

Московский государственный университет имени М.В. Ломоносова

Факультет вычислительной математики и кибернетики

Кафедра информационной безопасности

Мирпулатов Исломбек Пулат-угли

Определение потенциалов межатомного взаимодействия системы Ag - Cr

КУРСОВАЯ РАБОТА

**Преподаватель:**

к.ф-м.н.

Д.И. Бажанов

Москва, 2022

Оглавление

1. Описание задачи 3

2. Использованные формулы для расчета 4

3. Реализация 8

4. Алгоритм тернарного поиска 9

5. Метод оптимизации Нелдера Мида 10

6. Ускорение кода 12

6.1 Ускорение кода на чистом Python 12

6.2 Ускорение с помощью Numba 12

7. Результаты работы 14

8. Вывод 15

9. Листинг 16

atom.py: 16

crystal\_structure.py 17

utils.py 18

1. Описание задачи

В рамках задачи требуется определить потенциалы межатомного взаимодействия

(A-A, A-B, B-B) для системы A/B, где A - Cr (Хром), B - Ag (Серебро).

- полная энергия системы

- энергии притяжения

- энергия отталкивания

- расстояние между атомами i (Сорт А) и j (Сорт B)

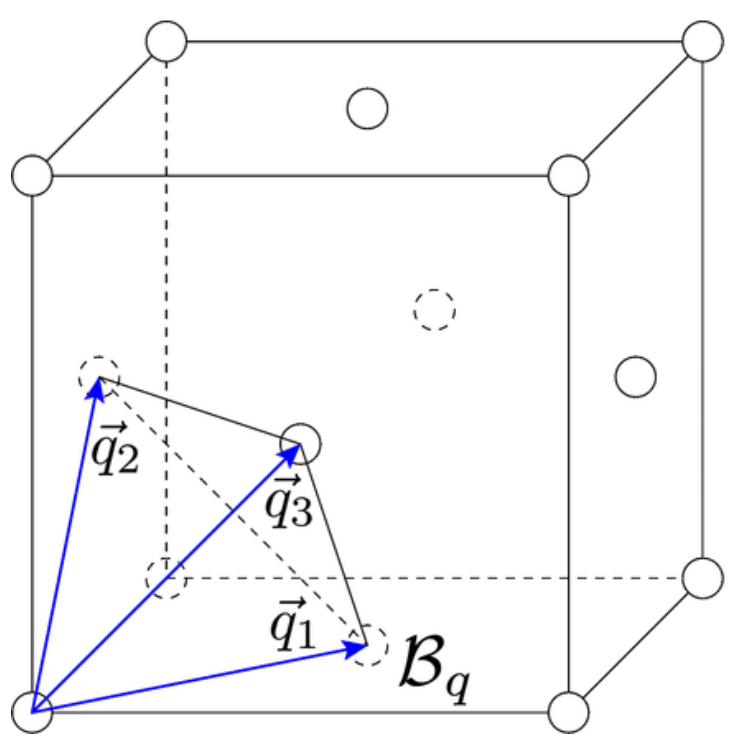
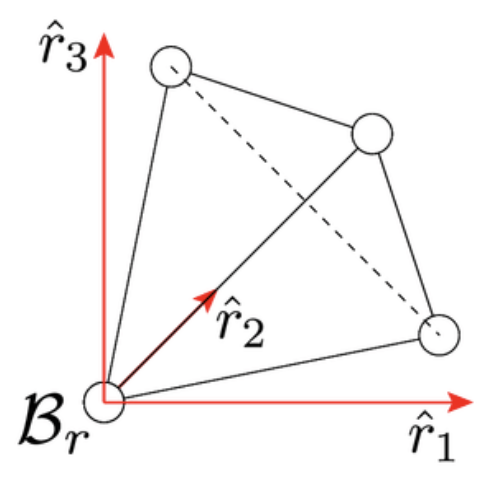
Параметры потенциалов: (для A-A, A-B и B-B взаимодействий 3 набора по 6 параметров) необходимо определить c помощью алгоритма минимизации функции ошибки относительно известных табличных параметров.

Все расчёты полных энергий E проводятся для кристаллической решётки размером 3 × 3 × 3 в единицах элементарной ячейки ГЦК структуры. Все расчёты статические, проводятся без релаксации атомных позиций. Атомы располагаются в узлах “идеальной” ГЦК решётки.

Полученное решение оптимизировать с помощью технологии распараллеливания.

2. Использованные формулы для расчета

Далее изображены ГЦК решетка Бравэ и элементарная ячейка из которой образуется кристаллическая решетка:



Для оптимизации параметров в качестве целевой функции минимизации был выбран следующий тип функции:

, где

TrueValue - основные значения кристаллической решетки, значения взятые из справочника

Value - значения, посчитанные с текущим набором параметров потенциалов

Для описания взаимодействия атомов сорта В-В в целевом функционале необходимо использовать следующие значения кристаллической решетки:

* a - параметр решетки
* - когезионная энергия
* B - модуль всестороннего растяжения, сжатия
* , , - константы упругости

В результате функция ошибки принимает следующий вид:

Для описания взаимодействия атомов сорта A-В в целевом функционале необходимо использовать следующие значения кристаллической решетки:

* - энергия растворимости

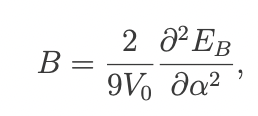
В результате функция ошибки принимает следующий вид:

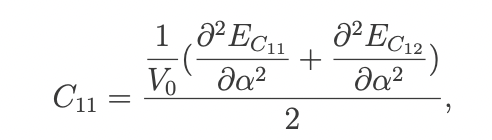
Для описания взаимодействия атомов сорта A-A в целевом функционале необходимо использовать следующие значения кристаллической решетки:

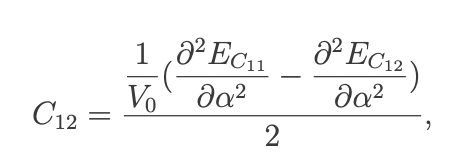
* - энергия связи димера в поверхностном слое
* - энергия связи димера на поверхности

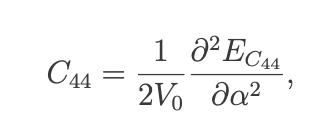
В результате функция ошибки принимает следующий вид:

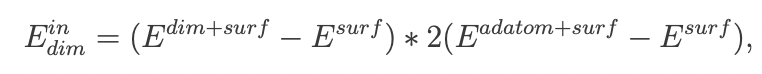
Сопутствующие формулы для предыдущих:

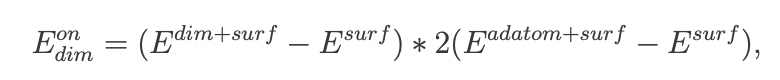


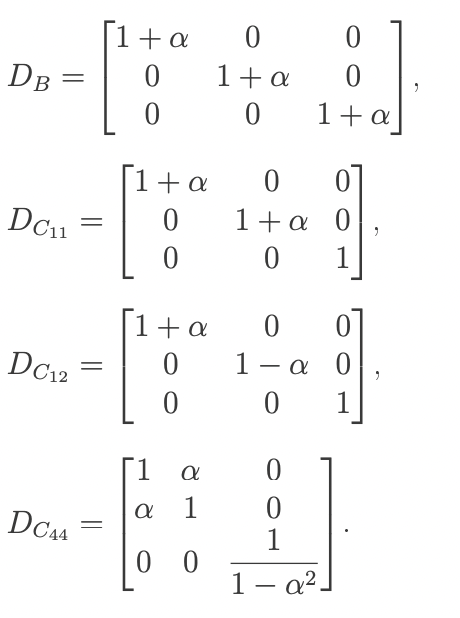












* - растояние для ближайшего соседа
* a - параметр решетки
* - радиус взаимодействия атомов
* - равновесный объем
* - количество ячеек x, y, z
* E - полная энергия
* - когезионная энергия
* n - количество атомов
* B - модуль всестороннего растяжения (сжатия). Равномерное растяжение (сжатие) происходит по трем эквивалентным направлениям кубической решетки
* - константа упругости. Равномерное растяжение в данном случае происходит в плоскости xy
* - константа упругости. Растяжение в данном случае происходит вдоль оси x и сжатие вдоль оси y
* - константа упругости
* - энергия связи димера в поверхностном слое
* - энергия связи димера на поверхности
* Матрицы D - матрицы деформации узлов, в которых располагаются атомы

3. Реализация

1. Реализован подсчет энергии взаимодействия между атомами учитывающих их тип.

2. Найдено значение для B-B, исходя из параметров потенциала взятых из таблицы. Значение найдено при помощи алгоритма тернарнарного поиска. (Описание в главе 4)

3. Реализован подсчет основных характеристик и параметров решетки.

4. Реализован алгоритм оптимизации Неледера Мида. (Описание в главе 5)

5. Найдены параметры для двух типов взаимодействий: В-В, А-В.

6. Построены графики зависимости полной энергии от расстояния.

4. Алгоритм тернарного поиска

Возьмём любые две точки  в этом отрезке: . Посчитаем значения функции  и . Дальше у нас получается три варианта:

* Если окажется, что , то искомый максимум не может находиться в левой части, т.е. в части . В этом легко убедиться: если в левой точке функция меньше, чем в правой, то либо эти две точки находятся в области "подъёма" функции, либо только левая точка находится там. В любом случае, это означает, что максимум дальше имеет смысл искать только в отрезке .
* Если, наоборот, , то ситуация аналогична предыдущей с точностью до симметрии. Теперь искомый максимум не может находиться в правой части, т.е. в части , поэтому переходим к отрезку .
* Если , то либо обе эти точки находятся в области максимума, либо левая точка находится в области возрастания, а правая — в области убывания (здесь существенно используется то, что возрастание/убывание строгие). Таким образом, в дальнейшем поиск имеет смысл производить в отрезке , но (в целях упрощения кода) этот случай можно отнести к любому из двух предыдущих.

Таким образом, по результату сравнения значений функции в двух внутренних точках мы вместо текущего отрезка поиска  находим новый отрезок . Повторим теперь все действия для этого нового отрезка, снова получим новый, строго меньший, отрезок, и т.д.

Рано или поздно длина отрезка станет маленькой, меньшей заранее определённой константы-точности, и процесс можно останавливать. Этот метод численный, поэтому после остановки алгоритма можно приближённо считать, что во всех точках отрезка  достигается максимум; в качестве ответа можно взять, например, точку .

5. Метод оптимизации Нелдера Мида

Метод Нелдера — Мида — метод оптимизации (поиска минимума) функции от нескольких переменных. Простой и в тоже время эффективный метод, позволяющий оптимизировать функции без использования градиентов. В рамках моей курсовой работы использовалась функция optimize из модуля scipy.optimize популярной библиотеки для языка python, которая используется для математических расчетов. []

Алгоритм заключается в формировании симплекса (**simplex**) и последующего его деформирования в направлении минимума, посредством трех операций:

1) **Отражение** (reflection);

2) **Растяжение** (expansion);

3) **Сжатие** (contract);

Симплекс представляет из себя геометрическую фигуру, являющуюся n — мерным обобщением треугольника. Для одномерного пространства — это отрезок, для двумерного — треугольник. Таким образом n — мерный симплекс имеет n + 1 вершину.

1) Пусть  функция , которую необходимо оптимизировать. На первом шаге выбираем три случайные точки (об этом чуть позже) и формируем симплекс (треугольник). Вычисляем значение функции в каждой точке: .

Сортируем точки по значениям функции   в этих точках, таким образом получаем двойное неравенство:

Мы ищем минимум функции, а следовательно, на данном шаге лучшей будет та точка, в которой значение функции минимально. Для удобства переобозначим точки следующим образом:

b = , g = , w = , где best, good, worst — соответственно.

2) На следующем шаге находим середину отрезка, точками которого являются g и b. Т.к. координаты середины отрезка равны полусумме координат его концов, получаем:

В более общем виде можно записать:

3) Применяем операцию отражения:

Находим точку , следующим образом:

Т.е. фактически отражаем точку относительно . В качестве коэффициента берут как правило 1. Проверяем нашу точку: если , то это хорошая точка. А теперь попробуем расстояние увеличить в 2 раза, вдруг нам повезет и мы найдем точку еще лучше.

4) Применяем операцию растяжения:

Находим точку  следующим образом:

В качестве γ принимаем γ = 2, т.е. расстояние увеличиваем в 2 раза.

Проверяем точку :

Если , то нам повезло и мы нашли точку лучше, чем есть на данный момент, если бы этого не произошло, мы бы остановились на точке .

Далее заменяем точку на .

5) Если же нам совсем не повезло и мы не нашли хороших точек, пробуем операцию сжатия.

Как следует из названия операции мы будем уменьшать наш отрезок и искать хорошие точки внутри треугольника.

Пробуем найти хорошую точку :

Коэффициент β принимаем равным 0.5, т.е. точка  на середине отрезка .

Алгоритм заканчивается, когда:

1) Было выполнено необходимое количество итераций.

2) Площадь симплекса достигла определенной величины.

3) Текущее лучшее решение достигло необходимой точности.

6. Ускорение кода

**6.1 Ускорение кода на чистом Python**

Для полного ускорения кода на Python необходимо сначала использовать все доступные способы ускорения на чистом Python, без использования дополнительных библиотек. К таковым относится:

* оптимизация вычислений
* уменьшение количества копирования объектов
* переход на константные объекты

Такие изменения позволяют оптимизировать код и значительно ускорить его исполнение, а затем применять дополнительные библиотеки, которые позволяют ещё сильнее ускорить код

**6.2 Ускорение с помощью Numba**

В данной части будут рассмотрен способ ускорения вычислений с использованием Numba. Для ускорения каждая функция была переписана в соответствии со строгими правилами использования Numba в Python. Все списки были заменены на кортежи, убраны классы сферы и луча, что понизило читаемость кода, но при этом позволило ему запускаться без ошибок. Классы в Numba имеют только экспериментальную поддержку, и основным их ограничением является поддержка только типа float32 для массивов numpy, в связи с этим возникали забавные ошибки из-за недостаточной точности вычисления.

**Преимущества:**

В качестве преимуществ Numba можно выделить высокую производительность вычислений, при использовании кода на python.

**Недостатки:**

Numba имеет множество ограничений, вот некоторые из которых пришлось столкнуться при адаптации кода программы:

* ограничения на использование классов,
* ограничения на использования стандартных Python функций (например в print нельзя использовать именные аргументы),
* ограничение на использование списков,
* ограничение на использование типизации аргументов функции (например если функция np.dot(a, b) использовалась аргументами типа float64, её нельзя будет использовать с аргументами типа float32). Также в Numba имеются параметры у @jit, как parallel=True, которые на сложных функциях наоборот замедлят вычисления.

7. Результаты работы

Расчеты проводились для системы A/B(001): Cr/Ag(001).

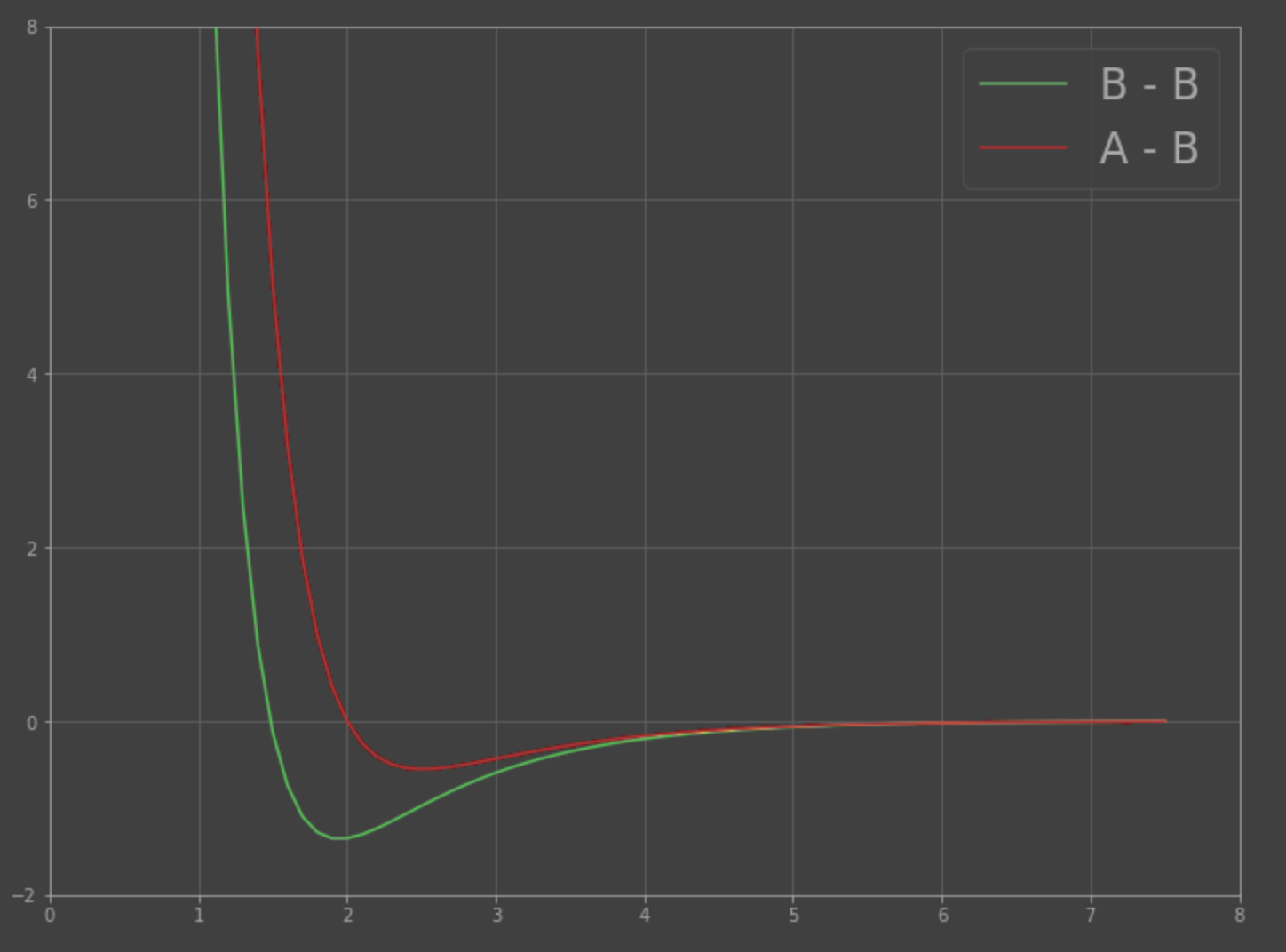
Параметры потенциалов для взаимодействия B - B, A - B оптимизировались с помощью алгоритма минимизации нулевого порядка Нелдера-Мида.

Для взаимодействия Ag - Ag получены следующие значения:

|  |  |  |
| --- | --- | --- |
| Переменная | Табличное значение | Полученное значение |
|  | 4.085 | 4.0857 |
|  | -2.960 | -2.9603 |
| B | 1.08 | 1.0836 |
|  | 1.32 | 1.3173 |
|  | 0.97 | 0.9668 |
|  | 0.51 | 0.5067 |

Для взаимодействия Cr - Ag получены следующие значения

|  |  |  |
| --- | --- | --- |
| Переменная | Табличное значение | Полученное значение |
|  | 0.881 | 0.88098 |

График зависимости полной энергии от расстояния:

8. Вывод

Выполнение данной курсовой работы помогло разобраться на практике с темой которой был посвящен курс “Параллельной обработки данных в естественнонаучных исследованиях”.

В роли основной задачи в рамках которой было изучены принципы параллельной обработки данных выступала задача определения потенциалов межатомного взаимодействия для системы вида A - B (001) Cr - Ag (001).

Были реализованы все формулы для подсчета характеристик, при условии достижения оптимальной скорости. А также были реализованы два алгоритма: тернарного поиска и алгоритма Нелдера-Мида. В результате были получены положительные результаты для систем A - B и B - B.

9. Листинг

**atom.py:**

import numpy as np

class Atom:

def \_\_init\_\_(self, \_type="Ag", x=1, y=1, z=1):

self.type = \_type

self.x = x

self.y = y

self.z = z

def \_\_add\_\_(self, other):

if isinstance(other, Atom):

return Atom(self.type, self.x + other.x, self.y + other.y, self.z + other.z)

def \_\_sub\_\_(self, other):

if isinstance(other, Atom):

return Atom(self.type, self.x - other.x, self.y - other.y, self.z - other.z)

def \_\_eq\_\_(self, other):

if isinstance(other, Atom):

return self.x == other.x and self.y == other.y and self.z == other.z

return False

def \_\_str\_\_(self):

return f"X: {self.x}\tY: {self.y}\tZ: {self.z}”

def transform(self, tansformation):

return Atom(

self.type,

self.x \* tansformation[0] + self.y \* tansformation[1] + self.z \* tansformation[2],

self.x \* tansformation[3] + self.y \* tansformation[4] + self.z \* tansformation[5],

self.x \* tansformation[6] + self.y \* tansformation[7] + self.z \* tansformation[8]

)

def distance(self):

return np.sqrt(self.x \* self.x + self.y \* self.y + self.z \* self.z)

**crystal\_structure.py**

import numpy as np

from POD.atom import Atom

class CrystalStructure:

def \_\_init\_\_(self, dim):

self.dim = dim

self.fcc = []

def face\_centred\_cubic(self, \_type):

for i in range(self.dim):

for j in range(self.dim):

for k in range(self.dim):

self.fcc.append(Atom(\_type, i, j, k))

self.fcc.append(Atom(\_type, i, j + 0.5, k + 0.5))

self.fcc.append(Atom(\_type, i + 0.5, j + 0.5, k))

self.fcc.append(Atom(\_type, i + 0.5, j, k + 0.5))

def calculate\_energy(self, trans, a1, a0, ksi, p, q, r0, o=None):

if o is None:

o = []

full\_energy = 0.

t = Atom().transform(trans)

cutoff = 1.7 \* max(t.x, max(t.y, t.z))

for X1 in self.fcc:

energy\_r = 0.

energy\_b = 0.

for X2 in self.fcc:

for dx in range(-1, 2):

for dy in range(-1, 2):

for dz in range(-1, 2):

if X1 != X2 or dx != 0 or dy != 0 or dz != 0:

rij = ((X2 + Atom('', dx \* self.dim, dy \* self.dim, dz \* self.dim)).transform(trans) -

X1.transform(trans)).distance()

if rij < cutoff:

if X1.type == "Ag" and X2.type == "Ag":

energy\_r += ((a1 / r0) \* (rij - r0) + a0) \* np.exp(-1 \* p \* (rij / r0 - 1))

energy\_b += ksi \* ksi \* np.exp(-2 \* q \* (rij / r0 - 1))

elif X1.type == "Cr" and X2.type == "Ag" or X1.type == "Ag" and X2.type == "Cr":

energy\_r += ((o[0] / o[5]) \* (rij - o[5]) + o[1]) \*\

np.exp(-1 \* o[3] \* (rij / o[5] - 1))

energy\_b += o[2] \* o[2] \* np.exp(-2 \* o[4] \* (rij / o[5] - 1))

energy\_b = -1 \* np.sqrt(energy\_b)

full\_energy += energy\_r + energy\_b

return full\_energy

def get\_size(self):

return len(self.fcc)

def change\_atom(self, t):

self.fcc[13].type = t

**utils.py**

import numpy as np

from POD.crystal\_structure import CrystalStructure

CONST = 0.8018993929636421

ALPHA = 10e-4

a0\_true = 4.085

e\_coh\_true = -2.960

b\_true = 1.08

c11\_true = 1.32

c12\_true = 0.97

c44\_true = 0.51

b\_pos = np.array([1 + ALPHA, 0, 0, 0, 1 + ALPHA, 0, 0, 0, 1 + ALPHA])

b\_neg = np.array([1 - ALPHA, 0, 0, 0, 1 - ALPHA, 0, 0, 0, 1 - ALPHA])

c11\_pos = np.array([1 + ALPHA, 0, 0, 0, 1 + ALPHA, 0, 0, 0, 1])

c11\_neg = np.array([1 - ALPHA, 0, 0, 0, 1 - ALPHA, 0, 0, 0, 1])

c12\_pos = np.array([1 + ALPHA, 0, 0, 0, 1 - ALPHA, 0, 0, 0, 1])

c12\_neg = np.array([1 - ALPHA, 0, 0, 0, 1 + ALPHA, 0, 0, 0, 1])

c44\_pos = np.array([1, ALPHA, 0, ALPHA, 1, 0, 0, 0, 1 / (1 - ALPHA \* ALPHA)])

c44\_neg = np.array([1, -ALPHA, 0, -ALPHA, 1, 0, 0, 0, 1 / (1 - ALPHA \* ALPHA)])

identity\_matrix = np.array([1., 0, 0, 0, 1., 0, 0, 0, 1.])

def get\_derivative(dim, positive, negative, energy, a1, a0, ksi, p, q, r0):

cs = CrystalStructure(dim)

cs.face\_centred\_cubic("Ag")

coh\_energy = (

cs.calculate\_energy(positive, a1, a0, ksi, p, q, r0)

+ cs.calculate\_energy(negative, a1, a0, ksi, p, q, r0)

) / cs.get\_size()

return (coh\_energy - 2. \* energy) / (ALPHA \* ALPHA)

def calculate\_characteristics(m0, dim, energy, a1, a0, ksi, p, q, r0):

v0 = m0 \* m0 \* m0 / 4

sq\_derivative\_energy\_b = get\_derivative(dim, m0 \* b\_pos, m0 \* b\_neg, energy, a1, a0, ksi, p, q, r0)

sq\_derivative\_energy\_c11 = get\_derivative(dim, m0 \* c11\_pos, m0 \* c11\_neg, energy, a1, a0, ksi, p, q, r0)

sq\_derivative\_energy\_c12 = get\_derivative(dim, m0 \* c12\_pos, m0 \* c12\_neg, energy, a1, a0, ksi, p, q, r0)

sq\_derivative\_energy\_c44 = get\_derivative(dim, m0 \* c44\_pos, m0 \* c44\_neg, energy, a1, a0, ksi, p, q, r0)

b = 2. \* sq\_derivative\_energy\_b \* CONST / (9. \* v0)

c11 = (sq\_derivative\_energy\_c11 \* CONST + sq\_derivative\_energy\_c12 \* CONST) / (2. \* v0)

c12 = (sq\_derivative\_energy\_c11 \* CONST - sq\_derivative\_energy\_c12 \* CONST) / (2. \* v0)

c44 = (sq\_derivative\_energy\_c44 \* CONST) / (2. \* v0)

return b, c11, c12, c44

def calculate\_energy\_sol(e\_coh\_a, e\_coh\_b, e\_b, e\_ab):

return e\_ab - e\_b - e\_coh\_a + e\_coh\_b

def error\_bb(w, dim, a0, e\_coh):

b, c11, c12, c44 = calculate\_characteristics(a0, dim, e\_coh, w[0], w[1], w[2], w[3], w[4], w[5])

err = (a0 - a0\_true) \* (a0 - a0\_true) / (a0\_true \* a0\_true) + \

(e\_coh - e\_coh\_true) \* (e\_coh - e\_coh\_true) / (e\_coh\_true \* e\_coh\_true) + \

(b - b\_true) \* (b - b\_true) / (b\_true \* b\_true) + \

(c11 - c11\_true) \* (c11 - c11\_true) / (c11\_true \* c11\_true) + \

(c12 - c12\_true) \* (c12 - c12\_true) / (c12\_true \* c12\_true) + \

(c44 - c44\_true) \* (c44 - c44\_true) / (c44\_true \* c44\_true)

err = np.sqrt(err / 6)

return err

def error\_ab(w, dim, m0, A1, A0, KSI, P, Q, R0, e\_coh\_b, e\_b):

e\_coh\_a = 4.10

e\_true = 0.881

A\_B = CrystalStructure(dim)

A\_B.face\_centred\_cubic("Ag")

A\_B.change\_atom("Cr")

e\_ab = A\_B.calculate\_energy(m0 \* identity\_matrix, A1, A0, KSI, P, Q, R0, w)

e\_sol = calculate\_energy\_sol(e\_coh\_a, e\_coh\_b, e\_b, e\_ab)

err = (e\_sol - e\_true) \* (e\_sol - e\_true) / (e\_true \* e\_true)

return err

def nrg(rij, o):

energy\_r = ((o[0] / o[5]) \* (rij - o[5]) + o[1]) \* np.exp(-1 \* o[3] \* (rij / o[5] - 1))

energy\_b = o[2] \* o[2] \* np.exp(-2 \* o[4] \* (rij / o[5] - 1))

energy\_b = -1 \* np.sqrt(energy\_b)

return energy\_r + energy\_b