Submission Phase

1. Do assignment ✓ (/smac-001/human_grading/view/courses/971628/assessments/5/submissions)

Evaluation Phase

2. Evaluate peers ✓ (/smac-001/human_grading/view/courses/971628/assessments/5/peerGradingSets)

Results Phase

3. See results ✓ (/smac-001/human_grading/view/courses/971628/assessments/5/results/mine)

Your effective grade is 18

Your unadjusted grade is 18, which is simply the grade you received from your peers.

See below for details.

Introduction

The week 2 videos of **Statistical Mechanics: Algorithms and Computations** study three algorithms for the simulation of hard disks in a box:

- event_disks_box.py (Event-driven molecular dynamics)
- direct_disks_box.py (Direct sampling)
- markov_disks_box.py (Markov-chain sampling)

In Homework Session 2, we will check that the statistical properties of these algorithms are all the same. We will then interpret the outcome. In a final step, we shall explicitly verify the equiprobability principle for hard disks in a box.

NB: We strongly rely on the algorithms of week 2, but provide some help, to facilitate the graphics output. Also, we provide a "code snippet" (a piece of code) for the event-driven molecular dynamics algorithm, in order to make things easier.

(A)

In this section, we consider four disks in a square box of edge length 1, in a box without periodic boundary conditions. We set the density equal to η = 0.18, which corresponds to a disk radius σ = 0.1196.

We consider a simple observable, the position x: the x-coordinate of the center of a disk. We will compute its probability distribution, as the normed histogram of x-positions. This histogram is the same for all disks, so we can collect data for one disk or for all of them.

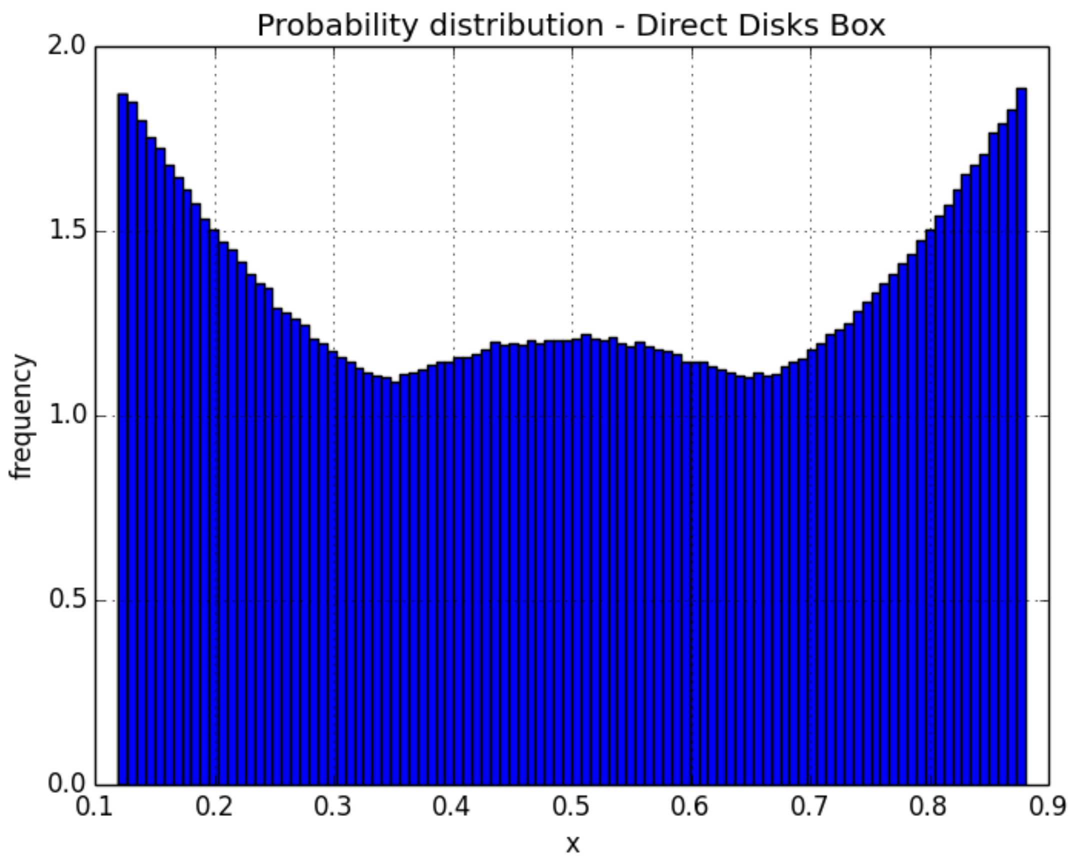
(A1)
Compute the histogram of the x-positions for the direct-sampling Monte Carlo algorithm by modifying the algorithm direct_disks_box_multirun.py. To avoid problems, simply cut and paste the below program into a file, and produce a histogram of the x-positions with 100 bins as output.

```
import random, pylab

def direct_disks_box(N, sigma):
    overlap = True
    while overlap == True:
        L = [(random.uniform(sigma, 1.0 - sigma), random.uniform(sigma, 1.0 - sigma))]
        for k in range(1, N):
            a = (random.uniform(sigma, 1.0 - sigma), random.uniform(sigma, 1.0 - sigma))
            min_dist_sq = min(((a[0] - b[0]) ** 2 + (a[1] - b[1]) ** 2) for b in L)
            if min_dist_sq < 4.0 * sigma ** 2:
                overlap = True
                break
            else:
                overlap = False
                L.append(a)
    return L

N = 4
sigma = 0.1197
n_runs = 1000000
histo_data = []
for run in range(n_runs):
    pos = direct_disks_box(N, sigma)
    for k in range(N): histo_data.append(pos[k][0])
pylab.hist(histo_data, bins=100, normed=True)
pylab.xlabel('x')
pylab.ylabel('frequency')
pylab.title('please fill in a title giving type of program and parameters')
pylab.grid()
pylab.savefig('direct_disks_histo.png')
pylab.show()
```

Before you upload the histogram in the below box, please make sure to give it an appropriate title saying which program was used in order to produce it and that the axes are correctly labelled. You may provide a short optional comment about the result, but it will not be graded: How do you reconcile the histogram output with the equiprobability principle?



Evaluation/feedback on the above work

Note: this section can only be filled out during the evaluation phase.

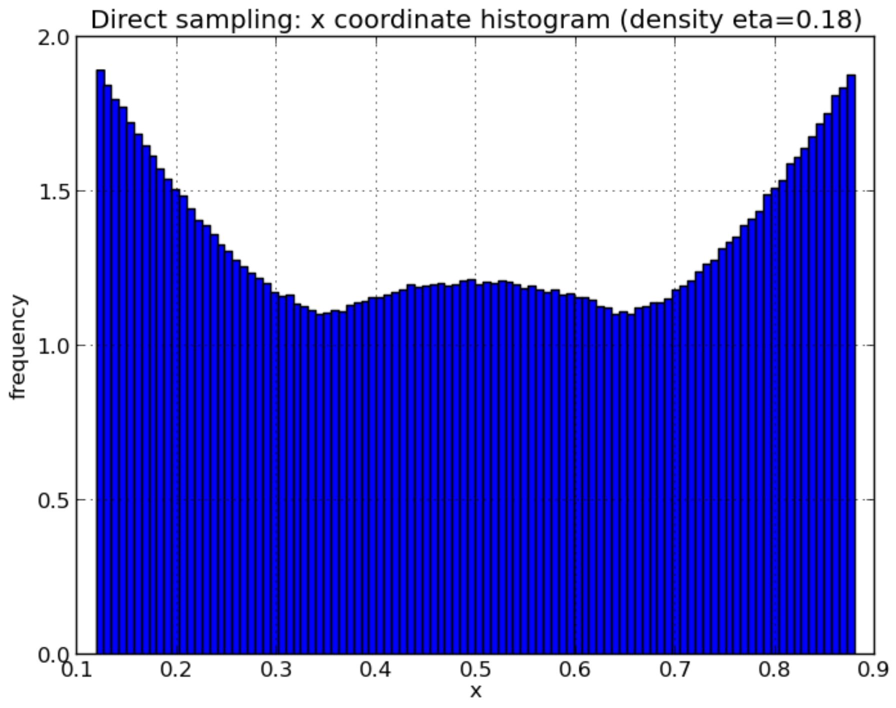
Here, you are asked to evaluate a scientific diagram. You can give a total of 2 points.

Give **1 point** if the diagram meets the following criteria of graphical quality (which will be the general criteria for the quality of diagrams in this course):

- The diagram shows exactly the quantities that were asked for (here: the histogram of positions over the x-coordinate)
- A title clearly indicates the program used to produce the output, for example "Direct sampling: x coordinate histogram (density eta=0.18)"
- All axes are correctly labelled (here: "Histogram of positions" and "x". Equivalent labels like "Histogram" or "Frequency" instead of "Histogram of positions" or "x-coordinate" for "x are also acceptable.)
- All axes have the correct scaling (here: both x-axis and y-axis are linear)
- The diagram shows at least the range of data points that was asked for (here: 100 bins.)

Give **1 point** if the diagram represents the correct results: the plot will show a histogram with values increasing towards the boundaries and a rounded bump in the center (see example below).

If the plot meets the criteria of graphical quality and shows the correct results, the student earned a total of 2 points. An example that would receive the full score is shown here:



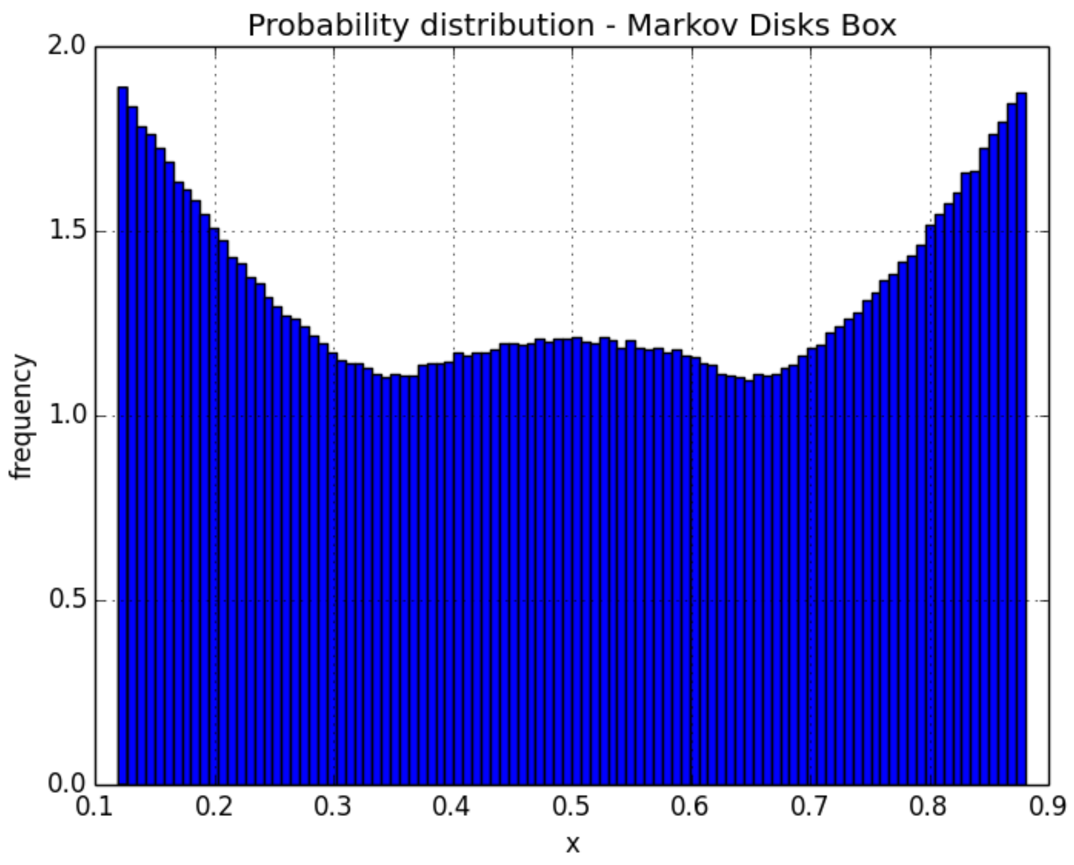
Score from your peers: **2**

(A2)

(A2.1)

Compute the histogram of the x-positions with 100 bins for the Markov-chain sampling algorithm, by modifying the algorithm markov_disks_box.py analogously to the above point A1. Set n_steps=2000000. As usual, make sure that there is an appropriate title saying which program was used to generate the histogram and that the axes are correctly labelled. Upload the histogram using the upload box below. Cut-and-paste your modified markov_disks_box.py program into the text box.

ATTENTION: This part is more subtle than it seems. You MUST record the x-positions at the right place (remember the story about the piles in the pebble game in Lecture 1!). To make sure you get the story right, compare your result to your result of A1, and don't hesitate to vary delta and to increase n_steps for high-quality comparisons.



To limit the computation time, n_steps has been set to 200. n_runs has been set to 1000000 (same as A1) to ensure good sampling of the histogram.

```
import random, pylab

def markov_disks_box(L, sigma):
    for steps in range(n_steps):
        a = random.choice(L)
        b = [a[0] + random.uniform(-delta, delta), a[1] + random.uniform(-delta, delta)
]
        min_dist = min((b[0] - c[0]) ** 2 + (b[1] - c[1]) ** 2 for c in L if c != a)
        box_cond = min(b[0], b[1]) < sigma or max(b[0], b[1]) > 1.0 - sigma
        if not (box_cond or min_dist < 4.0 * sigma ** 2):
            a[:] = b
    return L

L = [[0.25, 0.25], [0.75, 0.25], [0.25, 0.75], [0.75, 0.75]]
sigma = 0.1197
delta = 0.1
N = 4
n_steps = 200

n_runs = 1000000
histo_data = []
for run in range(n_runs):
    pos = markov_disks_box(L, sigma)
    for k in range(N): histo_data.append(pos[k][0])

pylab.hist(histo_data, bins=100, normed=True)
pylab.xlabel('x')
pylab.ylabel('frequency')
pylab.title('Probability distribution - Markov Disks Box')
pylab.grid()
pylab.savefig('markov_disks_histo.png')
pylab.show()
```

Evaluation/feedback on the above work

Note: this section can only be filled out during the evaluation phase.

Here, you are asked to evaluate a scientific diagram and to check the python code that generated the diagram. You can give a total of **2 points.**

Give **1 point** if the diagram meets the following criteria of graphical quality (which will be the general criteria for the quality of diagrams in this course):

- The diagram shows exactly the quantities that were asked for (here: the histogram of positions over the x-coordinate)
- A title clearly indicates the program used to produce the output, for example "Markov-chain sampling: x coordinate histogram (density eta=0.18)"
- All axes are correctly labelled (here: "Histogram of positions" and "x". Equivalent

labels like "Histogram" or "Frequency" instead of "Histogram of positions" or "x-coordinate" for "x are also acceptable.)
- All axes have the correct scaling (here: both x-axis and y-axis are linear)
- The diagram shows at least the range of data points that was asked for (here: 100 bins.)

Give **1 point** if the diagram represents the correct results and was generated by a correctly programmed python code. The correct plot will show a histogram with values increasing towards the boundaries and a rounded bump in the center (see example below).

Make sure that the line "for k in range(4): histo_data.append(L[k][0])". or equivalent is not inside the if:

Correct solution:

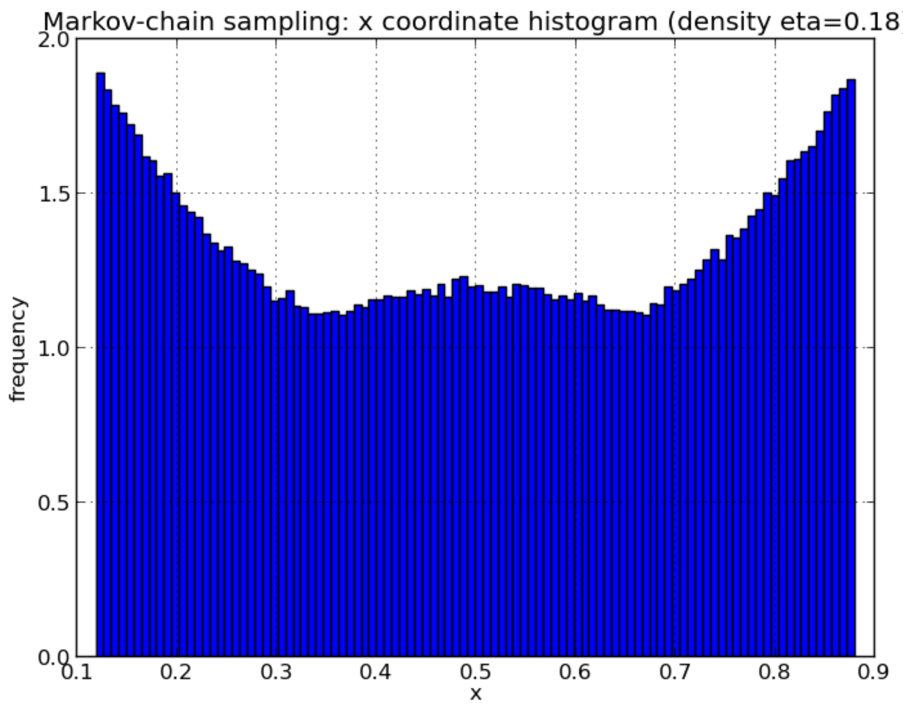
```
    if not (box_cond or min_dist_sq < 4.0 * sigma ** 2):
        a[:] = b
    for k in range(4): histo_data.append(L[k][0])
```

Incorrect solution:

```
    if not (box_cond or min_dist_sq < 4.0 * sigma ** 2):
        a[:] = b
        for k in range(4): histo_data.append(L[k][0])
```

ATTENTION: If the *incorrect* program has been used, the result will look qualitatively the same, but the values at the boundary will go up to higher
values like 2.0 or even slightly greater.

If the plot meets the criteria of graphical quality, shows the correct results generated by a correctly written program, the student earned a total of
2 points. An example plot that would receive the full score is shown here:



Score from your peers: **2**

(A2.2)
State the *three mathematical conditions* that guarantee that the histogram for the Markov-chain sampling algorithm agrees with the direct-sampling one (hint: these three conditions were discussed in tutorial 1 by Vivien, and again by Werner at the end of Lecture 2). Furthermore, explain briefly whether and why these three conditions are in fact satisfied by the four-disk system at density η = 0.18.

1. Reversibility - Detailed balance, $P^{eq}_{o} w_{o \to n} = P^{eq}_{n} w_{n \to o}$
2. Irreducibility - Regardless of the present configuration, any other configuration can be reached in finite time.
3. Aperiodicity - The same configuration is not always visited after a given period of time.

The Markov-chain sampling algorithm (due to use of *Tabula Rasa* method) samples all configurations with the same probability, thus satisfying condition 1. The density of 0.18 is not too high, so, the disks can move freely thereby satisfying condition 2. The algorithm is aperiodic by construction as the configurations evolve via a random walk.

Evaluation/feedback on the above work

Note: this section can only be filled out during the evaluation phase.

Here, you are asked to evaluate the correctness of the students' answer. You can give a maximum of **2 points.**

If the student can list the three conditions that guarantee the equivalence of the

Markov-chain sampling and the direct-sampling algorithm, i.e.

- Detailed balance (or global balance) (the student needs to list only one of the two)
- Aperiodicity
- Irreducibility

the student has earned **1 point.**

If the student furthermore knows that these conditions are in fact satisfied in the case of the four-disk system at density η = 0.18, the student earned **1 point**. At radius 0.12, we are clearly not in the situation shown by Werner in CM2 31:41, so that disks can easily pass around each other.

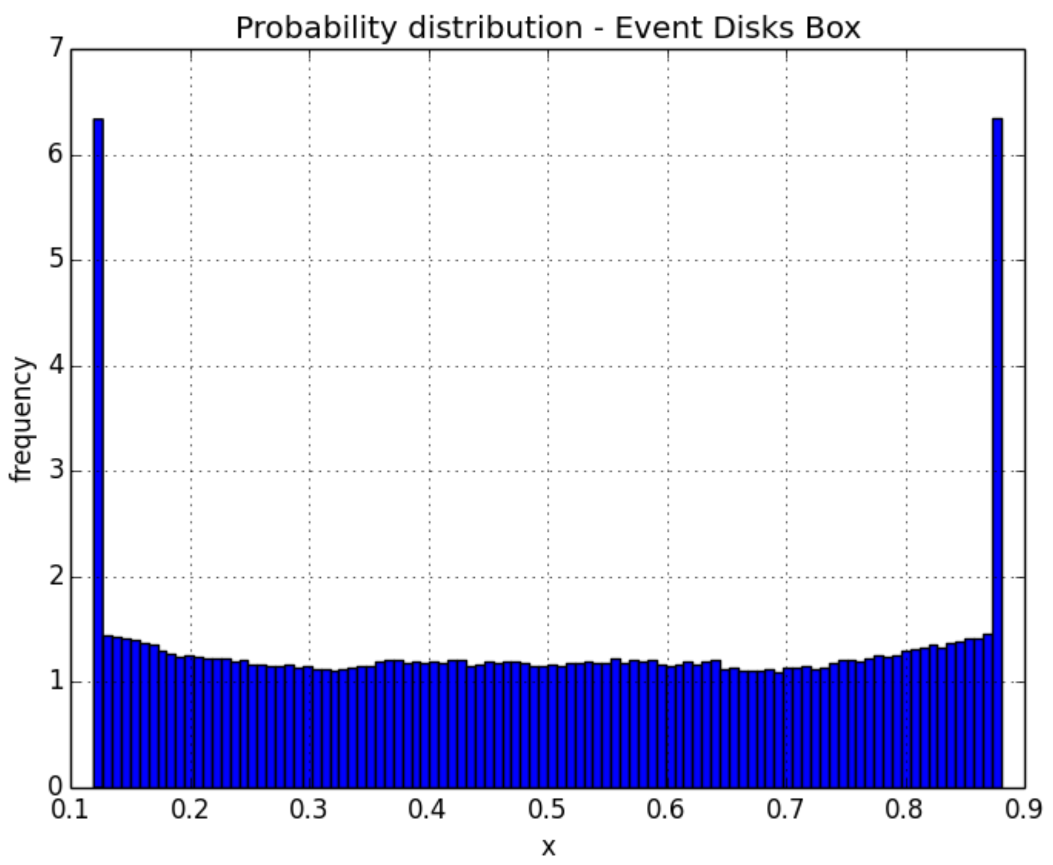
The following example answer would receive the full score:

"The three mathematical conditions are detailed balance, aperiodicity and irreducibility. Yes, we can actually prove that the three conditions are satisfied, as the density is low enough that there is no problem to for one disk to pass around the other."

Score from your peers: **2**

(A3)
We now consider the event_driven Molecular dynamics algorithm. In (A3.1), (A3.2), and (A3.3) always start from the algorithm event_disks_box.py, and NOT FROM event_disks_box_movie.py, which is too complicated for our task.

(A3.1.1)
Compute the histogram for the x-positions with 100 bins analogously to what you did in (A1) and (A2.1), but for the event-driven molecular dynamics algorithm **COLLECT THE HISTOGRAM FROM ALL PARTICLES AT ALL COLLISION TIMES** (wall **AND** pair collisions) and a total of n_events=200000 events. Make sure that there is an appropriate title that indicates the program used to generate the diagram and that the axes are labelled. Upload the diagram using the upload box below. Cut-and-paste your modified event-driven molecular dynamics program into the text box.


Probability distribution - Event Disks Box
frequency
x
0
1
2
3
4
5
6
7
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9

```
import math, pylab

def wall_time(pos_a, vel_a, sigma):
        if vel_a > 0.0:
                del_t = (1.0 - sigma - pos_a) / vel_a
        elif vel_a < 0.0:
                del_t = (pos_a - sigma) / abs(vel_a)
        else:
                del_t = float('inf')
        return del_t

def pair_time(pos_a, vel_a, pos_b, vel_b, sigma):
        del_x = [pos_b[0] - pos_a[0], pos_b[1] - pos_a[1]]
        del_x_sq = del_x[0] ** 2 + del_x[1] ** 2
        del_v = [vel_b[0] - vel_a[0], vel_b[1] - vel_a[1]]
        del_v_sq = del_v[0] ** 2 + del_v[1] ** 2
        scal = del_v[0] * del_x[0] + del_v[1] * del_x[1]
        Upsilon = scal ** 2 - del_v_sq * ( del_x_sq - 4.0 * sigma **2)
        if Upsilon > 0.0 and scal < 0.0:
                del_t = - (scal + math.sqrt(Upsilon)) / del_v_sq
        else:
                del_t = float('inf')
        return del_t

pos = [[0.25, 0.25], [0.75, 0.25], [0.25, 0.75], [0.75, 0.75]]
vel = [[0.21, 0.12], [0.71, 0.18], [-0.23, -0.79], [0.78, 0.1177]]
singles = [(0, 0), (0, 1), (1, 0), (1, 1), (2, 0), (2, 1), (3, 0), (3, 1)]
pairs = [(0, 1), (0, 2), (0, 3), (1, 2), (1, 3), (2, 3)]
sigma = 0.1197
t = 0.0
n_events = 200000

histo_data = []
N = 4

for event in range(n_events):
        wall_times = [wall_time(pos[k][l], vel[k][l], sigma) for k, l  in singles]
        pair_times = [pair_time(pos[k], vel[k], pos[l], vel[l], sigma) for k, l in pair
s]
        next_event = min(wall_times + pair_times)
        t += next_event
        for k, l in singles: pos[k][l] += vel[k][l] * next_event

        for k in range(N): histo_data.append(pos[k][0])

        if min(wall_times) < min(pair_times):
                collision_disk, direction = singles[wall_times.index(next_event)]
                vel[collision_disk][direction] *= -1.0
        else:
                a, b = pairs[pair_times.index(next_event)]
                del_x = [pos[b][0] - pos[a][0], pos[b][1] - pos[a][1]]
                abs_x = math.sqrt(del_x[0] ** 2 + del_x[1] ** 2)
                e_perp = [c / abs_x for c in del_x]
                del_v = [vel[b][0] - vel[a][0], vel[b][1] - vel[a][1]]
                scal = del_v[0] * e_perp[0] + del_v[1] * e_perp[1]

                for k in range(2):
                        vel[a][k] += e_perp[k] * scal
                        vel[b][k] -= e_perp[k] * scal

        print 'event',event
        print 'time',t
        print 'pos',pos
        print 'vel',vel


pylab.hist(histo_data, bins=100, normed=True)
pylab.xlabel('x')
pylab.ylabel('frequency')
pylab.title('Probability distribution - Event Disks Box')
pylab.grid()
pylab.savefig('event_disks_histo.png')
pylab.show()
```

Evaluation/feedback on the above work

Note: this section can only be filled out during the evaluation phase.

Here, you are asked to evaluate a scientific diagram and to check the python code that generated the diagram. You can give a total of **2 points.**

Give **1 point** if the diagram meets the following criteria of graphical quality (which will be the general criteria for the quality of diagrams in this course):

- The diagram shows exactly the quantities that were asked for (here: the histogram of positions over the x-coordinate)
- A title clearly indicates the program used to produce the output, for example "Event-driven (all collisions): x coordinate histogram (density eta=0.18)"
- All axes are correctly labelled (here: "Histogram of positions" and "x". Equivalent labels like "Histogram" or "Frequency" instead of "Histogram of
- positions" or "x-coordinate" for "x are also acceptable.)
- All axes have the correct scaling (here: both x-axis and y-axis are linear)
- The diagram shows at least the range of data points that was asked for (here: 100 bins.)

Give **1 point** if the diagram represents the correct results and was generated by a correctly programmed python code. The correct plot will show a histogram with values slowly increasing towards the boundaries, a rounded bump in the center and two very strong peaks at the left and right boundaries (see example below).

A correct program records the x-positions in one of the following 3 ways:

Version 1:

```
    for k, l in singles: pos[k][l] += vel[k][l] * next_event
    for k in range(4): histo_data.append(pos[k][0])  # <==========
```

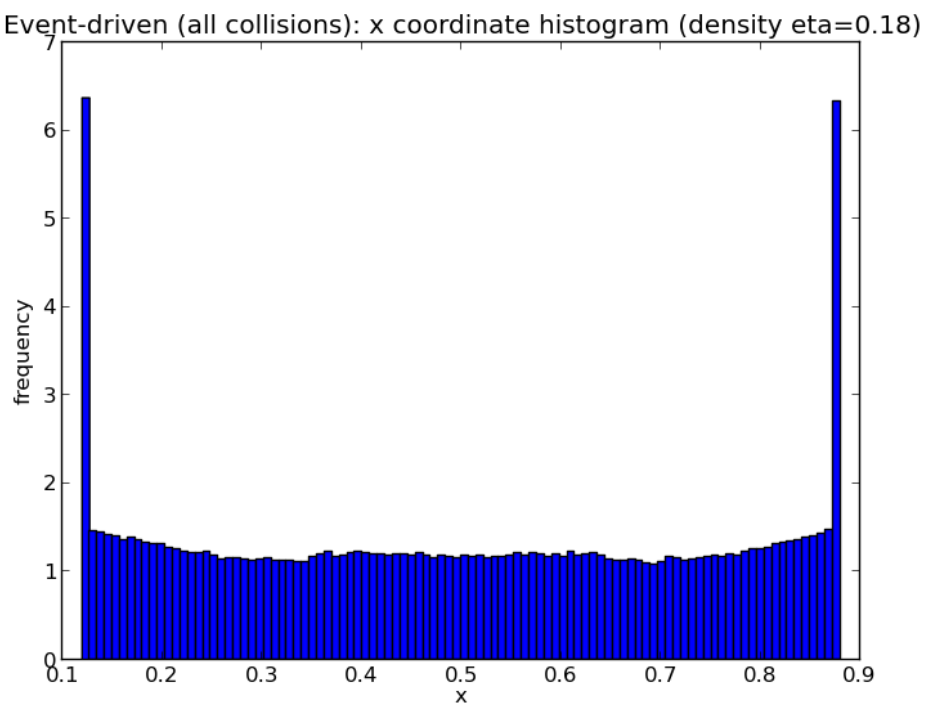
or version 2:

```
        for k in range(2):
            vel[a][k] += e_perp[k] * scal
            vel[b][k] -= e_perp[k] * scal
    for k in range(4): histo_data.append(pos[k][0])  # <===========
```

or version 3:

```
    if min(wall_times) < min(pair_times):
        for k in range(4): histo_data.append(pos[k][0])  # <==========
        for k, l in singles: pos[k][l] += vel[k][l] * next_event
        collision_disk, direction = singles[wall_times.index(next_event)
]
        vel[collision_disk][direction] *= -1.0
    else:
        for k in range(4): histo_data.append(pos[k][0])  # <==========
        for k, l in singles: pos[k][l] += vel[k][l] * next_event
```

If the plot meets the criteria of graphical quality, shows the correct results generated by a correctly written program, the student earned a total of **2 points.** An example plot that would receive the full score is shown here:



Score from your peers: **2**

(A3.1.2)
This histogram differs from the ones in (A1) and (A2.1). Briefly explain its outstanding feature: the large probability at x = σ and x = 1 - σ.

In **A1** and **A2.1**, the wall collisions are ignored which are being considered in **A3.1.1**. After the collision, the disk tends to remain near the wall edges. Due to this, there is increased probability of the disk being at x = σ and x = 1 - σ.

Evaluation/feedback on the above work

Note: this section can only be filled out during the evaluation phase.

Here, you have to check whether the student can correctly explain the outstanding feature of the histogram produced in A3.1.1.

The modified program records the x-coordinates of the four disk centers, whenever a collision, either between a disk and a wall or between two disks, occurs. Every time a disk collides with the left or the right wall its center is at x=σ or x=1−σ, respectively, no matter at which y-coordinate the collision happens. Therefore, center positions at the left and right walls are much more frequent than in the bulk of the box, where wall collisions occur only at y=σ and y=1−σ.

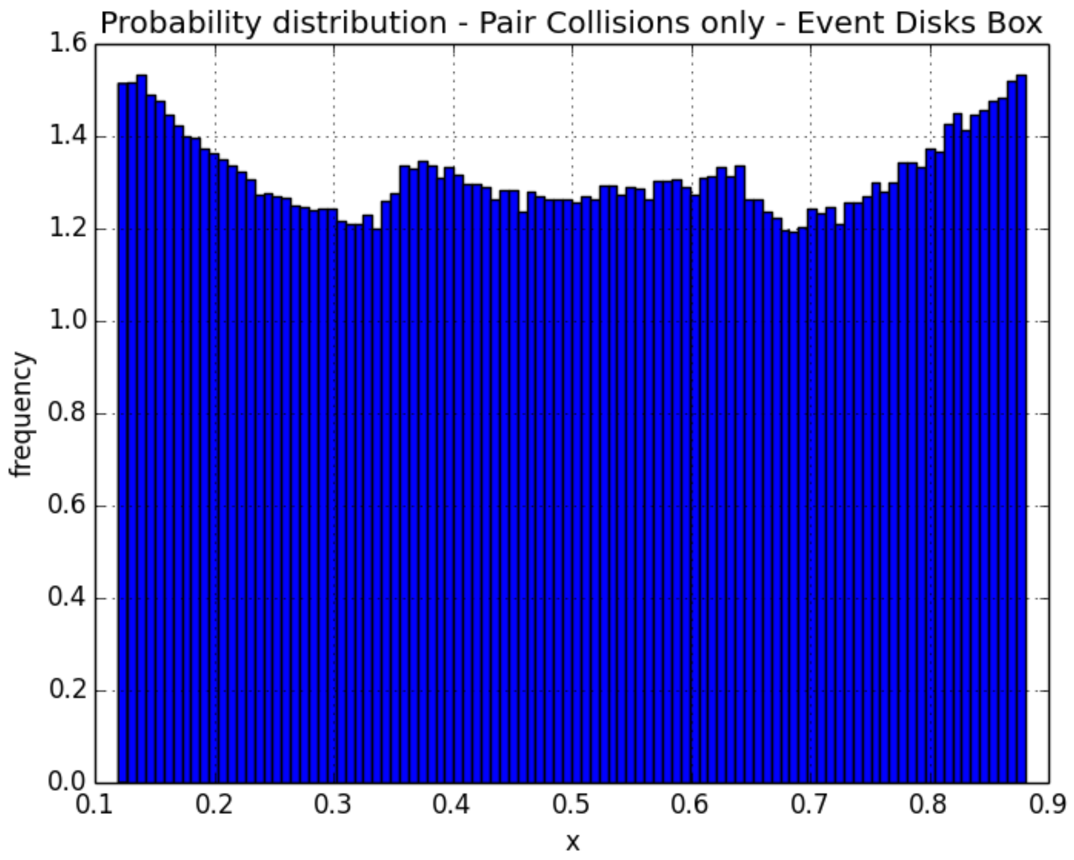
The student earned **1 points** if this is correctly explained in the student's answer.

For example, the following answers would receive the full score:

"The peaks at the left and the right of the histogram are due to wall collisions."

Score from your peers: **0.5**

(A3.2)
You may think that the difference between Monte Carlo and Molecular dynamics comes from taking into account wall collisions. To test this idea, compute the histogram of the x-positions for the event-driven molecular dynamics algorithm, but only at ALL PAIR COLLISION TIMES (that
is, drop the wall collisions from (A3.1.1)) for n_events = 500000. As usual, make sure that there is a title indicating the program used and that
the axes are labelled. Upload the histogram using the upload box below. Cut-and-paste your modified event-driven molecular dynamics program into the text box.


Probability distribution - Pair Collisions only - Event Disks Box
frequency
x
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9

```
import math, pylab

def wall_time(pos_a, vel_a, sigma):
        if vel_a > 0.0:
                del_t = (1.0 - sigma - pos_a) / vel_a
        elif vel_a < 0.0:
                del_t = (pos_a - sigma) / abs(vel_a)
        else:
                del_t = float('inf')
        return del_t

def pair_time(pos_a, vel_a, pos_b, vel_b, sigma):
        del_x = [pos_b[0] - pos_a[0], pos_b[1] - pos_a[1]]
        del_x_sq = del_x[0] ** 2 + del_x[1] ** 2
        del_v = [vel_b[0] - vel_a[0], vel_b[1] - vel_a[1]]
        del_v_sq = del_v[0] ** 2 + del_v[1] ** 2
        scal = del_v[0] * del_x[0] + del_v[1] * del_x[1]
        Upsilon = scal ** 2 - del_v_sq * ( del_x_sq - 4.0 * sigma **2)
        if Upsilon > 0.0 and scal < 0.0:
                del_t = - (scal + math.sqrt(Upsilon)) / del_v_sq
        else:
                del_t = float('inf')
        return del_t

pos = [[0.25, 0.25], [0.75, 0.25], [0.25, 0.75], [0.75, 0.75]]
vel = [[0.21, 0.12], [0.71, 0.18], [-0.23, -0.79], [0.78, 0.1177]]
singles = [(0, 0), (0, 1), (1, 0), (1, 1), (2, 0), (2, 1), (3, 0), (3, 1)]
pairs = [(0, 1), (0, 2), (0, 3), (1, 2), (1, 3), (2, 3)]
sigma = 0.1197
t = 0.0
n_events = 500000

histo_data = []
N = 4

for event in range(n_events):
        wall_times = [wall_time(pos[k][l], vel[k][l], sigma) for k, l  in singles]
        pair_times = [pair_time(pos[k], vel[k], pos[l], vel[l], sigma) for k, l in pair
s]
        next_event = min(wall_times + pair_times)
        t += next_event
        for k, l in singles: pos[k][l] += vel[k][l] * next_event

        if min(wall_times) < min(pair_times):
                collision_disk, direction = singles[wall_times.index(next_event)]
                vel[collision_disk][direction] *= -1.0
        else:
                for k in range(N): histo_data.append(pos[k][0])
                a, b = pairs[pair_times.index(next_event)]
                del_x = [pos[b][0] - pos[a][0], pos[b][1] - pos[a][1]]
                abs_x = math.sqrt(del_x[0] ** 2 + del_x[1] ** 2)
                e_perp = [c / abs_x for c in del_x]
                del_v = [vel[b][0] - vel[a][0], vel[b][1] - vel[a][1]]
                scal = del_v[0] * e_perp[0] + del_v[1] * e_perp[1]

                for k in range(2):
                        vel[a][k] += e_perp[k] * scal
                        vel[b][k] -= e_perp[k] * scal

        print 'event',event
        print 'time',t
        print 'pos',pos
        print 'vel',vel

pylab.hist(histo_data, bins=100, normed=True)
pylab.xlabel('x')
pylab.ylabel('frequency')
pylab.title('Probability distribution - Pair Collisions only - Event Disks Box')
pylab.grid()
pylab.savefig('event_disks_pair_collisions_histo.png')
pylab.show()
```

Evaluation/feedback on the above work

Note: this section can only be filled out during the evaluation phase.

Here, you are asked to evaluate a scientific diagram and to check the python code that generated the diagram. You can give a total of **2 points**.

Give **1 point** if the diagram meets the following criteria of graphical quality (which will be the general criteria for the quality of diagrams in this course):

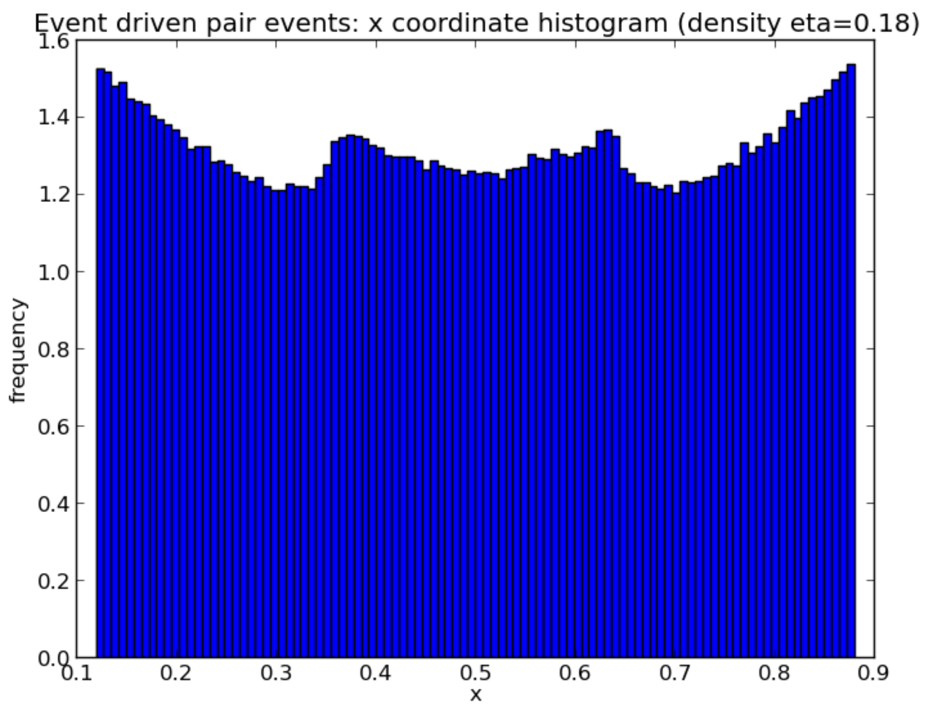
- The diagram shows exactly the quantities that were asked for (here: the histogram of positions over the x-coordinate)
- A title clearly indicates the program used to produce the output, for example "Event driven (pair collisions): x coordinate histogram (density eta=0.18)"
- All axes are correctly labelled (here: "Histogram of positions" and "x". Equivalent labels like "Histogram" or "Frequency" instead of "Histogram of
- positions" or "x-coordinate" for "x are also acceptable.)
- All axes have the correct scaling (here: both x-axis and y-axis are linear)
- The diagram shows at least the range of data points that was asked for (here: 100 bins.)

Give **1 point** if the diagram represents the correct results and was generated by a correctly programmed python code. The correct plot will again show a histogram with values slowly increasing towards the boundaries, and a rounded bump in the center, but the peaks due to wall collisions that were visible in (A3.1) are no longer present (see example below).

In order to only record the x-positions at pair collision events, a correct program records the x-positions in the following way:

```
    if min(wall_times) < min(pair_times):
        collision_disk, direction = singles[wall_times.index(next_event)
]
        vel[collision_disk][direction] *= -1.0
    else:
        for k in range(4): histo_data.append(pos[k][0])
```

If the plot meets the criteria of graphical quality, shows the correct results generated by a correctly written program, the student earned a total of **2 points**. An example plot that would receive the full score is shown here:



Score from your peers: **2**

(A3.3.1)
Finally (**grand finale!**), compute the histogram of the x-positions with 100 bins for the event-driven molecular dynamics algorithm, where you take the x-positions at regular time intervals t=0,1,2,3, using the following code snippet, to be introduced after the line "next_event = min(wall_times + pair_times)":

```
    ...
    next_event = min(wall_times + pair_times)
    t_previous = t
    for inter_times in range(int(t + 1), int(t + next_event + 1)):
        del_t = inter_times - t_previous
        for k, l in singles: pos[k][l] += vel[k][l] * del_t
        t_previous = inter_times
        for k in range(4): histo_data.append(pos[k][0])
    t += next_event
    ...
```

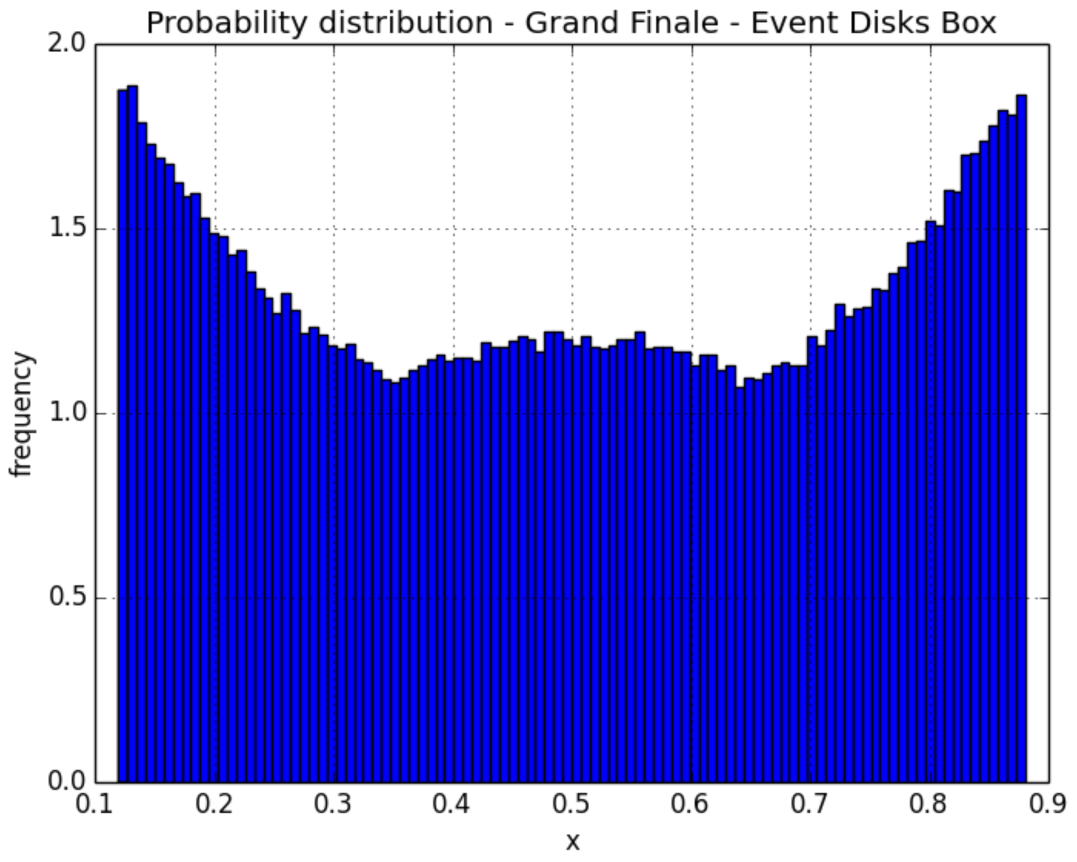
Further, replace the line

```
for k, l in singles: pos[k][l] += vel[k][l] * next_event
```

in the original event_disks_box.py by the line:

```
for k, l in singles: pos[k][l] += vel[k][l] * (t - t_previous)
```

Set n_events=1000000. As always, make sure that the histogram has an appropriate title, indicating which program was used to produce the graphical output and that the axes are labelled. Upload the histogram using the upload box below. Compare the histogram to what you obtained in sections (A1) and (A2.1). Are the probabilities the same? Write your answer into the textbox below.



Indeed, the histogram in **A3.3.1** is identical to those obtained in **A1** and **A2.1**.

Evaluation/feedback on the above work

Note: this section can only be filled out during the evaluation phase.

Here, you are asked to evaluate a scientific diagram and to check whether the student correctly interprets the diagram in comparison to the other diagrams obtained so far. You can give a total of **2 points.**

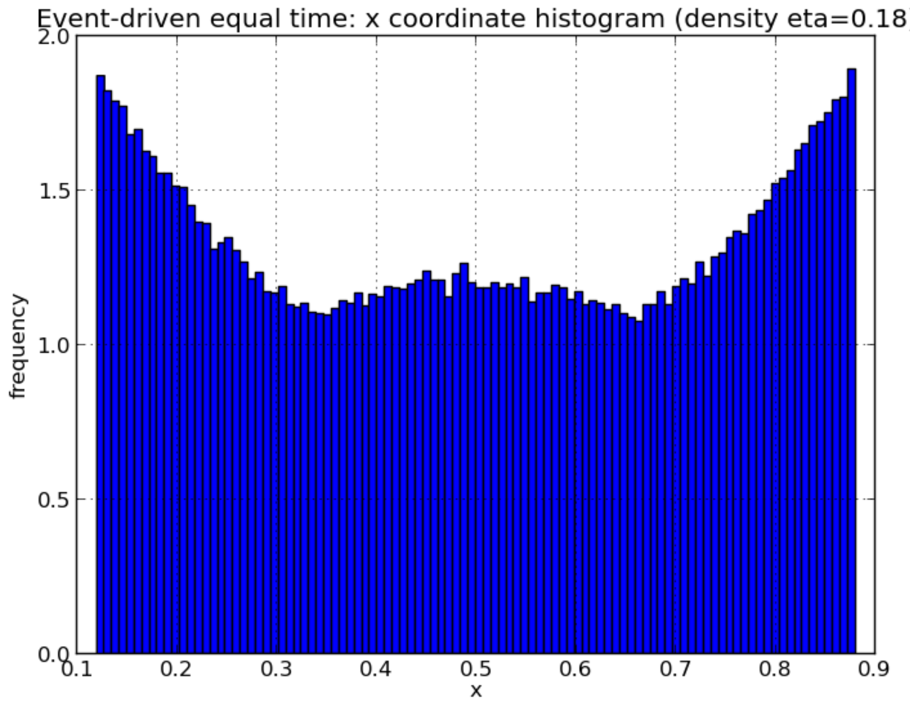
Give **1 point** if the diagram meets the following criteria of graphical quality (which will be the general criteria for the quality of diagrams in this course):

- The diagram shows exactly the quantities that were asked for (here: the histogram of positions over the x-coordinate)
- A title clearly indicates the program used to produce the output, for example "Event-driven (equal time): x coordinate histogram (density eta=0.18)"
- All axes are correctly labelled (here: "Histogram of positions" and "x". Equivalent labels like "Histogram" or "Frequency" instead of "Histogram of positions" or "x-coordinate" for "x are also acceptable.)
- All axes have the correct scaling (here: both x-axis and y-axis are linear)
- The diagram shows at least the range of data points that was asked for (here: 100 bins.)

Give **1 point** if the diagram represents the correct results: the plot will show a histogram with values increasing towards the boundaries and a rounded bump in the center (see example below) and will be equivalent to the diagrams obtained in (A1) and (A2.1). The student should answer

"Yes, the probabilities are the same as in (A1) and (A2.1) up to statistical errors."

If the plot meets the criteria of graphical quality and shows the correct results and the student noted that the probabilites are indeed the same as those obtained earlier, the student earned a total of **2 points**. An example plot that would receive the full score is shown here:



Score from your peers: **2**

(A3.3.2)
The histogram you obtained in section (A3.3.1) should be (up to statistical fluctuations) identical to what you obtained in sections (A1) and (A2.1). This
is a great finding - it establishes the equivalence between Newton's deterministic dynamics and Boltzmann's statistical dynamics. Does this equivalence follow from the detailed balance condition discussed by Werner in Lecture 1? Please comment briefly.

The equivalence between Newton's deterministic dynamics and Boltzmann's statistical dynamics does not only follow from the detailed balance condition but also from the ergodicity condition (irreducibility and aperiodicity) as explained in **A2.2**. For example, the irreducibility condition ensures that all parts of the configuration space are visited eventually.

The equivalence ensures that the time averaging (Newton's deterministic dynamics) and ensemble averaging (Boltzmann's statistical dynamics) of any system property lead to same results for a system satisfying the conditions as stated above.

Evaluation/feedback on the above work

Note: this section can only be filled out during the evaluation phase.

Here, you are asked to check whether the student can explain that this eqivalence does not follow from detailed balance. In fact, it is a deep result in Mathematics that is way beyond the scope of our course.

Give **1 point**, if the student provided this correct answer.

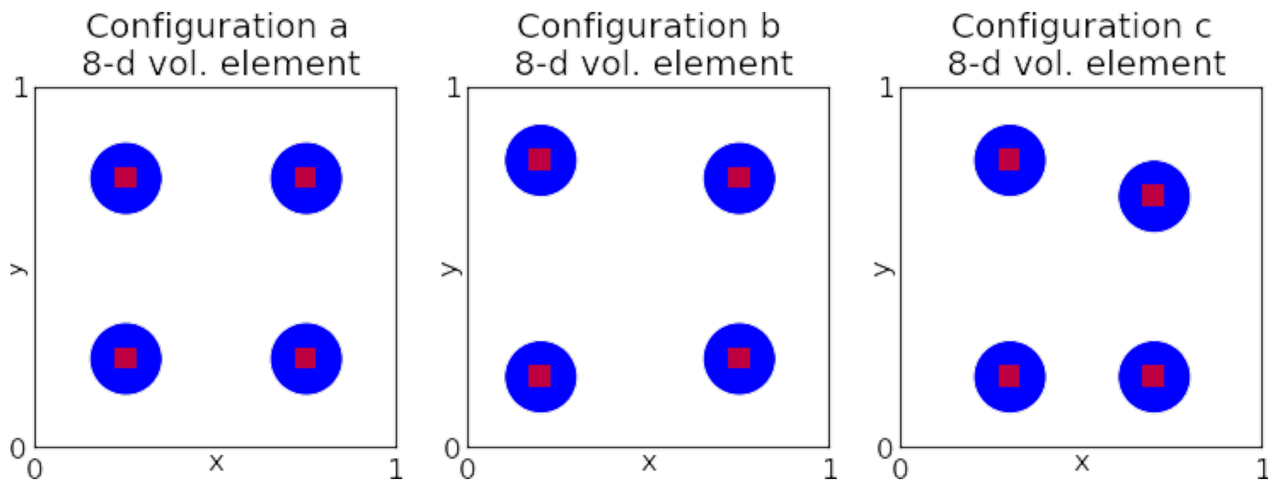
An acceptable answer would be:

"The eqivalence does not follow from detailed balance."

Score from your peers: **1**

(B) (more advanced, but not so difficult, after all)
In lecture 2 and tutorial 2, we insisted on the equiprobability principle, which governs the statistical physics of hard disks and hard spheres, yet observed a manifestly non-uniform probability of x-positions (the equiprobability principle manifestly does not apply to x-positions).

(B1)
To convince yourself that equiprobability IS satisfied, in the eight-dimensional space of allowed configurations of four-disk positions, consider the configuration a shown in the accompanying figure. The probability to sample this configuration EXACTLY is of course zero, so we must put little boxes around this configuration, as shown in red.



Using small boxes [x - 0.1, x + 0.1], modify the program direct_disks_box_multirun.py to show that the probability to sample configurations a, b, and c are the same (within the numerical precision), with

```
a = [(0.25, 0.25), (0.25, 0.75), (0.75, 0.25), (0.75,0.75)]
b = [(0.20, 0.20), (0.20, 0.80), (0.75, 0.25), (0.75,0.75)]
c = [(0.30, 0.20), (0.30, 0.80), (0.70, 0.20), (0.70,0.70)]
```

For your convenience, the program is already provided:

```
import random, math

def direct_disks_box(N, sigma):
    condition = False
    while condition == False:
        L = [(random.uniform(sigma, 1.0 - sigma), random.uniform(sigma, 1.0 - sigma))]
        for k in range(1, N):
            a = (random.uniform(sigma, 1.0 - sigma), random.uniform(sigma, 1.0 - sigma))
            min_dist = min(math.sqrt((a[0] - b[0]) ** 2 + (a[1] - b[1]) ** 2) for b in L)

            if min_dist < 2.0 * sigma:
                condition = False
                break
            else:
                L.append(a)
                condition = True
    return L

N = 4
sigma = 0.1
n_runs = 1000000
conf_a = [(0.25, 0.25), (0.25, 0.75), (0.75, 0.25), (0.75,0.75)]
conf_b = [(0.20, 0.20), (0.20, 0.80), (0.75, 0.25), (0.75,0.75)]
conf_c = [(0.30, 0.20), (0.30, 0.80), (0.70, 0.20), (0.70,0.70)]
hits = 0
Total = 0
del_xy = 0.1
configuration = conf_a
for run in range(n_runs):
    x_vec = direct_disks_box(N, sigma)
    cond = True
    for b in configuration:
        cond_b = min( max( abs(a[0] - b[0]), abs(a[1] - b[1]) ) for a in x_vec)  < del_xy
        cond *= cond_b
    if cond: hits += 1
print hits / float(n_runs), 'proportion of confs in eight-dimensional volume element.'
```

Cut-and-paste, run and modify your program for a disk radius σ =0 and for σ = 0.1. In two sentences, explain what this program does and write down the obtained frequencies for configurations a, b, and c, with four digits (e.g. 0.003) and provide a short explanation.

The program computes probability to sample several configurations for a given tolerance of 0.1. Three configurations (a, b and c) are considered.

σ = 0: Configuration a, freq=0.000; Configuration b, freq=0.000; Configuration c, freq=0.000
σ = 0.1: Configuration a, freq=0.001; Configuration b, freq=0.001; Configuration c, freq=0.001

When **σ = 0**, we don't have disks anymore but points. The probability to **exactly** sample such a configuration is infinitesimally small, as also output by the program for each of the three configurations.

For **σ = 0.1**, the probability is same (0.001) to sample each of the three configurations.

Evaluation/feedback on the above work

Note: this section can only be filled out during the evaluation phase.

In correcting part B, please realize that this is difficult material. Don't be too harsh.

Here, you have to check whether the student can correctly explain what the provided python program does and how it can be interpreted physically.

The program checks that there is one disk center in each of the four little boxes shown in the exercise. The frequencies are of the order 0.001, and they are approximately the same for configurations a, b, c because the equiprobability principle holds.

If the student can explain what the program does, give **1 point.** An acceptable answer would be

"The program checks that there is one disk center in each of the four little boxes shown in the exercise."

If the student furthermore can provide the frequencies which - rounded to four digits should 0.001 for all three configurations and understands that this is due to the equiprobability principle, give **1 point.** An acceptable answer would be:

"The probabilities for all three configurations are 0.001. They are all the same (up to statistical errors) because of the equiprobability principle."

If both parts of the question are correctly answered, give a total of **2 points**.

Score from your peers: **1.5**

(B2)
Provide an analytical expression for these probabilities, for both cases σ = 0 and for σ = 0.1, and check it against the results of section (B1).

Attention: for σ = 0.1, you need to know the acceptance ratio of direct_disks_box_multirun.py, modify your program to obtain it. Explain your analytical formula both for σ = 0 and σ =0.1 in the below text box and compare it to your results from part (B1).

Note that the outcome of your test for configurations a,b, and c comes as no surprise for the direct-sampling Monte Carlo algorithm (see Michael's discussion in tutorial 2). You could also implement this same test within the event-driven Molecular dynamics calculation, but you probably got the idea.

$Probability = \frac{(2\delta)^8}{(V/4!)}$ where

δ is tolerance (del_xy) and

$(V/4!)$ is total volume of possible configuration space with 4 identical disks.

$\sigma = 0$: Each of the eight coordinates can be from [0,1]. Therefore, $V = 1^8 = 1$.

$\sigma \neq 0$: Each of the eight coordinates has to be from $[\sigma, 1-\sigma]$ to avoid overlapping with walls. Also, the disks have to avoid overlapping with each other. Therefore, $V = L^8 \rho_{accept}$ where

$L = 1 - 2\sigma$ and

ρ_{accept} is probability of disks not overlapping.

For $\sigma = 0.1$, $V = 0.8^8 \rho_{accept}$.

From simulations, $\rho_{accept} = 0.34$ for $\sigma = 0.1$. Also, $delta = 0.1$.

Therefore, Probability=0.000 and 0.001 for $\sigma = 0$ and $\sigma = 0.1$ respectively as in **B1**.

Evaluation/feedback on the above work

Note: this section can only be filled out during the evaluation phase.

Here, you are asked to check whether the student was able to derive a correct formula for the probabilities that were numerically obtained by use of the provided python program. You can give a total of 2 points.

The correct formula for the case σ = 0 is

(2.0 * del_xy) ** 8 * 24

and the correct formula for the case of general σ is

(2.0 * del_xy) ** 8 * 24 / ((1.0 - 2.0 * sigma) ** 8 * p_accept)

where p_accept is the acceptance ratio of the direct-sampling algorithm (note that p_accept = 1 for σ = 0).
To obtain p_accept, one can for example compute the total number n_trials of trials needed to produce the n_runs samples, and is then given by

p_accept = n_runs / n_trials

The factor 24 = 4! is a combinatorial factor corresponding to the different ways of putting the four disks into the little red boxes.

If the student was able to derive the formula for the case σ = 0 (or a mathematically equivalent formula), give 1 point. If the student was able to
derive the formula for finite σ, give **1 point**. If the student computed the acceptance ratio for σ = 0.1, and checked it against the results of section B1, give 1 point.

Score from your peers: **3**

Overall evaluation/feedback

Note: this section can only be filled out during the evaluation phase.

peer 1 → Excellent.

peer 2 → *[This area was left blank by the evaluator.]*