Written by : Lance ,e-mail : 981103924@qq.com

队列模型实验报告

一、 实验目的

应用 M/M/1 队列编程思想，模拟商店购物排队结账的过程，熟悉离散事件推进方式、队列建立和提取方式。

二、 数学模型

1、 本次实验的模型核心是创建一条顾客到达事件的时间序列和一支等待队列，利用进程交互法，根据顾客到达事件及其时间节点作为整个模拟过程的分界点，对于每个到来的顾客的时间点，输出该时间点之前服务器进程发生的事件如结账离开、进入服务后，把当前时间推进到该顾客到来的时刻，然后根据队列和服务窗口情况模拟该顾客接下来的动作。

2、 本次实验中，顾客到达服从泊松分布，则时间间隔服从指数分布；每个顾客的收费时间按指数分布生成。

泊松分布的概率函数为

$$\mathrm{P(X=k)}=\frac{\lambda^k}{k!}e^{-\lambda},k=0,1\ldots$$

泊松分布的参数λ是单位时间内随机事件的平均发生率，因此泊松分布适用于描述单位时间内随机事件的发生数。在本次实验中输入的是顾客的平均到达时间t_{arr}，因此对应于泊松分布的参数$\lambda=\frac{1}{t_{arr}}$，然后在任何时间长度 t 内顾客到达数 N(t)服从参数 λt的泊松分布，则

$$\mathrm{P(N(t)=k)}=\frac{(\lambda t)^k}{k!}e^{-\lambda t},k=0,1\ldots$$

，求两个顾客到达的时间间隔 T 的概率分布，事件{T≤t}说明 t 内顾客到达数大于等于 1，因此 P{T≤t}=1-P{T>t}=1-P{N(t)=0}=1-$e^{-\lambda t}$，符合指数分布。

指数分布的概率函数为

$$\mathrm{F(x;\lambda)}=\begin{cases}1-e^{-\lambda x},x\geq 0\\0,x<0\end{cases}$$

其中λ是每单位时间事件发生次数，其概率函数描述了时间 x 内事件发生的概率。因此若要求顾客到达的时间间隔或者服务时间，先由均匀分布随机产生一

个概率μ，求解方程$\mu = 1 - e^{-\lambda x}$得出的 x 即为时间间隔或者服务时间。

3、 队列模型本身可以看作是一种“加入-离开”过程，与传染病模型的模式比较相像，请同学们给出自己对两类模型的比较分析。

队列模型只有一条队列，并且人一旦进入队列，离开队列之后不能重新进入队列。

传染病模型中的 SIS 模型相当于有两条队列，一条是健康人队列，一条是病人队列，人在这两条队列中互相转换，一旦离开这个队列后仍有可能回到这个队列；但传染病模型中的 SIR 模型相当于有三条队列，一条是健康人队列，一条是病人队列，一条是病愈者队列，人一旦进入某条队列，离开该队列后不能重新进入。

队列模型中，人离开队列的时间和上一个离开队列的人有关。

传染病模型中，人离开当前所在的队列和很多因素有关，例如感染率和感染人数和治愈率等。

4、 简述事件调度法、活动扫描法和进程交互法的异同。

事件调度法：核心是把时钟推进到下一事件发生的时间，并认为下一事件发生且执行它。

活动扫描法：使用固定的时间间隔和基于规则的方法来决定是否开始一个活动。在每一个时间步长，检查条件，如果条件为真则触发相应活动。

进程交互法：一个进程要处理实体在系统中流动时发生的所有事件，每个实体可能在进程中的不同时间点上被阻塞或延迟。

它们的相同之处都是根据当前的时钟执行当前可以发生的事件，事件都是按照时间顺序推进的。

但事件调度法可以跳过没有事件发生的仿真时段，而活动扫描法只能每次推进一个时间步长，进程交互法则根据当前时间点决定哪些事件可以发生，哪些进程需要被阻塞。

三、 编程实现

1. 在本次实验实现的单服务器排队模型中，实体为排队结账的顾客和收费服务台。

顾客属性有顾客编号、顾客排队间隔时间、顾客结账时间、顾客到达时间。顾客类定义代码如下：

```
class customer:
```

```
    def __init__(self, interTime, serTime, id,arrTime):
        self.id = id
        self.intervalTime = interTime
        self.serveTime = serTime
        self.arrivalTime = arrTime
```

收费服务台属性有当前时间、服务顾客的总时间、服务顾客的总数、顾客序列、等待队列和服务窗口（当服务窗口为 1 时为 M/M/1 模型）。服务台类定义代码如下：

```
class server :
    def __init__(self, customers, wqueue, num) :
        self.customers = customers
        self.wQueue = wqueue
        self.currentTime = 0.0
        self.serveTime = 0.0
        self.serveNum = 0
        self.subservers = []
        for i in range(num) :
            self.subservers.insert(len(self.subservers), subserver(i))
```

事件为顾客到达收费台、顾客结账结束离开。

活动为顾客接受服务、顾客进入等待队列、顾客因为等待队列满而放弃消费。

本实验采用时间的异步模型，仿真会跳过没有时间发生的时间段。

模拟过程如下：首先根据将要到达的顾客生成未来事件列表，即构成一个顾客流，再创建一个等待队列，当顾客没有到完或者等待队列中仍有顾客时，首先找出最快结束服务的服务窗口，若该窗口的服务结束的时间小于下一位顾客到达的时间， 则将当前时间推进到该窗口服务结束的时刻，并服务等待队列中队首的顾客，重复该操作至所有窗口为空或当前时间推进到了下一位顾客到来。接着，模拟下一位顾客到来的情形，将当前时间推进到该顾客到来的时间，对于该顾客而言，若所有窗口都有顾客且排队队列已满，则放弃消费离开，否则若有空窗口，则前去该窗口结账，否则若无空窗口但队列未满，则参与排队，重复上述过程至所有顾客都完成对应活动。模拟过程代码如下：

```
  def mmnSimulation(self) :
        while len(self.customers) != 0 or self.wQueue.size() != 0 :
            # 把在前面的事件输出
            while True :
                mSer = self.minSer()
```

```
                if mSer.isBusy and (len(self.customers) == 0 or mSer.endTime <= self.customers[0].ar
rivalTime) :
                    self.currentTime = mSer.endTime
                    print("id : ",mSer.id," Time : ", self.currentTime, " No. ", mSer.serCus.id," le
aves after serving\n")
                    self.wQueue.totalDelayTime += self.currentTime - mSer.serCus.arrivalTime
                    self.serveTime += mSer.serCus.serveTime
                    self.serveNum += 1
                    self.record("leaves after serving", mSer.serCus.id)
                    mSer.isBusy = False
                    mSer.endTime = float("inf")
                    if self.wQueue.size() !=0 :
                        cus = self.wQueue.getCus()
                        print("Time : ", self.currentTime, " No. ", cus.id," is serving\n")
                        self.record("serving", cus.id)
                        mSer.serveCus(cus, self.currentTime)
                else :
                    break
            if len(self.customers) == 0 and self.wQueue.size() == 0 :
                break
            # 推进到当前时间
            curCus = self.customers[0]
            del self.customers[0]
            self.currentTime = curCus.arrivalTime
            print("Time : ", self.currentTime, " No. ", curCus.id," arrives\n")
            self.record("arrives", curCus.id)

            if self.wQueue.size() >= self.wQueue.maxLen and self.isAllBusy() :
                print("Time : ", self.currentTime, " No. ", curCus.id," leaves cuz full queue\n")
                self.record("leaves cuz full", curCus.id)
                continue
            if self.isAllBusy() == False :
                ser = self.getFreeSer()
                ser.serveCus(curCus, self.currentTime)
                print("Time : ", self.currentTime, " No. ", curCus.id," is serving\n")
                self.record("serving", curCus.id)
                continue
            if self.wQueue.size() < self.wQueue.maxLen :
                self.wQueue.addCus(curCus)
                print("Time : ", self.currentTime, " No. ", curCus.id," starts waiting\n")
                self.record("waiting", curCus.id)
```

2. 本次实验中，顾客到达服从泊松分布，则时间间隔服从指数分布；收费时间按指数分布生成。生成顾客到达时间间隔和收费时间如下：

```
# produce the interval time sequence refer to poisson_distribution
arrTimes = npy.random.exponential(float(averArrTime),customerNum)
# produce the serval time sequence refer to exponential_distribution
serTimes = npy.random.exponential(averSerTime,customerNum)
customers = []
countTime = 0.0
# produce the customer sequence
for i in range(0,customerNum) :
    countTime += arrTimes[i]
    cus = customer(arrTimes[i], serTimes[i], i, countTime)
    customers.insert(len(customers), cus)
```

当顾客到达时，时间推进到该顾客到达的时间并输出提示信息“Time : current time No. customer id arrives”，如果队列未满且当前所有窗口有顾客结账的话，加入等待队列，输出提示信息“Time : current time No. customer id arrives and starts waiting”，如果队列满了，顾客放弃消费并输出提示信息“Time : current time No. customer id leaves because queue is full”,如果队列未满且当前有窗口无顾客消费，直接进入消费台，输出提示信息“Time : current time No. customer id is serving”；当顾客结账结束时，时间推进到该顾客结账结束的时间并输出提示信息“Time : current time No. customer id leaves after serving”。

3. 给出输入参数（平均到达时间，平均服务时间，顾客数目，队列最大常数，服务窗口数），输出参数（单位时间顾客到达数，单位时间服务人数，队列中平均等待顾客数，顾客平均等待时间，服务器服务率）的具体值，可以用图或表的方式展示多次仿真的结果值。

下面是 M/M/1 排队模型的仿真结果：

假定平均服务时间为 120s

（1）商店销售淡的时候，每日顾客不多约为 200 人，顾客平均到达时间较大，截取一段时间模拟

输入：

平均到达时间 432s

平均服务时间 120s

顾客数目 50 人

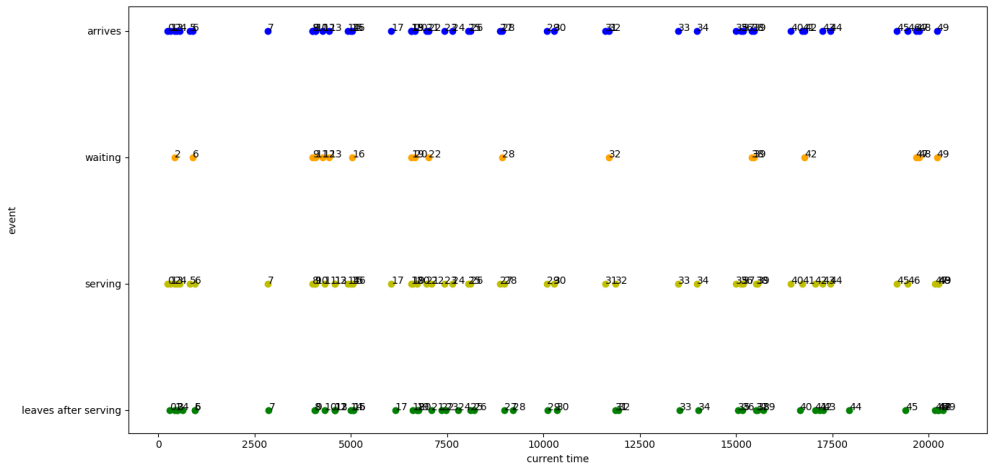
队列最大长度 10 人

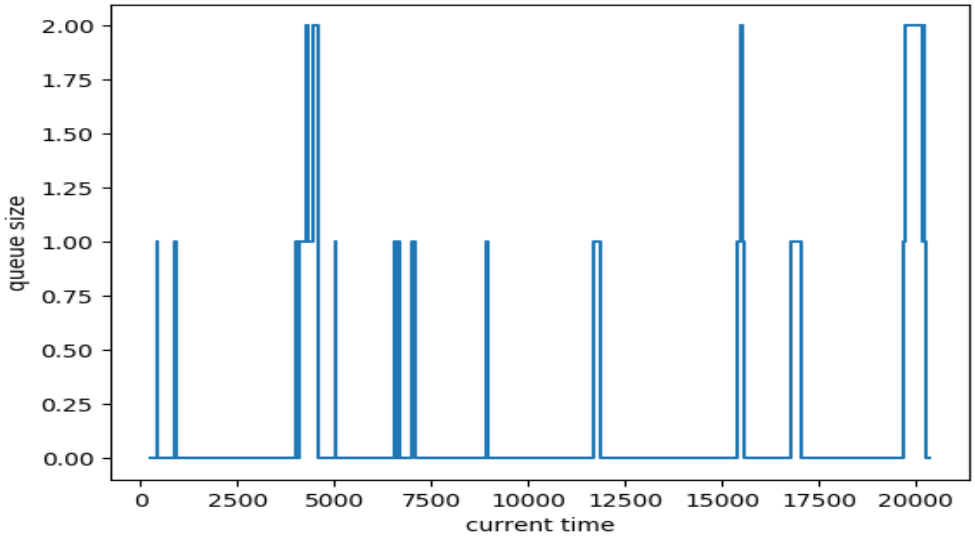
服务窗口数 1 个

```
请输入平均到达时间、平均服务时间、顾客数、队列最大长度、服务器数
432 120 50 10 1
```

输出：
单位时间顾客到达数 0.002472902938765979　人/秒
单位时间服务人数 0.00828221387045155　人/秒
队列中平均等待顾客数 0.429230673455488　人
顾客平均等待时间 174.91763319748515　秒
服务器服务率 0.29628571725759234

```
the number of arriving people per s :  0.002472902938765979  p/s

the number of serving people per s :  0.00828221387045155  p/s

average number of people in queue :  0.429230673455488  p

average number of waiting time :  174.91763319748515  s

rate of using the server :  0.29628571725759234
```





由此可见，在销售淡季的时候，顾客前来结账排队的概率很小，即使排队等待时间也不会很长，都能快速获得结账服务。

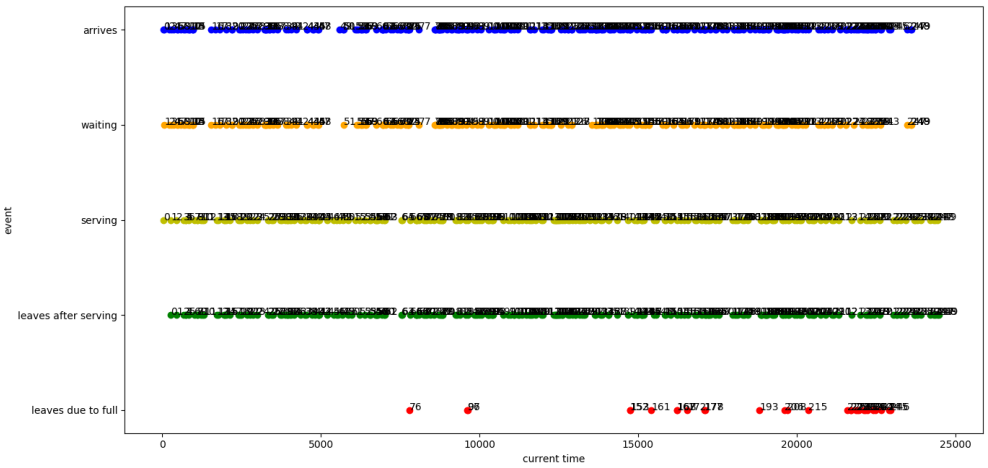
（2）商店销售旺的时候，每天顾客很多约 1000 人，顾客平均到达时间小，截一段时间模拟

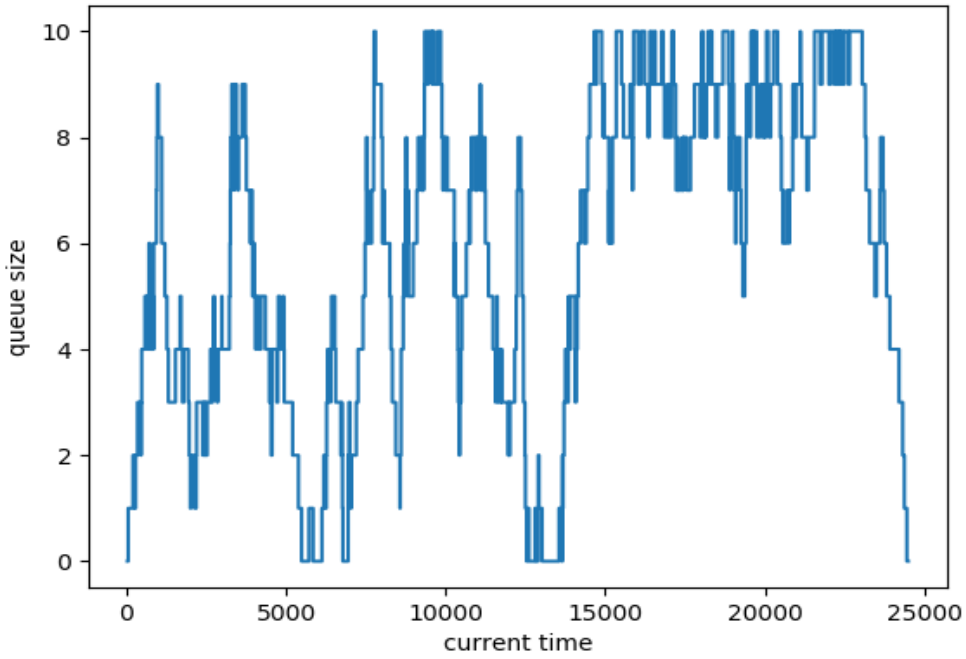
输入
平到达时间 86.4s
均服务时间 120s
顾客数目 250 人
队列最大长度 10 人
服务窗口数 1 个

```
请输入平均到达时间、平均服务时间、顾客数、队列最大长度、服务器数
86.4 120 250 10 1
```

输出：
单位时间顾客到达数 0.010582114728668771　人/秒
单位时间服务人数 0.009269696468387618　人/秒
队列中平均等待顾客数 6.857622374223935　人
顾客平均等待时间 759.8635247172947　秒
服务器服务率 0.9735818184313284

```
the number of arriving people per s :  0.010582114728668771  p/s
the number of serving people per s :  0.009269696468387618  p/s
average number of people in queue :  6.857622374223935  p
average number of waiting time :  759.8635247172947  s
rate of using the server :  0.9735818184313284
```





由此可见，在销售旺季的时候，每个顾客前来结账排队的概率很大，并且随着时间的推移越来越多顾客到达时，排队的可能性越大，也出现了比较多的因为

队列满而放弃消费的顾客，平均等待时间较长。

若商店平均服务效率提高即平均服务时间减少，假设为 100s，则仿真结果如下：

（1）商店销售淡的时候，每日顾客不多约为 200 人，顾客平均到达时间较大，截取一段时间模拟

输入：

平均到达时间 432s

平均服务时间 100s

顾客数目 50 人

队列最大长度 10 人

服务窗口数 1 个

```
请输入平均到达时间、平均服务时间、顾客数、队列最大长度、服务器数
432 100 50 10 1
```

输出：

单位时间顾客到达数 0.002449542011875063　人/秒

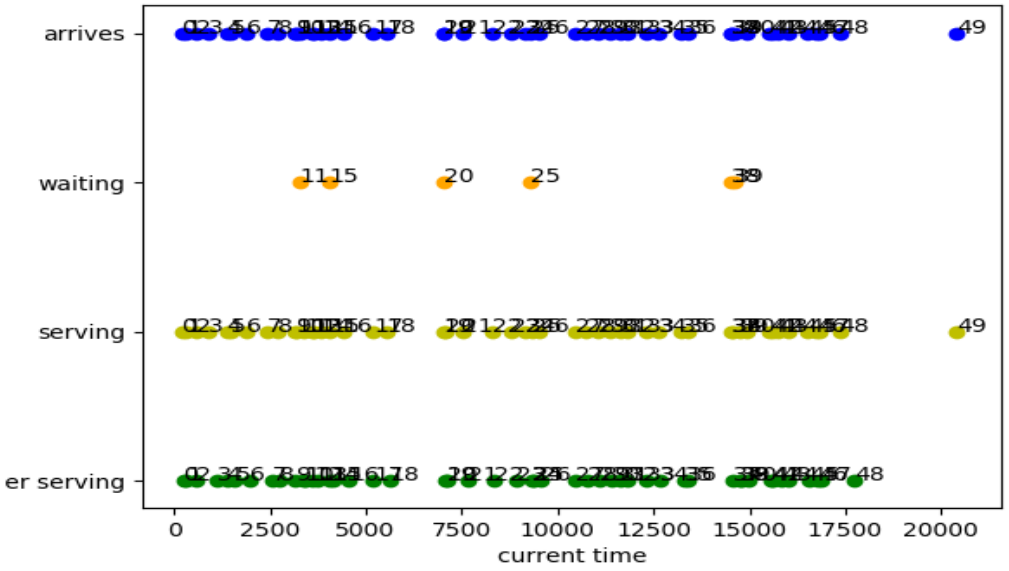
单位时间服务人数 0.012812083686743391　人/秒

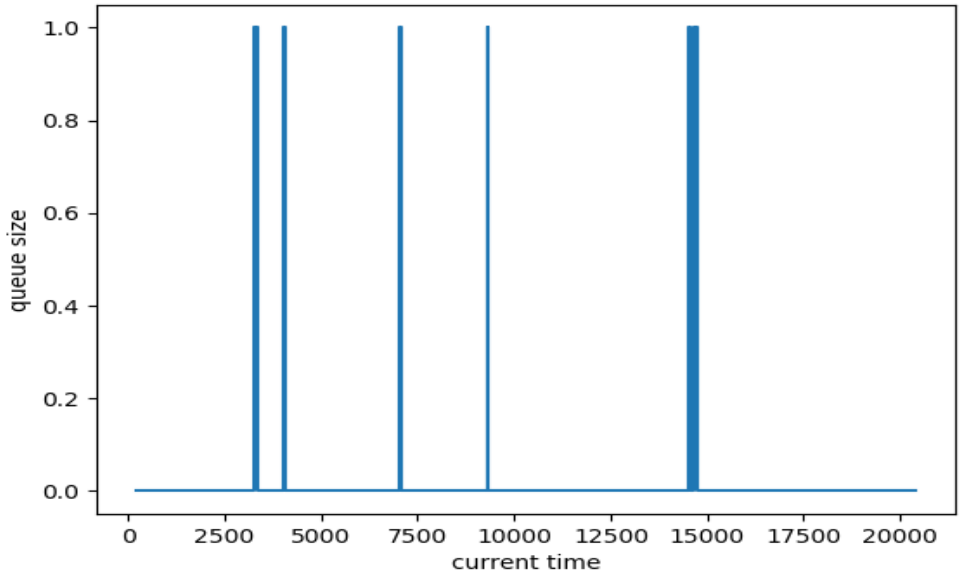
队列中平均等待顾客数 0.20487562527258926　人

顾客平均等待时间 85.34524391447619　秒

服务器服务率 0.1873661794858085

```
the number of arriving people per s :  0.002449542011875063  p/s

the number of serving people per s :  0.012812083686743391  p/s

average number of people in queue :  0.20487562527258926  p

average number of waiting time :  85.34524391447619  s

rate of using the server :  0.1873661794858085
```





对比可见，销售淡季时，在平均服务效率提升的情况下，顾客平均排队时间减少，队列平均等待顾客数减小。

（2）商店销售旺的时候，每天顾客很多约 1000 人，顾客平均到达时间小，截一段时间模拟

输入
平到达时间 86.4s
均服务时间 100s
顾客数目 250 人
队列最大长度 10 人
服务窗口数 1 个

```
请输入平均到达时间、平均服务时间、顾客数、队列最大长度、服务器数
86.4 100 250 10 1
```

输出：
单位时间顾客到达数 0.011845058179756214 人/秒
单位时间服务人数 0.011011259551758123 人/秒
队列中平均等待顾客数 5.3434446641222015 人
顾客平均等待时间 519.7554669890031 秒
服务器服务率 0.9336524698957096

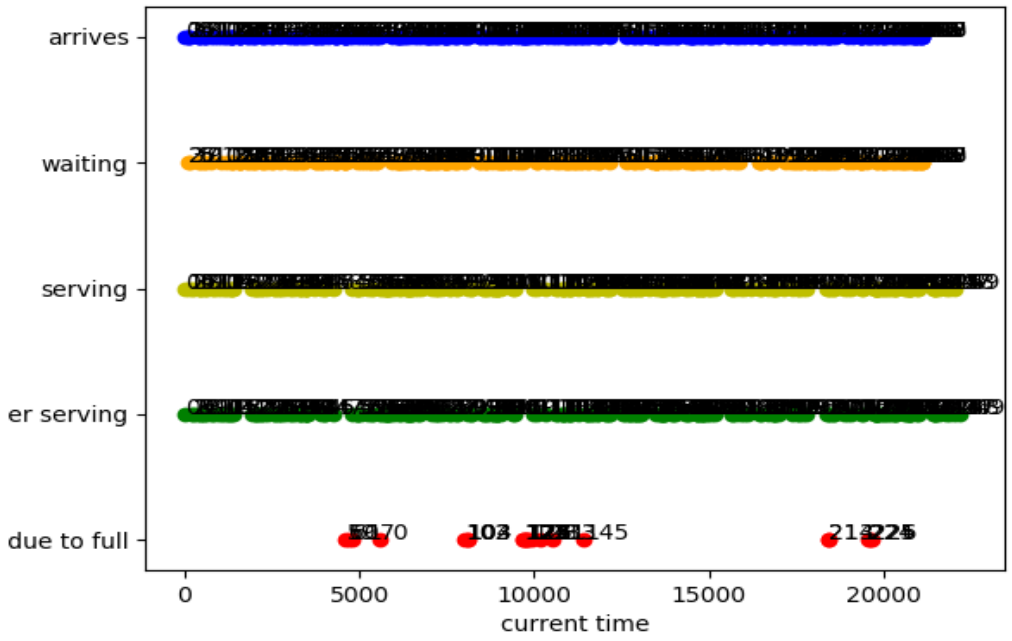
the number of arriving people per s : 0.011845058179756214 p/s

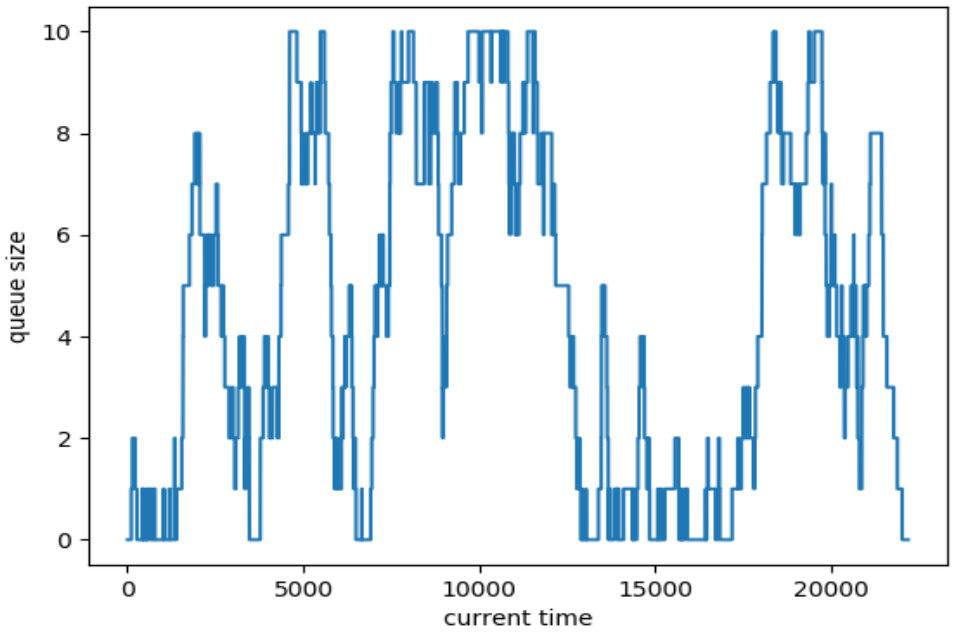
the number of serving people per s : 0.011011259551758123 p/s

average number of people in queue : 5.3434446641222015 p

average number of waiting time : 519.7554669890031 s

rate of using the server : 0.9336524698957096





对比可见，在销售旺季，在平均服务效率提升的情况下，顾客平均排队时间减小，队列平均等待顾客数减小，但仍存在因为队列满而放弃消费的顾客。

4. 编程模拟或纯数学分析 M/M/n 模型。

在笔者的编程实现下，模拟 M/M/n 模型仅需改变输入即可。

其中输出的参数均对应总服务台而非其中的服务窗口。

针对第三问给定的情况，下面是 M/M/n 模型仿真结果，以 n=3 为例：

（1）商店销售淡的时候，每日顾客不多约为 200 人，顾客平均到达时间较大，截取一段时间模拟

输入：

平均到达时间 432s

平均服务时间 120s

顾客数目 50 人

队列最大长度 10 人

服务窗口数 3 个

```
请输入平均到达时间、平均服务时间、顾客数、队列最大长度、服务器数
432 120 50 10 3
```

输出：

单位时间顾客到达数 0.002330360454285289 人/秒

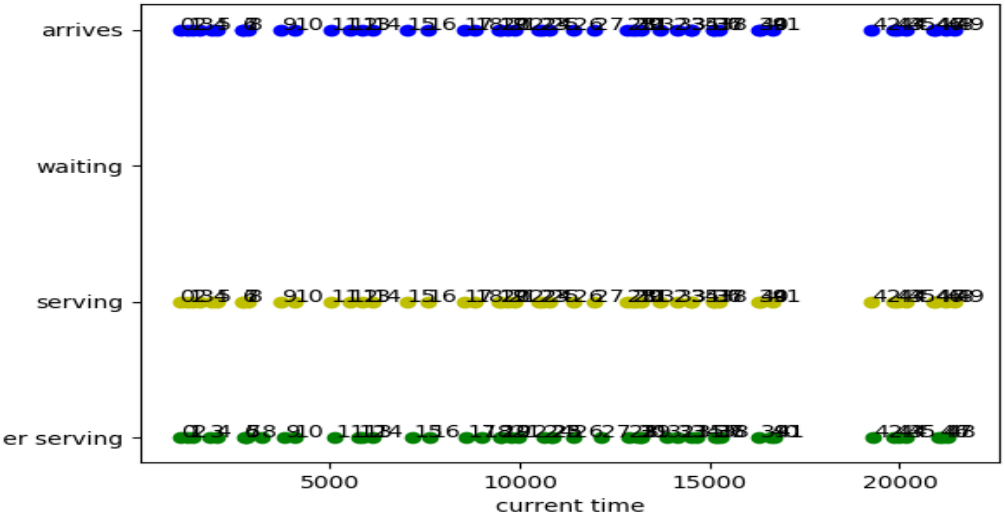
单位时间服务人数 0.009476561332247179 人/秒

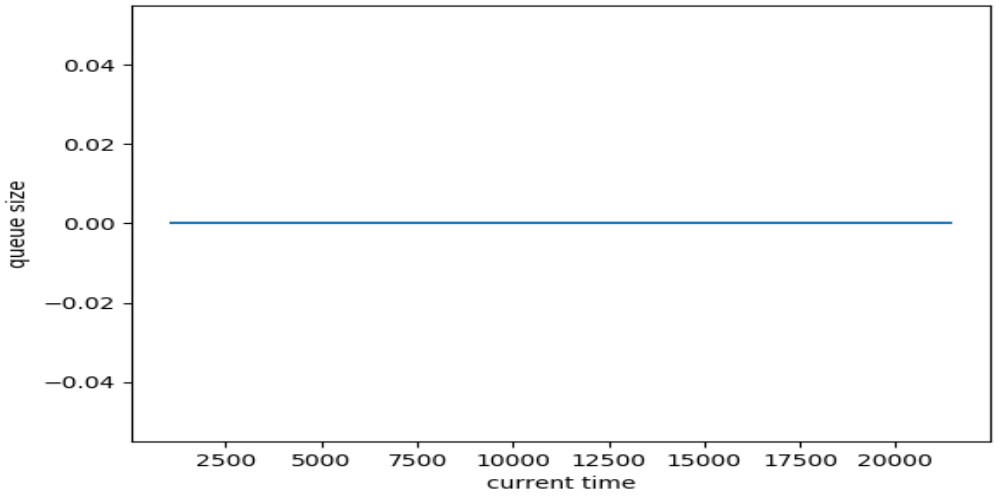
队列中平均等待顾客数 0.24098965491083185 人

顾客平均等待时间 105.52350846895945 秒

服务器服务率 0.24098965491083213

```
the number of arriving people per s :  0.002330360454285289  p/s

the number of serving people per s :  0.009476561332247179  p/s

average number of people in queue :  0.24098965491083185  p

average number of waiting time :  105.52350846895945  s

rate of using the server :  0.24098965491083213
```





由此可见，在销售淡季时，相同情况下，若服务窗口数为 3，则每位顾客到来都能直接得到结账服务，不需要排队。

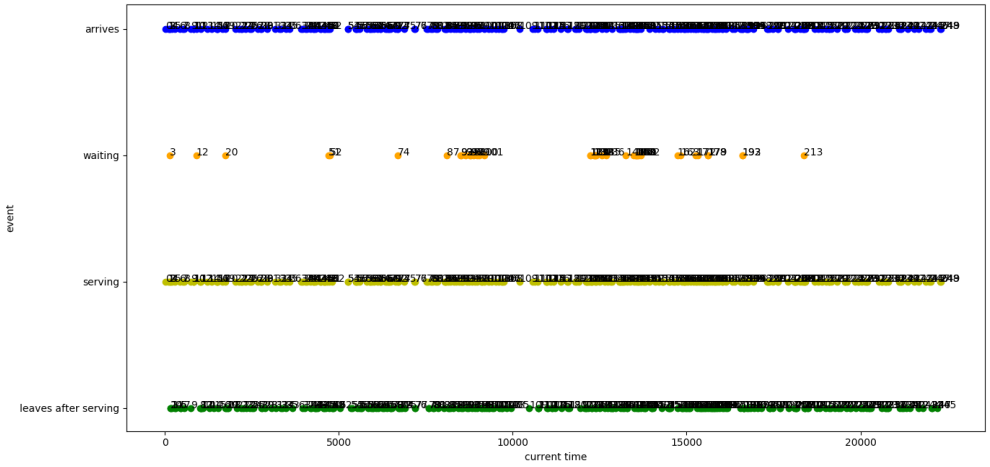
（2）商店销售旺的时候，每天顾客很多约 1000 人，顾客平均到达时间小，截一段时间模拟

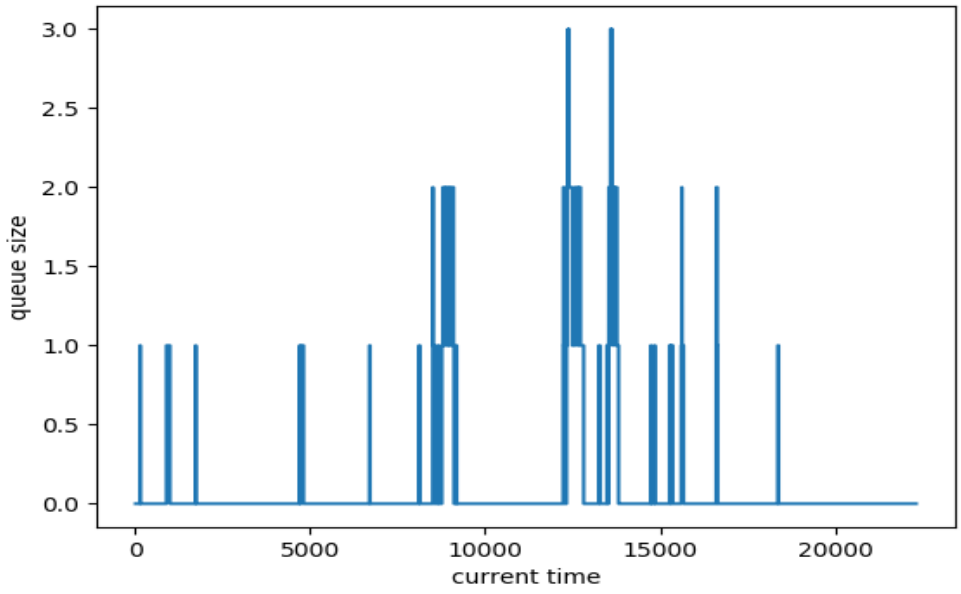
输入
平到达时间 86.4s
均服务时间 120s
顾客数目 250 人
队列最大长度 10 人
服务窗口数 3 个

```
请输入平均到达时间、平均服务时间、顾客数、队列最大长度、服务器数
86.4 120 250 10 3
```

输出：
单位时间顾客到达数 0.011205429999079001　人/秒
单位时间服务人数 0.00880509313050253　人/秒
队列中平均等待顾客数 1.3761602484118554　人
顾客平均等待时间 123.80232753632191　秒
服务器服务率 1.2624269152337702

```
the number of arriving people per s :  0.011205429999079001  p/s
the number of serving people per s :  0.00880509313050253  p/s
average number of people in queue :  1.3761602484118554  p
average number of waiting time :  123.80232753632191  s
rate of using the server :  1.2624269152337702
```





由此可见，在销售旺季，在相同条件下，服务窗口数为 3 时，顾客前来基本不用排队等候就能得到结账服务，也几乎不存在因队列满而放弃消费的情况。

四、 以绝味鸭脖销售店为例，给出初步的窗口优化策略

1. 该场景的拓扑结构：

绝味鸭脖销售店定位于大众人群，绝大多数的门店是社区店，由于是社区店，大多店中一般只有一个服务人员，因此适用于 M/M/1 模型。

2. 人流特点：

绝味鸭脖销售店一般开设在十字路口或者社区附近，在上下班时期或者就餐时间人流量较大，会出现高峰期，平时人流量较少。

3. 优化目的：

绝味鸭脖销售店由于是零售为主，服务人员不多，在高峰期的时候很容易出现顾客因队列满或人数过多而放弃消费，造成销售额的潜在损失，因此可以通过优化窗口来赚取该部分销售额。

4. 改进措施：

由上述仿真结果可以看到，当人流过多时，可以通过提高服务效率的方式减少顾客等待时间，但是优化幅度不大。也可以通过增加服务人员的方式来减少顾客等待时间，这样能过得很好的优化效果，但同时成本也会随之提高。因此若要优化窗口，可以加大员工培训力度，提升服务效率，同时在控制成本的情况下，对人流特别密集的门店，适当增加服务人数。