Simplex

November 29, 2022

1 Proyecto de Programación Lineal

1.1 Programación del Método Simplex

Integrantes del equipo: - Karen Arteaga Mendoza, 190161 - Federico Santacruz González, 190438 - Leopoldo Rodríguez Díaz Infante, 189584 - José Alberto Márquez Luján, 187917 - Santiago Fernández del Castillo Sodi, 189210

[1]:
```
import numpy as np
import pandas as pd # Para imprimir
np.set_printoptions(formatter={'float': lambda x: "{0:0.1f}".format(x)}) # Para␣
↪que imprima nada más 3 decimales
```

[2]:
```
def matriz_t0(A, b, c, M):
    '''
    Método para obtener la tabla cero del problema usando el método
    de la Gran M.

    EJEMPLO DE USO:
    >>> A = np.array([[3, 4, 1, 0],
                      [ 2,-1, 0,-1]])
    >>> b = np.array([20,2])
    >>> c = np.array([1, 1, 0, 0])
    >>> M = 100
    >>> matriz_t0(A, b, c, M)
    array([[  3.,   4.,   1.,   0.,   1.,   0.,  20.],
           [  2.,  -1.,   0.,  -1.,   0.,   1.,   2.],                    ␣
↪                                                [  1.,   1.,   0.,   0.,␣
↪100., 100.,   0.]])
    '''
    m = len(A)
    canon = np.eye(m, dtype=int)                        # Matriz identidad mxm
    mat1 = np.concatenate((A,canon), axis=1)            # Pegamos A con la␣
↪identidad
    cr = np.append(c,[M]*m)                             # Vector de costos␣
↪relativos
    mat1 = np.concatenate((mat1, np.array([cr])))       # Pegamos la matriz con␣
↪los costos relativos
```

```
    b_ext = np.array([np.append(b, 0)])              # Construcción del vector b

    # Regresamos la matriz extendida con b
    return np.concatenate((mat1, b_ext.T), axis=1).astype('float64')
```

[3]:
```
def reglaDeBland(tablaSimplex):
    '''
    Queremos la columna más a la izquierda con cr < 0.
    Si hay empate en el criterio de la variable de salida,
    elegimos la más arriba
    '''
    cr = tablaSimplex[-1][:-1] # costos relativos
    busq = np.where(cr < 0)[0]

    if len(busq) == 0: # No encontró; fin del problema
        return -1, -1
    else:
        colSal = busq[0]

    bk = tablaSimplex[:, -1]
    yk = tablaSimplex[:, colSal]

    by = np.empty(0)
    for b,y in zip(bk,yk):
        if y > 0:
            by = np.append(by, b/y)
        else:
            by = np.append(by, -1)

    valid = np.where(by >= 0)[0]

    if len(valid) == 0: # Todas las variables son menores que cero
        return -1, -2

    renglonSal = valid[by[valid].argmin()]

    return renglonSal, colSal
```

[4]:
```
def pivoteo(tablaSimplex):
    '''
    Dada una tabla Simplex, este método pivotea sobre el elemento
    que dictamina la regla de Bland y regresa el resultado.
    '''
    renglonSal, colSal = reglaDeBland(tablaSimplex)

    if colSal < 0: # Condiciones para detenerse
        return tablaSimplex, colSal
```

```
    m = len(tablaSimplex)

    valorPivoteo = tablaSimplex[renglonSal][colSal]
    tablaSimplex[renglonSal] = tablaSimplex[renglonSal] / valorPivoteo

    for i in range(m):
        if i != renglonSal and tablaSimplex[i][colSal] != 0:
            tablaSimplex[i] -= tablaSimplex[i][colSal] *␣
↪tablaSimplex[renglonSal]

    return tablaSimplex, 1
```

[5]:
```
def checkEmptiness(table, M):
    '''
    Método que recibe una tabla final y evalúa, usando
    el criterio de la Gran M, si la región del problema original
    es vacía o no. Si sí es vacía, regresa True; si no, False.
    '''
    m = len(table)

    table = np.array(table)

    ylast = table[-1][-m:-1]

    if np.any((ylast == 0)):
        for col in np.where(ylast == 0)[0]:
            column = table[:,col]
            if column.size == np.count_nonzero((column==0) | (column==1)) and 1␣
↪== np.count_nonzero(column==1):
                y = np.where(column == 1)[0]
                if table[:,-1][y] > 0:
                    return True

    return False
```

[6]:
```
def solver(A,b,c,M=100):
    '''
    Método para resolver un PPL planteado en su forma estándar. Utiliza
    el método de la Gran M y Simplex. Regresa la tabla en su forma final
    y el resultado de la función objetivo.
    '''
    t0 =  matriz_t0(A, b, c,M)

    for i in range(len(A)):
        t0[-1] = t0[i]*(-M) + t0[-1]
```

```
    # print("Tabla inicial:")
    # printMat(t0)
    # print('')

    contador = 1
    t1, z = pivoteo(t0) # Aquí z es la columna de salida y la usamos como␣
↪control para saber si terminó.

    while z >= 0:
        t1, z = pivoteo(t1)
        contador += 1

    if checkEmptiness(t1,M):
        print("ESPACIO DE SOLUCIÓN VACÍO; última versión de la tabla:")
        return t1, np.nan

    if z == -2: # no está acotado
        print("PROBLEMA NO ACOTADO; última versión de la tabla:")
        return t1, np.nan

    print(f"Tabla final después de {contador} iteraciones:")
    return t1, (-1)*t1[-1][-1]
```

[7]:
```
def printMat(t):
    df = pd.DataFrame(t)
    with pd.option_context('display.max_rows', None,
                           'display.max_columns', None,
                           'display.precision', 3,):
        display(df)
```

1.2 Pruebas con los problemas planteados por los alumnos

[8]:
```
c = np.array([1, 1, 0, 0])
A = np.array([[3, 4, 1, 0],
              [ 2,-1, 0,-1]])
b = np.array([20,2])
M = 100

t1, z = solver(A, b, c, M)
printMat(t1)
print(f"El valor de la función objetivo es: {z}")
```

```
Tabla final después de 4 iteraciones:
```

	0	1	2	3	4	5	6
0	0.0	5.5	1.0	1.5	1.0	-1.5	17.0

```
1  1.0 -0.5  0.0 -0.5    0.0   0.5   1.0
2  0.0  1.5  0.0  0.5  100.0  99.5  -1.0
```

```
El valor de la función objetivo es: 1.0000000000002132
```

[9]:
```
c1 = np.array([0, -9, -1, 0, 2, 1])
A1 = np.array([[0, 5, 50, 1, 1, 0],
               [1, -15, 2, 0, 0, 0],
               [0, 1, 1, 0, 1, 1]])
b1 = np.array([10, 2, 6])

t1, z1 = solver(A1, b1, c1, M)
printMat(t1)
print(f"\nEl valor de la función objetivo es: {z1}")
```

```
Tabla final después de 5 iteraciones:
```

	0	1	2	3	4	5	6	7	8	9
0	0.0	1.0	10.0	0.2	0.2	0.0	0.2	0.0	0.0	2.0
1	1.0	0.0	152.0	3.0	3.0	0.0	3.0	1.0	0.0	32.0
2	0.0	0.0	-9.0	-0.2	0.8	1.0	-0.2	0.0	1.0	4.0
3	0.0	0.0	98.0	2.0	3.0	0.0	102.0	100.0	99.0	14.0

```
El valor de la función objetivo es: -14.0
```

[10]:
```
c2 = np.array([-3, 1, 0, 0])
A2 = np.array([[-1, 1, 1, 0],
               [2, 2, 0, -1]])
b2 = np.array([5, 4])
t2, z2 = solver(A2, b2, c2, M)

printMat(t2)
print(f"\nEl valor de la función objetivo es: {z2}")
```

```
PROBLEMA NO ACOTADO; última versión de la tabla:
```

	0	1	2	3	4	5	6
0	0.0	2.0	1.0	-0.5	1.0	0.5	7.0
1	1.0	1.0	0.0	-0.5	0.0	0.5	2.0
2	0.0	4.0	0.0	-1.5	100.0	101.5	6.0

```
El valor de la función objetivo es: nan
```

[11]:
```
c3 = np.array([-40, -30, 0, 0])
A3 = np.array([[1, 1, 1, 0],
               [2, 1, 0, 1]])
b3 = np.array([12, 16])
t3, z3 = solver(A3, b3, c3, M)
```

```
printMat(t3)
print(f"\nEl valor de la función objetivo es: {z3}")
```

```
Tabla final después de 3 iteraciones:
     0    1     2     3      4      5      6
0  0.0  1.0   2.0  -1.0    2.0   -1.0    8.0
1  1.0  0.0  -1.0   1.0   -1.0    1.0    4.0
2  0.0  0.0  20.0  10.0  120.0  110.0  400.0


El valor de la función objetivo es: -400.0
```

1.3 Pruebas con los problemas planteados por el profesor

[12]:
```
# a) es un problema que no está acotado
c = np.array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,␣
 ↪1,-1,-1,0,0,0,0,0,0,0])
b = np.array([2,2,0,0,0,2])
A = np.array([
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0,␣
 ↪0, 0],
    [4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 0, 0, 0, 0,-1, 0, 0,␣
 ↪0, 0],
    [0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 0, 1,-1, 0,-1, 0,␣
 ↪0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,-1, 1, 0, 0,-1,␣
 ↪0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,-1, 0, 0,␣
 ↪0,-1, 0],
    [2, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,-1, 0, 0, 0,␣
 ↪0, 1]])

t, z = solver(A, b, c, 1000)
printMat(t)
print(f"\nEl valor de la función objetivo es: {z}")
```

```
PROBLEMA NO ACOTADO; última versión de la tabla:
    0    1    2    3    4    5    6    7    8    9    10   11   12   13   14  \
0  0.5  0.0  2.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0
1  0.5  1.0 -1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
2  0.0  0.0 -8.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
3  0.0  0.0 -4.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
4  0.0  0.0 -4.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
5 -2.0  0.0 -8.0 -4.0 -4.0 -4.0 -4.0 -4.0 -4.0 -4.0 -4.0 -4.0 -4.0 -4.0 -4.0
6  1.0  0.0  0.0  2.0  2.0  2.0  2.0  2.0  2.0  2.0  2.0  2.0  2.0  2.0  2.0
```

```
    15   16   17   18   19   20   21   22     23   24   25     26      27  \
0  1.0  2.0  1.0  1.0  0.0  0.0  0.0  0.0 -0.25  0.0  0.0 -0.25     1.0
1  0.0 -1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.25  0.0  0.0  0.25     0.0
2  0.0 -8.0  0.0  0.0  0.0  0.0  0.0  0.0  2.00  1.0 -1.0  0.00     0.0
3  0.0 -4.0  0.0  0.0  1.0  0.0  0.0  0.0  1.00  0.0 -1.0  0.00     0.0
4  0.0 -4.0  0.0  0.0  0.0 -1.0  1.0  0.0  1.00  0.0  0.0  0.00     0.0
5 -4.0 -8.0 -4.0  0.0  0.0  0.0  0.0  1.0  1.00  0.0  0.0  1.00     0.0
6  2.0  0.0  2.0  0.0  0.0  0.0  0.0  0.0  0.50  0.0 -1.0 -0.50  1001.0

       28      29      30      31      32   33
0     0.0    0.25     0.0     0.0   -0.25  1.5
1     0.0   -0.25     0.0     0.0    0.25  0.5
2     0.0   -2.00    -1.0     1.0    0.00  0.0
3     0.0   -1.00     0.0     1.0    0.00  0.0
4     0.0   -1.00     0.0     0.0    0.00  0.0
5    -1.0   -1.00     0.0     0.0    1.00  0.0
6  1000.0  999.50  1000.0  1001.0  999.50  1.0

El valor de la función objetivo es: nan
```

[13]:
```
# b) es un problema con región vacía
c = np.array([3, 6, -1, 2, 0, 0, 0, 0])
b = np.array([2, 10, 6, 5])
A = np.array([
    [1, 1, -1, 0, -1, 0, 0, 0],
    [1, 1, 2, 3, 0, 0, 1, 0],
    [1, 2, -1, 2, 0, 0, 0, 1],
    [0, 1, 0, 2, 0, -1, 0, 0]])

t, z = solver(A, b, c, 1000)
printMat(t)
print(f"\nEl valor de la función objetivo es: {z}")
```

```
ESPACIO DE SOLUCIÓN VACÍO; última versión de la tabla:
    0    1    2    3       4      5    6      7    8       9       10   11  \
0  0.0  0.5  0.0  1.0     0.0   -0.5  0.0    0.0  0.0     0.0     0.0  0.5
1  1.0  1.0 -1.0  0.0     0.0    1.0  0.0    1.0  0.0     0.0     1.0 -1.0
2  0.0 -1.5  3.0  0.0     0.0    0.5  1.0   -1.0  0.0     1.0    -1.0 -0.5
3  0.0  0.0  0.0  0.0    -1.0   -1.0  0.0   -1.0  1.0     0.0    -1.0  1.0
4  0.0  2.0  2.0  0.0  1000.0  998.0  0.0  997.0  0.0  1000.0  1997.0  2.0

       12
0     2.5
1     1.0
2     1.5
3     1.0
4 -1008.0
```

```
El valor de la función objetivo es: nan
```

[14]:
```python
# c) el valor de la función objetivo es 7.
c = np.array([3, 6, -1, 2, 7, 0, 0, 0])
b = np.array([2, 10, 5, 2])
A = np.array([
    [1, 1, -1, 0, 0, 0, 0, -1],
    [1, 1, 2, 3, 0, 1, 0, 0],
    [3, 0, 0, 1, -1, 0, 0, 0],
    [0, 1, 2, 0, 0, 0, 1, 0]])

t, z = solver(A, b, c, 1000)
printMat(t)
print(f"\nEl valor de la función objetivo es: {z}")
```

```
Tabla final después de 7 iteraciones:
```

	0	1	2	3	4	5	6	7	8	9	10	\
0	0.0	1.0	-1.0	-0.333	0.333	0.0	0.0	-1.0	1.0	0.0	-0.333	
1	0.0	0.0	3.0	3.000	0.000	1.0	0.0	1.0	-1.0	1.0	0.000	
2	1.0	0.0	0.0	0.333	-0.333	0.0	0.0	0.0	0.0	0.0	0.333	
3	0.0	0.0	3.0	0.333	-0.333	0.0	1.0	1.0	-1.0	0.0	0.333	
4	0.0	0.0	5.0	3.000	6.000	0.0	0.0	6.0	994.0	1000.0	1001.000	

	11	12
0	0.0	0.333
1	0.0	8.000
2	0.0	1.667
3	1.0	1.667
4	1000.0	-7.000

```
El valor de la función objetivo es: 6.999999999998712
```

1.4 Pruebas del profesor para la sesión en el aula

el 29 de noviembre de 2022

[15]:
```python
# A)
c = np.array([-1000, -100, -10, -1, 0, 0, 0, 0])
b = np.array([1, 100, 10000, 1000000])
A = np.array([
    [1,      0,  0, 0, 1, 0, 0, 0],
    [20,     1,  0, 0, 0, 1, 0, 0],
    [200,   20,  1, 0, 0, 0, 1, 0],
    [2000, 200, 20, 1, 0, 0, 0, 1]])
```

```
t, z = solver(A, b, c, 1000)
printMat(t)
print(f"\nEl valor de la función objetivo es: {z}")
```

```
Tabla final después de 10 iteraciones:
```

	0	1	2	3	4	5	6	7	8	9	10	\
0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	
1	20.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	
2	200.0	20.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	
3	2000.0	200.0	20.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	
4	1000.0	100.0	10.0	0.0	0.0	0.0	0.0	1.0	1000.0	1000.0	1000.0	

	11	12
0	0.0	1.0
1	0.0	100.0
2	0.0	10000.0
3	1.0	1000000.0
4	1001.0	1000000.0

```
El valor de la función objetivo es: -1000000.0
```

[16]:
```
# B)
c = np.append(np.array([1]*18), [-1, 0])
b = np.array([4, 1])
A = np.array([np.append(np.array([1]*19), [0]), np.append(np.array([1]*18), [0,␣
 ↪-1])])

t, z = solver(A, b, c, 1000)
printMat(t)
print(f"\nEl valor de la función objetivo es: {z}")
```

```
Tabla final después de 3 iteraciones:
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	\
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
1	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

	15	16	17	18	19	20	21	22
0	0.0	0.0	0.0	1.0	1.0	1.0	-1.0	3.0
1	1.0	1.0	1.0	0.0	-1.0	0.0	1.0	1.0
2	0.0	0.0	0.0	0.0	2.0	1001.0	998.0	2.0

```
El valor de la función objetivo es: -2.0
```

[17]:
```
# C)
c = np.array([8, -2, 1, 2, 5, 0, 0, 0, 0, 0])
```

```
b = np.array([2, 1, 10, 6, 5])
A = np.array([
    [ 1, 1,-1, 0, 0,-1, 0, 0, 0, 0],
    [-2,-1, 0,-4, 1, 0,-1, 0, 0, 0],
    [ 1, 1, 2, 3, 0, 0, 0, 1, 0, 0],
    [ 1, 2,-1, 2, 0, 0, 0, 0, 1, 0],
    [ 0, 1, 0, 2, 0, 0, 0, 0, 0,-1]])

t, z = solver(A, b, c, 1000)
printMat(t)
print(f"\nEl valor de la función objetivo es: {z}")
```

```
ESPACIO DE SOLUCIÓN VACÍO; última versión de la tabla:
     0    1    2     3    4      5    6    7      8       9    10     11  \
0   0.0  1.0  0.0   2.0  0.0    1.0  0.0  0.0    1.0     0.0 -1.0    0.0
1  -2.0  0.0  0.0  -2.0  1.0    1.0 -1.0  0.0    1.0     0.0 -1.0    1.0
2   3.0  0.0  0.0  -3.0  0.0   -5.0  0.0  1.0   -3.0     0.0  5.0    0.0
3  -1.0  0.0  1.0   2.0  0.0    2.0  0.0  0.0    1.0     0.0 -2.0    0.0
4   0.0  0.0  0.0   0.0  0.0   -1.0  0.0  0.0   -1.0    -1.0  1.0    0.0
5  19.0  0.0  0.0  14.0  0.0  995.0  5.0  0.0  996.0  1000.0  5.0  995.0

       12      13   14      15
0     0.0     1.0  0.0     4.0
1     0.0     1.0  0.0     5.0
2     1.0    -3.0  0.0     2.0
3     0.0     1.0  0.0     2.0
4     0.0    -1.0  1.0     1.0
5  1000.0  1996.0  0.0 -1019.0


El valor de la función objetivo es: nan
```