Detailed Explanation of Random Variable Generation

Generating random variables is a critical concept in simulations, statistical modeling, and machine learning. It involves creating random numbers or data that follow a specific probability distribution, such as uniform, normal, or exponential.

Key Methods of Generating Random Variables

1. Inverse Transform Sampling

This method is widely used to generate random variables from any distribution, given its cumulative distribution function (CDF).

Steps:

1. Generate a random number $U \sim \text{Uniform}(0, 1)$.
2. Set $X = F^{-1}(U)$, where F^{-1} is the inverse of the cumulative distribution function $F(x)$.

Example: Exponential Distribution For the exponential distribution with CDF:

$$F(x) = 1 - e^{-\lambda x}, \quad x \geq 0$$

The inverse CDF is:

$$F^{-1}(u) = -\frac{\ln(1-u)}{\lambda}$$

To generate an exponential random variable X:

- Generate $U \sim \text{Uniform}(0, 1)$.
- Compute $X = -\frac{\ln(1-U)}{\lambda}$.

Listing 1: Python Implementation: Inverse Transform Sampling

```python
import numpy as np

# Generate Exponential Random Variables
lambda_param = 1.0  # Rate parameter
u = np.random.uniform(0, 1, 1000)  # Generate 1000 uniform random numbers
x = -np.log(1 - u) / lambda_param  # Apply inverse transform
print(f"First 5 Exponential Random Variables: {x[:5]}")
```

2. Rejection Sampling

Rejection sampling is used to generate samples from a complex target distribution $f(x)$ by using a simpler proposal distribution $g(x)$.

Steps:

1. Choose a proposal distribution $g(x)$ and a constant c such that $f(x) \leq c \cdot g(x)$ for all x.
2. Repeat the following:
   (a) Sample $Y \sim g(x)$.
   (b) Generate $U \sim \text{Uniform}(0, 1)$.
   (c) Accept Y if $U \leq \frac{f(Y)}{c \cdot g(Y)}$.

Listing 2: Python Implementation: Rejection Sampling

```python
def target_pdf(x):
    return 0.5 * np.exp(-0.5 * x)  # Target distribution

def proposal_pdf(x):
    return np.exp(-x)  # Proposal distribution (Exponential with lambda=1)

def rejection_sampling(n):
    samples = []
    c = 2  # Constant such that f(x) <= c * g(x)
    while len(samples) < n:
        y = np.random.exponential(scale=1.0)  # Sample from proposal
            distribution
        u = np.random.uniform(0, 1)
        if u <= target_pdf(y) / (c * proposal_pdf(y)):
            samples.append(y)
    return np.array(samples)

# Generate samples
samples = rejection_sampling(1000)
print(f"First 5 Samples from Rejection Sampling: {samples[:5]}")
```

3. Box-Muller Method (For Normal Distribution)

The Box-Muller method generates samples from a standard normal distribution $N(0,1)$ using two independent uniform random variables.

Steps:

1. Generate $U_1, U_2 \sim \text{Uniform}(0,1)$.
2. Compute:
$$Z_1 = \sqrt{-2\ln(U_1)}\cos(2\pi U_2)$$
$$Z_2 = \sqrt{-2\ln(U_1)}\sin(2\pi U_2)$$
3. Z_1 and Z_2 are independent and follow $N(0,1)$.

Listing 3: Python Implementation: Box-Muller Method

```python
# Box-Muller Method for Normal Distribution
u1 = np.random.uniform(0, 1, 1000)
u2 = np.random.uniform(0, 1, 1000)

z1 = np.sqrt(-2 * np.log(u1)) * np.cos(2 * np.pi * u2)
z2 = np.sqrt(-2 * np.log(u1)) * np.sin(2 * np.pi * u2)

print(f"First 5 Normal Random Variables (Z1): {z1[:5]}")
print(f"First 5 Normal Random Variables (Z2): {z2[:5]}")
```

4. Sampling from Multivariate Distributions

When generating random variables from multivariate distributions, the most common approach is to use **Cholesky decomposition** for correlated Gaussian distributions.

Listing 4: Python Implementation: Multivariate Gaussian Sampling

```python
# Sampling from Multivariate Gaussian
mean = [0, 0]  # Mean vector
cov = [[1, 0.8], [0.8, 1]]  # Covariance matrix

# Generate samples
samples = np.random.multivariate_normal(mean, cov, size=1000)
print(f"First 5 Samples from Multivariate Gaussian: {samples[:5]}")
```

5. Monte Carlo Sampling

Example: Estimating π

1. Generate random points (x, y) uniformly in a square of side length 2 centered at the origin.
2. Count the number of points inside the unit circle ($x^2 + y^2 \leq 1$).
3. Estimate π using the ratio of points inside the circle to the total number of points.

Listing 5: Python Implementation: Monte Carlo Simulation for π

```
# Monte Carlo Simulation to Estimate Pi
n = 100000
x = np.random.uniform(-1, 1, n)
y = np.random.uniform(-1, 1, n)

# Points inside the unit circle
inside_circle = x**2 + y**2 <= 1
pi_estimate = 4 * np.sum(inside_circle) / n

print(f"Estimated Pi: {pi_estimate}")
```

Applications of Random Variable Generation

1. **Simulations**: Simulating physical systems, financial models, and queueing systems.
2. **Monte Carlo Methods**: Estimating integrals, solving optimization problems, and risk analysis.
3. **Machine Learning**: Generating synthetic data, initializing weights in neural networks, or sampling in probabilistic models (e.g., Bayesian inference).
4. **Cryptography**: Generating random keys or passwords.
5. **Optimization**: Stochastic optimization methods (e.g., Simulated Annealing).