# PS07

April 2, 2024

[46]:

```
import numpy as np

from numpy.random import default_rng
rng = default_rng()

def randomPoint():
    # return a random point in the rectangl [1/2,1] x [0,4]
    return (rng.uniform(1/2,1),rng.uniform(0,4))

def estimate(n):
    ll = [ randomPoint() for _ in range(n) ]    # make a list of n random points
    lr = [ (x,y) for (x,y) in ll if y <= 1/x ]  # find the points below the␣
  ↪curve
    return len(lr)/len(ll)                      # return the fraction of points␣
  ↪below the curve
```

[47]:

```
np.log(2)/2
```

[47]:

```
0.34657359027997264
```

[48]:

```
estimate(100000)
```

[48]:

```
0.3468
```

[49]:

```
[ (n,estimate(1000*n)) for n in range(10,40,2)]
```

[49]:

```
[(10, 0.3435),
 (12, 0.34475),
 (14, 0.3517857142857143),
 (16, 0.3469375),
 (18, 0.3486666666666667),
 (20, 0.34095),
 (22, 0.343),
 (24, 0.346875),
 (26, 0.3515),
 (28, 0.34567857142857145),
 (30, 0.3461),
 (32, 0.348375),
 (34, 0.34461764705882353),
 (36, 0.3483611111111111),
 (38, 0.34671052631578947)]
```

Day Sun Mon Tue Wed Thur Fri Sat DOW 0 1 2 3 4 5 6 Prob 0.16 0.08 0.04 0.08 0.12 0.25 0.27

[66]:

```python
new_dow_probs = { 0: .16,
              1: .08,
              2: .04,
              3: .08,
              4: .12,
              5: .25,
              6: .27 }
```

[61]:

```python
def customer_alt(d,dow_probs):
    pp = dow_probs[np.mod(d,7)]
    return rng.choice([1,0],p=[pp,1-pp])
```

[62]:

```python
class JFTE():
    def __init__(self,N,dow_probs):
        self.customers = [customer_alt(n,dow_probs) 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
```

```
               }


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()

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()
```

[67]:

```
import pandas as pd

def make_trials(dow_probs,trial_weeks = 2*52, num_trials = 10):
    return [ JFTE(7*trial_weeks,dow_probs=dow_probs) 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(new_dow_probs)
```

[56]:

```
stand_results = report_trials(stand_order,ten_trials)
stand_results
```

[56]:

```
   weeks  sales  lost_sales  storage_days  max_stock
0  104.0     91          12          3573         14
1  104.0    103          13          1805          7
2  104.0     97          10          1819          8
3  104.0    103           0          2083          6
4  104.0     99           0          3548          9
5  104.0     92           7          6062         16
6  104.0     86           2          6980         19
7  104.0     93           8          2630         12
8  104.0    102           6          3408         11
9  104.0     97           5          4687         14
```

[55]:
```
demand_results = report_trials(order_on_demand, ten_trials)
demand_results
```

[55]:
```
   weeks  sales  lost_sales  storage_days  max_stock
0  104.0     61          42           362          1
1  104.0     69          47           314          1
2  104.0     66          41           332          1
3  104.0     65          38           338          1
4  104.0     63          36           354          1
5  104.0     63          36           354          1
6  104.0     55          33           402          1
7  104.0     63          38           350          1
8  104.0     69          39           316          1
9  104.0     65          37           338          1
```

[57]:
```
stand_results.mean()
```

[57]:
```
weeks            104.0
sales             96.3
lost_sales         6.3
storage_days    3659.5
max_stock         11.6
dtype: float64
```

[59]:
```
demand_results.mean()
```

[59]:
```
weeks           104.0
sales            63.9
lost_sales       38.7
storage_days    346.0
max_stock         1.0
dtype: float64
```

[69]:
```
const_probs = { n: 1./7 for n in range(7) }
const_probs
```

[69]:
```
{0: 0.14285714285714285,
 1: 0.14285714285714285,
 2: 0.14285714285714285,
 3: 0.14285714285714285,
 4: 0.14285714285714285,
 5: 0.14285714285714285,
 6: 0.14285714285714285}
```

[71]:
```
## make a list of 10 trials. Each trial has length 2 years
## this time use constant probabilities
const_ten_trials = make_trials(const_probs)
```

[73]:
```
const_stand_results = report_trials(stand_order,const_ten_trials)
const_demand_results = report_trials(order_on_demand,const_ten_trials)
```

[74]:
```
const_stand_results
```

[74]:
```
   weeks  sales  lost_sales  storage_days  max_stock
0  104.0    102           1          3331         10
1  104.0     94           0          4322         12
2  104.0     98          15          1430          9
3  104.0     92           2          5146         14
4  104.0    100           9          1834          8
5  104.0    102           6          2704         10
6  104.0     98           6          2273          9
7  104.0     91           0          8034         16
8  104.0     95           2          3031         10
9  104.0    101           8          2369          9
```

[75]:
```
const_demand_results
```

[75]:
```
   weeks  sales  lost_sales  storage_days  max_stock
0  104.0     59          44           374          1
1  104.0     55          39           401          1
2  104.0     66          47           332          1
3  104.0     56          38           392          1
4  104.0     60          49           370          1
5  104.0     64          44           344          1
6  104.0     62          42           356          1
7  104.0     51          40           422          1
8  104.0     58          39           380          1
9  104.0     58          51           380          1
```

[76]:
```
const_stand_results.mean()
```

[76]:
```
weeks            104.0
sales             97.3
lost_sales         4.9
storage_days    3447.4
max_stock         10.7
dtype: float64
```

[77]:
```
const_demand_results.mean()
```

[77]:
```
weeks           104.0
sales            58.9
lost_sales       43.3
storage_days    375.1
max_stock         1.0
dtype: float64
```

[78]:

```
3447/97
```

[78]:

```
35.5360824742268
```

[79]:

```
375/58
```

[79]:

```
6.4655172413793105
```

[87]:

```
def required_profit(results):
  # we'll take the pandas DataFrame as argument

  means = results.mean()
  return means["storage_days"]/means["sales"]

required_profit(const_stand_results)
```

[87]:

```
35.430626927029806
```

[88]:

```
required_profit(const_demand_results)
```

[88]:

```
6.36842105263158
```

[82]:

```
six_month_ten_trials = make_trials(const_probs,trial_weeks=26,num_trials=20)

six_month_stand_results = report_trials(stand_order,six_month_ten_trials)
six_month_demand_results = report_trials(order_on_demand,six_month_ten_trials)

six_month_stand_results.mean()
```

[82]:

```
weeks            26.00
sales            22.45
lost_sales        2.80
storage_days    498.95
max_stock         6.15
dtype: float64
```

[84]:

```
six_month_demand_results.mean()
```

[84]:

```
26.0
```

[89]:

```
required_profit(six_month_stand_results)
```

[89]:

```
22.224944320712694
```

[90]:

```
required_profit(six_month_demand_results)
```

[90]:

```
6.239202657807309
```

[ ]: