UNIVERSIDAD LAICA ELOY ALFARO DE MANABÍ

INVESTIGACIÓN DE OPERACIONES

SÉPTIMO NIVEL PARALELO: A

ALUMNO(A):

MERA BURGOS ROSA MARÍA

TEMA:

DOCUMENTACIÓN DEL MÉTODO SIMPLEX EN PYTHON

DOCENTE: ING. JORGE ANIBAL MOYA DELGADO

FECHA: 24/02/2021

PERÍODO ACADÉMICO: 2020-2021 (2)

Manta – Manabí – Ecuador

ÍNDICE

1. DOCUMENTACIÓN DEL CÓDIGO EN PYTHON

- Para poder utilizar éste programa, es necesario cumplir con los siguientes requisitos:
  - Tener instalado **Python** (en lo posible la versión más reciente y estable 3.9.2), se lo puede hacer con el siguiente Link: https://www.python.org/ftp/python/3.9.1/python-3.9.1-amd64.exe
  - Tener instalado el módulo **Numpy**, desde la cmd de Windows colocar el siguiente comando: py -m pip install numpy
  - Opcionalmente tener instalado **Pandas** para visualizar las tablas de mejor manera: py -m pip install pandas

- Si ya se cuenta con lo anterior, se podrá correr el programa sin problemas. A continuación se explicará el código del programa que permite resolver ejercicios a través del método Simplex.

1.1. Librerías

Las siguientes librerías y módulos deberán ser importados principalmente, tal y como se muestra en la captura:

```
import sys
import numpy as np
from fractions import Fraction
```

- **Sys** es una librería nativa de Python, la cual utilizaremos ya que nos permite utilizar variables y funciones de Python, útiles para éste programa.
- **Numpy** es un módulo requerido en cuanto a que nos proporciona soporte para la creación de vectores y matrices, además de algunas funciones matemáticas.
- **Fraction** nos brindará el soporte necesario para que Python pueda reconocer números racionales o escritos en tipo de fracción.

1.2. Verificación de Panda

En la siguiente captura se ha declarado una sentencia Try-except, la cual permitirá asegurarse de que se está importando la librería “panda”, ya que se requiere para la formación de columnas y filas con etiquetas para las tablas que mostrarán los resultados:

```
try:
    import pandas as pd
    pandas_av = True
except ImportError:
    pandas_av = False
    pass
```

1.3. Variables y arreglos principales

Ahora se procede a declarar los arreglos que contendrán cada uno de los datos que nos solicite el programa para resolver un ejercicio determinado, tales como el número de variables, el número de restricciones, los valores de X1, X2 y X3 de la función objetivo y también los arreglos que contienen los resultados del ejercicio:

```
product_names = []
col_values = []
z_equation = []
final_rows = []
solutions = []
x = 'X'
z2_equation = []
removable_vars = []
no_solution = """
```

1.4. Función Main

En la siguiente captura, se define la función **Main**, la cual permite inicializar el programa en modo maximización y luego, el programa comenzará a pedirnos datos de un ejercicio determinado, y entre estos datos está la cantidad de variables y la cantidad de restricciones. Después se define un ciclo **for** que inicializará una variable “i” como contador:

```
def main():
    global decimals
    global const_num, prod_nums
    prob_type = int(1)
    print("""
    METODO SIMPLEX - ROSA MERA
    (MAXIMIZACION)
    """)
    print('\n----------------------------------------')
    global const_names
    const_num = int(input("CANTIDAD DE VARIABLES: >"))
    prod_nums = int(input("CANTIDAD DE RESTRICCIONES: >"))
    const_names = [x + str(i) for i in range(1, const_num + 1)]
    for i in range(1, prod_nums + 1):
        prod_val = i
        product_names.append(prod_val)
    print("----------------------------------------")
```

- Luego del código anterior, se invoca nuevamente a la sentencia Try-except con el fin de obtener el valor para X1, X2 y X3 de la función objetivo, pero considerando solamente valores numéricos, ya que si se ingresa una letra, el programa no la reconocerá como un valor válido y solicitará nuevamente el ingreso de un número:

```
if prob_type == 1:
    for i in const_names:
        try:
            val = float(Fraction(input("INGRESE EL VALOR DE %s DE LA FUNCION OBJETIVO: >" % i)))
        except ValueError:
            print("POR FAVOR, INGRESE UN NUMERO")
            val = float(Fraction(input("INGRESE EL VALOR DE %s DE LA FUNCION OBJETIVO: >" % i)))
        z_equation.append(0 - int(val))
    z_equation.append(0)

    while len(z_equation) <= (const_num + prod_nums):
        z_equation.append(0)
    print("----------------------------------------")
```

- En el siguiente fragmento de código, la función se encarga de solicitar los datos del ejercicio como en el fragmento anterior, pero ésta vez solicita los valores de X1, X2, X3, y el valor del menor o igual. Luego los valores son asociados a cada una de las variables y arreglos que guardan esos datos que ingresamos, y se van enlistando gracias al método “append”, y los valores resultantes que se calculan y luego se almacenan en los arreglos son mostrados debajo de la columna “Bi” y la fila Z. Por último, mediante la variable integer “decimals” se solicita el número de decimales que deseemos que sean mostrados en los valores de la tabla final de resultados, y después se hace un llamado a la función de maximización :

```
for prod in product_names:
    for const in const_names:
        try:
            val = float(Fraction(input("INGRESE EL VALOR DE %s : >" % (const))))
        except ValueError:
            print("EL VALOR DEBE SER UN NUMERO")
            val = float(Fraction(input("INGRESE EL VALOR DE %s: >" % (const))))
        col_values.append(val)
    equate_prod = float(Fraction(input('MENOR IGUAL QUE %s: >' % prod)))
    col_values.append(equate_prod)

final_cols = stdz_rows(col_values)
i = len(const_names) + 1
while len(const_names) < len(final_cols[0]) - 1:
    const_names.append('X' + str(i))
    solutions.append('X' + str(i))
    i += 1
solutions.append(' Z')
const_names.append('Bi')
final_cols.append(z_equation)
final_rows = np.array(final_cols).T.tolist()
print("----------------------------------------")
decimals = int(input('NUMERO DE DECIMALES A MOSTRAR : '))
print('\n----------------------------------------')
maximization(final_cols, final_rows)
```

1.5. Función maximization

En la captura que se muestra a continuación, se puede observar la función “maximization”, la cual se encarga de realizar todos los cálculos necesarios para resolver un determinado ejercicio a través del método simplex en base a los datos ingresados en el programa. Al inicio tenemos 2 arreglos y dos variables para almacenar los datos ingresados y calculados, y tenemos un mensaje que se mostrará como título de tabla para indicar el número de tabla que se está mostrando. En la sentencia Try-except se hace uso de “***np***”, el cual actúa como referencia del módulo *numpy* para que la tabla se vaya ordenando. El contador “i” se sigue incrementando para que el programa sepa hasta donde seguir pidiendo valores para Xn de acuerdo al número de variables que hayamos establecido al inicio. En el método While se establece la condición necesaria para cumplir con la regla de desarrollo del método simple, relacionando las variables y arreglos, actualizando los resultados en las variables correspondientes:

```
def maximization(final_cols, final_rows):
    row_app = []
    last_col = final_cols[-1]
    min_last_row = min(last_col)
    min_manager = 1
    print("TABLA 1")
    try:
        final_pd = pd.DataFrame(np.array(final_cols), columns=const_names, index=solutions)
        print(final_pd)
    except:
        print('  ', const_names)
        i = 0
        for cols in final_cols:
            print(solutions[i], cols)
            i += 1
    count = 2
    pivot_element = 2
    while min_last_row < 0 < pivot_element != 1 and min_manager == 1 and count < 6:
        print("----------------------------------------")
        last_col = final_cols[-1]
        last_row = final_rows[-1]
        min_last_row = min(last_col)
        index_of_min = last_col.index(min_last_row)
        pivot_row = final_rows[index_of_min]
        index_pivot_row = final_rows.index(pivot_row)
        row_div_val = []
        i = 0
        for _ in last_row[:-1]:
```

- También se tomaron en cuenta las validaciones respectivas en cuanto al procedimiento de división entre cero o un valor inválido:

```
    try:
        val = float(last_row[i] / pivot_row[i])
        if val <= 0:
            val = 10000000000
        else:
            val = val
        row_div_val.append(val)
    except ZeroDivisionError:
        val = 10000000000
        row_div_val.append(val)
    i += 1
```

- La siguiente parte del código es utilizada para hallar el valor del elemento pivote, la columna pivote y la fila pivote. Cada valor es mostrado a través del dato almacenado en la variable asignada para cada uno de éstos 3 elementos mencionados. Luego en el ciclo For se realizan los cálculos necesarios de división, multiplicación y resta requeridos para el desarrollo el ejercicio en base a los datos proporcionados en el programa.

```
min_div_val = min(row_div_val)
index_min_div_val = row_div_val.index(min_div_val)
pivot_element = pivot_row[index_min_div_val]
pivot_col = final_cols[index_min_div_val]
index_pivot_col = final_cols.index(pivot_col)
row_app[:] = []
for col in final_cols:
    if col is not pivot_col and col is not final_cols[-1]:
        form = col[index_of_min] / pivot_element
        final_val = np.array(pivot_col) * form
        new_col = (np.round((np.array(col) - final_val), decimals)).tolist()
        final_cols[final_cols.index(col)] = new_col

    elif col is pivot_col:
        new_col = (np.round((np.array(col) / pivot_element), decimals)).tolist()
        final_cols[final_cols.index(col)] = new_col
    else:
        form = abs(col[index_of_min]) / pivot_element
        final_val = np.array(pivot_col) * form
        new_col = (np.round((np.array(col) + final_val), decimals)).tolist()
        final_cols[final_cols.index(col)] = new_col
final_rows[:] = []
re_final_rows = np.array(final_cols).T.tolist()
final_rows = final_rows + re_final_rows

if min(row_div_val) != 10000000000:
    min_manager = 1
else:
    min_manager = 0
print('ELEMENTO PIVOTE: %s' % pivot_element)
print('COLUMNA PIVOTE: ', pivot_row)
print('FILA PIVOTE: ', pivot_col)
print("\n----------------------------------------")
solutions[index_pivot_col] = const_names[index_pivot_row]
```

- Así mismo, se realizan las validaciones respectivas de cada operación, para evitar el ingreso de valores inválidos y por lo tanto el cálculo erróneo de los datos.

```
print("TABLA %d" % count)
try:
    final_pd = pd.DataFrame(np.array(final_cols), columns=const_names, index=solutions)
    print(final_pd)
except:
    print("TABLA %d" % count)
    print('  ', const_names)
    i = 0
    for cols in final_cols:
        print(solutions[i], cols)
        i += 1
count += 1
last_col = final_cols[-1]
last_row = final_rows[-1]
min_last_row = min(last_col)
index_of_min = last_col.index(min_last_row)
pivot_row = final_rows[index_of_min]
row_div_val = []
i = 0
for _ in last_row[:-1]:
    try:
        val = float(last_row[i] / pivot_row[i])
        if val <= 0:
            val = 10000000000
        else:
            val = val
        row_div_val.append(val)
    except ZeroDivisionError:
        val = 10000000000
        row_div_val.append(val)
    i += 1
min_div_val = min(row_div_val)
index_min_div_val = row_div_val.index(min_div_val)
pivot_element = pivot_row[index_min_div_val]
```

1.6. Funciones para visualizar los resultados en las tablas

Finalmente, tenemos dos funciones adicionales para mejorar la visibilidad de los valores de X1, X2, X3, X4, Z y Bi en la tabla, de tal forma que se aprecien los datos de forma ordenada. Cada valor se almacena en una variable o arreglo, y por lo tanto éstas funciones cumplen precisamente con el propósito de asignar cada resultado contenido en la variable o arreglo, en una determinada posición, conformando las filas y columnas, al mismo tiempo que se hace uso de *numpy* para poder cuadrar las columnas y filas de tal forma que concuerde el título de columna y fila de la tabla con los correspondientes valores:

```
def stdz_rows2(column_values):
    final_cols = [column_values[x:x + const_num + 1] for x in range(0, len(column_values), const_num + 1)]
    sum_z = (0 - np.array(final_cols).sum(axis=0)).tolist()
    for _list in sum_z:
        z2_equation.append(_list)

    for cols in final_cols:
        while len(cols) < (const_num + (2 * prod_nums) - 1):
            cols.insert(-1, 0)

    i = const_num
    for sub_col in final_cols:
        sub_col.insert(i, -1)
        z2_equation.insert(-1, 1)
        i += 1

    for sub_col in final_cols:
        sub_col.insert(i, 1)
        i += 1

    while len(z2_equation) < len(final_cols[0]):
        z2_equation.insert(-1, 0)

    return final_cols
```

```
def stdz_rows(column_values):
    final_cols = [column_values[x:x + const_num + 1] for x in range(0, len(column_values), const_num + 1)]
    for cols in final_cols:
        while len(cols) < (const_num + prod_nums):
            cols.insert(-1, 0)

    i = const_num
    for sub_col in final_cols:
        sub_col.insert(i, 1)
        i += 1

    return final_cols
```

2. DEMOSTRACIÓN DEL CÓDIGO EN MODO CONSOLA

- Vamos a resolver el siguiente ejercicio:

Max Z= 3x1 + 4x2

S.R

2x1 + 3x2 <= 1200
2x1 + x2 <= 1000
4x2 <= 800

X1,x2 >= 0

- Descargamos el archivo desde el link que se encuentra en la parte de ANEXOS.
- Abrimos el Símbolo del sistema o **cmd** de Windows, colocamos **cd Desktop** y enter, después colocamos el nombre del archivo a abrir que en este caso es **metodosimplex.py**

```
C:\Users\rouss>cd Desktop

C:\Users\rouss\Desktop>metodosimplex.py
```

- Además, siguiendo nuestro ejercicio colocamos la cantidad de variables y la cantidad de restricciones:

Símbolo del sistema

```
C:\Users\rouss\Desktop>metodosimplex.py

    METODO SIMPLEX - ROSA MERA
    (MAXIMIZACION)


----------------------------------------
CANTIDAD DE VARIABLES: >2
CANTIDAD DE RESTRICCIONES: >3
----------------------------------------
```

- Luego ingresamos los valores de la función objetivo, en este caso solo va a ser para x1 y x2 porque ingresamos anteriormente 2 variables:

```
----------------------------------------
INGRESE EL VALOR DE X1 DE LA FUNCION OBJETIVO: >3
INGRESE EL VALOR DE X2 DE LA FUNCION OBJETIVO: >4
----------------------------------------
```

- Siguiendo con la ejecución del programa nos va a pedir que ingresemos los valores de las restricciones:

```
----------------------------------------
INGRESE EL VALOR DE X1 : >2
INGRESE EL VALOR DE X2 : >3
MENOR IGUAL QUE 1: >1200
INGRESE EL VALOR DE X1 : >2
INGRESE EL VALOR DE X2 : >1
MENOR IGUAL QUE 2: >1000
INGRESE EL VALOR DE X1 : >0
INGRESE EL VALOR DE X2 : >4
MENOR IGUAL QUE 3: >800
----------------------------------------
```

- También nos va a pedir que ingresemos el número de decimales que queremos que se muestren:

```
----------------------------------------
NUMERO DE DECIMALES A MOSTRAR : 2

----------------------------------------
```

- Ya habiendo ingresado lo anterior y por lo consiguiente haber dado enter nos va a mostrar las tablas, el elemento pivote con la columna y la fila de donde se encuentra:

```
----------------------------------------
TABLA 1
      X1    X2   X3   X4   X5       Bi
X3  12.0   6.0  1.0  0.0  0.0  20400.0
X4   9.0  15.0  0.0  1.0  0.0  25200.0
X5   6.0   6.0  0.0  0.0  1.0  12000.0
 Z  -5.0  -4.0  0.0  0.0  0.0      0.0
----------------------------------------
ELEMENTO PIVOTE: 12.0
COLUMNA PIVOTE:  [12.0, 9.0, 6.0, -5.0]
FILA PIVOTE:  [12.0, 6.0, 1, 0, 0, 20400.0]

----------------------------------------
TABLA 2
     X1    X2    X3   X4   X5      Bi
X1  1.0   0.5  0.08  0.0  0.0  1700.0
X4  0.0  10.5 -0.75  1.0  0.0  9900.0
X5  0.0   3.0 -0.50  0.0  1.0  1800.0
 Z  0.0  -1.5  0.42  0.0  0.0  8500.0
----------------------------------------
ELEMENTO PIVOTE: 3.0
COLUMNA PIVOTE:  [0.5, 10.5, 3.0, -1.5]
FILA PIVOTE:  [0.0, 3.0, -0.5, 0.0, 1.0, 1800.0]

----------------------------------------
```

- En nuestra tabla final nos mostrará los resultados que en este caso para el ejercicio que se está resolviendo para x1 = 1400, x2 = 600 y el valor de Z = 9400

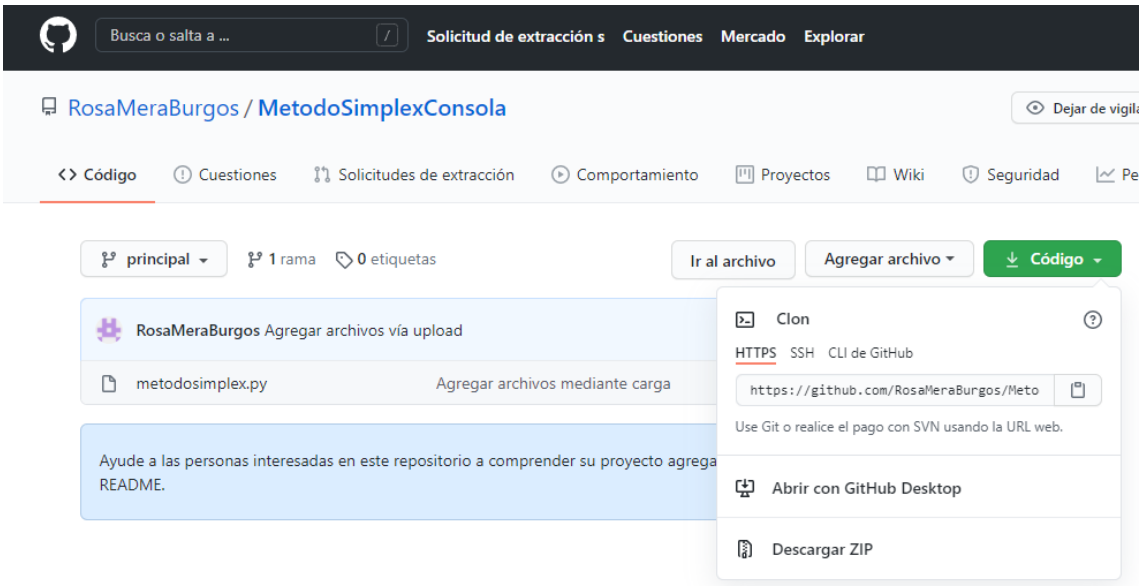
3. ANEXOS:

1. Descargar el archivo

- Ingresar al siguiente del repositorio del programa para descargar el archivo y ejecutar en modo consola:

https://github.com/RosaMeraBurgos/MetodoSimplexConsola

- En el botón verde que dice ‘Código’ hacer clic y colocar en Descargar ZIP, una vez descomprimido el archivo, copiar el programa al escritorio:



2. Ver video:

- Se puede visualizar el video o descargar a través del siguiente link a Google Drive: https://drive.google.com/file/d/1WtVF4pJ0K7Gn73Q_v9Khe-H_ztQjHugL/view?usp=sharing