Assignment 2

2DI66

Advanced Simulation

Quartile 3, 2022-2023

Full Name	Student ID	Study
G. Nieuwenhuizen	1640534	OML
P.F.A. Hüttl	1909053	exchange student (Math & CS department)
L.C.W.A. Verstraete	1415492	ME + AI&ES

Group 11

Eindhoven, March 14, 2023

Contents

1 Introduction

In this report, a model is presented, which describes the process of students getting food in MetaForum's canteen. Students arrive in groups at the canteen, get food and go to one of the three cashiers to pay either by card or by cash. When they go to the cashiers, the students always pick the shortest queue. This model is described in section 2 and results are presented in section 3. Then the model gets three different extensions that are elaborated in section 4. Lastly, conclusions are drawn in section 5

2 Simulation description

2.1 Customers

The customer class consists of the $arrivaltime, arrive, takenFood$ and $cashcard$ functions. The function describes all the actions of the customer. The random numbers for the simulation were generated using functions from the numpy library. The arrival time of the groups is generated first. This is done by generating a random number from the Poisson distribution and then adding this to the last arrival time. The process is continued until the arrival time is equal or greater than the time the canteen is open. After the arrival times are calculated. The $arrive$ function gets the arrival times and picks a random number for the group's size from the geometric distribution. Both numbers are paired and added to a dictionary with the customer's number of arrival as the key. The $takeFood$ function picks a random number from the exponential distribution and added to the dictionary of each customer. The random number stands for the time it takes the customer to choose food and get it. And lastly, the $cashcard$ function chooses a random number between 0 and 1 from a uniform distribution. If the number is below the $percentagecash$, the customer pays with cash. Otherwise, they pay with a card.

2.2 Service

The service class describes the process from arriving at the queues to finishing the payment.

Firstly the customer is assigned to the shortest queue in the *assignToQueue(queueInfo, customerInfoIndiv)* function. This is done by checking the length of the lists within *queueInfo*. *QueueInfo* is a list that contains a list per queue within which gives the numbers of the customers that are in the queue. The queue with the shortest length is chosen and if there are multiple shortest queues, a random one is picked. Then the queue number is added to the customer's individual info (*customerInfoIndiv*) and the customer's number of arrival is added to the assigned queue in *queueInfo*.

Then the waiting time of the customer is calculated with the *waitQueue(queueInfo, customerInfoIndiv, timePayment)* function. The *timePayment* list has a list for each queue with the times at which the payment for a customer is finished. It is assumed that the customer moves infinitely fast to its queue, so after the food is taken it immediately arrives at the queue.

Algorithm 1 waitQueue

```
if len(customer's queue) == 1 then
|  waittime = 0
else
|  waittime = time finished payment of person in front of customer - time customer arrives at queue
end
 append waittime to the customerInfo
```

The *servicetime(meancash, meancard, customerInfoIndiv)* calculates the servicetime for a customer. Within the *customerInfoIndiv*, the payment method is given, which is previously calculated. The servicetime is exponentially distributed and calculated for different means per payment method. As expected, the mean of cash payments is higher than the mean of card payments. The servicetime is then added to the *customerInfoIndiv*. If, for some reason, the payment method is not defined right in the *customerInfoIndiv*, the errormessage "card/cash not defined well" will appear.

Then, the *finish(customerInfoIndiv, timePayment)* is defined. This function calculates the timestamp a customer's payment is finished. The waittime and servicetime are present in the *customerInfoIndiv*.

Algorithm 2 finish

timeFinished = queueingtime + waittime + servicetime
append timeFinished to customerInfoIndiv

After a customer finishes the payment, the customer should be removed from the queue. This is done by *removeFromQueue(currentTime, timePaymentQueue, alreadyFinishedJobs, queueInfoQueue)*. The *currentTime* is given, to set a timestamp at which the queue should be updated. The *alreadyFinishedJobs* keeps track of the amount of finished customers, that are already removed from the queue.

Algorithm 3 removeFromQueue

finishedJobs = [timestamps in timePaymentQueue which are less or equal to the currentTime]
amountFinishedJobs = len(finishedJobs)
newlyFinishedJobs = amountFinishedJobs - alreadyFinishedJobs
alreadyFinishedJobs = amountFinishedJobs
queueInfoQueue = queueInfoQueue[newlyFinishedJobs:]

2.3 Simulation

In the simulation class the first step is to determine the arrivals, time to take food, and cash or card per customer by using functions from the customer class. This will result in a list of customers, and per customer there is a list that looks as follow: (customernumber, [GroupNr, arrivalTime, timeToTakeFood, timeToQueue, 'card' or 'cash']).
After the customers are initialized the current time, the number of queues, a list with lists per queue, and an array of zeros for the queues to keep track of the number of finished customers is defined. Third, the list of all the customers is sorted according to the time they arrive at the queue.
Then for all customers in the sorted list of customers it is checked if the time to the queue is higher than the current time and if so, the current time is replaced by the time the customer arrives at the queue. The queue will then be updated to check if customers have already left the queue. After that step, information about the customer will be added step by step by using functions from the service class. First, the customer will be assigned to a queue, and the queue number will be added to the customer information. Second, the waiting times are computed and added, and third the service time is computed and added. As last the finishing time will be added to the customer information. The customerInfoIndiv will after adding the finishing time looks like this: (customernumber, [GroupNr, arrivalTime, timeToTakeFood, timeToQueue, 'card' or 'cash', queue Nr, waitingTime, serviceTime, FinishedTime]). After adding information to the customer info list, this list is added to a list containing the information of all individual customers.

2.4 Results

There is chosen to make a separate class for computing the results of the simulation to make a more clear structure and overview. In this class the input variables are defined that are used in the simulation. The meangroupsize = 3 customers per group, the mean time to take food is 80 seconds, there are in total 3 servers and thus 3 queues, moreover the total time of the simulation is 3600 seconds (1 hour), and the mean time to pay with card is 12 seconds and with cash 20 seconds. There are other input values that change for the different questions. First, an input value "extension" is added this indicates if the basic model (0) is used or if extension 1,2, or 3 is used. Second, the results need to be computed for λ=1,2,3,4 so the Poisson arrival rate needs to be able to change easily. Third the percentage of cash payments changes for extension 3 to 0 so also needs to be able to change easily.

Besides, per customer there is a list with information, to make it more clear which index represents what information, the index numbers are first defined. These are as follows: groupNr = 0, arrTime = 1, timeFood = 2, timeQueue = 3, cashCard = 4, nrQueue = 5, waitTime = 6, serviceTime = 7, finishTime = 8.

3 Results and solutions

For the results of the simulation, different numbers of runs were used. A half-width of 3 seconds was chosen. For $\lambda = 1, 2, 3$, 500 runs were used and for $\lambda = 4$ there were 5000 runs. To estimate the number of simulation runs, the results of a shorter simulation were used. Then the needed number of runs was estimated using the formulas from the lecture. Since the numbers of runs for $\lambda = 1, 2, 3$ were in a similar range we chose a maximum of this number and added a safety margin. For $\lambda = 4$ the number of required runs was higher therefore a higher number of runs was chosen, again with a safety margin. The results of the estimation are in the appendix A.

3.1 Results

In Table 1 the results are given for lambda = 1,2,3,4 for the basic model. W_0 is the waiting time of queue 0, W_1 is the waiting time of queue 1, W_2 is the waiting time of queue 2, S is the sojourn time for an individual customer, and S_g is the sojourn time for a customer group. In Table 2 the confidence intervals for the waiting times of the queues are shown, in which LB is the lower bound and UB is the upper bound of the confidence interval. The confidence intervals for the sojourn times are shown in Table 3 in subsection 3.5, again LB is the lower bound and UB is the upper bound of the confidence interval.

Table 1: Results

λ	$\mathbf{E}[W_0]$	$\mathbf{sd}[W_0]$	$\mathbf{E}[W_1]$	$\mathbf{sd}[W_1]$	$\mathbf{E}[W_2]$	$\mathbf{sd}[W_2]$	$\mathbf{E}[\mathbf{S}]$	$\mathbf{sd}[\mathbf{S}]$	$\mathbf{E}[S_g]$	$\mathbf{sd}[S_g]$
1	1.06	4.64	1.10	4.82	1.07	4.62	96.28	81.15	148.46	100.55
2	5.45	13.50	5.35	13.21	5.42	13.38	100.82	82.67	153.01	102.66
3	20.67	29.85	20.51	29.55	20.67	29.54	115.48	87.44	168.17	106.56
4	127.75	90.71	128.07	90.90	127.77	90.78	222.67	127.39	282.83	142.71

Table 2: Confidence intervals for waiting times

λ	$\mathbf{E}[W_0]$	**LB**	**UB**	**Half-width**	$\mathbf{E}[W_1]$	**LB**	**UB**	**Half-width**	$\mathbf{E}[W_2]$	**LB**	**UB**	**Half-width**
1	1.06	0.97	1.15	0.09	1.10	1.00	1.19	0.10	1.07	0.98	1.16	0.09
2	5.45	5.18	5.73	0.28	5.35	5.07	5.64	0.29	5.42	5.15	5.68	0.27
3	20.67	19.60	21.74	1.07	20.51	19.48	21.54	1.03	20.67	19.63	21.70	1.04
4	127.75	125.43	130.07	2.32	128.07	125.74	130.41	2.34	127.77	125.44	130.11	2.33

3.2 Question 1

The sojourn time for an arbitrary customer can be found in Table 1. As can be seen the sojourn time increases when lambda increases, this is reasonable since if more customers are arriving the waiting time will become longer. When lambda is equal to 1, it means there is approximately arriving 1 group of three customers in one minute. In comparison, when lambda is equal to 4 it means approximately one group of three customers arrives every 15 seconds. The mean service time is approximately 15.2 seconds (12*0.4+20*0.6), which means there are approximately more customers arriving than customers leaving. The difference in length of the sojourn time depends mostly on the waiting time of the customers, since the time to take food and the service time does not depend on the number of arrivals.

Second, also the waiting time per queue can be found in Table 1. As can be seen the average waiting times are approximately equal between the queues, this indicates that all queues are evenly used. It is also shown that the waiting time for all queues is also indeed larger when lambda is larger.

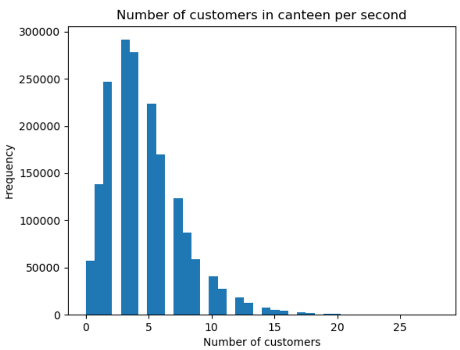
Third, there are on average 4.66 (λ=1), 9.74 (λ=2), 16.89 (λ=3), and 41.57 (λ =4) customers in the system. This is in line with the waiting times, since if there are more customers in the system there will also be more customers in the queue and therefore cause a longer waiting time.

3.3 Question 2

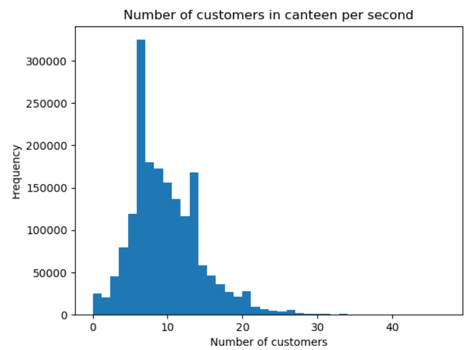
The sojourn time per group can also be found in Table 1, also in this case when lambda increases the sojourn time increases. This is again reasonable since if there are more arrivals the waiting time will increase and therefore the sojourn time will increase.

3.4 Question 3

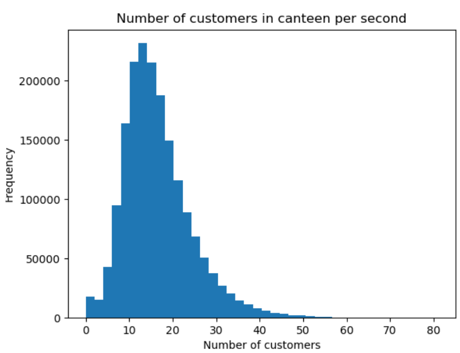
In Figure 1 histograms are shown per value of lambda, in every histogram the x-axis represents the number of customers in the canteen per second. The y-axis represents the frequency, this is the number of times a certain number of customers is in the canteen. The distribution for all values of different lambdas seems to be approximately the same, since it is all skewed to the left. However, for λ=1 the number of customers in the canteen is mostly between 0 and 25, for λ=2 between 0 and 30, for λ=3 between 0 and 40, and for λ=4 between 0 and 150. This indicates the mean is moving to the right of the graph, and thus increasing.



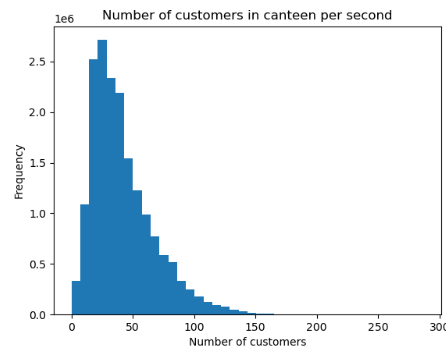
(a) Distribution λ=1



(b) Distribution λ=2



(c) Distribution λ=3



(d) Distribution λ=4

Figure 1: Histogram for different values of λ

3.5 Question 4

In Table 3 the confidence interval for sojourn times for an arbitrary customer and of a group of customers is shown. As can be seen the half-width increases when lambda increases, this means the number of runs does not result in the same accuracy for different values of lambda.

Table 3: Confidence intervals sojourn times

λ	**E[S]**	**LB**	**UB**	**Half-width**	**E**$[S_g]$	**LB**	**UB**	**Half-width**
1	96.28	95.73	96.83	0.55	148.46	147.29	149.63	1.17
2	100.82	100.34	101.28	0.47	153.01	152.13	153.89	0.88
3	115.49	114.41	116.57	1.08	168.17	166.85	169.49	1.32
4	222.67	220.35	224.98	2.32	282.83	280.28	285.38	2.55

4 Model extensions

Different model extensions are presented, which are applied one by one to the model described previously.

4.1 Model extension 1

In the first extension the service time at one of the servers takes longer since the cashier is new, there is determined this cashier will be working at server 0 (and thus queue 0). To add this extension there is added an if-statement to the serviceTime-function in the service class. The if-statement makes sure that when extension 1 is used and the customer is at queue 0 the service time is 25% longer. In Table 4 the results are shown for this extension. As can be seen, compared to the results without the extension, the mean waiting times for all queues become longer. For queue 0 this is most logical since the service time for this queue increases. For queues 1 and 2 it is less obvious, however since a customer will be longer at queue 0 it might also be that more customers will be assigned to queues 1 and 2 which will increase the waiting times in these queues too. The differences between the values for lambda are the same as the results with no extension, so when lambda increases, the waiting time and the sojourn time increase.

Table 4: Results with extension 1

λ	$\mathbf{E}[W_0]$	$\mathbf{sd}[W_0]$	$\mathbf{E}[W_1]$	$\mathbf{sd}[W_1]$	$\mathbf{E}[W_2]$	$\mathbf{sd}[W_2]$	$\mathbf{E}[\mathbf{S}]$	$\mathbf{sd}[\mathbf{S}]$	$\mathbf{E}[S_g]$	$\mathbf{sd}[S_g]$
1	1.68	6.91	1.27	5.35	1.33	5.63	98.04	81.81	150.78	101.11
2	8.48	18.97	6.49	14.91	6.42	14.60	103.40	83.02	155.43	102.29
3	36.11	44.60	27.78	34.94	27.97	35.36	126.12	91.42	178.58	109.73
4	248.00	159.52	197.50	127.26	197.69	127.15	307.99	168.68	376.96	184.10

4.2 Model extension 2

In this model extension, 15% of the groups of customers are going to get food from the cart in front of the canteen. In the customer file, the function $groupReduceFifteenPercent$ is added. This function checks the dictionary to get the number of groups in the canteen and then removes 15% of the groups. Afterwards, it rearranges the dictionary to have no break in the customer numbers. All the results from the simulation are given in Table 5

Table 5: Results with extension 2

λ	$\mathbf{E}[W]$	$CI_{95\%,\ \mathbf{E}[W]}$	$\mathbf{sd}[W]$	$\mathbf{E}[\mathbf{S}]$	$CI_{95\%,\ \mathbf{E}[\mathbf{S}]}$	$\mathbf{sd}[\mathbf{S}]$	$\mathbf{E}[S_g]$	$CI_{95\%,\ \mathbf{E}[S_g]}$	$\mathbf{sd}[S_g]$
1	0.89	[0.80, 0.98]	3.98	96.01	[95.45, 96.57]	80.75	148.69	[147.47, 149.11]	99.55
2	3.81	[3.62, 3.99]	10.87	99.01	[98.56, 99.46]	81.66	150.75	[149.84, 151.65]	101.00
3	12.41	[11.77, 13.03]	21.82	107.55	[106.84, 108.25]	84.61	159.72	[158.70, 160.74]	104.06
4	45.38	[44.74, 46.00]	47.84	140.39	[139.76, 141.02]	96.85	194.49	[193.79, 195.19]	114.86

In the Table 5, it can be seen that the sojourn time increases with a higher lambda, but the increase is slower than in the Table 1. The only time that depends on how many customers are in the canteen is the waiting time for the cashier. It can be also seen that the waiting time is increasing with more customers but compared to the canteen with no food cart in front the waiting time is shorter. The expected sojourn time is shorter than without the food cart since there are fewer customers in the canteen and the waiting time at the cashier is shorter.

4.3 Model extension 3

In this extension, the canteen is not accepting cash payments anymore. This means that the input which states the fraction of cash payments is set to 0. The expected sojourn time and expected waiting time are then affected.

The mean waiting time is a lot lower. For $\lambda = 1$, it is approximately halved and for $\lambda = 4$ the waiting time is even more than 6 times as small. The mean sojourn time also decreases. The sojourn time exists out of the

Table 6: Results with extension 3

λ	$\mathbf{E}[W_0]$	$\mathbf{sd}[W_0]$	$\mathbf{E}[W_1]$	$\mathbf{sd}[W_1]$	$\mathbf{E}[W_2]$	$\mathbf{sd}[W_2]$	$\mathbf{E}[\mathbf{S}]$	$\mathbf{sd}[\mathbf{S}]$	$\mathbf{E}[S_g]$	$\mathbf{sd}[S_g]$
1	0.45	2.42	0.44	2.37	0.45	2.36	92.64	80.57	145.06	100.02
2	2.08	6.69	2.14	6.90	2.07	6.71	94.33	81.08	147.13	101.48
3	6.33	12.98	6.27	12.74	6.29	12.97	98.11	81.65	150.04	101.90
4	19.69	26.09	19.64	25.93	19.68	26.03	111.63	85.49	163.64	104.94

time to take food, the servicetime and the waiting time. Since the mean servicetime and the mean waiting time decrease and the time to take food stays the same, this is completely as expected. With the decrease of the mean, the standard deviation also decreases.

5 Conclusions

Overall can be seen that the waiting times when lambda is 4 are a lot higher than the waiting times for values 1, 2, and 3 for lambda. This indicates the rate at which customers enter the queue is higher than leaving the queue. Besides when approximately four groups of three customers arrive at the system per minute, the waiting time will be around 125 seconds. The total time is around 220 seconds per customer, this means the waiting time is longer than the time a customer takes food and pays for the food. For a value for lambda of 1, 2, and 3 this is not the case, the waiting time is shorter than the time a customer takes food and pays for the food. When the waiting is longer than the time a customer is actually doing something, it might cause customers will go somewhere else, like in extension 2.

As can be seen in the results of extension 1, a new employee with a longer service time will increase the waiting times for all queues and not only for the specific queue. However, in extension 3 there is shown that the waiting times are decreasing significantly high for only card payments. It is a useful idea to combine these two. So when a new employee starts he or she will begin with only payments with a card and not cash. This gives the new employee time to practice and later start to also work with cash payments.

To reduce the waiting time, it would be beneficial to get rid of cash payments. However, at the moment it is assumed that there are as many arrivals as before. In practice, this could not be the case and the number of arrivals could be reduced, since some potential customers want to pay by cash. Although, when the queues are too long due to including cash payments, customers would also not come by, resulting in fewer arrivals. The best of the two (including or excluding cash) should be picked, but this is dependent on the arrival rate. After an analysis of the arrival rate and of the only card payments and long queues arrival rates, the best option between only card or card and cash payments can be made.

A Number runs

Table 7: Number of runs for $\lambda = 1$

lambda = 1, 1000 runs used, half-width = 3		
	Std	NrRuns
Q1sojourn	6,26	16,727009
Q1waiting0	1,11	0,525915
Q1waiting1	1,05	0,470596
Q1waiting2	1,11	0,525915
Q1nrCustomer	0,6	0,153664
Q2Sojourn	13,45	**77,217227**

Table 8: Number of runs for $\lambda = 2$

lambda = 2, 1000 runs used, half-width = 3		
	Std	NrRuns
Q1sojourn	5,45	12,67835
Q1waiting0	3,41	4,96339
Q1waiting1	3,2	4,370887
Q1waiting2	3,33	4,733235
Q1nrCustomer	0,99	0,41835
Q2Sojourn	10,27	**45,02052**

Table 9: Number of runs for $\lambda = 3$

lambda = 3, 1000 runs used, half-width = 3		
	Std	NrRuns
Q1sojourn	12,22	63,739998
Q1waiting0	11,84	59,837445
Q1waiting1	11,42	55,667516
Q1waiting2	11,84	59,837445
Q1nrCustomer	2,58	2,8412474
Q2Sojourn	15,08	**97,067158**

Table 10: Number of runs for $\lambda = 4$

lambda = 4, 1000 runs used, half-width = 3		
	Std	NrRuns
Q1sojourn	79,58	2703,196
Q1waiting0	79,93	2727,026
Q1waiting1	79,3	2684,207
Q1waiting2	80,68	2778,442
Q1nrCustomer	15,44	101,757
Q2Sojourn	87,15	**3241,936**

B Source code

B.1 Customer

```python
import numpy as np

class customer:

    def arrivaltime(poissonrate, totalTime):
        '''
        Calculates all the times when groups arrive.
        Param poissonrate: at which meantime rate the groups arrive.
        Param totalTime: Time of the simulation given in seconds.
        Return: a list of all the arrival times.
        '''
        TimesGroupArrives = []  #list to save all the different times when groups arrive
        t = np.random.poisson(60/poissonrate)
        while(t < totalTime):   # checks that arrival time is within the hour
            TimesGroupArrives.append(t)
            t += np.random.poisson(60/poissonrate)  # adds new arrival time
        return TimesGroupArrives


    def arrive(poissonrate, totalTime, meangroupsize):
        '''
        Puts groupsize and arrival time in a dictionary.
        Param poissonrate: at which the groups arrive.
        Param totalTime: Time of the simulation given in seconds.
        Param meangroupsize: integer of the mean group size.
        Returns: a dictionary of all the customers that are coming to the canteen.
        '''
        arrivalTimeArray = customer.arrivaltime(poissonrate, totalTime)
        Customers = {}  # creates dictonary to save all the information about the customer.
        customerAlreadyInTheCanteen = 0  # keeps track amount of customers in the canteen
        groupNr = 1  # keeps track of the number of groups that already arrived
        for i in range(len(arrivalTimeArray)): # goes over all the arrival times for the groups
            groupsize = np.random.geometric(1/meangroupsize)
            for j in range(groupsize):
                Customers[(j + customerAlreadyInTheCanteen)] = [groupNr, arrivalTimeArray[i]]
            customerAlreadyInTheCanteen += groupsize
            groupNr += 1
        return Customers


    def takeFood(mean, Customers):
        '''
        Calculates how long it takes a person to get the food.
        Param mean:  mean of the exponential distribution.
        Returns: a dictonary of all the customers that arrived during the time in the Canteen
        '''
        customersInTheCanteen = len(Customers) # calculates the number of all the customers that visit the canteen during lunch break
        for i in range(customersInTheCanteen):
            timeToGetFood = np.random.exponential(mean)  # chooses a random number for the time the customer needs to get the food
            timeToQueue = Customers[i][1]+timeToGetFood  # calculates the time it takes the customer to get to the queue
            Customers[i].append(timeToGetFood)
            Customers[i].append(timeToQueue)
        return Customers


    def cardcash(percentagecash, Customers):
        '''
        Decides if the customer pays with cash or a card.
```

```
        param percentagecash: int between 0 and 1
        returns: the dictionary customers with the added value cash or card
        '''
        customersInTheCanteen = len(Customers)
        for i in range(customersInTheCanteen):  # goes through all the customers in the canteen during lunch hour
            randomnumber = np.random.uniform(low=0.0, high=1.0) # picks a random number between 0 and 1 to decide if the customer pays with cash or card
            if randomnumber <= percentagecash:
                Customers[i].append("cash")
            else:
                Customers[i].append("card")
        return Customers


    def groupReduceFifteenPercent(customers):
        '''
        Reduces the groups in the dictionary by fifteen percent.
        Param customers: dictionary with all the customers that arrive.
        Returns: the reduced dictionary.
        '''
        amountOfGroups = list(customers.values())[-1][0] # number of groups during lunch break
        fifteenPercent = round(amountOfGroups*0.15)  # calculates 15% of the group
        actualPercentage = fifteenPercent/amountOfGroups # calculates the actual percentage of the groups that are going to the food cart
        customers = list(map(list, customers.items())) #changes into a list
        for i in range(fifteenPercent):
            deletedGroup = np.random.randint(amountOfGroups) # picks a random number for the groups that are not going to the canteen
            for j in range(len(customers)-1,0,-1):
                if customers[j][1][0] == deletedGroup:
                    del customers[j]  # deleting all the customers from the group
        reducedCustomersDictonary = dict(customers)
        newlySortedDict = {}  # rearranges dictionary to have a continuous customer number in the key
        for value in range(len(customers)):
            item = list(reducedCustomersDictonary.values())[value]
            key, value = value, item  # assigning new number to the customers
            newlySortedDict[key] = value
        return newlySortedDict, actualPercentage

# Order for customer in the canteen
# 0: arrival
# 1: take food
# 2: start to queue
# 3: card/cash
```

B.2 Service

```
from customer import customer
from numpy import argmin, random, zeros

class service:
    '''
    This class determines the service, which includes the queueing process
    '''
    groupNr = 0
    arrTime = 1
    timeFood = 2
    timeQueue = 3
    cashCard = 4
    nrQueue = 5
    waitTime = 6
    serviceTime = 7
    finishTime = 8

    def assignToQueue(queueInfo, customerInfoIndiv):
        '''
```

```
    Assigns a customer to shortest queue
    param queueInfo: list with a list per queue within which gives the customernumbers that are in the queue
    param customerInfoIndiv: info of the customer which have the customer number and list of group number, arrival time, taking food time, arriving time at queue and payment method
    returns: the customerinfo with the shortest queue added and the queueinfo with the customer added to the shortest queue
    '''
    MinQueues = [i for i in range(len(queueInfo)) if len(queueInfo[i]) == min(len(queueInfo[j]) for j in range(len(queueInfo)))]
    queue = random.choice(MinQueues) # if multiple queues are the shortest, pick random one
    customerInfoIndiv[1].append(queue)
    queueInfo[queue].append(customerInfoIndiv[0])
    return customerInfoIndiv, queueInfo

def waitQueue(queueInfo, customerInfoIndiv, timePayment):
    '''
    Calculate the waiting time for a customer
    param queueInfo: list with a list per queue within which gives the customernumbers that are in the queue
    param customerInfoIndiv: info of the customer which have the customer number and list of group number, arrival time, taking food time, arriving time at queue, payment method and queue number
    param timePayment:
    returns:customerInfoIndiv with the waitingtime added
    '''
    queue = customerInfoIndiv[1][service.nrQueue]
    if len(queueInfo[queue]) == 1: # only one customer in the queue, so no waiting time
        waittime = 0
    else:
        waittime = timePayment[queue][-1] - customerInfoIndiv[1][service.timeQueue]
    customerInfoIndiv[1].append(waittime)
    return customerInfoIndiv

def servicetime(extension, meancash, meancard,customerInfoIndiv):
    '''
    Calculate the service time for a customer
    param meancash: mean time for cash payments
    param meancard: mean time for card payments
    param customerInfoIndiv: info of the customer which have the customer number and list of group number, arrival time, taking food time, arriving time at queue, payment method, queue number and waitingtime
    returns: the customerinfo with the servicetime added
    '''

    ######## added code for extension 1!
    # For extension 1 an if statement is added, this makes sure when extension 1 is used and the customer is in
    # queue 0 the service time is 25% longer assuming the new employee works at queue 0
    if extension == 1 and customerInfoIndiv[1][service.nrQueue]==0:
        if customerInfoIndiv[1][service.cashCard] == "cash":
            servicetime = random.exponential(meancash*1.25) #servicetime cash
        elif customerInfoIndiv[1][service.cashCard] == "card":
            servicetime = random.exponential(meancard*1.25) #servicetime card
        else:
            print("card/cash not defined well")

        customerInfoIndiv[1].append(servicetime)
        return customerInfoIndiv
    # Else extension 1 is not used or extension 1 is used and the customer is not in the queue with the new employee
    else:
        if customerInfoIndiv[1][service.cashCard] == "cash":
            servicetime = random.exponential(meancash) #servicetime cash
        elif customerInfoIndiv[1][service.cashCard] == "card":
            servicetime = random.exponential(meancard) #servicetime card
```

```
            else:
                print("card/cash not defined well")
            customerInfoIndiv[1].append(servicetime)
            return customerInfoIndiv

    def finish(customerInfoIndiv, timePayment):
        '''
        Calculate the time a customer is finished
        param customerInfoIndiv: info of the customer which have the customer number and list of group number, arrival time, taking food time, arriving time at queue, payment method, queue number, waitingtime and servicetime
        param timePayment: list with a list per queue within which gives the time each customer of that queue is finished
        returns: the customerinfo with the finishing time added
        '''
        queue = customerInfoIndiv[1][service.nrQueue]
        serviceTime = customerInfoIndiv[1][service.serviceTime]
        waitTime = customerInfoIndiv[1][service.waitTime]
        timeFinished = customerInfoIndiv[1][service.timeQueue] + waitTime + serviceTime
        timePayment[queue].append(timeFinished)
        customerInfoIndiv[1].append(timeFinished)
        return customerInfoIndiv, timePayment

    def removeFromQueue(currentTime, timePaymentQueue, alreadyFinishedJobs, queueInfoQueue):
        '''
        Remove the finished customers from the queue
        param currentTime: time for which queue should be updated
        param timePaymentQueue: list per queue which gives the time each customer of that queue is finished
        param alreadyFinishedJobs: amount of finished customers queue already had
        param queueInfoQueue: list per queue which gives the time each customer of that queue is finished
        returns: the amountFinishedJobs and the updated queueInfoQueue
        '''
        finishedJobs = [time for time in timePaymentQueue if time <= currentTime]
        amountFinishedJobs = len(finishedJobs)
        newlyFinishedJobs = int(amountFinishedJobs - alreadyFinishedJobs)
        alreadyFinishedJobs = amountFinishedJobs
        queueInfoQueue = queueInfoQueue[newlyFinishedJobs:] # remove the newly finished customers from the list
        return amountFinishedJobs,queueInfoQueue
```

B.3 Simulation

```
from service import service
from customer import customer
from numpy import zeros, mean, random, std, sqrt, var, linspace, array
#from numpy.ndarray import flatten
import time
import matplotlib.pyplot as plt

class simulation:
    # simulate use cases
    # for 1 hour: 12.00h-13.00h


    def sim(extension, poissonratearrivals, totalTime, meangroupsize, meanFood, cashpayments, meancash, meancard):
        # First the arrival times are determined
        customers = customer.arrive(poissonratearrivals, totalTime, meangroupsize)
        # Output: for 1 customer (customernumber, [GroupNr, arrivalTime, timeToTakeFood, timeToQueue])

        # For extension 2 less groups arrive and thus a adjusted set of arrivals is used
        actuallPercentage = 0
        if extension == 2:
            customers, actuallPercentage = customer.groupReduceFifteenPercent(customers)
        # Adds to a customer the time it takes to take food
        customersGottenFood = customer.takeFood(meanFood, customers)
```

```
    # Output: for 1 customer (customernumber, [GroupNr, arrivalTime, timeToTakeFood, timeToQueue])

    # Adds to a customer if the person pays with cash or card
    customersCashCard = customer.cardcash(cashpayments, customersGottenFood)
    # Output: for 1 customer (customernumber, [GroupNr, arrivalTime, timeToTakeFood, timeToQueue, 'card' or 'cash'])


    # Initialise
    timeCurrent = 0 #start at t=0
    amountOfQueues = 3
    # Create an empty list per queue to be able to assign customers to a queue
    queues = [[] for i in range(amountOfQueues)]
    # Create an empty list per queue to be able to store the finishing times
    timeFinished = [[] for i in range(amountOfQueues)]
    # Create an array of zeros to keep track of the number of finished customers per queue
    finishedCustomers = zeros(amountOfQueues)

    # Sort the customers based on the time they arrive at the queue
    customerInfoSortqueue = sorted(customersCashCard.items(),key=lambda item: (item[1][3]))

    # start with an empty list to store all customer info
    listAll = []
    for i in range(len(customerInfoSortqueue)):
        # Set new time to time a customer arrives at the queues
        timeToQueue = customerInfoSortqueue[i][1][3]
        if timeCurrent <= timeToQueue:
            timeCurrent = timeToQueue

        # Update the queues: check for each queue if customers left the queue before the current time
        for j in range(len(queues)):
            finishedCustomers[j], queues[j] = service.removeFromQueue(timeCurrent, timeFinished[j], finishedCustomers[j], queues[j])

        # Assign the customer to the shortest queue
        customerInfoIndiv, queues = service.assignToQueue(queues, customerInfoSortqueue[i])
        # customerInfoIndiv output per customer: (customernumber, [GroupNr, arrivalTime, timeToTakeFood, timeToQueue, 'card' or 'cash', queue Nr])

        # Calculate the waiting time for the customer
        customerInfoIndiv = service.waitQueue(queues,customerInfoIndiv,timeFinished)
        # customerInfoIndiv output per customer: (customernumber, [GroupNr, arrivalTime, timeToTakeFood, timeToQueue, 'card' or 'cash', queue Nr, waitingTime])

        # Calculate the service time for the customer
        customerInfoIndiv = service.servicetime(extension, meancash, meancard, customerInfoIndiv)
        # customerInfoIndiv output per customer: (customernumber, [GroupNr, arrivalTime, timeToTakeFood, timeToQueue, 'card' or 'cash', queue Nr, waitingTime, serviceTime])

        # Calculate the time a customer is finished
        customerInfoIndiv, TimeFinished = service.finish(customerInfoIndiv, timeFinished)
        # customerInfoIndiv output per customer: (customernumber, [GroupNr, arrivalTime, timeToTakeFood, timeToQueue, 'card' or 'cash', queue Nr, waitingTime, serviceTime, FinishedTime])

        # Append all the information of a individual customer to a list
        listAll.append(customerInfoIndiv)

    # Define when the queues are empty
    timeEndEmptyQueue = [max(timeFinished[i]) for i in range(amountOfQueues)]
    timeEndEmptyQueues = max(timeEndEmptyQueue)
```

```
        return timeEndEmptyQueues, listAll, actuallPercentage




# sim = simulation.sim(simulation.extension, simulation.poissonratearrivals, simulation.
    totalTime, simulation.meangroupsize, simulation.meanfood, simulation.cashpayments,
    simulation.meancash, simulation.meancard)
# # sim = simulation.sim(simulation.poissonratearrivals, simulation.totalTime, simulation.
    meangroupsize, simulation.meanfood, 0, simulation.meancash, simulation.meancard)
# all_results = simulation.results(sim)
# # question3 = simulation.question3(50)
```

B.4 Results

```
from customer import customer
from service import service
from simulation import simulation
from numpy import zeros, mean, random, std, sqrt, var, linspace, array
#from numpy.ndarray import flatten
import time
import matplotlib.pyplot as plt
import numba as nb


class results:

    # input values
    extension = 0
    poissonratearrivals = 1
    meangroupsize = 3
    meanFood = 80 #seconds
    totalNrQueues = 3
    # listQueuedCustomersOld = zeros(totalNrQueues) # not used anymore
    totalTime = 3600 # in seconds
    cashpayments = 0.4    # 0.4 for basic, 0 for extension 3
    meancard = 12
    meancash = 20

    #Index
    groupNr = 0
    arrTime = 1
    timeFood = 2
    timeQueue = 3
    cashCard = 4
    nrQueue = 5
    waitTime = 6
    serviceTime = 7
    finishTime = 8

    #@nb.jit()
    def results(nrRuns):
        startTime = time.time()

        # Lists for all questions to store the mean and standard deviation per run
        Q1MeanSojourn = []
        Q1StdSojourn = []
        Q1MeanQueue0 = []
        Q1StdQueue0 = []
        Q1MeanQueue1 = []
        Q1StdQueue1 = []
        Q1MeanQueue2 = []
        Q1StdQueue2 = []
        Q1MeanNrCustomer = []
        Q1StdNrCustomer = []

        Q2MeanSojournGroup = []
        Q2StdSojournGroup = []
```

```python

    Q3listAll = []

    percentageExtension3 = []

    for i in range(nrRuns):
        # print("nrRun", i)
        # Run the simulation every time
        sim = simulation.sim(results.extension, results.poissonratearrivals, results.totalTime, results.meangroupsize, results.meanFood, results.cashpayments, results.meancash, results.meancard)

        percentageExtension3.append(sim[2])

        # Sojourn time arbitrary customer (individual)
        sojournTimeIndividual = []                  # an empty list to store all times
        for i in range(len(sim[1])):
            # compute for every customer the sojourn time
            sojournTimeIndividual.append(sim[1][i][1][results.finishTime] - sim[1][i][1][results.arrTime])
        meanQ1Sojourn = mean(sojournTimeIndividual) # compute mean sojourn time
        stdvQ1Sojourn = std(sojournTimeIndividual)  # compute std sojourn time
        Q1MeanSojourn.append(meanQ1Sojourn)         # add mean time to list of means
        Q1StdSojourn.append(stdvQ1Sojourn)          # add stdv to lsit of standard deviations


        # Expected time spend waiting in queue
        queueTimeList0 = []                         # an empty list to store all times
        queueTimeList1 = []                         # an empty list to store all times
        queueTimeList2 = []                         # an empty list to store all times
        for i in range(len(sim[1])):
            # Per queue add the waiting time to the list
            if sim[1][i][1][results.nrQueue] == 0:
                queueTimeList0.append(sim[1][i][1][results.waitTime])
            if sim[1][i][1][results.nrQueue] == 1:
                queueTimeList1.append(sim[1][i][1][results.waitTime])
            if sim[1][i][1][results.nrQueue] == 2:
                queueTimeList2.append(sim[1][i][1][results.waitTime])
        meanQ1Waiting0 = mean(queueTimeList0)       # compute mean waiting time queue 0
        stdQ1Waiting0 = std(queueTimeList0)         # compute stdv waiting time queue 0
        # print(meanQ1Waiting0)
        Q1MeanQueue0.append(meanQ1Waiting0)         # append the mean to the list of means of queue 0
        Q1StdQueue0.append(stdQ1Waiting0)           # append stdv to the list of stdv of queue 0

        meanQ1Waiting1 = mean(queueTimeList1)       # compute mean waiting time queue 1
        stdQ1Waiting1 = std(queueTimeList1)         # compute stdv waiting time queue 1
        Q1MeanQueue1.append(meanQ1Waiting1)         # append the mean to the list of means of queue 1
        Q1StdQueue1.append(stdQ1Waiting1)           # append stdv to the list of stdv of queue 1

        meanQ1Waiting2 = mean(queueTimeList2)       # compute mean waiting time queue 2
        stdQ1Waiting2 = std(queueTimeList2)         # compute stdv waiting time queue 2
        Q1MeanQueue2.append(meanQ1Waiting2)         # append the mean to the list of means of queue 2
        Q1StdQueue2.append(stdQ1Waiting2)           # append the stdv the list of stdv of queue 2

        # Expected number customers in the canteen
        # Create list of zeros for all seconds per run
        CustomersInCanteenSeconds = [0 for i in range(int(results.totalTime))]
        for i in range(len(sim[1])):
            j = 0
            for j in range(int(sim[1][i][1][results.arrTime]), int(sim[1][i][1][results.finishTime])): #change if stepsize is smaller than 1 seconde
```

```
                if 0<j <results.totalTime:
                    CustomersInCanteenSeconds[j] += 1

        AverageCustomersInCanteen = mean(CustomersInCanteenSeconds)          # compute mean number customers
        StandardDeviationCustomersInCanteen = std(CustomersInCanteenSeconds) # compute stdv number customers
        Q1MeanNrCustomer.append(AverageCustomersInCanteen)                   # append mean to list of means nr customers
        Q1StdNrCustomer.append(StandardDeviationCustomersInCanteen)          # append stdv to list of stdv nr customers


        #question 2
        allGroups = []                                  # create an empty list to store all groups
        # look for each group number and append it to the list if it is not already in it
        for i in range(len(sim[1])):
            Group = sim[1][i][1][results.groupNr]
            if Group not in allGroups:
                allGroups.append(Group)

        dictGroups = {} # create a dictionary with key=groupsnr and value=sojourntime of a group
        for i in allGroups:
            ALLTIME = []
            for g in range(len(sim[1])):
                G = sim[1][g][1][results.groupNr]
                if G == i:
                    arrivalTime = sim[1][g][1][results.arrTime]         # the arrival time is the same for all customers within a group
                    ALLTIME.append(sim[1][g][1][results.finishTime])    # append finish times for all customers within that group
                    maxTime = max(ALLTIME)                              # determine the latest finishing time
                dictGroups[i] = (arrivalTime, maxTime)                  # add to the dictionary the arrival time and latest finishing time
        sojournGroup = []  # create an empty list to store the sojourn times per group
        dictGroups = list(map(list, dictGroups.items()))
        for i in range(len(allGroups)):
            # for all groups compute the sojourn time and append to the list of sojourn times
            sojournGroup.append(dictGroups[i][1][1] - dictGroups[i][1][0])

        Q2MeanSojournGroup.append(mean(sojournGroup))       # compute mean sojourn time and append to list of mean sojourn times per run
        Q2StdSojournGroup.append(std(sojournGroup))         # compute stdv sojourn time and append to list of stdv sojourn times per run

        # Question 3
        # make one list with number of customers in the canteen of all runs
        customersInCanteenSeconds = [0 for i in range(int(results.totalTime))]
        for i in range(len(sim[1])):
            for j in range(int(sim[1][i][1][results.arrTime]), int(sim[1][i][1][results.finishTime])): #change if stepsize is smaller than 1 seconde
                if 0<j <results.totalTime:
                    customersInCanteenSeconds[j] += 1
        Q3listAll.append(customersInCanteenSeconds)



    print("extension", results.extension, "poisson rate", results.poissonratearrivals)

    print("Question 1 ")
    print("Sojourn time")
    print("Mean of mean sojourn time", mean(Q1MeanSojourn))
    # print("Stdv of mean sojourn time", std(Q1MeanSojourn))
```

```
        lb1 = mean(Q1MeanSojourn) - 1.96*sqrt(var(Q1MeanSojourn)/nrRuns)
        ub1 = mean(Q1MeanSojourn) + 1.96*sqrt(var(Q1MeanSojourn)/nrRuns)
        print("Half-width mean sojourn time", 1.96*sqrt(var(Q1MeanSojourn)/nrRuns))
        print("Confidence interval mean sojourn time", lb1, ",", ub1)
        print("Std sojourn time", mean(Q1StdSojourn))

        print("   ")
        print("Waiting time queue 0")
        print("Average waiting time queue 0 = ", mean(Q1MeanQueue0))
        print("stdv of mean waiting time", std(Q1MeanQueue0))
        lb1 = mean(Q1MeanQueue0) - 1.96*sqrt(var(Q1MeanQueue0)/nrRuns)
        ub1 = mean(Q1MeanQueue0) + 1.96*sqrt(var(Q1MeanQueue0)/nrRuns)
        print("Half-width average waiting time queue 0", 1.96*sqrt(var(Q1MeanQueue0)/nrRuns))
        print("Confidence interval average waiting time queue 0", lb1, ",", ub1)
        print("stdv waiting time queue 0 = ", mean(Q1StdQueue0))

        print("   ")
        print("Waiting time queue 1")
        print("Average waiting time queue 1 = ", mean(Q1MeanQueue1))
        print("std of mean waiting time queue 1 = ", std(Q1MeanQueue1))
        lb1 = mean(Q1MeanQueue1) - 1.96*sqrt(var(Q1MeanQueue1)/nrRuns)
        ub1 = mean(Q1MeanQueue1) + 1.96*sqrt(var(Q1MeanQueue1)/nrRuns)
        print("Half-width average waiting time queue 1", 1.96*sqrt(var(Q1MeanQueue1)/nrRuns))
        print("Confidence interval average waiting time queue 1", lb1, ",", ub1)
        print("stdv waiting time queue 1 = ", mean(Q1StdQueue1))

        print("   ")
        print("Waiting time queue 2")
        print("Average waiting time queue 2 = ", mean(Q1MeanQueue2))
        print("std mean waiting time queue 2 = ", std(Q1MeanQueue2))
        lb1 = mean(Q1MeanQueue2) - 1.96*sqrt(var(Q1MeanQueue2)/nrRuns)
        ub1 = mean(Q1MeanQueue2) + 1.96*sqrt(var(Q1MeanQueue2)/nrRuns)
        print("Half-width average waiting time queue 2", 1.96*sqrt(var(Q1MeanQueue2)/nrRuns))
        print("Confidence interval average waiting time queue 2", lb1, ",", ub1)
        print("stdv waiting time queue 2 = ", mean(Q1StdQueue2))

        print("  ")
        print("Number customers")
        print("Mean custumers in the canteen each second:", mean(Q1MeanNrCustomer))
        print("std mean custumers in the canteen each second:", std(Q1MeanNrCustomer))
        lb1 = mean(Q1MeanNrCustomer) - 1.96*sqrt(var(Q1MeanNrCustomer)/nrRuns)
        ub1 = mean(Q1MeanNrCustomer) + 1.96*sqrt(var(Q1MeanNrCustomer)/nrRuns)
        print("Half-width mean custumers in the canteen each second", 1.96*sqrt(var(Q1MeanNrCustomer)/nrRuns))
        print("Confidence interval mean custumers in the canteen each second", lb1, ",", ub1)
        print("Standard deviation customers in canteen:", mean(Q1StdNrCustomer))

        print("  ")
        print("Question 2")
        print("Mean sojourn time per group", mean(Q2MeanSojournGroup))
        print("Std mean sojourn time per group", std(Q2MeanSojournGroup))
        lb1 = mean(Q2MeanSojournGroup) - 1.96*sqrt(var(Q2MeanSojournGroup)/nrRuns)
        ub1 = mean(Q2MeanSojournGroup) + 1.96*sqrt(var(Q2MeanSojournGroup)/nrRuns)
        print("Half-width mean sojourn time per group", 1.96*sqrt(var(Q2MeanSojournGroup)/nrRuns))
        print("Confidence interval mean sojourn time per group", lb1, ",", ub1)
        print("Std sojourn time per group", mean(Q2StdSojournGroup))
        print(" ")


        # Question 3
        Q3listAllarray = array(Q3listAll)
        Q3listAllFlatten = Q3listAllarray.flatten()
        plt.hist(Q3listAllFlatten, bins= 40)
```

```
        plt.xlabel('Number of customers')
        plt.ylabel('Frequency')
        #plt.xlim(0,250) # add to create histograms with the same x-axis
        plt.title('Number of customers in canteen per second')
        plt.show()
        Mean = mean(Q3listAllFlatten)
        StandardDeviation = std(Q3listAllFlatten)
        print("Question 3")
        print("Mean:", Mean)
        print("Standard deviation:", StandardDeviation)


# =============================================================================
#         # Question 4
#         print("Question 4")
#         # Confidence interval individual customer
#         lb1 = mean(Q1MeanSojourn) - 1.96*sqrt(var(Q1MeanSojourn)/nrRuns)
#         ub1 = mean(Q1MeanSojourn) + 1.96*sqrt(var(Q1MeanSojourn)/nrRuns)
#         print("Half-width arbitrary customer", 1.96*sqrt(var(Q1MeanSojourn)/nrRuns))
#         print("Confidence interval arbitrary customer", lb1, ",", ub1)
#         # Confidence interval customer group
#         lb1 = mean(Q2MeanSojournGroup) - 1.96*sqrt(var(Q2MeanSojournGroup)/nrRuns)
#         ub1 = mean(Q2MeanSojournGroup) + 1.96*sqrt(var(Q2MeanSojournGroup)/nrRuns)
#         print("Half-width arbitrary group", 1.96*sqrt(var(Q2MeanSojournGroup)/nrRuns))
#         print("Confidence interval arbitrary group", lb1, ",", ub1) # I think this should be arbitrary group, so I changed the printed sentences customer -> group
# =============================================================================


        #ModelExtension 2:
        if results.extension ==2:
            print("Percentage of groups that are going to the food card: ",mean(percentageExtension3))

        totalTime = time.time() - startTime

        print("Total time", totalTime)
        return "Total time", totalTime




# results.Question2(10)
results.results(5000)
```

C Workload

We have all worked on every class, but per person spend more time on specific classes. Lore worked mostly on the service file, the simulation file and model extension 3 and on the report. Gilianne worked mostly on the simulation file, result file, model extension 1 and the report. Paulina worked mostly on the customer file, the results file, model extension 2 and on the report. Next to this, we all checked each other's code and helped each other when needed. We met in person and we feel like we all spend equally as much time on it.