ISyE 6669 HW 14

1. Let three binary variables x_1, x_2, x_3 represent "event i is chosen if $x_i = 1$, and event i is not chosen if $x_i = 0$" for $i = 1, 2, 3$. Write down all feasible binary solutions of the nonlinear constraint $(1 - x_1) \cdot (1 - x_2) \cdot x_3 = x_1$. Then, reformulate the nonlinear constraint using linear constraints to describe the same set of feasible binary solutions.

   Solution: The feasible binary solutions are:

   (0,1,1), (0,1,0), (0,0,0)

   The linear constraints are:

   $$x_1 = 0$$
   $$x_2 \geq x_3$$

2. Write *one* linear constraint to model the **disjunction** "Either event 1 is chosen or event 2 is chosen or event 3 is not chosen". Note that the "or" is inclusive, meaning "either A or B or C" includes the case A and B and C are all true. Use binary variable $x_i = 1$ if event i is chosen, and $x_i = 0$ if event i is not chosen.

   Solution: The single linear constraint is:

   $$x_1 + x_2 + (1 - x_3) \geq 1$$

3. Sharpen your pencil and try to see if you can crack the following Sudoku by hand first. Now write down a binary program to solve the Sudoku. Then code it in Python, solve it, and fill in your answer in the blanks. Hint: Try to define 9 binary variables for each cell of the Sudoku table. Each binary variable is defined for a number between 1, 2, ..., 9. Then, you can write down the logic relations between these binary variables.

   Solution: The final solution for the Sudoku is given below:

8	1	2	7	5	3	6	4	9
9	4	3	6	8	2	1	7	5
6	7	5	4	9	1	2	8	3
1	5	4	2	3	7	8	9	6
3	6	9	8	4	5	7	2	1
2	8	7	1	6	9	5	3	4
5	2	1	9	7	4	3	6	8
4	3	8	5	2	6	9	1	7
7	9	6	3	1	8	4	5	2

   The IP formulation of the problem is as follows:

$$
\begin{aligned}
\min_{i,j,k} \quad & \sum_{i=1}^{9}\sum_{j=1}^{9}\sum_{k=1}^{9} x_{ijk} && (1)\\
s.t. \quad & \sum_{i=1}^{9} x_{ijk} = 1 \forall \quad j = 1, ..., 9, k = 1, ..., 9 && (2)\\
& \sum_{k=1}^{9} x_{ijk} = 1, \forall \quad i = 1, ..., 9, j = 1, ..., 9 && (3)\\
& \sum_{j=1}^{9} x_{ijk} = 1, \forall \quad i = 1, ..., 9, k = 1, ..., 9 && (4)\\
& \sum_{i=1}^{9} x_{ijk} = 1, \forall \quad j = 1, 2, 3, k = 1, 2, 3 && (5)\\
& \sum_{i=1}^{9} x_{ijk} = 1, \forall \quad j = 1, 2, 3, k = 4, 5, 6 && (6)\\
& \sum_{i=1}^{9} x_{ijk} = 1, \forall \quad j = 1, 2, 3, k = 7, 8, 9 && (7)\\
& \sum_{i=1}^{9} x_{ijk} = 1, \forall \quad j = 4, 5, 6, k = 1, 2, 3 && (8)\\
& \sum_{i=1}^{9} x_{ijk} = 1, \forall \quad j = 4, 5, 6, k = 4, 5, 6 && (9)\\
& \sum_{i=1}^{9} x_{ijk} = 1, \forall \quad j = 4, 5, 6, k = 7, 8, 9 && (10)\\
& \sum_{i=1}^{9} x_{ijk} = 1, \forall \quad j = 7, 8, 9, k = 1, 2, 3 && (11)\\
& \sum_{i=1}^{9} x_{ijk} = 1, \forall \quad j = 7, 8, 9, k = 4, 5, 6 && (12)\\
& \sum_{i=1}^{9} x_{ijk} = 1, \forall \quad j = 7, 8, 9, k = 7, 8, 9 && (13)\\
& Fixed \quad constraints \quad here \quad (21 \quad total) && (14)\\
& x_{ijk} \in \{0, 1\}, \quad i = 1, ..., 9, j = 1, ..., 9, k = 1, ..., 9 && (15)
\end{aligned}
$$

The CVXPY version of the solution is below. Please see attached for the PuLP version of the solution.

```
# imports
import pandas as pd
import numpy as np
```

```
import cvxpy as cp

# define 9x9x9 matrix of binary decision variables
# note: cvxpy does not allow for more than 2 dimensions
# use a dictionary to circumvent this issue
x = {}
for i in range(9):
    x[i] = cp.Variable(shape=(9,9), boolean=True)

# definitions to simplify summation code
l = list(range(9))

# define (4) 9x9 matrices for row constraints, col constraints, square constrain
rowConstr = [[x]*9 for x in [None]*9] # each num appears once per row
colConstr = [[x]*9 for x in [None]*9] # each num appears once per col
sqrConstr = [[x]*9 for x in [None]*9] # each square can only contain a single nu
boxConstr = [[x]*9 for x in [None]*9] # each 3x3 box contains one of each num
fixedBoxes = [None]*21 # each num pre-filled
# constraints
constraints = []

# exactly one entry in each square
for row in l:
    for col in l:
        sqrConstr[row][col] = (1 == cp.sum([x[i][row,col] for i in l]))

# each row includes one of each number
for row in l:
    for num in l:
        rowConstr[num][row] = (1 == cp.sum([x[num][row, i] for i in l]))

# each col includes one of each number
for col in l:
    for num in l:
        colConstr[num][col] = (1 == cp.sum([x[num][i, col] for i in l]))

# each box includes all numbers
for num in l:
    boxConstr[0][num] = (1 == cp.sum(x[num][0:3, 0:3])) # top left
    boxConstr[1][num] = (1 == cp.sum(x[num][0:3, 3:6])) # top mid
    boxConstr[2][num] = (1 == cp.sum(x[num][0:3, 6:9])) # top right
    boxConstr[3][num] = (1 == cp.sum(x[num][3:6, 0:3])) # mid left
    boxConstr[4][num] = (1 == cp.sum(x[num][3:6, 3:6])) # mid mid
    boxConstr[5][num] = (1 == cp.sum(x[num][3:6, 6:9])) # mid right
    boxConstr[6][num] = (1 == cp.sum(x[num][6:9, 0:3])) # bot left
    boxConstr[7][num] = (1 == cp.sum(x[num][6:9, 3:6])) # bot mid
    boxConstr[8][num] = (1 == cp.sum(x[num][6:9, 6:9])) # bot right
```

```
# fixed values
fixedBoxes[0] = (x[7][0, 0] == 1)
fixedBoxes[1] = (x[2][1, 2] == 1)
fixedBoxes[2] = (x[5][1, 3] == 1)
fixedBoxes[3] = (x[6][2, 1] == 1)
fixedBoxes[4] = (x[8][2, 4] == 1)
fixedBoxes[5] = (x[1][2, 6] == 1)
fixedBoxes[6] = (x[4][3, 1] == 1)
fixedBoxes[7] = (x[6][3, 5] == 1)
fixedBoxes[8] = (x[3][4, 4] == 1)
fixedBoxes[9] = (x[4][4, 5] == 1)
fixedBoxes[10] = (x[6][4, 6] == 1)
fixedBoxes[11] = (x[0][5, 3] == 1)
fixedBoxes[12] = (x[2][5, 7] == 1)
fixedBoxes[13] = (x[0][6, 2] == 1)
fixedBoxes[14] = (x[5][6, 7] == 1)
fixedBoxes[15] = (x[7][6, 8] == 1)
fixedBoxes[16] = (x[7][7, 2] == 1)
fixedBoxes[17] = (x[4][7, 3] == 1)
fixedBoxes[18] = (x[0][7, 7] == 1)
fixedBoxes[19] = (x[8][8, 1] == 1)
fixedBoxes[20] = (x[3][8, 6] == 1)

# append constraints
for i in fixedBoxes:
    constraints.append(i)
for row in l:
    for col in l:
        constraints.append(sqrConstr[row][col])
        constraints.append(rowConstr[col][row])
        constraints.append(colConstr[col][row])
        constraints.append(boxConstr[row][col])

# define objective function as sum of all decision variables
prob = cp.Problem(cp.Minimize(sum([cp.sum(x[i]) for i in l])), constraints)

prob.solve()
print("Model fitting complete\n-------------------------\nSolution:")
solution = np.zeros(shape=(9,9))
for i in l:
    sol_mask = np.round(x[i].value) > 0.99
    solution[sol_mask] = i+1
print(solution)
print("-------------------------\nEnd")

# Model fitting complete
# -------------------------
# Solution:
```

```
# [[8.  1.  2.  7.  5.  3.  6.  4.  9.]
#  [9.  4.  3.  6.  8.  2.  1.  7.  5.]
#  [6.  7.  5.  4.  9.  1.  2.  8.  3.]
#  [1.  5.  4.  2.  3.  7.  8.  9.  6.]
#  [3.  6.  9.  8.  4.  5.  7.  2.  1.]
#  [2.  8.  7.  1.  6.  9.  5.  3.  4.]
#  [5.  2.  1.  9.  7.  4.  3.  6.  8.]
#  [4.  3.  8.  5.  2.  6.  9.  1.  7.]
#  [7.  9.  6.  3.  1.  8.  4.  5.  2.]]
# ------------------------------
# End
```

4. Consider the problem of investing in 10 projects. The expected profit from project i is p_i dollars. The required investment for project i is c_i. Formulate a binary linear integer program to determine which projects to invest in to maximize expected profit under the following constraints.

   - The investment budget is B.
   - You cannot invest in project 6 unless you invest in **either** projects 2 **or** 4. Here, either A or B also includes the case when both A and B are chosen.
   - If you invest in projects 3 and 9 simultaneously, then you cannot invest in project 10.
   - You can invest in the project 5 **or** project 7, but **not in both**.
   - You can invest in project 8 **if and only if** you invest in projects 1 and 5 simultaneously.

   Solution: Let $x_i \in 0, 1, i = 1, 2, ..., 10$ to denote whether to invest in project i (1 for investing, 0 for not). The the binary linear program can be formulated as follows:

$$
\begin{aligned}
\max_{x,y} \quad & \sum_{i=1}^{10} p_i x_i && (16)\\
s.t. \quad & \sum_{i=1}^{10} c_i x_i \leq B, && (17)\\
& x_6 \leq x_2 + x_4 && (18)\\
& x_{10} \leq 2 - x_3 - x_9, && (19)\\
& x_5 \leq 1 - x_7 && (20)\\
& x_1 \geq x_8, && (21)\\
& x_5 \geq x_8, && (22)\\
& x_8 \geq x_1 + x_5 - 1, && (23)\\
& x_i \in \{0, 1\}, i = 1, 2, ..., 10 && (24)
\end{aligned}
$$

   The formulation need not be unique.