ps08-sols

April 8, 2024

[10]:
```
import numpy as np
import math as math

def poisson(p,m):
   return (1.*p**m/ math.factorial(m))*np.exp(-p)

from numpy.random import default_rng
rng=default_rng()

def arrival(p=1./7,M = 10,rng=default_rng()):
  qq = list(map(lambda m:poisson(p,m),range(M)))
  qq = qq + [1-sum(qq,0)]

  return rng.choice(list(range(M+1)),p=qq)
```

[11]:
```
class JFTE():
    def __init__(self,N,prob=1./7):
        self.customers = [arrival(prob) for n in range(N)]
        self.num_days = N
        self.reset()

    def reset(self):
        self.stock = 1
        self.sales = 0
        self.lost_sales = 0
        self.storage_days = 0
        self.max_stock = 1

    def add_stock(self):
        self.stock = self.stock + 1
        if self.stock > self.max_stock:
            self.max_stock = self.stock

    def sale(self):
        self.stock = self.stock - 1
        self.sales = self.sales + 1

    def result(self):
        return { 'number_days': self.num_days,
                 'weeks': self.num_days/7.0,
                 'sales': self.sales,
                 'lost_sales': self.lost_sales,
                 'storage_days': self.storage_days,
                 'max_stock': self.max_stock
               }
```

[12]:
```
def stand_order(J,dow=6):
    ## dow = arrival day-of-week for standing order; should be in␣
 ↪[0,1,2,3,4,5,6]
    ## we'll assume that the first day of the ``days`` list is dow=0.

    N = J.num_days
    J.reset()

    # loop through the days
    for i in range(N):
        c = J.customers[i]            ## c is 1 if there is a customer on day␣
 ↪i, 0 otherwise

        if dow == np.mod(i,7):        ## add stock on the dow for order arrival
            J.add_stock()

        if c>0 and J.stock == 0:
            J.lost_sales = J.lost_sales + 1   ## lost sale if no stock

        if c>0 and J.stock > 0:               ## sale if adequate stock
            J.sale()

        J.storage_days = J.storage_days + J.stock    ## accumulate total␣
 ↪storage costs

    return J.result()
```

[13]:
```
def order_on_demand(J):
    J.reset()
    order_wait = np.inf                         ## order_wait represents␣
 ↪wait-time
                                                ## until next order arrival

    ## loop through the customers
    for c in J.customers:
        if c>0 and J.stock==0:                  ## record lost sale if no stock
            J.lost_sales = J.lost_sales + 1

        if c>0 and J.stock>0:                   ## record sale if adequate stock
```

```
            J.sale()

        J.storage_days += J.stock              ## accumulate storage days

        if  J.stock==0 and order_wait == np.inf:  ## reorder if stock is empty␣
↪and no current order
            order_wait = 5

        if order_wait == 0:                    ## stock arrives
            J.add_stock()
            order_wait = np.inf

        if order_wait>0:                       ## decrement arrival time for␣
↪in-transit orders
            order_wait -= 1

    return J.result()
```

We now create the trials

[14]:
```
import pandas as pd

def make_trials(trial_weeks = 2*52, num_trials = 10):
    return [ JFTE(7*trial_weeks) for _ in range(num_trials) ]

def report_trials(strategy,trials):

    results = [ strategy(t) for t in trials ]

    details = ['weeks', 'sales', 'lost_sales', 'storage_days', 'max_stock']

    sd = {i: [r[i] for r in results ] for i in details}

    return pd.DataFrame(sd)

## make a list of 10 trials. Each trial has length 2 years
ten_trials = make_trials()
```

[15]:
```
stand_results = report_trials(stand_order,ten_trials)
print(stand_results)
```

```
   weeks  sales  lost_sales  storage_days  max_stock
0  104.0     93           3          6413         17
1  104.0     98           2          4060         13
2  104.0    101           2          3785         13
3  104.0     99           0          5862         13
4  104.0     91           0          8886         20
5  104.0     78           0         13714         32
```

```
6  104.0     97           6          6582          19
7  104.0     99           1          3391          10
8  104.0    101           6          1500           7
9  104.0    103           5          2013           7
```

[16]:
```
stand_results.mean()
```

[16]:
```
weeks            104.0
sales             96.0
lost_sales         2.5
storage_days    5620.6
max_stock         15.1
dtype: float64
```

[20]:
```
stand_results.std()
```

[20]:
```
weeks              0.000000
sales              7.302967
lost_sales         2.415229
storage_days    3629.302970
max_stock          7.445356
dtype: float64
```

[17]:
```
demand_results = report_trials(order_on_demand, ten_trials)
demand_results
```

[17]:
```
   weeks  sales  lost_sales  storage_days  max_stock
0  104.0     62          34           356          1
1  104.0     56          44           393          1
2  104.0     61          42           362          1
3  104.0     56          43           394          1
4  104.0     58          33           380          1
5  104.0     51          27           422          1
6  104.0     62          41           356          1
7  104.0     61          39           362          1
8  104.0     63          44           350          1
9  104.0     61          47           362          1
```

[19]:
```
demand_results.mean()
```

[19]:
```
weeks           104.0
sales            59.1
lost_sales       39.4
storage_days    373.7
max_stock         1.0
dtype: float64
```

[ ]:
```

```