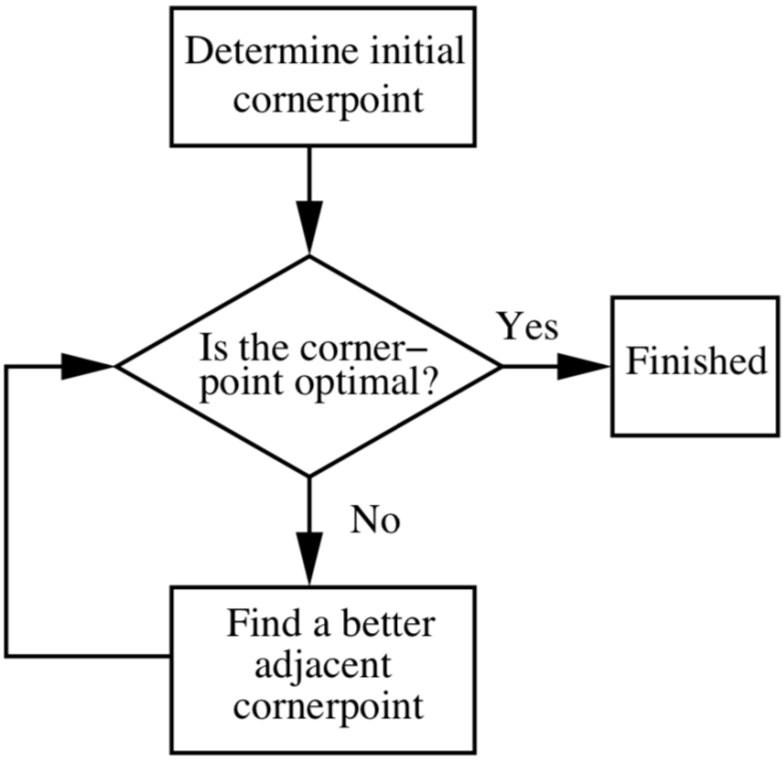
SF1811 HOME ASSIGNMENT 1
OPTIMIZATION MODELLING AND THE SIMPLEX METHOD

Geoffrey Lau Chung Yin

Introduction

Simplex method is created by mathematician George Dantzig in 1947. The purpose of simplex method is to relatively search along edges to other corner points for a better objective function value. The method first converts the linear programming question into a matrix standard form, which then we introduce slack variables to form equality statements. Basis is formed with a selection of variables in question, usually starting with slack variables. The basis is then checked if it forms a Basic Feasible Solution for the linear programming question, which determines if it is a corner point. If the basis is proven to be basic feasible solution, then reduced cost is calculated to determine if the corner point's optimality. If the reduced cost contains negative component, then the solution is not optimal, and changing variables for the basis is required. The process reiterates until the a basis with only positive components in its reduced cost is considered optimal, completing the whole simplex method. The following picture is describes the iterate process of simplex method.


Determine initial cornerpoint
Is the corner–point optimal?
Yes
Finished
No
Find a better adjacent cornerpoint

The program I wrote for this assignment is in python programming language, which takes in a linear programming question in matrix standard form with only equalities, and applying the simplex method step by step. At the end, the program also uses linprog function provided by scipystats to compare with my implementation.

Explanation of implementation

Suppose a company can create 4 products, A, B, C and D. Its factory runs 8 hours a day, containing two different manufacturing stage that is necessary for all products. The first processing stage can either handle 20 pieces of A, 16 pieces of B, 11 pieces of C or 5 pieces of D for each hour. The second processing stage can either handle 10 pieces of A, 15 pieces of B, 24 pieces of C or 10 pieces of D for each hour. The profit from each type of product is 12 dollars for A, 9 dollars for B, 8 dollars for C and 10 dollars for D per piece of product produced. How many of each type of product should the company produced?

The linear programming is then the following, with x_1, x_2, x_3, x_4 as number of units of type A,B,C,D produced in a day respectively.

$$\max\ 12x_1 + 9x_2 + 8x_3 + 10x_4$$
$$s.t.\ \frac{x_1}{20} + \frac{x_2}{16} + \frac{x_3}{11} + \frac{x_4}{5} \le 8$$
$$\frac{x_1}{10} + \frac{x_2}{15} + \frac{x_3}{24} + \frac{x_4}{10} \le 8$$
$$x_1, x_2, x_3, x_4 \ge 0$$

First, we change the question into standard form, and introduce slack variables to convert the inequality statements to equality statements. The question changes to,

$$\min\ -12x_1 - 9x_2 - 8x_3 - 10x_4$$
$$s.t.\ \frac{x_1}{20} + \frac{x_2}{16} + \frac{x_3}{11} + \frac{x_4}{5} + x_5 = 8$$
$$\frac{x_1}{10} + \frac{x_2}{15} + \frac{x_3}{24} + \frac{x_4}{10} + x_6 = 8$$
$$x_1, x_2, x_3, x_4, x_5, x_6 \ge 0$$

And written in matrix form for simplex method, it becomes

$$A = \begin{pmatrix} 1/20 & 1/16 & 1/11 & 1/5 & 1 & 0 \\ 1/10 & 1/15 & 1/24 & 1/10 & 0 & 1 \end{pmatrix}$$
$$b = (8 \quad 8)$$
$$C = (-12 \quad -9 \quad -8 \quad -10 \quad 0 \quad 0)$$
$$x = (x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6)$$

In this problem, we have used two slack variables, x_5 and x_6. The above values are first inputted into the program, labelled as part 1 in commentation of code.

Now, we form our initial basis for the first iteration of simplex method. As we can take the slack variables as our initial basis, so forming matrix beta (5,6) and mu (1,2,3,4).

$$A_\beta = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad A_\mu = \begin{pmatrix} 1/20 & 1/16 & 1/11 & 1/5 \\ 1/10 & 1/15 & 1/24 & 1/10 \end{pmatrix}$$
$$c_\beta = (0 \quad 0) \qquad c_\mu = (-12 \quad -9 \quad -8 \quad -10)$$

The above step is labelled as part 2 in commentation of code.

With the setup of basis completed, we proceed to test the basis if it is a basic feasible solution. This is done by finding a positive matrix that is able multiply with A_β to form b. In the program, the matrix variable "b_bar" is calculated by matrix multiplication of inverse matrix of A_β and matrix of b. This step is labelled as part 3 in commentation of code.

$$\bar{b} = {A_\beta}^{-1} b$$

If the basis does form a basic feasible solution, we begin the process of calculating its reduced cost to observe if the basis is optimal or not. To do this, we have to first calculate the simplex multiplier. This is done by

$$y = {A_\beta}^{T^{-1}} c_\beta$$

Then reduced cost is calculated with the following equation

$$r_\mu = c_\mu - {A_\mu}^T y$$

If the reduced cost consist of only positive values, then the basis at hand is optimal. The step above is labelled as part 4 in commentation of code.

In the case where the reduced cost is negative, we swap one of the variables in beta (5,6) with mu (1,2,3,4). For selection of variable in mu, we select the column responsible for the smallest value in the reduced cost to decrease the objective value in that particular direction. As for selection of variable to leave the basis, we consider which variable in the original basis creates the smallest factor with the new variable we chose while remaining feasible. To find the largest possible factor for our new variable, we can use the following equation,

$$\bar{a} = {A_\beta}^{-1} a_x$$
$$x_\beta = \bar{b} - \bar{a}t$$

where a_x is the new variable for the basis. For the solution still be feasible, x_β must be larger or equal to zero, therefore this creates an inequality

$$\bar{b}_i - \bar{a}_i t \geq 0$$

for *i* represents each column in the basis. I then subject the equation as below

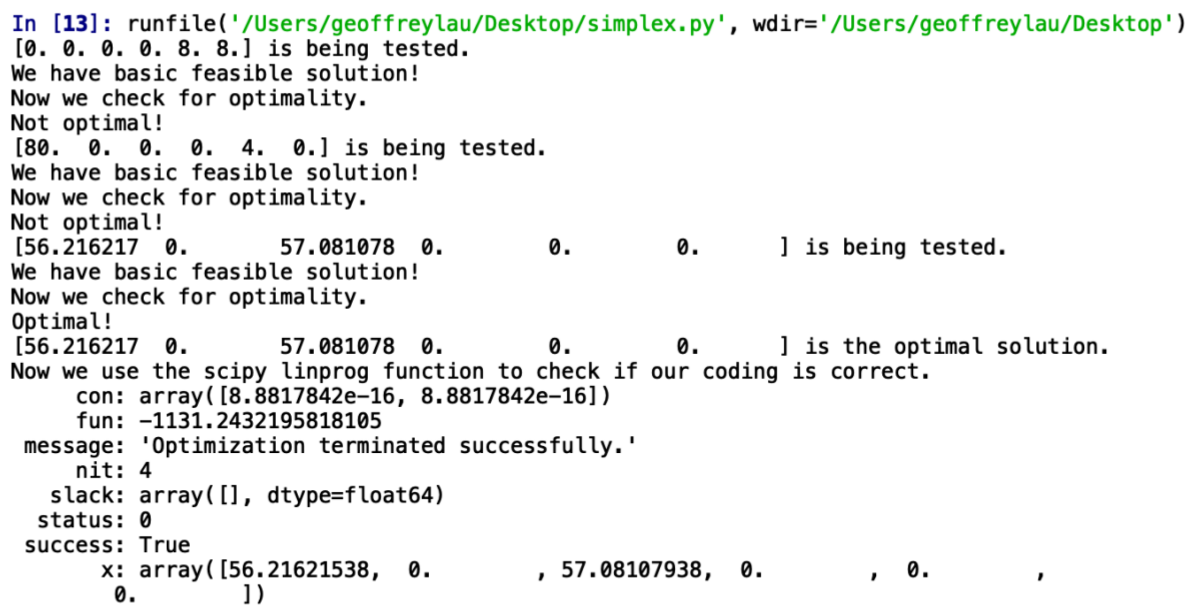
$$\frac{\bar{b}_i}{\bar{a}_i} = t$$

This calculates the factor we need for comparison, and since x_β must be larger or equal to zero any t values that is negative or equals to zero is not needed, eliminating the need of inequality calculation in the program. The program only need to choose the variable with the smallest t value bigger than zero to leave the basis. This step is labelled as part 5 in commentation of code.

From here, the calculation repeats itself with the new swapped basis, starting the iteration from recalculating "b_bar" to checking optimality. If the new basis is still not optimal, it will go through swapping basis again and commence next iteration. Otherwise, if the new basis is proven to be optimal, then the simplex method is finished.

The following picture is the output of the program running the question stated at the beginning. As shown, the implementation equals to the linear programming function from scipystats package.

Back to the example, this means that the company should obtain the highest profit with producing 56 pieces of A and 57 pieces of C in a day.



```
In [13]: runfile('/Users/geoffreylau/Desktop/simplex.py', wdir='/Users/geoffreylau/Desktop')
[0. 0. 0. 0. 8. 8.] is being tested.
We have basic feasible solution!
Now we check for optimality.
Not optimal!
[80.  0.  0.  0.  4.  0.] is being tested.
We have basic feasible solution!
Now we check for optimality.
Not optimal!
[56.216217  0.       57.081078  0.        0.        0.      ] is being tested.
We have basic feasible solution!
Now we check for optimality.
Optimal!
[56.216217  0.       57.081078  0.        0.        0.      ] is the optimal solution.
Now we use the scipy linprog function to check if our coding is correct.
     con: array([8.8817842e-16, 8.8817842e-16])
     fun: -1131.2432195818105
 message: 'Optimization terminated successfully.'
     nit: 4
   slack: array([], dtype=float64)
  status: 0
 success: True
       x: array([56.21621538,  0.        , 57.08107938,  0.        ,  0.        ,
        0.        ])
```

Appendix

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Oct 30 23:48:50 2019

@author: geoffreylau
"""

# initialization

import numpy as np
from scipy import optimize

# part 1

# input of all 3 variables, declaring how many slack variables are introduced

A = np.array([[1/20,1/16,1/11,1/5,1,0],[1/10,1/15,1/24,1/10,0,1]],dtype = 'f')
B = np.array([8,8],dtype = 'f')
C = np.array([-12,-9,-8,-10,0,0],dtype = 'f')

# amount of slack variables, as we can assume the first basis as slack variables in this assignment

slack = 2


# finding size of A

numrowsA = len(A)
numcolsA = len(A[0])




# part 2
# setting up beta and mu, choosing the last columns as basis

mu = np.arange(0,numcolsA - slack)
beta = np.arange(numcolsA - slack, numcolsA)

# A beta

A_beta = np.zeros((numrowsA,slack),dtype = 'f')
for y in range(slack):
    for x in range(numrowsA):
        A_beta[x][y] = A[x][beta[y]]
        x = x + 1
    y = y + 1

# A mu
A_mu = np.zeros((numrowsA,numcolsA - slack),dtype = 'f')
for y in range(numcolsA-slack):
    for x in range(numrowsA):
        A_mu[x][y] = A[x][mu[y]]
        x = x + 1
    y = y + 1

# C beta

C_beta = np.zeros(slack,dtype = 'f')
for x in range(slack):
    C_beta[x]=C[beta[x]]
    x = x + 1

# C mu

C_mu = np.zeros(numcolsA - slack,dtype = 'f')
for x in range(numcolsA - slack):
    C_mu[x]=C[mu[x]]
    x = x + 1

# begin iteration

optimal = False
feasible = True

while optimal == False:



# part 3


# calculate b bar

    A_beta_inv = np.linalg.inv(A_beta)
    b_bar = np.matmul(A_beta_inv,B)
    count = 0
    for x in range(len(b_bar)):
        if(b_bar[x]<0):
            count = count + 1
    x_current = np.array([0]*numcolsA,dtype = 'f')
    for x in range(len(beta)):
        x_current[beta[x]] = b_bar[x]
    print(x_current, "is being tested.")

# checking if it is a BFS

    if(count == 0):
        print ("We have basic feasible solution!")
        print ("Now we check for optimality.")
        feasible = True
    else:
        print ("Not feasible!")

# If not BFS we stop immediately by triggering boolean

        optimal = True
        feasible = False

    if(optimal == False and feasible == True):

# part 4

# Finding simplex multipliers

        y = np.matmul(np.linalg.inv(np.transpose(A_beta)),C_beta)

# Computing reduced cost

        r = np.subtract(C_mu,np.matmul(np.transpose(A_mu),y))



        count2 = 0
        for x in range(len(r)):
            if(r[x]>=0):
                count2 = count2 + 1
        if (count2 == len(r)):

# If optimal we can leave loop by triggering boolean

            print("Optimal!")
            optimal = True
        else:
            print("Not optimal!")

# Finding the smallest value in the reduced cost, locating its position in the matrix

        entering_index = np.argmin(r)

    if(optimal == False and feasible == True):

# part 5


# Finding the smallest value in the basis, locating its position in the matrix
# Using ratio to calculate the maximum t values, and choosing the smallest ratio to leave the basis

        a_temp = np.array([0]*numrowsA,dtype = 'f')
        x_beta_ratio = np.array([0]*numrowsA,dtype = 'f')
        for x in range(numrowsA):
            a_temp[x] = A_mu[x][entering_index]
        a_bar = np.matmul(np.linalg.inv(A_beta),a_temp)
        for x in range(numrowsA):
            x_beta_ratio[x] = b_bar[x] / a_bar[x]
            if (x_beta_ratio[x]<= 0):

# since equality, we dont want any ratio that is negative or 0, so we just randomly append...
# ... a huge value so it wont be the smallest ratio that we are choosing

                x_beta_ratio[x] = 10000000
        leaving_index = np.argmin(x_beta_ratio)
        # Replacing mu and beta with the two locations we calculated
        C_beta[leaving_index], C_mu[entering_index] = C_mu[entering_index], C_beta[leaving_index]
        beta[leaving_index], mu[entering_index] = mu[entering_index], beta[leaving_index]
        for x in range(numrowsA):
            A_beta[x][leaving_index], A_mu[x][entering_index] =
A_mu[x][entering_index],A_beta[x][leaving_index]

# end of iteration



# printing our optimal solution

x_opt = np.array([0]*numcolsA,dtype = 'f')
for x in range(len(beta)):
    x_opt[beta[x]] = b_bar[x]
print(x_opt, "is the optimal solution.")

# checking our program with the linear programming of scipystats

print("Now we use the scipy linprog function to check if our coding is correct.")
check = optimize.linprog(C,A_eq=A, b_eq=B, method = 'simplex')
print(check)
```