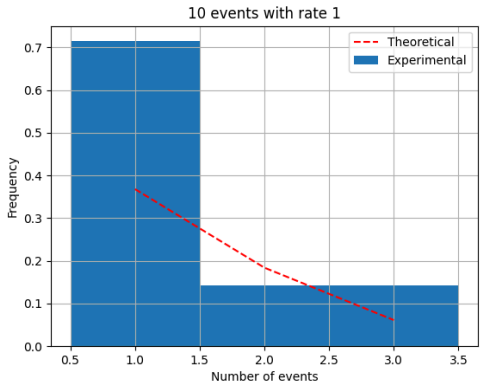
Traffic Engineering

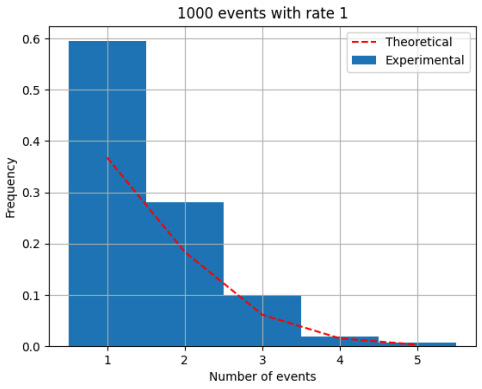
Lab 1 Report

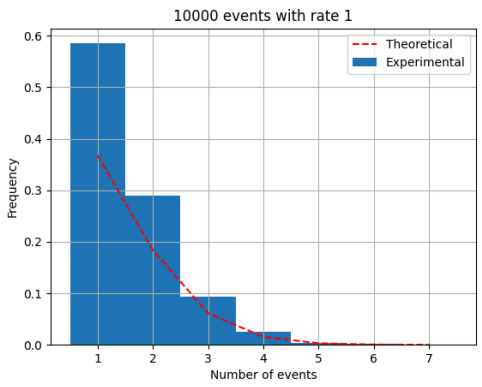
João Pargana, 93592
Samuel Barata, 94230

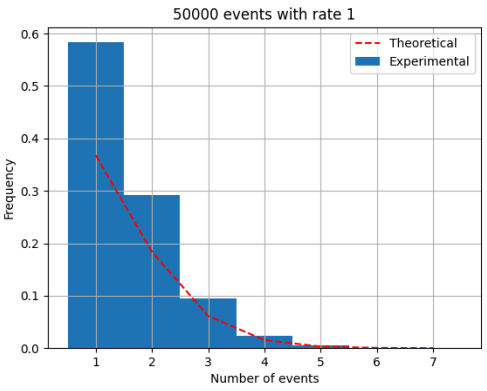
2.2 - Arrival process generation

The code developed is contained in the method *dois_dois* attached in a file called *main.py*

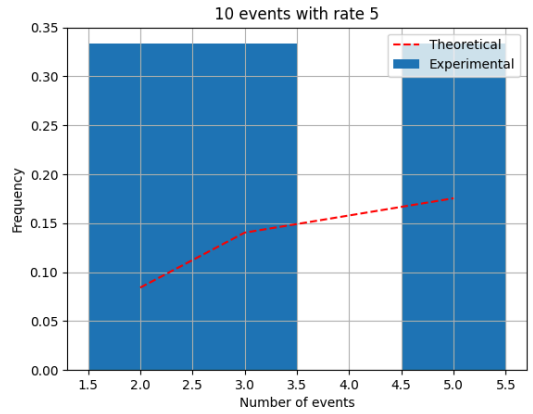


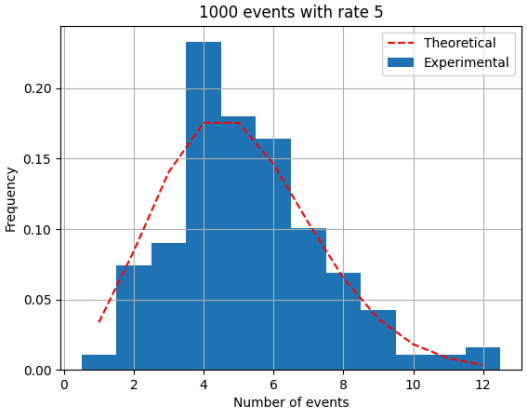


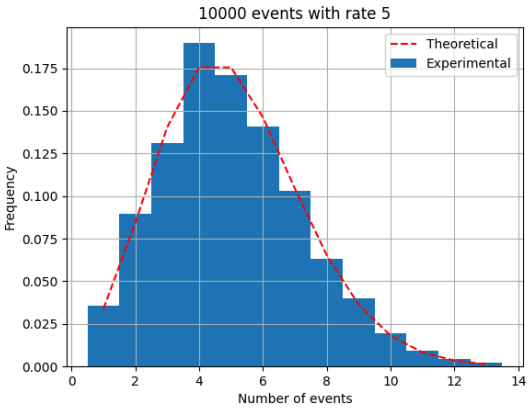


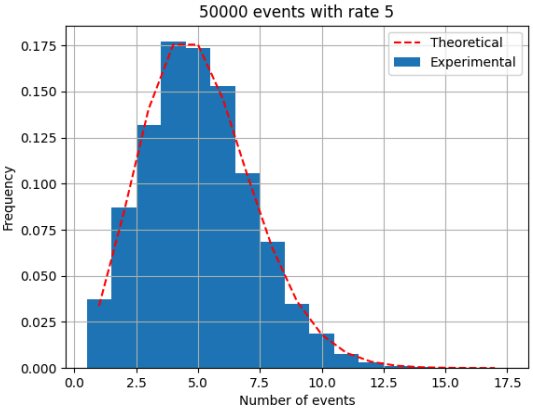


λ = 1. Here we can see that the experimental values obtained approximately match the theoretical values. For increasing values of N, the experimental values better match the theoretical ones. With λ =1, however, it is always skewed to look like an exponential distribution.

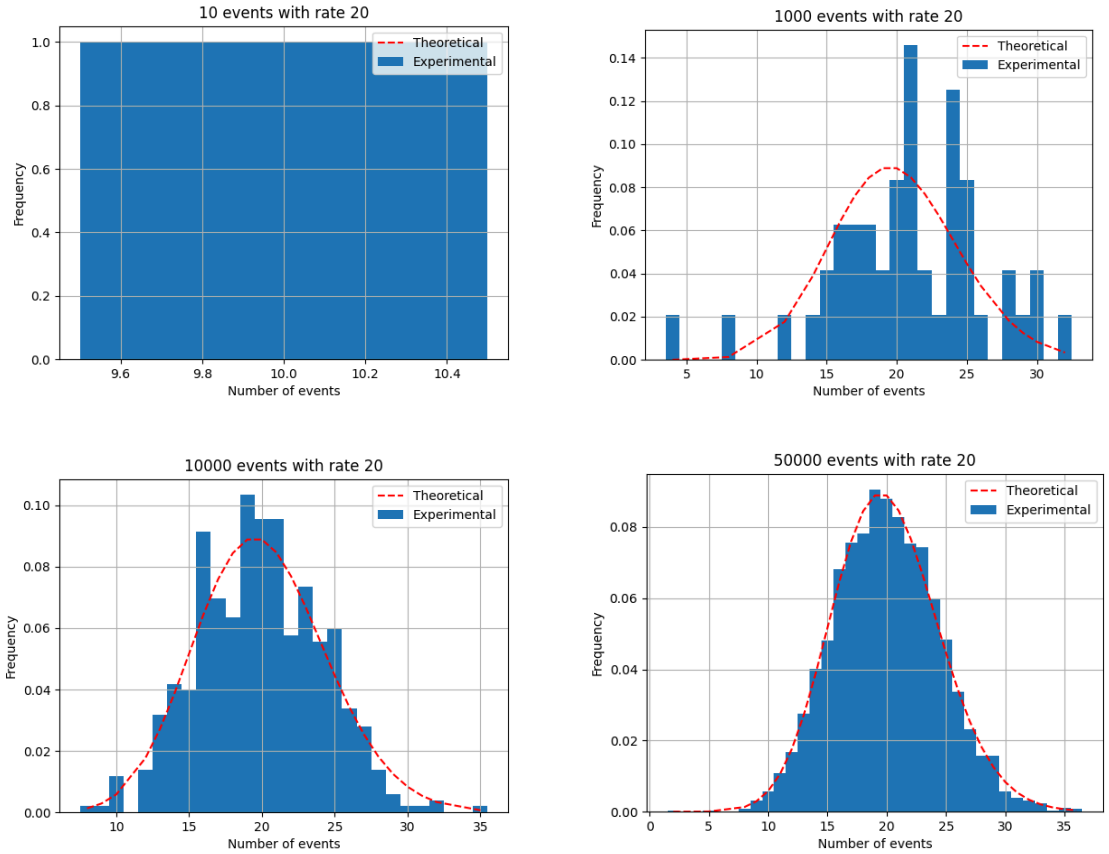








λ = 5. The plots here start to better represent a Poisson distribution as λ, and thus, the mean moves away from 1. As N increases, we still observe the experimental values better fitting the theoretical ones. At 10 events, we have the potential for high variance as all values hold a larger weight on the probability density which is why it appears as it does.



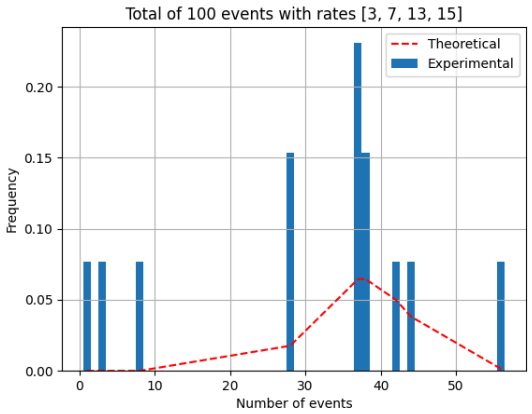
λ = 20. Again, we observe that as N increases, the experimental values better fit the theoretical ones. At 10 events, we still have the potential for high variance for the same reason as before, and at 1000 events, we can also observe the same thing, as 1000 events is still relatively little.

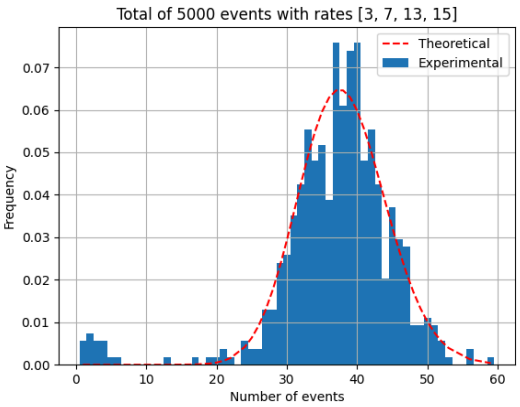
Conclusions

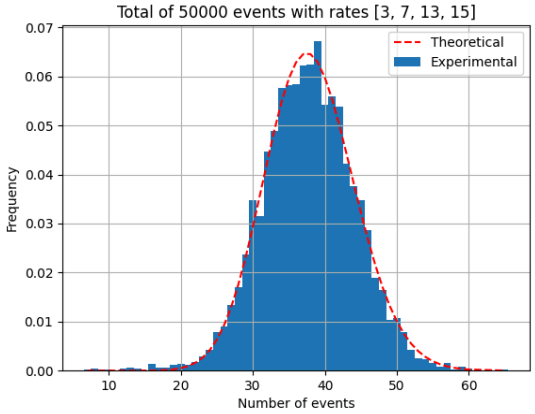
Low event counts leading to events having a higher weight on variance. Outliers, therefore, can stand out more than they would among a larger sample size. This is seen in the graphs where scaling event size leads to the experimental values having a better and smoother fit matching the theoretical values. We know that theoretically, the variance of a Poisson distribution is equal to λ, but with low event counts, the observed range is restricted due to small sample size. This means that even for high λ values, the spread of observed events can still be limited. This can be seen in the case of N=10, λ=20.
Increasing the λ values leads to a right shift of the bell curve, as the mean value increases accordingly. The distribution is more concentrated around λ and the expected probability of other events becomes more unlikely.

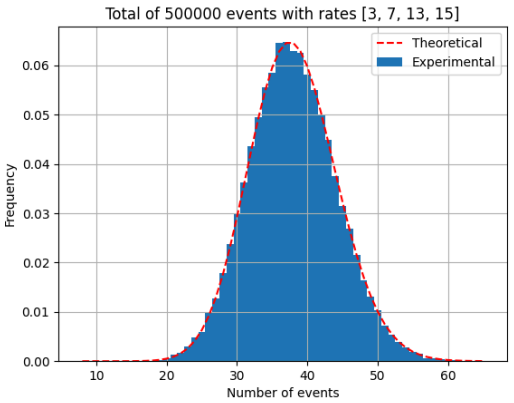
2.3 - Superposition of Poisson processes

The code developed is contained in the method *dois_tres* attached in a file called *main.py*









Conclusions

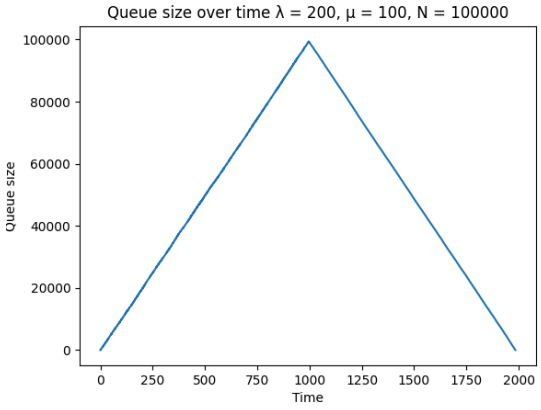
The λ of a superposition of Poisson processes results in a still Poissonian distribution with λ equal to the sum of the λ of all the Poisson processes that make it up. This can be seen in the observed and theoretical values where λ = 38. We ensured a balance between the Poisson processes by assigning each λ a weight equal to λ/(sum(λ)) which was then used to calculate how many events would be generated according to that distribution. Like in the previous section, a lower total event count leads to higher variance due to small sample size. As we increase the amount of simulated events, the observed values better fit the theoretical curve.

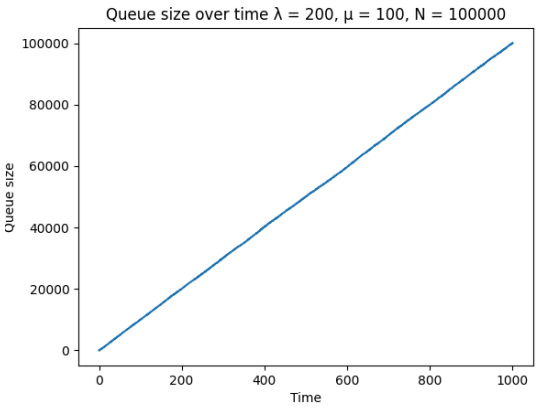
3 - M/M/1 queue simulation

The code developed is attached in a file called *mm1.py*

When we built our simulator we decided to stop generating packets at the specified count, but after analysing the results we concluded we had to change that since it would affect our statistics.
In the table below we can observe the differences in the average queue size, since after half the simulation the queue will be decreasing without new packets coming in (shown in the first graph). This will also affect average time in queue and consequently average time in the system.

N=100000 λ=100 μ=100	**Average time in system (Ws)**	**Average time in queue (Wq)**	**System utilisation (ρ)**	**Average queue size (Lq)**	**Events processed**
Finish processing	493.8539626	493.84401364	99.99951579	206521.55059592	199423/199423
Stop at 100000	250.34071092	250.33069756	99.9979656	49152.514718778	100000/200041

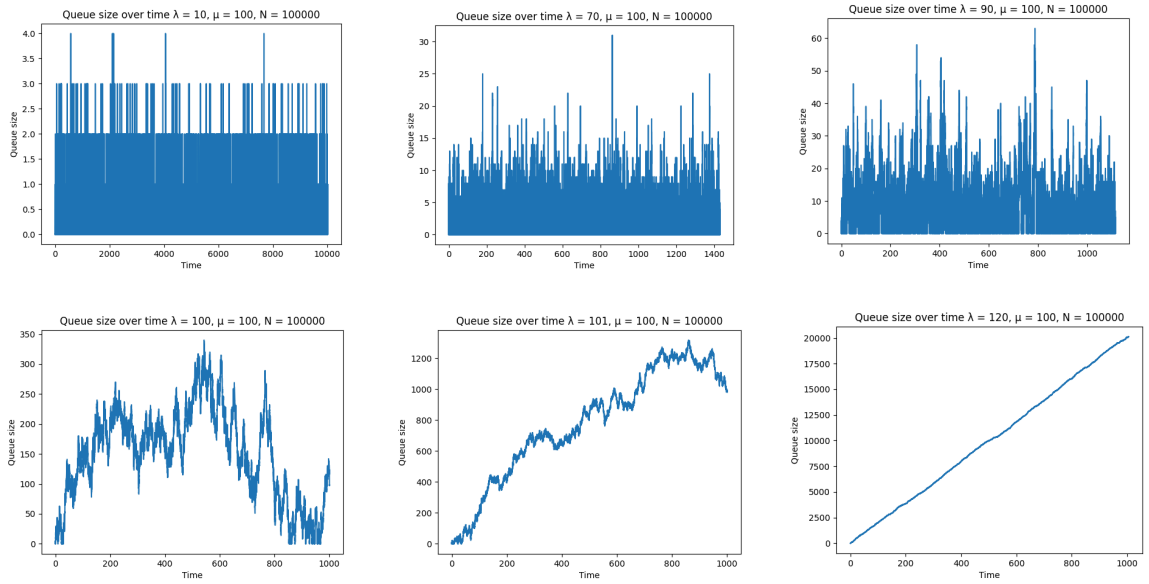




From here on out we'll always stop the simulations after N events have been processed.

3.1 - Experimentation with different service and arrival rates

100.000 events	Average time in system (Ws)		Average time in queue (Wq)		System utilisation (ρ)		Average queue size (Lq)	
λ = 10 μ =100	0.011149	0.01111	0.00112	0.0011	10.007%	10%	0.01113	.0111111
λ = 70 μ =100	0.033275	0.03333	0.02329	0.0233	69.7823	70%	1.6115	1.63333
λ = 90 μ =100	0.097245	0.1	0.08724	0.09	89.6931	90%	7.8053	8.1
λ = 100 μ =100	1.515012	inf	1.50505	inf	99.601%	100%	248.69	inf
λ = 101 μ =100	7.754840	-1	7.74485	-1.01	99.9%	101%	1032.2	-102.01
λ = 120 μ =100	83.698	-0.05	83.6883	-0.06	99.998%	120%	570088	-7.2

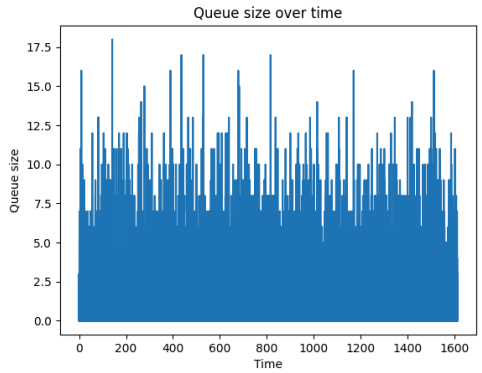


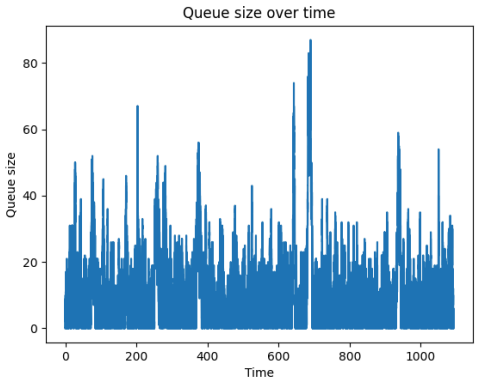
Analysing results

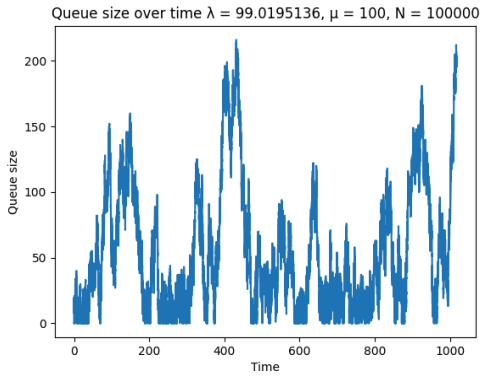
Just by looking at the graphs we can see a big difference between the top and bottom rows. The first three graphs have an arrival rate inferior to the service rate, this means that the server will be able to process events faster than they arrive, meaning the queue will be emptied multiple times.
The first graph of the second row has the arrival rate equal to service rate. As we can see the server struggles to keep up with arriving events but it's hard and only in a short period of time can it get the queue size back to 0.
In the last 2 graphs the server won't be able to keep up since events will be arriving faster than the speed they can be processed at.

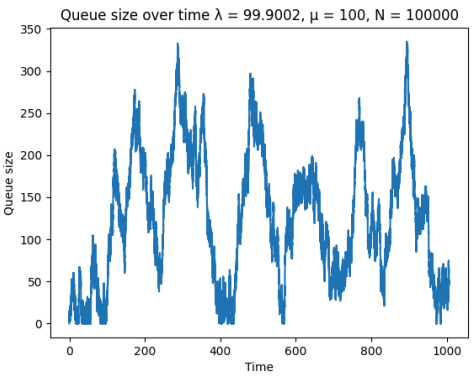
3.2 - Testing specific combinations for target queue sizes

100.000 events		Average time in system (Ws)		Average time in queue (Wq)		System utilisation (ρ)		Average queue size (Lq)	
λ=61.8034	**μ=100**	0.0266	0.0262	0.0166	0.0162	62.4%	61.1%	1.04	1
λ=91.608	**μ=100**	0.116	0.119	0.106	0.109	91.8%	91.6%	10.2	10
λ=99.0195136	**μ=100**	0.564	1.02	0.554	0.101	98.4%	99.0%	59.1	100
λ=99.9002	**μ=100**	1.27	10.0	1.26	10.0	99.5%	99.9%	208	1000





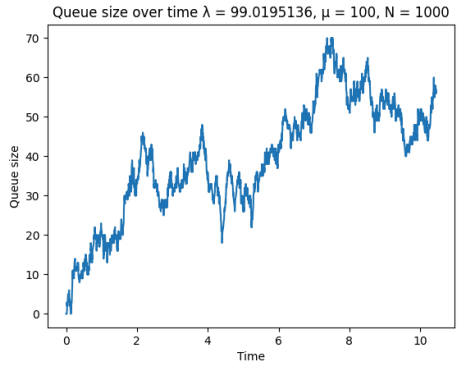


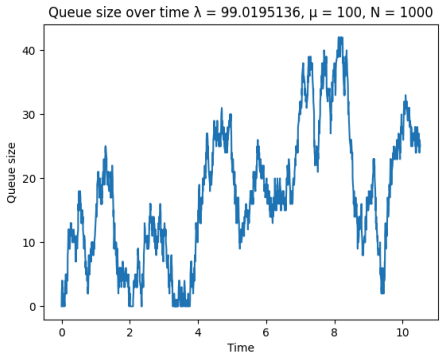


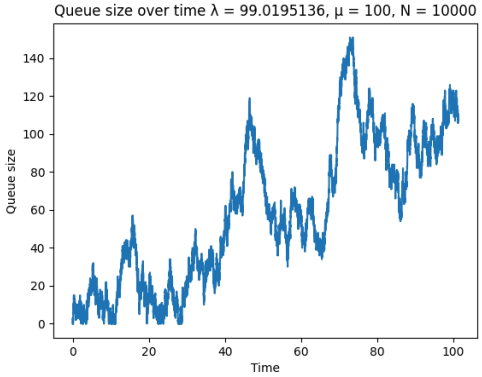
Analysing results

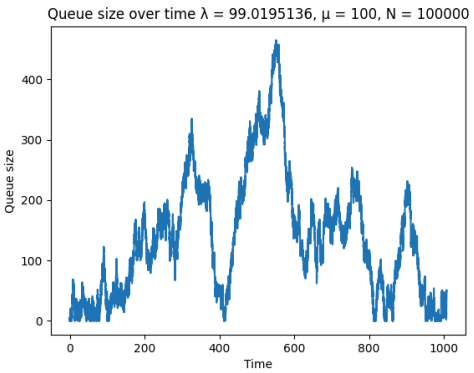
Analysing the results for Average queue sizes of 1, 10, 100, 1000 we can see that with queue size being 100 or more, due to the nature of queue size shrinking and growing by large amounts, we have a high variance. This can cause the experimental values to differ from the theoretical value.

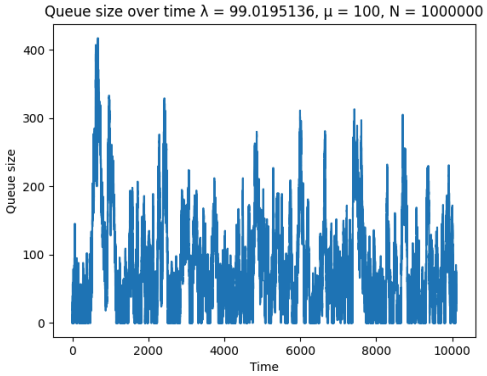
λ=99.0195136 μ=100	Average time in system (Ws)	Average time in queue (Wq)	System utilisation (ρ)	Average queue size (Lq)
N = 1.000	0.40682981	0.39638658	99.83098573%	589.66770217
N = 1.000	0.18952702	0.17923523	97.83830129%	44.28153274
N = 10.000	0.59547032	0.58543337	99.16812887%	118.21924538
N = 100.000	1.40238321	1.39237757	99.29757505%	140.37098867
N = 1.000.000	0.82788593	0.81789102	98.90061712%	88.97111497
N = 10.000.000	0.99496472	0.98496899	98.9561206%	93.8069459
Theoretical	1.01990195886	1.009901958864	99.0195136%	100

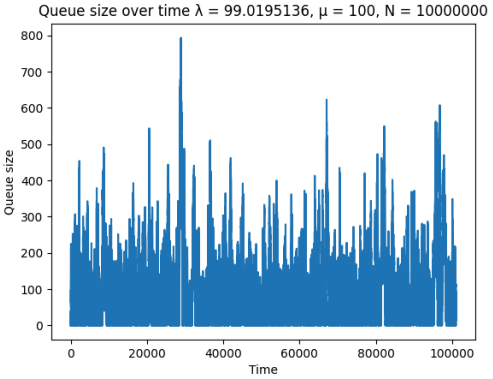












Analysing results

For an average queue size of 100 we used an arrival rate of 99.0195136 and a service rate of 100. Theoretically the server should be able to keep up with the requests but looking at the queue graphs we can see that with a low number of events the server rarely can get the queue back to 0.
We ran the test for the 1000 events twice since it differed so much from the theoretical values.
This is expected to happen since the arrival rate is almost equal to the service rate as we've seen back in section 3.1.
Only with 10.000.000 events did the simulation start to approximate theoretical results.