TEMA

April 23, 2021

1 Tema 1

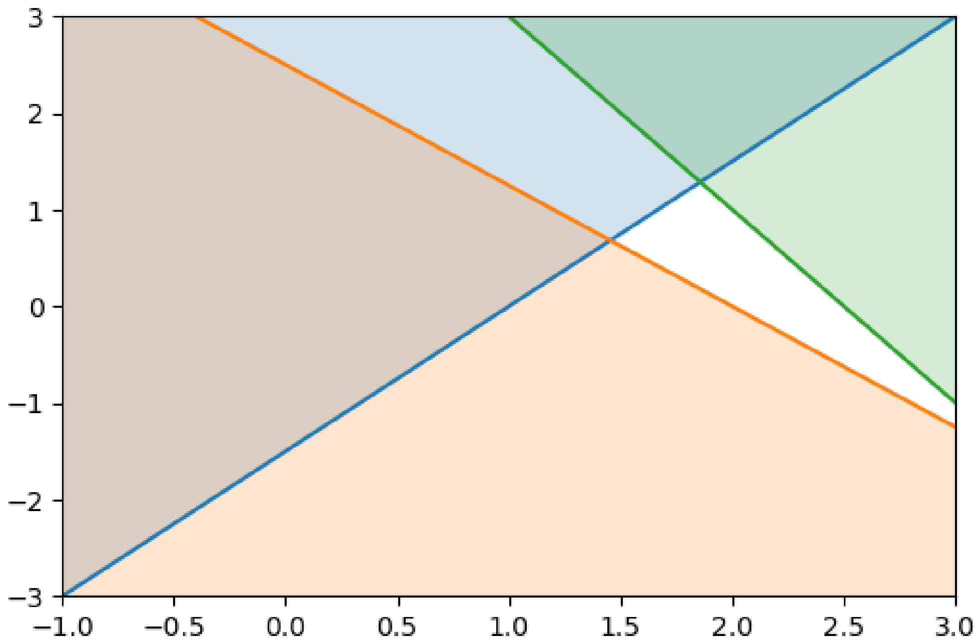
1.0.1 Reprezentarea grafica a tipurilor de programe liniare

[1]:
```
import numpy as np
import matplotlib.pyplot as plt # pentru grafice
import copy
```

Voi face cate un caz pentru fiecare tip de program liniar. 1. Incompatibil(fara solutii admisibile) 2. Compatibil (cu solutii admisibile) 2.1. Cu optim finit 2.1.1. Cu solutie optima unica 2.1.2 Cu o infinitate de solutii optime 2.2 Cu optim infinit

[2]:
```
'''
Pentru program incompatibil:
(max)f = 3 * x - y
3 * x - 2 * y >= 3 <=> y = (3 * x - 3) / 2
5 * x + 4 * y <= 10 <=> y = (10 - 5 * x) / 4
2 * x + y >= 5       <=> y = 5 - 2 * x
x >= 0, y >= 0
'''
x = np.linspace(-10, 10, 200)

plt.figure(dpi = 100)
plt.xlim(-1, 3)
plt.ylim(-3, 3)

y = (3 * x - 3) / 2
plt.plot(x, y)
plt.fill_between(x, y, y + 10, alpha = 0.2)


y = (10 - 5 * x) / 4
plt.plot(x, y)
plt.fill_between(x, y, y - 10, alpha = 0.2)

y = 5 - 2 * x
plt.plot(x, y)
plt.fill_between(x, y, y + 10, alpha = 0.2)
```

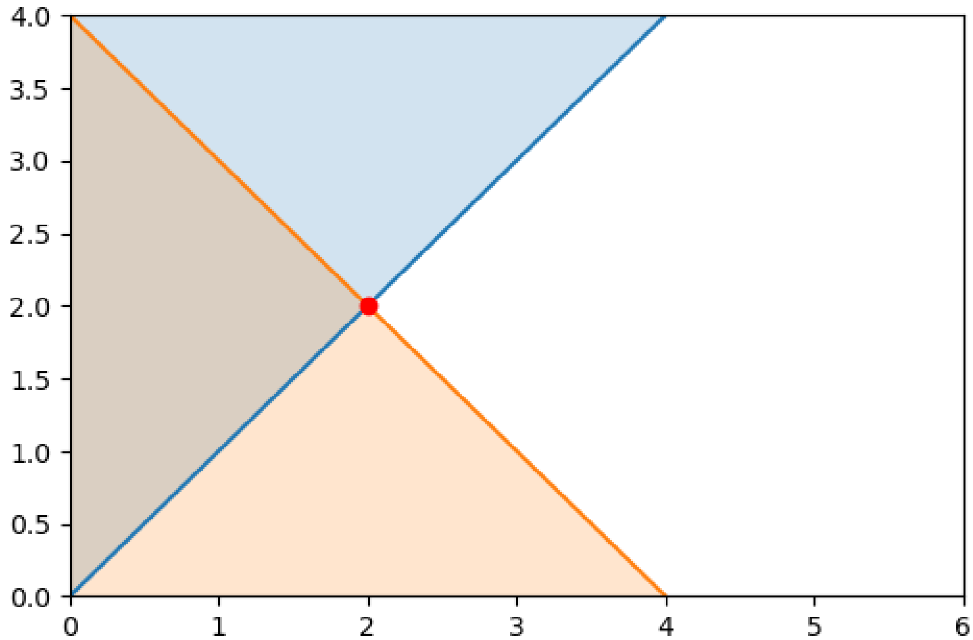
```
plt.show()
```



Dupa cum se vede in graficul de mai sus nu exista nicio zona in care graficele sa se intersecteze simultan => problema nu are solutii admisibile.

[3]:
```
'''
Program cu solutie optima finita
(max)f = x + y
x - y >= 0 <=> y = x
x + y <= 4 <=> y = 4 - x
x, y >= y

'''
x = np.linspace(-10, 10, 200)

plt.figure(dpi = 100)

plt.xlim(0, 6)
plt.ylim(0, 4)

y = x
plt.plot(x, y)
plt.fill_between(x, y, y + 10, alpha = 0.2)
```

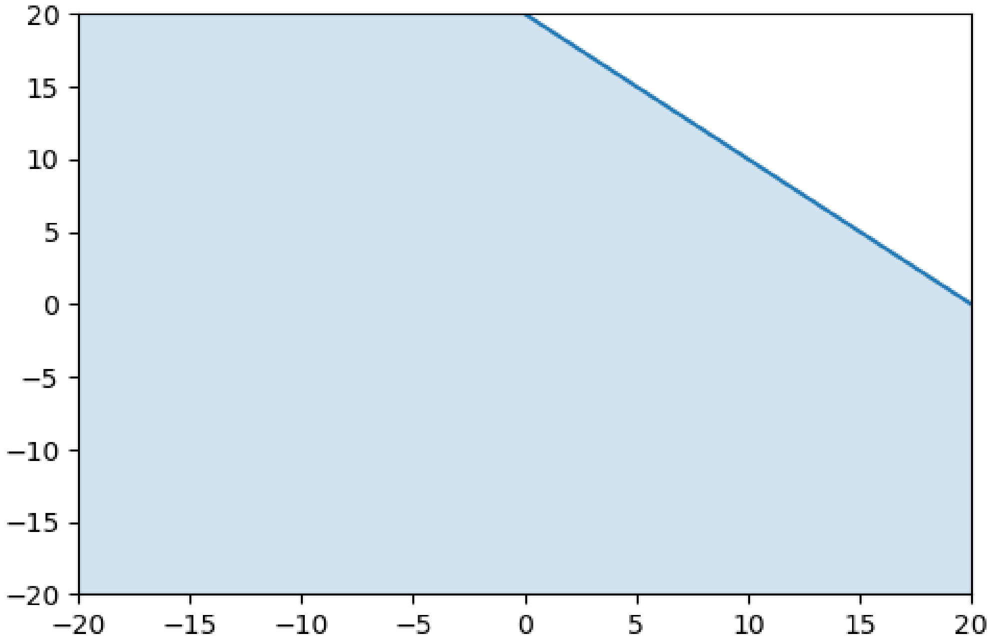
```
y = 4 - x
plt.plot(x, y)
plt.fill_between(x, y, y - 10, alpha = 0.2)


# plot-ez si punctul de optim (2, 6)
plt.scatter(2, 2, c='red', zorder=5)
plt.show()
```



Punctul de intersectie al dreptelor este si punctul ce maximizeaza functia

[4]:
```
'''
Sistem cu optim unic infinit
(max)f = x + y
x + y <= 20
'''

x = np.linspace(-20, 20, 200)

plt.figure(dpi = 100)

plt.xlim(-20, 20)
plt.ylim(-20, 20)
# x + y <= 20
y = 20 - x
```

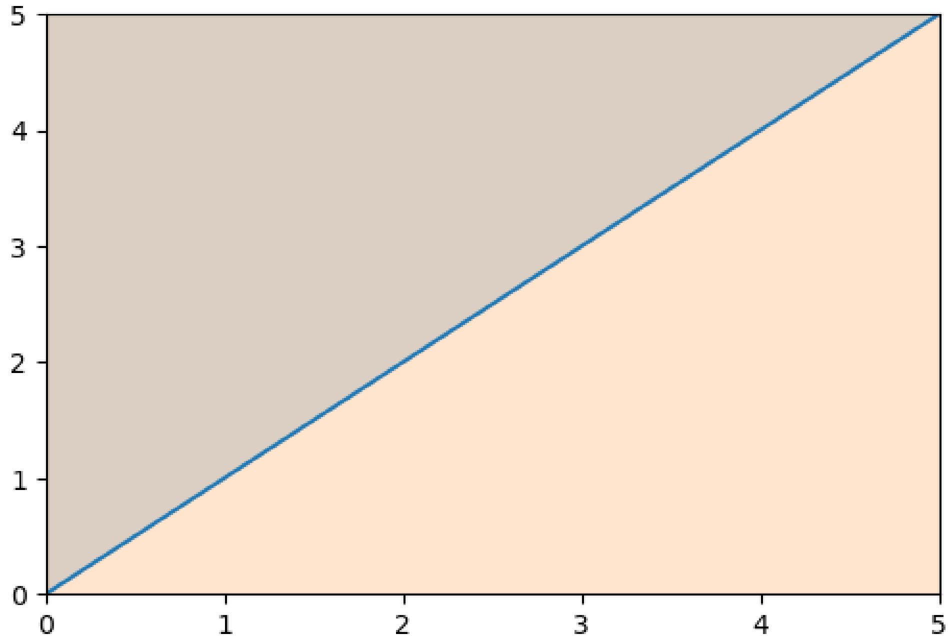
```
plt.plot(x, y)
plt.fill_between(x, y, y - 100, alpha = 0.2)
plt.show()
```



Toate punctele de pe dreapta x + y = 20 vor maximiza functia x + y

[5]:
```
'''
Program cu solutie optima infinita
(max)f = x  + y
x - y >= 0
y >= 4
x, y >= 0
'''

x = np.linspace(-10, 10, 200)
plt.figure(dpi = 100)
plt.xlim(0, 5)
plt.ylim(0, 5)

# x - y >= 0
y = x
plt.plot(x, y)
plt.fill_between(x, y, y + 10, alpha = .2)

# y >= 4
```

```
y = np.linspace(-10, 10, 200)
plt.plot(0 * y, y)
plt.fill_between(y, 0, 5, alpha = .2)

plt.show()
```



2 Tema 2

[10]:
```
import math
import numpy as np
```

Inversul unei matrici

[91]:
```
# Metoda pentru Gauss cu Pivotare Totala
def gauss_pivotare_totala(U, b, cerinta):
    # primim ca argumente matricea asociata sistemului si vectorul coloana b

    # verific daca sistemul are solutie unica prin calcularea determinantului
    if abs(np.linalg.det(U)) > 1e-15:
        # determinam dimensiunea matricei asociate sistemului
        n = U.shape[0]
        # concatenez vectorul coloana b la matricea sistemului U pentru a putea␣
↪aplica algoritmul
        U = np.append(U, b, axis = 1)
```

```
        # vector ce retine indicii coloanelor in vederea permutarii coloanelor(ce corespund vectorilor solutiilor)
        index = np.arange(0, n)
        for k in range(0, n-1):

            # se cauta elementul cu valoarea absoluta maxima din submatrice
            # amax gaseste maximul dintr-un array
            # where gaseste locul din matrice unde se afla acest maxim prin compararea fiecarul element cu maximul dat
            result_maxim = np.where(U[k:, k:n] == np.amax(U[k:, k:n]))
            coord = list(zip(result_maxim[0], result_maxim[1]))
            # determinam coordonatele corespunzatoare maximului
            # p -> linia
            # m -> coloana
            p = coord[0][0] + k # adunam k la aceste pozitii pentru a gasi pozitia lor raportata la toata matricea, nu la submatricea in care au fost cautate
            m = coord [0][1] + k
            # daca elementul gasit ca fiind maxim este egal cu 0 => Sistemul este incompatibil
            if U[p][m] == 0:
                if cerinta == "sistem":
                    return "Sistem incompatibil"
                elif cerinta == "inversa":
                    return "Nu se poate calcula inversa acestei matrici"

            if p != k:
                # daca indexul liniei pivotului este diferit de cel curent => trebuie sa interschimbam cele doua linii
                U[[k, p]] = U[[p, k]]

            if m != k:
                # daca indexul coloanei pivotului este diferit de cel curent => trebuie sa interschimbam cele doua coloane
                # deoarece am interschimbat doua coloane => trebuie sa schimbam si ordinea necunoscutelor sistemului cu ajutorul vectorului index
                U[:, [k, m]] = U[:, [m, k]]
                index [m], index [k] = index[k], index[m]

            # aplic pivotarea pe coloana curenta sub pivot
            for l in range(k+1, n):
                U[l] = U[l] - (U[l][k] / U[k][k]) * U[k]

        # daca ultimul element de pe ultima linie si ultima coloana este zero(corespunzator matricei sistemului) => sistemul este incompatibil
        if U[n-1][n-1] == 0:
```

```
            # in functie de cerinta afisez un mesaj corespunzator
            if cerinta == "sistem":
                return "Sistem incompatibil"
            elif cerinta == "inversa":
                return "Nu se poate calcula inversa acestei matrici"


        A = np.copy(U) # ii fac o copie matricei U
        U = A[0:, 0:n] # extrag matricea U asociata sistemului dupa ce s-a facut pivotarea
        size_b = b.shape[1] # aflu cate coloane are b
        solution = np.zeros((n, size_b)) # vector solutie cu numarul de linii egal cu n(numarul de linii din U) si coloane egal cu size_b(numarul de coloane din b)
        for i in range(size_b):
            # trebuie sa rezolv fiecare sistem in parte(in cazul unui sistem va fi de rezolvat doar unul)
            y = x = np.zeros((n, 1))
            # extragem toata matricea asociata lui b (la un sistem va fi un vector coloana, dar la inversa b va fi tot o matrice din care vom extrage pe rand cate o coloana)
            b = A[0:, n + i]
            # aplic algoritmul de substitutie descendenta(deoarece matricea U obtinuta este o matrice superior triunghiulara)
            # verificare pentru afisare
            if isinstance(sub_descendenta(U, b), int):
                if cerinta == "sistem":
                    return "Sistem incompatibil"
                elif cerinta == "inversa":
                    return "Nu se poate calcula inversa acestei matrici"
            else:
                y = sub_descendenta(U, b)
                for j in range(n):
                    x[index[j]] = y[j]
                for l in range(n):
                    solution[l][i] = x[l]


        # returnez solutia
        return solution
    else:
        # pe aceasta ramura va intra daca determinantul va fi mai mic decat acel prag dat
        if cerinta == "sistem":
            return "Sistem incompatibil"
        elif cerinta == "inversa":
```

```
            return "Nu se poate calcula inversa acestei matrici"
```

[92]:
```
def sub_ascendenta(U, b):

    # Algoritm pentru metoda substitutiei ascendente(elementele de deasupra␣
 ↪diagonalei principale sunt egale cu 0)
    # dimensiunea matricei asociate sistemului(este patratica)
    n = U.shape[0]
    # initializam x(vectorul solutie) cu zero
    x = np.zeros((n, 1))
    # daca determinantul este egal cu 0(aproximarea sa) => sistemul nu are␣
 ↪solutie unica
    # altfel  => are solutie unica(este compatibil determinat)
    # punem conditie ca acesta sa fie mai mare decat o aproximare a lui 0 data␣
 ↪de noi(in modul)
    if abs(np.linalg.det(U)) > 1e-15:
        # merg de la prima ecuatie catre ultima
        for i in range(0, n):
            suma = 0
            for j in range(i+1):
                # calculez suma produselor din fiecare ecuatie, exceptand␣
 ↪elementul ce trebuie aflat
                suma = suma + U[i][j] * x[j]
            suma = (b[i] - suma) / U[i][i]
            x[i] = suma
        return x
    else:
        # returnez -1 daca determinantul este egal cu 0
        return -1

def sub_descendenta(U, b):
    # Algoritm pentru metoda substitutiei ascendente(elementele de sub␣
 ↪diagonala principala sunt egale cu 0)
    # dimensiunea matricei asociate sistemului(este patratica)
    n = U.shape[0]
    x = np.zeros((n, 1))
    if abs(np.linalg.det(U)) > 1e-15:
        # parcurgem ecuatiile de la ultima(ce are doar o necunoscuta) spre prima
        for i in range(n-1, -1, -1):
            suma = 0
            for j in range(i+1, n):
                # calculez suma produselor elementelor aflate pana la ecuatia␣
 ↪data
                suma = suma + U[i][j] * x[j]
            suma = (b[i] - suma) / U[i][i]
            x[i] = suma
```

```
        return x
    else:
        # returnez -1 daca sistemul nu are solutie unica
        return -1
```

[93]:
```
# Inversa unei matrici folosind Gauss cu Pivotare Totala
B = np.array([
    [1, -1, 1],
    [2, 0, 3],
    [1, 1, -2]
])
b = np.identity(B.shape[0])
print("Inversa matricii date este: ")
print(gauss_pivotare_totala(B, b, "inversa"))
```

```
Inversa matricii date este:
[[ 0.375  0.125  0.375]
 [-0.875  0.375  0.125]
 [-0.25   0.25  -0.25 ]]
```

3 Lema substitutiei

[10]:
```
B = np.array([
    [1., 0., -1., 2.],
    [0., 1., 2., -1.],
    [-1., 2., 0., 1.],
    [2., -1., 1., 1.]
])
b = np.identity(B.shape[0])
N = B.shape[0]

# calculez inversa matricei B
B_inv = gauss_pivotare_totala(B, b, "inversa")
print(f"Inversa matricei B")
print(B_inv)
C = np.array([2., 1., 1., -1.], dtype=np.float)

# coloana pe care vreau sa o inlocuiesc in vectorul B
k = 1

# dupa aceasta inlocuire a coloanei k => matricea B_tilda
B_tilda = B.copy()
# in loc de coloana k il pun pe C
B_tilda[:, k] = C
```

```
# Fie matricea Y = B^-1 * C
# Operatorul @ se foloseste pentru inmultirea matricelor in python
Y = B_inv @ C

# Trebuie sa generez vectorul v(parametru al matricei E(eta))
# Trebuie sa schimbal semnul elementelor lui y
v = -Y / Y[k]
v[k] = 1 / Y[k]

# Generez matricea E - mai intai generez o matrice unitate
E_k = np.identity(N)
# adaug vectorul v pe coloana k a matricei E_k
E_k[:, k] = v

# inversa matricei B_tilda este produsul dintre vectorul E_k si inversa lui B
print(f"Inversul matricei B_tilda")
print(E_k * B_inv)
```

```
Inversa matricei B
[[ 1.8  1.4 -1.2 -1. ]
 [ 1.4  1.2 -0.6 -1. ]
 [-1.2 -0.6  0.8  1. ]
 [-1.  -1.   1.   1. ]]
Inversul matricei B_tilda
[[ 1.8        -1.52727273 -0.         -0.        ]
 [ 0.          0.27272727 -0.         -0.        ]
 [-0.         -0.43636364  0.8         0.        ]
 [-0.         -0.68181818  0.          1.        ]]
```

4 Tema 3 - Algoritmul simplex primal

[63]:
```
def exista_nr_negative(A):
    # ma uit pe ultima coloana sa vad daca mai exista numere negative
    nr_col = A.shape[1]
    exista_negativ = False
    index_negativ = -1
    n = A.shape[0]
    min_neg = A[n - 1][0]
    # Cat timp nu mai exista numere negative pe ultima linie
    for i in range(nr_col):
        if A[n - 1][i] <= min_neg and A[n-1][i] < 0:
            exista_negativ = True
            min_neg = A[n-1][i]
            index_negativ = i
```

```python
    return min_neg, index_negativ, exista_negativ


def gaseste_pivot(A):
    n = A.shape[0]
    nr_col = A.shape[1]
    min_neg, index_negativ, exista_negativ = exista_nr_negative(A)
    while exista_negativ == True:
        # daca tot mai exista numere negative pe ultima linie
        # am gasit coloana pivotului => acum trebuie sa mai gasesc linia
        # trebuie sa merg pe coloana pivotului
        aux = np.zeros((n-1, 1))
        for i in range(n-1):
            aux[i] = A[i][nr_col - 1] / A[i][index_negativ]

        # acum am gasit si linia pivotului
        min_aux_index = 0
        minn = aux[0]
        for i in range(aux.shape[0]):
            if aux[i] <= minn:
                minn = aux[i]
                min_aux_index = i

        # pivotul este la intersectia liniei si coloanei pe care tocmai le-am gasit
        pivot = A[min_aux_index][index_negativ]
        # trebuie sa fac acest pivot 1
        m = 1 / pivot
        # acum trebuie sa inmultesc toata linia corespunzatoare pivotului
        # cu aceasta valoare
        for i in range(nr_col):
            A[min_aux_index][i] *= m
        # break

        # acum trebuie sa fac elementele de deasupra si de desubtul pivotului 0
        # parcurg matricea, fara insa a modifica si linia pivotului
        for i in range(n):
            if i != min_aux_index:
                m = A[i][index_negativ] / A[min_aux_index][index_negativ]
                for j in range(nr_col):
                    A[i][j] = A[i][j] - m * A[min_aux_index][j]

        min_neg, index_negativ, exista_negativ = exista_nr_negative(A)
        #break
    return A
```

```
def gaseste_valori(A, termen):
    # trebuie sa vad daca pe coloana lui x si y este doar un 1 si restul␣
↪valorilor sunt 0
    n_linii = A.shape[0]
    n_col = A.shape[1]
    nr_one = 0
    index_termen = -1
    nr_zeros = 0

    for i in range(n_linii):
        if A[i][termen] == 1:
            nr_one += 1
            index_termen = i
        if A[i][termen] == 0:
            nr_zeros += 1

    if nr_one == 1:
        x = A[index_termen][n_col - 1]
    else:
        return False

    return x




if __name__ == '__main__':

    '''
    max P = 8 * x_0 + 8 * x_1
    2 * x_0 + 1 * x_1 <= 3
    1 * x_0 + 2 * x_1 <= 9
    x, y >= 0
    '''



    # Matricile asociate sunt
    # Am dat direct forma standard
    A = np.array([
        [2., 1., 1., 0., 0.],
        [1., 2., 0., 1., 0.],
        [-8., -8., 0., 0., 1.]
    ])
    A_copy = A.copy()
    b = np.array([
```

```
        [3.], [9.], [0.]
    ])
    A = np.append(A, b, axis=1)
    A = gaseste_pivot(A)
    print(A)
    print("Punctele de optim sunt:")
    for i in range(A.shape[1] - 4):
        find_valoare = gaseste_valori(A, i)
        if find_valoare != False:
            print(f"x_{i} = {find_valoare}")
        else:
            print(f"x_{i} = 0")

    find_valoare = gaseste_valori(A, A.shape[1] - 2)
    if  find_valoare != False:
        print(f"P = {find_valoare}")
    else:
        print(f"P = 0")

    print("max P = 8 * x_0 + 8 * x_1")
    print("2 * x_0 + 1 * x_1 <= 3")
    print("1 * x_0 + 2 * x_1 <= 9")
    print("x, y >= 0")
```

```
[[ 2.  1.  1.  0.  0.  3.]
 [-3.  0. -2.  1.  0.  3.]
 [ 8.  0.  8.  0.  1. 24.]]
Punctele de optim sunt:
x_0 = 0
x_1 = 3.0
P = 24.0
max P = 8 * x_0 + 8 * x_1
2 * x_0 + 1 * x_1 <= 3
1 * x_0 + 2 * x_1 <= 9
x, y >= 0
```

5 Algoritm simplex dual

[89]:
```
A = np.array([
    [2., 1., 3., 6.],
    [1., 2., 4., 8.],
    [3., 1., -2., 4.],
    [1., 1., 3., None]
])
nr_variabile = A.shape[1] - 1
```

```
print("Programul este:")
A_copy = A.copy()
# afisarea problemei de minimizare
for i in range(A.shape[0]):
    for j in range(A.shape[1]):
        if i < A.shape[0] - 1:
            if j < A.shape[1] - 1:
                print(f"x{j} * {A[i][j]}", end=' ')
            if j < A.shape[1] - 2:
                print("+",end='\t')
            if j == A.shape[1] - 1:
                print(f" >= {A[i][j]}")
        else:
            if j < A.shape[1] - 1:
                print(f"x{j} * {A[i][j]}", end=' ')
            if j < A.shape[1] - 2:
                print("+",end='\t')
            if j == A.shape[1] - 1:
                print(f" = P")

# ----- ADUC MATRICEA DE INTRARE LA FORMA STANDARD ------
# Calculez matricea transpusa asocitata problemei
A_transpusa = np.transpose(A)
print(f"Matricea A transpusa este {A_transpusa}")

A[A.shape[0] - 1][A.shape[1] - 1] = 0
# coloana termenilor liberi
b = np.zeros((A.shape[0], 1))
for i in range(A.shape[0]):
    b[i][0] = A_transpusa[i][A.shape[1] - 1]
print(f"Termenii liberi sunt:\n {b}")
A = np.delete(A_transpusa, A_transpusa.shape[1] - 1, 1)
for i in range(A.shape[1]):
    A[A.shape[0] - 1][i] = A[A.shape[0] - 1][i] * (-1)
# trebuie sa adaug matricea identitate
# adaug la matrice si variabilele pe care le-am adaugat la inegalitate pentru a␣
 ↪ajunge la egalitate
A_identity = np.identity(A.shape[0])
A = np.append(A, A_identity, axis = 1)
A = np.append(A, b, axis=1)
print("Problema de minimizare")
# Forma problemei duale pe care voi aplica algoritmul simplex primal -␣
 ↪(transformata intr-o problema de maximizare)
for i in range(A.shape[0]):
    for j in range(A.shape[1]):
        if i < A.shape[0] - 1:
            if j < A.shape[1] - 1 and A[i][j]:
```

```
                print(f"x{j} * {A[i][j]}", end=' ')
            if j < A.shape[1] - 2 and A[i][j]:
                print("+",end='\t')
            if j == A.shape[1] - 1:
                print(f" = {A[i][j]}")
        if i == A.shape[0] - 1:
            if j == A.shape[1] - 2:
                print(f"{A[i][j]} * P", end=' ')
            elif j < A.shape[1] - 1 and A[i][j]:
                print(f"x{j} * {A[i][j]}", end=' ')
            if j < A.shape[1] - 2 and A[i][j]:
                print("+",end='\t')
            if j == A.shape[1] - 1:
                print(f" = 0")
A = gaseste_pivot(A)
print(nr_variabile)
print(A.shape[1])
print(A)
nr_cnt_variabile = 0
print("Punctele de optim sunt:")
for i in range(nr_variabile, A.shape[1]):
    if nr_cnt_variabile >= nr_variabile:
        break
    print(f"x{nr_cnt_variabile} = {A[A.shape[0] - 1][i]}")
    nr_cnt_variabile += 1

print(f"P = C = {A[A.shape[0] - 1][A.shape[1] - 1]}")
```

```
Programul este:
x0 * 2.0 +      x1 * 1.0 +      x2 * 3.0  >= 6.0
x0 * 1.0 +      x1 * 2.0 +      x2 * 4.0  >= 8.0
x0 * 3.0 +      x1 * 1.0 +      x2 * -2.0  >= 4.0
x0 * 1.0 +      x1 * 1.0 +      x2 * 3.0  = P
Matricea A transpusa este [[2.0 1.0 3.0 1.0]
 [1.0 2.0 1.0 1.0]
 [3.0 4.0 -2.0 3.0]
 [6.0 8.0 4.0 None]]
Termenii liberi sunt:
 [[1.]
 [1.]
 [3.]
 [0.]]
Problema de minimizare
x0 * 2.0 +      x1 * 1.0 +      x2 * 3.0 +      x3 * 1.0 +       = 1.0
x0 * 1.0 +      x1 * 2.0 +      x2 * 1.0 +      x4 * 1.0 +       = 1.0
x0 * 3.0 +      x1 * 4.0 +      x2 * -2.0 +     x5 * 1.0 +       = 3.0
x0 * -6.0 +     x1 * -8.0 +     x2 * -4.0 +     1.0 * P  = 0
```

```
3
8
[[1.0 0.0 1.6666666666666665 0.6666666666666666 -0.3333333333333333 0.0
  0.0 0.3333333333333333]
 [0.0 1.0 -0.33333333333333326 -0.3333333333333333 0.6666666666666666 0.0
  0.0 0.33333333333333337]
 [0.0 0.0 -5.666666666666666 -0.6666666666666666 -1.6666666666666667 1.0
  0.0 0.6666666666666667]
 [0.0 0.0 3.333333333333333 1.3333333333333333 3.3333333333333335 0.0 1.0
  4.666666666666667]]
Punctele de optim sunt:
x0 = 1.3333333333333333
x1 = 3.3333333333333335
x2 = 0.0
P = C = 4.666666666666667
```