Practica 8: Ejercicios de integración numérica

Matematicas por Computadora

Alumno: Itzel Estrada Arcos (368980)
Docente: Ing. Jesus Padron

Viernes 4 de abril 2025

0.1 Introduction.

El método de cálculo de Romberg es un método estadístico que se aplica a la ley trapezoidal en un proceso conocido como extrapolación de Richardson. La regla trapezoidal compuesta se utiliza para construir la tabla, que se construye a partir de una secuencia mayor que el promedio, que se puede utilizar para calcular una integral única.
Este enfoque fue eficaz, se basó en cálculos previos y logró una rápida convergencia sin aumentar significativamente el número de características observadas. Esta sección del libro explica cómo funciona este cuadro y proporciona ejemplos que ilustran las ventajas de otros métodos más simples.

0.2 Desarrollo de la practica.

0.2.1 Proceso paso a paso.

Para la importación de módulos.
Cargué dos bibliotecas: `numpy`, para trabajar con arreglos numéricos y operaciones matemáticas eficientes, y `math`, que ofrece funciones matemáticas como `sin, cos, pi`, etc.

Para definición de la función `f`.
Esta función evalúa una expresión matemática que puedo ingresar como texto. Usé `eval` para interpretar ese texto como código Python, permitiendo usar variables como `x, y` funciones de math o numpy.

Para la evaluación de límites.
Hay otra función que también usa `eval`, pero esta vez para convertir los límites de integración escritos como texto (por ejemplo, `"math.pi/4"`) en valores numéricos reales.

Para la función principal de integración de Romberg.
Calculé la integral definida de una función utilizando el método de Romberg, que es una técnica numérica avanzada basada en la regla del trapecio y la extrapolación de Richardson.
Lo que hace el codigo:

Crea una matriz donde se almacenan los valores calculados en cada paso del método.
Inicia con una primera aproximación usando la regla del trapecio simple.
En cada iteración, se subdivide el intervalo a la mitad, se aplica la regla del trapecio compuesto, y se mejora la precisión con extrapolación.
En cada paso se imprimen los valores intermedios para observar el progreso del

cálculo.
Si se especifica una tolerancia, se revisa si la diferencia entre aproximaciones sucesivas es menor que esta, y si es así, se detiene el proceso y devuelve el resultado.

Para la función `main`.

En esta parte me encargé de interactuar con lo que llegaban a pedir los ejercicios. como:

La función a integrar.
Los límites de integración (que pueden ser expresiones como `math.pi/2`).
El número máximo de iteraciones.
Una tolerancia opcional para verificar convergencia.

Por ultimo, para ejecutar el programa.
Finalmente, cree una verificación que asegura que si el archivo se ejecuta directamente, se llame a la función `main`. Esto me permite que el programa se inicie correctamente cuando se ejecuta desde la terminal o un entorno de desarrollo.

0.3 Conclusion.

El método de Romberg me permitió calcular integrales de forma precisa y eficiente, combinando la regla del trapecio con la extrapolación de Richardson. La implementación en Python, apoyada en `numpy` y `math`, me facilitó la evaluación de funciones y límites, logrando una rápida convergencia con pocos recursos. Fue una forma práctica y clara de aplicar métodos numéricos al cálculo integral.

0.4 Resultados

Ejercicio 1

```
C:\Users\PITEL\PycharmProjects\Practica8\.venv\Scripts\python.exe C:
=== INTEGRACIÓN DE ROMBERG ===
Ingresa la función f(x) (ej: 'x**2 * math.sin(x)'): x**2*math.log(x)
Límite inferior (a, ej: '0' o 'math.pi/4'): 1
Límite superior (b, ej: '1.5' o 'math.pi/2'): 1.5
Máximo de iteraciones (n): 3
Tolerancia (opcional, ej: 1e-6): 1e-6

=== Iteración 0 ===
R[0, 0] = 0.2280741233

=== Iteración 1 ===
R[1,0] = 0.2012025114
R[1,1] = 0.1922453074

=== Iteración 2 ===
R[2,0] = 0.1944944732
R[2,1] = 0.1922584604
R[2,2] = 0.1922593373

=== Iteración 3 ===
R[3,0] = 0.1928180943
R[3,1] = 0.1922593013
R[3,2] = 0.1922593574
R[3,3] = 0.1922593577

¡Convergencia alcanzada en iteración 3!
Resultado final: R[3,3] = 0.192259357707
```

Figure 1: Eje. 1, int. a

```
C:\Users\PITEL\PycharmProjects\Practica8\.venv\Scripts\python.exe C:\
=== INTEGRACIÓN DE ROMBERG ===
Ingresa la función f(x) (ej: 'x**2 * math.sin(x)'): x**2*math.exp(-x)
Límite inferior (a, ej: '0' o 'math.pi/4'): 0
Límite superior (b, ej: '1.5' o 'math.pi/2'): 1
Máximo de iteraciones (n): 3
Tolerancia (opcional, ej: 1e-6): 1e-6

=== Iteración 0 ===
R[0, 0] = 0.1839397206

=== Iteración 1 ===
R[1,0] = 0.1677861928
R[1,1] = 0.1624016835

=== Iteración 2 ===
R[2,0] = 0.1624884051
R[2,1] = 0.1607224759
R[2,2] = 0.1606105287

=== Iteración 3 ===
R[3,0] = 0.1610798961
R[3,1] = 0.1606103931
R[3,2] = 0.1606029209
R[3,3] = 0.1606028001

Resultado final (sin convergencia): R[3,3] = 0.160602800130
```

Figure 2: Eje. 1, int. b

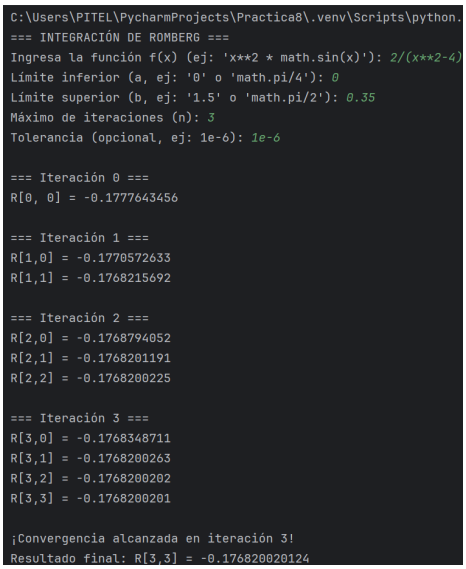


Figure 3: Eje. 1, int. c

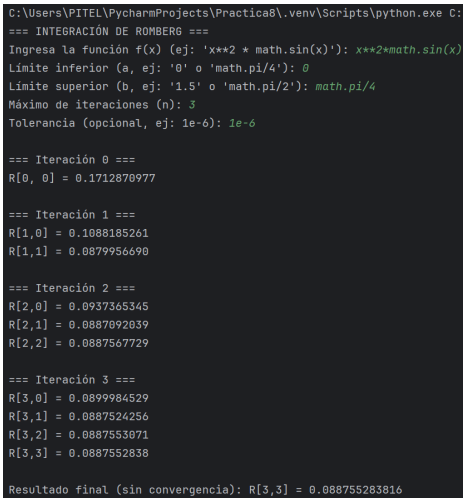


Figure 4: Eje. 1, int. d

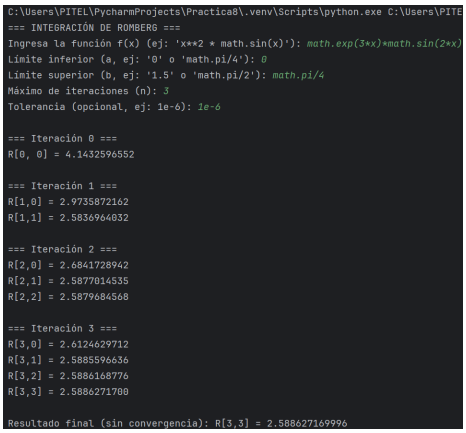


Figure 5: Eje. 1, int. e

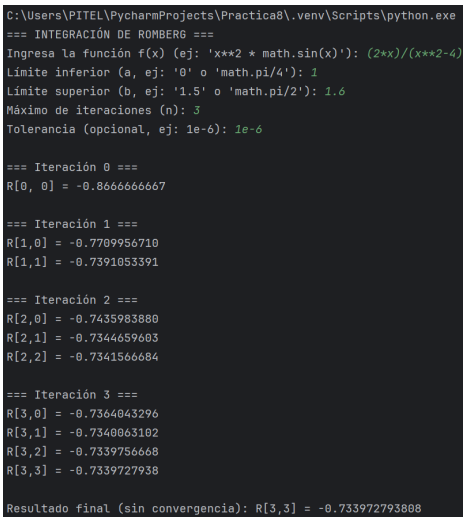


Figure 6: Eje. 1, int. f

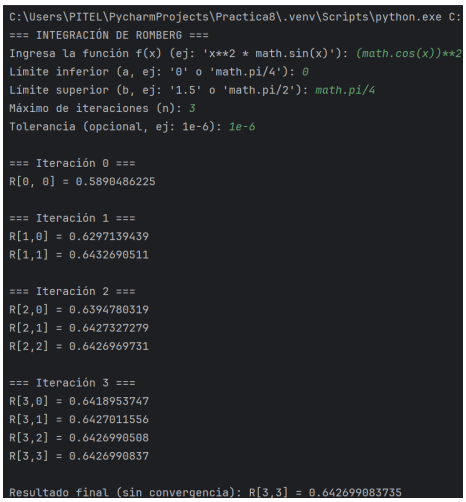


Figure 8: Eje. 1, int. h

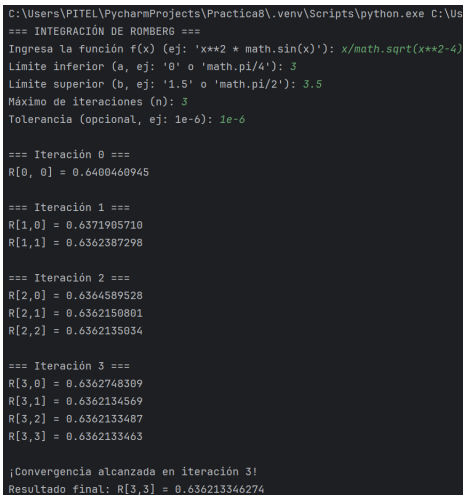


Figure 7: Eje. 1, int. g

Ejercicio 2

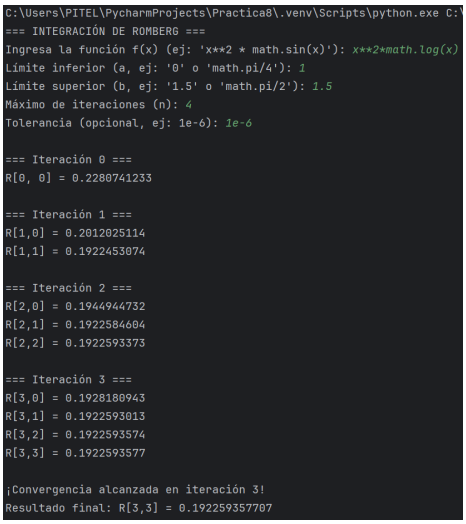


Figure 9: Eje. 2, int. a

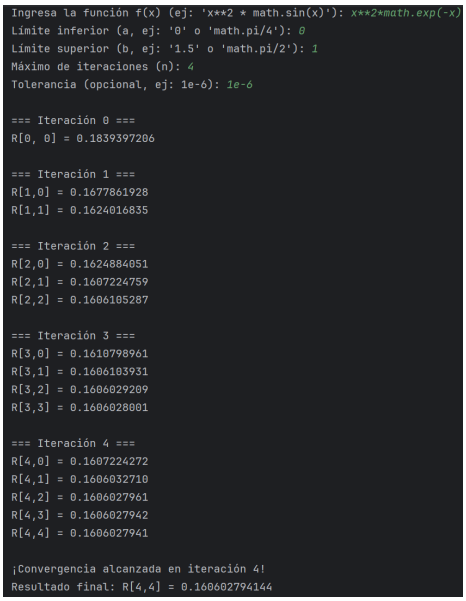


Figure 10: Eje. 2, int. b

```
C:\Users\PITEL\PycharmProjects\Practica8\.venv\Scripts\python.
=== INTEGRACIÓN DE ROMBERG ===
Ingresa la función f(x) (ej: 'x**2 * math.sin(x)'): 2/(x**2-4)
Límite inferior (a, ej: '0' o 'math.pi/4'): 0
Límite superior (b, ej: '1.5' o 'math.pi/2'): 0.35
Máximo de iteraciones (n): 4
Tolerancia (opcional, ej: 1e-6): 1e-6

=== Iteración 0 ===
R[0, 0] = -0.1777643456

=== Iteración 1 ===
R[1,0] = -0.1770572633
R[1,1] = -0.1768215692

=== Iteración 2 ===
R[2,0] = -0.1768794052
R[2,1] = -0.1768201191
R[2,2] = -0.1768200225

=== Iteración 3 ===
R[3,0] = -0.1768348711
R[3,1] = -0.1768200263
R[3,2] = -0.1768200202
R[3,3] = -0.1768200201

¡Convergencia alcanzada en iteración 3!
Resultado final: R[3,3] = -0.176820020124
```

Figure 11: Eje. 2, int. c

```
Ingresa la función f(x) (ej: 'x**2 * math.sin(x)'): x**2*math.sin(x)
Límite inferior (a, ej: '0' o 'math.pi/4'): 0
Límite superior (b, ej: '1.5' o 'math.pi/2'): math.pi/4
Máximo de iteraciones (n): 4
Tolerancia (opcional, ej: 1e-6): 1e-6

=== Iteración 0 ===
R[0, 0] = 0.1712870977

=== Iteración 1 ===
R[1,0] = 0.1088185261
R[1,1] = 0.0879956690

=== Iteración 2 ===
R[2,0] = 0.0937365345
R[2,1] = 0.0887092039
R[2,2] = 0.0887567729

=== Iteración 3 ===
R[3,0] = 0.0899984529
R[3,1] = 0.0887524256
R[3,2] = 0.0887553071
R[3,3] = 0.0887552838

=== Iteración 4 ===
R[4,0] = 0.0890659428
R[4,1] = 0.0887551061
R[4,2] = 0.0887552848
R[4,3] = 0.0887552844
R[4,4] = 0.0887552844

¡Convergencia alcanzada en iteración 4!
Resultado final: R[4,4] = 0.088755284435
```

Figure 12: Eje. 2, int. d

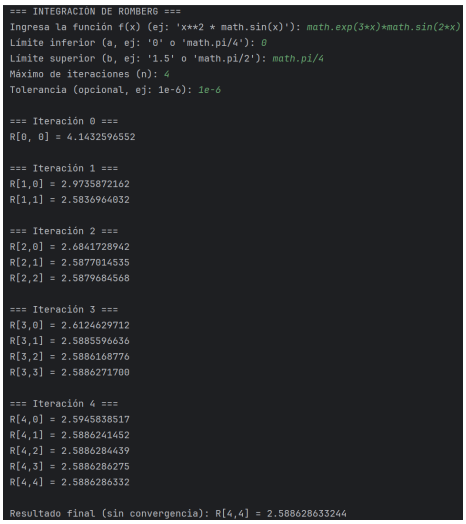


Figure 13: Eje. 2, int. e

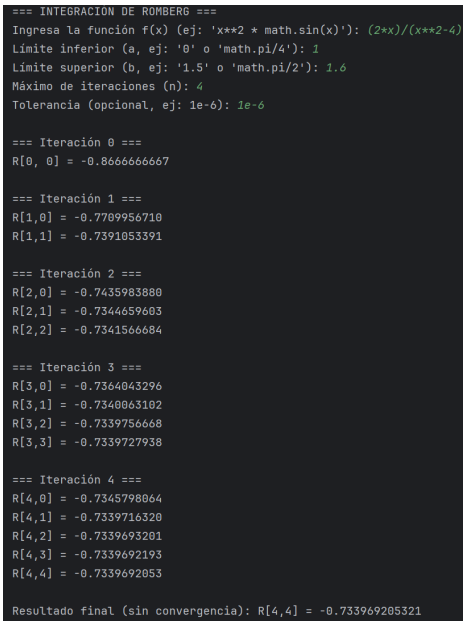


Figure 14: Eje. 2, int. f

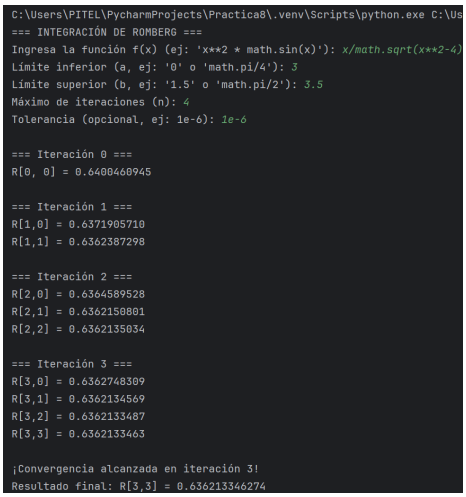


Figure 15: Eje. 2, int. g

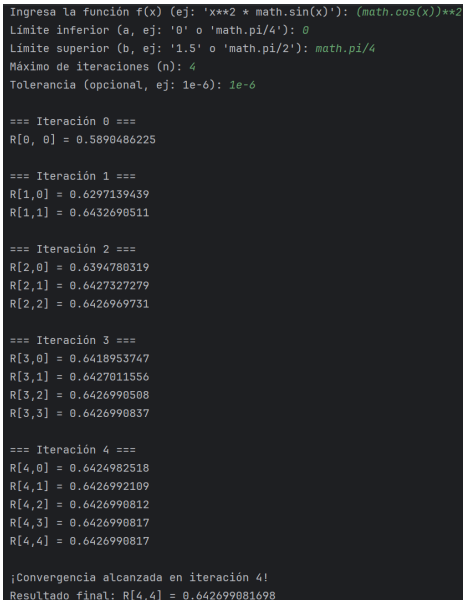


Figure 16: Eje. 2, int. h

Comprobación en Wolfram

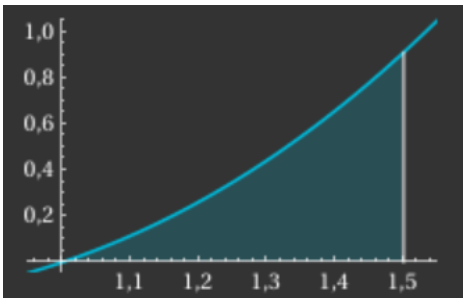


Figure 17: Integral a

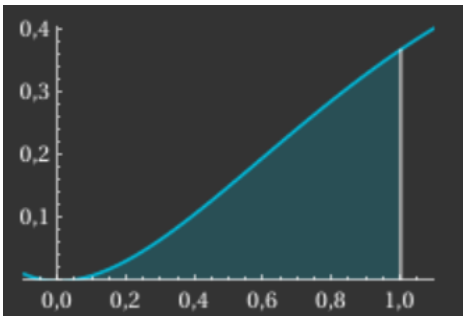


Figure 18: Integral b

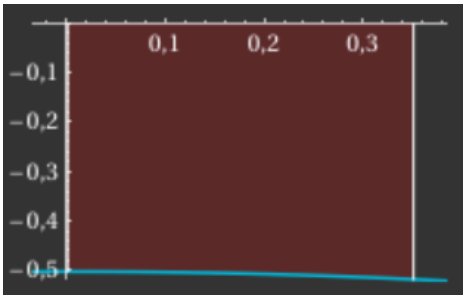


Figure 19: Integral c

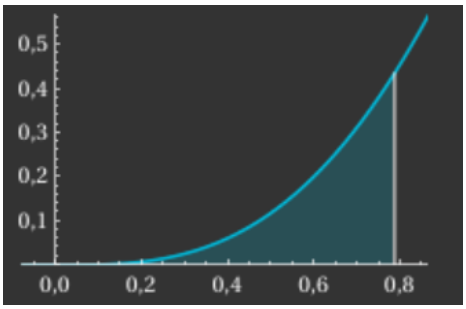


Figure 20: Integral d

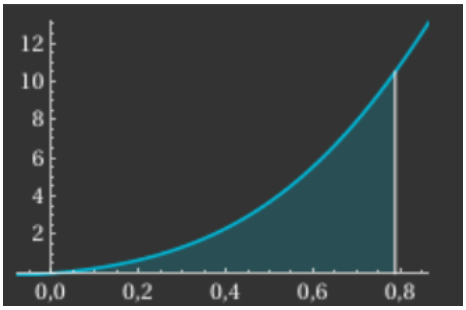


Figure 21: Integral e

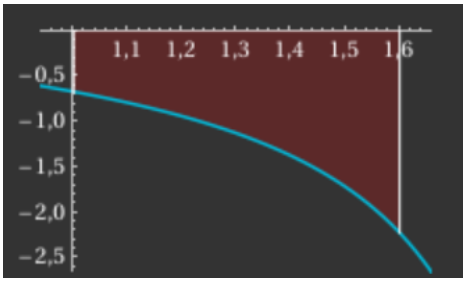


Figure 22: Integral f

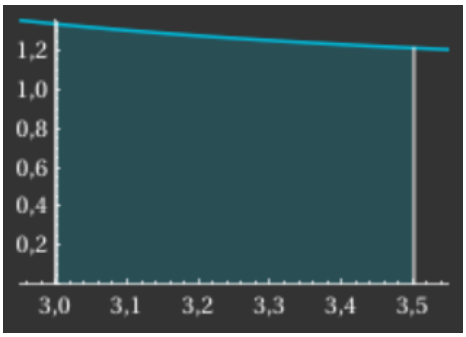


Figure 23: Integral g

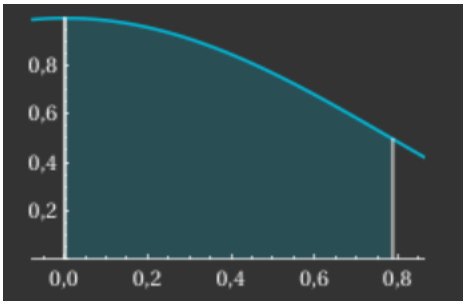


Figure 24: Integral h

Ejercicio 4

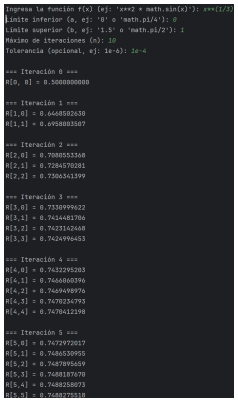


Figure 25: Eje. 4, int. a, parte 1

```
=== Iteración 6 ===
R[6,0] = 0.7489234104
R[6,1] = 0.7494654799
R[6,2] = 0.7495196389
R[6,3] = 0.7495312274
R[6,4] = 0.7495340213
R[6,5] = 0.7495347136
R[6,6] = 0.7495348863

=== Iteración 7 ===
R[7,0] = 0.7495717592
R[7,1] = 0.7497878755
R[7,2] = 0.7498093686
R[7,3] = 0.7498139675
R[7,4] = 0.7498150762
R[7,5] = 0.7498153510
R[7,6] = 0.7498154195
R[7,7] = 0.7498154366

=== Iteración 8 ===
R[8,0] = 0.7498298036
R[8,1] = 0.7499158183
R[8,2] = 0.7499243479
R[8,3] = 0.7499261729
R[8,4] = 0.7499266130
R[8,5] = 0.7499267220
R[8,6] = 0.7499267492
R[8,7] = 0.7499267560
R[8,8] = 0.7499267577

=== Iteración 9 ===
R[9,0] = 0.7499323953
R[9,1] = 0.7499665925
R[9,2] = 0.7499699774
R[9,3] = 0.7499707017
R[9,4] = 0.7499708763
R[9,5] = 0.7499709196
R[9,6] = 0.7499709304
```

Figure 26: Eje. 4, int. a, parte 2

```
R[9,7] = 0.7499709331
R[9,8] = 0.7499709338
R[9,9] = 0.7499709339

¡Convergencia alcanzada en iteración 9!
Resultado final: R[9,9] = 0.749970933933
```

Figure 27: Eje. 4, int. a, parte 3

Comprobación en Wolfram

```
=== INTEGRACIÓN DE ROMBERG ===
Ingresa la función f(x) (ej: 'x**2 * math.sin(x)'): 1.089 + 0.15*(x - 0.2) + 0.9*(x - 0.2)**2 + 2*(x - 0.2)**3
Límite inferior (a, ej: '0' o 'math.pi/4'): 0.2
Límite superior (b, ej: '1.5' o 'math.pi/2'): 0.3
Máximo de iteraciones (n): 10
Tolerancia (opcional, ej: 1e-6): 1E-4

=== Iteración 0 ===
R[0, 0] = 0.1022000000

=== Iteración 1 ===
R[1,0] = 0.1020500000
R[1,1] = 0.1020000000

=== Iteración 2 ===
R[2,0] = 0.1020125000
R[2,1] = 0.1020000000
R[2,2] = 0.1020000000

¡Convergencia alcanzada en iteración 2!
Resultado final: R[2,2] = 0.102000000000
```

Figure 28: Eje. 4, int. b

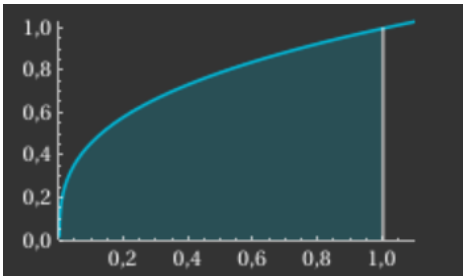


Figure 29: Integral a

Ejercicio 10

```
=== INTEGRACIÓN DE ROMBERG ===
Ingresa la función f(x) (ej: 'x**2 * math.sin(x)'): math.sqrt(1 + (math.cos(x))**2)
Límite inferior (a, ej: '0' o 'math.pi/4'): 0
Límite superior (b, ej: '1.5' o 'math.pi/2'): 48
Máximo de iteraciones (n): 10
Tolerancia (opcional, ej: 1e-6): 1e-4

=== Iteración 0 ===
R[0, 0] = 62.4373714087

=== Iteración 1 ===
R[1,0] = 57.2885616155
R[1,1] = 55.5722916871

=== Iteración 2 ===
R[2,0] = 56.4437506762
R[2,1] = 56.1621470297
R[2,2] = 56.2014707192

=== Iteración 3 ===
R[3,0] = 56.2630546521
R[3,1] = 56.2028226440
R[3,2] = 56.2055343516
R[3,3] = 56.2055988537

=== Iteración 4 ===
R[4,0] = 56.2187726653
R[4,1] = 56.2040120030
R[4,2] = 56.2040912936
R[4,3] = 56.2040683879
R[4,4] = 56.2040623861
```

Figure 30: Eje. 10 parte 1

```
=== Iteración 5 ===
R[5,0] = 58.3626837069
R[5,1] = 59.0773207208
R[5,2] = 59.2688746353
R[5,3] = 59.3175219899
R[5,4] = 59.3297316119
R[5,5] = 59.3327870071

=== Iteración 6 ===
R[6,0] = 58.4504426466
R[6,1] = 58.4796956265
R[6,2] = 58.4398539535
R[6,3] = 58.4266948951
R[6,4] = 58.4232014555
R[6,5] = 58.4223153068
R[6,6] = 58.4220929693

=== Iteración 7 ===
R[7,0] = 58.4655692832
R[7,1] = 58.4706114954
R[7,2] = 58.4700058866
R[7,3] = 58.4704844887
R[7,4] = 58.4706562126
R[7,5] = 58.4707026005
R[7,6] = 58.4707144167
R[7,7] = 58.4707173845
```

Figure 31: Eje. 10 parte 2

```
=== Iteración 8 ===
R[8,0] = 58.4692528251
R[8,1] = 58.4704806724
R[8,2] = 58.4704719509
R[8,3] = 58.4704793487
R[8,4] = 58.4704793286
R[8,5] = 58.4704791557
R[8,6] = 58.4704791011
R[8,7] = 58.4704790867
R[8,8] = 58.4704790831

=== Iteración 9 ===
R[9,0] = 58.4701656094
R[9,1] = 58.4704698709
R[9,2] = 58.4704691508
R[9,3] = 58.4704691063
R[9,4] = 58.4704690662
R[9,5] = 58.4704690562
R[9,6] = 58.4704690537
R[9,7] = 58.4704690531
R[9,8] = 58.4704690529
R[9,9] = 58.4704690529

¡Convergencia alcanzada en iteración 9!
Resultado final: R[9,9] = 58.470469052880
```

Figure 32: Eje. 10 parte 3

Comprobación en Wolfram

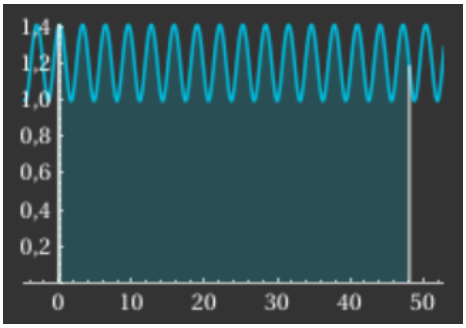


Figure 33: Integral

Inciso a (Iteraciones de R1,1 a R5,1)
Bansandome en los resultados que me proposiono el codigo de las interaciones del inciso a, declare que la aproximacion estimada es R3,1=56.443751

Inciso b (Iteraciones de R2,2 a R5,5)
Obtuve un cambio en la estabilidad de mis iteraciones, por lo tanto escogí la predominante, que fue R3,3=56.205535.

Inciso c (Iteraciones de R6,1 a R6,6)

Encontre que en la aproximacion pasada era muy inestable, por lo que esta parte me trajo mayor estabilidad en la parte R6,6=58.22093.

Inciso d (Iteraciones de R7,7 a R10,10)
Se continuo con la estabilidad, lo que me declaro mi aproximacion final R9,9=58.470469.

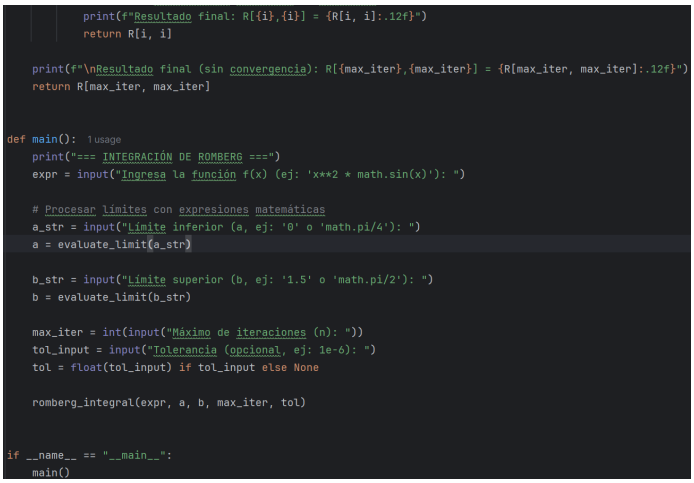
Inciso e (Explicacion de detalles)
La integracion de Romberg tuvo problemas con esta integral en especifico porque la funcion dentro de la raiz es oscilatoria. Esto significa que cambia constantemente de valor, lo que hace que la regla del trapecio, en la que se basa Romberg, no la pueda aproximar detalladamente con pocos puntos.
Como resultado, algunos valores en la tabla de Romberg fueron inestables, porque la extrapolación no pudo corregir los errores que se generaban en las primeras aproximaciones.

0.5 Anexos

Rombert.py

```python
import numpy as np
import math


def f(x, expr):  3 usages
    return eval(expr, {'np': np, 'math': math, 'x': x})


def evaluate_limit(limit_str):  2 usages
    """Evalúa expresiones como 'math.pi/4' en los límites."""
    return eval(limit_str, {'math': math, 'np': np})


def romberg_integral(expr, a, b, max_iter=10, tol=None):  1 usage
    R = np.zeros((max_iter + 1, max_iter + 1))
    h = b - a

    # Iteración inicial (trapecio simple)
    R[0, 0] = (h / 2) * (f(a, expr) + f(b, expr))
    print(f"\n=== Iteración 0 ===")
    print(f"R[0, 0] = {R[0, 0]:.10f}")

    for i in range(1, max_iter + 1):
        h /= 2
        # Regla del trapecio compuesto
        sumatoria = sum(f(a + (2 * k - 1) * h, expr) for k in range(1, 2 ** (i - 1) + 1))
        R[i, 0] = 0.5 * R[i - 1, 0] + h * sumatoria

        # Extrapolación de Richardson
        for j in range(1, i + 1):
            R[i, j] = R[i, j - 1] + (R[i, j - 1] - R[i - 1, j - 1]) / (4 ** j - 1)

        # Mostrar resultados
        print(f"\n=== Iteración {i} ===")
        for j in range(i + 1):
            print(f"R[{i},{j}] = {R[i, j]:.10f}")

        # Verificar convergencia
        if tol and abs(R[i, i] - R[i - 1, i - 1]) < tol:
            print(f"\n¡Convergencia alcanzada en iteración {i}!")
            print(f"Resultado final: R[{i},{i}] = {R[i, i]:.12f}")
```

Figure 34: Codigo parte 1



```
            print(f"Resultado final: R[{i},{i}] = {R[i, i]:.12f}")
            return R[i, i]

    print(f"\nResultado final (sin convergencia): R[{max_iter},{max_iter}] = {R[max_iter, max_iter]:.12f}")
    return R[max_iter, max_iter]


def main():  1 usage
    print("=== INTEGRACIÓN DE ROMBERG ===")
    expr = input("Ingresa la función f(x) (ej: 'x**2 * math.sin(x)'): ")

    # Procesar límites con expresiones matemáticas
    a_str = input("Límite inferior (a, ej: '0' o 'math.pi/4'): ")
    a = evaluate_limit(a_str)

    b_str = input("Límite superior (b, ej: '1.5' o 'math.pi/2'): ")
    b = evaluate_limit(b_str)

    max_iter = int(input("Máximo de iteraciones (n): "))
    tol_input = input("Tolerancia (opcional, ej: 1e-6): ")
    tol = float(tol_input) if tol_input else None

    romberg_integral(expr, a, b, max_iter, tol)


if __name__ == "__main__":
    main()
```

Figure 35: Codigo parte 2