Homework 2: Monte Carlo simulation of sodium incorporation in ferropericlase (Mg,Fe)O in the lower mantle under high pressure conditions

Initial implementation
By: Luisa Chavarria

1. **Project structure and documentation**

Provide a diagram or textual description of your code structure below.

```
cmse802_project_hw2/
├── src/
│   ├── lattice.py
│   ├── energy_calculation.py
│   ├── monte_carlo.py
│   ├── simulations.py
│
├── notebooks/
│   ├── hw2_data_analysis.ipynb
│
├── data/
│   └── constants.py
│
├── tests/
│   ├── test_lattice_creation.py
│
│
├── docs/
│   ├── papers
│   ├── README.md
│
├── results/
│   ├── report/
│
├── README.md
└── requirements.txt
```

Documentation strategy:

Scripts:

* constants.py - Includes some examples of constants that will be used in the model
* lattice.py - Includes the code to create the 3D cell for MgO or FeO
* energy_calculation_2.py - Includes the code for the calculation of the energy based on the Couloumb's law
* monte_carlo.py - Initial simulation using the Monte Carlo model
* simulations.py - This file contains the script that run multiple monte carlo simulations

Notebooks:

* Hw2_DataAnalysis - This Jupyter notebook contains the data analysis and visualizations of the functions in modules lattice.py and energy_calculation_2.py

2. Data/Problem Understanding

Algorithm foundation code:

The notebook Hw2_DataAnalysis contains the algorithm foundation code as well as the files monte_carlo.py where the initial simulation using the Monte Carlo model is provided.

Implementation Plan:

The implementation fo the algorithm require the following steps:

1) Data acquisition and input: The data can be obtained from material sciences databases like Materials project (https://next-gen.materialsproject.org/) that has information of Density functional theory (DFT) calculations about formation energies and elastic properties. For example, for MgO the material ID mp-1265 has information about the predicted formation energy of -2.995 eV/atom and for FeO the mp-18905 has a predicted formation energy of -1.071 eV/atom. The Open Quantum Materials Database (OQMD) is a DFT database where I can also find data about the structures (https://oqmd.org/)

2) Algorithm implementation:

The project will implement the Monte Carlo (MC) algorithm that consists on a sampling process where the next sample depends on the previous. For molecular simulations, the Metropolis Monte Carlo (MMC) method will be used to accept or reject which random sample or configuration is more suitable in the model. For the modeling, I'm going to follow the next steps:

* Define an initial (Mg,Fe)O lattice using a supercell approach

* Introduce Na substitutions at Mg or Fe sites

* Assign energies to different configurations based on defect formation energies.

* Accept or reject new configurations using the Boltzmann distribution:

In the Boltzmann distribution applying the MMC method, given an energy difference : Delta E = E_{new}-E_{old}

If Delta E < 0 (new state is lower in energy), accept the move

If Delta E > 0 (new state is higher in energy), accept with probability:

$$P_{\text{accept}} = \min\left(1, e^{-\Delta E/k_B T}\right)$$

3) Post-processing:

* Analyze results, aggregate outputs, and ensure results are interpretable for visualization or reporting
* Format data for output

4) Evaluation and iteration

* Evaluate the model and compare the model with experimental data in ferropericlase or in other minerals
* Adjust parameters of the model until obtaining values that can be contrasted with experimental data

Dependencies:

Libraries/packages:

* Numerical processing: numpy, scipy, math
* Data visualization: matplotlib, seaborn, plotly
* File handling: pandas
* Optimization: cvxpy, scipy.optimize

3. Unit Testing Framework

The modeling will require tests for the following modules:

1. Lattice
2. Energy_calculation
3. Monte Carlo simulation step
4. Simulation

However, I'm going to show some test only for the first or second module.

Test Case 1. Lattice

Test objective: ensure that the lattice initialize with the right number of ions inside the structure

Inputs:

Materials: MgO

Lattice size: 5

Expected Output:

I expect a 3D lattice with a size 5x5x5 with alternating 0 for Mg or Fe and 1 for O.

Test steps:

1. Call lattice.py (material="MgO", L=5).
2. Check the shape of the lattice: it should be (5, 5, 5).
3. Verify that alternating ions (Mg/O) are placed correctly, with 0 (Mg) and 1 (O) alternating in a 3D checkerboard pattern.

Edge Cases:

Material set to a value that isn't supported (e.g., material="NaCl"), expecting an exception or error message.

Testing Implementation

The script test_lattice_creation.py can be executed in the terminal using the following command:

- python -m unittest test_lattice_creation.py

This is how the code looks like:

```python
class TestLatticeCreation(unittest.TestCase):

    def test_create_mgo_lattice(self):
        """ Test lattice creation for MgO (should alternate between Mg and O). """
        lattice = create_lattice("MgO", 5)

        # Check if the shape is correct
        self.assertEqual(lattice.shape, (5, 5, 5))

        # Verify alternating ions: Mg=0, O=1
        for i in range(5):
            for j in range(5):
                for k in range(5):
                    if (i + j + k) % 2 == 0:
                        self.assertEqual(lattice[i, j, k], 0)  # Mg
                    else:
                        self.assertEqual(lattice[i, j, k], 1)  # O

    def test_create_feo_lattice(self):
        """ Test lattice creation for FeO (should alternate between Fe and O). """
        lattice = create_lattice("FeO", 5)

        # Check if the shape is correct
        self.assertEqual(lattice.shape, (5, 5, 5))

        # Verify alternating ions: Fe=0, O=1
        for i in range(5):
            for j in range(5):
                for k in range(5):
                    if (i + j + k) % 2 == 0:
                        self.assertEqual(lattice[i, j, k], 0)  # Fe
                    else:
                        self.assertEqual(lattice[i, j, k], 1)  # O

    def test_invalid_material(self):
        """ Test that invalid material input raises an error. """
        with self.assertRaises(ValueError):
            create_lattice("NaCl", 5)  # Invalid material should raise ValueError

if __name__ == '__main__':
    unittest.main()
```

This test contains three functions:

- test_create_mgo_lattice: test if the lattice function creates a 5x5x5 lattice for MgO correctly
- test_create_feo_lattice: test if the lattice function creates a 5x5x5 lattice for FeO correctly
- test_invalid_material: tests if the function raises a ValueError when an invalid material is passed

4. Initial Implementation

The Hw2_DataAnalysis.ipynb Jupyter notebook contains some of the core functionality implementation for the creation of the lattice and initial calculation of energies for the cell.

2. Set parameters for the simulation

Define the simulation parameters

```
material = 'MgO'        # Can be 'MgO' or 'FeO'
lattice_size = 5      # Lattice size (5x5x5)
num_na_ions = 5        # Number of Na⁺ ions to be inserted
temperature = 300       # Temperature in Kelvin
num_simulations = 10     # Number of simulations to run
steps = 1000            # Number of Monte Carlo steps

k_e = 8.99e9  # Coulomb's constant in N·m²/C²
r_Mg = 72e-12  # Ionic radius of Mg2+ in meters
r_Fe = 78e-12  # Ionic radius of Fe2+ in meters
r_Na = 102e-12  # Ionic radius of Na+ in meters
r_O = 140e-12  # Ionic radius of O2- in meters
k_B = 1.38e-23  # Boltzmann constant in J/K
```

[62] ✓ 0.0s Python

3. Create the initial lattice

Use the create_lattice function to generate the lattice for the selected material (MgO or FeO)

This module generates a 3D lattice representing MgO or FeO. Both of them have a cubic structure or also named rocksalt similar to NaCl. It is one of the simplest crystal structures and can be a model for cation substitution. Each Mg2+ or Fe2+ ion have an octahedral coordination. This means that they are surrounded by 6 O2- ions

Use the function create_lattice of the module lattice.py to generate the 3D lattice. For this example, I'm using a cell with a dimenssion of 5x5x5. Some authors like Vinograd et al., (2006) used a supercell with a 3x3x3 configuration. For the initial implementation, a larger cell can give more information about the energy distribution when sodium is incorporated inside the structure.

This function takes the variables material and lattice size Material can be MgO or FeO. For future implementations, I want to include different proportions of Fe inside the cell.

```
# Create the lattice for the chosen material

lattice = create_lattice(material=material, L=lattice_size)

lattice  # The output is an array with the structure of the 5X5X5 cell
```

[69] ✓ 0.0s Pythor

```
array([[[0, 1, 0, 1, 0],
        [1, 0, 1, 0, 1],
        [0, 1, 0, 1, 0],
        [1, 0, 1, 0, 1],
        [0, 1, 0, 1, 0]],

       [[1, 0, 1, 0, 1],
        [0, 1, 0, 1, 0],
        [1, 0, 1, 0, 1],
        [0, 1, 0, 1, 0],
        [1, 0, 1, 0, 1]],

       [[0, 1, 0, 1, 0],
        [1, 0, 1, 0, 1],
        [0, 1, 0, 1, 0],
        [1, 0, 1, 0, 1],
        [0, 1, 0, 1, 0]],

       [[1, 0, 1, 0, 1],
        [0, 1, 0, 1, 0],
```

4. Calculate the energy

This module calculates the Coulombic interactions between ions using Coulomb's law.

The Coulomb's law calculates the amount of force between two charged particles. The closer two charges are, the stronger the force between them.

$$E_{\text{Coulomb}} = \frac{k_e \cdot Q_1 \cdot Q_2}{r^2}$$

where ke is Coulomb's constant (8.99e9 N*m^2/C^2), Q1 and Q2 are the charges of the interacting ions, and r is the distance between them.

Use the file energy_calculation_2.py to access the function energy_change. This function calculate the electrostatic interaction and estimate the change of energy based on the Coulombic energy between Na+ and its neighbors. The neighbors list contains the six nearest neighbors as it has an octahedral coordination in the 3D configuration.

In the code, it is assumed that the atoms have a periodic arrange.

```
#k_e = 8.99e9  # Coulomb's constant in N·m²/C²
#r_Mg =  20e-12   # Ionic radius of Mg2+ in meters 72e-12
#r_Fe = 20e-12  # Ionic radius of Fe2+ in meters  78e-12
#r_Na = 20e-12  # Ionic radius of Na+ in meters  102e-12
#r_O = 20-12  # Ionic radius of O2- in meters 140e-12
#k_B = 1.38e-23  # Boltzmann constant in J/K

i, j, k = 1, 1, 1 # Position of the Na+ cation
target_ion = 'Mg' # Fe is also an option


energy = energy_change(i, j, k, lattice, target_ion)

# Print the result
print(f"The energy change due to Na+ insertion at ({i}, {j}, {k}) is: {energy} J")
```

[64] ✓ 0.0s Pythor

```
The energy change due to Na+ insertion at (1, 1, 1) is: -107880000000.0 J
```

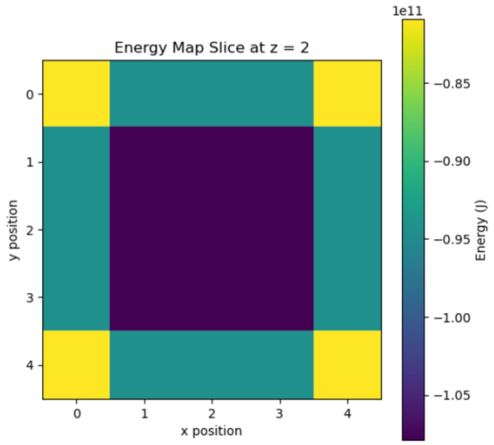
Visualization and Analysis:

5. Map of the distribution of energies

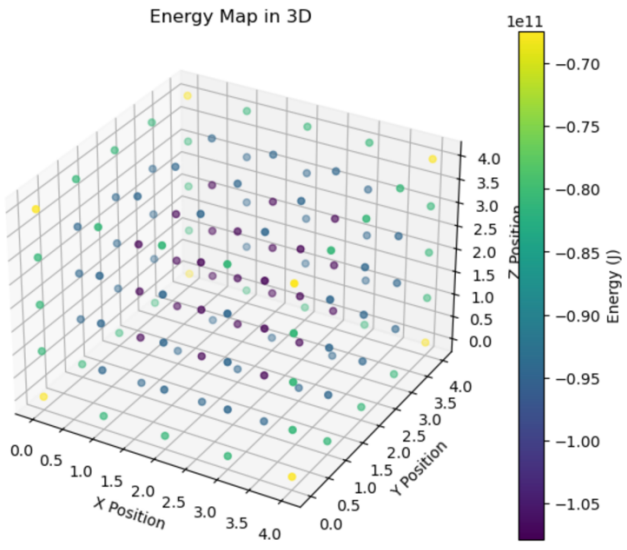
This module looks to create visualizations of energies through maps. The code iterates in the lattice position and calculate the energy at each point and then plot the energy slice

```
# Initialize an empty array to store the energy values for each lattice position
energy_map = np.zeros((lattice_size, lattice_size, lattice_size))

# Loop through all positions in the lattice to calculate the energy at each point
for i in range(lattice_size):
    for j in range(lattice_size):
        for k in range(lattice_size):
            energy_map[i, j, k] = energy_change(i, j, k, lattice, target_ion)

#Visualize the energy map

# Plotting a slice of the energy map
slice_index = lattice_size //2   # Choose a central slice for visualization
energy_slice = energy_map[slice_index, :, :]  # Take a slice along one axis

# Plotting the energy slice
plt.figure(figsize=(6, 6))
plt.imshow(energy_slice, cmap='viridis', interpolation='nearest')
plt.colorbar(label='Energy (J)')
plt.title(f"Energy Map Slice at z = {slice_index}")
plt.xlabel('x position')
plt.ylabel('y position')
plt.show()
```

[72] ✓ 0.0s Pythor




```
# 3D plot that shows the distribution of energy at different positions in the 3D cell



from mpl_toolkits.mplot3d import Axes3D

# Get the x, y, z coordinates and energy values
x, y, z = np.indices((lattice_size, lattice_size, lattice_size))
energies = energy_map.flatten()

# Plotting in 3D
fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(111, projection='3d')
scatter = ax.scatter(x.flatten(), y.flatten(), z.flatten(), c=energies, cmap='viridis', s=20)
fig.colorbar(scatter, label='Energy (J)')
ax.set_xlabel('X Position')
ax.set_ylabel('Y Position')
ax.set_zlabel('Z Position')
ax.set_title('Energy Map in 3D')
plt.show()
```



5. Progress report

Project Progress Assessment:

The literature review and the cell and energy calculations took more weeks that expected. So, for this initial implementation, I'm including the initial code for the Monte Carlo simulations that need to be tested in the following weeks. Also, the original idea needed to be modified to simplify the model. So, initially, I wanted to model variables like the effect of pressure and temperature for the solid solution (Mg,Fe)O, however, the initial implementation only considered the end members MgO and FeO for the calculation of energies in the incorporation of sodium.

This is the updated Gantt chart:

	Timeline										
Task name	**W5**	**W6**	**W7**	**W8**	**W9**	**W10**	**W11**	**W12**	**W13**	**W14**	**W15**
Literature Review											
Cell and Energy calculations											
Monte Carlo Implementation											
Run Simulations and Analyze Trends											
Compare with experimental data											
Finalize Analysis and visualizations											
Final Report and presentation											
Milestones	Homework 1					Homework 2		Homework 3		Project Due	

Course Concept application:

1. Concept: Version control.

Application in Project: GitHub has been used to upload the Git repository and the control versions.

2. Concept: Statistical analysis of stochastic models.

Application in Project: I will use statistical analysis applied to probability distributions using the Monte Carlo algorithm, specifically applying the Metropolis Monte Carlo method and the Boltzman Distribution for the calculus of energies and simulations of prefered sites of Na in (Mg,Fe)O]

3.Concept: High performant Python and Optimization.

Application in Project: I want to use methods of optimization for the Monte Carlo simulations. They can be computational intensive, so using methods like numpy or numba can help to reduce the processing time of the simulations.