OPTIMIZATION TECHNIQUE

LAB CYCLE 1

SIMPLEX METHOD

CODE :

```
import numpy as np
from fractions import Fraction # so that numbers are not displayed in d
ecimal.

print("\n        ****SiMplex Algorithm ****\n\n")

# inputs

# A will contain the coefficients of the constraints
A = np.array([[1, 1, 0, 1], [2, 1, 1, 0]])
# b will contain the amount of resources
b = np.array([8, 10])
# c will contain coefficients of objective function Z
c = np.array([1, 1, 0, 0])

# B will contain the basic variables that make identity matrix
cb = np.array(c[3])
B = np.array([[3], [2]])
# cb contains their corresponding coefficients in Z
cb = np.vstack((cb, c[2]))
xb = np.transpose([b])
# combine matrices B and cb
table = np.hstack((B, cb))
table = np.hstack((table, xb))
# combine matrices B, cb and xb
# finally combine matrix A to form the complete simplex table
table = np.hstack((table, A))
# change the type of table to float
table = np.array(table, dtype ='float')
# inputs end

# if min problem, make this var 1
MIN = 0

print("Table at itr = 0")
print("B \tCB \tXB \ty1 \ty2 \ty3 \ty4")
for row in table:
  for el in row:
```

```
        # limit the denominator under 100
    print(Fraction(str(el)).limit_denominator(100), end ='\t')
  print()
print()
print("Simplex Working....")

# when optimality reached it will be made 1
reached = 0
itr = 1
unbounded = 0
alternate = 0

while reached == 0:

  print("Iteration: ", end =' ')
  print(itr)
  print("B \tCB \tXB \ty1 \ty2 \ty3 \ty4")
  for row in table:
    for el in row:
      print(Fraction(str(el)).limit_denominator(100), end ='\t')
    print()

  # calculate Relative profits-> cj - zj for non-basics
  i = 0
  rel_prof = []
  while i<len(A[0]):
    rel_prof.append(c[i] - np.sum(table[:, 1]*table[:, 3 + i]))
    i = i + 1

  print("rel profit: ", end =" ")
  for profit in rel_prof:
    print(Fraction(str(profit)).limit_denominator(100), end =", ")
  print()
  i = 0

  b_var = table[:, 0]
  # checking for alternate solution
  while i<len(A[0]):
    j = 0
    present = 0
    while j<len(b_var):
      if int(b_var[j]) == i:
        present = 1
        break;
      j+= 1
    if present == 0:
      if rel_prof[i] == 0:
        alternate = 1
```

```
      print("Case of Alternate found")
      # print(i, end =" ")
    i+= 1
  print()
  flag = 0
  for profit in rel_prof:
    if profit>0:
      flag = 1
      break
    # if all relative profits <= 0
  if flag == 0:
    print("All profits are <= 0, optimality reached")
    reached = 1
    break

  # kth var will enter the basis
  k = rel_prof.index(max(rel_prof))
  min = 99999
  i = 0;
  r = -1
  # min ratio test (only positive values)
  while i<len(table):
    if (table[:, 2][i]>0 and table[:, 3 + k][i]>0):
      val = table[:, 2][i]/table[:, 3 + k][i]
      if val<min:
        min = val
        r = i  # leaving variable
    i+= 1

    # if no min ratio test was performed
  if r ==-1:
    unbounded = 1
    print("Case of Unbounded")
    break

  print("pivot element index:", end =' ')
  print(np.array([r, 3 + k]))

  pivot = table[r][3 + k]
  print("pivot element: ", end =" ")
  print(Fraction(pivot).limit_denominator(100))

    # perform row operations
  # divide the pivot row with the pivot element
  table[r, 2:len(table[0])] = table[
      r, 2:len(table[0])] / pivot

  # do row operation on other rows
```

```
  i = 0
  while i<len(table):
    if i != r:
      table[i, 2:len(table[0])] = table[i,
        2:len(table[0])] - table[i][3 + k] * table[r, 2:len(table[0])]
    i += 1


  # assign the new basic variable
  table[r][0] = k
  table[r][1] = c[k]

  print()
  print()
  itr+= 1


print()

print("***************************************************************"
)
if unbounded == 1:
  print("UNBOUNDED LPP")
  exit()
if alternate == 1:
  print("ALTERNATE Solution")

print("optimal table:")
print("B \tCB \tXB \ty1 \ty2 \ty3 \ty4")
for row in table:
  for el in row:
    print(Fraction(str(el)).limit_denominator(100), end ='\t')
  print()
print()
print("value of Z at optimality: ", end =" ")

basis = []
i = 0
sum = 0
while i<len(table):
  sum += c[int(table[i][0])]*table[i][2]
  temp = "x"+str(int(table[i][0])+1)
  basis.append(temp)
  i+= 1
# if MIN problem make z negative
if MIN == 1:
  print(-Fraction(str(sum)).limit_denominator(100))
else:
```

```
  print(Fraction(str(sum)).limit_denominator(100))
print("Final Basis: ", end =" ")
print(basis)

print("Simplex Finished...")
print()
```

OUTPUT :

```
Iteration:  2
B       CB      XB      y1      y2      y3      y4
3       0       3       0       1/2     -1/2    1
0       1       5       1       1/2     1/2     0
rel profit:  0, 1/2, -1/2, 0,

pivot element index: [0 4]
pivot element:  1/2


Iteration:  3
B       CB      XB      y1      y2      y3      y4
1       1       6       0       1       -1      2
0       1       2       1       0       1       -1
rel profit:  0, 0, 0, -1,
Case of Alternate found

All profits are <= 0, optimality reached
```

```
                                        ****SiMplex Algorithm ****


Table at itr = 0
B       CB      XB      y1      y2      y3      y4
3       0       8       1       1       0       1
2       0       10      2       1       1       0

Simplex Working....
Iteration:  1
B       CB      XB      y1      y2      y3      y4
3       0       8       1       1       0       1
2       0       10      2       1       1       0
rel profit:  1, 1, 0, 0,

pivot element index: [1 3]
pivot element:  2
```

```
***************************************************************
ALTERNATE Solution
optimal table:
B       CB      XB      y1      y2      y3      y4
1       1       6       0       1       -1      2
0       1       2       1       0       1       -1

value of Z at optimality:  8
Final Basis:  ['x2', 'x1']
Simplex Finished...
```