Danmarks Tekniske Universitet

DTU

Stochastic Simulation - Exercises

Course 02443

Authors

Luigi Pampanin - s243128
Samuele Dilengite - s243599

June 16, 2025

Contents

The Python code used in this assignment is available in the following GitHub repository.

1 Exercise 1: generating random numbers

1.1 Implementing LCG and evaluate the performance

The linear congruential generator is defined by the recurrence relation:

$$x_i = (ax_{i-1} + c) \bmod M$$

A Python implementation using integer arithmetic is provided below.

Listing 1: LCG implemented using integer operations

```
def LCG(seed, a, c, m, n):
    X = [seed]
    for _ in range(n - 1):
        X.append((a * X[-1] + c) % m)
    return X
```

Generation of 10.000 pseudo-random numbers 10,000 pseudo-random numbers were generated and grouped into 10 bins. The figure below displays the frequency distribution.

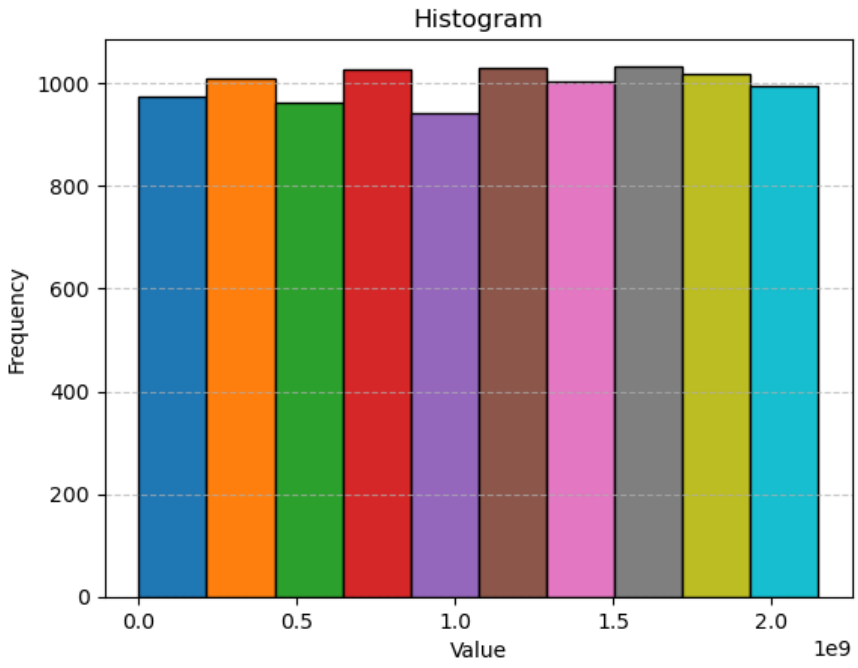


Figure 1: Histogram of LCG output

Evaluation of quality of the generator Different test were applied to test for uniformity and independence, two of the main feature to look for in a random number generator. For uniformity the **chi-squared** test was applied, which evaluates deviation from expected bin frequencies. The test did not indicate significant deviation, with a p-value comfortably above the 0.05 significance level (p-value = 0.45). This suggests that the numbers are distributed uniformly enough for practical purposes.
The **Kolmogorov-Smirnov** was applied as well. The KS test compared the empirical cumulative distribution function (CDF) of the normalized LCG values to the theoretical CDF of the Uniform(0,1) distribution. The maximum absolute difference between the empirical and theoretical CDFs was small, and the corresponding p-value did not provide evidence against the null hypothesis. This indicates that the sample does not significantly deviate from a uniform distribution in shape.
To evaluate the independence of the generated pseudo-random sequence, a **lag-h correlation** test was applied. The normalized LCG output was used to compute the lag-1 correlation coefficient c_1, and the result was compared to the theoretical distribution:

$$c_h \sim \mathcal{N}\left(0.25,\ \frac{7}{144n}\right)$$

The observed value of c_1 was then transformed into a z-score:

$$z = \frac{c_1 - 0.25}{\sqrt{7/(144n)}}$$

The computed z-score was close to 0, indicating that the correlation between successive values in the sequence is consistent with the null hypothesis of independence. No statistically significant autocorrelation was detected at lag 1, 2 and 4.

Experimenting with different values for the parameters Multiple parameter sets were tested. Results indicate:

- Poor performance with $a = 5$, $c = 1$, $m = 16$ due to short cycles and strong structure (Figure 2).
- Good performance with $a = 1664525$, $c = 1013904223$, $m = 2^{32}$; these values were chosen looking up on internet for numbers that respected the **Maximum cycle lenght** theorem. This generator passed uniformity and independence tests (Figure 1).

1.2 Testing a system available generator

To benchmark the custom linear congruential generator (LCG), the same statistical tests were applied to a random sample generated by the built-in NumPy function `np.random.rand`. The sample size was identical ($n = 10{,}000$), and the tests included a histogram analysis (Figure 3), chi-squared test, Kolmogorov-Smirnov (KS) test, and lag correlation tests at lags 1, 2, and 4.

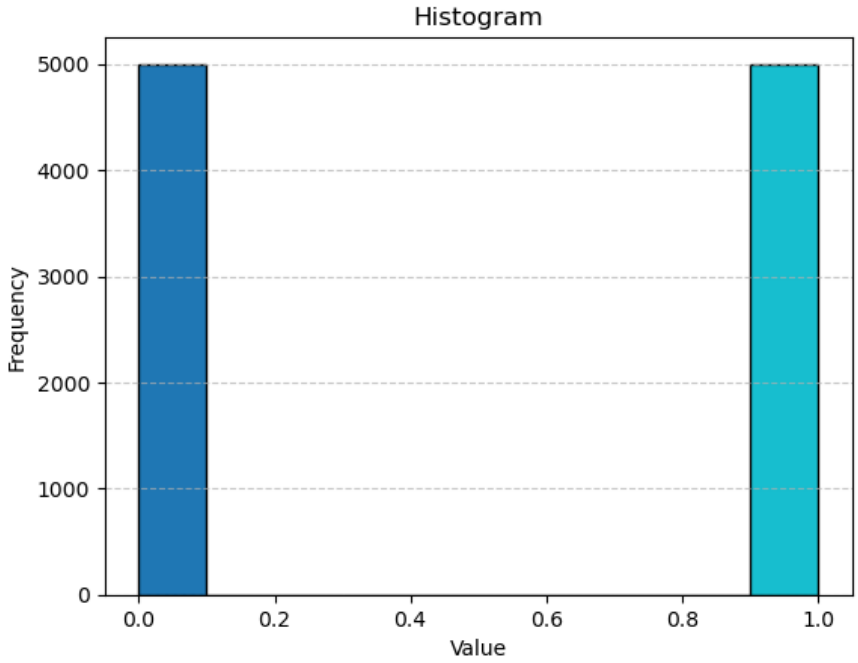


Figure 2: Histogram of bad LCG output

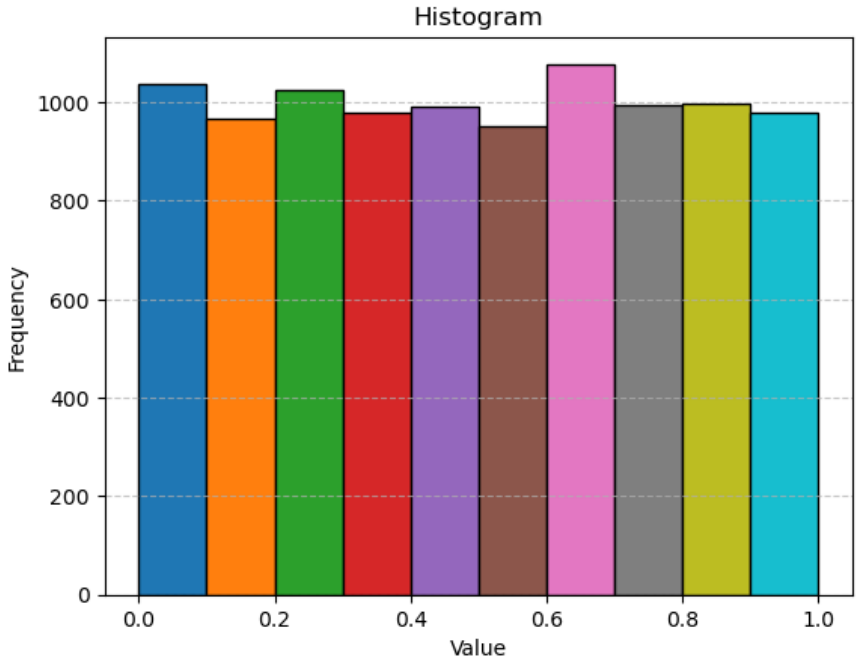


Figure 3: Histogram of np.random.rand output

The custom chi-squared test returned a p-value above the 0.05 threshold (p-value = 0.92), indicating no significant deviation from the expected uniform distribution. This confirms that the sample passes the test for uniformity.

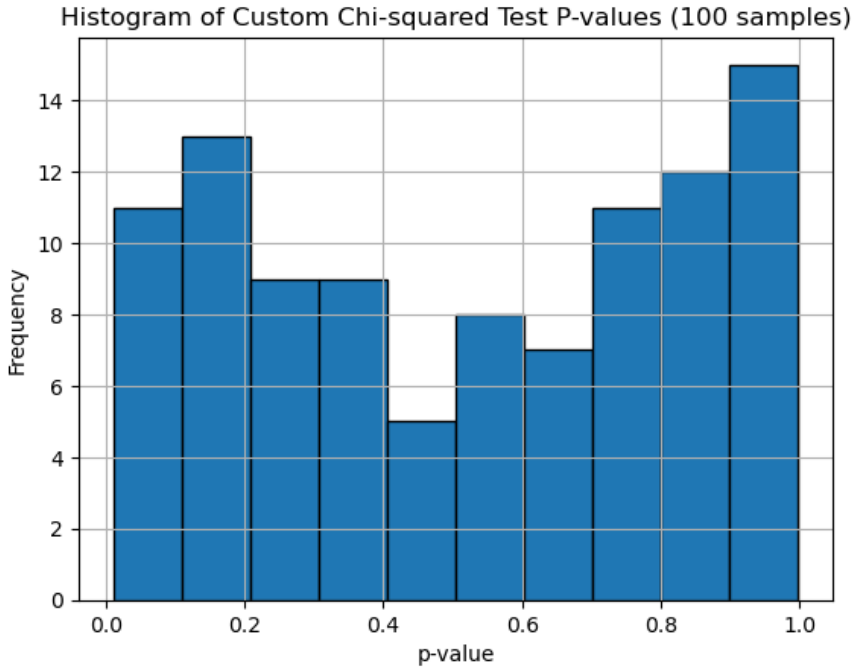


Figure 4: Histogram of p-values

The KS test showed excellent agreement between the empirical cumulative distribution function and the theoretical uniform CDF. The resulting p-value further supported the assumption of uniformity.

The lag correlation coefficients c_1, c_2, and c_4 were all close to the expected theoretical mean of 0.25. Corresponding z-scores were within acceptable ranges, suggesting no statistically significant dependence between values at those lags.

The system generator exhibited high-quality random behavior, passing all uniformity and independence tests. Compared to the LCG implementation, the system RNG provided slightly more consistent statistical properties across all tested metrics, as expected from a well-established, optimized pseudo-random number generator.

1.3 Reflection on Testing a Single Sample

Performing statistical tests on a single sample of pseudo-random numbers can give an indication of the generator's quality, but it is not fully sufficient. Due to the stochastic nature of the tests, even a well-performing generator may occasionally produce a sample that fails a statistical test purely by chance. At the same time, a poor generator might pass by luck.

To obtain a more reliable assessment, the tests were repeated on multiple independently generated samples. The resulting p-values were collected and visualized in a histogram.

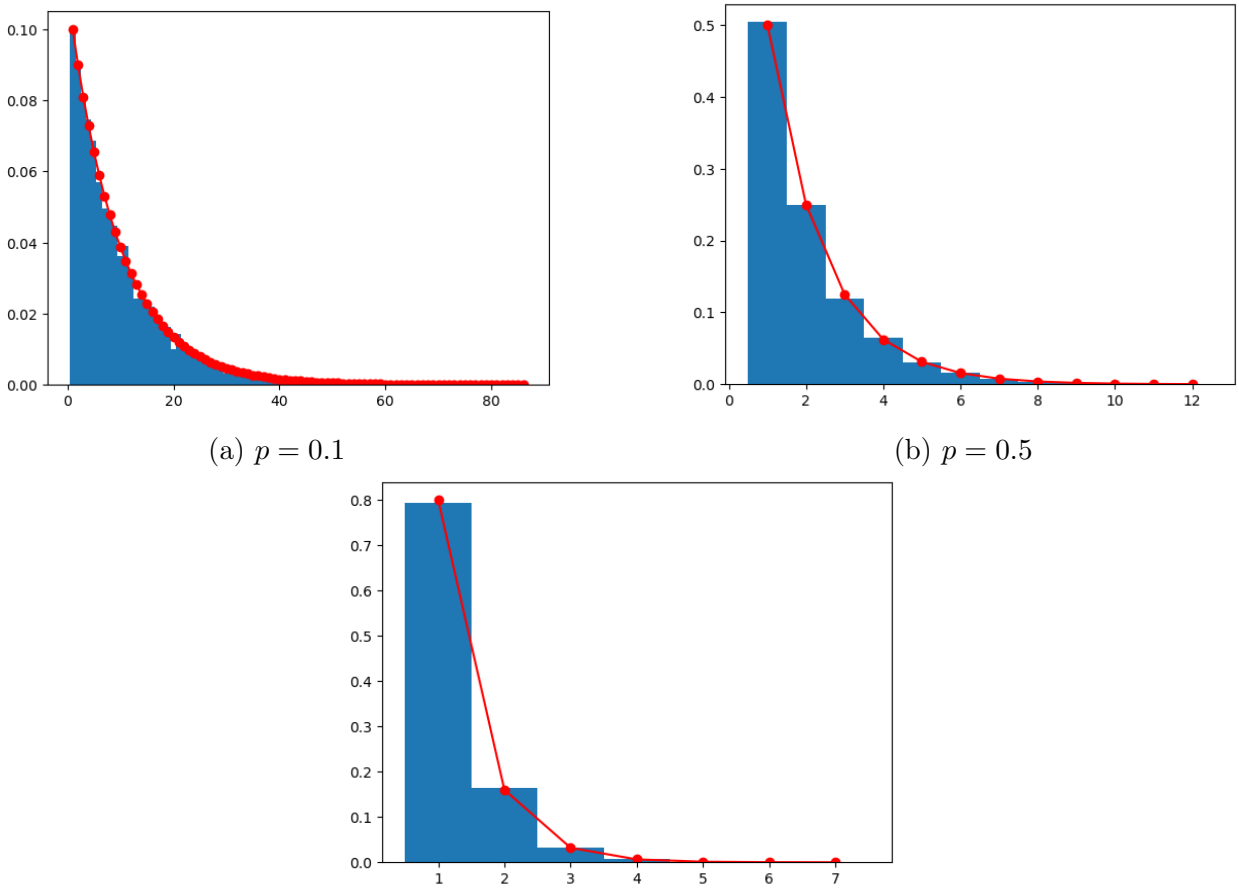
The p-values obtained from repeated tests appeared to follow a uniform distribution (Figure 4), as expected under the null hypothesis for a good RNG. This consistency across samples strengthens the conclusion that the LCG, with the final chosen parameters, performs well in terms of both uniformity and independence.

In conlusion, relying on a single test result is not sufficient for RNG validation. Repeating tests across multiple samples provides a more robust evaluation and helps detect subtle flaws that may not appear in a single run.

2 Exercise 2: generating discrete random variables

1) Geometric distribution

To simulate the geometric distribution, we use the crude method, as it provides an explicit formula to choose the correct categories for the values of the variable. In particular, we can use the formula $X = \lfloor (\frac{\log(U)}{\log(1-p)}) \rfloor + 1$, where U represents values coming from a uniform distribution over the interval $(0, 1)$. This means that we generate samples from a uniform distribution, and then the values generated by the geometric distribution are found applying the formula.



(a) $p = 0.1$

(b) $p = 0.5$

(c) $p = 0.8$

Figure 5: Simulated (Histogram) and Theoretical (Red line) geometric distribution for different values of p

As can be seen from Figure 5, the simulated distributions follow the expected behaviour, and we can notice that, for $p = 0.1$, there are more classes and way less samples fall in the first classes, this is because the geometric distribution describes the probability of the first success happening at a specific value, and having such a low probability of success leads to

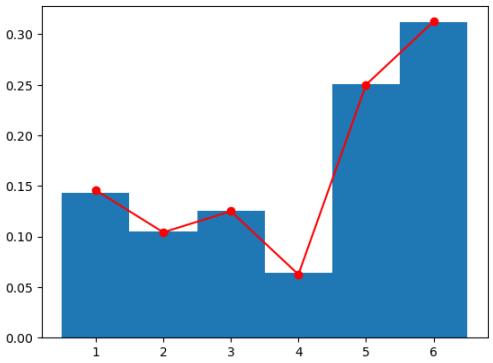
distribute more the classes, whereas for $p = 0.5$, half of the samples fall in the first class, and for $p = 0.8$, the majority of samples fall in the first class, given the high probaibility of success.

2) Simulating 6-point distribution

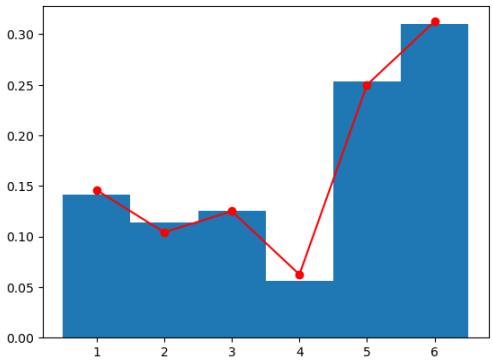
To simulate the given discrete random variable using crude method, we simply generate samples from a uniform distribution, then assign values to the classes ($\{X = x_i\}$ depending on the formula $\sum_{j=1}^{i-1} p_j < U \leq \sum_{j=1}^{i} p_j$, where p_j is $\mathbb{P}(X = x_i)$, i.e. we look at where the uniform samples fall in the cumulative density function of X, and assign the samples accordingly.

When using the rejection method, we select a constant c, such that $c \geq p_i \, \forall \, i$, then we generate one the classes with uniform probability, and accept the sample in that class, I, if $U_2 \leq p_i/c$, where U_2 is a sample obtained from a uniform distribution, otherwise we reject it and we go back at the first step.

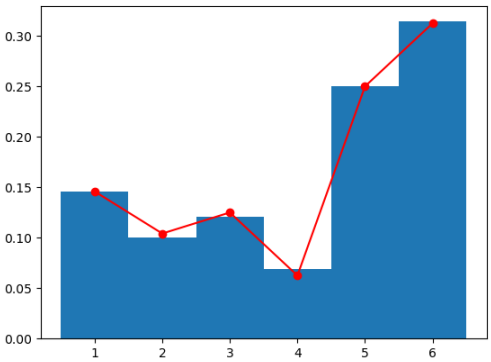
For the alias method, we first need to generate two tables: the $F(I)$-values (part of the mass belonging to I), and the $L(I)$-values (the alias of class I). Then, we generate a class I with equal probability and test if $U_2 \leq F(I)$, if the result is positive, we assign $X = I$, otherwise $X = L(I)$.



(a) Crude method



(b) Rejection method



(c) Alias method

Figure 6: Simulated 6-point random variable with different methods

Figure 6 Illustrates the results from the three different methods, showing that the results resemble quite well the expected distribution.

3) Comparing the methods

To compare the three different methods, we can rely on different tools, such as visual inspection and statistical tests.
Looking at the histograms, the three methods seem to provide accurate approximations of the random variable, thoroughly following the theoretical distribution.

Using a more quantitative approach, we run the χ^2-test for goodness of fit. In order for the results to be accurate, we need to make sure that the expected values for every class are at least 5, and since the sample is quite large for this probability, this condition is verified.

The results of the tests show that there is no statistical evidence to reject the hypothesis that the generated distributions differ from the theoretical one.

4) Choosing the best suited method

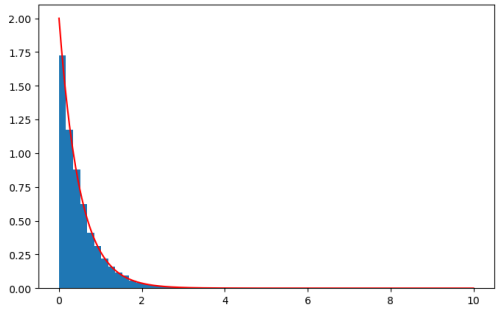
The methods have some differences which can lead to choose one instead of another one in different situations. For example, the Crude method provides an accurate way to sample from a discrete variable. However, it requires to go through every class looking for the one

where the sample belongs to, which, in the best way can be carried out with a binary search, having complexity of $O(\log(n))$, making it computationally expensive for high number of samples. The rejection method works efficiently when the classes are balanced; indeed, if there is a class which has a much higher probability, this would affect the rejection ratio, leading to reject a lot of samples in the classes with lower probability. Finally, the Alias method runs in constant time once the table is set up: $O(1)$. However, the construction of the table requires preprocessing before running the algorithm.

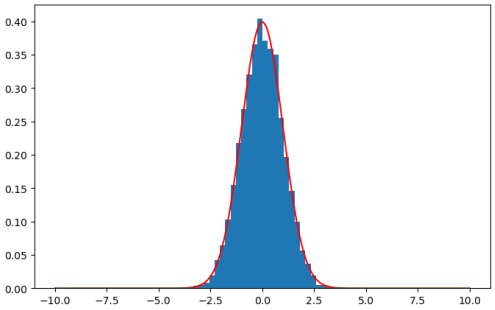
3 Exercise 3: generating continuous random variables

1) sample continuos variables

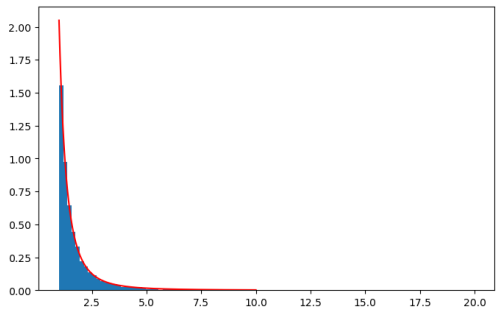
We generate the Exponential and Pareto distributions using the inverse method, i.e. computing the inverse of the cumulative density function and applying it to samples from a uniform distribution, while we generate the Normal distribution using the Box-Meuller method, using polar coordinates transformation. The results are illustrated in Figure 7. We point out that, since the Pareto distribution has heavy tail, when sampling, we draw some samples of high value, which would make the plot not visible, therefore, for plotting the results we decided to remove these values.



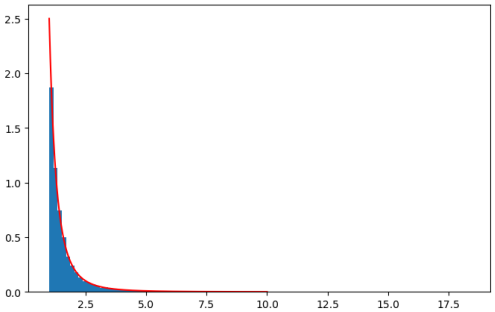
(a) Exponential distribution



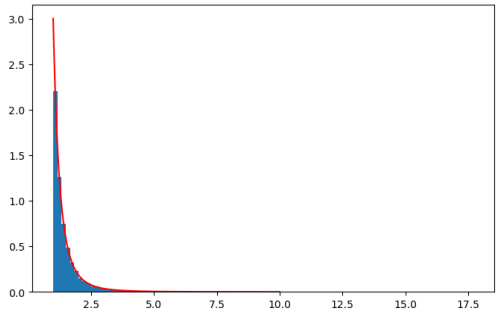
(b) Normal distribution



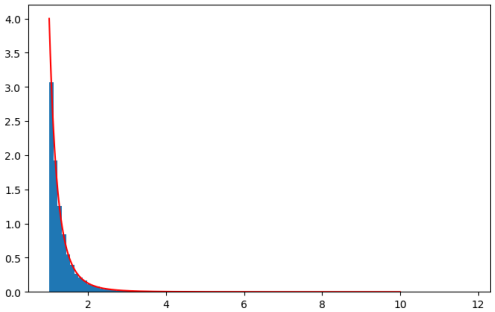
(c) Pareto k = 2.05



(d) Pareto k = 2.5



(e) Pareto k = 3



(f) Pareto K = 4

Figure 7: Continuos distributions simulated

Looking at the histograms in Figure 7 and using statistical tests like the Kolmogorov-Smirnov test, we can conclude that there is no statistical evidence to conclude that the generated distributions are different from the theoretical.

2) Pareto distribution

We compare the expected and sample first and second moments for the Pareto distribution for different values of the parameter k.

k	theoretical mean	sample mean	theoretical variance	sample variance
2.05	1.95	1.99	37.18	27.54
2.5	1.66	1.68	2.22	3.97
3	1.5	1.5	0.75	1.08
4	1.33	1.33	0.22	0.26

Table 1: Values of mean and variance for different values of k for Pareto distribution

From Table 1 we see that, for every value of k, the theoretical and sample mean are very similar, whereas for small values of k, in particular close to 2, the variances are way different. This can be due to the fact that, the theoretical variance of the Pareto distribution diverges for $k \to 2^+$, therefore, when $k = 2.05$, the theoretical variance will be higher than the sample variance. Furthermore, since the Pareto distribution has heavy tails, it can happen that we sample from more extreme values, leading to higher sample variances for the other values of k.

3) Normal distribution

We generate samples from a Normal distribution using the Box-Meuller method from Point 2, and compute 100 95% confidence intervals using 10 as sample size. Since we know that the samples should represent a normal distribution with average 0 and variance 1, we compute how many of the confidence intervals generated actually comprise these values.
To compute the CI for the mean, we use a t-test, supposing we don't know the real variance, and for the variance we use a χ^2- test. The Intervals have the following form, respectively:

$$\left[\mu - \frac{t_{1-\alpha/2,(n-1)}}{\sqrt{n}}, \mu + \frac{t_{1-\alpha/2,(n-1)}}{\sqrt{n}}\right]$$

$$\left[\frac{(n-1)s^2}{\chi^2_{1-\alpha/2,(n-1)}}, \frac{(n-1)s^2}{\chi^2_{\alpha/2,(n-1)}}\right]$$

The CI for the mean is the one shown in the slides, whereas the one for the variance is a known result from the literature.

Running the code multiple times, we get that the around 93-97 of the CI contain the real values for the mean and the variance, showing pretty accurate results even though the sample size of 10 is not big.

4) Pareto distribution using composition

To generate the Pareto distribution using composition, we follow the apporach of the slides, where we first generate samples from a random variable $Y \sim \mathcal{E}(\mu)$, and then we

generated the x values from $X \sim \mathcal{E}(Y)$, giving us samples form a Pareto distribution with parameters $k = 1, \beta = \mu$. Again, for plotting reasons, we remove values higher than 100 to make the plot visible. The plot is shown in Figure 8.

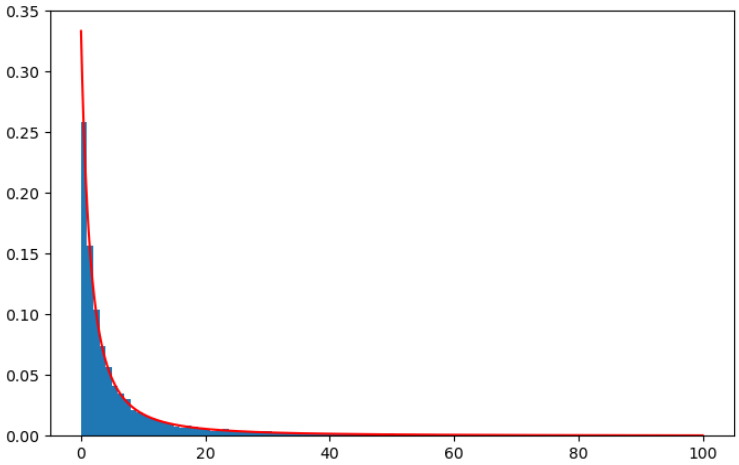


Figure 8: Pareto distribution using Composition.

4 Exercise 4: Discrete event simulation

In this exercise we write the code to run a discrete event simulation program for a blocking system, where there are $m = 10$ service units and no waiting rooms. We assume a mean service time, mst = 8 time units, a mean time between customers, mtbc = 1 time unit and we simulate it 10 times for 10000 customers.

Simulation as Poisson process

First we assume that the arrival process is a Poisson process, i.e. the interval time between arrival of customers is governed by an Exponential distribution: $I_c \sim \mathcal{E}(mtbc)$. Also the service time distribution is governed by an Exponential distribution: $S \sim \mathcal{E}(mst)$.

To write the program, we, as explained before, generate the intervals between arrivals from an Exponential distribution, and we recover the the arrival times with a cumulative sum. Then, when simulating, for every customer arriving, we check whether there are available services, if that is the case the customer is served by that service, otherwise the service is blocked and the count incremented. Finally, we compute the blocking fraction as the ratio between the number of customers blocked and total number of customers.
To generate the confidence interval we run the simulation ten times and use the following formula:

$$\left[\bar{\theta} + \frac{S_\theta}{\sqrt{(n)}} t_{\alpha/2}(n-1);\ \bar{\theta} + \frac{S_\theta}{\sqrt{(n)}} t_{1-\alpha/2}(n-1)\right]$$

Where $\bar{\theta} = \sum_{i=1}^{n} \hat{\theta}_i / n$, $S_\theta^2 = \frac{1}{n-1}(\sum_{i=1}^{n} \hat{\theta}_i^2 - n\bar{\theta}^2)$ are the usual sample mean and variance.

For this simulation, the generated CI for the blocking fraction is $[12.2275\%; 12.2284\%]$.
The result is also in agreement with the theoretical solution which is $B = 12.1661$.

Arrival process as renewal process

We run again the simulation changing the distribution of the arrival times, first to an Erlang distribution, then to a Hyper Exponential and compute again the confidence intervals.

The Erlang distribution is a Gamma distribution with an integer value k as shape parameter. Moreover, to keep the mean = 1, we set the scale parameter = $1/k$. The Confidence intervals for different values of k are illustrated in Table 2.

k	CI lower	CI upper
1	12.2316	12.2323
2	9.5787	9.5792
5	7.4516	7.4523
10	6.7616	6.7623

Table 2: Confidence intervals for the simulation with Erlang distribution to model arrival intervals

To model the arrival intervals with a Hyperexponential distribution with parameters: $p_1 = 0.8$, $\lambda_1 = 0.8333$, $p_2 = 0.2$, $\lambda_2 = 5.0$, we use the composition method to generate it and then run again the simulation.
The confidence interval in this case is: [14.2036, 14.2043].

Changing serving time distribution

We run the simulation changing the distribution of the serving time.
First, we use a constant time service, set to be equal to the mean time to serve. The confidence interval is: [12.2106, 12.2113].
Then, we model it as a Pareto distribution with different $k = 1.05$ and $k = 2.05$. The confidence interval are, respectively: [0.1019, 0.1020], and [12.0516, 12.0523].
Finally, we choose as distributions a uniform distribution modelled by a random variable $U \sim \mathcal{U}(0, 2 * mst)$, that the mean remains equal to mst. The confidence interval is: [12.2296, 12.2303]. The last distribution is a normal distribution with mean = mst, and variance = 3. The value of the variance has been chosen such that negative values are rare, since they would represent negative service times. The confidence interval is: [11.7597, 11.7602].

Comparison and interpretation of results

We make a comparison between the different CI found in the previous points.
When using exponential distributions to generate both arrival and service times, the CI has values around the expected theoretical solution. When, in point 2, we use the Erlang distribution, as it can be seen from the plot of the density for different values of k in Figure 9, the distribution tends to be shifted towards higher values, therefore leading to higher values in between customers, therefore blocking a lower amount of customers. We notice that, for $k = 1$, the Erlang distribution is an exponential, and indeed produced the same results.

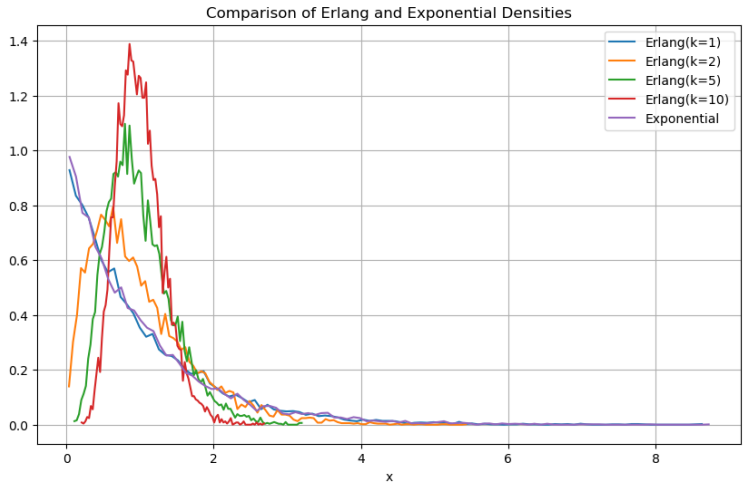


Figure 9: Plot of Erlang distribution for different values of the parameter.

When using a Hyperexponential distribution for the arrival times, in particular with the parameters illustrated above, we add the probability to sample from an exponential with

higher parameter, which leads to more values close to 0, reducing the time between arrivals and so producing more blockings.

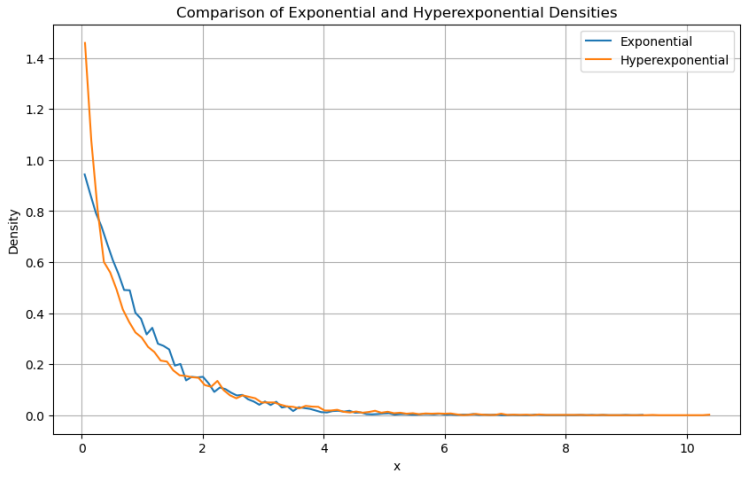


Figure 10: Plot of hyperexponential distribution vs exponential.

In part 3, instead, we change the distribution of service time. First, using a constant time service, we notice the the CI is roughly the same as for the exponential case. The same happens in the other cases, using a Pareto distribution with $k = 2.05$ and choosing a uniform and a gaussian distribution. However, using a Pareto with $k = 1.05$ we see that the blocking ratio is way lower. Comparing the two pareto distributions we notice that, for k = 1.05, the distribution presents way lower values with some sporadic high value, but not enough to balance the other low values. Therefore, the times to serve a client is way lower, resulting in fewer blocks. Whereas, for k = 2.05, we get much higher values for the distribution, which leads to higher service times and therefore a higher number of blocks.

5 Exercise 5: Variance reduction methods

Crude Monte Carlo Estimator
We estimate the integral

$$I = \int_0^1 e^x \, dx = e - 1 \approx 1.7183$$

using crude Monte Carlo:

$$\hat{I}_{\text{crude}} = \frac{1}{n} \sum_{i=1}^{n} e^{U_i}, \quad U_i \sim \mathcal{U}(0,1).$$

With $n = 100$ samples:

$$\hat{I}_{\text{crude}} \approx 1.7243, \quad 95\% \text{ CI: } [1.6764, 1.7722].$$

Antithetic Variables
We use paired variables $U_i \sim \mathcal{U}(0,1)$ and $1 - U_i$, with:

$$Y_i = \frac{1}{2}(e^{U_i} + e^{1-U_i}).$$

Then:

$$\hat{I}_{\text{anti}} = \frac{1}{n} \sum_{i=1}^{n} Y_i \approx 1.7206, \quad 95\% \text{ CI: } [1.6824, 1.7587].$$

Control Variates
Using $U \sim \mathcal{U}(0,1)$ as control variable with $\mathbb{E}[U] = 0.5$, define:

$$Z_i = e^{U_i} + c(U_i - 0.5), \quad c = -\frac{\text{Cov}(e^U, U)}{\text{Var}(U)}.$$

The estimator is:

$$\hat{I}_{\text{control}} = \frac{1}{n} \sum_{i=1}^{n} Z_i \approx 1.7210, \quad 95\% \text{ CI: } [1.6827, 1.7594].$$

Stratified Sampling
Divide $[0,1]$ into 10 equal strata. In each stratum i, draw n samples $U_i \sim \mathcal{U}(\frac{i}{10}, \frac{i+1}{10})$. The estimator is:

$$\hat{I}_{\text{stratified}} = \frac{1}{10n} \sum_{i=1}^{10} \sum_{j=1}^{n} e^{U_{i,j}}.$$

With $n = 100$ per stratum:

$$\hat{I}_{\text{stratified}} \approx 1.7198, \quad 95\% \text{ CI: } [1.6861, 1.7535].$$

Control Variates in Blocking System Simulation

We revisit the blocking system of Exercise 4, where a Poisson arrival process and exponential service times are used to estimate the blocking probability.

Let X_i be the indicator variable for whether customer i is blocked. The crude estimator is:

$$\hat{B}_{\text{crude}} = \frac{1}{n}\sum_{i=1}^{n} X_i$$

To reduce variance, we introduce a control variate Y_i that is correlated with X_i and has a known expected value. A suitable example is the number of busy servers at arrival time.

The control variate estimator becomes:

$$\hat{B}_{\text{cv}} = \frac{1}{n}\sum_{i=1}^{n} \left(X_i + c(Y_i - \mathbb{E}[Y])\right), \quad c = -\frac{\text{Cov}(X,Y)}{\text{Var}(Y)}$$

Simulation results show that the variance of the estimator $\hat{B}_{\text{cv}}$ is significantly reduced compared to $\hat{B}_{\text{crude}}$, leading to narrower confidence intervals for the estimated blocking probability.

Common Random Numbers in Blocking System

We compare the blocking probability under two different arrival processes in the loss system of Exercise 4:

- Poisson arrivals (Exercise 4.1): $\hat{B}_{\text{Poisson}} = 0.10584$
- Renewal process with hyperexponential interarrival times (Exercise 4.2b): $\hat{B}_{\text{HyperExp}} = 0.08143$

The crude estimate for the difference is:

$$\hat{\Delta}_{\text{indep}} = \hat{B}_{\text{Poisson}} - \hat{B}_{\text{HyperExp}} = 0.02441, \quad 95\%\text{ CI: } [0.01994, 0.02888]$$

Using **common random numbers (CRNs)**, we synchronize the random number streams used to generate arrivals and services in both simulations. This introduces positive correlation, reducing the variance of the difference estimator.

With CRNs, we obtain:

$$\hat{\Delta}_{\text{CRN}} = 0.02439, \quad 95\%\text{ CI: } [0.02057, 0.02821]$$

Although the point estimates are almost identical, the confidence interval under CRNs is narrower, demonstrating the variance reduction benefit of common random numbers when estimating differences between two related systems.

Importance Sampling for $\mathbb{P}(Z > a)$

Estimate $\mathbb{P}(Z > a)$, where $Z \sim \mathcal{N}(0,1)$, using: - Crude Monte Carlo:

$$\hat{P}_{\text{crude}} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}_{\{Z_i > a\}}$$

- Importance sampling with $Y \sim \mathcal{N}(a, \sigma^2)$:

$$\hat{P}_{\text{IS}} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}_{\{Y_i > a\}} \cdot \frac{\phi(Y_i)}{\phi_a(Y_i)},$$

where ϕ is the standard normal PDF, and ϕ_a is the density of $\mathcal{N}(a, \sigma^2)$.

Results show lower variance and better accuracy for IS, especially for large a, e.g., $a = 4$.

Importance Sampling for the Integral Using Exponential

We use importance sampling with proposal $g(x) = \lambda e^{-\lambda x}$ over $x \in [0, 1]$ to estimate:

$$I = \int_0^1 e^x dx.$$

The estimator becomes:

$$\hat{I}_{\text{IS}} = \frac{1}{n} \sum_{i=1}^{n} \frac{e^{X_i} \mathbf{1}_{\{X_i \leq 1\}}}{g(X_i)}, \quad X_i \sim g(x).$$

After testing several values of λ, variance was minimized near $\lambda \approx 1.2$, though overall variance reduction was limited.

Pareto Distribution and First Moment Importance Sampling

We consider using the first moment distribution (i.e., $xf(x)$) for importance sampling of a Pareto distribution $f(x) \propto x^{-(k+1)}$. This gives:

$$g(x) \propto xf(x) \propto x^{-k}$$

But this is not normalizable on $[1, \infty)$ when $k \leq 1$, making the method infeasible.

However, the insight suggests tailoring $g(x)$ more carefully,for instance via a beta or truncated distribution matching e^x on $[0, 1]$—for effective variance reduction in Question 8.

Considering the first moment distribution as sampling distribution, we have:

$$G_1(x) = \frac{\int_0^x tf(t)dt}{\mathbb{E}[X]} \implies g_1(x) = G_1'(x) = \frac{xf(x)}{\mathbb{E}[X]}$$

Using the Importance Sampling estimator becomes:

$$\mathbb{E}[X] = \int_0^\infty tf(t)dt = \int_0^\infty \frac{tf(t)}{g_1(t)} g_1(t)dt = \int_0^\infty \frac{tf(t)}{tf(t)/\mathbb{E}[X]} g_1(t)dt = \mathbb{E}[X]$$

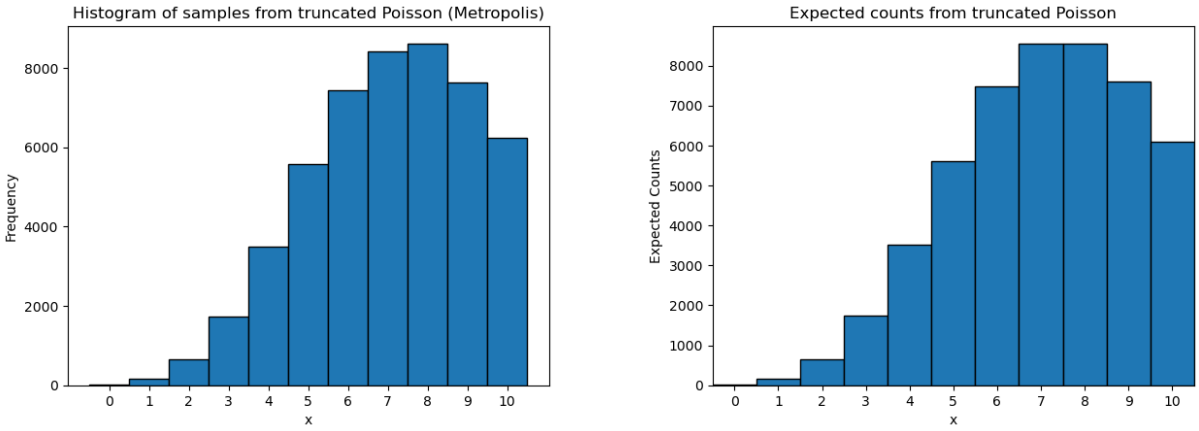
This means that the importance sampling estimator using g_1 would be a perfect estimator, as it is a constant, that is the variance is 0. However, we are assuming we can sample from g_1, which requires to know the expected value, which is our quantity of interest! This makes the approach not meaningful per se. However, this suggests that if we can find a sampling distribution which is close to the quantity of interest, we would get an estimator with very

low variance. In generale, chossing as sampling density, a g which is proportional to $xf(x)$, would reduce the variance of the estimator.
To apply this to the problem in Question 8, we would use a density function g proportional to our qunatiy of interest, i.e. $g \, \alpha \, e^X \mathbb{I}_{(0,1)}$.

6 Exercise 6: Markov Chain Monte Carlo

1) Sampling from Erlang system

In this exercise we are given a distribution $f(x) = cg(x)$ we need to sample from. In order not to compute the constant c, we use the Markov Chain Monte Carlo (MCMC) method. In particular, we propose a state, X_i, and then we make another proposal for the state Y, if $g(Y) \geq g(X_i)$, we accept Y, otherwise we accept Y with probability $g(Y)/g(X_i)$. To get a more accurate result, we run a warm-up phase to start from a better proposal. The plot of the generated (MCMC) and expected distribution is shown in figure Figure 14a and Figure 14b, respectively.



(a) Truncated Poisson generated by MCMC

(b) Expected truncated Poisson

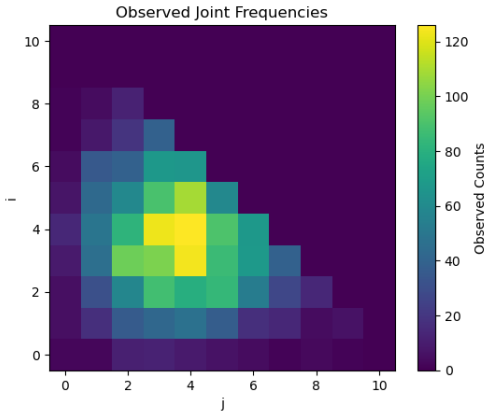
Figure 11: Comparison of MCMC-generated and expected truncated Poisson distributions

Running a χ^2 test, there is no statistical significance of difference between the two distributions

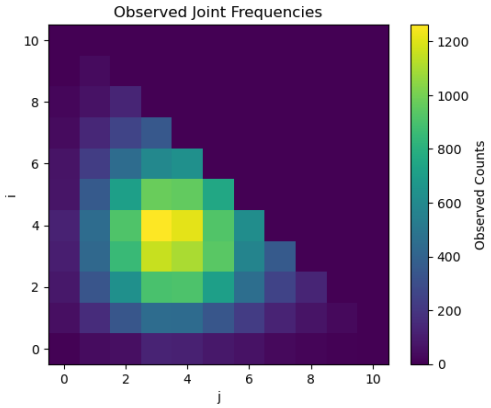
2) Joint truncated Poisson

The joint density for a truncated Poisson distribution is given by: $p(i,j) = c\frac{A_1^i}{i!}\frac{A_2^j}{j!}$ $0 \leq i + j \leq m$, m number of lines.

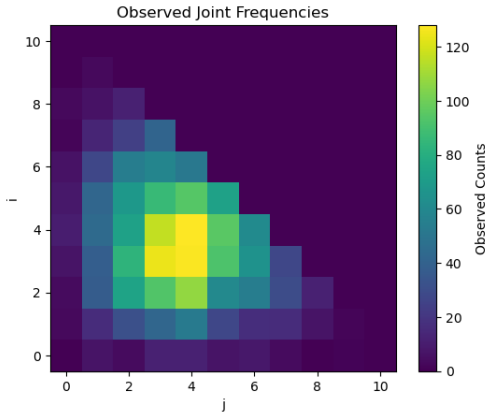
We generate samples from this distribution in three cases: using Metropolis-Hastings algorithm in the standard case, using coordinate wise to sample from the distribution, and using Gibbs sampling. The plot of the generated distributions are shown in figure Figure 12. Also in this case, the tests don't show statistically significant difference.



(a) Standard Metropolis-Hastings



(b) Coordinate wise



(c) Gibbs sampling

Figure 12: Distribution generated using the three methods

3) Bayesian statistical problem

We consider a problem where the observations are $X_i \sim \mathcal{N}(\Theta, \Psi)$, where $(\Xi, \Gamma) = (\log(\theta), \log(\Psi))$ is a standard Normal with correlation $\rho = 0.5$.

a)

we generate a pair (θ, ψ) by generating a pair (Ξ, Γ) from a Normal distribution as explained above, and then taking the exponential, i.e. we are generating a couple of the parameters from the prior distribution.

b)

Using the values for the parameter, we generate 10 samples from X_i.

c)

To derive the posterior distribution, we apply Bayes theorem:

$$f(\theta,\psi)|x_i) = \frac{f(x_i|\theta,\psi)f(\theta,\psi)}{f(x_i)} = c\, f(x_i|\theta,\psi)f(\theta,\psi)$$

Where $c = f(x_i)$. So, our objective is to sample from the posterior without having to compute c. The expression for the posterior becomes:

$$f(\theta,\psi \mid x_i) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\psi}} \exp\left(-\frac{(x_i-\theta)^2}{2\psi}\right) = \frac{1}{(\psi 2\pi)^{\frac{n}{2}}} \exp\left(-\frac{1}{2\psi}\sum_{i=1}^{n}(x_i-\theta)^2\right) =$$

$$= \frac{1}{(\psi 2\pi)^{\frac{n}{2}}} \exp\left(-\frac{n}{2\psi}\left(\hat{Var}(x_i)(x) + (\bar{X}-\theta)^2\right)\right)$$

Where we used the identity: $\sum_{i=1}^{n}(x_i-\theta)^2 = n(\hat{Var}(x_i) + (\bar{x}-\theta)^2)$.
We can then notice that, up to a constant, we have:

$$f(\theta,\psi \mid x_i)\, \alpha\, f(\bar{X} \mid \theta,\psi) f(Var(x_o) \mid \hat{\theta},\psi) f(\theta,\psi)$$

and exploit the fact that we know that $\bar{X}$ and $\hat{Var}(x_i)$ are independent with distribution: $\bar{X} \sim \mathcal{N}(\theta, \psi/n)$ and $\frac{n\hat{Var}(x_i)}{\psi} \sim \chi^2(n-1)$.

d)

We use a MCMC to sample from the posterior. As before, we use a warm-up phase and then we extract values in another loop. We select our first guess for (θ,ψ) using a standard normal distribution, making sure that the values are positive. The guess is updated summing to the current value a value obtained again from a normal standard distribution. To evaluate the choices, we use the posterior distribution computed in the previous point. The plot of the distribution, together with the values of θ and ψ we found in the previous points il shown in Figure 13

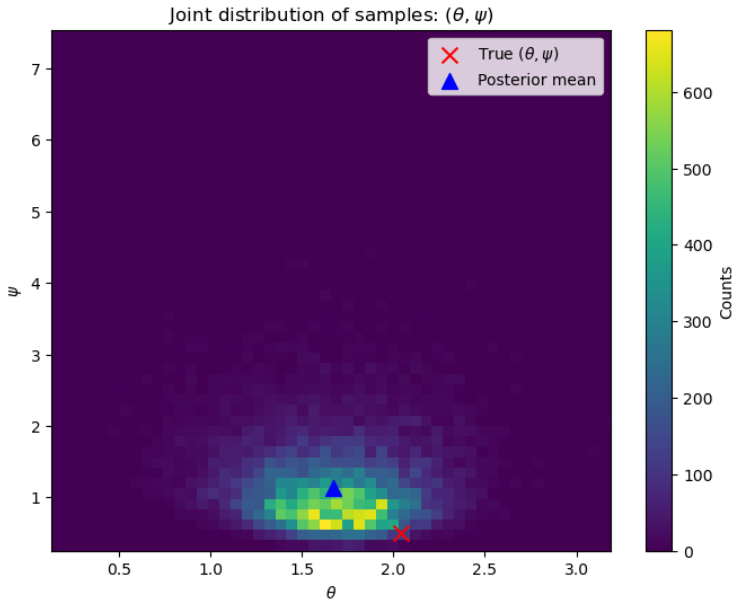
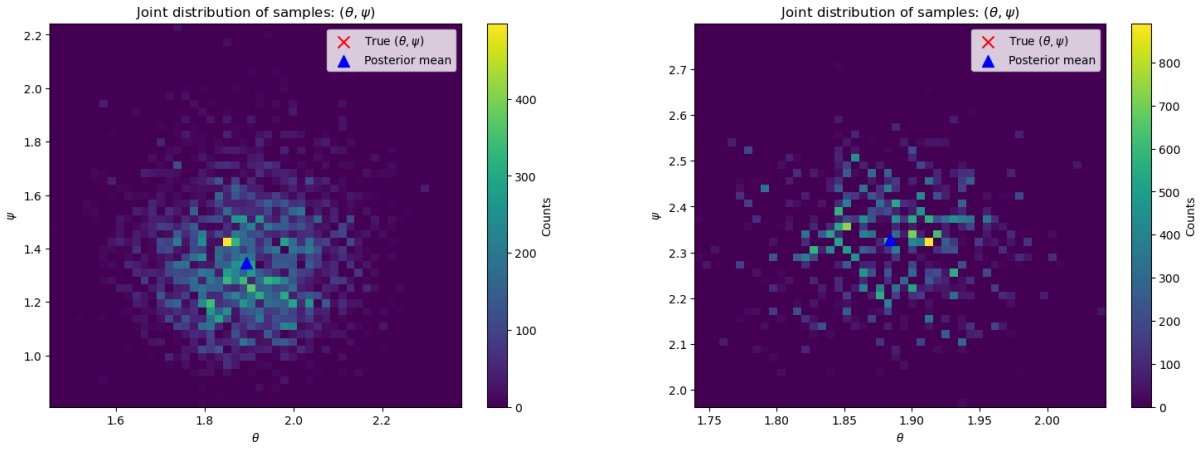


Figure 13: Distribution found using MCMC.

e)

We run again the code, using $n = 100$ and $n = 1000$ samples. We expect that in this case, values of the posterior are close to the values of the prior, as we are sampling more points. Figure **??** shows the plot of the distributions in these cases.



(a) n=100 (b) n=1000

In the case $n = 1000$ way less values are shown, this is because the posterior takes such small values that a great number of proposals is rejected, leading to have a lot of values only in one class.

7 Exercise 7: Simulated Annealing for the Travelling Salesman Problem

In this exercise, the Travelling Salesman Problem (TSP) is approached using the Simulated Annealing algorithm. The objective is to find a route that visits each station exactly once and returns to the starting point, minimizing the total travel cost.

The algorithm is first applied to a set of stations defined by their coordinates in the plane. The cost between stations is given by the Euclidean distance. To validate the implementation, the algorithm was debugged using stations arranged on a circle, where the optimal route is intuitively known.

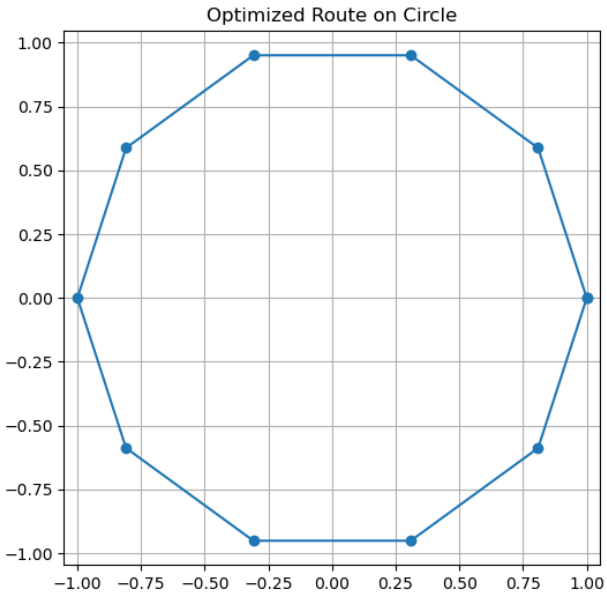


Figure 15: Debugging of the algorithm on a circle

Once validated, the method was tested on randomly positioned stations, and the final route was visualized.

A cooling schedule of the form $T_k = \frac{1}{\sqrt{1+k}}$ was used, and new routes were proposed by randomly swapping two cities in the current path. The Metropolis acceptance criterion was applied to decide whether to accept the new solution, allowing for occasional uphill moves to escape local minima (Figure 16).

In the second part of the exercise, the code was adapted to work with a cost matrix instead of coordinates. This required only minor modifications, as the energy evaluation step was adjusted to sum the corresponding entries in the cost matrix. The simulated annealing procedure again produced a significant improvement in route cost compared to the initial random route.

Overall, simulated annealing proved to be an effective heuristic for solving the TSP, demonstrating good performance in both geometric and matrix-based formulations.

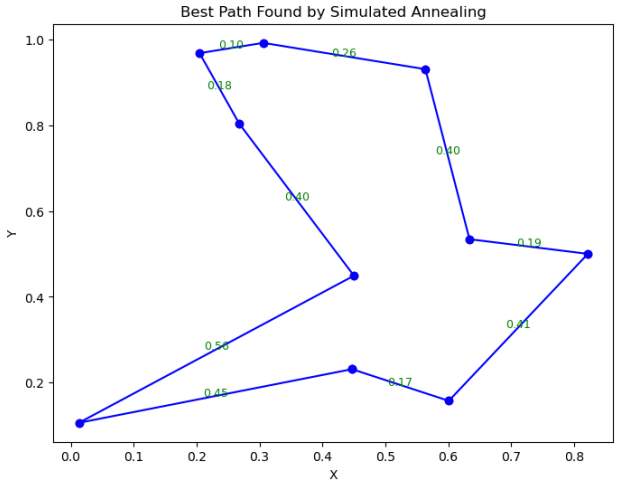


Figure 16: Best routh found through simulated annealing

8 Exercise 8: Bootstrap

8.1 Exercise 13

Given a sample $X_1, \ldots, X_{10}$ from an i.i.d. distribution with unknown mean μ, the objective is to estimate the probability

$$p = \mathbb{P}\left(a < \frac{1}{n}\sum_{i=1}^{n} X_i - \mu < b\right),$$

with $a = -5$, $b = 5$, and the observed values:

$$56,\ 101,\ 78,\ 67,\ 93,\ 87,\ 64,\ 72,\ 80,\ 69.$$

The bootstrap method is applied by first centering the data using the sample mean $\bar{X}$. The centered data approximates the distribution of $\bar{X} - \mu$. Then, 10,000 bootstrap samples are generated by resampling the centered data with replacement. For each sample, the mean is computed, and the proportion of bootstrap means falling in the interval (a, b) is calculated.

This proportion serves as the estimate of p. The result from the simulation yields:

$$\hat{p} = 0.7630.$$

This value represents the estimated probability that the sample mean deviates from the true mean by less than 5 units.

8.2 Exercise 15

Given the sample of size $n = 15$:

$$5,\ 4,\ 9,\ 6,\ 21,\ 17,\ 11,\ 20,\ 7,\ 10,\ 21,\ 15,\ 13,\ 16,\ 8,$$

the goal is to estimate the variance of the sample variance, $\mathrm{Var}(S^2)$, using the bootstrap method.

The procedure involves computing the sample variance S^2 for each of 10,000 bootstrap samples generated by resampling the original data with replacement. The variability of these bootstrap sample variances provides an empirical estimate of $\mathrm{Var}(S^2)$.

The final result, $Var(S^2) = 58.49$, is obtained by calculating the variance of the 10,000 bootstrap estimates of S^2. This approach offers a non-parametric estimate of the uncertainty associated with the sample variance.

8.3 Bootstrap analysis of mean and median for Pareto samples

A sample of size $N = 200$ is drawn from a Pareto distribution with shape parameter $k = 1.05$ and scale parameter $\beta = 1$. The goal is to estimate the variance of both the sample mean and the sample median using bootstrap resampling and to compare their precision.

The computed sample mean is approximately 7.9554, while the sample median is 1.9868. These values differ substantially due to the heavy-tailed nature of the Pareto distribution, which tends to inflate the mean more than the median.

To estimate the variance of the sample mean, $B = 10{,}000$ bootstrap resamples are generated from the observed data. The mean of each resample is calculated, and the variance of these means yields the bootstrap estimate:

$$\widehat{\mathrm{Var}}(\bar{X}) = 5.8384.$$

The same bootstrap samples are used to compute the median for each replicate. The variance of these medians provides an estimate of the variance of the sample median:

$$\widehat{\mathrm{Var}}(\tilde{X}) = 0.0326.$$

The estimated variance of the sample median is significantly smaller than that of the sample mean. This indicates higher precision of the median in this setting. The sample mean, being sensitive to extreme values, exhibits greater variability, whereas the median remains more stable under the influence of the heavy tail of the Pareto distribution.

List of Figures

References