University of Padova

Physics of Data

Numerical Methods in Soft Matter

Marco Zenari

marco.zenari.2@studenti.unipd.it

ID: 2097012

June 22, 2024

Disclaimer:
In this report I present only the results of the exercises. The code, analysis, and calculations can be found in the attached folder. In the attached folder, the data obtained from the simulations are omitted to reduce the weight of the attachment and can be found in the public repository: repository for data of simulations.

Contents

1 Sampling **4**
- 1.1 Sampling random points within D-dimensional domains by hit and miss 4
- 1.2 Inversion Method 5
- 1.3 Inversion Method II 7
- 1.4 Additional Excercises 7

2 Sampling via Transformation of Coordinates **9**
- 2.1 Sampling uniformly points within a unit radius disk 9
- 2.2 Box Muller Transformation 11
- 2.3 Rejection method 14

3 Importance Sampling **16**
- 3.1 Importance Sampling I 16
- 3.2 Importance Sampling II 17
- 3.3 Importance Sampling III 18

4 Markov Chains **20**
- 4.1 Markov Chain Equation 20
- 4.2 Digraph 20
- 4.3 Irreducible 20
- 4.4 Regularity 20
- 4.5 Stationarity 20

5 Simulation of a 2D Ising model by the Metropolis algorithm **21**
- 5.1 Exercise 21
  - 5.1.1 Programm and simulation 21
  - 5.1.2 Equilibration time, averages and fluctuations 21
- 5.2 Excerise 22
  - 5.2.1 Integrated correlation time and critical slowing-down 22
  - 5.2.2 Finite-size analysis and estimates of the critical exponents 22

6 Advanced Simulations of a 2D Ising model **26**
- 6.1 Wolff Cluster Algorithm 26
  - 6.1.1 Cluster size statistics 26
  - 6.1.2 Autocorrelation time 28
- 6.2 Multiple Markov Chains 29

7 Continuous time Markov processes, Gillespie algorithm **31**
- 7.1 Lotka-Volterra 31
- 7.2 Brusselator 32

8 Off-lattice simulations: basics **34**
- 8.1 Reduced units 34
- 8.2 Off-lattice Monte Carlo 34
- 8.3 Off-lattice Monte Carlo of Hard Spheres 35

8.4 Off-lattice Monte Carlo of Lennard Jones particles 37

9 Integration schemes **39**
9.1 Harmonic Oscillators 39
9.2 Algorithms 40
9.3 The harmonic oscillator #2 41

10 Interaction potentials & thermostats **42**
10.1 Exercise: Canonical Fluctuations 42
10.2 Lennard-Jones fluid in the microcanonical ensembles. 42
10.3 Exercise 10.3 A 44

11 Langevin and Brownian Dynamics **46**
11.1 On the validity of the Langevin approach 46
11.2 Simple Brownian motion 46
11.3 Exercise 48
11.3.1 One Trap 48
11.3.2 Double Harmonic Trap - Escape time 52

12 Reweighting techniques **54**
12.1 Change of measure 54
12.2 Single Histogram Method 54
12.3 Multiple Histogram Method (MHM) 55
12.4 Umbrella Sampling (optional) 57

13 Langevin simulation of many particles **60**
13.1 Cell list 60
13.2 Active matter 60

Chapter 1

Sampling

1.1 Sampling random points within D-dimensional domains by hit and miss

Rectangle
Reference files in *cap1_Sampling* folder:

- *ex_1_1_rectangle.c*;
- *nmsm.c* (module with costum functions implemented during the course);
- *nmsm.h* (header to the module nmsm.c);
- *script.sh* (a simple bash script to compile and run the code).

The goal of this exercise is to implement a function that uniformly samples in a given interval $[inf, sup]$ and use it to estimate the area of a rectangle $[a, b] \times [c, d]$ using the hit-and-miss method.
I started by implementing a function that uniformly samples numbers in the interval $[0, 1]$ with the following code:

```
double uniform(){
        //rand() samples an integer between 0 and RAND_MAX
        //from random generator of stdlib
        double out = (double)rand()/(RAND_MAX+1.);
        return out;
}
```

To obtain a uniform sampler in a $[inf, sup]$ interval, it is sufficient to reparametrize the output of the previous function as done in the function below:

```
double uniform_interval(double inf, double sup){
        //reparametrize the uniform sample U[0,1] in U[inf, sup]
        double out = inf + (sup - inf)*uniform();
        return out;
}
```

Finally, by sampling with the previous function two numbers, respectively, in $x \sim U[a, b]$ and $y \sim U[c, d]$ we obtain a pair of coordinates (x, y) that by construction are inside the rectangle and all the throws of the hit and miss are actually hits. The area is then given by $A = (b-a)(d-c)\frac{\#hits}{\#throws} = (b-a)(d-c)$, as expected.

Disk
Reference files in *cap1_Sampling* folder:

- *ex_1_1_disk.c*;
- *nmsm.c* (module with costum functions implemented during the course);

- *nmsm.h* (header to the module nmsm.c);
- *script.sh* (a simple bash script to compile and run the code).

The goal of this exercise is to compute the area of a disk with unit radius ($r = 1$) with the hit-and-miss method. This will give us an estimate of the constant π by confronting with the analytical result $A = \pi$. In order to do that, I used the **uniform_interval()** function previous described to uniformly sample $(x, y) \sim [-1, 1] \times [-1, 1]$. I did that for $N = 10^5$ times and counted the number of hits, which is the number of (x, y) samples such that $x^2 + y^2 < 1$. The estimation of the area of the disk and since of the constant π, is then given by $A_{est} = 4\frac{\#hits}{N} \simeq 3.1497$.

1.2 Inversion Method

Power law distribution
Reference files in *cap1_Sampling* folder:

- *ex_1_2.c*;
- *notes.pdf*(hand-written calculations of the exercises);
- *data/data_power_sample_*.txt*(folder and txt files containing the samples);
- *data_processing.ipynb*(jupyter notebook containing the analysis of the saved data);
- *nmsm.c* (module with costum functions implemented during the course);
- *nmsm.h* (header to the module nmsm.c);
- *script.sh* (a simple bash script to compile and run the code).

The goal of this exercise is to sample from a power-law distribution by means of inversion method. I solved the exercise in the most general framework, by writing a function that samples from the distribution

$$p(x) = cx^n \tag{1.1}$$

in the interval

$$x \in [inf, sup]. \tag{1.2}$$

C is a constant that can be obtained by imposing the normalization of the distribution, and it is equal to

$$c = \frac{n+1}{sup^{n+1} - inf^{n+1}}, \tag{1.3}$$

the complete calculation can be found in the file *notes.pdf*, point (1). To implement the inversion method I computed the cumulative distribution function:

$$F(x) = c\left[\frac{x^{n+1}}{n+1} - \frac{inf^{n+1}}{n+1}\right], \tag{1.4}$$

calculation in point (2) of *notes.pdf.* The cdf can be easily inverted as done in point (3) of *notes.pdf* and finally, sampling a uniform variable $\xi \sim U[0, 1]$ we can obtain a variable sampled from our power law as

$$x = \left[\left(\frac{n+1}{c}\xi + inf^{n+1}\right)^{\frac{1}{n+1}}\right]. \tag{1.5}$$

My C function to implement this sampler is the following (you can find it in *nmsm.c*):

```
double power_sampler(double n, double inf, double sup){
        double r = uniform(); //sample r in U[0,1]
        double c = (n+1)/(pow(sup, n+1)-pow(inf, n+1)); //normalization factor
        double base = (r+c*pow(inf, n+1)/(n+1))*(n+1)/c;
        double out = pow(base, 1/(n+1)); //from inversion method
        return out;
}
```

In the file *es_1_2.c* I sampled $N = 10^4$ points for each proposed power exponent in the proposed interval, saving the data in the *.txt* files. In *data_processing.ipynb* I read the data, did histograms, and did the comparison with the analytic power law. In the following figures are presented the results.

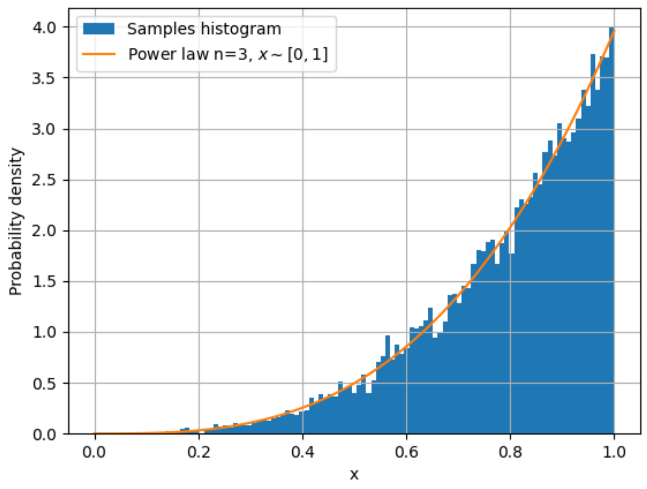

Figure 1.1: Histogram of the sample from a power law distribution with exponent $n = 3$ in the interval $[0, 1]$ and comparison with the analytical power law.

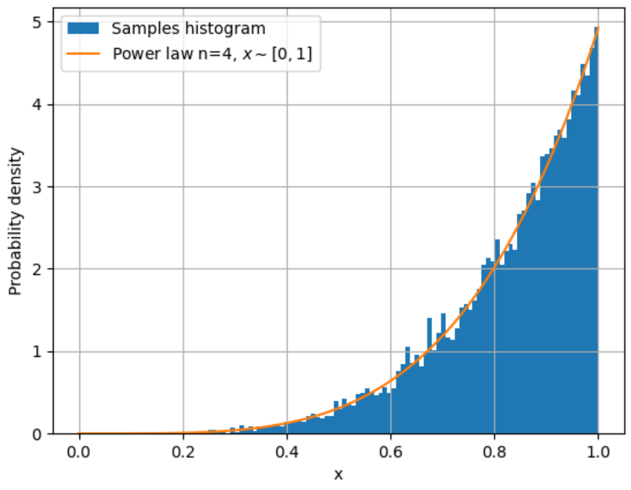

Figure 1.2: Histogram of the sample from a power law distribution with exponent $n = 4$ in the interval $[0, 1]$ and comparison with the analytical power law.

1.3 Inversion Method II

Power law distribution with n=2

Reference files in *cap1_Sampling* folder:

- *ex_1_2.c*;
- *notes.pdf* (hand-written calculations of the exercises);
- *data/data_power_sample_*.txt* (folder and txt files containing the samples);
- *data_processing.ipynb* (jupyter notebook containing the analysis of the saved data);
- *nmsm.c* (module with costum functions implemented during the course);
- *nmsm.h* (header to the module nmsm.c);
- *script.sh* (a simple bash script to compile and run the code).

In this exercise, I have used the code described in the previous section in the particular case $\rho(x) = cx^2$ for $x \in [0, 2]$. Follows the results.

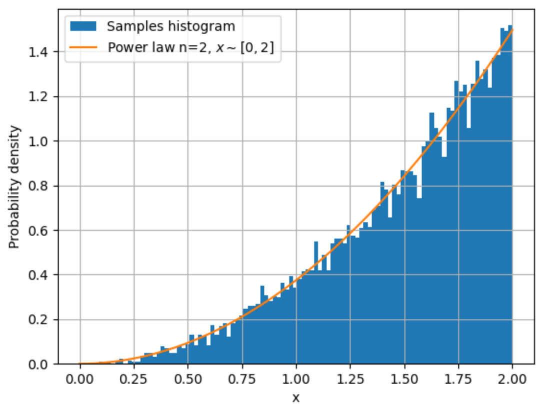

Figure 1.3: Histogram of the sample from a power law distribution with exponent $n = 2$ in the interval $[0, 2]$ and comparison with the analytical power law.

1.4 Additional Excercises

Reference files in *cap1_Sampling* folder:

- *additional_exercises.ipynb* (jupyter notebook with calculations and code).;
- *plots* (folder with the plot).

In this exercise, I have implemented the inversion method to sample from the following proposed distributions:

$$\rho(x) = \mu \exp(-\mu x) \qquad \mu = 1, \tag{1.6}$$

$$\rho(x) = 2x \exp\left(-x^2\right), \tag{1.7}$$

$$\rho(x) = \frac{1}{(a + bx)^n} \qquad a = 1, b = 1, n = 2. \tag{1.8}$$

The results for the cumulative function of the distributions and its inverse function can be found, together with the implementation, in the file *additional_exercises.ipynb*. In the following figure, we plot the histogram of the samples and the corresponding probability distributions.

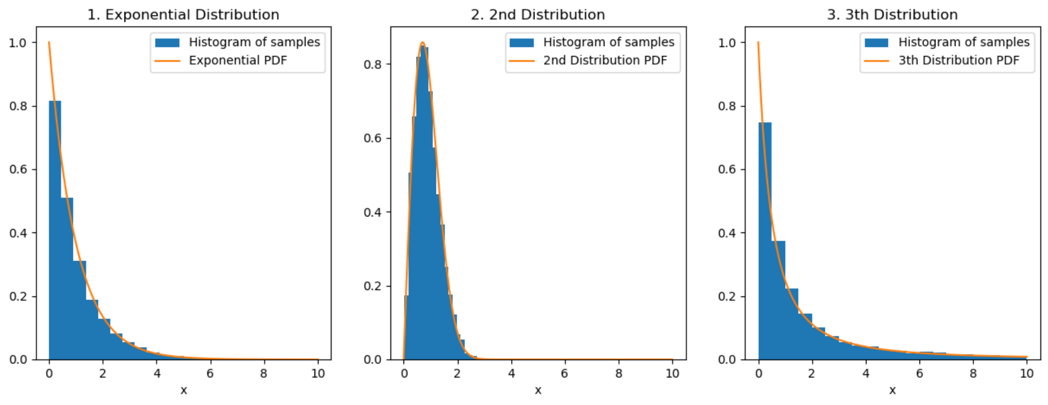

Figure 1.4: Sample from different distributions using inversion method.

Chapter 2

Sampling via Transformation of Coordinates

2.1 Sampling uniformly points within a unit radius disk

Reference files in *cap2_Sampling_via_Transformation_of_Coordinates* folder:

- *ex_2_1disk.c*;
- *notes.pdf*(hand-written calculations of the exercises);
- *data/disk_sample_*.txt*(folder and txt files containing the samples);
- *data_processing.ipynb*(jupyter notebook containing the analysis of the saved data);
- *nmsm.c* (module with costum functions implemented during the course);
- *nmsm.h* (header to the module nmsm.c);
- *script.sh* (a simple bash script to compile and run the code).

The goal of this exercise is to uniformly sampling from a unitary disk (with radius $r = 1$). The naive approach of sampling two numbers $r = \xi_1$ and $\theta = 2\pi\xi_2$ where $\xi_1, \xi_2 \sim U[0,1]$ does not work.
An intuitive explanation of why it does not work is that sampling the two random number as described above is equivalent to uniformly sampling from a rectangle of dimension $1 \times 2\pi$ and, trying to deform such rectangle into a disk, resulting in one of the edges being compressed in the center of the disk, giving rise to a clearly non uniform sampling in the disk. It is easy to show this fact with a simulation, as done in the file *ex_2_1.c*.
The samples are then saved in the file *data/disk_sample_wrong.txt* and analyzed in the *data_processing.ipynb* notebook.
In the following figure I plot the density of the sampled points with this method with a Gaussian kernel, showing that they are actually not uniformly distributed, but more points are sampled from the center of the disk.

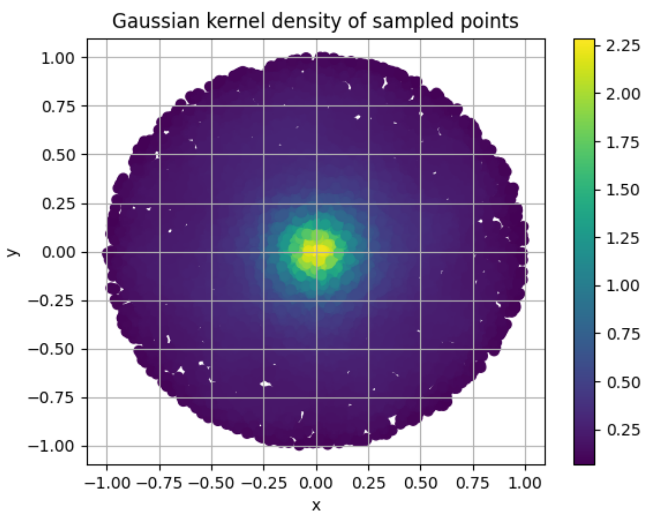

Figure 2.1: Gaussian kernel density plot of points sampled from a disk in the naive way. More points are sampled from the center then from the edge of the disk.

A better way to uniformly sample from a unitary disk is to use the polar transformation of coordinates. The idea is to consider the probability density $P(x, y)$ and the probability density $P(r, \theta)$ where the two sets of coordinates are connected by the polar transformation

$$x = r \cdot \cos(\theta) \tag{2.1}$$

$$y = r \cdot \sin(\theta). \tag{2.2}$$

The Jacobian of the transformation is $J = r$ (details on calculation on *notes.pdf* point (4)). Therefore the two probability distribution are connected by the following equation

$$p(x, y) = \frac{p(r, \theta)}{r}. \tag{2.3}$$

In cartesian coordinates the probability distribution is easy to compute

$$p(x, y) = \frac{1}{\pi}, \tag{2.4}$$

calculations are in point (5) of *notes.pdf*. Therefore $P(r, \theta) = \frac{r}{\pi}$ and we can compute the marginal distribution

$$P(r) = 2r, \tag{2.5}$$

calculations in point (6) of *notes.pdf*. The probability of θ given r is then obtained as $P(\theta|r) = P(r, \theta)/P(r)$ and it is equal to (point (7) of *notes.pdf*):

$$P(\theta|r) = \frac{1}{2\pi}. \tag{2.6}$$

Now it is sufficient to use inversion method to sample from $P(r)$ and then again to sample from $P(\theta|r)$. The details of the calculation can be found in point (8) of *notes.pdf* and the final result is that, sampling $\xi_1, \xi_2 \sim U[0, 1]$ we can obtain uniformly sampled polar coordinates on a unitary disk by means of the relations:

$$r = \sqrt{\xi_1} \tag{2.7}$$

$$\theta = 2\pi\xi_2. \tag{2.8}$$

The code for this sampling with transformation of coordinates is implemented in *ex_2_1.c*, the data are saved in *data/disk_sample_right.txt* and analyzed int the notebook *data_processing.ipynb*.

The results are shown in the following figure, where the density of the points is computed with a gaussian kernel.

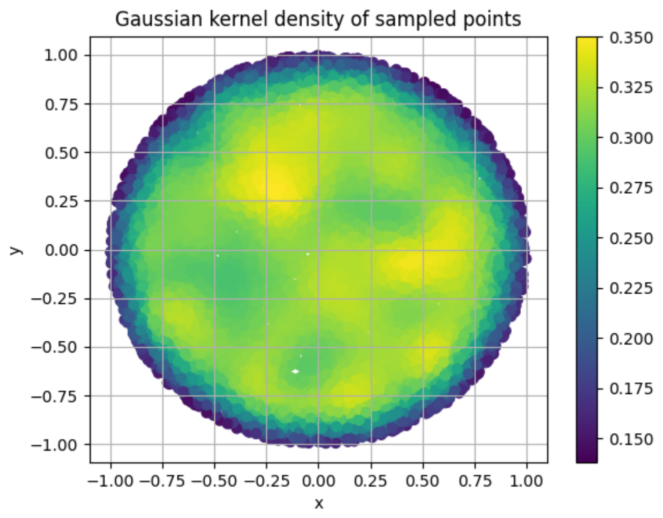

Figure 2.2: Gaussian kernel density plot of points sampled from a disk with transformation of coordinates. The sampling is uniform in the disk with some boundary effects at the edge.

2.2 Box Muller Transformation

Reference files in *cap2_Sampling_via_Transformation_of_Coordinates* folder:

- *ex_2_2.c*;
- *notes.pdf* (hand-written calculations of the exercises);
- *data/gaussian_sample*.txt*(folder and txt files containing the samples);
- *data_processing.ipynb*(jupyter notebook containing the analysis of the saved data);
- *nmsm.c* (module with costum functions implemented during the course);
- *nmsm.h* (header to the module nmsm.c);
- *script.sh* (a simple bash script to compile and run the code).

The goal of this exercise is to design an algorithm based on the transformation of coordinates able to sample from a normal distribution $\mathcal{N}(0,1)$ and then extending it to a general gaussian distribution $\mathcal{N}(\mu,\sigma^2)$. We can start from the following 2D gaussian probability density

$$P(x,y) = \frac{1}{2\pi} e^{-\frac{x^2+y^2}{2}}, \tag{2.9}$$

and apply the polar coordinates transformation as done in the previous case. The jacobian of the transformation is still $J = r$ and the new probability distribution is given by

$$P(r,\theta) = \frac{r}{2\pi} e^{-\frac{r^2}{2}}, \tag{2.10}$$

that can be factorized defining $P(\theta) = \frac{1}{2\pi}$ and $P(r) = re^{-\frac{r^2}{2}}$. Now we can sample from $P(r)$ and $P(\theta)$ by means of the inversion method, obtaining

$$r = \sqrt{-2\ln(1-\xi_1)} \tag{2.11}$$
$$\theta = 2\pi\xi_2, \tag{2.12}$$

where $\xi_1, \xi_2 \sim U[0,1]$; the calculations can be found at point (9) of *notes.pdf*. Remembering now the transformation to polar coordinates 2.1 we obtain two gaussian samples $x, y \sim \mathcal{N}(0,1)$ from two uniformly distributed variables $\xi_1, \xi_2 \sim U[0,1]$ as following:

$$x = \sqrt{(-2\ln(\xi_1)} \cdot \cos(2\pi\xi_2) \tag{2.13}$$

$$y = \sqrt{(-2\ln(\xi_1)} \cdot \sin(2\pi\xi_2). \tag{2.14}$$

It follows from the properties of the Gaussian distribution that if a sample $x \sim \mathcal{N}(0,1)$ we can obtain a sample $y \sim \mathcal{N}(\mu, \sigma^2)$ by applying the following transformation:

$$y = \mu + \sigma \cdot x. \tag{2.15}$$

The code for this algorithm is implemented in *nmsm.c* as follow:

```
    double * box_muller(double mu, double sigma){
        //creating a pointer allocation memory
        double * x = (double *)malloc(2*sizeof(double));
        double u1 = uniform(); //sampling a uniform rv
        double u2 = uniform(); //sampling a uniform rv
        //using box-muller transformations to obtain two normal gaussian rv
        x[0] = sqrt(-2*log(u1))*cos(2*PI*u2);
        x[1] = sqrt(-2*log(u1))*sin(2*PI*u2);
        //reparametrize the rv to a N[mu, sigma^2]
        x[0] = mu + x[0]*sigma;
        x[1] = mu + x[1]*sigma;
        //returning the pointer
        return x;
}
```

This function is used in the code *ex_2_2.c* to sample points from a gaussian distribution with mean μ and standard variation σ. The sampled points (N=5000) are saved in the files *data/gaussian_sample*.txt* and histogrammed in the notebook *data_processing.ipynb*. The results are shown in the following figures.

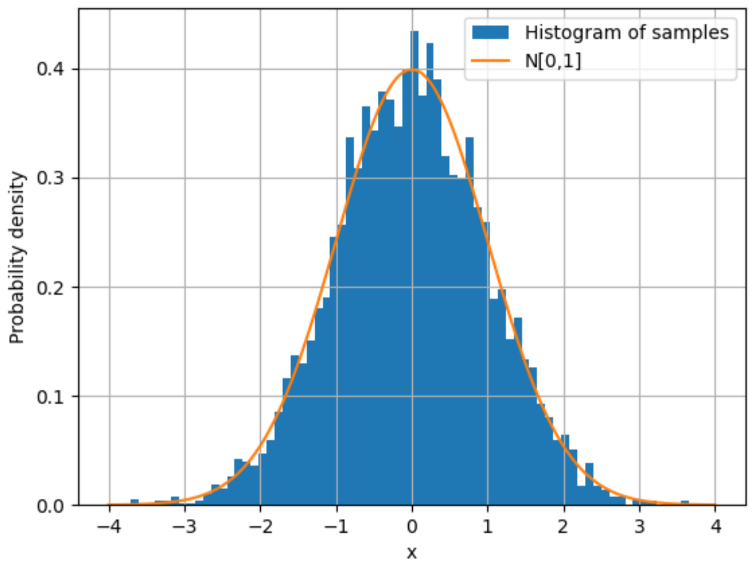

Figure 2.3: Histogram of points sampled from a distribution $\mathcal{N}(0,1)$ with box-muller transformation and comparison with analytical distribution.

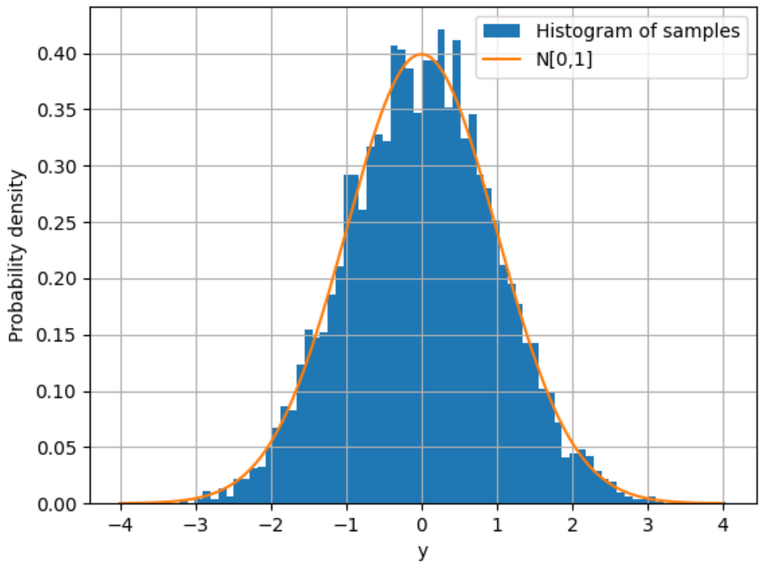

Figure 2.4: Histogram of points sampled from a distribution $\mathcal{N}(0,1)$ with box-muller transformation and comparison with analytical distribution.

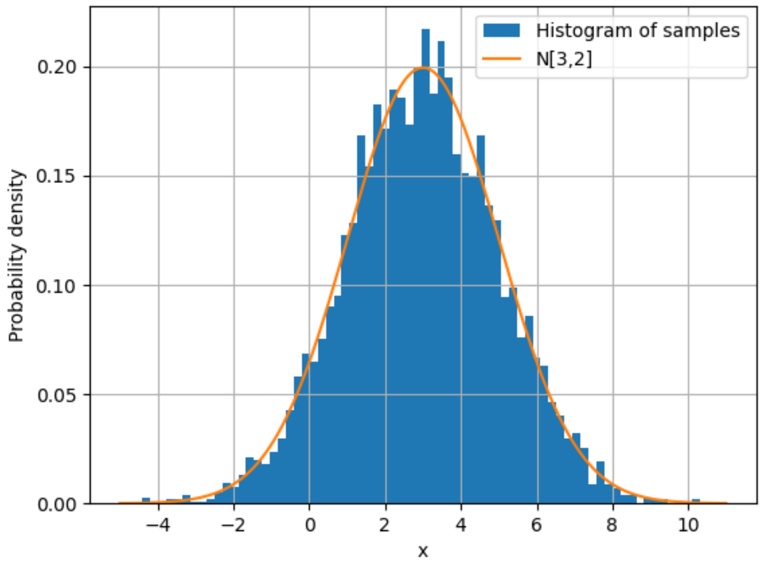

Figure 2.5: Histogram of points sampled from a distribution $\mathcal{N}(3,2)$ with box-muller transformation and comparison with analytical distribution.

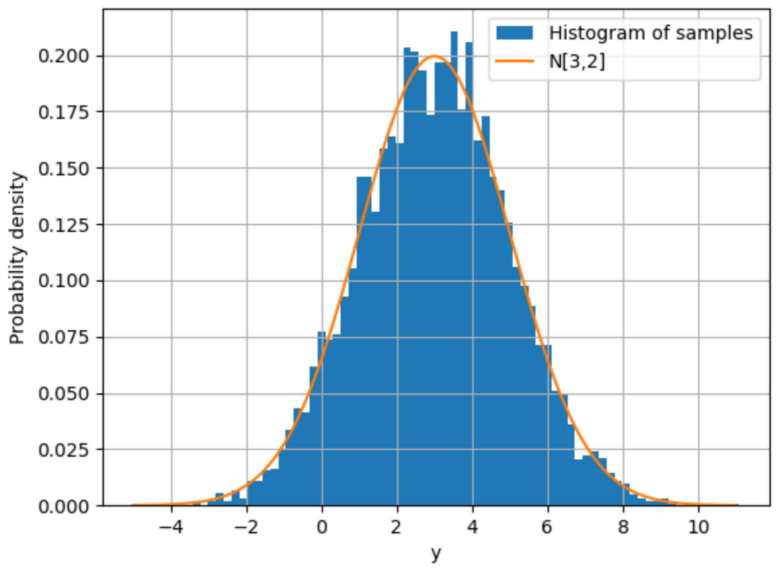

Figure 2.6: Histogram of points sampled from a distribution $\mathcal{N}(3,2)$ with box-muller transformation and comparison with analytical distribution.

2.3 Rejection method

Half-Gaussian

Reference files in *cap2_Sampling_via_Transformation_of_Coordinates* folder:

- *ex_2_3.c*;
- *notes.pdf* (hand-written calculations of the exercises);
- *data/data_for_importance_sampling.txt* (folder and txt files containing the samples);
- *data/rejection_data*.txt* (data files with the samples for different parameters used in rejection method).
- *data_processing.ipynb* (jupyter notebook containing the analysis of the saved data);
- *nmsm.c* (module with costum functions implemented during the course);
- *nmsm.h* (header to the module nmsm.c);
- *script.sh* (a simple bash script to compile and run the code).

The goal of this exercise is to sample from the distribution

$$f(x) = \frac{2}{\sqrt{\pi}} e^{-x^2}, \tag{2.16}$$

with $x \sim [0, \infty]$ by using the rejection method. The idea of the method is to find a distribution $g(x)$, from which we know how to sample, and such that $f(x) \leq cg(x)$ for some constant c. Then we can sample $X \sim g(x)$ and accept X as a sample from $f(x)$ with probability $f(x)/(cg(x))$. This can be easily done by sampling a uniformly distribution number $\xi \sim U[0,1]$ and keep the sample if $\xi \geq f(x)/(cg(x))$. Following the hint from the exercise I choose as $g(x)$:

$$g(x) = \begin{cases} A & 0 \leq x \leq p \\ \frac{A}{p} x e^{(p^2 - x^2)} & x > p \end{cases} . \tag{2.17}$$

This distribution depend on two parameters, A and p, and a relation between them can be found imposing the normalization condition (point (10) of *notes.pdf*), obtaining

$$A = \frac{2p}{2p^2+1}. \tag{2.18}$$

With this choice, it is possible to show that for $c \geq \frac{1+p^2}{\sqrt{\pi}p}$ we have that $cg(x) > f(x)$ (point (11) of *notes.pdf*).

Now we need to sample from $g(x)$ with the inversion method. The cumulative distribution of $g(x)$ is given by:

$$F(x) = \begin{cases} Ax & 0 \leq x \leq p \\ -\frac{A}{2p}e^{(p^2-x^2)} + K & x > p \end{cases}, \tag{2.19}$$

where K is an integration constant that can be obtained by imposing connection conditions, obtaining $K = Ap + \frac{A}{2p}$ (point (12) of *notes.pdf*). Substituting the relation between A and p into the one obtained for K we have $K = 1$. This result was also straightforward to obtain considering that the cumulative function must converge to 1, $F(x) \xrightarrow{x\to\infty} 1$. The next step is to invert $F(x)$. This can be done in each interval of the domain, providing to compute the corrects intervals in the co-domain. The complete calculations are in point (13) of *notes.pdf*, the result is

$$F^{-1}(y) = \begin{cases} \frac{y}{A} & 0 \leq y \leq Ap \\ \sqrt{p^2 - \ln\left(\frac{2p}{A}(k-y)\right)} & Ap < y \leq 1 \end{cases} \tag{2.20}$$

Now we can sample $\xi \sim U[0,1]$ and obtain a sample $X \sim g(x)$ through $F^{-1}(\xi)$. Finally we can compute the ration $f(X)/cg(X)$ and keep X as a sample of $f(x)$ if a random sample $\xi_1 \sim U[0,1]$ is such that $\xi_1 < f(X)/(cg(X))$. The code is implemented in *ex_2_3.c* and *nmsm.c*. The samples are saved in the files *data/rejection_data*.txt* for different values of the parameter p. Good results are obtained with $p = 0.1$ and the data are saved in *data/data_for_importance_sampling.txt* and will be used in the next exercise. The analysis of the samples is done in the notebook *data_processing.ipynb* and the result for $p = 0.1$ is shown in the following figure in comparison to the analytical probability distribution.

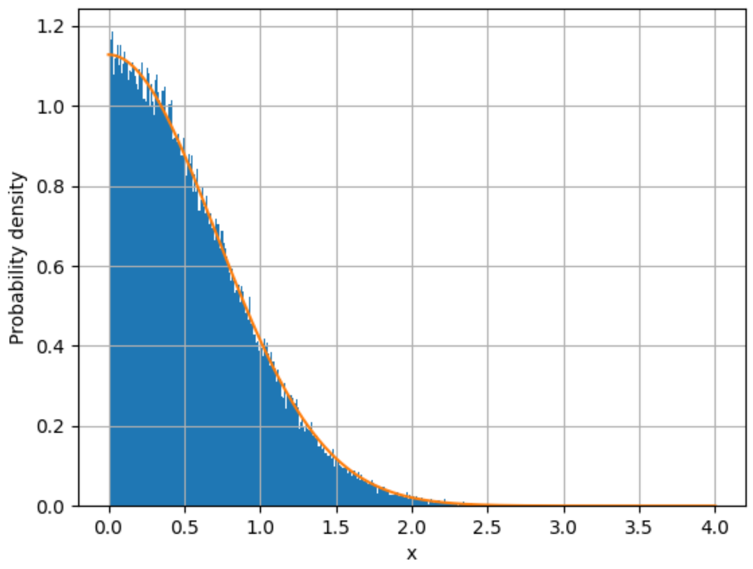

Figure 2.7: Histogram of the samples ($N = 10^5$)from a Half-Gaussian distribution obtained with rejection method ($p = 0.1$) and comparison with analytical probability distribution.

Chapter 3

Importance Sampling

3.1 Importance Sampling I

Integral of a slow varying function modulated by a Gaussian
Reference files in *cap3_Importance_Sampling* folder:

- *ex_3_1.c*;
- *data/data_for_importance_sampling.txt*(folder and txt files containing the samples);
- *nmsm.c* (module with costum functions implemented during the course);
- *nmsm.h* (header to the module nmsm.c);
- *script.sh* (a simple bash script to compile and run the code).

The goal of this exercise is to compute the integral $I = \int_0^\infty e^{-x^2} g(x)$, where $g(x)$ is a slowly varying function with Monte Carlo methods in two ways: Crude Monte Carlo and Importance Sampling. For the computation a used $g(x) = x$. I start by applying the crude Monte Carlo method. The idea is to sample $x_i \sim U[inf, sup]$, where $[sup, inf]$ is the integration interval. The integral I is then estimated by

$$I_N = \frac{b-a}{N} \sum_{i=1}^{N} f_i, \tag{3.1}$$

where N is the number of samples and f_i is the function that we are integrating calculated in the sample, $f_i = f(x_i)$. The result obtained with this method is presented in Table (3.1) together with the confrontation with the analytical result and the result with importance sampling.
A more effective method is to use importance sampling. The idea is to sample from a distribution $W(x)$ as similar as possible to the function that we want to integrate (we want the ratio to be as constant as possible). For this exercise, I choose $W(x) = \frac{2}{\sqrt{\pi}} e^{-x^2}$ and use the samples saved in the previous exercise (2.3) in the file *data/data_for_importance_sampling.txt*. With this method the the estimation of the integral reduces to

$$I_N = \frac{1}{N} \sum_{i=1}^{N} \frac{f(x_i)}{W(x_i)} = \frac{\sqrt{\pi}}{2N} \sum_{i=1}^{N} g(x_i). \tag{3.2}$$

The code is implemented in the file *ex_3_1.c* and the results are summarized in Table (3.1).

Analytical result	Crude Monte Carlo	Importance Sampling
0.5	0.464067	0.494076

Table 3.1: Compariosn between the analytical result of the integral $\int_0^\infty x e^{-x^2}$ and the estimation obtained with two monte carlo methods: crude monte carlo and importance sampling with $N = 10000$ samples.

3.2 Importance Sampling II

Estimate the integral of $\cos(x)$

Reference files in *cap3_Importance_Sampling* folder:

- *ex_3_2.c*;
- *notes.pdf* (hand-written calculations of the exercises);
- *nmsm.c* (module with costum functions implemented during the course);
- *nmsm.h* (header to the module nmsm.c);
- *script.sh* (a simple bash script to compile and run the code).

The goal of this exercise is to estimate the integral

$$I = \int_0^{\frac{\pi}{2}} \cos(x)dx, \tag{3.3}$$

using importance sampling. Following the hint I used as distribution $g(x)$ for the importance sampling a second order polynomial $g(x) = a + bx^2$. In order to obtain the most efficient sampling the idea is to choose $g(x)$ such that $f(x)/g(x)$ is as constant as possible. In this case $f(x) = \cos(x)$ is a concave function in the interval of integration $[0, \frac{\pi}{2}]$ and therefore it is reasonable to choose $b < 0$ to have a concave $g(x)$ function. Using the constrain of normalization for $g(x)$ I obtain a relation between the parameters a and b,

$$a = \frac{2}{\pi} - b\frac{\pi^2}{12}, \tag{3.4}$$

the calculations are reported in the file *notes.pdf* at point (14). Another condition on a and b can be found by imposing that $g(\frac{\pi}{2}) = 0$ (done at point (15) of *notes.pdf*) obtaining

$$b = -\frac{12}{\pi^3} \tag{3.5}$$

$$a = \frac{3}{\pi}. \tag{3.6}$$

Therefore, the distribution $g(x)$ that I have chosen is given by $g(x) = \frac{3}{\pi} - \frac{12}{\pi^3}x^2$. The problem with this approach is that it is not trivial to invert the cumulative function of $g(x)$ because it is a third-order degree polynomial. For this reason I decided to sample from $g(x)$ using the rejection method, in a similar way as explained in section (2.3). As function to overestimate $g(x)$ I choose $g_1(x) = a_1 - b_1x$. Imposing the normalization condition and setting $a_1 = 1$ to start from the same point of $f(x)$ I obtain (point (16) of *notes.pdf*)

$$g_1(x) = 1 - \left(\frac{4}{\pi} - \frac{8}{\pi^2}\right)x. \tag{3.7}$$

This distribution is easy to integrate to obtain the cumulative function that it is also easy to invert, leading to the possibility to sample from $g_1(x)$ with the inversion method (point (17) of *notes.pdf*). It is sufficient to choose $c = 1.5$ to have $cg_1(x) > g(x)$. A plot of $f(x), g(x)$ and $cg_1(x)$ is represented in figure (3.1).

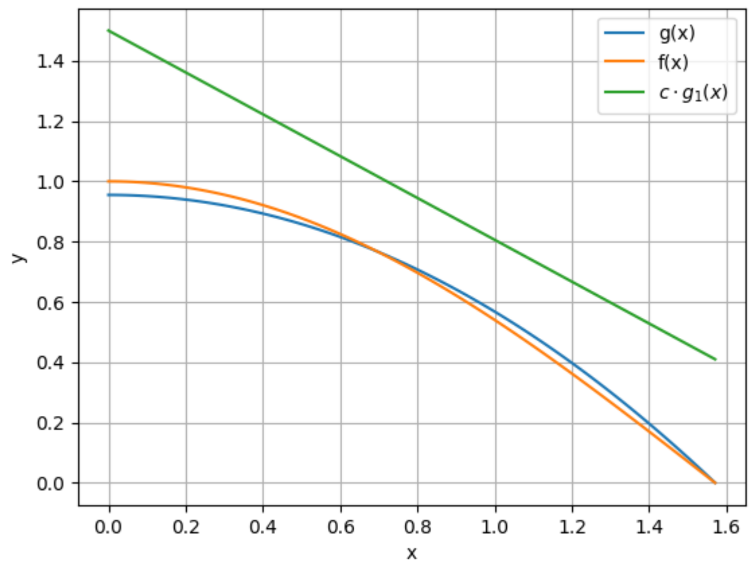

Figure 3.1: Plot of the function that we want to integrate $f(x) = \cos(x)$, the function used for importance sampling $g(x)$ and the function used for rejection method $g_1(x)$ multiplied by the constant $c = 1.5$.

The code for this exercise is implemented in the file *ex_3_2.c* and uses functions defined in the module *nmsm.c*. In order to obtain an error inferior to the 1% we need to sample at least $N = 1000$ points, for which I obtain an estimation of the integral $I_N = 1.01$ to be compared with the analytical result $I = 1$.

3.3 Importance Sampling III

Comparison of variances with or without Importance Sampling
Reference files in *cap3_Importance_Sampling* folder:

- *importance_sampling_variance.ipynb*;
- *homework2.pdf* (hand-written analytical solution of the exercises).

The goal of this exercise is to compute the variance of the expected value of a function with and without importance sampling. We consider the function

$$f(x) = \begin{cases} 0 & x < T \\ 1 & x \geq T \end{cases}, \tag{3.8}$$

and we want to compute the average respect to the probability distribution function $\rho(x) = e^{-x}$ with $x >= 0$. The problem with this approach is that the function $f(x)$ is different from zero only above a threshold T while the pdf $\rho(x)$ is high in the region $x \leq T$. In order to improve the variance on the computation with Monte Carlo method we introduce the importance sampling with the function $g(x, a) = ae^{-ax}$ defined for $x \geq 0$. The analytical results for the mean $< f >_\rho$, variance without importance sampling $\sigma^2(f)$ and variance with importance sampling $\sigma^2(fp/g, a)$ can be found in the file *homework2.pdf*. In the same file can be found the optimization procedure used to minimize $\sigma^2(fp/g, a)$, with respect to the parameter a. The best parameter a^* found is

$$a^* = \frac{2T + 2 - \sqrt{4T^2 + 1}}{2T}. \tag{3.9}$$

In the following figures I present the results for the standard deviation with and without importance sampling and the ratio between them.

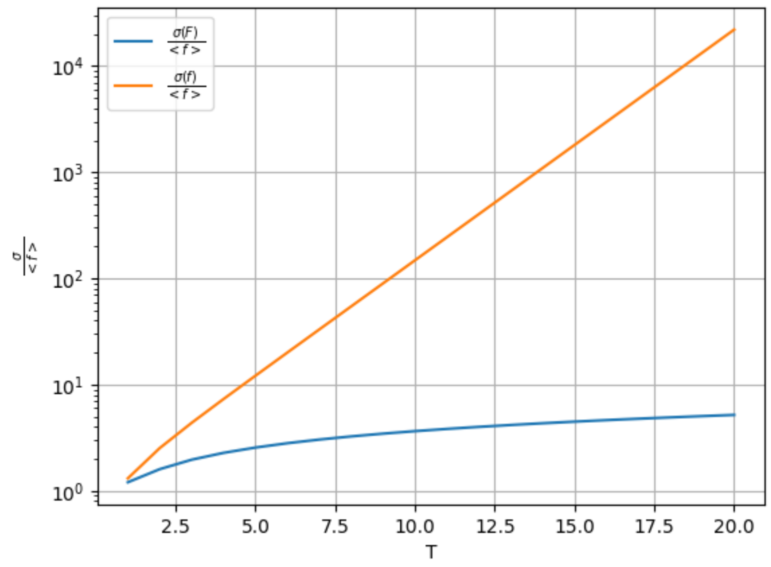

Figure 3.2: Plot of the standard deviation of the function f divided by the average $< f >$ computed with (F) and without (f) importance sampling. Logscale on the y-axis.

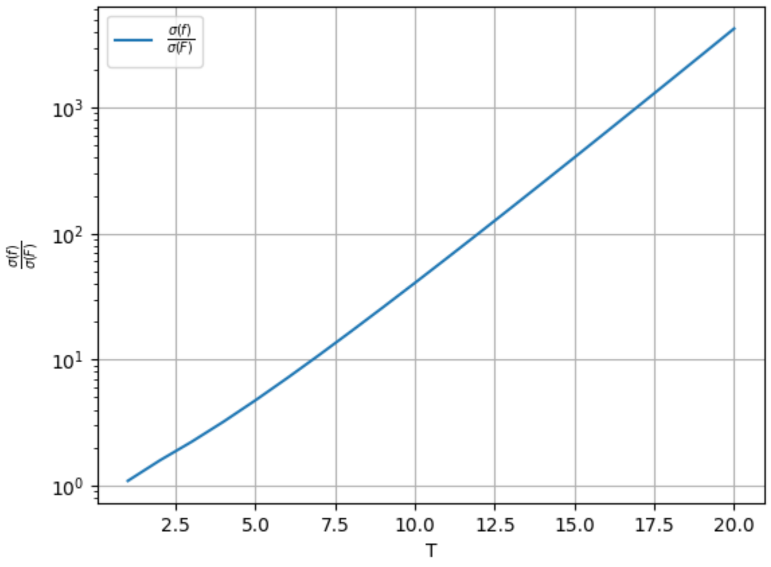

Figure 3.3: Plot of the ratio between the standard deviation of the function f, $\sigma(f)$ and the standard deviation with importance sampling $\sigma(F)$. Logscale on the y-axis.

It is evident that the most T increases the less effective is computing the average with $\rho(x)$ respect to using importance sampling with $g(x, a^*)$.

Chapter 4

Markov Chains

Reference files in *cap4_Markov_Chains* folder:

- *homework2.pdf* (hand-written analytical solution of the exercises).

All the exercises on Markov chains have been solved analytically and can be found in the file *homework2.pdf*.

4.1 Markov Chain Equation

Solved analytically in the file *homework2.pdf*.

4.2 Digraph

Solved analytically in the file *homework2.pdf*.

4.3 Irreducible

Solved analytically in the file *homework2.pdf*.

4.4 Regularity

Solved analytically in the file *homework2.pdf*.

4.5 Stationarity

Solved analytically in the file *homework2.pdf*.

Chapter 5

Simulation of a 2D Ising model by the Metropolis algorithm

Reference files in *cap5_Ising_metropolis* folder:

- *data*(folder with the saved data from simulations);
- *data_processing.ipynb*(jupyter notebook with the analysis of the simulations);
- *ising_glauber_metropolis.c* (code for simulations);
- *nmsm.c* (module with costum functions implemented during the course);
- *nmsm.h* (header to the module nmsm.c);
- *script.sh* (a simple bash script to compile and run the code).

5.1 Exercise

5.1.1 Programm and simulation

In the file *ising_glauber_metropolis.c* you can find my C implementation to simulate a 2D Ising model with Glauber (spin-flip) dynamics and metropolis algorithm. The code takes as input parameters the number of spins along each direction L, and the temperature at which the simulations are computed β. The code randomly initializes a state vector of $N = L^2$ and computes its initial energy. Then it updates the state of the system with Glauber dynamics and metropolis acceptance probability. If the Glauber move is accepted, the state of the system is updated and the observables energy and magnetization are also updated. The Glauber proposal is repeated for N times each sweep, and the evolution is carried out for $5 \cdot 10^5$ steps. At each step, the total energy and the total magnetization are saved in the *txt* file in the *data* folder. I have run the simulation for 4 values of $L = 10, 20, 30, 40$ and 22 values of β around the theoretical critical $\beta_c \simeq 0.44$.

5.1.2 Equilibration time, averages and fluctuations

In this section we analyze one of the simulations obtained for $L = 30$ and $\beta = 0.445$. The temperature is below the critical temperature, and therefore we expect the system to be in the ferromagnetic phase. The plot of the first 800 sweep times of the observables can be found in figure 5.1. The averages of the observable have been computed sampling points from simulation after the equilbration time and every auto-correlation time, computed as described in the following section. From the observable we can also compute their variance, similarly as for the average that are proportionals to the specific heat at constant volume and the magnetic susceptibility. The results are proposed in the following sections and the detailes of the computation can be found in the file *data_processing.ipynb*.

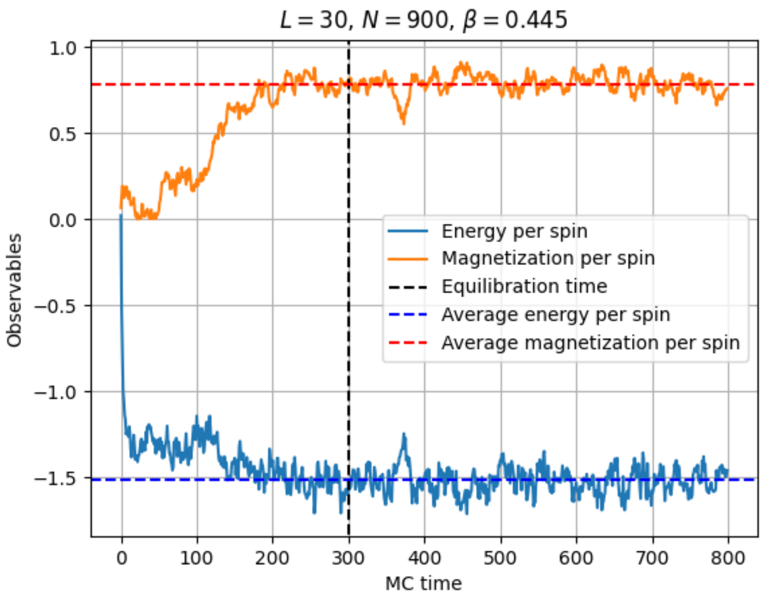

Figure 5.1: Plot of the energy per spin and the magnetization per spin of a simulation with $N = 900$ spins at $\beta = 0.445$. The dotted lines represents the equilibration time, the average magnetization per spin and the average energy per spin.

5.2 Excerise

5.2.1 Integrated correlation time and critical slowing-down

From each observable time series, I compute the corresponding autocorrelation function by integration, computing the integrals as sums in the discretized limit. Note that the time-step in these simulations is $\Delta = 1$. From the autocorrelation function, I compute the auto-correlation time as the time at which the auto-correlation function reaches $1/e$ of its initial value, as by definition. The details of the computation can be found in the file *data_processing.ipynb*. The standard deviation on the observable can then be computed by sampling every autocorrelation time or correcting its value as seen in the lectures.

5.2.2 Finite-size analysis and estimates of the critical exponents

Finally, I have used the data from the simulations to compute the specific heat and magnetic susceptibility for each simulation and plotted as a function of the temperature. The results are plotted in the following figures.

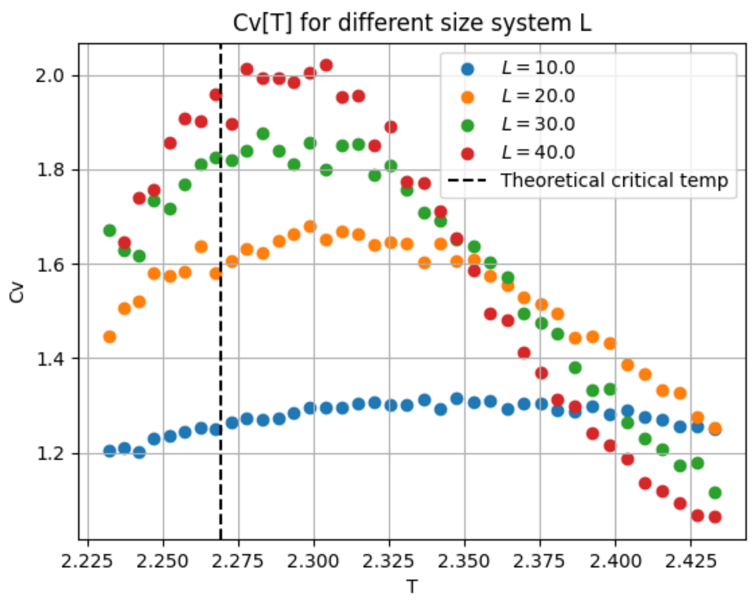

Figure 5.2: Specific heat as a function of temperature for different values of lattice extension L.

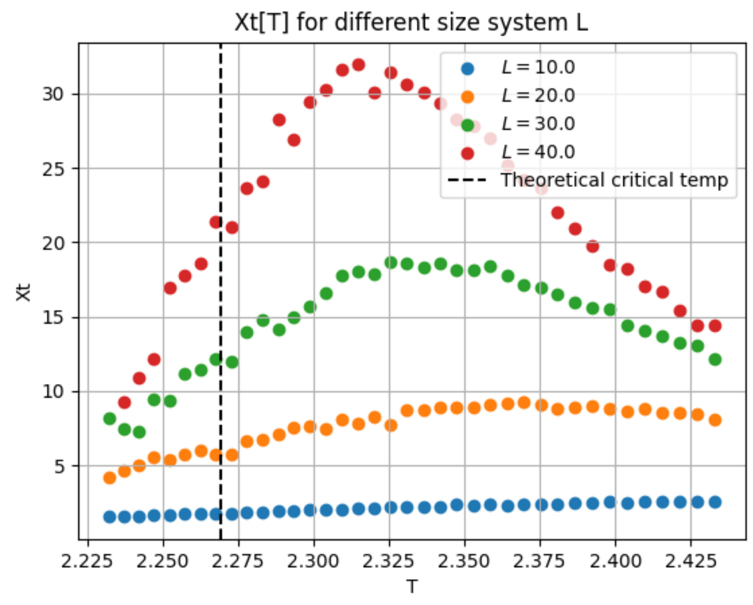

Figure 5.3: Magnetic susceptibility as a function of temperature for different values of lattice extension L.

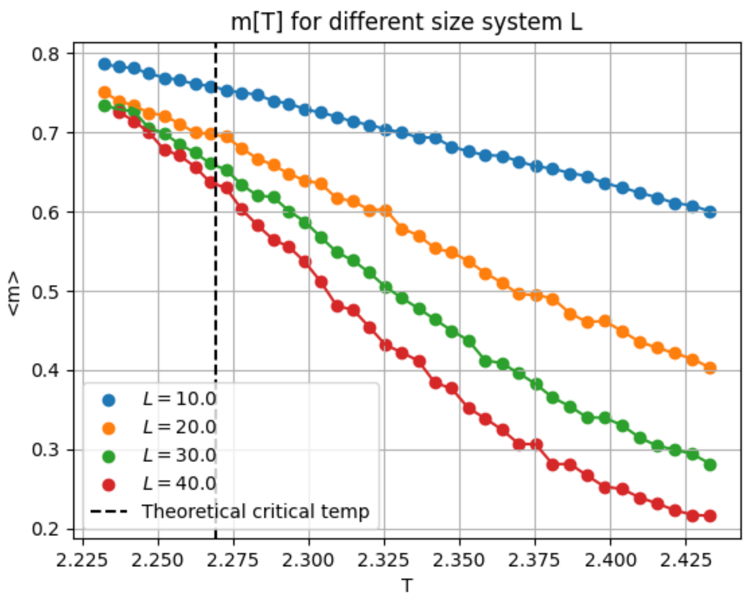

Figure 5.4: Magnetization per spin as a function of temperature for different values of lattice extension L.

From the plots I have estimated the value of the critical temperature for different values of L as the temperature for which the specific heat shows its maximum. This critical temperature differs from the theoretical one because of the effect of finite size of the lattice. The estimated critical temperatures are reported in the following table.

L	$\beta_c[L]$
10	0.426
20	0.437
30	0.437
40	0.437

Table 5.1: Critical temperatures for different finite size lattice L.

Finally, I have estimated the critical exponents β and γ by fitting the average magnetization and the magnetic susceptibility as a function of L. The fits are presented in the following figures and the results for the critical exponents are presented in the following table.

Exponent	Estimated	Theoretical prediction
β	0.126	0.125
γ	1.77	1.75

Table 5.2: Comparison between estimated exponents and theoretical prediction.

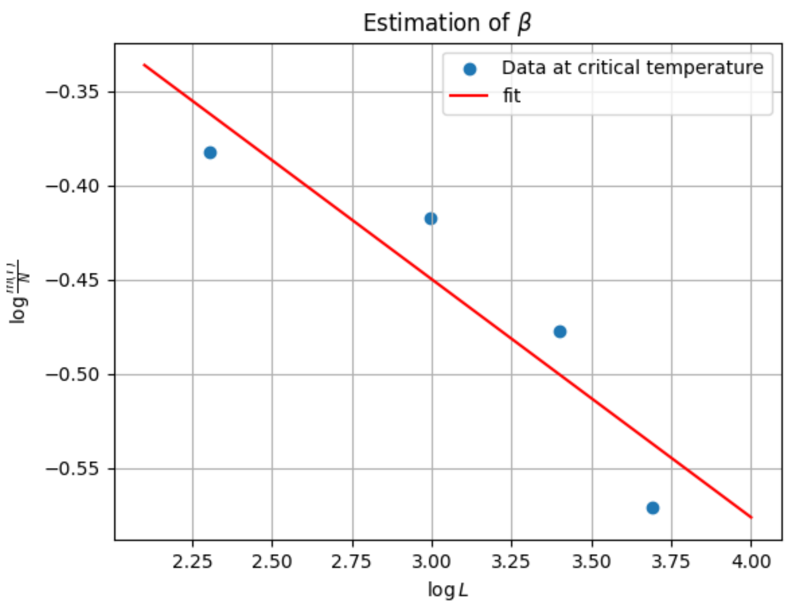

Figure 5.5: Fit for the estimation of β critical exponent.

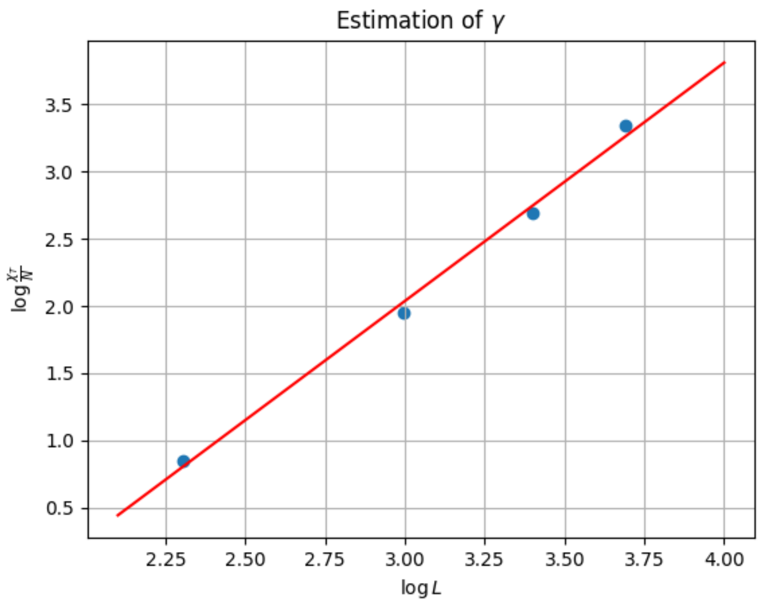

Figure 5.6: Fit for the estimation of γ critical exponent.

Chapter 6

Advanced Simulations of a 2D Ising model

Reference files in *cap6_Advanced_Ising_wolff_MMC* folder:

- *data_**(folders with the saved data from simulations);
- *data_processing.ipynb*(jupyter notebook with the analysis of the simulations);
- *data_**(folders with the saved plots from analysis);
- *ising_glauber_metropolis.c* (code for simulations with metropolis);
- *ising_wolff.c* (code for simulations with Wolff Cluster algorithm);
- *ising_MMC.c* (code for simulations with Multiple Chain Monte Carlo);
- *nmsm.c* (module with costum functions implemented during the course);
- *nmsm.h* (header to the module nmsm.c);
- *script.sh* (a simple bash script to compile and run the code).

6.1 Wolff Cluster Algorithm

The algorithm is implemented in the file *ising_wolff.c* and uses functions saved in *nmsm.c*.

6.1.1 Cluster size statistics

I have performed three different simulations for three different temperatures $\beta = \beta_c/2, \beta_c, 2\beta_c$ with $L = 50$ and $N = 2500$ for 20000 steps. The simulations are saved in the folder *data_cluster_distributions* as time series of the energy of the system, the magnetization of the systems and the size of the Wolff cluster at each iteration of the algorithm. In the following figures I report the histograms of the frequency of the cluster distribution for the different temperatures in logartimic scale.

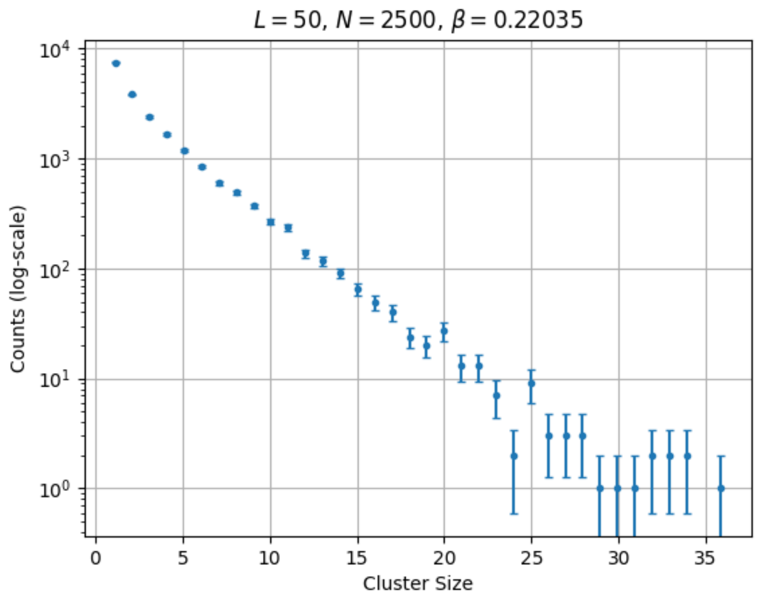

Figure 6.1: Cluster distribution in the Wolff algorithm for $\beta = \beta_c/2$.

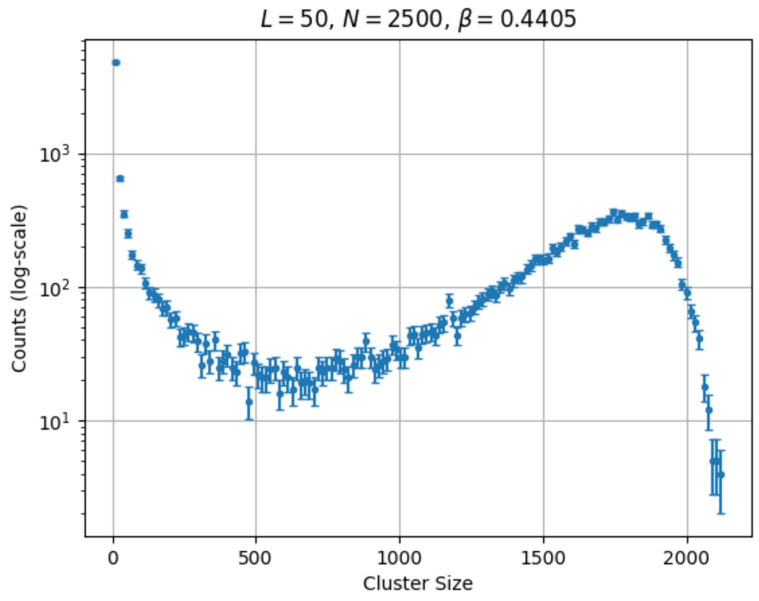

Figure 6.2: Cluster distribution in the Wolff algorithm for $\beta = \beta_c$.

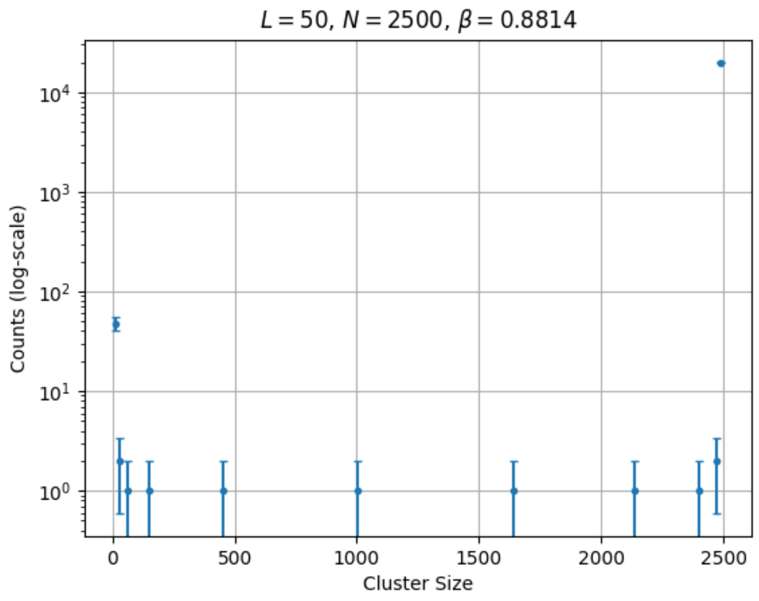

Figure 6.3: Cluster distribution in the Wolff algorithm for $\beta = 2\beta_c$.

From the histograms we can see that when the system is in the ferromagnetic phase ($\beta = 2\beta_c$) the cluster is composed of 0 or N spins. At the critical temperature we have a bimodal distribution of the cluster size while in the paramagnetic phase $\beta = \beta_c/2$ the cluster size is typically very small. This behavior is expected and the consequence is that in the limit $\beta \to 0$ the cluster size is typically $S(\beta) \simeq 1$ and the Wolff algorithm is not efficient, since it leads to similar results to Glauber metropolis, but with a much more complicated algorithm. In the ferromagnetic phase, instead, the cluster size is significant and the Wolff algorithm is more efficient.

6.1.2 Autocorrelation time

In this section, I have used the code in the files *ising_glauber_metropolis.c* and *ising_wolff.c* to simulate the Ising model at different values of L with the two algorithms: metropolis and Wolff. The aim is to compare the autocorrelation in the time series of the observable magnetization. I have used $L = 10, 20, 30, 40$ and simulated the system at the corresponding critical temperature estimated in the previous chapter (Table 5.1). The simulations are saved in the folder *data_autocorrelation* and in the following figures we plot the autocorrelation time (processed in *data_processing.ipynb* as described in the previous chapter) as a function of L, in log-log scale. Since from the theory at the critical point the relation between the two is a power law, $\tau \propto L^z$ we expect the log-log plot to be linear, and with a simple fit we can estimate the coefficient z. In the following figure, we plot the autocorrelation time of the magnetization τ_{mag} as a function of L in log-log scale and the corresponding fit.

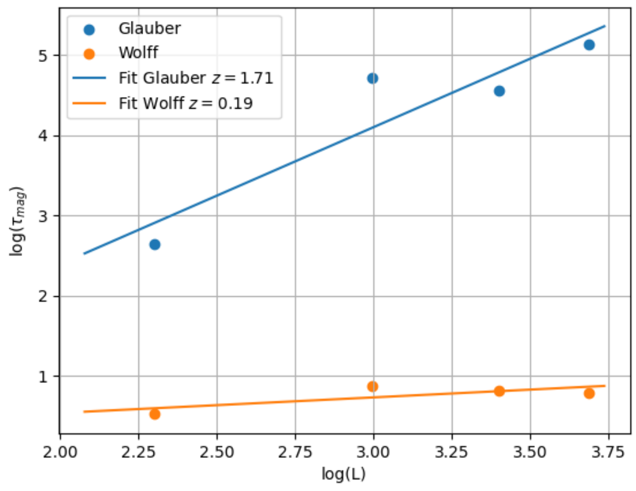

Figure 6.4: Plot of the auto-correlation time of the magnetization as a function of L for the Glauber-metropolis algorithm and the wolff cluster algorithm. The exponent of the power was is obtained with a linear fit.

As expected the Wolff algorithm is more efficient than the Glauber-metropolis algorithm in terms of critical-slowing-down. For the Wolff algorithm τ_{mag} corresponds to the auto-correlation time multiplied by $S(\beta)/L^2$ where $S(\beta)$ is the average cluster size at the corresponding temperature β.

6.2 Multiple Markov Chains

The simulations of the Ising model with multiple Markov chains are implemented in the file *ising_MMC.c*. I have used 6 different Markov chains with temperatures $\beta = 0.41, 0.42, 0.43, 0.44, 0.45, 0.46$ and have used both Glauber-Metropolis and Wolff as algorithms. I have simulated the system with both the algorithms allowing and forbidding the swap between Markov chains, and I have saved the simulations in the folder *data_MMC*. For all simulations, I have used $L = 20$ and 30000 timesteps. To verify that the energy distributions of the different chains are overlapped, I have histogrammed the energy per spin after the equilibration time. The results are shown in the following figures.

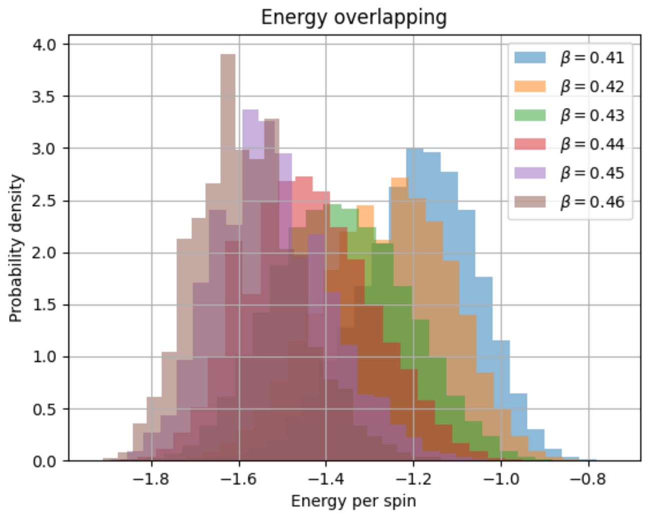

Figure 6.5: Energy overlapping of Multiple Markov Chaina algorithm with Glauber-Metropolis.

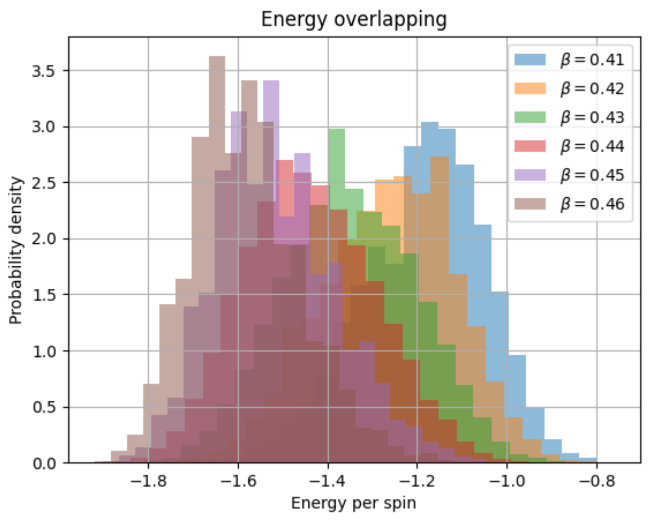

Figure 6.6: Energy overlapping of Multiple Markov Chains algorithm with Wolff.

I have then computed the autocorrelation time for the magnetization for each chain when the Swapping is allowed and when it is not (single Markov Chain Dynamic). In the following table I report the results.

	Glauber SWAP	Glauber NO SWAP	Wolff SWAP	Wolff NO SWAP
β	τ_{mag}	τ_{mag}	τ_{mag}	τ_{mag}
0.41	22	39	3	3
0.42	19	58	3	3
0.43	32	55	3	3
0.44	38	52	3	3
0.45	26	43	2	2
0.46	28	30	2	2

Table 6.1: Auto-correlation time for magnetization when the swap is allowed and when it is not in the Multiple Markov Chains for the Glauber-Metropolis algorithm and the Wolff cluster algorithm.

The tables shows a good improvement in the auto-correlation time when the MMC i used with the Glauber-Metropolis algorithm while when implementing the Wolff algorithm using MMC does not decrease the auto-correlation time.

Chapter 7

Continuous time Markov processes, Gillespie algorithm

Reference files in *cap7_Gillespie* folder:

- *cap7_Gillespie*(jupyter notebook with code and analysis);
- *plots* (folder with the plots).

The code with the implementation of the Gillespie algorithm can be found in the notebook *cap7_Gillespie* in Python. This code is used to study the Lotka-Volterra system and the Brusselator as suggested.

7.1 Lotka-Volterra

In this section, I simulate the Lotka-Volterra model with $k_1 = 3$, $k_2 = 1/100$, $k_3 = 5s^{-1}$ starting from different initial conditions. In the following figures are plotted the results.

We can see the oscillatory behavior of the predator-prey system that evolves following the Lotka-Volterra model. The populations of prey and predator oscillate around their stationary solution (prey $= k_3/k_2$, predator $=k_1/k_2$) with oscillations amplitude that are somehow related to the initial condition: if the initial condition is close to the stationary the oscillations amplitudes are smaller then if the initial conditions are far from the stationary solution. We can also see that the predator oscillations are in a delayed phase, as expected from a cause-effect point of view: a lot of preys will be followed by an increase of predators, while a low number of preys will be followed by a low number of predators. If eventually the predators go extinct, the prey grow exponentially, as expected since the only rate different from zero is $\omega_1 = k_1 \cdot X_1$. This can happen if we start with too many predators with respect to the number of preys: having no food, they will die. Another possibility is to increase the death rate of predators k_3 and reduce the predation rate k_2.

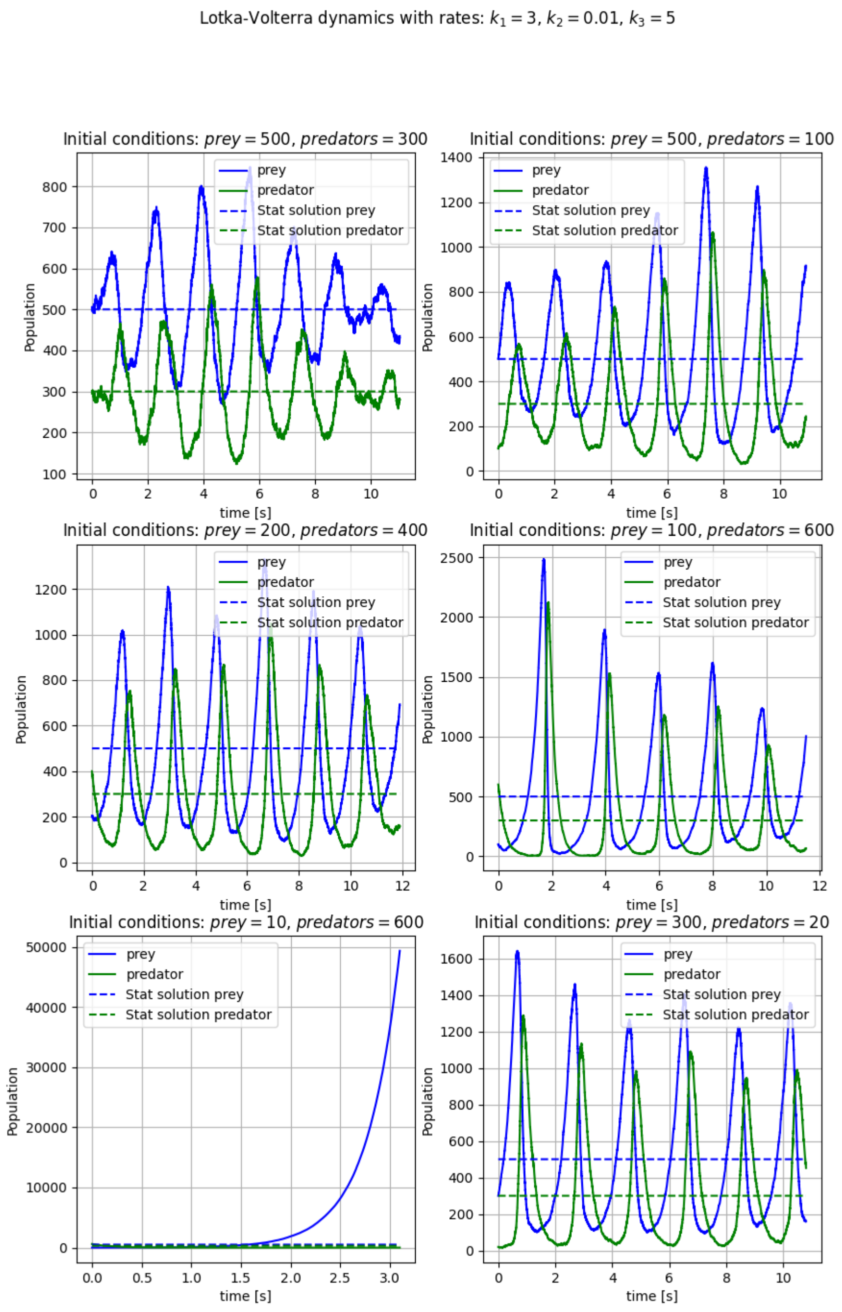

Figure 7.1: Lotka-Volterra simulations with Gillespie algorithm with different initial conditions.

7.2 Brusselator

In this section, I simulate the Brusselator with $a = 2$, $b = 5$ and different values of the volume $\Omega = 10^2, 10^3, 10^4$. The results are shown in the following plots.

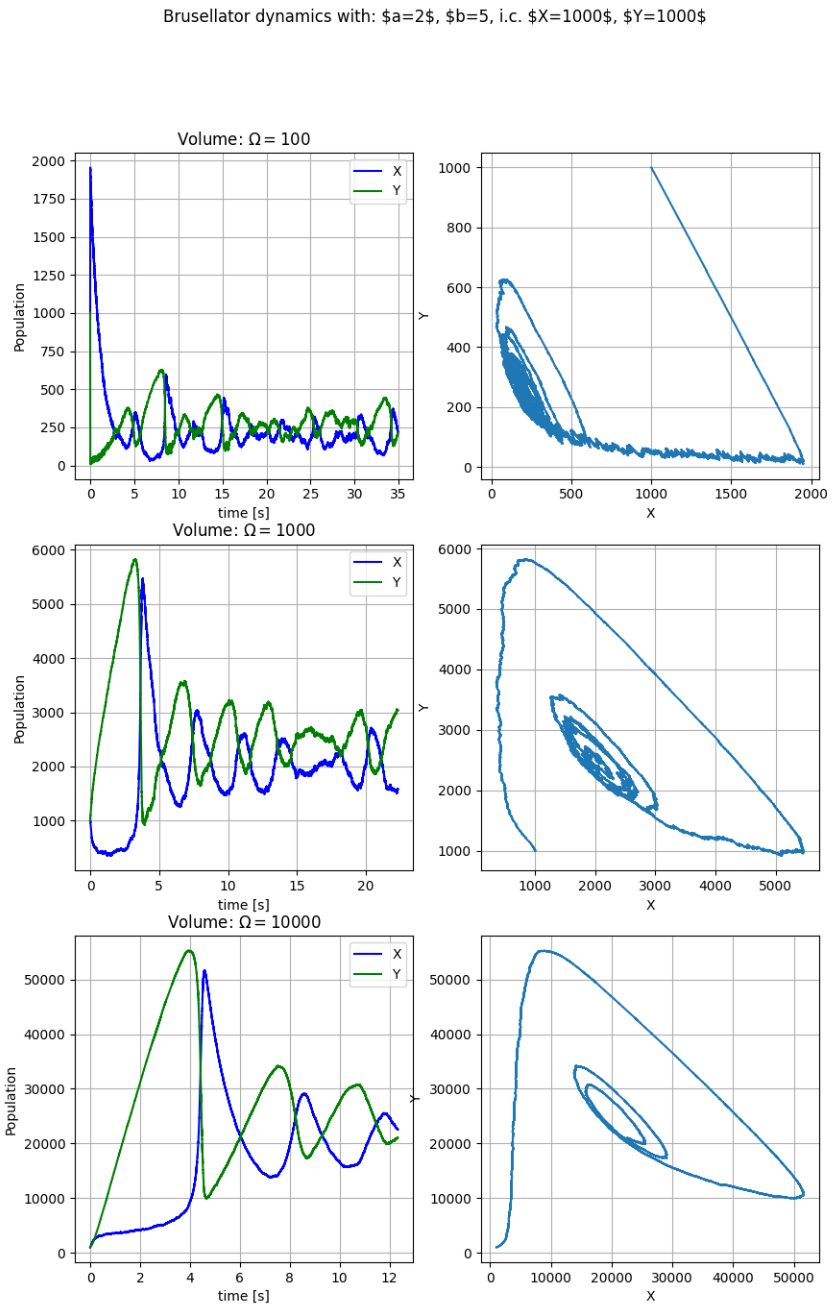

Figure 7.2: Brusselator simulations with Gillespie algorithm with different volumes values $\Omega = 10^2, 10^3, 10^4$.

If we start with a small volume Ω, lets say $\Omega = 100$ the third reaction is more likely to occur, giving rise to a spike of creations of molecule of type X. When the number of X is high enough, because of the fact that both the second and the fourth reactions are proportional to X there is suddenly a decrease in X in favor of Y. This leads to a high frequency oscillatory behavior. When Ω is of the magnitude of order of X and Y the frequency of oscillations is smaller and in the limit of large volume Ω the frequency of oscillations increases. In the phase space this means that the greater the volume is, the greater the dimension of the attractor will be.

Chapter 8

Off-lattice simulations: basics

8.1 Reduced units

In this section we consider the reduced units for common physical quantities. For a general observable O I indicate with the hat $\hat{O}$ the corresponding typical unit and with the star the observable in reduced units O^*, in such a way that $O = O^*\hat{O}$.
Given a temperature of $T^* = 2$ in reduced units we can easily compute the corresponding temperature in SI units for Argon and Krypton by considering that the typical unit of temperature is obtained from the typical unit of energy by dividing for the Boltzmann constant, $\hat{T} = \hat{E}/K_b$ and we have $\hat{T}_{Ar} = 119.8$ K and $\hat{T}_{Kr} = 164.0$ K. The temperature in SI units is then $T_{Ar} = T^*\hat{T}_{Ar} = 239.6$ K and $T_{Kr} = T^*\hat{T}_{Kr} = 328.0$ K.
To compute the typical integration timestep we need to compute the typical time unit for Argon and Krypton, knowing that $\hat{\tau} = \sqrt{\hat{m}\hat{\sigma}^2/\hat{E}}$. Using the data given in the exercise and using as typical unit for the mass the atomic mass of Argon and Krypton, $\hat{m}_{Ar} = 6.63 \cdot 10^{-26}$ Kg and $\hat{m}_{Kr} = 1.39 \cdot 10^{-25}$ Kg I obtain $\hat{\tau}_{Ar} = 2.16 \cdot 10^{-12}$ s and $\hat{\tau}_{Kr} = 2.65 \cdot 10^{-12}$ s. The typical integration time step is $\hat{dt} = 0.001\hat{\tau}$, giving $\hat{dt}_{Ar} = 2.16 \cdot 10^{-15}$ s and $\hat{dt}_{Kr} = 2.65 \cdot 10^{-15}$ s.
The friction coefficient is defined as $\gamma = F/(m * v)$ where F is the friction force, m is the mass of the particle and v is the velocity of the particle. Therefore, the typical unit of the friction coefficient is $\hat{\gamma} = \hat{F}/(\hat{m}\hat{v})$, where $\hat{v} = \hat{\sigma}/\hat{\tau}$ and $\hat{f} = \hat{E}/\hat{\sigma}$. Therefore, $\gamma^* = \gamma/\hat{\gamma}$ with $\hat{\gamma} = \hat{E}\hat{\tau}/(\hat{\sigma}^2\hat{m})$.
For a spherical particle, the viscosity coefficient η is related to the friction coefficient by the Stokes law $\gamma = 6\pi R\eta$ where R is the radius of the particle. Therefore, in reduced units, we have $\eta^* = \eta/\hat{\eta}$ with $\hat{\eta} = 1/(6\pi) \cdot \hat{E}\hat{\tau}/(\hat{\sigma}^3\hat{m})$.

8.2 Off-lattice Monte Carlo

Reference files in *cap8_Off_lattice_simulations_basic* folder:

- *OLMC_code_base*(folder with the code);

In the folder *OLMC_code_base* you can find all the files used to produce a base code for an off-lattice Monte Carlo simulation. The files with the extension **.h* contains useful definitions for the code that is implemented in the file *main.c*. The file *input.dat* contains the input parameters for the simulations, coded as *key world = value*, and is read by the *read_input_file()* function defined in the *init.h* file. The code reads the input file, initializes the random generator, and allocates the necessary memory with the function *allocate()* defined in the file *utils.h*. The function *initialization()* randomly initializes the system and *print_config()* is a debugging function used to print in stdout the configuration of the system. The function *compute_tot_energy()* uses the definitions in the file *energy.h* to compute the energy of the system, and the function *monte_carlo_sweep()* computes a Monte Carlo sweep with metropolis algorithm. Finally, the function *clean()* frees the allocated memory. Other files are the *random.h* that contains usefuls sampling functions, *definitions.h* that contains the struct definition of a particle and of the system that is simulated, *Makefile* that compiles and execute the code. All the suggestions of the exercise are implemented in the *monte_carlo.h* file, like the Monte Carlo sweep being composed of N displacement move trials and each Monte Carlo proposal move and acceptance with Metropolis Algorithm. The Monte Carlo move is implemented by sampling a random particle and proposing a random displacement that is accepted according to the metropolis algorithm, considering the energy of the system before and after the displacement. If the move is accepted, the state of the system is updated, and the new total energy is computed.

8.3 Off-lattice Monte Carlo of Hard Spheres

Reference files in *cap8_Off_lattice_simulations_basic* folder:

- *OLMC_code_hard_spheres*(folder with the code and the output data);
- *data_processing_hard_spheres.ipynb* (jupyter notebook with analysis of simulations);
- *plots* (folder with plots).

In this exercise, I modify the code of the previous one to simulate the Hard Spheres model. In this model, each particle is represented as a sphere of radius $\sigma = 1$ and the interaction potential is $V = 0$ if the particles do not overlap, and $V = \infty$ if the particles overlap. I have chosen as a measure of the energy the total number of overlaps in the system. I have implemented Monte Carlo in the following way: the Monte Carlo move is proposed as in the previous exercise, by sampling a random particle and a random displacement of maximum length (d_{max}), while the metropolis acceptance can be simplified by accepting every move that maintains or decreases the number of overlaps. I have simulated the system for $N = 100$ particles at different densities $\rho = 0.05, 0.3, 0.5, 1$ varying the dimensions of the box and using different values for the maximum displacement parameter $d_{max} = 0.01\sigma, 0.1\sigma, 0.5\sigma, 0.8\sigma, 1\sigma$. I have repeated the simulations 10 times for each configuration saving the time series of energy and acceptance ration in the folder *OLMC_code_hard_spheres*
output. The parameters have been changed with a bash script that is used to write the *input.dat* file. In the following figures I plot the results for the observable energy (computed as number of overlaps) and acceptance ratio as functions of the maximum displacement at different densities. The observables have been averaged over the last 10^3 Monte Carlo sweeps (simulations uses 10^6 MC sweeps) and over the 10 different realizations. For the initialization of the system, I have used two different approaches: the first one is random initialization inside the box, while the second one is the cubic-lattice initialization.

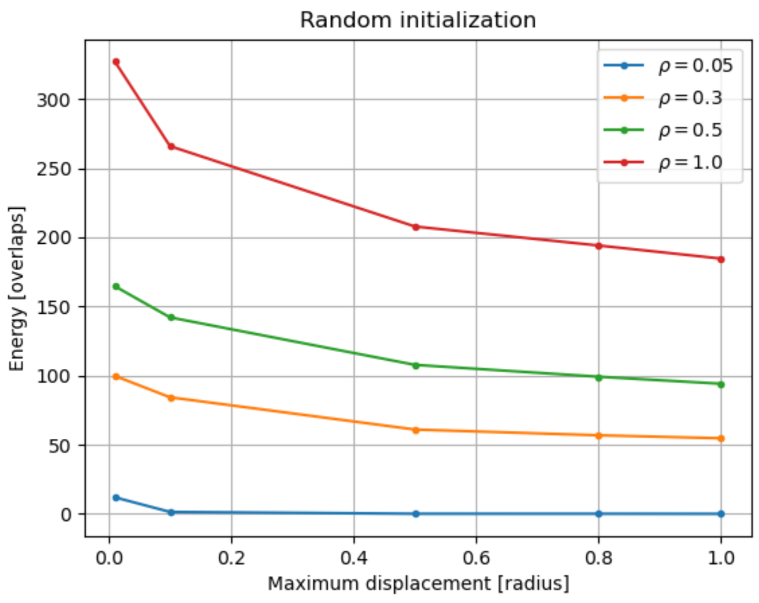

Figure 8.1: Energy overlaps as a function of the maximum displacement hyper-parameter with random initialization of the particles.

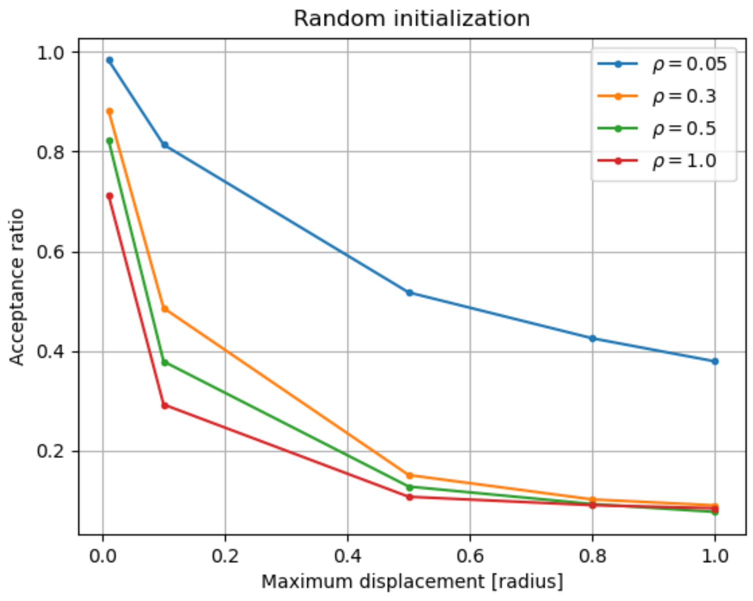

Figure 8.2: Acceptance Ratio as a function of the maximum displacement hyper-parameter with random initialization of the particles.

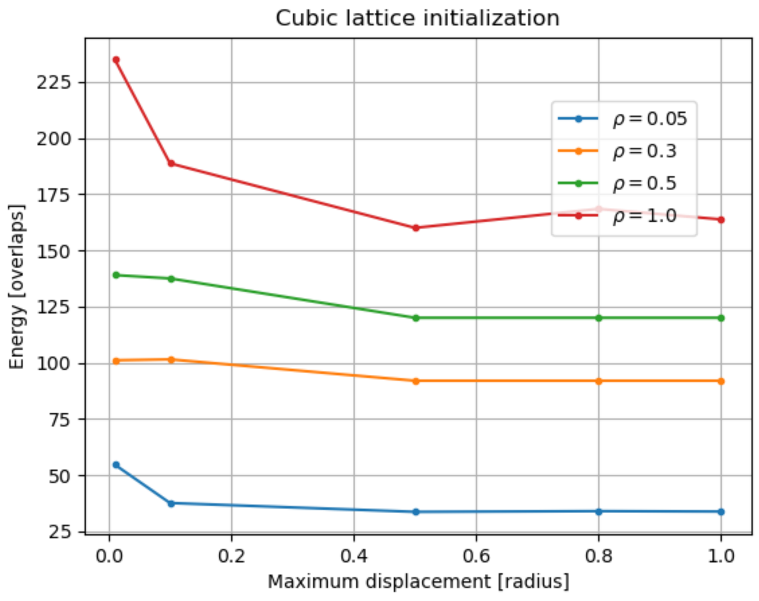

Figure 8.3: Energy overlaps as a function of the maximum displacement hyper-parameter with cubic-lattice initialization of the particles.

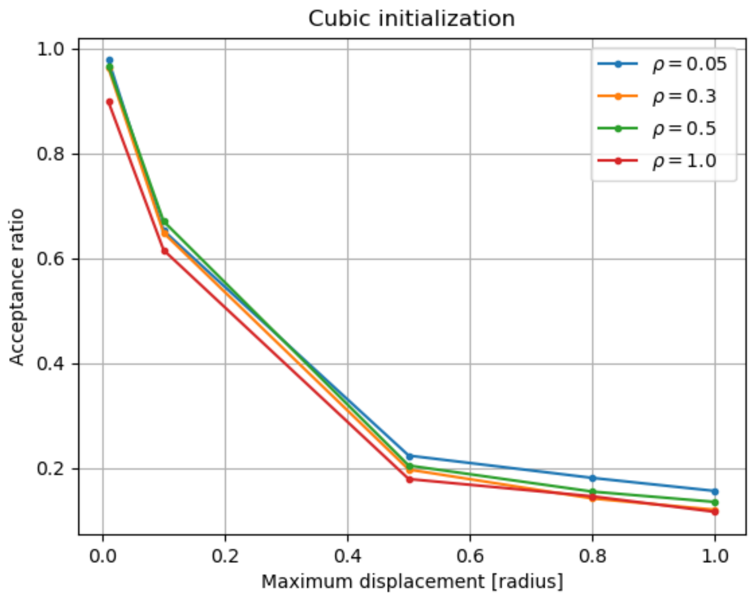

Figure 8.4: Acceptance Ration a function of the maximum displacement hyper-parameter with cubic initialization of the particles.

Ideally, we want a high acceptance ratio (to exploit at best the computational resources), and we want the system to relax to equilibrium as fast as possible, meaning that the energy is at its lowest. From the plots, we can see that, keeping fixed the number of MC sweeps, the energy (measured as number of overlaps) is lower with higher maximum displacements and seems to tend to a plateau in its value. However, the acceptance ratio is higher for small maximum displacement, suggesting that the choice of this hyperparameter needs to be made taking into consideration a trade-off between energy and acceptance ratio. For the number of overlaps we can see that using a non-random initialization can help, as in the cubic-lattice initialization we can see a smaller number of overlaps and the reach of the plateau with the maximum-displacement. As expected, at higher densities, the energy is higher (there are more overlaps), and the acceptance ratio is smaller (it is more probable that the random displacement brings an overlap), suggesting that the metropolis-algorithm is more efficient for small-density fluids.

8.4 Off-lattice Monte Carlo of Lennard Jones particles

Reference files in *cap8_Off_lattice_simulations_basic* folder:

- *OLMC_code_lennard_jones*(folder with the code and the output data);
- *data_processing_lenanrd_jones.ipynb* (jupyter notebook with analysis of simulations);
- *plots* (folder with plots).

In this exercise, I simulate with Monte Carlo a fluid of particles that interacts with the paradigmatic Lennard-Jones potential. I compute the pressure of the fluid at different densities at two Temperatures $T^* = 2.0, 0.9$ that are respectively above and below the critical temperature of a gas-liquid transition. The implemented potential is the one suggested in the text of the exercise with the tail corrections for energy (per particle) and for the pressure. I plot in the following figure the pressure-density diagram and compare it with the tabulated results for the EOS of a Lennard-Jones fluid.

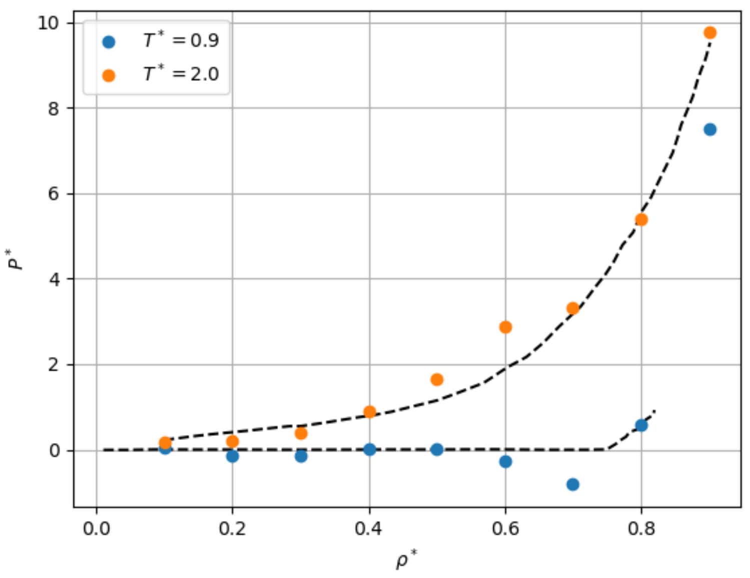

Figure 8.5: Pressure-Density diagram for a Lennard-Jones fluid in the liquid phase ($T^* = 0.9$) and in the gas phase ($T^* = 2.0$).

For each configuration I have simulated 5 realizations of 30000 Monte Carlo Sweeps, and for the averages I have discarded the first 5000 samples and applied a thinning of 10. From the results, we can see that the expected trend is followed in the gas ($T^* = 2$) phase, while in the liquid ($T^* = 0.9$) we observe a different trend, with negative pressures that are not feasible. The discrepancy between the tabulated EOS and the results of the simulations for $T^* = 2$ is probably due to the lack of statistics of my simulations. Another improvement could be to change the hyperparameter d_{max} based on the density. In fact, I have tuned it to $d_{max} = 0.3$ based on a test with low density, but the acceptance ratio of Metropolis with such d_{max} is very small 0.2 at higher densities.

Chapter 9

Integration schemes

9.1 Harmonic Oscillators

Reference files in *cap9_Integration_schemes* folder:

- *first_order_integrators.c*(code);
- *data*(folder with the simulations);
- *plots*(folder with the plots);
- *cap9_analytical_calculations.pdf*(pdf with analytical results of the exercise).

In this exercise I implement two simple first-order integrators, as suggested. The first one is shown to be non-symplectic while the second one is. The second one also has a constant of motion known as Shadow Hamiltonian. The analytical results of the exercise (points a and b) can be found in the file *cap9_analytical_calculations.pdf.* The code with the implementation of the integrators is in the file *first_order_integrators.c* and the simulations are saved in the folder *data.* Follows the analysis suggested by the text with the time steps $dt = 0.01$ and $dt = 0.001$. Note that the code for the analysis of the simulations is missing in this exercise due to a technical problem, but it consists only in plots of the time series saved in *data* and the absolute error is defined as $x - x_{exact}$ where x is the integrated value of the position, while x_{exact} is the analytical solution of the harmonic oscillator.

As we can see from the plots, the symplectic integrator has a lower absolute error using the same integration parameters. We note also that the symplectic integrator maintains a shadow Hamiltonian (as expected), while the non-symplectic one has a drift of the energy with time.

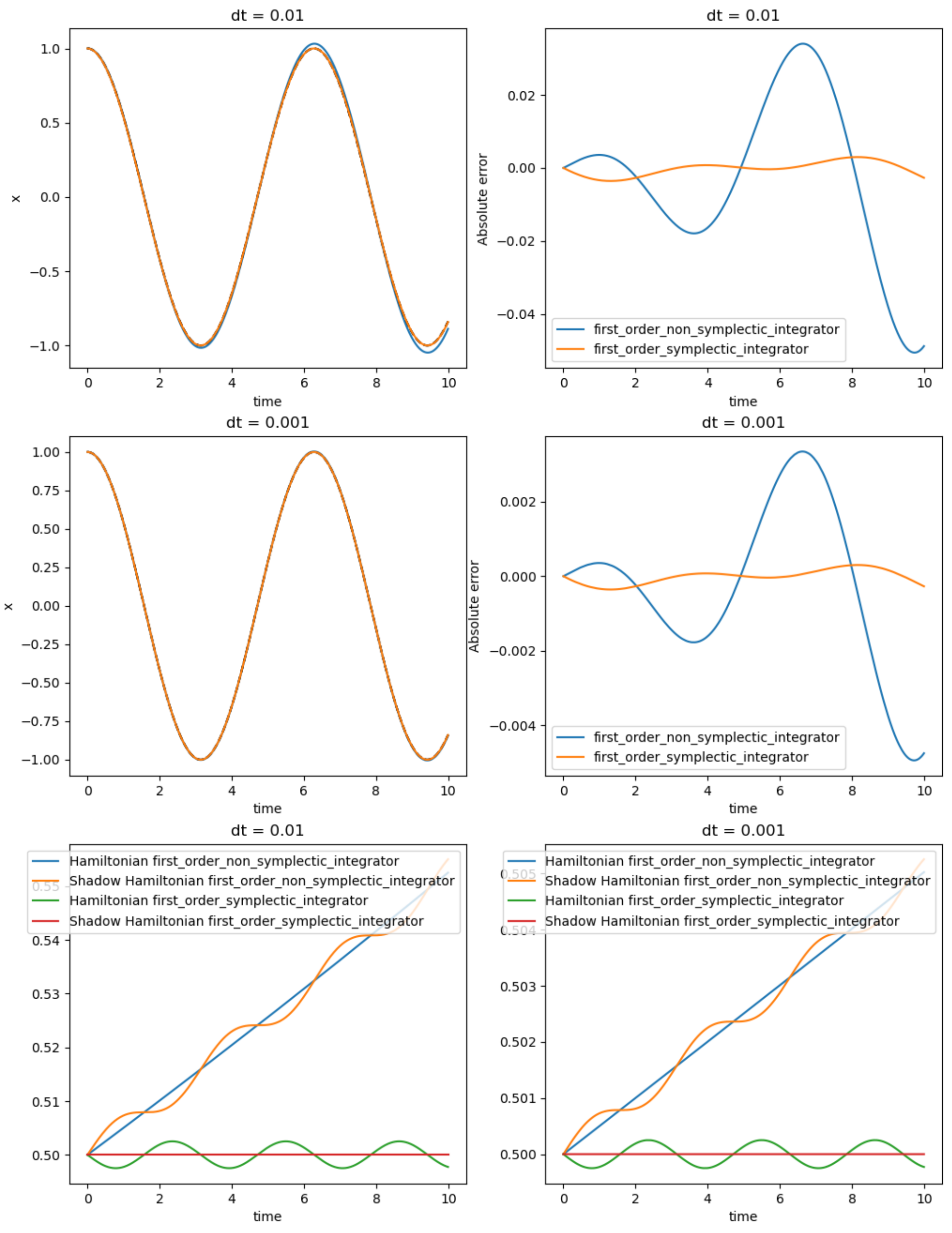

Figure 9.1: Results of the integration of the 1D harmonic oscillator with two simple first-order integrators. The focus is on the difference between the symplectic one and the non-symplectic-one.

9.2 Algorithms

Reference files in *cap9_Integration_schemes* folder:

- *higher_order_integrators.c*(code).

In this exercise, I have implemented the integration schemes Verlet and Velocity-Verlet for the 1D harmonic oscillator. The code can be found in the file *higher_order_integrators.c*.

9.3 The harmonic oscillator #2

Reference files in *cap9_Integration_schemes* folder:

- *higher_order_integrators.c*(code);
- *data*(folder with simulations);
- *plots*(folder with plots);
- *analysis_higher_order_integrators.ipynb*(jupyter notebook with analysis of simulations).

In this exercise, I have used the integrators implemented in the previous one to study the harmonic oscillator. The results presented in the following plots are obtained analyzing the simulations saved in *data* as done in *analysis_higher_order_integrators.ipynb*.

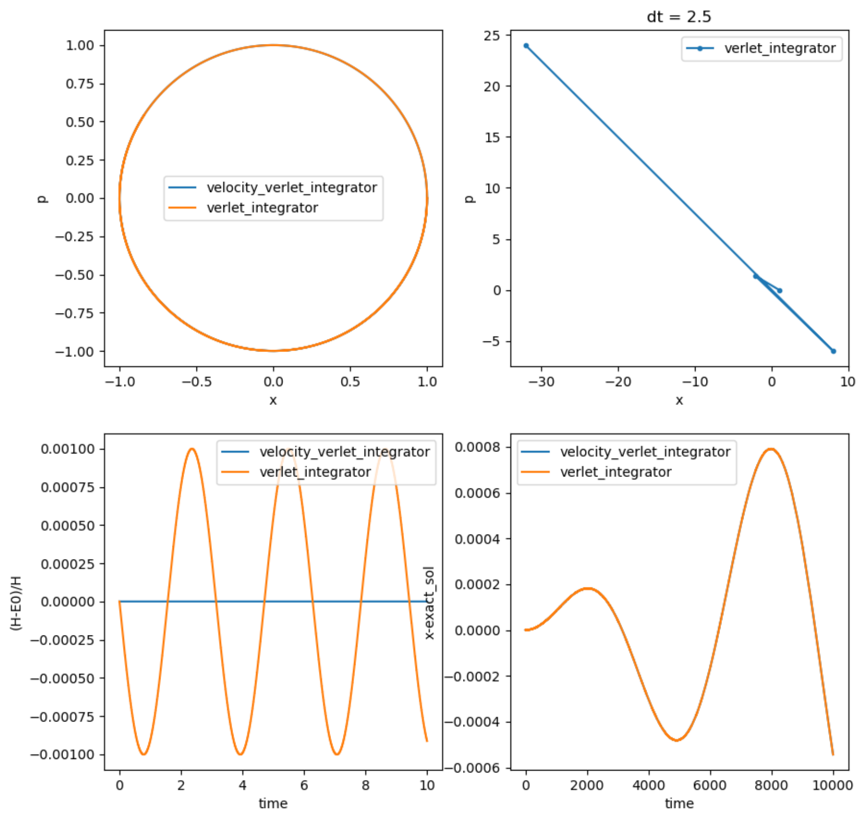

Figure 9.2: Results of the integration of the 1D harmonic oscillator with the Verlet integrator and the Velocity-Verlet integrator. In the upper-left plot we can see the phase portrait of the harmonic oscillator ($m = 1$, $k = 1$). In the upper right we observe the instability of the Velocity-Verlet integrator in the case $\omega dt > 2$. In the lower-left we plot the relative error of the energy respect to the initial energy for the two integrators. In lower-right we see the absolute error of the trajectories for the two integrators.

As expected, the trajectory in the phase space is a circle ($m = 1$, $k = 1$) while if the time steps dt are such that $\omega dt > 2$ the Velocity-Verlet algorithm is unstable. We can also see that the Velocity-Verlet integrator is symplectic, while this is not true for the Verlet algorithm. Finally, we see that the discrepancy between the integrated value of the position x and the exact solution x_{exact} of the harmonic oscillator increases with time for both integrators.

Chapter 10

Interaction potentials & thermostats

10.1 Exercise: Canonical Fluctuations

Reference files in *cap10_Interaction_potentials_and_Thermostats* folder:

- *ex_10_1.pdf* (analytical calculations).

The exercise is solved in the file *ex_10_1.pdf*.

10.2 Lennard-Jones fluid in the microcanonical ensembles.

Reference files in *cap10_Interaction_potentials_and_Thermostats* folder:

- *code_Lennard_Jones_fluid*(folder with the code for this exercise);
- *code_Lennard_Jones_fluid*
  output(folder with simulations data);
- *analysis.ipynb*(Jupyter notebook with analysis of results);
- *plots*(folder with plots of simulations);

In this exercise, I have simulated a Lennard Jones fluid of $N = 200$ particles of mass $m = 1$ in a box of length $L = 10$ with periodic boundary conditions. The lennard-jones potential is

$$V_{LJ}(r) = 4\epsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right] \tag{10.1}$$

where we have used $\epsilon = 1$ and $\sigma = 1$. We have also introduced a cut-off r_c of the potential, changing it from $r_c = 1.2$ to $r_c = 4$ with a step of 0.2. The density of the system is $\rho = 0.2$ and we are interested in studying how the distribution of particles changes at equilibrium when changing r_c. The positions of the particles have been initialized as a cubic lattice evolved for 100 Monte Carlo sweeps (using the code developed in Chapter 8). The velocities were sampled from a Maxwell-Boltzmann distribution with $T = 1$ as described in the lecture notes. The system has then evolved with the velocity-verlet algorithm for $N = 1000$ steps with timestep $dt = 0.01$. Finally, the radial distribution function was computed as described in the lecture notes at every step and averaged at the end over all the steps. In the following figures, we present the results for some of the r_c considered. The other plots can be found in the folder *plots*.

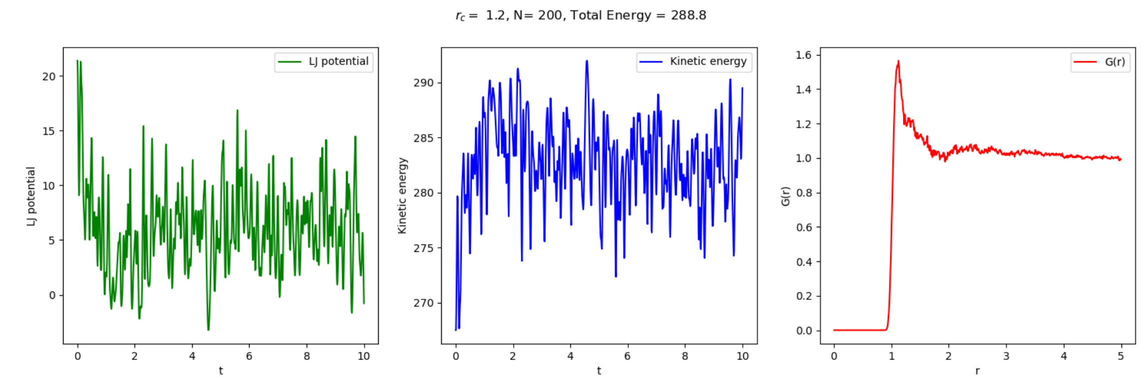

Figure 10.1: Results for energy and distribution function for Lennard-Jones fluid in the Microcanonical Ensemble with cut-off $r_c = 1.2$.

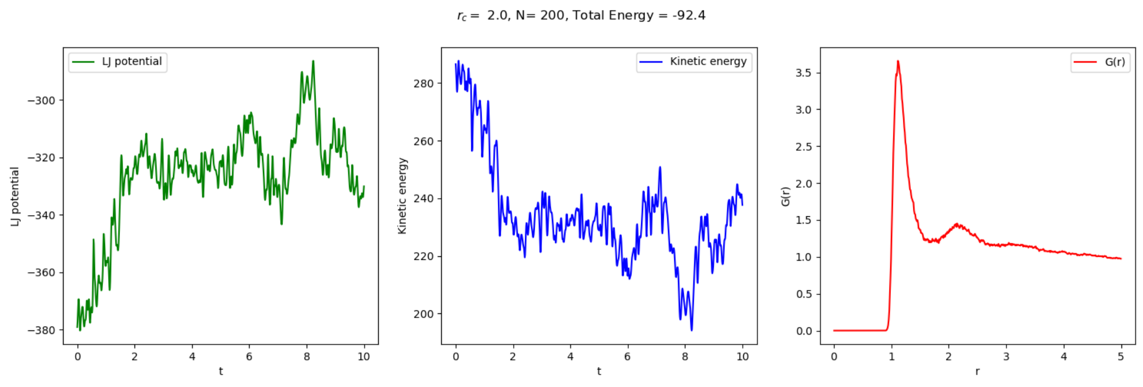

Figure 10.2: Results for energy and distribution function for Lennard-Jones fluid in the Microcanonical Ensemble with cut-off $r_c = 2.0$.

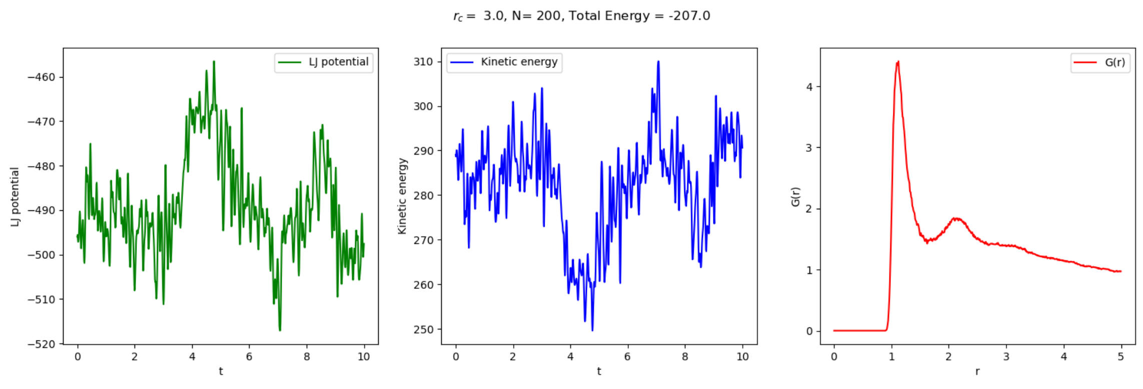

Figure 10.3: Results for energy and distribution function for Lennard-Jones fluid in the Microcanonical Ensemble with cut-off $r_c = 3.0$.

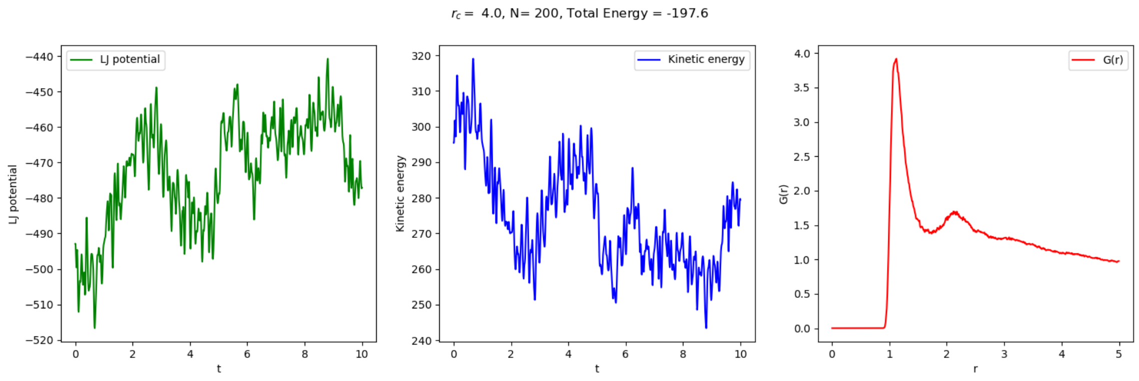

Figure 10.4: Results for energy and distribution function for Lennard-Jones fluid in the Microcanonical Ensemble with cut-off $r_c = 4.0$.

From the plots, we can see that when r_c is small and of the order of σ (e.g. $r_c = 1.2$), the interactions are mainly repulsive and therefore the system is the gas phase. This is shown by a positive energy of the system and a radial distribution function without a structure of neighbors. Increasing r_c, the system displays a bonded state ($E < 0$) and a structure in the radial distribution function and a structure of first neighbors in the radial distribution function, suggesting to be in the liquid phase.

10.3 Exercise 10.3 A

Reference files in *cap10_Interaction_potentials_and_Thermostats* folder:

- *code_Lennard_Jones_fluid_with_thermostats*(folder with the code for this exercise);
- *code_Lennard_Jones_fluid_with_thermostats output*(folder with simulations data);
- *analysis_thermostats.ipynb*(Jupyter notebook with analysis of results);
- *plots*(folder with plots of simulations);

In this exercise, I have followed the suggestions in the text and implemented two different choices as thermostats: Velocity Rescaling and Andersen Thermostat. The code for the Lennard Jones Fluid simulation is the same as for the previous exercise, and in the file *thermostats.h* I have added the new functions used. In the file *main.c* I have removed unnecessary parts for this exercise and added the line for the thermostats that is inside the integration cycle. From the simulations I save the time series of the kinetic energy in a txt file saved in the folder *output*.

I have considered the following number of particles $N = 50, 75, 100, 125, 150, 175, 200$ in a box of length $L = 100$, so that the density of the fluid is $\rho \leq 2$. The initial conditions are set as in the previous exercise and the simulations are computed for 5000 timesteps with a step $dt = 0.01$. As frequency for the Andersen Thermostat I have chosen $\omega = 10$. In the following figures, I present the results for the two cases.

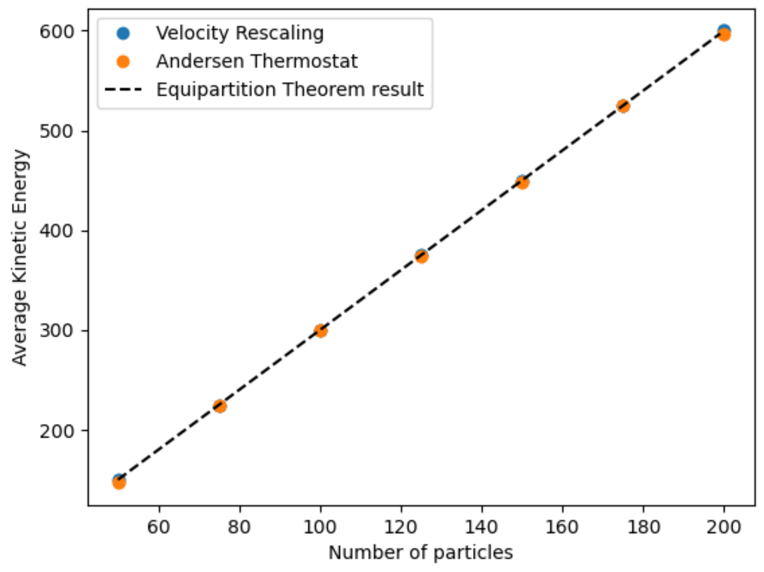

Figure 10.5: Plot of the average kinetic energy of the systems in the simulations as a function of the number of particles. Two different strategies for the thermostat are considered.

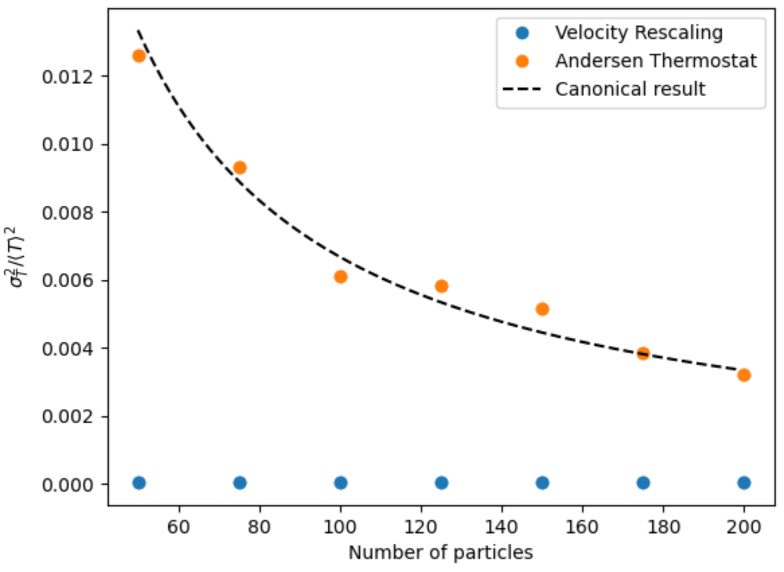

Figure 10.6: Plot of fluctuation of kinetic temperature of the systems in the simulations as a function of the number of particles. Two different strategies for the thermostat are considered.

From the first plot, we can see that both strategies reproduce Kinetic Energy expected from the theory of equipartition theorem. The second plot shows instead that only a canonical thermostat, as the Andersen Thermostat, reproduces correctly the fluctuations of temperature ad described by the theory of Canonical Ensemble. Note that better results could be obtained with longer and repeated simulations.

Chapter 11

Langevin and Brownian Dynamics

11.1 On the validity of the Langevin approach

Reference files in *cap11_Langevin_and_Brownian_Dynamics* folder:

- *ex_11_1.pdf* (pdf with the hand-written calculations of the exercise).

The solution of the exercise can be found in the file *es_11_1.pdf* with hand-written calculations.

11.2 Simple Brownian motion

Reference files in *cap11_Langevin_and_Brownian_Dynamics* folder:

- *code_brownian_motion*(folder with source code);
- *code_brownian_motion*
  output(folder with saved simulations);
- *plots*(folder with save plots);
- *analysis_brownian_motion*(jupyter notebook with analysis of simulations).

In this exercise, I have implemented two integrators for the Langevin dynamics: the first one is a first-order integrator and it can be used in the overdamped limit, while the second one is a second-order integrator and it is useful in the underdamped limit. In both cases I have considered different temperatures from $T^* = 0.1$ to $T^* = 1.9$ keeping fixed $\gamma\tau = 1.$ and different values of the friction coefficient from $\gamma\tau = 10$ to $\gamma\tau = 90$ keeping fixed $T^* = 1$. For every simulation I have considered 500 noninteracting particles inside a box of length $L = 20\sigma$ with $\sigma = 1$ with periodic boundary conditions. As initial condition I have chosen to start the particles in the center of the box. In the underdamped case, I have initialized the velocities according to the stationary Maxwell-Boltzmann distribution.
As osservable I have considered the Mean Square Displacement (MSD) $\langle (x(t) - x_0)^2 \rangle$. The results are shown in the following figures.

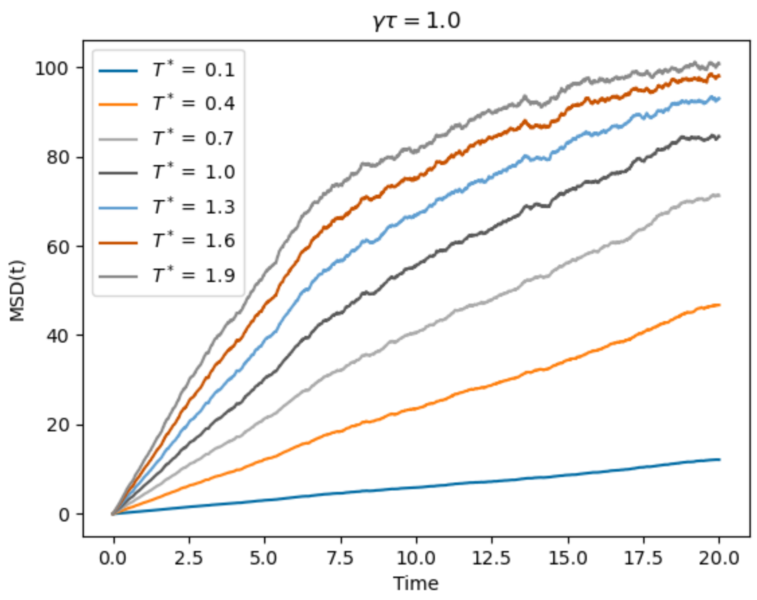

Figure 11.1: Mean Square Displacement in the overdamped case for different Temperatures.

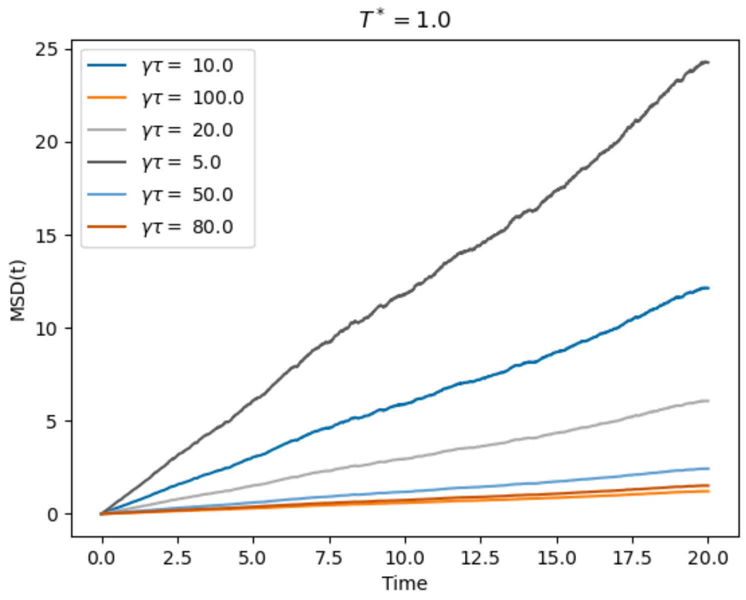

Figure 11.2: Mean Square Displacement in the overdamped case for different values of friction coefficient γ.

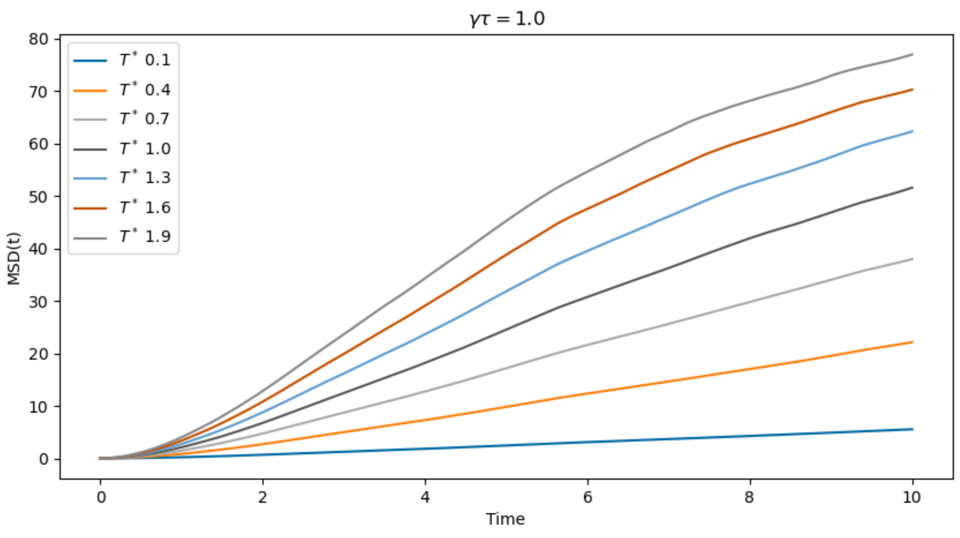

Figure 11.3: Mean Square Displacement in the underdamped case for different Temperatures.

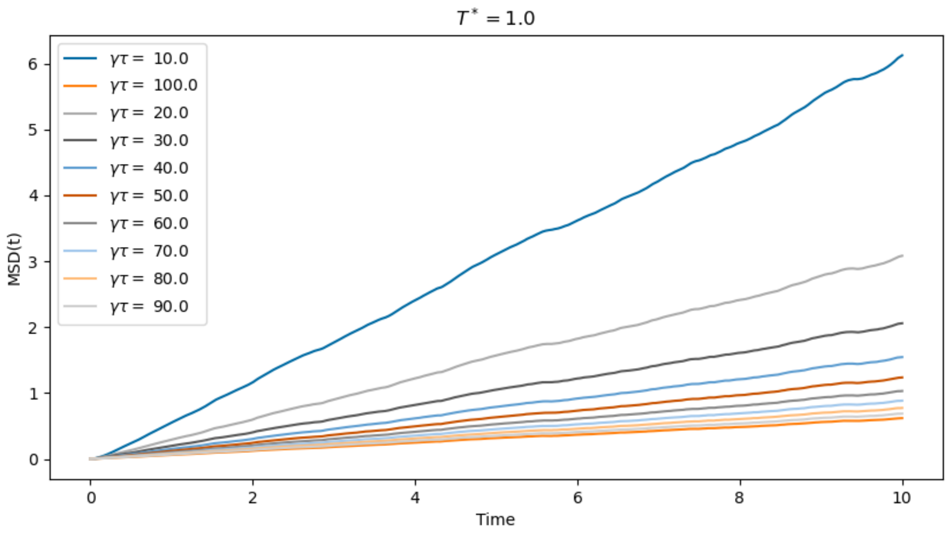

Figure 11.4: Mean Square Displacement in the underdamped case for different values of friction coefficient γ.

The simulations represent the average over different trials of a Brownian motion of a particle starting from the center of a box with periodic boundary conditions. This can be seen as a diffusion process, and accordingly we expect $MSD \propto t$, where the proportional constant is 2 times the diffusion constant D. From the figures, we can see the linearity of MSD respect to time in the first part of the simulation and a saturation of the MSD when the particles reach the extremes of the box. Changing T^* and γ we can also see an agreement with the Einstein relation $D\gamma = k_B T$: increasing T keeping fixed γ the diffusion coefficient increases and increasing γ keeping fixed T the diffusion coefficient D decreases. In the underdamped limit we can see that at the beginning of the simulations there is a trend of the type MSD $\propto t^2$, i.e. the ballistic regime.

11.3 Exercise

11.3.1 One Trap

Reference files in *cap11_Langevin_and_Brownian_Dynamics* folder:

- *code_colloide_harmonic_trap*(folder with source code);
- *code_colloide_harmonic_trap*
  output(folder with saved simulations);
- *plots*(folder with save plots);
- *analysis_harmonic_trap*(jupyter notebook with analysis of simulations).

In this exercise, I consider a 2D box of length $L = 20\sigma$ with a harmonic trap placed at the center $r_c = (10, 10)$ with potential

$$V(r) = \frac{1}{2}(\vec{r} - \vec{r}_c)^2. \tag{11.1}$$

I consider $N = 400$ realizations of 10^4 timesteps of length $dt = 0.01$ with parameters $\sigma = 1$, $\epsilon = 1$ and $m = 1$. The particles are randomly initialized on a radius circle $r = 3$ around the center of the trap with a velocity distributed according to the equilibrium distribution. The following is an example of a simulation:

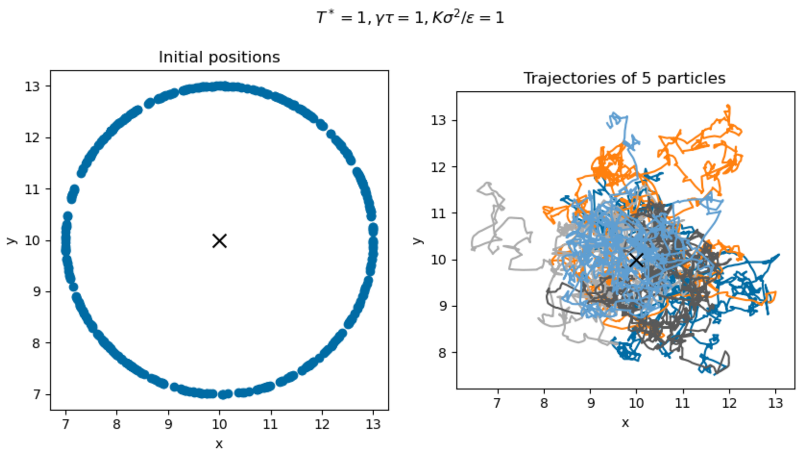

Figure 11.5: Example of simulation of a colloid in an harmonic trap with parameter $T^* = 1$, $\gamma\tau = 1$ and $K\sigma^2/\epsilon = 1$. On the left the initial conditions for $N = 400$ realizations and on the right example of trajectory for 5 realizations.

In particular, I consider the average positions of the particle over the realizations and its standard deviation, as well as the autocorrelation time of the distance of the particle from the trap, along the two axis. I consider simulations at different values of the parameters T^*, $\gamma\tau$ and $K\sigma^2/\epsilon$ keeping one fixed and changing the others.

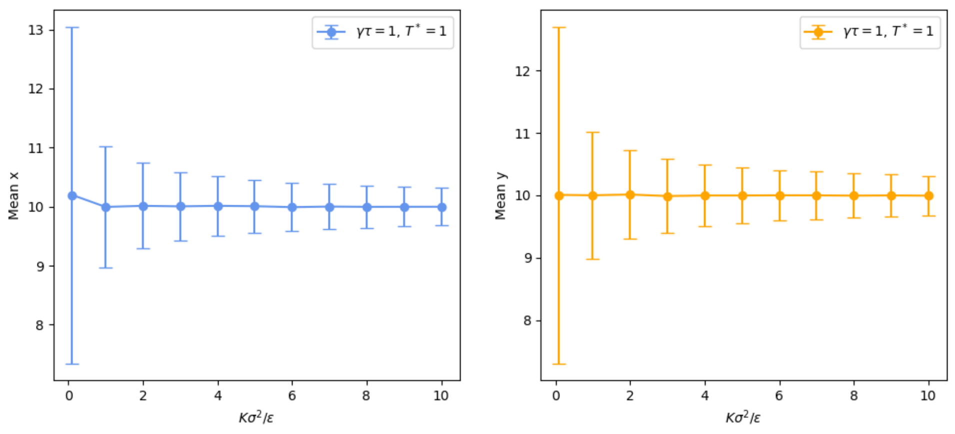

Figure 11.6: Average position and standard deviations of the simulations over 400 realizations changing the parameter $K\sigma^2/\epsilon$.

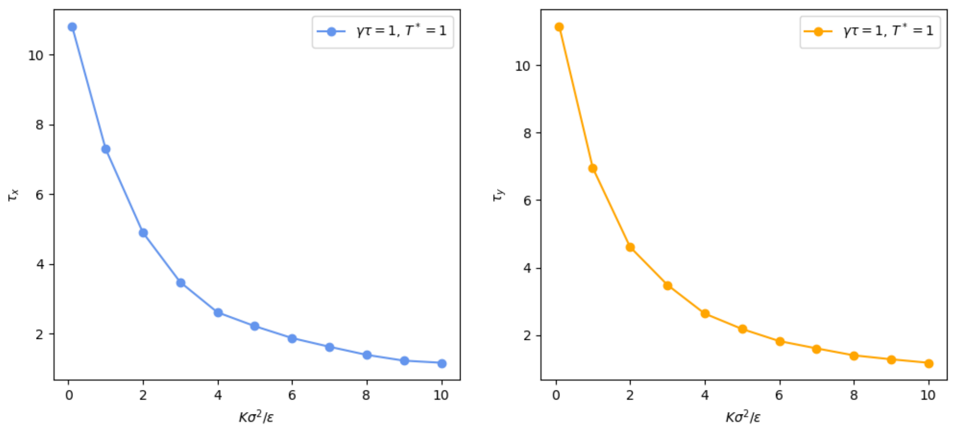

Figure 11.7: Autocorrelation time of the distance of the particle from the center of the trap over 400 realizations of the simulations changing the parameter $K\sigma^2/\epsilon$.

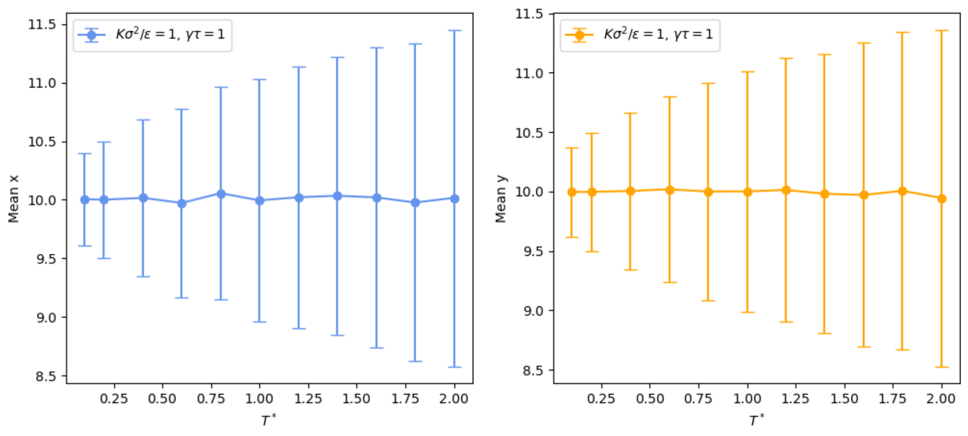

Figure 11.8: Average position and standard deviations of the simulations over 400 realizations changing the parameter T^*.

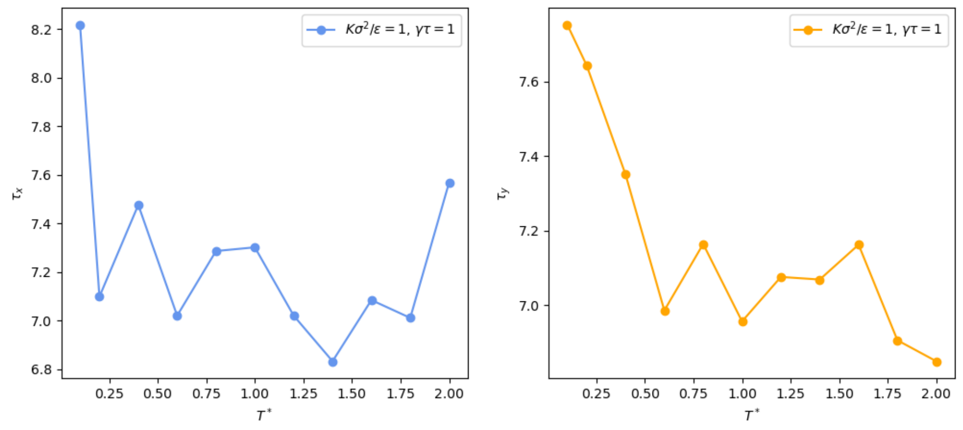

Figure 11.9: Autocorrelation time of the distance of the particle from the center of the trap over 400 realizations of the simulations changing the parameter T^*.

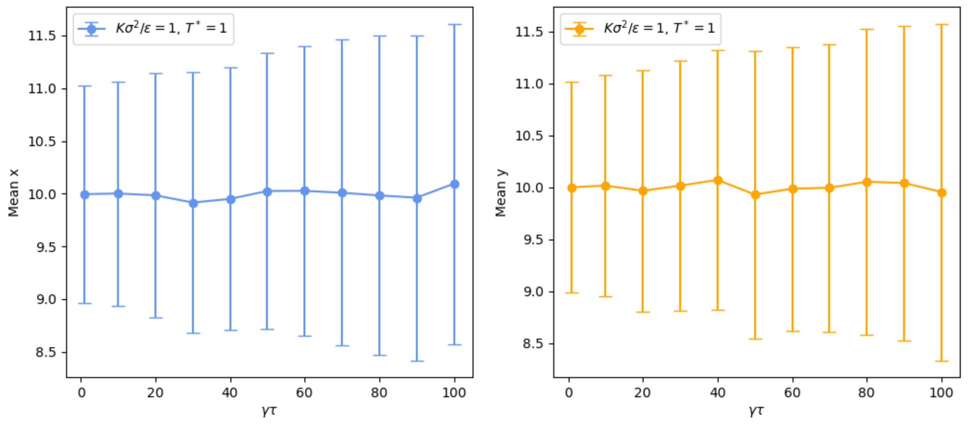

Figure 11.10: Average position and standard deviations of the simulations over 400 realizations changing the parameter $\gamma\tau$.

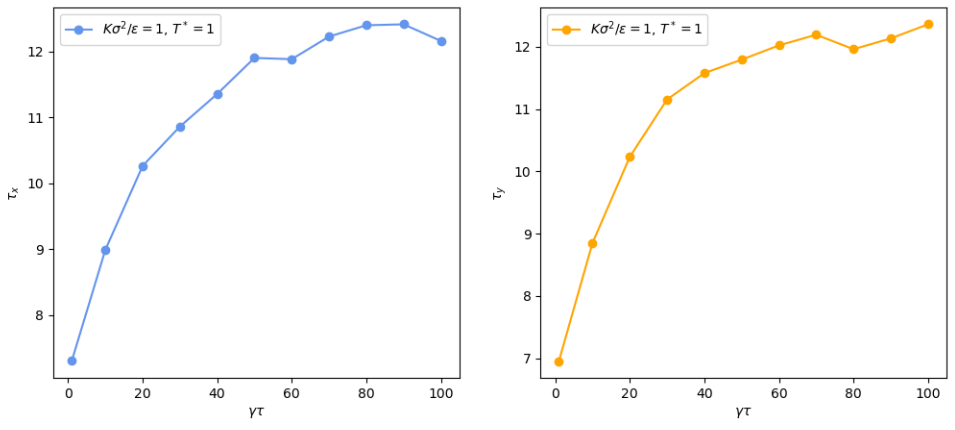

Figure 11.11: Autocorrelation time of the distance of the particle from the center of the trap over 400 realizations of the simulations changing the parameter $\gamma\tau$.

For the positions, we see that on average the particles are attracted to the center of the box, i.e.

on the center of the trap. The standard deviation is a metric of how spread the particles are around the center of the trap.
Increasing the temperature, we see that the standard deviation increases, as the particles have more energy to escape the trap. Moreover, the velocities on average are greater at higher temperatures and the autocorrelation time of the distance from the trap is lower.
Increasing the friction coefficient γ makes it more difficult for the particle to escape the trap, and therefore the standard deviation of the position decreases. Moreover, the velocities are lower and therefore the autocorrelation time of the distance from the trap increases.
Increasing the harmonic coupling K the particle is more attracted to the trap, making the standard deviation of the postion lower. Moreover, being in the neighborhood of the trap, the stochastic fluctuations of the position of the particle are more important, and the distance from the center of the trap has a lower autocorrelation time.

11.3.2 Double Harmonic Trap - Escape time

Reference files in *cap11_Langevin_and_Brownian_Dynamics*folder:

- *code_double_trap*(folder with source code);
- *code_double_trap*
  output(folder with saved simulations);
- *plots*(folder with save plots);
- *analysis_double_harmonic_trap*(jupyter notebook with analysis of simulations).

In this exercise, I consider two harmonic traps inside a 2D box of length $L = 20\sigma$ and positioned at $r_1 = (8, 10)$ and $r_2 = (12, 10)$. Then I have simulated the same particle as in the previous exercise with $m = 1$, $\sigma = 1$ and $\epsilon = 1$ for $N = 400$ realizations. In the following figure I plot two trajectories as an example.

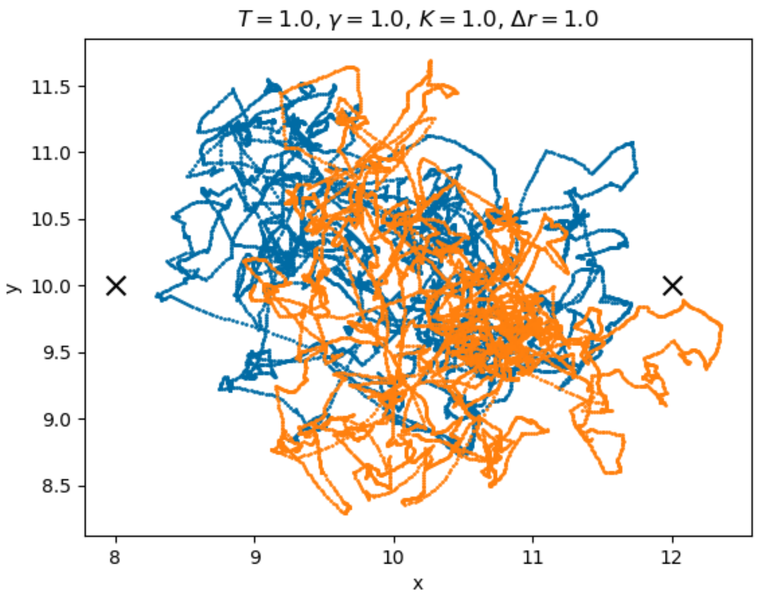

Figure 11.12: Example of trajectories of 2 particles trapped between two harmonic traps.

As initial position for the particle, I have considered $r = (8 + \Delta r, 10)$ with $\Delta r = 0, ...2.$ and I have studied the average Mean First Time Passage (MFPT) to go from the first trap to the second and its standard deviation.

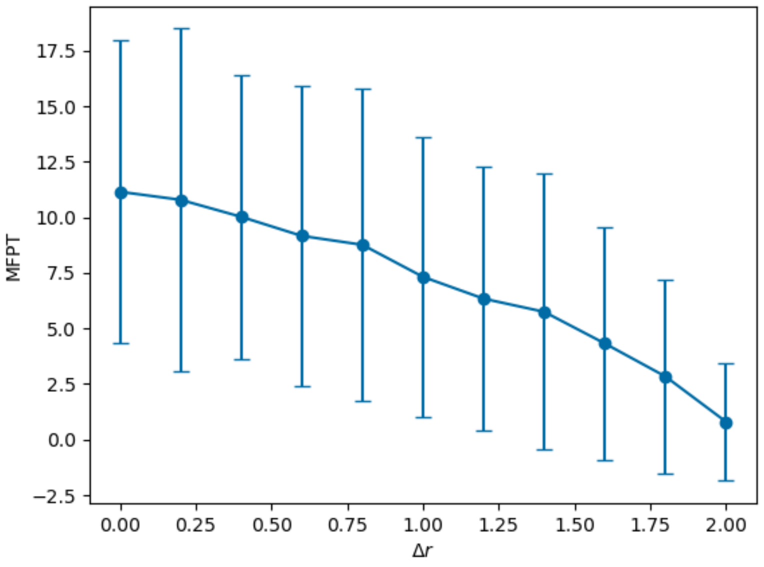

Figure 11.13: MFPT and its standard deviation changing the initial position of the particle.

As expected, the MFPT decreases by starting the simulation further from the first harmonic trap and also its standard deviation.

Chapter 12

Reweighting techniques

12.1 Change of measure

Reference files in *cap12_Reweighting_Techniques* folder:

- *ex_12_1.pdf* (pdf with the hand-written calculations of the exercise).

The solution of the exercise can be found in the file *es_12_1.pdf* with hand-written calculations.

12.2 Single Histogram Method

Reference files in *cap12_Reweiglattice, Techniques* folder:

- *reweighting.ipynb*(jupyter notebook with the implementation of the histogram method);
- *plots*(folder with the plofor
- *data_ISING_2D* (folder with the data of the simulations).

In this exercise, I use the data saved from Chapter 5 of the simulations of a 2D Ising model for different values of $\beta = 0.441, ..., 0.447$. The aim is to estimate the average energy at a temperature β' given the data from the simulations at a temperature β. The code that implements this algorithm can be found in the file *reweighting.ipynb*. The results are presented in the following figure.

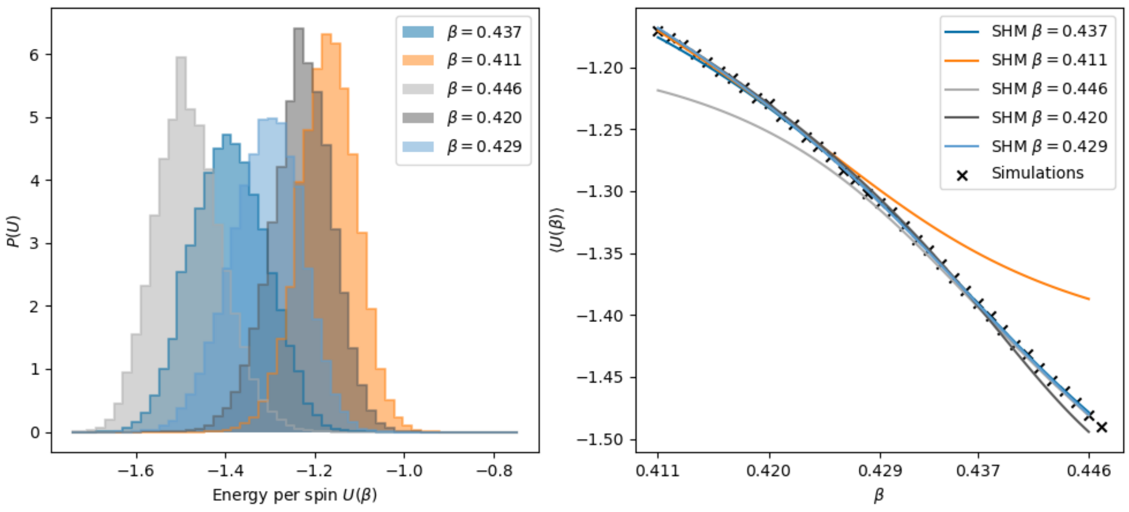

Figure 12.1: Single Histogram Method for reweighting. On the left: energy per spin distribution at the value of temperature chose to implement the methods. On the right: the estimation of the average energy per spin obtained different β compared with the actual data obtained from the simulations.

From the plot, we can see that if the energy distributions of the chosen temperatures overlap, the single histogram method gives good estimations in terms of reweighting the averages of observables. When the two energy distributions do not overlap much, e.g., for $\beta = 0.411$ and $\beta = 0.446$, using the single histogram methods yields poor results.
From the plot, we can see that this method works well in a small range around the temperature used for estimating the observables in the simulations, but when we exit from this range the estimation diverges from the real value of the averages.

12.3 Multiple Histogram Method (MHM)

Reference files in *cap12_Reweighting_Techniques* folder:

- *reweighting.ipynb*(jupyter notebook with the implementation of the histogram method);
- *plots*(folder with the plots)
- *data_ISING_2D* (folder with the data of the simulations).

In this exercise, I use the data saved from Chapter 5 of the simulations of a 2D Ising model for different values of $\beta = 0.441, ..., 0.447$. The aim is to estimate the average energy at a temperature β' given the data from the simulations at a temperature β and the specific heat at a temperature β for different sizes of the system. As values of the length of the square lattice, I have used $L = 10, 20, 30, 40$. The simulations are made up of 500000 timesteps. I have applied a burn-in of 1000 and a thinning of 200 in such a way that the samples used for histogramming are not correlated. The code that implements the multiple histogram method for reweighting can be found in the file *reweighting.ipynb*. The results are presented in the following figures.

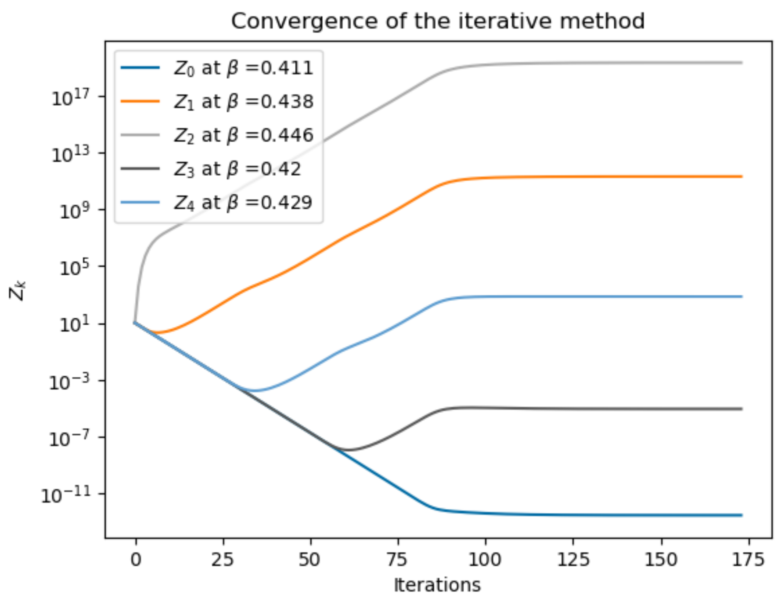

Figure 12.2: Convergence of the iterative method used to estimate the partition function Z_k at different temperatures β_k.

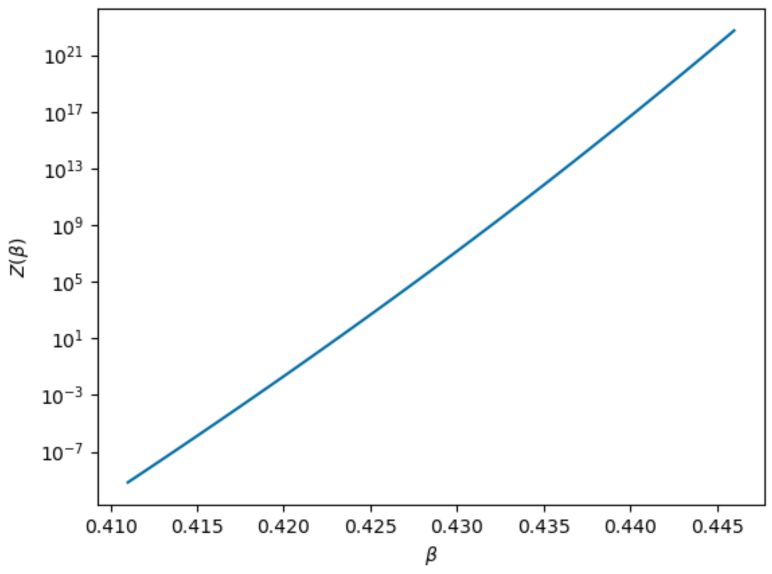

Figure 12.3: Plot of the partition function Z as a function of the inverse temperature β, extrapolated from the previous estimated Z_k.

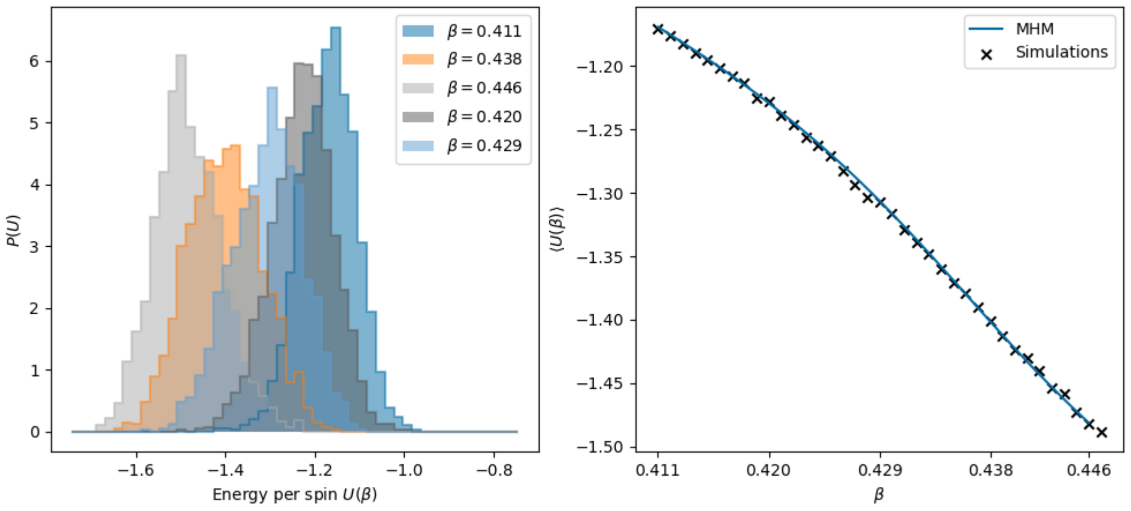

Figure 12.4: Average energy per spin at different inverse temperatures β obtained using the Multiple Histogramm Method.

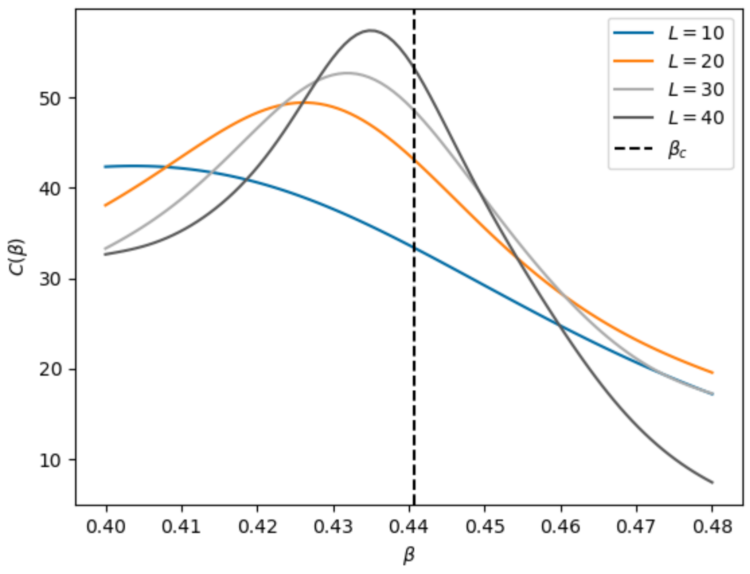

Figure 12.5: Estimated Specific Heat as a function of the Inverse temperature β from simulations with different sizes of the lattice L.

From the plots, we can see that the MHM interpolates the average energy per spin of the simulations in a more robust way than the SHM. We can also see that, as expected, increasing the size of the of the lattice L the bump of the specific heat approaches the theoretical critical temperature.

12.4 Umbrella Sampling (optional)

Reference files in *cap12_Reweighting_Techniques* folder:

- *umbrella_sampling.ipynb*(jupyter notebook with the implementation of the umbrella sampling method);
- *plots*(folder with the plots).

In this exercise, I have implemented the Umbrella sampling method for a simple system: a particle moving in 1D in a mexican hat potential

$$H(x) = -x^2 + x^4. \tag{12.1}$$

Firstly, I tried a simple Monte Carlo sampling for this system using metropolis. As observable, I have considered the average position. The results with this naive approach are presented in the following figure.

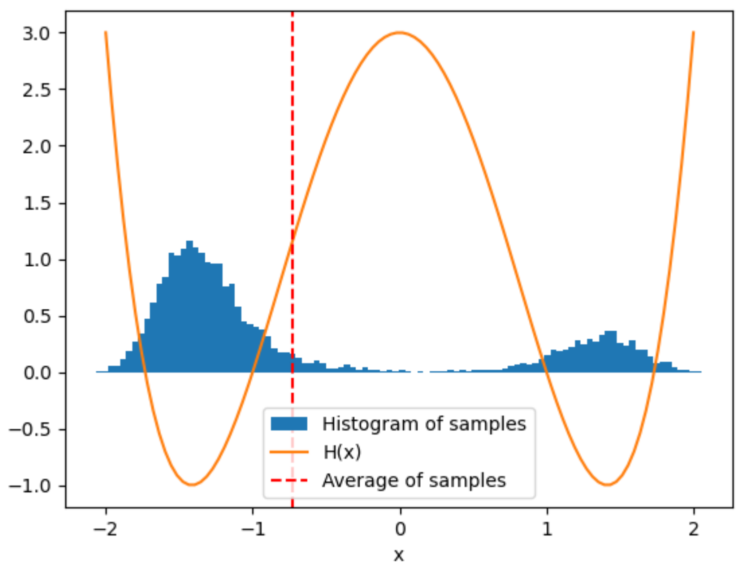

Figure 12.6: Samples obtained using the metropolis algorithm for a 1D system with mexican hat potential. The algorithm is not ergodic.

I have then implemented the Umbrella Sampling procedure introducing an auxiliary potential $U(x)$ obtained as minus a non-normalized gaussian function,

$$U(x) = -A \exp\left(-\frac{x^2}{2\sigma^2}\right), \tag{12.2}$$

the effective hamiltonian obtained is $H_{eff}(x) = H(x) + U(x)$:

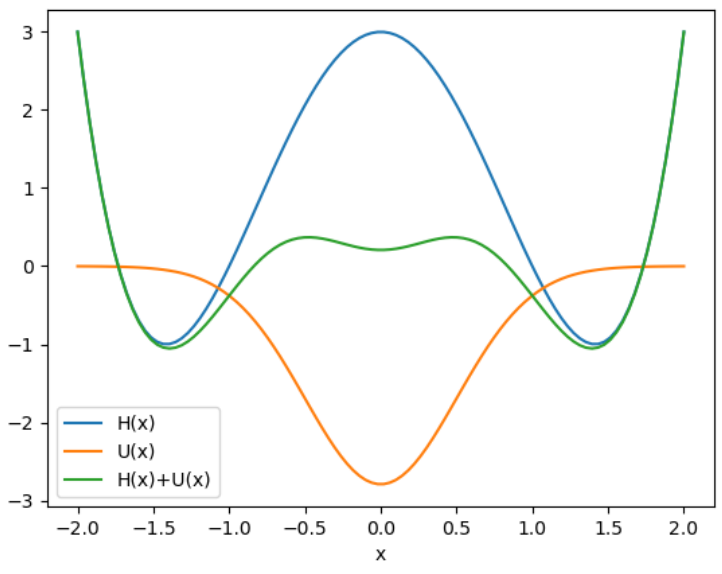

Figure 12.7: Hamiltonian of the problem, auxiliary potential and effective hamiltonian.

The results obtained using the effective Hamiltonian in the Umbrella sampling are presented in the following figure.

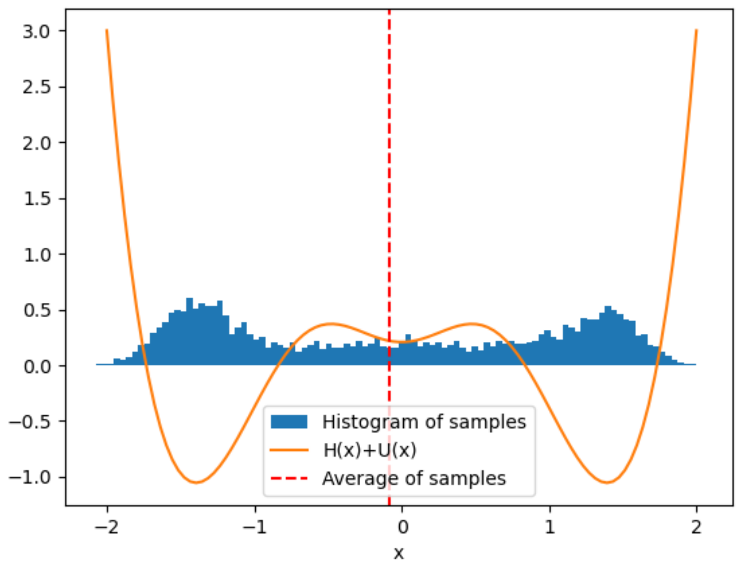

Figure 12.8: Sampling of the position using the Umbrella Sampling for a 1D system with mexican hat potential. The algorithm is ergodic.

As shown by the plot, using the Umbrella Sampling the algorithm is able to sample all the possible configurations of the system and it is therefore ergodic.

Chapter 13

Langevin simulation of many particles

13.1 Cell list

Reference files in *cap13_Langevin_simulation_of_many_particles* folder:

- *C-sim_modified*(folder with the given code modified);

The C code provided is an implementation of a simulation of N small particles and a larger probe, *particle 0*. Initially, particle 0 is placed at the center of a 2D box, while the other particles are placed randomly. The bigger particle experiences a harmonic force with respect to a trap initially placed in the center of the box that moves uniformly with a given velocity v_{trap}. The bigger particle interacts with all the small particles with a repulsive force that is exponential in the distance between particle 0 and small particles. The same force is experienced by the small particles, according to the third principle of mechanics. The small particles also interact with each other with a repulsive Gaussian force and experience thermal fluctuations. Their dynamic is simulated according to a Langevin equation. If the parameter $L_p = 1$ the small particles are independent, i.e. does not experience a polymer bond interaction.
To reduce the computational cost of the simulations, the cell-list strategy is implemented. In particular, in this case we have to consider two types of pair interactions. The first is the interaction of particle 0 with the other small particles. The second one are the interactions between the small particles. For the second type of interaction the strategy implemented is the standard cell-lists, where is registered which particles are sitting in which cells. For the first type of interaction, the implemented strategy consists of keeping track of the small particles that sit in the cell Dv0 relative to the position of the particle 0. This second strategy is useful since the interaction considered is particle 0 - small particles and we can therefore easily compute the forces considering the relative distance particle 0 - small particles.

13.2 Active matter

Reference files in *cap13_Langevin_simulation_of_many_particles* folder:

- *C-sim_modified*(folder with the given code modified);
- *plots*(folder with the plots of the results);
- *analysis.ipynb*(jupyter notebooke with the analysis of the simulations).

In this exercise, I have modified the reference code given to implement a simple simulation of an example of active matter. The modified code can be found in the folder *C-sim_modified.* In these simulations, the small particles are organized in pairs, each pair representing a bacterium, while particle 0 is a probe. The organization in pairs has been done by setting the parameter $L_p = 2$ and the interaction between small particles has been modified to have a harmonic force with a resting length $\Lambda = 0.5$. I have also modified the code to introduce an active propulsion force of magnitude f, directed from the first particle of a pair to the second. In the following figure I report a snapshot of one of the simulations.

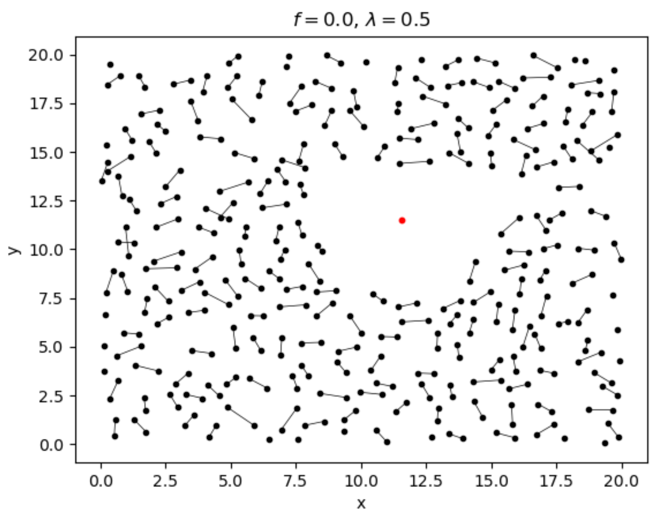

Figure 13.1: Snapshot of the box of one of the simulations at an intermediate time. The black dumbells represent the bacterium while the red particle is the probe. Note that the black particles not connected with a segment are actually connected with another particle that is projected to another edge of the box by the boundary conditions.

In particular, we want to study the Mean Square Displacement (MSD) of the probe as a function of the magnitude of the propulsion force f. I have considered the values $f = 0, 2, 4, 6, 8, 10$ for the propulsion force and repeated the simulations 10 times for each combination of parameters. During thermalization, a harmonic trap was used to keep the probe in the center of the box. After thermalization, the stifness of the trap was set to zero, $k_{trap} = 0$. In the following figure I present the results.

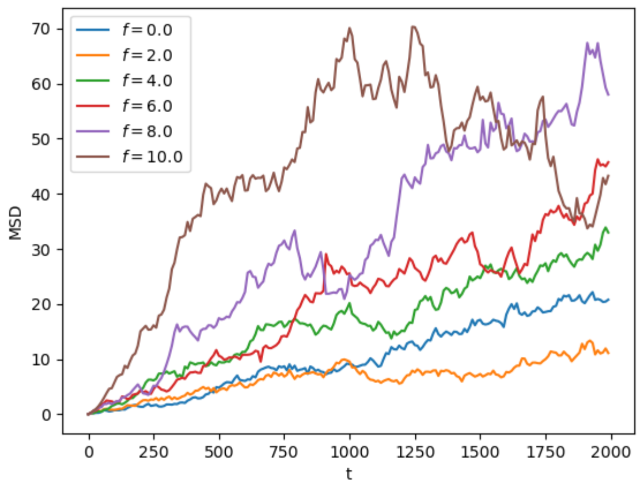

Figure 13.2: Mean Square Displacement of the probe unite inside a field of bacterium changin the propulsion force of the bacterium f.

From the plot we can see that when the propulsion force increases, the MSD of the probe particle increases, even though it is subject to fluctuations that should be averaged on more realizations (here we repeat each simulation 10 times). This trend is expected because having the bacterium a higher propulsion force, they act as a bath with higher thermal energy and the fluctuations on the probe that follow are also higher, yielding a higher Mean Square Displacement.