Solutions to Homework 1

1 Solution to Problem 1

(a) Since we know $U_i = Z_i/m$ for all $i \geq 0$, the inequality $U_{i-2} < U_i < U_{i-1}$ is equivalent to $Z_{i-2} < Z_i < Z_{i-1}$. Given this Fibonacci generator, $Z_i \in \{0, 1, ..., m-1\}$ for all $i \geq 0$, which implies that

$$Z_i = \begin{cases} Z_{i-1} + Z_{i-2} & \text{if } Z_{i-1} + Z_{i-2} < m; \\ Z_{i-1} + Z_{i-2} - m & \text{otherwise.} \end{cases}$$

Thus, if $Z_{i-1} + Z_{i-2} < m$, $Z_i = Z_{i-1} + Z_{i-2}$ and the inequality $Z_{i-2} < Z_i < Z_{i-1}$ implies that $Z_{i-2} < 0$, which forms a contradiction. Moreover, if $Z_{i-1} + Z_{i-2} \geq m$, $Z_i = Z_{i-1} + Z_{i-2} - m$ and $Z_{i-2} < Z_i < Z_{i-1}$ implies that $Z_{i-1} > m$, which forms another contradiction. In conclusion, this generator can never produce three consecutive output values satisfying the inequality $U_{i-2} < U_i < U_{i-1}$.

(b) For a "perfect" random-number generator, U_{i-2}, U_{i-1}, $U_i \overset{\text{i.i.d.}}{\sim} U(0,1)$. Therefore,

$$\begin{aligned} P(U_{i-2} < U_i < U_{i-1}) &= \int_0^1 \int_{u_{i-2}}^1 \int_{u_{i-2}}^{u_{i-1}} 1^3 \, du_i \, du_{i-1} \, du_{i-2} \\ &= \int_0^1 \int_{u_{i-2}}^1 (u_{i-1} - u_{i-2}) \, du_{i-1} \, du_{i-2} \\ &= \int_0^1 \frac{1}{2}(u_{i-2} - 1)^2 \, du_{i-2} = \frac{1}{6}. \end{aligned}$$

2 Solution to Problem 2

(a) The CDF of the distribution is

$$F(x) = \int_0^x \frac{e^y}{e-1} dy = \frac{e^x - 1}{e-1}, \quad 0 \leq x \leq 1.$$

Thus, $x = F^{-1}(u) = \log\big(1 + (e-1)u\big)$, and we have the following inverse method algorithm:

0. Input: N (sample size);
1. Generate $U_1, ..., U_N \overset{\text{i.i.d.}}{\sim} U(0,1)$;
2. For $i = 1, ..., N$:
$$X_i = \log\big(1 + (e-1)U_i\big);$$
3. Output: $X_1, ..., X_N$.

(b) The CDF of the distribution is

$$\begin{aligned} F(x) &= \frac{1}{\pi}\int_{-\infty}^{x} \frac{1}{1+(y-1)^2}dy \\ &= \frac{1}{\pi}\int_{-\pi/2}^{\arctan(x-1)} \frac{1}{1+\tan^2\theta}\cdot\frac{1}{\cos^2\theta}d\theta \qquad (y-1=\tan\theta) \\ &= \frac{1}{\pi}\arctan(x-1)+\frac{1}{2}, \quad x\in\mathbb{R}. \end{aligned}$$

Thus, $x = F^{-1}(u) = \tan(\pi u - \pi/2) + 1$, and we have the following inverse method algorithm:

0. Input: N (sample size);
1. Generate $U_1, ..., U_N \overset{\text{i.i.d.}}{\sim} U(0,1)$;
2. For $i = 1, ..., N$:
$$X_i = \tan(\pi U_i - \pi/2) + 1;$$
3. Output: $X_1, ..., X_N$.

(c) The CDF of the distribution is

$$F(x) = \int_2^x f(y)dy = \begin{cases} \frac{(x-2)^2}{4} & \text{if } 2 \le x \le 3; \\ -\frac{1}{12}x^2 + x - 2 & \text{if } 3 \le x \le 6. \end{cases}$$

Thus, the inverse CDF is of form

$$x = F^{-1}(u) = \begin{cases} 2 + 2\sqrt{u} & \text{if } 0 \le u \le 1/4; \\ 6 - \sqrt{12(1-u)} & \text{if } 1/4 \le u \le 1. \end{cases}$$

Moreover, we have the following inverse method algorithm:

0. Input: N (sample size);
1. Generate $U_1, ..., U_N \overset{\text{i.i.d.}}{\sim} U(0,1)$;
2. For $i = 1, ..., N$:
$$X_i = \begin{cases} 2 + 2\sqrt{U_i} & \text{if } 0 \le U_i \le 1/4; \\ 6 - \sqrt{12(1-U_i)} & \text{if } 3/4 \le U_i \le 1; \end{cases}$$
3. Output: $X_1, ..., X_N$.

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import math
import time
np.random.seed(1)
```

#3 Gnerating Poisson Random Variables

(a) Implement the inverse method and the ad-hoc method

(i) Let X and $F(\cdot)$ respectively denote the random variable and CDF of the Poisson distribution with mean λ. It would be difficult to explicitly formulate $F^{-1}(\cdot)$. However, given $0 \leq u \leq 1$, we can calculate F^{-1} as follows

$$F^{-1}(u) = \inf_{k \geq 0} \left\{ e^{-\lambda} \sum_{i=0}^{k-1} \frac{\lambda^i}{i!} < u \leq e^{-\lambda} \sum_{i=0}^{k} \frac{\lambda^i}{i!} \right\}$$

One can use a list to keep record of quantiles of the Poisson distribution to reduce the computation time. The implementation is in the function **generate_poisson_inverse**

(ii) The idea of the ad-hoc method is explained on page 10 in the lecture slide *generatingrv*. The code is implemented in the function **generate_poisson_adhoc**

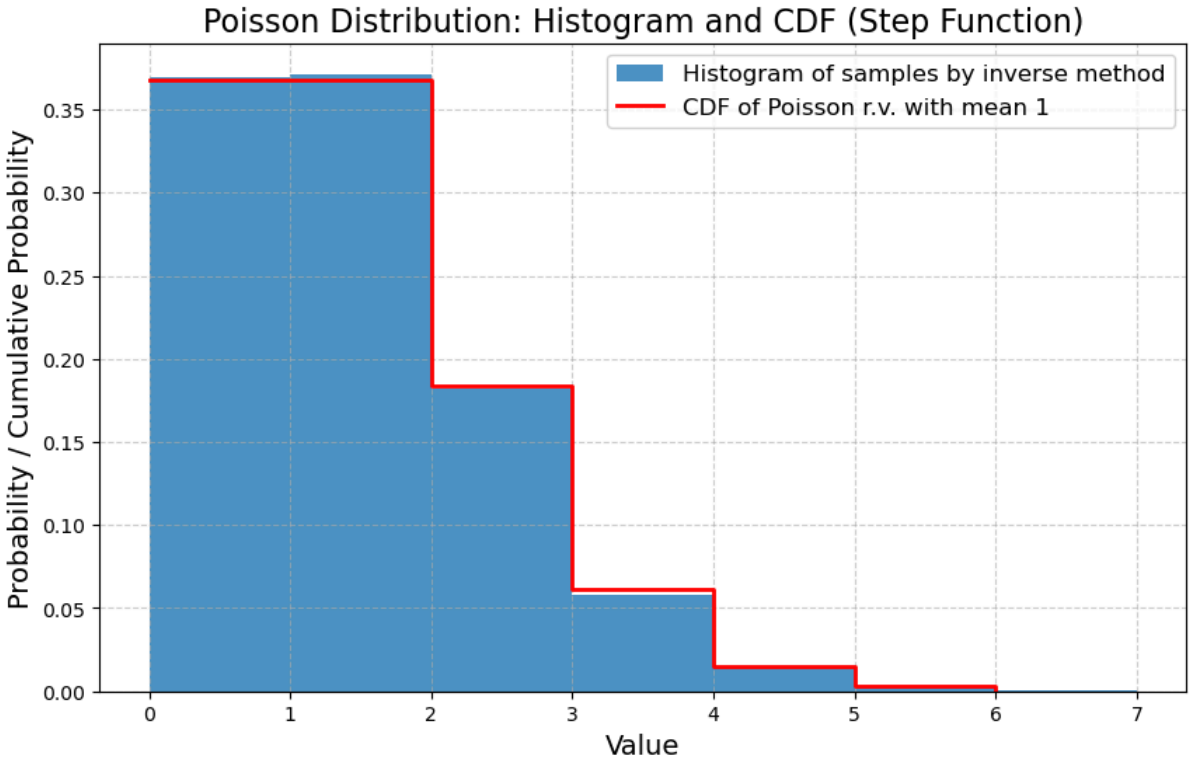
In [2]:
```
mass_poisson = lambda lam, k : math.exp(-lam) * math.pow(lam, k) / math.factorial(k)
# A function that calculates the probability when a poisson r.v. with mean lam equals to k

def generate_poisson_inverse(lam, n):
    # lam is the mean of the poisson distribution, n is the number of samples
    # this function returns a list of n samples
    qt = []
    qt.append(mass_poisson(lam, 0))
    l = len(qt)
    samples = []
    for i in range(n):
        u = np.random.uniform(0,1)
        if (u>qt[-1]):
            while u > qt[-1]:
                temp = qt[-1] + mass_poisson(lam, l)
                qt.append(temp)
                l += 1
            samples.append(l-1)
        else:
            k=0
            while(qt[k]<u):
                k+=1
            samples.append(k)
    return samples

inv_cdf_exp = lambda mu, u : -(1/mu * math.log(1-u))
# The inverse function of the CDF of the exponential distribution with rate mu
def generate_poisson_adhoc(lam, n):
    samples = []
    for i in range(n):
        k=0; s=0
        while(s < lam):
            u = np.random.uniform(0,1)
            s += inv_cdf_exp(1, u)
            # s+= np.random.exponential(1) # if one use this, it is acceptable, and it would be much faster
            k += 1
        samples.append(k-1)
    return samples
```

(b) Compare the simulated date with the standard Poisson distribution

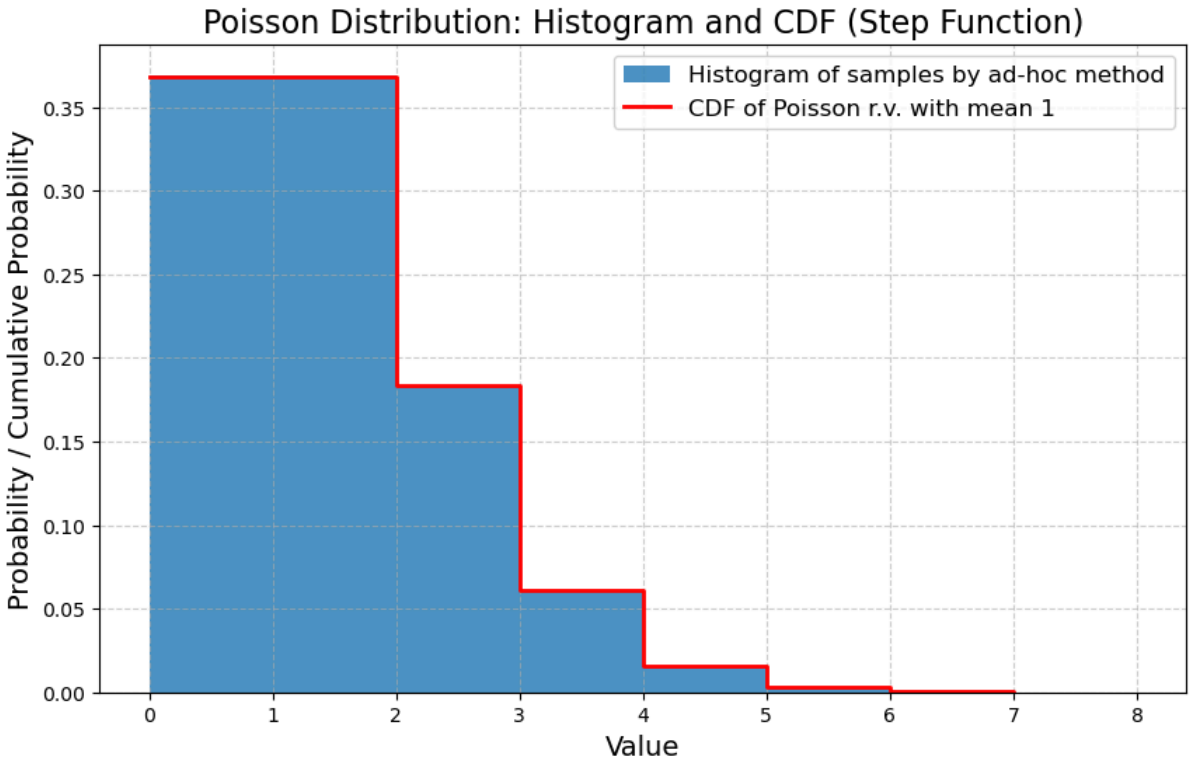
In [3]:
```
def plot(samples, method):
    plt.figure(figsize=(10, 6))
    hist_values, bins, _ = plt.hist(samples, bins=np.arange(0, max(samples) + 2),
                                    density=True, alpha=0.8, label='Histogram of samples by '+method+' method')
    x = np.arange(0, max(samples) + 1)
    cdf = [mass_poisson(1,i) for i in x]

    # Plot the CDF as a step function
    plt.step(x, cdf, where='post', color='red', linewidth=2, label='CDF of Poisson r.v. with mean 1')

    # Add labels, title, and legend
    plt.title('Poisson Distribution: Histogram and CDF (Step Function)', fontsize=16)
    plt.xlabel('Value', fontsize=14)
    plt.ylabel('Probability / Cumulative Probability', fontsize=14)
```

```
    plt.legend(fontsize=12)
    plt.grid(True, linestyle='--', alpha=0.6)

    # Show the plot
    plt.show()
```

In [4]:
```
# The distribution of samples generateb by the inverse method.
samples1 = generate_poisson_inverse(1,10000)
plot(samples1, 'inverse')
```



In [5]:
```
# The distribution of samples generateb by the ad-hoc method.
samples1 = generate_poisson_adhoc(1,10000)
plot(samples1, 'ad-hoc')
```



(c) Compare the efficiency of the above two methods with **np.random.poisson()**

In [6]:
```
num = 100000
time_inverse = time.time()
```

```
generate_poisson_inverse(1, num)
time_inverse = time.time() - time_inverse

time_adhoc = time.time()
generate_poisson_adhoc(1, num)
time_adhoc = time.time() - time_adhoc

time_np = time.time()
np.random.poisson(1, num)
time_np = time.time() - time_np

print("The inverse method uses %.4f seconds to genertate 100000 samples." % time_inverse)
print("The ad-hoc method uses %.4f seconds to genertate 100000 samples." % time_adhoc)
print("The np.random.poisson() method uses %.4f seconds to genertate 100000 samples." % time_np)
```

```
The inverse method uses 0.1239 seconds to genertate 100000 samples.
The ad-hoc method uses 0.2790 seconds to genertate 100000 samples.
The np.random.poisson() method uses 0.0020 seconds to genertate 100000 samples.
```

It is worth mentioning that if one use the **np.random.exponential()** to generate the exponential sample in the ad-hoc method, the method would be much faster, and it may take around 0.08 seconds.

#4 Sampling from Posterior Distribution

Consider a Bayesian statistic model in which the data are i.i.d. Poisson random variables with mean θ. Suppose the prior of the parameter $\theta > 0$ is a $\mathrm{Gamma}(\alpha, \beta)$ with $\alpha \geq 1, \beta > 0$ and its PDF is

$$f_0(\theta) = \frac{\beta^\alpha}{\Gamma(\alpha)} \cdot \theta^{\alpha-1} e^{-\beta\theta}, \quad \theta > 0.$$

(c) Implement the A-R algorithm. It should take the parameters (α, β) of the prior distribution and the i.i.d. data samples as input by the users. Illustrate the efficacy of your implementation using any examples you like and report the results.

Answer

(a) Derive the posterior distribution $f_1(\theta)$ given i.i.d. data $X_1, \ldots, X_n$.

The posterior distribution $f_1(\theta)$ is given by

$$\begin{aligned} f_1(\theta) := f(\theta \mid X_1, \ldots, X_n) &= \frac{f(\theta, X_1, \ldots, X_n)}{f(X_1, \ldots, X_n)} \\ &\propto f_0(\theta) \prod_{i=1}^n f(X_i \mid \theta) \propto \theta^{\alpha-1} e^{-\beta\theta} \prod_{i=1}^n \theta^{X_i} e^{-\theta} \\ &= \theta^{\alpha + \sum_{i=1}^n X_i - 1} e^{-(\beta+n)\theta}, \end{aligned}$$

where in the second line we only retain the terms that are dependent on θ.

Therefore, the posterior distribution is a $\mathrm{Gamma}(\alpha + \sum_{i=1}^n X_i, \beta + n)$.

(b) Design an A-R algorithm to sample from the posterior distribution $f_1(\theta)$ and write down the pseudo codes.

For the posterior distribution $f_1(\theta)$, we have known $X_1, \ldots, X_n$, so this problem is equivalent to sampling from a Gamma distribution by using the A-R algorithm. So we will derive the algorithm how to use A-R to sample Gamma(α, β) with $\alpha \geq 1$ and $\beta > 0$.

If we use exponential distribution with rate μ as the proposed distribution, then

$$c^*(\mu) = \max_{\theta \geq 0} \frac{\mathrm{Gamma}(\theta; \alpha, \beta)}{\exp(\theta; \mu)} = \max_{\theta \geq 0} \frac{\frac{\beta^\alpha}{\Gamma(\alpha)} \cdot \theta^{\alpha-1} \exp(-\beta\theta)}{\mu \exp(-\mu\theta)} = \max_{\theta \geq 0} \frac{\beta^\alpha}{\Gamma(\alpha)\mu} \theta^{\alpha-1} \exp(-\beta\theta + \mu\theta).$$

Note that if $\mu < \beta$, then $c^*(\mu)$ is bounded. The above optimal solution is (you can take a log and then set its derivative as 0)

$$\theta^*(\mu) = \frac{\alpha - 1}{\beta - \mu}.$$

And hence,

$$c^*(\mu) = \frac{\beta^\alpha}{\Gamma(\alpha)\mu} \left(\frac{\alpha - 1}{\beta - \mu} \right)^{\alpha-1} \exp(-(\alpha - 1))$$

Therefore, the best choice of μ is

$$\mu^* = \text{argmin}_{0<\mu<\beta} \frac{\beta^\alpha}{\Gamma(\alpha)\mu}\left(\frac{\alpha-1}{\beta-\mu}\right)^{\alpha-1}\exp(-(\alpha-1)) \overset{\text{take log}}{=} \text{argmin}_{0<\mu<\beta} -\log(\mu) - (\alpha-1)\log(\beta-\mu) = \beta/\alpha$$

with

$$c^*(\mu^*) = \frac{\alpha^\alpha}{\Gamma(\alpha)}\exp(-(\alpha-1)).$$

Algorithm of sampling Gamma(α, β)

1. Generate $\theta \sim \exp(\beta/\alpha)$.
2. Generate $U \sim \text{Uniform}(0,1)$.
3. If $U \leq f(\theta)/(c^* g(\theta)) = (\beta\theta/\alpha)^{\alpha-1}\exp((\alpha-1)(1-\beta\theta/\alpha))$, return X;
4. Otherwise, go back to step 1.

(c) Implement the A-R algorithm.

In [7]:
```
# The code
failure_num_gamma = 0
def AR_Gamma(alpha, beta):
    theta = -np.log(np.random.random()) / (beta / alpha) # step 1
    U = np.random.random()
    r = beta * theta / alpha
    if U <= np.exp((alpha-1)*(np.log(r)+ 1-r)):
        return theta
    else:
        global failure_num_gamma
        failure_num_gamma += 1
        return AR_Gamma(alpha, beta)
```

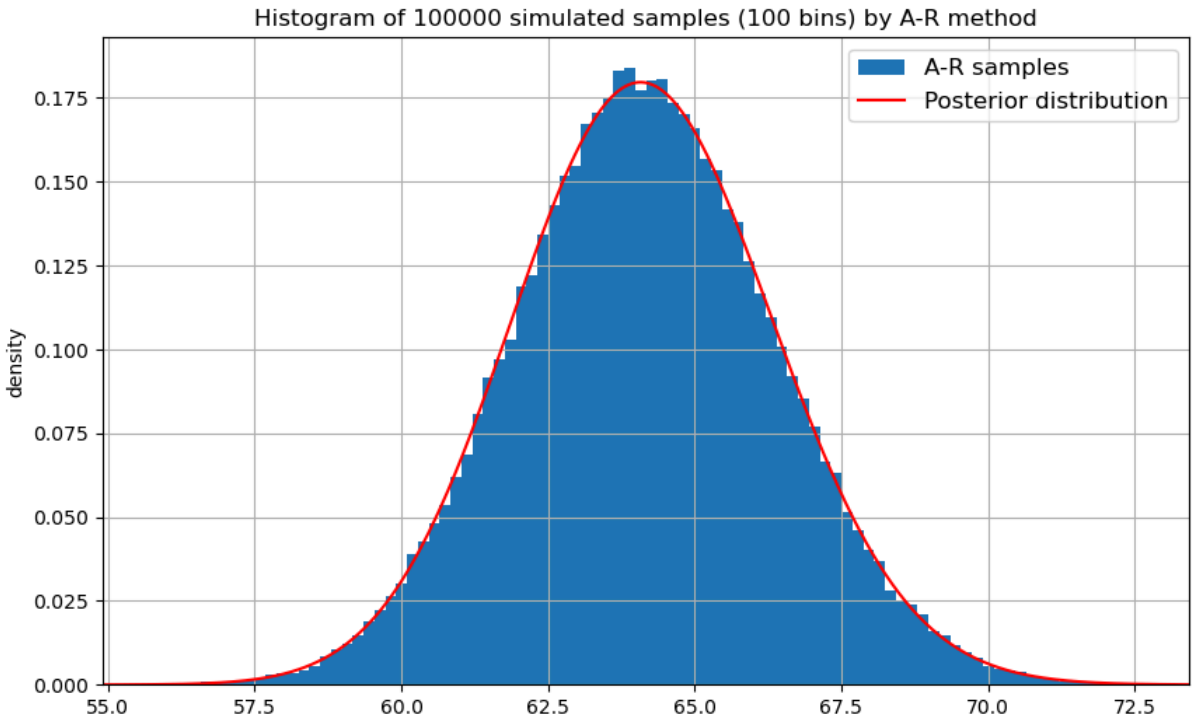
In [8]:
```
# Input
alpha, beta = 200, 3
theta = AR_Gamma(alpha, beta)
N = 10
X = np.random.poisson(lam=theta, size=N)
print(f"Inputs: alpha={alpha}, beta={beta}, theta={theta}, X={X}" )
```

```
Inputs: alpha=200, beta=3, theta=63.75027196342887, X=[67 70 61 65 68 63 58 63 65 54]
```

In [9]:
```
alpha_post = alpha + X.sum()
beta_post = beta + N
failure_num_gamma = 0
samples = [AR_Gamma(alpha_post, beta_post) for i in range(100000)]
print('Theoretical efficiency: ', 1/np.exp(alpha_post * np.log(alpha_post) - math.lgamma(alpha_post)- (alpha_post-1)))
print('Practical efficiency:',100000/(100000+failure_num_gamma))
```

```
Theoretical efficiency:  0.0319341838812487
Practical efficiency: 0.031976035879670336
```

In [10]:
```
plt.figure(figsize=(10, 6))
plt.hist(samples, bins=100, density=True, label='A-R samples')
x = np.linspace(np.min(samples), np.max(samples), 1000)
pdf = np.exp((alpha_post-1)*np.log(x) - beta_post*x + alpha_post*np.log(beta_post) - math.lgamma(alpha_post))
plt.plot(x,pdf,color='r', label='Posterior distribution')
plt.grid()
plt.xlim(np.min(samples), np.max(samples))
plt.ylabel('density')
plt.title('Histogram of {} simulated samples (100 bins) by A-R method'.format(100000))
plt.legend(fontsize=12)
plt.show()
```


Histogram of 100000 simulated samples (100 bins) by A-R method
A-R samples
Posterior distribution
density
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
55.0
57.5
60.0
62.5
65.0
67.5
70.0
72.5

In [ ]: