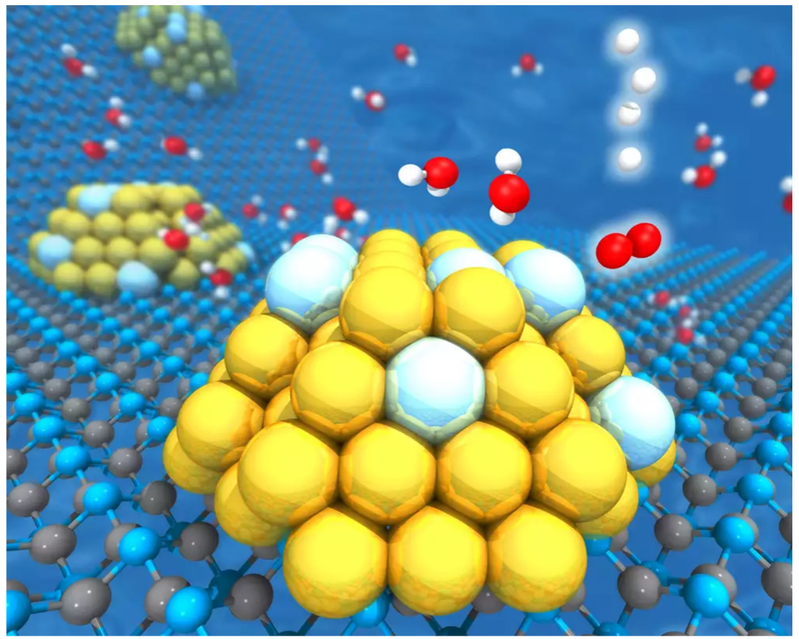
Ex4: Nanoparticle Energy Minimization

Thermodynamics for Modelling and Simulation
WiSe 2022/23

Abhishek Khetan
askhetan@modes.rwth-aachen.de
Schinkelstr. 8
Raum 212 (2.O.G.)
52062 Aachen

www.modes.rwth-aachen.de

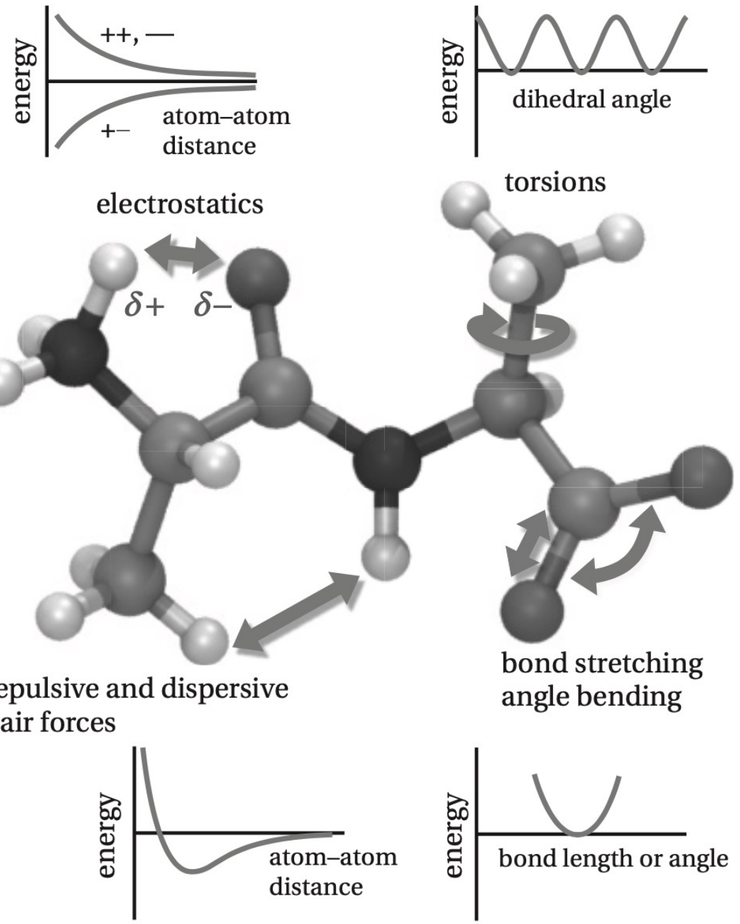


https://www.mpie.de/atomistic

MODES
Junior Professorship for Multi-scale Modeling of Heterogeneous Catalysis in Energy Systems
RWTH AACHEN UNIVERSITY

Total energy from the Hamiltonian, and forces

$$H(\mathbf{p}^N, \mathbf{r}^N) = K(\mathbf{p}^N) + U(\mathbf{r}^N) = E_n^{kin} + E_e^{kin} + E_{n-e}^{pot} + E_{e-e}^{pot} + E_{n-n}^{pot}$$



$$U(\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_N) = \sum_{\text{bonds } i} a_i (d_i - d_{i,0})^2$$

$$+ \sum_{\text{angles } i} b_i (\theta_i - \theta_{i,0})^2$$

$$+ \sum_{\text{torsions } i} \left(\sum_n c_{i,n} \cos(\omega_i n)^n \right)$$

$$+ \sum_{\text{atom pairs } ij} \frac{q_i q_j}{4\pi\epsilon_0 r_{ij}}$$

electrostatics

$$+ \sum_{\text{atom pairs } ij} 4\epsilon_{ij} \left[\left(\frac{r_{ij}}{\sigma_{ij}} \right)^{-12} - \left(\frac{r_{ij}}{\sigma_{ij}} \right)^{-6} \right]$$

atom–atom repulsion at close distances

van der Waals attractions at long distances

bonded interactions (intramolecular)

nonbonded interactions (intra- and intermolecular)

Forces

$$-\frac{dU}{d\mathbf{r}_i}(\mathbf{r}_1, \mathbf{r}_2, \ldots) = \mathbf{F}_i \qquad \text{for all atoms } i$$

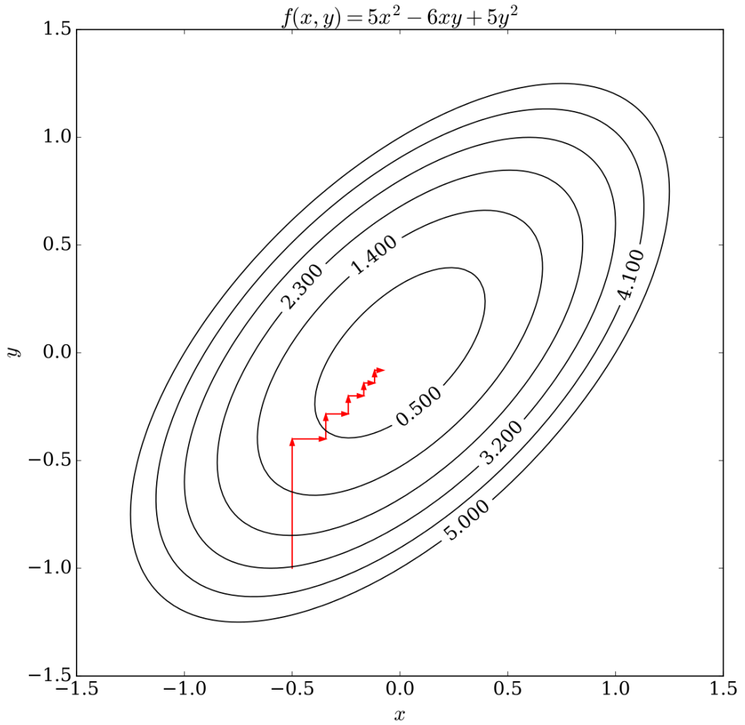
Locating the minima: Univariate search or Line search

Algorithm:

1. Start with an initial set of coordinates $\mathbf{r}_o^N = (\ldots r_1\hat{\mathbf{x}}_1 + r_2\hat{\mathbf{x}}_2 + r_3\hat{\mathbf{x}}_3 \ldots)$ and a search direction along $\hat{\mathbf{x}}_1$

2. Assign $\mathbf{r}_1^N = \mathbf{r}_o^N + \delta_1\hat{\mathbf{x}}_1$ and $\mathbf{r}_2^N = \mathbf{r}_1^N + \delta_1\hat{\mathbf{x}}_1$

3. Are $U(\mathbf{r}_2^N) > U(\mathbf{r}_1^N)$ AND $U(\mathbf{r}_o^N) > U(\mathbf{r}_1^N)$?



If NO, then
4. Shift values as $\mathbf{r}_o^N = \mathbf{r}_1^N$, $\mathbf{r}_1^N = \mathbf{r}_2^N$, $\mathbf{r}_2^N = \mathbf{r}_1^N + \delta_1\hat{\mathbf{x}}_1$, and **go back to step 3**

If YES, then
4. Locate the minimum $\mathbf{r}_{\min}^N$ between $\mathbf{r}_o^N$, $\mathbf{r}_1^N$ and $\mathbf{r}_2^N$ by bisecting or fitting a parabola

5. Assign $\mathbf{r}_o^N = \mathbf{r}_{\min}^N$, a new search direction along $\hat{\mathbf{x}}_2$ and **go back to step 2.** Repeat this over **all atoms and all directions** until the change in energy ΔU in every direction is less than some tolerance value (typically ~10^{-6}).

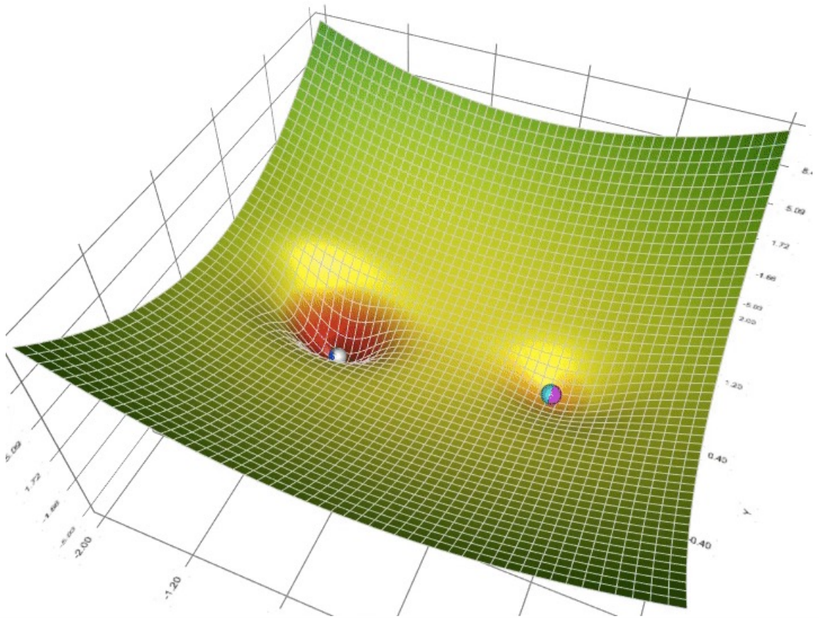
Locating the minima: Line search with steepest descent

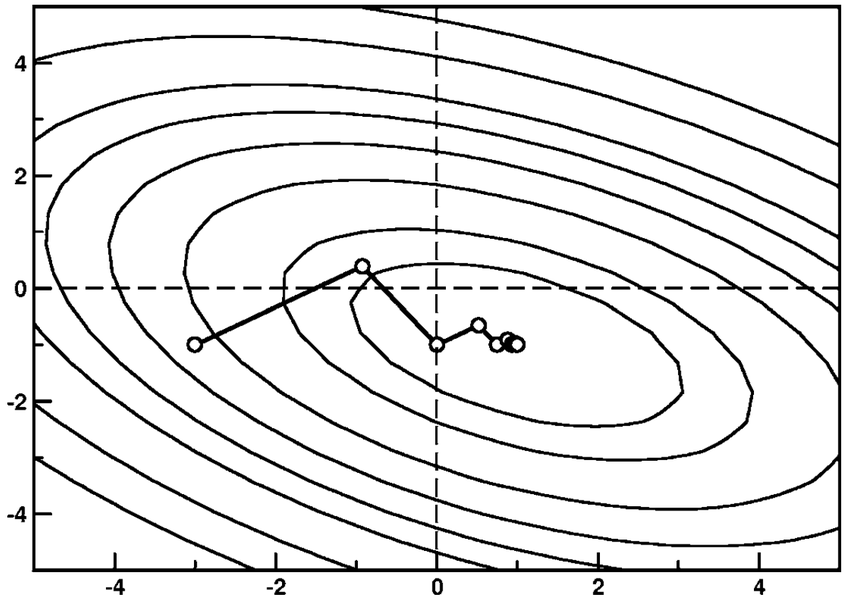
Algorithm:

1. Start with an initial set of coordinates $\mathbf{r}_{\mathrm{o}}^{N}$ and perform a line search along direction of $\mathbf{f}^{N}$ to find a new minimum point $\mathbf{r}_{\mathrm{min}}^{N}$

2. Assign $\mathbf{r}_{\mathrm{o}}^{N} = \mathbf{r}_{\mathrm{min}}^{N}$. Compute new set of forces $\mathbf{f}^{N}$ at $\mathbf{r}_{\mathrm{min}}^{N}$. Perform line search along direction of $\mathbf{f}^{N}$

3. Repeat Step 2 over until the change in energy ΔU is less than some tolerance value (typically ~10^{-6})





Locating the minima: Conjugate gradients

Algorithm:

1. Start with an initial set of coordinates $\mathbf{r}_{\mathrm{o}}^{N}$ and perform a (steepest descent) line search along direction of $\mathbf{d}_{\mathrm{o}}^{N} = \mathbf{f}_{\mathrm{o}}^{N}$ to find a new minimum point $\mathbf{r}_{1}^{N}$

2. Assign $\mathbf{r}_{\mathrm{o}}^{N} = \mathbf{r}_{1}^{N}$ $(or\ \ \mathbf{r}_{\mathrm{i}}^{N} = \mathbf{r}_{\mathrm{i}-1}^{N})$. Compute new set of forces $\mathbf{f}_{1}^{N}$ at $\mathbf{r}_{\mathrm{o}}^{N}$. Perform line search along direction of $\mathbf{d}_{1}^{N}$ such that:

$$\mathbf{d}_{1}^{N} = \mathbf{f}_{1}^{N} + \gamma_1 \mathbf{d}_{\mathrm{o}}^{N} \quad (or\ \ \mathbf{d}_{\mathrm{i}}^{N} = \mathbf{f}_{\mathrm{i}}^{N} + \gamma_i \mathbf{d}_{\mathrm{i}-1}^{N})$$

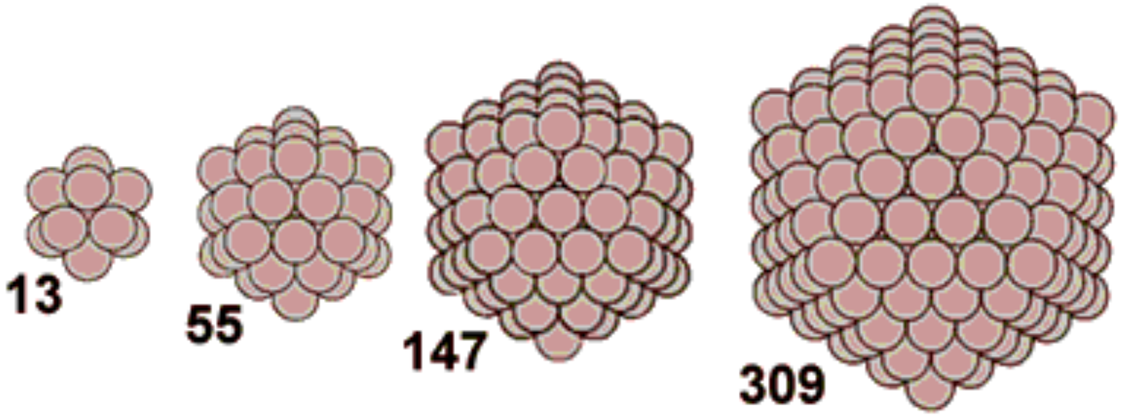
where:

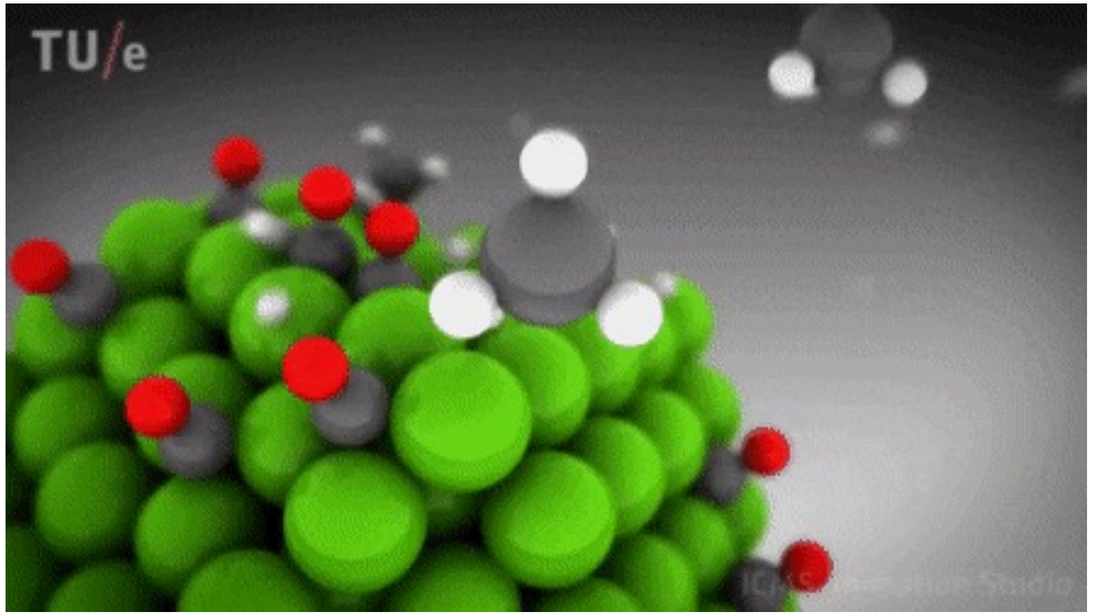
$$\gamma_1 = \frac{\mathbf{f}_{1}^{N} . \mathbf{f}_{1}^{N}}{\mathbf{f}_{\mathrm{o}}^{N} . \mathbf{f}_{\mathrm{o}}^{N}} \quad (or\ \ \gamma_i = \frac{\mathbf{f}_{\mathrm{i}}^{N} . \mathbf{f}_{\mathrm{i}}^{N}}{\mathbf{f}_{\mathrm{i}-1}^{N} . \mathbf{f}_{\mathrm{i}-1}^{N}})$$

3. Repeat Step 2 with until the change in energy ΔU is less than some tolerance value (typically ~10^{-6})

First assignment: Energy minimization of nanoclusters

Example “magic number” icosahedral clusters





Task: write a code to perform energy minimization for nanoparticles/nanoclusters of varying size (N) using the conjugate-gradient method.

$$\mathbf{d}_i^N = \mathbf{f}_i^N + \gamma_i^N \mathbf{d}_{i-1}^N, \text{ where } \gamma_i^N = \frac{(\mathbf{f}_i^N - \mathbf{f}_{i-1}^N).\mathbf{f}_i^N}{\mathbf{f}_{i-1}^N.\mathbf{f}_{i-1}^N}$$

MODES Junior Professorship for Multi-scale Modeling of Heterogeneous Catalysis in Energy Systems
RWTH AACHEN UNIVERSITY

Next assignment: Energy minimization of clusters using python

1. Energy model

$$U^* = \sum_i \alpha |\mathbf{r}_i|^2 + \sum_{i<j} u(r_{ij}) \qquad \alpha = 0.0001N^{-2/3}$$

use $\epsilon = \sigma = 1$ in the force field, $r_c = 2.5\sigma$

2. Truncate the pairwise potential:

conventional

$$u(r_{ij}) = 4\epsilon\left[\left(\frac{r_{ij}}{\sigma}\right)^{-12} - \left(\frac{r_{ij}}{\sigma}\right)^{-6}\right]$$

to implement in assignment

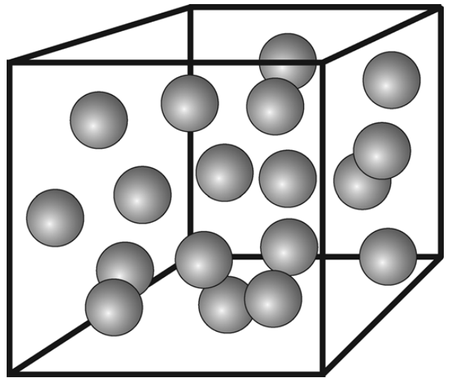
$$u(r_{ij}) = \begin{cases} 4\epsilon\left[\left(\frac{r_{ij}}{\sigma}\right)^{-12} - \left(\frac{r_{ij}}{\sigma}\right)^{-6}\right] - 4\epsilon\left[\left(\frac{r_c}{\sigma}\right)^{-12} - \left(\frac{r_c}{\sigma}\right)^{-6}\right] & r_{ij} \le r_c \\ 0 & r_{ij} > r_c \end{cases}$$

$$r_c = 2.5\sigma$$

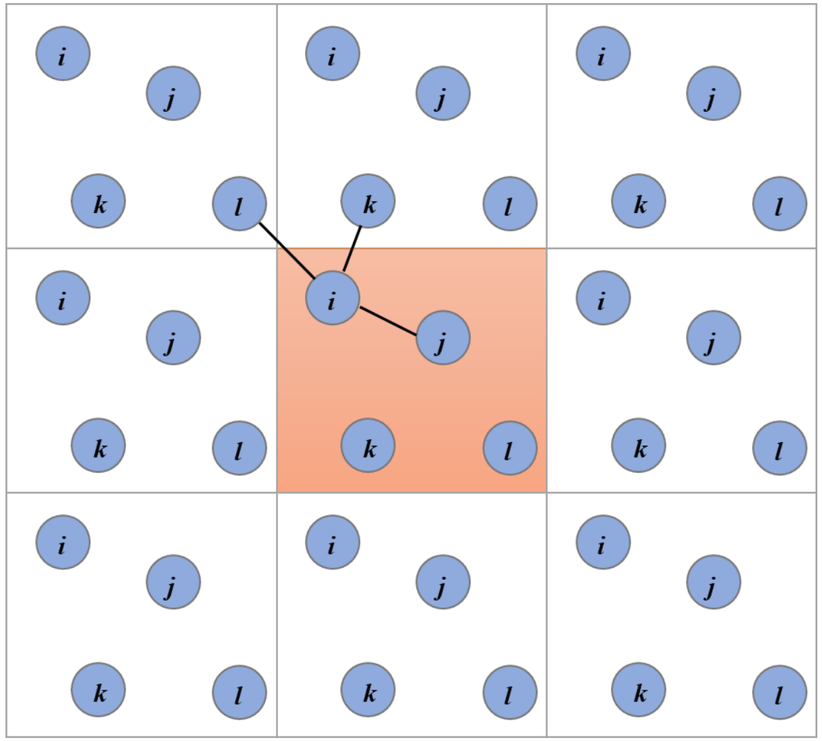
MODES Junior Professorship for Multi-scale Modeling of Heterogeneous Catalysis in Energy Systems
RWTH AACHEN UNIVERSITY

Next assignment: Energy minimization of clusters using python

1. Periodic boundary conditions



$$i = 1, 2, \ldots, N$$



2. Find minimum distance $\mathbf{r}_{ij}^{\mathbf{o}}$ as:

$$\mathbf{r}_{ij}^{0} = \mathbf{r}_{ij} - L\ \mathrm{nint}\left(\mathbf{r}_{ij}/L\right)$$

where L is the length of the cubic box

Tasks: Energy minimization of nanoclusters

(1) perform energy minimization with:

-- N = 2 to 25

--for each N perform M = 1000 simulations starting each time with a new random initial structure

--use $\epsilon = \sigma = 1$ in the force field, $r_c = 2.5\sigma$

--use fractional tolerance value of Line search LStol= 10^{-8} ($|U_i - U_{i-1}| < LStol \times |U_i|$), $\delta = 0.001$, MaxLSSteps = 100000

[Hint: $\delta = 0.001$ is just a recommended value. If you want to adapt it to make it larger at the beginning of the LineSearch and smaller at the end, go ahead and use your creativity such that you may make your LineSearch more efficient. You can make it dependent on the fractional change in energy]

--use fractional tolerance value of Energy change ECtol= 10^{-10} ($|U_i - U_{i-1}| < ECtol \times |U_i|$), MaxCGSteps = 100000

--use size L such that $\frac{N}{V} = 0.05$ where $V = L^3$

(2) From the M = 1000 simulations at each N, save the positions and potential energy U_{min}^N of the lowest minimum and the number of CGSteps (i) in which it is reached. Plot U_{min}^N and CGSteps required to reach it as a function of N.

(3) For this minimum case at each N, plot the evolution of potential energy as a function of the CGSteps and visualize the particle.

(4) The macroscopic scaling estimate for nanoparticle energy is given by $U_{macro}(N) = a + bN^{\left(\frac{2}{3}\right)} + cN$. Fit the minimum potential energy data to this formula using least-squares minimization in python and find the constants a, b, c. Finally plot the $U_{\min}^N - U_{macro}(N)$ as a function of N. Do you notice anything different at certain numbers?

Deliverables: Energy minimization of nanoclusters

(1) complete python code as Jupyter notebook shared over the git that includes separate functions to:

--initialize positions

input: number of atoms (N) and box size (L)

output: positions [(N,3) array]

--calculate energy

input: positions [(N,3) array] and box size (L)

output: potential energy

--calculate forces

input: positions [(N,3) array] and box size (L)

output: forces [(N,3) array]

[Hint: It is possible that energy and forces are need both at the same time. Can you make a combined function as well?]

--perform line search

input: positions [(N,3) array], search direction [(N,3) array], δ, LStol=10^{-8}, MaxLSSteps= 10^{5}

output: positions [(N,3) array] and potential energy

--perform conjugate gradient minimization

input: positions [(N,3) array], ECtol=10^{-10} and MaxCGSteps= 10^{5}

output: positions [(N,3) array] and potential energy

[Hint: Line search will be called from within the conjugate gradient minimizer – which variables must be globally declared?]

--perform the minimization for different starting positions

--plot the results from the tasks directly in the Jupyter notebook

(2) PDF file with plots from tasks (2, 3, 4) with clear axes and figure captions specifying the simulation parameters

MODES Junior Professorship for Multi-scale Modeling of Heterogeneous Catalysis in Energy Systems
RWTH AACHEN UNIVERSITY

Extra tips: Energy minimization of clusters using python

--For initialization, use the random number generator in 3 directions always going from -L/2 to +L/2 such that the origin is in the centre of the box.

--The LineSearch has to be implemented by moving all atoms at once along the direction, not one by one. Before using the direction in LineSearch, don't forget to normalize it. Also before normalization, check if any absolute value of any force component is too large (going to +/- infinity) - if so, clip it to something more finite like +/- 10e4. Also, gamma is an Nx1 vector.

--For locating the minimum at the end you can use the bisection method, or fit the data to a parabola or a combination of both. There are NO RESTRICTIONS on that.

--This problem is a classic Lennard Jones simulation problem for which a paper was published in 1997 (https://link.springer.com/content/pdf/10.1023/A:1008276425464.pdf). You can already compare your own minimum for each N to that reported in this paper, just to see how good your simulations are.

Solution strategy (if you don’t know where to start):Try using “readymade” line search and conjugate gradient minimizers (I make no preferable suggestions) to first build the outer loop for getting the final answers. Hopefully, then you can see the parts of the total program building up, get a feeling of the outer loops of N and M. Eventually you can replace the readymade minimizers with your own.

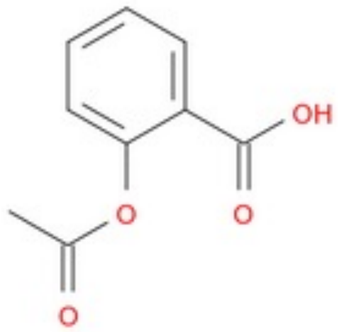
Grading Scheme (60 + 10bonus)

(25) Task1- Code for Line Search + Conjugate Gradient + Position Initialization + Energy + Forces
(10) Results - completeness and accuracy. Even if your code is not perfect - you get points here at least if it produces some resonable numbers
(5) Task2 code + plot
(10) Task3 code + plots + visuals (for each N)
(10) Task4 code + plot
(10) Bonus points for $\frac{\mathrm{N}}{\mathrm{V}} = 0.01$ only when completing all the prior tasks by yourself (i.e. no coping or using built-in functions)

MODES Junior Professorship for Multi-scale Modeling of Heterogeneous Catalysis in Energy Systems
RWTH AACHEN UNIVERSITY

Visualization software and geometry file formats: xyz

- Many software, many formats
  - xyz
  - pdb
  - gro (GROMACS)
  - mol/sdf
  - cif
  - POSCAR (VASP)
  - …….



C9H8O4

Aspirin.xyz

```
21
Aspirin
C 2.05639 -0.16543 0.39727
C 0.68890 2.02854 -0.64816
C 2.74071 1.02327 0.12533
C 2.05734 2.11782 -0.40016
C -3.59396 1.03209 0.04589
C 0.67738 -0.27507 0.15097
C 0.00274 0.84234 -0.35887
O 0.63293 -2.54526 0.93360
O -1.76452 1.43252 1.56571
O -1.24894 -1.68411 0.12975
C 0.05278 -1.58881 0.44958
C -2.15199 1.12841 0.44513
O -1.36167 0.82832 -0.65825
H -1.47652 -2.60111 0.39232
H 2.60948 -1.01173 0.80284
H 0.15708 2.88266 -1.05823
H 3.80777 1.09126 0.32364
H 2.59044 3.04081 -0.61405
H -3.81640 0.02049 -0.30287
H -4.22480 1.24591 0.91440
H -3.81296 1.76576 -0.73332
```

Visualization: for visualizing the trajectory and final geometry from pos_list

--For example, my pos_list looks like this. It has 3 frames (from 3 steps) and the number of atoms is = 5:

```
[array([[0.61939829, 1.32879669, 0.30544959],
    [1.51199518, 1.32836717, 0.98258992],
    [1.75734328, 0.27676396, 1.26700374],
    [1.37274924, 0.50322375, 0.23760851],
    [0.6883065 , 0.56524291, 1.12244757]]),
array([[ 1.31389696, 0.10802717, 1.12654671],
    [ 0.9389867 , 1.06058787, 1.58187981],
    [ 0.97235851, 0.96121552, 0.46641162],
    [ 0.22703922, 0.37178331, 1.0599827 ],
    [ 1.92408014, 1.03970773, 1.04522908]]),
array([[ 0.29985348, 0.98878387, 0.47166138],
    [ 1.31391836, 0.647785 , 0.80428378],
    [-0.31763666, 1.23579606, 1.36868183],
    [ 0.55865088, -0.09698025, 0.4385713 ],
    [ 0.41837656, 0.39822014, 1.42955791]])]
```

--Install software : ase (https://wiki.fysik.dtu.dk/ase/)

Visualization

--After you've installed ase - this should be your code to create the trajectory. Please note that C5 in the code means 5 carbon atoms. So for 20 atoms you have to replace that by C20. You can adjust the factor (fac=1.0) to make the simulations prettier if necessary. For me it doesn't seem to make a difference.

```
from ase import Atoms
from ase.visualize import view
from ase.io.trajectory import Trajectory
traj = Trajectory('nano.traj', 'w')
i=0
fac=1.0
while i < len(pos_list):   # pos_list is the list of positions
    print(i)
    nano = Atoms('C5', pos_list[i]*fac)
    i+=1
    traj.write(nano)
```

--Finally when this is done successfully a file called nano.traj. will be written in the same folder. This is the file that can be visualized. Go to your command line in the same folder and use:

```
$ ase gui nano.traj
```

Thank you for your attention!