U2267 Probability: Simulation Project

系級:通訊二　　姓名:鐘婉庭　　ID:411186028

執行:Windows11 並使用線上編譯器 ideone.com 做測試

Problem 1:

內容:用提供之 rand function 程式測試 uniform random variable 的累積性質是否與理想一樣。

已知 $f_X(\chi)$是符合 uniform(0,1)的隨機變數，而是 $F_X(\chi)$是符合 uniform(0,1)隨機變數的 CDF，根據定義可得:

$$f_x(x) = \begin{cases} \frac{1}{b-a} = \frac{1}{1-0} = 1 & , \quad for \quad x > 0 \\ 0 & , \quad else \end{cases}$$

$$F_x(x) = \int f_x(x)\ dx = \begin{cases} x & , \quad for \quad x > 0 \\ 0 & , \quad else \end{cases}$$

χ	0.2	0.4	0.6	0.8	0.95
$\hat{F}(\chi)$	0.197000	0.394900	0.603200	0.801900	0.949900
$F_{/X}(\chi)$	0.201090	0.400740	0.598630	0.799600	0.950470
$F_X(\chi)$	0.2	0.4	0.6	0.8	0.95

結果討論:

$\hat{F}(x)$為使用 10000 個樣本測得的數據，跑出的結果為在樣本數中有多少比例的隨機變數小於輸入值。而我想看看如果改為更多樣本數據是否會更接近理想值，因此多設了一個隨機產生 100000 個數字的 $F_{/X}(x)$，也發現當數據增加確實有更接近理想值了。

Problem 2:

內容:用提供之 main function 程式與自己寫的 gen_exp function 測試 exponential(lambda) random variable 的累積性質是否與理想一樣。其中，$F_X(\chi)$是 exponential(1)的 CDF。

根據定義可得:

$$f_x(x) = \begin{cases} \lambda e^{-\lambda x} & , \quad for \quad x > 0 \\ 0 & , \quad else \end{cases}$$

$$F_x(x) = \int_0^x f_t(t)\ dt = \int_0^x \lambda e^{-\lambda t}\ dt = \begin{cases} 1 - e^{-\lambda x} & , \quad for \quad x > 0 \\ 0 & , \quad else \end{cases}$$

推導過程(圖一計算過程):

$$F_X(x) = S = 1 - e^{-\lambda x}$$

$$1 - S = e^{-\lambda x}$$

$$\ln(1-S) = -\lambda x \ln e$$

$$x = \frac{\ln(1-S)}{-\lambda} = \frac{\log(1-S)}{-\lambda}$$

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>

double gen_exp_411186028(double lambda){

    double S =((double)rand()/((double)RAND_MAX+1.0));
    return -log(1- S) / lambda;
  }

```

圖二(function 部分)

結果討論:

前面與問題一相同,rand 會生成一個在[0, RAND_MAX]範圍內的隨機整數,並轉成 double 型態除以(RAND_MAX + 1.0),將整數變成 0~1 的範圍。Function 中的隨機數 e 用來計算指數分布,lambda 之值在 main function 中做設定。寫完後發現無法跑,原因是因為我們使用的數學函數 log()尚未成功引用,因此要在程式的最開始部分加入 #include <math.h>,以確保使用到 log 函數的地方能夠正確編譯。而在測試中也發現,因為隨機變數的關係,每次的數據皆不同,因此下表中數據採用三次的平均值做紀錄。

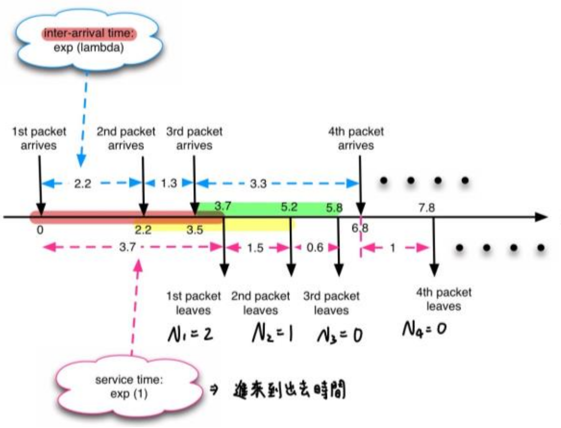
表一 數據結果

x	1	2	3	4	5
F^(x)	0.62856	0.86413	0.94933	0.98023	0.99277
$F_{/X}(x)$	0.630950	0.865790	0.950290	0.981210	0.993450
$F_X(x)$	0.632120	0.864664	0.950212	0.981684	0.993262

* $1-e^{-1}=0.63212$; $1-e^{-2}=0.86466$; $1-e^{-3}=0.95021$; $1-e^{-4}=0.98168$; $1-e^{-5}=0.99326$

Problem 3:

內容:每個封包的進來時間間隔符合 exp(lambda),每個封包的處裡時間是符合 exp(1),Ni 是指第 i 個封包離開時有幾個封包還未處理,Ti 是指從封包進來到完成離開所需要的時間。當輸入 lambda 時將輸出處裡一萬個封包後得出的平均 T 與 N。



圖一(封包時間線概念圖)

程式宣告解釋:

```
int sim(double lambda) {
    int N = -1;
    double resttime = 0.0;
    double T = 0.0;
    double Ti[10000];
    int Ni[10000];
    double Ttotal = 0.0;
    double Ntotal = 0.0;
    double inter_arrival_time[10000];
    double service_time[10000];
```

N: 一萬個封包離開時有幾個封包還未處理的平均數量

N i [10000]:第 i 個封包離開時有幾個封包還未處理

Ntotal:每次離開還未處裡的封包數總和

T: 一萬個封包從進來到完成離開所需要的平均時間

Ti[10000]:第 i 個封包進來到完成離開所需要的時間

Ttotal:每個封包從進來到完成離開所需要的時間總和

resttime:在一個服務時間中完成一個封包後剩餘的時間

inter_arrival_time[i]:i 與 i+1 個封包進來的時間間隔

service_time[i]:第 i 個的封包需要的服務時間

```
24      for (int i=0; i<10000; i++){
25      inter_arrival_time [i] = gen_exp_411186028(lambda);
26      service_time [i] = gen_exp_411186028(1);}
27      Ti[0] = service_time [0];
28      for (int i=1; i<10000; i++){
29          if(service_time [i]>inter_arrival_time [i-1]){
30          Ti[i] = service_time [i] - inter_arrival_time [i-1];
31          Ttotal += Ti[i];}}
32      for (int i=0; i<10000; i++){
33          resttime += Ti[i];
34          if (resttime >= inter_arrival_time[i] && i < 10000) {
35              N += 1;
36              resttime -= inter_arrival_time[i];}
37          else{
38              resttime += service_time[i+1];
39              Ni[i] = N;}
40          Ntotal += Ni[i];}
41      N=Ntotal/10000;
42      T=(Ttotal+service_time [0])/10000;
```

圖三(problem3 code 部分)

程式構思想法:

因為後面需要使用不同的 i 項做加減運算,所以需要先將一萬個的數據先跑完,以免要與下一個數據加減的時候發現還不知道下一個數據是什麼,無法運算。

T (從封包進來到完成離開所需要的平均時間):

程式範圍: 前面兩個 for 迴圈(24-31 行)

用累積分布函數概念可以求得 T,以圖一黃色螢光筆部分為 T_2 做例子,service time 的第 2 個的 CDF(累積到 5.2)再扣掉 inter-arrival time 的第 1 個的 CDF(累積到 2.2)。將此概念運用至 10000 個 T 中。求出每個的等待時間後分別存至 Ti[10000]內,並將其累加至 Ttotal,最後除 10000 可得平均值,值得注意的是因為累加的部分只有在 i=1 加到 i=9999,並沒有存入 i=0 之值,因此要在最總和地方再加上最後 service_time[0]之值。

而在第一個迴圈(24 行)中因為使用 service_time [i] - inter_arrival_time [i-1],i 若

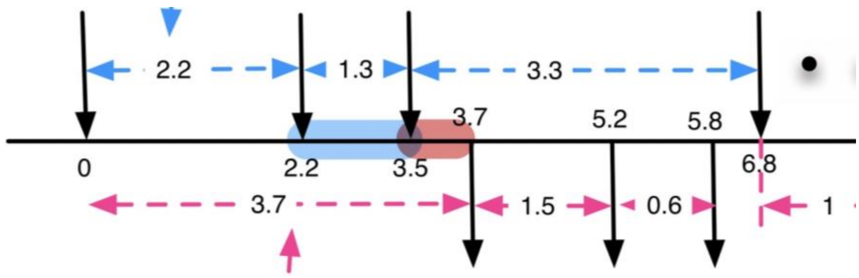
為 0 會出現負的，因此設此迴圈的開始為 i-1，也因為這樣所以需要自己設 Ti [0](27 行)，也就是 service_time 的第一個數據。

N(每個封包離開時有幾個封包未處理的平均數量):

程式範圍: 第三個 for 迴圈(32-40 行)

因為不能影響到 Ti 的值，在這裡多設一個變數 resttime 用來當作處理完一個封包所剩時間。並將 resttime 與 inter_arrival_time(iat)比較大小，如果 resttime 大於 iat，就把 N(初始值-1)+1，得到目前的 N 為 0，表示這段時間他本來就要完成一個封包，所以等待的封包數為 0，而判斷後 resttime 扣掉剛剛的 iat，如果扣掉後的 resttime 還比下一個 iat 大表示已經有封包進來在等了，N 就要在加 1，此時的 N 就會等於 1，扣下去直到後面的判斷 resttime 比較小，就直接跳到 else 裡面，並把下個 service_time 加上去，把時間累積加給下一個判斷是否有封包進來了。用以下圖四做解釋，3.7 扣掉 2.2 後的藍色加紅色時間為第一次相減後的 resttime，此時的 N 為-1+1=0，經比較後得知仍然大於 1.3，因此此時的 N=0+1=1，再把此時的 resttime(藍色加紅色)與 1.3 比較，此時的 N=1+1=2，再扣 1.3 後的 resttime 為紅色部分，接著與下一個 3.3 比較，發現小於 iat，所以跳到 else 裡面，把此時的 N 存入到 N[i]，並把所剩 resttime 與下一個 service_time 相加，繼續給下一個 N 判斷。

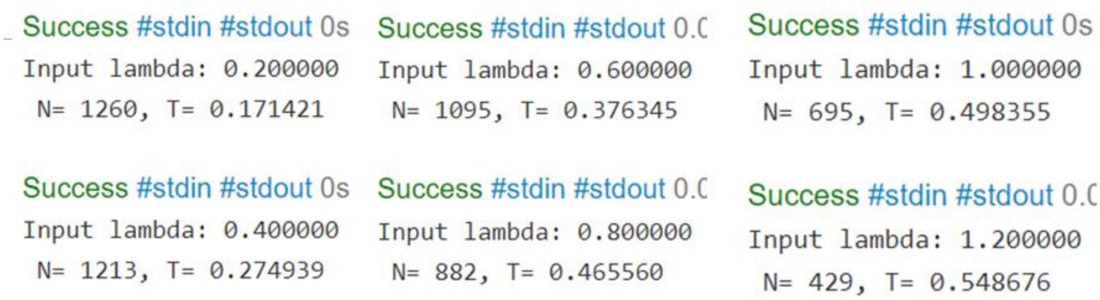
這裡也是把每一個 N[i]的值做累加存進 Ntotal，最後除以 10000 取平均值。



圖四 (封包時間線概念圖)

結果與討論:

lambda	0.2	0.4	0.6	0.8	1.0	1.2
N	1260	1213	1095	882	695	429
T	0.171421	0.274939	0.376345	0.465560	0.498355	0.548676


Success #stdin #stdout 0s
Input lambda: 0.200000
N= 1260, T= 0.171421

Success #stdin #stdout 0.C
Input lambda: 0.600000
N= 1095, T= 0.376345

Success #stdin #stdout 0s
Input lambda: 1.000000
N= 695, T= 0.498355

Success #stdin #stdout 0s
Input lambda: 0.400000
N= 1213, T= 0.274939

Success #stdin #stdout 0.C
Input lambda: 0.800000
N= 882, T= 0.465560

Success #stdin #stdout 0.C
Input lambda: 1.200000
N= 429, T= 0.548676

可從數據看出，隨著 lambda 增加，封包未處理的平均數量有降低的趨勢。根據

exponential(lambda) random variable 的基本定義:

$$PDF: f_x(x) = \begin{cases} \lambda e^{-\lambda x} & , \quad for \quad x > 0 \\ 0 & , \quad else \end{cases}$$

$$\Rightarrow CDF: F_x(x) = \begin{cases} 1 - e^{-\lambda x} & , \quad for \quad x > 0 \\ 0 & , \quad else \end{cases}$$

可以推得以下:

lambda 越小，到達時間越長，到達率(1/到達時間)就越低，同時等待的服務的封包數量可能會減少，因為到達的時間間隔較長，也就有更多時間可以服務先前到達的封包，使得等待服務的封包數量得以減少。

照邏輯推斷，如果服務率(1/服務時間)較大(服務時間較短)，或到達率較小(到達時間間隔長)，等待被服務的封包數量就會較少，因此 N 會減少，而因為等待時間 T[i]增加，平均等待時間(T)可能會增加。而隨著 lambda 增加的測試數據結果，N 越來越低，T 也有增加的趨勢，符合邏輯推斷。

Problem 4:

內容:與第三題封包概念相同，不同的是當系統正在服務一個封包時，此時進入的封包將不會繼續等待被服務而是直接被丟棄,我們要計算在送出一萬個封包中最後的丟棄比率。

程式構思想法:

我認為可以沿用第三題的時間比較方式，先判斷 service_time 是否大於 iat，如果大於就將他扣掉該個的 iat，並將剩下的 resttime 再與 service_time 做比較，若仍大於他就扣掉並將 discard 數量+1，resttime 會在更新，不斷地將他與下一個 iat 比大小，直到小於 iat 為止。