Algorithms for random variates generator

Chiara Iannicelli

November 2023

1 Introduction

Random variate generators play a crucial role in simulating and generating random numbers following specific probability distributions.
Here are some well-known algorithms for generating random variates for common probability distributions:

- **Uniform Distribution:** Linear Congruential Generator (LCG): A simple and widely used method for generating pseudo-random numbers with a uniform distribution. The formula is $X_{n+1} = (aX_n + c)\, mod\, m$

- **Normal Distribution:**
  - Box-Muller Transform: Converts two independent uniform random variables into two independent standard normal random variables.
  - Marsaglia's Polar Method: Another method for generating standard normal random variables using polar coordinates.

- **Exponential Distribution:** Inverse Transform Sampling: If U is a uniform random variable on (0, 1), then $X = -\frac{1}{\lambda} ln(1 - U)$ follows an exponential distribution with rate λ.

- **Poisson Distribution:** Poisson Process Simulation: Using the properties of a Poisson process to generate Poisson-distributed random variables.

- **Gamma Distribution:** Sum of Exponentials: A gamma distribution with integer shape parameter k can be generated as the sum of k independent exponential random variables.

- **Beta Distribution:** Transformation Method: Using the cumulative distribution function (CDF) and its inverse.

- **Cauchy Distribution:** Ratio of Normals: Generating random variables from the ratio of two independent standard normal variables.

- **Log-Normal Distribution:** Exponential of a Normal: If Z is a standard normal random variable, then $X = e^{\mu+\sigma Z}$ follows a log-normal distribution.

- **Chi-Square Distribution:** Sum of Squares of Normals: A chi-square distribution with k degrees of freedom can be generated as the sum of the squares of k independent standard normal variables.

- **F Distribution:** Ratio of Chi-Squares: If X and Y are independent chi-square random variables with k_1 and k_2 degrees of freedom, respectively, then $F = \frac{\frac{X}{k_1}}{\frac{Y}{k_2}}$ follows an F-distribution.

These algorithms are foundational in statistical simulations and are often implemented in programming languages and statistical software libraries.

Keep in mind that some distributions, especially non-standard or complex ones, may require specialized methods, and there are advanced algorithms for generating random variates in such cases.

2 Simulations

Here follows the possible code to simulate some of the alghoritm above.

2.1 Uniform Distribution (Linear Congruential Generator):

```python
import numpy as np
import matplotlib.pyplot as plt

# Set seed for reproducibility
np.random.seed(42)

# Number of samples
num_samples = 1000

# Uniform Distribution (Linear Congruential Generator)
uniform_samples = np.random.rand(num_samples)

# Plot histogram for visualization
plt.hist(uniform_samples, bins=30, edgecolor='black')
plt.title('Uniform Distribution')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.show()

```

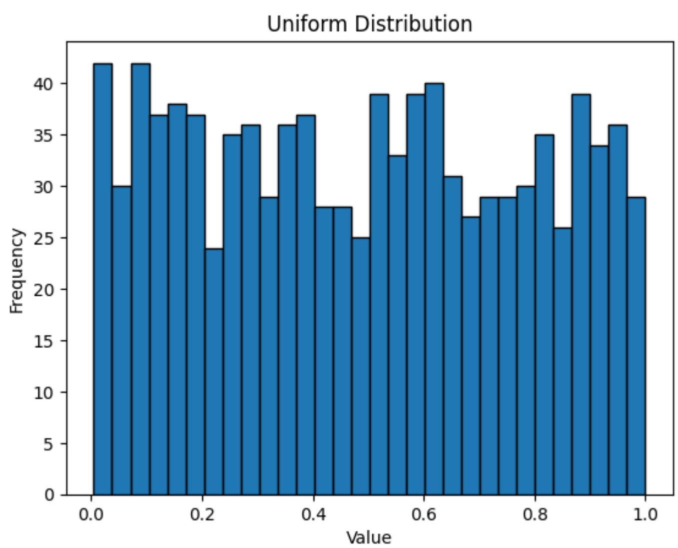

Figure 1: Uniform Distribution

2.2 Normal Distribution (Box-Muller Transform):

```python
import numpy as np
import matplotlib.pyplot as plt

# Set seed for reproducibility
np.random.seed(42)

# Number of samples
num_samples = 1000

# Normal Distribution (Box-Muller Transform)
uniform_samples = np.random.rand(2 * num_samples)
normal_samples_box_muller = np.sqrt(-2 * np.log(uniform_samples[::2])) * np.cos(2 * np.pi * uniform_samples[1::2])

# Plot histogram for visualization
plt.hist(normal_samples_box_muller, bins=30, edgecolor='black')
plt.title('Normal Distribution (Box-Muller)')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.show()

```

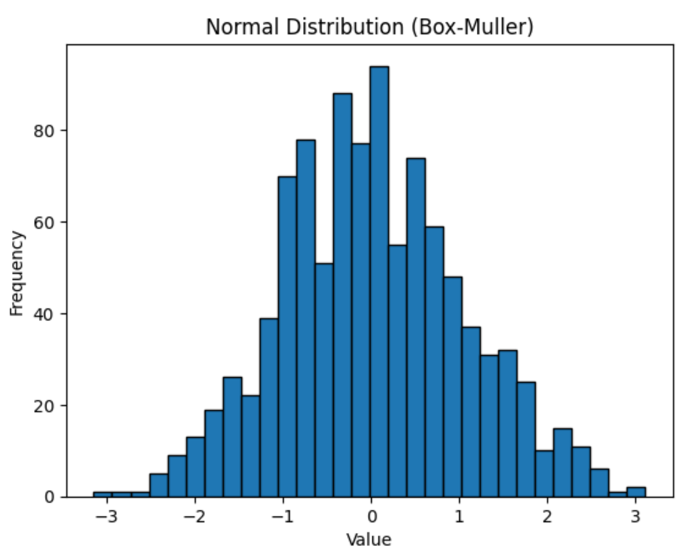

Figure 2: Normal Distribution

2.3 Exponential Distribution (Inverse Transform Sampling):

```python
import numpy as np
import matplotlib.pyplot as plt

# Set seed for reproducibility
np.random.seed(42)

# Number of samples
num_samples = 1000

# Exponential Distribution (Inverse Transform Sampling)
lambda_param = 0.5
exponential_samples = -np.log(1 - np.random.rand(num_samples)) / lambda_param

# Plot histogram for visualization
plt.hist(exponential_samples, bins=30, edgecolor='black')
plt.title('Exponential Distribution')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.show()
```

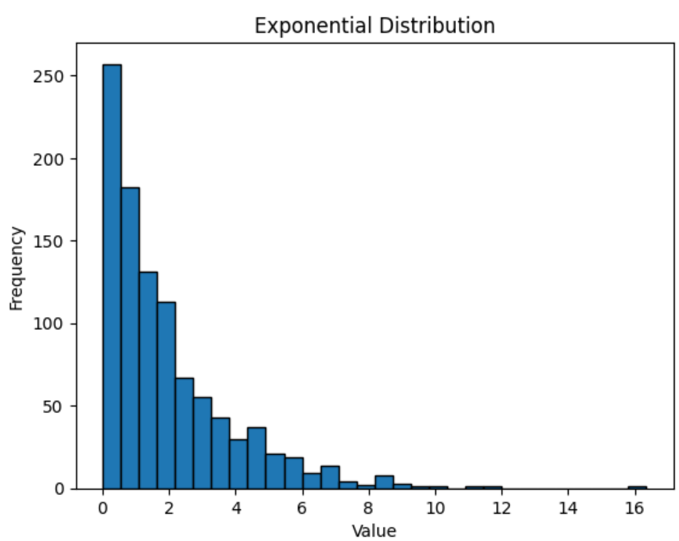

Figure 3: Exponential Distribution

2.4 Poisson Distribution (Poisson Process Simulation):

```
import numpy as np
import matplotlib.pyplot as plt

# Set seed for reproducibility
np.random.seed(42)

# Number of samples
num_samples = 1000

# Poisson Distribution (Poisson Process Simulation)
lambda_poisson = 3
poisson_samples = np.random.poisson(lambda_poisson, num_samples)

# Plot histogram for visualization
plt.hist(poisson_samples, bins=30, edgecolor='black')
plt.title('Poisson Distribution')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.show()

```

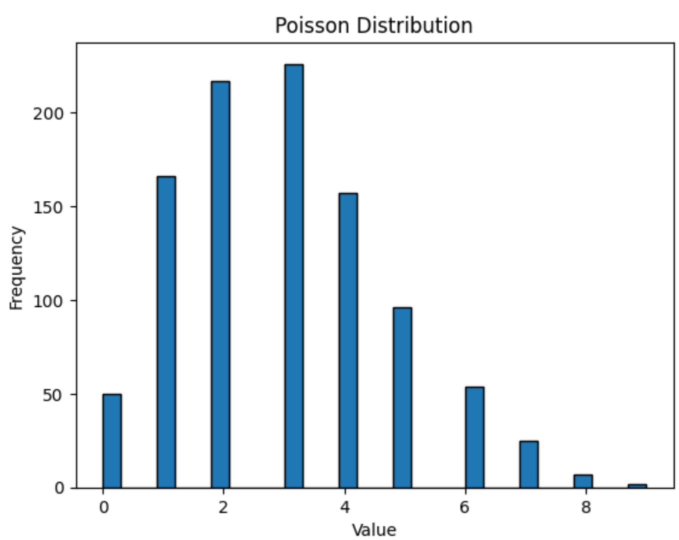

Figure 4: Poisson Distribution

2.5 Gamma Distribution (Sum of Exponentials):

```
import numpy as np
import matplotlib.pyplot as plt

# Set seed for reproducibility
np.random.seed(42)

# Number of samples
num_samples = 1000

# Gamma Distribution (Sum of Exponentials)
k_gamma = 2
gamma_samples = -np.log(np.random.rand(k_gamma,
    num_samples)).sum(axis=0)

# Plot histogram for visualization
plt.hist(gamma_samples, bins=30, edgecolor='black')
plt.title('Gamma Distribution')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.show()

```

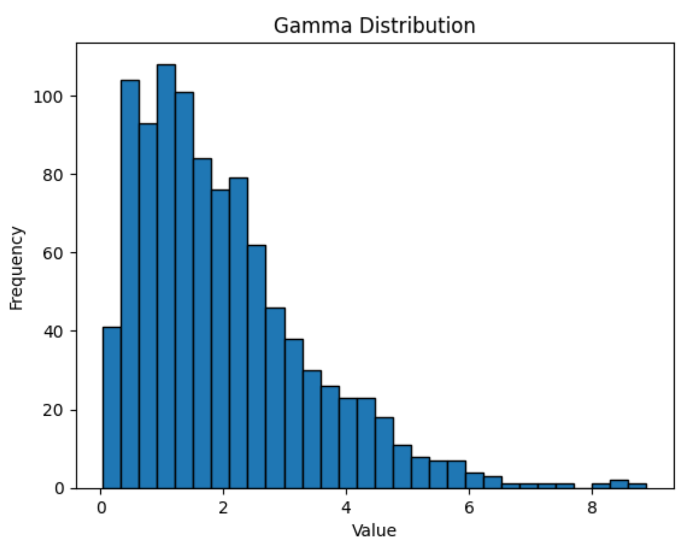

Figure 5: Gamma Distribution

2.6 Beta Distribution (Transformation Method):

```python
import numpy as np
import matplotlib.pyplot as plt

# Set seed for reproducibility
np.random.seed(42)

# Number of samples
num_samples = 1000

# Beta Distribution (Transformation Method)
alpha_beta = 2
beta_samples = np.random.beta(alpha_beta, alpha_beta, num_samples)

# Plot histogram for visualization
plt.hist(beta_samples, bins=30, edgecolor='black')
plt.title('Beta Distribution')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.show()

```

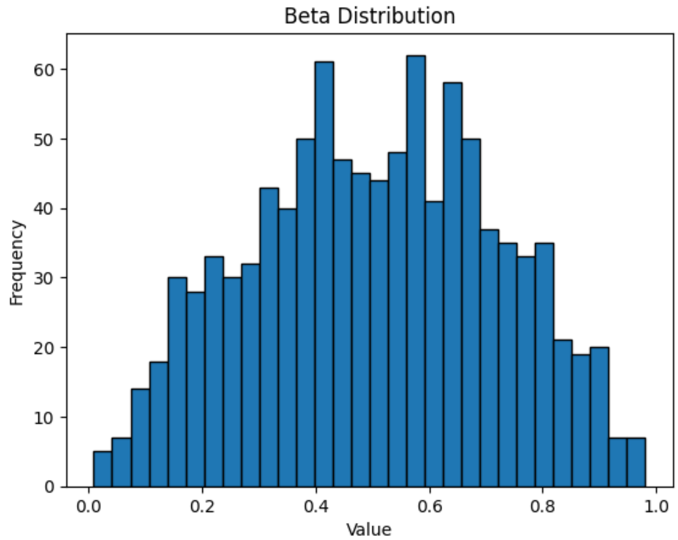

Figure 6: Beta Distribution

2.7 Cauchy Distribution (Ratio of Normals):

```
import numpy as np
import matplotlib.pyplot as plt

# Set seed for reproducibility
np.random.seed(42)

# Number of samples
num_samples = 1000

# Cauchy Distribution (Ratio of Normals)
cauchy_samples = normal_samples_box_muller[::2] /
↪  normal_samples_box_muller[1::2]

# Plot histogram for visualization
plt.hist(cauchy_samples, bins=30, edgecolor='black')
plt.title('Cauchy Distribution')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.show()

```

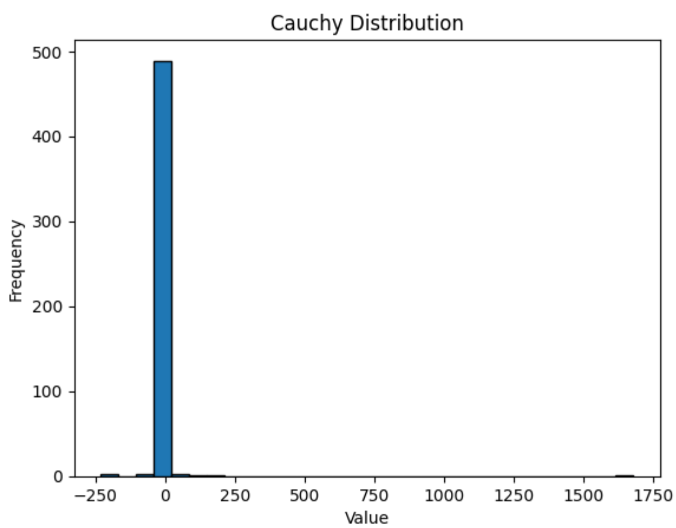

Figure 7: Cauchy Distribution

2.8 Log-Normal Distribution (Exponential of a Normal):

```
import numpy as np
import matplotlib.pyplot as plt

# Set seed for reproducibility
np.random.seed(42)

# Number of samples
num_samples = 1000

# Log-Normal Distribution (Exponential of a Normal)
log_normal_samples = np.exp(normal_samples_box_muller)

# Plot histogram for visualization
plt.hist(log_normal_samples, bins=30, edgecolor='black')
plt.title('Log-Normal Distribution')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.show()

```

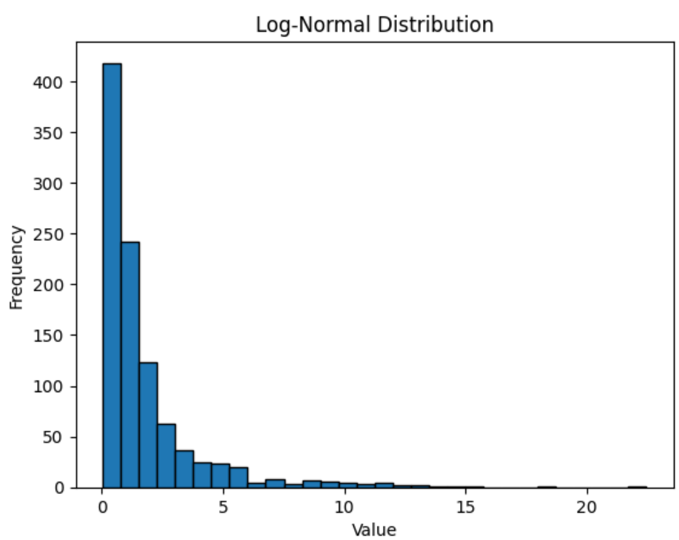

Figure 8: Log-Normal Distribution

2.9 Chi-Square Distribution (Sum of Squares of Normals):

```
import numpy as np
import matplotlib.pyplot as plt

# Set seed for reproducibility
np.random.seed(42)

# Number of samples
num_samples = 1000

# Chi-Square Distribution (Sum of Squares of Normals)
k_chi_square = 3
chi_square_samples = (normal_samples_box_muller ** 2).sum(axis=0)

# Plot histogram for visualization
plt.hist(chi_square_samples, bins=30, edgecolor='black')
plt.title('Chi-Square Distribution')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.show()

```

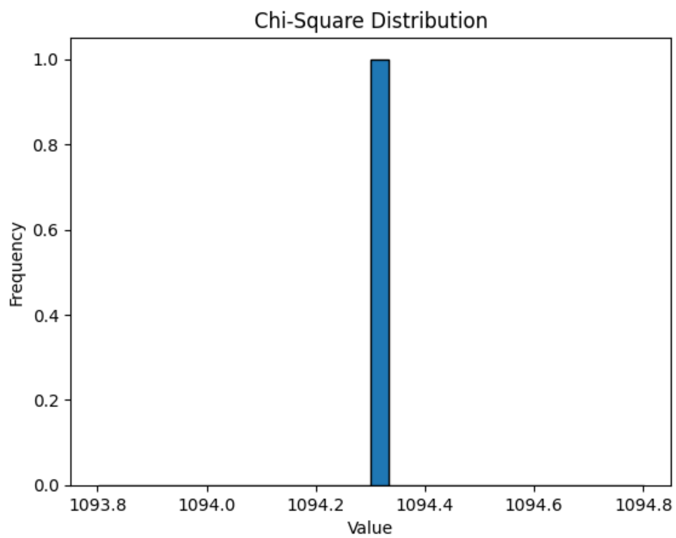

Figure 9: Chi-Square Distribution

2.10 F Distribution (Ratio of Chi-Squares):

```
import numpy as np
import matplotlib.pyplot as plt

# Set seed for reproducibility
np.random.seed(42)

# Number of samples
num_samples = 1000

# F Distribution (Ratio of Chi-Squares)
k1_f = 3
k2_f = 4
chi_square1_f = (normal_samples_box_muller[:k1_f] ** 2).sum(axis=0)
chi_square2_f = (normal_samples_box_muller[k1_f:k1_f+k2_f] **
↪  2).sum(axis=0)
f_samples = (chi_square1_f / k1_f) / (chi_square2_f / k2_f)

# Plot histogram for visualization
plt.hist(f_samples, bins=30, edgecolor='black')
plt.title('F Distribution')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.show()

```

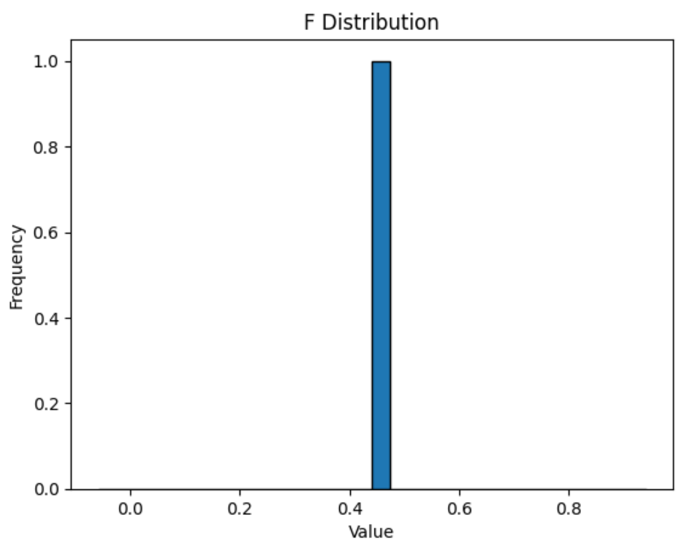

Figure 10: F Distrbution