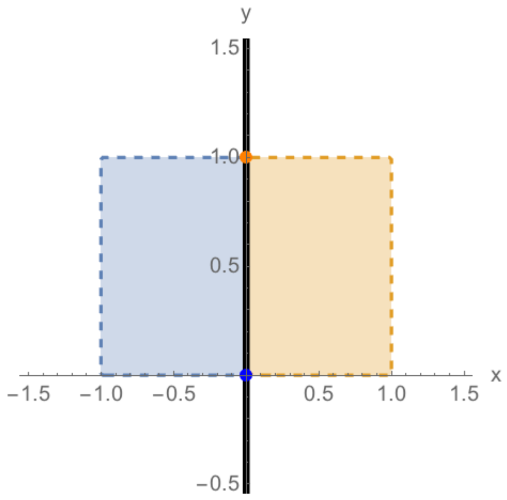
Caleb Logemann
MATH 566 Discrete Optimization
Homework 2

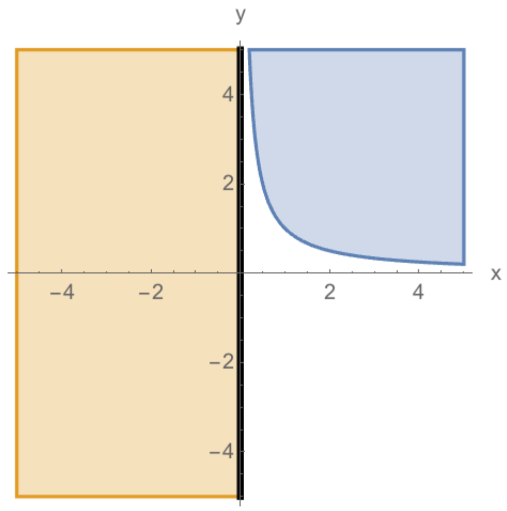
1. (a) Give an example of bounded convex sets C and D in $\mathbb{R}^d$, where $C \cap D = \varnothing$, but there is no hyperplane stricly seperating C and D. That is find $a \in \mathbb{R}^d$ and $b \in \mathbb{R}$, such that

$$\mathbf{a}^T\mathbf{c} > b, \forall \mathbf{c} \in C \quad \text{and} \quad \mathbf{a}^T\mathbf{d} < b, \forall \mathbf{d} \in D$$

Let $C = \left\{[x, y]^T | -1 < x < 0, 0 < y < 1\right\} \cup \left\{[0, 0]^T\right\}$ and let $D = \{(x, y) | 0 < x < 1, 0 < y < 1\} \cup \left\{[0, 1]^T\right\}$. In this case C and D are bounded convex sets and $C \cap D = \varnothing$. Let $\mathbf{a} = [-1, 0]^T$ and $b = 0$, then for $\mathbf{c} \in C$, $\mathbf{a}^T\mathbf{c} \geq 0$ and if $\mathbf{c} = [0, 0]$, then $\mathbf{a}^T\mathbf{c} = 0$. Also for $\mathbf{d} \in D$, $\mathbf{a}^T\mathbf{d} \leq 0$ and for $\mathbf{d} = [0, 1]^T$, $\mathbf{a}^T\mathbf{d} = 0$. Therefore there is no hyperplane strictly seperating C and D. A plot of these regions is shown below



(b) Give an example of closed convex sets C and D that cannot be strictly seperated.
Let $C = \left\{[x, y]^T | xy >= 1, x >= 0\right\}$, and let $D = \left\{[x, y]^T | x \leq 0\right\}$. In this case C and D are closed convex sets and $C \cap D = \varnothing$. Then it is required that $\mathbf{a} = [1, 0]^T$ and $b = 0$. For $\mathbf{c} \in C$, $\mathbf{a}^T\mathbf{c} > 0$. Also $\mathbf{d} \in D$, $\mathbf{a}^T\mathbf{d} \leq 0$. Therefore there is no hyperplane strictly seperating C and D. A plot of these regions is shown below



2. Dualize your diet problem. Take the data from your diet problem from HW1, create the dual of the program, and solve it. Your answer should contain the solution of the dual and the interpretation of the results of the dual.

   The original diet problem can be written as

$$\begin{array}{ll} \min & \mathbf{c}^T\mathbf{x} \\ \text{s.t.} & A\mathbf{x} \geq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{array}$$

   where

$$A^T = \begin{bmatrix} 125 & 35 & 0 & 0 & 0.275 \\ 190 & 7 & 2 & 7 & 0.130 \\ 110 & 27 & 1 & 0 & 0 \\ 50 & 13 & 0 & 0 & 0 \\ 190 & 27 & 1 & 4 & 0.360 \\ 100 & 18 & 1 & 2 & 0.06 \\ 60 & 9 & 2 & 2 & 0.2 \\ 20 & 4 & 2 & 1 & 0.38 \\ 70 & 12 & 3 & 4 & 0.37 \\ 60 & 25 & 4 & 2 & 0.105 \\ 120 & 2 & 0 & 1 & 0.025 \\ 140 & 24 & 1 & 4 & 0.44 \\ 80 & 1 & 0 & 8 & 0.69 \\ 60 & 0 & 0 & 11 & 0.52 \\ 150 & 25 & 1 & 4 & 0.025 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} 2779 \\ 383 \\ 38 \\ 60 \\ 1.5 \end{bmatrix} \quad \mathbf{c} = \begin{bmatrix} 0.75 \\ 0.13 \\ 0.34 \\ 0.13 \\ 0.48 \\ 0.17 \\ 0.21 \\ 0.21 \\ 0.21 \\ 0.25 \\ 0.42 \\ 0.26 \\ 0.50 \\ 1.15 \\ 0.29 \end{bmatrix}$$

   and $\mathbf{x}$ is the number of servings of each type of food.

   The dual of the diet program can be written as follows

$$\begin{array}{ll} \min & \mathbf{b}^T\mathbf{y} \\ \text{s.t.} & A^T\mathbf{y} \geq \mathbf{c} \\ & \mathbf{y} \geq \mathbf{0} \end{array}$$

   The dual of the program can be written in Sage as follows.

```
# create linear program
p = MixedIntegerLinearProgram(maximization=True)
y = p.new_variable(nonnegative=True)

A = matrix([[125, 35, 0, 0, 0.275], [190, 7, 2, 7, 0.130], [110, 27, 1,
    ↪ 0, 0],
[50, 13, 0, 0, 0], [190, 27, 1, 4, .36], [100, 18, 1, 2, .06], [60, 9,
    ↪ 2, 2, .2],
[20, 4, 2, 1, .38], [70, 12, 3, 4, .37], [60, 25, 14, 2, 0.105],
[120, 2, 0, 1, 0.025], [140, 24, 1, 4, 0.44], [80, 1, 0, 8, 0.69],
[60, 0, 0, 11, 0.52], [150, 25, 1, 4, 0.025]])
b = transpose(matrix([2779, 383, 38, 60, 1.4*1.5]))
c = transpose(matrix([0.75, 0.13, 0.34, 0.13, 0.48, 0.17, 0.21, 0.21,
    ↪ 0.21, 0.25, 0.42, 0.26, 0.50, 1.15, 0.29]))
```

```
p.set_objective(b[0,0]*y[1] + b[1, 0]*y[2] + b[2, 0]*y[3] + b[3, 0]*y[4]
    ↪ + b[4,0]*y[5])
p.add_constraint(A[0,0]*y[1] + A[0,1]*y[2] + A[0,2]*y[3] + A[0,3]*y[4] +
    ↪ A[0,4]*y[5] <= c[0,0])
p.add_constraint(A[1,0]*y[1] + A[1,1]*y[2] + A[1,2]*y[3] + A[1,3]*y[4] +
    ↪ A[1,4]*y[5] <= c[1,0])
p.add_constraint(A[2,0]*y[1] + A[2,1]*y[2] + A[2,2]*y[3] + A[2,3]*y[4] +
    ↪ A[2,4]*y[5] <= c[2,0])
p.add_constraint(A[3,0]*y[1] + A[3,1]*y[2] + A[3,2]*y[3] + A[3,3]*y[4] +
    ↪ A[3,4]*y[5] <= c[3,0])
p.add_constraint(A[4,0]*y[1] + A[4,1]*y[2] + A[4,2]*y[3] + A[4,3]*y[4] +
    ↪ A[4,4]*y[5] <= c[4,0])
p.add_constraint(A[5,0]*y[1] + A[5,1]*y[2] + A[5,2]*y[3] + A[5,3]*y[4] +
    ↪ A[5,4]*y[5] <= c[5,0])
p.add_constraint(A[6,0]*y[1] + A[6,1]*y[2] + A[6,2]*y[3] + A[6,3]*y[4] +
    ↪ A[6,4]*y[5] <= c[6,0])
p.add_constraint(A[7,0]*y[1] + A[7,1]*y[2] + A[7,2]*y[3] + A[7,3]*y[4] +
    ↪ A[7,4]*y[5] <= c[7,0])
p.add_constraint(A[8,0]*y[1] + A[8,1]*y[2] + A[8,2]*y[3] + A[8,3]*y[4] +
    ↪ A[8,4]*y[5] <= c[8,0])
p.add_constraint(A[9,0]*y[1] + A[9,1]*y[2] + A[9,2]*y[3] + A[9,3]*y[4] +
    ↪ A[9,4]*y[5] <= c[9,0])
p.add_constraint(A[10,0]*y[1] + A[10,1]*y[2] + A[10,2]*y[3] + A[10,3]*y
    ↪ [4] + A[10,4]*y[5] <= c[10,0])
p.add_constraint(A[11,0]*y[1] + A[11,1]*y[2] + A[11,2]*y[3] + A[11,3]*y
    ↪ [4] + A[11,4]*y[5] <= c[11,0])
p.add_constraint(A[12,0]*y[1] + A[12,1]*y[2] + A[12,2]*y[3] + A[12,3]*y
    ↪ [4] + A[12,4]*y[5] <= c[12,0])
p.add_constraint(A[13,0]*y[1] + A[13,1]*y[2] + A[13,2]*y[3] + A[13,3]*y
    ↪ [4] + A[13,4]*y[5] <= c[13,0])
p.add_constraint(A[14,0]*y[1] + A[14,1]*y[2] + A[14,2]*y[3] + A[14,3]*y
    ↪ [4] + A[14,4]*y[5] <= c[14,0])

# print solution
print('Objective Value: {}'.format(p.solve()))
for i, v in p.get_values(y).iteritems():
    print('y_%s = %s' % (i, v))
```

```
Objective Value: 3.95434986206
y_1 = 0.000327791491215
y_2 = 0.00715914040947
y_3 = 0.00299840278786
y_4 = 0.0
y_5 = 0.08929867867
```

The objective solution is $3.95 a day which is identical to the solution found in homework 1. The solution values are $0.0003277 per calorie, $0.007159 per carb, $0.002998 per gram of fiber, $0 per gram of protein, and $0.089298 per gram of sodium.

3. A paper mill manufactures rolls of paper of a standard width, 3 meters. But customers want to buy rolls of shorter width, and the mill has to cut such rolls from the 3m rolls. Let us consider an order of

   - 97 rolls of width 135 cm,
   - 610 rolls of width 108 cm,
   - 395 rolls of width 93 cm, and
   - 211 rolls of width 42 cm.

   What is the smallest number of 3m rolls that have to be cut in order to satisfy this order?

   First we must enumerate all of the different ways that a 3m roll can be cut up into these lengths.

   (a) $c_1 = 2 \times 135$
   (b) $c_2 = 135 + 108 + 42$
   (c) $c_3 = 135 + 93 + 42$
   (d) $c_4 = 135 + 3 \times 42$
   (e) $c_5 = 2 \times 108 + 2 \times 42$
   (f) $c_6 = 108 + 2 \times 93$
   (g) $c_7 = 108 + 93 + 2 \times 42$
   (h) $c_8 = 108 + 4 \times 42$
   (i) $c_9 = 3 \times 93.$
   (j) $c_{10} = 2 \times 93 + 2 \times 42$
   (k) $c_{11} = 93 + 4 \times 42$
   (l) $c_{12} = 7 \times 42$

   Any other cuts leave excess paper that could be used for the order so won't be necessary.

   Then we can create the integer linear program.

$$\begin{array}{ll}
\min & \sum_{i=1}^{12} (c_i) \\
\text{s.t.} & 2c_1 + c_2 + c_3 + c_4 \geq 97 \\
& c_2 + 2c_5 + c_6 + c_7 + c_8 \geq 610 \\
& c_3 + 2c_6 + c_7 + 3c_9 + 2c_{10} + c_{11} \geq 395 \\
& c_2 + c_3 + 3c_4 + 2c_5 + 2c_7 + 4c_8 + 2c_{10} + 4c_{11} + 7c_{12} \geq 211 \\
& c_i \geq 0 \quad 1 \leq i \leq 12 \\
& c_i \in \mathbb{Z} \quad 1 \leq i \leq 12
\end{array}$$

   This integer linear program can be written in Sage as follows

```
# create linear program
p = MixedIntegerLinearProgram(maximization=False)
c = p.new_variable(integer=True, nonnegative=True)
p.set_objective(c[1] + c[2] + c[3] + c[4] + c[5] + c[6] + c[7] + c[8] +
    ↪ c[9] + c[10] + c[11] + c[12])
p.add_constraint(2*c[1] + c[2] + c[3] + c[4] >= 97)
p.add_constraint(c[2] + 2*c[5] + c[6] + c[7] + c[8] >= 610)
p.add_constraint(c[3] + 2*c[6] + c[7] + 3*c[9] + 2*c[10] + c[11] >= 395)
```

```
p.add_constraint(c[2] + c[3] + 3*c[4] + 2*c[5] + 2*c[7] + 4*c[8] + 2*c
    ↪ [10] + 4*c[11] + 7*c[12] >= 211)
# print solution
print('Objective Value: {}'.format(p.solve()))
for i, v in p.get_values(c).iteritems():
    print('c_%s = %s' % (i, int(round(v))))
```

The solution given by Sage is

```
Objective Value: 453.0
c_1 = 49
c_2 = 0
c_3 = 0
c_4 = 0
c_5 = 206
c_6 = 198
c_7 = 0
c_8 = 0
c_9 = 0
c_10 = 0
c_11 = 0
c_12 = 0
```

This means that 453 3 m rolls are needed in order to fill this order. Of the 453 rolls, 49 should be cut into two 135 cm rolls, 206 shold be cut into two 108 cm rolls and two 42 cm rolls, and the remaining 198 rolls should be cut into a 108 cm roll and two 93 cm rolls