Computations

Sohyun Park

Table of contents

# 1. Localised external potential

The localised external potential added to the system (Turci and Wilding 2021):

where is the Heaviside function, , and represents the repulsive barrier strength.

import numpy as np  
import matplotlib.pyplot as plt  
  
# Parameters  
sigma = 1.0 # Particle diameter  
d = sigma # Barrier width  
ew\_values = np.arange(20, 55, 5) # Barrier strength values from 20 to 50 incrementing by 5  
  
# Define the potential function  
def V\_ext(x, epsilon\_w, d):  
 H = lambda x: np.heaviside(x, 0.5) # Heaviside step function  
 return epsilon\_w \* (np.cos(np.pi \* x / d) + 1) \* H(d - x) \* H(x + d)  
  
# x range  
x = np.linspace(-2 \* d, 2 \* d, 500)  
  
# Plot  
plt.figure(figsize=(10, 6))  
for ew in ew\_values:  
 V = V\_ext(x, ew, d)  
 plt.plot(x, V, label=f"$\\epsilon\_w = {ew}$")  
  
plt.axvline(-d, color='gray', linestyle='--', label=r"$x = -d$", linewidth=0.8)  
plt.axvline(d, color='gray', linestyle='--', label=r"$x = d$", linewidth=0.8)  
plt.axhline(0, color='black', linewidth=0.5)  
plt.xlabel("$x$", fontsize=12)  
plt.ylabel("$V\_{\\text{ext}}(x)$", fontsize=12)  
plt.legend(fontsize=10, loc='upper right')  
plt.grid(alpha=0.5)  
plt.show()

|  |
| --- |
| Figure 1.1: External potential for different |

|  |
| --- |
| Note |
| Intermediate values can be thought of as representing a thin porous membrane with nonzero crossing probability. |

Callout tip

|  |
| --- |
| Tip |
| Note that there are five types of callouts, including: note, tip, warning, caution, and important. |

# 2. Asymmetry order parameter

The degree of asymmetry of the instantaneous density profile with respect to the barrier location is quantified as:

In the steady state, the average (over time and distinct initial conditions) provides a measure of the typical asymmetry of the liquid region with respect to .

import pandas as pd  
import numpy as np  
import re  
import os  
import matplotlib.pyplot as plt  
  
def load\_profile(filename):  
 """  
 Load LAMMPS profile dump file into a pandas DataFrame.  
 Reads multiple timesteps and returns a DataFrame containing data for all timesteps.  
 """  
 data = []  
 with open(filename, 'r') as f:  
 lines = f.readlines()  
  
 i = 0  
 while i < len(lines):  
 line = lines[i].strip()  
 # Skip comment lines and empty lines  
 if line.startswith('#') or line == '':  
 i += 1  
 continue  
 tokens = line.split()  
 if len(tokens) == 3:  
 # Try to parse the line as Timestep and Number-of-chunks  
 try:  
 timestep = int(tokens[0])  
 chunk\_count = int(tokens[1])  
 # We can ignore total\_count  
 except ValueError:  
 # If parsing fails, skip this line  
 i += 1  
 continue  
 # Read the data for this timestep  
 data\_start = i + 1  
 data\_end = data\_start + chunk\_count  
 for j in range(data\_start, data\_end):  
 if j >= len(lines):  
 break # Prevent index out of range  
 dl = lines[j].strip()  
 if dl == '':  
 continue  
 dl\_tokens = dl.split()  
 if len(dl\_tokens) != 4:  
 continue  
 try:  
 chunk = int(dl\_tokens[0])  
 coord1 = float(dl\_tokens[1])  
 ncount = int(float(dl\_tokens[2])) # Convert to float first to handle possible decimals  
 density = float(dl\_tokens[3])  
 data.append({  
 'Timestep': timestep,  
 'Chunk': chunk,  
 'Coord1': coord1,  
 'Ncount': ncount,  
 'density': density  
 })  
 except ValueError:  
 continue # Skip lines with invalid data  
 i = data\_end  
 else:  
 i += 1  
  
 df = pd.DataFrame(data)  
 return df # Return the DataFrame with all timesteps  
  
def calculate\_asymmetry(data, lx, rho\_ld, rho):  
 """  
 Compute the asymmetry order parameter A(t) with proper normalization.  
 """  
 x\_bins = data['Coord1'] # Bin center positions along x-axis  
 densities = data['density'] # Density per bin  
  
 # Ensure that x\_bins and densities are sorted according to x\_bins  
 sorted\_indices = np.argsort(x\_bins)  
 x\_bins = x\_bins.iloc[sorted\_indices].reset\_index(drop=True)  
 densities = densities.iloc[sorted\_indices].reset\_index(drop=True)  
  
 bin\_width = abs(x\_bins.iloc[1] - x\_bins.iloc[0]) # Calculate bin width  
  
 # Split the bins into left and right of the barrier (x = 0)  
 left\_mask = x\_bins < 0  
 right\_mask = x\_bins > 0  
  
 left\_bins = densities[left\_mask]  
 right\_bins = densities[right\_mask]  
  
 left\_x = x\_bins[left\_mask]  
 right\_x = x\_bins[right\_mask]  
  
 # Integrate density over the left and right regions  
 left\_integral = (left\_bins \* bin\_width).sum()  
 right\_integral = (right\_bins \* bin\_width).sum()  
  
 # Numerator for A(t)  
 numerator = abs(right\_integral - left\_integral)  
  
 # Denominator normalization  
 denominator = (rho - rho\_ld) \* lx  
  
 # Asymmetry order parameter  
 asymmetry = numerator / denominator if denominator != 0 else 0  
  
 return asymmetry  
  
def process\_dump\_files(folder, lx, rho\_ld, rho):  
 """  
 Process all dump files in the given folder to calculate asymmetry A(t).  
 """  
 results = {}  
  
 # Regex to extract `ew` and `ly` from file names  
 pattern = re.compile(r"wet\.ew\.(\d+)\.ly\.(\d+)\.dump")  
  
 for filename in os.listdir(folder):  
 # Skip files that do not end with '.dump'  
 if not filename.endswith(".dump"):  
 continue  
  
 match = pattern.match(filename)  
 if match:  
 ew = int(match.group(1))  
 ly = int(match.group(2))  
  
 # Ensure results dictionary is organized by `ly`  
 if ly not in results:  
 results[ly] = []  
  
 filepath = os.path.join(folder, filename)  
 try:  
 data = load\_profile(filepath)  
 # Compute asymmetry at each timestep  
 asymmetry\_list = []  
 grouped = data.groupby('Timestep')  
 for timestep, df\_timestep in grouped:  
 asymmetry = calculate\_asymmetry(df\_timestep, lx=lx, rho\_ld=rho\_ld, rho=rho)  
 asymmetry\_list.append(asymmetry)  
 # Average asymmetry over time  
 average\_asymmetry = np.mean(asymmetry\_list)  
 results[ly].append((ew, average\_asymmetry))  
 except Exception as e:  
 print(f"Error processing {filepath}: {e}")  
 continue  
  
 # Sort results by `ew` for each `ly`  
 for ly in results:  
 results[ly].sort(key=lambda x: x[0])  
  
 return results  
  
def plot\_asymmetry(results):  
 """  
 Plot asymmetry A(t) as a function of ew for different system sizes Ly.  
 """  
 plt.figure(figsize=(8, 6))  
  
 # Assign colors and markers for each system size Ly  
 num\_ly = len(results)  
 colors = plt.cm.viridis(np.linspace(0, 1, num\_ly))  
 markers = ['o', 's', 'D', '^', 'v', 'x', '\*', '+', 'p', 'h']  
  
 for i, (ly, data) in enumerate(sorted(results.items())):  
 ew, asymmetry = zip(\*data)  
 plt.plot(ew, asymmetry, label=f"$L\_y = {ly}$", color=colors[i % len(colors)], marker=markers[i % len(markers)])  
  
 # Add shading to represent different regimes (adjust ranges as needed)  
 plt.axvspan(10, 20, color='black', alpha=0.1, label="Unpinned")  
 plt.axvspan(20, 30, color='yellow', alpha=0.2, label="Asymmetric")  
 plt.axvspan(30, 50, color='pink', alpha=0.2, label="Symmetric")  
  
 plt.xlabel(r"$\epsilon\_w$", fontsize=14)  
 plt.ylabel(r"$\langle \mathcal{A}(t) \rangle$", fontsize=14)  
 plt.legend(fontsize=12)  
 plt.grid(True)  
 plt.tight\_layout()  
 plt.show()  
  
def plot\_asymmetry\_vs\_time(filename, lx, rho\_ld, rho):  
 """  
 Compute and plot the asymmetry order parameter A(t) as a function of Timestep for a given dump file.  
 """  
 data = load\_profile(filename)  
 # Compute asymmetry at each timestep  
 asymmetry\_list = []  
 timesteps = []  
 grouped = data.groupby('Timestep')  
 for timestep, df\_timestep in grouped:  
 asymmetry = calculate\_asymmetry(df\_timestep, lx=lx, rho\_ld=rho\_ld, rho=rho)  
 asymmetry\_list.append(asymmetry)  
 timesteps.append(timestep)  
  
 # Sort the data by timestep  
 timesteps, asymmetry\_list = zip(\*sorted(zip(timesteps, asymmetry\_list)))  
  
 # Plot A(t) vs Timestep  
 plt.figure(figsize=(10, 6))  
 plt.plot(timesteps, asymmetry\_list, linestyle='-')  
 plt.xlabel('Timestep', fontsize=14)  
 plt.ylabel(r'$\mathcal{A}(t)$', fontsize=14)  
 plt.grid(True)  
 plt.tight\_layout()  
 plt.show()  
  
 # Optionally, return the data for further analysis  
 return timesteps, asymmetry\_list  
  
# Main script  
if \_\_name\_\_ == "\_\_main\_\_":  
 folder = "dumps" # Folder containing dump files  
 lx = 240.0 # Total system size in x-direction (2 \* lx in your LAMMPS script)  
 rho\_ld = 0.15 # Low-density phase value (from your context)  
 rho = 0.5 # Total system density (from your LAMMPS input)  
  
 # Option 1: Process all dump files and calculate average asymmetry  
 # Uncomment the following lines to perform this operation  
 # results = process\_dump\_files(folder, lx=lx, rho\_ld=rho\_ld, rho=rho)  
 # if results:  
 # plot\_asymmetry(results)  
 # else:  
 # print("No results to plot.")  
  
 # Option 2: Plot asymmetry vs time for a specific dump file  
 # Specify the filename  
 filename = os.path.join(folder, 'wet.241206.0213.eps.20.ly.120.dump') # Replace with your specific file  
 if os.path.exists(filename):  
 timesteps, asymmetry\_list = plot\_asymmetry\_vs\_time(filename, lx=lx, rho\_ld=rho\_ld, rho=rho)  
 else:  
 print(f"File {filename} does not exist.")

|  |
| --- |
| Figure 2.1: Asymmetry order parameter over time for = 20 |

# 3. Slurm Jobs

* **slurm-10797760**: It took 20 minutes to compute 2,500,000 steps.  
  → This means I need about 27 hours to compute 4 × 50,000,000 steps for four epsilons.
* **slurm-10802470**: Lost atoms in all epsilons.
* **slurm-10814916**: Cancelled, since I realised what was happening was:  
  ABPs interacted to form clusters before the wall was even there.  
  So, I commented out the following lines:
* # variable relaxation equal ceil(10\*${taur}/${dt})   
  # thermo\_style custom step time v\_tscaled pe density press   
  # thermo ${snapshot} # output thermo data every snapshot steps   
  # run ${relaxation}
* **slurm-10815371**: Lost atoms again in eps=20 even when dt = 0.00002\*${taur}.
* **slurm-10815451**: Lost atoms again for all epsilons even when dt = 0.00001\*${taur}…  
  I realised I didn’t include timestep ${dt}! Now it works perfectly.
* **slurm-10824222**: All of them were extremely successful! It took about 8 hours to compute 4 different potentials for 4 × 25,000,000 steps.  
  Removed OMPI error by adding export OMPI\_MCA\_mca\_base\_component\_show\_load\_errors=0.

Turci, Francesco, and Nigel B. Wilding. 2021. “Wetting Transition of Active Brownian Particles on a Thin Membrane.” *Physical Review Letters* 127 (23): 238002. <https://doi.org/10.1103/PhysRevLett.127.238002>.