INSTITUTO SUPERIOR TÉCNICO

TRAFFIC ENGINEERING

METI

Lab Report I

Analysis of queues and Poisson distributions

2018/2019

Grupo 6
André Mendes - 78079
Filipe Fernandes - 78083

1 Introduction

The goal of this laboratory guide is to simulate, analyze and discuss the properties of Poisson arrival processes and its application to queue processing. We are students attending a master degree in telecommunications and informatics engineering so we understand the impact and importance that these theme may have not only in this course but also in our future as engineers.

This report is composed by three sections. We start by explaining the code developed and used algorithms, on the second section we analyze and discuss obtained results. We conclude the report by stating main conclusions of this work.

2 Developed code and used algorithms

For this work all code were written in Matlab R_2017_b, all source code are available on appendix.

2.1 Arrival process generation

For the arrival process generation we started by writing the code of exponential distribution (δt) with parameter λ. Then we developed an algorithm to simulate a Poisson process (annex A).

Firstly we generated a sequence of 50 events with time interval δt between successive events and parameter $\lambda = 5$ obtaining the results in Figures 1 and 2.

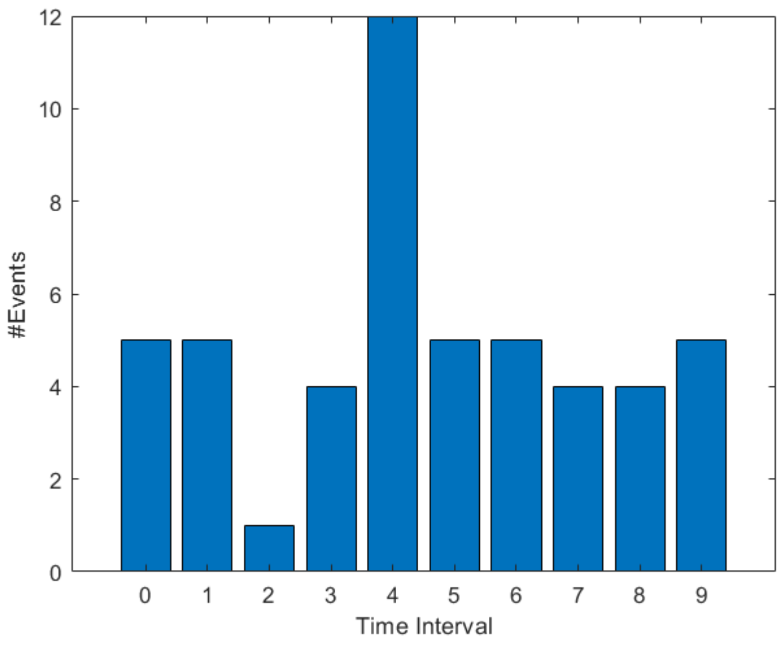

Figure 1: Number of events of Poisson process per time interval with $N = 50$ and $\lambda = 5$.

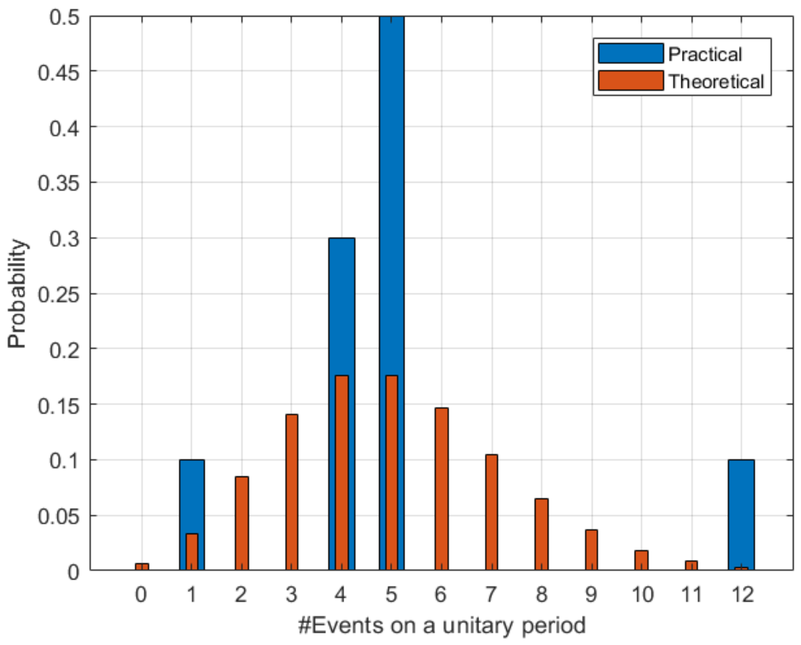

Figure 2: Practical vs Theoretical Values of Poisson process with $N = 50$ and $\lambda = 5$.

The Poisson process was simulated using an algorithm which receives as arguments parameters n (number of events) and comparison (from exponential distribution). We created a cycle with n iterations (events) where in each one we stored the temporal instances on a list (lambdat variable in annex A) from the following formula:

$$\delta t = -\frac{\log(1-u)}{\lambda}$$

Then we counted the number of events on a time interval (Figure 1) and using these values (stored in the nrocorrencias variable in annex A) calculated the probability of observing a given number of events on a unitary period with the formula below.

$$P(X = k) = \frac{\lambda^k}{k!}e^{-\lambda}$$

2.2 Superposition of Poisson processes

Using the code developed on the section above we were asked to develop a similar algorithm in which three Poisson processes (different λ values) would be super-positioned. In this case the difference is that we need calculate the number of iterations (events) which depends on the highest λ because it limits the number of events to be simulated. For this we used the equation: *ceil(*λ **(n/*λ*maximum))*. After knowing the number of events to be simulated, the process of the previous section was repeated on this one and Figure 6 was obtained.

2.3 M/M/1 queue simulation

Now the objective is to calculate the average of elements in a MM1 system queue. The algorithm created for this purpose receives as an input: λ, μ and the maximum number of iterations to be done by the system (input N in annex C). This iterations can be element arrivals or departures to the queue. Two lists were created with the number of maximum iteration as length, that store the arrivals and departures total times (variables total departures and total arrivals in annex C). Finally for each time instance, the number of elements in the queue were saved in a list (variable queue size per time instant in annex C). To find out the average number of elements in the queue, the sum of all of them was divided for the total number of time instances.

3 Obtained Results

In this section we will present the obtained results using the algorithms we described on the previous section.

3.1 Arrival process generation

On previous section, in Figures 1 and 2, we presented the results of Poisson process with N = 50. We conclude that the Poisson curve isn't acceptable because of insufficient number of events. So we ran again our algorithm but this time with N = 20000 and keeping $\lambda = 5$. The result is displayed on figure 3. Using 20000 events we can see that the Poisson curve is much more accurate, and the practical and theoretical values are now much closer than in figure 2.

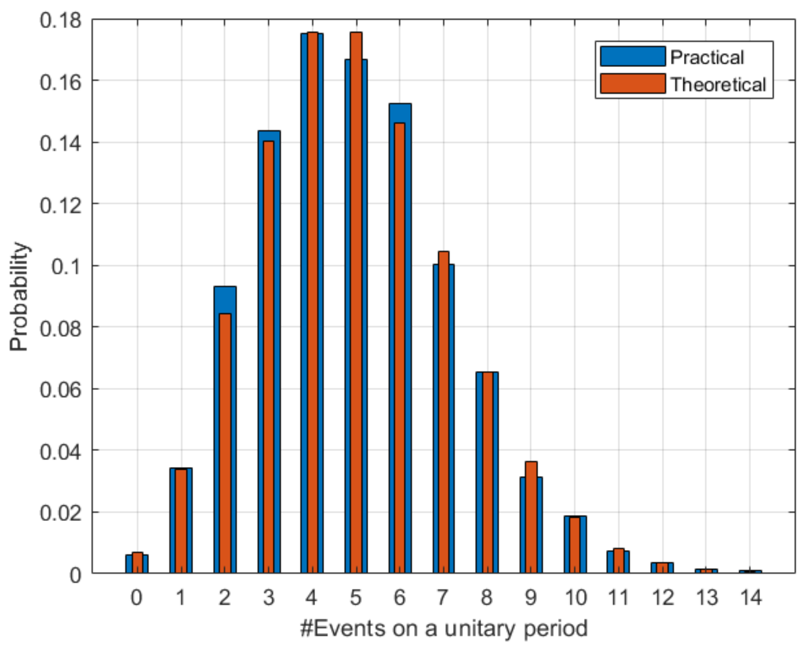

Figure 3: Practical vs Theoretical Values of Poisson process with N = 20000 and $\lambda = 5$

To better understand the arrival process generation we ran again our algorithm keeping the number of events but varying λ parameter. Using $\lambda = 10$ and $\lambda = 1$, results are showed in figures 4 and 5, respectively.

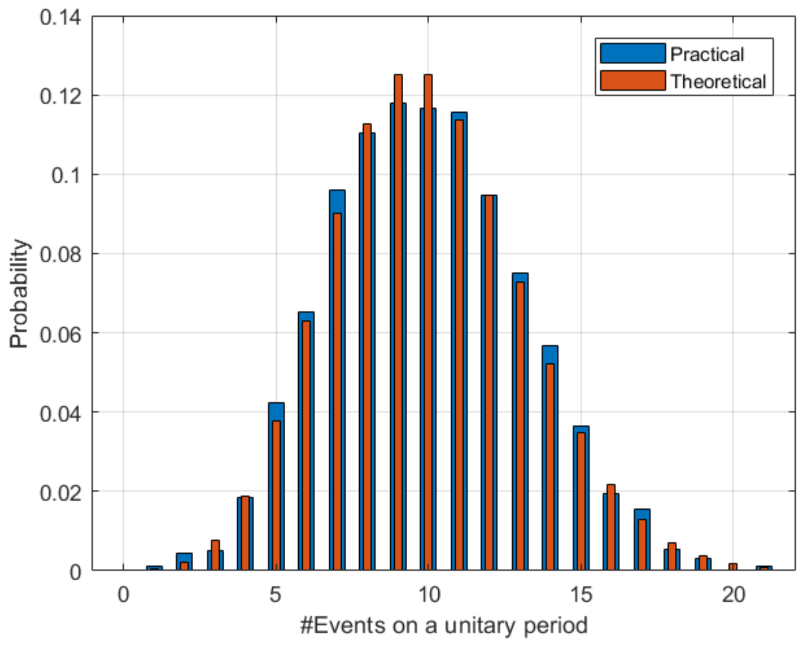

Figure 4: Practical vs Theoretical Values of Poisson process with N $= 20000$ and $\lambda = 10$

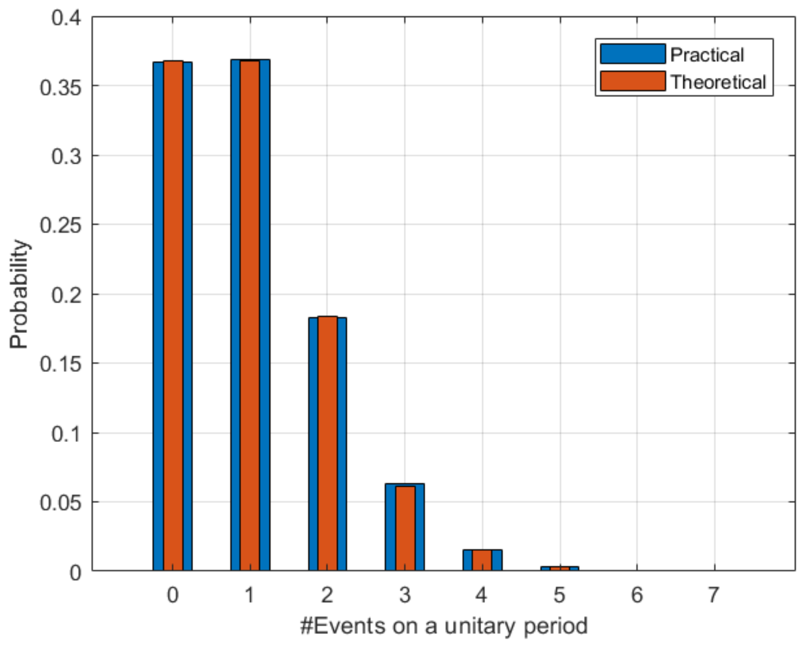

Figure 5: Practical vs Theoretical Values of Poisson process with N $= 20000$ and $\lambda = 1$

By analyzing the results we can see that the higher probability reached has bigger values if the λ parameter is smaller.

3.2 Superposition of Poisson processes

The results obtained for the superposition of Poisson processes with N = 20000 and $\lambda = 1$, $\lambda = 5$, $\lambda = 10$ are displayed on figure 6.

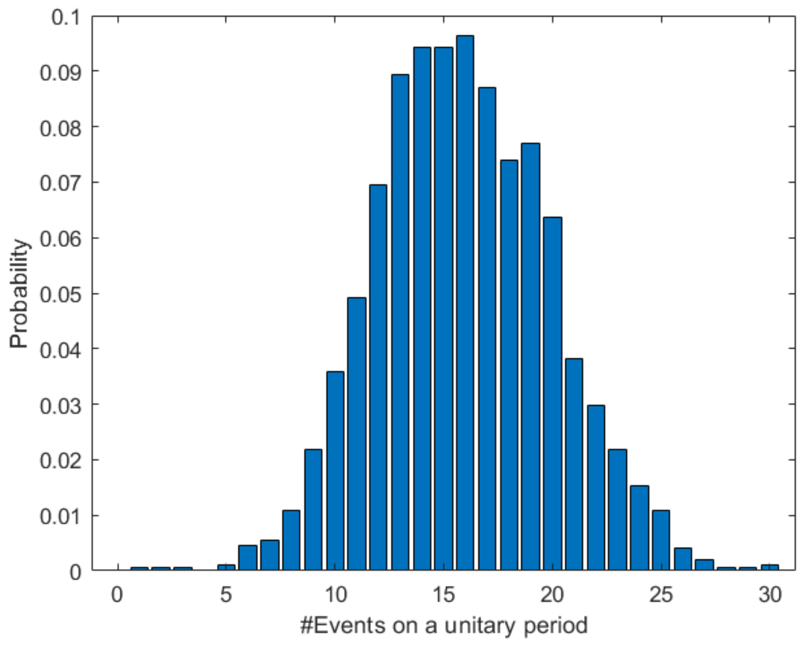

Figure 6: Practical vs Theoretical Values of Poisson process with N $= 20000$, $\lambda = 1$, $\lambda = 5$, $\lambda = 10$

As expected the superposition of Poisson processes is also a Poisson process in this case with $\lambda = 16$.

3.3 M/M/1 queue simulation

On figures 7, 8 and 9 we can observe the average queue size of M/M/1 simulator for different input parameters.

For the results displayed below we used as input N = 10000 events and tried different values for λ and μ.

On the next table we present the practical and theoretical results we obtained for each input parameter. The practical results were achieved using the algorithm we developed and the theoretical values were calculated using the following formula: $\lambda/(\mu - \lambda)$.

Average time on queue	$\lambda = 19$, $\mu = 20$	$\lambda = 20$, $\mu = 20$	$\lambda = 20$, $\mu = 19$
Practical	14.4708	52.8063	190.3968
Theoretical	19	Inf	-20

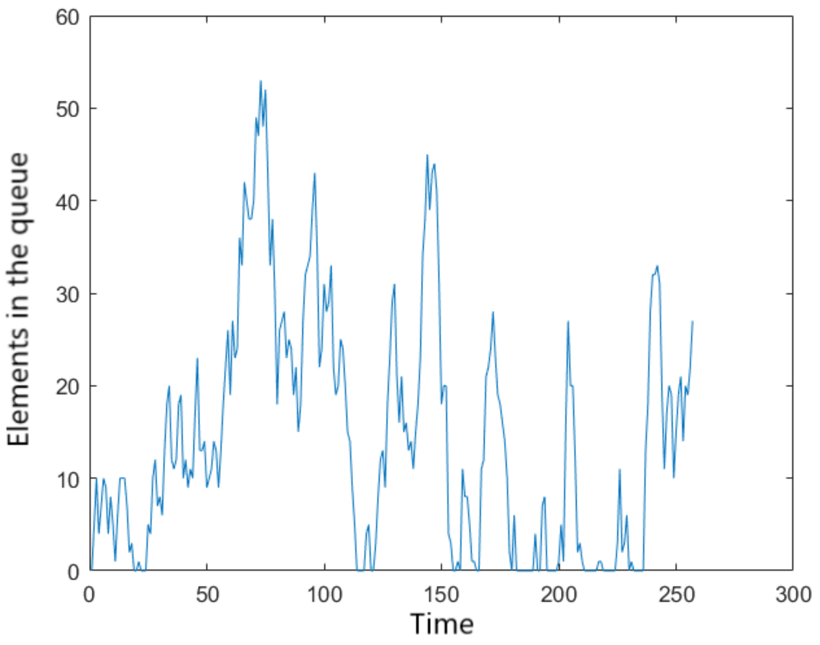

Figure 7: Number of elements on queue per time instant with $\lambda = 19$, $\mu = 20$

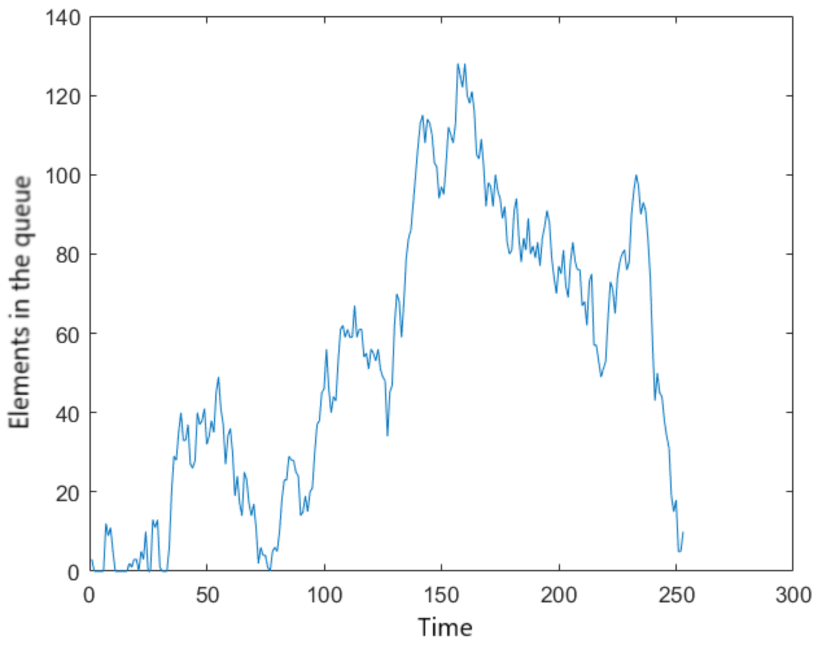

Figure 8: Number of elements on queue per time instant with $\lambda = 20$, $\mu = 20$

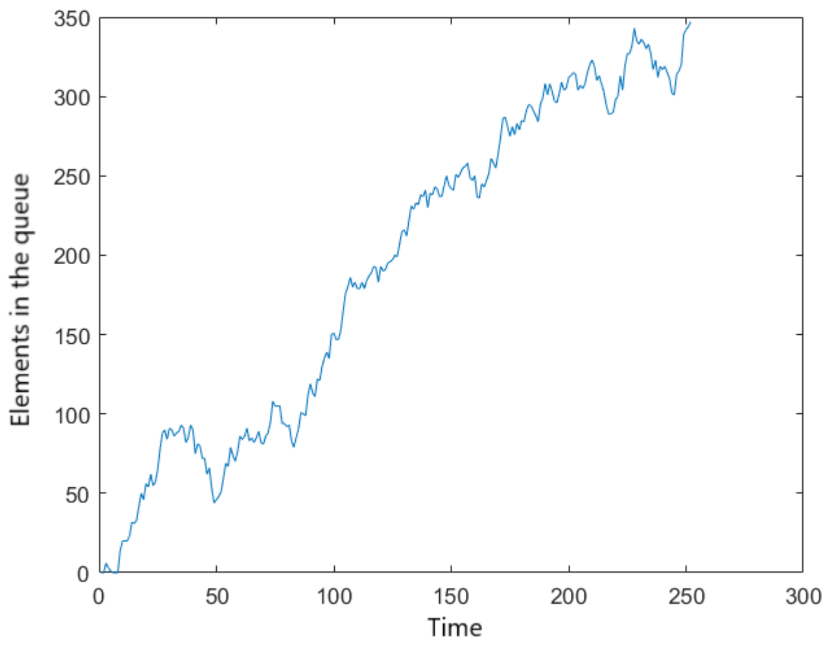

Figure 9: Number of elements on queue per time instant with $\lambda = 20$, $\mu = 19$

4 Conclusions

Regarding the Poisson process we conclude that the number of events is crucial to the success of the simulation. As we have observed, with only N = 10 events the simulation isn't much accurate in comparison with the theoretical values. But with a bigger number of events like N = 20000, theoretical and experimental values are pretty close. We also conclude that the Poisson curve gets smaller to events with `unitary period` $> \lambda$.

By thinking of a superposition of Poisson processes we know that this is also a Poisson process. So as expected the λ parameter of the superposition is the sum of all λ's. In this case: $\lambda = \lambda 1 + \lambda 2 + \lambda 3$; $\lambda = 10 + 5 + 1 = 16$.

Lastly, the analysis of the M/M/1 queue let us observe that the increment of the arrival rate keeping the same processing rate, $\lambda > \mu$ will generate a bigger queue until it gets limited by the memory capacity of the server. For this type of queue we conclude and it only has a good behaviour if $\lambda < \mu$, even if both parameters have the same value the behaviour is uncertain.

Annex:

A

Annex A - Poisson process

```
function lambdat = ex22(n,lambda)
    i = 1;
    lambdat = 1:n;
    while i <= n
        if i == 1
            number = rand;
            lambdat(i) = - log (1 - number) / lambda;
        else
            number = rand;
            lambdat(i) = lambdat(i-1) + (- log (1 - number) / lambda)
                ;

        end
        i = i + 1;

    end

    nrocorrencias = 1:ceil(lambdat(n));
    nrocorrencias = zeros(1, ceil(lambdat(n)));
    ii = 1;
    while ii <= n
        index=ceil(lambdat(ii));
        nrocorrencias(index)=nrocorrencias(index)+1;
        ii = ii + 1;
    end
    zerosss = 0:ceil(lambdat(length(lambdat)))-1;
    figure
    bar(zerosss,nrocorrencias)
    ylabel('#Events')
    xlabel('Time␣Interval')


    pratical = probability(nrocorrencias);

    theoretical= 1:length(pratical);
    iii = 1;
    while iii <= length(theoretical)
        theoretical(iii)= poissonarrival(iii-1,lambda);
        iii = iii + 1;

    end

    zeross = 0:max(nrocorrencias);
    figure
    w1=0.5;
    bar(zeross,pratical,w1)
    w2=0.25;
    hold on
    bar(zeross ,theoretical ,w2)
    hold off
    grid on
    ylabel('Probability ')
    xlabel('#Events␣on␣a␣unitary␣period ')
    legend({'Practical ','Theoretical '},'Location ','northeast ')

end



function result = poissonarrival(k,lambda)

result = lambda^k / factorial(k)* exp(-lambda);

end


function probabilities = probability(array)
    probabilities= 1:max(array)+1;
    count=0;
    i = 0;
    while i <= max(array)
        j = 1;
        while j <= length(array)
            if i==array(j)
                count = count + 1;
            end
            j = j + 1;
        end


        probabilities(i+1)=count/length(array);
        count=0;
        i = i + 1;
    end

end
```

B Annex B - Superposition of Poisson processes

```
function lambdat = ex23(n,lambda1 ,lambda2 ,lambda3)
    lambdat = zeros(3,n);
    j = 1;
    lambdas = [lambda1 ,lambda2 ,lambda3];
    time=n/max(lambdas);
    limit(1)= ceil(lambda1*time);
    limit(2)= ceil(lambda2*time);
    limit(3)= ceil(lambda3*time);
    while j <= 3
        i = 1;
        while i <= limit(j)
            if  i==1
                number1 = rand;
                lambdat(j,i) = - log (1 - number1) / lambdas(j);
            else
                number2 = rand;
                lambdat(j,i) = lambdat(j,i-1) + (- log (1 - number2)
                   / lambdas(j));
            end
            i = i + 1;

        end
        j = j + 1;
    end

    lambdat=sort(lambdat(lambdat>0));
    nrocorrencias = 1:ceil(lambdat(length(lambdat)));
    nrocorrencias = zeros(1, ceil(lambdat(length(lambdat))));
    ii = 1;
    while ii <= length(lambdat)
        index=ceil(lambdat(ii));
        nrocorrencias(index)=nrocorrencias(index)+1;
        ii = ii + 1;
    end
    pratical = probability(nrocorrencias);
    zeross = 0:max(nrocorrencias);
    bar(zeross,pratical)
    ylabel('Probability')
    xlabel('#Events␣on␣a␣unitary␣period')
end

function probabilities = probability(array)
    probabilities= 1:max(array)+1;
    count=0;
    i = 0;
    while i <= max(array)
        j = 1;
        while j <= length(array)
            if i==array(j)
                count = count + 1;
            end
            j = j + 1;
        end


        probabilities(i+1)=count/length(array);
        count=0;
        i = i + 1;
    end
end
```

C Annex C - M/M/1 Queue Simulation

```
function ex3(N, lambda ,u)

    arrivals = 1:N;
    departures = 1:N;

    for i = 1:(N)
        u1 = rand;
        arrivals(i) = -(log(1-u1))/lambda;
        u2 = rand;
        departures(i) = -(log(1-u2))/u;
    end

    total_arrivals = 1:N;
    total_departures = 1:N;

    for i = 1:(N)
        if i==1
            total_arrivals(i) = arrivals(i);
            total_departures(i) = departures(i);
        else
           total_departures(i) = total_departures(i-1) + departures(i
              );
           total_arrivals(i) = total_arrivals(i-1) + arrivals(i);
        end
    end

    queue_size_per_time_instant = 1:(ceil(sum(departures)));
    queue_size = 0;
    arrivals_index = 1;
    departures_index = 1;
    total_index = 1;
    limit_events_flag = 0;


    for i = 1:ceil(max(total_arrivals))
        while (arrivals_index <= N) && (total_arrivals(arrivals_index
           ) <= i)
            if (total_index <= N)
                queue_size = queue_size + 1;
            else
                limit_events_flag = 1;
            end
            arrivals_index = arrivals_index + 1;
            total_index = total_index + 1;
        end

        while (departures_index <= N) && (total_departures(
           departures_index) <= i)
            if (total_index <= N)
```

```
                queue_size = queue_size - 1;
            else
                limit_events_flag = 1;
            end
            departures_index = departures_index + 1;
            total_index = total_index + 1;
        end

        if queue_size < 0
            queue_size = 0;
        end

        if limit_events_flag == 1
            break;

        end

        queue_size_per_time_instant(i) = queue_size;

    end

    queue_size_per_time_instant_final = 1:(i-1);

    for i = 1:(i-1)
        queue_size_per_time_instant_final(i) =
           queue_size_per_time_instant(i);
    end


    disp('Pratical␣Average')
    avrg = sum(queue_size_per_time_instant_final)/length(
       queue_size_per_time_instant_final);
    disp(mean(avrg));
    plot(queue_size_per_time_instant_final);
    disp('Theoretical␣Average')
    theo_result = lambda/(u-lambda);
    disp(theo_result);

end
```