BAQUE PILOSO JORGE LUIS
Una fábrica de bancos y mesas desea maximizar sus beneficios donde la contribución marginal de bancos es de $ 100 por unidad y de mesas $ 120 por unidad. Cuenta con 60 horas semanales para el ensamble y utiliza 3 horas semanales para bancos y 6 horas semanales para mesas. Para el proceso de terminado cuenta con 32 horas semanales y utiliza 4 horas semanales para bancos y 2 horas para mesas.
Al fabricante no le conviene producir menos de cinco bancos semanales y menos de 2 mesas semanales ya que no cubre sus costos fijos.
Se pide:
a. Solución por el método gráfico.
b. Solución método simplex
c. Dual del Primal

In [*]:

```
from typing import List
import matplotlib.pyplot as plt
import numpy as np
from tkinter import *
from tkinter import ttk
import tkinter.messagebox
# ------PARTE VISUAL------ #
# Creación de la clase ventana, permite manejar toda la parte de IU de la aplicación


class ventana:
    def __init__(self):
        # Se crea un objeto de tipo TK
        self.v0 = Tk()
        # Establece tamaño y posición de la ventana
        self.v0.geometry("492x500+400+100")
        # Agrega titulo a la ventana
        self.v0.title("Software de Método Gráfico -Proyecto de I.O")
        # Cambia el color del fondo
        self.v0.config(bg="#8BB2E4")
        # Establece la dimensión de la ventana fija
        self.v0.resizable(False, False)
        # Crear un label que contiene el titulo
        self.lTitulo = Label(self.v0, text="Método Gráfico", fg="red", font=(
            "Helvetica", 16), pady="15", bg="#8BB2E4")
        # Establece la posición del titulo dentro de la venta
        self.lTitulo.grid(row=0, column=0)
        # Crea un nuevo frame dentro de la ventana
        self.frame0 = Frame(self.v0)
        # Establece propiedades del frame0
        self.frame0.config(padx="10", pady="10", bg="#8BB2E4",
                           borderwidth=1, relief="ridge")
        # Ajusta su posición dentro de la ventana
        self.frame0.grid(row=1, padx=10)
        # Crea un nuevo frame dentro de la ventana
        self.frame1 = LabelFrame(self.frame0, text="Función Objetivo: Ejemplo 2x + 8y",
                                 padx="10", pady="10", labelanchor="n", bg="#8BB2E4")
        # Ajusta su posición dentro de la ventana
        self.frame1.grid(row=0, column=0)
        # Crea un nuevo frame dentro de la ventana
        self.frame2 = Frame(self.frame0, bg="#8BB2E4")
        # Ajusta su posición dentro de la ventana
        self.frame2.grid(row=0, column=1, rowspan=2, padx=10, pady=20)
        # widgets del frame 0
        self.labelT = Label(self.frame1, text="Tipo:",
                            font=("Helvetica", 12), bg="#8BB2E4")
        self.labelT.grid(row=0, column=0, padx=10)
        self.ValorMax_Min = StringVar()
        self.CBMax_Min = ttk.Combobox(
            self.frame1, textvariable=self.ValorMax_Min, width="10", font=("Helvetica", 10))
        self.CBMax_Min['values'] = ('Maximizar', 'Minimizar')
        self.CBMax_Min.current(0)
        self.CBMax_Min.grid(row=0, column=1, padx=10)
        self.valorFO = StringVar()
        self.valorLFO = StringVar()
        self.labelZ = Label(self.frame1, text="Z= ", font=(
            "Helvetica", 12), width="4", bg="#8BB2E4")
        self.labelZ.grid(row=0, column=2)
        self.entradaFO = Entry(
```

```
            self.frame1, textvar=self.valorFO, width="10", font=("Helvetica", 10))
        self.entradaFO.grid(row=0, column=3, padx=10)
        self.BtnAgregarFO = Button(self.frame2, text="Agregar FO", command=lambda: self.Agr
            self.valorFO, self.ValorMax_Min, self.valorLFO))
        self.BtnAgregarFO.grid(row=0, column=0)
        self.BtnNuevo = Button(self.frame2, text="Nuevo",
                               command=lambda: self.nuevo(self.valorLFO))
        self.BtnNuevo.grid(row=1, column=0, padx=20, pady=20)
        self.labelFO = Label(self.frame0, textvar=self.valorLFO, font=(
            "Helvetica", 16), relief="ridge", width="22", bg="#FFF")
        self.labelFO.grid(row=1, column=0)
        # Crear mas frames
        self.frame3 = Frame(self.v0)
        self.frame3.config(padx="10", pady="10", bg="#8BB2E4",
                           borderwidth=1, relief="ridge")
        self.frame3.grid(row=2, padx=10, pady=10)
        self.frame4 = LabelFrame(self.frame3, text="Restricciones: Ejemplo 2x + 8y <= Ò >=
                                 padx="10", pady="10", labelanchor="n", bg="#8BB2E4")
        self.frame4.grid(row=0, column=0, pady=10)
        self.frame5 = LabelFrame(self.frame3, bg="#8BB2E4")
        self.frame5.grid(row=0, column=1, padx=10, pady=10, rowspan=2)
        # Widgets Restricciones
        self.lista = Listbox(self.frame3, width=28,
                             height=5, font=("Helvetica", 14))
        self.lista.grid(row=1, column=0)
        self.valorRes = StringVar()
        self.entradaRes = Entry(
            self.frame4, textvar=self.valorRes, font=("Helvetica", 10))
        self.entradaRes.grid(row=0, column=0, padx=10)
        self.BtnAgregarRes = Button(
            self.frame4, text="Agregar", command=lambda: self.insertarListbox(self.lista, s
        self.BtnAgregarRes.grid(row=0, column=1, pady=10)
        self.BtnMostrarGrafica = Button(
            self.frame5, text="Mostrar Grafica", command=lambda: graficar())
        self.BtnMostrarGrafica.grid(row=0, column=1, padx=10, pady=20)
        self.BtnReiniciar = Button(
            self.frame5, text="\nMostrar Resultados", command=lambda: hallarResultados())
        self.BtnReiniciar.grid(row=1, column=1, padx=10, pady=20)
        self.BtnReiniciar = Button(
            self.frame5, text="\nReiniciar", command=lambda: self.limpiar_listbox(self.list
        self.BtnReiniciar.grid(row=2, column=1, padx=10, pady=20)
    # Función que se ejecuta cuando se toca el boton "Agregar FO"

    def AgregarFO(self, fo, max_min, label):
        if fo.get() == "":
            tkinter.messagebox.showinfo(
                "ERROR", "No se ha ingresado función objetivo")
        else:
            foCopia = fo.get()
            try:
                foCopia = foCopia.replace("x", "/")
                foCopia = foCopia.replace("y", "/")
                foCopia = foCopia.split("/")
                global Objfo
                Objfo.llenarAtributos(
                    float(foCopia[0]), float(foCopia[1]), max_min.get())
                label.set(max_min.get()+" Z=  "+fo.get())
            except:
                tkinter.messagebox.showinfo(
                    "ERROR", "Error al ingresar los datos")
    # Función que se ejecuta cuando se toca el boton "Nuevo"
```

```
    def nuevo(self, label):
        global Objfo
        label.set("")
        Objfo = fObjetivo()
        listaEcuaciones = []
        listaRestricciones = []
        listaPuntos = []
        print(listaEcuaciones)
    # Función que se ejecuta cuando se toca el boton "Agregar Restricciones"

    def insertarListbox(self, listbox, rest):
        if rest.get() == "":
            tkinter.messagebox.showinfo(
                "ERROR", "No ha ingresado restricciones")
        else:
            try:
                ecuacion = rest.get()
                if ecuacion.find("<=") > 1:
                    ecuacion = ecuacion.replace("<=", "")
                    tipo = 1
                elif ecuacion.find(">=") > 1:
                    ecuacion = ecuacion.replace(">=", "")
                    tipo = 0
                else:
                    ecuacion = ecuacion.replace("=", "")
                    tipo = 2
                ecuacion = ecuacion.replace("x", "/")
                ecuacion = ecuacion.replace("y", "/")
                ecuacion = ecuacion.replace("<=", "/")
                ecuacion = ecuacion.replace(">=", "/")
                ecuacion = ecuacion.replace("=", "/")
                ecuacion = ecuacion.split("/")
                a = float(ecuacion[0])
                b = float(ecuacion[1])
                c = float(ecuacion[2])
                listbox.insert(END, rest.get())
                listaEcuaciones.append((a, b, c, tipo))
                rest.set("")
            except:
                tkinter.messagebox.showinfo(
                    "ERROR", "Error al ingresar los datos")
    # Función que se ejecuta cuando se toca el boton "Reiniciar"

    def limpiar_listbox(self, listbox):
        while listbox.size() > 0:
            listbox.delete(0)
            global listaRestricciones
            listaRestricciones = []
            global listaEcuaciones
            listaEcuaciones = []
            print(listaEcuaciones)
            global listaPuntos
            listaPuntos = []
            Objfo.pFactible = []
    # Función que se ejecuta cuando empieza la aplicación

    def run(self):
        self.v0.mainloop()
```

```
# ------PARTE LÓGICA------ #
# Permite Almacenar la restricciones que recien se agregan
listaEcuaciones = []
# Almacena los puntos
listaPuntos = [(0, 0)]
# Esta lista contiene letras que sirven para marcar los puntos
letras = ["0", "A", "B", "C", "D", "E", "F", "G",
          "H", "I", "J", "K", "L", "M", "N", "O", "P", "Q"]
# Almacena la lista de objetos de tipo restriccion
listaRestricciones = []
# Almacena los valores de z obtenido de evaluar cada punto
listaResultados = []


def hallarVertices():
    i = 0
    j = i+1
    for restriccion in listaRestricciones:
        listaPuntos.append((restriccion.valorX, 0.))
        listaPuntos.append((0., restriccion.valorY))
    while i < len(listaRestricciones)-1:
        while j < len(listaRestricciones):
            try:
                M1 = np.array([[listaRestricciones[i].valora, listaRestricciones[i].valorb]
                               [listaRestricciones[j].valora, listaRestricciones[j].valorb]
                M2 = np.array([listaRestricciones[i].valorc,
                               listaRestricciones[j].valorc])
                punto = np.linalg.solve(M1, M2)
                listaPuntos.append((punto[0], punto[1]))
            except:
                pass
            j += 1
        i += 1
        j = i+1


def dibujarPuntos(plt):
    i = 0
    for vertice in listaPuntos:
        plt.annotate(letras[i], xy=(vertice[0], vertice[1]), fontsize=12)
        plt.scatter(vertice[0], vertice[1], color="b")
        i += 1


def puntosNegativos():
    i = 0
    while i < len(listaPuntos):
        if listaPuntos[i][0] < 0 or listaPuntos[i][1] < 0:
            listaPuntos.remove(listaPuntos[i])
        i += 1


def puntosRepetidos(Puntos):
    lista_nueva = []
    for i in Puntos:
        if i not in lista_nueva:
            lista_nueva.append(i)
    Puntos[0:len(listaPuntos)] = lista_nueva


def hallarResultados():
```

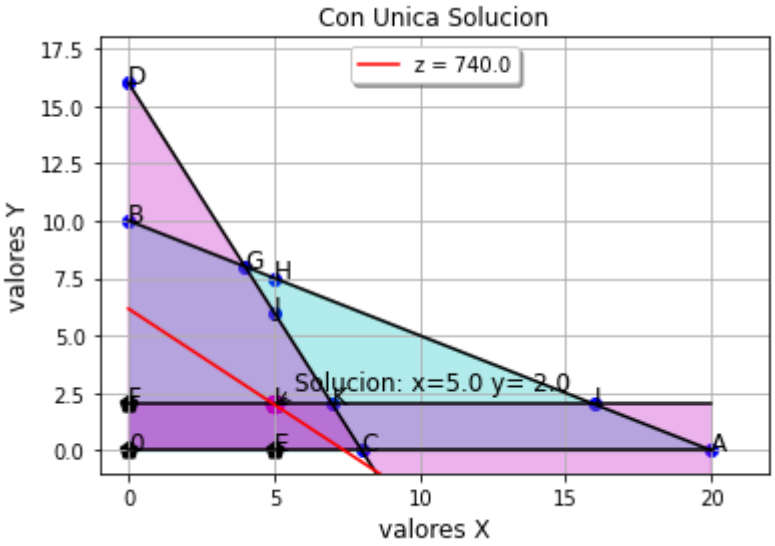
```
    i = 0
    for p in listaPuntos:
        listaResultados.append(
            (letras[i], p[0], p[1], Objfo.a*p[0]+Objfo.b*p[1]))
        i += 1


def graficar():
    mayorX = 0
    mayorY = 0
    colores = ["b", "g", "c", "m", "y"]
    for ecuacion in listaEcuaciones:
        restic = Restriccion(
            ecuacion[0], ecuacion[1], ecuacion[2], ecuacion[3])
        listaRestricciones.append(restic)
    if len(listaRestricciones) > 0:
        for res in listaRestricciones:
            res.hallarPuntos()
            abs(res.valorX)
            if abs(res.valorX) > mayorX:
                mayorX = abs(res.valorX)
            if res.valorY > mayorY:
                mayorY = res.valorY
        hallarVertices()
        puntosNegativos()
        puntosRepetidos(listaPuntos)
        dibujarPuntos(plt)
        hallarResultados()
        for res in listaRestricciones:
            if mayorX <= 0:
                mayorX = mayorY = 10
            res.valorX = mayorX
            res.despejarY()
            res.dibujar(plt)
            seleccion = np.random.randint(len(colores))
            if res.tipo == 1:
                plt.fill_between(res.listaX, res.listaY,
                                 color=colores[seleccion], alpha=0.3)
        plt.grid(True)
        plt.ylim(-1, mayorY+2)
        plt.xlim(-1, mayorX+2)
        plt.xlabel('valores X', fontsize=12)
        plt.ylabel('valores Y', fontsize=12)
        global Objfo
        Objfo.puntosFactible(listaResultados, listaRestricciones, plt)
        Objfo.solucion(listaResultados, mayorX)
        plt.legend(loc='upper center', shadow=True)
        plt.show()
    else:
        tkinter.messagebox.showinfo("ERROR", "No ha ingresado restricciones")


class fObjetivo:
    def __init__(self):
        self.a = 0
        self.b = 0
        self.tipo = ""
        self.z = 0
        self.pFactible = []

    def llenarAtributos(self, a, b, tipo):
```

```
        self.a = a
        self.b = b
        self.tipo = tipo

    def puntosFactible(self, resultados, restricciones, plt):
        i = 0
        j = 0
        cuenta = 0
        for p in resultados:
            for restriccion in restricciones:
                if restriccion.tipo == 1:
                    if restriccion.valora*p[1]+restriccion.valorb*p[2] <= restriccion.valor
                        cuenta += 1
                elif restriccion.tipo == 0:
                    if restriccion.valora*p[1]+restriccion.valorb*p[2] >= restriccion.valor
                        cuenta += 1
                else:
                    if restriccion.valora*p[1]+restriccion.valorb*p[2] == restriccion.valor
                        cuenta += 1
            if cuenta == len(restricciones):
                self.pFactible.append((p[1], p[2], p[3]))
                plt.scatter(p[1], p[2], color="k", marker=(5, 0), s=80)
            cuenta = 0

    def solucion(self, resultados, mayorX):
        p = []
        cuenta = 0
        menor = 100
        mayor = 0
        print(len(self.pFactible))
        if len(self.pFactible) < 1:
            plt.title("Solucion No Factible")
        elif len(self.pFactible) == 1:
            if self.tipo == "Maximizar":
                plt.title("Con Solucion No Acotada")
            else:
                p = (self.pFactible[0][0], self.pFactible[0][1])
                print(p)
                plt.title("Con Unica Solucion")
                listaX = np.arange(0, float(mayorX)+1)
                listaY = (mayor-self.a*listaX)/self.b
                plt.plot(listaY, color="r", label="z = "+str(mayor))
                plt.annotate(r'Solucion: x='+str(p[0])+' y= '+str(p[1])+'', xy=(p[0], p[1])
                    p[0]+5, p[1]+5), textcoords='offset points', fontsize=12, arrowprops=di
                plt.scatter(p[0], p[1], color="m", s=80)
        else:
            if self.tipo == "Maximizar":
                for f in self.pFactible:
                    if f[2] >= mayor:
                        mayor = f[2]
                        p = (f[0], f[1])
                for f in self.pFactible:
                    if f[2] == mayor:
                        cuenta += 1
                if cuenta >= 2:
                    plt.title("Con Multiples Soluciones")
                    listaX = np.arange(0, float(mayorX)+1)
                    listaY = (mayor-self.a*listaX)/self.b
                    plt.plot(listaY, color="r", label="z = "+str(mayor))
                else:
                    plt.title("Con Unica Solucion")
```

```
                    listaX = np.arange(0, float(mayorX)+1)
                    listaY = (mayor-self.a*listaX)/self.b
                    plt.plot(listaY, color="r", label="z = "+str(mayor))
                    plt.annotate(r'Solucion: x='+str(p[0])+' y= '+str(p[1])+'', xy=(p[0], p
                        p[0]+5, p[1]+5), textcoords='offset points', fontsize=12, arrowprop
                    plt.scatter(p[0], p[1], color="m", s=80)
            else:
                print("Minimizar")
                for f in self.pFactible:
                    if f[2] <= menor:
                        menor = f[2]
                        p = (f[0], f[1])
                for f in self.pFactible:
                    if f[2] == menor:
                        cuenta += 1
                if cuenta >= 2:
                    listaX = np.arange(0, float(mayorX)+1)
                    listaY = (menor-self.a*listaX)/self.b
                    plt.plot(listaY, color="r", label="z = "+str(menor))
                    plt.title("Con Solucion Multiple")
                else:
                    listaX = np.arange(0, float(mayorX)+1)
                    listaY = (menor-self.a*listaX)/self.b
                    plt.plot(listaY, color="r", label="z = "+str(menor))
                    plt.annotate(r'Solucion: x='+str(p[0])+' y= '+str(p[1])+'', xy=(p[0], p
                                xytext=(p[0]+5, p[1]+5), textcoords='offset points', fonts
                                arrowprops=dict(arrowstyle="->", connectionstyle="arc3,rad
                    plt.scatter(p[0], p[1], color="m", s=80)
                    plt.title("Con Unica Solucion")


class Restriccion:
    def __init__(self, a, b, c, tipo):
        self.valorX = 0
        self.valorY = 0
        self.listaX = []
        self.listaY = []
        self.valora = a
        self.valorb = b
        self.valorc = c
        self.tipo = tipo

    def hallarPuntos(self):
        if self.valorb == 0:
            self.valorX = self.valorc/self.valora
        elif self.valora == 0:
            self.valorY = self.valorc/self.valorb
        else:
            self.valorX = (self.valorc)/self.valora
            self.valorY = (self.valorc)/self.valorb

    def despejarY(self):
        self.listaX = np.arange(0, float(self.valorX)+1)
        if self.valorb == 0:
            self.listaY = (0*self.listaX)
        elif self.valora == 0:
            self.listaY = (self.valorc+0*self.listaX)/self.valorb
        else:
            self.listaY = (self.valorc-self.valora*self.listaX)/self.valorb

    def dibujar(self, ventana):
```

```
        ventana.plot(self.listaY, "k")


Objfo = fObjetivo()
principal = ventana()
principal.run()
```

4



In [ ]:

In [ ]: