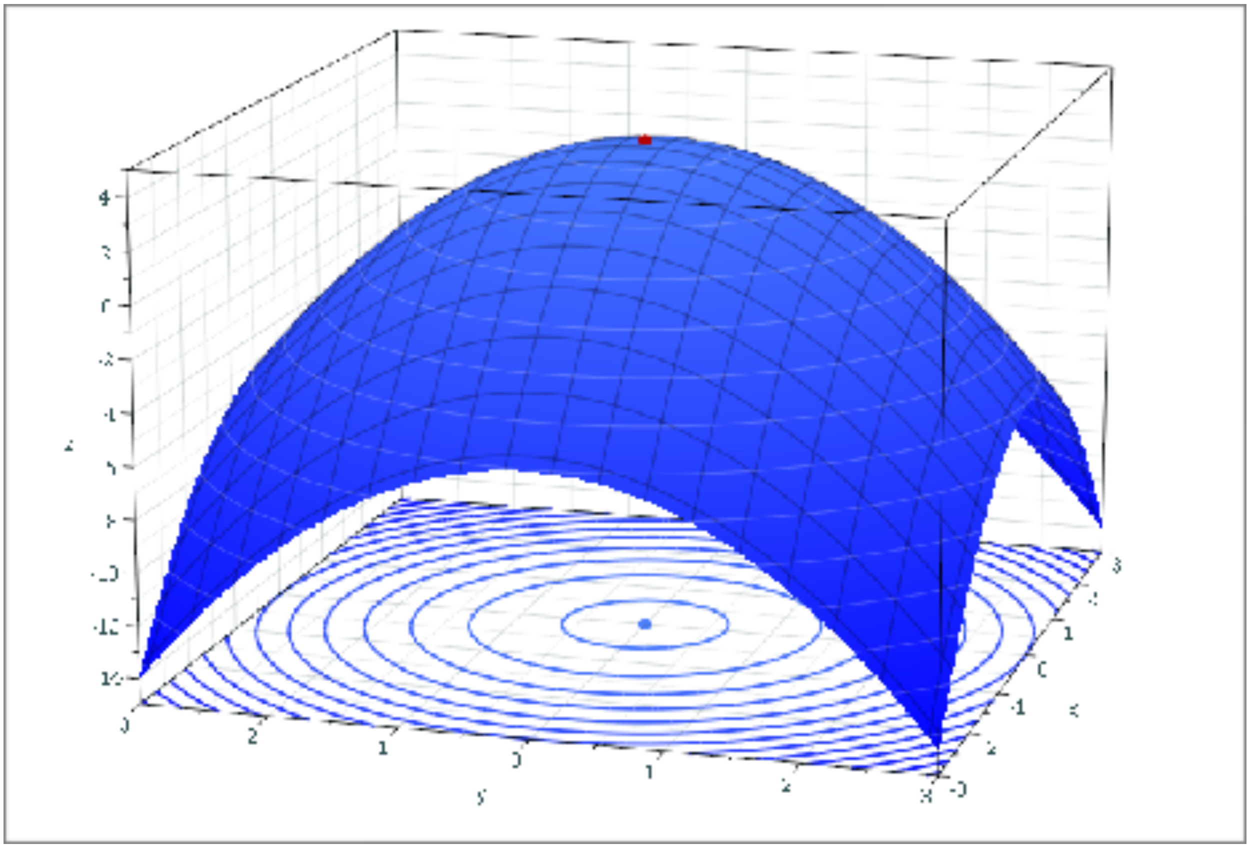
ISYE 4133: Advanced Optimization



Homework 1

Jorge Cruz Serrallés

GT ID: 903131070

jjcs@gatech.edu

Fall 2019

Problem 1: (10 points) Transforms the following linear programs in the standard form.

(a)

$$
\begin{aligned}
\text{minimize } & x_1 + 2x_2 \\
\text{subject to } & \\
& x_1 + x_2 \geq 10 \\
& 2x_1 + 5x_2 \leq 40 \\
& x_1, x_2 \geq 0
\end{aligned}
\quad \Longrightarrow \quad
\begin{aligned}
\text{minimize } & x_1 + 2x_2 \\
\text{subject to } & \\
& x_1 + x_2 - s_1 = 10 \\
& 2x_1 + 5x_2 - s_2 = 40 \\
& x_1, x_2, s_1, s_2 \geq 0
\end{aligned}
$$

(b)

$$
\begin{aligned}
\text{maximize } & x_1 - x_2 \\
\text{subject to } & \\
& x_1 + x_2 \leq 4 \\
& 2x_1 + 5x_2 = 30 \\
& x_1 \leq 0 \\
& x_2 \geq 0
\end{aligned}
\quad \Longrightarrow \quad
\begin{aligned}
\text{minimize } & x_1 + x_2 \\
\text{subject to } & \\
& -x_1 + x_2 + s_1 = 4 \\
& -2x_1 + 5x_2 = 30 \\
& x_1, x_2, s_1 \geq 0
\end{aligned}
$$

Problem 2: (10 points) Implement and solve the linear programs in the above question in the given form as well as the standard form in Gurobi. Note the optimal solutions of the given LP and the LP in its standard form. Show how these solutions are related.

(a)

Original Code:

```
from gurobipy import *

try:

    # Create my model
    m = Model('hw1p1a_regular')

    # Create my variables
    x = m.addVars(2, vtype= GRB.CONTINUOUS, name=['x1','x2'])

    # Set an objective function
    m.setObjective(x[0]+2*x[1], GRB.MINIMIZE)

    # Add regular constraints
    m.addConstr(x[0]+x[1] >= 10)
    m.addConstr(2*x[0]+5*x[1] <= 40)

    # Add non-negativity constraints
    m.addConstrs((x[i] >= 0 for i in range(2)))

    # Optimize model
    m.optimize()

    for v in m.getVars():
        print('%s %g' % (v.varName, v.x))
```

```
    print('Obj: %g' % m.objVal)

except GurobiError as e:
    print('Error code ' + str(e.errno) + ": " + str(e))

except AttributeError:
    print('Encountered an attribute error')
```

Original Output:

```
Optimize a model with 4 rows, 2 columns and 6 nonzeros
Coefficient statistics:
  Matrix range     [1e+00, 5e+00]
  Objective range  [1e+00, 2e+00]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+01, 4e+01]
Presolve removed 2 rows and 0 columns
Presolve time: 0.00s
Presolved: 2 rows, 2 columns, 4 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    0.0000000e+00   2.500000e+00   0.000000e+00      0s
       1    1.0000000e+01   0.000000e+00   0.000000e+00      0s

Solved in 1 iterations and 0.00 seconds
Optimal objective  1.000000000e+01
x1 10
x2 0
Obj: 10
```

Standard Code:

```
from gurobipy import *

try:

    # Create my model
    m = Model('hw1p1a_standard')
```

```python
    # Create my variables
    x = m.addVars(2, vtype= GRB.CONTINUOUS, name=['x1','x2'])
    s = m.addVars(2, vtype= GRB.CONTINUOUS, name=['s1','s2'])

    # Set an objective function
    m.setObjective(x[0]+2*x[1], GRB.MINIMIZE)

    # Add regular constraints
    m.addConstr(x[0]+x[1]-s[0] == 10)
    m.addConstr(2* x[0]+5*x[1]+s[1] == 40)

    # Add non-negativity constraints
    m.addConstrs((x[i] >= 0 for i in range(2)))
    m.addConstrs((s[i] >= 0 for i in range(2)))

    # Optimize model
    m.optimize()

    for v in m.getVars():
        print('%s %g' % (v.varName, v.x))

    print('Obj: %g' % m.objVal)

except GurobiError as e:
    print('Error code ' + str(e.errno) + ": " + str(e))

except AttributeError:
    print('Encountered an attribute error')
```

Standard Output:

```
Optimize a model with 6 rows, 4 columns and 10 nonzeros
Coefficient statistics:
  Matrix range     [1e+00, 5e+00]
  Objective range  [1e+00, 2e+00]
  Bounds range     [0e+00, 0e+00]
```

```
  RHS range         [1e+01, 4e+01]
Presolve removed 4 rows and 2 columns
Presolve time: 0.00s
Presolved: 2 rows, 2 columns, 4 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    0.0000000e+00   2.500000e+00   0.000000e+00      0s
       1    1.0000000e+01   0.000000e+00   0.000000e+00      0s

Solved in 1 iterations and 0.00 seconds
Optimal objective  1.000000000e+01
x1 10
x2 0
s1 0
s2 20
Obj: 10
```

Explanation:

Both answers are quite similar, except for the the second slack variable being used as a 20 point shift.

(b)

Original Code:

```
from gurobipy import *

try:

    # Create my model
    m = Model('hw1a1b_regular')

    # Create my variables
    x1 = m.addVar(name="x1",lb = -GRB.INFINITY, ub = 0.0)
    x2 = m.addVar(name="x2",lb = 0.0)

    # Set an objective function
    m.setObjective(x1-x2, GRB.MAXIMIZE)

    # Add regular constraints
```

```python
    m.addConstr(x1+x2 <= 4)
    m.addConstr(2*x1+5*x2 == 30)

    # Optimize model
    m.optimize()

    print(' worked')
    for v in m.getVars():
        print('%s %g' % (v.varName, v.x))

    print('Obj: %g' % m.objVal)

except GurobiError as e:
    print('Error code ' + str(e.errno) + ": " + str(e))

except AttributeError:
    print('Encountered an attribute error')
```

Original Output:

```
Optimize a model with 2 rows, 2 columns and 4 nonzeros
Coefficient statistics:
  Matrix range     [1e+00, 5e+00]
  Objective range  [1e+00, 1e+00]
  Bounds range     [0e+00, 0e+00]
  RHS range        [4e+00, 3e+01]
Presolve removed 2 rows and 2 columns
Presolve time: 0.00s
Presolve: All rows and columns removed
Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0   -1.0666667e+01   0.000000e+00   0.000000e+00      0s

Solved in 0 iterations and 0.00 seconds
Optimal objective -1.066666667e+01
 worked
x1 -3.33333
x2 7.33333
Obj: -10.6667
```

Standard Code:

```
from gurobipy import *

try:

    # Create my model
    m = Model('hw1a1b_standard')

    # Create my variables
    x = m.addVars(2, vtype= GRB.CONTINUOUS,  name=['x1','x2'],lb
 = 0.0)
    s = m.addVars(1, vtype= GRB.CONTINUOUS, name=['s'],lb = 0.0)

    # Set an objective function
    m.setObjective(x[0]+x[1], GRB.MINIMIZE)

    # Add regular constraints
    m.addConstr(-x[0]+x[1]+s[0] == 4)
    m.addConstr(-2*x[0]+5*x[1] == 30)

    # Optimize model
    m.optimize()

    print(' worked')
    for v in m.getVars():
        print('%s %g' % (v.varName, v.x))

    print('Obj: %g' % m.objVal)

except GurobiError as e:
    print('Error code ' + str(e.errno) + ": " + str(e))

except AttributeError:
    print('Encountered an attribute error')
```

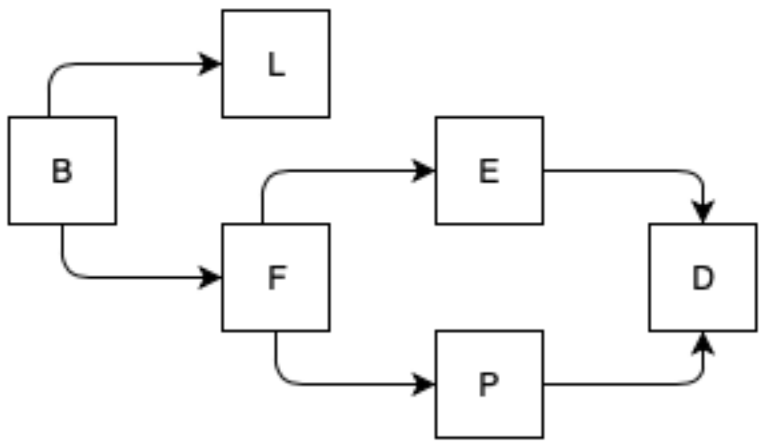
Standard Output:

```
Optimize a model with 2 rows, 3 columns and 5 nonzeros
Coefficient statistics:
  Matrix range     [1e+00, 5e+00]
  Objective range  [1e+00, 1e+00]
  Bounds range     [0e+00, 0e+00]
  RHS range        [4e+00, 3e+01]
Presolve removed 2 rows and 3 columns
Presolve time: 0.00s
Presolve: All rows and columns removed
Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    1.0666667e+01   0.000000e+00   0.000000e+00      0s

Solved in 0 iterations and 0.00 seconds
Optimal objective  1.066666667e+01
 worked
x1 3.33333
x2 7.33333
s[0] 0
Obj: 10.6667
```

Explanation:

The regular and standard form problems have the same absolute objective value output.

Problem 3: (10 points) Formulate a linear program that solves the problem. Explain your formulation. Note, there is no limit on the number of tasks that can be done in parallel. Solve the linear program using Gurobi and report the optimal solution.



Formulation:

$$
\begin{aligned}
\text{minimize } & S_D + d_D \\
\text{subject to} & \\
S_D &\geq S_E + D_E \\
S_D &\geq S_P + D_P \\
S_E &\geq S_F + D_F \\
S_P &\geq S_F + D_F \\
S_F &\geq S_B + D_B \\
S_L &\geq S_B + D_B \\
S_t \geq 0 \forall t &\in \{B, F, E, P, D, L\}
\end{aligned}
$$

Where duration $= d_t = \{3,2,3,4,1,2\} \forall t \in \{B, F, E, P, D, L\}$

Formulation: I optimized for the end time of task d (start time of D + duration time of D) given that it is the last task that can be done, requires the completion of E and P, and is the last task that must be completed before the project is considered completed. If the duration data was random (or had some sort of variance), I would have created a more general objective function.

Problem 4: (10 points) For m $\in$ {10,20,50,100,500,1000,10000} and n $\in$ {10,20,50,100,1000,10000} generate matrices $A \in R^{m\times n}$ whose entries are uniformly random between [0,1]. Similarly generate $b \in R^m$ randomly with entries randomly between [0,1000]. Also generate a cost function $c \in R^n$ with entries randomly between [0, 1000].

(a) Formulate the linear program $\{\min c^T x : Ax \geq b, x \geq 0\}$ for the above random data. Solve 10 instances of the program for each pair of values of m and n with a time out of 2 minutes. Note the time taken and objective value for each run and average over the 10 runs for each pair of (m, n).

Continuous Code:

```
from gurobipy import *
import numpy as np
import csv

with open('problem_4_data.csv', mode='w') as problem_4_data:
    p4_writer = csv.writer(problem_4_data, delimiter=',', quotec
har='"', quoting=csv.QUOTE_MINIMAL)
    p4_writer.writerow(["Rows", "Columns", "Avg Obj Val", "Avg R
T", "Iterations"])
    #Define Proportions


    # m=[10,20,50,100,500,1000,10000]
    # n=[10,20,50,100,1000,10000]
    """
    m == 10000 and n == 10000 were not used due to time constrai
nts
    """
    m=[10,20,50,100,500,1000]
    n=[10,20,50,100,1000]

    for r in m:
```

```
        for c in n:

            outDict = {}

            for i in range(10):
                i += 1

                # Create A
                A = np.random.rand(r,c)*1000
                # Create b
                b = np.random.randint(0,1000,r)
                # Create cost function
                cost = np.random.randint(0,1000,c)
                # Create dictionary for output

                try:

                    # Create my model
                    m = Model('hw1p4a')
                    m.setParam('TimeLimit',
120, 0) # Sets time limit to 2 minutes
                    m.setParam('OutputFlag', 0) # Mutes output

                    # Create my variables
                    x = m.addVars(c, vtype= GRB.CONTINUOUS)

                    # Create objective function expression
                    expression = quicksum(np.transpose(cost)
[j]*x[j] for j in range(c))

                    #Set objective function
                    m.setObjective(expression, GRB.MINIMIZE)

                    # Add regular constraints
                    m.addConstrs(
```

```
                        (quicksum(A[i,j] * x[j] for j in range(c
))
                            >= b[i]
                        for i in range(r)))

                    # Optimize model
                    m.optimize()

                    # Append results to output dictionary
                    outDict[i] = (m.objVal, m.runTime)

                except GurobiError as e:
                    print('Error code ' + str(e.errno) + ": " +
str(e))

                except AttributeError:
                    print('Encountered an attribute error')

            average_objVal = float(sum(v[0] for v in outDict.val
ues()))/float(len(outDict))
            average_runTime = float(sum(v[1] for v in outDict.va
lues()))/float(len(outDict))
            print(
                    "Rows: " + str(r) +
                    ", Cols: " + str(c) +
                    ", Avg Obj: " + str(average_objVal) +
                    ", Average RT: " + str(average_runTime) +
                    ", Iterations: " + str(10) + "\n"
                    )
            p4_writer.writerow([str(r), str(c), str(average_objV
al), str(average_runTime), str(10)])
```

Continuous Output:

Rows	Columns	R x C	Avg Obj Val	Avg RT	Iterations
10	10	100	267.10200758122500	0.00044130047300328130	10
10	20	200	110.86204777630800	0.00054297447204580090	10
10	50	500	73.19295681167900	0.00102500915527344100	10
10	100	1000	37.27019339827524100	0.00087711811065673800	10
10	1000	10000	1.31524896860941800	0.00264210730968770	10
20	10	200	460.59829600281700	0.00033552643836962900	10
20	20	400	89.44367118888790	0.00036120414733888670	10
20	50	1000	53.53874777249660	0.00062456130981445300	10
20	100	2000	30.02700684482770	0.00080893039703369100	10
20	1000	20000	1.50941435470871	0.00324449539184570	10
50	10	500	544.79162118238900	0.00042042732238769500	10
50	20	1000	265.17163027834700	0.00060355663299560600	10
50	50	2500	84.55353273960980	0.00102241039276123000	10
50	100	5000	53.11174726616620	0.00164160723454590	10
50	1000	50000	2.30658642000739	0.01066208911895750	10
100	10	1000	477.69986535075800	0.00050191879272460900	10
100	20	2000	283.32501621971100	0.00075963006744384800	10
100	50	5000	125.37113520660000	0.00155045986175537000	10
100	100	10000	62.77388203973680	0.00324301719665527000	10
100	1000	100000	2.09408416835273	0.03261761665344240	10
500	10	5000	744.61940100063800	0.00212132930755615000	10
500	20	10000	366.24953308778100	0.00318148136138916000	10
500	50	25000	169.08908246645000	0.00685853958129883000	10
500	100	50000	88.22653674865060	0.01696019172668460	10
500	1000	500000	3.59398448796250	0.19888641334259000	10
1000	10	10000	535.02832051830900	0.00396080017089844	10
1000	20	20000	399.44947552835600	0.00713269710540772	10
1000	50	50000	115.25829962305500	0.01647379398345950	10
1000	100	100000	100.94081781187200	0.03492882251739500	10
1000	1000	1000000	0.35078038071109	0.48083478332244900	10

(b) Update the formulation to insist the variables are integers. Repeat the experiment. Note the time taken and objective value for each run.

Integer Code:

```
from gurobipy import *
import numpy as np
import csv

with open('problem_4_data_integer.csv', mode='w') as problem_4_data:
    p4_writer = csv.writer(problem_4_data, delimiter=',', quotechar='"', quoting=csv.QUOTE_MINIMAL)
    p4_writer.writerow(["Rows", "Columns", "Avg Obj Val", "Avg RT", "Iterations"])
    #Define Proportions


    # m=[10,20,50,100,500,1000,10000]
    # n=[10,20,50,100,1000,10000]
    """
    m == 10000 and n == 10000 were not used due to time constraints
    """
    m=[10,20,50,100,500,1000]
    n=[10,20,50,100,1000]

    for r in m:
        for c in n:

            outDict = {}

            for i in range(10):
                i += 1

                # Create A
                A = np.random.rand(r,c)*1000
```

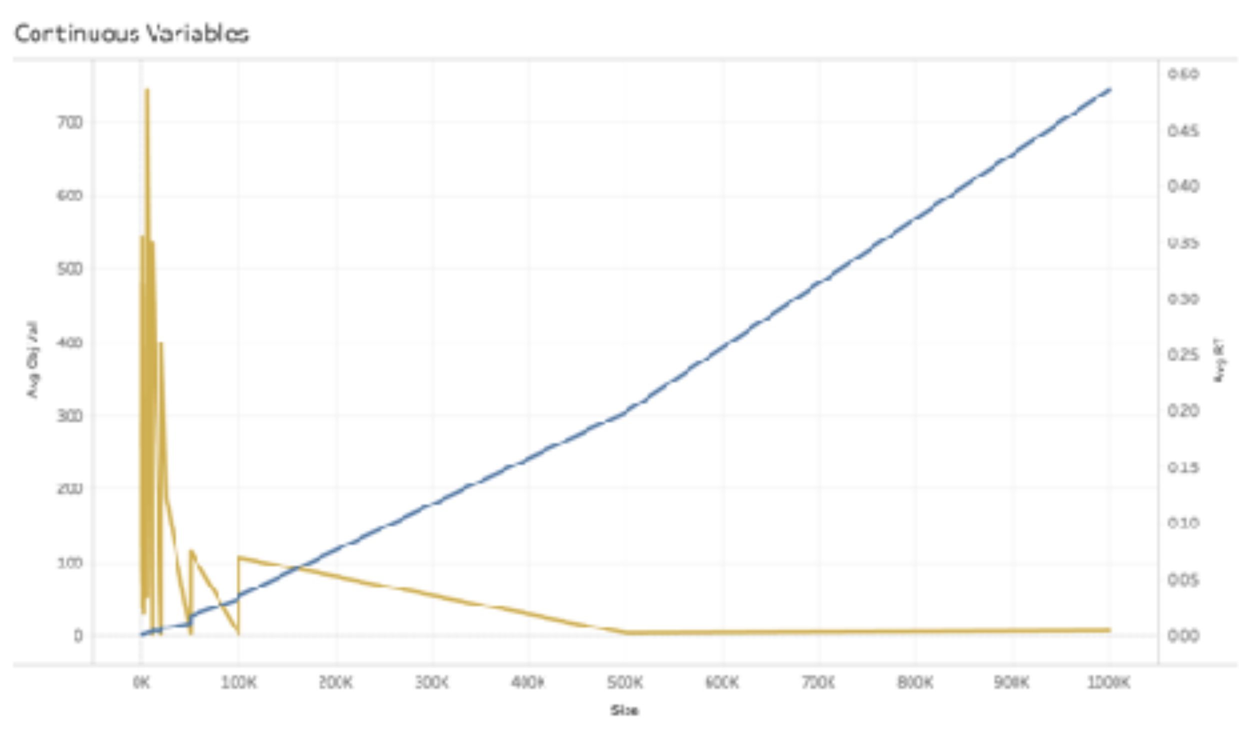
```
                # Create b
                b = np.random.randint(0,1000,r)
                # Create cost function
                cost = np.random.randint(0,1000,c)
                # Create dictionary for output

                try:

                    # Create my model
                    m = Model('hw1p4a')
                    m.setParam('TimeLimit',
120, 0) # Sets time limit to 2 minutes
                    m.setParam('OutputFlag', 0) # Mutes output

                    # Create my variables
                    x = m.addVars(c, vtype= GRB.INTEGER)

                    # Create objective function expression
                    expression = quicksum(np.transpose(cost)
[j]*x[j] for j in range(c))

                    #Set objective function
                    m.setObjective(expression, GRB.MINIMIZE)

                    # Add regular constraints
                    m.addConstrs(
                        (quicksum(A[i,j] * x[j] for j in range(c
))
                            >= b[i]
                        for i in range(r)))

                    # Optimize model
                    m.optimize()

                    # Append results to output dictionary
                    outDict[i] = (m.objVal, m.runTime)
```

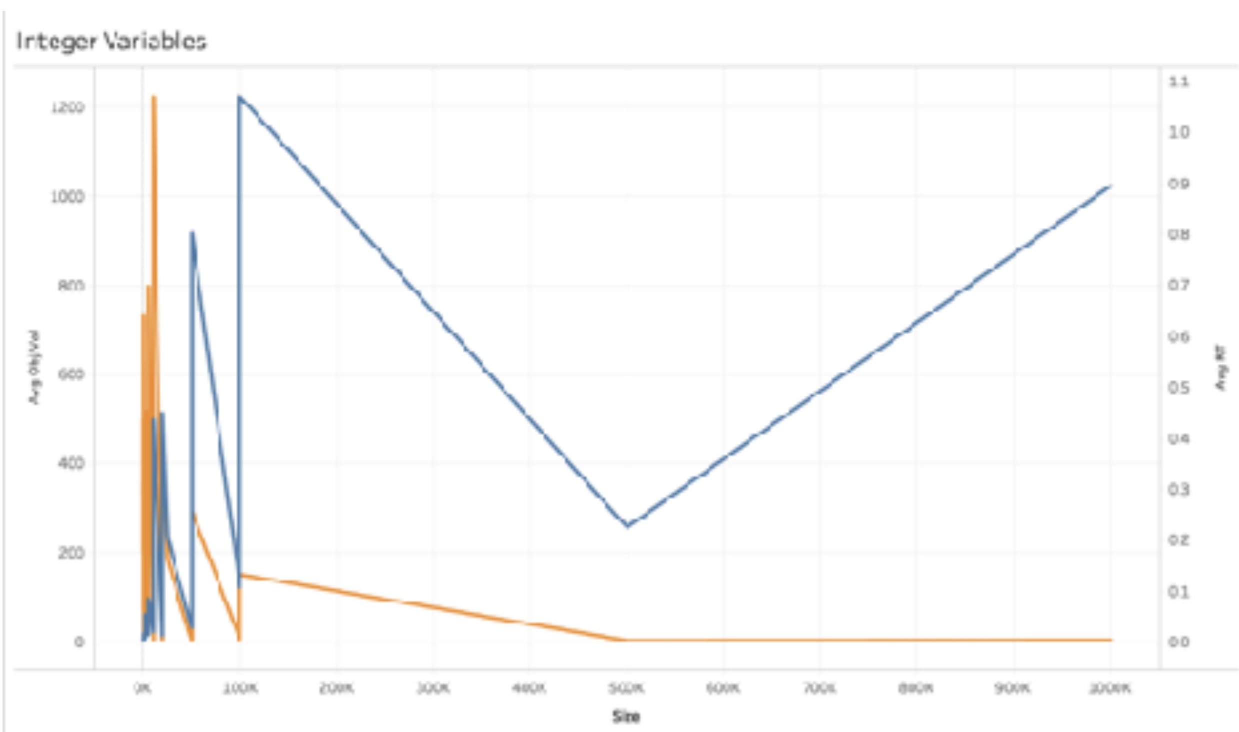
```
                except GurobiError as e:
                    print('Error code ' + str(e.errno) + ": " +
str(e))

                except AttributeError:
                    print('Encountered an attribute error')

            average_objVal = float(sum(v[0] for v in outDict.val
ues()))/float(len(outDict))
            average_runTime = float(sum(v[1] for v in outDict.va
lues()))/float(len(outDict))
            print(
                    "Rows: " + str(r) +
                    ", Cols: " + str(c) +
                    ", Avg Obj: " + str(average_objVal) +
                    ", Average RT: " + str(average_runTime) +
                    ", Iterations: " + str(10) + "\n"
                    )
            p4_writer.writerow([str(r), str(c), str(average_objV
al), str(average_runTime), str(10)])
```

Integer Output:

Rows	Columns	R x C	Avg Obj Val	Avg RT	Iterations
10	10	100	357.1	0.00406896415710449	10
10	20	200	208.3	0.00240321159362793	10
10	50	500	69.3	0.00335636138916016	10
10	100	1000	17.5	0.00216133594512939	10
10	1000	10000	2.6	0.0205343961715698	10
20	10	200	499.8	0.00373208522796631	10
20	20	400	238.5	0.0056162834167480	10
20	50	1000	84.5	0.0056933164596558	10
20	100	2000	45.5	0.0060351591110229	10
20	1000	20000	0.8	0.0132387399673462	10
50	10	500	732.6	0.017433500289917	10
50	20	1000	350.8	0.0187209606170654	10
50	50	2500	123.1	0.0176380157470703	10
50	100	5000	35.1	0.0148109674453735	10
50	1000	50000	1.6	0.028776979446411	10
100	10	1000	410.6	0.0168401002883911	10
100	20	2000	517.9	0.0648672817608643	10
100	50	5000	160.8	0.0546288661956787	10
100	100	10000	67.2	0.0627960038070680	10
100	1000	100000	4.8	0.1074607133865360	10
500	10	5000	799.0	0.066533309868316	10
500	20	10000	324.9	0.264074373324633	10
500	50	25000	184.6	0.204796772281951	10
500	100	50000	137.0	0.368858377686547	10
500	1000	500000	2.8	0.226439001235700	10
1000	10	10000	1224.6	0.486733790632019	10
1000	20	20000	508.7	0.461021838188171	10
1000	50	50000	298.6	0.802328579634216	10
1000	100	100000	160.6	1.07012298107147	10
1000	1000	1000000	8.8	0.6947257518768	10

(c) Plot the time and objective value as the y-axis and size (m + n) as the x-axis.





Problem 5: (10 points) A furniture manufacturing company makes two models of tables for libraries and other university facilities. Both models use the same table tops but model A has 4 short (18-inch) legs and model B has 4 longer ones (30 inches). It takes 0.10 labor hour to cut and shape a short leg from stock, 0.15 labor hour to do the same for a long leg and 0.50 labor hour to produce a tabletop. An additional 0.30 labor hour is needed to attach the set of leges for either model after all parts are available. Estimated profit is 30 for each model A sold and 45 for each model B sold. Plenty of top material is on hand but the company wants to decide how to use the available 5000 feet of leg stock and 800 labor hours to maximize profit assuming that everything made can be sold.

(a) Formulate a LP to choose the optimal plan. Assume that the number of tables and legs manufactured can take fractional values.

$$\begin{aligned} \text{maximize } & 30Q_A + 45Q_B \\ \text{subject to } & \\ 72Q_A + 120Q_B &\leq 60{,}000 \\ (0.4+0.5+0.3)Q_A + (0.6+0.5+0.3)Q_B &\leq 800 \\ Q_A, Q_B &\geq 0 \end{aligned}$$

(b) Solve the linear program using Gurobi.

Code:

```
from gurobipy import *

try:

    # Create my model
    m = Model('hw1p5a')

    # Create my variables
```

```
    q = m.addVars(2, vtype= GRB.CONTINUOUS, name=['qa','qb'])

    # Set an objective function
    m.setObjective(30*q[0]+45*q[1], GRB.MAXIMIZE)

    # Add regular constraints
    m.addConstr(72*q[0] + 120*q[1] <= 60000)
    m.addConstr(1.2*q[0] + 1.4 * q[1] <= 800)

    # Add non-negativity constraints
    m.addConstrs((q[i] >= 0 for i in range(2)))

    # Optimize model
    m.optimize()

    for v in m.getVars():
        print('%s %g' % (v.varName, v.x))

    print('Obj: %g' % m.objVal)

except GurobiError as e:
    print('Error code ' + str(e.errno) + ": " + str(e))

except AttributeError:
    print('Encountered an attribute error')
```

Output:

```
Optimize a model with 4 rows, 2 columns and 6 nonzeros
Coefficient statistics:
  Matrix range     [1e+00, 1e+02]
  Objective range  [3e+01, 4e+01]
  Bounds range     [0e+00, 0e+00]
  RHS range        [8e+02, 6e+04]
Presolve removed 2 rows and 0 columns
Presolve time: 0.00s
Presolved: 2 rows, 2 columns, 4 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    2.5000000e+04   3.659990e+02   0.000000e+00      0s
       2    2.3333333e+04   0.000000e+00   0.000000e+00      0s

Solved in 2 iterations and 0.00 seconds
Optimal objective  2.333333333e+04
qa 277.778
qb 333.333
Obj: 23333.3
```

(c) Can you justify the assumption that the variables can take fractional values?

Yes! You can partially build a table. If these constraints were for a calendar month (say, January), then it would be optimal to use as much of our time and stock resources to build as many tables as possible. Any table partially built could be finished in February.