EEC 264 Term Project - Spring 2024

Ankit Goel, Date: 06/12/2024

1 Generation of Random Variables

1.1 Verification of Uniform Distribution

Objective: To verify that if X has a probability distribution $F_X(x)$, then $Y = F_X(X)$ is uniformly distributed over $[0, 1]$.

Proof:

1. Let $Y = F_X(X)$.
2. To find the CDF of Y:
$$F_Y(y) = P(Y \leq y) = P(F_X(X) \leq y)$$
3. Since F_X is a monotonically increasing function:
$$F_Y(y) = P(X \leq F_X^{-1}(y)) = F_X(F_X^{-1}(y)) = y$$
4. Therefore, $F_Y(y) = y$, which is the CDF of a uniform distribution over $[0, 1]$.

Conclusion: If X has a probability distribution $F_X(x)$, then $Y = F_X(X)$ is uniformly distributed over $[0, 1]$.

1.2 Generating Rayleigh and Cauchy Distributed Variables

Objective: To use the inverse transform method to generate Rayleigh and Cauchy distributed random variables.

Rayleigh Distribution:

- CDF: $F_{X1}(x) = 1 - \exp\left(-\frac{x^2}{2}\right)$
- Inverse CDF: $X_1 = \sqrt{-2\ln(1 - Y)}$, where $Y \sim U(0, 1)$

Cauchy Distribution:

- CDF: $F_{X2}(x) = \frac{1}{\pi}\arctan(x) + \frac{1}{2}$

- Inverse CDF: $X_2 = \tan(\pi(Y - 0.5))$, where $Y \sim U(0, 1)$

Implementation in Python:

```
# Rayleigh distribution: f_X1(x) = x * exp(-x^2 / 2) for x >= 0
def generate_rayleigh(n):
    Y = np.random.uniform(0, 1, n)
    X1 = np.sqrt(-2 * np.log(1 - Y))
    return X1

# Cauchy distribution: f_X2(x) = 1 / (pi * (1 + x^2))
def generate_cauchy(n):
    Y = np.random.uniform(0, 1, n)
    X2 = np.tan(np.pi * (Y - 0.5))
    return X2

# Generate samples
rayleigh_samples = generate_rayleigh(100000)
cauchy_samples = generate_cauchy(100000)

# Plot the histograms
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.hist(rayleigh_samples, bins=50, density=True, alpha=0.6, color='b')
plt.title('Rayleigh Distributed Samples')
plt.xlabel('X1')
plt.ylabel('Density')

plt.subplot(1, 2, 2)
plt.hist(cauchy_samples, bins=50, density=True, alpha=0.6, color='r')
plt.title('Cauchy Distributed Samples')
plt.xlabel('X2')
plt.ylabel('Density')

plt.tight_layout()
plt.show()
```

Conclusion: The histograms confirm that X_1 follows a Rayleigh distribution and X_2 follows a Cauchy distribution.

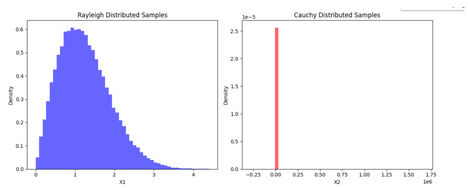

Figure 1: Rayleigh and Cauchy distributed samples

2 Monte Carlo Simulation of Test Performance

2.1 Verification of Unbiased Estimator

Objective: To verify that $E[Z(N)] = p$, where $Z(N) = \frac{1}{N}\sum_{i=1}^{N} X_i$ and X_i are i.i.d. Bernoulli random variables with $P(X_i = 1) = p$.

Proof:

1. $Z(N) = \frac{1}{N}\sum_{i=1}^{N} X_i$
2. By linearity of expectation:
$$E[Z(N)] = E\left[\frac{1}{N}\sum_{i=1}^{N} X_i\right] = \frac{1}{N}\sum_{i=1}^{N} E[X_i]$$
3. Since $E[X_i] = p$:
$$E[Z(N)] = \frac{1}{N} \cdot N \cdot p = p$$

Conclusion: The estimator $Z(N)$ is unbiased, as $E[Z(N)] = p$.

2.2 Standard Deviation and Relative Error

Objective: To evaluate the standard deviation $\sigma_{Z(N)}$ of $Z(N)$ and ensure that the relative error $\frac{\sigma_{Z(N)}}{p}$ does not exceed 10%.

Derivation:

1. Variance of $Z(N)$:
$$\mathrm{Var}(Z(N)) = \frac{1}{N^2}\sum_{i=1}^{N} \mathrm{Var}(X_i) = \frac{p(1-p)}{N}$$
2. Standard deviation:
$$\sigma_{Z(N)} = \sqrt{\frac{p(1-p)}{N}}$$
3. Relative error:
$$\frac{\sigma_{Z(N)}}{p} = \frac{\sqrt{\frac{p(1-p)}{N}}}{p} = \frac{\sqrt{1-p}}{\sqrt{N} \cdot \sqrt{p}} \leq 0.1$$

4. Solving for N:

$$\sqrt{N} \geq \frac{\sqrt{1-p}}{0.1\sqrt{p}}$$

$$N \geq \left(\frac{\sqrt{1-p}}{0.1\sqrt{p}}\right)^2$$

$$N \geq \frac{1-p}{0.01p}$$

Final Formula:

$$N \geq \frac{1-p}{0.01p}$$

Example Calculation in Python:

```
# a) Unbiased Estimator
def monte_carlo_estimator(N, p):
    samples = np.random.binomial(1, p, N)
    Z_N = np.mean(samples)
    return Z_N

# b) Standard Deviation and Relative Error
def calculate_std_and_relative_error(N, p):
    samples = np.random.binomial(1, p, N)
    Z_N = np.mean(samples)
    sigma_Z_N = np.std(samples) / np.sqrt(N)
    relative_error = sigma_Z_N / p
    return sigma_Z_N, relative_error

# Example: Estimating for p = 0.5
p = 0.7
N = 1000
Z_N = monte_carlo_estimator(N, p)
sigma_Z_N, relative_error = calculate_std_and_relative_error(N, p)

print(f"Estimator Z(N): {Z_N}")
print(f"Standard Deviation Z(N): {sigma_Z_N}")
print(f"Relative Error Z(N)/p: {relative_error}")

# Ensure N is sufficient for relative error <= 10%
required_N = (1 - p) / (0.01 * p)
print(f"Required N for <= 10% relative error: {required_N}")
```

3 Conclusion

The required number of trials N to ensure the relative error $\frac{\sigma_{Z(N)}}{p}$ does not exceed 10% is given by:

$$N \geq \frac{1-p}{0.01p}$$

This ensures that the estimator $Z(N)$ is within 10% of the true value p.

Project 2: Monte Carlo Simulation of Signal Detection EEC 264 Term Project - Spring 2024

4 Detection of Known Signals

Consider the hypothesis testing problem:

$$H_0 : Y(t) = V(t)$$

$$H_1 : Y(t) = A\sin(2\pi t/T) + V(t)$$

where $V(t)$ is white Gaussian noise with variance $\sigma^2 = 1$, and $0 \leq t \leq T-1$ with $T = 100$. The two hypotheses are equally likely.

4.1 Probability of Error of the Minimum Probability of Error Detector

Objective: To obtain an expression for the probability of error P_E of the minimum probability of error detector.

Derivation: The likelihood ratio test for Gaussian noise is:

$$Ł = \frac{f_{Y|H_1}(Y)}{f_{Y|H_0}(Y)}$$

Given that $V(t)$ is white Gaussian noise with variance $\sigma^2 = 1$:

$$Ł = \frac{\exp\left(-\frac{1}{2}\sum_{t=0}^{T-1}(Y(t) - A\sin(2\pi t/T))^2\right)}{\exp\left(-\frac{1}{2}\sum_{t=0}^{T-1}Y(t)^2\right)}$$

Simplifying, the decision rule becomes:

$$\sum_{t=0}^{T-1} Y(t)\sin(2\pi t/T) \underset{H_0}{\overset{H_1}{}} \frac{A}{2}$$

The probability of error P_E can be expressed using the Q-function:

$$P_E = Q\left(\frac{A\sqrt{T}}{2}\right)$$

4.2 Values of A for Specific P_E

Objective: To find values of A that result in $P_E = 0.1$ and $P_E = 0.01$, and compare these values to empirical values obtained by Monte Carlo simulation.

Calculation:

Using the inverse Q-function:

$$A = \frac{2Q^{-1}(P_E)}{\sqrt{T}}$$

```
# Parameters for colored noise
alpha = 0.8
q = 1 - alpha**2

# Generate colored noise
def generate_colored_noise(T, alpha, q):
    V = np.zeros(T)
    N = np.random.normal(0, np.sqrt(q), T)
    for t in range(1, T):
        V[t] = alpha * V[t-1] + N[t]
    return V

# Monte Carlo simulation for colored noise detection
def monte_carlo_colored_detection(T, A, alpha, q, num_trials=10000):
    errors = 0
    for _ in range(num_trials):
        V = generate_colored_noise(T, alpha, q)
        Y = np.sin(2 * np.pi * np.arange(T) / T) + V
        decision = np.sum(Y * np.sin(2 * np.pi * np.arange(T) / T)) > A / 2
        if not decision:
            errors += 1
    P_E_est = errors / num_trials
    return P_E_est

# Estimate probability of error for colored noise
P_E_colored = monte_carlo_colored_detection(T, A1, alpha, q)
print(f"Estimated P_E for colored noise: {P_E_colored}")
```