# Math 6367 Optimization 2 HW 01 Prof. Dr. Ronald H.W. Hoppe

**20 March 2019**
**Jonathan Schuba**

## Assignment:

Consider the following two-stage stochastic linear program which represents an investment decision problem in two resources, $x_1$ and $x_2$ which are needed in the second stage to cover at least 80% of the demand $\xi$.

$$\begin{aligned} \text{minimize} \quad & 3x_1 + 2x_2 + E_\xi(\min(-15y_1, -12y_2) \\ \text{subject to} \quad & 3y_1 + 2y_2 \le x_1 \\ & 2y_1 + 5y_2 \le x_2 \\ & 0.8\xi_1 \le y_1 \le \xi_1 \\ & 0.8\xi_2 \le y_2 \le \xi_2 \\ & x_1, x_2, y_1, y_2 \ge 0 \end{aligned}$$

Where $(\xi_1, \xi_2) = (4,4), (4,8), (6,4)$, or $(6,8)$ with probability $1/4$ each.

Formulate the two-stage stochastic linear program in standard form and solve it by the L-shaped method starting from $x = 0$.

## Formulation of the two-stage problem

The standard form required by the L-shaped method is the following.

$$\begin{aligned} \text{minimize} \quad & c^\top x + Q(x) \\ \text{subject to} \quad & Ax = b \\ & x \ge 0 \end{aligned}$$

where

$$\begin{aligned} Q(x) &= E_\xi(Q(x, \xi(\omega))) \\ Q(x, \xi(\omega)) &= \min_y \{ q(\omega)^\top y \mid Wy = h(\omega) - T(\omega)x \ , \ y \ge 0 \} \end{aligned}$$

We note that there are four different realizations of our random vector $(\xi_1, \xi_2)$. Therefore, $q(\omega), h(\omega)$ and $T(\omega)$ may have four different versions, which we will have to associate with each realization of the random vector.

For example, consider the last realization $(\xi_1, \xi_2) = (6, 8)$. The second stage problem

$$Q(x, \xi(\omega)) = \min_y \{ q(\omega)^\top y \mid Wy = h(\omega) - T(\omega)x \ , \ y \geq 0 \}$$

becomes

$$\begin{aligned} \underset{y}{\text{minimize}} \quad & q(\omega_4)^\top y \\ \text{subject to} \quad & Wy = h(\omega_4) - T(\omega_4)x \\ & y \geq 0 \end{aligned}$$

where

$$q(\omega_4) = \begin{bmatrix} -15 \\ -12 \end{bmatrix}$$

$$W = \begin{bmatrix} 3 & 2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 2 & 5 & 0 & 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$h(\omega_4) = \begin{bmatrix} 0 \\ 0 \\ -0.8 * 6 \\ -0.8 * 8 \\ 6 \\ 8 \end{bmatrix}$$

$$T(\omega_4) = \begin{bmatrix} -1 & 0 \\ 0 & -1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

This is clearly equivelent to the original problem, with slack variables introduced to convert the inequalities to equalities. We also note the only $h(\omega)$ has any dependence on the realization of the random variables. The $q$ and $T$ are constant, as is $W$.

We now identify the vector $c = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$ and note that there are no first stage equality or inequality constraints, so $A$ and $b$ are none.

## Variables as python input

We will be using Python 3.6 for this assignment, with numpy and cvxpy packages. Numpy is a package for matrix and array math, and cvxpy is a python port of the cvx package for solving convex optimization problems.

This example is relatively easy to input, since the matrices used in the second stage do not depend on any interactions between the current solution iterate and the random variables. However, the example presented in class (from page 188 of the book) has an interesting feature that the matrices $T$ and $h$ change depending on whether the current solution is less than or greater than the random variable being considered. This made us realize that blindly putting matricies into the solver was not adequate. We need a way to programmically change these matrices for each portion of step 2 and 3 of the algorithm.

We implemented a class called L_Shaped_Algorithm which accepts functions, rather than matricies, for $T$ and $h$. The user can supply a function that generates the correct matrix for a given random variable and solution iterate.

In this case, these driver functions for $T$ and $h$ are simple, but in general, they can take in the current iterate and random variable realization, and return the correct matrix.

```
In [1]: import numpy as np

        c = np.array([3,2])

        W = np.array([[3, 2,1,0,0,0,0,0],
                      [2, 5,0,1,0,0,0,0],
                      [-1,0,0,0,1,0,0,0],
                      [0,-1,0,0,0,1,0,0],
                      [1, 0,0,0,0,0,1,0],
                      [0, 1,0,0,0,0,0,1]])


        p = []  # probability for each realization of the random variable
        q = []  # vector q for each realization
        s = []  # random variable values for each realization
        for s1 in [6,4]:
            for s2 in [8,4]:
                p.append(1/4)
                q.append(np.array([-15,-12]))
                s.append(np.array([s1,s2]))

        def T_driver(x, s):
            return np.array([[-1,0],
                             [0,-1],
                             [0,0],
                             [0,0],
                             [0,0],
                             [0,0]])

        def h_driver(x, s):
            return np.array([0, 0, -0.8*s[0], -0.8*s[1], s[0], s[1]])
```

The call to solve the problem is simple, and informative. We initialize an L_Shaped_Algorithm object with the desired properties, and call the solve method on it. In each stage, a simple printout of key information is provided. For more detailed information, set verbose = True.

```
In [2]: from l_shaped_algorithm_cvx import L_Shaped_Algorithm

        Solver = L_Shaped_Algorithm(c = c,
                                    A_eq = None,
                                    b_eq = None,
                                    A_ineq = None,
                                    b_ineq = None,
                                    W = W,
                                    h_driver = h_driver,
                                    T_driver = T_driver,
                                    q = q,
                                    realizations = s,
                                    probabilities = p,
                                    max_iter = 100,
                                    precision=10e-6,
                                    verbose=False, debug=False)
        x_opt = Solver.solve()
```

```
===========================================
=============== Iteration 1 ===============
===========================================
----------------- Step 1 -----------------
objective value =  -0.0
x_nu            =  [-0. -0.]
theta_nu        =  -inf

----------------- Step 2 -----------------
Feasibility cut identified
objective      =  11.2
dual objective =  11.2
dual variables =  [-0.3937  -0.34414 -1.      -1.       0.      -0.     ]
Dk =  [0.3937  0.34414]
dk =  11.2

===========================================
=============== Iteration 2 ===============
===========================================
----------------- Step 1 -----------------
objective value =  65.09004
x_nu            =  [ 0.      32.54502]
theta_nu        =  -inf

----------------- Step 2 -----------------
Feasibility cut identified
objective      =  11.2
dual objective =  11.2
dual variables =  [-0.66654  0.      -1.      -1.       0.       0.     ]
Dk =  [ 0.66654 -0.     ]
dk =  11.2
```

```
===========================================
=============== Iteration 3 ===============
===========================================
----------------- Step 1 -----------------
objective value =  77.05383
x_nu            =  [16.8032  13.32212]
theta_nu        =  -inf

----------------- Step 2 -----------------
Feasibility cut identified
objective      =  5.65558
dual objective =  5.655576
dual variables =  [-0.  -0.2 -0.4 -1.  -0.   0. ]
Dk =  [0.  0.2]
dk =  8.32

===========================================
=============== Iteration 4 ===============
===========================================
----------------- Step 1 -----------------
objective value =  133.60959
x_nu            =  [16.8032 41.6   ]
theta_nu        =  -inf

----------------- Step 2 -----------------
Feasibility cut identified
objective      =  3.4656
dual objective =  3.465601
dual variables =  [-0.33333 -0.      -1.      -0.66667  0.      -0.     ]
Dk =  [0.33333 0.     ]
dk =  9.06667

===========================================
=============== Iteration 5 ===============
===========================================
----------------- Step 1 -----------------
objective value =  164.8
x_nu            =  [27.2 41.6]
theta_nu        =  -inf

----------------- Step 2 -----------------
No feasibility cuts needed

----------------- Step 3 -----------------
Optimality cut made
E =  [1.01911 1.08259]
e =  -61.1045
```

```
===========================================
=============== Iteration 6 ===============
===========================================
----------------- Step 1 -----------------
objective value =  30.94
x_nu            =  [27.2 41.6]
theta_nu        =  -133.86

----------------- Step 2 -----------------
No feasibility cuts needed

----------------- Step 3 -----------------

Optimal Solution Found

Objective Value  =  30.94
Optimal Solution =  [27.2 41.6]
```