3 Tonks gas *(4 points)*

Consider a one-dimensional gas of N particles of length a confined to a strip of length L. The particles cannot overlap with each other (hard core repulsion) and otherwise do not interact with each other (no attraction like in the van der Waals gas).

1. Calculate the canonical partition sum Z by integrating over all possible values for the midpoints of the gas particles. ***(2 points)***

The canonical partition sum Z is as usual a sum over all configurations in phase space:

$$\begin{aligned} Z &= \frac{1}{N!}\frac{1}{h^N}\int \mathrm{d}^N p \int \mathrm{d}^N x\, e^{-\beta\left(\frac{p^2}{2m}+U(x)\right)} \\ &= \frac{1}{N!}\left(\frac{1}{h}\int \mathrm{d}p\, e^{-\beta\frac{p^2}{2m}}\right)^N \int \mathrm{d}^N x\, e^{-\beta U(x)} \\ &= \frac{1}{N!\lambda^N}\int \mathrm{d}x_1 \dots \mathrm{d}x_N\, e^{-\beta U(x)}\,. \end{aligned} \tag{1}$$

Here, we write λ for the thermal wavelength as usual and $U(x)$ for the potential of the system which depends on the midpoints $x_1, ..., x_N$ of the particles. We see that the partition sum reduces to an integral over all possible values for the midpoints of the gas particles. For hard core repulsion, we know that the weight $e^{-\beta U(x)}$ is either 0 (if any two particles overlap which is not allowed) or 1 (if no particles overlap). Thus, we can reduce the integral to the case where $x_1 < x_2 < ... < x_N$ and multiply by $N!$ to include all possibilities. Clearly, x_i must then lie between $x_{i-1} + a$ and $L - (N-i)a - a/2$. With successive integration we get

$$\begin{aligned} Z &= \frac{1}{\lambda^N}\int_{a/2}^{L-(N-1)a-a/2} \mathrm{d}x_1 \int_{x_1+a}^{L-(N-2)a-a/2} \mathrm{d}x_2 \dots \int_{x_{N-1}+a}^{L-a/2} \mathrm{d}x_N \\ &= \frac{1}{\lambda^N}\int_{a/2}^{L-(N-1)a-a/2} \mathrm{d}x_1 \int_{x_1+a}^{L-(N-2)a-a/2} \mathrm{d}x_2 \dots \int_{x_{N-2}+a}^{L-a-a/2} \mathrm{d}x_{N-1}\, (L - 3a/2 - x_{N-1}) \\ &= \frac{1}{2\lambda^N}\int_{a/2}^{L-(N-1)a-a/2} \mathrm{d}x_1 \int_{x_1+a}^{L-(N-2)a-a/2} \mathrm{d}x_2 \dots \int_{x_{N-3}+a}^{L-2a-a/2} \mathrm{d}x_{N-2}\, (L - 5a/2 - x_{N-2})^2 \\ &= \dots \\ &= \frac{1}{i!\lambda^N}\int_{a/2}^{L-(N-1)a-a/2} \mathrm{d}x_1 \int_{x_1+a}^{L-(N-2)a-a/2} \mathrm{d}x_2 \dots \int_{x_{N-i-1}+a}^{L-ia-a/2} \mathrm{d}x_{N-i}\, (L - (2i+1)a/2 - x_{N-i})^i \\ &= \dots \\ &= \frac{1}{(N-1)!\lambda^N}\int_{a/2}^{L-(N-1)a-a/2} \mathrm{d}x_1\, (L - (2N-1)a/2 - x_1)^{N-1} \\ &= \frac{1}{N!\lambda^N}(L - Na)^N\,. \end{aligned} \tag{2}$$

This makes also sense from a physical point of view: It's like we have N independent particles which can move freely in a one-dimensional volume $(L - Na)$.

2. Calculate the free energy $F = -k_BT \ln Z$ and the pressure $p = -\partial_L F$. Evaluate the virial coefficient B_2 from the appropriate Mayer function and show that your result agrees with the virial expansion based on the exact solution. ***(2 points)***

The free energy is

$$F = -k_BT \left(N \ln \left(\frac{L - Na}{\lambda} \right) - \ln (N!) \right) . \tag{3}$$

The pressure is

$$p = \frac{Nk_BT}{L - Na} . \tag{4}$$

The virial coefficient B_2 for one-dimensional hard particles with radius $a/2$ is

$$B_2 = -\frac{1}{2} \int_0^\infty \mathrm{d}r \left(e^{-\beta U(r)} - 1 \right) = \frac{1}{2} \underline{\int_0^{a/2}} \mathrm{d}r = \frac{a}{4} , \tag{5}$$

since $e^{-\beta U(r)}$ can be either 0 or 1 as explained above.

4 Computer exercise: Ising model *(10 bonus points)*

Consider a two-dimensional lattice of $N \times N$ spins. Every spin can take values $s_{i,j} = \pm 1$ and the system is governed by the Hamiltonian

$$\mathcal{H} = -J \cdot \sum_{NN} s_{i,j} s_{k,l} \tag{6}$$

with J being the interaction strength. The sum is only over nearest neighbors (NN), meaning that the spin $s_{i,j}$ at $x = i, y = j$ interacts only with spins at $s_{i\pm 1,j}$ and $s_{i,j\pm 1}$. Assume periodic boundary conditions in both directions, meaning that, for instance, spin $s_{0,j}$ interacts with $s_{1,j}$ and with $s_{N-1,j}$ on "the other side". In the following, put $J = 1$ and $k_B = 1$ for simplicity.

1. Write a code (and submit it along with your results) that implements the following algorithm (called "importance sampling with the Metropolis algorithm")

- Initialize the spins with $s = \pm 1$ randomly chosen in an $N \times N$ array.

```python
def initialize_spins(N):

    grid = np.ndarray((N, N))
    for i in range(N):
        for j in range(N):
            grid[i][j] = random.choice([-1, +1])

    return grid
```

- Pick a spin (i, j) at random. Calculate the energy change δE upon flipping only this spin (i, j). If $\delta E < 0$, accept the spin flip. If $\delta E > 0$, accept the flip with Boltzmann probability $\exp(-\beta \delta E)$, otherwise reject.

```python
def flip_random_spin(grid, T):

    def boltzmann_prob(dE, T):
        beta = 1 / (k_B * T)
        return np.exp(-beta * dE)

    # choose random grid cell
    i, j = random.choice(range(N)), random.choice(range(N))
    dE = get_flip_energy(grid, i, j)

    flip = False
    if dE < 0:  # flip spin
        flip = True
    if dE >= 0:  # flip only with Boltzmann probability
        if random.uniform(0, 1) < boltzmann_prob(dE, T):
            flip = True

    if flip:
        grid[i][j] *= -1
    return grid
```

Here, the flip energy is given by

```python
def get_flip_energy(grid, i, j):

    def apply_periodic_bounds(cell_idx, N):
        if cell_idx >= N:
            cell_idx -= N
        elif cell_idx < 0:
            cell_idx += N
        return cell_idx

    dE = 0  # calculate energy difference after flip
    for di in [-1, 0, 1]:  # loop over neighbors
        for dj in [-1, 0,  1]:
            if di == dj == 0:
                continue  # no self-interaction
            # get row/col index for neighbor
            i_neighbor = apply_periodic_bounds(i+di, N)
            j_neighbor = apply_periodic_bounds(j+dj, N)
            # get spin of neighbor
            s_neighbor = grid[i_neighbor][j_neighbor]
            # subtract current state's energy, add new state's energy
            dE -= -J * s_neighbor * grid[i][j]
            dE += -J * s_neighbor * (-grid[i][j])

    return dE
```

- After every N^2 of such spin "tests", evaluate the mean magnetization $\langle M\rangle = \frac{1}{N^2}\sum_{i,j} s_{i,j}$ (note that the sum here is over all spins)

  Get the magnetization for a given grid:

```python
def get_magnetization(grid):
    N = grid.shape[0]

    magnetization = 0
    for i in range(N):  # sum over all spins
        for j in range(N):
            magnetization += grid[i][j]

    return magnetization / N**2
```

Calculate mean magnetization after every N^2 "tests", do this for different temperatures and plot.

```python
def main():

    temperatures = [1.5, 3]
    nr_of_runs = 300

    magnetizations = []
    for T in temperatures:
        grid = initialize_spins(N)

        magnetization = []
        for run_idx in tqdm(range(nr_of_runs)):
            # flip random spins
            for _ in range(N**2):
                grid = flip_random_spin(grid, T)
            # get magnetization
            magnetization.append(get_magnetization(grid))
            # plot spins in grid
            if run_idx in [0, 20, 50]:
                plot_grid(grid, T, run_idx)

        magnetizations.append(magnetization)

    plot_magnetizations(magnetizations, temperatures)
```

Plotting is done with this function:

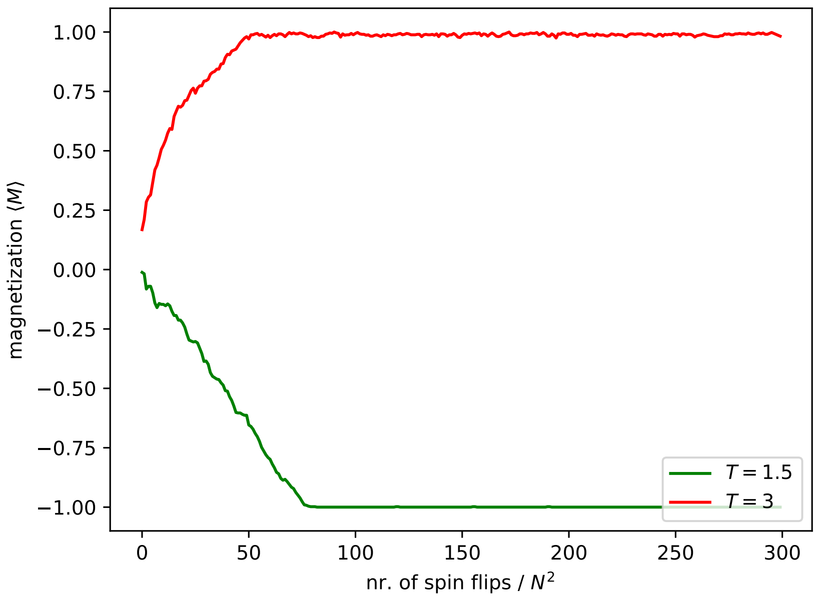
```python
def plot_magnetizations(magnetizations, temperatures):

    def get_color_gradient(from_color, to_color, nr_of_samples):
        gradient = Color(from_color).range_to(Color(to_color), nr_of_samples)
        colors = [c.rgb for c in gradient]
        return colors

    colors = get_color_gradient('green', 'red', len(temperatures))
    for idx, magnetization in enumerate(magnetizations):
        T = temperatures[idx]
        plt.plot(
            magnetization,
            label=f'$T={T}$',
            color=colors[idx],
        )

    plt.xlabel('nr. of spin flips / $N^2$')
    plt.ylabel(r'magnetization $\langle M\rangle$')
    plt.legend(loc='lower right')
    plt.savefig('../figures/magnetization_vs_time.pdf')
    plt.close()


def plot_grid(grid, T, idx):
    plt.imshow(grid, cmap='gray')
    plt.savefig(f'../figures/grid_{T}_{idx}.pdf')
    plt.close()
```

- Why does this simple method sample phase space (rather) efficiently?

Choosing spins at random is an unbiased way of doing the calculation, since all grid cells are equally important.

2. Study the system numerically for $N = 32$ or higher and for the two temperatures $T = 3$ and $T = 1.5$ (note again that $J = 1 = k_B$, hence T is the only parameter). Continue running the algorithm, until $\langle M \rangle$ does not change anymore except for small fluctuations around a constant value. Discuss your results.

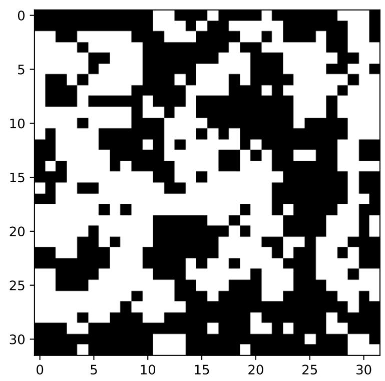
The code above leads to the creation of the following plot.

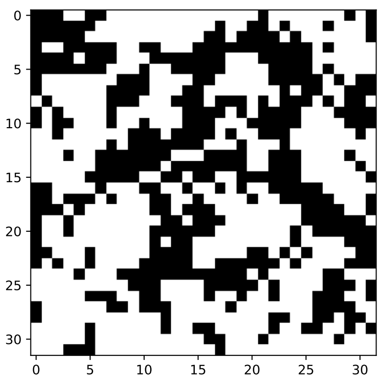


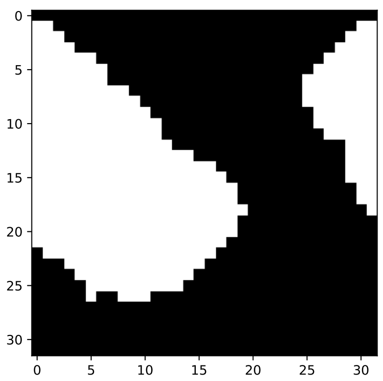
From this we can verify a few expectations.

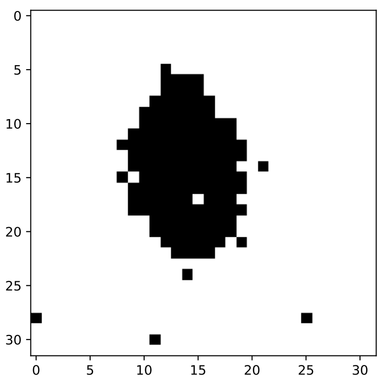
1. The state of least energy is one where all spins align. The system tends toward this state over time.
2. Whether all spins align to be $+1$ or -1 is arbitrary and can change from run to run, since it depends both on the random initialization as well as the randomness in the flip probability calculations.
3. For higher temperatures, spins might flip with a non-zero probability even if the energy change is positive. For this reason, one can see in the plot that the system at $T = 3$ approximates the low-energy state a bit more slowly than the system at $T = 1.5$ does.

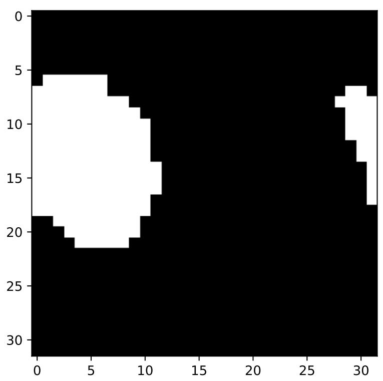
Visualization of the grid's state at various times.

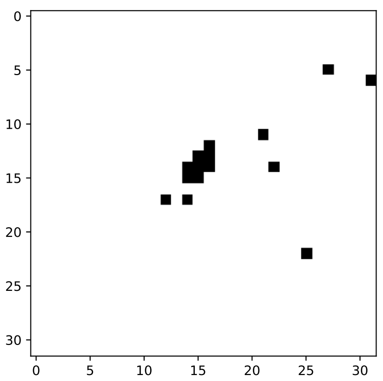

(a) spins for $T = 1.5$ after initialization


(b) spins for $T = 1.5$ after initialization


(c) spins for $T = 1.5$ after 20 loops over the grid


(d) spins for $T = 3$ after 20 loops over the grid


(e) spins for $T = 1.5$ after 50 loops over the grid


(f) spins for $T = 3$ after 50 loops over the grid