UNIVERSIDAD PANAMERICANA

Materia: Programación Orientada a Objetos (COM102)

Profesor: Gerardo Bárcena Ruiz

Fecha de entrega: 2022-Abril-29

Ciclo: 1222

Nombre del proyecto: Graficadora de curvas de nivel

Miembros del Equipo		
ID	**Nombre**	**Carrera**
0244643	Sara Rocio Miranda Mateos	Ing. Inteligencia de datos y ciberseguridad
0241833	Arantza Méndez Rodríguez	Ing. Mecatrónica
0241529	María Fernanda Rosas Vernis	Ing. Animación y videojuegos

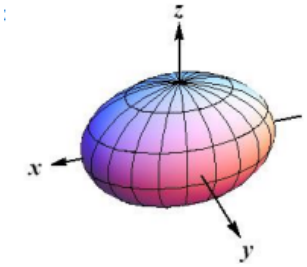
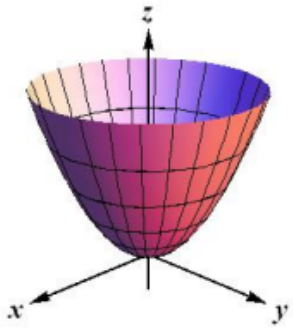
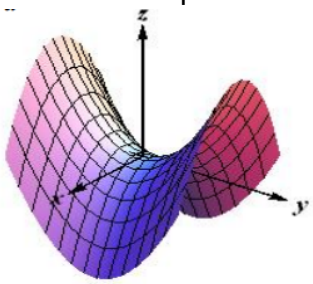
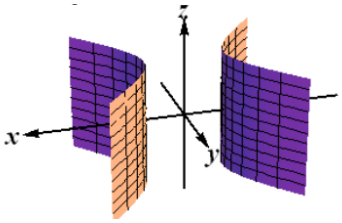
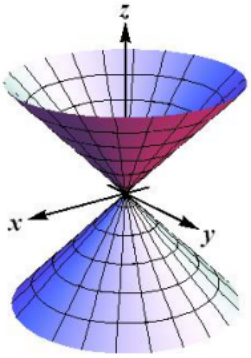
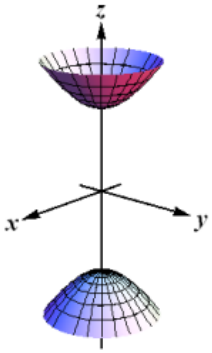
Rúbricas		
ID	**1-identify**	
	IP	**ASA**

Descripción:

Nuestro proyecto es una calculadora destinada para resolver operaciones de cálculo vectorial (curvas de nivel). Contamos con un menú que ofrece al usuario 6 opciones de superficies cuadráticas. Dentro de cada opción se le muestra la función a resolver y los parámetros que puede modificar. Al llenar toda la información requerida el programa regresará una gráfica de la superficie en 3D y todos los datos serán guardados dentro de un apartado específico para el historial.

La motivación para desarrollar esta calculadora surge del hecho de que para materias de cálculo avanzado, las superficies cuadráticas son parte del temario. Es por esto, que al realizar una plataforma que grafique desde elipsoides hasta cilindros, el estudio de estos puede facilitarse.

Asimismo, consideramos importante presentar las 6 superficies cuadráticas incluidas en este proyecto:

Elipsoide 	Paraboloide elíptico 	Paraboloide hiperbólico 
Cilindro hiperbólico 	Cono Elíptico 	Hiperboloide de dos hojas 

Objetivos:

- Ayudar al usuario a visualizar gráficas de superficies cuadráticas con un programa 3D.
- Ampliar nuestro conocimiento sobre las bibliotecas de python y aplicar sus diferentes funcionalidades apropiadamente.
- Facilitar al usuario el estudio de temas complejos de cálculo, en este caso, las superficies cuadráticas.

Desarrollo:

Aplicamos bibliotecas usadas anteriormente en el semestre para lograr que nuestra calculadora cumpliera los objetivos del proyecto. Tkinter, is, PIL, matplotlib, numpy, math y os fueron esenciales para la realización del mismo.

Contamos con dos clases madres, la primera "Graficador" es a donde llegan todos los parámetros de nuestras funciones para lograr que la gráfica se muestre en 3D. Lo siguiente es nuestro "PlotWin", el cuerpo principal del proyecto, tenemos funciones para asignar imágenes a las ventanas del tkinter, tenemos funciones dedicadas a sólo los botones. Como nuestro proyecto es una calculadora con varias pestañas (el menú, las funciones y el historial) queríamos que al usuario se le facilitará moverse entre ellas con botones y que no tuviera que cerrar el programa cada que quisiera realizar otra acción.

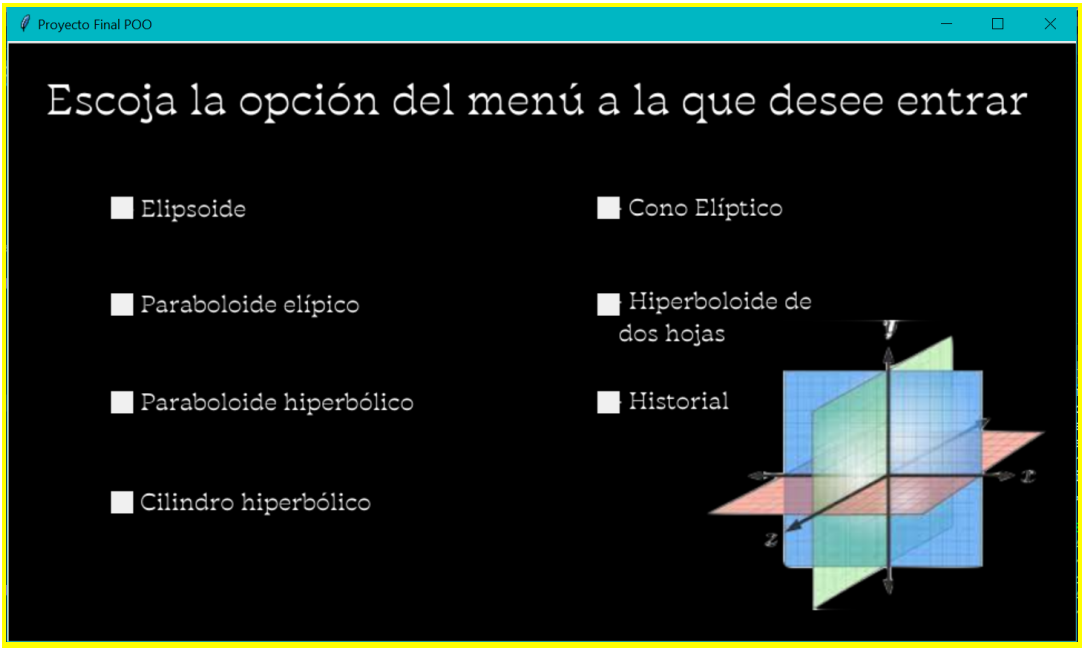


Fig 1. Menú que muestras las opciones de superficies disponibles y el historial

Después, tenemos una función que se designa todos los parámetros a las funciones dependiendo de qué variables necesiten, para esto se hizo uso de banderas.

```
if (band == 1): # 1 = XYZ CON ABC
    self.lblZ = tkinter.Label(self.window, text="Z = ", bg="black", fg ="white")
    self.lblZ.place(x=350, y=352)
    self.txtZ = tkinter.Entry(self.window, bg="black", fg ="white", width=5)
    self.txtZ.place(x=400, y=350)
    prSetTxt(self.txtZ, '1')

    self.lblC = tkinter.Label(self.window, text="C = ", bg="black", fg ="white")
    self.lblC.place(x=350, y=392)
    self.txtC = tkinter.Entry(self.window, bg="black", fg ="white", width=5)
    self.txtC.place(x=400, y=390)
    prSetTxt(self.txtC, '1')

if (band2 == 1): # Z = X + Y CON ABC
    self.graph = tkinter.Button(self.window, borderwidth = 0, text = "GRAPH",
                                command=self.btngraph1)
    self.graph.place(x=650, y=352, height = 50, width=75)
elif(band2 == 2): # 1 = XYZ CON ABC
    self.graph = tkinter.Button(self.window, borderwidth = 0, text = "GRAPH",
                                command=self.btngraph2)
    self.graph.place(x=650, y=352, height = 50, width=75)
```

Luego tenemos funciones que guardan todas las operaciones que la calculadora es capaz de realizar, algunas de las superficies cuadráticas requieren de ellas, otras logramos graficarlas directamente con una función diferente.

```
def gets(self, ec):
    self.a = (float(self.txtA.get()) * float(self.txtA.get()))
    self.b = (float(self.txtB.get()) * float(self.txtB.get()))
    if((self.a  == 0) or (self.b == 0)):
        messagebox.showinfo(
            title="ERROR",
            message="El numero no debe ser CERO")

    if(ec==1):
        self.funcion = ('(',self.txtY.get(), '*Y*Y/', str(self.b), ')-(',
                    self.txtX.get(), '*X*X/', str(self.a),')' )
    elif(ec==2):
        self.c = (float(self.txtC.get()) * float(self.txtC.get()))
        self.funcion = ('(',self.txtX.get(), '*X*X/', str(self.a), ')+(',
                    self.txtY.get(), '*Y*Y/', str(self.b),')', '-(', self.txtZ.get(), '*Z*Z/',str(self.c),')' )
    elif(ec==3):
        self.funcion = ('(',self.txtX.get(), '*X*X/', str(self.a), ')+(',
                    self.txtY.get(), '*Y*Y/', str(self.b),')')
    elif(ec==4):
        self.funcion = ('(',self.txtX.get(), '*X*X/', str(self.a), ')+(',
                    self.txtY.get(), '*Y*Y/', str(self.b),')')
    elif(ec==5):
        self.c = (float(self.txtC.get()) * float(self.txtC.get()))
        self.funcion = ('(',self.txtX.get(), '*X*X/', str(self.a), ')+(',
                    self.txtY.get(), '*Y*Y/', str(self.b),')*Z*Z/',str(self.c),')')

    self.lblf = tkinter.Label(self.window, text=self.funcion, bg="black", fg ="white")
    self.lblf.place(x=400, y=430)

def btngraph1(self): #grafica directamente la funcion

    ecuacion = self.ec
    self.gets(ecuacion)
    self.txtfunc = tkinter.Entry(self.window, borderwidth = 0, bg="black", fg ="black", width=5)
    self.txtfunc.place(x=980, y=430)
    prSetTxt(self.txtfunc, self.funcion)

    self.showPlot(self.txtfunc.get(), self.txtRi.get(),
        self.txtRf.get(), self.txtstep.get(),0, self.nombre, self.colmap)
```

Cada una de las superficies cuadráticas tiene su propia función, la más compleja de ellas es la del elipsoide, ya que para poder graficarlo tuvimos que usar una fórmula diferente a todas las demás, consideramos que fue uno de nuestros obstáculos ya que necesitábamos adaptar esta función a todos los parámetros que ya teníamos contemplados.

```
def elipsoide(self): #FALTAN PARAMETROS A,B,C 1,0
    self.portadadir = elipsoide
    self.fondos(self.portadadir)
    self.ec = 5
    #poner parametros
    self.parametros(1,0)

    self.graph = tkinter.Button(self.window, borderwidth = 0, text = "GRAPH",
                                command=self.btn_elip)
    self.graph.place(x=650, y=352, height = 50, width=75)


    self.salir()
#elipsoide
```

```
def btn_elip(self):

    ecuacion = self.ec
    self.gets(ecuacion)

    coefs = (float(self.txtA.get()), float(self.txtB.get()), float(self.txtC.get()))
    rx, ry, rz = 1/np.sqrt(coefs) #LOS X SE MULTIPLICAN CON LOS PARAMETROS

    # angulos para los elipses
    u = np.linspace(0, 2 * np.pi, 100)
    v = np.linspace(0, np.pi, 100)

    #ecuacion para elipsoides
    x = rx * np.outer(np.cos(u), np.sin(v))
    y = ry * np.outer(np.sin(u), np.sin(v))
    z = rz * np.outer(np.ones_like(u), np.cos(v))

    lFig = Figure(figsize=(5, 4), dpi=120)
    lAxis = Axes3D(lFig)
    lAxis.plot_surface(x, y, z,
                   rstride=1, cstride=1,
                   cmap= self.colmap)
    # para el radio
    max_radius = max(rx, ry, rz)
    for axis in 'xyz':
        getattr(lAxis, 'set_{}lim'.format(axis))((-max_radius, max_radius))


    lWin = tkinter.Tk()
    lWin.title('Elipsoide')
    canvas = FigureCanvasTkAgg(lFig, master=lWin)
    canvas.draw()
    canvas.get_tk_widget().pack(side=tkinter.TOP,
                                fill=tkinter.BOTH, expand=1)
    txt = ('Elipsoide \n'+str(self.funcion)+'\n')
    self._log(txt)
```

Cada plantilla de las superficies cuadráticas le muestra al usuario la fórmula que se debe usar para la figura y una imagen de cómo debería quedar su gráfica. Los parámetros ya están predeterminados pero son editables, de ahí, se muestra un texto que es la fórmula que se usa pero con los parámetros que se prefirieron.

Fig 2. Ejemplo que muestra la interfaz al seleccionar una de las superficies.

A continuación, cuando el usuario oprima el botón de graficar, una ventana aparecerá con la gráfica esperada y el color que se le designó.

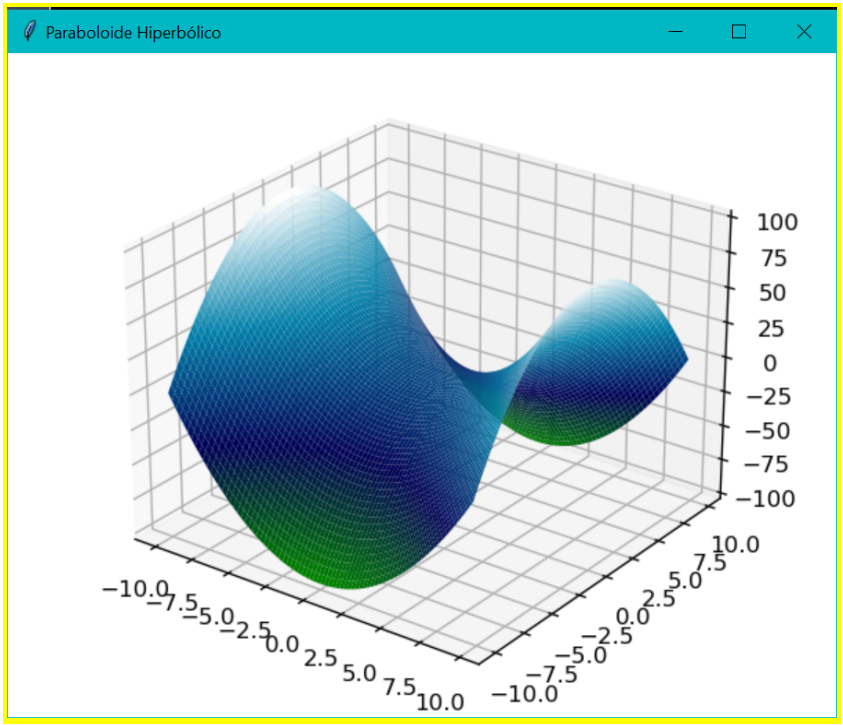


Fig.3 Es así como se grafican cada una de las opciones de la calculadora.

Por último, en el apartado del historial, el código fue diseñado para que guarde el nombre de la función y sus parámetros. Toda la información se guarda en un txt y se imprime en una caja de texto.

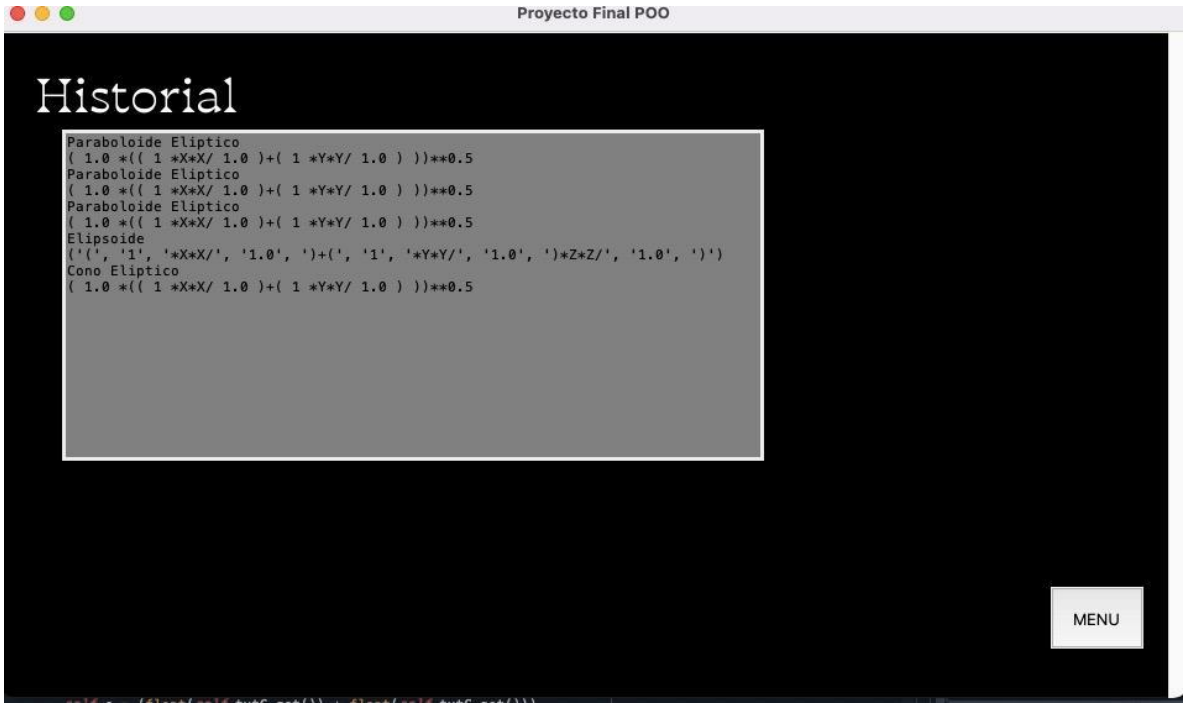
```
    def txt(self):
        self.sbLog = tkinter.Scrollbar(self.window)
        self.sbLog.pack(side=tkinter.RIGHT, fill=tkinter.Y)
        self.txtLog = tkinter.Text(self.window, bg="gray", width=80, height=20,
                                   wrap=tkinter.NONE, yscrollcommand=self.sbLog.set)
        self.txtLog.place(x=50, y=80)
        self.sbLog.config(command=self.txtLog.yview)

        lFile = open(rutatxt, "rt")
        if os.path.isfile(rutatxt):
            lFile = open(rutatxt, "rt")
            for lLine in lFile:
                self.txtLog.insert(tkinter.END, lLine)
            lFile.close()
```

```
def btngraph1(self): #grafica directamente la funcion

    ecuacion = self.ec
    self.gets(ecuacion)
    self.txtfunc = tkinter.Entry(self.window, borderwidth = 0, bg="black", fg ="black", width=5)
    self.txtfunc.place(x=980, y=430)
    prSetTxt(self.txtfunc, self.funcion)
    self.showPlot(self.txtfunc.get(), self.txtRi.get(),
        self.txtRf.get(), self.txtstep.get(),0, self.nombre, self.colmap)

    txt = (self.nombre+'\n'+self.txtfunc.get()+'\n')
    self._log(txt)
#btngraph1
```



Conclusiones:

Esta graficadora de curvas de nivel es una alternativa para los estudiantes que se encuentran en semestres donde las superficies cónicas son parte de los temarios, además de que en temas de tal complejidad, tener herramientas visuales siempre puede ser una ventaja al momento de estudiar o ver por primera vez el tema. En tal sentido, al poder hacerlo a través de un programa que permita al usuario experimentar con los parámetros y observar cómo estos son afectados, dará como resultado un estudio interactivo y eficaz de la materia.

Para finalizar, consideramos que poder tener la oportunidad de resolver "nuestro propio reto" permitió que la curiosidad del equipo explorará los diferentes usos de objetos y herramientas de Python. Gracias a que hubo distintos escenarios de programación durante las clases, nos fue posible tener antecedentes tanto teóricos como prácticos que nos facilitaron el desarrollo de este proyecto.