Chapter 1: Introduction & random variables simulation

UE Computational statistics

Pierre Pudlo
Aix-Marseille Université / Faculté des Sciences

About this chapter

Objectives:

- Understand the fundamental principles of simulation in statistics.
- Learn basic methods for simulating random variables.

Content:

- Role of simulation in computational statistics.
- Simulation of uniform, normal, and other classical distributions (inverse transform method, rejection method, Box-Muller method).
- Introduction to Gaussian vectors: simulation via Cholesky decomposition.

1. Introduction

Monte Carlo methods

Definition

Describe methods relying on **random sampling to solve mathematical problems**.

Highlights the use of **randomness and probability as fundamental components**

- refers to the famous casino district in Monaco
- first used in the 1940s by mathematicians working on nuclear physics simulations, particularly in the Manhattan Project

Why simulate?

1. **Approximating theoretical results**

- Intractable integrals, complex distributions.
- E.g. Get the distribution of an estimator using Monte Carlo simulations

2. **Validation of statistical models**

- Evaluate the performance of models/methods under controlled conditions
- E.g. testing how well a regression model performs when certain assumptions are violated (such as normality…)

3. **Hypothesis testing, risk analysis and decision-making**

- Compute the type-II risk of a test, the p-values of permutation tests, etc.
- Evaluation the risk of an investment portfolio using simulated price paths

4. **Bootstrap and resampling methods**

- Get confidence intervals, standard errors, etc. without relying on asymptotic results

5. **Bayesian inference**

- Approximate the posterior distribution, rarely available in closed form
- Get credible intervals, etc.

6. **Machine learning**

- Uncertainty quantification
- Reinforcement learning with, e.g., Markov Decision Processes (AlhaGo: Monte Carlo Tree Search, etc.)
- Generative models (E.g., Generative Adversarial Networks, Variational Autoencoders, etc.)
- Stochastic optimization (E.g. Stochastic Gradient Descent, Bayesian optimization)
- …

Example: Estimating the probability of a customer churn

Aim

Estimate the probability that a customer will churn with the next month.

Include uncertainty due to variability in the coefficients.

1. Fit a logistic regression model to the data

$$\text{churn}_i \sim \mathscr{B}(p_i) \quad \text{with } \log\left(\frac{p_i}{1-p_i}\right) = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}$$

2. Assume that the inference error is normally distributed and **simulate many sets** of coefficients $(\beta_0, \beta_1, \beta_2, \ldots)$ from

$$\beta_i^{\mathrm{MC}} \sim \mathcal{N}(\widehat{\beta}_i, \widehat{\sigma}_i^2)$$

where $\widehat{\sigma}_i$ is the standard error of the estimate $\widehat{\beta}_i$

3. For a specific customer with covariates $(x_1^\star, \ldots, x_p^\star)$, calculate the many churn probabilities given by the sets of coefficients

$$p^{\star,\mathrm{MC}} = \frac{1}{1 + \exp\left(-\beta_0^{\mathrm{MC}} - \beta_1^{\mathrm{MC}} x_1^\star - \ldots - \beta_p^{\mathrm{MC}} x_p^\star\right)}$$

4. Analyze the distribution of the churn probabilities

Python code

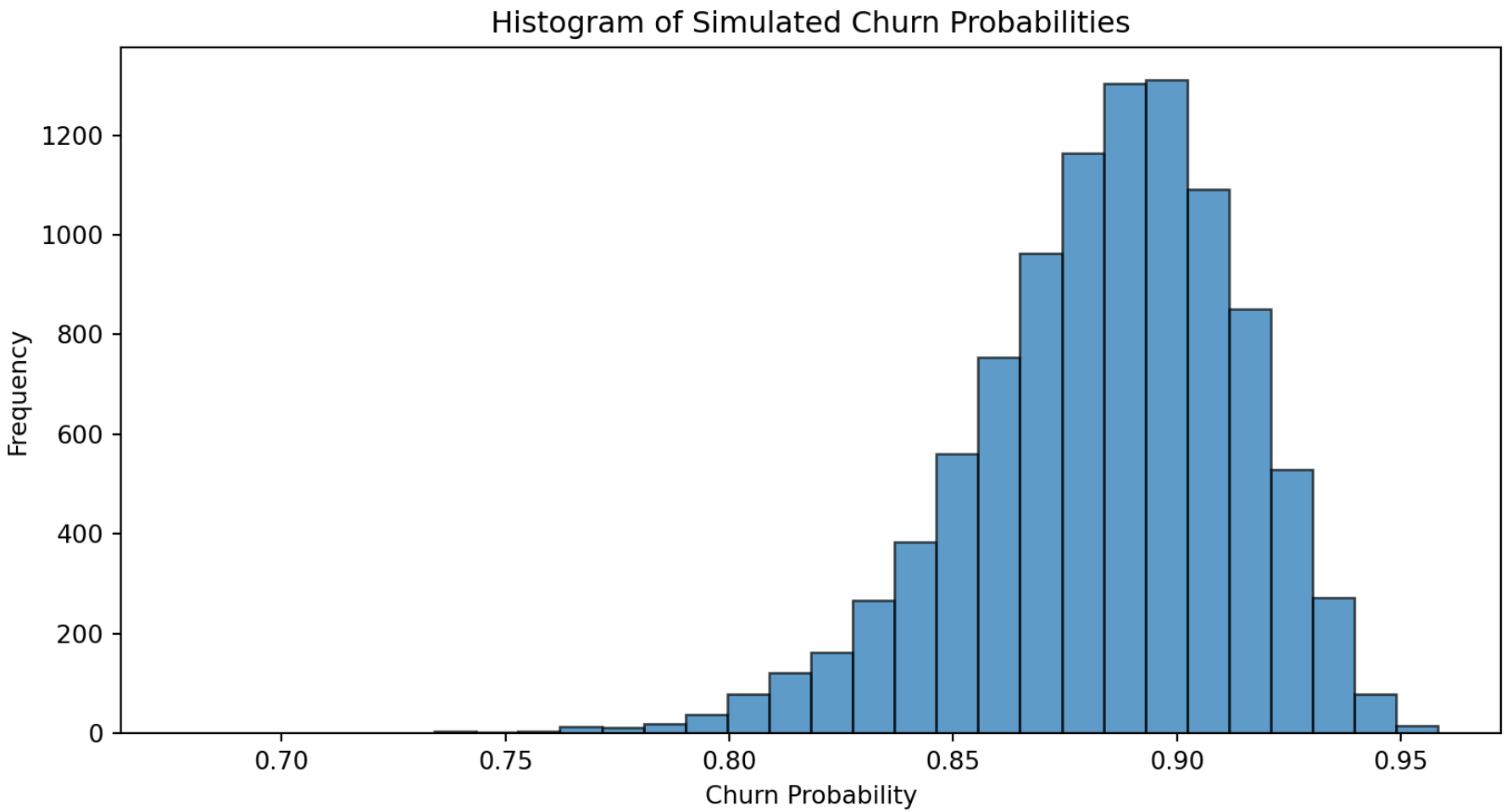
```python
import numpy as np
from scipy.stats import norm

# Example logistic regression coefficients and standard errors
coefficients = np.array([0.5, -0.3, 1.2])  # Intercept, Feature1, Feature2
std_errors = np.array([0.1, 0.2, 0.15])
customer_features = np.array([1, 0.8, 1.5])  # Intercept term, Feature1, Feature2

n_simulations = 10000
simulated_coefficients = np.random.normal(loc=coefficients,\
    scale=std_errors, size=(n_simulations, len(coefficients)))
churn_probabilities = 1 / (1 + np.exp(-np.dot(simulated_coefficients, \
    customer_features)))

# Analyze results
mean_probability = np.mean(churn_probabilities)
confidence_interval = np.percentile(churn_probabilities, [2.5, 97.5])
```

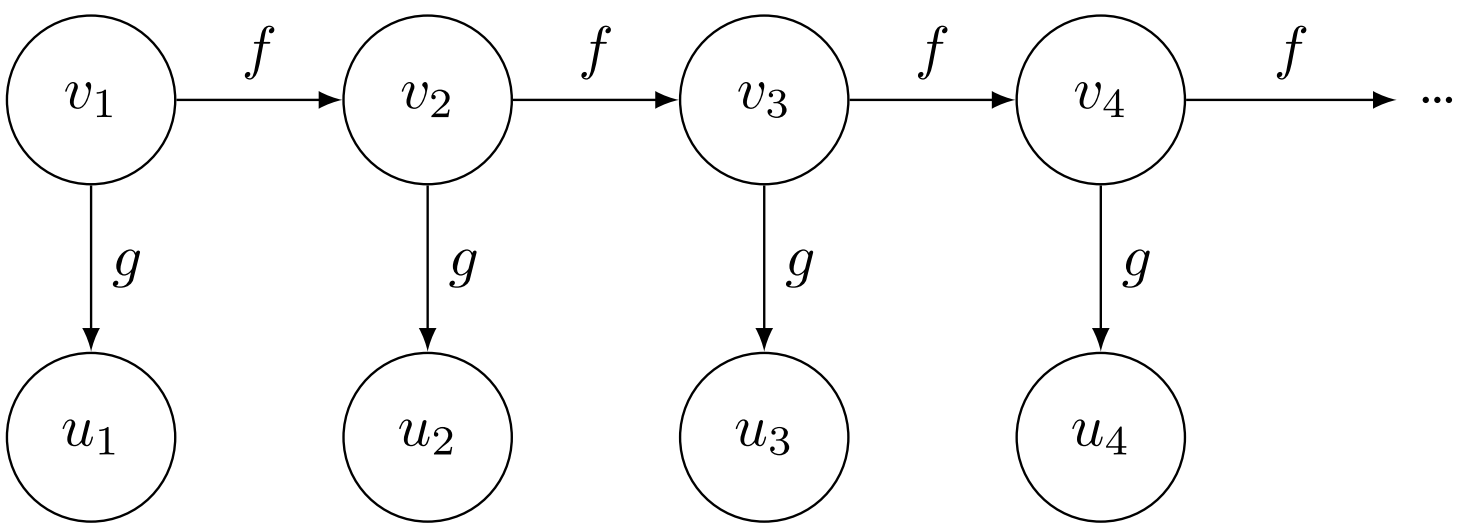
```
Estimated churn probability: 0.884
95% prediction interval: [0.815, 0.933]
```



2. Simulate random variables

2.1 Random Number Generators (RNGs)

- **Random number generators** (RNGs) are algorithms that produce sequences of numbers that appear random.
- RNGs produce a sequence $u_1, u_2, \ldots,$ that can be considered as realizations of independent random variables $U_1, U_2, \ldots, \sim \mathscr{U}([0; 1))$
- RNGs are based on a sequence defined by a **recurrence relation**: $v_{i+1} = f(v_i)$. Real numbers in $[0, 1)$ are then obtained by $u_i = g(v_i)$.
- v_1 is called the **seed** of the RNG.



Example: Linear Congruential Generator

- One of the oldest and simplest RNGs, where $v_i \in \mathbb{N}$
- Set a, c, m as integer constants
- Recurrence relation: $v_{i+1} = f(v_i) = (av_i + c) \mod m$
- The sequence $u_i = g(v_i) = v_i/m$ is then used as the sequence of random numbers

```
class LCG:
  def __init__(self, seed, a=1664525, c=1013904223, m=2**32):
    self.state = seed
    self.a, self.c, self.m = a, c, m
  def __iter__(self):
    return self
  def __next__(self):
    self.state = (self.a * self.state + self.c) % self.m
    return self.state / self.m
lcg_gen = LCG(12345)
next(lcg_gen)
```

```
0.02040268573909998
```

```
next(lcg_gen)
```

```
0.01654784823767841
```

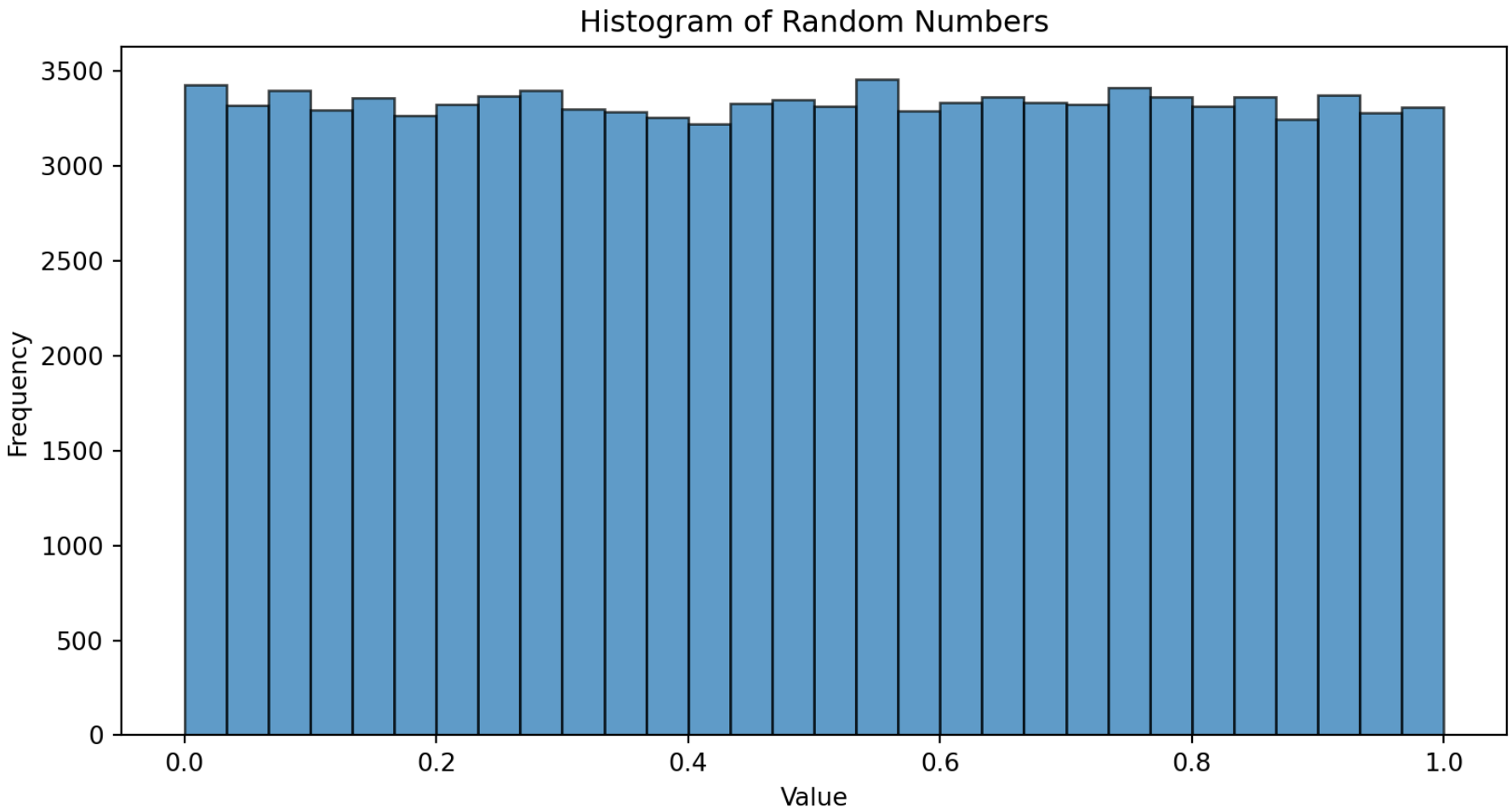
```
next(lcg_gen)
```

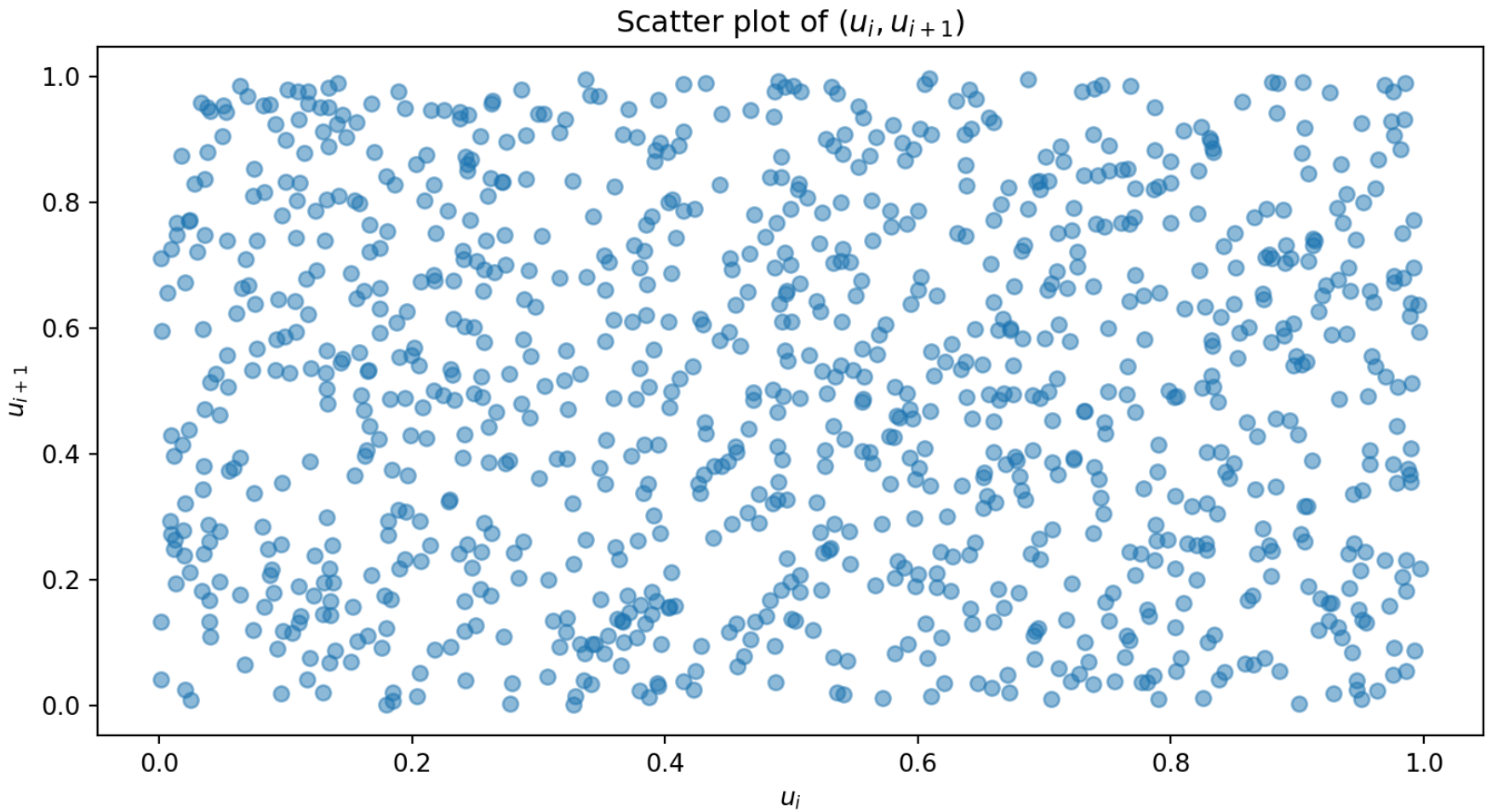
```
0.5431557944975793
```

amU Aix Marseille Université

```
1 next(lcg_gen)
```

```
0.6349040560889989
```


Histogram of Random Numbers
Frequency
Value
0
500
1000
1500
2000
2500
3000
3500
0.0
0.2
0.4
0.6
0.8
1.0


Scatter plot of (u_i, u_{i+1})
1.0
0.8
0.6
0.4
0.2
0.0
u_{i+1}
0.0
0.2
0.4
0.6
0.8
1.0
u_i

More efficient RNGs

Linear Congruential Generators have some drawbacks, and have been replaced by more efficient RNGs, such as:

- **Mersenne Twister** (MT19937): widely used in practice, period of $2^{19937} - 1$
- **Permuted Congruential Generator (PCG)**: a family of RNGs with good statistical properties and fast generation
- **Xorshift**: simple and fast RNGs with good statistical properties
- **L'Ecuyer's Combined Generators**: combines multiple RNGs to improve the quality of the generated numbers
- **...**

Use the RNGs that are implemented in your programming language or statistical software, expect for specific needs (e.g. cryptographic applications, parallel computation,...).

2.2 Inverse of the CDF method

- **Inverse transform method**: given a random variable X with CDF $F(x) = \mathbb{P}(X \leq x)$, we can generate X by inverting F if possible
- Let $U \sim \mathscr{U}([0;1))$, then $X' = F^{-1}(U)$ has the same distribution as X

Example: Exponential distribution

Let $X \sim \mathrm{Exp}(\lambda)$ with CDF $F(x) = 1 - \exp(-\lambda x)$, then $F^{-1}(u) = -\frac{1}{\lambda}\log(1-u)$

Since $1 - U$ is also $\mathscr{U}([0;1))$, we can also use $X' = -\frac{1}{\lambda}\log(U)$

- Note that F^{-1} is the quantile function of X: $F^{-1}(p)$ is the quantile of X of order p

The pseudo-inverse of the CDF is

$$F^{-\#}(p) = \inf\{x \in \mathbb{R} : F(x) \geq p\}, \quad p \in (0;1)$$

Exercise

- Compute the pseudo-inverse of the CDF of a Dirac distribution at x_0.
- Compute the pseudo-inverse of the CDF of a Bernoulli distribution with parameter p.
- Given a CDF $F(x)$, what is the distribution of $F^{-\#}(U)$ when $U \sim \mathscr{U}([0;1))$?

Other distributions linked with the exponential

Assume $U_1, U_2, \ldots \sim \mathscr{U}([0;1))$ are independent.

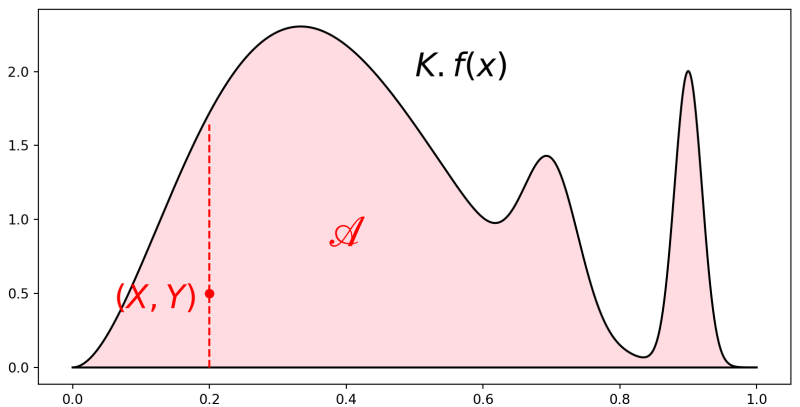
$$Y = -2\sum_{j=1}^{k} \log(U_j) \sim \chi^2_{2k}, \quad k \in \mathbb{N}^\star$$

$$Y = -\frac{1}{\beta}\sum_{j=1}^{a} \log(U_j) \sim \mathrm{Gamma}(a, \beta), \quad a \in \mathbb{N}^\star, \beta > 0$$

$$Y = \frac{\sum_{j=1}^{a} \log(U_j)}{\sum_{j=1}^{a+b} \log(U_j)} \sim \mathrm{Beta}(a, b), \quad a, b \in \mathbb{N}^\star$$

These formulas provide **a way to simulate** random variables **from these distributions**.

2.3 Rejection method



Let $f(x)$ be probability density function (PDF), and $K > 0$ a constant.

Set $\mathscr{A} = \{(x, y) : 0 \leq y \leq Kf(x)\}$.

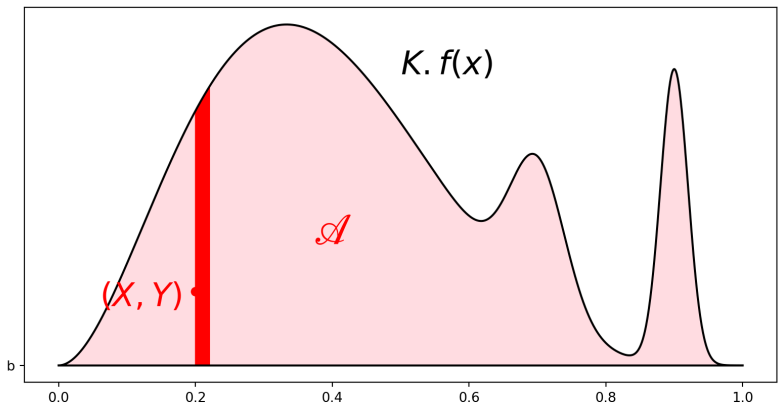
Theorem

$$(X, Y) \sim \mathscr{U}(\mathscr{A}) \iff$$

$$\begin{cases} X \sim \text{law with pdf } f \\ [Y|X = x] \sim \mathscr{U}([0; Kf(x)]) \end{cases}$$

Proof: Obvious? □

$$(X,Y) \sim \mathscr{U}(\mathscr{A}) \iff \begin{cases} X \sim \text{law with pdf } f \\ [Y|X=x] \sim \mathscr{U}([0;Kf(x)]) \end{cases} \quad ?$$



Proof: $\Longrightarrow$?

$\mathbb{P}(X \in [x;x+h]) \propto$ red area
$\propto KF(x+h) - KF(x)$
$= hKf(x) + o(h)$

Hence $X \sim f$

On the other hand,

$f(y|x) = \frac{f(x,y)}{f(x)} \propto f(x,y)$

$\propto \mathbf{1}\{(x,y) \in \mathscr{A}\}$

$\propto \mathbf{1}\{0 \leq y \leq Kf(x)\}$

Hence $[Y|X=x] \sim \mathscr{U}([0;Kf(x)])$.

$\Longleftarrow$? Since the laws of $[X]$ and $[Y|X]$ characterize entirely the law of (X,Y), the reciprocal is also true and we have the result. $\square$

Rejection algorithm

Assume

- know how to simulate $\sim g(x)$ and
- want to simulate $\sim f(x)$
- there exists $K > 0$ such that $\forall x,\ f(x) \leq Kg(x)$

Then, the rejection algorithm is:

- accepted $\leftarrow$ false
- while not accepted
  - $x \sim g$
  - $u \sim \mathscr{U}([0; 1))$
  - accepted $\leftarrow \left(u \leq f(x)/(Kg(x))\right)$
- return x

- accepted $\leftarrow$ false
- while not accepted
  - $x \sim g$
  - $u \sim \mathscr{U}([0;1))$
  - accepted $\leftarrow \left(u \leq f(x)/(Kg(x))\right)$
- return x

Set $y = Kg(x)u$ in the loop.

Theorem $\implies$ (x, y) is a draw from

$$\mathscr{U}(\mathscr{B}),\ \mathscr{B} = \{(x, y) : 0 \leq y \leq Kg(x)\}$$

It is accepted when $y \leq f(x)$, i.e. when (x, y) is in

$$\mathscr{A} = \{(x, y) : 0 \leq y \leq f(x)\}$$

Set $y = Kg(x)u$ in the loop.

Theorem $\implies$ (x, y) is a draw from

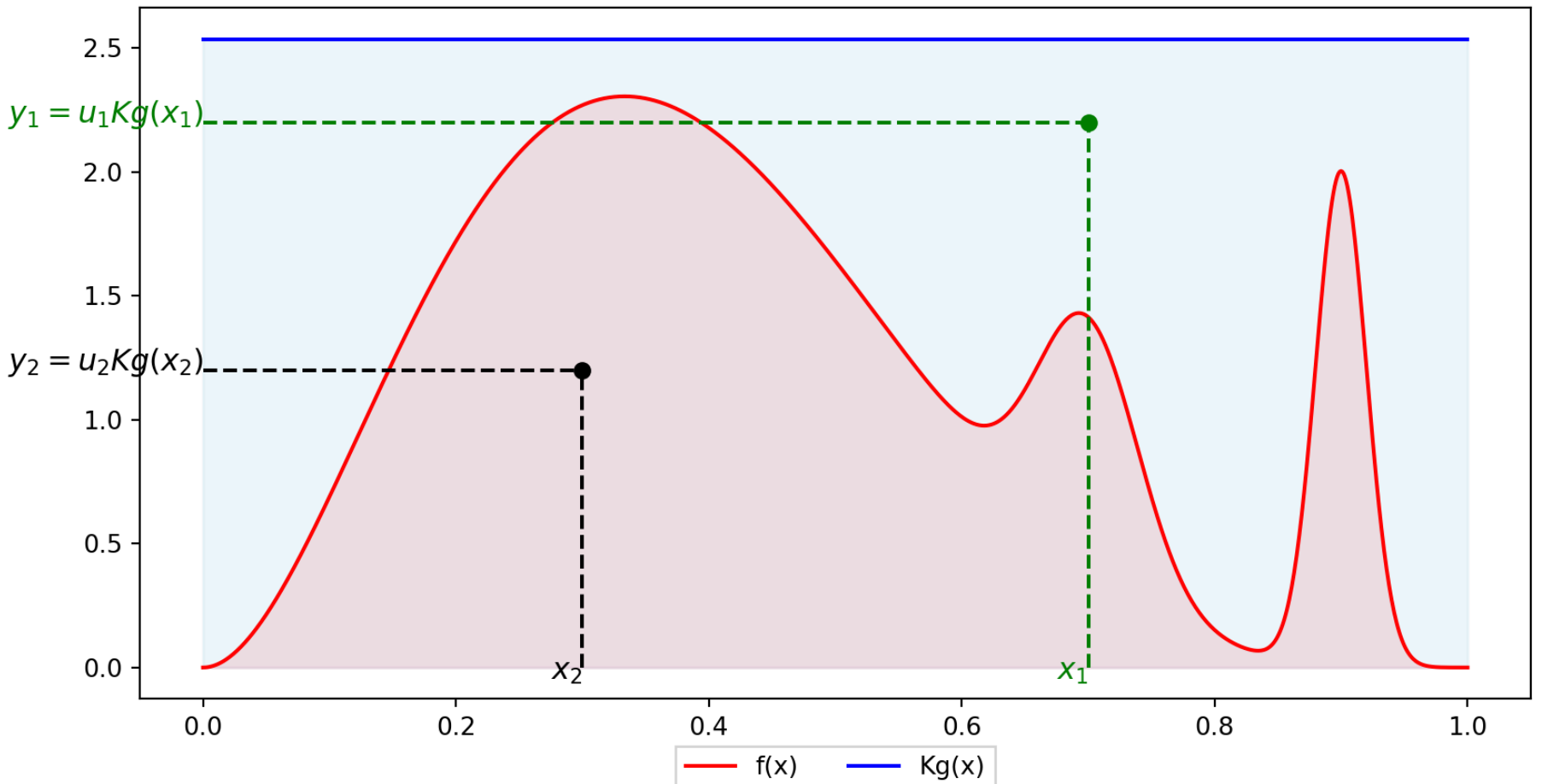
$$\mathscr{U}(\mathscr{B}),\ \mathscr{B} = \{(x, y) : 0 \leq y \leq Kg(x)\}$$

It is accepted when $y \leq f(x)$, i.e. when (x, y) is in

$$\mathscr{A} = \{(x, y) : 0 \leq y \leq f(x)\}$$

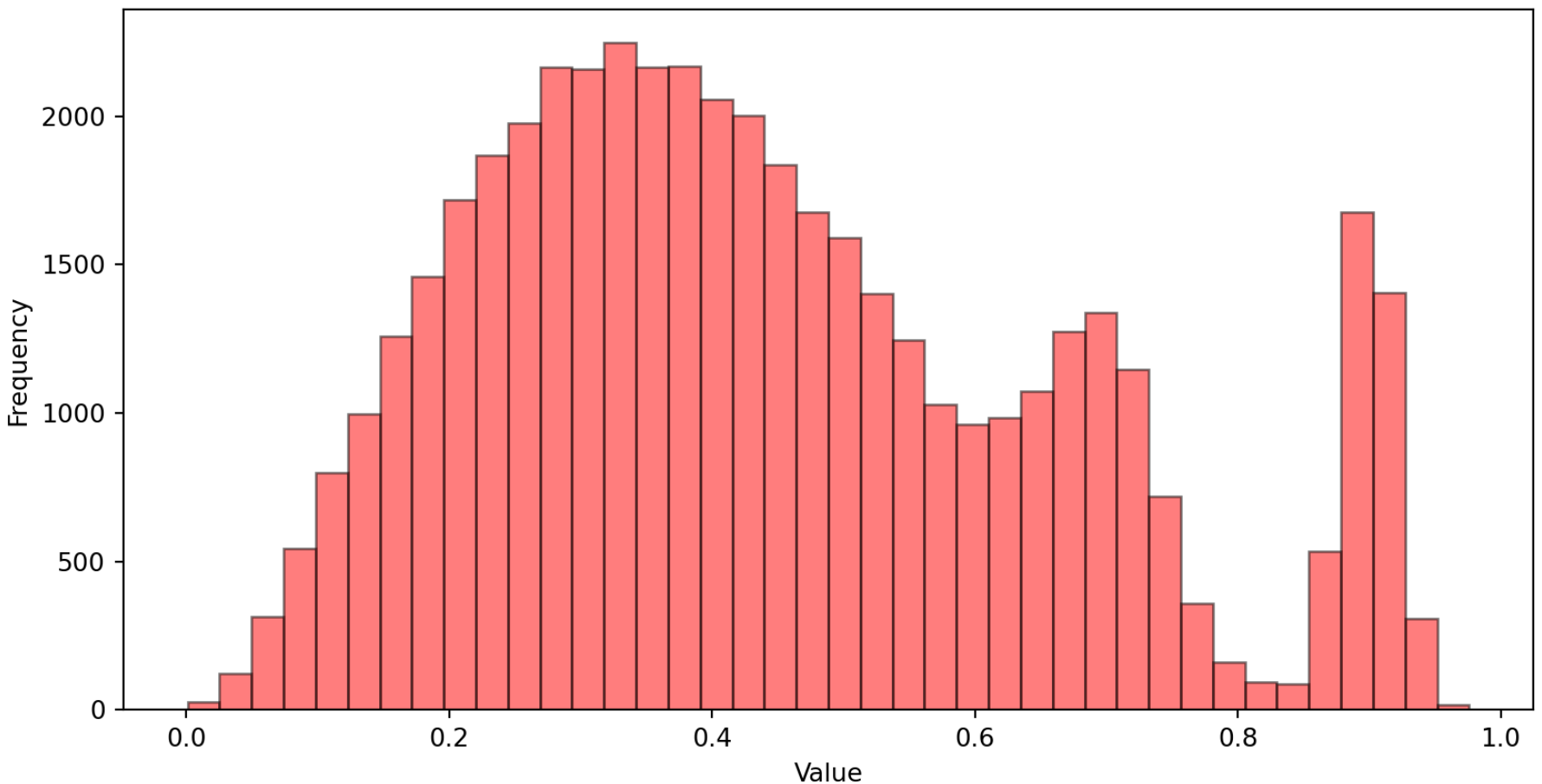
Since $\mathscr{A} \subset \mathscr{B}$, when accepted, (x, y) is a draw from $\mathscr{U}(\mathscr{A})$.

Theorem $\implies$ $X \sim f$ $\square$



The first draw x_1 is rejected because $y_1 > f(x_1)$, while the second draw x_2 is accepted because $y_2 \leq f(x_2)$.

When repeated $N = 100,000$ times, the accepted x's are distributed according to $f(x)$:



Efficiency of the rejection algorithm

- The algorithm is efficient if the acceptance probability is high
- The acceptance probability is $\dfrac{\text{area}(\mathscr{A})}{\text{area}(\mathscr{B})}$ where

$$\mathscr{A} = \{(x, y) : 0 \leq y \leq f(x)\}, \quad \mathscr{B} = \{(x, y) : 0 \leq y \leq Kg(x)\}$$

$\Longrightarrow$ choose g and K wisely such that $Kg(x)$ is close to $f(x)$ and such that

$$\forall x,\ f(x) \leq Kg(x)$$

Not always obivous to find g and K...

2.4 The Box-Muller method

Assume $U_1, U_2 \sim \mathscr{U}([0;1))$ are independent.

Then, the random variables

$$\begin{cases} Z_1 = \sqrt{-2\log(U_1)}\cos(2\pi U_2) \\ Z_2 = \sqrt{-2\log(U_1)}\sin(2\pi U_2) \end{cases}$$

are independent and follow a **standard normal distribution** $\mathscr{N}(0,1)$

Exercise

Prove that, if (R, Θ) are the polar coordinates of a random point $(X_1, X_2) \sim \mathscr{N}(0,1)^{\otimes 2}$, then, $R^2 \sim \mathrm{Exp}(1/2)$ and $\Theta \sim \mathscr{U}([0; 2\pi))$.

Prove then that the Box-Muller method is exact.

Other methods

Problem

The Cauchy distribution has density $g(x) = \dfrac{1}{\pi(1+x^2)}$.

1. Compute its CDF $G(x)$ and get a method to simulate according to the Cauchy distribution
2. Show that we can use the rejection method to simulate then according to the standard normal distribution and that the best K is $\sqrt{2\pi/e}$. What is the acceptance rate?
3. Use now the double-exponential distribution with density $h(x|\alpha) = \dfrac{\alpha}{2} e^{-\alpha|x|}$ to simulate according to the standard normal distribution. Show that the best K is $\sqrt{\dfrac{2}{\pi}} \alpha^{-1} e^{-\alpha^2/2}$.
4. Which value of α gives the largest acceptance rate?

2.5 Mixture representation

- The principle of a mixture representation is to represent a density f as the marginal of another distribution. For example,

$$f(x) = \sum_{i \in \mathscr{I}} \pi_i f_i(x), \quad f(x) = \int_{\mathscr{Y}} f(x|y) g(y) dy$$

  i.e., in the first case,

$$Y \text{ s.t. } \mathbb{P}(Y = i) = \pi_i, \quad [X|Y = i] \sim f_i$$

  and in the second case,

$$Y \sim g, \quad [X|Y = y] \sim f(\cdot|y)$$

- If we can simulate according to g and $f(\cdot|y)$, we can simulate according to f.

Exercise

1. Show that, if

$$Y \sim \text{Gamma}(n, (1-p)/p), \quad [X|Y=y] \sim \mathscr{P}(y),$$

   then X follows a negative binomial distribution $\text{NegBin}(n, p)$.

2. Show that a Student distribution with ν degrees of freedom, $t(\nu)$, can be represented as a mixture of centered normal distributions $\mathcal{N}(0, \sigma^2)$

3. Compute the probability mass function of the beta-binomial distribution defined as the law of X when

$$[X|Y=y] \sim \text{Bin}(n, y), \quad Y \sim \text{Beta}(a, b)$$

3. Simulation of Gaussian vectors

The univariate case and the problem

Set $\mu \in \mathbb{R}$ and $\sigma > 0$.

- If $Z \sim \mathcal{N}(0,1)$, then $X = \mu + \sigma Z \sim \mathcal{N}(\mu, \sigma^2)$
- Since we ca use other algorithms (e.g. Box-Muller) to simulate $\mathcal{N}(0,1)$, we can simulate any univariate Gaussian $\mathcal{N}(\mu, \sigma^2)$
- Note that we have to **multiply** the standard normal Z **by** the standard deviation σ, which is **the square root of the variance**

The problem to simulate a **vector** $X \in \mathbb{R}^d$ with mean μ and covariance matrix Σ is to define properly the **square root of the variance matrix** Σ

Variance matrices

Fix a random vector $X \in \mathbb{R}^d$. Its variance matrix Σ is a **symmetric positive semi-definite matrix**:

$$\Sigma = \mathbb{E}[(X-\mu)(X-\mu)^T], \quad \text{with } \mu = \mathbb{E}[X].$$

- $\Sigma^T = \mathbb{E}\Big[\big[(X-\mu)(X-\mu)^T\big]^T\Big] = \mathbb{E}[(X-\mu)(X-\mu)^T] = \Sigma$
- $\forall x \in \mathbb{R}^d, \quad x^T\Sigma x = \mathbb{E}\Big[\big(x^T(X-\mu)\big)^2\Big] \geq 0$

It can be **diagonalized in an orthornormal basis**, i.e.,

$$\Sigma = P\Lambda P^T, \quad \text{whith } P \in \mathscr{O}_d \text{ and } \Lambda = \text{diag}(\lambda_1, \ldots, \lambda_d)$$

where $\mathscr{O}_d$ is the set of orthogonal matrices and $\lambda_i \geq 0$ are the eigenvalues of Σ.

The matrix L defined by $L = P\sqrt{\Lambda}, \quad$ where $\sqrt{\Lambda} = \mathrm{diag}(\sqrt{\lambda_1}, \ldots, \sqrt{\lambda_d})$ is such that

$$LL^T = \Sigma.$$

This is **a square root of the variance** matrix Σ.

Definition

A matrix L of size $d \times d$ is **a square root** of a symmetric, positive, semi-definite matrix A of size $d \times d$ if $LL^T = A$.

It is **far from being unique**! If $R \in \mathscr{O}_d$, then LR is also a square root of Σ:

$$LR(LR)^T = LRR^TL^T = LL^T = \Sigma$$

If $d = 1$, $\mathscr{O}_1 = \{-1, 1\}$, and the square root is unique up to a sign. If $d > 1$, the cardinal of $\mathscr{O}_d$ is infinite…

The Cholesky decomposition

- The Cholesky decomposition is **a way to compute a square root** of a variance matrix Σ
- It is based on the fact that any variance matrix Σ can be written as $\Sigma = LL^T$ with L a **lower triangular matrix**
- The Cholesky decomposition is then the algorithm to compute L

It is **available in most programming languages** and statistical software:

- in R, through the `chol` function (in the `base` package)
- in Python, through the `numpy.linalg.cholesky` function

Simulate a Gaussian vector

How to simulate a Gaussian vector with mean μ and variance matrix Σ, i.e.,

$$\sim \mathcal{N}_d(\mu, \Sigma)?$$

1. Simulate $Z = (Z_1, \ldots, Z_d) \sim \mathcal{N}_d(0, I_d)$ with independent $Z_i \sim \mathcal{N}(0, 1)$
2. Compute a lower triangular matrix L such that $LL^T = \Sigma$ with the Cholesky algorithm
3. Return $X = \mu + LZ$

Proof. LZ is a linear transform of a Gaussian vector, thus it is a Gaussian vector. Hence,

$$X = \mu + LZ \sim \mathcal{N}_d(\mathbb{E}(X), \mathbb{V}(X)).$$

Moreover, $\mathbb{E}(X) = \mu + L\mathbb{E}(Z) = \mu$ and

$$\mathbb{V}(X) = \mathbb{E}\Big[(X - \mu)(X - \mu)^T\Big] = L\mathbb{E}(ZZ^T)L^T = LL^T = \Sigma. \quad \square$$

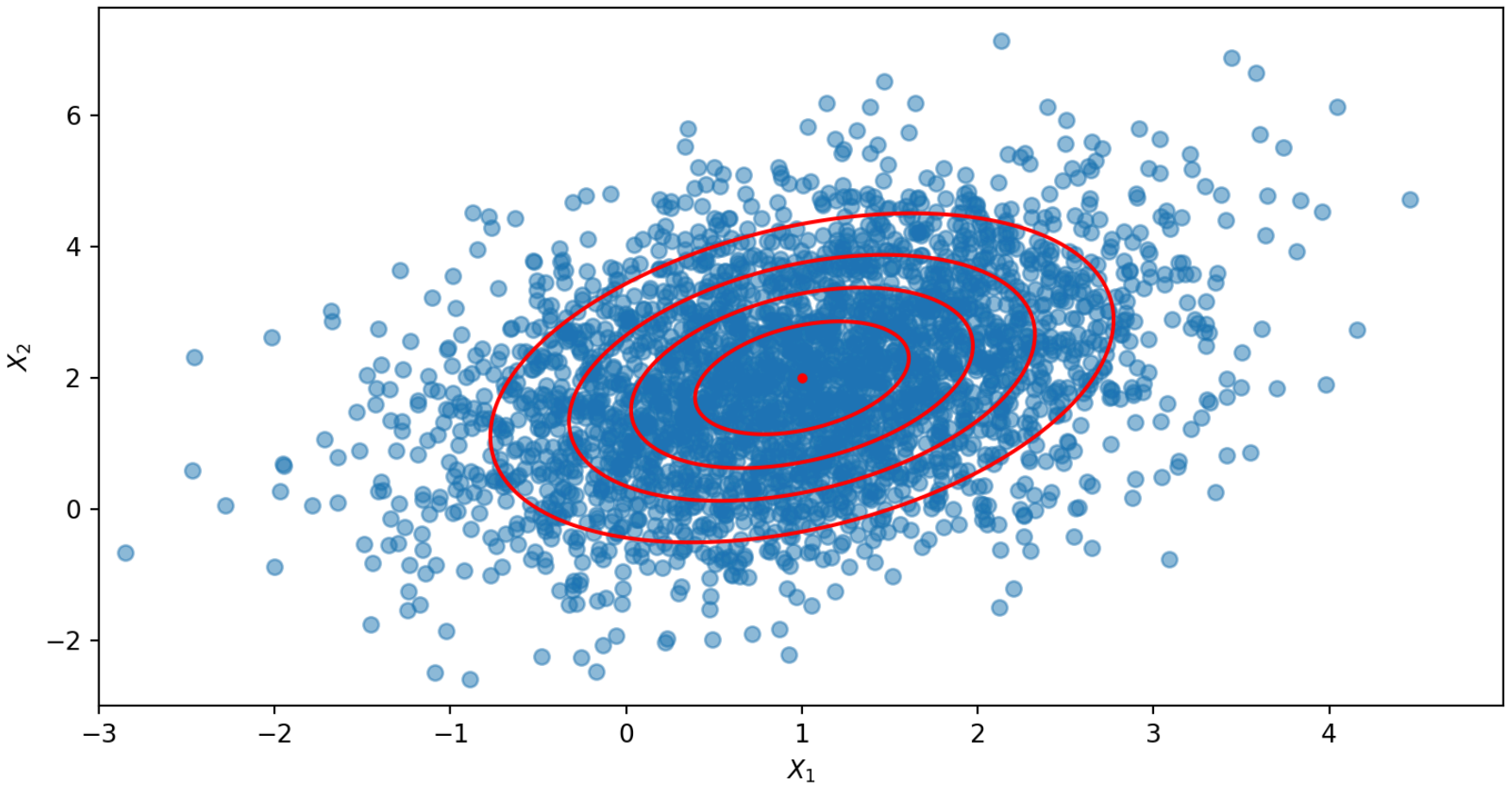
Example

```python
import numpy as np
from numpy.linalg import cholesky
# Parameters
mu = np.array([1, 2])
Sigma = np.array([[1, 0.5], [0.5, 2]])
# Cholesky decomposition
L = cholesky(Sigma)
print(L)
```

```
[[1.         0.        ]
 [0.5        1.32287566]]
```

```python
# Simulate a Gaussian vector
Z = np.random.normal(size=2)
X = mu + L @ Z
print(X)
```

```
[2.20282522 3.84654839]
```



$N = 3,000$ simulated Gaussian vectors in blue and the contour plot of the targeted Gaussian distribution in red

Exercise

1. Approximate, with a Monte Carlo algorithm, the probability

$$\mathbb{P}\left(X_1 \geq 3, X_2 \geq 4\right), \quad \text{when } X = (X_1, X_2) \sim \mathcal{N}_2\left(\mu, \Sigma\right)$$

with

$$\mu = \begin{pmatrix} 1 \\ 2 \end{pmatrix}, \quad \Sigma = \begin{pmatrix} 1 & 0.5 \\ 0.5 & 2 \end{pmatrix}$$

2. What happens to the MC method when we want to compute

$$\mathbb{P}(X_1 \geq 5, X_2 \geq 7)?$$

Questions ?