Homework 5

We will use both tools to solve a section allocation problem! (problem from Convex.jl homepage). The set-up is as follows:

- n students
- m discussion sessions
- Each section should have between 6 to 10 students
- Given an $n \times m$ preference matrix P
  - P_{ij} is student i's ranking for section j
  - 1 means top choice
  - 10000 or a large number means the student can not attend the section
- Goal is to get an allocation matrix X
  - $X_{ij} = 1$ if student i is assigned to section j
  - 0 otherwise
  - Minimize sum(X.*P)

I have provided 3 files: `allocation_JuMP.jl` (see below), `allocation_Convex.jl` (see below), and `data.jl`, which you can download from the course homepage. You do not have to use these files, if you want to you can just copy the text from them and use the IJulia notebook if you prefer that. However, in `data.jl` the preference matrix P is stored, and this file is loaded in the two `allocation_...` files I provided. If you copy this text into the notebook you can skip the `include("data.jl")`-part of the code!

Good Luck!

allocation_JuMP.jl

```
using JuMP

# data.jl has our preference matrix, P
include("data.jl");

# Define the model
m = <WRITE YOUR CODE HERE>

# Define the variables
<WRITE YOUR CODE HERE>

# We want every student to be assigned to exactly one section.
# So, every row must have exactly one non-zero entry
# In other words, the sum of all the columns for every row is 1
# We also want each section to have between 6 and 10 students,
# so the sum of all the rows for every column should be between these

# Specify the constraints
# HINT: you can do it with for-loops of
# the other way I mentioned in class ;)
<WRITE YOUR CODE HERE>


# Our objective is simple sum(X .* P), which can be
# more efficiently represented as vec(X)' * vec(P)
# Since each entry of X is either 0 or 1,
# this is basically summing up the rankings of students
# that were assigned to them.
# If all students got their first choice,
# this value will be the number of students since
# the ranking of the first choice is 1.

# Specify the objective
<WRITE YOUR CODE HERE>

# Solve it
@time status = solve(m)
println("Objective value: ", getObjectiveValue(m))
# print the value
# should also be 65.0 ...
```

allocation_Convex.jl

```
using Convex, GLPKMathProgInterface
# data.jl has our preference matrix, P
include("data.jl");

# Specify the variables
# HINT: check what size P has ;)
X = <WRITE YOUR CODE HERE>

# We want every student to be assigned to exactly one section.
# So, every row must have exactly one non-zero entry
# In other words, the sum of all the columns for every row is 1
# We also want each section to have between 6 and 10 students,
# so the sum of all the rows for every column should be between these

# Specify the constraints
# HINT: you can do it on one line, but it's also
# okay to do it on multiple lines.
# if you wish to add a constraint later on
# you can do so by:
# constraints += ...
constraints = <WRITE YOUR CODE HERE>

# Our objective is simple sum(X .* P), which can be
# more efficiently represented as vec(X)' * vec(P)
# Since each entry of X is either 0 or 1,
# this is basically summing up the rankings of students
# that were assigned to them.
# If all students got their first choice,
# this value will be the number of students since
# the ranking of the first choice is 1.
p = <WRITE YOUR CODE HERE>

# I specified the solver for you :)
@time solve!(p, GLPKSolverMIP())
p.optval
# correct answer is 65.0 ...
```