**Researches about theory (R)**

8\_R.

Do a research about the following topics:  
  
- The law of large numbers LLN, the various definitions of convergence  
  
- The convergence of the Binomial to the normal and Poisson distributions  
  
- The central limit theorem [in anticipation of a topic we will study later]

“In probability theory, the law of large numbers (LLN) is a theorem that describes the result of performing the same experiment a large number of times. According to the law, the average of the results obtained from a large number of trials should be close to the expected value and will tend to become closer to the expected value as more trials are performed.” (Wikipedia)

Chart, scatter chart

Description automatically generated

This is an example with the experiment of dice rolls. We can see on the X axis the number of trials (in this case the n value is 1000), instead on the Y axis we have the possible values (6 cases: 1,2,3,4,5,6). The mean of this values is 3.5, so we can plot the precise expected value in the chart (the blue line). Starting the experiment and so moving from the left side of the chart towards the right side we can see that with a low number of trials we can have values not so close to the expected one, but instead as we move towards the right, we can observe the mean is stabilized closely to the expected value. We will see that this experiment can be replicated several times (infinitely) and we have the same shape of the chart.

There are two different versions of the law of large numbers that are described below. They are called the strong law of large numbers and the weak law of large numbers. Stated for the case where X1, X2, ... is an infinite sequence of independent and identically distributed (i.i.d.) Lebesgue integrable random variables with expected value E(X1) = E(X2) = ...= µ, both versions of the law state that – with virtual certainty – the sample average:

{\displaystyle {\overline {X}}\_{n}={\frac {1}{n}}(X\_{1}+\cdots +X\_{n})}

converges to the expected value:

|  |  |
| --- | --- |
| {\displaystyle {\overline {X}}\_{n}\to \mu \quad {\textrm {as}}\ n\to \infty .} |  |

The “weak” law of large numbers (also called Khinchin law) states that the sample average converges in probability towards the expected value

Shape

Description automatically generated with medium confidence

That is, for any positive number *ε*,



Interpreting this result, the weak law states that for any nonzero margin specified (*ε*), no matter how small, with a sufficiently large sample there will be a very high probability that the average of the observations will be close to the expected value; that is, within the margin.

The “strong” law of large numbers (also called Kolmogorov’s law) states that the sample average converges almost surely to the expected value.

Shape

Description automatically generated with medium confidence

That is,



What this means is that the probability that, as the number of trials *n* goes to infinity, the average of the observations converges to the expected value, is equal to one.

In probability theory, there exist several different notions of convergence of random variables. The convergence of sequences of random variables to some limit random variable is an important concept in probability theory, and its applications to statistics and stochastic processes. A sequence X1, X2… of real-valued random variables is said to converge in distribution or converge in law to a random variable X if:



For every number x in R at which F is continuous.

For sufficiently large values of n the binomial law is approximated by other laws.

When n tends to infinity, leaving lambda = np fixed, the binomial distribution tends to the Poisson distribution P (lambda) = P (np). In statistics this approximation is usually accepted when n> = 20 and p <= 1/20, or when n> = 100 and np <= 10.

By the central limit theorem, when n tends to infinity, leaving p fixed, the binomial distribution tends to the normal distribution N (np, npq), with mean np and variance npq. In statistics this approximation is usually accepted when np> 5 and nq> 5.

More precisely, the central limit theorem states that



In probability theory, the central limit theorem (CLT) establishes that, in many situations, when independent random variables are summed up, their properly normalized sum tends toward a normal distribution (informally a bell curve) even if the original variables themselves are not normally distributed. The theorem is a key concept in probability theory because it implies that probabilistic and statistical methods that work for normal distributions can be applicable to many problems involving other types of distributions.

**Applications / Practice (A)     [work on this at least 30' a day, all days]**

8\_A. Exercise (also partially described in video 04)  
  
Generate and represent m "sample paths" of n point each (m, n are program parameters), where each point represents a pair of:

time index t, and **relative frequency** of success f(t),  
  
where f(t) is the sum of t Bernoulli random variables with distribution B(x, p) = p^x(1-p)^(1-x) observed at the various times up to t: j=1, ..., t..

At time n (last time) and one other chosen inner time 1<j<n (where j is a user parameter) represent with a histogram the distribution of f(t).

See also what happens if you replace the relative frequency f(t) with the **absolute frequency** n(t) or by **normalized relative frequency**: f(t) / sqrt(p(1-p)/n).  
  
Comment briefly on the result.

(The general scheme of this exercise, will also be "reused" in next homeworks where we will consider other more interesting stochastic processes.)

VIDEO + CODE

It’s clear that if we set N to be low (in my application you can insert at least 20 as input for N and no maximum value limit), for example with n = 20 or n = 30 we have lower probability that the results (Bernoulli trials) will fall around the expected probability P (the one we set at the starting point before running the chart plotting). Instead, if try with high values of N (like n = 500 or even n = 2000 or 3000) we have way higher probability of having the results really close to the expected final value. Moreover, we can see (still with an high number of trials) that the LLN is reflected into the histograms that we produce at the middle and at the end of the chart. If we look at the expected value at the end of the chart, the bar in the histogram is probably the highest, because in that point most of the paths fallin that portion. If we move from the expected value, we will see lower bars (similar to the Gauss chart).

**Researches about applications (RA)**

6\_RA. Do a web research about the various methods proposed to compute the running median (one pass, online algorithms).  
Store (cite all sources and attributions) the algorithm(s) that you think is(are) a good candidate, explaining briefly how it works and possibly try a quick demo.

from heapq import heappush,heappop, heapify,\_heapify\_max

arr = [11, 8, 9, 6, 20, 5, 7, 14, 12, 3]

# `low` is a max-heap & `high` is a min-heap

low = []

high = []

# Init median to zero

median = 0

for i in arr:

# array value is less than median, add to median

# add the value to max-heap & heapify

# else add to min-heap & heapify

if i < median:

# as we don't have inbuilt function called \_heappush\_max

# so use heappush [O(log N) per push] and call \_heapify\_max [O(N)]

heappush(low, i)

\_heapify\_max(low)

else:

heappush(high, i)

# The size of height difference between heaps

# is more than 1, then move the element & heapify both

if len(low) > len(high)+1:

heappush(high, heappop(low))

\_heapify\_max(low)

elif len(high) > len(low) + 1:

heappush(low, heappop(high))

\_heapify\_max(low)

# Calculate Median:

# If lengths equal then median is average of both heaps

# Else median is larger heap element

if len(low) == len(high):

median = float(low[0] + high[0])/2.0

else:

median = float(low[0]) if len(low) > len(high) else float(high[0])

print("Median: " , median)

# Time Complexity = O(log N) + O(1) ≈ O(log N).

# Space complexity = O(N) linear space to hold input in heap containers.

PS C:\Users\edoardottt\Desktop> python3.exe .\test\_median.py

Median: 11.0

Median: 9.5

Median: 9.0

Median: 8.5

Median: 9.0

Median: 8.5

Median: 8.0

Median: 8.5

Median: 9.0

Median: 8.5

The algorithm creates two heaps: one called low to store the first sorted half of the inputs and the other called high to store the other sorted half. The low heap is a “max heap”, this means that the root is the highest value, and the children have lower values, the leaves have the lowest values. On the contrary the high heap is a “min heap”, this means that the root is the lowest value, and the children have higher values, the leaves have the highest values. We initialize the median to zero and we start receiving inputs. If the value is lower than the median, we push the value in the low heap and the restore the rules of a max heap, otherwise we do the same action but inserting the value in the high heap. If the number of items inside the low heap is greater than the number of the high heap plus one, we remove the highest value in the low heap and we put it into the high heap (always restoring the rules of a heap), otherwise we do the opposite: removing the lowest value from the high heap and putting it into the low heap (and restoring the rules of a heap). At the end we compute with the obvious formula the running median (taking the mean of the highest value of low heap and the lowest value of high heap if the two heaps have the same length; one of the two if the size are not equal, we could say the ‘exceeding’ one).

It’s interesting also the one posted here <https://www.geeksforgeeks.org/median-of-stream-of-integers-running-integers/>

**References**

<https://en.wikipedia.org/wiki/Law_of_large_numbers>

<https://stats.stackexchange.com/questions/489948/difference-between-uniform-laws-of-large-numbers-and-law-of-large-numbers?rq=1>

<https://en.wikipedia.org/wiki/Convergence_of_random_variables>

<https://stats.stackexchange.com/questions/22557/central-limit-theorem-versus-law-of-large-numbers>

<https://medium.com/mind-boggling-algorithms/streaming-algorithms-running-median-of-an-array-using-two-heaps-cd1b61b3c034>

<https://gist.githubusercontent.com/anilpai/48d058d6a1e982d57ab55700c2e440c0/raw/7a79ae3f2952d84552d5c9646f09e351dfaaa8db/running_median.py>

<https://www.geeksforgeeks.org/median-of-stream-of-integers-running-integers/>

<https://en.wikipedia.org/wiki/Central_limit_theorem>