HW2

February 8, 2018

0.1 Homework 2: More linear programs

Name: Zihao Qiu
Campus ID: 9079810942
Email: zqiu34@wisc.edu

0.2 Problem 1 (Polyhedron modeling)

0.2.1 (a)

Assume x=(x_1, x_2, x_3). It's obvious that points in the regular cube satisify the inequalities as follows:

$$\begin{cases} -1 \le x_1 \le 1 \\ -1 \le x_2 \le 1 \\ -1 \le x_3 \le 1 \end{cases}$$

So we can get A and b:

$$A = \begin{pmatrix} 1 & 0 & 0 \\ -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & -1 \end{pmatrix} \tag{1}$$

$$b = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \tag{2}$$

0.2.2 (b)

It's obvious that points in the regular octahedron satisify the inequalities as follows:

$$\begin{cases} x_1 + x_2 + x_3 \leq 1 \\ x_1 + x_2 - x_3 \leq 1 \\ x_1 - x_2 + x_3 \leq 1 \\ -x_1 + x_2 + x_3 \leq 1 \\ x_1 - x_2 - x_3 \leq 1 \\ -x_1 + x_2 - x_3 \leq 1 \\ -x_1 - x_2 + x_3 \leq 1 \\ -x_1 - x_2 - x_3 \leq 1 \end{cases}$$

So we can get A and b:

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & -1 \\ 1 & -1 & 1 \\ -1 & 1 & 1 \\ 1 & -1 & -1 \\ -1 & 1 & -1 \\ -1 & -1 & 1 \\ -1 & -1 & -1 \end{pmatrix} \tag{3}$$

$$b = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \tag{4}$$

0.3 Problem 2 (Standard form with equality constraints)

0.3.1 (a)

First we can deal with $-1 \leq z_2 \leq 5$. Let $x_2 = z_2 + 1$, we have $0 \leq x_2 \leq 6$. $x_2 \geq 0$ is OK. In order to cope with $x_2 \leq 6$, we can add a slack variable s_2. Let $x_2 + s_2 = 6$ and we can have $s_2 \geq 0$

In the same way, we can convert $-1 \leq z_3 \leq 5$ into $x_3 \geq 0$, $s_3 \geq 0$ and $x_3 + s_3 = 6$. The same rule can also apply on $-2 \leq z_4 \leq 2$ and we can have $x_4 \geq 0$, $s_4 \geq 0$ and $x_4 + s_4 = 4$

Because z_1 is not constrained. So we can use u-v to replace z_1, where $u \geq 0$ and $v \geq 0$.

To deal with $-z_1 + 6z_2 - z_3 + z_4 \geq -3$, first we can turn it into $z_1 - 6z_2 + z_3 - z_4 \leq 3$, then we can add a slack variable s_5 to convert the inequality into equality: $z_1 - 6z_2 + z_3 - z_4 + s_5 = 3$, where $s_5 \geq 0$. After that, we can replace z_1 with u-v, replace z_2 with $x_2 - 1$ and so on. Finally we can get $u - v - 6x_2 + x_3 - x_4 + s_5 = -4$.

In the same way, we can convert $7z_2 + z_4 = 5$ and $z_3 + z_4 \leq 2$ into $7x_2 + x_4 = 14$ and $ x_3 + x_4 + s_6 = 5$.

The final LP problem with equality constraints can be described as follows:

$$\begin{aligned} -\min_{u,v,x_2,s_2,x_3,s_3,x_4,s_4,s_5,s_6} \quad & -3u+3v+x_2-1 \\ \textit{subject to}: u-v-6x_2+x_3-x_4+s_5 &= -4 \\ 7x_2+x_4 &= 14 \\ x_3+x_4+s_6 &= 5 \\ x_2+s_2 &= 6 \\ x_3+s_3 &= 6 \\ x_4+s_4 &= 4 \\ u\geq 0, v\geq 0, x_2\geq 0, s_2 &\geq 0 \\ x_3\geq 0, s_3\geq 0, x_4\geq 0, s_4 &\geq 0 \\ s_5\geq 0, s_6 &\geq 0 \end{aligned}$$

So we can have A, b, c, x:

$$x = \begin{pmatrix} u \\ v \\ x_2 \\ s_2 \\ x_3 \\ s_3 \\ x_4 \\ s_4 \\ s_5 \\ s_6 \\ 1 \end{pmatrix} \tag{5}$$

$$c = \begin{pmatrix} -3 \\ 3 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -1 \end{pmatrix} \tag{6}$$

$$A = \begin{pmatrix} 1 & -1 & -6 & 0 & 1 & 0 & -1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 7 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \end{pmatrix} \tag{7}$$

$$b = \begin{pmatrix} -4 \\ 14 \\ 5 \\ 6 \\ 6 \\ 4 \end{pmatrix} \tag{8}$$

0.3.2 (b)

First I use the version that the program gives us:

```
In [1]: using JuMP, Clp

        m = Model(solver = ClpSolver())

        @variable(m, z1)
        @variable(m, -1 <= z2 <= 5)
        @variable(m, -1 <= z3 <= 5)
        @variable(m, -2 <= z4 <= 2)

        @constraint(m, -z1 + 6z2 - z3 + z4 >= -3)
        @constraint(m, 7z2 + z4 == 5)
        @constraint(m, z3 + z4 <= 2)

        @objective(m, Max, 3z1 - z2)

        @time status = solve(m)

        println(status)
        println("z1: ", getvalue(z1))
        println("z2: ", getvalue(z2))
        println("z3: ", getvalue(z3))
        println("z4: ", getvalue(z4))
        println("objective value: ", getobjectivevalue(m))
```

```
  2.941261 seconds (1.42 M allocations: 73.950 MiB, 0.81% gc time)
Optimal
z1: 8.571428571428571
z2: 0.42857142857142855
z3: -1.0
z4: 2.0
objective value: 25.28571428571429
```

Then I use the standard form with equality constraints:

```
In [10]: using JuMP, Clp
```

```
        m = Model(solver = ClpSolver())

        @variable(m, u >= 0)
        @variable(m, v >= 0)
        @variable(m, x2 >= 0)
        @variable(m, s2 >= 0)
        @variable(m, x3 >= 0)
        @variable(m, s3 >= 0)
        @variable(m, x4 >= 0)
        @variable(m, s4 >= 0)
        @variable(m, s5 >= 0)
        @variable(m, s6 >= 0)

        @constraint(m, x2 + s2 == 6)
        @constraint(m, x3 + s3 == 6)
        @constraint(m, x4 + s4 == 4)
        @constraint(m, u - v -6x2 + x3 - x4 + s5 == -4)
        @constraint(m, 7x2 + x4 == 14)
        @constraint(m, x3 + x4 + s6 == 5)

        @objective(m, Min, -3u + 3v + x2 -1)

        @time status = solve(m)

        println(status)
        println("u: ", getvalue(u))
        println("v: ", getvalue(v))
        println("x2: ", getvalue(x2))
        println("s2: ", getvalue(s2))
        println("x3: ", getvalue(x3))
        println("s3: ", getvalue(s3))
        println("x4: ", getvalue(x4))
        println("s4: ", getvalue(s4))
        println("s5: ", getvalue(s5))
        println("s6: ", getvalue(s6))
        println("objective value: ", -getobjectivevalue(m))
```

```
  0.000438 seconds (70 allocations: 6.563 KiB)
Optimal
u: 8.571428571428571
v: 0.0
x2: 1.4285714285714286
s2: 4.571428571428571
x3: 0.0
s3: 6.0
x4: 4.0
s4: 0.0
s5: 0.0
```

```
s6: 1.0
objective value: 25.28571428571429
```

We can see that both versions have the same objective value. Note that z_1=u-v, z_2=x_2-1, z_3=x_3-1, z_4=x_4-2. So in fact, two versions of LP get the optimal value on the same input values.

0.4 Problem 3 (Alloy blending)

Note that the grade of Carbon is 2%-3%, So the weight of Carbon in 500 tons of steel is 10tons-15tons. In the same way, we can get the weight of Copper is 2tons-3tons and the weight of Manganese is 6tons-8.25tons.

Assume that we need $x_1$tons of Iron alloy 1, $x_2$tons of Iron alloy 2, $x_3$tons of Iron alloy 3, $x_4$tons of Copper 1, $x_5$tons of Copper 2, $x_6$tons of Alumimum 1 and $x_7$tons of Alumimum 2, and then the LP problem can be described as follows:

$$
\begin{aligned}
\min_{x_1,x_2,x_3,x_4,x_5,x_6,x_7} \quad & 200x_1 + 250x_2 + 150x_3 + 220x_4 + 240x_5 + 200x_6 + 165x_7 \\
\textit{subject to}: \quad & 10 \le 0.025x_1 + 0.03x_2 \le 15 \\
& 2 \le 0.003x_3 + 0.9x_4 + 0.96x_5 + 0.004x_6 + 0.006x_7 \le 3 \\
& 6 \le 0.013x_1 + 0.008x_2 + 0.04x_5 + 0.012x_6 \le 8.25 \\
& x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 = 500 \\
& 0 \le x_1 \le 400 \\
& 0 \le x_2 \le 300 \\
& 0 \le x_3 \le 600 \\
& 0 \le x_4 \le 500 \\
& 0 \le x_5 \le 200 \\
& 0 \le x_6 \le 300 \\
& 0 \le x_7 \le 250
\end{aligned}
$$

```
In [1]: using JuMP, Clp

        m = Model(solver = ClpSolver())

        @variable(m, 0 <= x1 <= 400)
        @variable(m, 0 <= x2 <= 300)
        @variable(m, 0 <= x3 <= 600)
        @variable(m, 0 <= x4 <= 500)
        @variable(m, 0 <= x5 <= 200)
        @variable(m, 0 <= x6 <= 300)
        @variable(m, 0 <= x7 <= 250)

        @constraint(m, x1 + x2 + x3 + x4 + x5 + x6 + x7 == 500)
        @constraint(m, 10 <= 0.025x1 + 0.03x2 <= 15)
        @constraint(m, 2 <= 0.003x3 + 0.9x4 + 0.96x5 + 0.004x6 + 0.006x7 <= 3)
        @constraint(m, 6 <= 0.013x1 + 0.008x2 + 0.04x5 + 0.012x6 <= 8.25)
```

```
    @objective(m, Min, 200x1 + 250x2 + 150x3 + 220x4 + 240x5 + 200x6 + 165x7)

    @time status = solve(m)

    println(status)
    println("x1: ", getvalue(x1))
    println("x2: ", getvalue(x2))
    println("x3: ", getvalue(x3))
    println("x4: ", getvalue(x4))
    println("x5: ", getvalue(x5))
    println("x6: ", getvalue(x6))
    println("x7: ", getvalue(x7))
    println("objective value: ", getobjectivevalue(m))
```

```
  2.567154 seconds (1.42 M allocations: 73.843 MiB, 0.93% gc time)
Optimal
x1: 400.0
x2: 0.0
x3: 39.77630199231039
x4: 0.0
x5: 2.761272282418735
x6: 57.462425725270876
x7: 0.0
objective value: 98121.63579168124
```

So we need 400 tons of Iron alloy 1, 39.78 tons of Iron alloy3, 2.76 tons of Copper2 and 57.46 tons of Aluminum 1. The minimum production cost is \$98121.64.

0.5 Problem 4 (Stigler's diet)

0.5.1 (a)

First I formulate Stigler's diet problem as an LP.

Let x = $(x_1,x_2,\cdots,x_{77})$ to be a vector to describe how much dollar I should spend on the corresponding food. For example, I should spend x_1 dollars on 'Wheat Flour(Enriched)', spend x_2 dollars on 'Macaroni' and so on. The the problem can be formulated as follows:

data[f,i] is the amount of nutrient i contained in food f. lower[i] is the minimum daily requirement of nutrient i.

$$\min_x \sum_{i=1}^{77} x_i$$

$$\begin{aligned} subject\ to: \sum_{i=1}^{77} x_i * data[i,1] &\geq lower[1] \\ \sum_{i=1}^{77} x_i * data[i,2] &\geq lower[2] \\ \sum_{i=1}^{77} x_i * data[i,3] &\geq lower[3] \\ \sum_{i=1}^{77} x_i * data[i,4] &\geq lower[4] \\ \sum_{i=1}^{77} x_i * data[i,5] &\geq lower[4] \\ \sum_{i=1}^{77} x_i * data[i,6] &\geq lower[4] \\ \sum_{i=1}^{77} x_i * data[i,7] &\geq lower[4] \\ \sum_{i=1}^{77} x_i * data[i,8] &\geq lower[4] \\ \sum_{i=1}^{77} x_i * data[i,9] &\geq lower[4] \end{aligned}$$

The I solve this problem with Julia. The codes are as follows:

```
In [4]: # STARTER CODE FOR STIGLER'S DIET PROBLEM
        using NamedArrays

        # import Stigler's data set
        raw = readcsv("C:\\Users\\Administrator\\Desktop\\cs524\\stigler.csv")
        (m,n) = size(raw)

        n_nutrients = 2:n      # columns containing nutrients
        n_foods = 3:m          # rows containing food names

        nutrients = raw[1,n_nutrients][:]   # the list of nutrients (convert to 1-D array)
        foods = raw[n_foods,1][:]           # the list of foods (convert to 1-D array)

        # lower[i] is the minimum daily requirement of nutrient i.
        lower = Dict( zip(nutrients,raw[2,n_nutrients]) )

        # data[f,i] is the amount of nutrient i contained in food f.
        data = NamedArray( raw[n_foods,n_nutrients], (foods,nutrients), ("foods","nutrients") );
```

```
In [5]: using JuMP, Clp
```

```
        m = Model(solver = ClpSolver())

        @variable(m, x[1:length(n_foods)] >= 0)

        @constraint(m, sum(x[i] * data[i,1] for i in 1:length(n_foods)) >= lower["Calories (1000
        @constraint(m, sum(x[i] * data[i,2] for i in 1:length(n_foods)) >= lower["Protein (g)"])
        @constraint(m, sum(x[i] * data[i,3] for i in 1:length(n_foods)) >= lower["Calcium (g)"])
        @constraint(m, sum(x[i] * data[i,4] for i in 1:length(n_foods)) >= lower["Iron (mg)"])
        @constraint(m, sum(x[i] * data[i,5] for i in 1:length(n_foods)) >= lower["Vitamin A (100
        @constraint(m, sum(x[i] * data[i,6] for i in 1:length(n_foods)) >= lower["Thiamine (mg)"
        @constraint(m, sum(x[i] * data[i,7] for i in 1:length(n_foods)) >= lower["Riboflavin (mg
        @constraint(m, sum(x[i] * data[i,8] for i in 1:length(n_foods)) >= lower["Niacin (mg)"])
        @constraint(m, sum(x[i] * data[i,9] for i in 1:length(n_foods)) >= lower["Ascorbic Acid

        @objective(m, Min, sum(x[i] for i=1:length(n_foods)))

        @time status = solve(m)

        println(status)

        for i in 1:length(n_foods)
            println(foods[i], ' ',getvalue(x[i]))
        end
        println("total value (one yaer): ", getobjectivevalue(m)*365)
        println("objective value (one day): ", getobjectivevalue(m))
```

```
  0.000825 seconds (70 allocations: 48.813 KiB)
Optimal
Wheat Flour (Enriched) 0.02951906167648827
Macaroni 0.0
Wheat Cereal (Enriched) 0.0
Corn Flakes 0.0
Corn Meal 0.0
Hominy Grits 0.0
Rice 0.0
Rolled Oats 0.0
White Bread (Enriched) 0.0
Whole Wheat Bread 0.0
Rye Bread 0.0
Pound Cake 0.0
Soda Crackers 0.0
Milk 0.0
Evaporated Milk (can) 0.0
Butter 0.0
Oleomargarine 0.0
Eggs 0.0
Cheese (Cheddar) 0.0
```

```
Cream 0.0
Peanut Butter 0.0
Mayonnaise 0.0
Crisco 0.0
Lard 0.0
Sirloin Steak 0.0
Round Steak 0.0
Rib Roast 0.0
Chuck Roast 0.0
Plate 0.0
Liver (Beef) 0.0018925572907052643
Leg of Lamb 0.0
Lamb Chops (Rib) 0.0
Pork Chops 0.0
Pork Loin Roast 0.0
Bacon 0.0
Ham, smoked 0.0
Salt Pork 0.0
Roasting Chicken 0.0
Veal Cutlets 0.0
Salmon, Pink (can) 0.0
Apples 0.0
Bananas 0.0
Lemons 0.0
Oranges 0.0
Green Beans 0.0
Cabbage 0.011214435246144865
Carrots 0.0
Celery 0.0
Lettuce 0.0
Onions 0.0
Potatoes 0.0
Spinach 0.005007660466725203
Sweet Potatoes 0.0
Peaches (can) 0.0
Pears (can) 0.0
Pineapple (can) 0.0
Asparagus (can) 0.0
Green Beans (can) 0.0
Pork and Beans (can) 0.0
Corn (can) 0.0
Peas (can) 0.0
Tomatoes (can) 0.0
Tomato Soup (can) 0.0
Peaches, Dried 0.0
Prunes, Dried 0.0
Raisins, Dried 0.0
Peas, Dried 0.0
```

```
Lima Beans, Dried 0.0
Navy Beans, Dried 0.061028563526693246
Coffee 0.0
Tea 0.0
Cocoa 0.0
Chocolate 0.0
Sugar 0.0
Corn Syrup 0.0
Molasses 0.0
Strawberry Preserves 0.0
total value (one yaer): 39.66173154546625
objective value (one day): 0.10866227820675685
```

My cheapest annual cost is $39.66. It's a little bit lower than $39.93.
In my optimal diet, there are 'Wheat Flour (Enriched)', 'Liver (Beef)', 'Cabbage', 'Spinach' and 'Navy Beans, Dried'

0.5.2 (b)

I can delete all the food items related to 'meat' in 'stigler.csv'. Then I get the file 'stigler_vegetarian.csv'. Use the code above and then I get the final result.

```
In [6]: # STARTER CODE FOR STIGLER'S DIET PROBLEM (VEGETARIAN)
        using NamedArrays

        # import Stigler's data set
        raw = readcsv("C:\\Users\\Administrator\\Desktop\\cs524\\stigler_vegetarian.csv")
        (m,n) = size(raw)

        n_nutrients = 2:n      # columns containing nutrients
        n_foods_veg = 3:m          # rows containing food names

        nutrients = raw[1,n_nutrients][:]   # the list of nutrients (convert to 1-D array)
        foods_veg = raw[n_foods_veg,1][:]           # the list of foods (convert to 1-D array)

        # lower[i] is the minimum daily requirement of nutrient i.
        lower = Dict( zip(nutrients,raw[2,n_nutrients]) )

        # data[f,i] is the amount of nutrient i contained in food f.
        data = NamedArray( raw[n_foods_veg,n_nutrients], (foods_veg,nutrients), ("foods","nutrie

In [7]: using JuMP, Clp

        m = Model(solver = ClpSolver())

        @variable(m, x[1:length(n_foods_veg)] >= 0)

        @constraint(m, sum(x[i] * data[i,1] for i in 1:length(n_foods_veg)) >= lower["Calories (
        @constraint(m, sum(x[i] * data[i,2] for i in 1:length(n_foods_veg)) >= lower["Protein (g
        @constraint(m, sum(x[i] * data[i,3] for i in 1:length(n_foods_veg)) >= lower["Calcium (g
        @constraint(m, sum(x[i] * data[i,4] for i in 1:length(n_foods_veg)) >= lower["Iron (mg)"
        @constraint(m, sum(x[i] * data[i,5] for i in 1:length(n_foods_veg)) >= lower["Vitamin A
        @constraint(m, sum(x[i] * data[i,6] for i in 1:length(n_foods_veg)) >= lower["Thiamine (
        @constraint(m, sum(x[i] * data[i,7] for i in 1:length(n_foods_veg)) >= lower["Riboflavin
        @constraint(m, sum(x[i] * data[i,8] for i in 1:length(n_foods_veg)) >= lower["Niacin (mg
        @constraint(m, sum(x[i] * data[i,9] for i in 1:length(n_foods_veg)) >= lower["Ascorbic A

        @objective(m, Min, sum(x[i] for i=1:length(n_foods_veg)))

        @time status = solve(m)

        println(status)

        for i in 1:length(n_foods_veg)
            println(foods_veg[i], ' ',getvalue(x[i]))
        end
        println("total value (one yaer): ", getobjectivevalue(m)*365)
        println("objective value (one day): ", getobjectivevalue(m))
```

```
  0.000721 seconds (70 allocations: 39.219 KiB)
Optimal
Wheat Flour (Enriched) 0.03545558140888771
Macaroni 0.0
Wheat Cereal (Enriched) 0.0
Corn Flakes 0.0
Corn Meal 0.0
Hominy Grits 0.0
Rice 0.0
Rolled Oats 0.0
White Bread (Enriched) 0.0
Whole Wheat Bread 0.0
Rye Bread 0.0
Pound Cake 0.0
Soda Crackers 0.0
Milk 0.0
Evaporated Milk (can) 0.008591461668763544
Butter 0.0
Oleomargarine 0.0
Eggs 0.0
Cheese (Cheddar) 0.0
Cream 0.0
Peanut Butter 0.0
Mayonnaise 0.0
Crisco 0.0
Plate 0.0
Apples 0.0
```

```
Bananas 0.0
Lemons 0.0
Oranges 0.0
Green Beans 0.0
Cabbage 0.011249517312443502
Carrots 0.0
Celery 0.0
Lettuce 0.0
Onions 0.0
Potatoes 0.0
Spinach 0.005112832613199646
Sweet Potatoes 0.0
Peaches (can) 0.0
Pears (can) 0.0
Pineapple (can) 0.0
Asparagus (can) 0.0
Green Beans (can) 0.0
Corn (can) 0.0
Peas (can) 0.0
Tomatoes (can) 0.0
Tomato Soup (can) 0.0
Peaches, Dried 0.0
Prunes, Dried 0.0
Raisins, Dried 0.0
Peas, Dried 0.0
Lima Beans, Dried 0.0
Navy Beans, Dried 0.04862804357316852
Coffee 0.0
Tea 0.0
Cocoa 0.0
Chocolate 0.0
Sugar 0.0
Corn Syrup 0.0
Molasses 0.0
Strawberry Preserves 0.0
total value (one yaer): 39.79866435040896
objective value (one day): 0.10903743657646292
```

We can see that vegetarian will spend at least $39.80 a year to meet the RDA. Vegetarian will use ‘Wheat Flour (Enriched)’, ‘Evaporated Milk (can)’, ‘Cabbage’, ‘Spinach’ and ‘Navy Beans, Dried’.