Assignment-1

Sanyam Kaul: CS23MTECH14011

Mayuresh Rajesh Dindorkar: CS23MTECH14007

Shrenik Ganguli: CS23MTECH14014

Sreyash Mohanty: CS23MTECH14015

Assumption

- Polytope is non-degenerate.
- Polytope is bounded
- Rank of A is n
- Initial feasible point is given

Instructions to execute code: -

- Kindly re-start the kernel every-time before runnning the code

---

In [1]:
```
# ============================================================================
# Sanyam Kaul: CS23MTECH14011
# Mayuresh Rajesh Dindorkar: CS23MTECH14007
# Shrenik Ganguli: CS23MTECH14014
# Sreyash Mohanty: CS23MTECH14015
#
# Assumption
# 1. Polytope is non-degenerate.
# 2. Polytope is bounded
# 3. Rak of A is n
# 4. Initial feasible point is given
# ============================================================================

# importing libraries
import numpy as np
import numpy.linalg as la
import csv


class LO_Assignment_1:
    def __init__(self, m, n, A, B, C):
        self.epslion = 1e-8

        self.A = A
        self.B = B
        self.C = C
        self.X = np.empty([n])
        self.n = n
        self.m = m
        self.assert_input_dimesions()

    # Verifying that input has correct dimensions
```

```
    def assert_input_dimesions(self):

        assert(self.B.shape == (self.m,))
        assert(self.C.shape == (self.n,))
        assert(self.X.shape == (self.n,))

        self.execute_simplex_algorithm()

    #method to extract linearly independent rows
    def get_linearly_independ_rows(self, A, B, X):
        indices = np.where(np.abs((A @ X) - B) < self.epslion)[0]
        return A[indices], indices

    # calculating min ratio to calculate beta value
    def get_min_ratio(self, A_non_tight, B_non_tight, v):
        return (B_non_tight - np.dot(A_non_tight, self.X)) / ((A_non_tight @ v) + 1

    #extracting non tight rows to calculate beta value
    def extract_non_tight_rows(self, matrix, indices):
        return matrix[~np.isin(np.arange(len(matrix)), indices)]

    # Method used to move towards vertex
    def move_towards_vertex(self, negative_alphas_list, A_tight_row_matrix_inv, inc
        if len(negative_alphas_list) == 0:
           return None
        else:
            negative_alphas_list = negative_alphas_list[0]
            v = -A_tight_row_matrix_inv[negative_alphas_list]
            B_non_tight = self.extract_non_tight_rows(B, indices)
            A_non_tight = self.extract_non_tight_rows(A, indices)
            min_ratio = self.get_min_ratio(A_non_tight, B_non_tight, v)
            beta = np.min(min_ratio[min_ratio >= 0])
            return beta * v

    # Method to calculate direction vectos using linearly independent rows which is
    def get_direction_vector(self):
        A_tight_row_matrix, indices = self.get_linearly_independ_rows(self.A, self.
        A_tight_row_matrix_inv = la.inv(A_tight_row_matrix.T)
        alphas = (A_tight_row_matrix_inv @ self.C)
        negative_alphas_list = np.where(alphas < 0)[0]
        return self.move_towards_vertex(negative_alphas_list, A_tight_row_matrix_ir

    # Method to run simplex algorithm
    def execute_simplex_algorithm(self) -> None:
        while True:
            direction_vector= self.get_direction_vector()
            if direction_vector is not None:
                self.X = self.X + direction_vector
                print(f"Arrived at new point Z': {self.X}\tCost at this Z': {self.C
                print('-----------------------------------------------------------
            else:
                break

        print(f"Optimal Value (C.X) :{np.dot(self.C,self.X)}\tOptimal vertex (X) :{

if __name__ == '__main__':

    # Reading the csv file for test case input
    file_path = 'Assignment1.csv'
    with open(file_path, 'r') as file:
        reader = csv.reader(file)
        data = list(reader)

    # Extracting parameters from read csv file
```

```python
    X = np.array([float(x) for x in data[0][:-1]])
    C = np.array([float(x) for x in data[1][:-1]])
    B = np.array([float(x) for x in [row[-1] for row in data[2:]]])
    A = np.array([[float(x) for x in row[:-1]] for row in data[2:]])
    m = A.shape[0]
    n = A.shape[1]

    # Printing extracted parameters
    print('A:')
    print(A)
    print(f'B: {B}')
    print(f'C: {C}')
    print(f'X: {X}')
    print(f'm: {m}')
    print(f'n: {n}\n')

    # Calling constructor and passing parameters
    LO_Assignment_1(m, n, A, B, C)
```

```
A:
[[ 2. -1.]
 [ 1.  2.]
 [-1.  1.]
 [-1.  0.]
 [ 0. -1.]]
B: [4. 9. 3. 0. 0.]
C: [2. 5.]
X: [2. 0.]
m: 5
n: 2

Arrived at new point Z': [3.4 2.8]      Cost at this Z': 20.79999999999328

--------------------------------------------------------------------------------
Arrived at new point Z': [1. 4.]        Cost at this Z': 21.999999999991466

--------------------------------------------------------------------------------
Optimal Value (C.X) :21.999999999991466 Optimal vertex (X) :[1. 4.]
```

In [ ]: