Exercise4V2

October 8, 2020

1 Question 1

The first thing I did to tackle this question was to simulate the randomisation of the order of the queue and the seating position. I then devised a way to check for aisle and seat interference through the use of the np.floor function and the modulus operator to calculate the row and column of each invidiual seat, assuming that the layout of the airplane is 4 seats-AISLE-4 seats.

My first initial simulation was purely to calculate the average time it took each person to board and find their seat. I then expanded this out to simulate the boarding of 100 passengers across 1000 planes to get the average time and standard deviation for the total boarding.

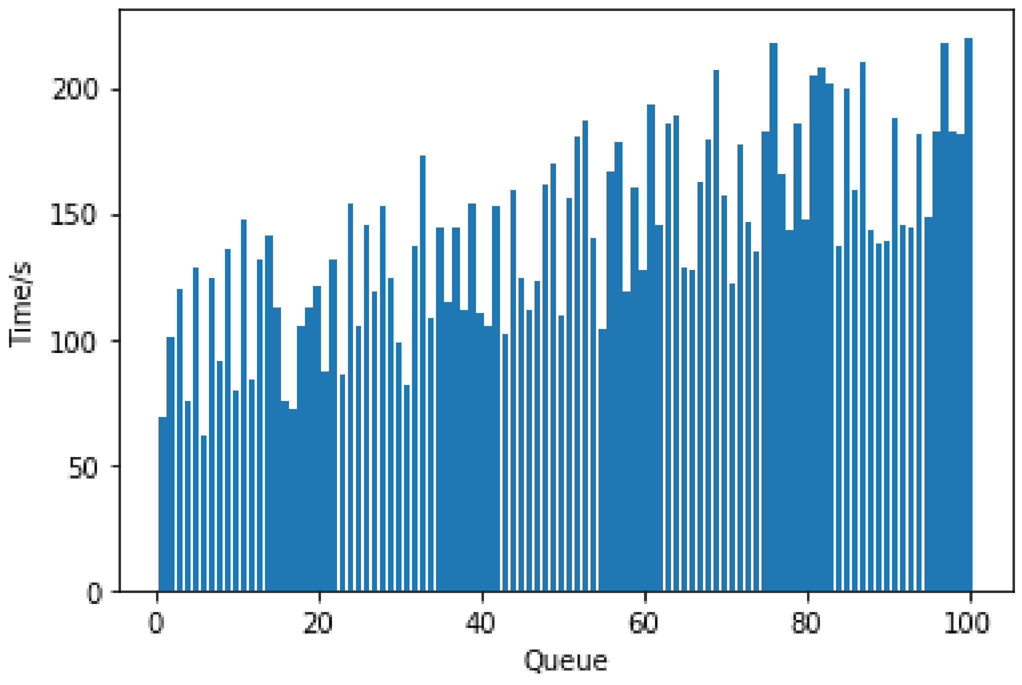
I must admit that I recognise that my model has problems such as the fact that in this model 1 boarding of a passenger is done at one time leading to a ridiculously high total boarding time of +2hrs, however I could not come up with a suitable way of implementing this. I tried to edit my program to alow for parallel boardings with a time gap between however I could not do this and struggled to think of how to implement it.

I struggled with this question a lot because I was unsure what it was asking us to do.I assumed that we simulate over many different planes boarding 100 people. My programming skills aren't the best and I would have found the analytical version of this problem much more to my taste. Most of my time was spent trying to implement parallel boarding into my model. A simulation with time steps as mentioned in the question of the exercise probably would have made more sense, however how to implement this if you loop over until every seat is taken how then can you implement interferences and the distance/velocities?

[18]:
```python
import numpy as np
import scipy as sci
from matplotlib import pyplot as plt
import random
from random import choice
import sys


#init
airplane = []
random.seed(random.randint(1,25000))

seatstaken = 0
N = 100 #number of people
```

```
if N % 4 != 0:
    print('ERROR: N not divisible by 4.')
    sys.exit()
seats = np.zeros(N)
times = np.zeros(N)
velocity = 0.5 #0.5 m/s speed
distance = 25 #25m from gate (first person) to first seat row (seat no 1,2,3,4)
queuedist = 0.5 #0.5m between people in boarding queue
rowdist = 1 #1m between seat rows
#so it takes (25m + (0.5*arrayindex) + (1*rowindexnumber)) * 0.5m/s time to get␣
↪from queue pos to seat row
#var form: (distance + (queuedist * i) + (rowdist * rownumber starting from 0))␣
↪* velocity
# + 30s for aisle interference
# if person[i]_row > person[i-1]_row then +30s
# + 15 secs for seat interference
# if person[i] and person[i-1] == same row and person[i-1] in column 2 or 3␣
↪then + 15s

#constructing airplane
for i in range (0,N):
    airplane.append(i)

queue = []
j=1
while j < N+1: #randomisizing the queue
    p = random.randint(1,N)
    if p not in queue:
        queue.append(p)
        j+=1


r= random.sample(range(0,N),N)
print(r)
for i in range(0,N):
    #r = random.randint(0,N-1)
    #s = random.randrange(0,11)
    #print(r[i])
    #seats[r] = queue[i]
    times[i] = times[i] + ((distance +(queuedist*i)+(rowdist * np.floor(r[i]/
↪4)))/ velocity)
    if seats[r[i]] == 0:
        seats[r[i]] = queue[i]
   # print('Person %d is assigned to seat %d. Column %d, Row %d' % (queue[i],␣
↪r[i]+1,(r[i] % 4)+1, np.floor(r[i]/4)+1))
    if np.floor(r[i]/4) > np.floor(r[i-1]/4):
        times[i]= times[i] + 30
        #print('Aisle Interference')
    if np.floor(r[i]/4) == np.floor(r[i-1]/4) and (r[i] %4) == 2 or (r[i] %4)␣
↪== 3:
        times[i] = times[i] +15
        #print('Seat Interference')








#print(airplane)
#print(random.randint(1,25))
#print(choice(airplane))
#print('QUEUE')
#print(queue)
#print('SEATS')
#print(seats)
#print('TIMES')
#print(times)
print('Avg time to Board and find seat is %d seconds' %(np.mean(times)))
print('Std Dev is %d seconds' %(np.std(times)))


plt.bar(np.sort(queue), times)
plt.xlabel('Queue')
plt.ylabel('Time/s')
#count, bins, ignored = plt.hist(times,100,density=True)
plt.show()
#print(queue[0])
#print(len(lis))
#for i in range(1,101):
#    print('i = %d count = %d'% (i, lis.count(i)))
#for i in range(0,100):
    #print('Seat %d is assigned to person %d. Person %d is in column %d row␣
↪%d'% (i+1, lis[i],lis[i], (i % 4)+1, np.floor(i/4)+1 ))
```

```
[11, 42, 76, 44, 89, 13, 77, 70, 98, 41, 87, 19, 81, 97, 96, 21, 12, 16, 28, 45,
7, 61, 30, 75, 35, 82, 59, 94, 92, 40, 5, 52, 95, 23, 62, 31, 57, 49, 73, 15, 3,
66, 22, 74, 60, 34, 27, 69, 85, 20, 54, 71, 83, 47, 2, 64, 86, 25, 46, 38, 79,
68, 90, 93, 29, 26, 32, 65, 91, 78, 6, 53, 48, 24, 56, 99, 51, 33, 58, 37, 63,
67, 80, 10, 72, 50, 88, 14, 1, 0, 36, 8, 4, 18, 9, 17, 84, 43, 39, 55]
Avg time to Board and find seat is 142 seconds
Std Dev is 37 seconds
```



2 Simulation over 1000 planes with 100 passengers boarding

[17]:
```python
import numpy as np
import scipy as sci
from matplotlib import pyplot as plt
import random
from random import choice
import sys


#init
airplane = []
random.seed(random.randint(1,25000))

seatstaken = 0
AI=0
SI=0
S = 1000 #1000 boarding sims
N = 100 #number of people
if N % 4 != 0:
    print('ERROR: N not divisible by 4.')
    sys.exit()
seats = np.zeros(N)
times = np.zeros(N)
```

```
Totaltime =0
Avgtime = 0
Stdtime=0
totaltimes = np.zeros(S)
Avgtimes = np.zeros(S)
Stdtimes = np.zeros(S)
velocity = 0.5 #0.5 m/s speed
distance = 25 #25m from gate (first person) to first seat row (seat no 1,2,3,4)
queuedist = 0.5 #0.5m between people in boarding queue
rowdist = 1 #1m between seat rows
#so it takes (25m + (0.5*arrayindex) + (1*rowindexnumber)) * 0.5m/s time to get␣
↪from queue pos to seat row
#var form: (distance + (queuedist * i) + (rowdist * rownumber starting from 0))␣
↪* velocity
# + 30s for aisle interference
# if person[i]_row > person[i-1]_row then +30s
# + 15 secs for seat interference
# if person[i] and person[i-1] == same row and person[i-1] in column 2 or 3␣
↪then + 15s

#constructing airplane
for i in range (0,N):
    airplane.append(i)

queue = []
j=1
while j < N+1: #randomisizing the queue
    p = random.randint(1,N)
    if p not in queue:
        queue.append(p)
        j+=1


print(queue)
#print(r)
k = 0
while k < S:

    for i in range(0,N):
        r= random.sample(range(0,N),N)
        times[i] += (i*queuedist)/velocity +(distance / velocity) + (rowdist *␣
↪np.floor(r[i]/4))/velocity #time to get to seat row
        if np.floor(r[i]/4) > np.floor(r[i-1]/4):
            times[i]= times[i] + 30
            #print('Aisle Interference')
            AI += 1
        if np.floor(r[i]/4) == np.floor(r[i-1]/4) and (r[i] %4) == 2 or (r[i]␣
↪%4) == 3:
            times[i] = times[i] +15
            #print('Seat Interference')
            SI +=1
        #print(Totaltime)

    Totaltime = np.sum(times) #total time per plane for all passengers to board␣
↪and find a seat
    Avgtime = np.mean(times) #avg time for a person to board and find seat
    Stdtime = np.std(times) #std for person to board and find seat
    times = np.zeros(N)
    totaltimes[k] = Totaltime
    Avgtimes[k] = Avgtime
    Stdtimes[k] = Stdtime

    #print('Total time to board plane %d: %f seconds' %(k,Totaltime))
    #print('Total time to board plane %d: %f mins' %(k,Totaltime/60))
    #print('Total time to board plane %d: %f hours' %(k,Totaltime/3600))



    #print(k)
    k+=1

print('SIM FOR %d PEOPLE BOARDING %d PLANES' %(N, S))
#print(totaltimes)
print('Average total boarding time:%f seconds or %f hours' % (np.
↪mean(totaltimes),np.mean(totaltimes)/3600 ))
print('Standard Deviation in total boarding time:%f seconds or %f hours' % (np.
↪std(totaltimes), np.std(totaltimes)/3600))
print('Average time for passenger to board and find seat is %f seconds' %(np.
↪mean(Avgtimes)))
print('Standard deviation in time taken for a person to board and find a seat␣
↪is: %f seconds' %(np.mean(Stdtimes)))
```

```
[19, 97, 48, 73, 10, 26, 35, 2, 54, 18, 14, 6, 11, 66, 59, 45, 33, 95, 27, 78,
98, 57, 41, 60, 99, 46, 4, 28, 80, 87, 92, 91, 30, 89, 24, 39, 70, 8, 50, 93, 7,
56, 1, 96, 81, 77, 36, 23, 72, 61, 79, 65, 84, 90, 38, 63, 37, 53, 49, 88, 58,
44, 100, 76, 20, 62, 21, 40, 47, 52, 94, 55, 22, 69, 51, 25, 74, 32, 3, 71, 43,
42, 85, 15, 12, 5, 64, 34, 29, 13, 16, 17, 83, 31, 67, 9, 75, 68, 86, 82]
SIM FOR 100 PEOPLE BOARDING 1000 PLANES
Average total boarding time:14185.988000 seconds or 3.940552 hours
Standard Deviation in total boarding time:275.575877 seconds or 0.076549 hours
Average time for passenger to board and find seat is 141.859880 seconds
Standard deviation in time taken for a person to board and find a seat is:
```

```
39.221809 seconds
```

```
[ ]:
```

Statistical Methods Fall 2020 Answers to Problem Set 4

Jake Muff
Student number: 015361763
07/10/2020

1 Question 2: Test Statistic

For this question we have two gaussian distributions. For the pions $\mu = 0$ as it is centered at 0 and $\sigma = 1$. For the kaons $mu = 3.0$ and $\sigma = 1$.
So for this question the hypothesis H_0 is for the pions with a probability distribution of

$$g(w|H_0) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(w-0)^2}$$

And the hypothesis H_1 with a pdf

$$g(w|H_0) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(w-3)^2}$$

So that

$$H_0 = \pi \ , \ H_1 = K$$

1. What is the kaon selection efficiency when requiring $w > 2.0$?

$$\epsilon_K = \int_{-\infty}^{2} g(w|K) dw$$
$$= \int_{-\infty}^{2} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(w-3)^2}$$
$$= 0.158655$$

Or, alternatively calculate

$$\alpha = \int_{2}^{\infty} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(w-3)^2} dw$$

$$= 0.841345$$

Where

$$\epsilon_K = 1 - \alpha$$

So

$$\epsilon_K = 0.158655$$

2. What is the probability that a pion will be accepted as a kaon when requiring $w > 2.0$?

$$\int_2^\infty \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(w)^2}$$
$$= 0.0227501$$

3. Suppose a sample of particles consists of 95 % pions and 5 % kaons. What is the purity of the kaon sample selected by $w > 2.0$?
   Purity from GC can be defined as

$$p_K = \frac{\int_{-\infty}^{w} a_K \cdot g(w|K)dw}{\int_{-\infty}^{w}(a_K g(w|K) + (1 - a_K)g(w|\pi))dw}$$

   With $a_K = 5\%$. This gives us

$$p_K = 0.00847$$
$$p_K = 0.847\%$$

4. Design a kaon selection for the same sample that gives real kaons in at least 4 cases out of 5. What will be the requirement on w?
   Question is basically asking what should the cut value be at a minimum for a sample of kaons with a purity of 80%. To answer this use the equation from GC

$$\frac{g(w|K)}{g(w|\pi)} > 0.8$$

$$\frac{\frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(w-3)^2}}{\frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}w^2}} > 0.8$$

   Which reduces down to

$$e^{\frac{6w-9}{2}} > 0.8$$

   So

$$w > 1.4256$$