模擬與統計計算

HW3

N26120838 吳定洋

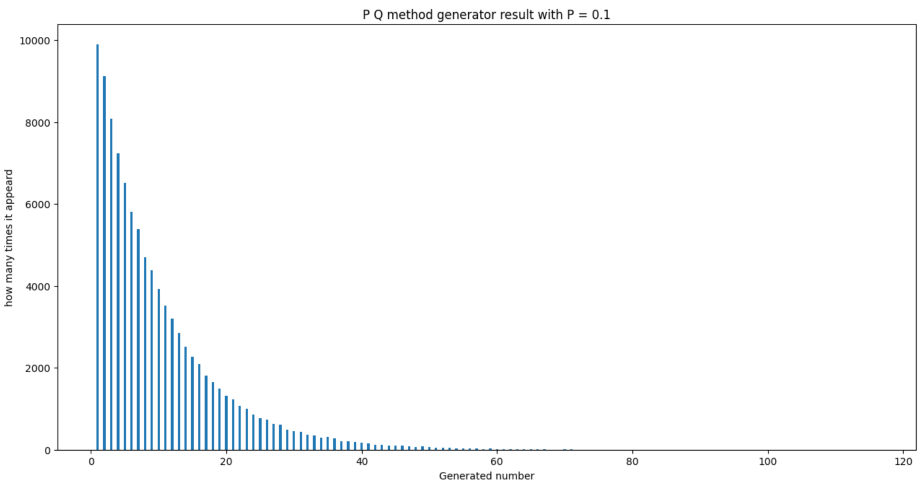
這個作業要我們分別寫三個程式，都是用來生成幾何隨機數的。

（一） p q 法

　　第一個是用 p q 的方法，先設定 p 然後不斷生成生成 uniform random number，計算種共生成幾次 uniform random number 才能小於 p。p 和生成數字次數都可以做更改。

```python
def geometric_random_generator_qp(p):
    trials = 0
    while True:
        trials += 1
        if random.random() <= p:
            return trials

p = 0.005  # Probability of success
result = []
for i in range(1000000):
    result.append(geometric_random_generator_qp(p))
```

　　首先我先測試了 p = 0.1 生成了 10K 個 geometric random number，其生成數字直方圖統計如下，x 軸是 geometric number，y 軸是該 x 在這 10K 次 trails 中產生的次數。



（二）traditional inverse transform 方法

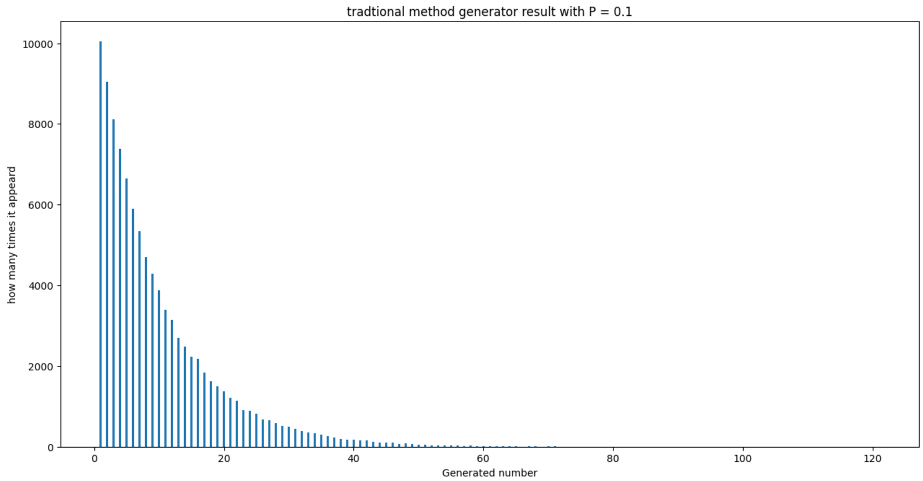
再來是 traditional inverse transform 的方法，一樣要設定一個成功機率 p，然後參照老師講義上公式實作

$$X = \begin{cases} x_0 & \text{If } U < p_0 \\ x_1 & \text{If } p_0 \le U < p_0 + p_1 \\ \vdots & \\ x_j & \text{If } \sum_{i=0}^{j-1} p_i \le U < \sum_{i=0}^{j} p_i \\ \vdots & \end{cases}$$

，在一邊測 random u 是否滿足對應的 geometric number 時，我同時把 $\sum_{i=0}^{j} p_i$ 都存到 all_round list 中，這樣我後續要使用時直接到 list 找就好，不用再重新跑一次上面那個公式，不然非常耗費時間，事實上這個方法的 time complexity 已經很高，後續會探討。

```python
def geometric_random_generator_traditional(p):
    u = random.random()  # Generate a uniform random number between 0 and 1
    q= 1-p
    trail = 0

    while True:
        trail += 1
        if 0 <= u < all_round[0]:
            return 1
        if len(all_round) <= trail:
            all_round.append(q * all_round[trail-1])

        if sum(all_round[0:trail-1]) <= u <sum(all_round[0:trail]):
            return trail



p = 0.005 # Probability of success
all_round = [p]

result = []
for i in range(1000000):
    result.append(geometric_random_generator_traditional(p))
```

一樣我先測試了 p = 0.1 生成了 10K 個 geometric random number，其生成數字直方圖統計如下，x 軸是 geometric number，y 軸是該 x 在這 10K 次 trails 中產生的次數。

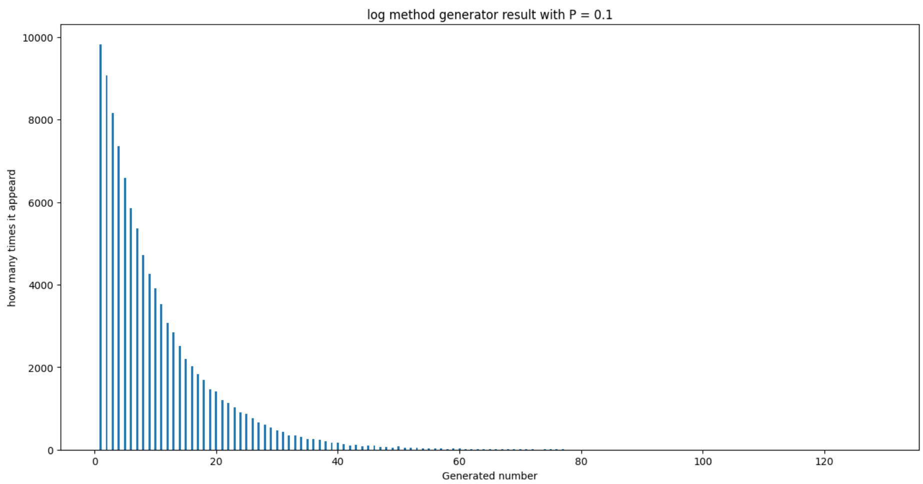


（三） X= Int(log(U)/log(q))+1 法

最後是 X= Int(log(U)/log(q))+1 的方法，直接套用推導出來 Geometric Random Variable 的公式來做。

```
def geometric_random_generator_log(p):
    u = random.random()  # Generate a uniform random number between 0 and 1
    q= 1-p
    x = math.ceil(math.log(u) / math.log(q))  # 實作inverse transform 公式 取math.log(u) / math.log(q)整數後+1
    return x


p = 0.005  # Probability of success
result = []
for i in range(1000000):
    result.append(geometric_random_generator_log(p))
```

一樣我先測試了 p = 0.1 生成了 10K 個 geometric random number，其生成數字直方圖統計如下，x 軸是 geometric number，y 軸是該 x 在這 10K 次 trails 中產生的次數。

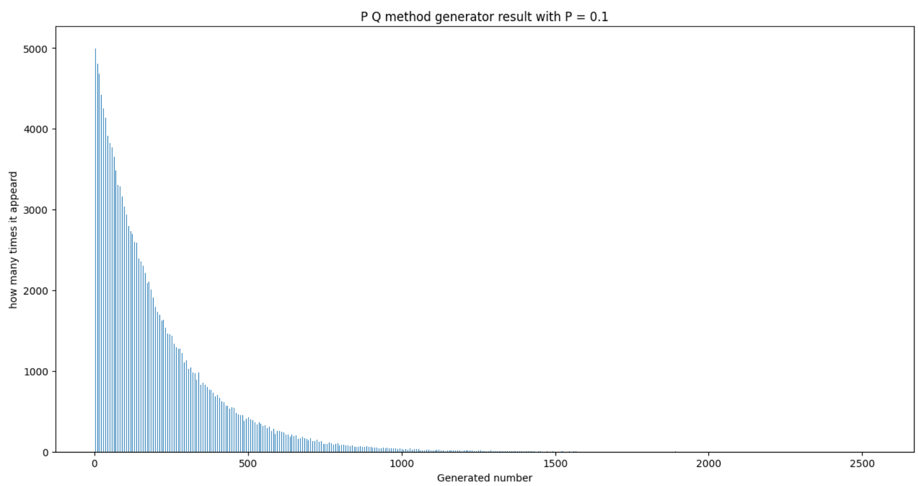


（四）時間複雜度探討

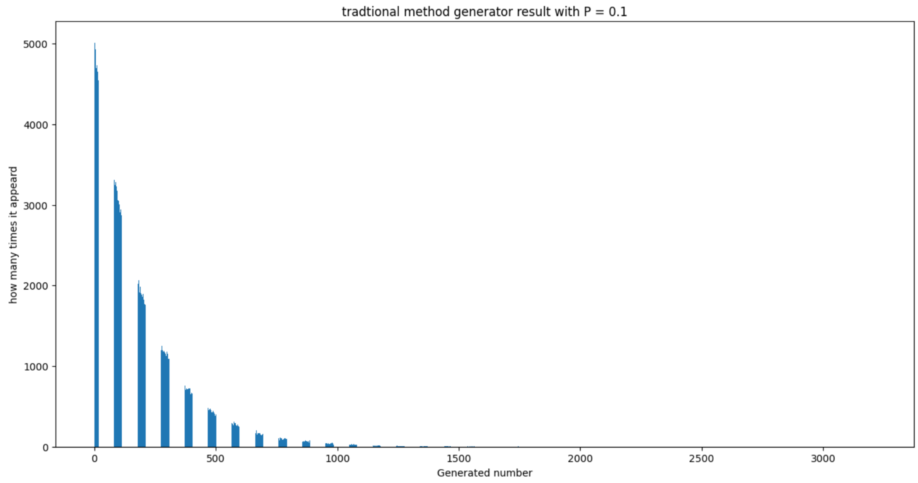
- 第一個 p q 法，我程式的時間複雜度，在 geometric_random_generator_qp(p)中 O(1/p)，p 為 constant，所以可以看成 O(1)，整個程式的時間複雜度取決於我要產生幾個 geometric number，所以 total 是 O(n*1)，n 為我產生 geometric number 的次數。
- 第二個 traditional inverse transform 方法，在一開始我 all_round list 中什麼都還沒計算，在 worst case 中我要計算 sum(all_round[0:trail-1]) <= u <sum(all_round[0:trail])時，我需要計算 trail 次，所以這邊 time complexity 是 O(trail)，整個程式我要取 n 個 geometric number 所以種共的 time complexity 為 O(max(trail)*n)。
- 第三個 X= Int(log(U)/log(q))+1 非常簡單，直接代公式的，而那些數學計算都是 O(1)，種共要得到 n 個 geometric number，所以 total time complexity 為 O(n)。

（五）直方圖探討

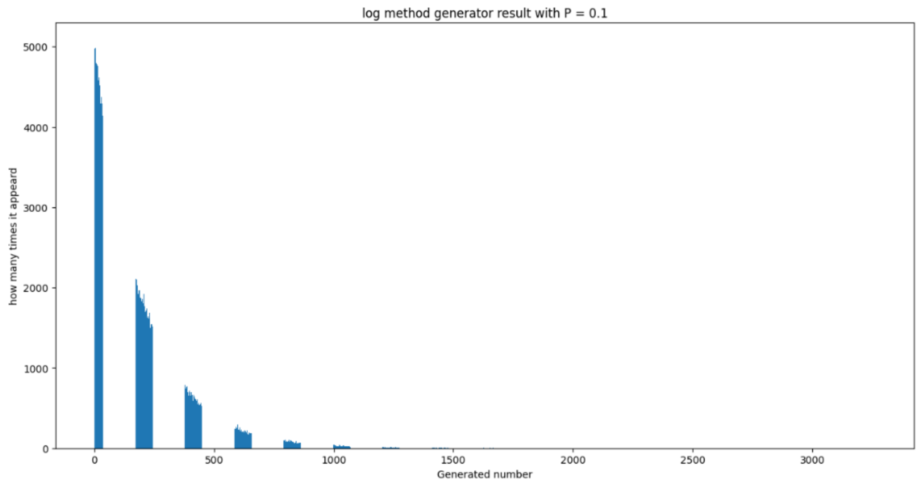
上面用到的三個 geometric random number generator 老師在上課中有從 p q 法逐步推導到 Int(log(U)/log(q))+1 方法，所以直方圖模擬出來幾乎都一模一樣，基於好奇我又用這 3 個程式分別模擬了 p = 0.005 取樣 100 萬次的結果。圖上的標題可以忽略，因為那是在 python 中產生的圖，我忘記改標題名稱了，而後面兩張圖應該是因為 p 太小，導致 geometric number 太多，所以圖片整個顯示不出來，其實結果是 x 是連續的整數。



圖表 1 p q 法 p = 0.005 取樣 100 萬次



圖表 2 傳統法 p = 0.005 取樣 100 萬次



圖表 3 log 法 p = 0.005 取樣 100 萬次

根據這些圖片的猜測，這應該是老師課程在鋪墊 **binomial distribution**。