The Chinese University of Hong Kong, Shenzhen
DDA4002 2024-2025 Spring

Solutions to Assignment 1

Yinqi (Marcus) Zhong
121090847

February 12, 2025

1 Theoretic Questions

1.1 Quality of Random Number Generator

a

Since $U_i = \frac{Z_i}{m}$ 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 RNG $Z_i = (Z_{i-1} + Z_{i-2})\ mod\ m$, the recurrence turns out that Z_i is directly dependent on the sum of Z_{i-1} and Z_{i-2}, which would be:

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

Thereforeif $Z_{i-2} < Z_i < Z_{i-1}$, when $Z_{i-1} + Z_{i-2} < m$, $Z_{i-2} < 0$, which is contradictory. When $Z_{i-1} + Z_{i-2} > m$, Z_{i-1} would be larger than m, which is also contradictory.

So, for the arrangement of 3 consecutive output would never be $U_{i-2} < U_i < U_{i-1}$, but it would be $U_{i-2} < U_{i-1} < U_i$, $\forall i > 1$.

b

A **perfect random-number generator** is the case where the numbers generated are independent and uniformly distributed. $U_{i-2}, U_{i-1}, U_i \overset{iid}{\sim} U(0,1)$.

This means, for any 3 consecutive numbers, their probability of any particular order should be equally likely. So, the probability of $U_{i-2} < U_i < U_{i-1}$ would be:

$$p_{(a)} = \frac{1}{A_1^3} = \frac{1}{3!} = \frac{1}{6}$$

1.2 Inverse Method to Generate Random Variables

The general step of the Inverse Method is as follows:

1. Find the CDF $F(x)$: the integral of the PDF, which is $F(x) = P(X \leq x)$
2. Solve x as a uniform distributed random variable U: $F(x) = U$, which is $F^{-1}(u) \equiv inf\{x : F(x) \geq u\}$

a

$$f(x) = \frac{e^x}{e-1}, 0 \leq x \leq 1$$

$$F(x) = \int_0^x \frac{e^t}{e-1} dt = \frac{1}{e-1} \cdot [e^t]_0^x = \frac{e^x - 1}{e-1}$$

Set $F(x) = U$, where U is uniformly distributed on [0,1]:

$$\frac{e^x - 1}{e-1} = U$$

$$e^x = 1 + (e-1)U$$

$$x = \ln(1 + (e-1)U)$$

The inverse method algorithm would be:

1. Input N
2. Generate $U_1, ..., U_N \overset{iid}{\sim} U(0,1)$
3. Generate random variables for i = 1, ..., N:

$$X_i = \ln(1 + (e-1)U_i)$$

4. Output: $X_1, ..., X_N$

b

$$f(x) = \left\{\pi \left[1 + (x-1)^2\right]\right\}^{-1}$$

Since the integral of $\frac{1}{x^2+1}$ is $arctangent(x)$, the CDF of $F(x)$ is:

$$F(x) = \int_{-\infty}^x \left\{\pi \left[1 + (x-1)^2\right]\right\}^{-1} = \frac{1}{\pi} \cdot \int_{-\infty}^x \frac{1}{1 + (t-1)^2} dt = \frac{1}{\pi} \cdot [arctan(k)]_{-\infty}^{x-1}$$

Since $arctan(k) \in (-\frac{\pi}{2}, \frac{\pi}{2})$,

$$F(x) = \frac{1}{\pi}[arctan(x-1) - (-\frac{\pi}{2})] = \frac{1}{\pi} \cdot arctan(x-1) + \frac{1}{2}$$

Set $F(x) = U$, where U is uniformly distributed on [0,1]:

$$U = F(x) = \frac{1}{\pi} \cdot arctan(x-1) + \frac{1}{2}$$

$$arctan(x-1) = \pi \cdot (U - \frac{1}{2})$$

Take tangent of both side, we get:

$$x = tan(\pi \cdot (y - \frac{1}{2})) + 1$$

The inverse method algorithm would be:

1. Input N
2. Generate $U_1, ..., U_N \overset{iid}{\sim} U(0,1)$
3. Generate random variables for i = 1, ..., N:

$$X_i = tan(\pi \cdot (U_i - \frac{1}{2})) + 1$$

4. Output: $X_1, ..., X_N$

c

$$f(x) = \begin{cases} \frac{x-2}{2} & \text{if } 2 \le x \le 3, \\ 1 - \frac{x}{6} & \text{if } 3 \le x \le 6. \end{cases}$$

For $2 \le x \le 3$,

$$F(x) = \int_2^x \frac{t-2}{2} dt = \frac{(x-2)^2}{4}$$

For $3 \le x \le 6$,

$$F(x) = F(3) + \int_3^x (1 - \frac{t}{6}) dt = -2 + x - \frac{x^2}{12}$$

So, the CDF $F(X)$ would be

$$F(x) = \begin{cases} \frac{(x-2)^2}{4} & \text{if } 2 \le x \le 3, \\ -2 + x - \frac{x^2}{12} & \text{if } 3 \le x \le 6. \end{cases}$$

Set $F(x) = U$, where U is uniformly distributed on [0,1]:

For $2 \le x \le 3$,

$$U = \frac{(x-2)^2}{4}$$

$$x = 2 + 2\sqrt{U}, \text{for } U \in [0, \frac{1}{4}]$$

For $3 \le x \le 6$,

$$U = -2 + x - \frac{x^2}{12}$$

$$-\frac{x^2}{12} + x - (U+2) = 0$$

$$x = \frac{12 \pm \sqrt{48 - 48U}}{2}$$

Because $x \le 6$

$$x = 6 - \sqrt{12 - 12U}, \text{for } U \in [\frac{1}{4}, 1]$$

The inverse method algorithm would be:

1. Input N
2. Generate $U_1, ..., U_N \overset{iid}{\sim} U(0,1)$
3. Generate random variables for i = 1, ..., N:

$$X_i = \begin{cases} 2 + 2\sqrt{U_i} & \text{if } U_i \in [0, \frac{1}{4}], \\ 6 - 2\sqrt{3 - 3U_i} & \text{if } U_i \in [\frac{1}{4}, 1]. \end{cases}$$

4. Output: $X_1, ..., X_N$

2 Numeric Experiments

2.1 Generating Poisson Random Variables

a

Inverse method: The CDF of a Poisson distribution with mean λ is

$$F(k) = \sum_{n=0}^{k} \frac{\lambda^n \cdot e^{-\lambda}}{n!}$$

In python, generating a Poisson RV, the procedure is:

- Generate uniform RV : $U \in [0, 1]$
- Find minimal k that satisfies $F(k) \geq U$

Ad-Hoc Method: Poisson distribution can be derived from the sum of exponentially distributed inter-arrival times. For Poisson Distribution with rate λ, inter-arrival times follow exponential distribution with mean $\frac{1}{\lambda}$.

In python, may generate exponential random variables and sum them until the total exceeds t, get the count of generation times as k.

b

Generating 10000 samples from Poisson distribution with mean 1 using the 3 methods respectively. The graph comparison is as follow.

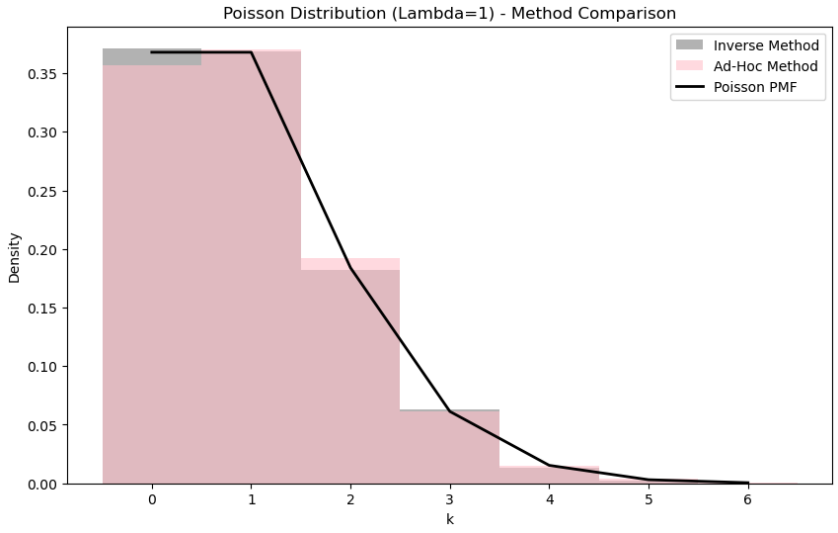


Figure 1: Poisson Distribution (Lambda=1) - Method Comparison

c

The output of the calculation time is:

Inverse Method Time: 0.41811 seconds

Ad-Hoc Method Time: 0.08483 seconds

Numpy Method Time: 0.00290 seconds

As observed, Numpy method computes fastest, then adhoc method. Inverse Method takes most time. The implementation **cannot** beat the numpy function.

2.2 Sampling from Posterior Distribution

The prior PDF:

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

a

Given $X1, ...Xn$ $X_i \overset{\text{i.i.d.}}{\sim} Poisson(\theta), \quad i = 1, \ldots, n$

The likelihood :

$$L(\theta \mid X_1, ...X_n) = \Pi_{i=1}^n \frac{\theta^{X_i} \cdot e^{-\theta}}{X_i!} \simeq \theta^{X_i} \cdot e^{-n\theta}$$

Then, the posterior distribution is calculated as proportional to the product of likelihood and prior:

$$f_1(\theta|X_1, \ldots, X_n) \propto L(\theta|X_1, \ldots, X_n) \cdot f_0(\theta) \propto \theta^{\sum X_i} \cdot e^{-n\theta} \cdot \theta^{\alpha-1} \cdot e^{-\beta\theta}$$

Which can be simplified as :

$$f_1(\theta|X_1, \ldots, X_n) \propto \theta^{\sum X_i + \alpha - 1} \cdot e^{-(n+\beta)\theta}$$

This is the form of a **Gamma Distribution**, and the updated parameters are:

- $\alpha' = \alpha + \sum X_i$
- $\beta' = \beta + n$

$$\theta|X_1, \ldots, X_n \sim \text{Gamma}(\alpha + \sum X_i, \beta + n)$$

b

In this AR Algorithm, the **target distribution** is $f(x) = f_1(\theta|X_1, \ldots, X_n)$, we need to define a **proposal distribution** needs to be defined, it satisfies $f(x) \leq c \cdot g(x), \forall x \in R, c > 1$.

The pseudo code for the AR algorithm would be:

1. Choose a Proposal Distribution: Let Gamma Distribution as the proposal, let's say it is $q(\theta) \sim Gamma(\alpha_{proposal}, \beta_{proposal})$
2. Generate $\theta^* \sim q(\theta)$
3. Independently draw a random number $U \sim unif(0, 1)$
4. If $U \leq f_1(\theta^*|X_1, \ldots, X_n)/c \cdot q(\theta^*)$ (c is a scaling factor to ensure that the ratio is always less than or equal to 1), stop and return X (Acceptance).
5. Otherwise, repeat the process back to step 2 (Rejection).

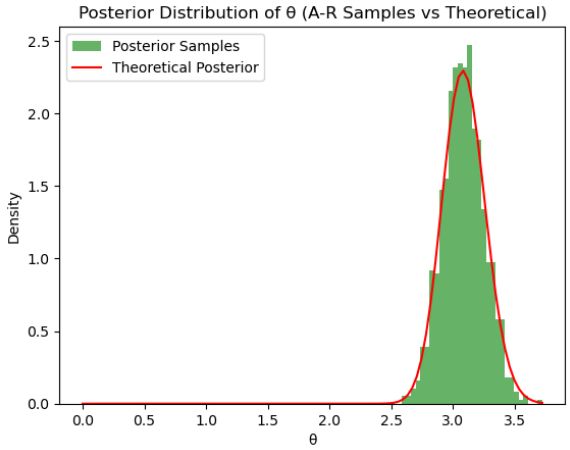
c

I implemented the A-R algorithm by python.

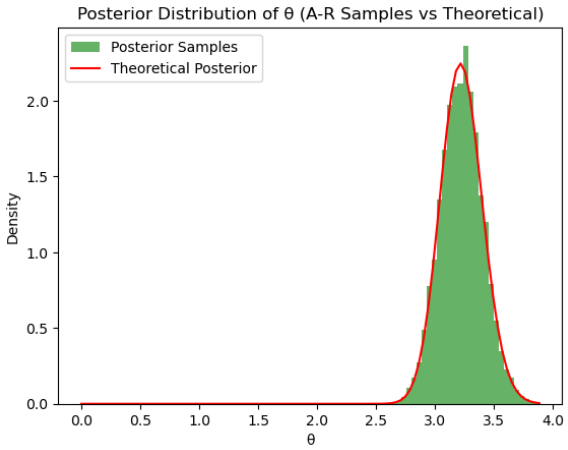
There are 4 parameters that can be manually adjusted to tset the efficacy of the implementation.

- prior α
- prior θ
- sample size of the posterior samples
- data input: The observed data samples. In this problem, I use numpy.random to generate data following **poisson distribution** with mean 3. The value of the mean, or the distribution can be adjusted.

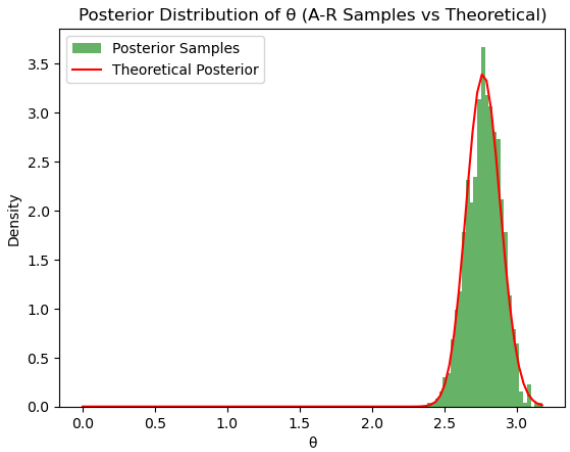
The comparison of the posterior distribution pdf and the samples histogram with different parameters are listed below:



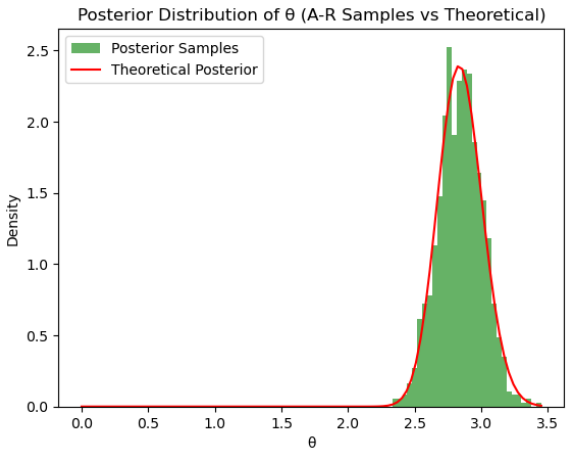
(a) Alpha = 2, Beta = 2, n = 1000



(b) Alpha = 2, Beta = 2, n = 10000



(c) Alpha = 2, Beta = 1, n = 1000



(d) Alpha = 1, Beta = 2, n = 1000

Figure 2: Comparison of different parameters

Obviously, the histogram of the posterior samples **closely matches the theoretical posterior**. The swift of the prior parameters does not impact on the accuracy of the sampling, the histograms and samples can all successfully match. Also, as the number of samples increases, the samples from the A-R algorithm seem to **converge** towards the true posterior, meaning that the algorithm is functioning as expected.