```
%pip install scipy --upgrade
%pip install gurobipy
import gurobipy as grb
import math
import matplotlib.pyplot as plt
import scipy.stats as stats
import scipy.optimize as opt
import numpy as np
```

```
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (1.13.0)
Requirement already satisfied: numpy<2.3,>=1.22.4 in /usr/local/lib/python3.10/dist-packages (from scipy) (1.25.2)
Requirement already satisfied: gurobipy in /usr/local/lib/python3.10/dist-packages (11.0.1)
```

Q1-(1)

```
projSelected=np.array([1,1,1,1,1,1,1,1])
iniCost=np.array([250,650,250,500,700,30,350,70])
probSuccess=np.array([0.9,0.7,0.6,0.4,0.8,0.6,0.7,0.9])
minRev=np.array([600,1250,500,1600,1150,150,750,220])
modeRev=np.array([750,1500,600,1800,1200,180,900,250])
maxRev=np.array([900,1600,750,1900,1400,250,1000,320])

projCount=len(projSelected)
S=10
revenues=np.zeros((S,projCount))
success=np.zeros((S,projCount))
finalProfit=np.zeros((S,projCount))

for i in range(projCount):
    success[0:S,i]=np.random.binomial(1,probSuccess[i],S)
    revenues[0:S,i]=np.random.triangular(minRev[i],modeRev[i],maxRev[i],S)
    finalProfit[0:S,i]=((np.multiply(success[:,i],revenues[:,i]))-iniCost[i])*projSelected[i]

totalProfit=np.sum(finalProfit)
avgProfit=totalProfit/S
print("average total profit:",avgProfit)
```

```
average total profit: 2027.3458921494966
```

Q1-(2)

Project(1,2,5,6,7)

```
projSelected=np.array([1,1,0,0,1,1,1,0])
iniCost=np.array([250,650,250,500,700,30,350,70])
probSuccess=np.array([0.9,0.7,0.6,0.4,0.8,0.6,0.7,0.9])
minRev=np.array([600,1250,500,1600,1150,150,750,220])
modeRev=np.array([750,1500,600,1800,1200,180,900,250])
maxRev=np.array([900,1600,750,1900,1400,250,1000,320])

projCount=len(projSelected)
S=10000
revenues=np.zeros((S,projCount))
success=np.zeros((S,projCount))
finalProfit=np.zeros((S,projCount))
```

```
for i in range(projCount):
    success[0:S,i]=np.random.binomial(1,probSuccess[i],S)
    revenues[0:S,i]=np.random.triangular(minRev[i],modeRev[i],maxRev[i],S)
    finalProfit[0:S,i]=((np.multiply(success[:,i],revenues[:,i]))-iniCost[i])*projSelected[i]


totalProfit=np.sum(finalProfit)
avgProfit=totalProfit/S
print("profit:",avgProfit)

probSuccessList=projSelected*probSuccess
#print(probSuccessList)
probAllSuccess1=np.prod(probSuccessList[probSuccessList!=0])

print("probability of all selected projected being successful:",probAllSuccess1)
```

```
profit: 1450.5967081102729
probability of all selected projected being successful: 0.21167999999999998
```

optimal project selection

```
iniCost=np.array([250,650,250,500,700,30,350,70])
probSuccess=np.array([0.9,0.7,0.6,0.4,0.8,0.6,0.7,0.9])
minRev=np.array([600,1250,500,1600,1150,150,750,220])
modeRev=np.array([750,1500,600,1800,1200,180,900,250])
maxRev=np.array([900,1600,750,1900,1400,250,1000,320])


model1_2 = grb.Model('Q1_2')
I=8
x_vars=model1_2.addVars(range(I),vtype=grb.GRB.BINARY, name="x")


np.random.seed(5566)
S=10000
simSuccess=np.zeros((S,I))
simRev=np.zeros((S,I))
avgProfit=np.zeros(I)


for i in range(I):
    simSuccess[0:S,i]=np.random.binomial(1,probSuccess[i],S)
    simRev[:,i]=np.random.triangular(minRev[i],modeRev[i],maxRev[i],S)
    simRev=np.multiply(simRev,simSuccess)


model1_2.addConstr(grb.quicksum(x_vars[i]*iniCost[i]
                   for i in range(I))<=2000)
```

```
<gurobi.Constr *Awaiting Model Update*>
```

```
obj = grb.quicksum(x_vars[i]*(simRev[s,i]-iniCost[i])
                   for i in range(I)
                   for s in range(S))/S

model1_2.setObjective(obj,grb.GRB.MAXIMIZE)
model1_2.update()
model1_2.optimize()

solution = model1_2.getAttr('x', x_vars )
print(solution)

print(model1_2.Status == grb.GRB.OPTIMAL)

for v in model1_2.getVars():
    print(v.VarName, v.X)

optobj = model1_2.getObjective()
print(optobj.getValue())
```

```
Gurobi Optimizer version 11.0.1 build v11.0.1rc0 (linux64 - "Ubuntu 22.04.3 LTS")

CPU model: Intel(R) Xeon(R) CPU @ 2.20GHz, instruction set [SSE2|AVX|AVX2]
Thread count: 1 physical cores, 2 logical processors, using up to 2 threads

Optimize a model with 1 rows, 8 columns and 8 nonzeros
Model fingerprint: 0xdd01773e
Variable types: 0 continuous, 8 integer (8 binary)
Coefficient statistics:
  Matrix range     [3e+01, 7e+02]
  Objective range  [9e+01, 4e+02]
  Bounds range     [1e+00, 1e+00]
  RHS range        [2e+03, 2e+03]
Found heuristic solution: objective 1359.8718340
Presolve removed 1 rows and 8 columns
Presolve time: 0.00s
Presolve: All rows and columns removed

Explored 0 nodes (0 simplex iterations) in 0.01 seconds (0.00 work units)
Thread count was 1 (of 2 available processors)

Solution count 2: 1515.93 1359.87

Optimal solution found (tolerance 1.00e-04)
Best objective 1.515933512516e+03, best bound 1.515933512516e+03, gap 0.0000%
{0: 1.0, 1: 1.0, 2: 0.0, 3: 1.0, 4: 0.0, 5: 1.0, 6: 1.0, 7: 1.0}
True
x[0] 1.0
x[1] 1.0
x[2] 0.0
x[3] 1.0
x[4] 0.0
x[5] 1.0
x[6] 1.0
x[7] 1.0
1515.9335125162736
```

```
resultList=[]
for v in model1_2.getVars():
    resultList.append(v.X)

result=np.array(resultList)

probSuccessList=result*probSuccess
#print(probSuccessList)
probAllSuccess2=np.prod(probSuccessList[probSuccessList!=0])
print("profit：",optobj.getValue())
print("probability of all selected projected being successful:",probAllSuccess2)
```

```
profit： 1515.9335125162736
probability of all selected projected being successful: 0.095256
```

選擇Project(1,2,5,6,7)

- profit：1450.4413096372475
- probability：0.21167999999999998

optimal project selection

- profit：1515.9335125162736
- probability：0.095256

若直接選擇Project(1,2,5,6,7)，獲利程度較低，但全部成功的機率較高；反之則獲利程度較高，但全部成功的機率較低。

Q2-(1)

```
seats=19
demands=np.array(range(14,26,1))
probDemand=np.array([0.03,0.05,0.07,0.09,0.11,0.15,0.18,0.14,0.08,0.05,0.03,0.02])
probShow=0.9
price=150
cost=325
penalty=0

def profit(sellCount):
    expShow=probShow*sellCount
    revenue=price*sellCount
    global penalty

    for demand,prob in zip(demands,probDemand):
        realShow=probShow*min(demand,sellCount)
        if (realShow>seats):
            extra=realShow-seats
            penalty+=prob*cost*extra

    netProfit=revenue-penalty
    return -netProfit

initial=[seats]
bounds=[(14,25)]

result=opt.minimize(profit,initial,bounds=bounds)
expSeat=result.x[0]
totalProfit=-result.fun

lossSales=0
if round(expSeat) < seats:
        lossSales = (seats-expSeat) * price

print("tickets sell:",round(expSeat))
print("expected total profit",totalProfit)
print("penalty：",penalty)
print("loss sales：",lossSales)
```

```
tickets sell: 25
expected total profit 3653.475
penalty： [193.04999994]
loss sales： 0
```

Q2-(2)

```
def extraCost(sellCount):
    expShow=probShow*sellCount
    revenue=price*sellCount
    extraCost=0

    for demand,prob in zip(demands,probDemand):
        realShow=probShow*min(demand,sellCount)
        if (realShow>seats):
            extra=realShow-seats
            extraCost+=prob*cost*extra

    return extraCost


initial=[seats]
bounds=[(14,25)]

result=opt.minimize(extraCost,initial,bounds=bounds)

profit=round(result.x[0])*price-result.fun
print("ticket sell:",round(result.x[0]))
print("expected total profit:",profit)
print("penalty:",result.fun)

expDemand=np.sum(demands*probDemand)
#print(expDemand)

lostSales=max(0,expDemand-round(result.x[0]))
print("lost sales:",lostSales)
```

```
ticket sell: 19
expected total profit: 2850
penalty: 0
lost sales: 0.3200000000000003
```

Q2-(3)

最大化利潤(Q2-(1))

- ticket sell：25
- profit：3460.4250000585
- penalty：386.09999988
- lost sales：0

最小化penalty(Q2-(2))

- ticket sell：19
- profit：2850
- penalty：0
- lost sales：0.3200000000000003

若選擇最大化利潤，獲利程度較高，但penalty較多，表示有些顧客得不到服務；反之則penalty較少，讓每個買票的顧客都能得到服務，但獲利程度較低。

Q2-(4)

```
from scipy.stats import binom
maxExpProfit = -np.inf
optReservation = 0
penalty=0
lossSales=0

for reservations in range(14,26):
        expProfit=0
        for demand, prob in zip(demands,probDemand):

            expShow=int(reservations*probShow)
```

```
    expShow=int(reservations*probShow)
    prob_Show=binom.pmf(range(seats+1),reservations,probShow)


    revenue=min(expShow,seats)*price
    penalty=max(expShow-seats,0)*cost
```