Stochastic Simulation
Professor: Prof. Fabio Nobile
Assistant: Eliott Van Dieren

Week of 2025-10-06 to 2025-10-10

Lab 5 of Thursday 9th October 2025

Exercice 1.

A simulator would like to produce an unbiased estimate of $\mathbb{E}(XY)$, where the two independent random variables X and Y have bounded first moments and can be generated by a stochastic simulation. To this end, they simulate $R \in \mathbb{N}$ replications $X_1, \dots, X_R$ of X and, independently of this, R replications $Y_1, \dots, Y_R$ of Y. They thus have the following two natural estimators for $\mathbb{E}(XY)$ at their disposal:

$$\mathrm{Est}_1 := \left(\frac{1}{R}\sum_{r=1}^{R} X_r\right)\left(\frac{1}{R}\sum_{r=1}^{R} Y_r\right) \quad \text{and} \quad \mathrm{Est}_2 := \frac{1}{R}\sum_{r=1}^{R} X_r Y_r \,.$$

1) Verify that both estimators Est_1 and Est_2 are unbiased.

2) Show that $\mathrm{Var}(\mathrm{Est}_1) < \mathrm{Var}(\mathrm{Est}_2)$.

3) Use the delta method to show that $\sqrt{R}(\mathrm{Est}_1 - \mu_x\mu_y) \xrightarrow{d} N(0, \tau^2)$. Find τ^2 explicitly and derive a $1-\alpha$ asymptotic confidence interval.

Solution

1) Introducing the notation $\bar{X}_R = \frac{1}{R}\sum_{i=1}^{R} X_i$ and $\bar{Y}_R = \frac{1}{R}\sum_{r=1}^{R} Y_j$, it holds by the independence of the random variables X_i and Y_j, and the identical distribution of elements in the sequence $X_1, X_2, \dots, X_R$ and ditto for elements in the sequence $Y_1, Y_2, \dots, Y_R$, that for any $1 \leqslant i, j \leqslant R$,

$$\mathbb{E}[X_i Y_j] = \mathbb{E}[X_i]\mathbb{E}[Y_j] = \mathbb{E}[X]\mathbb{E}[Y] = \mathbb{E}[XY].$$

Consequently,

$$\mathbb{E}\big[\bar{X}_R \bar{Y}_R\big] = \frac{1}{R^2}\sum_{i,j=1}^{R} \mathbb{E}[X_i Y_j] = \frac{1}{R^2}\sum_{i,j=1}^{R} \mathbb{E}[XY] = \mathbb{E}[XY],$$

and

$$\mathbb{E}\left[\frac{1}{R}\sum_{r=1}^{R} X_r Y_r\right] = \frac{1}{R}\sum_{r=1}^{R} \mathbb{E}[XY] = \mathbb{E}[XY].$$

2) Let us recall first that since the sequence $X_1, X_2, \dots$ is iid,

$$\mathrm{Var}(\bar{X}_R) = \mathrm{Cov}(\bar{X}_R, \bar{X}_R) = \frac{1}{R}\,\mathrm{Var}(X),$$

and

$$\mathbb{E}\big[\bar{X}_R\big] = \mathbb{E}[X].$$

(And, similarly, $\mathrm{Var}(\bar{Y}_R) = \frac{1}{R}\mathrm{Var}(Y), \quad \mathbb{E}\big[\bar{Y}_R\big] = \mathbb{E}[Y]$.)

Using that $\bar{X}_R$ and $\bar{Y}_R$ are independent,

$$\begin{aligned}
\mathrm{Var}(\mathrm{Est}_1) &= \mathbb{E}\big[\bar{X}_R^2\bar{Y}_R^2\big] - \big(\mathbb{E}\big[\bar{X}_R\bar{Y}_R\big]\big)^2 \\
&= \mathbb{E}\big[\bar{X}_R^2\big]\mathbb{E}\big[\bar{Y}_R^2\big] - \big(\mathbb{E}\big[\bar{X}_R\bar{Y}_R\big]\big)^2 \\
&= \Big(\mathrm{Var}(\bar{X}_R) + \big(\mathbb{E}\big[\bar{X}_R\big]\big)^2\Big)\Big(\mathrm{Var}(\bar{Y}_R) + \big(\mathbb{E}\big[\bar{Y}_R\big]\big)^2\Big) - \big(\mathbb{E}\big[\bar{X}_R\bar{Y}_R\big]\big)^2 \\
&= \mathrm{Var}(\bar{X}_R)\,\mathrm{Var}(\bar{Y}_R) + \mathrm{Var}(\bar{X}_R)\big(\mathbb{E}\big[\bar{Y}_R\big]\big)^2 + \mathrm{Var}(\bar{Y}_R)\big(\mathbb{E}\big[\bar{X}_R\big]\big)^2 \\
&= \frac{1}{R^2}\mathrm{Var}(X)\,\mathrm{Var}(Y) + \frac{1}{R}\mathrm{Var}(X)(\mathbb{E}[Y])^2 + \frac{1}{R}\mathrm{Var}(Y)(\mathbb{E}[X])^2.
\end{aligned}$$

And for the second estimator we use that since the sequence $X_1Y_1, X_2Y_2, \ldots$ is i.i.d, we may argue as above to derive that

$$\mathrm{Var}(\mathrm{Est}_2) = \frac{1}{R}\mathrm{Var}(XY) = \frac{1}{R}\Big(\mathrm{Var}(X)\,\mathrm{Var}(Y) + \mathrm{Var}(X)(\mathbb{E}[Y])^2 + \mathrm{Var}(Y)(\mathbb{E}[X])^2\Big).$$

So whenever $\mathrm{Var}(X)\,\mathrm{Var}(Y) > 0$,

$$\mathrm{Var}(\mathrm{Est}_2) - \mathrm{Var}(\mathrm{Est}_1) = \mathrm{Var}(X)\,\mathrm{Var}(Y)\frac{R-1}{R^2} > 0.$$

3) We have that,

$$\sqrt{R}\Big(\begin{bmatrix} \bar{X}_R \\ \bar{Y}_R \end{bmatrix} - \begin{bmatrix} \mu_x \\ \mu_y \end{bmatrix}\Big) \to N(0, \Sigma),$$

where $\Sigma = \begin{bmatrix} \mathrm{var}(X) & 0 \\ 0 & \mathrm{var}(Y) \end{bmatrix}$. Then for $h(x,y) = xy$, and after applying the delta method, we get that

$$\sqrt{R}\big(h(\bar{X}, \bar{Y}) - \mu_x\mu_y\big) \to N(0, \tau^2)$$

with $\tau^2 = [\mu_y, \mu_x]\Sigma\begin{bmatrix} \mu_y \\ \mu_x \end{bmatrix} = \mathrm{Var}(X)\mathbb{E}[Y]^2 + \mathrm{Var}(Y)\mathbb{E}[X]^2$. The confidence interval then follows easily.

Exercice 2.

We are interested in estimating the expectation $\mathbb{E}[X]$ of a random variable X using several sampling methods. We will work with three different distributions of X.

- $X \sim \mathrm{Pareto}(x_m = 1, \gamma = 3.1)$ (i.e. with PDF $p(y) = \mathbb{1}_{y>x_m} x_m^\gamma \gamma y^{-(\gamma+1)}$), $\mathbb{E}[X] = \frac{\gamma x_m}{\gamma - 1}$ for $\gamma > 1$, else $\mathbb{E}[X] = \infty$ and $\mathrm{Var}(X) = \frac{x_m^2\gamma}{(\gamma-1)^2(\gamma-2)}$ for $\gamma > 2$, else $\mathrm{Var}(X) = \infty$. Lastly, the normalised third absolute moment is $\beta = \mathbb{E}[|X - \mathbb{E}[X]|^3]/\,\mathrm{Var}(X)^{3/2}$ for $\gamma > 3$ For $\gamma = 3.1, x_m = 1$, we have $\mathbb{E}[X] = \frac{3.1}{2.1}$, $\mathrm{Var}(X) = \frac{3.1}{2.1^2 \cdot 1.1}$ and $\beta \approx 18.9$
- $X \sim \mathrm{Lognormal}(\mu = 0, \sigma = 1)$, $\mathbb{E}[X] = \exp\Big(\mu + \frac{\sigma^2}{2}\Big)$, $\mathrm{Var}(X) = (\exp(\sigma^2) - 1)\exp(2\mu + \sigma^2)$, and

$$\beta = \frac{4\,\mathrm{Erf}(\frac{\sigma}{2\sqrt{2}}) - 3e^{\sigma^2}\,\mathrm{Erf}(\frac{3\sigma}{2\sqrt{2}}) + e^{3\sigma^2}\,\mathrm{Erf}(\frac{5\sigma}{2\sqrt{2}})}{(e^{\sigma^2} - 1)^{3/2}}.$$

For $\mu = 0, \sigma = 1$, we have $\mathbb{E}[X] = \sqrt{e}$, $\mathrm{Var}(X) = e(e-1)$ and $\beta \approx 6.35$.

- $X \sim U([-1,1])$, $\mathbb{E}[X] = 0$, $\mathrm{Var}(X) = \frac{1}{3}$, and $\beta = 3\sqrt{3}/4$.

We will start with a Crude Monte Carlo estimator

$$\overline{I}_N := \frac{1}{N}\sum_{k=1}^{N} X_k,$$

for $X_1, X_2, \dots, X_N$ i.i.d. samples of X.

1) By *a priori* analysis, determine $N(\alpha, \epsilon)$, $\alpha = 0.05$ such that

$$\mathbb{P}(|\overline{I}_N - \mathbb{E}[X]| > \epsilon) < \alpha$$

using Chebycheff's inequality, the Berry-Esseen Theorem and the CLT

$$\frac{\overline{I}_N - \mathbb{E}[X]}{\sqrt{\mathrm{Var}(X)/N}} \sim N(0,1).$$

   a) Provide the analytical expressions for $N(\alpha, \epsilon)$ using the three different techniques. Furthermore, compute $N(\alpha, \epsilon)$ for $\epsilon = 0.1$ and $\alpha = 0.05$ for each of the three distributions above, and compare the obtained values for the three techniques. What is the order of growth of $N(\alpha = 0.05, \epsilon)$ w.r.t ϵ?

   b) Generate M independent MC approximations $\overline{I}_N^j$, $j = 1, 2, \dots, M$ with $M = 1000$. Compute empirically $\mathbb{P}(|\overline{I}_N - \mathbb{E}[X]| > \epsilon)$, and compare with the target $\alpha = 0.05$.

2) Algorithm 1 proposes a sequential Monte Carlo method to compute the expectation $\mathbb{E}[X]$ of a random variable X, where the sample size is doubled at each iteration until the estimated $1-\alpha$ confidence interval based on a central limit theorem approximation is smaller than a prescribed tolerance ϵ. The algorithm then outputs the final sample size $N(\epsilon, \alpha)$, as well as the estimated value $\bar{X}_N$.

   Algorithm 1 can be particularly sensitive to the choice of initial sample size N_0, and as such, we would like to assess the robustness of such an algorithm in estimating $\mathbb{E}[X]$ for different distributions of X. For some values of N_0 ranging between 10 and 50, consider $\alpha = 10^{-1.5}$ and $\epsilon = 1/10$.

   a) Repeat the simulation $K = 20\alpha^{-1}$ times and record the sample sizes $\{N^{(i)}\}_{i=1}^K$ as well as the computed sample means $\{\bar{X}_{\epsilon,\alpha}^{(i)}\}_{i=1}^K$ returned by the algorithm for each run $i = 1, ..., K$.

   b) Estimate the probability of failure $\overline{p}$ of the algorithm :

$$\overline{p}_K(N_0, \epsilon, \alpha) = \frac{1}{K}\sum_{i=1}^{K} \mathbb{1}_{|\bar{X}_{\epsilon,\alpha}^{(i)} - \mathbb{E}[X]| > \epsilon}.$$

   Then check whether $\overline{p}_K(N_0, \epsilon, \alpha) \leqslant \alpha$ holds.

   c) Repeat your experiment for different values of ϵ and α. Discuss your results.

Hint. You may generate Pareto(x_m, α) r.v. by inversion.

Algorithm 1: Sample Variance Based SMC

Input: N_0, distribution λ, accuracy $\epsilon > 0$, confidence $1 - \alpha > 0$.
Output: $\overline{X}_{\epsilon,\alpha}$ (i.e, approximation of $\mathbb{E}[X]$ with $X \sim \lambda$), N.

Set $k = 0$, generate N_k i.i.d. replica $\{X_i\}_{i=1}^{N_k}$ of $X \sim \lambda$ and

$$\bar{X}_{N_k} = \frac{1}{N_k} \sum_{i=1}^{N_k} X_i, \tag{2.1}$$

$$\overline{\sigma}^2_{N_k} := \frac{1}{N_k - 1} \sum_{i=1}^{N_k} (X_i - \overline{X}_{N_k})^2. \tag{2.2}$$

while $\bar{\sigma}_{N_k} C_{1-\alpha/2} / \sqrt{N_k} > \epsilon$ **do**
Set $k = k + 1$ and $N_k = 2N_{k-1}$.
Generate a new batch of N_k i.i.d. replicas $\{X_i\}_{i=1}^{N_k}$ of $X \sim \lambda$.
Compute the sample variance $\overline{\sigma}^2_{N_k}$ by (2).
end while
Set $N = N_k$, generate i.i.d. samples $\{X_i\}_{i=1}^{N}$ of λ and compute the output sample mean $\overline{X}_{\epsilon,\alpha}$.

Algorithm 2: One-at-a-time Sample Variance Based SMC

Input: N_0, distribution λ, accuracy $\epsilon > 0$, confidence $1 - \alpha > 0$.
Output: $\overline{X}_{\epsilon,\alpha}$ (i.e, approximation of $\mathbb{E}[X]$ with $X \sim \lambda$), N.

Set $k = 0$, generate N_k i.i.d. samples $\{X_i\}_{i=1}^{N_k}$ of λ and compute the sample variance

$$\overline{\sigma}^2_{N_k} := \frac{1}{N_k - 1} \sum_{i=1}^{N_k} (X_i - \overline{X}_{N_k})^2. \tag{2.3}$$

while $\bar{\sigma}_{N_k} C_{1-\alpha/2} / \sqrt{N_k} > \epsilon$ **do**
Set $k = k + 1$ and $N_k = N_{k-1} + 1$.
Generate a new i.i.d. sample $X^{(N_k+1)}$ of λ.
Compute

$$\bar{\mu}_{N_k+1} = \frac{N_k}{N_k + 1} \bar{\mu} + \frac{1}{N_k + 1} X^{(N_k+1)} \tag{2.4}$$

$$\bar{\sigma}^2_{N_k+1} = \frac{N_k - 1}{N_k} \sigma^2_{N_k} + \frac{1}{N_k + 1} (X^{(N_k+1)} - \bar{\mu}_{N_k})^2 \tag{2.5}$$

end while
Set $N = N_k$, generate i.i.d. samples $\{X_i\}_{i=1}^{N}$ of λ and compute the output sample mean $\overline{X}_{\epsilon,\alpha}$.

3) One drawback of Algorithm 1 is that it requires to generate a batch of samples at each iteration, which may be computationally expensive. Compare Algorithm 1 with the sequential Monte Carlo method in Algorithm 2, where one realization is added at a time. Do you observe any difference in the performance of the two algorithms ? Is one more robust numerically than the other?

4) Repeat the points above by replacing the CLT stopping criterion in Algorithm 1 and Algorithm 2 by the Chebycheff and Berry-Esseen based stopping criteria.

Solution

1) For determining the respective lower bounds for $N(\alpha, \epsilon)$, we have:

- Chebycheff's inequality states that for a random variable Y with finite variance σ^2, we have
$$\mathbb{P}(|Y - \mathbb{E}[Y]| \geqslant k) \leqslant \frac{\sigma^2}{k^2}.$$
Applying this to our estimator $\overline{I}_N$ with $Y = \overline{I}_N$, $\mathbb{E}[Y] = \mathbb{E}[X]$, and $\sigma^2 = \mathrm{Var}(X)/N$, we get
$$\mathbb{P}(|\overline{I}_N - \mathbb{E}[X]| \geqslant \epsilon) \leqslant \frac{\mathrm{Var}(X)}{N\epsilon^2}.$$
To ensure this probability is less than α, we set
$$\frac{\mathrm{Var}(X)}{N\epsilon^2} < \alpha \implies N > \frac{\mathrm{Var}(X)}{\alpha\epsilon^2}.$$
Thus, the Chebycheff bound is
$$N_{\mathrm{Cheb}}(\alpha, \epsilon) = \frac{\mathrm{Var}(X)}{\alpha\epsilon^2}.$$

- The Central Limit Theorem (CLT) states that for large N, the distribution of the standardized sample mean approaches a normal distribution. Specifically,
$$\frac{\overline{I}_N - \mathbb{E}[X]}{\sqrt{\mathrm{Var}(X)/N}} \sim N(0, 1).$$
Using the properties of the normal distribution, we want to find N such that
$$\mathbb{P}\big(|\overline{I}_N - \mathbb{E}[X]| \geqslant \epsilon\big) < \alpha.$$
This is equivalent to
$$\mathbb{P}\left(\left|\frac{\overline{I}_N - \mathbb{E}[X]}{\sqrt{\mathrm{Var}(X)/N}}\right| \geqslant \frac{\epsilon\sqrt{N}}{\sqrt{\mathrm{Var}(X)}}\right) < \alpha.$$
Using the standard normal distribution, we find the critical value $z_{\alpha/2}$ such that
$$\mathbb{P}\big(|Z| \geqslant z_{\alpha/2}\big) = \alpha,$$
where $Z \sim N(0, 1)$. Thus, we set
$$\frac{\epsilon\sqrt{N}}{\sqrt{\mathrm{Var}(X)}} = z_{\alpha/2} \implies N = \frac{z_{\alpha/2}^2 \,\mathrm{Var}(X)}{\epsilon^2}.$$
Thus, the CLT bound is
$$N_{\mathrm{CLT}}(\alpha, \epsilon) = \frac{z_{\alpha/2}^2 \,\mathrm{Var}(X)}{\epsilon^2}.$$

- The Berry-Esseen theorem provides a bound on the convergence rate of the distribution of the standardized sample mean to the normal distribution. It states that for a random variable X with finite third absolute moment $\beta = \mathbb{E}[|X - \mathbb{E}[X]|^3]/\operatorname{Var}(X)^{3/2}$, we have

$$\sup_x \left| \mathbb{P}\left(\frac{\sqrt{N}(\bar{I}_N - \mathbb{E}[X])}{\sigma} \leqslant x \right) - \Phi(x) \right| \leqslant \frac{C\beta}{\sqrt{N}},$$

where Φ is the CDF of the standard normal distribution and C is a constant (often taken as 0.4748). To ensure that

$$\mathbb{P}\Big(|\bar{I}_N - \mathbb{E}[X]| \geqslant \epsilon \Big) < \alpha,$$

we use the BE theorem to write

$$\mathbb{P}\left(\left| \frac{\sqrt{N}(\bar{I}_N - \mathbb{E}[X])}{\sigma} \right| \geqslant \frac{\epsilon\sqrt{N}}{\sigma} \right) \leqslant 2 - 2\Phi\left(\frac{\epsilon\sqrt{N}}{\sigma} \right) + 2\frac{C\beta}{\sqrt{N}}.$$

Using the standard normal distribution, we find the critical value N such that

$$1 - \Phi\left(\frac{\epsilon\sqrt{N}}{\sigma} \right) + \frac{C\beta}{\sqrt{N}} = \alpha/2.$$

Here are the results of the experiments for $\epsilon = 0.1$ and $\alpha = 0.05$:

Distribution	μ	Var	σ	β	$N_{\text{Chebyshev}}$	N_{CLT}	N_{BE}
Pareto (x_m=1, γ=3.1)	1.4762	0.6390	0.7994	18.7900	1279	246	127349
Lognormal (μ=0, σ=1)	1.6487	4.6708	2.1612	6.3500	9342	1795	14545
Uniform [-1,1]	0.0000	0.3333	0.5774	1.2990	667	129	610

Table 1: Summary statistics and required sample sizes for different distributions.

Distribution / Method	N	Empirical Probability	Target α
Uniform [-1,1] — Chebyshev	667	0.0000	0.0500
Uniform [-1,1] — CLT	129	0.0507	0.0500
Uniform [-1,1] — BE	610	0.0001	0.0500
Pareto (x_m=1, γ=3.1) — Chebyshev	1279	0.0006	0.0500
Pareto (x_m=1, γ=3.1) — CLT	246	0.0416	0.0500
Pareto (x_m=1, γ=3.1) — BE	127349	0.0000	0.0500
Lognormal (μ=0, σ=1) — Chebyshev	9342	0.0000	0.0500
Lognormal (μ=0, σ=1) — CLT	1795	0.0506	0.0500
Lognormal (μ=0, σ=1) — BE	14545	0.0000	0.0500

Table 2: Empirical probabilities versus target α for different distributions and methods.

Python code:

```
import numpy as np
import scipy.stats as st
import matplotlib.pyplot as plt
import scipy.optimize as so
```

```
np.random.seed(42)

alpha = 0.05
eps = 0.1


# Distribution definitions

### U[-1,1]
a = -1
b = 1
mu_unif = (a+b)/2
var_unif = (b-a)**2/(12)
sigma_unif = np.sqrt(var_unif)
beta_unif = 3*np.sqrt(3)/4  # Provided in problem, tested numerically using MC.

### Pareto (xm=1, gamma=3.1)
xm = 1
gamma = 3.1
mu_pareto = gamma * xm / (gamma - 1)
var_pareto = xm**2 * gamma / ((gamma - 1)**2 * (gamma - 2))
sigma_pareto = np.sqrt(var_pareto)
beta_pareto = 18.79 # Provided in problem, tested numerically using MC.

### Lognormal (mu_l=0,sigma_l=1)
mu_l = 0
sigma_l = 1
mu_logn = np.exp(mu_l + sigma_l**2/2)
var_logn = ( np.exp(sigma_l**2) - 1 ) * np.exp(2*mu_l + sigma_l**2)
sigma_logn = np.sqrt(var_logn)
beta_logn = 6.35 # Provided in problem, tested numerically using MC, matches theoretical
↪  value.

# Functions for N bounds
# Chebyshev: N >= var / (alpha * eps^2)
def N_chebyshev(var, alpha, eps):
    return np.ceil(var / (alpha * eps**2))

# CLT normal approximation: N >= (z_{1-alpha/2}^2 * var) / eps^2
def N_clt(var, alpha, eps):
    z = st.norm.ppf(1 - alpha/2)
    return np.ceil((z**2 * var) / (eps**2))

def N_BE(sigma, beta, alpha, eps):
    K_BE = 0.4748
    # We need to solve the equation (see slides):
    # 1 - alpha/2 - Phi(sqrt(N)*eps/sigma) + K_BE * beta / sqrt(N) = 0
    f = lambda N: 1 - alpha/2 - st.norm.cdf(np.sqrt(N)*eps/sigma) + K_BE * beta / np.sqrt(N)
    # Use a root-finding method to find N
    N_solution = so.bisect(f, 10, 1e9)
    return int(np.ceil(N_solution))

vals = {
    'Pareto (xm=1, gamma=3.1)': (mu_pareto, var_pareto, beta_pareto),
    'Lognormal (mu=0,sigma=1)': (mu_logn, var_logn, beta_logn),
    'Uniform [-1,1]': (mu_unif, var_unif, beta_unif)
}

# compute for each distribution
results = {}
for name, (mu, var, beta) in vals.items():
    sigma = np.sqrt(var)
```

```
    n_cheb = N_chebyshev(var, alpha, eps)
    n_clt = N_clt(var, alpha, eps)
    n_be = N_BE(sigma, beta, alpha, eps)
    results[name] = (mu, var, sigma, beta, n_cheb, n_clt, n_be)

print(f"{'Distribution':<30} {'mu':<10} {'var':<15} {'sigma':<15} {'beta':<10}
 ↪ {'N_Chebyshev':<15} {'N_CLT':<15} {'N_BE':<10}")
for name in results.keys():
    mu, var, sigma, beta, n_cheb, n_clt, n_be = results[name]
    print(f"{name:<30} {mu:<10.4f} {var:<15.4f} {sigma:<15.4f} {beta:<10.4f} {n_cheb:<15}
     ↪ {n_clt:<15} {n_be}")

print("-" * 80)
# -- Monte Carlo simulation --
M = 10000  # Number of independent MC approximations
for name in ["Uniform [-1,1]", "Pareto (xm=1, gamma=3.1)", "Lognormal (mu=0,sigma=1)"]:
    mu, var, sigma, beta, n_cheb, n_clt, n_be = results[name]

    # sampler using scipy distributions
    if name == 'Uniform [-1,1]':
       sampler = lambda size: st.uniform.rvs(loc=a, scale=b-a, size=size)
    elif name == 'Pareto (xm=1, gamma=3.1)':
       sampler = lambda size: st.pareto.rvs(b=gamma, scale=xm, size=size)
    elif name == 'Lognormal (mu=0,sigma=1)':
        sampler = lambda size: st.lognorm.rvs(s=sigma_l, size=size)
    else:
       raise ValueError("Unknown distribution")

    for technique in ['Chebyshev', 'CLT', 'BE']:
        N_key = f'N_{technique}'
        N = int(results[name][{'Chebyshev':4, 'CLT':5, 'BE':6}[technique]])
        samples = sampler((M, N))
        I_estimates = samples.mean(axis=1)
        prob_empirical = np.mean(np.abs(I_estimates - mu) > eps)
        print(f"{name} using {technique} N={N}: Empirical probability = {prob_empirical:.4f},
         ↪ Target alpha = {alpha:.4f}")

    print("-" * 80)
```

2) A possible `Python` implementation for this part of the exercise is given below. Notice that since $\mathbb{E}[\bar{p}_N] \sim \alpha$, $\mathrm{Var}[\bar{p}_N] \sim \alpha(1-\alpha)/N \sim \alpha/N$, the choice $N = 20\alpha^{-1}$ gives a 95% confidence interval $\alpha \pm \frac{\sqrt{\alpha}}{\sqrt{N}} C_{1-\alpha/2} \sim \alpha\left(1 \pm \frac{C_{1-\alpha/2}}{\sqrt{20}}\right)$. The results of the numerical tests are plotted in Figure 1 for Algorithm 1 and in Figure 2 for Algorithm 2. For the estimate of $\overline{p}_N$ we have included errorbars of magnitude $\sigma_N := 2\sqrt{\overline{p}_N(1-\overline{p}_N)/N}$ for all three distributions. Random variables from the Pareto distribution are sampled using inversion sampling with $F^{-1}(x) = (1-x)^{-1/3.1}$. For Algorithm 1, we observe that only the heavy-tailed Pareto$(x_m = 1, \alpha = 3.1)$ distribution fails to approximate its mean with the sought confidence as $p_N - 2\sigma_N > \delta$ for all N_0-values (and the failure is more pronounced for lower N_0-values), while for the other two distributions, one always has $p_N + 2\sigma_N < \delta$. When using Algorithm 2, the Pareto distribution once again fails to approximate the mean for all N_0 but the failure rate of the algorithm is higher for a given N_0 than Algorithm 1. We also observe that the algorithm fails to approximate the mean for all N_0 for the other distributions as well.

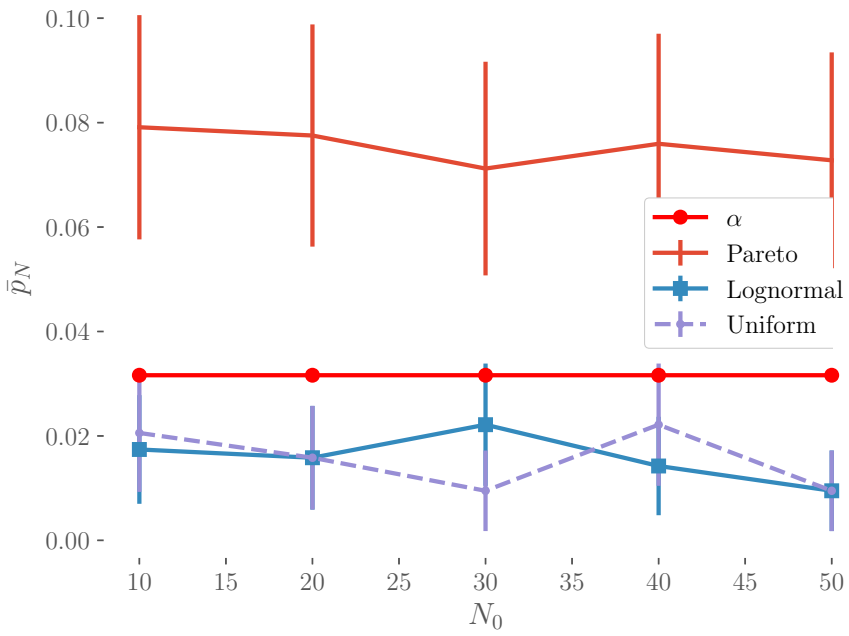

Figure 1: Numerical estimates of the probability of failure for (the SMC) Algorithm 1 when varying the initial number of samples N_0 along with error bars

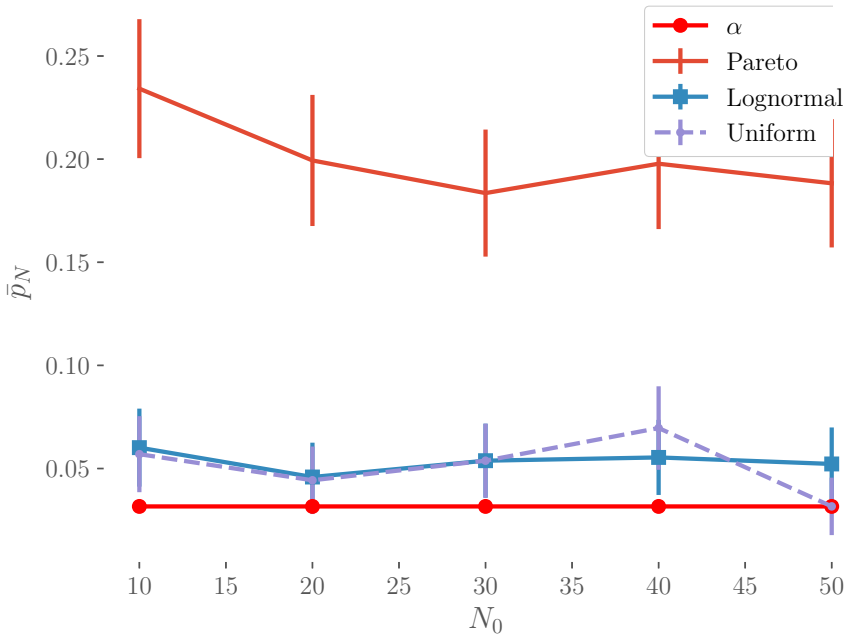

Figure 2: Numerical estimates of the probability of failure for (the SMC) Algorithm 2 when varying the initial number of samples N_0 along with error bars

Python code:

```
import numpy as np
import scipy.stats as st
import matplotlib.pyplot as plt
from matplotlib import rc
#graphical parameters
################################################################################
plt.style.use('ggplot')
rc('font',**{'family':'serif','serif':['Computer Modern Roman'],
     'size' : '12'})
rc('text', usetex=True)
rc('lines', linewidth=2)
plt.rcParams['axes.facecolor']='w'
################################################################################

def Algo1(N0, lamda, alpha, eps):
    M = N0
    X = lamda(M)
    sig = np.std(X)

    while 2*(1.-st.norm.cdf(np.sqrt(M)*eps / sig)) > alpha:

            M = 2 * M
            X = lamda(M)
            sig = np.std(X)

    barX = np.mean(lamda(M))

    return barX, M

def Algo2(N0, lamda, alpha, eps):
    M = N0
    X = lamda(M)
    barX = np.mean(X)
    sig = np.std(X)

    while 2*(1.-st.norm.cdf(np.sqrt(M)*eps / sig)) > alpha:
        Xnew = lamda(1)
        barX = M*barX/(M+1) + 1*Xnew/(M+1)
        sig = np.sqrt(  (M-1)*sig**2/M + (Xnew-barX)**2/(M+1)  )
        M = M +1

    barX = np.mean(lamda(M))
    return barX, M

# Variable introduction
alpha = 10 ** (-1.5)
epsilon = 1./10
N0 = np.arange(10, 51, 10)

pNPareto = np.zeros(len(N0))
randPareto = lambda n: (1 - st.uniform.rvs(size = (n,))) ** (-1/ 3.1)
meanPareto = 3.1/2.1

pNLognormal = np.zeros(len(N0))
randLognorm = lambda n: np.exp(st.norm.rvs(size = (n,) ))
meanLognormal = np.exp(1./2)

pNUniform = np.zeros(len(N0))
randUni = lambda n: (2 * st.uniform.rvs(size = (n,)) - 1.)
meanUniform = 0

# Number of times to run each simulation for a given N0
```

```
N = int(20/alpha)

# Algorithm 1
for i in range(len(N0)):
    for j in range(N):
        print('Running simulation ',j,'/',N, 'for N0 = ', N0[i], ' for Algorithm 1')
        barX,M = Algo1(N0[i], randPareto, alpha, epsilon)
        pNPareto[i] = pNPareto[i] + float(np.abs(barX - meanPareto) > epsilon) / N

        barX,M = Algo1(N0[i], randLognorm, alpha, epsilon)
        pNLognormal[i] = pNLognormal[i] + float(np.abs(barX - meanLognormal) > epsilon) / N

        barX,M = Algo1(N0[i], randUni, alpha, epsilon)
        pNUniform[i] = pNUniform[i] + float(np.abs(barX - meanUniform) > epsilon) / N

plt.figure()
eBars = 2 * np.sqrt(pNPareto * (1 - pNPareto) / N)
plt.errorbar(N0, pNPareto, yerr = eBars, label = 'Pareto')

eBars = 2 * np.sqrt(pNLognormal * (1 - pNLognormal) / N)
plt.errorbar(N0, pNLognormal, yerr = eBars, fmt = '-s', label = 'Lognormal')

eBars = 2 * np.sqrt(pNUniform * (1 - pNUniform) / N)
plt.errorbar(N0, pNUniform, eBars, fmt = '--.', label = 'Uniform')

plt.plot(N0, alpha * np.ones(len(N0)), 'r-o', label = r'$\alpha$')
plt.xlabel(r'$N_0$')
plt.ylabel(r'$\bar{p}_N$')
plt.legend()
plt.show()

# Algorithm 2
for i in range(len(N0)):
    for j in range(N):
        print('Running simulation ',j,'/',N, 'for N0 = ', N0[i], ' for Algorithm 2')
        barX,M = Algo2(N0[i], randPareto, alpha, epsilon)
        pNPareto[i] = pNPareto[i] + float(np.abs(barX - meanPareto) > epsilon) / N

        barX,M = Algo2(N0[i], randLognorm, alpha, epsilon)
        pNLognormal[i] = pNLognormal[i] + float(np.abs(barX - meanLognormal) > epsilon) / N

        barX,M = Algo2(N0[i], randUni, alpha, epsilon)
        pNUniform[i] = pNUniform[i] + float(np.abs(barX - meanUniform) > epsilon) / N

plt.figure()
eBars = 2 * np.sqrt(pNPareto * (1 - pNPareto) / N)
plt.errorbar(N0, pNPareto, yerr = eBars, label = 'Pareto')

eBars = 2 * np.sqrt(pNLognormal * (1 - pNLognormal) / N)
plt.errorbar(N0, pNLognormal, yerr = eBars, fmt = '-s', label = 'Lognormal')

eBars = 2 * np.sqrt(pNUniform * (1 - pNUniform) / N)
plt.errorbar(N0, pNUniform, eBars, fmt = '--.', label = 'Uniform')

plt.plot(N0, alpha * np.ones(len(N0)), 'r-o', label = r'$\alpha$')
plt.xlabel(r'$N_0$')
plt.ylabel(r'$\bar{p}_N$')
plt.legend()
plt.show()
```

Exercice 3.

Consider the problem of pricing a Barrier option with maturity $T > 0$ based on the stock price S, which is given as the solution to the stochastic differential equation

$$dS = rS\,dt + \sigma S\,dW\,, \quad S(0) = S_0\,,$$

where W denotes a standard one-dimensional Wiener process. One can show that $S_t = S_0 e^{X_t}$, where $X_t = (r - \sigma^2/2)t + \sigma W_t$ with W being a standard Wiener process. It follows that S_t has a log-normal distribution for any $t > 0$. For $m \in \mathbb{N}$, let $t_i = i\Delta t$ with $\Delta t = T/m$ denote the discrete observation times of the stock price S (e.g. daily at market closure). The payoff of a call option subject to a lower barrier is then given by

$$\Psi(S_{t_0}, S_{t_1}, \dots, S_T) = (S_T - K)_+ \mathbb{I}_{\left\{B \leqslant \min_{i=0,\dots,m}(S_{t_i})\right\}}\,,$$

where $B < S_0$ denotes the Barrier and $K \leqslant S_0$ the strike price. Here, $z_+ = (|z| + z)/2$ denotes the positive part of z. Estimate the expected payoff $\mathbb{E}(\Psi(S_{t_0}, S_{t_1}, \dots, S_T))$ with antithetic variables, using the process parameters $m = 1000$, $r = 0.5$, $\sigma = 0.3$, $T = 2$, $S_0 = 5$, and $K = 10$. Specifically, investigate the variance reduction effect for different barrier values B.

Solution

Instead of using Ψ as given in the exercise, we write is as a function of m increments of the m independent normally distributed random variables $\Delta W_i = W_{t_i} - W_{t_{i-1}}$, $i = 1, \dots, m$:

$$\Psi(S_{t_0}, \dots, S_{t_m}) = \tilde{\Psi}(\Delta W_1, \Delta W_2, \dots, \Delta W_m)\,.$$

This is of course possible, since S_0 is deterministic and because the trajectory of the process S observed at discrete times $t_1, \dots, t_m$ is a function of independent increments ΔW_i of the Wiener process, each following a $\mathcal{N}(0, t_i - t_{i-1})$ distribution. Then we can apply the basic algorithm for antithetic variables as described in the lecture notes

$$\hat{\mu}_{\mathrm{AV}} = \frac{1}{N/2} \sum_{i=1}^{N/2} \frac{\tilde{\Psi}(\boldsymbol{\Delta W}^{(i)}) + \tilde{\Psi}(-\boldsymbol{\Delta W}^{(i)})}{2}\,,$$

where each $\boldsymbol{\Delta W}^{(i)} \sim \mathcal{N}(0, \Delta t I_m)$. An exemplary `Python` implementation of the function $\tilde{\Psi}$ is the following:

```
import numpy as np
#----------------------------------------
#Defines the gen psi fucntion
def genPsi(xi,t,r,sigma,K,B,S0):
    dt=np.diff(t)
    W=np.append([0],np.cumsum(np.sqrt(dt)*xi))
    S=S0*np.exp((r-sigma**2/2)*t+sigma*W);
    rval = (abs(S[-1]-K)+ (S[-1]-K))/2*(B<=min(S));
    return rval
#----------------------------------------
```

Using this implementation with $B = 4$, $N = 100000$, and 95% confidence interval, one obtains for example:

```
B =   4
MC confidence interval  (alpha= 0.05  N=  100000)  mean=  4.168  +-  0.033
N*variance MC estimator:  29.176290337018568
AV confidence interval  (alpha=0.05  N=    50000)  mean=  4.160  +-  0.022
N*variance AV estimator:  13.075383003446746
```

Python code:

```
import numpy as np
import scipy.stats as st

np.random.seed(42)

m = 1000
r = 0.5
sigma = 0.3
T = 2
S0 = 5
K = 2*S0
Blist = np.arange(0,S0)
nB = np.size(Blist)
dt = T/m
t=np.arange(0,m)*dt
N = np.int64(1E5)
M = np.int64(N/2)
#Defines for the confidence intervals
alpha=0.05

cval = st.norm.ppf(1-alpha/2);

#Defines the gen psi fucntion
def genPsi(xi,t,r,sigma,K,B,S0):
    dt=np.diff(t)
    W=np.append([0],np.cumsum(np.sqrt(dt)*xi))
    S=S0*np.exp((r-sigma**2/2)*t+sigma*W);
    rval = (abs(S[-1]-K)+ (S[-1]-K))/2*(B<=min(S));
    return rval

#Begins the for loop
for j in range(nB):
    B=Blist[j]
    print('B =  ',B)
    psi = np.zeros(N)
    for i in range(N):
        xi = np.random.standard_normal(m-1)
        psi[i] = genPsi(xi,t,r,sigma,K,B,S0)
    print("MC confidence interval (alpha= ",alpha, ' N= ',N,') mean= ',
          np.mean(psi), " +- ", cval*np.std(psi)/np.sqrt(N))
    NvarMC = np.var(psi)
    print("N*variance MC estimator: " ,NvarMC)
    psiAV = np.zeros(M)
    for i in range(M):
        xi = np.random.standard_normal(m-1);
        psiAV[i] = (genPsi(xi,t,r,sigma,K,B,S0) + genPsi(-xi,t,r,sigma,K,B,S0))/2
    print("AV confidence interval (alpha= ",alpha, ' N/2= ',M,') mean= ',
          np.mean(psiAV), " +- ", cval*np.std(psiAV)/np.sqrt(M))
    NvarAV = 2*np.var(psiAV)
    print('N*variance AV estimator: ',NvarAV)
```