Graficadora de Funciones Complejas en Python

Baruc Samuel Cabrera García

6 de enero de 2025

Seminario de Titulación

Profesores responsables: Dr. Manuel Cruz Lópes, Dr. Otto Romero

ÃƆndice

1. Introducción

El conjunto de los números complejos consiste de todos los números que se pueden expresar en la forma

$$z = x + iy, \quad x, y \in \mathbb{R}, \quad i := \sqrt{-1}$$

El análisis de los números complejos es un área extensa de investigación. Al momento de querer visualizar tales funciones, nos encontramos con el problema de que es difícil el visualizar las gráficas, ya que son objetos geométricos que viven en el espacio euclidiano de dimensión 4. De esta problemática es que surge la intención de esta investigación. Se desarrolló un programa en Python, el cual en base a la declaración de funciones base, constructoras y graficadoras, es capaz de graficar funciones complejas con base a la parametrización de curvas de interés, así como gráficas 3D de ciertas propiedades de interés para ciertas funciones.

La importancia de esta investigación recae en las posibles aplicaciones para fines didácticos, como de investigación. El programa que se realizó a lo largo del semestre, permitirá a futuros estudiantes el visualizar los números complejos, así como algunos los resultados de la teoría de funciones de variable compleja.

2. Discusión

El enfoque principal de esta investigación, es el desarrollar un código de Python el cual pueda ser usado por diferentes usuarios que requieran del uso de gráficas, y así graficar funciones complejas de interés. Por esto mismo, se buscó que el código tuviera la mayor cantidad de herramientas posibles para que el usuario pueda realizar sus propias gráficas de la forma más eficiente.

A continuación, explicaremos el funcionamiento y propósito de algunas secciones del código "Graficadora_de_Funciones_C

2.1. Funciones base

Recordemos que un número complejo $z \in \mathbb{C}$ esta conformado por sus partes reales e imaginarias, esto es

$$z = x + iy, \quad x, y \in \mathbb{R}.$$

Esto se puede resolver en Python con las funciones de la librería "cmath", que en este caso son tales que

$$x = z.real, \quad y = z.imag \quad .$$

Luego, recordemos que el conjugado $\overline{z}$ de un número complejo z es tal que

$$\begin{aligned} z = x + iy, \quad \overline{z} &= x - iy, \\ &= z.conjugate. \end{aligned}$$

La norma compleja se puede expresar de la forma

$$|z| = |x + iy| = \sqrt{x^2 + y^2} = \sqrt{(x + iy)(x - iy)} = \sqrt{z\overline{z}}.$$

Con base en lo anterior, en Python programamos las funciones base, las cuales nos servirán a definir otras funciones complejas basadas en la librería "numpy". En la figura 1 se ven las definiciones declaradas con base en los aspectos ya mencionados.

Figura 1:

```
# Funcion que dado un numero z = x + jy, regresa (x,y)
def split(z):#Regresa la parte real e imaginaria de un numero z
  return z.real, z.imag

# Funcion que regresa la parte real de un numero z
def Re(z): # Funcion
  return z.real

# Funcion que regresa la parte imaginaria de un numero z
def Im(z):
  return z.imag

# Funcion que dado un numero z = x + jy, regresa x - jy (el conjugado de z)
def conjug(z):
  return z.conjugate()

# Funcion que regresa la norma de un numero z
def norma_c(z):
  return (z*conjug(z))**(1/2)
```

Ahora, procederemos a recordar cómo se definen ciertas funciones para un número complejo $z \in \mathbb{C}$ de la forma $z = x + iy$:

$$e^z = e^x \cdot (\cos(y) + i\sin(y)), \quad \arg(z) = \arctan(\frac{y}{x}), \quad \log(z) = \log(|z|) + i\arg(z),$$
$$\sin(z) = \frac{e^{iz} - e^{-iz}}{2i}, \qquad \cos(z) = \frac{e^{iz} + e^{-iz}}{2}, \quad \tan(z) = \frac{\sin(z)}{\cos(z)}.$$

Notemos que la función $\arg(z)$ nos indica el ángulo de z con respecto al eje x, pero esta definición tiene como contradominio a $[-\pi/2, \pi/2]$, ya que nos da ángulos en $[0, \pi/2]$ para los cuadrantes $(x, y), (-x, -y)$, y ángulo en $[-\pi/2, 0]$ para los cuadrantes $(x, -y), (-x, y)$. Pero realmente buscamos que el contradominio sea $[0, 2\pi)$. Para esto, proponemos usar la definición $\arg(z) = \arctan 2(y, x)$ en su lugar, pero esta tiene como contradominio a $(-\pi, \pi]$, ya que nos da ángulos en $[0, \pi]$ para los cuadrantes $(x, y), (-x, y)$, y ángulo en $(-\pi, 0]$ para los cuadrantes $(x, -y), (-x, -y)$. Finalmente, definimos a $\arg(z)$ de la forma siguiente para tener un contradominio $[0, 2\pi)$.

$$\arg(z) = \begin{cases} 2\pi + \arctan 2(y, x), & y < 0, \\ \arctan 2(y, x), & \text{o.c.} \end{cases}$$

En la figura 2 se ven las definiciones de las funciones ya mencionadas.

Figura 2:

```python
# Funcion que regresa la exponencial compleja de z
def exp_c(z):
  x, y = split(z)
  return np.exp(x)*( np.cos(y) + 1j*np.sin(y) )

# Funcion que regresa el coseno complejo de z
def cos_c(z):
  return (exp_c(1j*z) + exp_c(-1j*z))/2

# Funcion que regresa el seno complejo de z
def sen_c(z):
  return (exp_c(1j*z) - exp_c(-1j*z))/(1j*2)

# Funcion tangente compleja
def tan_c(z):
  return sen_c(z)/cos_c(z)

# Funcion que regresa el argumento de un numero z (angulo de z respecto al eje real)
def arg_c(z):
  x, y = split(z)
  # El angulo estara en el intervalo [0,2pi
  return np.where(y < 0, 2*pi + np.arctan2(y, x), np.arctan2(y, x)) # Quitamos el signo al angulo

# Variante de arg_c que escala el resultado de  [0,2pi) a [0,360)
def arg_c_grados(z):
  return arg_c(z) * (360/(2*pi))

# Funcion logaritmo compleja de un numero z
def log_c(z):
  return np.log(norma_c(z)) + 1j*arg_c(z)
```

Con el uso de estas funciones base, es que el usuario tendrá las herramientas necesarias para generar una gran cantidad de funciones propias.

2.2. Funciones constructoras

Durante el desarrollo del código, se llegaron a considerar algunos ejemplos para graficar, pero resultaba agobiante el tener que declarar nuevas funciones por cada ligero ajuste. Por lo tanto, se generaron funciones constructoras, las cuales según algunos parámetros dados, declaran funciones adecuadas a los mismos. Estas funciones se dividen en dos tipos, los cuales serán explicados a continuación, junto con algunos ejemplos de estos tipos.

1. **Funciones de variable compleja de la forma** $f : \mathbb{C} \to \mathbb{C}$

   a) **Función de Moebius:** La función de Moebius es de la forma $f(z) = (az + b)/(cz + d)$, donde $a, b, c, d \in \mathbb{C}$, y son tales que $ad \neq bc$.

b) **Polinomios:** Recordemos que hay dos formas de expresar un polinomio, estas son según sus coeficientes $A = \{a_j\}_{j=0}^{n-1}$ (en este caso, $n-1$ será su grado), y según sus ceros $P = \{p_j\}_{j=0}^{n-1}$. Estas formas respectivamente son

$$f_1(z) = \sum_{k=0}^{n-1} a_k z^k, \quad f_2(z) = \prod_{j=0}^{n-1} (z - p_j).$$

Se declararon funciones constructoras para estos tipos de polinomios, según arreglos de coeficientes ordenados A, y arreglos de ceros P. También se implementó un parámetro extra «$flag$», el cual indica si se regresa el polinomio ($flag = True$) o el recíproco del polinomio ($flag = False$).

2. **Parametrizaciónes de curvas $g : \mathbb{R} \to \mathbb{C}$.**

a) **Recta entre dos números complejos** $start, end$**:** Ya se sabe que tal función es de la forma

$$\begin{aligned} g_0(t) : [0,1] &\to \mathbb{C}, \\ t &\mapsto end \cdot t + (1-t) \cdot start. \end{aligned}$$

Pero es de interés que esta función se adapte para poder utilizar un intervalo $[a, b]$ en lugar de $[0, 1]$, para esto se genera una función biyectiva

$$\begin{aligned} b(t) : [a,b] &\to [0,1], \\ t &\mapsto \frac{t-a}{b-a}. \end{aligned}$$

Así, podemos generar una función que une con una recta a los puntos $start, end$ con un intervalo $[a, b]$, y tal función es de la forma $g(t) = g_0(b(t))$.

b) **Círculo de radio r, con centro c:** Esta función está dada por $g(t) = c + e^{it + \log(r)}$, y toma como dominio al intervalo $[0, 2\pi)$.

c) **Circunferencias concéntricas :** Es de interés el generar una parametrización para n circunferencias centradas en un número complejo c, donde los radios tienen tamaño de paso Dr. Esta función tendrá como dominio a $[0, n)$, y el valor n indica el número de circunferencias concéntricas generadas. En este caso, se declaran n funciones $\{h_j\}_{j=0}^{n-1}$, donde h_j es la función de la circunferencia con radio $Dr \cdot (j+1)$, y es de la forma

$$\begin{aligned} h_j : [j, j+1) &\to \mathbb{C}, \\ t &\mapsto circ_j((t-j) \cdot 2\pi). \end{aligned} \qquad \begin{aligned} circ_j : [0, 2\pi) &\to \mathbb{C}, \\ t &\mapsto c + e^{it + \log(Dr \cdot (j+1))}. \end{aligned}$$

Así, la parametrización para las circunferencias concéntricas están dadas por

$$\begin{aligned} g_n : [0, n) &\to \mathbb{C}, \\ t &\mapsto h_{\lfloor t \rfloor}(t). \end{aligned}$$

d) **Rectas paralelas:** Función que regresa $2n$ rectas paralelas a una recta dada por una función (junto con su intervalo asociado), donde todas las rectas paralelas están separadas por una distancia r. Sean p, q los puntos que une la recta dada, definimos los vectores

$$u = (q-p)/(norma\_c(q-p)), \quad v = \overline{u}.$$

Con estos vectores se declara el vector de desplazamiento $d_j = (v \cdot (-(n-j)))$, donde j es el índice de una de las rectas paralelas. Dicho esto, se declaran $2n+1$ funciones $\{r_j\}_{j=0}^{2n}$, donde r_j es de la forma

$$\begin{aligned} r_j : [j, j+1) &\to \mathbb{C}, \\ t &\mapsto recta_j(t). \end{aligned}$$

$recta_j(t)$ une a los puntos $(p + d_j)$ y $(q + d_j)$, para un intervalo $[j, j+1)$. Por lo tanto la función de las $2n$ rectas paralelas esta dada por

$$\begin{aligned} g : [0, 2n) &\to \mathbb{C}, \\ t &\mapsto r_{\lfloor t \rfloor}(t). \end{aligned}$$

e) **Malla de rectas:** Función que genera una malla de n rectas horizontales y n rectas verticales, centradas en un número c, cada una de longitud l, y separadas por una distancia $2r$. La función en cuestión toma dos casos en relación a sí n es par o impar:

1) n **es par:** Las n primeras rectas en ser generadas, son las horizontales, comenzando desde abajo, y las n rectas siguientes son las verticales, comenzando desde la izquierda. Considerando la primera recta horizontal y la primera recta vertical, tenemos que estas unen a los puntos p_1, p_2 y q_1, q_2 respectivamente. Tales puntos son de la forma

$$p_1 = c - \left(\frac{l}{2}\right) - i\left(2r\left(\frac{n}{2}-1\right)+r\right), \quad q_1 = c - \left(2r\left(\frac{n}{2}-1\right)+r\right) + i\left(\frac{l}{2}\right),$$
$$p_2 = c + \left(\frac{l}{2}\right) - i\left(2r\left(\frac{n}{2}-1\right)+r\right), \quad q_2 = c - \left(2r\left(\frac{n}{2}-1\right)+r\right) - i\left(\frac{l}{2}\right).$$

Para la $j-th$ recta (para $j < n$, la recta es horizontal), declaramos la función

$$h_j : [j, j+1) \to \mathbb{C},$$
$$t \mapsto recta_{1,j}(t).$$

Donde $recta_{1,j}(t)$ une a los puntos $(p_1 + i(2rj))$ y $(p_2 + i(2rj))$ para un intervalo $[j, j+1)$. Mientras que para la $h-th$ recta (para $h \geq n$, la recta es vertical), declaramos la función

$$v_h : [h, h+1) \to \mathbb{C},$$
$$t \mapsto recta_{2,h}(t).$$

Donde $recta_{2,h}(t)$ une a los puntos $(q_1 + 2r(h-n))$ y $(q_2 + 2r(h-n))$ para un intervalo $[h, h+1)$. De este modo, la parametrización para la malla en este caso está dada por

$$m : [0, 2n) \to \mathbb{C},$$
$$t \mapsto \begin{cases} h_{\lfloor t \rfloor}(t), & t < n, \\ v_{\lfloor t \rfloor}(t), & \text{o.c .} \end{cases}$$

2) n **es impar:** Las n primeras rectas en ser generadas, son las horizontales, comenzando desde abajo, y las n rectas siguientes son las verticales, comenzando desde la izquierda. Considerando la primera recta horizontal y la primera recta vertical, tenemos que estas unen a los puntos p_1, p_2 y q_1, q_2 respectivamente. Tales puntos son de la forma

$$p_1 = c - \left(\frac{l}{2}\right) - i\left(2r\left(\frac{n-1}{2}\right)\right), \quad q_1 = c - \left(2r\left(\frac{n-1}{2}\right)\right) + i\left(\frac{l}{2}\right),$$
$$p_2 = c + \left(\frac{l}{2}\right) - i\left(2r\left(\frac{n-1}{2}\right)\right), \quad q_2 = c - \left(2r\left(\frac{n-1}{2}\right)\right) - i\left(\frac{l}{2}\right).$$

Para la $j-th$ recta (para $j < n$, la recta es horizontal), declaramos la función

$$h_j : [j, j+1) \to \mathbb{C},$$
$$t \mapsto recta_{1,j}(t).$$

Donde $recta_{1,j}(t)$ une a los puntos $(p_1 + i(2rj))$ y $(p_2 + i(2rj))$ para un intervalo $[j, j+1)$. Mientras que para la $h-th$ recta (para $h \geq n$, la recta es vertical), declaramos la función

$$v_h : [h, h+1) \to \mathbb{C},$$
$$t \mapsto recta_{2,h}(t).$$

Donde $recta_{2,h}(t)$ une a los puntos $(q_1 + 2r(h-n))$ y $(q_2 + 2r(h-n))$ para un intervalo $[h, h+1)$. De este modo, la parametrización para la malla en este caso está dada por

$$m : [0, 2n) \to \mathbb{C},$$
$$t \mapsto \begin{cases} h_{\lfloor t \rfloor}(t), & t < n, \\ v_{\lfloor t \rfloor}(t), & \text{o.c .} \end{cases}$$

3. Imágenes de dominios bajo funciones holomorfas

El principal objetivo de esta investigación es el de permitir al usuario el graficar una gran variedad de funciones complejas. Por esta razón, se han desarrollado múltiples opciones para que el usuario pueda graficar las funciones de su interés.

Por el momento, únicamente se explicará el funcionamiento de una de estas opciones, ya que las otras son derivadas de esta misma.

Primero explicaremos el funcionamiento de la función que genera la curva parametrizada Γ según una parametrización g y un intervalo $[a, b]$ (ó $[a, b)$ según sea el caso). Esta función considera dos casos:

1. g **es continua:** En el caso que $g()$ sea una función continua, simplemente se genera un arreglo T de 100000 puntos equidistantes en el intervalo $[a, b]$, y se regresa $\Gamma = g(T)$.

2. g **es continua por pedazos:** En el caso que $g()$ sea continua por pedazos, se tiene que la función está definida en $(b - a)$ secciones, esto es

$$g(t) = \begin{cases} g_0(t), & t \in [a, a+1), \\ g_1(t), & t \in [a+1, a+2), \\ \vdots & \vdots \\ g_{(b-a)-1}(t), & t \in [b-1, b). \end{cases}$$

   Por lo tanto, se generan arreglos $\{T_j\}_{j=0}^{(b-a)-1}$, donde T_j es un arreglo de 100000 puntos equidistantes en el intervalo $[a+j, (a+j)+1-\epsilon]$, donde $\epsilon > 0$ es un número lo suficientemente pequeño para simular $[a+j, (a+j)+1)$. De esta forma definimos $\Gamma_j := g(T_j)$, y se regresa $\Gamma = [\Gamma_0, \Gamma_1, ..., \Gamma_{(b-a)}]$.

En la figura 3 se puede apreciar la declaración de esta función.

Figura 3:

```
# Dada una parametrizacion con intervalo [a,b] o [a,b), regresa un vector de 100000 puntos
# que representen a la curva
def curve_generator(g, a, b, continua = True):
    if continua:
        T = np.linspace(a,b,100000)
        return g(T)
    else: # En caso se que g() no sea una parametrizacion continua, obtenemos sus fragmentos por separado
        Curve = []

        for i in range(b-a):
            T = np.linspace(i,i+1-epsilon,100000) # Se toma [i, i+1-epsilon] para no mezclar fragmentos
            Curve.append(g(T))
        return Curve
```

Ahora, pasaremos a la explicación de la función graficadora. En este caso se explicará cómo funciona la función que grafica una curva parametrizada $\Gamma = g([a, b])$ (ó $\Gamma = g([a, b))$), junto a su transformación $f(\Gamma)$. Del mismo modo que en la función previa, se consideran dos casos en relación a si $g()$ es una función continua o continua por pedazos:

1. g **es continua:** En el caso en que $g()$ sea una función continua, se genera la curva Γ utilizando la función vista la figura 3, luego con el apoyo de la función *split*(), obtenemos los arreglos $X0, Y0$, los cuales representan la parte real e imaginaria respectivamente de Γ. Del mismo modo, utilizando a *split*() con $f(\Gamma)$, obtenemos los arreglos $X1, Y1$, los cuales representan la parte real e imaginaria respectivamente de $f(\Gamma)$. Utilizando estos arreglos, generamos dos gráficas en dos planos (x, y), para así representar a Γ y a $f(\Gamma)$.

2. g **es continua por pedazos:** En el caso en que $g()$ sea una función continua por pedazos, es bastante similar al caso donde es continua. Se genera de igual forma a la curva Γ utilizando la función vista la figura 3, luego con el apoyo de la función $split()$, obtenemos los arreglos $X0_j, Y0_j$, los cuales representan la parte real e imaginaria respectivamente de Γ_j. En el primer plano (x, y), graficamos $[X0_j, Y0_j]$ para representar a Γ. Del mismo modo, utilizando a $split()$ con $f(\Gamma_j)$, obtenemos los arreglos $X1_j, Y1_j$, los cuales representan la parte real e imaginaria respectivamente de $f(\Gamma_j)$. Luego en el segundo plano (x, y), graficamos $[X1_j, Y1_j]$ para representar a $f(\Gamma)$.

Los dos casos anteriores se pueden apreciar en la figura 4.

Figura 4:

```python
# Funcion graficadora.
# Para graficar, se manda una parametrizacion g junto con su intervalo [a,b] o [a,b),
# así como la funcion f sobre la cual graficaremos.
# Tambien se considera la bandera continua, la cual indica si la parametrizacion es de una curva continua
def graficadora(g, a, b, f, color = 'b', continua = True):

  # Graficamos Gamma
  if continua:
    # Curva a mandar a funcion
    Gamma = curve_generator(g, a, b)
    X0, Y0 = split(Gamma)

    # Cuva resultado de funcion
    Gamma2 = f(Gamma)
    X1, Y1 = split(Gamma2)

    plt.subplot(1, 2, 1)
    plt.plot(X0, Y0, c = 'red')
    plt.xlabel('Parte real')
    plt.ylabel('Parte Imaginaria')
    plt.title('Gráfico de g(t)')
    plt.axis('equal')

    plt.subplot(1, 2, 2)
    plt.plot(X1, Y1, c =  color)
    plt.xlabel('Parte real')
    plt.ylabel('Parte Imaginaria')
    plt.title('Gráfico de f(g(t))')
    plt.axis('equal')

  else:
    Gammas = curve_generator(g, a, b, continua)
    plt.subplot(1, 2, 1)
    for i in range(len(Gammas)):
      X0, Y0 = split(Gammas[i])
      plt.plot(X0, Y0, c = 'red')
      plt.xlabel('Parte real')
      plt.ylabel('Parte Imaginaria')
      plt.title('Gráfico de g(t)')
      plt.axis('equal')

    plt.subplot(1, 2, 2)
    for i in range(len(Gammas)):
      X1, Y1 = split(f(Gammas[i]))
      plt.plot(X1, Y1, c =  color)
      plt.xlabel('Parte real')
      plt.ylabel('Parte Imaginaria')
      plt.title('Gráfico de f(g(t))')
      plt.axis('equal')


  plt.tight_layout()
  #plt.grid(True)
  plt.show()
```

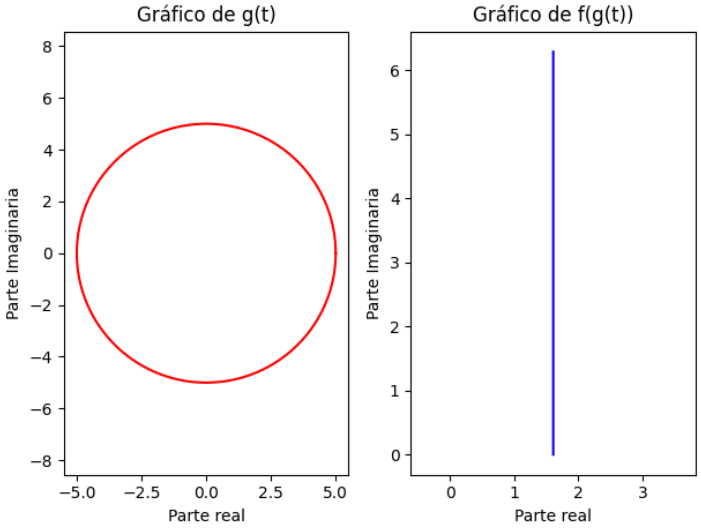
A continuación, presentaremos algunos ejemplos de imágenes de dominios de ciertas parametrizaciónes de la sección 2.2, bajo ciertas funciones holomorfas $f(z)$.

- **Ejemplo 1:** Parametrización de una recta al ser transformada por $f(z) = \exp(z)$.

Figura 5: ../Proyecto Final/Ex 4.png

- **Ejemplo 2:** Parametrización de una circunferencia al ser transformada por $f(z) = \log(z)$.

Figura 6:



- **Ejemplo 3:** Parametrización de circunferencias concéntricas al ser transformada por $f(z) = z^3$.

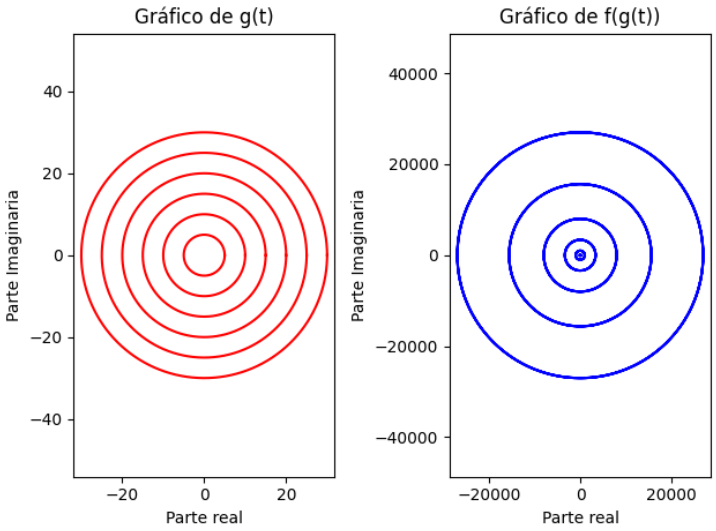


Figura 7:

- **Ejemplo 4:** Parametrización de rectas paralelas al ser transformada por $f(z) = \exp(z)$

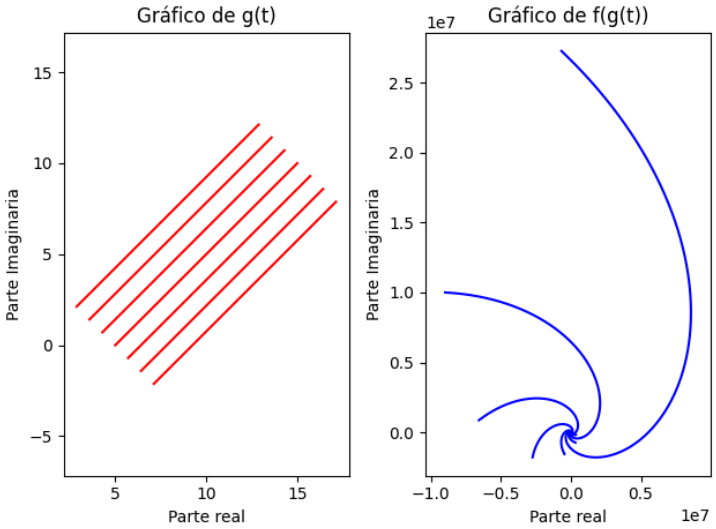


Figura 8:

- **Ejemplo 5:** Parametrización de malla transformada por $f(z) = z^2$

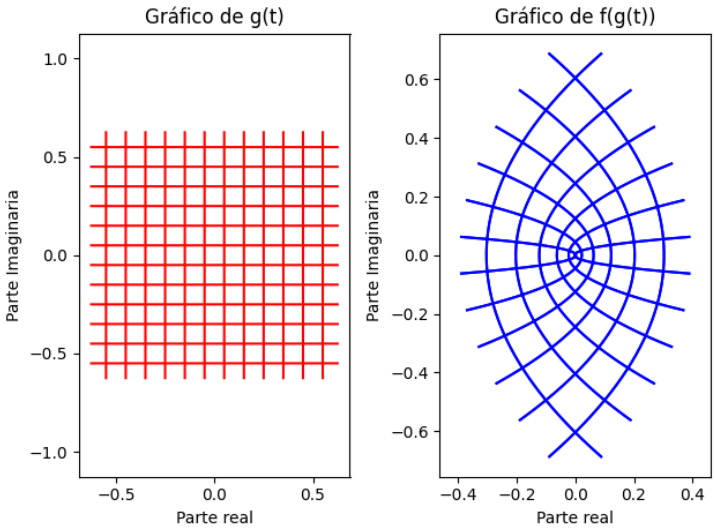


Figura 9:

- **Ejemplo 6:** Parametrización de malla transformada por $f(z) = z^3$

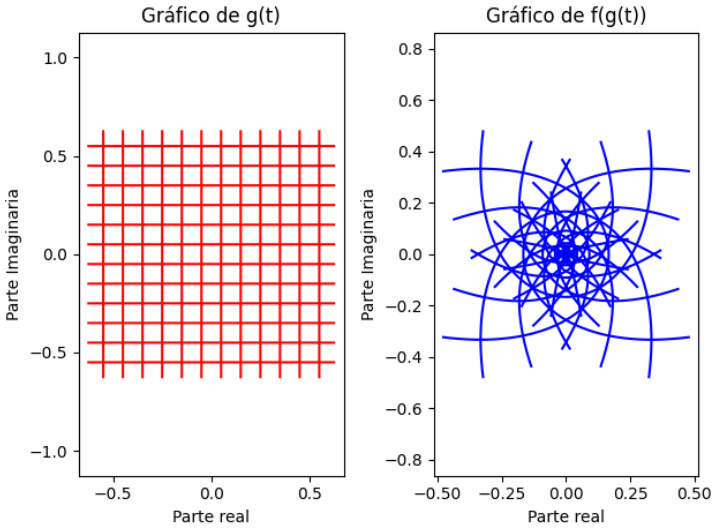


Figura 10:

- **Ejemplo 7:** Parametrización de malla transformada por $f(z) = 1/z$.

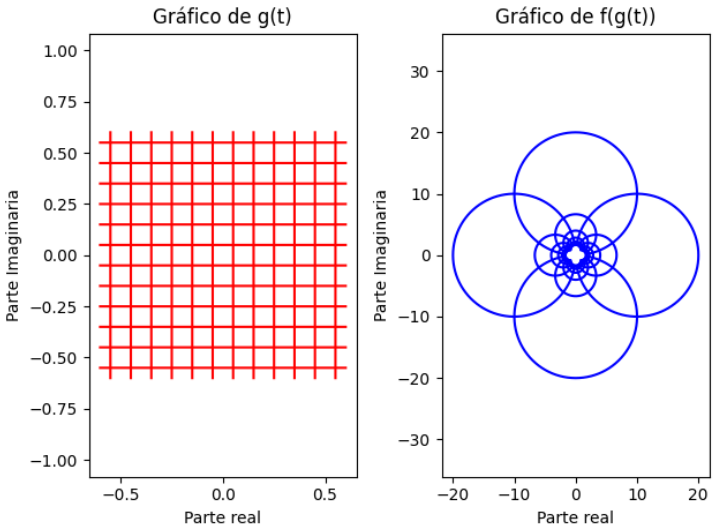


Figura 11:

- **Ejemplo 8:** Parametrización de malla transformada por $f(z) = e^z$

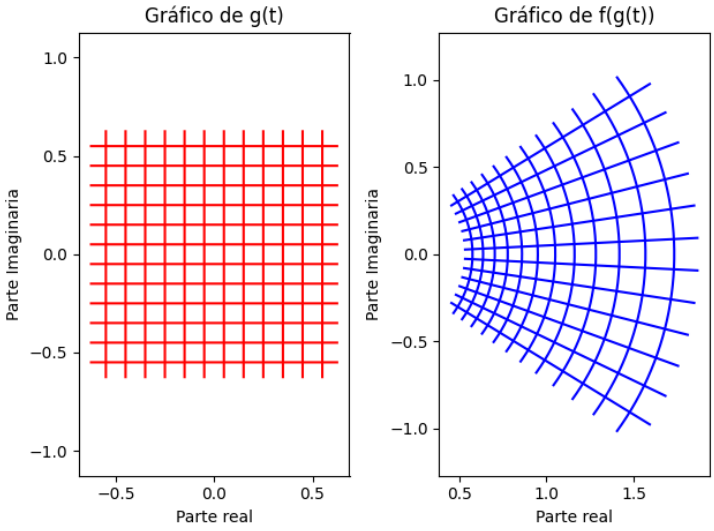


Figura 12:

- **Ejemplo 9:** Parametrización de malla transformada por $f(z) = \log(z)$

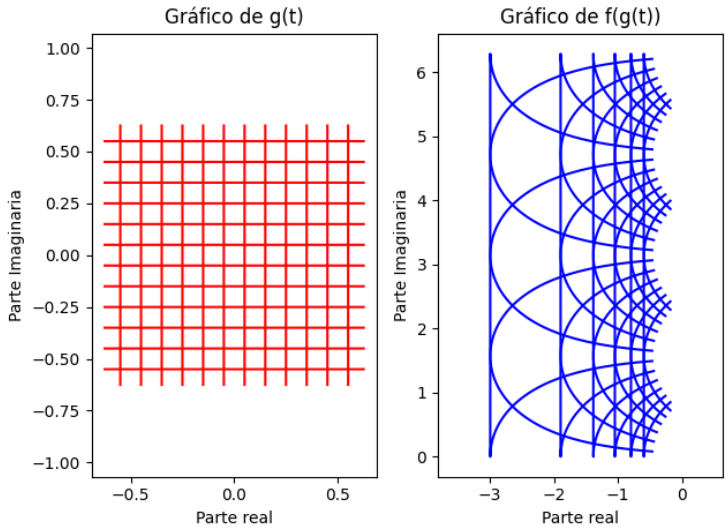


Figura 13:

4. Representación gráfica de funciones complejas

Ya hemos visto los métodos para graficar a Γ y a $f(\Gamma)$ en dos planos (x, y), esto nos permite tener una mejor comprensión sobre las funciones $f : \mathbb{C} \to \mathbb{C}$ y sus gráficas, a pesar de no poder visualizarlas en sus totalidad. Pero algo que si podemos visualizar en su totalidad son funciones $g : \mathbb{C} \to \mathbb{R}$, ya que se pueden visualizar en un plano (x, y, z).

De esta forma introducimos las funciones de graficación $\mathbb{C} \to \mathbb{R}$, las cuales toman un dominio $\Gamma \subset \mathbb{C}$, y muestran $g(f(\Gamma))$, donde f es una función de variables complejas, y $g : \mathbb{C} \to \mathbb{R}$ es una función que nos permite enfocarnos en un componente de $f(\Gamma)$.

Primero, explicaremos el como obtener Γ. Para estos casos, se toma a $\Gamma \subset \mathbb{C}$, cómo

$$\Gamma := X + iY,$$

donde X, Y son superficies de malla en base a arreglos x, y de 100 puntos equidistantes según ciertos límites. De esta forma, tenemos un dominio continuo en $\mathbb{C}$. Luego, obtenemos $Z = g(f(\Gamma))$, y finalmente tenemos tres mallas X, Y, Z, con las cuales podemos llevar a cabo la graficación en $\mathbb{C} \to \mathbb{R}$.

A continuación, presentaremos algunos ejemplos para la función $f(z) = z^2$, y ciertas funciones $g(z)$.

- **Ejemplo 1:** $f(z) = z^2$, $g(z) = Re(z)$

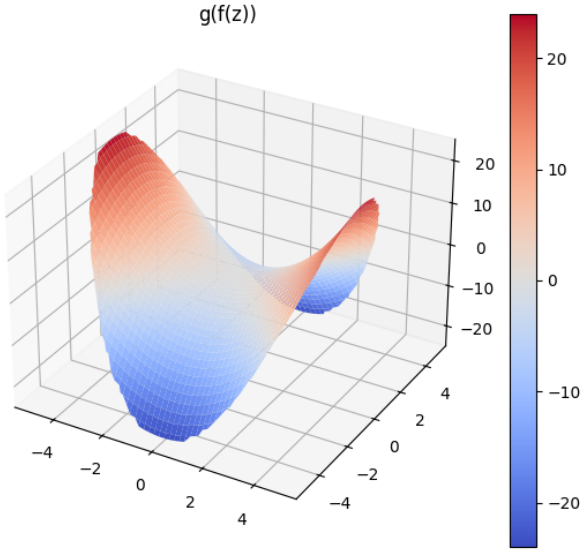


Figura 14:

- **Ejemplo 2:** $f(z) = z^2$, $g(z) = Im(z)$

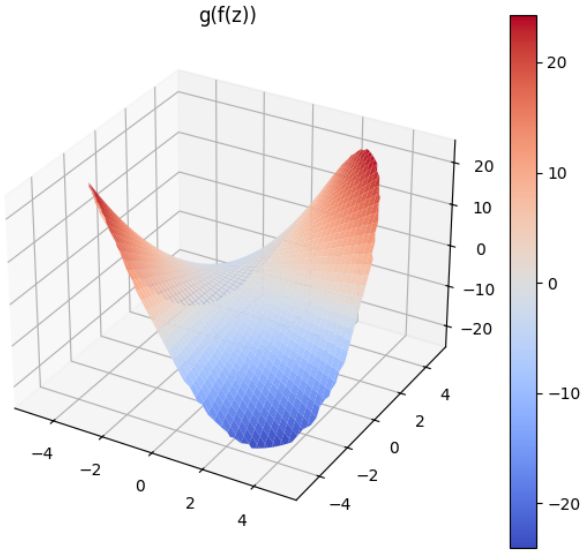


Figura 15:

- **Ejemplo 3:** $f(z) = z^2$, $g(z) = |z|$

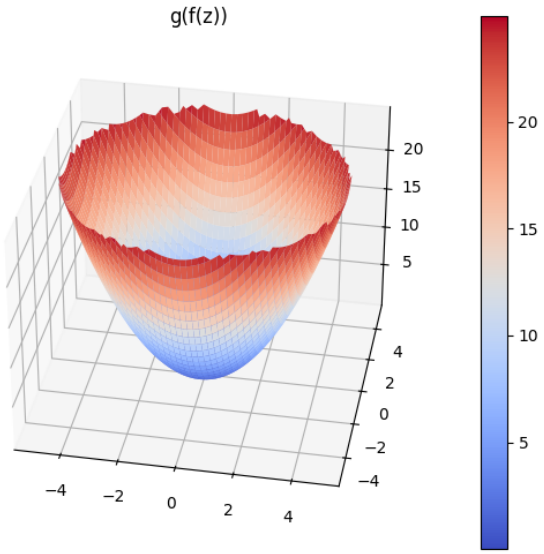


Figura 16:

- **Ejemplo 4:** $f(z) = z^2$, $g(z) = arg\_c(z)$

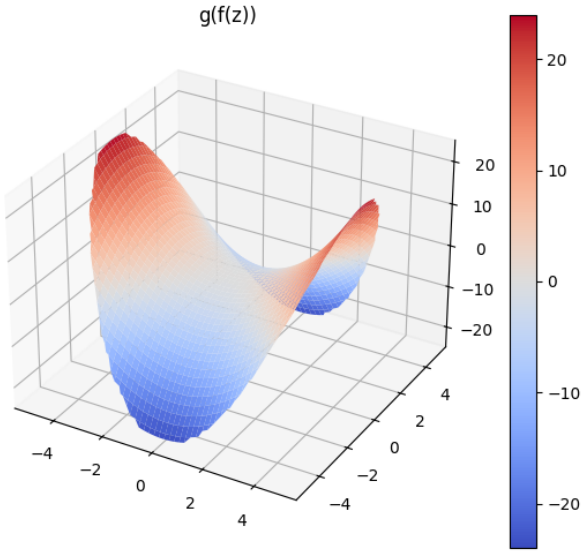


Figura 17:

5. Curvatura Gaussiana de la gráfica de una función holomorfa

Sea $\Omega \subset \mathbb{C}$ una región conexa y abierta, y sea $f : \Omega \to \mathbb{C}$ una función holomorfa, escribimos

$$f(x, y) = u(x, y) + iv(x, y), \quad z = x + iy \in \Omega,$$

donde $u, v : \Omega \to \mathbb{R}$ son funciones diferenciables. Definimos a la gráfica de f como el conjunto

$$\Gamma(f) = \left\{(z, f(z)) \in \mathbb{C}^2 : z \in \Omega\right\}.$$

Luego, considerando que $\Gamma(f) \subset \mathbb{R}^4$ es una superficie real, podemos definir una parametrización para este conjunto, de la forma

$$\begin{aligned} &\psi : \mathbb{C} \to \mathbb{R}^4, \\ &(x, y) \mapsto (x, y, u(x, y), v(x, y)). \end{aligned}$$

Dada esta parametrización, es que se tiene el siguiente Teorema, que se puede apreciar más a detalle en "The Gaussian Curvature of Graphs of Holomorphic Functions".

Teorema (Cruz-López, Romero). Sea $f = u + iv : \Omega \to \mathbb{C}$ una función holomorfa. Para todo $z \in \Omega$, la curvatura Gaussiana K de la gráfica de f en el punto $(z, f(z)) \in \Gamma(f)$ es dada por

$$\begin{aligned} K &= -\frac{\Delta E}{2\left[1 + |f'(z)|^2\right]^3} \\ &= -\frac{\Delta E}{2E^3}. \end{aligned}$$

Donde $E = 1 + u_x^2 + v_x^2$. ■

El Teorema anterior es aplicado en la elaboración del código, para generar una función constructora de la función K, en base a las funciones u, v. Cabe resaltar que u, v tiene que ser expresiones que utilicen a $x0, y0$ en lugar de x, y. Esto se realiza debido a que el código necesita crear símbolos distintos a los usados en el resto del código para derivar. Por ejemplo, para la función $f(z) = z^2$, se tiene que

$$u(x, y) = x^2 - y^2, \quad v(x, y) = 2xy.$$

Por lo tanto se tienen que u, v deben ser expresiones de la forma

$$u(x, y) = (x0)^2 - (y0)^2, \quad v(x, y) = 2(x0 \cdot y0).$$

Ahora, para graficar, se emplea un proceso análogo al de las gráficas de funciones $g : \mathbb{C} \to \mathbb{R}$. Con la diferencia en que en lugar de requerir de funciones f, g, únicamente se requiere de las expresiones u, v, ya que Z estará dada por $Z = K(X, Y)$.

A continuación,presentaremos algunos ejemplos para ciertas funciones $f(x+iy) = u(x, y)+iv(x, y)$, junto con sus respectivas funciones $K(x, y)$.

- **Ejemplo 1:**

$$f(x+iy) = (x+iy)^3 = \left(x^3 - 3xy^2\right) + i\left(3yx^2 - y^3\right),$$
$$K(x,y) = \frac{72 \cdot \left(-x^2 - y^2\right)}{\left(36x^2y^2 + 9 \cdot (x^2 - y^2)^2 + 1\right)^3}.$$

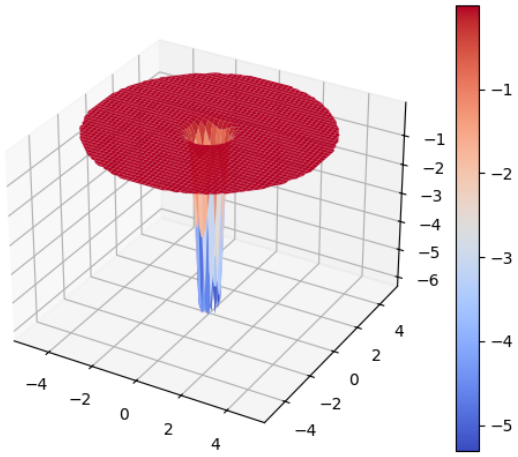


Figura 18:

- **Ejemplo 2:**

$$f(x+iy) = (x+iy)^2 = \left(x^2 - y^2\right) + i\left(2xy\right),$$
$$K(x,y) = \frac{-8}{\left(4x^2 + 4y^2 + 1\right)^3}.$$

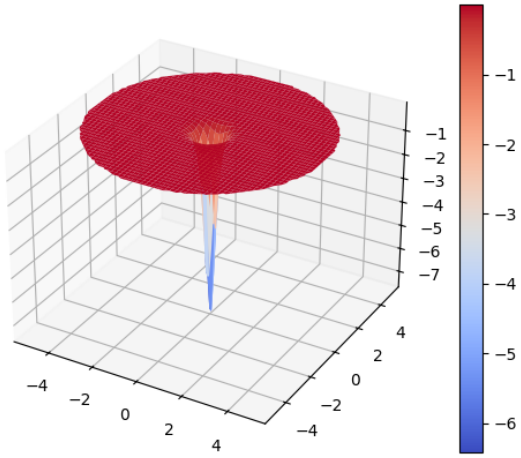


Figura 19:

- **Ejemplo 3:**

$$f(x+iy) = \frac{1}{(x+iy)} = \frac{x}{x^2+y^2} + i\left(\frac{-y}{x^2+y^2}\right),$$

$$K(x,y) = \frac{-8\cdot\left(x^4+2x^2y^2+y^4\right)^3}{\left(x^4+2x^2y^2+y^4+1\right)^3\cdot\left(x^6+3x^4y^2+3x^2y^4+y^6\right)}.$$

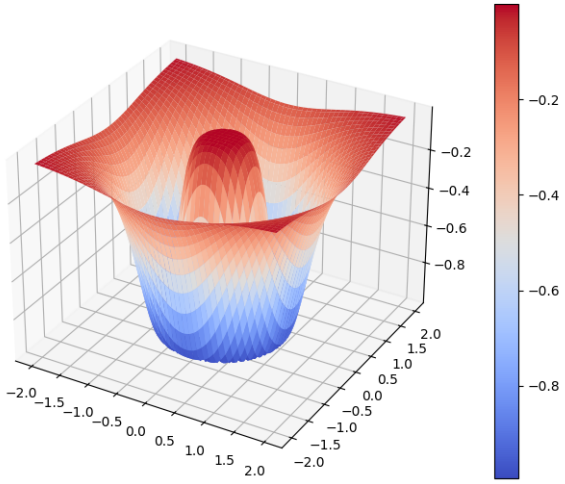


Figura 20:

- **Ejemplo 4:**

$$f(x+iy) = \frac{1}{(x+iy)^2} = \left(\frac{x^2-y^2}{(x^2+y^2)^2}\right) + i\left(\frac{-2xy}{(x^2+y^2)^2}\right),$$

$$K(x,y) = \frac{-72\cdot(x^2+y^2)^{18}}{\left(4x^2\cdot(x^2-3*y^2)^2+4y^2\cdot(3x^2-y^2)^2+(x^2+y^2)^6\right)^3\cdot\left(x^8+4x^6y^2+6x^4y^4+4x^2y^6+y^8\right)}.$$

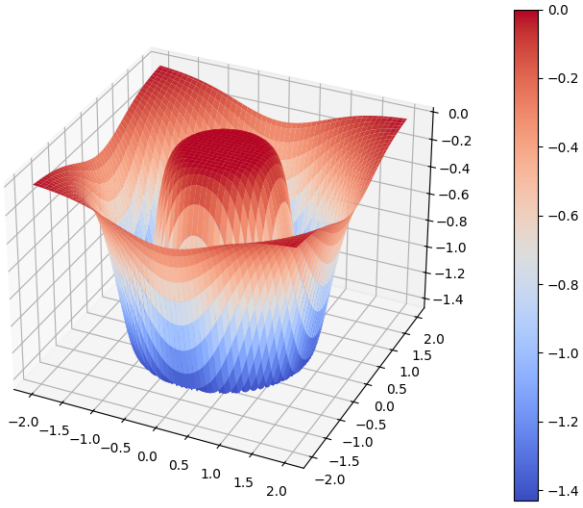


Figura 21:

- **Ejemplo 5:**

$$\begin{aligned} f(x+iy) &= \frac{1}{10}\left(((x+iy)+i+1)((x+iy)-i+1)((x+iy)+i-1)((x+iy)-i-1)\right) \\ &= \left(\frac{1}{10x^4} - \frac{3}{5x^2y^2} + \frac{1}{10y^4} + \frac{2}{5}\right) + i\left(\frac{2}{5x^3y} - \frac{2}{5xy^3}\right), \end{aligned}$$

$$K(x,y) = \frac{-45000 \cdot (x^2+y^2)^2}{(4x^6+12x^4y^2+12x^2y^4+4y^6+25)^3}.$$

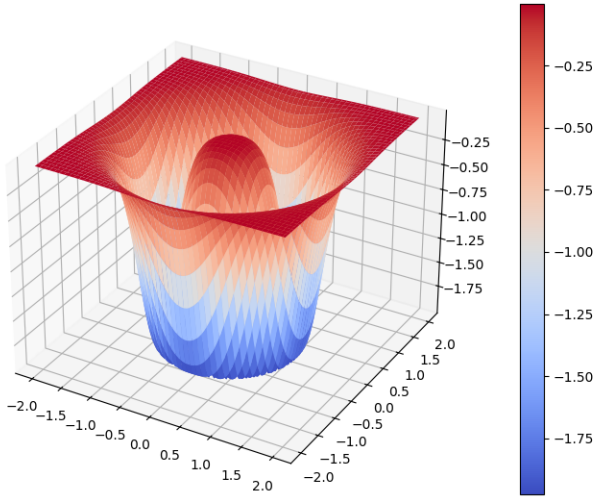


Figura 22:

- **Ejemplo 6:**

$$\begin{aligned} f(x+iy) &= \frac{1}{4}\left(((x+iy)+2)\left((x+iy)-2i\right)\left((x+iy)-2\right)\right) \\ &= \left(\frac{1}{4x^3}-\frac{3}{4xy^2}+xy-x\right)+i\left(\frac{3}{4x^2y}-\frac{1}{4y^3}-\frac{1}{2x^2}+\frac{1}{2y^2}-y+2\right), \end{aligned}$$

$$K(x,y)=\frac{-2048\cdot(9x^2+9y^2-12y+4)}{(9x^4+18x^2y^2+9y^4-24x^2y-24y^3-8x^2+40y^2-32y+32)^3}.$$

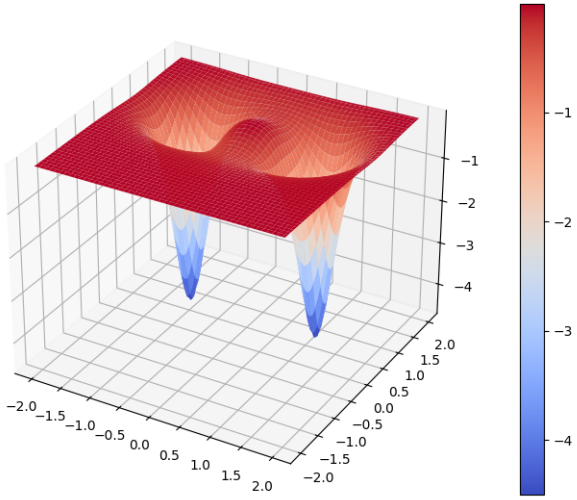


Figura 23:

6. Conclusión

El trabajar con números complejos es una tarea complicada hasta cierto punto, en especial cuando no se tiene una imagen clara del comportamiento de los mismos al ser transformados por funciones complejas. Por esta razón, el desarrollar un código en Python que permite graficar libremente funciones complejas, resulta en ser una herramienta de gran eficacia en versatilidad.

Con el código en cuestión, se ha podido representar de manera visual el comportamiento de funciones complejas, así como sus propiedades. Lo que permite tener una comprensión más profunda sobre los conceptos de las áreas relacionadas con la variable compleja.

La capacidad del código para manejar diferentes tipos de funciones, así como de adaptarse y evolucionar según las propias necesidades del usuario y de sus capacidades, hace que el código sea especialmente útil para investigadores, maestros y estudiantes en Matemáticas. Pero sus posibles aplicaciones no se limitan únicamente al área de las matemáticas, sino que pueden abarcar áreas como la ingeniería, mecánica de fluidos, electromagnetismo y física.

Esta investigación destaca la importancia de las herramientas computacionales en la investigación y educación. En un futuro, se podrían considerar mejoras adicionales a este código ya implementado, como la optimización para funciones más avanzadas o incorporación de una interfaz de usuario (similar a Geogebra) para facilitar su uso y llegar a un público más amplio.

El código en cuestión ofrece una solución accesible para la graficación de funciones complejas, generando un recurso didáctico y de investigación, el cual puede ser empleado en una gran multitud de áreas, y para una gran cantidad de propósitos.

7. Bibliografía

1. Cabrera García, Baruc Samuel. **Graficadora de Funciones Complejas.py**, **Ejemplos**. Disponibles en https://drive.google.com/drive/folders/1fnLVO3JzpxgE-D6vCnIUJ_xVFk-lnZuD?usp=sharing

2. Lutz, M. (2013). **Learning Python: Powerful Object-Oriented Programming**. O'Reilly Media, Inc.

3. Maggi, Claudio; Martín, Héctor; De Rosa Maria Anna. **Aplicaciones Gráficas de Funciones Complejas.** https://frrq.cvg.utn.edu.ar/pluginfile.php/28233/mod_resource/content/1/Aplicaciones%20gr%C3%A1ficas%20de%20funciones%20complejas%20Maggi.pdf

4. Manuel Cruz-López; Otto Romero. **The Gaussian Curvature of Graphs of Holomorphic Functions**. Enviado para su publicación.

5. Marsden, J. E. (1973). **Basic complex analysis**.

6. **Números complejos: qué son, origen, características, relevancia** - Ferrovial. (2022, 2 noviembre). Ferrovial. https://www.ferrovial.com/es/stem/numeros-complejos/#:~:text=Su%20uso%20abarca%20distintas%20ramas,o%20las%20telecomunicaciones%20es%20fundamental.