UNIVERSITAT POLITÈCNICA DE CATALUNYA
RANDOMIZED ALGORITHMS

LAB. 1: **SIMULATION OF THE GALTON BOARD**

https://github.com/luigibruzzese/UPC-RA-MIRI-Lab.git

STUDENT: BRUZZESE LUIGI
DATE: 14/10/2024

Introduction

This report is related to the first assignment of the Randomized Algorithms course held at UPC in Barcelona.
The assignment is related to a simulation of the Galton board, a device used to simulate the behavior of probabilistic distributions and illustrate the central limit theorem.
The organization of the report is the following: section 1 explains how the problem has been translated into a python script, in order to obtain the expected results in mixed textual and graphical views; in section 2, the correlation between the specific problem and the probabilistic model it tries to explain is analyzed; then, section 3 generalizes the previous results by changing the "basic" settings of the parameters.
Together with this report, the python script whose results are shown here and a README file with the instructions for running it are delivered.

1. Coding the problem

As suggested in the instructions of the assignment, the Galton Board has been translated into a triangle matrix: ***n*** is the dimension of the matrix, which corresponds to the dimension of the board, i.e. the number of final cells in which the balls can fall at the end of the game, and ***N*** is the number of balls we use for simulation.
Notice that the script takes these two values as parameters when running it (see the README attached file): if they're not specified, it uses the trial values of *N = 500* and *n = 50*, that have been used for the first simulations done in section 2.
In its way to the final cell, a ball starts at position (0,0) of our matrix and, at each step, from position (*i, j)* can move either to the bottom (going to (i+1, j), as if, on the real board, it goes to the left) or to the right (going to (i, j+1)); the decision is taken randomly with, by default, the same probability of moving either to the left or to the right (i.e., $\boldsymbol{p = 0.5}$. Notice that this value can be changed for analyzing more general results that will be explained in section 4).
Due to this kind of movement, after *n* steps (which is the dimension of the matrix/board), the ball arrives at some cell (*i, n-i)*: we identify *i* as the number of the **final cell** (*i* goes from 0 to n-1, indeed we have *n* final cells in total).

After the simulation of this game, our data are ready to be analyzed: in our script, ***x*** is the vector with all the number that identify the final cells (so a simple vector with all the numbers from 0 to n-1), while *bag* is the vector that contains, for each cell, the number of balls fallen into it, with a 1:1 relationship with *x* (i.e., position *i* of the vector contains the number of balls in cell *i*).
If the values of *N* and *n* are not "so small", a cartesian graph that puts the values of the vector *x* on the x-axis and the corresponding values of the vector *bag* on the y-axis shows a trend like the plotting of a binomial distribution: the next section tries to explain this behavior.

2. The probabilistic model

From the original *bag* vector, we derive a normalized version of it by dividing each value for *N*, the number of balls. In that sense, we obtain a vector (called *normalizedBag* in the script), in which each value corresponds to the ratio between the number of balls in that cell and the total number of balls, i.e., an estimation of the **probability**, following our data, that, taken a ball, it falls in cell *i* with respect to all the other cells.

This probability is very important because it explains to us the reason for the similarity with a binomial distribution of our data.
Let's indicate with *i* the index of each cell, as already done during the computation, and with $P_{i,n}$ the probability that a ball, starting at (0,0), ends at cell *i* (i.e., at position (i, n-i), following our convention). If we calculate it analytically, we obtain that its value is

$$P_{i,n} = \binom{n}{i} p^i (1-p)^{n-i} \qquad (1)$$

with *p* the probability that regulates the movement of going to the right (in our case $p = \frac{1}{2}$, so that it can also be considered the movement of going to the left, it's equivalent). That happens because we can see each movement as **an independent Bernoulli trial** in which we have two possibilities (either right or left) and we decide with probability *p* of going to the right; following this idea, the whole process for a single ball can be considered as a Binomial probability (sum of *n* Bernoulli r.v.) in which we make *n* steps to reach the end. Indeed, if we want to reach the final cell *i*, we must perform exactly *i* movements to the right (that's why p^i) and *n-i* movements to the left to end there (in our case the process is symmetric since $p = \frac{1}{2}$).
Another way to justify this formula is to consider that we have $\binom{n}{i}$ ways to arrive at the cell *i*, and at each step we decide independently where to go following the probability p.

Now we can understand why, in a plot in which we take on the x-axis the **values of all the possible cells *i*** and on the y-axis the **value of the number of balls in the corresponding cell *i* divided by N**, we obtain a kind of binomial distribution: on the y-axis we're considering a **random variable Y** that, if we take a ball from the bag, estimates the cell in which the ball will end up at the end of the process. It consists, indeed, of a binomial random variable that counts the number of successes *i* of the ball, where a success, for each step, is when the ball goes to the right, with respect to the whole number of "trials" *n:*

$$Y \sim Bin(n,p) \qquad (2)$$

In that way, on the y-axis we have the probability that the ball has gone *i* times to the right, i.e. $P(Y = i)$, that corresponds to the number of balls (among all the N balls) in the cell *i* from our sample.

The following figure shows what we obtain with 500 balls in a 50-dimensional board, so *n = 50* and *N = 500* (notice that on the y-axis there are two scales: the one on the left shows the value calculated for the *normalizedBag*, while the one on the right shows the values of the original *bag,* i.e. the effective number of balls fallen into the corresponding cell):

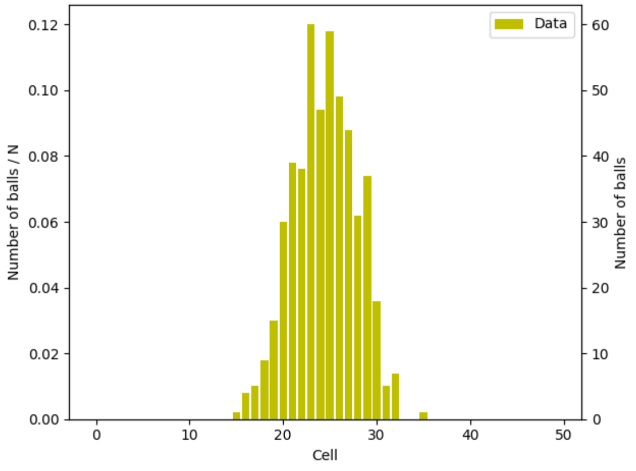


Figure 1

It corresponds to the first plot of the python script.
Now, we can plot the binomial approximation (2) using (1) with $p = \frac{1}{2}$ (that's our case), by printing the corresponding probabilities calculated in the points of the x-axis (i.e., the number of cells, the *x* vector):

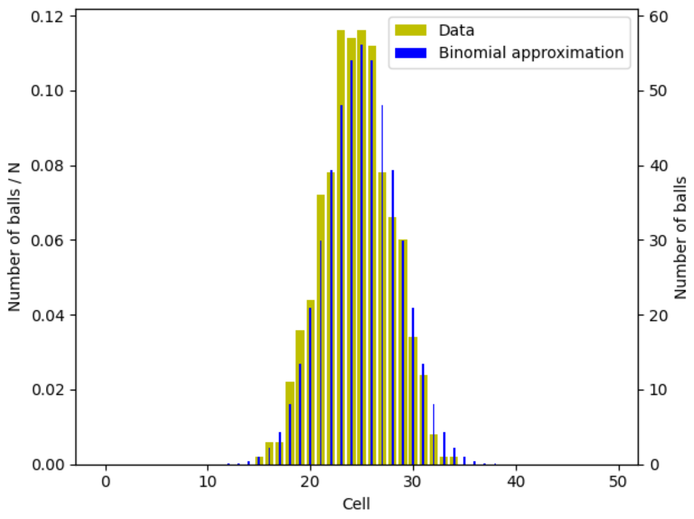


Figure 2

As we can see, even if we're using relatively "small" numbers of *N* and *n* (just to better show the plots), the behavior is already what we expected, even if with an error (the so-called *MSE_bin_points*, that is the MSE calculated between the real data and the binomial approximation):

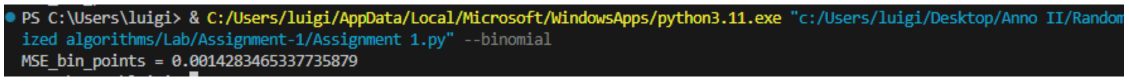

```
PS C:\Users\luigi> & C:/Users/luigi/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/luigi/Desktop/Anno II/Random
ized algorithms/Lab/Assignment-1/Assignment 1.py" --binomial
MSE_bin_points = 0.0014283465337735879
```

Figure 3

Now that we've understood completely the problem, we can go on and see how the binomial approximation that is visible here tends to another distribution, this time a continuous one, which is the Normal distribution. In fact, even with these small numbers of N and n, the plot already shows a behavior like the well-known bell of the Gaussian distribution.
My approach has been to let *scipy*, a python library that works with *matplotlib*, approximate the behavior of the binomial points (already calculated for the previous computation) to a Gaussian, by the usage of its function *curve_fit*. This function requires the shape of the function we want to approximate data with and the vectors of the x-axis and y-axis points to fit (respectively, in my case, *x* and *binPoints*): that's why I've defined a function called *gaussianFunc* in the code, that has this shape (the standard one of a Gaussian):

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

where μ and σ are the parameters the *curve_fit* function must guess, and represent, respectively, the median and the standard deviation of the Gaussian distribution.
The code also calculates the *MSE_bin_gauss*, which is the MSE between the binomial approximation and the gaussian one.

Let’s see which the results are using the same value of *N* and *n* of above:

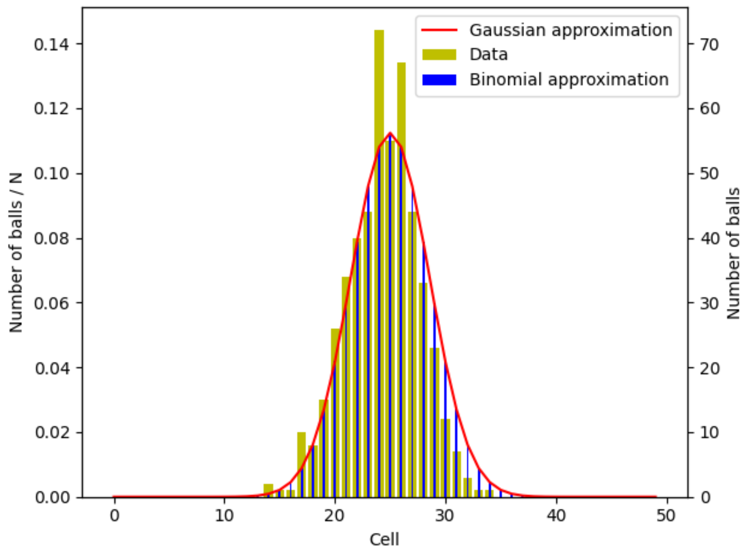
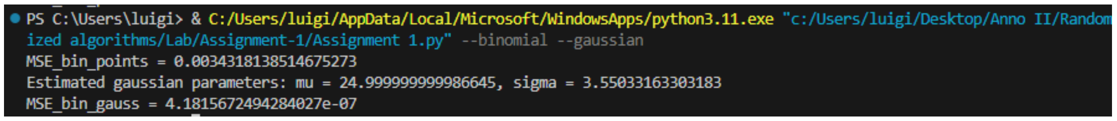


Figure 4

The Gaussian already seems to well approximate the binomial and, so, the distribution of the data, with the following parameters:



```
● PS C:\Users\luigi> & C:/Users/luigi/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/luigi/Desktop/Anno II/Random
  ized algorithms/Lab/Assignment-1/Assignment 1.py" --binomial --gaussian
  MSE_bin_points = 0.0034318138514675273
  Estimated gaussian parameters: mu = 24.999999999986645, sigma = 3.55033163303183
  MSE_bin_gauss = 4.1815672494284027e-07
```

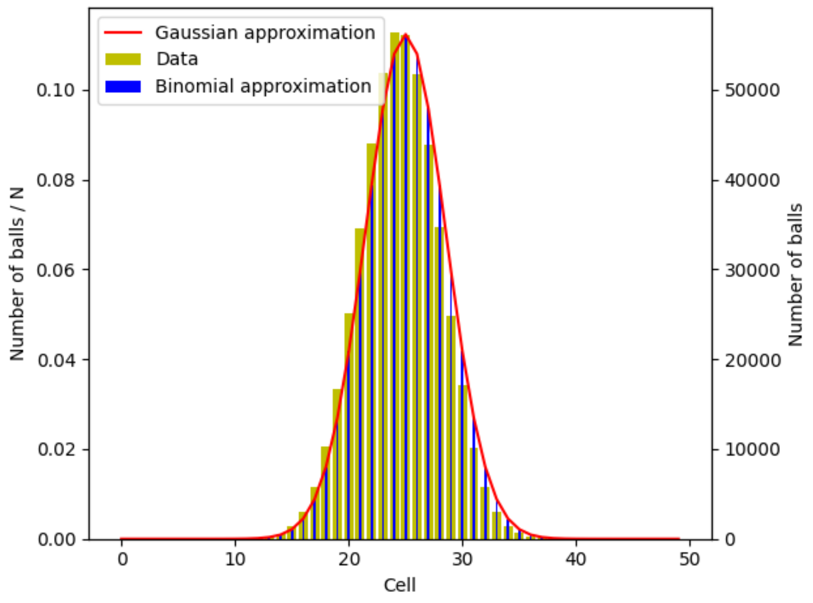
Figure 5

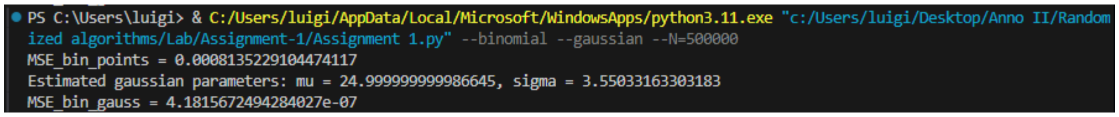
Indeed, the expected value μ is very close to 25, that is n/2, and the error between the binomial and the gaussian distributions is already quite small. The standard deviation $\sigma = 3.55$, so that the variance is $\sigma^2 = 3.55^2 = 12.6025$ that is not so far from $n/4 = 12.5$. That’s not a case and more general results about this approximation are reported in the next section.

3. Playing with the model

This last section has the aim of using the previous model to see more general probabilistic results.
In particular, the idea is to change the values of *n* and *N* to generalize the previous (basic) model and look at the different MSEs to see when the approximations get better results (both the binomial for the points and the gaussian for the binomial).
The following experiments can be done directly by setting the parameters from the command line while running the program (see the README file attached).

First, we can look at what happens by increasing only the value of the number of balls *N,* letting *n* remain the same:



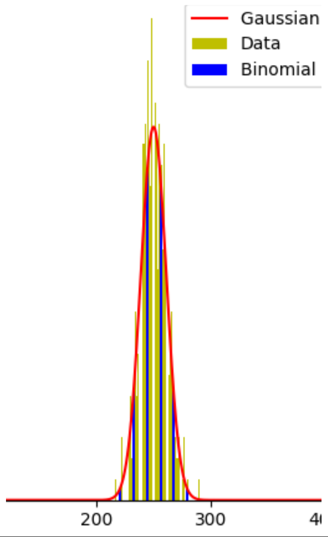


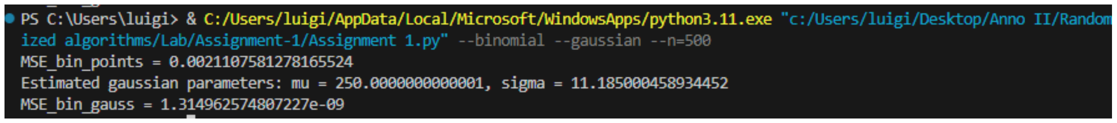
```
● PS C:\Users\luigi> & C:/Users/luigi/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/luigi/Desktop/Anno II/Random
  ized algorithms/Lab/Assignment-1/Assignment 1.py" --binomial --gaussian --N=500000
  MSE_bin_points = 0.0008135229104474117
  Estimated gaussian parameters: mu = 24.999999999986645, sigma = 3.55033163303183
  MSE_bin_gauss = 4.1815672494284027e-07
```

Figures 6, 7

The values related to the gaussian distribution are the same as *figure 5*, since we're not changing the values on the x-axis, but **the binomial approximation now describes very well our data**: the MSE is 10 times reduced.

On the other hand, let's see what happens by increasing the value of the board (*n*), letting *N* remain the same:





```
● PS C:\Users\luigi> & C:/Users/luigi/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/luigi/Desktop/Anno II/Random
  ized algorithms/Lab/Assignment-1/Assignment 1.py" --binomial --gaussian --n=500
  MSE_bin_points = 0.0021107581278165524
  Estimated gaussian parameters: mu = 250.0000000000001, sigma = 11.185000458934452
  MSE_bin_gauss = 1.314962574807227e-09
```

Figures 8, 9

The approximation between the points and the binomial is not very accurate (the MSE remains approximately the same with respect to the case of Figure 5), but the gaussian approximates better the binomial (the MSE is reduced by a factor of 10^2!).
The values of μ and σ^2 are closer than before to ${}^{n}/_{2} = 250$ and ${}^{n}/_{4} = 125 = 11.18033989^2$:
it seems that, more than before,

$$Y \sim Bin\left(n, {}^{1}/_{2}\right) \rightarrow N\left({}^{n}/_{2}, {}^{n}/_{4}\right) \quad as\ n \rightarrow \infty \qquad (3)$$

This result is explained by the Central limit theorem, which tells us that the sum of *n* random variables (in our case, *n* Bernoulli random variables) approaches a normal distribution as *n* increases.

In the end, for "big" values of *n* and *N*, our plot is definitively uniform and the gaussian approximation is very good also with respect to the data:

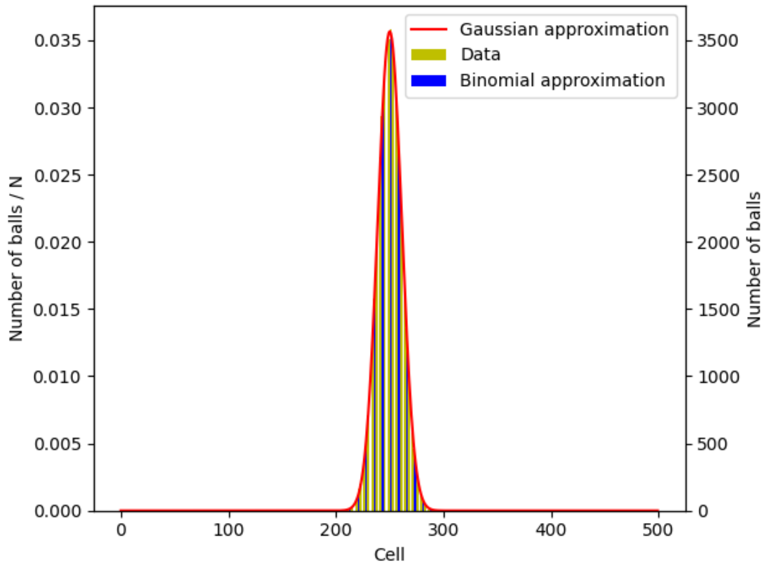
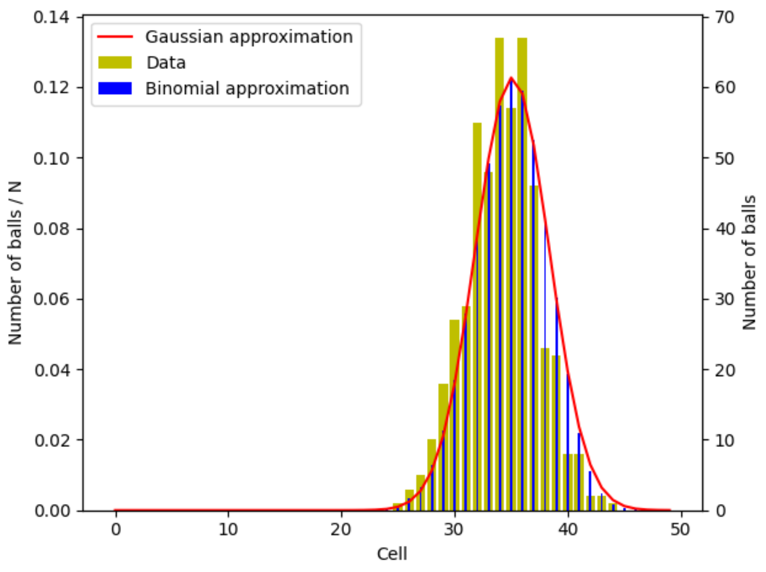


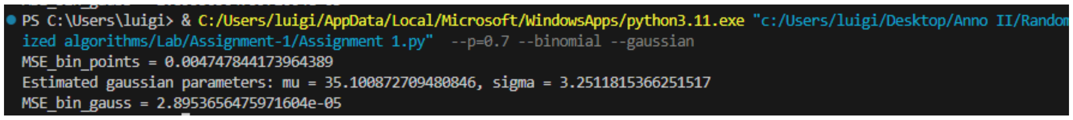
Figure 10

4. Conclusions

We can change a little bit the way in which the Galton board works by changing the value of *p*, i.e., the probability, at each step, that a ball falls to the right.
This change can be done by setting the parameter from the command line (see the README file, default is 0.5, used to show the previous results).

The following figures show what happens for the initial values of *n = 50, N = 500* and setting *p = 0.7*:





```
● PS C:\Users\luigi> & C:/Users/luigi/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/luigi/Desktop/Anno II/Random
  ized algorithms/Lab/Assignment-1/Assignment 1.py"  --p=0.7 --binomial --gaussian
  MSE_bin_points = 0.004747844173964389
  Estimated gaussian parameters: mu = 35.100872709480846, sigma = 3.2511815366251517
  MSE_bin_gauss = 2.8953656475971604e-05
```

Figures 11, 12

As expected, the data are “moved” to the right, since the probability of going to the right is more than before. The gaussian approximation has a similar standard deviation as before, but the mean is moved to the right too, as expected.

In the end, from this analysis the power of the probabilistic models and the Central Limit Theorem emerges even if with a “small” and easy simulator like in our case. It’s important to notice that in a real Galton board (i.e., a board created by hand), other factors must be considered (such as mechanical laws, gravity…) that can influence the experiment and lead to results that are not as accurate as the ones provided by a simulator. However, the general results achieved by going to asymptotical values of the parameters, describe very well the mechanism even in a real simulation, since these factors can be considered irrelevant with respect to the general “law” that regulates the process.