Management Science - Individual Assignment 2

Thanh Dat Nguyen - 532618tn

20 December 2022

Question 1

Binary Program Formulation

In this section, we define the binary program. Our decision variables are binary variables s, which denote every possible swimmer-style combination:

$$\begin{aligned} s_j^i \in \{0,1\}, i \in I, j \in J \\ I = \{N, M, L, A, C\} \\ J = \{BA, BR, BU, FR\} \end{aligned} \tag{1}$$

In this problem, the objective is to minimize the time it takes to finish the relay:

$$min \sum_{i \in I} \sum_{j \in J} s_j^i t_j^i \tag{2}$$

Where t stands for the time it takes for a swimmer i to do a lap with a particular style j. The function is subject to the following constraints - each swimmer can only swim at most once and each style has to be swum once:

$$\begin{aligned} \sum_{i \in I} s_j^i = 1, \forall j \in J \\ \sum_{j \in J} s_j^i \leq 1, \forall i \in I \end{aligned} \tag{3}$$

Gurobi Implementation

[224]:
```
swimmers = ["N", "M", "L", "A", "C"]
styles = ["BA","BR","BU","FR"]
times = {("N","BA"): 40.1, ("N","BR"): 41.7, ("N","BU"): 41.1, ("N","FR"): 33.3,
         ("M","BA"): 39.6, ("M","BR"): 37.6, ("M","BU"): 31.9, ("M","FR"): 31.2,
         ("L","BA"): 35.3, ("L","BR"): 37.9, ("L","BU"): 36.2, ("L","FR"): 33.1,
         ("A","BA"): 30.6, ("A","BR"): 34.1, ("A","BU"): 33.3, ("A","FR"): 29.1,
         ("C","BA"): 36.9, ("C","BR"): 37.2, ("C","BU"): 29.0, ("C","FR"): 31.6}

model = gp.Model()
s = model.addVars(times, vtype = gp.GRB.BINARY, name = "relay",obj = times)
#the objective for minimization not defined because it's the default
#there have to be 4 swimmers, one for each style and the swimmers can't
repeat in a relay
model.addConstrs(gp.quicksum(s[i,j] for i in swimmers) == 1 for j in styles)
model.addConstrs(gp.quicksum(s[i,j] for j in styles) <= 1 for i in swimmers)
#optimizing the model
model.optimize()
model.printAttr("X")
```

```
Optimal solution found (tolerance 1.00e-04)
Best objective 1.287000000000e+02, best bound 1.287000000000e+02, gap 0.0000%
```

```
    Variable            X
-------------------------
 relay[M,FR]            1
 relay[L,BR]            1
 relay[A,BA]            1
 relay[C,BU]            1
```

Solving the problem in Gurobi tells us the following solution: have Carol do the butterfly stroke, Asma do the backstroke, Lina do the breaststroke and Mary do the freestyle. Under this solution, it would take the team 128.7 seconds to finish the relay.

Question 2

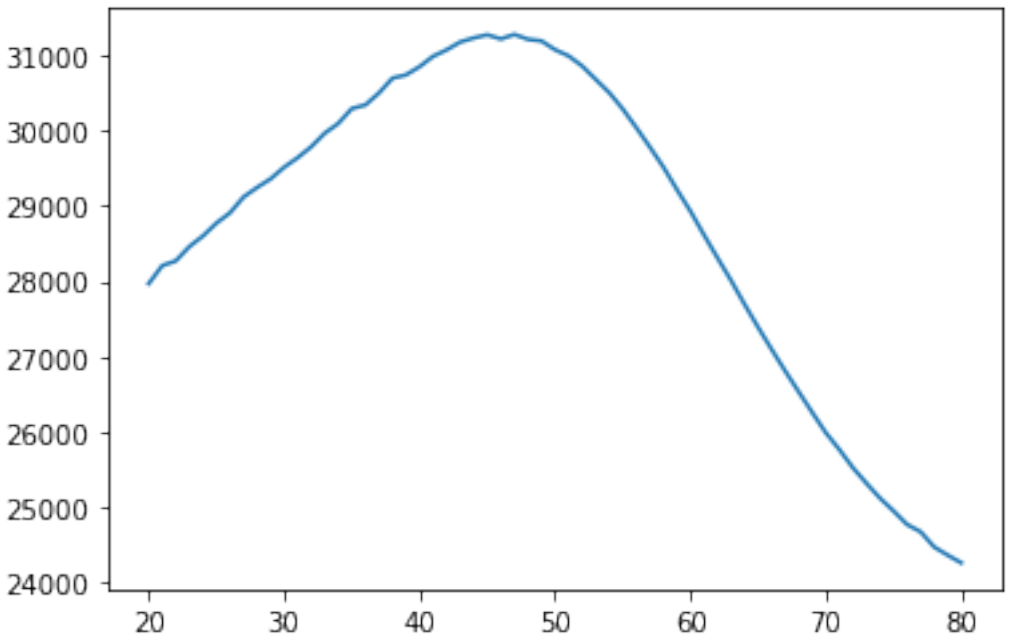
Part A

[107]:
```
seats = 100
price1 = 150
price2 = 500
alpha = range(20,81)
n = 10000
expRevenues2 = []

for limit in alpha:
    #generate customers period 1
    customers1 = np.random.poisson(lam = 75, size = n)
    #sales based on booking limit
    p1sales = np.minimum(customers1,limit)
    #capacity
    remCap = 100 - p1sales
    #generate customers period 2
    customers2 = np.random.poisson(lam = 50, size = n)
    #sales based on capacity
    p2sales = np.minimum(customers2, remCap)
    #average the revenue across the 10k sims
    exprevenue = np.average(150*p1sales + 500*p2sales)
    #store our revenues for each booking limit
    expRevenues2.append(exprevenue)

sn.lineplot(alpha, expRevenues2)
```

[107]:
```
<AxesSubplot:>
```



https://www.overleaf.com/project/639863c918fa5a11e53fa176

[119]:
```
limit_table = pd.DataFrame(
    {'Booking Limit': alpha,
     'Expected Revenue': expRevenues2
    })
limit_tablelimit_table["Expected Revenue"]==max(limit_table["Expected Revenue"])
```

[119]:
```
    Booking Limit  Expected Revenue
27             47          31268.89
```

[120]:
```
max(expRevenues2)
```

[120]:
```
31268.89
```

As we can see from the plot, in this particular simulation, the highest expected revenue comes from setting the booking limit at 47 for the first period. The optimal revenue here would be $31268.89.

Part B

[133]:
```
seats = 100
price1 = 150
price2 = 500
alpha = range(20,81)
#adding possible range of p
p = range(150,301)
n = 10000
expRevenues = []
```

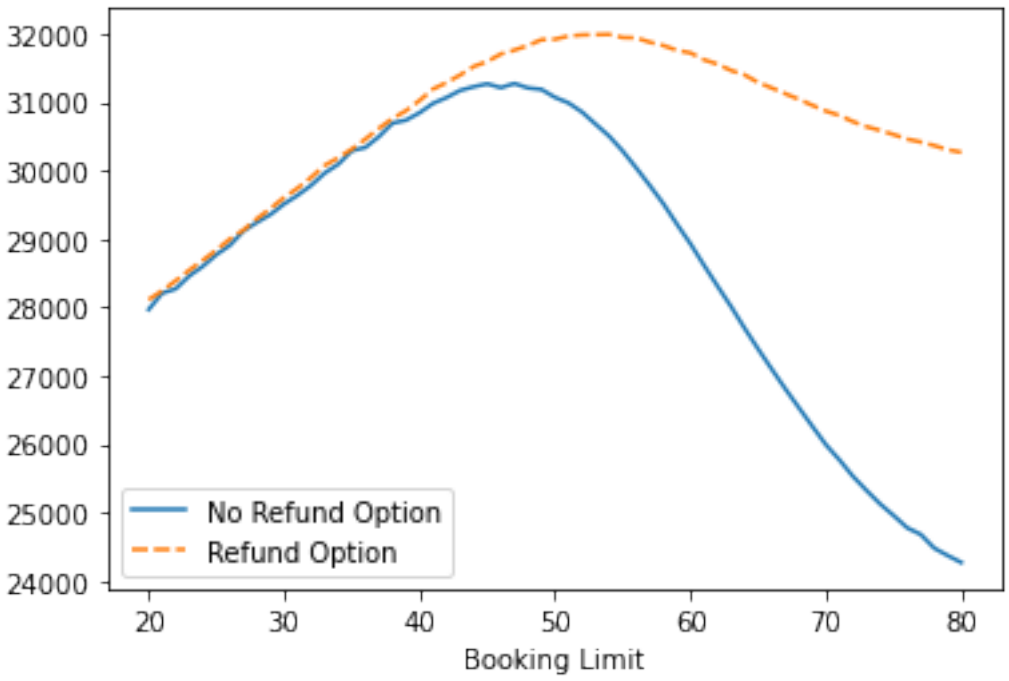
```
bestChoice = []


for limit in alpha:
    for refund in p:
        customers1 = np.random.poisson(lam = 75, size = n)
        p1sales = np.minimum(customers1,limit)
        #probability of refund
        probRefund = (refund-150)/200
        #number of refundable tickets
        refTickets = np.random.binomial(p1sales, probRefund)
        remCap = 100 - p1sales
        #remaining cap including refundables
        remCapRef = remCap + refTickets
        customers2 = np.random.poisson(lam = 50, size = n)
        #sales can now go up to including refunds
        p2sales = np.minimum(customers2, remCapRef)
        #tickets that are refunded - can't be negative
        refunded = np.maximum(p2sales-remCap, 0)
        exprevenue = np.average(150*p1sales + 500*p2sales - refund*refunded)
        expRevenues.append(exprevenue)
    #for each limit we automatically store maximized revenue (best p)
    bestoption = max(expRevenues)
    bestChoice.append(bestoption)
    #clear the expected revenues list (for each limit we want a fresh one)
    expRevenues = []

bestChoice
```

[137]:
```
limit_table2 = pd.DataFrame(
    {'Booking Limit': alpha,
     'No Refund Option': expRevenues2,
     'Refund Option': bestChoice
    })
dfAir= limit_table2.set_index(["Booking Limit"])
sn.lineplot(data = dfAir)
```

[137]:
```
<AxesSubplot:xlabel='Booking Limit'>
```



[138]:
```
dfAir.loc[dfAir["Refund Option"]==max(dfAir["Refund Option"])]
```

[138]:

Booking Limit	No Refund Option	Refund Option
54	30498.885	31986.945

[139]:
```
np.average(dfAir["Refund Option"]) - np.average(dfAir["No Refund Option"])
```

[139]:
```
1970.5955163934414
```

[140]:
```
max(dfAir["Refund Option"]) - max(dfAir["No Refund Option"])
```

[140]:
```
718.0550000000003
```

The optimal revenue (assuming we always choose the optimal refund level) is $31,986.945. Based on the optimal revenues for both schemes, the value of the callable products scheme is $718.06. The optimal booking limit with the callable products scheme is considerably higher at 54, compared to the 47 of the scheme without these products. As we can also see from the graph, the callable products scheme offsets the decrease in revenues caused by a higher booking limit, while keeping the same performance as the No Refund Option for lower booking limits. It is, therefore, highly recommended that the airline adopts this scheme.

Question 3

Part A

Stochastic Program Formulation

For this problem we formulate the decision variables PF (the amount sent from the production facility to a warehouse) and S (the amount shipped from a warehouse to a distribution center per scenario).

$$\begin{gathered} PF_w, w \in W \\ S^d_{s,w}, s \in S, w \in W, d \in DC \\ W = \{A, B\} \\ DC = \{1, 2, 3, 4\} \\ S = \{a, b, c\} \end{gathered} \tag{4}$$

We are maximizing the profit, which is given by the following objective value (summing costs and revenues):

$$max - 10 \sum_{w \in W} PF_w - \sum_{w \in W} p_w PF_w + \sum_{w \in W} \sum_{d \in DC} \sum_{s \in S} Pr_s (20 - c^d_w) S^d_{s,w} \tag{5}$$

Where p_w stands for the transportation cost from the production facility to the warehouse, Pr_s stands for the probability of a given scenario s and c^d_w stands for the transportation cost from a warehouse to a distribution center. This is subject to second stage constraints:

$$\begin{gathered} \sum_{d \in DC} S^d_{s,w} \leq PF_w, \forall s \in S, \forall w \in W \\ \sum_{w \in W} S^d_{s,w} \leq D^d_s, \forall d \in D, \forall s \in S \\ S^d_{s,w} \geq 0, \forall s \in S, \forall w \in W, \forall d \in DC \\ PF_w \geq 0, \forall w \in W \end{gathered} \tag{6}$$

The amount shipped from each warehouse should be less or equal to the amount sent from a production facility to this warehouse. The amount we send from the warehouses to a distribution center should be lower than that distribution center's demand for each scenario.

Gurobi Implementation

[223]:
```
productionCost = {"A": -11,"B":-11.5}
profit = {("A",1): 18.5, ("A",2):18, ("A",3):18, ("A",4):17,
        ("B",1): 18, ("B",2):18.8, ("B",3):18.9, ("B",4):17.5}
demands = {("a",1): 5000, ("a",2): 3000, ("a",3): 7800, ("a",4): 4000,
          ("b",1): 3000, ("b",2): 1000, ("b",3): 6000, ("b",4): 3000,
          ("c",1): 4200, ("c",2): 4100, ("c",3): 3000, ("c",4): 5400}
warehouses = ["A","B"]
distributionCenters = [1,2,3,4]
Scenarios = ["a","b","c"]
ScenarioProbs={"a": 0.3, "b": 0.2, "c": 0.5}

model = gp.Model()
```

```
#first stage variables
production = model.addVars(warehouses, name = "PF")
#second stage variables
shipping = model.addVars(Scenarios, warehouses,
                         distributionCenters, name = "S")

#objective function
obj = gp.LinExpr()
obj+= gp.quicksum(productionCost[w]*production[w]
                  for w in warehouses)
obj+= gp.quicksum(ScenarioProbs[s]*profit[(w,dc)]*shipping[(s,w,dc)]
                  for s in Scenarios for dc in distributionCenters
                  for w in warehouses)
model.setObjective(obj, gp.GRB.MAXIMIZE)

#second stage constraints (all of them)
#you can't transport more than you produce for each warehouse
model.addConstrs(gp.quicksum(shipping[(s,w,dc)]
                             for dc in distributionCenters)<=
                 production[w] for s in Scenarios for w in warehouses)
#what you ship shouldn't be higher than the demand
model.addConstrs(gp.quicksum(shipping[(s,w,dc)]
                             for w in warehouses) <= demands[(s,dc)]
                 for dc in distributionCenters for s in Scenarios)

model.optimize()
model.printAttr("X")
```

```
Optimal objective  1.049990000e+05

    Variable            X
-------------------------
       PF[A]         6700
       PF[B]        10000
    S[a,A,1]         5000
    S[a,A,2]          800
    S[a,A,4]          900
    S[a,B,2]         2200
    S[a,B,3]         7800
    S[b,A,1]         3000
    S[b,B,2]         1000
    S[b,B,3]         6000
    S[b,B,4]         3000
    S[c,A,1]         4200
    S[c,A,4]         2500
    S[c,B,2]         4100
    S[c,B,3]         3000
    S[c,B,4]         2900
```

Based on our model solution, we should send 6700 products to warehouse A and 10000 products to warehouse B. The expected profit of this solution is $104,999.

Part B(2)

Mathematical Notation

Here the problem is mostly the same as above. Decision variables:

$$\begin{gathered} PF_w, w \in W \\ S^d_{s,w}, s \in S, w \in W, d \in DC \\ I_w, w \in W \\ W = \{A, B\} \\ DC = \{1, 2, 3, 4\} \\ S = \{a, b, c\} \end{gathered} \tag{7}$$

A new variable is how many warehouses to use I (binary variable). The new objective function with the fixed costs is therefore:

$$max - 100000 \sum_{w \in W} I_w - 10 \sum_{w \in W} PF_w - \sum_{w \in W} p_w PF_w + \sum_{w \in W} \sum_{d \in DC} \sum_{s \in S} Pr_s (20 - c^d_w) S^d_{s,w} \tag{8}$$

This is subject to constraints:

$$\begin{gathered} \sum_{d \in DC} S^d_{s,w} \leq PF_w \forall s \in S, \forall w \in W \\ \sum_{w \in W} S^d_{s,w} \leq D^d_s \forall d \in D, \forall s \in S \\ PF_w = PF_w I_w, \forall w \in W \\ S^d_{s,w} \geq 0, \forall s \in S, \forall w \in W, \forall d \in DC \\ PF_w \geq 0, \forall w \in W \\ I_w \in \{0, 1\} \end{gathered} \tag{9}$$

Gurobi Implementation

[211]:
```
dfdemand = pd.read_csv('demand.csv')
dfTC = pd.read_csv('TranportationCosts.csv')
```

[ ]:
```
#we transform the warehouse and distribution center columns to index columns
#this allows to easily access the data using the W/DC name
dfTCindex = dfTC.set_index(["Warehouse"])
dfdemandIX = dfdemand.set_index(["DistributionCenter"])
#PF cost we take from the TC dataframe
#transp. cost we take from the TC dataframe
#demands we take from the demand dataframe
warehouses = dfTC["Warehouse"]
distributionCenters = dfdemand["DistributionCenter"]
Scenarios = list(dfdemandIX.columns)
```

```
ScenarioProbs = 0.01
fixedCost = -100000
profit = 20
prodCost = -10


model = gp.Model()
#first stage variables
production = model.addVars(warehouses, name = "PF")
nwarehouses = model.addVars(warehouses, vtype = gp.GRB.BINARY, name = "I")
#second stage variables
shipping = model.addVars(Scenarios, warehouses, distributionCenters, name = "S")

#objective function
obj = gp.LinExpr()
obj+= gp.quicksum(fixedCost*nwarehouses[w] for w in warehouses)
obj+= gp.quicksum((prodCost-dfTCindex.loc[w,"PFacility"])*production[w]
                  for w in warehouses)
obj+= gp.quicksum(ScenarioProbs*(profit-dfTCindex.loc[w,dc])*shipping[(s,w,dc)]
                  for s in Scenarios
                  for dc in distributionCenters
                  for w in warehouses)
model.setObjective(obj, gp.GRB.MAXIMIZE)

#second stage constraints (all of them)
#you can't transport more than you produce for each warehouse
model.addConstrs(gp.quicksum(shipping[(s,w,dc)]
                             for dc in distributionCenters)<= production[w]
                 for s in Scenarios for w in warehouses)
#should be lower than demand
model.addConstrs(gp.quicksum(shipping[(s,w,dc)]
                             for w in warehouses) <= dfdemandIX.loc[dc,s]
                 for dc in distributionCenters for s in Scenarios)
#indicator constraint
model.addConstrs(production[w] == production[w]*nwarehouses[w]
                 for w in warehouses)

model.optimize()
```

[221]:
```
model.getAttr("ObjVal")
#model.printAttr("X")
```

[221]: 2124910.455000004

The adjusted model concludes that the fixed cost for each warehouse is so high that it only recommends using one warehouse in Stuttgart and subsequently send all the products from this warehouse to the different distribution centers. We produce and send 278,441 products to this single warehouse. The expected profit is $2,124,910.455.

[ ]:
```
#we create a new object to calculate the profit per scenario
array_shipping  = model.getAttr('X', shipping)
array_shipping
```

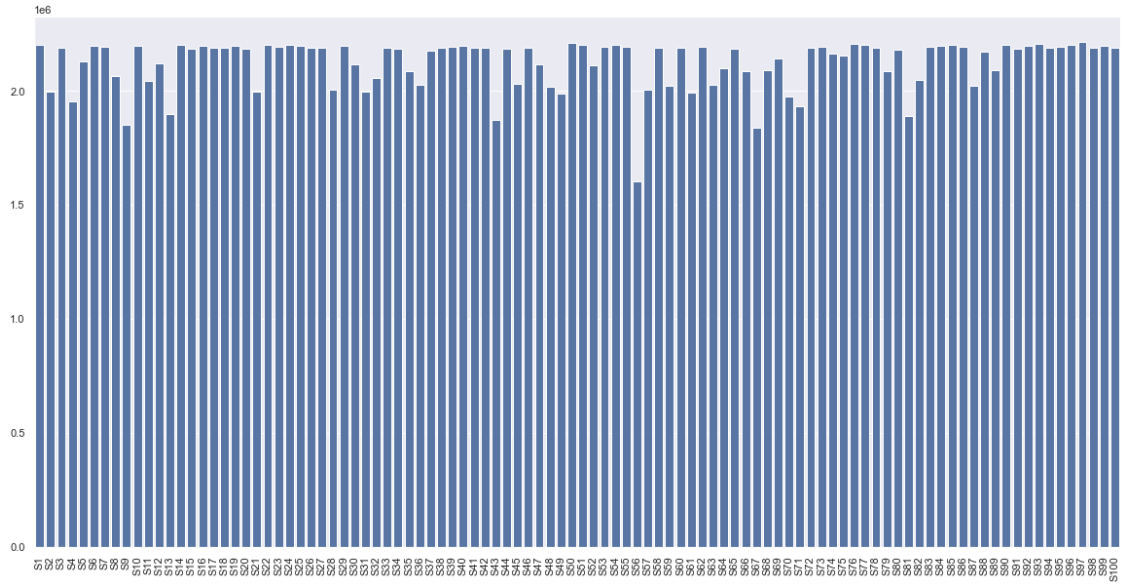
[ ]:
```
scenarioprofit = 0
TotalProfit = {}

#run through all scenarios
for s in Scenarios:
    #for w in warehouses:
    #here only Stuttgart is the warehouse
    #for each DC combination calculate profit
    for dc in distributionCenters:
        #add the profit for that scenario
        scenarioprofit+=array_shipping[s,"WH-Stuttgart",dc]*\
        (20-dfTCindex.loc["WH-Stuttgart",dc])
    #add the fixed cost and the PF cost
    finalprofit = scenarioprofit - 100000 + \
    (prodCost-dfTCindex.loc["WH-Stuttgart","PFacility"])*278441
    #store the scenario profit in a dictionary
    TotalProfit[s] = finalprofit
    scenarioprofit = 0

TotalProfit
```

[205]:
```
#these are just preparations to be able to create a chart
keys = list(TotalProfit.keys())
values = list(TotalProfit.values())
sn.set(rc={'figure.figsize':(20,10)})
chart = sn.barplot(x = keys, y = values, color = 'b', edgecolor = 'w')
chart.set_xticklabels(chart.get_xticklabels(), rotation=90)
chart
```

[205]: <AxesSubplot:>



[213]:
```
min(values)
```

[213]: 1601924.15

The worst case profit among all scenarios is $1,601,924.15, which happens under scenario 56.