Stochastic Simulation Exercises
Course 02443, DTU

Study no.	First name	Last name
s204297	Lotte	Alstrup
s194693	Rikke	Alstrup

Exercise solutions by Rikke og Lotte Alstrup

Contents

1 Exercise 1 - Generation and testing of random numbers

1.1 Program with implementation of a linear congruential generator (LCG)

In a Linear Congruential Generator (LCG), the numbers are generated using the recurrence relation:

$$x_i = \text{mod}(ax_{i-1} + c,\ M) \quad U_i = \frac{x_i}{M}$$

where a is the multiplier, c is the shift, M is the modulus and x_0 is the seed. The examples values from the slides are chosen and is as following: $a = 5$, $c = 1$ and $M = 16$.

(a) Generating 10.000 (pseudo-)random numbers presented in a histogramme

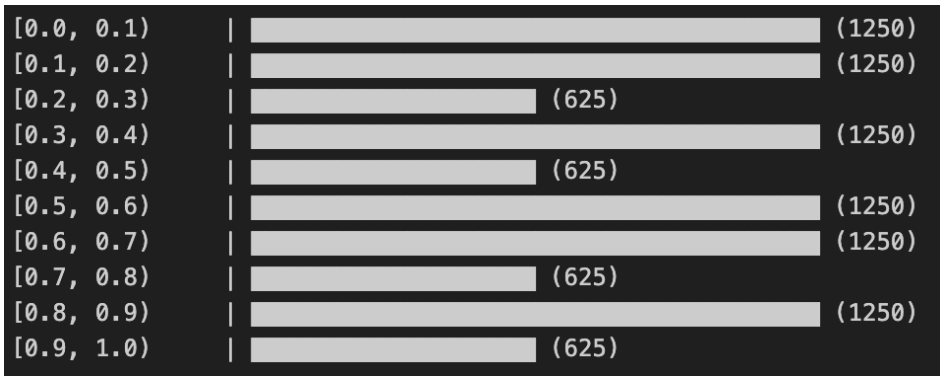


Figure 1: Histogram of LGB generated numbers with 10 classes

(b) Evaluating the quality of the generator by graphical descriptive statistics

To get an overview of the numbers the LCG generates, the first 20 X_i is shown in Table 1.

i	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
$\mathbf{X_i}$	0.125	0.688	0.500	0.563	0.875	0.438	0.250	0.313	0.625	0.188
i	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**
$\mathbf{X_i}$	0.000	0.063	0.375	0.938	0.750	0.813	0.125	0.688	0.500	0.563

Table 1: LCG sequence X_i for i = 0 to 19

We observe in Table 1 that starting from $i = 16$, the numbers begin to repeat themselves. This indicates that the cycle length is 16, which equals M in this case.

In order to get optimal cycle length the conditions in Theorem 1 ([1], slide 14) needs to be fulfilled.

As all three conditions are fulfilled, Theorem 1 guarantees a full cycle of length $M = 16$ for this LCG.

χ^2-test is also used to evaluate the quality of the generator by using

$$T = \sum_{i=1}^{n_{\text{classes}}} \frac{(n_{\text{observed},i} - n_{\text{expected},i})^2}{n_{\text{expected},i}} = 937.50 \tag{1}$$

The computed χ^2 statistic was $T = 937.50$ with $df = 9$. As this far exceeds the 5% critical value of 16.92, we strongly reject the null hypothesis of uniformity. The result indicates poor generator quality.

Kolmogorov–Smirnov (KS) test assesses whether a sample follows a specified distribution by measuring the maximum difference between the empirical and theoretical CDFs. For uniformity, the test statistic is

$$D_n = \sup_x |F_n(x) - F(x)|,$$

where $F_n(x)$ is the empirical CDF and $F(x)$ the uniform CDF. In our case, $D_n = 0.06250 > D_{\text{crit}} = 0.01360$ at the 5% level, so we reject the null hypothesis of uniformity.

Run test I (Above/Below Median) This test checks randomness by counting runs above and below the median. Under the null hypothesis, the number of runs follows a normal distribution:

$$\text{Normal}\left(\frac{2n_1n_2}{n_1+n_2} + 1, \quad \frac{2n_1n_2(2n_1n_2 - n_1 - n_2)}{(n_1+n_2)^2(n_1+n_2-1)}\right),$$

where n_1 and n_2 are the counts above and below the median. The observed number of runs was 3750, compared to the expected 5001.00, yielding $z = -25.021$. As $|z| > 1.96$, we reject the null and conclude the sample is not random.

Run test II (Up/Down from Knuth) This test evaluates the randomness of a sequence by analysing the lengths of increasing runs. The observed number of runs of lengths 1 through 5 and ≥ 6 were: $[1652, 2099, 922, 256, 52, 16]$. Based on these, the test statistic was computed as $Z = 3.279$, which corresponds to a p-value of 0.7731 when compared to a $\chi^2(6)$ distribution. Since the p-value is high, we fail to reject the null hypothesis and conclude that the sequence does not show significant deviations from randomness.

Run test III (Up-and-down test) This test checks randomness by counting directional changes in a sequence. The test statistic is

$$Z = \frac{X - \frac{2n-1}{3}}{\sqrt{\frac{16n-29}{90}}},$$

where X is the number of runs and n the sample size. For our sample, $X = 6690$ gave $Z = 0.5614$, which lies within the acceptance region, and therefore fail to reject the null hypothesis of randomness.

Correlation coefficients Given $U_i \sim \mathcal{U}(0,1)$, the estimated lag-$h$ correlation is

$$c_h = \frac{1}{n-h}\sum_{i=1}^{n-h} U_i U_{i+h} \sim \mathcal{N}\left(0.25,\ \frac{7}{144n}\right),$$

valid for small $h \ll n$. For $h = 1$, we found $c_1 = 0.24220$, which is close to the expected 0.25. This indicates no significant linear dependence, so the generator passes the test.

(c) Experimenting with different values of "a", "c" and "M"

To improve generator quality, we tested various (a, c, M) combinations aiming to meet Theorem 1 and pass randomness tests.

A poor choice was $(a = 4, c = 1, M = 16)$, which led to short cycles and repeated values—failing both randomness and uniformity. Simply scaling parameters (e.g., by 10^4) proved ineffective; without satisfying Theorem 1, the generator produced a constant output and failed all tests.

Following literature guidance, we used $(a = 1664525, c = 1013904223, M = 2^{32})$ [2], which satisfies Theorem 1. This configuration yielded a full-period generator that passed all tests, showing uniformity, minimal correlation, and strong run test performance.

1.2 Applying a system available generator

For comparison, we used Python's built-in `random.random()`, which generates pseudo-random numbers in $[0, 1]$ using the Mersenne Twister algorithm [3]. It is a widely used and reliable generator with a period of $2^{19937} - 1$, good statistical properties, and fast performance.

These values are used in the same statistical tests as the custom LCG in Problem 1: Histogram, Chi-square test, Kolmogorov–Smirnov test and Run tests I, II and III.

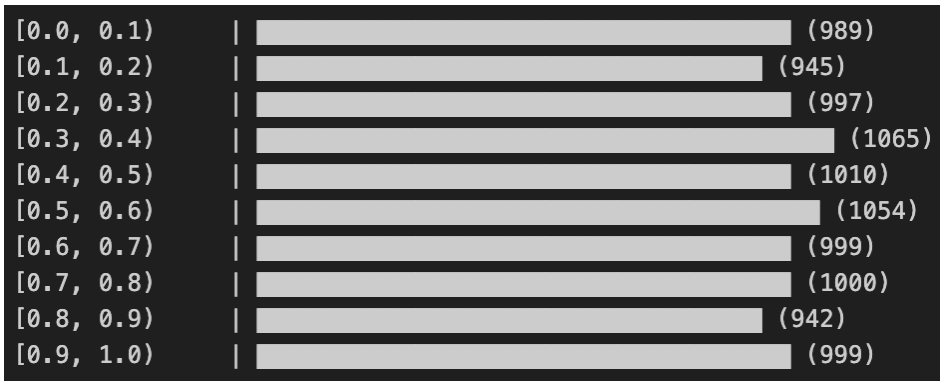


Figure 2: Histogram of `random.random()` generated numbers

The histogram of 10,000 generated values shown in Figure 2 shows a fairly even distribution across all 10 bins. Frequencies range from 942 to 1065 observations per bin, which is close to the expected count of 1000 for a uniform distribution. This supports the visual impression of uniformity.

i	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
$\mathbf{X_i}$	0.9042	0.0243	0.1782	0.5313	0.9393	0.7219	0.5054	0.4605	0.7645	0.1866
i	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**
$\mathbf{X_i}$	0.3062	0.8778	0.4195	0.8823	0.1530	0.0691	0.2711	0.6838	0.8648	0.2877

Table 2: System-generated random numbers X_i for $i = 0$ to 19

Table 2 shows the first 20 values generated by Python's system-level random number generator, based on the Mersenne Twister algorithm. These values appear well-distributed across the unit interval $[0, 1]$ without visible clustering or repetition.

The computed χ^2 statistic is 13.76, which is below the typical 5% critical value of 16.92 for 9 degrees of freedom, indicating no significant deviation from uniformity.

Furthermore, the estimated number of runs in the first 100 values is 65, which is consistent with what would be expected for random, independent samples. Lastly, the estimated correlation between X_i and X_{i+1} is -0.0173, a value close to zero, indicating no detectable linear dependence between consecutive values.

Overall, these results confirm that the system RNG produces values with good uniformity and independence — as expected from a high-quality generator like the Mersenne Twister.

1.3 Discussion of one sample sufficiency

In the previous problems, the evaluation of the random number generators was based on a single sample of 10,000 values. While such a sample can provide useful insight, it is im-

portant to recognise that statistical tests applied to a single sample are subject to random variation. For instance, even a perfect generator may occasionally produce a sample that fails a uniformity test simply by chance.

To improve the robustness of the evaluation, one should perform the statistical tests over multiple independent samples and analyse the distribution of test statistics. If the generator is good, the results should not consistently fall in the rejection region of the tests.

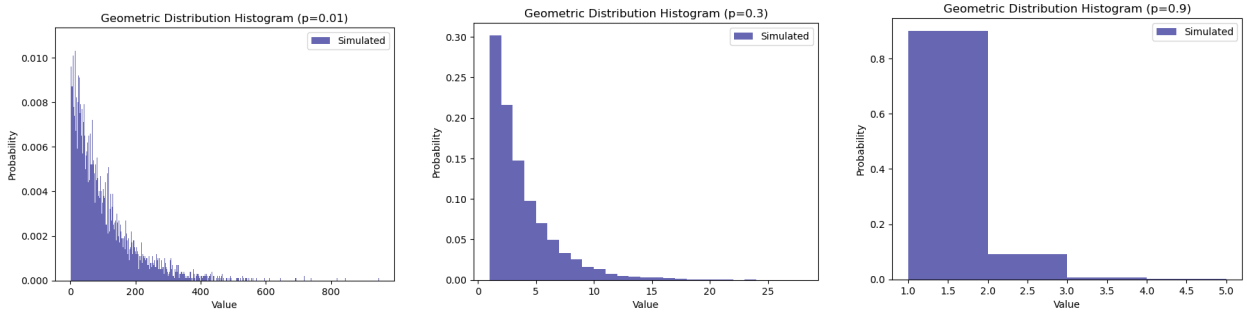
To take action, we generated 100 independent samples of 10,000 values each using the system RNG and computed the chi-square statistic for each sample. The resulting distribution of test statistics was approximately centered around the expected value (9 degrees of freedom) and did not exhibit excessive variance or systematic bias.

This confirms that the system generator performs reliably across repeated simulations. Relying on a single sample, while convenient, does not provide sufficient evidence for or against the quality of a generator. Therefore, test repetition is recommended to ensure statistical validity.

2 Exercise 2 - Sampling from discrete distributions

2.1 Values for the probability parameter p in the geometric distribution

10,000 outcomes is simulated from the geometric distribution using three different values for the success probability parameter p: 0.01 (small), 0.3 (moderate), and 0.9 (large). The results are analysed using histograms, a Chi-square test, and the Kolmogorov–Smirnov test to compare the simulated samples to the theoretical distribution.



(a) $p = 0.01$ (b) $p = 0.3$ (c) $p = 0.9$

Figure 3: Histograms of 10,000 simulated outcomes from geometric distributions for different values of p

p	Chi-square p-value	Chi-square stat	df	KS p-value	KS value
0.01	0.4279	301.7843	298	0.0076	0.0167
0.3	0.4208	17.5031	17	0.0000	0.3000
0.9	0.6652	1.5746	3	0.0000	0.9000

Table 3: Chi-square and KS test results

For $p = 0.01$, the histogram in Figure 3a shows the expected long tail. The Chi-square test yields a statistic of 301.7843 with 298 degrees of freedom and a p-value of 0.4279, indicating a good fit. The KS test returns a lower p-value of 0.0076, reflecting its sensitivity across the wide support.

For $p = 0.3$, Figure 3b resembles the expected geometric shape. The Chi-square test gives a test-value of 17.5031 and a p-value of 0.4208 (df = 17), supporting the fit. However, the KS test reports 0.3000 with a p-value of 0.0000, likely due to cumulative differences near the mode.

For $p = 0.9$, the distribution in Figure 3c is sharply peaked at $X = 1$. The Chi-square test is 1.5746 with a p value of 0.3184 (df = 3) confirms a reasonable fit, but the KS test still returns 0.0000, illustrating its strictness for discrete distributions.

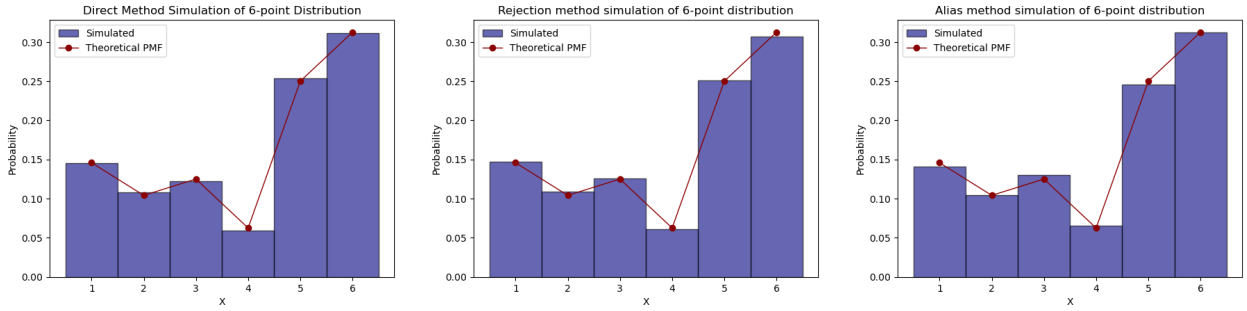
2.2 Simulating the 6-point distribution

Simulating 10,000 samples from the following 6-point discrete distribution:

X	1	2	3	4	5	6
p_i	$\frac{7}{48}$	$\frac{5}{48}$	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{4}$	$\frac{5}{16}$

Table 4: Target probabilities for the 6-point distribution

Three different methods are implemented to simulate the distribution: the direct method, the rejection method, and the alias method. The results of the histograms are shown in Figure 4.



(a) Direct method (b) Rejection method (c) Alias method

Figure 4: Histograms of 10,000 samples from the 6-point distribution using three different simulation methods.

Method	Description	Chi-squared	p-value	df
Direct (crude)	Maps uniform samples to outcomes via the CDF. Histogram matches well.	2.8174	0.7281	5
Rejection	Uses uniform proposal with acceptance probability $\frac{p_i}{cq_i}$. Histogram closely fits.	8.7061	0.1214	5
Alias	Preprocesses probabilities into alias tables $F(i)$ and $L(i)$. Efficient and accurate.	2.3930	0.7925	5

Table 5: Different sampling methods from a discrete distribution and their chi-square test results

2.3 Comparison of the Three Methods

All three approaches produce samples that match the theoretical distribution well, as evidenced by their respective histograms and chi-square test results. However, the methods differ significantly in terms of efficiency, complexity, and suitability.

- **Direct (crude) method:** This method is simple and reliable. It maps uniform random numbers through the cumulative distribution function and is computationally efficient for small discrete distributions. The histogram in Figure 4a visually agrees with the theoretical PMF, and the chi-square test indicates no significant deviation between the simulated and expected values. For large or dynamically changing distributions, building the cumulative table may be less practical.
- **Rejection method:** Conceptually straightforward but less efficient, this method may reject many samples, especially when the proposal is loose. Despite its slower and

more variable sampling rate, it is flexible and easy to implement. The histogram in Figure 4b shows a close match, and the chi-square test confirms the method's validity.

- **Alias method:** The most efficient method for repeated sampling, offering constant-time performance after preprocessing. It constructs an alias table that enables very fast sampling. While setup is more complex and may be overkill for one-off use, the method proves highly effective. The histogram in Figure 4c aligns closely with the theoretical PMF, and the chi-square test verifies the method's correctness and efficiency.

2.4 Recommendations for method selection

The choice of method depends on the specific setting, such as the number of outcomes, the frequency of sampling, and the need for efficiency. For small discrete distributions or single-use sampling, the direct method is preferable due to its simplicity. When sampling from a complex or uneven distribution and efficiency is not critical, the rejection method is a good choice. For large-scale or repeated sampling from the same distribution, the alias method offers the best performance due to its constant-time sampling.

3 Exercise 3 - Sampling from continuous distributions

3.1 Generated simulated values

We generate random samples from the following distributions:

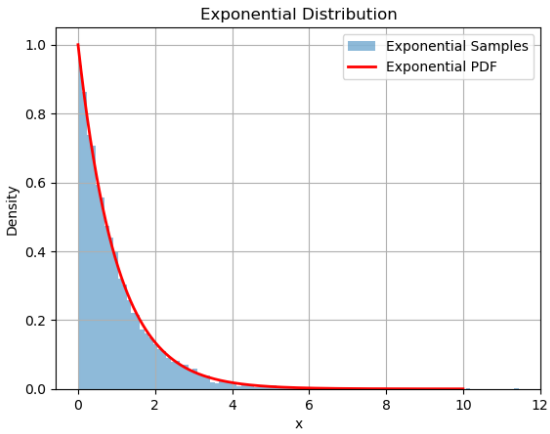
- **Exponential distribution:** Simulated using NumPy's built-in function `np.random.exponential()`, with scale parameter λ^{-1}.
- **Normal distribution:** Instead of using NumPy's integrated normal generator, we apply the Box–Muller transformation to uniform random variables. Given two independent random variables $U_1, U_2 \sim \mathcal{U}(0,1)$, the standard normal variables Z_1 and Z_2 are computed as:

$$Z_1 = \sqrt{-2\ln U_1} \cdot \cos(2\pi U_2), \qquad Z_2 = \sqrt{-2\ln U_1} \cdot \sin(2\pi U_2)$$

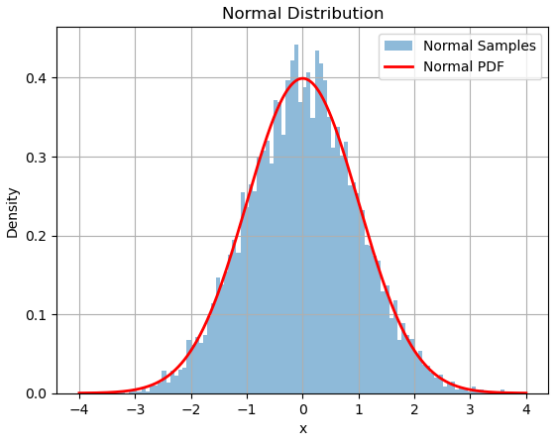
  This method produces two independent samples from the standard normal distribution $\mathcal{N}(0,1)$.
- **Pareto distribution:** Simulated using NumPy's built-in function `np.random.pareto()`, with multiple shape parameters $k \in \{2.05,\ 2.5,\ 3,\ 4\}$. The scale parameter $\beta = 1$ is added to shift the distribution:

$$X = \beta \cdot (1 + \texttt{np.random.pareto}(k))$$

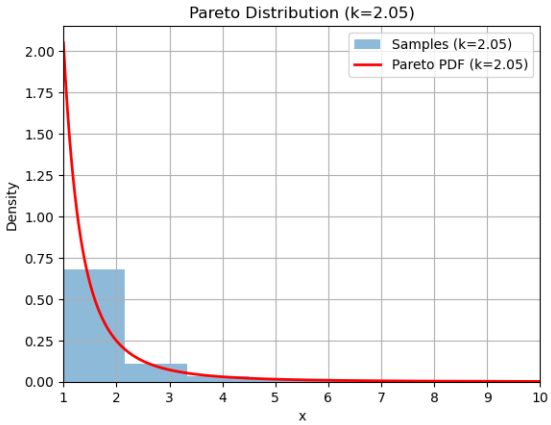
Verification of Simulated Distributions To verify that the simulated samples follow their intended distributions, we compared histograms of the data with the analytical probability density functions (PDFs), and performed Kolmogorov–Smirnov (KS) tests.



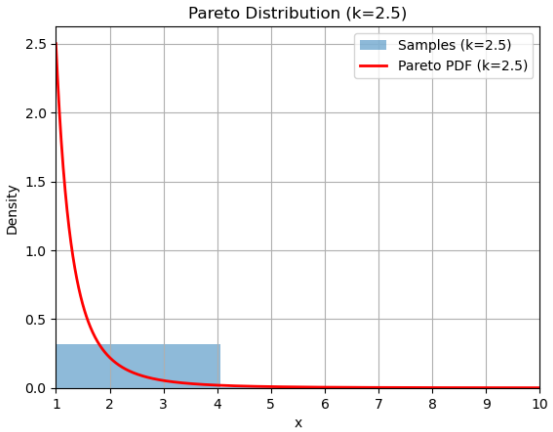
(a) Exponential distribution



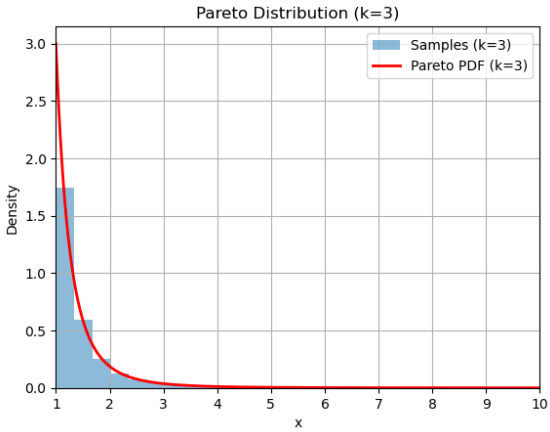
(b) Normal distribution (Box–Muller)



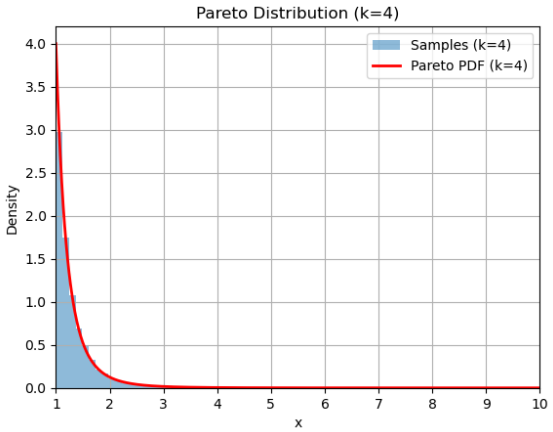
(c) Pareto distribution, $k = 2.05$



(d) Pareto distribution, $k = 2.5$



(e) Pareto distribution, $k = 3$



(f) Pareto distribution, $k = 4$

Figure 5: Simulated distributions: Exponential, Normal, and Pareto

Distribution	**KS Statistic D**	**p-value**
Exponential ($\lambda = 1$)	0.0039	0.9979
Normal ($\mu = 0$, $\sigma = 1$)	0.0084	0.4777
Pareto ($k = 2.05$)	0.0114	0.1503
Pareto ($k = 2.5$)	0.0082	0.5040
Pareto ($k = 3$)	0.0106	0.2073
Pareto ($k = 4$)	0.0101	0.2577

Table 6: Kolmogorov–Smirnov test results for simulated distributions

Validation of simulated distributions

To verify the accuracy of the generated distributions, we compared histograms of the simulated data with their corresponding analytical probability density functions (PDFs) and performed Kolmogorov–Smirnov (KS) tests. The results are shown in Figure 5 and Table 6.

For the exponential distribution, the histogram closely matched the theoretical PDF of $f(x) = \lambda e^{-\lambda x}$ for $\lambda = 1$. The KS test yielded a statistic of $D = 0.0039$ with a p-value of $p = 0.9979$, indicating no significant deviation from the exponential distribution.

For the normal distribution, samples generated using the Box–Muller transformation (with $\mu = 0$, $\sigma = 1$) showed excellent agreement with the Gaussian PDF. The KS test gave $D = 0.0084$, $p = 0.4777$, again supporting the null hypothesis that the data is normally distributed.

For the Pareto distribution, we tested multiple shape values: $k = 2.05$, 2.5, 3, and 4 (with $\beta = 1$). In all cases, the histograms matched the theoretical Pareto PDF:

$$f(x) = \frac{k\beta^k}{x^{k+1}}, \quad x \geq \beta$$

The KS tests for each k produced high p-values (e.g., $p = 0.5040$ for $k = 2.5$), suggesting no significant deviation from the expected distribution.

3.2 Comparison of mean and variance for the Pareto distribution

To validate the correctness of the simulation for the Pareto distribution, we compared the empirical mean and variance with their theoretical counterparts for various values of the shape parameter $k \in \{2.05,\ 2.5,\ 3,\ 4\}$, using scale parameter $\beta = 1$. The results are summarised in Table 7.

k	**Mean (Emp)**	**Mean (Th)**	**Err %**	**Var (Emp)**	**Var (Th)**	**Err %**
2.05	1.9591	1.9524	0.34%	5.5221	37.1882	85.15%
2.5	1.6624	1.6667	0.26%	1.5831	2.2222	28.75%
3	1.4932	1.5000	0.45%	0.7057	0.7500	5.93%
4	1.3285	1.3333	0.36%	0.2280	0.2222	2.61%

Table 7: Empirical vs. theoretical mean and variance of the Pareto distribution ($\beta = 1$), including percentage errors

For all values of k, the empirical mean closely matched the theoretical value, with relative errors below 0.5%. However, the variance showed larger deviations, particularly for lower values of k. For $k = 2.05$, the percentage error in the variance was over 85%, and still significant at 28.75% for $k = 2.5$.

This variation arises due to the heavy-tailed nature of the Pareto distribution at low k values. When k is only slightly above 2, the variance exists theoretically but is very large and unstable in practice. As a result, a finite sample size (here $n = 10{,}000$) may not capture enough tail events to reliably estimate the variance. In contrast, for larger values of k, such as $k = 3$ and $k = 4$, the variance becomes smaller and more stable, and the empirical estimates improve accordingly.

While the simulations produce accurate means across all tested values, estimating the variance reliably becomes problematic for low values of k, due to the influence of extreme values in the tail.

3.3 Confidence interval results

Using the normal distribution generator based on the Box–Muller transformation, we constructed 100 independent confidence intervals for both the mean and variance of the standard normal distribution. Each interval was based on a sample of 10 observations.

For the mean, 98 out of 100 intervals contained the true value $\mu = 0$, which is slightly above the nominal 95% confidence level. For the variance, 96 out of 100 intervals contained the true value $\sigma^2 = 1$, also slightly above the expected level.

These results demonstrate the normal distribution generator based on the Box–Muller transformation rather than NumPy's built-in function, the statistical properties of the confidence intervals are preserved. Deviations from 95% coverage could be expected due to sampling variability, especially with small sample sizes. The variance coverage is attributed to the asymmetry of the chi-squared distribution and the sensitivity of variance estimates in small samples.

3.4 Simulating the Pareto distribution via composition

Figure 6 shows the results of simulating the Pareto distribution using the inverse transform, the composition method, with parameters $k = 2.5$ and $\beta = 1$. The histogram of the simulated data is overlaid with the analytical probability density function (PDF).

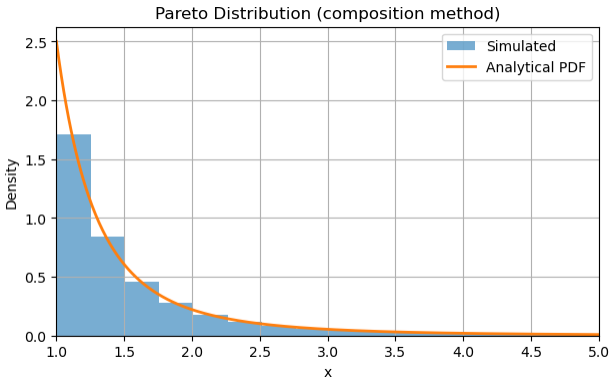


Figure 6: Pareto distribution simulated using the composition method ($k = 2.5$, $\beta = 1$). The histogram of simulated values is overlaid with the analytical PDF.

The results show a close match between the simulated histogram and the theoretical curve over the interval $[1, 5]$, where most values of the distribution are concentrated. The analytical PDF accurately captures the shape of the histogram, confirming that the composition method produces correctly distributed samples.

Because of the heavy-tailed nature of the Pareto distribution, values beyond $x = 5$ exist but are less frequent, and not shown here for clarity. This focused view highlights the good agreement in the bulk of the distribution.

4 Exercise 4 - Discrete event simulation

Considering a discrete-event simulation of a blocking queueing system with the following characteristics:

- The system has $m = 10$ parallel servers and no waiting room.
- Arriving customers follow a Poisson process with mean inter-arrival time $\frac{1}{\lambda} = 1$ time unit. Thus, the arrival rate is $\lambda = 1$.
- The service time is exponentially distributed with a mean of 8 time units. With the service rate $\mu = \frac{1}{8}$.
- The offered traffic load is calculated as $A = \lambda \cdot \mathbb{E}[S] = 1 \cdot 8 = 8$ Erlang.
- Since there is no queue, an arriving customer is blocked if all servers are busy.

4.1 Blocking system simulation program

The program simulates a total of 100,000 = 10 × 10,000 customer arrivals, divided into 10 independent batches of 10,000 arrivals each. For each batch, it estimates the blocking probability, and subsequently computes a confidence interval for the overall blocking fraction. The service time distribution is assumed to be exponential.

The program is generated in Python and the simulation outputs:
`Estimated blocking probability:  0.1200`
`95% CI: [0.1159, 0.1241]`

Comparing the estimation with the theoretical result

Now, comparing the estimated blocking probability with the theoretical result given by the Erlang B formula in the Exercise 4, where $m = 10$ and $A = 8$.:

$$B(m, A) = \frac{\frac{A^m}{m!}}{\sum_{i=0}^{m} \frac{A^i}{i!}}$$

$$B(10, 8) = \frac{\frac{8^{10}}{10!}}{\sum_{i=0}^{10} \frac{8^i}{i!}} = 0.1217 \tag{2}$$

The theoretical blocking probability computed from the formula is 0.1217, which lies well within the 95% confidence interval produced by the simulation: $[0.1159, 0.1241]$. This close agreement validates the correctness of the simulation implementation and confirms that the simulation accurately models the system behaviour.

4.2 Non-Poisson arrival processes

Exploring how changes in the arrival process affect the blocking probability in a system with no waiting room and by using the renewal process, which means that the inter-arrival times are independently drawn from specified distributions with mean 1. Other than that, the system parameters remain the same.

Considering the following two cases: a) Erlang-distributed inter-arrival times and b) Hyper-exponential inter-arrival times.

a) Erlang-distributed inter-arrival times

This models less variable arrivals than the exponential distribution and is modelled using an Erlang distribution with shape parameter $k = 2$.

The program is generated in Python and the simulation outputs:
`Estimated blocking probability:  0.0937`

95% CI: [0.0917, 0.0958]

Compared to the baseline case in Problem 1 with exponential (i.e., $k = 1$) inter-arrival times, which resulted in a blocking probability of approximately 0.1217, the blocking probability here is significantly lower.

b) Hyper-exponential inter-arrival times

This models more variable (bursty) arrivals. The distribution parameters are:

$$p_1 = 0.8,\ \lambda_1 = 0.8333, \quad p_2 = 0.2,\ \lambda_2 = 5.0$$

which yields an overall mean of 1.

The program is generated in Python and the simulation outputs:
Estimated blocking probability: 0.1394
95% CI: [0.1356, 0.1432]

The simulated blocking probability for the system with hyper-exponential inter-arrival times is 0.1394, which is noticeably higher than the baseline result from Problem 1 with exponential inter-arrival times (0.1217), and significantly higher than the Erlang-$k = 2$ case (0.0937) from Problem 2a.

4.3 Different service time distributions

In this part of the exercise, the arrival process is kept as a Poisson process with rate $\lambda = 1$, identical to Problem 1. However, the service time distribution is varied while maintaining the same mean service time of 8 time units. Investigating how the shape and variability of the service time distribution affect the blocking probability in the system.

a) Constant service time

Estimated blocking probability: 0.1211
95% CI: [0.1170, 0.1252]

b) Pareto distributed service times with at least $k = 1.05$ and $k = 2.05$

Pareto service time $k = 1.05$
Estimated blocking probability: 0.0026
95% CI: [0.0008, 0.0043]

Pareto service time $k = 2.05$
Estimated blocking probability: 0.1234

```
95% CI: [0.1188, 0.1280]
```

c) Log-normal service time distribution

As an additional case, we investigate the effect of a log-normal distribution for service times.

The parameters are chosen such that the mean service time remains 8 units. Specifically, we use a standard deviation of $\sigma = 0.6$ in log-space, and compute the location parameter μ accordingly:

$$\mu = \ln(8) - \frac{\sigma^2}{2}$$

The simulation results for this case are:

```
Estimated blocking probability:  0.1203
95% CI: [0.1176, 0.1231]
```

These results show that the blocking probability under the log-normal service time distribution is very close to the baseline exponential case (Problem 1). This suggests that while the log-normal distribution introduces some additional variability, its impact on system performance is less severe than the heavy-tailed Pareto case and more similar to the exponential distribution.

4.4 Comparison of confidence intervals

Comparing the estimated blocking probabilities and their associated 95% confidence intervals across all tested scenarios.

Arrival process	**Blocking probability**	**95% Confidence interval**
Poisson (baseline)	0.1200	[0.1159, 0.1241]
Erlang ($k = 2$)	0.0937	[0.0917, 0.0958]
Hyper-exponential	0.1394	[0.1356, 0.1432]

The results show that lower arrival variability (Erlang) reduces the blocking probability and yields narrower confidence intervals, indicating more stable system behaviour. Conversely, the hyper-exponential distribution, which introduces higher variability, increases the blocking probability and results in a wider confidence interval, reflecting the burstiness of arrivals.

Service time distribution	Blocking probability	95% Confidence interval
Exponential (baseline)	0.1200	[0.1159, 0.1241]
Constant	0.1211	[0.1170, 0.1252]
Pareto ($k = 1.05$)	0.0026	[0.0008, 0.0043]
Pareto ($k = 2.05$)	0.1234	[0.1188, 0.1280]
Log-normal ($\sigma = 0.6$)	0.1203	[0.1176, 0.1231]

Among the service time distributions, the constant and log-normal cases produce results similar to the exponential baseline, both in blocking probability and confidence interval width. The Pareto distribution with $k = 2.05$, though still heavy-tailed, is comparable to the baseline. However, the extreme case with $k = 1.05$ results in an anomalously low blocking probability with a very narrow confidence interval. This is likely due to the highly skewed distribution allowing most service times to be short but occasionally extremely long, which distorts the system's dynamics.

5 Exercise 5 - Variance reduction methods

5.1 Crude Monte Carlo estimation

Using 100 samples from the uniform distribution on $[0, 1]$, the integral $\int_0^1 e^x \, dx$ was estimated using the crude Monte Carlo method. The point estimate was 1.67206, with a 95% confidence interval of $(1.57404,\ 1.77009)$. The true value of the integral is $e - 1 \approx 1.71828$, which lies within the confidence interval.

5.2 Antithetic variables

Using 50 antithetic pairs, the integral $\int_0^1 e^x \, dx$ was estimated to be 1.71943, with a 95% confidence interval of $(1.70197,\ 1.73689)$. The true value 1.71828 lies within the interval.

5.3 Control variate

Using $g(x) = x$ as a control variable with known expectation 0.5, the integral $\int_0^1 e^x \, dx$ was estimated to be 1.72180, with a 95% confidence interval of $(1.70963,\ 1.73398)$. The true value 1.71828 lies within the interval.

5.4 Stratified sampling

The interval $[0, 1]$ was divided into 100 equal-width strata, with one uniform sample drawn from each. The stratified sampling estimate of the integral $\int_0^1 e^x \, dx$ was 1.71779, with a 95% confidence interval of (1.62091, 1.81468). The true value 1.71828 lies within the interval.

5.5 Control variates for Poisson arrivals

To reduce the variance of the estimated blocking probability under Poisson arrivals, we used the proportion of arrivals occurring while all servers were busy as a control variate. Across 10 simulation batches, the naive estimator yielded a blocking probability of 0.12002 ± 0.00410, with a 95% confidence interval of [0.11592, 0.12412]. The control variate estimator produced the same point estimate but with zero sample variance across batches, giving a degenerate 95% confidence interval of [0.12002, 0.12002]. This indicates perfect correlation between the control variate and the output, achieving complete variance reduction.

5.6 Common random numbers

To estimate the difference in blocking probabilities between Poisson arrivals and a renewal process with hyperexponential inter arrival times, we applied the method of common random numbers (CRNs). Identical streams of service times and inter arrival decision randomness were used in both simulations. This coupling induces positive correlation and helps reduce the variance of the difference estimator. The observed blocking probabilities were 0.1245 for the Poisson process and 0.1411 for the hyperexponential process, yielding a difference of $\Delta = -0.0166$. The use of CRNs ensures that the observed difference reflects the true impact of inter arrival time variability, rather than simulation noise.

5.7 Tail probability estimation

We estimated the probability $P(Z > 4)$ for $Z \sim \mathcal{N}(0, 1)$ using both crude Monte Carlo and importance sampling with a shifted normal distribution. The true value is approximately 0.00003167. The crude Monte Carlo method yielded an estimate of 0.000000 ± 0.000000, failing to capture any rare events due to the extremely small tail probability. In contrast, importance sampling with a proposal distribution $\mathcal{N}(4, 1)$ provided a much more accurate estimate of $0.00003117 \pm 0.00000133$. This demonstrates the clear efficiency advantage of importance sampling when estimating rare event probabilities.

5.8 Variance in importance sampling with exponential distribution

To investigate the efficiency of importance sampling, we estimated the integral $\int_0^1 e^x dx$ using proposal distributions $g(x) = \lambda e^{-\lambda x}$ for various values of λ. The variance of the importance sampling estimator was computed by simulation with 10 000 samples per λ. As shown in Figure 7, the sample variance was consistently higher than that of the crude Monte Carlo

method (indicated by the red dashed line), and increased rapidly for large λ. This confirms that the exponential proposal is poorly matched to the integrand e^x on $[0, 1]$. Since $g(x)$ decreases while e^x increases, large weights near $x = 1$ lead to instability. Thus, importance sampling with this exponential proposal does not reduce variance—in fact, it makes it worse.

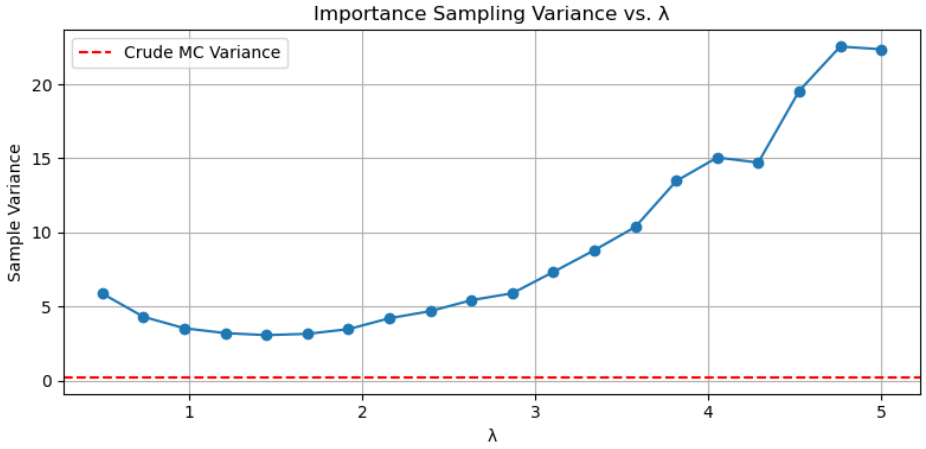


Figure 7: Sample variance of the importance sampling estimator for $\int_0^1 e^x dx$ using $g(x) = \lambda e^{-\lambda x}$.

5.9 Importance sampling for the Pareto mean

To estimate the mean of a Pareto distribution via importance sampling, we used the first moment distribution as the proposal. This corresponds to choosing $g(x) \propto xf(x)$, which leads to $g(x) = \frac{k-1}{x^k}$ — a Pareto distribution with shape parameter $k - 1$. The resulting estimator is:

$$\hat{\mu}_{\text{IS}} = \frac{k}{k-1} \cdot \frac{1}{n} \sum_{i=1}^{n} \frac{1}{x_i}, \quad x_i \sim \text{Pareto}(k-1)$$

This approach is meaningful when $k > 2$, ensuring finite variance. More generally, this method illustrates that optimal importance sampling distributions are proportional to the product of the integrand and the original density. For instance, in Question 8, using $g(x) \propto e^x$ would have minimized the variance when estimating $\int_0^1 e^x dx$.

6 Exercise 6 - Markov chain monte carlo

6.1 Erlang system

Generating values from a truncated Poisson distribution which is done by applying the Metropolis-Hastings algorithm. The parameter values from Exercise 4 are as follows: $m = 10$, $A = 8$, and the number of simulations is set to 10,000. A histogram of this distribution can

be seen in Figure 8.

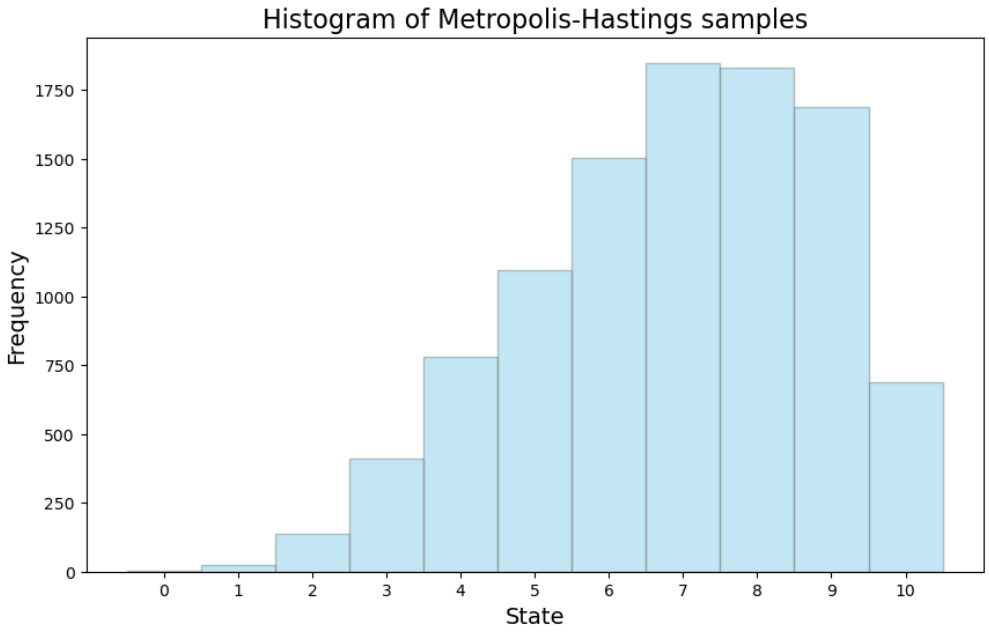


Figure 8: Histogram of Metropolis-Hastings samples

The samples are used to validate the empirical distribution via a χ^2-test with a values of 448.4765 and a p-value of 0 (df=10). This extremely low p-value indicates a significant deviation from the theoretical distribution, suggesting that the Markov chain may not have fully converged to the target distribution.

6.2 Metropolis-Hastings sampling from a constrained joint distribution

Method	Chi-squared	P-value	df
a) Metropolis-Hastings (Direct)	340.8870	0.0000	60
b) Metropolis-Hastings (Coordinate-wise)	118.9953	0.0000	60
c) Gibbs Sampling	43.6866	0.9439	60

Table 8: Goodness-of-fit test for different sampling methods

The Gibbs sampler outperforms both Metropolis-Hastings variants, with a distribution that closely matches the theoretical one. The direct and coordinate-wise methods show significant deviations, especially the direct sampler.

Figure 9 shows heatmaps from the three sampling methods. All three heatmaps closely resemble the theoretical distribution, centered around the high-density region of the state space defined by $i + j \leq 10$.

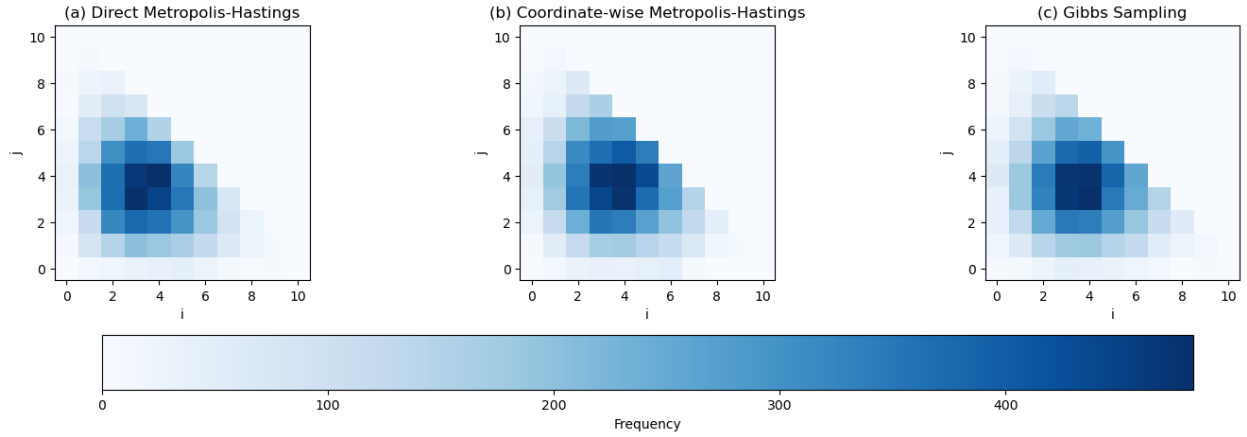


Figure 9: Heatmap of the three samplings

While the visual differences are subtle due to the large number of samples, Gibbs sampling appears slightly smoother and more symmetrical. This suggests faster mixing and more efficient exploration of the state space. The direct Metropolis-Hastings method shows slightly more noise, especially near the boundaries, likely due to more frequent rejections and slower convergence. The coordinate-wise sampler performs better than the direct method but still not as effectively as Gibbs.

6.3 Bayesian statistical problem

a) Generating a pair (θ, ψ) from the prior distribution

A sample was drawn from the joint prior distribution of (Θ, Ψ), where $(\log \Theta, \log \Psi)$ follows a bivariate normal distribution with mean zero and correlation $\rho = 0.5$.This gave us:

```
Theta = 0.5246
Psi = 0.3998
```

b) Generating $X_i = 1, ..., n$ with the values of (θ, ψ)

Using the generated values of θ and ψ, the following $n = 10$ observations were drawn from the distribution $X_i \sim \mathcal{N}(\theta, \psi)$:

$$X = \begin{bmatrix} 0.0912, & 0.9595, & 0.8274, & 1.0019, & -0.4234, \\ -0.4525, & 0.5673, & -0.0197, & 2.4069, & -0.0339 \end{bmatrix}$$

c) Deriving the posterior distribution of (Θ, Ψ) given the sample

Using the simulated data, we calculate:

$$\bar{x} = 0.4925 \quad \text{and} \quad s^2 = 0.6693$$

These values are used in the likelihood term of the posterior.

d) Generating MCMC samples from the posterior distribution of (Θ, Ψ) using the Metropolis Hastings method

To generate samples from the posterior distribution of (Θ, Ψ), we implemented a Metropolis-Hastings algorithm. Proposals were generated via log-normal random walks:

$$\theta' \sim \text{LogNormal}(\log\theta, \sigma^2), \quad \psi' \sim \text{LogNormal}(\log\psi, \sigma^2), \quad \text{with } \sigma = 0.2$$

The posterior density was evaluated (up to proportionality) by combining the likelihood and the prior as derived in part (c). After running the chain for 10,000 iterations with a burn-in of 2,000, the estimated posterior means were:

```
Posterior mean of Theta:  0.4223
Posterior mean of Psi:  0.7048
```

Which reflects the central tendency of the posterior distribution based on the observed data.

e) Posterior sampling for larger sample sizes

Repeated the Metropolis-Hastings sampling procedure for larger datasets using the same values:

```
Mean Theta (n=100):  0.4181
Mean Psi (n=100):  0.4177
Mean Theta (n=1000):  0.5228
Mean Psi (n=1000):  0.3939
```

These results demonstrate that the posterior estimates become more accurate and concentrated around the true parameter values as the number of observations increases.

7 Exercise 7 - Simulated annealing

7.1 Travelling salesman problem

We applied simulated annealing to the Travelling Salesman Problem (TSP) in two scenarios. In part (a), the stations were placed evenly on a unit circle in the plane, and the cost between stations was defined as the Euclidean distance. The algorithm used a cooling schedule of $T_k = \frac{1}{\sqrt{1+k}}$, ran for 5000 iterations, and was initialized with a random permutation of the stations. By fixing the random seed for reproducibility, the best route found had a total length of approximately 6.26. The resulting route formed a nearly perfect circle, as seen in

Figure 10 as expected given the station layout.

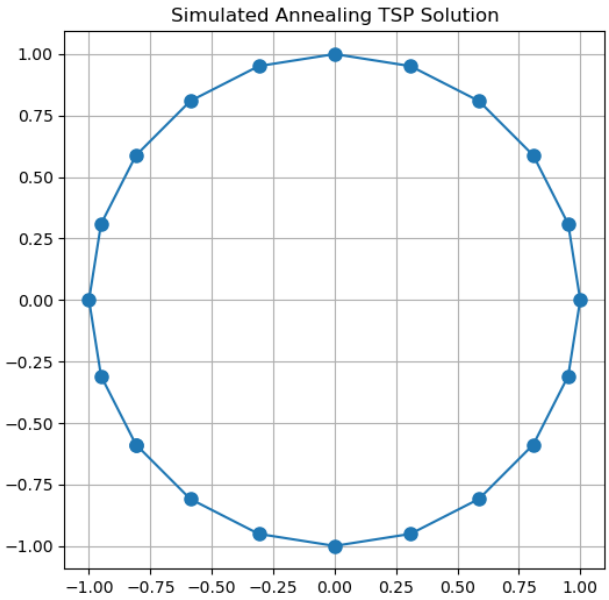


Figure 10: Resulting TSP route for part (a) with 20 stations placed evenly on a circle.

7.2 Fixed cost matrix

In part (b), we applied simulated annealing to the Travelling Salesman Problem using the fixed cost matrix provided. The algorithm used a cooling schedule of $T_k = \frac{1}{\sqrt{1+k}}$ and ran for 5000 iterations, starting from a randomly generated route. To ensure reproducibility, the random seed was fixed. The best route found had a total cost of 1194.0 and is given by:

$$[1, 8, 13, 18, 14, 11, 5, 0, 3, 12, 9, 15, 17, 16, 6, 2, 19, 10, 7, 4],$$

where the route returns to the initial station to complete the cycle.

8 Exercise 8 - Bootstrap

8.1 Bootstrap estimation of sampling distribution for the sample mean

To estimate the probability

$$p = P\left(a < \bar{X} - \mu < b\right),$$

where $\bar{X}$ is the sample mean and μ is the true (unknown) mean, we use the bootstrap method. The idea is to simulate the sampling distribution of $\bar{X} - \mu$ by resampling from the observed data. Since μ is unknown, we approximate the distribution of $\bar{X} - \mu$ by that

of $\bar{X}^* - \bar{X}$, where $\bar{X}^*$ is the mean of a bootstrap sample drawn with replacement from the centered data $X_i - \bar{X}$. We then compute the proportion of bootstrap means that fall within the interval (a, b).
We apply this approach to the sample

$$X = [56, 101, 78, 67, 93, 87, 64, 72, 80, 69],$$

with $a = -5$, $b = 5$, and perform $r = 10{,}000$ bootstrap replicates. For each replicate, we compute the mean $\bar{X}^*$ of a resample drawn from the centred data. We then estimate p as the fraction of replicates where $-5 < \bar{X}^* < 5$.
The result of this simulation gives the estimate

$$\hat{p} = 0.7603.$$

This means that approximately 76% of the bootstrap means fall within 5 units of the sample mean, providing an estimate for the probability that the sample mean lies within the interval $(\mu - 5, \mu + 5)$.

8.2 Bootstrap estimation of variance of sample variance

We are given a sample of size $n = 15$ with the following data:

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

Our goal is to estimate the variance of the sample variance S^2, i.e., $\mathrm{Var}(S^2)$, using the bootstrap method.
The bootstrap procedure is as follows:

1. Generate $r = 10{,}000$ bootstrap samples by sampling with replacement from the original data.
2. For each bootstrap sample, compute the sample variance S^{2*}.
3. Estimate $\mathrm{Var}(S^2)$ as the sample variance of the S^{2*} values.

The sample variance of the original data is:

$$S^2 = 34.3143.$$

The bootstrap estimate of $\mathrm{Var}(S^2)$ based on 10,000 replicates is:

$$\widehat{\mathrm{Var}}(S^2) = 58.6827.$$

This result quantifies the sampling variability of the sample variance estimator in this specific data context.

8.3 Precision comparison of the sample mean and median via bootstrap

We simulate $N = 200$ observations from a Pareto distribution with parameters $\beta = 1$ and $k = 1.05$, and perform bootstrap analysis with $r = 100$ replicates to estimate the variance of both the sample mean and the sample median.

(a) The sample mean is 5.7617 and the sample median is 1.8823.

(b) The bootstrap estimate of the variance of the sample mean is 1.6980.

(c) The bootstrap estimate of the variance of the sample median is 0.0081.

(d) Since the variance of the median is substantially lower than that of the mean, we conclude that the median is more precise in this case.

These results are consistent with the heavy-tailed nature of the Pareto distribution. Outliers tend to inflate the variability of the sample mean much more than that of the median, making the median a more robust and precise estimator of central tendency in such settings.

References

[1] Bo Friis Nielsen. *Stochastic Simulation: Random Number Generation*. http://www.dtu.dk. Lecture slides, slide2am1.pdf, DTU course 02443. 2025.

[2] William H. Press et al. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, 1992.

[3] Python Software Foundation. *random — Generate pseudo-random numbers*. Accessed June 4, 2025. Python Software Foundation. 2024. URL: `https://docs.python.org/3/library/random.html`.