Exercises - Section 2: Lecture 8 – Quadratic Modeling - Solutions

The following four pieces of mathematical models require quadratic terms. In each example, x, y, and z denote variables and all other letters denote data. For each one, write mathematical expressions and gurobipy code. Use auxiliary variables when helpful.

a. maximize $(x + y + z)^2$

DESCRIPTION	MATH	GUROBIPY	`m=gp.Model()`

Variables	x, y, z, w	`x = m.addVar(vtype=gp.GRB.CONTINUOUS, name="x")`
		`y = m.addVar(vtype=gp.GRB.CONTINUOUS, name="y")`
		`z = m.addVar(vtype=gp.GRB.CONTINUOUS, name="z")`
		`w = m.addVar(vtype=gp.GRB.CONTINUOUS, name="w")`	

Objective	Maximize w^2	`m.setObjective(w*w, gp.GRB.MAXIMIZE)`	
Constraint that defines the auxiliary variable w	$w = x + y + z$	`m.addConstr(w==x+y+z)`	

Auxiliary variables cut down the number of multiplications and additions required, from 9 to 3.

b. minimize $\sum_{i=1}^{N} \sum_{j=1}^{N} a_{ij} x_i x_j$

DESCRIPTION	MATH	GUROBIPY	`m=gp.Model()`
Variables	x	`x = m.addVars(range(1,N+1), vtype=gp.GRB.CONTINUOUS, name="x")`	
Objective	Minimize $\sum_{i=1}^{N} \sum_{j=1}^{N} a_{ij} x_i x_j$	`m.setObjective(sum(a[i,j]*x[i]*x[j] for i in range(1,N+1) for j in range(1,N+1)), gp.GRB.MINIMIZE)`	

Here, adding auxiliary variables would not help; there are N^2 terms that require two multiplications, and N^2 additions, whether auxiliary variables are used or not.

c. $(x + y + z)^2 \leq R$

DESCRIPTION	**MATH**	**GUROBIPY**	`m=gp.Model()`

Variables	x, y, z, w	`x = m.addVar(vtype=gp.GRB.CONTINUOUS, name="x")`
		`y = m.addVar(vtype=gp.GRB.CONTINUOUS, name="y")`
		`z = m.addVar(vtype=gp.GRB.CONTINUOUS, name="z")`
		`w = m.addVar(vtype=gp.GRB.CONTINUOUS, name="w")`	

Constraint that defines the auxiliary variable w	$w = x + y + z$	`m.addConstr(w==x+y+z)`	
Quadratic constraint	$w^2 \leq R$	`m.addConstr(w*w<=R)`	

Auxiliary variables cut down the number of multiplications and additions required, from 9 to 3.

d. $\sum_{i=1}^{N} \sum_{j=1}^{N} a_{ij} x_i x_j \leq R$

DESCRIPTION	**MATH**	**GUROBIPY**	`m=gp.Model()`
Variables	x	`x = m.addVars(range(1,N+1), vtype=gp.GRB.CONTINUOUS, name="x")`	
Quadratic constraint	$\sum_{i=1}^{N} \sum_{j=1}^{N} a_{ij} x_i x_j \leq R$	`m.addConstr(sum(a[i,j]*x[i]*x[j] for i in range(1,N+1) for j in range(1,N+1)) <= R)`	

Here, adding auxiliary variables would not help; there are N^2 terms that require two multiplications, and N^2 additions, whether auxiliary variables are used or not.

2. An investment portfolio manager wants to determine how much money to invest in each of 1000 stocks (the S&P 500 plus 500 smaller stocks). The manager's data science team has built models to predict the expected return (per dollar invested) relative to the market of each stock i (denoted by α_i) and they have calculated the historical covariance β_{ij} for each pair of stocks i and j. The covariances are used as a proxy for investment risk; if x_i is the amount invested in stock i, then the total risk can be written as $\sum_i \sum_j \beta_{ij} x_i x_j$. The portfolio has a total of B dollars available for investment.

a. Create mathematical and gurobipy models that the manager can use to determine how much money to invest in each stock in order to maximize the total expected return, while adhering to the budget and keeping the risk below a specific tolerance T. Use auxiliary variables for the quadratic terms if needed.

ENGLISH	MATH	GUROBIPY	`m=gp.Model()`

<u>Data</u>
Number of stocks
Investment budget
Risk tolerance
Predicted returns
Covariances
	
	N
	B
	T
	α_i
	β_{ij}
		`N = 1000`
		`B # read from file`
		`T # read from file`
		`alpha # read from file (dictionary)`
		`beta # read from file (dictionary)`	

<u>Variables</u> Amount invested in each stock	x_i	`x = m.addVars(range(1,N+1), vtype=gp.GRB.CONTINUOUS,name="x")`	
<u>Objective</u> Maximize expected return	Maximize $\sum_{i=1}^{N} \alpha_i x_i$	`m.setObjective(sum(alpha[i]*x[i] for i in range(1,N+1)), gp.GRB.MAXIMIZE)`	

<u>Constraints</u>
Can't invest beyond budget
 	
Total risk within tolerance
 	
Can't invest negative dollars
	$\sum_{i=1}^{N} x_i \leq B$
	
	$\sum_{i=1}^{N} \sum_{j=1}^{N} \beta_{ij} x_i x_i \leq T$
	
	all $x_i \geq 0$
		`m.addConstr(sum(x[i] for i in range(1,N+1)) <= B)`
		
		`m.addConstr(sum(beta[i,j]*x[i]*x[j] for i in range(1,N+1) for j in range(1,N+1)) <= T)`
		
		`# implied by variable declaration`	

b. Create mathematical and gurobipy models that the manager can use to determine how much money to invest in each stock in order to minimize the risk, while adhering to the budget and having an expected return of at least R. Use auxiliary variables for the quadratic terms if needed.

ENGLISH	**MATH**	**GUROBIPY** `m=gp.Model()`

<u>Data</u>
Number of stocks
Investment budget
Minimum return required
Predicted returns
Covariances
	
	N
	B
	R
	α_i
	β_{ij}
		
		`N = 1000`
		`B # read from file`
		`R # read from file`
		`alpha # read from file (dictionary)`
		`beta # read from file (dictionary)`

<u>Variables</u> Amount invested in each stock	x_i	`x = m.addVars(range(1,N+1), vtype=gp.GRB.CONTINUOUS,name="x")`
<u>Objective</u> Minimize estimated risk	Minimize $\sum_{i=1}^{N}\sum_{j=1}^{N}\beta_{ij}x_ix_i$	`m.setObjective(sum(beta[i,j]*x[i]*x[j] for i in range(1,N+1) for j in range(1,N+1)), gp.GRB.MINIMIZE)`

<u>Constraints</u>
Can't invest beyond budget
 	
Total risk within tolerance
Can't invest negative dollars
	$\sum_{i=1}^{N} x_i \leq B$
	
	$\sum_{i=1}^{N} \alpha_i x_i \geq R$
	
	all $x_i \geq 0$
		`m.addConstr(sum(x[i] for i in range(1,N+1)) <= B)`
		
		`m.addConstr(sum(alpha[i]*x[i] for i in range(1,N+1)) >= R)`
		
		`# implied by variable declaration`

3. A data scientist would like to run a regression with special restrictions: the regression constant (a_0) must be zero, the sum of all regression coefficients $a_1, \dots, a_m$ must be zero, and no coefficient a_j can be greater than 1 or less than -1. The data scientist has n data points, each consisting of a response y_i and predictors x_{ij}. Create mathematical and gurobipy models that the data scientist can use to find the constrained regression solution. Use auxiliary variables for the quadratic terms if it would be helpful.

ENGLISH	**MATH**	**GUROBIPY**	`m=gp.Model()`

<u>Data</u>
Number of data points
Number of parameters
Known data (responses)
Known data (predictors)
	
	N
	M
	y_i
	x_{ij}
		
		`N # read from file`
		`M # read from file`
		`y # read from file (dictionary)`
		`x # read from file (dictionary)`	

<u>Variables</u>
Regression coefficient for each predictor
 	
Error term for each data point
	a_j
	
	u_i
		`a = m.addVars(range(M+1), lb=-1, ub=-1, vtype=gp.GRB.CONTINUOUS, name=”a”)`
		
		`u = m.addVars(range(1,N+1), lb=-GRB.INFINITY, vtype=gp.GRB.CONTINUOUS, name="u")`	

<u>Objective</u> Minimize sum of squared error (the usual linear regression objective)	Minimize $\sum_{i=1}^{N} u_i^2$	`m.setObjective(sum(u[i]*u[i] for i in range(1,N+1), gp.GRB.MINIMIZE)`	

<u>Constraints</u>
Define error variable for each data point
 	
Regression constant is zero
 	
Sum of all regression coefficients is zero
 	
Each regression coefficient is between -1 and 1
	$u_i = y_i - (a_0 + \sum_{j=1}^{M} a_j x_{ij})$ for all i
	
	$a_0 = 0$
	
	$\sum_{j=1}^{M} a_j = 0$
	
	$-1 \leq a_j \leq 1$ for all j
		`m.addConstrs((u[i] == y[i] - sum(a[j]*x[i,j] for j in range(1,M+1))) for i in range(1,N+1))`
		
		`m.addConstr(a[0] == 0)`
		
		`m.addConstr(sum(a[j] for j in range(M+1)) == 0)`
		
		`# implied by variable declaration`	

Notes about this model: (1) It’s necessary to be careful with the ranges of variables. There are $M + 1$ regression coefficients (the constant term a_0 as well as one regression coefficient per variable) but the data is defined just from 1 to N. (2) Because each data point’s error can be negative, it’s necessary to override gurobipy’s default non-negativity constraint by adding `lb=-GRB.INFINITY` in the variable definition.

4. An appliance manufacturer is getting ready to launch production of three new models of oven. The three oven models do the same thing (cook food), but they have different features. Oven A has the most features (e.g., convection, self-cleaning, Sabbath mode, on/off timer), Oven B has only a subset of those features (e.g., self-cleaning, Sabbath mode, on/off timer), and Oven C has even fewer of the features (e.g., self-cleaning, Sabbath mode). The table below shows the cost to produce each model of oven.

Model	**Production cost**
A	\$600
B	\$400
C	\$375

The manufacturer would like to maximize its total profit (the sum over all ovens sold of selling price minus production cost) on the new ovens, and the prices they charge will affect the number of ovens they sell. The manufacturer's marketing experts and data scientists believe that the number of ovens sold of each model will depend on the price of its model, and how much different that price is from the prices of the next model up and/or down. They have collaborated to come up with the following estimates. If P_i is the selling price of oven model i and Q_i is the number of ovens of model i they can sell, then:

$$Q_A = 3{,}000 - 5P_A + (7{,}000 - 25(P_A - P_B))$$

$$Q_B = 20{,}000 - 25P_B - \big(7{,}000 - 25(P_A - P_B)\big) - 40(P_B - P_C)$$

$$Q_C = 10{,}000 - 20P_C + 40(P_B - P_C)$$

Create mathematical and gurobipy models that the manufacturer can use to maximize its profit. [Remember to use auxiliary variables for quadratic terms if they're helpful, and don't forget obvious restrictions like the prices and quantities sold can't be negative.] Solve the gurobipy model.

Note: Because this is a nonconvex model, in the gurobipy code you'll need to include the statement `m.setParam("NonConvex",2)` before optimizing.

ENGLISH	MATH	GUROBIPY `m=gp.Model()` **`m.setParam("NonConvex",2)`**

<u>Data</u>
Number oven models
 	
Production cost of each model
 	
Constant ($j = 0$) and elasticity coefficient for each model ($j = 1,2,3$) in each model's equation
	M
	
	c_j
	
	α_{ij}
		`M # read from file`
		`# Let indices 1, 2, and 3 denote oven models A, B, and C`
		`# read from file`
		
		`alpha # read from file`
		`# alphas for model A are 10000, -30, 250, 0`
		`# alphas for model B are 13000, 25, -90, 40`
		`# alphas for model C are 10000, 0, 40, -60`

<u>Variables</u>
Price of each oven model
 	
Estimated number sold of each oven model
	p_j
	
	q_i
		`p = m.addVars(range(1,M+1), vtype=gp.GRB.CONTINUOUS, name="p")`
		
		`q = m.addVars(range(1,M+1), vtype=gp.GRB.CONTINUOUS, name="q")`

<u>Objective</u> Maximize total profit	Maximize $\sum_{i=1}^{M} q_i(p_i - c_i)$	`m.setObjective(sum(q[i]*(p[i]-c[i]) for i in range(1,M+1)), gp.GRB.MAXIMIZE)`

<u>Constraints</u>
Quantity sold is set according to elasticity equations
 	
All prices are non-negative
 	
All quantities sold are non-negative
	$q_i = \alpha_{i0} + \sum_{j=1}^{M} \alpha_{ij} p_j$ for all i
	
	$p_i \geq 0$ for all i
	
	$q_i \geq 0$ for all i
		`m.addConstrs((q[i] == alpha[i,0] + sum(alpha[i,j]*p[j] for j in range(1,M+1))) for i in range(1,M+1))`
		
		`# implied by variable declaration`
		
		`# implied by variable declaration`

Note: Above, the quantities are set as continuous variables because they're just estimates anyway, and rounding off with large numbers isn't a big deal. However, you could also set the quantities to be integer, since ovens have to be sold in integer quantities:

```
q = m.addVars(range(1,M+1), vtype=gp.GRB.INTEGER, name="q")
```

If you do that, you'll get the same answer but it might take Gurobi a couple of minutes to solve it, because it has to eliminate small roundoffs.

The two gurobipy models can be found in files `Pt. 2 2.8 question 4-continuous.py` and `Pt. 2 2.8 question 4-integer.py`.

Either way, the solution is:

Model A:	$741.18 selling price	1000 ovens sold
Model B:	$529.41 selling price	3500 ovens sold
Model C:	$490.44 selling price	1750 ovens sold

NOTES: