week10-02–monte-carlo-simulation

March 22, 2024

1 George McNinch Math 87 - Spring 2024

2 Week 10

2.1 # Monte-Carlo simulation

3 A modeling application of Monte-Carlo methods: fish tanks!

In this notebook we are going to discuss a modeling example.

Suppose that you have been promoted to inventory manager at **Jane's Fish Tank Emporium** (JFTE).

JFTE sells only 150 gallon fish tanks that are bulky, so it prefers to not keep more in stock than are needed at any given point in time.

Suppose that on average JFTE sells one tank per week.

JFTE can order new tanks at any point, but they must wait 5 days for the arrival of a new tank once it has been ordered.

The task is to design a good strategy for ordering fish tanks.

4 Relevant questions & parameters??

- profit from the sale of a tank?
- cost of storage for an unsold tank in stock?
- what does "on average, one tank is sold per week" really mean??
- what strategies are even possible?

Let's consider some extremal cases first:

- if the profit per tank is large and the storage costs for an in-stock tank relatively small, then a good strategy is to keep a relatively large inventory.
- if the profit per tank is small and the storage costs for an in-stock tank are relatively large, then a good strategy is to keep little-or-no inventory and order as required.

It is difficult to formulate too many generalities without knowing further information.

An important rule of modeling we'd like to follow is this:

Start with a relatively simple model, but build it to allow incremental additions of complexity.

5 Simplifying assumptions

1. Let's assume that "on average, JFTE sells one tank per week" means that on any given day, there is a $\frac{1}{7}$ chance of an interested customer entering the store.
2. If an interested customer arrives but there is no stock, the potential sale is then *lost* (thus our model doesn't acknowledge rainchecks or instructions to a customer to "try next week").
3. The cost of storing a tank is high enough that you only want to store tanks you expect to sell "soon".

These assumptions suggest two strategies, which we want to compare.

Strategy A. Set a *standing order* to have one tank delivered each week.
Strategy B. Order a new tank whenever one is sold – *on-demand ordering*

We are going to use a Monte-Carlo simulation to compare these two strategies.

6 Our simulation

The first step is to simulate arrival of customers. We are going to make a list of N days for our simulation, and for each day we are going to use a random selection to "decide" whether a customer arrives.

For each day, we would like to keep track of various information:

- does a customer arrive? (determined randomly)
- is there a tank in stock? (ordering is determined by our strategy)

So let's create a `python` data structure which keeps track of the required information. We'll just use a `class` named `JFTE` which has instance variables `customers`, `stock`, `sales` etc.

When we construct an instance of the class, we indicate the number of days `N` for our simulation. We create a list corresponding to `days`, and the random number generated "decides" whether or not a customer will arrive on the given day.

We now implement our *strategies* as functions which take as argument an instance of the class `JFTE` and return an instance of the class `result`.

[1]:
```python
import numpy as np
import itertools as it

from numpy.random import default_rng
rng = default_rng()
```

[2]:
```python
def customer(prob=1./7):
    return rng.choice([1,0],p=[prob,1-prob])


class JFTE():
    def __init__(self,N,prob=1./7):
        self.customers = [customer() for n in range(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 num_days(self):
        return len(self.customers)

    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 result(self.num_days(),self.sales,self.lost_sales,
                      self.storage_days,self.max_stock)
```

[3]:
```
class result():
    def __init__(self,num_days,sales,lost_sales,storage_days,max_stock):
        self.num_days = num_days
        self.sales = sales
        self.lost_sales = lost_sales
        self.storage_days = storage_days
        self.max_stock = max_stock

    def report(self):
        entries = [f"weeks:        {self.num_days/7.}",
                   f"sales:        {self.sales}",
                   f"lost sales:   {self.lost_sales}",
                   f"storage_days: {self.storage_days}  (effective)",
                   f"max stock:    {self.max_stock}",
                    ]
        return "\n".join(entries)
```

The first strategy is to have a standing order made each week on the same day.

[4]:
```
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()

    for i in range(N):
        c = J.customers[i]
        if dow == np.mod(i,7):
            J.add_stock()
        if c>0 and J.stock == 0:
            J.lost_sales = J.lost_sales + 1
        if c>0 and J.stock > 0:
            J.sale()
        J.storage_days = J.storage_days + J.stock
    return J.result()
```

The second strategy is to have a order placed as soon as a sale is made.

[13]:
```
def order_on_demand(J):
    J.reset()
    order_wait = np.inf
    for c in J.customers:
        if c>0 and J.stock==0:
            J.lost_sales = J.lost_sales + 1
        if c>0 and J.stock>0:
            J.sale()

        J.storage_days = J.storage_days + J.stock
        if  order_wait == np.inf and J.stock==0:
            order_wait = 5
        if order_wait == 0:
            J.add_stock()
            order_wait = np.inf
        if order_wait>0:
            order_wait = order_wait - 1
    return J.result()
```

[6]:
```
J = JFTE(2*52*7)

J1 = stand_order(J,dow=6)
J2 = order_on_demand(J)
```

[7]:
```
print(J1.report())
```

```
weeks:        104.0
sales:        98
lost sales:   4
storage_days: 1865  (effective)
```

```
max stock:    7
```

[8]:
```
print(J2.report())
```

```
weeks:         104.0
sales:         59
lost sales:    43
storage_days: 374  (effective)
max stock:    1
```

[9]:
```
import pandas as pd

JL = list(map(JFTE,10*[2*52*7]))

def report_trials(JL):
    JS = list(map(stand_order,JL))
    JD = list(map(order_on_demand,JL))
    rdict = {"sales        - standing":list(map(lambda x:x.sales,     JS)),
             "lost sales   - standing":list(map(lambda x:x.lost_sales,JS)),
             "storage_days- standing":list(map(lambda x:x.storage_days,JS)),
             "max stock    - standing":list(map(lambda x:x.max_stock,JS)),
             "sales        - demand":  list(map(lambda x:x.sales,     JD)),
             "lost sales   - demand":  list(map(lambda x:x.lost_sales,JD)),
             "storage_days- demand":  list(map(lambda x:x.storage_days,JD)),
             "max stock    - demand":list(map(lambda x:x.max_stock,JD))
              }
    return pd.DataFrame(rdict)
```

[10]:
```
report_trials(JL)
```

[10]:
```
   sales        - standing  lost sales   - standing  storage_days- standing  \
0                       96                        0                    3541
1                      104                        9                    1600
2                      105                        8                    3262
3                      104                       23                     769
4                       97                        6                    3744
5                      103                       10                    1277
6                       95                       12                    2195
7                       98                        5                    1949
8                       99                        4                    3109
9                      104                       14                    2121

   max stock    - standing  sales        - demand  lost sales   - demand  \
0                       10                     60                     36
1                        7                     64                     49
2                        9                     66                     47
3                        4                     66                     61
4                       11                     58                     45
```

```
5                      6                    65                    48
6                     10                    63                    44
7                      7                    60                    43
8                      9                    64                    39
9                     11                    69                    49

   storage_days- demand  max stock   - demand
0                   368                     1
1                   345                     1
2                   337                     1
3                   332                     1
4                   380                     1
5                   343                     1
6                   352                     1
7                   370                     1
8                   344                     1
9                   314                     1
```