Session 26: Portfolio Optimization Solutions

In this lab, you will practice your Gurobi coding skills by analyzing a large-scale portfolio optimization case.

1. Problem

Trojan investment is exploring new methods for updating its portfolio of US stocks based on mixed integer linear and quadratic optimization. In particular, it would like to optimize the trade-off between returns and risk, given the presence of transaction costs and managerial overhead. In particular, transaction cost implies that the new portfolio must not be too different from the current portfolio. Managerial overhead means that if the company invest in any stock, there should be a sufficiently large stake, and the number of stocks in the portfolio cannot be too large. The abstract formulation is given below.

Data:

- S: the set of stocks.
- w_i: the old weight of stock $i \in S$ before optimization. (The "weight" of a stock is % of total funds invested in the stock; weights of all stocks should add to one.)
- R_i: the expected annual return of stock $i \in S$.
- C_{ij}: the estimated covariance between stocks $i, j \in S$.
- σ_{target}: the maximum volatility of the final portfolio.
- Δ: the total movement allowed between the old weights and the new weights.
- k: the maximum # of stocks allowed in the portfolio.
- ϵ: the minimum non-zero weight allowed.

Decision variables:

- x_i: the new weight of stock i. (Continuous)
- δ_i: difference in weight for stock i. (Continuous)
- z_i: whether to use stock i. (Binary)

Objective and constraints: All summations are over the set S of stocks.

$$
\begin{array}{rll}
\text{Maximize:} & \sum_i R_i x_i & \text{(Average Return)} \\
\text{subject to:} & & \\
\text{(Valid weights)} & \sum_i x_i = 1 & \\
\text{(Risk tolerance)} & \sum_{i,j} C_{ij} x_i x_j \le \sigma^2_{target} & \\
\text{(Change in weights 1)} & x_i - w_i \le \delta_i & \text{for each stock } i. \\
\text{(Change in weights 2)} & -(x_i - w_i) \le \delta_i & \text{for each stock } i. \\
\text{(Change in weights 3)} & \frac{1}{2} \sum_i \delta_i \le \Delta & \\
\text{(Non-negligible weights)} & \epsilon z_i \le x_i \le z_i & \text{for each stock } i. \\
\text{(Simplicity)} & \sum_i z_i \le k & \\
\text{(Non-negativity)} & x_i \ge 0 &
\end{array}
$$

2. Data

The file “26-data.xlsx” (emailed to everyone and available on NBViewer along with other handouts and notes) contains two sheets. The sheet “s&p500” contains the stock prices of every stock on the S&P500 for 10 years. The sheet “oldPortfolio” contains the weights on the current portfolio. The following code can be used to load the data and calculate the returns R_i and covariances C_{ij}.

```
[1]: import pandas as pd
     import numpy as np

     oldPortfolio=pd.read_excel('26-data.xlsx',sheet_name='oldPortfolio'\
                                ,index_col=0)['Weight']
     oldPortfolio
```

```
Stock
AMGN    0.306342
CNC     0.231379
FFIV    0.290586
FL      0.019480
LEG     0.152214
Name: Weight, dtype: float64
```

```
[2]: rawPrices=pd.read_excel('26-data.xlsx',sheet_name='s&p500'\
                             ,index_col=0).fillna(method='ffill')
     logPrices=np.log(rawPrices)
     priceChange=logPrices.diff(1).iloc[1:,:].fillna(0)
     C=priceChange.cov()*252           # About 252 business days in a year
     R=priceChange.mean()*252
```

```
[3]: R.head()
```

```
MMM     0.101382
AOS     0.252184
ABT     0.084367
ABBV    0.096193
ACN     0.141367
dtype: float64
```

```
[4]: C.iloc[:5,:5]
```

	MMM	AOS	ABT	ABBV	ACN
MMM	0.049054	0.042544	0.021191	0.008905	0.031119
AOS	0.042544	0.098905	0.025834	0.010012	0.039423
ABT	0.021191	0.025834	0.042142	0.012491	0.023052
ABBV	0.008905	0.010012	0.012491	0.039773	0.008844
ACN	0.031119	0.039423	0.023052	0.008844	0.063869

3. Optimizing for Given Parameters

Solve the optimization problem for the following parameters:

- σ_{target}: 0.25

- Δ: 0.3
- k: 20
- ϵ: 0.001

The code should save the result in an excel file "26-output.xlsx" with a single sheet, in the same format as the "oldPortfolio" sheet above.

```
[14]: stdMax=0.25
      maxChange=0.3
      k=20
      eps=0.001
      S=R.index

      from gurobipy import Model, GRB
      mod=Model()

      x=mod.addVars(S)
      z=mod.addVars(S,vtype=GRB.BINARY)
      delta=mod.addVars(S)

      totRet=sum(R.loc[i]*x[i] for i in S)
      mod.setObjective(totRet,sense=GRB.MAXIMIZE)

      mod.addConstr(sum(x[i] for i in S)==1)
      totCov=sum(C.loc[i,j]*x[i]*x[j] for i in S for j in S)
      risk=mod.addConstr(totCov<=stdMax**2)
      numUsed=sum(z[i] for i in S)
      simplicity=mod.addConstr(numUsed<=k)
      totChange=sum(delta[i] for i in S)/2
      change=mod.addConstr(totChange<=maxChange)

      for i in S:
          if i in oldPortfolio.index:
              old=oldPortfolio.loc[i]
          else:
              old=0
          mod.addConstr(x[i]-old<=delta[i])
          mod.addConstr(-x[i]+old<=delta[i])
          mod.addConstr(x[i]<=z[i])
          mod.addConstr(x[i]>=eps*z[i])
      mod.setParam('outputflag',False)
      #mod.setParam('OptimalityTol',1e-6)
      mod.optimize()
```

```
Parameter OptimalityTol unchanged
   Value: 1e-06  Min: 1e-09  Max: 0.01  Default: 1e-06
Optimize a model with 2015 rows, 1509 columns and 5533 nonzeros
Model has 1 quadratic constraint
Variable types: 1006 continuous, 503 integer (503 binary)
Coefficient statistics:
  Matrix range     [1e-03, 1e+00]
  QMatrix range    [9e-07, 2e+01]
```

```
  Objective range  [3e-04, 5e-01]
  Bounds range     [1e+00, 1e+00]
  RHS range        [2e-02, 2e+01]
  QRHS range       [6e-02, 6e-02]
Presolve removed 542 rows and 44 columns
Presolve time: 0.06s
Presolved: 1473 rows, 1465 columns, 4454 nonzeros
Variable types: 962 continuous, 503 integer (503 binary)

Root relaxation: objective 2.990471e-01, 25 iterations, 0.00 seconds

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

H    0     0                       0.2429456   26.78178      -      -    0s
H    0     0                       0.2543559   26.78178      -      -    0s
H    0     0                       0.2566798   26.78178      -      -    0s
*    0     0               0       0.2566798    0.25866  0.77%      -    0s
H    0     0                       0.2566851    0.25866  0.77%      -    0s
     0     0     cutoff    0         0.25669    0.25669  0.00%      -    1s

Explored 1 nodes (83 simplex iterations) in 1.23 seconds
Thread count was 4 (of 4 available processors)

Solution count 4: 0.256685 0.25668 0.254356 0.242946

Optimal solution found (tolerance 1.00e-04)
Warning: max constraint violation (6.2116e-06) exceeds tolerance
Best objective 2.566851201519e-01, best bound 2.566851201519e-01, gap 0.0000%
```

```
[7]: import numpy as np
     print('Return:',totRet.getValue())
     print('Risk:',np.sqrt(totCov.getValue()))
     print('# stocks:',numUsed.getValue())
     print('Change in portfolio:',totChange.getValue())
     data=[]
     for i in S:
         if x[i].x>0:
             data.append([i,x[i].x])
     df=pd.DataFrame(data,columns=['Stock','Weight'])
     df.to_excel('26-output.xlsx',index=False)
```

```
Return: 0.2566851201519479
Risk: 0.2500124229840874
# stocks: 8.0
Change in portfolio: 0.30000000000000004
```

4. Tradeoff between multiple objectives

The following example illustrates how to analyze problems with multiple objectives. It is based on Q1 from session 23, or DMD Example 8.1.

Decision variables: Let A, G, D denote the fraction of total investment to put in the assets Advent, GSS, and Digital.

Objective and constraints:

$$\begin{aligned} \text{Maximize:} \quad & 11A + 14G + 7D \\ \text{subect to:} \quad & \\ \text{(Fractions)} \quad & A + G + D = 1 \\ \text{(Target risk)} \quad & \sqrt{16A^2 + 22G^2 + 10D^2 + 6AG + 2GD - 10AD} \leq \sigma \\ \text{(Nonnegativity)} \quad & A, G, D \geq 0 \end{aligned}$$

[8]:
```
from gurobipy import Model, GRB
import numpy as np
mod2=Model()
sigma=GRB.INFINITY
A=mod2.addVar()
G=mod2.addVar()
D=mod2.addVar()
ret=11*A+14*G+7*D
riskSquared=16*A*A+22*G*G+10*D*D+6*A*G+2*G*D-10*A*D
mod2.setObjective(riskSquared)
mod2.addConstr(A+G+D == 1)
mod2.setParam('outputflag',False)
mod2.optimize()
print('Minimum risk possible:',np.sqrt(riskSquared.getValue()))
```

```
Minimum risk possible: 1.8928303077552984
```

[9]:
```
mod2.setObjective(ret,sense=GRB.MAXIMIZE)
riskConstraint=mod2.addConstr(riskSquared<=GRB.INFINITY)
mod2.setParam('outputflag',False)
mod2.optimize()
print('Maximum possible return:',ret.getValue())
print('Corresponding sigma:',np.sqrt(riskSquared.getValue()))
```

```
Maximum possible return: 13.999999999968766
Corresponding sigma: 4.690415759786275
```

[10]:
```
sigmaList=np.linspace(1.893,5,20)
retList=[]
for sigma in sigmaList:
    riskConstraint.QCRHS=sigma**2
    mod2.optimize()
    retList.append(ret.getValue())
import matplotlib.pyplot as plt
plt.plot(sigmaList,retList,'ro')
plt.title('Tradeoff between risk and return')
plt.xlabel('Risk')
plt.ylabel('Return')
plt.show()
```

```
<Figure size 640x480 with 1 Axes>
```

(Optional) 4.1 Exercise

Analyze the tradeoff between return and risk (σ_{target}), as well as return and change in portfolio (Δ) in the problem for Trojan investment.

4.1.1 Tradeoff between return and risk

[11]:
```
mod.setObjective(totCov,sense=GRB.MINIMIZE)
mod.optimize()
print('Minimum total std:',np.sqrt(totCov.getValue()))
```

```
Optimize a model with 2015 rows, 1509 columns and 5533 nonzeros
Model has 126756 quadratic objective terms
Model has 1 quadratic constraint
Variable types: 1006 continuous, 503 integer (503 binary)
Coefficient statistics:
  Matrix range     [1e-03, 1e+00]
  QMatrix range    [9e-07, 2e+01]
  Objective range  [0e+00, 0e+00]
  QObjective range [2e-06, 3e+01]
  Bounds range     [1e+00, 1e+00]
  RHS range        [2e-02, 2e+01]
  QRHS range       [6e-02, 6e-02]

Loaded MIP start with objective 0.0495691

Presolve removed 542 rows and 44 columns
Presolve time: 0.06s
Presolved: 1473 rows, 1465 columns, 4449 nonzeros
Presolved model has 126756 quadratic objective terms
Variable types: 962 continuous, 503 integer (503 binary)

Root relaxation: objective 2.536668e-02, 258 iterations, 0.02 seconds

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

     0     0    0.02537    0    6    0.04957    0.02537  48.8%     -    0s
H    0     0                       0.0253667    0.02537  0.00%     -    0s

Explored 1 nodes (258 simplex iterations) in 0.28 seconds
Thread count was 4 (of 4 available processors)

Solution count 2: 0.0253667 0.0495691

Optimal solution found (tolerance 1.00e-04)
Best objective 2.536668343954e-02, best bound 2.536668343954e-02, gap 0.0000%
Minimum total std: 0.15926921686108744
```

[12]:
```
mod.setObjective(totRet,sense=GRB.MAXIMIZE)
risk.QCRHS=GRB.INFINITY
```

```
mod.optimize()
print('Maximum return:',totRet.getValue())
print('Corresponding total std:',np.sqrt(totCov.getValue()))
```

```
Optimize a model with 2015 rows, 1509 columns and 5533 nonzeros
Model has 1 quadratic constraint
Variable types: 1006 continuous, 503 integer (503 binary)
Coefficient statistics:
  Matrix range     [1e-03, 1e+00]
  QMatrix range    [9e-07, 2e+01]
  Objective range  [3e-04, 5e-01]
  Bounds range     [1e+00, 1e+00]
  RHS range        [2e-02, 2e+01]

Loaded MIP start with objective 0.232978

Presolve removed 542 rows and 44 columns
Presolve time: 0.03s
Presolved: 1473 rows, 1465 columns, 4454 nonzeros
Variable types: 962 continuous, 503 integer (503 binary)

Root relaxation: objective 3.145449e-01, 16 iterations, 0.00 seconds

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

*    0     0               0       0.3145449    0.31454  0.00%     -    0s

Explored 0 nodes (16 simplex iterations) in 0.06 seconds
Thread count was 4 (of 4 available processors)

Solution count 2: 0.314545 0.232978

Optimal solution found (tolerance 1.00e-04)
Best objective 3.145448532023e-01, best bound 3.145448532023e-01, gap 0.0000%
Maximum return: 0.3145448532023129
Corresponding total std: 1.257137907894366
```

[13]:
```
sigmaList=np.linspace(0.114,2.10,20)
retList=[]
for sigma in sigmaList:
    risk.QCRHS=sigma**2
    mod.optimize()
    retList.append(totRet.getValue())
import matplotlib.pyplot as plt
plt.plot(sigmaList,retList,'ro')
plt.title('Tradeoff between risk and return')
plt.xlabel('Risk')
plt.ylabel('Return')
plt.show()
```

```
Optimize a model with 2015 rows, 1509 columns and 5533 nonzeros
```

```
Model has 1 quadratic constraint
Variable types: 1006 continuous, 503 integer (503 binary)
Coefficient statistics:
  Matrix range     [1e-03, 1e+00]
  QMatrix range    [9e-07, 2e+01]
  Objective range  [3e-04, 5e-01]
  Bounds range     [1e+00, 1e+00]
  RHS range        [2e-02, 2e+01]
  QRHS range       [1e-02, 1e-02]

MIP start did not produce a new incumbent solution

Presolve removed 542 rows and 44 columns
Presolve time: 0.05s
Presolved: 1473 rows, 1465 columns, 4454 nonzeros
Variable types: 962 continuous, 503 integer (503 binary)

Root relaxation: objective 2.970790e-01, 25 iterations, 0.00 seconds

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

     0     0 infeasible    0               - infeasible      -     -    0s

Explored 0 nodes (218 simplex iterations) in 0.17 seconds
Thread count was 4 (of 4 available processors)

Solution count 0

Model is infeasible or unbounded
Best objective -, best bound -, gap -
```

```
        ---------------------------------------------------------------------------

        AttributeError                            Traceback (most recent call last)

        <ipython-input-13-5fed06f3b4c7> in <module>
          4     risk.QCRHS=sigma**2
          5     mod.optimize()
    ----> 6     retList.append(totRet.getValue())
          7 import matplotlib.pyplot as plt
          8 plt.plot(sigmaList,retList,'ro')


        linexpr.pxi in gurobipy.LinExpr.getValue()


        var.pxi in gurobipy.Var.__getattr__()
```

```
var.pxi in gurobipy.Var.getAttr()


AttributeError: b"Unable to retrieve attribute 'x'"
```

4.1.2 Tradeoff between return and transaction cost

```
[ ]: risk.QCRHS=0.25**2
     DeltaList=np.linspace(0,1,20)
     retList=[]
     for Delta in DeltaList:
         change.RHS=Delta
         mod.optimize()
         retList.append(totRet.getValue())
     import matplotlib.pyplot as plt
     plt.plot(DeltaList,retList,'ro')
     plt.title('Tradeoff between return and transaction cost')
     plt.xlabel('Change in portfolio')
     plt.ylabel('Return')
     plt.show()
```