ME596 Midterm

February 2, 2017

0.1 Problem 1.

We can restate the problem mathematically as

$max \quad p = -0.04h^2 - 4(0.02)h + (0.10)blh$ subject to the contraints:$b + 2h = 17l + 2h = 22b, l, h > 0.$

There are three design variables, however we can make the following substitutions:

$$b = 17 - 2hl = 22 - 2h,$$

which when expanded by order and simplified yeilds

$$p = -0.4h^3 - 7.84h^2 + 37.32h.$$

If we apply the neccesary condition

$$\frac{dp}{dh} = 0 = 0.4(3)h^2 - 7.84(2)h + 37.32,$$

and find the roots using the quadratic formula we find that $h^{(1,2)} = [3.130, 9.937]$.
Taking the second derivative and evaluating at the stationary points,

$$\left.\frac{d^2p}{dx^2}\right|_{h=3.130} < 0.$$

Thus we have found the maximum. The function value evaluated at this point is $f = 50.68$ (or 51 cents, a paltry profit for a welded box!). The remaining design variables are, respectively

$$b = 10.74l = 15.27.$$

0.2 Problem 2.

What follows is an attempt at a Simplex algorithem in Python3. The algorithem as implimented requires the problem to already be in cannonical tableu form. Starting from the original statement of the optimization problem

Maximize $\quad x_1 + x_2 + 2x_3$subject to $\quad 2x_1 + x_2 + 2x_3 \leq 8x_1 + x_2 + x_3 \geq 2 - x_1 + x_2 + 2x_3 = 1x_1 \geq 0, x_2 \geq 0, x_3 \geq 0.$

It can be seen that all the variables are defined, so no difference substitutions are neccessary. Furthermore due to the inequality constraints one slack variable and one surplus variable is required. The objective function and constraints may be restated as a cannonical tableu thusly:

f = [0.0, -1.0, -1.0, -2.0, 0.0, 0.0] x1 = [8.0, 2.0, 1.0, 2.0, -1.0, 0.0] x2 = [2.0, 1.0, 1.0, 1.0, 0.0, 1.0] x3 = [1.0, -1.0, 1.0, 2.0, 0.0, 0.0]

where the first column is the **b** vector, and the second through fourth columns represent the coefficient **A** matrix.

In [1]:
```
#First put problem in standard form and construct cannonical matrix
#      ^^ program will NOT do this.

#Note: this is coded in Python3 *not* MATLAB

 #initial feasible solution is given by initial tableu construction

from __future__ import division

def printTableu(tableu):
    print('----------------------------------------------')

    for row in tableu:
        newrow = [ '%2.2f' % elem for elem in row ]
        print(newrow)
    print('----------------------------------------------')
    return


def pivoter(tableu, row, col): #where row and col are the index values of min b[i]/a[i][
    j = 0
    pivot = tableu[row][col]
    for x in tableu[row]: #normalize entire tableu by the pivot value
        tableu[row][j] = tableu[row][j] / pivot
        j += 1

    i = 0
    for xi in tableu: #Gauss-Jordan elimination
        if i != row: #ignore the pivot row
            ratio = xi[col]
            j = 0
            for xij in xi: #subtract by columns
                xij -= ratio * tableu[row][j]
                tableu[i][j] = xij
                j += 1

        i += 1 #move across rows

    return tableu

def simplex(tableu):
     THETA_INFINITE = -1
     optimal = False
```

```
    unbounded  = False
    n = len(tableu[3])
    m = len(tableu) - 1
    while ((not optimal) and (not unbounded)):
        min = 0.0
        pivotCol = j = 0
        while(j < (n-m)): #find min of C[j]
             cj = tableu[3][j]
             if (cj < min) and (j > 0):
                 min = cj
                 pivotCol = j
             j += 1
        if min == 0.0: #if not C[j] < 0 then break the loop
            optimal = True
            continue

        pivotRow = i = 0
        minTheta = THETA_INFINITE
        for xi in tableu:
             if (i > 0):
                  xij = xi[pivotCol]
                  if xij > 0: #to avoid infinite and negative numbers
                      theta = (xi[0] / xij) #test criteria for pivot column -> min b[i]
                      if (theta < minTheta) or (minTheta == THETA_INFINITE):
                          minTheta = theta
                          pivotRow = i

             i += 1
        if minTheta == THETA_INFINITE:
            unbounded = True
            continue

        tableu = pivoter(tableu, pivotRow, pivotCol)

    print('\n Unbounded = {}'.format(unbounded))
    print('Optimal = {} \n'.format(optimal))
    print("Final tableu")
    printTableu(tableu)
    return tableu

f  = [ 0.0, -1.0, -1.0, -2.0, 0.0,  0.0]
x1 = [ 8.0, 2.0,  1.0,  2.0, -1.0,  0.0]
x2 = [ 2.0, 1.0,  1.0,  1.0, 0.0, 1.0]
x3 = [ 1.0, -1.0,  1.0,  2.0, 0.0,  0.0]


tableu = []
tableu.append(x1)
```

```
        tableu.append(x2)
        tableu.append(x3)
        tableu.append(f)

        print("Initial tableu")
        printTableu(tableu)

        tableu = simplex(tableu)
```

```
Initial tableu
----------------------------------------------
['8.00', '2.00', '1.00', '2.00', '-1.00', '0.00']
['2.00', '1.00', '1.00', '1.00', '0.00', '1.00']
['1.00', '-1.00', '1.00', '2.00', '0.00', '0.00']
['0.00', '-1.00', '-1.00', '-2.00', '0.00', '0.00']
----------------------------------------------

 Unbounded = False
Optimal = True

Final tableu
----------------------------------------------
['4.00', '0.00', '-1.00', '0.00', '-1.00', '-2.00']
['2.00', '1.00', '1.00', '1.00', '0.00', '1.00']
['3.00', '0.00', '2.00', '3.00', '0.00', '1.00']
['2.00', '0.00', '0.00', '-1.00', '0.00', '1.00']
----------------------------------------------
```

This result unfortunately does not match the results obtained by hand calculation of the simplex algorither for the given tableu. Let try using the Python Scipy library's built-in simplex function and compare.

```
In [2]: import numpy as np
        from scipy.optimize import linprog
        from numpy.linalg import solve

        c = np.array([-1, -1, -2])
        A_ub = np.array([[2, 1, 2],[-1, -1, -1]])
        b_ub = np.array([[8],[-2]])
        A_eq = np.array([[-1, 1, 2]])
        b_eq = np.array([1])
        bounds = ((0, None), (0, None), (0, None))
        res = linprog(c, A_ub, b_ub, A_eq, b_eq, bounds=bounds)
        print(res)
```

```
     fun: -5.6666666666666661
 message: 'Optimization terminated successfully.'
     nit: 3
```

```
   slack: array([ 0.,  2.])
  status: 0
 success: True
       x: array([ 2.33333333,  0.        ,  1.66666667])
```

Happily this result *does* math those of the by-hand calculations, thus it seems safe to assume that there are lingering issues with the previous simplex implimentation as currently coded.

0.3 Problem 3.

We wish to minimize the weight. What follows is one approach to making the objective function and constraints of this optimization problem well defined.

We expect weight $\sim V$ where V is volume. The only stated assumption in this analysis is that the truss members have a slenderness ratio $L >> (A_1)^{1/2}$ such that Eule'rs formula may be applied. If we define the inscribed angle θ, between members AB and AC as

$$\theta = \arctan \frac{H}{L},$$

then the objective function becomes

$$V = A_1 L + A_2 L \cos \arctan \frac{H}{L}.$$

We must also define the constraints of the problem. Stated in plain words the constraints are that the members AB and AC, which are in tension and compression respectively, cannot fail by yeilding or buckling. We can find the loads in each member by applying the method of joints at joint A. Here we have

$$F_{AB} = \frac{P}{\sin\theta} = P \arcsin \arctan \frac{H}{L} F_{AC} = F_{AB} \cos\theta = P \arctan \arctan \frac{H}{L}.$$

where P is the load. The yeilding critera can be stated as

$$\frac{F_{AB}}{A_2} < \sigma_{yeild},$$

which can be simplified as

$$\frac{P \arcsin \arctan \frac{H}{L}}{A_2} < \sigma_{yeild}.$$

The second constraint in the buckling condition in the compressive member AC. Euler's formula for the critical buckiling load can be simplifed by dividing out order 1 constants such that

$$F_{critical} = \frac{\pi^2 EI}{L^2} \sim \frac{1}{L^2} > F_{AC}.$$

We may now fully state the optimization problem:

$$V = A_1 L + A_2 L \cos \arctan \frac{H}{L} \text{ subject to the contraints:} \frac{P \arcsin \arctan \frac{H}{L}}{A_2} < \sigma_{yeild} P \arctan \arctan \frac{H}{L} < \frac{1}{L^2}.$$

0.4 Problem 4.

We wish to use Lagrangian multipliers to check for the points that satisfy the neccessary and sufficient conditions for the problem

$$\text{Maximize} \quad f(x) = x_1^2 + x_2^2 - 3x_1x_2 \text{subject to} \quad x_1^2 + x_2^2 = 6 \leq 8.$$

Noting the equality constraint (and consequent lack of need to apply Kuhn-Tucker conditions) we can restate the problem as the minimization of a the negative objective function, and adding the Lagrangian multiplier such that

$$H(x_1, x_2, \lambda) = f(x_1, x_2) + \lambda g(x_1, x_2) = -x_1^2 - x_2^2 + 3x_1x_2 + \lambda(x_1^2 + x_2^2 - 6).$$

Applying the necessary conditions:

$$\frac{\partial H}{\partial x_1} = 0 = -2x_1 + 3x_2 + 2\lambda x_1 \frac{\partial H}{\partial x_2} = 0 = -2x_2 + 3x_1 + 2\lambda x_2,$$

and solving for λ

$$\lambda = \frac{2x_1 - 3x_2}{2x_1} = \frac{2x_2 - 3x_1}{2x_2} = -\frac{1}{2}.$$

Substituting into the constraints $g(x_1, x_2)$ we have

$$2x_1^2 = 6x_1 = x_2 = \sqrt{3}f = -21.$$

Applying the sufficient conditions (using the partial derivative subscript notation):

$$H_{x_1x_1} = H_{x_2x_2} = -2 + 2\lambda = -3H_{x_1x_2} = Hx_2x_1 = 3g_{x_1} = g_{x_2} = ... = 2x_1 = 2\sqrt{3}.$$

Therefore (noting the simplifying complimentariness of the g_{x_n} vectors and their transposes), we take the sign of the second derivative to be the determinant of the 2×2 Hessian matrix such that

$$\left.\frac{\partial^2 f}{\partial x_1^2}\right|_{x_2=0} = -3 - 3 - 3 - 3 = -12 < 0.$$

Therefore we have a maximum.

```
In [ ]:
```