Telecommunications Engineering
Universidad Carlos III de Madrid
Statistics

Assignment 3: Probability Models

Group	Students	Signatures
	● Alonso Herreros Copete	[illegible]

IMPORTANT: *The teachers of this course apply a 'zero tolerance' policy regarding academic dishonesty. Students that sign up this document agree to deliver an original work. The breach of this commitment will result in academic punishment.*

Observations:

Solve the exercises in the **Assignment3.pdf** file. **Note:** It is advisable to consult the manual for basic operation of MATLAB / Octave available on the website of the course.

Code available on GitHub: https://github.com/alonso-herreros/uni-stat-lab3

EXERCISE 1.

From a study on the number of friends request received *weekly* in the social network's account of UC3M School of Engineering undergraduate students, it is known that the number of *undesired* friends request (i.e. people that the student actually does not know) received by a student is distributed according to a Poisson with variance 7.

a) Compute by simulation in MATLAB/Octave the probability that two students would receive more than 15 *undesired* friend requests.

X_1, X_2 = numbers of undesired friend requests in a week for two different people

$X_1, X_2 \sim Poisson(\lambda_X)$

$X_3 = X_1 + X_2$ = total number of undesired requests received by two people in a week.

$X_3 \sim Poisson(\lambda_X + \lambda_X)$

A simple simulation was done in MATLAB using the built-in function `poissrnd` to generate a vector `x3` with random values according to the Poisson distribution of parameter $2 \cdot \lambda_X = 14$. In order to somewhat check that the simulation was running correctly, the mean was calculated, knowing it's theoretical value should be the same as the parameter in the Poisson distribution ($2 \cdot \lambda_X = 14$). This helps catch errors before calculating the probability with incorrect code.

Then, in order to calculate the probability of any value in the vector being greater than 15, the number of occurrences of numbers greater than 15 in the vector `x3` was counted using `sum(x3 > 15)`, which first turns `x3` into a logical vector, with values 1 where a number greater than 15 was found and values 0 where the number was less than or equal to 15. The `sum` function then adds all the ones. By dividing this by the number of entries in the vector, we get the probability of a value being greater than 15.

```
N = 10000; % number of simulations
lambdaX = 7; % requests/week

disp("a) Probability of two people getting more than 15 undesired friend requests");

% There is a built-in function to do exactly what we need
x3 = poissrnd(2*lambdaX, N, 1); % Generate N samples of X3 ~ Poisson(2*lambdaX)
disp("By built-in function:")
disp(" Mean (should be close to 14) = " + mean(x3));
disp(" P(X3 > 15) = " + sum(x3 > 15)/N);
```

```
a) Probability of two people getting more than 15 undesired friend requests


By built-in function:
 Mean (should be close to 14) = 14.0153
 P(X3 > 15) = 0.3339
```

In order to more closely simulate what these functions represent, a custom function that simulates the Poisson process was written and tested. This function makes use of the relationship between the Poisson process and the Exponential distribution in the following way: since the corresponding exponential distribution represents the time until an event occurs, we can generate random values according to this exponential distribution until their sum is greater than the allotted time period. This represents generating the intervals between one event and the next until they add up to more than one week. By counting how many random intervals fit in one week, we obtain the number of times our random event “receiving an undesired friend request” has occurred.

```
function [n] = custom_poisson(lambda)
    elapsed = 0; % Time elapsed since the start of the week
    n = -1; % Requests recevied this week. Starting at -1 so on the first loop it becomes 0

    while elapsed < 1
        % Add 1 to the number of requests received this week. At first it becomes 0. If the loop is repeated, it becomes 1, etc.
        n = n + 1;
        % Generate a random time t, according to the distribution of T3 ~ Exp(1/2*lambdaX). This is the time until the next undesired friend request.
        % t = exprnd(1/lambda); <- this is a built-in function that does exactly what we need here, but I'm going to do it manually.
        % X ~ Exp(1/lambda) -> F(x) = 1 - e^(-lambda*x)
        % F(x) = U -> 1 - e^(-lambda*x) = U -> e^(-lambda*x) = 1 - U -> -lambda*x = ln(1 - U) -> x = -ln(1 - U)/lambda
        % So, we generate a random number U ~ U(0, 1) and use it to generate a random time t ~ Exp(1/lambda)
        U = rand();
        t = -log(1 - U)/lambda;
        % Add the time until the next undesired friend request to the elapsed time
        elapsed = elapsed + t;
        % If this time is still before the end of the week, then this request was still received in the same week. The loop will repeat
    end
end
```

```
% Now, using the step-by step processes defined in the custom function
x3_custom = zeros(N, 1); % Start the vector at it's final size
for i = 1:N % Simulate N times
    % Add the number of requests received by each person
    x3_custom(i) = custom_poisson(lambdaX) + custom_poisson(lambdaX);
end
disp("By custom simulation:");
disp(" Mean (should be close to 14) = " + mean(x3_custom));
disp(" P(X3 > 15) = " + sum(x3_custom > 15)/N);
```

```
By custom simulation:
 Mean (should be close to 14) = 14.0019
 P(X3 > 15) = 0.3325
```

As we can see, both methods return similar conclusions. The mean in both cases is close to the expected value, indicating that the simulations likely to be accurate, and they return similar values for the probability we were trying to find: near 33%

b) Compute by simulation in MATLAB/Octave the probability that in a particular week, 180 or more students from a selection of 250 of the campus, had received between 5 and 10 (both included) undesired friend requests each.

Once again, the built-in function `poissrnd` can do the simulation for us, and then we can use that data to find the probability of this specific event. The function was used to generate a matrix where each row represents one iteration of the simulation, and each element in that row represents the number of undesired friend requests received by each one of the 250 selected members.

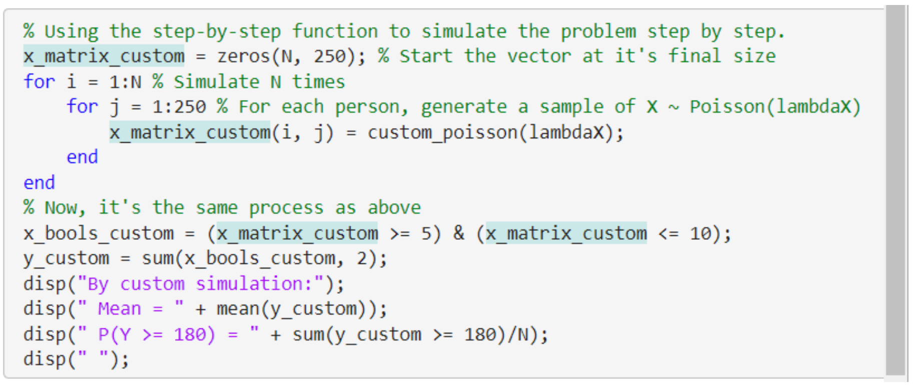
We can turn this into a logical matrix where each element is a 1 if the student received between 5 and 10 undesired friend requests, and a 0 if they received less than 5 or more than 10. Then, we can count the number of students which did receive between 5 and 10 by counting the number of ones, that is, by adding all of the ones with the `sum` function. This gives us a column vector `y` of individual simulations, where each entry is the number of students out of 250 who received between 5 and 10 undesired friend requests. It represents values of a random variable Y, described by the Binomial distribution of 250 trials and success being the random variable X between 5 and 10: $B(250,\ P(5 \leq X \leq 10))$.

Finally, by counting the number of occurrences greater than or equal to 180 in this y vector, we can obtain the number of simulations in which our main event occurred, and by dividing this by the total number of simulations we can find the probability of the event.

```
disp("b) Probability of 180 or more students out of 250 people getting between 5 and 10 undesired friend requests each");
% Generate N samples of X ~ Poisson(lambdaX) for each of the 250 people
x_matrix = poissrnd(lambdaX, N, 250);
% Create a matrix of booleans, where each element is true if the person received between 5 and 10 undesired friend requests
x_bools = (x_matrix >= 5) & (x_matrix <= 10);
% Turn it into a vector of the number of people who received between 5 and 10 undesired friend requests, one entry per week
y = sum(x_bools, 2);
disp("Let Y = Number of students out of 250 who received between 5 and 10 undesired friend requests each")
disp("Y ~ B(250, P(5 <= X <= 10))")
disp("By built-in function:");
disp(" Mean = " + mean(y));
disp(" P(Y >= 180) = " + sum(y >= 180)/N);
```

```
b) Probability of 180 or more students out of 250 people getting between 5 and 10 undesired friend requests each

Let Y = Number of students out of 250 who received between 5 and 10 undesired friend requests each
Y ~ B(250, P(5 <= X <= 10))
By built-in function:
 Mean = 182.1708
 P(Y >= 180) = 0.6543
```

As in the previous example, this was also calculated using the custom step-by-step function. Each entry of a similar matrix was populated using the custom function, and then the same procedure was followed to count the number of students who got between 5 and 10 undesired friend requests, count the number of times that count turned out to be greater than 180, and calculate the probability of 180 or more students getting between 5 and 10 undesired friend requests.



```matlab
% Using the step-by-step function to simulate the problem step by step.
x_matrix_custom = zeros(N, 250); % Start the vector at it's final size
for i = 1:N % Simulate N times
    for j = 1:250 % For each person, generate a sample of X ~ Poisson(lambdaX)
        x_matrix_custom(i, j) = custom_poisson(lambdaX);
    end
end
% Now, it's the same process as above
x_bools_custom = (x_matrix_custom >= 5) & (x_matrix_custom <= 10);
y_custom = sum(x_bools_custom, 2);
disp("By custom simulation:");
disp(" Mean = " + mean(y_custom));
disp(" P(Y >= 180) = " + sum(y_custom >= 180)/N);
disp(" ");
```

```
By custom simulation:
 Mean = 182.1869
 P(Y >= 180) = 0.6549
```

EXERCISE 2.

a) The waiting time (in seconds) that a student takes to login to a PC in a computer lab, is a random variable X with density function given by:

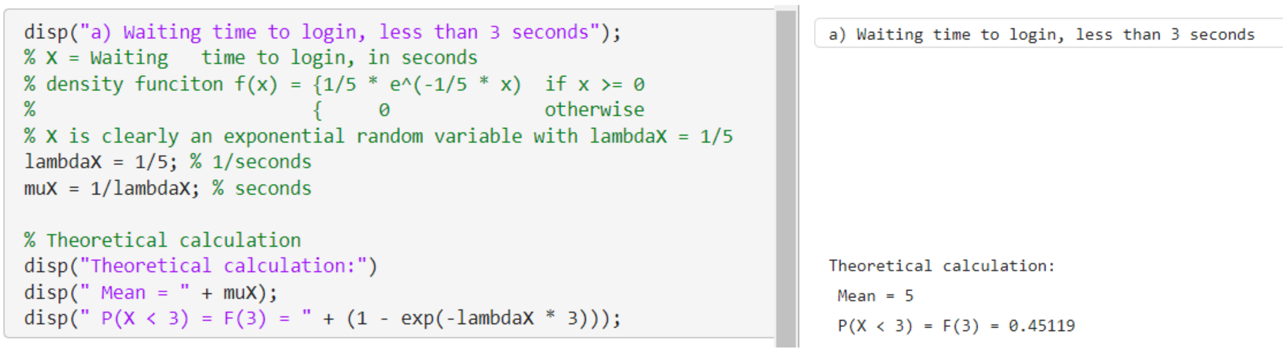
$$f(x) = \begin{cases} \frac{1}{5}e^{-\frac{1}{5}x} & if\ x \geq 0 \\ 0 & otherwise \end{cases}$$

Compute by simulation in MATLAB/Octave the probability that a student does not take more than 3 seconds to login. Compare the result obtained in the simulation with the theoretical result.

First, the theoretical result was calculated. Since X clearly follows an exponential distribution with parameter $\lambda_X = \frac{1}{5}$, this was quite simple. The probability of X being less than 3 is equal to its distribution function evaluated at 4:

$$P(X < 3) = F(3) = \int_{-\infty}^{3} f(x)dx = \int_{0}^{3} \frac{1}{5}e^{-\frac{1}{5}x}dx = 1 - e^{-\frac{1}{5}\cdot 3} = 0.45119$$

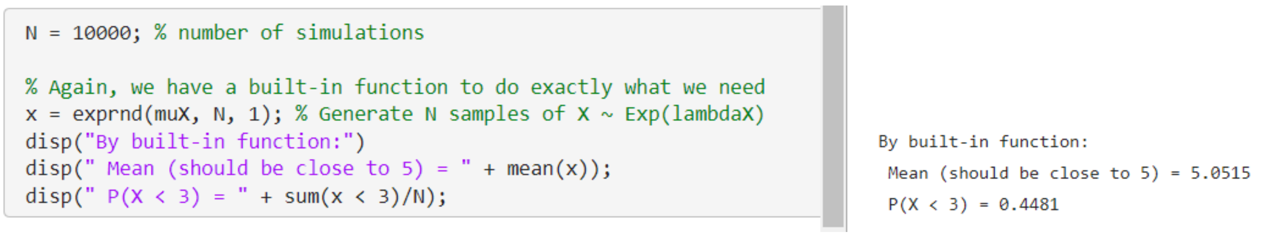
The mean was also calculated in order to check for simple errors quickly.



```
disp("a) Waiting time to login, less than 3 seconds");
% X = Waiting   time to login, in seconds
% density funciton f(x) = {1/5 * e^(-1/5 * x)  if x >= 0
%                         {     0              otherwise
% X is clearly an exponential random variable with lambdaX = 1/5
lambdaX = 1/5; % 1/seconds
muX = 1/lambdaX; % seconds

% Theoretical calculation
disp("Theoretical calculation:")
disp(" Mean = " + muX);
disp(" P(X < 3) = F(3) = " + (1 - exp(-lambdaX * 3)));
```

```
a) Waiting time to login, less than 3 seconds

Theoretical calculation:
 Mean = 5
 P(X < 3) = F(3) = 0.45119
```

As in the previous exercise, there is a built-in function (`exprnd`) which can generate random values from an exponential distribution. It was used to easily generate 10000 random samples of X, and the mean was calculated using the `mean(x)` function to catch possible errors early. Then, to calculate the probability of one value being less then 3, the total number of values less than 3 was counted, and then divided by the total number of simulations, in order to get the simulated probability.

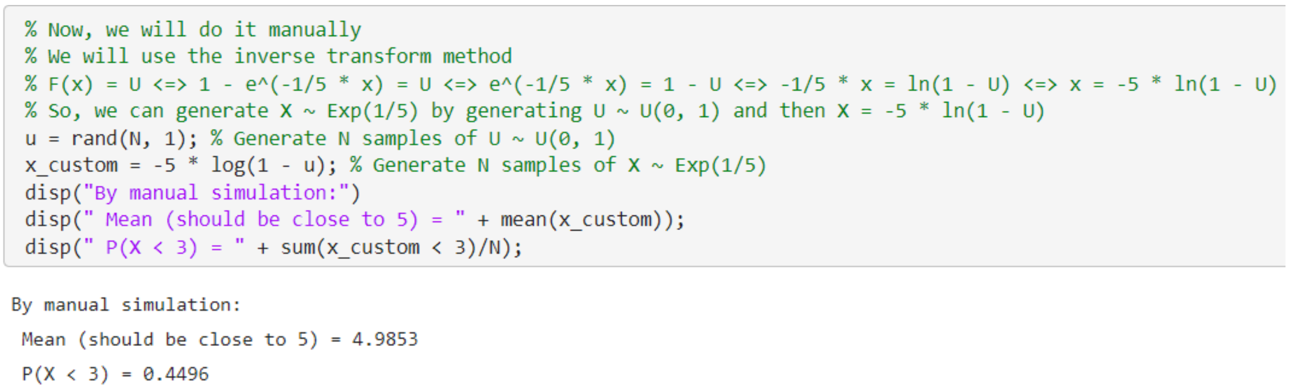


```
N = 10000; % number of simulations

% Again, we have a built-in function to do exactly what we need
x = exprnd(muX, N, 1); % Generate N samples of X ~ Exp(lambdaX)
disp("By built-in function:")
disp(" Mean (should be close to 5) = " + mean(x));
disp(" P(X < 3) = " + sum(x < 3)/N);
```

```
By built-in function:
 Mean (should be close to 5) = 5.0515
 P(X < 3) = 0.4481
```

As we can see, the simulation returned a probability of 44.8%, which is close to the theoretical value.

Just for the sake of completeness, the inverse transform method was also used to generate a similar vector of random values. By solving the equation $u = F(x)$ for x, we can generate a uniform random variable $U \sim U(0, 1)$ and assign each value to a value of x, and they will follow the distribution of X. After generating the random values, the calculations were quite simple.



```
% Now, we will do it manually
% We will use the inverse transform method
% F(x) = U <=> 1 - e^(-1/5 * x) = U <=> e^(-1/5 * x) = 1 - U <=> -1/5 * x = ln(1 - U) <=> x = -5 * ln(1 - U)
% So, we can generate X ~ Exp(1/5) by generating U ~ U(0, 1) and then X = -5 * ln(1 - U)
u = rand(N, 1); % Generate N samples of U ~ U(0, 1)
x_custom = -5 * log(1 - u); % Generate N samples of X ~ Exp(1/5)
disp("By manual simulation:")
disp(" Mean (should be close to 5) = " + mean(x_custom));
disp(" P(X < 3) = " + sum(x_custom < 3)/N);
```

```
By manual simulation:
 Mean (should be close to 5) = 4.9853
 P(X < 3) = 0.4496
```

The simulated probability was 44.9%, which is also close to the theoretical value.

b) Let Y be the r.v. that represents the duration (in hours) of the connection time from a PC of the computer lab to a social network is given by the density function:

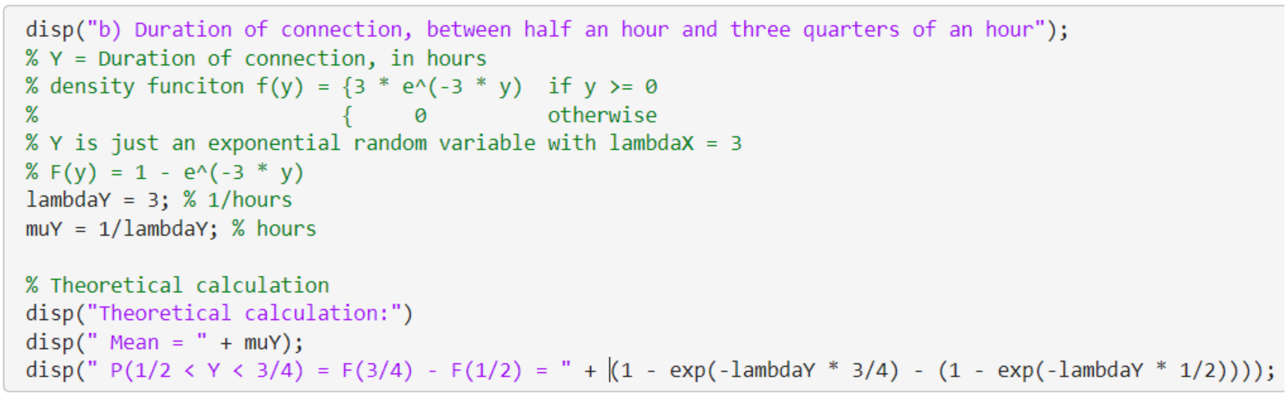
$$f(y) = \begin{cases} 3e^{-3y} & if\ y \geq 0 \\ 0 & otherwise \end{cases}$$

Compute by simulation in MATLAB/Octave the probability that a student is connected to the social network between half an hour and three quarters of an hour in the computer lab. Compare the result obtained in the simulation with the theoretical result.

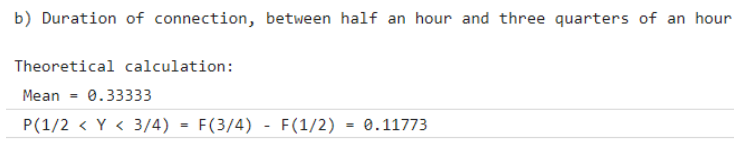
Once again, Y clearly follows an exponential distribution with parameter $\lambda_Y = 3$. The probability of Y being between $\frac{1}{2}$ and $\frac{3}{4}$ is approximately 11.8%. It can be calculated as follows:

$$P\left(\frac{1}{2} < Y < \frac{3}{4}\right) = P\left(\left\{Y < \frac{3}{4}\right\} - \left\{Y < \frac{1}{2}\right\}\right) = P\left(Y < \frac{3}{4}\right) - P\left(Y < \frac{1}{2}\right) = F\left(\frac{3}{4}\right) - F\left(\frac{1}{2}\right) =$$

$$= \left(1 - e^{-3\cdot\frac{3}{4}}\right) - \left(1 - e^{-3\cdot\frac{1}{2}}\right) = 0.11773$$

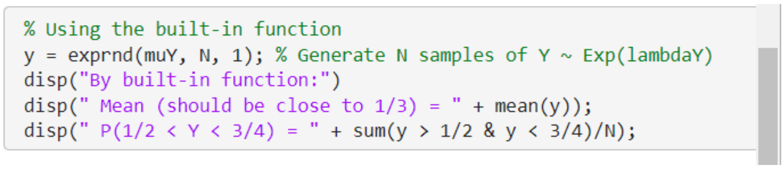


```
disp("b) Duration of connection, between half an hour and three quarters of an hour");
% Y = Duration of connection, in hours
% density funciton f(y) = {3 * e^(-3 * y)  if y >= 0
%                         {     0          otherwise
% Y is just an exponential random variable with lambdaX = 3
% F(y) = 1 - e^(-3 * y)
lambdaY = 3; % 1/hours
muY = 1/lambdaY; % hours

% Theoretical calculation
disp("Theoretical calculation:")
disp(" Mean = " + muY);
disp(" P(1/2 < Y < 3/4) = F(3/4) - F(1/2) = " + (1 - exp(-lambdaY * 3/4) - (1 - exp(-lambdaY * 1/2))));
```

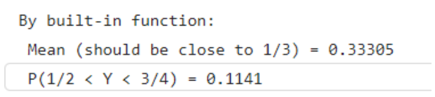


```
b) Duration of connection, between half an hour and three quarters of an hour

Theoretical calculation:
 Mean = 0.33333
 P(1/2 < Y < 3/4) = F(3/4) - F(1/2) = 0.11773
```

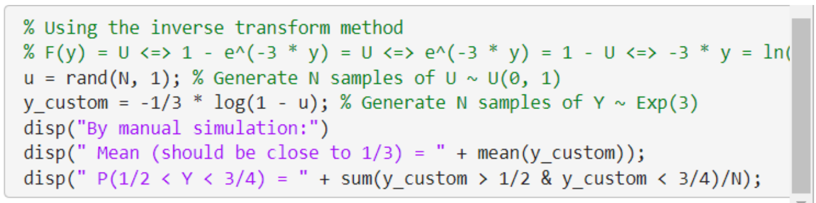
Once again, this experiment was simulated both using the built-in function `exprnd` and the inverse transform method.



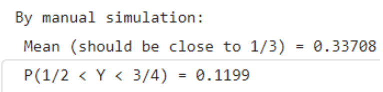
```
% Using the built-in function
y = exprnd(muY, N, 1); % Generate N samples of Y ~ Exp(lambdaY)
disp("By built-in function:")
disp(" Mean (should be close to 1/3) = " + mean(y));
disp(" P(1/2 < Y < 3/4) = " + sum(y > 1/2 & y < 3/4)/N);
```



```
By built-in function:
 Mean (should be close to 1/3) = 0.33305
 P(1/2 < Y < 3/4) = 0.1141
```



```
% Using the inverse transform method
% F(y) = U <=> 1 - e^(-3 * y) = U <=> e^(-3 * y) = 1 - U <=> -3 * y = ln(
u = rand(N, 1); % Generate N samples of U ~ U(0, 1)
y_custom = -1/3 * log(1 - u); % Generate N samples of Y ~ Exp(3)
disp("By manual simulation:")
disp(" Mean (should be close to 1/3) = " + mean(y_custom));
disp(" P(1/2 < Y < 3/4) = " + sum(y_custom > 1/2 & y_custom < 3/4)/N);
```



```
By manual simulation:
 Mean (should be close to 1/3) = 0.33708
 P(1/2 < Y < 3/4) = 0.1199
```

The results were quite close to the expected value: 11.41 % and 11.99%

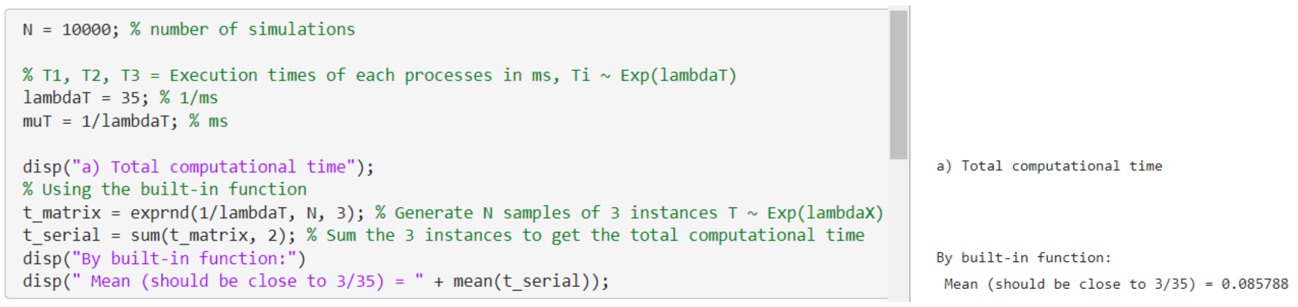
EXERCISE 3.

A computational study is based on three processes running in series. Let T_1, T_2 and T_3 be the execution time for processes 1, 2 and 3 respectively, and let T be the total execution time of the computational study. Assume that T_1, T_2 and T_3 are independent random variables and identically distributed by an Exponential distribution with parameter $\lambda = 35$ millisecs^{-1}.

a) Compute by simulation using MATLAB/Octave the average time of execution of the computational study.

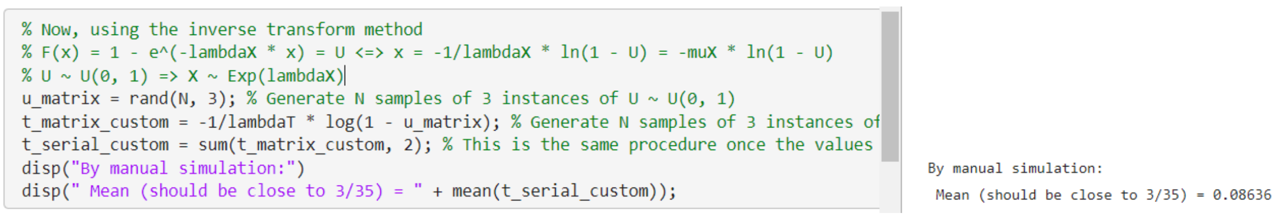
The simulation was performed by generating random values for the variables T_1, T_2 and T_3. They were all stored in the matrix `t_matrix`, with one row per simulation and one column per process. Since the processes run in series, the second process runs when the first one ends, and the third one starts when the second one ends, meaning that the total time is the sum of the time taken by each one (assuming there is no time in between), so $T = T_1 + T_2 + T_3$. Using this, the matrix was turned into a one-dimensional vector `t_serial`, representing the total time running all 3 processes in series would take for each simulation. Finally, the mean was calculated with the `mean` function.

The simulations was first done using the built-in function `exprnd`, and then with the inverse transform method. The theoretical value was also calculated by simply calculating $3 \cdot \frac{1}{\lambda} = \frac{3}{35} = 0.08571$ in order to catch simple mistakes early.



```
N = 10000; % number of simulations

% T1, T2, T3 = Execution times of each processes in ms, Ti ~ Exp(lambdaT)
lambdaT = 35; % 1/ms
muT = 1/lambdaT; % ms

disp("a) Total computational time");
% Using the built-in function
t_matrix = exprnd(1/lambdaT, N, 3); % Generate N samples of 3 instances T ~ Exp(lambdaX)
t_serial = sum(t_matrix, 2); % Sum the 3 instances to get the total computational time
disp("By built-in function:")
disp(" Mean (should be close to 3/35) = " + mean(t_serial));
```

```
a) Total computational time

By built-in function:
 Mean (should be close to 3/35) = 0.085788
```



```
% Now, using the inverse transform method
% F(x) = 1 - e^(-lambdaX * x) = U <=> x = -1/lambdaX * ln(1 - U) = -muX * ln(1 - U)
% U ~ U(0, 1) => X ~ Exp(lambdaX)
u_matrix = rand(N, 3); % Generate N samples of 3 instances of U ~ U(0, 1)
t_matrix_custom = -1/lambdaT * log(1 - u_matrix); % Generate N samples of 3 instances of
t_serial_custom = sum(t_matrix_custom, 2); % This is the same procedure once the values
disp("By manual simulation:")
disp(" Mean (should be close to 3/35) = " + mean(t_serial_custom));
```

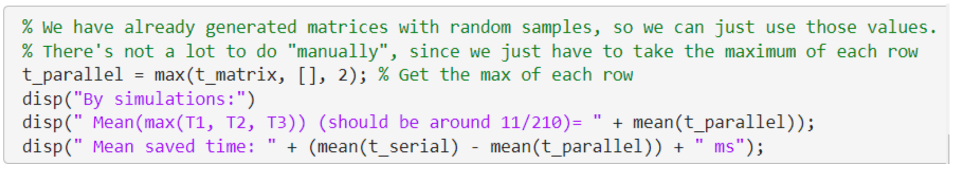
```
By manual simulation:
 Mean (should be close to 3/35) = 0.08636
```

As we can see, both values are close to the theoretical value.

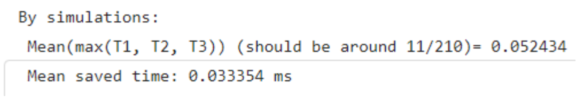
b) Assume that the three processes can be parallelized, i.e. processes 1, 2 and 3 can be run in parallel to complete the computation study. How much time would be saved to perform the computational study as opposed to what would be the case if the process must be executed in series?

In this case, all three processes can be started simultaneously, and the computation study will finish when all three are complete. The final time will be equal to the time of the one which takes the longest. In this case, $T = max\left(T_1, T_2, T_3\right)$

For this simulation, the random value matrix from the previous section was reused, but instead of adding all three values in each simulation, the matrix was reduced to a one-dimensional vector by taking the maximum value in each row. The simulated improvement in time can be calculated by taking the difference between the mean total time with the processes running in series and the mean total time with the processes running in parallel.



```
% We have already generated matrices with random samples, so we can just use those values.
% There's not a lot to do "manually", since we just have to take the maximum of each row
t_parallel = max(t_matrix, [], 2); % Get the max of each row
disp("By simulations:")
disp(" Mean(max(T1, T2, T3)) (should be around 11/210)= " + mean(t_parallel));
disp(" Mean saved time: " + (mean(t_serial) - mean(t_parallel)) + " ms");
```



```
By simulations:
 Mean(max(T1, T2, T3)) (should be around 11/210)= 0.052434
 Mean saved time: 0.033354 ms
```

With this, we have an average saved time of 0.03335 ms.

The theoretical mean value was also calculated in order to check the results of the simulation.

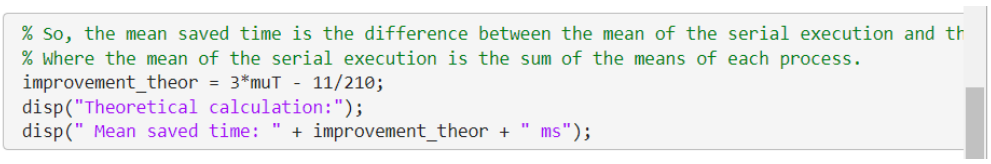
$$T = max\left(T_1, T_2, T_3\right) \Rightarrow F_T(t) = P(T \leq t) = P(max\left(T_1, T_2, T_3\right) \leq t) =$$

$$= P\left(T_1 \leq t \cap T_2 \leq t \cap T_3 \leq t\right) = P\left(T_1 \leq t\right) * P\left(T_2 \leq t\right) * P\left(T_3 \leq t\right) = F_{T1}(t)^3$$

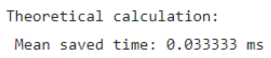
$$f_T(t) = \frac{d}{dt} F_T(t)$$

$$\int_{-\infty}^{\infty} t * f_T(t) dt$$

Instead of calculating this by hand, I used Python to calculate the result of the integral (more details in the files), which returned the theoretical expected time of $\frac{11}{210} = 0.5238$. The theoretical improvement in time was also found by taking the difference of this new time with the old theoretical time.



```
% So, the mean saved time is the difference between the mean of the serial execution and th
% Where the mean of the serial execution is the sum of the means of each process.
improvement_theor = 3*muT - 11/210;
disp("Theoretical calculation:");
disp(" Mean saved time: " + improvement_theor + " ms");
```



```
Theoretical calculation:
 Mean saved time: 0.033333 ms
```

c) If you could choose between a system running in series but where each process has parameter λ = 40 millisecs-1 (i.e. each one runs faster) or the old system (each process with λ = 35 millisecs^{-1}) but parallelized, which one would you select and why?

In order to determine the best option, the new configuration was simulated as well, and then the results were compared. The procedure was as simple as the first section of this exercise, changing the value of the parameter of the distribution from $\lambda_X = 35$ to $\lambda_X' = 40$.

```
disp("c) Serial with λ = 40 vs parallel with λ = 35");
% Let's create the new matrix with lambda = 40
% Using the built-in function, I don't think there's any need to do this manually anymore.
lambdaT_ = 40;
t_matrix_ = exprnd(1/lambdaT_, N, 3); % Generate N samples of 3 instances T ~ Exp(lambdaX)
t_serial_ = sum(t_matrix_, 2); % Sum the 3 instances to get the total computational time
disp("Mean of serial execution with λ = 40 (should be close to 3/40): " + mean(t_serial_));
disp("Mean of parallel execution with λ = 35: " + mean(t_parallel));
```

```
c) Serial with λ = 40 vs parallel with λ = 35

Mean of serial execution with λ = 40 (should be close to 3/40): 0.074933
Mean of parallel execution with λ = 35: 0.052409
```

The parallel execution with slower time for each process turned out to still be faster than the series execution with slightly faster time for each process, so the option I would choose if possible would be the parallel processes. The theoretical calculations are not necessary to determine the winner between both options.