Exercise Session 1

MDMC Spring 2023

Lorenzo Agosta, Virginia Carnevali, Simon Dürr, Sophia Johnson, Nikolaos Lempesis, Andrea Levy

March 14, 2023

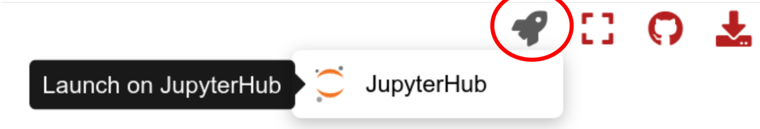
Exercise General Information

- Interviews
  - 10-15 minues discussion to test your understanding of the exercise
  - Good occasion to discuss your doubts and questions
- Report feedback
  - After your interview, you will get a detailed feedback via Moodle
    - No grade
    - Overall comment and detailed correction of the exercises

Exercises contribute to 1/2 of final grade! We count the best 5 out of the 6 reports for your exercise grade.

Notebooks reminder

- Always access the notebooks via the rocket button on the top right of the code files and choose JupyterHub to launch noto.epfl.ch
- **Make sure to access noto this way each time you begin the exercise to ensure you have the latest version!**



Exercise structure

🎯 Learning goals	📖 Chapter in script	📚 Resources
Follow a (re)introduction to statistical mechanics Learn differeces between canonical and microcanonical ensembles Understand how to calculate the Boltzmann distribution for a harmonic oscillator	Chapter 2 - Statistical Mechanics in a Nutshell	Understanding Molecular Simulation, Frenkel & Smit, p.9-17

Exercise 2 - Intro & Tips

Today we'll be building a tool to compute the probability distribution of the energy of a system using Boltzmann statistics. The focus of the exercise is to refresh your notions of statistical mechanics, which will be always the fundations for the simulatio tecniques we'll see during the course.

Tips!

- The theoretical part is about statistical thermodynamics. Be sure to know what we mean by
  - Thermodynamic ensembles
  - Microstate
  - Partition function and why its utility
  - Boltzmann distribution

Exercise 2 - Intro & Tips

- The practical part is about computing the Boltzmann distribution of a fictious harmonic oscillator
- Modify the code below to translate into code concepts from statistical mechanics

```python
# MODIFY HERE
# set number of energy levels and temperatures here
n_energy_levels= 0
reducedTemperatures=[0, 0, 0, 0]


def calculateStateOccupancy(T, i):
    # No degeneracy
    return 1  # MODIFY HERE

def calculateStateOccupancy_s1(T, i):
    # Degeneracy s+1
    return 1  # MODIFY HERE

def calculateStateOccupancy_s2(T, i):
    # Degeneracy s+2
    return 1  # MODIFY HERE


functions ={
    "no_degeneracy": calculateStateOccupancy,
    "s+1": calculateStateOccupancy_s1,
    "s+2": calculateStateOccupancy_s2
}

for f in functions.keys():
    fig, ax =plt.subplots(1)
    ax.set_title(f)
    ax.set_xlabel("Energy level")
    ax.set_ylabel("Occupancy")

    calculateOccupancy=functions[f]

    for reducedTemperature in reducedTemperatures:
        distribution = [] # For each state there is one entry
        partition_function=np.float64(0.0)

        for i in range(n_energy_levels):
            stateOccupancy=calculateOccupancy(reducedTemperature, i)
            distribution.append(1.0) # MODIFY HERE
            partition_function+=1.0  # MODIFY HERE
```

Exercise 2 - Intro & Tips

- Remember there is also Exercise 8 (often skipped in previous years!)

```
# INSERT YOUR WORKING CODE HERE
# ADAPT it for the linear rotor case
# defining a new calculateStateOccupancy_rotor function and a "linear rotor" key

# Then run the following cell (after having edited it where indicated!)
```

```
for f in functions.keys():
    fig, ax =plt.subplots(1)
    ax.set_title(f)
    ax.set_xlabel("Reduced Temperature")
    ax.set_ylabel("Partition Function")

    calculateOccupancy=functions[f]
    Z = []  # This variable will store the partition function
            # for different reduced temperatures

    for reducedTemperature in reducedTemperatures:
        distribution = [] # For each state there is one entry
        partition_function=np.float64(0.0)

        for i in range(n_energy_levels):
            stateOccupancy=calculateOccupancy(reducedTemperature, i)
            distribution.append(1.0) # MODIFY HERE
            partition_function+=1.0  # MODIFY HERE
        Z.append(1.0) # MODIFY HERE
    ax.plot(reducedTemperatures, Z, label="Z Exact")
    ax.plot(reducedTemperatures, Z, label="Z Approximated") # MODIFY HERE
                                                            # insert approximated Z

    ax.legend()
    plt.show()
```