Sima Shafaei

CSE620: mini project simplex

Due: October 7st, 2020

1) stating the LP problem using equations

The LP problem in the standard form can be written in the following form:

$$\begin{aligned} &\text{minimize } c^T x \\ &subject\ to: Ax = b \\ &x \geq 0 \end{aligned}$$

Where c and x are n dimensional column vector, b is a m-dimensional and non-negative column vector and A is m*n matrix. Other form of LP problem such as canonical form can be converted into standard form by adding new Slack and surplus variables.

2) describing the Simplex algorithm in pseudocode and equations

Consider a system of equations with *n* variables and *m* equations where $n \geq m$. A basic solution for this system is obtained in the following way:

a) Set *n-m* variables equal to zero. These variables are called non-basic variables (N.B.V).

b) Solve the system for the *m* remaining variables. These variables are called basic variables (B.V.)

$$\Longrightarrow [B|N]x = b \Longrightarrow B\,x_B + N\,x_N = b$$

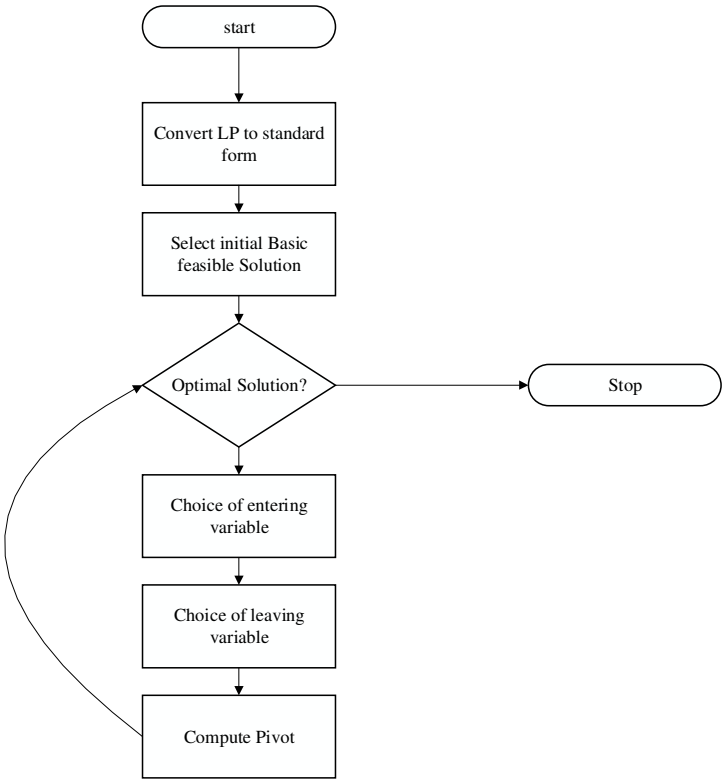
$$\Longrightarrow solve\ m\ resulting\ equation\ B\,x_B = b \Longrightarrow x_B = B^{-1}\,b$$

c) The vector of variables obtained is called the basic solution (x^1) (it contains both basic and non-basic variables).

d) compute the cost of basic solution

$$c^T x^1 = c_B^T x_B + c_N^T x_N = c_B^T (B^{-1} b) + c_N^T\, 0 = c_B^T B^{-1} b$$

Simplex method is a greedy search that moves from one basic solution to another basic solution with greatest reduced cost. To move from one basic solution to another we have to choose a variable for leaving the basic variable and another variable to entering. This is called pivot operation
Therefore, the algorithm of simplex method would be as follows:



3) describing the source of the Simplex code that you used, including its source code and simple examples of how to use it (and on which platform)

I used scipy.optimization.linprog library to optimize linear function using simplex method. To use this library, we should first present the optimization problem in the following form

$$\begin{aligned} \min_x \ & c^T x \\ \text{such that } & A_{ub} x \leq b_{ub}, \\ & A_{eq} x = b_{eq}, \\ & l \leq x \leq u, \end{aligned}$$

where x is a vector of decision variables; c, b_{ub}, b_{eq}, l, and u are vectors; and A_{ub} and A_{eq} are matrices.

Therefore, if you have "greater than" inequality constraint, the problem is not presented in the form accepted by linprog but it is easy to convert the “greater than” inequality constraint to a “less than” inequality constraint by multiplying both sides by a factor of −1. In this form l can be $-\infty$ and u can be ∞.

After converting problem to an acceptable format it is very easy to use linprog to optimize a function. You just need to assign A_{ub}, A_{eq}, b_{ub}, b_{eq}, l, u, and c. and call the function. For example, for the instance problem in lecture:

$\min x_1 + x_2 + x_3 + x_4$

$subject\ to: x_1 + 2x_2 - x_3 - x_4 = 1$

$-x_1 - 5x_2 + 2x_3 + 3x_4 = 1$

$x_i \geq 0\ \forall i$

A_{eq}, b_{eq} and c matrix would be as follows:

$$A_{eq} = \begin{bmatrix} 1 & 2 & -1 & -1 \\ -1 & -5 & 2 & 3 \end{bmatrix}, b_{eq} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, c^T = [1 \quad 1 \quad 1 \quad 1]$$

Following code shows optimization of above example using this module

```
import numpy as np
from scipy.optimize import linprog

A=[[1,2,-1,-1],[-1,-5,2,3]]
b=[[1,1]]
b=np.transpose(b)

c=[[1,1,1,1]]
c=np.transpose(c)

res = linprog(c, A_eq=A, b_eq=b, method="simplex")
print(res)
```

```
     con: array([0.00000000e+00, 6.66133815e-16])
     fun: 2.9999999999999996
 message: 'Optimization terminated successfully.'
     nit: 2
   slack: array([], dtype=float64)
  status: 0
 success: True
       x: array([2., 0., 0., 1.])
```

The result of function is printed. nit is the number of iteration before convergence. *x* is the values of the decision variables that minimizes the objective function while satisfying the constraints. *fun* is the optimal value of the objective function.

List the values of m, n, the problem instance statement, the system time needed for solving the LP and the solution that you found for each of the above combination of m and n

The values of m(number of constraints) and n(number of variables) is defined as follows:

	4	10	20	30	40	50
2	m=2,n=4	m=2,n=10	m=2,n=20	m=2,n=30	m=2,n=40	m=2,n=50
6	m=6,n=4	m=6,n=10	m=6,n=20	m=6,n=30	m=6,n=40	m=6,n=50
10	m=10,n=4	m=10,n=10	m=10,n=20	m=10,n=30	m=10,n=40	m=10,n=50
14	m=14,n=4	m=14,n=10	m=14,n=20	m=14,n=30	m=14,n=40	m=14,n=50

The instance problem created randomly for different values of m and n using the following code:

```
import time
constraints=[2,6,10,14]
variables=[4,10,20,30,40,50]
optimization_time=np.zeros((4,6))
optimization_iteration=np.zeros((4,6))
optimization_result=np.zeros((4,6))
slack_variables=np.zeros((4,6))
for i,m in enumerate(constraints):
    for j,n in enumerate(variables):
        A=np.random.randint(40, size=(m,n))
        A=A-20
        b=np.random.randint(10, size=(m,1))
        c=np.random.randint(20, size=(n,1))
        start_time=time.time()
        res = linprog(c, A_eq=A, b_eq=b, method="simplex")
        end_time=time.time()
        optimization_time[i][j]=end_time-start_time
        optimization_result[i][j]=res.fun
        optimization_iteration[i][j]=res.nit
        slack_variables[i][j]=len(res.slack)
```

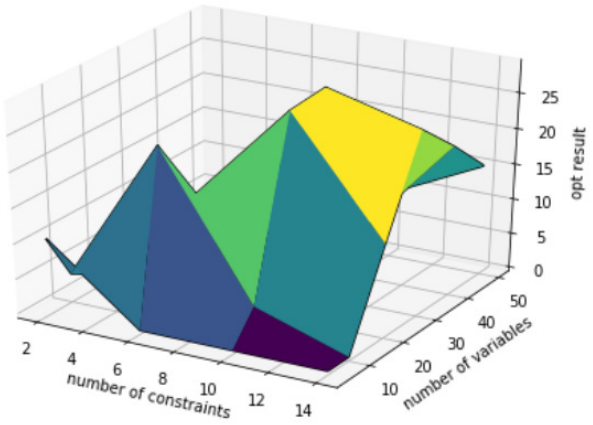
The I used the following code to tabulate and graph the solution of optimization problem:

```
fig = plt.figure()
ax = Axes3D(fig)

X, Y = np.meshgrid(constraints, variables)
ax.plot_wireframe(X,Y,np.transpose(optimization_result),color='black')
cset = ax.plot_surface(X,Y,np.transpose(optimization_result),cmap='viridis')
ax.set_xlabel('number of constraints')
ax.set_ylabel('number of variables')
ax.set_zlabel('opt result')
plt.show()

headers=[4,10,20,30,40,50]
table=np.c_[np.transpose([constraints]),optimization_result]
print(tabulate(table,headers, tablefmt="github"))
```

Figure 1Error! Reference source not found. shows optimization value for different number of variable and constraints. In the table, First row shows number of variables and first column shows the number of constraints.

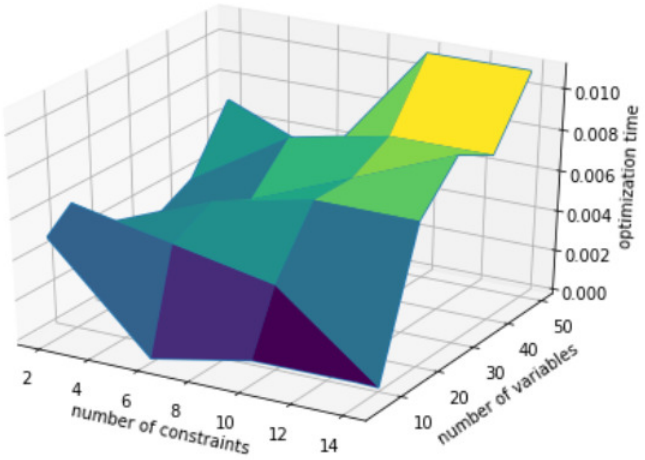


	4	10	20	30	40	50
2	10.4198	3.54167	0.394137	1.79259	0	0.105882
6	0	24.0794	14.6435	4.93041	3.19496	1.86235
10	0	4.33957	28.4747	28.9449	5.72457	8.75578
14	0	0	12.8691	25.5712	20.6176	15.137

Figure 1: optimization result for different number of variable and constraints

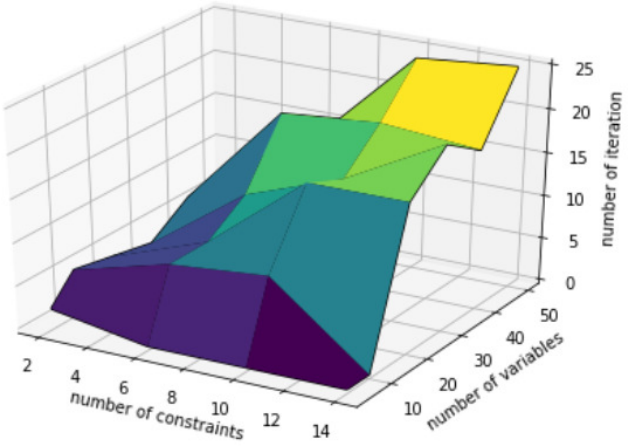
Tabulate and graph the time versus m and n

Figure 2 shows optimization time for different number of variable and constraints. In the table, first row shows the number of variables and first column shows the number of constraints. we also showed the effect of number of variable and constraints on number of iteration in Figure 3.



	4	10	20	30	40	50
2	0.00499034	0.00597835	0.00398874	0.00299168	0.00398922	0.00698209
6	0	0.00498676	0.00598407	0.00498152	0.00698066	0.004987
10	0.000997066	0.00398898	0.00698137	0.00698137	0.00797892	0.0109711
14	0.00103235	0	0.00694442	0.00901508	0.00797844	0.0109718

Figure 2: table and graph of optimization time versus different number of variables and constraints



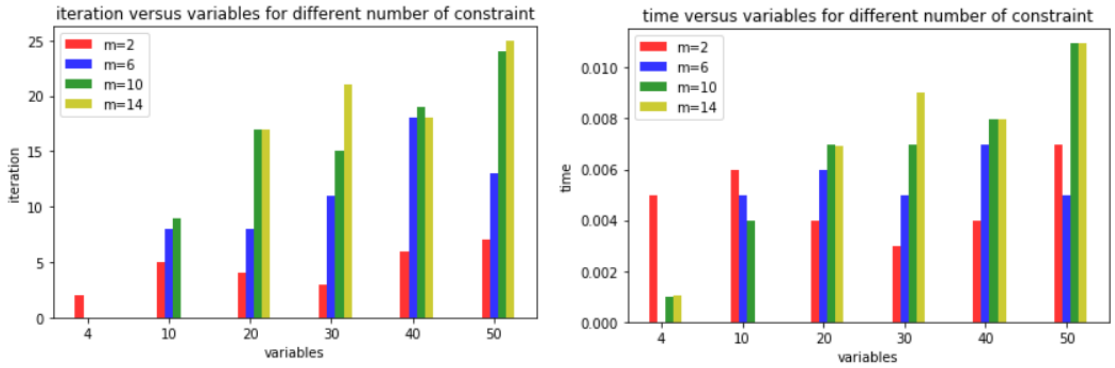
	4	10	20	30	40	50
2	2	5	4	3	6	7
6	0	8	8	11	18	13
10	0	9	17	15	19	24
14	0	0	17	21	18	25

Figure 3: table and graph of number of iteration versus different number of variables and constraints

An analysis of the results: what do you observe as the problem size and constraints vary?

To get more clear diagram we present the result in the format of Figure 4 and Figure 5. Bar charts in Figure 4 (part a and b) respectively show the iteration and time versus number of variables.

As it is clear, for higher value of n (n≥20), if the number of variables remains constant, by increasing number of constraints, time of algorithm and number of iteration will increase in average.

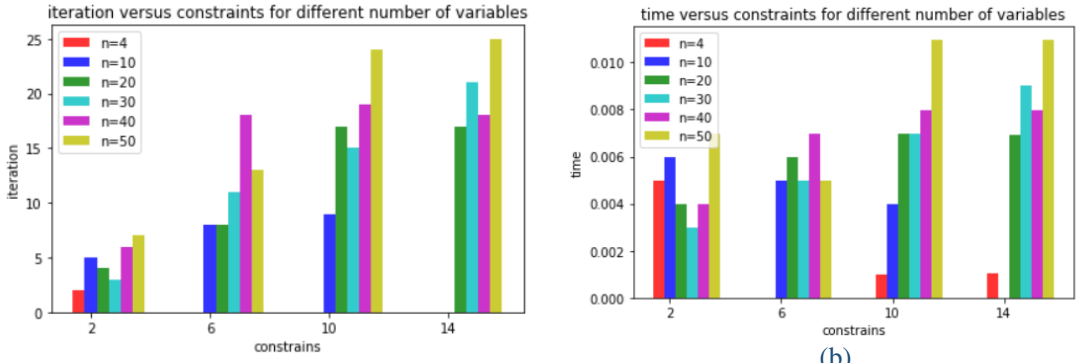


(a) (b)

Figure 4:a) number of iteration vs number of variables for different value of m (number of constraints) b) time of optimization vs number of variables for different value of m (number of constraints)

Figure 5 (part a and b) respectively show the iteration and time versus number of constraints. based on this diagram we can conclude that for higher value of m (m≥10), if the number of constraints remains

constant, by increasing number of variables, time of algorithm and number of iteration will increase linearly in average



(a) (b)

Figure 5: a) number of iteration vs number of constrains for different value of n (number of variables) b) time of optimization vs number of constrains for different value of m (number of variables)

conclusion

Although in the worst case, the simplex method requires an exponential number of pivots, in practical experience, the number of pivots seems to be linear in the number of variables and constraints, on average.