=[0,7]
y_dT3=[52*kB*(Tm+dT3),52*kB*(Tm+dT3)]
axes[0,1].plot(x_dT3,y_dT3,label="525$k_B$T",color="r")
axes[0,1].plot(r*1e9,DG_hom_perfect_dT3,label="Perfect")
axes[0,1].plot(r*1e9,DG_hom_lam_dT3,label="Lamellar TB")
axes[0,1].plot(r*1e9,DG_hom_5fold_dT3,label="5-fold TB")
axes[0,1].set_yscale("log")
axes[0,1].set_ylim([1.0e-20,1.0e-17])
axes[0,1].set_xlim([0,4.0])
axes[0,1].yaxis.set_major_locator(mticker.LogLocator(numticks=999))
axes[0,1].yaxis.set_minor_locator(mticker.LogLocator(numticks=999, subs="auto"))
axes[0,1].set_xticks(ticks,minor=True)
axes[0,1].legend(frameon=False,fontsize=9,loc='best',bbox_to_anchor=(0.0, 0.06, 1, 1))
axes[0,1].set_title("Nucleation on Al ΔT=440 K",fontsize=10)
```

```
y_dT1=[52*kB*(Tm+dT1),52*kB*(Tm+dT1)]
axes[1,0].plot(x_dT1,y_dT1,label="52kBT",color="r")
axes[1,0].plot(r*1e9,DG_het_perfect_dT1,label="Perfect")
axes[1,0].plot(r*1e9,DG_het_lam_dT1,label="lamellar")
axes[1,0].plot(r*1e9,DG_het_5fold_dT1,label="5fold")
axes[1,0].set_yscale("log")
axes[1,0].set_ylim([1.0e-20,3.0e-17])
axes[1,0].yaxis.set_major_locator(mticker.LogLocator(numticks=999))
axes[1,0].yaxis.set_minor_locator(mticker.LogLocator(numticks=999, subs="auto"))
axes[1,0].set_xticks(ticks,minor=True)
#axes[1,0].legend(frameon=False)
axes[1,0].set_ylabel("ΔG (J)",fontstyle="italic")
axes[1,0].set_xlabel("r (nm)",fontstyle="italic")
axes[1,0].set_title("Nucleation on $Al_2O_3$ ΔT=140 K",y=1.0, pad=-10,fontsize=10)


axes[1,1].plot(x_dT3,y_dT3,label="52kBT",color="r")
axes[1,1].plot(r*1e9,DG_het_perfect_dT3,label="Perfect")
axes[1,1].plot(r*1e9,DG_het_lam_dT3,label="Lamellar")
axes[1,1].plot(r*1e9,DG_het_5fold_dT3,label="5fold")
axes[1,1].set_yscale("log")
axes[1,1].set_ylim([1.0e-20,3.0e-17])
axes[1,1].set_xlim([0,4.0])
axes[1,1].yaxis.set_major_locator(mticker.LogLocator(numticks=999))
axes[1,1].yaxis.set_minor_locator(mticker.LogLocator(numticks=999, subs="auto"))
axes[1,1].set_xticks(ticks,minor=True)
#axes[1,1].legend(frameon=False)
axes[1,1].set_xlabel("r (nm)",fontstyle="italic")
axes[1,1].set_title("Nucleation on $Al_2O_3$ ΔT=440 K",y=1.0, pad=-10,fontsize=10)
plt.subplots_adjust(wspace=0.2,hspace=0.3)
plt.savefig('dG.png', dpi=1200,bbox_inches='tight',pad_inches = 0)
plt.show()

plt.clf()


coef_hom_perfect=[0,0,0];rmax1_hom_perfect=[];rmax2_hom_perfect=[]
coef_hom_perfect2=[0,0,0];rc1_hom_perfect=[];rc2_hom_perfect=[]
for Ti in T:
    coef_hom_perfect=[0,0,0]
    coef_hom_perfect2=[0,0,0]
    a1,a2,a3=V_hom(h,γt,γsl,L,Tm,Ti)
    coef_hom_perfect[0]+=a1;coef_hom_perfect[1]+=a2;coef_hom_perfect[2]+=a3
    a1,a2,a3=S_sl(h,γt,γsl,L,Tm,Ti)
    coef_hom_perfect[0]+=a1;coef_hom_perfect[1]+=a2;coef_hom_perfect[2]+=a3
    coef_hom_perfect[2]+=-52*kB*(Tm+Ti)
    rmax1_hom_perfect.append(np.roots(coef_hom_perfect)[0]*1e9)
    rmax2_hom_perfect.append(np.roots(coef_hom_perfect)[1]*1e9)

    b1,b2=dV_hom(h,γt,γsl,L,Tm,Ti)
```

```
    coef_hom_perfect2[0]+=b1;coef_hom_perfect2[1]+=b2;
    b1,b2=dS_sl(h,γt,γsl,L,Tm,Ti)
    coef_hom_perfect2[0]+=b1;coef_hom_perfect2[1]+=b2;
    rc1_hom_perfect.append(np.roots(coef_hom_perfect2)[0]*1e9)
    rc2_hom_perfect.append(np.roots(coef_hom_perfect2)[1]*1e9)

coef_hom_lamellar=[0,0,0];rmax1_hom_lamellar=[];rmax2_hom_lamellar=[]
coef_hom_lamellar2=[0,0,0];rc1_hom_lamellar=[];rc2_hom_lamellar=[]
for Ti in T:
    coef_hom_lamellar=[0,0,0]
    coef_hom_lamellar2=[0,0,0]
    a1,a2,a3=V_hom(h,γt,γsl,L,Tm,Ti)
    coef_hom_lamellar[0]+=a1;coef_hom_lamellar[1]+=a2;coef_hom_lamellar[2]+=a3
    a1,a2,a3=S_sl(h,γt,γsl,L,Tm,Ti)
    coef_hom_lamellar[0]+=a1;coef_hom_lamellar[1]+=a2;coef_hom_lamellar[2]+=a3
    a1,a2,a3=S_lam(h,γt,γsl,L,Tm,Ti)
    coef_hom_lamellar[0]+=a1;coef_hom_lamellar[1]+=a2;coef_hom_lamellar[2]+=a3
    coef_hom_lamellar[2]+=-52*kB*(Tm+Ti)
    rmax1_hom_lamellar.append(np.roots(coef_hom_lamellar)[0]*1e9)
    rmax2_hom_lamellar.append(np.roots(coef_hom_lamellar)[1]*1e9)

    b1,b2=dV_hom(h,γt,γsl,L,Tm,Ti)
    coef_hom_lamellar2[0]+=b1;coef_hom_lamellar2[1]+=b2;
    b1,b2=dS_sl(h,γt,γsl,L,Tm,Ti)
    coef_hom_lamellar2[0]+=b1;coef_hom_lamellar2[1]+=b2;
    b1,b2=dS_lam(h,γt,γsl,L,Tm,Ti)
    coef_hom_lamellar2[0]+=b1;coef_hom_lamellar2[1]+=b2;
    rc1_hom_lamellar.append(np.roots(coef_hom_lamellar2)[0]*1e9)
    rc2_hom_lamellar.append(np.roots(coef_hom_lamellar2)[1]*1e9)



coef_hom_5fold=[0,0,0];rmax1_hom_5fold=[];rmax2_hom_5fold=[]
coef_hom_5fold2=[0,0,0];rc1_hom_5fold=[];rc2_hom_5fold=[]
for Ti in T:
    coef_hom_5fold=[0,0,0]
    coef_hom_5fold2=[0,0,0]
    a1,a2,a3=V_hom(h,γt,γsl,L,Tm,Ti)
    coef_hom_5fold[0]+=a1;coef_hom_5fold[1]+=a2;coef_hom_5fold[2]+=a3
    a1,a2,a3=S_sl(h,γt,γsl,L,Tm,Ti)
    coef_hom_5fold[0]+=a1;coef_hom_5fold[1]+=a2;coef_hom_5fold[2]+=a3
    a1,a2,a3=S_5fold_hom(h,γt,γsl,L,Tm,Ti)
    coef_hom_5fold[0]+=a1;coef_hom_5fold[1]+=a2;coef_hom_5fold[2]+=a3
    coef_hom_5fold[2]+=-52*kB*(Tm+Ti)
    rmax1_hom_5fold.append(np.roots(coef_hom_5fold)[0]*1e9)
    rmax2_hom_5fold.append(np.roots(coef_hom_5fold)[1]*1e9)
    b1,b2=dV_hom(h,γt,γsl,L,Tm,Ti)
    coef_hom_5fold2[0]+=b1;coef_hom_5fold2[1]+=b2;
    b1,b2=dS_sl(h,γt,γsl,L,Tm,Ti)
    coef_hom_5fold2[0]+=b1;coef_hom_5fold2[1]+=b2;
    b1,b2=dS_5fold_hom(h,γt,γsl,L,Tm,Ti)
```

```
    coef_hom_5fold2[0]+=b1;coef_hom_5fold2[1]+=b2;
    rc1_hom_5fold.append(np.roots(coef_hom_5fold2)[0]*1e9)
    rc2_hom_5fold.append(np.roots(coef_hom_5fold2)[1]*1e9)
#    print(coef_hom_5fold2[0],coef_hom_5fold2[1],Ti)


tickes2=[0,150,300,450]
fig2,axes2=plt.subplots(2,3, sharey=False,sharex=True)
plt.xticks(tickes2,fontsize=10)

#plt.plot(Tm-T,rmax1_hom_perfect,label="rmax1_hom_perfect")
axes2[0,0].plot(-T,rmax2_hom_perfect,label="$r_{max}$")
axes2[0,0].plot(-T,rc1_hom_perfect,label="$r_{c}$")
#plt.plot(Tm-T,rc2_hom_perfect,label="rc2_hom_perfect")
axes2[0,0].legend(frameon=False,fontsize=11,loc='best',bbox_to_anchor=(0.25, 0.6, 0.2, 0.2))
axes2[0,0].set_ylabel("r (nm)",fontstyle="italic")
axes2[0,0].set_title("Nucleation on Al",fontsize=10)
axes2[0,0].text(320,16,"perfect",fontsize=11)
axes2[0,0].set_xlim([0,500])

axes2[0,1].plot(-T,rmax2_hom_lamellar,label="$r_{max}$")
axes2[0,1].plot(-T,rc1_hom_lamellar,label="$r_{c}$")
#plt.plot(Tm-T,rc2_hom_perfect,label="rc2_hom_perfect")
#axes2[0,0].set_ylabel("r (nm)",fontstyle="italic")
axes2[0,1].set_title("Nucleation on Al",fontsize=10)
axes2[0,1].text(220,23,"Lamellar TB")

axins2=axes2[0,1].inset_axes([0.48,0.23,0.4,0.3])
tickes4=[150,300]
axins2.set_xticks(tickes4)
axins2.plot(-T,rmax2_hom_lamellar,label="$r_{max}$")
axins2.plot(-T,rc1_hom_lamellar,label="$r_{c}$")
axins2.set_xlim([150,300])
axins2.set_ylim([0,2.5])
axins2.tick_params(axis='both', which='major', labelsize=9)
axes2[0,1].indicate_inset_zoom(axins2)


#plt.plot(Tm-T,rmax1_hom_5fold,label="rmax1_hom_5fold")
axes2[0,2].plot(-T,rmax2_hom_5fold,label="rmax2_hom_5fold")
axes2[0,2].plot(-T,rc1_hom_5fold,label="rc1_hom_5fold")
#plt.plot(Tm-T,rc2_hom_5fold,label="rc2_hom_5fold")
#axes2[0,2].legend(frameon=False)
#axes2[0,0].set_ylabel("r (nm)",fontstyle="italic")
axes2[0,2].set_title("Nucleation on Al",fontsize=10)
axes2[0,2].text(260,21.5,"5-fold TB")

axins3=axes2[0,2].inset_axes([0.48,0.35,0.4,0.3])
tickes5=[150,300]
axins3.set_xticks(tickes5)
axins3.plot(-T,rmax2_hom_5fold,label="$r_{max}$")
```

```
axins3.plot(-T,rc1_hom_5fold,label="$r_{c}$")
axins3.set_xlim([150,300])
axins3.set_ylim([0,2.5])
axins3.tick_params(axis='both', which='major', labelsize=9)
axes2[0,2].indicate_inset_zoom(axins3)



coef_het_perfect=[0,0,0];rmax1_het_perfect=[];rmax2_het_perfect=[]
coef_het_perfect2=[0,0,0];rc1_het_perfect=[];rc2_het_perfect=[]
for Ti in T:
    coef_het_perfect=[0,0,0]
    coef_het_perfect2=[0,0,0]
    a1,a2,a3=V_het(h,γt,γsl,L,Tm,Ti,theta)
    coef_het_perfect[0]+=a1;coef_het_perfect[1]+=a2;coef_het_perfect[2]+=a3
    a1,a2,a3=S_sl(h,γt,γsl,L,Tm,Ti)
    coef_het_perfect[0]+=a1;coef_het_perfect[1]+=a2;coef_het_perfect[2]+=a3
    coef_het_perfect[2]+=-52*kB*(Tm+Ti)
    rmax1_het_perfect.append(np.roots(coef_het_perfect)[0]*1e9)
    rmax2_het_perfect.append(np.roots(coef_het_perfect)[1]*1e9)

    b1,b2=dV_het(h,γt,γsl,L,Tm,Ti,theta)
    coef_het_perfect2[0]+=b1;coef_het_perfect2[1]+=b2;
    b1,b2=dS_sl(h,γt,γsl,L,Tm,Ti)
    coef_het_perfect2[0]+=b1;coef_het_perfect2[1]+=b2;
    rc1_het_perfect.append(np.roots(coef_het_perfect2)[0]*1e9)
    rc2_het_perfect.append(np.roots(coef_het_perfect2)[1]*1e9)
#    print(coef_het_perfect2[0],coef_het_perfect2[1],Ti)

coef_het_lamellar=[0,0,0];rmax1_het_lamellar=[];rmax2_het_lamellar=[]
coef_het_lamellar2=[0,0,0];rc1_het_lamellar=[];rc2_het_lamellar=[]
for Ti in T:
    coef_het_lamellar=[0,0,0]
    coef_het_lamellar2=[0,0,0]
    a1,a2,a3=V_het(h,γt,γsl,L,Tm,Ti,theta)
    coef_het_lamellar[0]+=a1;coef_het_lamellar[1]+=a2;coef_het_lamellar[2]+=a3
    a1,a2,a3=S_sl(h,γt,γsl,L,Tm,Ti)
    coef_het_lamellar[0]+=a1;coef_het_lamellar[1]+=a2;coef_het_lamellar[2]+=a3
    a1,a2,a3=S_lam(h,γt,γsl,L,Tm,Ti)
    coef_het_lamellar[0]+=a1;coef_het_lamellar[1]+=a2;coef_het_lamellar[2]+=a3
    coef_het_lamellar[2]+=-52*kB*(Tm+Ti)
    rmax1_het_lamellar.append(np.roots(coef_het_lamellar)[0]*1e9)
    rmax2_het_lamellar.append(np.roots(coef_het_lamellar)[1]*1e9)

    b1,b2=dV_het(h,γt,γsl,L,Tm,Ti,theta)
    coef_het_lamellar2[0]+=b1;coef_het_lamellar2[1]+=b2;
    b1,b2=dS_sl(h,γt,γsl,L,Tm,Ti)
    coef_het_lamellar2[0]+=b1;coef_het_lamellar2[1]+=b2;
    b1,b2=dS_lam(h,γt,γsl,L,Tm,Ti)
    coef_het_lamellar2[0]+=b1;coef_het_lamellar2[1]+=b2;
    rc1_het_lamellar.append(np.roots(coef_het_lamellar2)[0]*1e9)
```

```
    rc2_het_lamellar.append(np.roots(coef_het_lamellar2)[1]*1e9)



coef_het_5fold=[0,0,0];rmax1_het_5fold=[];rmax2_het_5fold=[]
coef_het_5fold2=[0,0,0];rc1_het_5fold=[];rc2_het_5fold=[]
for Ti in T:
    coef_het_5fold=[0,0,0]
    coef_het_5fold2=[0,0,0]
    a1,a2,a3=V_het(h,γt,γsl,L,Tm,Ti,theta)
    coef_het_5fold[0]+=a1;coef_het_5fold[1]+=a2;coef_het_5fold[2]+=a3
    a1,a2,a3=S_sl(h,γt,γsl,L,Tm,Ti)
    coef_het_5fold[0]+=a1;coef_het_5fold[1]+=a2;coef_het_5fold[2]+=a3
    a1,a2,a3=S_5fold_het(h,γt,γsl,L,Tm,Ti,theta)
    coef_het_5fold[0]+=a1;coef_het_5fold[1]+=a2;coef_het_5fold[2]+=a3
    coef_het_5fold[2]+=-52*kB*(Tm+Ti)
    rmax1_het_5fold.append(np.roots(coef_het_5fold)[0]*1e9)
    rmax2_het_5fold.append(np.roots(coef_het_5fold)[1]*1e9)
    b1,b2=dV_het(h,γt,γsl,L,Tm,Ti,theta)
    coef_het_5fold2[0]+=b1;coef_het_5fold2[1]+=b2;
    b1,b2=dS_sl(h,γt,γsl,L,Tm,Ti)
    coef_het_5fold2[0]+=b1;coef_het_5fold2[1]+=b2;
    b1,b2=dS_5fold_het(h,γt,γsl,L,Tm,Ti,theta)
    coef_het_5fold2[0]+=b1;coef_het_5fold2[1]+=b2;
    rc1_het_5fold.append(np.roots(coef_het_5fold2)[0]*1e9)
    rc2_het_5fold.append(np.roots(coef_het_5fold2)[1]*1e9)

#plt.plot(Tm-T,rmax1_het_perfect,label="rmax1_het_perfect")
axes2[1,0].plot(-T,rmax2_het_perfect,label="rmax2_het_perfect")
axes2[1,0].plot(-T,rc1_het_perfect,label="rc1_het_perfect")
#plt.plot(Tm-T,rc2_het_perfect,label="rc2_het_perfect")
#axes2[1,0].legend(frameon=False)
axes2[1,0].set_ylabel("r (nm)",fontstyle="italic")
axes2[1,0].set_xlabel("ΔT (K)",fontstyle="italic")
axes2[1,0].set_title("Nucleation on $Al_2O_3$",fontsize=10)
axes2[1,0].text(320,17,"perfect")


#plt.plot(Tm-T,rmax1_het_lamellar,label="rmax1_het_lamellar")
axes2[1,1].plot(-T,rmax2_het_lamellar,label="rmax2_het_lamellar")
axes2[1,1].plot(-T,rc1_het_lamellar,label="rc1_het_lamellar")
#plt.plot(Tm-T,rc2_het_lamellar,label="rc2_het_lamellar")
#axes2[1,1].legend(frameon=False)
axes2[1,1].set_xlabel("ΔT (K)",fontstyle="italic")
axes2[1,1].set_title("Nucleation on $Al_2O_3$",fontsize=10)
axes2[1,1].text(220,40,"Lamellar TB")
axins=axes2[1,1].inset_axes([0.48,0.2,0.4,0.3])
tickes3=[300,500]
axins.set_xticks(tickes3)
axins.plot(-T,rmax2_het_lamellar,label="$r_{max}$")
axins.plot(-T,rc1_het_lamellar,label="$r_{c}$")
```

```
#plt.plot(Tm-T,rc2_hom_perfect,label="rc2_hom_perfect")
#axes2[0,0].set_ylabel("r (nm)",fontstyle="italic")
axins.set_xlim([300,500])
axins.set_ylim([0,5])
axins.tick_params(axis='both', which='major', labelsize=9)
axes2[1,1].indicate_inset_zoom(axins)


#plt.plot(Tm-T,rmax1_het_5fold,label="rmax1_het_5fold")
axes2[1,2].plot(-T,rmax2_het_5fold,label="rmax2_het_5fold")
axes2[1,2].plot(-T,rc1_het_5fold,label="rc1_het_5fold")
#plt.plot(Tm-T,rc2_het_5fold,label="rc2_het_5fold")
#axes2[1,2].legend(frameon=False)
axes2[1,2].set_xlabel("ΔT (K)",fontstyle="italic")
axes2[1,2].set_title("Nucleation on $Al_2O_3$",fontsize=10)
axes2[1,2].text(260,23,"5-fold TB")
plt.subplots_adjust(wspace=0.4,hspace=0.3)
plt.savefig('r.png', dpi=1200,bbox_inches='tight',pad_inches = 0)
plt.show()
```

7. Python code for fcc phase growth in Regions A and B

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
def bootstrap_replicate_1d(data,target):
    """Generate bootstrap replicate of 1D data."""
    bs_sample = np.random.choice(data, len(data))
#    print(len(data))
    n_target=0;n=0;
    for i in range(len(bs_sample)):
        n+=1
        if (bs_sample[i]==target):
            n_target+=1
#            print(n_target)

    return n_target/n
    #return bs_sample.groupby(["CNA"]).mean()
def draw_bs_reps(data,target, size=1):
    """Draw bootstrap replicates."""

    # Initialize array of replicates: bs_replicates
    bs_replicates = np.empty(size)

    # Generate replicates
    for i in range(size):
        bs_replicates[i] = bootstrap_replicate_1d(data["CNA"],target)

    return bs_replicates
time_A=[]
fcc_A_mean=[]
fcc_A_std=[]
```

```
timestep=0.0005
frequency=500
size=200
n_step1=398
step=4

for i in range(0,n_step1+1,16):
   name="CNA."+str(i)
   data_array = np.loadtxt('data_1_A/'+name,skiprows=2,delimiter=" ")
#   print(data_array.shape)
   data = pd.DataFrame(data_array,columns=["number", "CNA"])
#   print(data.head())

   bs_replicates_fcc_A = draw_bs_reps(data,1.0,size)
   n_fcc_A_mean= np.mean(bs_replicates_fcc_A)
   n_fcc_A_std= np.std(bs_replicates_fcc_A)

   time_A.append(i*frequency*timestep)
   fcc_A_mean.append(n_fcc_A_mean)
   fcc_A_std.append(n_fcc_A_std)
   print(i,n_fcc_A_mean, n_fcc_A_std)
laststep_1=np.floor(n_step1/step)*4
#for j in range(0,n_step2+1,160):
n_step2=712
for j in range(0,n_step2+1,16):
   name="CNA."+str(j)
   data_array = np.loadtxt('data_2_A/'+name,skiprows=2,delimiter=" ")
#   print(data_array.shape)
   data = pd.DataFrame(data_array,columns=["number", "CNA"])
#   print(data.head())

   bs_replicates_fcc_A = draw_bs_reps(data,1.0,size)
   n_fcc_A_mean= np.mean(bs_replicates_fcc_A)
   n_fcc_A_std= np.std(bs_replicates_fcc_A)

   time_A.append((j+laststep_1)*frequency*timestep)
   fcc_A_mean.append(n_fcc_A_mean)
   fcc_A_std.append(n_fcc_A_std)
   print(laststep_1+j,n_fcc_A_mean, n_fcc_A_std)

time_B=[]
fcc_B_mean=[]
fcc_B_std=[]

for i in range(0,n_step1+1,16):
   name="CNA."+str(i)
   data_Brray = np.loadtxt('data_1_B/'+name,skiprows=2,delimiter=" ")
#   print(data_Brray.shape)
   data = pd.DataFrame(data_Brray,columns=["number", "CNA"])
#   print(data.head())
```

```
    bs_replicates_fcc_B = draw_bs_reps(data,1.0,size)
    n_fcc_B_mean= np.mean(bs_replicates_fcc_B)
    n_fcc_B_std= np.std(bs_replicates_fcc_B)

    time_B.append(i*frequency*timestep)
    fcc_B_mean.append(n_fcc_B_mean)
    fcc_B_std.append(n_fcc_B_std)
    print(i,n_fcc_B_mean, n_fcc_B_std)

laststep_1=np.floor(n_step1/step)*4
#for j in range(0,n_step2+1,160):

for j in range(0,n_step2+1,16):
    name="CNA."+str(j)
    data_Brray = np.loadtxt('data_2_B/'+name,skiprows=2,delimiter=" ")
#    print(data_Brray.shape)
    data = pd.DataFrame(data_Brray,columns=["number", "CNA"])
#    print(data.head())

    bs_replicates_fcc_B = draw_bs_reps(data,1.0,size)
    n_fcc_B_mean= np.mean(bs_replicates_fcc_B)
    n_fcc_B_std= np.std(bs_replicates_fcc_B)

    time_B.append((j+laststep_1)*frequency*timestep)
    fcc_B_mean.append(n_fcc_B_mean)
    fcc_B_std.append(n_fcc_B_std)
    print(laststep_1+j,n_fcc_B_mean, n_fcc_B_std)
plt.plot(time_A, fcc_A_mean, color='blue',label="Region A")
plt.fill_between(time_A,                                    np.array(fcc_A_mean)-np.array(fcc_A_std),
np.array(fcc_A_mean)+np.array(fcc_A_std),alpha=0.3, color="blue")

plt.plot(time_B, fcc_B_mean, color='red',label="Region B")
plt.fill_between(time_B,                                    np.array(fcc_B_mean)-np.array(fcc_B_std),
np.array(fcc_B_mean)+np.array(fcc_B_std),alpha=0.3, color="red")
plt.xlabel("time (ps)",size=15)
plt.xticks(fontsize=15)
plt.ylabel("fcc phase fraction",size=15)
plt.yticks(fontsize=15)
plt.title("Al-OO",size=15)
plt.legend(fontsize=15)
plt.savefig("image/PF_OO_AB.jpeg",bbox_inches='tight',dpi=1200) #facecolor='black',
```


[1] J. Bragard, A. Karma, Y.H. Lee, M. Plapp, Linking Phase-Field and Atomistic Simulations to Model Dendritic Solidification in Highly Undercooled Melts, Interface Science 10(2) (2002) 121-136.
[2] J.J. Hoyt, M. Asta, A. Karma, Method for Computing the Anisotropy of the Solid-Liquid Interfacial Free Energy, Physical Review Letters 86(24) (2001) 5530-5533.
[3] J. Hoyt, M. Asta, A. Karma, Method for computing the anisotropy of the solid-liquid interfacial free energy, Physical review letters 86(24) (2001) 5530.
[4] J.J. Hoyt, M. Asta, Atomistic computation of liquid diffusivity, solid-liquid interfacial free energy, and kinetic coefficient in Au and Ag, Physical Review B 65(21) (2002) 214106.