Assignment-4

Sanyam Kaul: CS23MTECH14011

Mayuresh Rajesh Dindorkar: CS23MTECH14007

Shrenik Ganguli: CS23MTECH14014

Sreyash Mohanty: CS23MTECH14015

Assumption

- Rank of A is n

Instructions to execute code: -

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

In [5]:

```
# ============================================================================
# Sanyam Kaul: CS23MTECH14011
# Mayuresh Rajesh Dindorkar: CS23MTECH14007
# Shrenik Ganguli: CS23MTECH14014
# Sreyash Mohanty: CS23MTECH14015
#
#
#
# Assumption
# 1. Polytope is non-degenerate.
# 2. Rank of A is n
# ============================================================================


import csv
import numpy as np
from numpy import linalg as la

class LO_Assignment_4():

    def __init__(self, A, B, C, z, m, n ):
        self.m = m
        self.n = n
        self.A = A
        self.B = B
        self.C = C
        self.eps = 1e-8
        self.X = z.reshape((n,1))

        self.check_dimensions(self.A, self.B, self.C, self.X, self.n, self.m)

    # Checking input dimensions
    def check_dimensions(self, A, B, C, X, n, m):
        try:
            assert(A.shape == (m, n))
            assert(B.shape == (m, 1))
            assert(C.shape == (n, 1))
            assert(X.shape == (n, 1))
```

```
        except AssertionError:
            self._return_error()
            return

        self.B = self.get_non_degenerate()  # Making LP non degenerate
        print(self.B)
        if self.B is not None:
            print("Non-feasible problem\n") if self.n != np.linalg.matrix_rank(self

    # checking non-degeneracy
    def get_non_degenerate(self) -> np.ndarray:
        itr = 0
        while True:
            i = self.m - self.n
            B_ = self.B
            if (itr < 1000):
                # Perturbing B by adding noise

                B_[:i] = self.B[:i] + np.random.uniform(1e-6, 1e-5, size=(i,1))
                itr += 1

            else:
                # Checking for a larger range
                B_[:i] = self.B[:i] + np.random.uniform(0.1, 10, size=(i,1))

            Z = np.dot(self.A, self.X) - B_
            inds = np.where(np.abs(Z) < self.eps)[0]
            if len(inds) == self.n:  # Converted to non degenerate
                break
            print(itr)
        return B_

    # method to check point feasibility
    def check_point_feasibility(self, A, X, B):
        return np.all(np.dot(A, X) <= B)

    # check if any constraint is tight
    def check_any_constraint_tight(self, A, X, B):
        return True if np.any(B - np.dot(A, X) < pow(10, -4)) else False

    # Approaching polytope boundary
    def approach_polytope_boundary(self, A, X, B, C):

        dir_vec = C
        alpha = 0.01
        while not self.check_any_constraint_tight(A, X, B):
            if not self.check_point_feasibility(A, (X + alpha * dir_vec), B):
                alpha /= 10
            else:
                print('\nApproaching boundry: -')
                cost = np.dot(X.T, C)
                print("X: X.ravel()\t Cost: cost.ravel()")
                X = X + alpha * dir_vec
        return X

    def obtain_initial_vertex(self, A, B):
        rank = la.matrix_rank(A)
        initial_vertex = np.dot(la.pinv(A[:rank]), B[:rank])
        return initial_vertex

    def calculate_alpha_value(self, A, X, B, C):
        get_independent_rows = lambda A, X, B: (B - np.dot(A, X) < self.eps).T[0]
        lin_ind = A[get_independent_rows(A, X, B)]
        alpha = np.dot(la.pinv(lin_ind.T), C)
```

```
        return alpha

    def calculate_beta(self, A, X, B, is_min_ratio_positive, min_ratio):
        beta = np.min(((B - np.dot(A, X)).T[0] / min_ratio + 1e-12)[is_min_ratio_po
        return beta

    # method to check vertex optimality
    def find_optimal_vertex(self, A, X, B, C):
        if np.all(self.calculate_alpha_value(A, X, B, C) >= 0):
            calculated_cost = np.dot(X.T, C).ravel()
            print(f"X: {X.ravel()}\t cost: {calculated_cost}")
            return X
        get_independent_rows = lambda A, X, B: (B - np.dot(A, X) < pow(10, -4)).T[0
        tight_rows_matrix = get_independent_rows(A, X, B)
        direction_matrix = -la.pinv(A[tight_rows_matrix])
        print('dm', direction_matrix.shape)
        cost_list = []
        for i in range(0, direction_matrix.shape[1], 1):
            direction_vector = direction_matrix[:, i].reshape(-1, 1)
            min_ratio = (np.dot(A, direction_vector)).T
            is_min_ratio_positive = min_ratio > 0
            if np.any(is_min_ratio_positive):
                beta = self.calculate_beta(A, X, B, is_min_ratio_positive, min_rati
                z_prime = X + direction_vector * beta
                calculated_cost = np.dot(z_prime.T, C)
                cost_list.append((calculated_cost, i, z_prime))
                print(f'Z_prime: {z_prime.ravel()}\t cost: {calculated_cost}')
            else:
                print("This is unbounded case")
                return []

        _, index, z_prime = max(cost_list)
        return self.find_optimal_vertex(A, z_prime, B, C)

    # method to execute simplex algo
    def execute_simplex_algo(self):
        temp_A, temp_B, temp_C = self.A, self.B, self.C
        if self.X.shape != self.C.shape:
            temp_A = np.append(np.append(self.A, np.zeros((1, self.n)), axis = 0),
            temp_A[-1][-1] = -1
            temp_B = np.append(self.B, [abs(min(self.B))], axis = 0)
            temp_C = np.zeros((self.n +1, 1))
            temp_C[-1] = 1

        self.X = self.approach_polytope_boundary(temp_A, self.X, temp_B, temp_C)
        print('\nReached polytope boundary, now approaching vertex: -\n')
        self.X = self.obtain_initial_vertex(temp_A, temp_B)
        opt_vertex = self.find_optimal_vertex(temp_A, self.X, temp_B, temp_C)
        if len(opt_vertex) == 0:
            print('Polytope is unbounded- optimal value does not exit\n')
        else:
            print(f'Optimal vertex: {self.X.T[0]}\n')

#Calculating initial feasible point feasible point
def feasible_point( A, B, C, m, n) -> np.ndarray:
    inds = np.where(B < 0)[0]
    if len(inds) == 0:
        # Start at the origin, when LP has inequalities with positive
        # right-hand sides
        return np.zeros(C.shape)
    else:
        for _ in range(
                1000):  # Calculate X such that all constraints are satisfied
            rand_rows = np.random.choice(m, n)
```

```
            A_rand = A[rand_rows]
            B_rand = B[rand_rows]
            try:
                A_inv = la.inv(A_rand)
                X_ = np.dot(A_inv, B_rand)
                X_inds = np.where(np.abs(X_) < 1e-8)[0]
                Z = np.dot(A, X_) - B

                pos_rows = np.where(Z > 0)[0]
                if ((len(X_inds) == A_rand.shape[1]) and (
                        len(pos_rows) <= 0)):   # Infeasible
                    raise Exception("Infeasible\n")
                elif (len(pos_rows) > 0):
                    continue
                else:
                    return X_
            except la.LinAlgError:
                continue
        raise Exception("Infeasible\n")  # Infeasible

def read_linear_programming_input(file_path):
    with open(file_path, 'r') as file:
        reader = csv.reader(file)
        data = list(reader)

    # Extracting data
    c = np.array([float(x) for x in data[0][:-1]])
    b = np.array([float(x) for x in [row[-1] for row in data[1:]]])
    A = np.array([[float(x) for x in row[:-1]] for row in data[1:]])
    return c, b, A

def main():
    file_path = 'Assignment4.csv'
    C, B, A = read_linear_programming_input(file_path)

    m = A.shape[0]
    n = A.shape[1]
    z = feasible_point(A, B, C, m, n)
    B = B.reshape((m,1))
    C = C.reshape((n,1))

    print(f'Initial feasible point z: {z}\n')
    print(f'Cost Vector C: {C}')
    print(f'Constraint Vector B: {B}\n')
    print(f'Co-efficient Matrix A:\n {A}')

    LO_Assignment_4(A, B, C, z, m, n)

if __name__=="__main__":
    main()
```

```
Initial feasible point z: [0. 0.]

Cost Vector C: [[1.]
 [2.]]
Constraint Vector B: [[1.]
 [4.]
 [0.]
 [0.]]

Co-efficient Matrix A:
 [[ 1. -3.]
 [-1.  2.]
 [-1.  0.]
 [ 0. -1.]]
[[1.00000608]
 [4.00000198]
 [0.        ]
 [0.        ]]

Reached polytope boundary, now approaching vertex: -

dm (2, 4)
Z_prime: [-14.0000181   -5.00000806]     cost: [[-24.00003422]]
Z_prime: [-3.40000503e+01 -5.86197757e-14]       cost: [[-34.00005034]]
This is unbounded case
Polytope is unbounded- optimal value does not exit
```

In [ ]: