Simulación ITAM
10 de septiembre de 2018 2018-II

Tarea 1. Fecha de entrega: jueves 30 de agosto 2018

Lecturas

- Explained: Monte Carlo simulations
- As Forecats Go, You Can Bet on Monte Carlo
- Stan Ulam, John Von Neumann and the Monte Carlo Method
- Dagpunar, capítulos 1 y 2

Instrucciones

El propósito de los primeros ejercicios sobre `R` es que practiquen algunas de las tareas comunes que se realizarán durante el curso en diferentes momentos, *y que salgan las dudas en los momentos apropiados*. **Esta primera tarea es individual**.
Resuelve la siguiente tarea usando `RStudio` y generando un documento en word, en html y pdf y entrega los tres archivos en un sólo archivo zipeado enviado por correo a:
jorge.delavegagongora@gmail.com
y poner como asunto:
`[S18-II] Tarea 1 NombreDelAlumno`

También puedes usar otro software si quieres y verificar que se pueden completar las tareas indicadas.

Problemas

1. Sea X el número de 'unos' obtenido en 12 lanzamientos de un dado honesto. Entonces X tiene una distribución binomial. Calcular una tabla con los valores de la función de distribución para $x = 0, 1, \ldots, 12$ por dos métodos: usando la función `cumsum` y usando la función `pbinom`. También determinar cuánto vale $P(X > 7)$.

   Solución.

   - Sabemos que la fórmula de la función de masa de probabilidad binomial es $P(X = x) = \binom{n}{x} p^x (1-p)^{n-x}$ con $x \in \{0, 1, \ldots n\}$. En el ejemplo dado, $n = 12$, $p = 1/6$. Entonces los posibles valores $P(X = x)$ son los siguientes:

```
options(scipen=12) #da margen para evitar la notación científica
x <- 0:12
binomial <- function(x,n,p){ choose(n,x)*p^x*(1-p)^(n-x)} #define la función binomial
probas <- binomial(0:12,n=12,p=1/6)
probas
```

```
 [1] 0.1121566547846151 0.2691759714830763 0.2960935686313839
 [4] 0.1973957124209226 0.0888280705894152 0.0284249825886128
 [7] 0.0066324959373430 0.0011369993035445 0.0001421249129431
[10] 0.0000126333255949 0.0000007579995357 0.0000000275636195
[13] 0.0000000004593937
```

```
cumsum(probas) # es el equivalente a la función de distribución

 [1] 0.1121567 0.3813326 0.6774262 0.8748219 0.9636500 0.9920750 0.9987075
 [8] 0.9998445 0.9999866 0.9999992 1.0000000 1.0000000 1.0000000
```

- La función `pbinom` calcula directamente las probabilidades acumuladas, o equivalentemente, la función de distribución. Hay que revisar los parámetros de esta función para especificarla correctamente.

```
pbinom(0:12, size=12, prob=1/6)

 [1] 0.1121567 0.3813326 0.6774262 0.8748219 0.9636500 0.9920750 0.9987075
 [8] 0.9998445 0.9999866 0.9999992 1.0000000 1.0000000 1.0000000
```

- Para el tercer punto, lo podemos hacer directamente usando la función `pbinom` o `dbinom` de varias maneras:

```
pbinom(7, size=12, prob=1/6,lower.tail=F)

[1] 0.0001555443

1-pbinom(7, size=12, prob=1/6)

[1] 0.0001555443

sum(dbinom(8:12,size=12,prob=1/6))

[1] 0.0001555443
```

□

2. (Estaturas de presidentes gringos). En un artículo de Wikipedia, se reportan las estaturas de los Presidentes de los Estados Unidos y los de sus oponentes en elecciones. Se ha notado que mientras más alto sea el presidente típicamente gana la elección.

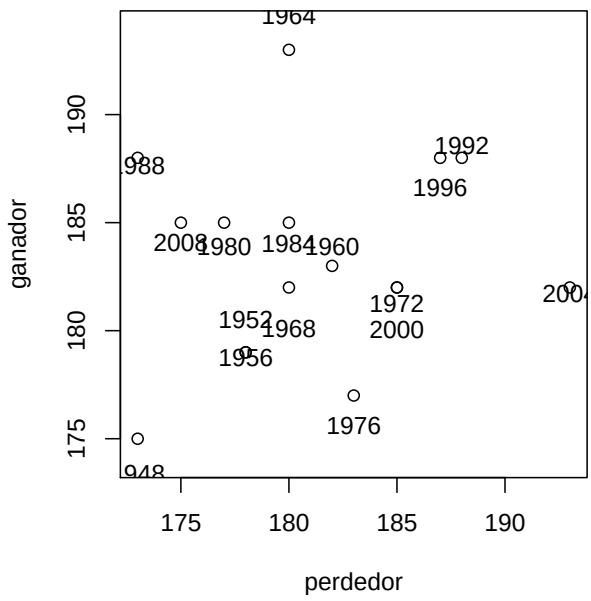
Year	Winner	Height	Opponent	Height
2008	Barack Obama	6 ft 1 in 185 cm	John McCain	5 ft 9 in 175 cm
2004	George W. Bush	5 ft 11.5 in 182 cm	John Kerry	6 ft 4 in 193 cm
2000	George W. Bush	5 ft 11.5 in 182 cm	Al Gore	6 ft 1 in 185 cm
1996	Bill Clinton	6 ft 2 in 188 cm	Bob Dole	6 ft 1.5 in 187 cm
1992	Bill Clinton	6 ft 2 in 188 cm	George H.W. Bush	6 ft 2 in 188 cm
1988	George H.W. Bush	6 ft 2 in 188 cm	Michael Dukakis	5 ft 8 in 173 cm
1984	Ronald Reagan	6 ft 1 in 185 cm	Walter Mondale	5 ft 11 in 180 cm
1980	Ronald Reagan	6 ft 1 in 185 cm	Jimmy Carter	5 ft 9.5 in 177 cm
1976	Jimmy Carter	5 ft 9.5 in 177 cm	Gerald Ford	6 ft 0 in 183 cm
1972	Richard Nixon	5 ft 11.5 in 182 cm	George McGovern	6 ft 1 in 185 cm
1968	Richard Nixon	5 ft 11.5 in 182 cm	Hubert Humphrey	5 ft 11 in 180 cm
1964	Lyndon B. Johnson	6 ft 4 in 193 cm	Barry Goldwater	5 ft 11 in 180 cm
1960	John F. Kennedy	6 ft 0 in 183 cm	Richard Nixon	5 ft 11.5 in 182 cm
1956	Dwight D. Eisenhower	5 ft 10.5 in 179 cm	Adlai Stevenson	5 ft 10 in 178 cm
1952	Dwight D. Eisenhower	5 ft 10.5 in 179 cm	Adlai Stevenson	5 ft 10 in 178 cm
1948	Harry S. Truman	5 ft 9 in 175 cm	Thomas Dewey	5 ft 8 in 173 cm

Hagan una gráfica de dispersión de puntos con la estatura del perdedor vs. el ganador.

Solución.

Aquí se necesita crear dos vectores de datos para poder hacer la gráfica. Se puede hacer en pies o en centimetros. Yo la haré en centímetros.

```r
par(pty="s") #haz la gráfica cuadrada
perdedor <- c(175,193,185,187,188,173,180,177,183,185,180,180,182,178,178,173)
ganador <- c(185,182,182,188,188,188,185,185,177,182,182,193,183,179,179,175)
año <- c(2008,2004,2000,1996,1992,1988,1984,1980,1976,1972,1968,1964,1960,1956,1952,1948)
plot(perdedor,ganador,asp=1)
text(perdedor,jitter(ganador,amount=2),año)
```



□

3. La función `rpois` genera observaciones aleatorias de una distribución Poisson. Usen la función `rpois` para simular un número grande ($n = 1000$ y $n = 10000$) muestras Poisson con parámetro $\lambda = 0.61$. Encuentren la función de masa de probabilidad, media, y varianza para las muestras. Comparen con los valores teóricos de la densidad Poisson.

Solución.

- Para el primer caso: (a varianza y la media casi deben ser la misma)

```r
P1 <- rpois(n=1000,lambda=0.61) #genera la muestra Poisson
Z <- table(P1)/1000   #Calcula las frecuencias relativas de los números observados en la muestra
dpois(0:(length(Z)-1),lambda=0.61)-table(P1)/1000 #Calcula desviaciones respecto a los valores observados

P1
           0            1            2            3            4
-0.0076491309 -0.0035559699  0.0110904292  0.0005550539 -0.0008653543

mean(P1)

[1] 0.591

var(P1)

[1] 0.5903093
```

- Para el segundo caso:

```
P2 <- rpois(n=10000,lambda=0.61) #genera la muestra Poisson
Z <- table(P2)/10000   #Calcula las frecuencias relativas de los números observados en la muestra
Z

P2
     0      1      2      3      4      5
0.5372 0.3363 0.1022 0.0211 0.0029 0.0003

dpois(0:(length(Z)-1),lambda=0.61)-table(P2)/10000 #calcula las desviaciones respecto a los valores observados

P2
             0              1              2              3              4
 0.00615086907 -0.00485596986 -0.00110957081 -0.00054494606  0.00023464573
             5
 0.00008242678

mean(P2)

[1] 0.6171

var(P2)

[1] 0.6081484
```

□

4. Escriban una función en `R` llamada `sd.n` que regrese el valor estimado de $\hat{\sigma}$ de una muestra de tamaño n, utilizando la fórmula del estimado máximo verosímil de la varianza.

Solución.

La desviación estándar obtenida por máxima verosimilitud es la versión sesgada de la muestra, que sólo se divide por n y no por $n-1$. Entonces podemos escribir:

```
#Si x es la muestra de tamaño length(x)=n
sd.n <- function(x){
        sum((x-mean(x))^2)/length(x)
}
#Por ejemplo
 x <- rnorm(1000,mean=2,sd=10)
 sd.n(x)


[1] 102.5514
```

□

5. Escriban una función `norma` que calcule la norma Euclideana de un vector numérico de longitud n. Evaluar la norma de los vectores $(0,0,0,1)$, $(2,5,2,4)$ y $(1,2,3,4,5,6,7,8,9,10)$.

Solución.

```
norma <- function(x) sqrt(sum(x^2))
#Entonces:
norma(c(0,0,0,1))

[1] 1

norma(c(2,5,2,4))

[1] 7

norma(1:10)

[1] 19.62142
```

□

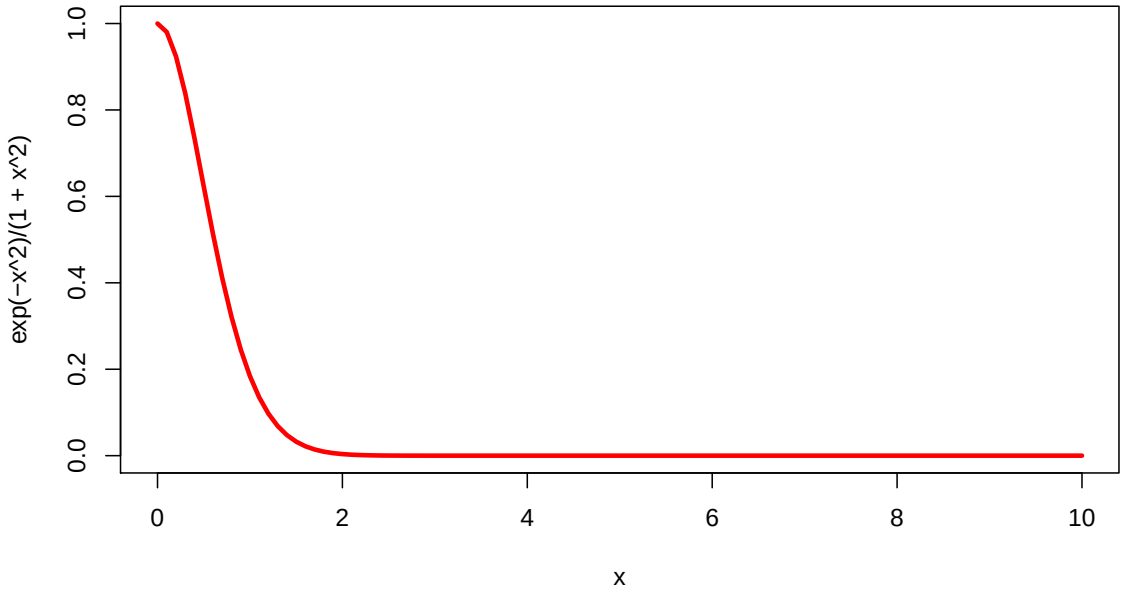
6. Usar la función `curve` para graficar la función $f(x) = e^{-x^2}/(1+x^2)$ en el intervalo $0 \leq x \leq 10$. Luego usar la función `integrate` para calcular el valor de la integral

$$\int_0^\infty \frac{e^{-x^2}}{1+x^2} dx$$

El límite superior se especifica usando el argumento `upper=Inf` en la función `integrate`.

Solución.

```
curve(exp(-x^2)/(1+x^2),from=0,to=10,col="red",lwd=3)
```




```
integrate(function(x)exp(-x^2)/(1+x^2),lower=0,upper=Inf)
```

```
0.6716467 with absolute error < 0.000083
```

□

7. Construir una matriz con 10 renglones y 2 columnas que contienen datos provenientes de una normal estándar:

```
x <- matrix(rnorm(20),10,2)
```

Esta es una muestra de 10 observaciones de una distribución normal bivariada. Usen la función `apply` y la función `norma` que crearon en un ejercicio anterior para calcular las normas euclideanas para cada una de las 10 observaciones.

Solución.

```
set.seed(10) #fijo la semilla para no estar cambiando la muestra
x <- matrix(rnorm(20),10,2) #crea la matriz
x #muestra los datos
```

```
             [,1]        [,2]
 [1,]  0.01874617  1.10177950
 [2,] -0.18425254  0.75578151
 [3,] -1.37133055 -0.23823356
 [4,] -0.59916772  0.98744470
 [5,]  0.29454513  0.74139013
 [6,]  0.38979430  0.08934727
 [7,] -1.20807618 -0.95494386
 [8,] -0.36367602 -0.19515038
 [9,] -1.62667268  0.92552126
[10,] -0.25647839  0.48297852
```

```
apply(x,1,norma) #calcula la norma por renglón
```

```
 [1] 1.1019390 0.7779169 1.3918702 1.1550104 0.7977570 0.3999032 1.5399240
 [8] 0.4127274 1.8715378 0.5468541
```

□

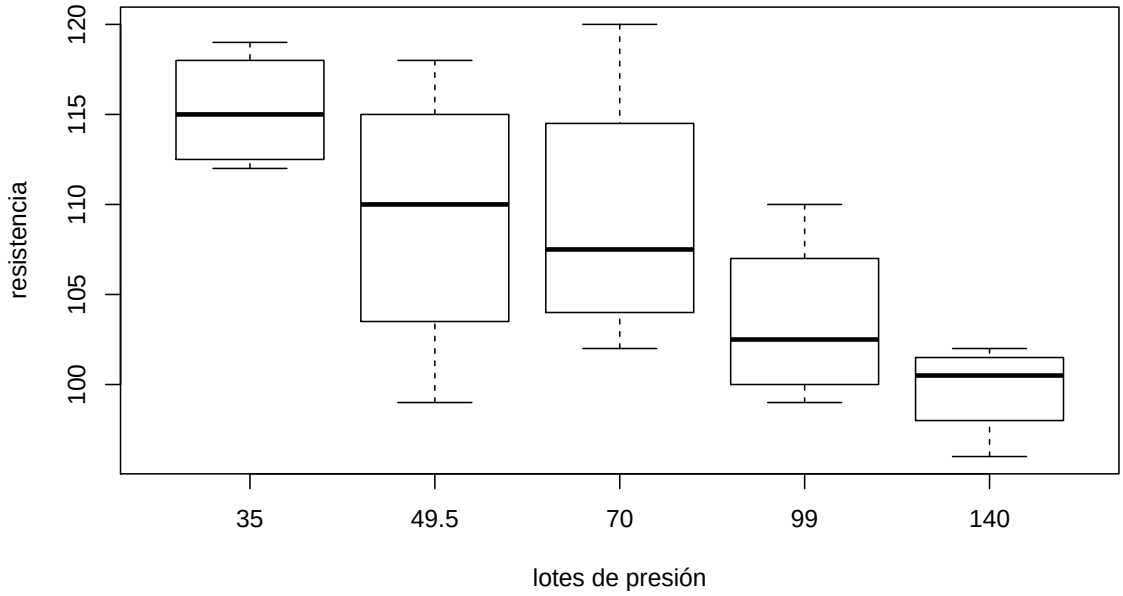
8. Los siguientes datos describen el factor de desgaste de papel manufacturado bajo diferentes presiones durante el prensado. Cuatro hojas de papel fueron seleccionadas y probadas para cada uno de los cinco lotes manufacturados:

Presión (lotes)	Factor de resistencia (hojas)
35.0	112 119 117 113
49.5	108 99 112 118
70.0	120 106 102 109
99.0	110 101 99 104
140.0	100 102 96 101

Metan estos datos en un `dataframe` con dos variables: factor de resistencia y presión. Hacer un boxplot para comparar los diferentes factores de resistencia para cada presión.

Solución.

```
datos <- data.frame(presion=rep(c(35,49.5,70,99,140),rep(4,5)),
                                        resistencia = c(112,119,117,113,
                                                        108,99,112,118,
                                                        120,106,102,109,
                                                        110,101,99,104,
                                                        100,102,96,101))
with(datos,boxplot(resistencia ~ presion, xlab="lotes de presión",ylab="resistencia"))
```



□

8. Este ejercicio está relacionado con el modelo de linea de espera que programamos, `mm1` en el archivo `Queue.R` que revisamos en laboratorio el martes 21 de agosto:

   a) Modifiquen el código del programa para incorporar las medidas adicionales de desempeño:

   - El tiempo total promedio de los n clientes en el sistema. (hint: piensen en las variables W_i = el tiempo total que pasa en el sistema el cliente i (espera + servicio). Entonces es similar al tema de estimar $\bar{D}$).
   - La longitud máxima de la cola
   - La máxima espera en cola

   Solución.

   Las modificaciones a realizar son las siguientes:

   1) Agrega una variable llamada `tiempo_en_sistema` en el módulo de inicialización de variables. Esta variable servirá para medir el tiempo total en el sistema, que es en espera más el tiempo de servicio.
   2) En `llegadas`, modifica el tiempo en sistema incrementando por el tiempo de servicio para los clientes que no esperan.
   3) En `salidas`, considera el incremento incrementado por el tiempo de servicio para los clientes que esperan.
   4) Agrega en el reporte, las variables: `longitud_max`, `espera_max`

   Ver el código modificado en el Anexo.

   □

b) Ejecuten el modelo `mm1` 100 veces con $\lambda_A = 5$, $\lambda_S = 4$ y $n = 1000$ y hacer un histograma para cada una de las estadísticas de desempeño, y calcular estadísticas descriptivas (media, varianza y coeficiente de variación, min, max, etc) para cada una de ellas.

Solución.

Definimos una lista para guardar los resultados de las 100 corridas. Después podemos extraer las variables relevantes y hacer las gráficas necesarias de cada una de las variables para hacer el cálculo solicitado.

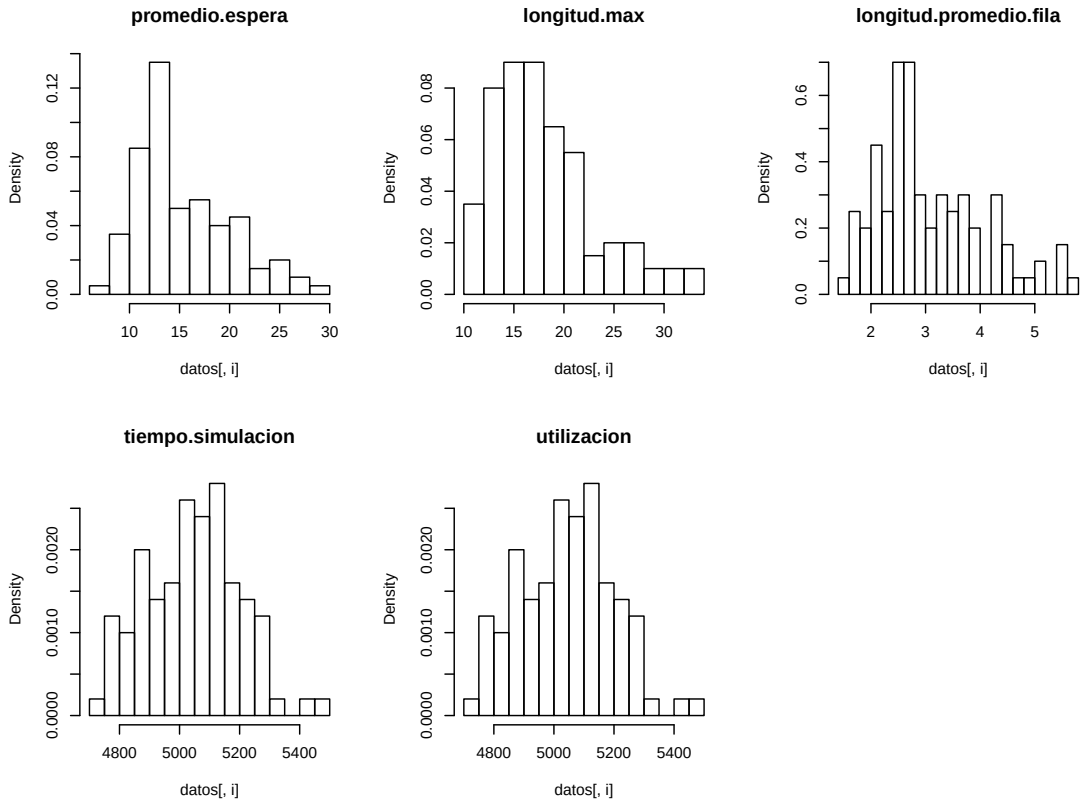
```
options(width=200)
source("~/Dropbox/Academia/ITAM/SimS18-II/scripts/Queue/Queue.R")
corrida <- list(NULL)
for(i in 1:100) corrida[[i]] <- mm1(lambdaA = 5,lambdaS = 4,n = 1000)
datos <- data.frame(
promedio.espera = unlist(lapply(corrida,function(x)x$promedio.espera)),
longitud.max = unlist(lapply(corrida,function(x)x$longitud_max)),
longitud.promedio.fila = unlist(lapply(corrida,function(x)x$longitud.promedio.fila)),
tiempo.simulacion = unlist(lapply(corrida,function(x)x$tiempo.simulacion)),
utilizacion = unlist(lapply(corrida,function(x)x$tiempo.simulacion)))
apply(datos,2,summary)
```

```
        promedio.espera longitud.max longitud.promedio.fila tiempo.simulacion utilizacion
Min.           7.436557        11.00               1.479248          4745.497    4745.497
1st Qu.       12.058206        15.00               2.425538          4941.576    4941.576
Median        13.918509        18.00               2.770934          5051.289    5051.289
Mean          15.403528        18.53               3.078623          5044.638    5044.638
3rd Qu.       18.182300        21.00               3.674945          5142.453    5142.453
Max.          28.695283        34.00               5.620074          5481.103    5481.103
```

```
par(mfrow=c(2,3))
for(i in 1:5)hist(datos[,i],main=names(datos)[i],prob=T,breaks=15)
datos
```

```
    promedio.espera longitud.max longitud.promedio.fila tiempo.simulacion utilizacion
1          9.371532           12               1.823929          5138.100    5138.100
2         18.725900           19               3.774933          4983.557    4983.557
3         12.669616           18               2.474501          5120.070    5120.070
4         13.339635           18               2.645525          5069.058    5069.058
5         18.892221           23               3.968989          4766.280    4766.280
6         11.314614           14               2.232106          5071.739    5071.739
7         22.505524           30               4.366046          5225.478    5225.478
8         13.894354           20               2.746359          5059.190    5059.190
9         16.865770           20               3.308472          5121.308    5121.308
10        18.890149           15               3.665213          5169.517    5169.517
11        21.367388           22               4.257751          5018.859    5018.859
12        18.606757           25               3.704140          5271.730    5271.730
13        12.436811           18               2.409942          5267.315    5267.315
14        13.835137           18               2.828613          4891.139    4891.139
15        25.231789           21               4.988334          5070.431    5070.431
16        20.876534           22               4.261631          4899.246    4899.246
17        20.072906           19               3.998728          5040.113    5040.113
18        12.254320           13               2.475062          5006.249    5006.249
19        14.832349           16               2.929837          5068.603    5068.603
20        21.583005           21               4.432315          4875.092    4875.092
21        13.398701           17               2.548001          5268.381    5268.381
22        13.629300           18               2.743532          4967.793    4967.793
23         9.908586           13               1.875054          5287.817    5287.817
24        14.533115           19               2.998229          4853.748    4853.748
25        13.555664           19               2.593727          5226.326    5226.326
26        13.971199           13               2.791600          5004.728    5004.728
27        21.169682           27               4.413823          4830.354    4830.354
28        12.596297           12               2.480607          5114.252    5114.252
29        10.916761           15               2.134488          5131.477    5131.477
30        13.359906           16               2.676703          4998.757    4998.757
31        11.350695           15               2.144305          5303.677    5303.677
32        16.443987           19               3.235039          5095.176    5095.176
33        13.639210           15               2.707250          5043.388    5043.388
34        12.917601           14               2.643459          4886.629    4886.629
35        12.737953           14               2.534661          5026.152    5026.152
36        13.841002           19               2.658697          5205.935    5205.935
37        10.214629           12               1.997977          5112.485    5112.485
38        13.020031           18               2.524388          5157.697    5157.697
39        27.685019           27               5.589935          4954.576    4954.576
```

40	14.204455	23	2.787804	5096.695	5096.695
41	12.669166	18	2.543798	4983.736	4983.736
42	11.166472	21	2.188263	5110.666	5110.666
43	11.720054	12	2.292446	5112.467	5112.467
44	19.468705	19	3.960302	4944.102	4944.102
45	15.861362	18	3.331590	4760.898	4760.898
46	11.622920	13	2.283147	5090.745	5090.745
47	17.389424	21	3.343669	5200.701	5200.701
48	15.691060	19	3.075168	5102.505	5102.505
49	13.942664	16	2.754064	5068.487	5068.487
50	24.181502	21	5.188800	4767.846	4767.846
51	10.318906	12	1.894850	5446.876	5446.876
52	13.308227	21	2.560448	5197.615	5197.615
53	19.535526	20	3.772805	5177.984	5177.984
54	20.533082	24	4.305727	4790.081	4790.081
55	17.742093	17	3.531583	5023.835	5023.835
56	10.592282	12	2.111633	5021.280	5021.280
57	8.923533	13	1.720708	5185.965	5185.965
58	17.394759	26	3.497820	4973.378	4973.378
59	12.550226	15	2.565792	4891.366	4891.366
60	16.819585	17	3.384840	4989.621	4989.621
61	28.695283	33	5.620074	5140.804	5140.804
62	11.750921	15	2.318729	5067.828	5067.828
63	13.398960	16	2.707995	4947.926	4947.926
64	7.436557	13	1.479248	5133.450	5133.450
65	16.330137	22	3.481840	4862.539	4862.539
66	16.925236	28	3.430329	4933.997	4933.997
67	25.847844	29	5.404705	4782.471	4782.471
68	22.847138	32	4.528311	5098.279	5098.279
69	10.219254	13	2.117825	4832.282	4832.282
70	18.014586	18	3.652105	4932.658	4932.658
71	18.040814	18	3.715737	4948.876	4948.876
72	13.429333	14	2.613484	5225.106	5225.106
73	10.598173	14	2.046733	5178.093	5178.093
74	16.053981	17	3.290845	4878.376	4878.376
75	8.895247	17	1.689638	5269.256	5269.256
76	20.698877	32	4.236836	4885.584	4885.584
77	20.566942	22	4.389670	4745.497	4745.497
78	27.717593	26	5.405618	5127.553	5127.553
79	9.279064	11	1.794180	5171.759	5171.759
80	9.708454	14	1.776575	5481.103	5481.103
81	22.276205	20	4.633804	4828.904	4828.904
82	20.242022	28	3.971446	5096.889	5096.889
83	15.525452	19	3.136107	4964.890	4964.890
84	11.712043	15	2.443706	4794.882	4794.882
85	12.160635	16	2.430737	5009.941	5009.941
86	11.279813	14	2.173500	5200.675	5200.675
87	8.443028	13	1.608887	5247.745	5247.745
88	11.680943	17	2.349403	5002.157	5002.157
89	14.176894	16	2.927264	4843.053	4843.053
90	14.103415	15	2.915554	4837.302	4837.302
91	10.861027	13	2.156627	5036.119	5036.119
92	13.653553	18	2.604544	5274.489	5274.489
93	12.941547	16	2.531975	5111.246	5111.246
94	15.271333	16	3.056062	5002.568	5002.568
95	16.830341	16	3.473358	4850.699	4850.699
96	13.400669	17	2.678829	5014.469	5014.469
97	11.169324	15	2.170232	5147.403	5147.403
98	25.471059	34	5.149781	4946.047	4946.047
99	14.218296	21	2.900094	4902.703	4902.703
100	16.389186	26	3.170772	5168.926	5168.926



Anexo

A continuación se lista el código generado para simular la fila con las modificaciones en donde dice `Tarea 1`

```
# Adaptación del programa de Law y Kelton (2000) para modelo M/M/1
# Elaborado por: Jorge de la Vega
# Fecha: marzo 2016 para el curso Simulación 2016

#Este programa simula una linea de espera con un servidor, de acuerdo a los principios de
#DES (Discrete event Simulation).Considera la ejecución de procedimientos en bloques.

## Variables consideradas:

# En relación a los clientes:
# lambda_A = tiempo promedio de interarribo de los clientes. Es la media de la distribución de llegadas
# n        = número de clientes que entraran al sistema. Es fijado de antemano.
# lt_A     = vector con los tiempos de arribo (de longitud Qlim)
# Di      = tiempo de espera de un cliente en un momento dado
# le       = lista de eventos con su tiempo y su tipo

# En relación al servicio:
# lambda_S = tiempo promedio de servicio.

# En relación a los indicadores del sistema
# area_q               = variable auxiliar para calcular la longitud promedio de la fila.
# area_status_servidor = variable auxiliar para calcular la utilización del servidor.
# q_t                  = longitud de la fila en un instante dado.
# num_eventos = numero de eventos para la función de tiempo
# clientes_enespera     = clientes que han esperado en fila
# servidor              = Estado del servidor
# sig_tipo_evento       = Siguiente tipo de evento (1= llegada 2= salida)
# reloj                 = reloj de simulación (e)
# tiempo_ultimo_evento  = tiempo del evento más reciente
# tiempo_sig_evento     = tiempo al siguiente evento
# total_esperas         = El total de esperas D para todos los clientes


# La siguiente función es la función principal que llama a cada subrutina para llevar a
# cabo la simulación, de acuerdo al flujo que se describió en clase.

mm1 <- function(lambdaA = 15, lambdaS = 10, n = 100){
```

```
inicializa(lambdaA = lambdaA, lambdaS = lambdaS, n = n) #inicializa simulación
#ejecuta la simulación mientras se cumple la condición de que
#los clientes en espera alcanzan el valor fijo de clientes que pasan por el sistema
while (clientes_enespera < n){
tiempo() #determina el siguiente evento y avanza los relojes
actualiza_estadisticas() #actualiza los acumuladores
if(sig_tipo_evento == 1) llegadas() else salidas() #llama la función de evento
next
}
return(reporte()) #Genera el reporte al final de la simulación.
}


# Esta función inicializa los parámetros de la simulación, así como las variables de estado del sistema
# tiempo_ultimo_evento = tiempo del evento más reciente
inicializa  <- function(lambdaA, lambdaS, n){
lambdaA  <<- lambdaA   # media del de llegadas
lambdaS  <<- lambdaS   # tiempo promedio de servicio
n        <<- n          # El número de clientes en el sistema que se consideran antes de detener la simulación
num_eventos <<- 2       # número de eventos distintos (llegadas y salidas)
lt_A     <<- vector(mode="numeric", length = 1) #inicializa el vector para guardar los tiempos de arribo
reloj    <<- 0             # inicializa reloj de simulación
servidor <<- 0          # inicializa variables de estado 0 = libre, 1= ocupado
q_t      <<- 0
tiempo_ultimo_evento <<- 0

#incializa los contadores estadísticos
clientes_enespera <<- 0
total_esperas     <<- 0
tiempo_en_sistema <<-0   #Tarea 1
area_q            <<- 0
area_status_servidor <<- 0

#inicializa la lista de eventos: el primer arribo y el tiempo de salida. Se asigna un tiempo de salida muy grande, ya que no hay
#clientes esperando. Con esto se garantiza que el siguiente evento sea una llegada.
tiempo_sig_evento <<- c(reloj + rexp(1, 1/lambdaA), 1e30)
le <<- c(e=reloj,tipo=0,q=q_t)
}


# Esta función se usa para comparar el tiempo_sig_evento[1] (llegadas) y
# tiempo_sig_evento[2] (salidas) y
# definir sig_tipo_evento que sea igual al mínimo de esos dos. Después avanza el reloj
# de simulación al tiempo de ocurrencia del tipo de evento escogido, min_tiempo_sig_evento.
tiempo <- function(){
min_tiempo_sig_evento <<- 1e29  #valor inicial del mínimo
sig_tipo_evento <<- 0

#Determina el tipo de evento del siguiente evento a ocurrir (una llegada o una salida)
#de acuerdo a su tamaño
for(i in 1:num_eventos){
if( tiempo_sig_evento[i] < min_tiempo_sig_evento ){
min_tiempo_sig_evento <<- tiempo_sig_evento[i]
sig_tipo_evento <<- i
}
}

#verifica si la lista de eventos está vacía
if(sig_tipo_evento == 0)
stop(print(paste("La lista de eventos está vacía en el tiempo:", reloj, sep=" ")))

#La lista de eventos no está vacía, avanza el reloj de simulación
reloj <<- min_tiempo_sig_evento
le <<- rbind(le,c(reloj,sig_tipo_evento,q=q_t))
}


llegadas <- function(){
tiempo_sig_evento[1] <<- reloj + rexp(1, 1/lambdaA) #Programa un evento de llegada
if(servidor == 1){
q_t <<- q_t + 1 #aumenta la cola en 1
lt_A[q_t] <<- reloj  #guarda el tiempo de llegada de este cliente en la lista de eventos.
} else {
Di <<- 0
total_esperas <<- total_esperas + Di
clientes_enespera <<- clientes_enespera + 1
servidor <<- 1
tiempo_sig_evento[2] <<- reloj + rexp(1, 1/lambdaS) #tiempo de salida
tiempo_en_sistema <<- tiempo_en_sistema + tiempo_sig_evento[2] -reloj   #Tarea 1
}
}


#Esta Función sigue el diagrama de flujo que vimos en clase.
salidas <- function(){
if(q_t == 0){
servidor <<- 0
tiempo_sig_evento[2] <<-  1e30
} else {
q_t <<- q_t - 1
Di <<- reloj - lt_A[1]
total_esperas <<- total_esperas + Di
clientes_enespera <<- clientes_enespera + 1
tiempo_sig_evento[2] <<- reloj + rexp(1, 1/lambdaS)
tiempo_en_sistema <<- tiempo_en_sistema + tiempo_sig_evento[2] -reloj  #Tarea 1
for(i in 1:q_t) lt_A[i] <<- lt_A[i+1]
}
}

#Función de reporte
reporte <- function(){
#print(paste("Promedio de espera en la fila:", round(total_esperas/clientes_enespera, 2), "minutos", sep=" "))
```

```
#print(paste("Número promedio de clientes esperando en la fila:",round(area_q/reloj, 2), sep = " "))
#print(paste("Utilización del servidor:",100*round(area_status_servidor/reloj, 2), "%", sep = " "))
#print(paste("El tiempo de simulación fue de:", round(reloj,2), "minutos", sep = " "))
return(list(promedio.espera = total_esperas/clientes_enespera,
tiempo_total_promedio_en_sistema = tiempo_en_sistema/reloj, #Tarea1
longitud_max = max(le[,3]),   #Tarea 1
espera_max = max(total_esperas), #Tarea 1
longitud.promedio.fila = area_q/reloj,
utilizacion = area_status_servidor/reloj,
tiempo.simulacion = reloj,
le = le))
}


#Función de actualización de estadísticas
actualiza_estadisticas <- function(){
tiempo_desde_ultimo_evento <<- reloj - tiempo_ultimo_evento
tiempo_ultimo_evento <<-  reloj
area_q <<- area_q + q_t * tiempo_desde_ultimo_evento
area_status_servidor <<- area_status_servidor + servidor * tiempo_desde_ultimo_evento
}
```