Introduction to Linear Programming (3/1)

Learning Objectives:

- Define linear programs (LP) and differentiate LPs from other optimization models. (Model)
- Manually solve linear programs with two variables graphically and illustrate the following concepts on the graph: objective function, feasible region, optimal solution, shadow cost, vertex, binding constraint, non-binding constraing. (Analyze)
- Model a production planning problem using linear programming. (Model)

Textbook readings: DMD Chapter 7.1-3, 7.6, 7.8

What is a Linear Program?

A **Linear Program (LP) is an optimization model that maximizes a linear function of the decision variables subject to linear constraints on the decision variables.**

A **linear function** of variables $x_1, x_2, \cdots, x_n$ is one that can be expressed in the form:

$$f(x) = a_0 + \sum_{i=1}^{n} a_i x_i$$

where $a_0, a_1, \cdots, a_n$ are all numerical constants (not variables).

For example, if the variables are x, y, z, the following are linear functions:

- x
- $3x + 5y$
- $2[2(3x + 5y) + 7z + 3] + 4(x + y)$, because it can be simplified to $6 + 16x + 24y + 14z$.

The following are not linear functions:

- xy
- $x^2 + 7y$
- $3(x + y)(x + z) + 5$

A **linear constraint** takes one of the following three forms:

$$LHS \leq RHS$$
$$LHS \geq RHS$$
$$LHS = RHS$$

where both the left hand side (LHS) and right hand side (RHS) are linear functions of decision variables.

For example, if x, y and z are the decision variables, then the following are linear constraints

$$x \geq 0$$
$$5 = x + y$$
$$3x + 5y \leq 5y + 3z + 10$$

However, the following are not linear constraints, because one of the two sides is not a linear function of the decision variables.

$$x^2 \leq 5$$
$$5 = xy$$
$$3x + 5z \leq (5y + 3z)x + 10$$

A linear program cannot have strict constraints like $>$ or $<$, but must be $\geq$ or $\leq$.

Geometry of Linear Programs

See DMD section 7.3 and the video recording of the lecture. You must learn to draw the feasible region of a linear program with 2 variables by hand and be able to illustrate the following on the graph.

- **direction of objective function**: which direction corresponds to increasing objective?
- **feasible region**: set of decision variables satisfying all constraints.
- **set of optimal solutions**: which points in the feasible region yield the highest objective value?
- **vertex**: a "corner" of the feasible region.
- **binding constraint**: a constraint in which at the optimal solution, the LHS and RHS are equal.
- **non-binding constraint**: a constraint in which at the optimal solution, the LHS and RHS are not equal.
- **shadow price** of a constraint: the change in the optimal objective value when the right hand side (RHS) of the constraint is increased by one unit.

Generic Production Planning Problem

The following generalizes the example given in class and in DMD section 7.3. The purpose is to illustrate how to succinctly express large linear programs using index and summation notation. Once expressed in this way, it is straightforward to later create large linear programs in Python using for loops.

Suppose we are a plant able to produce n products, each of which takes one of m resources. Let $I = \{1, 2, \cdots, n\}$ be the set of products and $J = \{1, 2, \cdots, m\}$ be the set of resources. The amount of resource j product i requires is a_{ji}. (Hence a_{21} is the amount of resource 2 that product 1 requires.) To capacity for resource j is c_j. The profit we make from producing each unit of product i is p_i.

Suppose we produce x_i amount of product each $i \in I$. The goal is to maximize the total profit, which is $\sum_{i=1}^{n} p_i x_i$. The constraints are as follows:

- for each resource, we cannot use more than what we have, so $\sum_i^n a_{ji} x_i$, which is the amount of resource j we use, must not exceed the capacity c_j.
- we cannot make negative amounts of any product, so $x_i \geq 0$ for every product j.

The **linear program formulation** is as below. (A LP formulation is a way of expressing the optimization problem as a linear program, with the objective, the constraints, and the decision variables clearly written out.)

Decision variables: $x_1, x_2, \cdots, x_n$, where x_i is the amount of product i to produce.

$$
\begin{array}{lll}
\text{Maximize:} & \displaystyle\sum_{i=1}^{n} p_i x_i & \\
\text{Subject to:} & \displaystyle\sum_{i=1}^{n} a_{ji} x_i \leq c_j & \text{for every resource } j \in J, \\
& x_i \geq 0 & \text{for every product } i \in I.
\end{array}
$$

Installing Gurobi

Step 1 (Installing Gurobi via conda):

In Anaconda prompt (Windows) or in a terminal (Mac or Linux), type:

```
conda install -c http://conda.anaconda.org/gurobi gurobi
```

Type yes for everything to install.

Step 2 (Obtaining license):

To use Gurobi, you need to obtain a **free academic license** for your computer. To do so, following steps 2-4 of the instructions on the following webpage, under the heading "***To obtain a free named-user academic license***": http://www.gurobi.com/academia/for-universities. Note that after registering on the Gurobi website using your USC.edu email address, you also need to run the command `grbgetkey` followed by the license key, as indicated in step 4 of the above instructions.

Step 3: (Test import):

In Anaconda prompt (Windows) or terminal (Mac or Linux), start a Python shell using

```
python
```

Once the shell starts, type the following and see it there is an error.

```
import gurobipy
```

If there is no error, then you move on to the following session to test solving a LP in Gurobi within Jupyter notebook. If there is an error, try to Google the error message and "Gurobi" together to see if you can figure out what's wrong. Come to office hour (Friday 3:30-5:30pm) or email the professor for help.

If the error message is `Segmentation Fault` and you are using a Mac, go to the following step.

Step 4: (Only for Mac users who obtained the Segmentation Fault Error above)

Unfortunately, there is a bug with the latest version of Anaconda for Python 3, which makes it unable to use Gurobi. This bug has not been resolved since Fall 2017. See discussion here.

The simplest work around unfortunately is to create a new Anaconda environment that uses Python 2.7 and install Gurobi in that environment. If you follow the instructions below, you can avoid the Python 2.7 conflicting with the Python 3.6 that you already have. **DO NOT DOWNLOAD ANACONDA 2.7 FROM THE WEBSITE AND INSTALL THE PACKAGE THAT WAY. IT WILL MESS UP YOUR SYSTEM.**

Creating a new environment using Python 2.7: Type in a terminal the following three lines, answering `yes` whenever it prompts you. The first line adds the Gurobi package to the conda search path. The second creates a new environment called `py27` (feel free to change the name), with Python version 2.7, as well as the packages you need, which are jupyter, pandas, scipy, numpy, and gurobi. The third line activates the environment you created.

```
conda config --add channels http://conda.anaconda.org/gurobi
conda create -n py27 python=2.7 jupyter pandas scipy numpy gurobi
source activate py27
```

If the first line results in a "Permission denied error", then you can replace the above three lines with

```
conda create -n py27 python=2.7 jupyter pandas scipy numpy
source activate py27
conda install -c http://conda.anaconda.org/gurobi gurobi
```

Working with environments: When you start a new terminal, you are in the default environment with Python 3.6 without gurobi. When you activate the environment using `source activate py27`, the prefix `py27` will appear in your terminal, showing that you are now working in the `py27` environment. Only when you start Python or Jupyter notebook when you see this prefix, will Gurobi work. To go back to the original Python 3.6 environment (without Gurobi), you can type

```
source deactivate
```

Testing Gurobi again in Python 2.7: Now, activate the `py27` environment by typing `source activate py27` in terminal, and go back to step 3 to test. Then proceed to the next section to test a bigger chunk of code. (When you test the LP in the next section, make sure you start jupyter notebook in the new environment, by typing:

```
source activate py27
jupyter notebook
```

Email the professor if you have problems.

Testing Gurobi

IMPORTANT: Before next class, please make sure that your Gurobi works by copy/pasting the code in the cell below into Jupyter notebook and seeing if the output is the same. (Please come to office hour or email me before next class if any issue arises. I will not help you debug during class next session.)

The following code solves the Gemstone Tool Company LP from DMD section 7.3.

Maximize :	$130W + 100P$	
subject to:	$1.5W + P \leq 27$	(Constraint on steel)
	$W + P \leq 21$	(Constraint on molding)
	$0.3W + 0.5P \leq 9$	(Constraint on assembly)
	$W \leq 15$	(Limit on demand for wrenches)
	$P \leq 16$	(Limit on demand for pliers)
	$W, P \geq 0$	(Non-negativity)

[1]:
```
import gurobipy as grb

mod=grb.Model()                          # Defining model
W=mod.addVar(lb=0,name='W')             # Defining variable W with lower bound (lb) 0.
P=mod.addVar(lb=0,name='P')
c1=mod.addConstr(1.5*W+P<=27,'Steel')  # Adding each constraint and naming it (the name
c2=mod.addConstr(W+P<=21,'Molding')
c3=mod.addConstr(0.3*W+0.5*P<=9,'Assembly')
c4=mod.addConstr(W<=15,'W-Demand')
c5=mod.addConstr(P<=16, 'P-Demand')

# Setting the objective function and specifying that we are maximizing it
mod.setObjective(130*W+100*P,sense=grb.GRB.MAXIMIZE)
```

```
    # Calling the Gurobi solver to solve the model we inputed.
    mod.optimize()

    print('\n-------RESULTS--------------')
    print('Optimal Objective: ',mod.ObjVal)
    print ('Optimal W=',W.x)
    print ('Optimal P=',P.x)
    print('Shadow price of constraint c1=',c1.PI)
    print('Shadow price of constraint c2=',c2.PI)
    print('Shadow price of constraint c3=',c3.PI)
    print('Shadow price of constraint c4=',c4.PI)
    print('Shadow price of constraint c5=',c5.PI)
```

```
Optimize a model with 5 rows, 2 columns and 8 nonzeros
Coefficient statistics:
  Matrix range     [3e-01, 2e+00]
  Objective range  [1e+02, 1e+02]
  Bounds range     [0e+00, 0e+00]
  RHS range        [9e+00, 3e+01]
Presolve removed 2 rows and 0 columns
Presolve time: 0.02s
Presolved: 3 rows, 2 columns, 6 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    2.7000000e+03   3.250000e+00   0.000000e+00      0s
       3    2.4600000e+03   0.000000e+00   0.000000e+00      0s

Solved in 3 iterations and 0.03 seconds
Optimal objective  2.460000000e+03

-------RESULTS--------------
Optimal Objective:  2460.0
Optimal W= 12.0
Optimal P= 9.0
Shadow price of constraint c1= 60.0
Shadow price of constraint c2= 40.0
Shadow price of constraint c3= 0.0
Shadow price of constraint c4= 0.0
Shadow price of constraint c5= 0.0
```

```
[2]: mod.write('example.lp') # Write model into a file (note extension)
     mod2=grb.read('example.lp') # Read from lp file
     mod2.optimize()            # Solving the new model
```

```
Optimize a model with 5 rows, 2 columns and 8 nonzeros
Coefficient statistics:
  Matrix range     [3e-01, 2e+00]
  Objective range  [1e+02, 1e+02]
  Bounds range     [0e+00, 0e+00]
  RHS range        [9e+00, 3e+01]
```

```
Presolve removed 2 rows and 0 columns
Presolve time: 0.02s
Presolved: 3 rows, 2 columns, 6 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    2.7000000e+03   3.250000e+00   0.000000e+00      0s
       3    2.4600000e+03   0.000000e+00   0.000000e+00      0s

Solved in 3 iterations and 0.03 seconds
Optimal objective  2.460000000e+03
```