最佳化決策模式設計與應用

DESIGN AND APPLICATIONS OF OPTIMAL DECISION MAKING MODELS

Project 1

李艾霓 / H24076095 / 國立成功大學 / 統計學系111級

2019/06/27

※pdf基本上內容和ipynb檔是相同的，只是刪除了部分程式碼，更便於閱讀。

1. 問題描述及假設

The Problem 問題描述

2

- 阿中是販賣活螃蟹店家的老闆，每天固定要去批貨來賣。他希望用數學模型來幫助自己***決定每天所需要訂購的螃蟹品種數量***以便可以**獲得最大的收益**及盡可能滿足客戶的需求。訂購太多如果沒有賣出去會造成每日庫存成本上升，以及承擔螃蟹死亡的成本損失風險；相對地，訂購太少會無法滿足客戶需求，造成再買率降低。

NCKU IIM

Project Assumptions 問題假設-1

3

- 府城螃蟹專賣四種品項，分別為沙母、一般紅蟳、頂級沙幼母、二級沙幼母。
- 阿中老闆每日清晨在開店前要至上游盤商批貨，根據一周七天的訂單需求及庫存螃蟹來決定今日的每一種品項所需訂購量。
- 因為螃蟹為易腐性商品，預期壽命為訂購當日開始後三天內，每一天會有5%的螃蟹量損失(無條件進位，取整數)，第三天結束若沒有賣完則不能繼續當作庫存。
- 假設一天只採購一次，採購時只有10個籃子可以裝貨，一個籃子可裝之螃蟹數量依品種不同而有差異。10個籃子可以不用全部使用，但是**如果有使用的話就需要裝滿**。
- 螃蟹販售價格及成本在一周內不會變動，皆為已知常數。

NCKU IIM

須注意的是，在這裡，我們更進一步定義「耗損率」發生在每日閉店後要計算剩餘螃蟹時發現，故在庫存之前即先淘汰掉，並不會計入庫存，也不影響當日販售(當日購買的螃蟹應是經過挑選，理應不會馬上耗損，故閉店之後才計算)。

2. 參數設定

2.1 載入預設參數

這是一個假想的7天需求情況:

	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7
品種							
沙母	50	82	10	17	34	13	71
一般紅蟳	70	46	24	46	45	47	36
頂級沙幼母	16	62	34	44	13	45	32
二級沙幼母	20	53	65	100	81	21	51

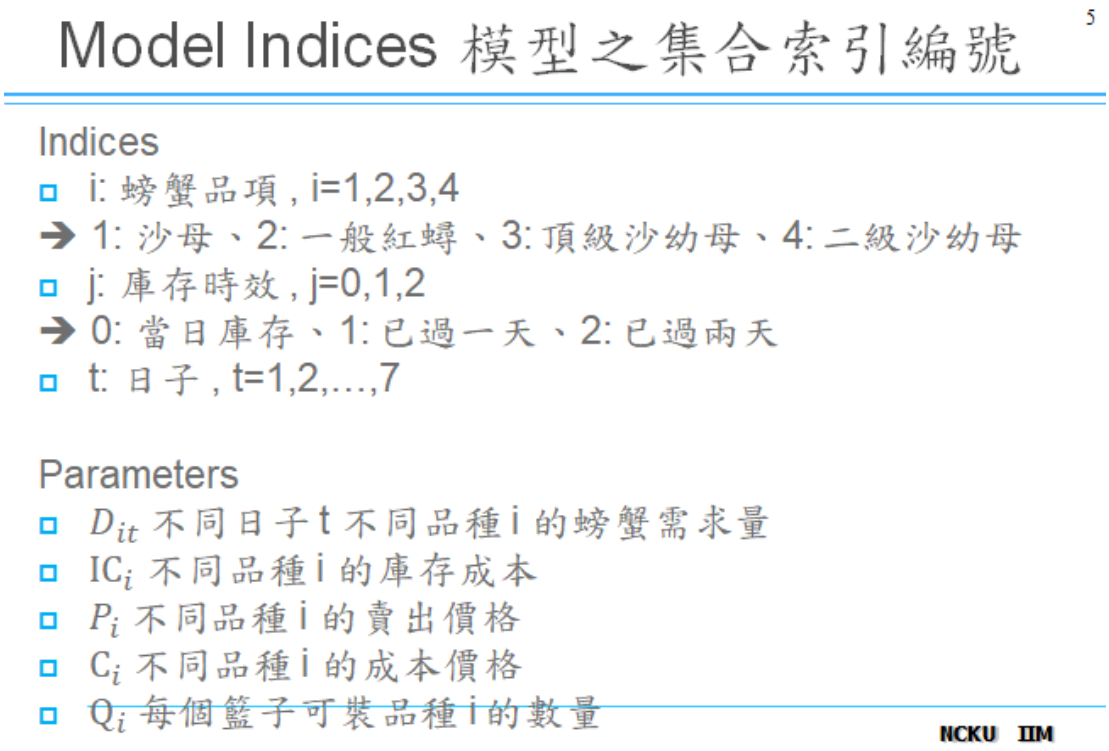
假想的初始庫存量:

	1天庫存	2天庫存
品種		
沙母	3	0
一般紅蟳	20	17
頂級沙幼母	13	4
二級沙幼母	21	0

而這些則是參考老師ppt所使用的參數:

	販售價格	進貨成本	庫存成本/天	每個籃子可裝	耗損率
品種					
沙母	900	700	10	6	0.05
一般紅蟳	380	300	10	20	0.03
頂級沙幼母	650	520	10	13	0.04
二級沙幼母	550	480	10	13	0.06

2.2 參數說明



此外，定義 **Ri 每個品種 i 的耗損率**

2.3 寫入參數

```
I = range(1,5)
J = range(3)
T = range(1,8)
```

```
D = {}
for i in I:
    for t in T:
        D[i,t] = demand.iat[i-1,t-1]
```

```
IC = {}
P = {}
C = {}
Q = {}
R = {}
for i in I:
    IC[i] = param.iat[i-1,2]
    P[i] = param.iat[i-1,0]
    C[i] = param.iat[i-1,1]
    Q[i] = param.iat[i-1,3]
    R[i] = param.iat[i-1,4]
```

3. 模型

3.1 建立模型

```
model = Model('Crab')
```

3.2 設定變數

變數 **x [i, t]** 表示第 i 種螃蟹在第 t 天要購買的籃數。

變數 **sell [i, t, j]** 則表示第 i 種螃蟹在第 t 日販售出第 j 天庫存的數量。

```
x = {}
sell = {}
for i in I:
    for t in T:
        x[i,t] = model.addVar(ub=10, vtype='I', name='x%d%d'%(i,t))
        for j in J:
            sell[i,t,j] = model.addVar(vtype='I', name='sell%d%d%d'%(i,t,j))
```

由上面的變數經過轉換以方便後續運算:

buy [i, t] 表示第 i 種螃蟹在第 t 天實際購買隻數 (籃數 * 籃子可裝)

```
buy = {}
for i in I:
    for t in T:
        buy[i,t] = x[i,t]*Q[i]
```

begin [i, t, j] 表示第 i 種螃蟹在第 t 天批貨結束之後、開店之前計算出第 j 天存貨可販賣的隻數 (j=0表示當日清晨購買)

end [i, t, j] 表示第 i 種螃蟹在第 t 天閉店之後計算出第 j 天未賣完需庫存的隻數 (已扣除耗損隻數，未四捨五入，故可能不是整數，而是作為一種「期望值」)

```
begin = {}
end = {}
for i in I:
    for t in T:
        for j in range(2,-1,-1):
            if t == 1 and j in [1,2]:  # 第一天的1天及2天庫存已給定
                begin[i,1,j] = inventory.iat[i-1,j-1]

            elif j != 0:  # j==2 or j==3
                begin[i,t,j] = end[i,t-1,j-1]
            else:  # j==0
                begin[i,t,j] = buy[i,t]

            end[i,t,j] = (begin[i,t,j]-sell[i,t,j])*(1-R[i])
```

```
model.update()
```

3.3 設定限制式

這裡將所有限制式分為三個部分：

```
# 一天不可買超過十籃
for t in T:
    model.addConstr(quicksum(x[i,t] for i in I) <= 10)
```

```
# 銷售量不可能超過當日需求量
for (i,t) in D.keys():
    model.addConstr(quicksum(sell[i,t,j] for j in J) - D[i,t] <= 0)
```

```
# 銷售量不可超過可販售的數量
for i in I:
    for t in T:
        for j in J:
            model.addConstr(sell[i,t,j] - begin[i,t,j] <= 0)
```

3.4 設定目標式

我們的目標式是使獲利最大，為了易讀性，我們分別算出銷售額以及成本，再以 **profit = earn-cost** 表達。

```
earn = quicksum(sell[i,t,j]*P[i] for (i,t,j) in sell)
```

```
cost = quicksum(buy[i,t]*C[i] for (i,t) in D.keys())  # 買進成本(當日買進個數 * 單位買進成本)
cost += quicksum(end[i,t,j]*IC[i] for (i,t,j) in sell)  # 庫存成本
```

```
profit = earn-cost
model.setObjective(profit, GRB.MAXIMIZE)
```

3.5 最佳化

```
model.optimize()
```

```
Optimize a model with 119 rows, 112 columns and 336 nonzeros
Variable types: 0 continuous, 112 integer (0 binary)
Coefficient statistics:
  Matrix range     [9e-01, 2e+01]
  Objective range  [4e+02, 7e+03]
  Bounds range     [1e+01, 1e+01]
  RHS range        [3e+00, 1e+02]
Found heuristic solution: objective 38132.164000
Presolve removed 52 rows and 2 columns
Presolve time: 0.00s
Presolved: 67 rows, 110 columns, 218 nonzeros
Variable types: 0 continuous, 110 integer (0 binary)

Root relaxation: objective 1.346306e+05, 80 iterations, 0.00 seconds

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0 134630.572    0   33 38132.1640 134630.572   253%     -    0s
```

```
H    0     0                    120728.13476 134630.572  11.5%     -    0s
     0     0 132560.557    0   41 120728.135 132560.557  9.80%     -    0s
H    0     0                    126070.36192 132560.557  5.15%     -    0s
     0     0 132063.666    0   62 126070.362 132063.666  4.75%     -    0s
H    0     0                    126515.19384 132063.666  4.39%     -    0s
     0     0 131896.486    0   60 126515.194 131896.486  4.25%     -    0s
     0     0 131819.161    0   59 126515.194 131819.161  4.19%     -    0s
     0     0 131816.074    0   61 126515.194 131816.074  4.19%     -    0s
     0     0 131749.411    0   63 126515.194 131749.411  4.14%     -    0s
     0     0 131733.497    0   67 126515.194 131733.497  4.12%     -    0s
     0     0 131722.844    0   68 126515.194 131722.844  4.12%     -    0s
     0     0 131721.379    0   68 126515.194 131721.379  4.12%     -    0s
     0     0 131663.348    0   73 126515.194 131663.348  4.07%     -    0s
     0     0 131516.456    0   71 126515.194 131516.456  3.95%     -    0s
     0     0 131497.050    0   73 126515.194 131497.050  3.94%     -    0s
     0     0 131495.665    0   76 126515.194 131495.665  3.94%     -    0s
     0     0 131457.317    0   60 126515.194 131457.317  3.91%     -    0s
     0     0 131438.539    0   58 126515.194 131438.539  3.89%     -    0s
     0     0 131421.519    0   64 126515.194 131421.519  3.88%     -    0s
     0     0 131421.039    0   57 126515.194 131421.039  3.88%     -    0s
     0     0 131418.966    0   57 126515.194 131418.966  3.88%     -    0s
     0     0 131418.966    0   57 126515.194 131418.966  3.88%     -    0s
H    0     0                    128179.76731 131418.966  2.53%     -    0s
     0     2 131418.966    0   57 128179.767 131418.966  2.53%     -    0s
H  197   119                    128253.38037 131361.510  2.42%   3.3    0s
H  353   223                    128403.13637 131359.985  2.30%   3.2    0s
H  468   269                    128420.83109 131333.826  2.27%   3.3    0s
H  851   468                    128877.54814 130971.223  1.62%   3.6    0s
H  855   450                    129057.92984 130971.223  1.48%   3.6    0s
H  931   463                    129458.20320 130971.223  1.17%   3.7    0s
H21198 10452                    130047.64349 130430.061  0.29%   2.8    3s

Cutting planes:
  Gomory: 24
  MIR: 62
  StrongCG: 7
  Zero half: 1

Explored 29752 nodes (95353 simplex iterations) in 4.98 seconds
Thread count was 4 (of 4 available processors)

Solution count 10: 130048 129458 129058 ... 126070

Optimal solution found (tolerance 1.00e-04)
Best objective 1.300476434900e+05, best bound 1.300476434900e+05, gap 0.0000%
```

4. 檢視結果

4.1 最佳值

```
Optimal value: 130047.64348999996
```

4.2 每日購買籃數

	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7
品種							
沙母	7	5	1	2	5	2	8
一般紅蟳	3	1	1	2	2	3	1
頂級沙幼母	0	4	3	3	1	5	1
二級沙幼母	0	0	5	3	2	0	0

4.3 供給及需求的關係

recall一開始的需求表：

	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7
品種							
沙母	50	82	10	17	34	13	71
一般紅蟳	70	46	24	46	45	47	36
頂級沙幼母	16	62	34	44	13	45	32
二級沙幼母	20	53	65	100	81	21	51

再對照實際賣出量：

	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7
品種							
沙母	45	30	6	12	30	12	48
一般紅蟳	53	46	20	40	40	47	32
頂級沙幼母	13	52	34	43	13	45	32
二級沙幼母	20	0	65	39	26	0	0

將每日的需求量減去賣出量：

	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7
品種							
沙母	5	52	4	5	4	1	23
一般紅蟳	17	0	4	6	5	0	4
頂級沙幼母	3	10	0	1	0	0	0
二級沙幼母	0	53	0	61	55	21	51

由上表可以發現，若盡量滿足頂級沙幼母的需求則可使獲利最大化。而一般紅蟳每籃獲利與頂級沙幼母相近，因此紅蟳的賣出量也與需求量很接近。 另外也能發現，為了使獲利最大化，會選擇讓沙母與二級沙幼母供不應求。

但在正常情況下，若有如此大量的需求，店家理應會增加批貨的量來盡量滿足需求並增加自己的獲利，故可知我們所預設的需求量應是大於實際的，可以調整我們預設的參數再測試看看。另，即使在大量的需求下，模型也會盡量減少庫存(且幾乎沒有1天和2天庫存)(見下方)，對照實際的庫存量也可以發現我們預設的初始庫存應該也是過高的。

4.4 庫存

```
品種  天  0天庫存  1天庫存  2天庫存
 1    1    0.00    0.00    0.00
 1    2    0.00    0.00    0.00
 1    3    0.00    0.00    0.00
 1    4    0.00    0.00    0.00
 1    5    0.00    0.00    0.00
 1    6    0.00    0.00    0.00
 1    7    0.00    0.00    0.00

品種  天  0天庫存  1天庫存  2天庫存
 2    1   26.19    0.00    0.00
 2    2    0.00    0.18    0.00
 2    3    0.00    0.00    0.18
 2    4    0.00    0.00    0.00
 2    5    0.00    0.00    0.00
 2    6   12.61    0.00    0.00
 2    7    0.00    0.59    0.00

品種  天  0天庫存  1天庫存  2天庫存
 3    1    0.00    0.00    0.96
 3    2    0.00    0.00    0.00
 3    3    4.80    0.00    0.00
 3    4    0.00    0.77    0.00
 3    5    0.00    0.00    0.74
 3    6   19.20    0.00    0.00
 3    7    0.00    0.19    0.00

品種  天  0天庫存  1天庫存  2天庫存
 4    1    0.00    0.94    0.00
 4    2    0.00    0.00    0.88
 4    3    0.00    0.00    0.00
 4    4    0.00    0.00    0.00
 4    5    0.00    0.00    0.00
 4    6    0.00    0.00    0.00
```

```
     4      7     0.00      0.00      0.00
```

5. 更改參數並重新測試

5.1 更改後參數

我們大致上將需求減少，看看是否能更符合真實情況。

	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7
品種							
沙母	10	37	43	18	23	14	12
一般紅蟳	34	27	20	21	62	23	9
頂級沙幼母	8	12	10	12	34	17	17
二級沙幼母	6	34	23	51	12	13	11

庫存也做了調整：

	1天庫存	2天庫存
品種		
沙母	2	0
一般紅蟳	17	0
頂級沙幼母	10	0
二級沙幼母	1	0

下面則只更改了二級沙幼母的耗損率(0.06 -> 0.02)。

	販售價格	進貨成本	庫存成本/天	每個籃子可裝	耗損率
品種					
沙母	900	700	10	6	0.05
一般紅蟳	380	300	10	20	0.03
頂級沙幼母	650	520	10	13	0.04
二級沙幼母	550	480	10	13	0.02

5.2 放入模型

```
D = {}
```

```
for i in I:
    for t in T:
        D[i,t] = demand.iat[i-1,t-1]
IC = {}
P = {}
C = {}
Q = {}
R = {}
for i in I:
    IC[i] = param.iat[i-1,2]
    P[i] = param.iat[i-1,0]
    C[i] = param.iat[i-1,1]
    Q[i] = param.iat[i-1,3]
    R[i] = param.iat[i-1,4]
model = Model('Crab')
x = {}
sell = {}
for i in I:
    for t in T:
        x[i,t] = model.addVar(ub=10, vtype='I', name='x%d%d'%(i,t))
        for j in J:
            sell[i,t,j] = model.addVar(vtype='I', name='sell%d%d%d'%(i,t,j))
buy = {}
for i in I:
    for t in T:
        buy[i,t] = x[i,t]*Q[i]
begin = {}
end = {}
for i in I:
    for t in T:
        for j in range(2,-1,-1):
            if t == 1 and j in [1,2]:  # 第一天的1天及2天庫存已給定
                begin[i,1,j] = inventory.iat[i-1,j-1]

            elif j != 0:  # j==2 or j==3
                begin[i,t,j] = end[i,t-1,j-1]
            else:  # j==0
                begin[i,t,j] = buy[i,t]

            end[i,t,j] = (begin[i,t,j]-sell[i,t,j])*(1-R[i])
model.update()
# 一天不可買超過十籃
for t in T:
    model.addConstr(quicksum(x[i,t] for i in I) <= 10)
# 銷售量不可能超過當日需求量
for (i,t) in D.keys():
    model.addConstr(quicksum(sell[i,t,j] for j in J) - D[i,t] <= 0)
# 銷售量不可超過可販售的數量
for i in I:
    for t in T:
        for j in J:
            model.addConstr(sell[i,t,j] - begin[i,t,j] <= 0)
earn = quicksum(sell[i,t,j]*P[i] for (i,t,j) in sell)
cost = quicksum(buy[i,t]*C[i] for (i,t) in D.keys())  # 買進成本(當日買進個數 * 單位
買進成本)
cost += quicksum(end[i,t,j]*IC[i] for (i,t,j) in sell)  # 庫存成本
profit = earn-cost
model.setObjective(profit, GRB.MAXIMIZE)
```

```
model.optimize()
```

```
Academic license - for non-commercial use only
Optimize a model with 119 rows, 112 columns and 336 nonzeros
Variable types: 0 continuous, 112 integer (0 binary)
Coefficient statistics:
  Matrix range     [9e-01, 2e+01]
  Objective range  [4e+02, 7e+03]
  Bounds range     [1e+01, 1e+01]
  RHS range        [1e+00, 6e+01]
Found heuristic solution: objective 14631.968000
Presolve removed 48 rows and 5 columns
Presolve time: 0.00s
Presolved: 71 rows, 107 columns, 221 nonzeros
Variable types: 0 continuous, 107 integer (4 binary)

Root relaxation: objective 8.224825e+04, 73 iterations, 0.00 seconds

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0 82248.2528    0   35 14631.9680 82248.2528   462%     -    0s
H    0     0                     66012.644420 82248.2528  24.6%     -    0s
     0     0 81018.7114    0   48 66012.6444 81018.7114  22.7%     -    0s
     0     0 80057.3636    0   45 66012.6444 80057.3636  21.3%     -    0s
                                      ......
                                      ......
                                      ......
                                      ......
 35845304 185147 74900.2669  109   17 74886.0302 74900.2669  0.02%   2.8 3490s
 35883988 169821 74899.2129   94   19 74886.0302 74899.2129  0.02%   2.8 3495s
 35929176 151856 74897.9572   91   17 74886.0302 74897.9572  0.02%   2.8 3500s
 35975114 131916     cutoff   74      74886.0302 74896.5126  0.01%   2.8 3505s
 36014966 106612 74894.7528   68   22 74886.0302 74894.7528  0.01%   2.8 3510s

Cutting planes:
  Gomory: 30
  MIR: 202
  StrongCG: 23
  Inf proof: 1
  Zero half: 2

Explored 36041663 nodes (99098364 simplex iterations) in 3513.29 seconds
Thread count was 4 (of 4 available processors)

Solution count 10: 74886 74886 74886 ... 74658

Optimal solution found (tolerance 1.00e-04)
Best objective 7.488602822000e+04, best bound 7.489351263494e+04, gap 0.0100%
```

5.3 結果分析

相較於原本的參數，這次模型花了較長的時間才求得最佳解，且最佳值相較於原本確實隨著需求的減少而降低了。

```
Optimal value: 74886.02821999993
```

每日購買籃數：

	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7
品種							
沙母	1	6	7	4	3	2	2
一般紅蟳	1	1	1	1	3	1	0
頂級沙幼母	0	2	0	1	3	1	1
二級沙幼母	3	1	2	3	1	1	0

需求降低之後，不再是每天都買到十籃那麼多，而是隨著每日的需求而有所變動，這應該較符合這個模型的初衷：滿足需求的同時避免庫存太多而負擔螃蟹死亡的風險。

需求 - 供給：

	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7
品種							
沙母	2	1	1	0	0	2	0
一般紅蟳	0	7	0	1	2	3	9
頂級沙幼母	0	1	0	4	0	0	4
二級沙幼母	0	0	12	0	2	0	11

沒有每天都買到10籃，但這次幾乎所有的需求都被盡量滿足了！

庫存：

```
品種  天  0天庫存  1天庫存  2天庫存
 1    1    0.00    0.00    0.00
 1    2    0.00    0.00    0.00
 1    3    0.00    0.00    0.00
 1    4    5.70    0.00    0.00
 1    5    0.00    0.66    0.00
 1    6    0.00    0.00    0.63
 1    7    0.00    0.00    0.00

品種  天  0天庫存  1天庫存  2天庫存
 2    1    0.00    2.91    0.00
 2    2    0.00    0.00    0.88
 2    3    0.00    0.00    0.00
 2    4    0.00    0.00    0.00
 2    5    0.00    0.00    0.00
 2    6    0.00    0.00    0.00
 2    7    0.00    0.00    0.00

品種  天  0天庫存  1天庫存  2天庫存
 3    1    0.00    1.92    0.00
 3    2   14.40    0.00    0.88
 3    3    0.00    4.22    0.00
```

```
     3      4     4.80     0.00     0.22
     3      5     8.64     0.77     0.00
     3      6     0.00     4.45     0.74
     3      7     0.00     0.00     0.44

   品種     天   0天庫存   1天庫存   2天庫存
     4      1    33.32     0.00     0.00
     4      2     0.00    12.07     0.00
     4      3    14.70     0.00     0.07
     4      4     0.00     2.65     0.00
     4      5     2.94     0.00     0.63
     4      6     0.00     2.88     0.00
     4      7     0.00     0.00     0.86
```

可以看到庫存是被計算得很剛好的，正好在兩天之內將可以販售的螃蟹都販賣完畢，幾乎可以說是沒有任何的浪費。

6. Discussion

由於該最佳化求解是在每日各品種需求量已知下進行，但確切需求量往往不易獲得，需要利用過往經驗、顧客消費水平、競爭店家等來估算。因此應用實際狀況時，可能需求量的求得也會是個課題。另外，誠如上述，為了使獲利最大化會選擇讓一些品種供不應求，但如此一來會降低顧客回頭率，而回頭率會影響之後的需求量，因此供不應求所影響回頭率的情況也是之後需要考慮的問題。