LINEAR PROGRAMMING PROJECT REPORT

1. My PUID is 33676933
   Because my last didgit is 3, I selected the **Model 2** as my problem.

   My LP model is

   Model 2: Pig Farming

   A farmer is raising pigs for market, and he wishes to determine the quantities of the available types of feed that should be given to each pig to meet certain nutritional requirements at a *minimum cost*. The number of units of each type of basic nutritional ingredient contained within a kilogram of each feed type is given in the following table, along with the daily nutritional requirements and feed costs:

Nutritional ingredient	*Kilogram of corn*	*Kilogram of tankage*	*Kilogram of alfalfa*	*Minimum daily requirement*
Carbohydrates	90	20	40	200
Protein	30	80	60	180
Vitamins	10	20	50	150
Cost (in cents)	35	30	25	

   Formulate the linear programming model for this problem.

   I formulated the above the question as shown below.

   Minimise 35x1+30x2+25x3
   S.T 90x1+20x2+40x3 >= 200
   30x1+80x2+60x3 >= 180
   10x1+20x2+50x3 >= 150
   x1,x2,x3 >= 0

2. **PYTHON CODE**

```
# Firstly, we should input whether the objective function is to maximise or minimise.
# If it is maximize, we enter 1, and the objective function is converted into a minimisation problem by multiplying by -1.
# if its minimize, just enter 2.
# I am taking the N matrix for the coefficients of the variables in the objective function.
# B and C matrix for the RHS values and coefficients of the constraints respectively.
# So we need to enter those values according the question given.
def input_N():
    inp=int(input("1.Maximize\n2.Minimize\nSelect an option(1 or 2) :"))
    input1=input("Input N matrix")
    input2=input('Input A matrix')
    input3=input('Input B matrix')

    n = input1.split()
    for i in range(len(n)):
        n[i]=int(n[i])
    if inp==1:
        for i in range(len(n)):
            n[i]=-1*n[i]

    a = input2.split()
    for i in range(len(a)):
        a[i]=int(a[i])
```

```
    b=input3.split()
    for i in range(len(b)):
        b[i]=int(b[i])
    rows=int(input("Enter number of rows of C matrix  :"))
    column=int(input("Enter no of cols of C matrix :"))
    print("Enter the elements of C :")
    c=[[int(input()) for i in range(column)]for j in range(rows)]
# here we have to enter the number of rows and columns of the C matrix of the given question
# and should also enter constraint element values, i.e. values after converting them into the standard form, and adding artificial variables.
# THUS THE 1st REQUIREMENT OF THE PROJECT GUIDELINE IS SATISFIED.

    print(n)
    print(a)
    print(b)
    for i in range(rows):
        print(c[i])
    arr=[]
    for i in range(len(c)):
        length=len(c[i])-len(a)
        col=c[i][length:]
        arr.append(col)
    for i in range(len(arr)):
        print(arr[i])

    if [1,0,0] in arr and [0,1,0] in arr and [0,0,1] in arr:
        print("Identity matrix available...")
    else:
        print("Identity matrix not available")
        # We are initially Checking if a basic feasible solution with identity matrix as corresponding basis is readily available or not.
        # But because that we are initially adding the artificial variables, we are obtaining an initial basis feasible solution(Identity matrix) ready hand.
        # SO THE 4th REQUIREMENT IN THE PROJECT GUIDELINES IS SATISFIED.

        # Thus we can move to the two phase method
        # The main use of the phase one is to get to an extreme point of the feasible region,
        # and in phase two, we move from this feasible point to an optimal point, taking the assumption that an optimum exists.



        # Phase1
        # In the phase 1, our objective function changes, we need to minimize only the artificial variables.


        # Inital Tableau
    name1=[]
    for i in range(len(n)):
        nm='X'+str(i+1)
        name1.append(nm)

    name2=[]
    for i in range(len(a)):
        nm='A'+str(i+1)
        name2.append(nm)
```

```
    namelis=name1+name2
    namelis.append('RHS')

    for i in range(len(n)):
        n[i]=0

    for i in range(len(a)):
        a[i]=-1*a[i]

    row1=n+a
    row1.append(0)
    for i in range(len(namelis)):
        print('{0: <5}'.format(namelis[i]),end='')
    print()
    for i in range(len(row1)):
        print('{0: <5}'.format(str(row1[i])),end='')
    print()

    for i in range(len(c)):
        for j in range(len(c[i])):
            print('{0: <5}'.format(str(c[i][j])),end='')
        print(b[i])
        # For the first table in phase 1, I have done according to text
book, where we first have -1 as the reduced cost of the artificial
variables
        # then by row operations we convert it into zero's.
        # after getting this table, then we can start the iteration part.

    #Step 1
    print("Step 1")
    for i in range(len(row1)):
        if(row1[i]==-1):
            for j in range(rows):
                if (c[j][i]==1):
                    row1[len(row1)-1]+=b[j]
                    for k in range(len(row1)-1):
                        row1[k]=row1[k]+c[j][k]


    namelis.insert(0,'X0')
    namelis.append("Min.R")
    row1.insert(0,'*')
    c[0].insert(0, 'A1')
    c[1].insert(0, 'A2')
    c[2].insert(0, 'A3')
    # so at first we have the artificial variables as our basis
    for i in range(len(namelis)):
        print('{0: <5}'.format(namelis[i]),end='')
    print()
    for i in range(len(row1)):
        print('{0: <5}'.format(str(row1[i])),end='')
    print()

    for i in range(len(c)):
        for j in range(len(c[i])):
            print('{0: <5}'.format(str(c[i][j])),end='')
        print(b[i])

    while(any(row1[i]>0 for i in range(1,len(row1)-1))):
        max=row1[1]
```

```
        maxindex=1
        for i in range(1,len(row1)-2):
            if row1[i]>max:
                max=row1[i]
                maxindex=i
        mr=[]
        for i in range(rows):
            if(c[i][maxindex]>0 ):
                mr.append(b[i]/c[i][maxindex])
                # According to the max positive value in the zeroth row, we
decide our pivoting column.
                # and according to the minimum ratio we decide which is the
pivoting row
                # thus we get our pivoting element.

        mrval=mr[0]
        mrindex=0
        mrcount=1
        indexls=[mrindex]
        for i in range(1, len(mr)):
            if mr[i]<mrval:
                mrval=mr[i]
                mrcount=1
                mrindex=i
                indexls=[mrindex]
            elif mr[i]==mrval:
                mrcount+=1
                indexls.append(i)
    # Here I am using the lexicographic rule to prevent the simplex method
from cycling.
        # So when the two or more minimum ratios are same, with the help of
lexicographic rule,
        # we are able to select the pivoting element.
        # THUS THE 5TH REQUIREMENT IN THE PROJECT GUIDELINES IS SATISFIED

        if mrcount!=1:
            pivls=[]
            for i in range(len(indexls)):
                lexval=c[indexls[i]][1]/c[indexls[i]][maxindex]
                pivls.append(lexval)

            minlex=pivls[0]
            minindex=0
            for i in range(len(pivls)):
                if pivls[i]<minlex:
                    minlex=pivls[i]
                    minindex=i

            pivindex=indexls[minindex]
        else:
            pivindex=mrindex

        pivotelement= c[pivindex][maxindex]
        print('Pivot element =',pivotelement)
        for i in range(1,len(row1)-2):
            c[pivindex][i]/=pivotelement
        b[pivindex]/=pivotelement


        mult = row1[maxindex]
        for i in range(1,len(row1)-2):
```

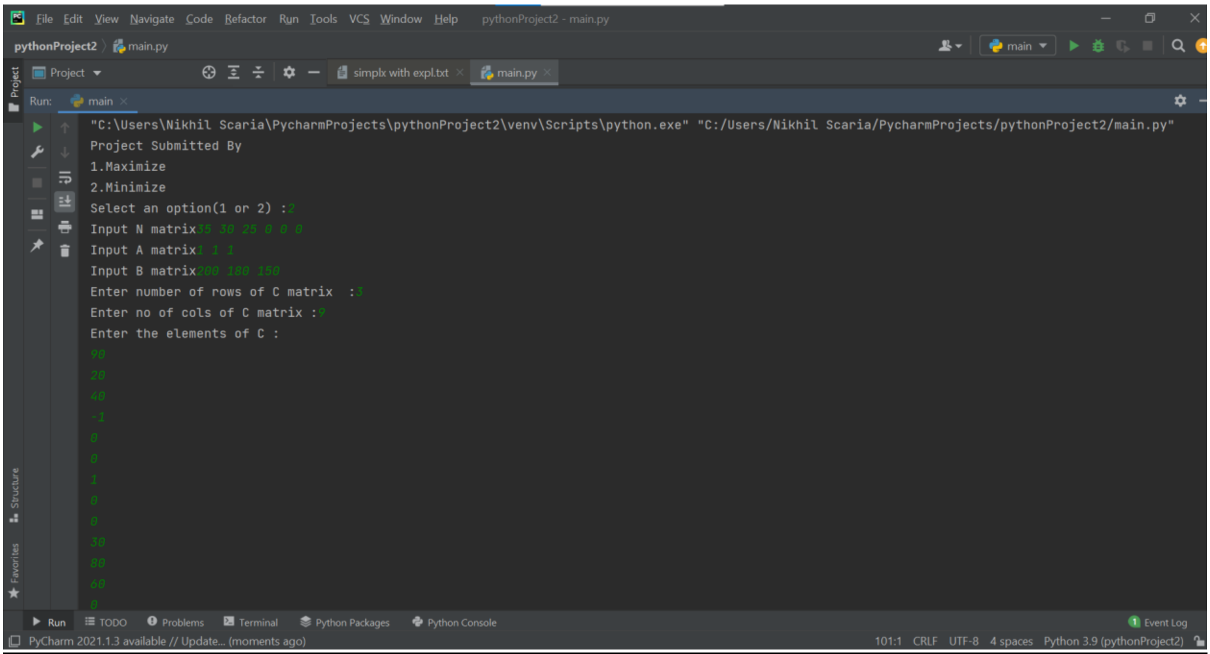
```
            row1[i]-=mult*c[pivindex][i]
        row1[len(row1)-1]-=mult*b[pivindex]
        print()
        for i in range(rows):
            if pivindex!=i:
                mult = c[i][maxindex]
                for j in range(1, len(row1) - 1):
                    c[i][j] -= (mult * c[pivindex][j])
                b[i] -= (mult * b[pivindex])

        c[pivindex][0]=namelis[maxindex]

        for i in range(1,len(row1)-1):
            row1[i]=round(row1[i],4)

        for i in range(len(c)):
            for j in range(1,len(c[i])):
                c[i][j]=round(c[i][j],4)

        for i in range(len(namelis)):
            print('{0: <10}'.format(namelis[i]), end=' ')
        print()
        for i in range(len(row1)):
            print('{0: <10}'.format(str(row1[i])), end=' ')
        print()

        for i in range(len(c)):
            for j in range(len(c[i])):
                print('{0: <10}'.format(str(c[i][j])), end=' ')
            print(b[i])
        # For checking feasibility
        # we are checking the RHS of the optimal tableau in the phase 1,
        # if the RHS value is zero, then the given LP is feasible, if not
its infeasible.
        # THUS THE 2nd REQUIREMENT IN THE PROJECT GUIDELINE IS SATISFIED.

    if(row1[len(row1) - 1] == 0):
        print("Since R.H.S is 0 the LP is feasible")
    else:
        print("LP is not feasible")


    for i in range(len(c)):
    if 'A' in c[i][0].split():
        if all(c[i][j] == 0 for j in range(1, len(c[i]))):
           print("The LP constraints are redundant. So, we remove the AR")
           del c[i]
           del b[i]
    else:
        print("No artificial variables present\n\n")
    # By the above code we check for Redundancy.
    # If there is an artificial variable in the basis of the final tableau
in phase 1,
    # and if there is any other legitimate non basic variable with zero
coefficient corresponding to the row of the artificial variable in the
basis, then that row is redundant, and we just remove that whole specific
row and continue to phase two.
    # if there is no such artificial variable in basis, then the LP is not
redundant.
    # THUS THE 3rd REQUIREMENT IN THE PROJECT GUIDELINE IS SATISFIED.
```

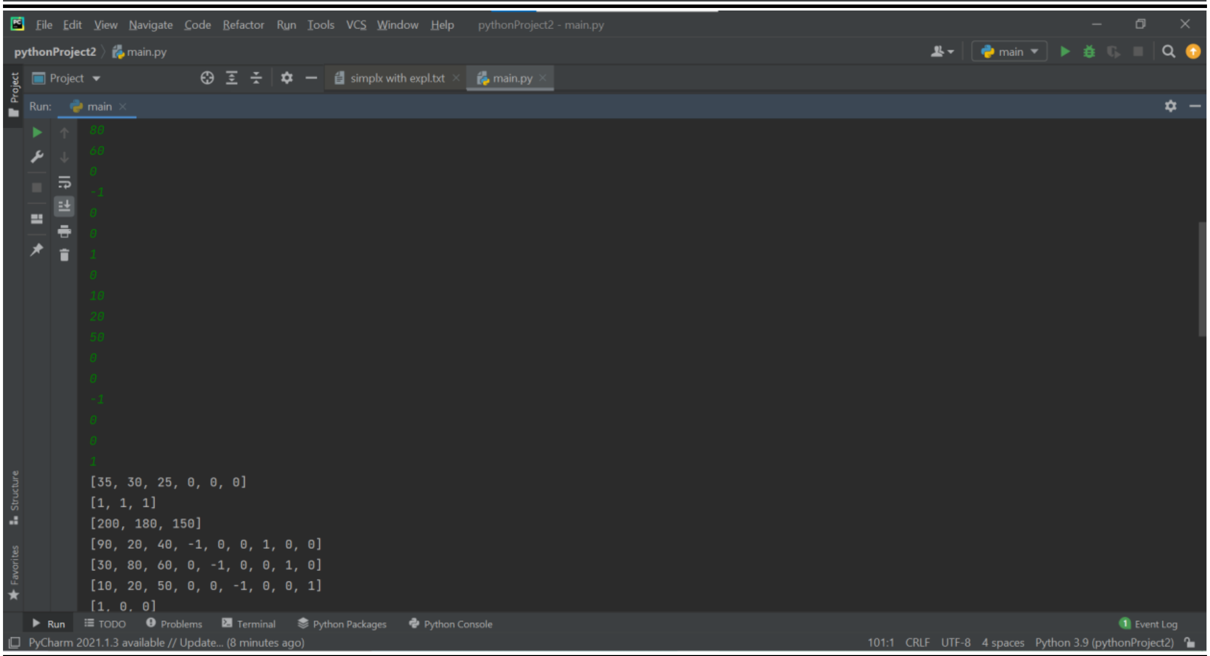
```python
print("Starting Phase 2")
# now we start the phase two, with the original objective function,
# and the artificial variables are removed.
numph=input1.split()
for i in range(len(numph)):
    numph[i]=-1*int(numph[i])

for i in range(1,len(numph)):
    row1[i]=numph[i-1]

for i in range(len(n)+1):
    print('{0: <10}'.format(namelis[i]),end='')
print('{0: <10}'.format(namelis[len(namelis)-2]))

for i in range(len(n)+1):
    print('{0: <10}'.format(str(row1[i])),end='')
print('{0: <10}'.format(row1[len(row1)-1]))

for i in range(len(c)):
    for j in range(len(n)+1):
        print('{0: <10}'.format(str(c[i][j])),end='')
    print(b[i])

for i in range(len(c)):
    for j in range(1,len(numph)):
        if c[i][0]==namelis[j]:
            if row1[j]!=0:
                multi=-1*row1[j]
                for k in range(len(c)):
                    if c[k][j]==1:
                        for p in range(1,len(numph)+1):
                            row1[p]=row1[p]+multi*c[k][p]
                        row1[len(row1)-1]+=multi*b[k]
for i in range(1,len(numph)+1):
    row1[i]=round(row1[i],4)

print()
for i in range(len(n)+1):
    print('{0: <10}'.format(namelis[i]),end='')
print('{0: <10}'.format(namelis[len(namelis)-2]))

for i in range(len(n)+1):
    print('{0: <10}'.format(str(row1[i])),end='')
print('{0: <10}'.format(row1[len(row1)-1]))

for i in range(len(c)):
    for j in range(len(n)+1):
        print('{0: <10}'.format(str(c[i][j])),end='')
    print(b[i])
if(any(row1[i]>0 for i in range(1,len(numph)))):
    while (any(row1[i] > 0 for i in range(1, len(numph) ))):
        max = row1[1]
        maxindex = 1
        for i in range(1, len(numph) ):
            if row1[i] > max:
                max = row1[i]
                maxindex = i



        if all(c[i][maxindex]<=0 for i in range(rows)):
```

```
                print("The given LP is unbounded....")
                recession = []
                for i in range(len(n)):
                    recession.append(0)
                recession[maxindex-1]=1
                for i in range(len(c)):
                    st=c[i][0].split()
                    nod=int(st[1])
                    recession[nod-1]=-1*(c[i][maxindex])
                print("Recession direction =",recession)
                break
            # with the above code, we check whether the LP is unbounded or
not, and if it is unbounded, the code will give us the recession direction
and the code gets terminated.
            # THUS THE 6th REQUIREMENT OF THE PROJECT GUIDELINE IS
SATISFIED.


            mr = []
            for i in range(rows):
                if (c[i][maxindex] > 0):
                    mr.append(b[i] / c[i][maxindex])

            mrval = mr[0]
            mrindex = 0
            mrcount = 1
            indexls = [mrindex]
            for i in range(1, len(mr)):
                if mr[i] < mrval:
                    mrval = mr[i]
                    mrcount = 1
                    mrindex = i
                    indexls = [mrindex]
                elif mr[i] == mrval:
                    mrcount += 1
                    indexls.append(i)

            if mrcount != 1:  # Here using Lexigraphic RULE
                pivls = []
                for i in range(len(indexls)):
                    lexval = c[indexls[i]][1] / c[indexls[i]][maxindex]
                    pivls.append(lexval)

                minlex = pivls[0]
                minindex = 0
                for i in range(len(pivls)):
                    if pivls[i] < minlex:
                        minlex = pivls[i]
                        minindex = i

                pivindex = indexls[minindex]
            else:
                pivindex = mrindex

            pivotelement = c[pivindex][maxindex]
            print('Pivot element =', pivotelement)
            for i in range(1, len(row1) - 2):
                c[pivindex][i] /= pivotelement
            b[pivindex] /= pivotelement

            mult = row1[maxindex]
```

```
            for i in range(1, len(numph)):
                row1[i] -= mult * c[pivindex][i]
            row1[len(row1) - 1] -= mult * b[pivindex]
            print()

            for i in range(rows):
                if pivindex != i:
                    mult = c[i][maxindex]
                    for j in range(1, len(numph)):
                        c[i][j] -= (mult * c[pivindex][j])
                    b[i] -= (mult * b[pivindex])

            c[pivindex][0] = namelis[maxindex]

            for i in range(1, len(numph)):
                row1[i] = round(row1[i], 4)

            for i in range(len(c)):
                for j in range(1, len(c[i])):
                    c[i][j] = round(c[i][j], 4)

            for i in range(len(namelis)):
                print('{0: <10}'.format(namelis[i]), end=' ')
            print()
            for i in range(len(row1)):
                print('{0: <10}'.format(str(row1[i])), end=' ')
            print()

            for i in range(len(c)):
                for j in range(len(c[i])):
                    print('{0: <10}'.format(str(c[i][j])), end=' ')
                print(b[i])
    else:
        print('We have Reached the optimal value.....! ')
        print("Optimal objective function value=" , row1[len(row1)-1])
        # Thus when all the values in the zeroth row become <= 0, the code
is terminated
        # and we have reached the optimal points with the optimal objective
function value.
        # THUS THE 6th REQUIREMENT OF THE PROJECT GUIDELINE IS SATISFIED.


input_N()
```

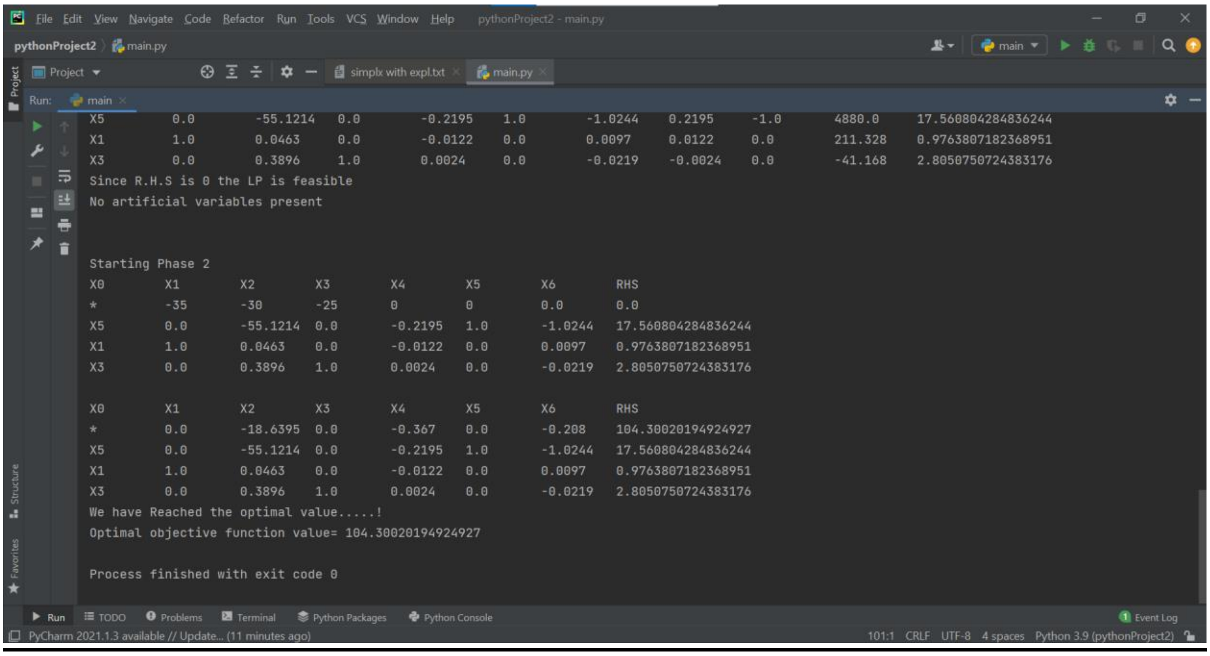
3. Output from the PYTHON code





```
[35, 30, 25, 0, 0, 0]
[1, 1, 1]
[200, 180, 150]
[90, 20, 40, -1, 0, 0, 1, 0, 0]
[30, 80, 60, 0, -1, 0, 0, 1, 0]
[10, 20, 50, 0, 0, -1, 0, 0, 1]
[1, 0, 0]
[0, 1, 0]
[0, 0, 1]
Identity matrix available...
X1   X2   X3   X4   X5   X6   A1   A2   A3   RHS
0    0    0    0    0    0    -1   -1   -1   0
90   20   40   -1   0    0    1    0    0    200
30   80   60   0    -1   0    0    1    0    180
10   20   50   0    0    -1   0    0    1    150
Step 1
X0   X1   X2   X3   X4   X5   X6   A1   A2   A3   RHS  Min.R
*    130  120  150  -1   -1   -1   0    0    0    530
A1   90   20   40   -1   0    0    1    0    0    200
A2   30   80   60   0    -1   0    0    1    0    180
A3   10   20   50   0    0    -1   0    0    1    150
Pivot element = 50
```

```
Pivot element = 50

X0        X1        X2        X3        X4        X5        X6        A1        A2        A3        RHS       Min.R
*         100.0     60.0      0.0       -1.0      -1.0      2.0       0.0       0.0       0         80.0
A1        82.0      4.0       0.0       -1.0      0.0       0.8       1.0       0.0       -40       80.0
A2        18.0      56.0      0.0       0.0       -1.0      1.2       0.0       1.0       -60       0.0
X3        0.2       0.4       1.0       0.0       0.0       -0.02     0.0       0.0       1         3.0
Pivot element = 18.0

X0        X1        X2        X3        X4        X5        X6        A1        A2        A3        RHS       Min.R
*         0.0       -251.1111 0.0       -1.0      4.5556    -4.6667   0.0       -5.5556   0         80.0
A1        0.0       -251.1111 0.0       -1.0      4.5556    -4.6667   1.0       -4.5556   4880.0    80.0
X1        1.0       3.1111    0.0       0.0       -0.0556   0.0667    0.0       0.0556    -60       0.0
X3        0.0       -0.2222   1.0       0.0       0.0111    -0.0333   0.0       -0.0111   13.0      3.0
Pivot element = 4.5556

X0        X1        X2        X3        X4        X5        X6        A1        A2        A3        RHS       Min.R
*         0.0       0.0       0.0       0.0       0.0       0.0       -1.0      -1.0      0         0.0
X5        0.0       -55.1214  0.0       -0.2195   1.0       -1.0244   0.2195    -1.0      4880.0    17.560804284836244
X1        1.0       0.0463    0.0       -0.0122   0.0       0.0097    0.0122    0.0       211.328   0.9763807182368951
X3        0.0       0.3896    1.0       0.0024    0.0       -0.0219   -0.0024   0.0       -41.168   2.8050750724383176
Since R.H.S is 0 the LP is feasible
No artificial variables present
```



Thus, we can see that we have got an optimal objective function value as 104.300201 and the optimal points are,

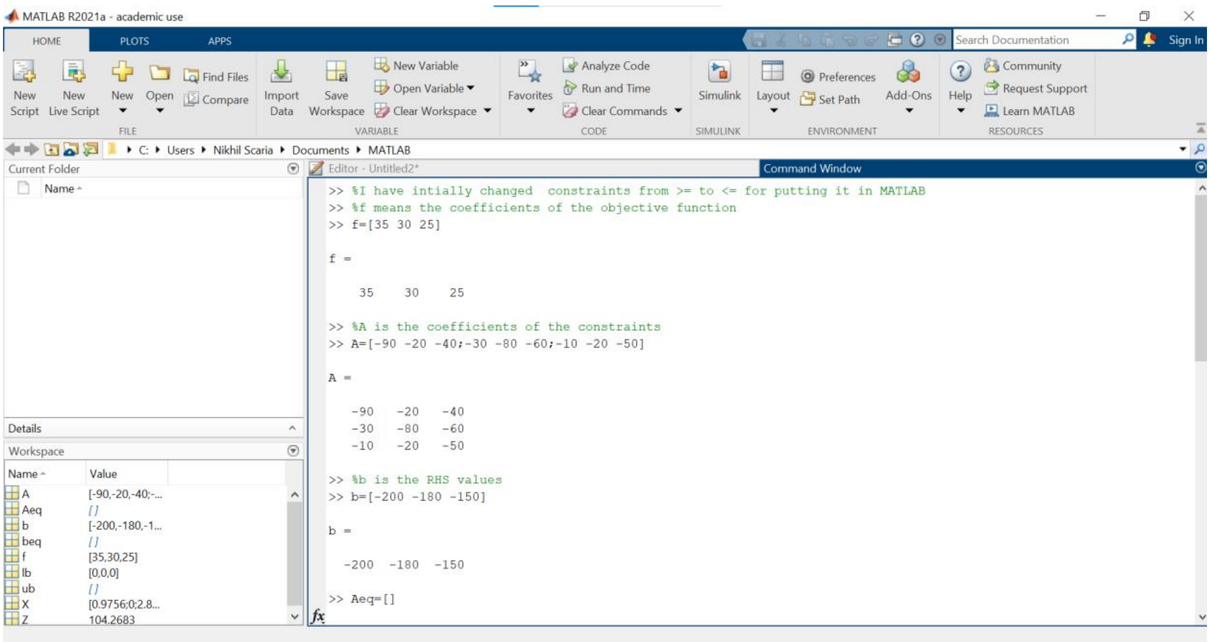
x1 = 0.9763807182

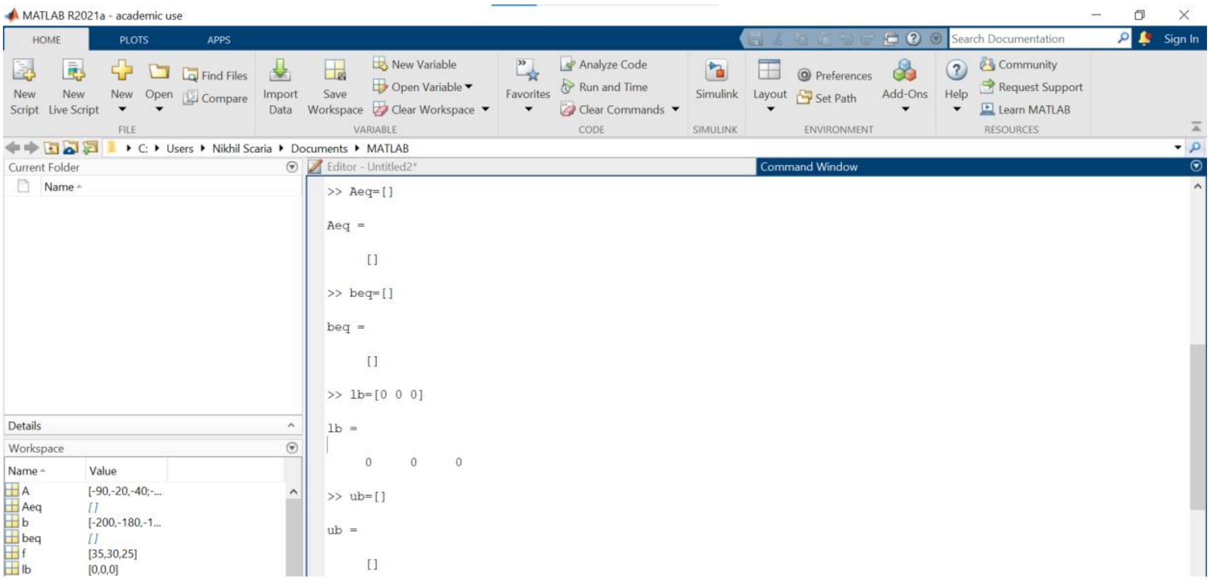
x2 = 0

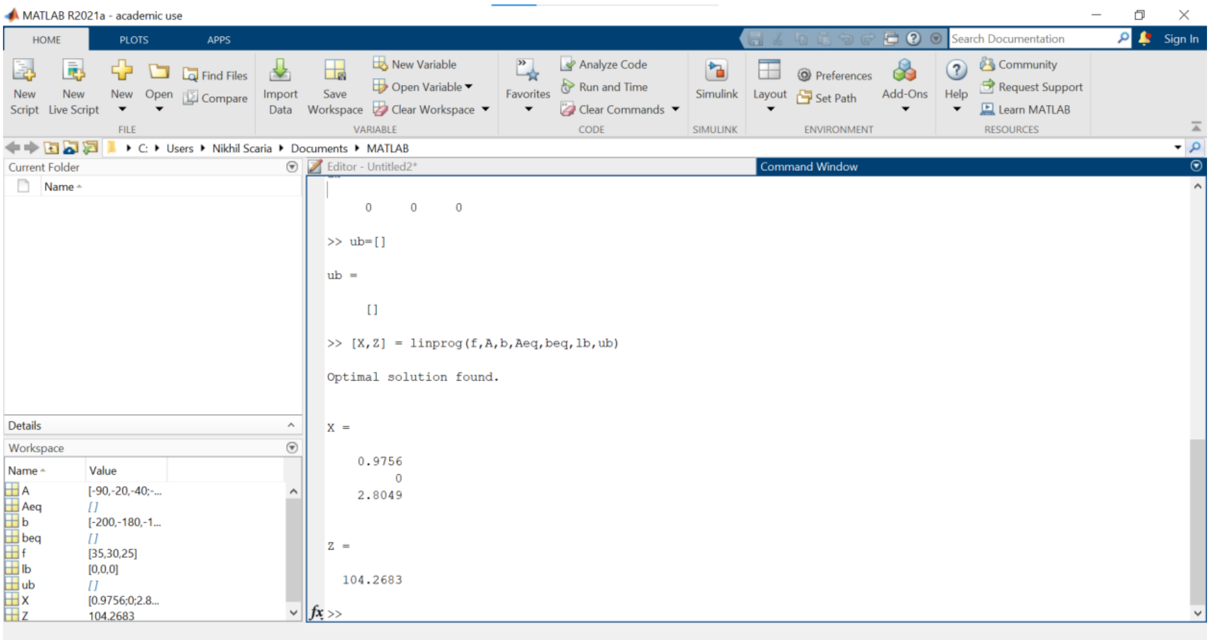
x3 = 2.8050750724

Thus, the answer for the given question would be 104 cents = $1.04

4. <u>USING COMMERCIAL SOLVER – MATLAB</u>







By using the MATLAB as commercial solver, we get optimal objective function value as 104.2683, and the optimal points as x1=0.9756, x2=0, x3=2.8049

Thus, the answer for the given question is 104.2683 cents = $1.04

Hence, we can see that we obtained the same value from the python code and from the commercial solver.