Solutions to Sample Final Exam C

Learning Objective:

- Create Python code to automate a given task.
- Formulate linear optimization models to inform a business decision.

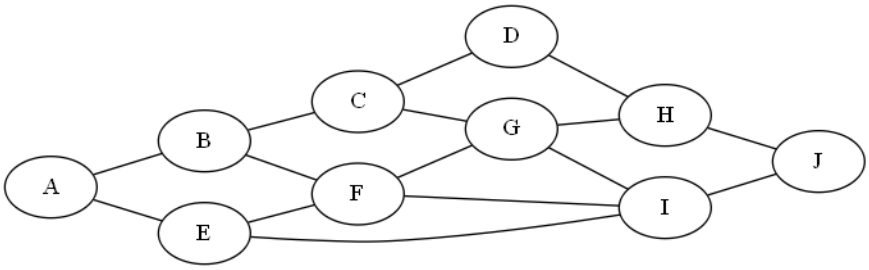
Instructions:

The final exam tests your mastery of skills taught in Weeks 7-12, which culminates in creating linear optimization models to inform a given business decision. There are three questions, worth a total of 30 points. The exam is 100 minutes, and is open-notes but closed-computer. You can bring paper notes or books of any kind, but no computers, tablets, or cell phones are allowed. Do not share your solutions with a student who has not yet completed an exam and do not look at other people's solutions. **Any violation of academic integrity will result in a zero grade for the exam for everyone involved.**

As long as you fulfill all the specifications described in the problem description, it doesn't matter how you model the problem or how efficient is your code. As with the midterm, partial credits will be given for any fragments of correct solutions, or for English descriptions that would lead to a solution.

Q1. Campus Security Planning (Concrete Formulation; 13 points)

USC would like to protect every street around its campus by stationing security staff at intersections, with the goal that every street has at least one staff stationed at one of the two ends. A street is defined to be the segment connecting two neighboring intersections, which are the two ends of the street. For example, the following map has 10 intersections and 15 streets. The street connecting A-B, can be protected by staffing either at intersection A or intersection B (these are the two ends of the street). Staffing at one of these would suffice, although it's okay to staff at both ends as well. Staffing someone at intersection A would protect the streets A-B and A-E simultaneously. Similarly, staffing someone at intersection F would simultaneously protect the streets B-F, E-F, F-G and F-I, and so on.



For the above map, formulate a linaer optimization problem to minimize the number of staff needed, subject to protecting all 15 of the streets, as well as satisfying the following constraints:

- If someone is stationed at intersection I, then someone must be also stationed at J. However, it is okay to station only J but not I.
- The segments connecting intersections A, B, C and D are part of a large avenue, and the segments connecting E, F, G and H is another large avenue. One of these two large avenues must have at least 3 out of 4 intersections stationed.
- Intersections A and F must either both be stationed or both be unstationed.
- The segments C-G, G-H and B-F are especially dangeous. At least 2 out of 3 of these especially dangeous segments must have staff stationed at both ends.

Decision Variables:

- $X_A, X_B, \cdots, X_J$ for whether to station staff at each intersection. (binary)
- L_1: whether to station at least 3 out of 4 intersections for the large avenue connecting A-B-C-D. (binary)
- L_2: whether to station at least 3 out of 4 intersections for the large avenue connecting E-F-G-H. (binary)
- Y_{CG}, Y_{GH}, Y_{BF}: whether to station both ends of the streets C-G, G-H and B-F respectively. (binary)

Objective and Constraints:

$$
\begin{aligned}
\text{Min.} \quad & X_A + X_B + \cdots + X_J \\
\text{s.t.} \quad & X_A + X_B \geq 1 \\
& X_B + X_C \geq 1 \\
& X_C + X_D \geq 1 \\
& X_A + X_E \geq 1 \\
& X_B + X_F \geq 1 \\
& X_C + X_G \geq 1 \\
& X_D + X_H \geq 1 \\
& X_E + X_F \geq 1 \\
& X_F + X_G \geq 1 \\
& X_G + X_H \geq 1 \\
& X_E + X_I \geq 1 \\
& X_F + X_I \geq 1 \\
& X_G + X_I \geq 1 \\
& X_H + X_I \geq 1 \\
& X_I + X_J \geq 1 \\
& X_I \leq X_J \\
& X_A + X_B + X_C + X_D \geq 3L_1 \\
& X_E + X_F + X_G + X_H \geq 3L_2 \\
& L_1 + L_2 \geq 1 \\
& X_A = X_F \\
& X_C + X_G \geq 2Y_{CG} \\
& X_G + X_H \geq 2Y_{GH} \\
& X_B + X_F \geq 2Y_{BF} \\
& Y_{CG} + Y_{GH} + Y_{BF} \geq 2
\end{aligned}
$$

Q2. Diet Optimization (Abstract Formulation; 9 points)

Luke would like to optimize his diet to minimize cost while satisfying minimal nutritional requirements, as represented by being within a certain range of intake for each kind of nutrition. The first table below shows his favorite foods, as well as the nutrition information per serving and cost per serving. The second table shows the upper and lower bounds on his total daily intake of each nutrient.

Foods	Calories	Protein	Fat	Sodium	Costs
1. ice cream	330	8	15	180	\$1.59
2. chicken	420	32	10	300	\$2.89
3. pizza	320	15	20	820	\$1.99

Foods	Calories	Protein	Fat	Sodium	Costs
4. fries	380	4	19	270	$1.89
5. macaroni	320	12	10	830	$2.09
6. milk	100	8	2.5	125	$0.89
7. salad	320	31	2	123	$2.49

	Calories	Protein	Fat	Sodium
Lower bound	1800	91	0	0
Upper bound	2200	9999	65	1779

The above table specifies what is the minimum and maximum amount of calories that Luke would like to have in his diet, as well as the amount of protein, fat and sodium. For example, having only one serving of ice cream and one serving of chicken would not give him enough calories, but having 6 servings of ice cream would be enough calories, but that would be too much fat. Luke is okay with fractional servings, but he would like to make sure that the number of servings of each item is either zero, or between 1 and 3. (For example, 0 or 1.2 servings of pizza are okay, but 0.2 or 3.2 are not.)

Write an abstract formulation of a linear optimization model to help him decide his optimal diet. You need to define data variables so as to generalize the problem to arbitrarily large data sets, so your formulation should not reference any of the concrete numbers above. In other words, the numbers in the above tables along with the range of 1 and 3 are only for illustration purposes, but all of these should be encoded by appropriate data variables.

Data:

- I: set of foods.
- J: set of nutrients.
- c_i: cost of food i.
- a_{ij}: amount of nutrient j in one serving of food i.
- l_j: lower bound in daily intake of nutrient j.
- u_j: upper bound of nutrient j.
- s_i: minimum number of servings of food i to have if Luke would have any such food at all.
- t_i: maximum number of servings of food i to have.

Decision Variables:

- X_i: amount of food i in the diet. (continuous)
- Z_i: whether to include food i at all. (binary)

Objective and Constraints:

$$\text{Min.} \quad \sum_{i \in I} c_i X_i$$

s.t.

$$l_j \le \sum_{i \in I} a_{ij} X_i \le u_j \qquad \text{for each nutrient } j \in J.$$

$$s_i Z_i \le X_i \le t_i Z_i \qquad \text{for each food } i \in I.$$

$$X_i \ge 0 \qquad \text{for each food } i \in I.$$

Q3. Feature Selection in Linear Regression (Gurobi Coding; 8 points)

A challenge in predictive analytics is to select a few good features out of many thousands of possibilities, so that a linear regression model estimated using these few features still has good predictive power. In this exercise, you will implement a reusable tool that solves this problem using mixed integer programming.

The input file is of Excel format and has one sheet. Each row represents an observation. The first column corresponds to the dependent variable the user wishes to predict. Each following column corresponds to a feature that may be used for prediction. The name of the dependent variable as well as the features are given as the first row. An sample input file is given below.

	A	B	C	D	E	F	G	H	I
1	Y	X1	X2	X3	X4	X5	X6	X7	X8
2	4.03	0.5	1	-0.06	4	0	-4.5	-0.96	1.43
3	6.93	2.2	0	-0.11	3	1	-1.8	0.38	1.76
4	9.8	1	0	1.37	4	1	14.1	0.03	3.51
5	3.81	0.4	1	-0.1	-1	0	-3.7	0.68	1.67
6	2.03	-0.8	0	-2.43	-2	1	2.8	-1.56	2.05

You should implement the following optimization model. Although the objective function is quadratic, you can just multiply the terms using "$*$" and it will be fine.

Data:

- I: the set of rows (`range(5)` in the example above, but should be inferred from the data).
- J: the set of features ("X1", "X2", $\cdots$, "X8" in the example above, but should be inferred from the data).
- y_i: the value of the dependent variable in row i.
- x_{ij}: the value of feature j in row i.
- k: the maximum number of features used.
- M: a big constant.

Decision Variables:

- α: the coefficient of the constant term. (Continuous)
- β_j: the coefficient for feature j. (Continuous)
- p_i: the predicted value for observation i. (Continuous)
- z_j: whether to use feature j. (Binary)

Objective and Constraints:

$$\begin{aligned}
&\text{Minimize} && \sum_{i\in I}(y_i - p_i)^2 && \\
&\text{s.t} && && \\
&\text{(Linear prediction)} && p_i = \alpha + \sum_{j\in J} x_{ij}\beta_j && \text{for each row } i \in I. \\
&\text{(Big M)} && -Mz_j \le \beta_j \le Mz_j && \text{for each feature } j \in J. \\
&\text{(Using few features)} && \sum_{j\in J} z_j \le k &&
\end{aligned}$$

Note that the variables α, β_j and p_i are NOT constrained to be non-negative.

Write a function called "regress" with three input parameters:

- inputFile: the path to an input file of the above format.
- k: the value of the parameter k (see list of data variables above).
- M: the value of the parameter M (see list of data variables above).

The function should return solve the above model and return a pandas Series object with at most k+1 elements. The first entry has index (label) "Constant" and the value corresponds to the optimal value of α. For the subsequent elements, the index corresponds to the name of the features used, and the value correspond to the estimated coefficient β_j. Only the non-zero coefficients β_j should be included. See below for the Series returned using the input file above, and $k = 2$, $M = 100$.

Write your function below. You must import all packages needed.

[1]:
```
import pandas as pd
from gurobipy import Model, GRB
def regress(inputFile,k,M):
    data=pd.read_excel(inputFile)
    I=data.index
    y=data.iloc[:,0]
    x=data.iloc[:,1:]
    J=x.columns
    mod=Model()
    const=mod.addVar(lb=-GRB.INFINITY)
    beta=mod.addVars(J,lb=-GRB.INFINITY)
    z=mod.addVars(J,vtype=GRB.BINARY)
    p=mod.addVars(I,lb=-GRB.INFINITY)
    mod.setObjective(sum((y.loc[i]-p[i])*(y.loc[i]-p[i]) for i in I))
    for i in I:
        mod.addConstr(p[i]==const+sum(x.loc[i,j]*beta[j] for j in J))
    for j in J:
        mod.addConstr(-M*z[j]<=beta[j])
        mod.addConstr(beta[j]<=M*z[j])
    mod.addConstr(sum(z[j] for j in J)<=k)
    mod.setParam('outputflag',False)
    mod.optimize()
    ans=pd.Series(dtype=float)
    ans['Constant']=const.x
    for j in J:
        if beta[j].x!=0:
            ans[j]=beta[j].x
    return ans
```

[3]:
```
# Test code
coefficients=regress('sample-final-C-input.xlsx',2,100)
coefficients
```

```
Constant    4.084086
X3          2.051074
X5          2.969166
dtype: float64
```