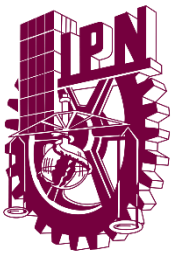


INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

CALCULADORA DE RESIDUOS Y CONTORNOS PARA FUNCIONES RACIONALES Y TRASCENDENTALES: IMPLEMENTACION EN PHYTON

Alumnos:

- Alva Pérez Alan Freddy
- Pensamiento Robledo Sabrina Erika
- Hernández Castañeda Nailea

Profesor: Correa Coyac David

Grupo: 4CM3

Materia: Matemáticas Avanzadas para la Ingeniería

Índice

Abstracto

Este trabajo presenta el desarrollo e implementación de una calculadora computacional en Python diseñada para el análisis de funciones complejas, con énfasis en el cálculo de residuos, la visualización de contornos y la estimación de integrales mediante el teorema del residuo. La herramienta aborda funciones racionales (ej. $f(z) = \frac{1}{z^2+1}$) y trascendentes (ej. $f(z) = \frac{e^z}{z}$) automatizando la identificación de polos, el cálculo simbólico de residuos (utilizando la librería sympy) y la graficación de contornos en el plano complejo (mediante matplotlib). Además, se aplica el teorema del residuo para evaluar integrales complejas y reales, validando los resultados con casos teóricos conocidos. La calculadora provee una interfaz intuitiva que facilita el estudio de singularidades y su relación con la geometría del contorno, demostrando su utilidad en la docencia y la investigación en análisis complejo.

Descripción del problema

Contexto

El análisis de funciones complejas y el cálculo de residuos juegan un papel fundamental en diversas áreas de las matemáticas aplicadas, la física y la ingeniería. En particular, el teorema del residuo permite evaluar integrales complejas y reales que serían difíciles (o incluso imposibles) de resolver mediante técnicas tradicionales de cálculo. Este teorema establece una conexión profunda entre el comportamiento local de una función alrededor de sus singularidades (polos) y el valor global de una integral sobre un contorno cerrado.

Sin embargo, el cálculo manual de residuos y la selección adecuada de contornos presentan desafíos significativos, especialmente en funciones trascendentes o con polos de orden superior. La identificación precisa de singularidades, la aplicación de fórmulas de residuos y la verificación de las condiciones de convergencia requieren un alto grado de atención y destreza analítica. Además, la visualización geométrica de contornos y su relación con la ubicación de los polos es crucial para garantizar la correcta aplicación del teorema, algo que no siempre es intuitivo al trabajar únicamente con herramientas algebraicas.

En este contexto, surge la necesidad de una herramienta computacional que automatice estos procesos, reduzca errores humanos y permita una exploración interactiva del plano complejo. Implementar dicha herramienta en Python ofrece ventajas clave, gracias a su capacidad para integración simbólica (con sympy), cálculo numérico eficiente (con numpy) y visualización dinámica (con matplotlib). Esta calculadora no solo agiliza el estudio teórico, sino que también sirve como recurso pedagógico para ilustrar conceptos abstractos del análisis complejo mediante ejemplos concretos y visualizaciones interactivas.

Problema específico

El análisis de integrales complejas mediante el teorema del residuo presenta tres desafíos fundamentales que justifican el desarrollo de esta herramienta computacional:

- Complejidad en el cálculo manual de residuos:
  a. En funciones racionales con polos múltiples (ej. $f(z) = \frac{1}{(z^2+1)^3}$), el cálculo de residuos requiere derivadas de orden superior que son propensas a errores.
  b. Para funciones trascendentales (ej. $f(z) = \frac{e^z}{z^4+1}$), la identificación y clasificación de singularidades esenciales resulta particularmente compleja.
- Dificultades en la visualización geométrica:

    a. La selección óptima del contorno de integración depende críticamente de la distribución espacial de los polos.
    b. La representación gráfica manual de contornos y singularidades no permite una exploración dinámica de diferentes configuraciones.
- Limitaciones en la verificación numérica.
    a. La validación de resultados teóricos requiere implementaciones numéricas independientes.
    b. Las herramientas existentes no integran visualización con cálculo simbólico en un único entorno.

Caso ilustrativo.

Al evaluar $\int_{-\infty}^{\infty} \frac{\cos(x)}{x^2+1} dx$ mediante la integración compleja:

- El proceso manual requiere:
    a. Identificación de los polos.
    b. Cálculo de los residuos.

Cada paso conlleva posibles errores que nuestra calculadora busca minimizar mediante automatización y visualización.

Esta problemática motiva el desarrollo de una solución computacional que unifique:

- Cálculo simbólico preciso de los residuos.
- Visualización del plano complejo.
- Estimación numérica de integrales.

La implementación en Phyton aprovecha las ventajas de:

- Sympy para calculo exacto de residuos.
- Matplotlib para visualizaciones.
- Numpy para verificaciones numéricas.

Este enfoque integrado supera las limitaciones de los métodos tradicionales, proporcionando una herramienta versátil para casos de estudio en análisis complejo.

Objetivos

El desarrollo de esta calculadora de residuos y contornos en Phyton persigue tres objetivos fundamentales, articulados en torno a las necesidades identificadas en el análisis complejo.

Objetivo principal:

Desarrollar una herramienta integral que automatice el proceso completo del análisis de funciones complejas, desde el cálculo de residuos hasta la estimación de

integrales mediante el teorema del residuo, combinando precisión simbólica con visualización.

Objetivos específicos:

- Implementación computacional.
  a. Crear algoritmos eficientes para el cálculo automático de residuos en funciones racionales y trascendentales.
  b. Desarrollar módulos para la clasificación precisa de singularidades (polos simples, múltiples y esenciales).
  c. Implementar métodos numéricos para validación de los resultados simbólicos.
- Visualización avanzada.
  a. Diseñar representaciones graficas de contornos y su relación con la distribución de polos.
  b. Generar visualizaciones que ilustren el teorema del residuo en acción.

Fundamentación matemática

Teoría del residuo

Primero se explicará que es un residuo y como puede usarse para evaluar integrales del tipo:

$$\oint_c f(z)dz$$

Estas serán integrales de contorno tomadas alrededor de una trayectoria simple cerrada C. Si $f(z)$ es analítica en todas partes sobre C y dentro de C, entonces tal integral es cero debido al teorema de la integral de Cauchy.

Si $f(z)$ tiene una singularidad en el punto $z = z_0$ en el interior de C, pero de otra forma es analítica sobre C y dentro de C, entonces $f(z)$ tiene una serie de Laurent

$$f(z) = \sum_{n=0}^{\infty} a_n(z - z_0) + \frac{b_1}{z - z_0} + \frac{b_2}{(z - z_0)^2} + \cdots$$

que converge para todos los puntos próximos a $z = z_0$ (excepto en $z = z_0$ mismo), en algún dominio de la forma $0 < |z = z_0| < R$. Ahora, el coeficiente b_1 de la primera potencia negativa $\frac{1}{z-z_0}$ de esta serie de Laurent esta dado por la formula integral

$$\frac{1}{2\pi i}\oint_c \frac{f(z*)}{(z* - z_0)^{n+1}} dz*$$

Es decir

$$b_1 = \frac{1}{2\pi i}\oint_c f(z)dz$$

pero como las series de Laurent pueden obtenerse por varios métodos sin usar las fórmulas integrales para los coeficientes, b_1 puede encontrarse aplicando uno de tales métodos y luego aplicar la fórmula para b_1 a fin de evaluar la integral:

$$\oint_c f(z)dz = 2\pi i b_1$$

Aquí se integra en el sentido contrario al movimiento de las manecillas del reloj alrededor de la trayectoria simple cerrada C que contiene a $z = z_0$ en su interior.

El coeficiente b_1 se denomina residuo de $f(z)$ en $z = z_0$, y se denota por

$$b_1 = Res\ f(z)_{z=z_0}$$

Antes de proseguir con la integración, se pregunta lo siguiente. Para obtener un residuo, un simple coeficiente de una serie de Laurent, ¿es necesario obtener toda la serie o existe una manera más económica? Para los polos, la respuesta es que sí existe una manera más económica. Entonces, la fórmula para un residuo en un polo simple es:

$$Res\ f(z)_{z=z_0} = b_1 = \lim_{z\to z_0}(z - z_0)f(z)$$

Por otra parte, si $f(z)$ tiene un polo de $m - esimo$ orden en $z = z_0$, entonces el residuo esta dado por

$$Res\ f(z)_{z=z_0} = \frac{1}{(m-1)!}\lim_{z\to z_0}\{\frac{d^{m-1}}{dz^{m-1}}[(z - z_0)^m f(z)]\}$$

Teorema del residuo

Hasta el momento es posible evaluar integrales de funciones analíticas $f(z)$ sobre curvas cerradas C cuando $f(z)$ tiene un solo punto singular dentro de C. A continuación, se verá que el método de integración por residuos puede extenderse al caso en que haya varios puntos singulares de $f(z)$ dentro de C. Esta extensión es sorprendentemente sencilla, como se muestra a continuación.

Sea $f(z)$ una función analítica dentro de una trayectoria simple cerrada C y sobre C, excepto por un número finito de puntos singulares $z_1, z_2, \dots, z_k$ dentro de C. Entonces

$$\oint_c f(z)dz = 2\pi i\sum_{j=1}^{k} Res\ f(z)_{z=z_j}$$

la integral se efectúa en el sentido contrario al movimiento de las manecillas del reloj alrededor de la trayectoria C.

Este teorema importante tiene varias aplicaciones en relación con las integrales complejas y reales.

Contornos y polos

En el análisis complejo, un contorno es una curva cerrada en el plano complejo que se utiliza como trayectoria de integración en una integral de línea. La visualización de estos contornos es fundamental para comprender la aplicación de teoremas como el de Cauchy y el teorema de los residuos. Usualmente, los contornos se representan gráficamente como círculos o curvas simples cerradas que rodean una región del plano.

Uno de los contornos más comunes es el círculo centrado en el origen, descrito por $|z| = r$ que puede ser recorrido en sentido antihorario (orientación positiva) u horario (orientación negativa). La dirección del recorrido afecta directamente el signo del valor de la integral.

La relación entre la forma y posición del contorno y la ubicación de los polos (singularidades aisladas de una función) es esencial en el cálculo de integrales complejas. Si todos los polos de una función están fuera del contorno, y la función es analítica dentro y sobre él, la integral sobre ese contorno es cero, según el teorema de Cauchy. Sin embargo, si el contorno encierra uno o más polos, la integral no es nula y su valor puede determinarse con el teorema de los residuos, que establece que:

$$\oint_c f(z)dz = 2\pi i \sum_{j=1}^{k} Res\, f(z)_{z=z_j}$$

Por lo tanto, al visualizar un contorno, no solo importa su forma geométrica, sino también qué puntos del plano complejo encierra, ya que la presencia o ausencia de singularidades en su interior determina el comportamiento de la integral.

Implementación del código

Bibliotecas utilizadas

El desarrollo de esta calculadora de residuos y contornos se fundamenta en tres librerías principales de Python, cada una con un rol específico en el proceso de cálculo simbólico, numérico y visualización:

- SymPy: Cálculo Simbólico.

    a. Función principal: Realiza operaciones algebraicas y análisis complejo de forma exacta (no aproximada).
    b. Aplicación en el proyecto:
        - Cálculo preciso de residuos (sympy.residue).
        - Identificación de singularidades (sympy.singularities).
        - Manipulación simbólica de funciones racionales y trascendentes.
        - Simplificación de expresiones complejas.

Ventaja clave: Proporciona resultados exactos (ej. π en lugar de 3.1416), esencial para demostraciones matemáticas rigurosas.

- NumPy: Operaciones Numéricas.
    a. Función principal: Soporta cálculos numéricos eficientes con arreglos multidimensionales.
    b. Aplicación en el proyecto:
        - Generación de contornos (ej. círculos mediante numpy.linspace y numpy.exp(1j*t)).
        - Evaluación numérica de funciones complejas en puntos discretos.
        - Cálculo de integrales reales mediante aproximaciones (verificación de resultados simbólicos).

Ventaja clave: Acelera operaciones vectorizadas y permite manejar grandes conjuntos de datos para visualización.

- Matplotlib: Visualización.
    a. Función principal: Generación de gráficos 2D interactivos y estáticos.
    b. Aplicación en el proyecto:
        - Representación de contornos en el plano z (ej. círculo s, semicírculos).
        - Mapeo de la transformación $w = f(z)$ en el plano complejo.
        - Visualización de polos (con marcadores rojos) y su relación con el contorno.
        - Gráficos de funciones reales para integrales sobre $\mathbb{R}$.

Ventaja clave: Permite superponer múltiples elementos (contornos, polos, etiquetas) en una misma figura, facilitando la interpretación geométrica.

En la integración, estas bibliotecas trabajan en conjunto:

- SymPy obtiene resultados analíticos exactos (residuos, polos).
- NumPy discretiza contornos y evalúa funciones numéricamente.
- Matplotlib traduce los datos en visualizaciones intuitivas.

Estructura del código

Módulo de cálculo general de residuos.

```
entrada = input(f"\nIngrese f(z) ({ejemplo}): ")
try:
    f = syp.sympify(entrada, locals=variables_permitidas)
    print(f"\nFunción aceptada: f(z) = {f}")
except Exception as e:
    print(f"Error: {e}")
    exit()

radio = float(input("\nRadio del contorno circular (ej: 3): "))
t = npy.linspace(0, 2*npy.pi, 500)
z_vals = radio * npy.exp(1j * t)
```

```
try:
    polos = syp.singularities(f, z)
    print("\nPolos encontrados:")
    for p in polos:
        print(f"• z = {p}")

    polos_dentro = [p for p in polos if abs(p) < radio]
    print(f"\nPolos dentro del contorno (|z| < {radio}):")
    for p in polos_dentro:
        print(f"• z = {p}")

    print("\nResiduos correspondientes:")
    suma_residuos = 0
    for p in polos_dentro:
        res = syp.residue(f, z, p)
        suma_residuos += res
        print(f"Residuo en z = {p} → {res}")

    integral = 2*syp.pi*syp.I*suma_residuos
    print(f"\nValor de la integral ∮f(z)dz = {integral}")
```

```
except Exception as e:
    print(f"Error en cálculo: {e}")
    polos = []
    polos_dentro = []

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(npy.real(z_vals), npy.imag(z_vals), 'b-', label=f'Contorno |z|={radio}')
for p in polos:
    color = 'red' if abs(p) < radio else 'gray'
    plt.scatter(syp.re(p), syp.im(p), color=color,
                label=f'Polo en z={p}' if abs(p) < radio else None)
plt.title("Plano z")
plt.xlabel("Re(z)"); plt.ylabel("Im(z)")
plt.grid(True); plt.legend()
plt.gca().set_aspect('equal')

plt.subplot(1, 2, 2)
try:
    f_vals = [complex(f.subs(z, val)) for val in z_vals]
    plt.plot([v.real for v in f_vals], [v.imag for v in f_vals], 'purple', label='f(z)')
    plt.title("Transformación bajo f(z)")
    plt.xlabel("Re(f(z))"); plt.ylabel("Im(f(z))")
    plt.grid(True); plt.legend()
    plt.gca().set_aspect('equal')
except:
    plt.title("No se pudo graficar f(z)")

plt.suptitle(f"Análisis de f(z) = {f}", y=1.02)
plt.tight_layout()
plt.show()
```

- Ingreso y validación de la función:
  a. Usa sympy.sympify para interpretar la entrada del usuario con un diccionario seguro de variables y funciones permitidas.
  b. Detecta errores de sintaxis o funciones no autorizadas.
- Configuración del contorno circular:
  a. Genera puntos en un círculo de radio definido por el usuario mediante numpy.linspace y numpy.exp(1j*t).
- Cálculo de polos y residuos:
  a. Identifica singularidades con sympy.singularities.
  b. Filtra polos dentro del contorno y calcula sus residuos usando sympy.residue.
  c. Estima la integral compleja con el teorema del residuo: $2\pi i \times \Sigma$ residuos.
- Visualización:

a. Plano z: Muestra el contorno y los polos (en rojo si están dentro del contorno).
b. Plano w = f(z): Grafica la transformación de la función sobre el contorno.

Módulo de integrales reales.

```
entrada = input("\nIngrese f(x) como función de z (ej: 1/(z**2 + 1)): ")
try:
    f = syp.sympify(entrada, locals={'z': z})
    print(f"\nFunción aceptada: f(z) = {f}")
except Exception as e:
    print(f"Error: {e}")
    exit()

if not f.is_rational_function(z):
    print("La función no es racional")
    exit()

numerador, denominador = syp.fraction(f)
grado_num = syp.degree(numerador, z)
grado_den = syp.degree(denominador, z)

if grado_den < grado_num + 2:
    print("No cumple condición de grados (den ≥ num + 2)")
    exit()

polos = syp.singularities(f, z)
polos_superiores = [p for p in polos if syp.im(p) > 0]

if not polos_superiores:
    print("No hay polos en el semiplano superior")
    exit()

print("\nPolos en semiplano superior:")
for p in polos_superiores:
    print(f"• z = {p}")
```

```
print("\nResiduos correspondientes:")
suma_residuos = 0
for p in polos_superiores:
    res = syp.residue(f, z, p)
    suma_residuos += res
    print(f"Residuo en z = {p} → {res}")

integral_real = 2*syp.pi*syp.I*suma_residuos
print(f"\nValor de la integral ∫f(x)dx (-∞ a ∞) ≈ {integral_real}")
```

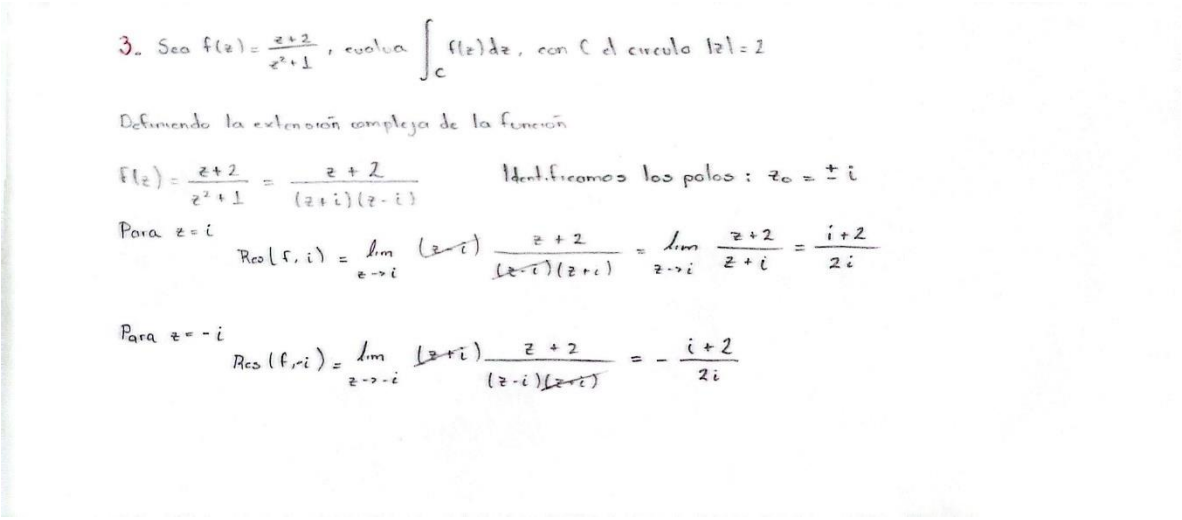
```
R = 10
theta = npy.linspace(0, npy.pi, 100)
semicircle = R * npy.exp(1j * theta)
eje_real = npy.linspace(-R, R, 200)
contorno = npy.concatenate([eje_real, semicircle[::-1]])

plt.subplot(1, 2, 1)
plt.plot(npy.real(contorno), npy.imag(contorno), 'b-', label='Contorno')
for p in polos:
    color = 'red' if syp.im(p) > 0 else 'gray'
    plt.scatter(syp.re(p), syp.im(p), color=color,
                label=f'Polo en z={p}' if syp.im(p) > 0 else None)
plt.title("Contorno de integración (semiplano superior)")
plt.xlabel("Re(z)"); plt.ylabel("Im(z)")
plt.grid(True); plt.legend()
plt.gca().set_aspect('equal')
plt.ylim(-1, R+1)

plt.subplot(1, 2, 2)
x_vals = npy.linspace(-5, 5, 400)
try:
    f_real = syp.lambdify(z, f, 'numpy')
    y_vals = f_real(x_vals)
    plt.plot(x_vals, y_vals, 'purple', label='f(x)')
    plt.fill_between(x_vals, y_vals, alpha=0.3, color='purple')
    plt.title("Función real f(x)")
    plt.xlabel("x"); plt.ylabel("f(x)")
    plt.grid(True); plt.legend()
except:
    plt.title("No se pudo graficar f(x)")

plt.suptitle(f"Integral ∫({entrada})dx de -∞ a ∞ ≈ {integral_real}", y=1.02)
plt.tight_layout()
plt.show()
```

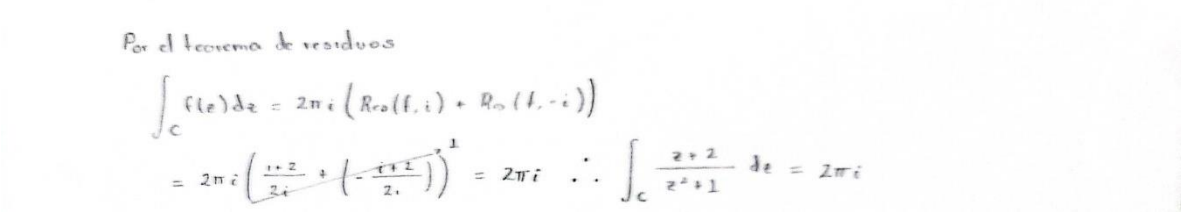
- Validación de la función.
  a. Verifica que sea racional, con denominador de grado al menos 2 unidades mayor que el numerador.
  b. Rechaza funciones con polos reales (para garantizar convergencia).
- Detección de polos en el semiplano superior:
  a. Filtra singularidades con $Im(z) > 0$ usando sympy.singularities
- Cálculo de residuos e integral:
  a. Aplica el método del residuo con un contorno semicircular (eje real + semicírculo en ∞).
  b. La integral real se obtiene como 2πi × Σ residuos en el semiplano superior.
- Visualización:
  a. Contorno de integración: Dibuja el semicírculo y los polos relevantes.
  b. Función real: Muestra $f(x)$ en el eje real con el área bajo la curva.

Resultados

Validación numérica

Residuos calculados.

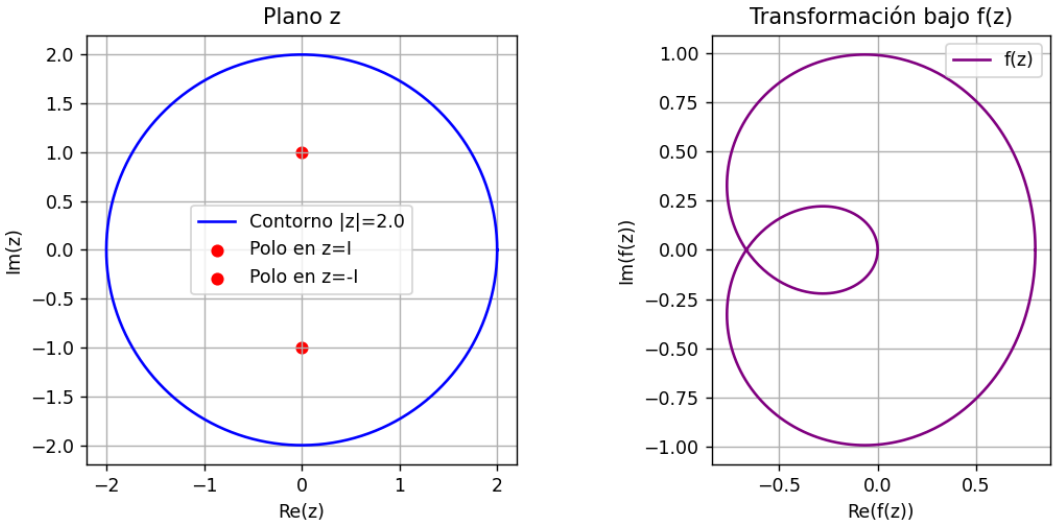




En el programa, colocando la función propuesta en el ejercicio en papel se obtiene:

```
Ingrese f(z) (Ej: (z**2 + 1)/(z - 2)): (z+2)/(z**2 +1)

Función aceptada: f(z) = (z + 2)/(z**2 + 1)

Radio del contorno circular (ej: 3): 2

Polos encontrados:
• z = I
• z = -I

Polos dentro del contorno (|z| < 2.0):
• z = I
• z = -I

Residuos correspondientes:
Residuo en z = I → 1/2 - I
Residuo en z = -I → 1/2 + I

🎯 Valor de la integral ∮f(z)dz = 2*I*pi
```

Donde su representación gráfica es:



Aplicaciones a integrales

Cálculo de la integral impropia

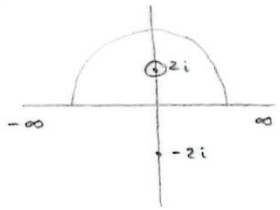
6. Usa el teorema de residuos para calcular $\int_{-\infty}^{\infty} \frac{1}{x^2+4}\,dx$

Definiendo la extensión compleja de la función

$$f(z) = \frac{1}{z^2+4} = \frac{1}{(z+2i)(z-2i)}$$

Identificación de polos $z_0 = \pm 2i$

Eleccion de un contorno



Obtencion del residuo

Para $z = 2i$

$$\text{Res}(f, 2i) = \lim_{z \to 2i} \cancel{(z-2i)} \frac{1}{(z+2i)\cancel{(z-2i)}}$$

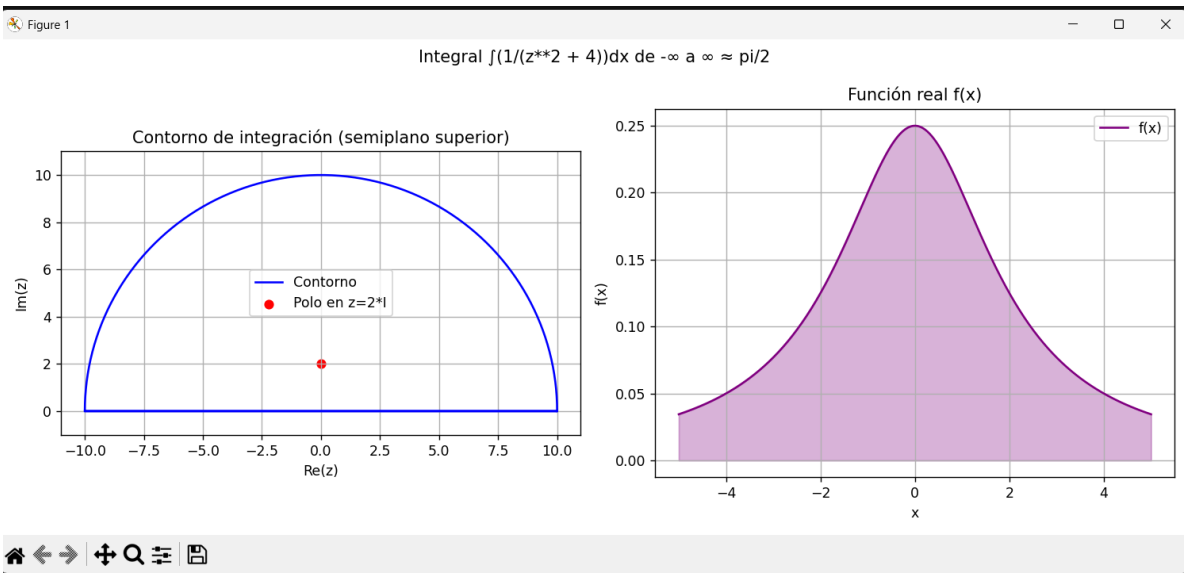
$$= \lim_{z \to 2i} \frac{1}{z+2i} = \frac{1}{4i}$$

Aplicando el teorema del residuo:

$$\oint_C f(z)\,dz = 2\pi i \left(\frac{1}{4i}\right) = \frac{\pi}{2}$$

Capturas del sistema:

```
Calculadora de residuos de funciones racionales y trascendentales.

1. Calculo de residuo.

2. Aplicaciones a integrales reales.

Seleccione una opción (1/2): 2

Requisitos para la función f(z):
1. f(z) debe ser racional (cociente de polinomios)
2. El denominador no debe tener raíces reales
3. Grado(denominador) ≥ Grado(numerador) + 2

Ingrese f(x) como función de z (ej: 1/(z**2 + 1)): 1/(z**2 + 4)

Función aceptada: f(z) = 1/(z**2 + 4)

Polos en semiplano superior:
• z = 2*I

Residuos correspondientes:
Residuo en z = 2*I → -I/4

Valor de la integral ∫f(x)dx (-∞ a ∞) ≈ pi/2
```



Conclusiones

Fue posible adquirir nuevos conocimientos prácticos en programación, así como solidificar los temas vistos a lo largo del curso; específicamente se logra el cálculo de polos y residuos de una función compleja (válida) que se ingrese, así como la representación visual de contornos en el plano complejo. A través de herramientas bastante poderosas en el lenguaje pyhton, como las bibliotecas Sympy, Matplotlib, y Numpy fue posible lograrlo y crear una representación gráfica de las funciones. Así se obtuvo una automatización exacta del proceso, permitiendo la reducción significativa de errores y aceleración de los cálculos.

Tuvimos, además una comprensión más intuitiva del comportamiento de las funciones complejas mediante representaciones gráficas interactivas. La herramienta facilitó la evaluación de integrales reales complejas, incluso en casos donde los métodos clásicos fallan, utilizando el teorema del residuo.

Además de cumplir con los objetivos propuestos, la calculadora se podría usar como un recurso valioso en entornos tanto académicos como profesionales. Su utilidad radica también en fomentar el aprendizaje activo al poder experimentar con diferentes funciones, contornos y configuraciones, fortaleciendo así la conexión entre el álgebra simbólica, la teoría matemática y la interpretación geométrica.

Referencias

Kreyszig, E. (1999). Matemáticas avanzadas para ingeniería. Volumen I y II (3.ª ed.). México: Limusa Wiley.

The SymPy Development Team. (s.f.). SymPy documentation. SymPy. https://docs.sympy.org/latest/index.html

The Matplotlib Development Team. (s.f.). Matplotlib: Visualization with Python. https://matplotlib.org/stable/index.html

NumPy Developers. (s.f.). NumPy documentation. https://numpy.org/doc/

W3Schools. (s.f.). Python Matplotlib Pyplot. https://www.w3schools.com/python/matplotlib_pyplot.asp