Assignment1

Nicola Perin

April 11th, 2023

Contents

1 1D Random walks: properties; comparison numerical/analytical results; convergence

1.1 (a) Plot the instantaneous position for runs corresponding to different seeds

The following are the plots of the instantaneous position and the squared instantaneous position of 5 random walkers generated with multiple runs changing the seed of the random number generator each time. Since the probability of going left or right are equal, the expected behaviour for x_i is to stay around zero, and the expected behaviour for x_i^2 is to grow linearly with the number of steps. We can see from the plots how a single walker can deviate substantially from the expected behaviour.

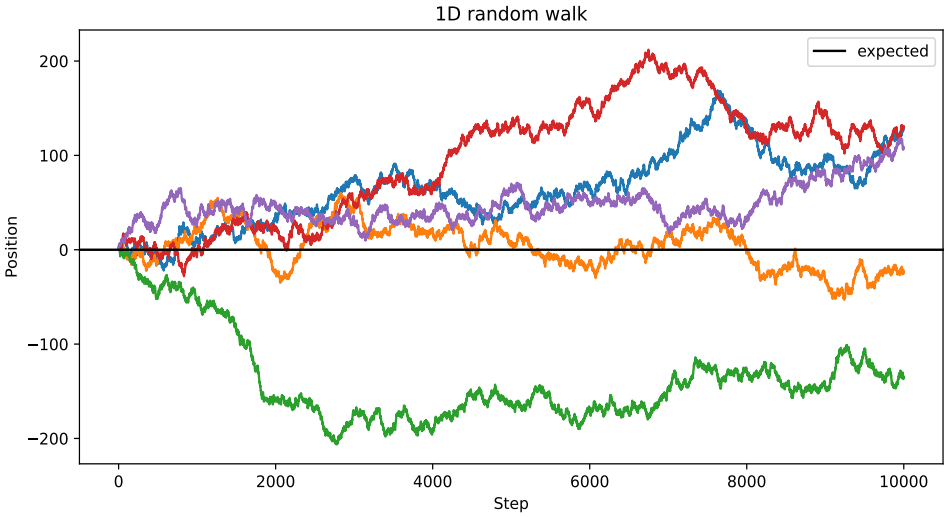


Figure 1: Plot of the instantaneous position x_i of 5 different random walkers

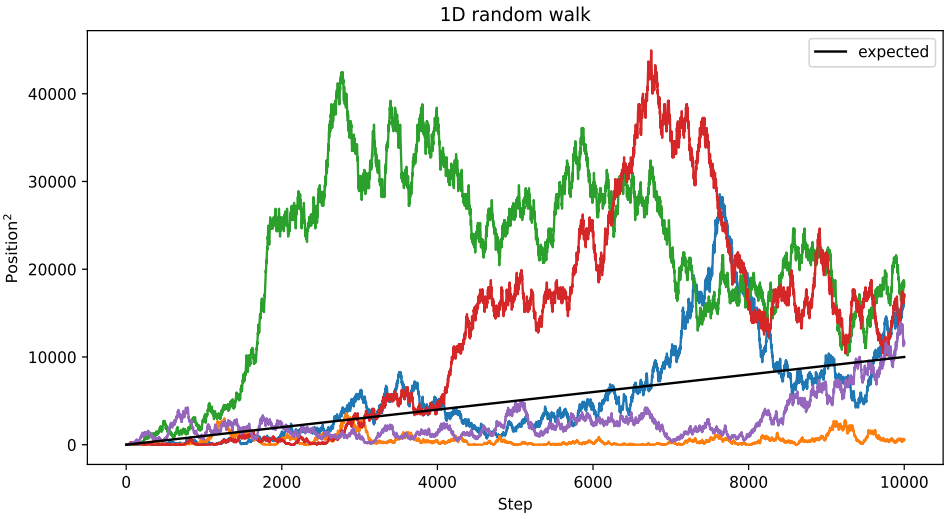


Figure 2: Plot of the instantaneous x_i^2 for runs corresponding to 5 different seeds

Now we look at how the average final values $< x_N >$ and $< x_N >^2$ compare with the theoretical values, which in our case are 0 and N respectively, as the number of runs increases exponentially from 10^1 to 10^5. The scale of the x-axis is logarithmic since i chose exponentially equispaced points.

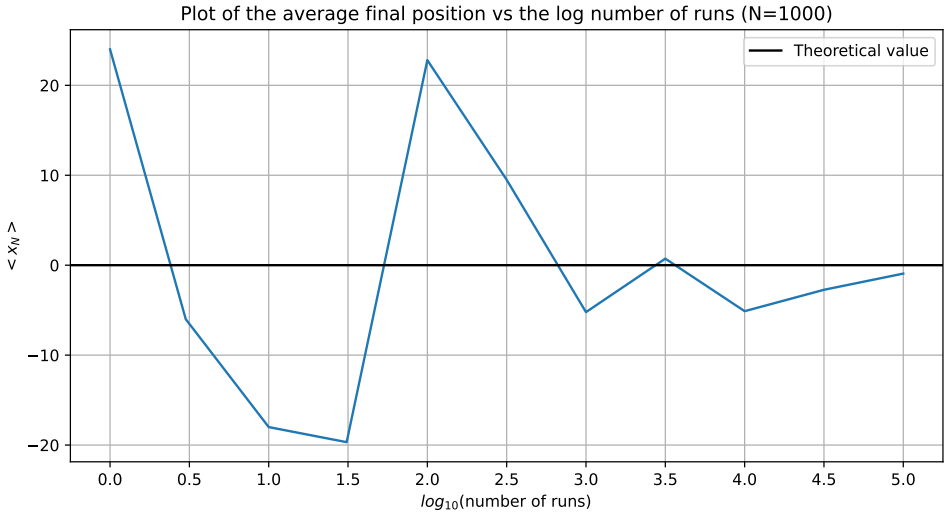


Figure 3: Plot of the average final x_N over many runs

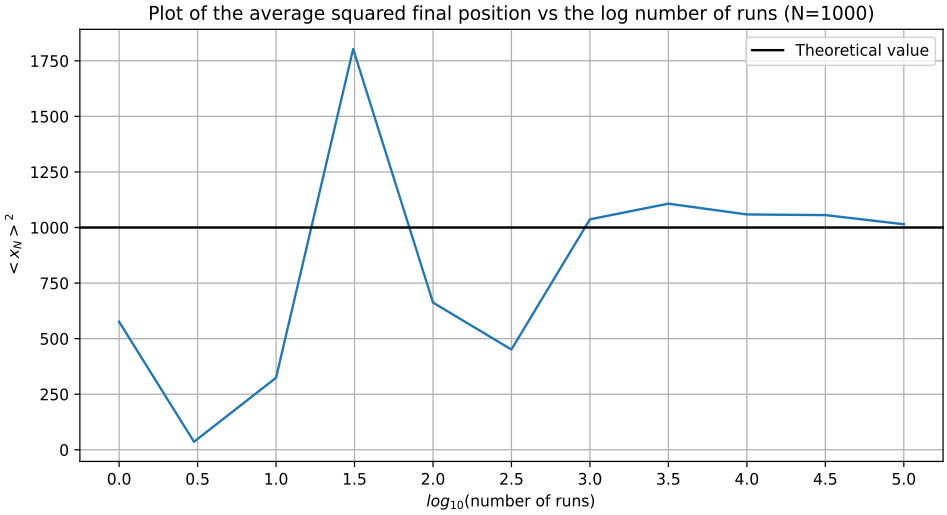


Figure 4: Plot of the average final x_N^2 over many runs

We can clearly see how as the number of runs increases, the average values converge to the theoretical one.

1.2 (b) Averages over many walkers

The following plots show the behaviour of the average instantaneous quantities $< x_i >$ and $< (\Delta x_i)^2 >$ over a large number of runs.

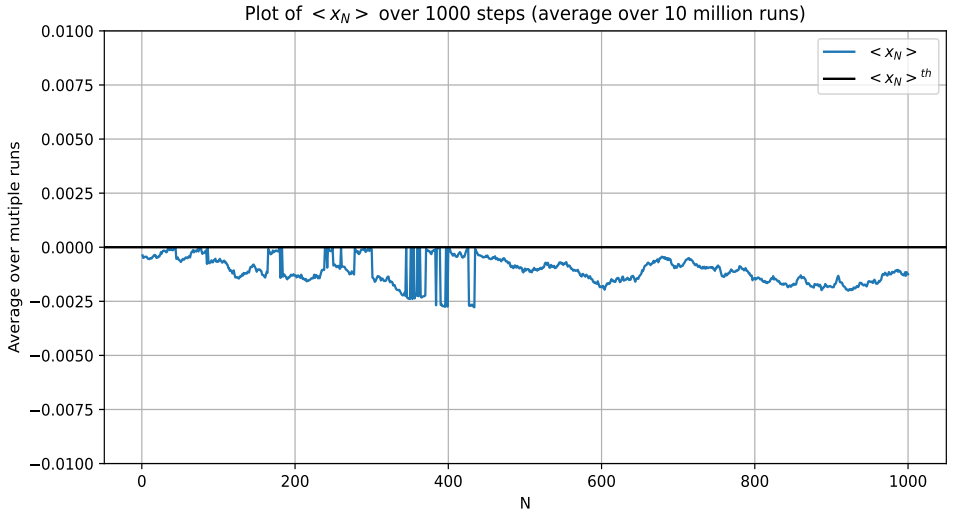


Figure 5: Plot of $< x_i >$ vs i

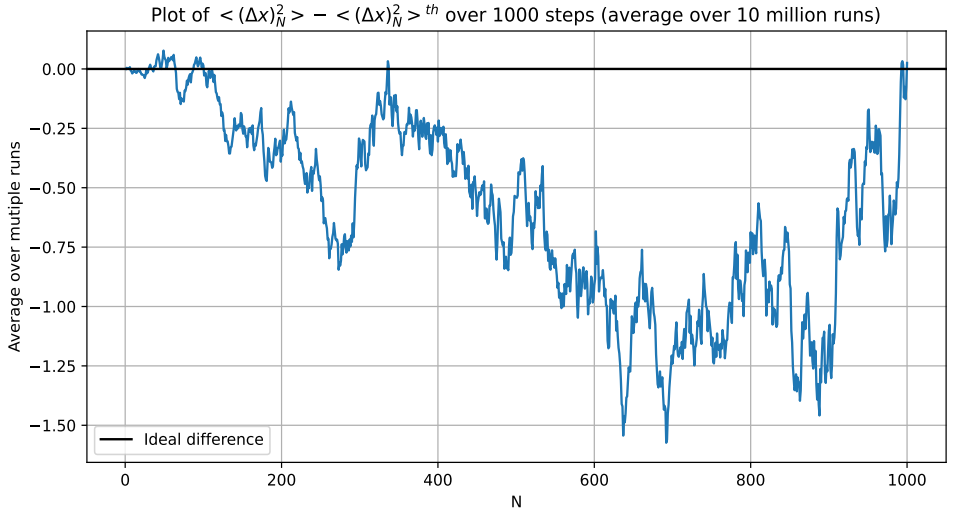


Figure 6: Plot of $< (\Delta x_i)^2 > -i$ vs i

In this case, the behaviour of both $< x_i >$ and of $< (\Delta x_i)^2 >$ is really close to the expected one. For this reason, the scale of the y-axis is restricted to an interval of the same order of magnitude of the plot. In the second case, the way to do this is to plot the difference with the ideal value i instead of just $< (\Delta x_i)^2 >$.

1.3 (c) Accuracy of the mean square displacement

This plot shows how the larger is the number of walks for the average, the smaller $\Delta = |\frac{<(\Delta x_N)^2>^{calc}}{<(\Delta x_N)^2>^{th}} - 1|$ is. For $N = 1000$, at least 1000 runs are needed to have an accuracy $\Delta \leq 5\%$.

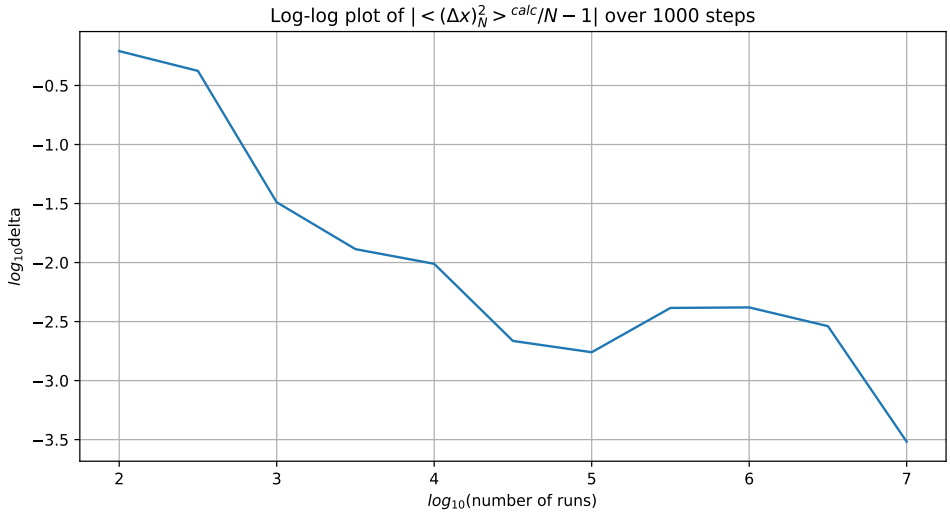


Figure 7: Plot of Δ vs i

1.4 (e) Dependence of $< (\Delta x_N)^2 >$ on N

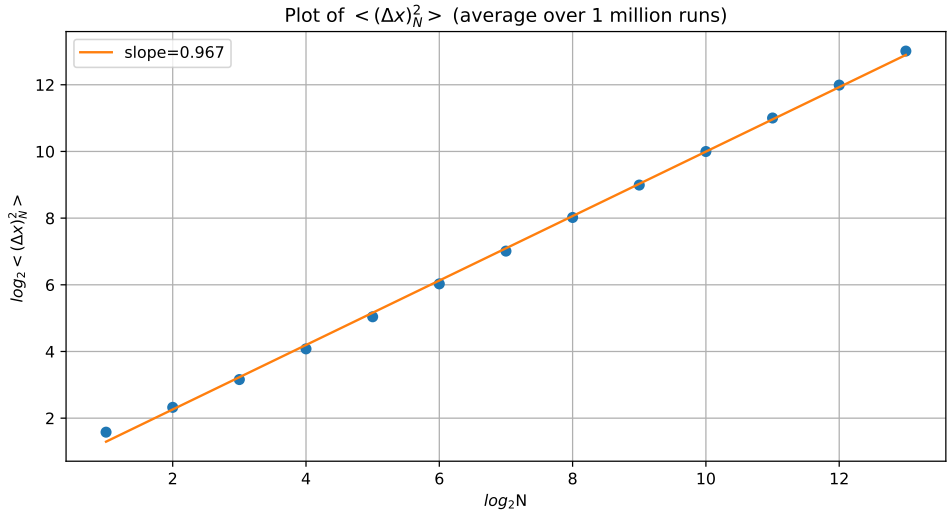
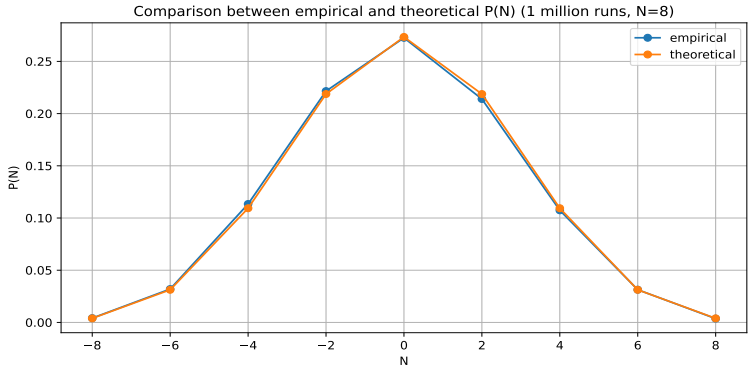


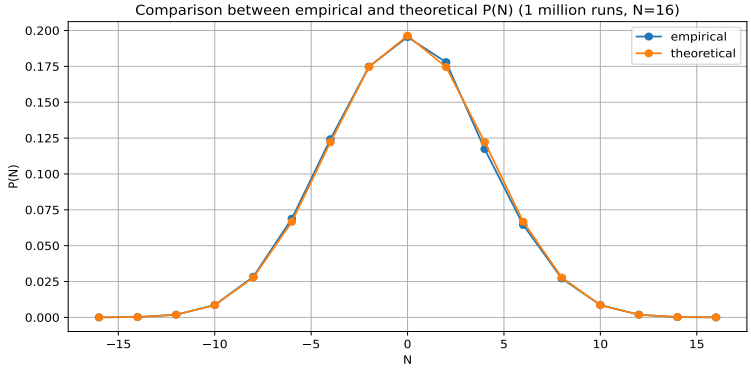
Figure 8: Loglog plot of $< (\Delta x_N)^2 >$ vs N

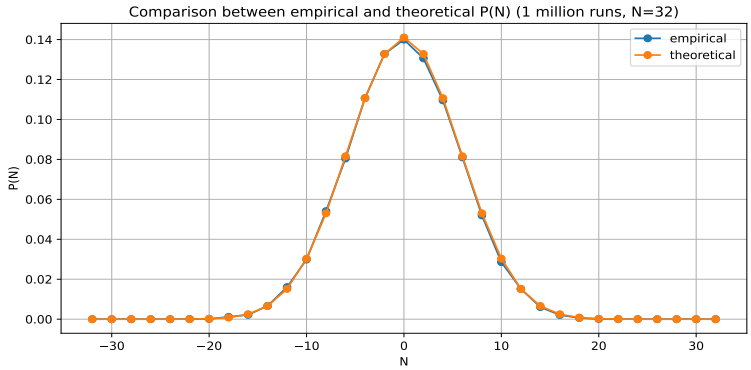
The plot shows the behaviour of $< (\Delta x_N)^2 >$ as N increases exponentially from 2^0 to 2^{13}. The fitted line has a slope of $2\nu^{calc} = 0.967$, which gives $\nu^{calc} = 0.484$, close to $\nu^{th} = 0.5$.

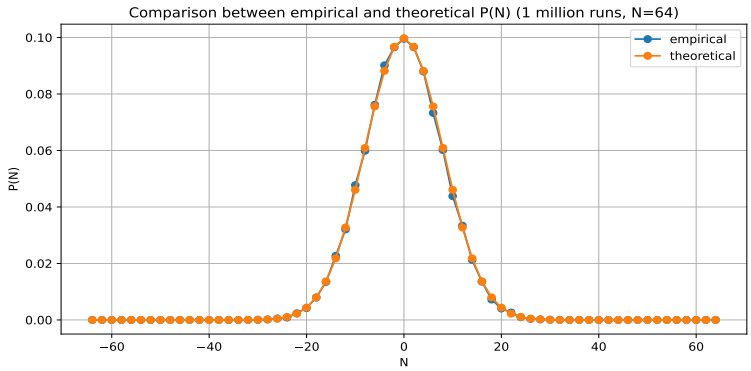
1.5 (f-g) Comparison between $P_N(x)$ and $P_N(x)^{th}$

The following are the plots of $P_N(x)$ and $P_N(x)^{th}$ vs N for $N = 8, 16, 32, 64$. In the first two cases, double precision allows to compute exactly the factorial; for $N > 20$ i decided to use the Stirling approximation.





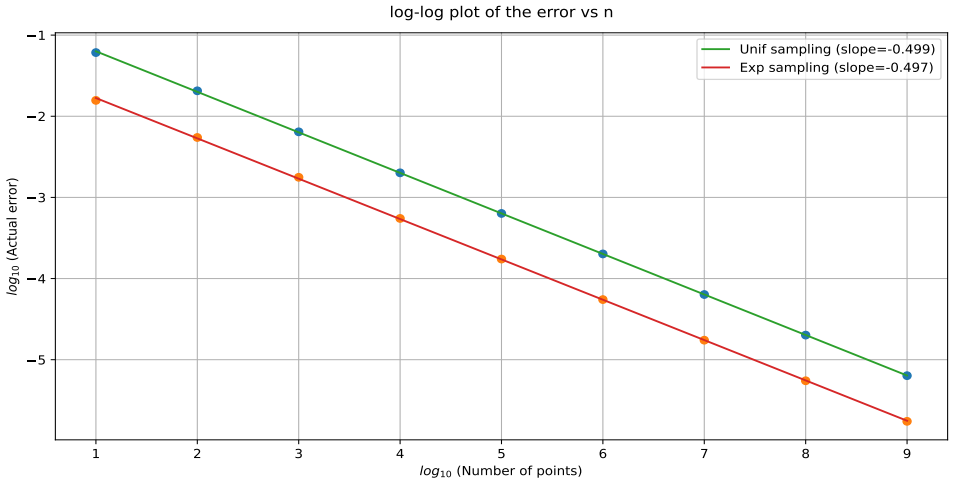




2 MC: sample mean and importance sampling

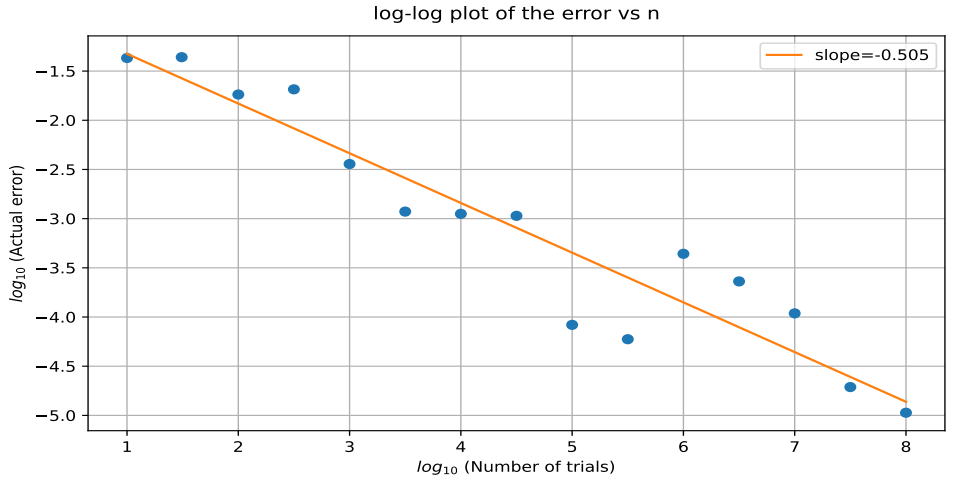
2.1 (a-b-c) Numerical estimate of $\int_0^1 e^{-x^2} dx$; errors

The following plot shows the dependence of the error between the exact and the approximate value of the integral. The expected behaviour is $\frac{\sigma_n}{\sqrt{n}} \propto \frac{1}{\sqrt{n}}$, which in a log-log plot corresponds to a slope of -0.5. I plotted two sets of points and the corresponding fitted line: one is relative to the uniform sampling and the other to the importance sampling (exponential in this case). I chose an exponentially equispaced number of sampled points, from 10^1 to 10^9. In both cases the fitted line has a slope that is close to the ideal value. The lines are therefore parallel with the one relative to the importance sampling being lower. This is expected since it's the more efficient of the two methods.



3 Variance reduction methods

3.1 (a-b) MC sample mean with uniformly distributed points



The plot shows the dependence of the absolute error to the number of points. The line is a linear fit of the points; its slope is 0.505, very close to the expected $\frac{1}{\sqrt{n}}$ behaviour.

We can compute σ_n to check if it's a good estimate of the error. It turns out that for $n = 10^4$, the actual absolute error is $1.12 * 10^{-3}$, while σ_n has the much bigger value of $2.21 * 10^{-1}$.

3.2 (c-d) Variance reduction techniques

To improve the error estimate, I applied the following two different methods of variance reduction:

- Average of the averages
- Block averages

In both cases I found that the estimated errors are consistent with the actual error. In the first, for $n = 10^4$, $\frac{\sigma_n}{\sqrt{n}} = 2.25 * 10^{-3}$, which is compatible with the estimated $\sigma_m = 2.19 * 10^{-3}$.

In the second case i checked that the error estimate wouldn't change with the block size. The results are in the following table:

block size	$\frac{\sigma_s}{\sqrt{s}}$
10000	$2.15 * 10^{-3}$
1000	$2.21 * 10^{-3}$
100	$2.20 * 10^{-3}$
10	$2.24 * 10^{-3}$

I also compared the efficiency of the two: I used the jupyter magic command %timeit, which runs a function several times measuring the execution time, and then computes the average and the standard deviation.

Fixing $n = 10^4$, the function implementing the average of the averages takes $7.24ms \pm 571\mu s$ per loop, while the function implementing the block averages takes just $779\mu s \pm 6.15\mu s$ per loop. The first method is about an order of magnitude less efficient, despite delivering the same result.

4 Appendix - Parallel computation with MPI

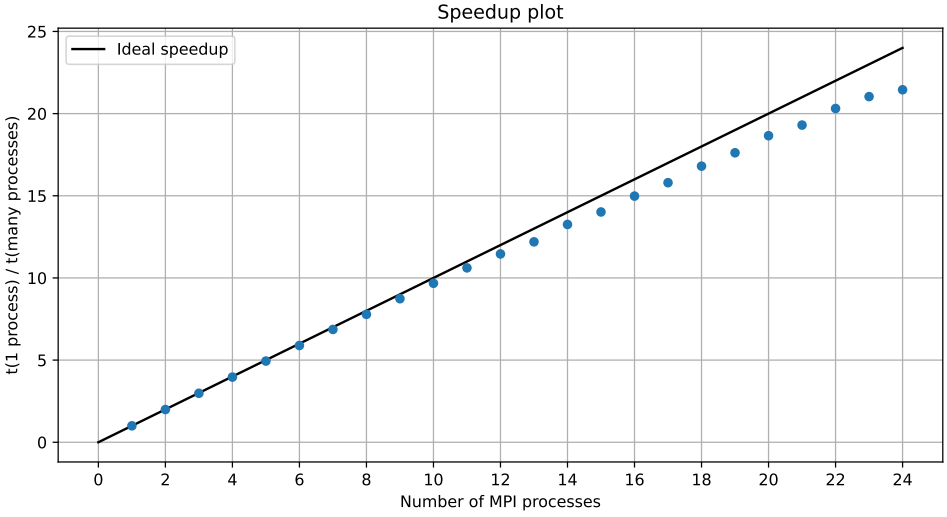
I decided to implement a parallel version of the program for the first exercise. My approach followed the distributed memory paradigm, which means that many parallel processes are created, each with its own memory. Computational tasks can only operate on local data, and if remote data are required, they must communicate with one or more remote processes. This is carried out by a message passing interface (MPI), which consists in calls to the routines contained in the MPI module.

In this case, the most demanding portion of the code are the two nested do loops over the number of steps and the number of runs. The number runs is divided among the processes and at the end the partial results are gathered and summed using the MPI_Reduce() function.

What follows is the plot of the measured speedup obtained using up to 24 processes, compared with the ideal case. The speedup is defined as

$$runtime(1_{process})/runtime(i_{processes})$$

where $i = 1, 24$ in this case.



The program achieves a good speedup, especially considering that the runtime includes also the time it takes to write data to file, which is done by just one process and it doesn't scale. This result is to be expected, since the program performs $O(n_{runs} * N)$ operations on a problem whose size grows as $O(N)$; assuming that both n_{runs} and N can go to infinity, this is a cpu-bound program with minimal communication required.

The test was performed on one of the THIN nodes of the Orfeo data center present in Area Science Park. The node is equipped with a pair of Intel Xeon CPUs, each with 12 cores. I must note that using MPI inside a single computer leads to a much lower communication time, since for all the processes, their memory space is in the RAM of the same computer.

The parameters chosen for the test are $N = 10^4$ and $n_{runs} = 10^5$. The runs were repeated 5 times for each number of processes, in order to obtain the mean and standard deviation. Since the times are very consistent, the error bars are smaller than the dot size in the plot.