Simulate_a_Simple_Supply_Chain

March 23, 2019

1 Simulate a Simple Supply Chain

2 Yu-Ting Shen

2.1 Problem description

There are ten store outlets that get replenishments from a single warehouse. Initially, there are 100 units of product at each of the outlets. Inventory changes due to the following reasons only: Sales, expirations, damages, replenishment.

- Sales Demand at each outlet follow Normal distribution: N(70, 6^2).
- Expirations follow the Poisson distribution: P(3)
- Damages follow the Poisson distribution: P(2)
- At every time step, 100 items are dispatched from the warehouse to each of the outlets. The lead time for those items to arrive at any of the outlet follows the Poisson distribution: P(3 timesteps).

Your task is to simulate sales, damages, expirations, replenishments and inventory at each outlet and keep track of these quantities. Also, calculate the quantity Lost Sales (LS) for each timestep, where;

LS = unfulfilled Sales due to stockouts.

Run the simulation for 50 timesteps (t=1, 2...50). At the end of the simulation, output all the tracked quantities in tabular format for one outlet of your choice.

We are looking for good coding practices and overall approach to the solving the problem. Use Python.

In [1]:
```
# from graphviz import Digraph

# g = Digraph('G', filename='supply_chain.gv')

# g.attr(size='10,10')
# g.attr('node', shape='box')
```

```
# g.node('warehouse')

# for i in range(1, 11):
#     g.edge('warehouse', 'outlet_' + str(i) + '\n 100', label='P(3t)')

# g.view() # produce file
# g # render in jupyter notebook.

# When uncomment this part, the jupyter notebook cannot output to pdf format.
```

2.1.1 Distributions

In [2]:
```
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import norm, poisson

# Sales Demand
mu = 70
sigma = 6
x = np.linspace(mu - 3 * sigma, mu + 3 * sigma, 100)

# Expirations
rv_e = poisson(3)
arr_e = []
for num_e in range(0, 20):
    arr_e.append(rv_e.pmf(num_e))

# Damages
rv_d = poisson(2)
arr_d = []
for num_d in range(0, 20):
    arr_d.append(rv_d.pmf(num_d))

# Making plots
fig, ax = plt.subplots(1, 3, figsize=(20, 5))

ax[0].grid(True)
ax[0].plot(x, norm.pdf(x, mu, sigma))
ax[0].set_title('Sales Demand')
ax[0].set_xlabel('Number of products (sold)')
ax[0].set_ylabel('Probability')

ax[1].grid(True)
ax[1].plot(arr_e, linewidth=2.0)
ax[1].set_xticks([i for i in range(0, 20, 2)])
ax[1].set_title('Expiration')
ax[1].set_xlabel('Number of products (expired)')
ax[1].set_ylabel('Probability')
```

```
ax[2].grid(True)
ax[2].plot(arr_d, linewidth=2.0)
ax[2].set_xticks([i for i in range(0, 20, 2)])
ax[2].set_title('Damage')
ax[2].set_xlabel('Number of products (Damaged)')
ax[2].set_ylabel('Probability')

plt.show()
```

```
<Figure size 2000x500 with 3 Axes>
```

2.1.2 Create random number generators

- The random number generators follow

1. normal distribution: $N(70, 6^2)$
2. poisson distribution: $P(3)$ for expirations and $P(2)$ for damages

```
In [3]: def num_sales_demand():
            num = np.random.normal(70, 6, 1) # return a numpy.ndarray
            num = np.asscalar(num) # convert numpy.ndarray to scalar
            return int(num) # number of product must be integer

        def num_expiration():
            num = np.random.poisson(3, 1)
            num = np.asscalar(num)
            return int(num)

        def num_damage():
            num = np.random.poisson(2, 1)
            num = np.asscalar(num)
            return int(num)
```

```
In [4]: # test
        # np.random.seed(42) # set random seed for testing
        # print(num_sales_demand())
        # print(num_expiration())
        # print(num_damage())
```

2.1.3 Given:

- At every time step, 100 items are dispatched from the warehouse to each of the outlets.
- The lead time for those items to arrive at any of the outlet follows the Poisson distribution: P(3 timesteps).

So the products take **time** to ship and arrive store at **t + time**.

- t: leave warehouse

- time: arrive store

In [5]:
```
def arrived(t):
    time = np.random.poisson(3, 1)
    time = np.asscalar(time)
    return t, t + time # return the starting and arriving time
```

Definitions:

- $N_s(t)$: number of sold products at time t
- $N_e(t)$: number of expired products at time t
- $N_d(t)$: number of damaged products at time t
- $N_w(t)$: number of replenished items at time t

In [6]:
```
# Initial condition (i.e. t=0)
Ns = [0]
Ne = [0]
Nd = [0]
time = [(0, 0)] # (starting time, arriving time)

for i in range(1, 51):
    Ns.append(num_sales_demand())
    Ne.append(num_expiration())
    Nd.append(num_damage())
    time.append(arrived(i))
```

In [7]:
```
# show results
# print(Ns)
# print(Ne)
# print(Nd)
# print(time)

# print(len(Ns))
# print(len(Ne))
# print(len(Nd))
# print(len(time))
```

In [8]:
```
Nw = [0] * 51

for i in range(1, 51):
    arrive_timestep = time[i][1]
    if arrive_timestep < 51:
        Nw[arrive_timestep] += 100 # 100 items arrived
```

In [9]:
```
# show results
# time[i][0]: timestemp for starting to ship
# time[i][i]: arrived timestemp

# for i in range(1, 51):
#     print(i, 'starting time=', time[i][0], 'arriving time=', time[i][1], 'replenishm
```

2.1.4 The number of products in stock at timestep t is:

- $N(t) = N(t-1) - N_s(t) - N_e(t) - N_d(t) + N_w(t)$

```
In [10]: def num_in_stock(N_t_minus_1, Ns_t, Ne_t, Nd_t, Nw_t):
             num = N_t_minus_1 - Ns_t - Ne_t - Nd_t + Nw_t

             if num < 0:
                 return 0
             return num
```

2.1.5 Lost Sales:

- Lost Sales (LS) = unfulfilled Sales due to stockouts.
- LS = sales demand - number of products in stock can be sold
- number of products in stock can be sold = $N(t-1) + N_w(t) - N_e(t) - N_d(t)$
  - If this value is negative, then this means no product can be sold. $\rightarrow$ Set to zero.

```
In [11]: # Initial conditions
         N_stock = [100]
         LS = [0]

         for i in range(1, 51):
             N = num_in_stock(N_stock[-1], Ns[i], Ne[i], Nd[i], Nw[i])

             # Calculate lost sales
             product_can_be_sold = max(0, N_stock[-1] + Nw[i] - Ne[i] - Nd[i])
             if Ns[i] - product_can_be_sold > 0:
                 ls = Ns[i] - product_can_be_sold
             else:
                 ls = 0

             N_stock.append(N)
             LS.append(ls)
```

```
In [12]: # show
         # print(N_stock)
         # print(len(N_stock))
         # print(LS)
         # print(len(LS))
```

2.1.6 Table

```
In [13]: # show in table

         import pandas as pd

         df = pd.DataFrame({'Sales_Demand': Ns,
                            'Expirations': Ne,
```

```
                       'Damages': Nd,
                       'Replenished': Nw,
                       'In_stock': N_stock,
                       'Lost_Sales': LS})
df.index.name = 'Timestep'
df
```

Out[13]:

```
          Sales_Demand  Expirations  Damages  Replenished  In_stock  \
Timestep
0                    0            0        0            0       100
1                   71            4        3            0        22
2                   65            3        0            0         0
3                   73            0        2            0         0
4                   63            2        1          200       134
5                   65            4        3          100       162
6                   75            2        2            0        83
7                   63            7        1          200       212
8                   70            1        3          100       238
9                   72            4        3            0       159
10                  78            3        4            0        74
11                  60            1        0          200       213
12                  67            3        1          100       242
13                  76            1        4          100       261
14                  72            1        3          100       285
15                  76            2        0          100       307
16                  67            4        2          200       434
17                  69            2        1            0       362
18                  67            6        3          100       386
19                  66            6        3          100       411
20                  75            3        3          200       530
21                  67            2        2          100       559
22                  69            4        3            0       483
23                  69            3        1          100       510
24                  60            3        2          100       545
25                  73            2        0            0       470
26                  69            2        5            0       394
27                  78            2        1          200       513
28                  66            5        3            0       439
29                  76            4        0          200       559
30                  69            3        2          100       585
31                  64            2        3            0       516
32                  68            1        3          200       644
33                  74            2        2          200       766
34                  84            2        2          200       878
35                  66            3        3            0       806
36                  76            5        3            0       722
37                  47            3        3          300       969
38                  77            2        1            0       889
```

39	80	4	1	100	904
40	68	3	5	100	928
41	70	3	1	0	854
42	69	2	1	200	982
43	78	4	3	100	997
44	70	3	3	200	1121
45	68	4	1	0	1048
46	68	5	2	0	973
47	71	2	3	200	1097
48	76	2	1	100	1118
49	74	3	3	100	1138
50	61	6	1	100	1170

	Lost_Sales
Timestep	
0	0
1	0
2	46
3	73
4	0
5	0
6	0
7	0
8	0
9	0
10	0
11	0
12	0
13	0
14	0
15	0
16	0
17	0
18	0
19	0
20	0
21	0
22	0
23	0
24	0
25	0
26	0
27	0
28	0
29	0
30	0
31	0
32	0

```
33                    0
34                    0
35                    0
36                    0
37                    0
38                    0
39                    0
40                    0
41                    0
42                    0
43                    0
44                    0
45                    0
46                    0
47                    0
48                    0
49                    0
50                    0
```

```
In [ ]:
```