Caleb Logemann
MATH 566 Discrete Optimization
Homework 3

1. Show that

$$A\mathbf{x} = \mathbf{b} \text{ has a nonnegative solution iff } \forall \mathbf{y} \in \mathbb{R}^m \text{ with } \mathbf{y}^T A \geq \mathbf{0}^T \text{ implies } \mathbf{y}^T\mathbf{b} \geq 0$$

implies

$$A\mathbf{x} \leq \mathbf{b} \text{ has a nonnegative solution iff } \forall \mathbf{y} \in \mathbb{R}^m,\ \mathbf{y} \geq \mathbf{0} \text{ with } \mathbf{y}^T A \geq \mathbf{0}^T \text{ implies } \mathbf{y}^T\mathbf{b} \geq 0.$$

Proof. First let me rewrite the original theorem in different notation for clarity. Let $C \in \mathbb{R}^{m \times q}$ and$\mathbf{d} \in \mathbb{R}^m$, then $C\mathbf{u} = \mathbf{d}$ has a nonnegative solution if and only if for all $\mathbf{v} \in \mathbb{R}^m$ with $\mathbf{v}^T C \geq \mathbf{0}^T$ then $\mathbf{v}^T\mathbf{d} \geq 0$.

Consider the matrix inequality $A\mathbf{x} \leq \mathbf{b}$ for $A \in \mathbb{R}m \times n$ and $\mathbf{b} \in \mathbb{R}^m$. Let $\mathbf{z} = \mathbf{b} - A\mathbf{x}$, $\mathbf{u} = [\mathbf{x}, \mathbf{z}]^T$, $C = [A, I]$ where I is the $m \times m$ identity matrix. Thus

$$C\mathbf{u} = A\mathbf{x} + I\mathbf{z} = A\mathbf{x} + \mathbf{b} - A\mathbf{x} = \mathbf{b}$$

This transformation is like adding slack variables to a linear program to transform inequality to equality. Note that if $A\mathbf{x} \leq \mathbf{b}$ has a nonnegative solution, then $\mathbf{z} = \mathbf{b} - A\mathbf{x} \geq \mathbf{0}$, thus $\mathbf{u} \geq \mathbf{0}$ so $C\mathbf{u} = \mathbf{b}$ has a nonnegative solution. ALso if $C\mathbf{u} = \mathbf{b}$ has a nonnegative solution, then $\mathbf{u} \geq \mathbf{0}$ and obviously $\mathbf{x} \geq \mathbf{0}$. Therefore $A\mathbf{x} \leq \mathbf{b}$ has a nonnegative solution if and only if $C\mathbf{u} = b$ has a nonnegative solution. Thus we can say that

□

2. Some university in Iowa was measuring the loudness of the fan's screaming during the first touchdown of the local team. The measurements contain loudness in dB and the number of people at the stadium in thousands

# fans	53	55	59	61.5	61.5
dB	90	94	95	100	105

Find a line $y = ax + b$ best fitting the data. There are several different notions of best fitting. Commonly used is least squares that is minimizing $\sum_i (ax_i + b - y_i)^2$. But big outliers move the result a lot (and it is troublesome to do it using linear programming). Use the one that minimizes the sum of differences. That is

$$\sum_i |ax_i + b - y_i|$$

Write a linear program that solves the problem and solve if for the "measured" data.

The linear program's objective function should minimize the sum of the absolute values of the errors. However a linear program's objective function must be linear and therefore can't contain any absolute values funtions. Instead let $|ax_i + b - y_i| \leq e_i$. When $\sum_i e_i$ is minimized, then those inequalities become equalities. Furthermore, $|ax_i + b - y_i| \leq e_i$ can be transformed into two linear inequalities that can act as constraints for the linear program. Those two inequalties are

$$e_i \geq ax_i + b - y_i$$
$$-e_i \leq ax_i + b - y_i$$

Rearranging this inequalites results in

$$-e_i + ax_i + b \leq y_i$$
$$e_i + ax_i + b \geq y_i$$

Thus the linear program becomes

$$\begin{array}{ll} \min & \sum_i e_i \\ \text{s.t.} & -e_i + ax_i + b \leq y_i \quad \forall i \\ & e_i + ax_i + b \geq y_i \quad \forall i \end{array}$$

There are no restrictions on a or b they can be any real number. The values of e_i must be nonnegative, but we do not require a specific constraint for that as the other constraints implies this.

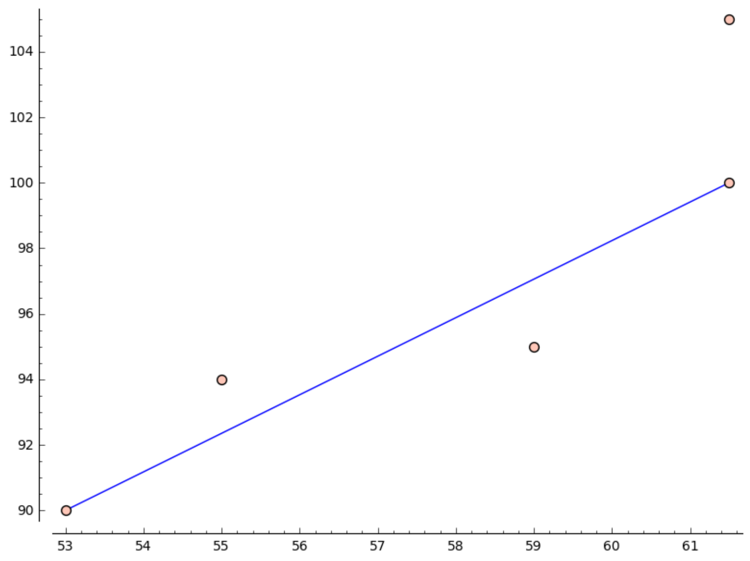
The following implements this linear program for the given data.

```
import numpy as np
# initialize data
x = vector([53, 55, 59, 61.5, 61.5])
y = vector([90, 94, 95, 100, 105])
n = x.length()

# create linear program
milp = MixedIntegerLinearProgram(maximization=False)
u = milp.new_variable(nonnegative=True)
a = milp.new_variable(nonnegative=False)
b = milp.new_variable(nonnegative=False)

# set objective and constraints
ones = matrix(np.full((1,n), 1))
milp.set_objective((ones*u)[0])
milp.add_constraint(identity_matrix(n)*u + b[0] + a[0]*x >= y)
milp.add_constraint(-identity_matrix(n)*u + b[0] + a[0]*x <= y)

print('Objective Value: {}'.format(milp.solve()))
a1 = milp.get_values(a[0])
b1 = milp.get_values(b[0])
print('a = {}'.format(a1))
print('b = {}'.format(b1))
print('Errors')
for i, v in milp.get_values(u).iteritems():
    print('u_%s = %s' % (i, v))

sp = scatter_plot(zip(x, y))
z = var('z')
lp = plot(a1*z + b1, (z, min(x), max(x)))
show(sp + lp)
```

The out put of the previous code is

```
Objective Value: 8.70588235294
a = 1.17647058824
```

```
b = 27.6470588235
Errors
u_0 = 0.0
u_1 = 1.64705882353
u_2 = 2.05882352941
u_3 = 0.0
u_4 = 5.0
```



3. Suppose you are preparing a schedule for classes. You have fixed number of classes and students. Every student told you which classes (s)he wants to attend. However, you do not have enough time slots to run all classes sequentially so you need to make some classes run in parallel. Create a schedule and argue why it is the best schedule in the sense that people as few conflicts as possible. You should be able to justify the optimality of the schedule in some sense. Make schedule with 3 and with 4 timeslots. Assume that there is no limit on how many classes can run in one timeslot. Use the following data

```
[[0,0,0,1,0,1,0,1,0,0,1,0,1,0,0,1,1,1,1,1,1,1,1,0,1,1,1,1,0,0,1,0,0,0,1,1,1,1,1,1,0,1,1,1,1,1,1,0,0,1,0,1,0,0],
[1,1,1,1,0,1,0,1,0,0,0,1,1,1,0,0,0,0,0,0,1,0,1,0,0,0,0,0,1,1,0,1,0,0,1,0,0,0,1,0,1,0,1,0,0,0,0,1,0,0,1,1,1,0],
[1,0,1,0,1,0,1,0,1,1,1,1,0,0,1,1,1,1,1,1,0,0,0,1,1,1,0,1,0,0,0,1,0,0,1,1,0,1,1,1,0,1,1,1,1,1,0,0,0,1,1,0,0,0],
[0,0,1,0,0,0,1,0,0,1,1,0,0,0,0,1,0,0,0,0,0,0,0,0,1,1,0,0,0,1,0,0,1,0,0,0,1,0,0,0,0,0,0,0,1,1,1,0,1,1,0,0,0,0],
[1,0,0,1,0,1,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,1,0,1,0,0,1,0,1,0,1,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0],
[0,1,1,1,1,0,0,0,1,1,1,1,1,0,1,0,1,1,0,1,0,0,0,0,0,1,0,0,0,1,1,0,0,1,1,0,1,1,1,1,0,0,0,1,1,1,1,0,0,1,0,1,0,1],
[0,0,1,1,0,1,1,0,0,1,1,1,0,1,1,0,1,1,0,1,0,1,1,0,0,1,1,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,1,1,0,1,1,0,0,1,0,1,1,0],
[1,0,0,0,0,1,1,0,1,0,0,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,1,1,0,0,0,1,0,0,0,1,0,0,0,0,0,0,1,0,1,0,1,0,1,0,0,0,0,0],
[0,1,0,0,0,1,1,0,1,0,1,0,0,0,0,1,1,1,0,1,0,1,1,1,1,1,0,0,0,0,1,0,1,0,0,0,1,0,1,1,0,0,1,1,1,0,1,0,0,0,0,1,0,1],
[1,1,1,1,0,1,0,1,0,0,0,1,1,1,0,0,0,0,0,0,1,0,1,0,0,0,0,0,1,1,0,1,0,0,1,0,0,0,1,0,1,0,1,0,0,0,0,1,0,0,1,1,1,0],
[1,0,1,0,1,0,1,0,1,1,1,1,0,0,1,1,1,1,1,1,0,0,0,0,1,1,0,1,0,0,0,1,0,0,1,1,0,1,1,1,0,1,1,1,1,1,0,0,0,1,1,0,0,0],
[0,0,1,0,0,0,1,0,0,1,1,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,1,1,0,1,1,0,0,0,0],
[1,0,0,1,0,1,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,1,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0]]
```

Rows corresponds to one class, columns corresponds to one students. 1 means the student wants to attend the class.

Depending on the number of timeslots and the number of classes a student wants to take there is a maximum number of classes that a student can actually attend. For example if there are 4 timeslots and the student wants to take 7 classes the maximum number of classes that they can take is 4. If there is a class that this student wants to take in all 4 timeslots then there really isn't any conflict, because the student can fill up their schedule with classes. On the other hand if there are 4 timeslots and the student only wants to take 2 different classes, then the maximum number of classes that the student can take is 2.

This observation leads me to take the approach of maximizing the number of timeslots that a student can take a class they are interested in for each student. In order to do this I will introduce some notation. Let $s_{ij} = 1$ if student j wants to take class i and $s_{ij} = 0$ if student j does not want to take class i. Thus s_{ij} is the ij entry of the given data. Let c_{ij} be a binary variables, where $c_{ij} = 1$ if class i is scheduled for timeslot j, and $c_{ij} = 0$ otherwise. Let w_{ij} be a binary variable, where $w_{ij} = 1$ if student i wants to take a class offered in timeslot j. Otherwise $w_{ij} = 0$, if there is no class in timeslot j that student i wants to take. Lastly let n_t be the number of timeslots, n_c be the number of classes, and n_s be the number of students. Since we are using binary variables, this implies the linear program will in fact be an integer linear program.

Thus the objective function of our linear program will be maximizing

$$\sum_{i=1}^{n_s}\sum_{j=1}^{n_t} w_{ij}.$$

This double sum is the total number of class periods for all students which contains a class that they want to take.

The following constraints are also necessary. First every class can only be schedule once, so

$$\sum_{j=1}^{n_t}(c_{ij}) = 1$$

This means that each class can only be in one timeslot.

We also need constraints to enforce the definition of the binary variables w_{ij}. In other words the variables w_{ij} depends on the variables c_{ij}. Let n_{ij} be the total number of classes student i is interested in taking during timeslot j. This can be calculated as

$$n_{ij} = \sum_{k=1}^{n_c} (s_{ki} c_{kj})$$

If $n_{ij} \geq 1$, then $w_{ij} = 1$. The constraint

$$n_{ij} \leq M w_{ij}$$

enforces this when M is sufficiently large. This works because w_{ij} is binary and can only be 0 or 1, so if $n_{ij} > 0$, then $w_{ij} > 0$ which implies $w_{ij} = 1$. In this problem M being sufficiently large means that M must be larger than the largest possible value of n_{ij}, which in this case means that $M > n_c$.

If on the other hand $n_{ij} = 0$, then $w_{ij} = 0$. The constraint

$$n_{ij} \geq w_{ij}$$

enforces this condition. If $n_{ij} = 0$, then $w_{ij} \neq 1$ as $0 \ngeq 1$, so $w_{ij} = 0$. This also doesn't contradict the previous constraint as if $n_{ij} = 1$, then $n_{ij} = 1 \geq 1 = w_{ij}$.

Thus the full integer linear program is

$$\begin{array}{lll}
\max * & \sum_{i=1}^{n_s} \sum_{j=1}^{n_t} w_{ij} & \\
\text{s.t.} & \sum_{j=1}^{n_t} (c_{ij}) = 1 & 1 \leq i \leq n_c \\
& n_{ij} \leq M w_{ij} & 1 \leq i \leq n_s \quad 1 \leq j \leq n_t \\
& n_{ij} \geq w_{ij} & 1 \leq i \leq n_s \quad 1 \leq j \leq n_t \\
& 0 \leq c_{ij} \leq 1 & 1 \leq i \leq n_c \quad 1 \leq j \leq n_t \\
& 0 \leq w_{ij} \leq 1 & 1 \leq i \leq n_s \quad 1 \leq j \leq n_t \\
& c_{ij} \in \mathbb{Z} & 1 \leq i \leq n_c \quad 1 \leq j \leq n_t \\
& w_{ij} \in \mathbb{Z} & 1 \leq i \leq n_s \quad 1 \leq j \leq n_t
\end{array}$$

The following sage script implements this integer linear program.

```
nStudents = s.ncols()
nClasses = s.nrows()
nTimeSlots = 3
f(x) = min_symbolic(x, nTimeSlots)
# for each student take the minimum of desired courses and nTimeSlots
maxNumClasses = vector(map(f, sum(A)))

milp = MixedIntegerLinearProgram(maximization=True)
# variable to store what period each class is being held
# c[i,j] = 1 if class i is scheduled for time slot j, 0 otherwise
c = milp.new_variable(binary=True)
# w[i,j] = 1 if student i wants to take at least one class in period j
# w[i,j] = 0 otherwise, student doesn't want to take any classes
# offered in period j
w = milp.new_variable(binary=True)

# objective function will minimize difference of maximum possible
# courses and number classes actually taken
# sum of all w variables
W = 0
```

```
for i in range(nStudents):
    W = W + milp.sum([w[i,j] for j in range(nTimeSlots)])
milp.set_objective(W)

for i in range(nClasses):
    milp.add_constraint(milp.sum([c[i,j] for j in range(nTimeSlots)]) ==
        ↪ 1)

# sufficiently large number, number larger than largest possible,
# nij, number of classes student i wants to take in period j.
M = nClasses + 1
for i in range(nStudents - 1):
    for j in range(nTimeSlots):
        nij = milp.sum([s[k,i]*c[k,j] for k in range(nClasses)])
        # for w[i,j] = 1  if nij >= 1
        milp.add_constraint(nij <= M*w[i,j])
        # for w[i,j] = 0 if nij = 0
        milp.add_constraint(w[i,j] <= nij)

print(str(int(milp.solve())) + '␣out␣of␣' + str(sum(maxNumClasses)))
c = milp.get_values(c)
for j in range(nTimeSlots):
    S = str(j+1) + ':␣'
    for i in range(nClasses):
        if c[i,j] >= 1:
            S = S + 'Class␣' + str(i+1) + ',␣'
    S = S[:-2]
    print(S)
```

Importing the given data as matrix s is not shown. As described at the beginning of the problem there is a maximum number of classes that can be taken depending on the number of timeslots. This is shown as part of the output.

Running this script with 3 timeslots results in

```
157 out of 158
1: Class 3, Class 8, Class 9, Class 10
2: Class 2, Class 4, Class 5, Class 11, Class 12
3: Class 1, Class 6, Class 7, Class 13
```

With 4 timeslots

```
194 out of 202
1: Class 2, Class 7, Class 8, Class 12
2: Class 5, Class 9, Class 11
3: Class 1, Class 6, Class 13
4: Class 3, Class 4, Class 10
```