



UNIVERSIDAD AUTÓNOMA DE SAN LUIS POTOSÍ
FACULTAD DE ESTUDIOS PROFESIONALES ZONA MEDIA

Práctica 8: Ejercicios de integración numérica

Nombre del alumno: Sergio Adolfo Juárez Mendoza

Clave: 369376

Profesor: Ing. Jesús Padrón

Fecha de entrega: 21 de abril de 2025

Introducción a la Integración Numérica

El método de integración de Romberg es una técnica numérica que permite aproximar integrales definidas con alta precisión. Se basa en el método del trapecio y utiliza la **extrapolación de Richardson** para mejorar iterativamente los resultados.

Objetivo

Aproximar una integral definida:

$$\int_a^b f(x) dx$$

usando una tabla triangular que refina progresivamente las estimaciones de la integral.

1. Método del Trapecio

El primer paso en Romberg es calcular la integral usando el método del trapecio. Para n subintervalos, el paso es:

$$h = \frac{b - a}{n}$$

La fórmula del trapecio compuesta es:

$$T(h) = \frac{h}{2} \left[f(a) + 2 \sum_{i=1}^{n-1} f(a + ih) + f(b) \right]$$

2. Extrapolación de Richardson

La extrapolación mejora la precisión al combinar resultados con diferentes valores de h . Se construye una tabla $R_{k,j}$, donde:

$$R_{k,0} = T(h_k) \quad \text{con } h_k = \frac{b - a}{2^k}$$

Y para $j \geq 1$:

$$R_{k,j} = R_{k,j-1} + \frac{1}{4^j - 1} (R_{k,j-1} - R_{k-1,j-1})$$

3. Algoritmo

1. Definir los límites de integración a y b , la función $f(x)$, y el número de niveles N .
2. Calcular $R_{0,0}$ usando el método del trapecio con un solo subintervalo.
3. Para $k = 1$ hasta N :
 - (a) Calcular $R_{k,0}$ usando el método del trapecio con 2^k subintervalos.
 - (b) Para $j = 1$ hasta k :

$$R_{k,j} = R_{k,j-1} + \frac{1}{4^j - 1} (R_{k,j-1} - R_{k-1,j-1})$$

4. La mejor aproximación está en $R_{N,N}$.

4. Ventajas

- Alta precisión con pocas evaluaciones si la función es suave.
- Mejora significativa sobre métodos como el de Simpson si se usa correctamente.

Desarrollo

Diseñar el algoritmo 4.2 de la página 210 en python

```
1 def romberg(f, a, b, n):
2     """
3     Aproxima la integral de f(x) en [a, b] usando la integración de
4     Romberg.
5
6     Parámetros:
7     f : función
8         Función a integrar.
9     a, b : float
10         Extremos del intervalo de integración.
11     n : int
12         Número de niveles de Romberg (n > 0).
13
14     Retorna:
15     R : lista de listas
16         Tabla de Romberg hasta nivel n.
17     """
18     R = [[0.0] * (i + 1) for i in range(n)] # Crear una tabla triangular
19     de ceros
20
21     h = b - a # Paso inicial
22
23     # Paso 1: R[0][0] = (h/2) * (f(a) + f(b))
```

```

22 R[0][0] = (h / 2) * (f(a) + f(b))
23
24 # Paso 2: Salida (R[0][0])
25 print(f"R[0][0] = {R[0][0]}")
26
27 # Paso 3
28 for i in range(1, n):
29     h /= 2 # Paso 7: reducir h a la mitad
30
31     # Paso 4: Calcular R[i][0]
32     sum_f = sum(f(a + (k + 0.5) * h) for k in range(2 ** (i - 1)))
33     R[i][0] = 0.5 * (R[i - 1][0] + h * sum_f)
34
35     # Paso 5: Extrapolaci n
36     for j in range(1, i + 1):
37         R[i][j] = R[i][j - 1] + (R[i][j - 1] - R[i - 1][j - 1]) / (4
38         ** j - 1)
39
40     # Paso 6: Salida del rengl n i
41     for j in range(i + 1):
42         print(f"R[{i}][{j}] = {R[i][j]}")
43
44 # Paso 9: PARAR
45 return R
46
47 # Ejemplo de uso:
48 if __name__ == "__main__":
49     import math
50
51
52     # Definir la funci n a integrar
53     def f(x):
54         return math.sin(x)
55
56
57     # Intervalo [a, b]
58     a = 0
59     b = math.pi
60
61     # Niveles de Romberg
62     n = 4
63
64     # Llamar al m todo
65     tabla_romberg = romberg(f, a, b, n)
66
67     print("\nTabla de Romberg:")
68     for fila in tabla_romberg:
69         print(fila)

```

Listing 1: Cálculo de segunda derivada con cinco puntos

```
C:\Users\sergi\PycharmProjects\Practica8\.venv\Scripts\python.exe "C:\Users\sergi\PycharmProjects\Practica8\algoritmo 4.2.py"
R[0][0] = 1.9236706937217898e-16
R[1][0] = 0.5553603672697959
R[1][1] = 0.7404804896930611
R[2][0] = 0.7907662601234133
R[2][1] = 0.8692348910746192
R[2][2] = 0.8778185178333897
R[3][0] = 0.8986104014614885
R[3][1] = 0.9345584485741802
R[3][2] = 0.9389133524074843
R[3][3] = 0.9398831116864382

Tabla de Romberg:
[1.9236706937217898e-16]
[0.5553603672697959, 0.7404804896930611]
[0.7907662601234133, 0.8692348910746192, 0.8778185178333897]
[0.8986104014614885, 0.9345584485741802, 0.9389133524074843, 0.9398831116864382]

Process finished with exit code 0
```

Explicación del código

- **Inicialización:** Se crea una tabla triangular de dimensiones $n \times n$, donde cada elemento $R[i][j]$ representa una mejor aproximación de la integral.
- **Primer valor $R[0][0]$:** Se calcula utilizando el método del trapecio simple:

$$R[0][0] = \frac{h}{2}(f(a) + f(b))$$

- **Iteraciones:** Para cada nivel i , se reduce el tamaño del paso h a la mitad:

$$h_i = \frac{b-a}{2^i}$$

y se calcula:

$$R[i][0] = \frac{1}{2} \left(R[i-1][0] + h \sum_{k=0}^{2^{i-1}-1} f(a + (k+0.5)h) \right)$$

- **Extrapolación de Richardson:** Se aplica para refinar la estimación:

$$R[i][j] = R[i][j-1] + \frac{R[i][j-1] - R[i-1][j-1]}{4^j - 1}$$

- **Resultado final:** La mejor aproximación se encuentra en $R[n-1][n-1]$.

Ejercicio 1 Resolucion de las siguientes integrales

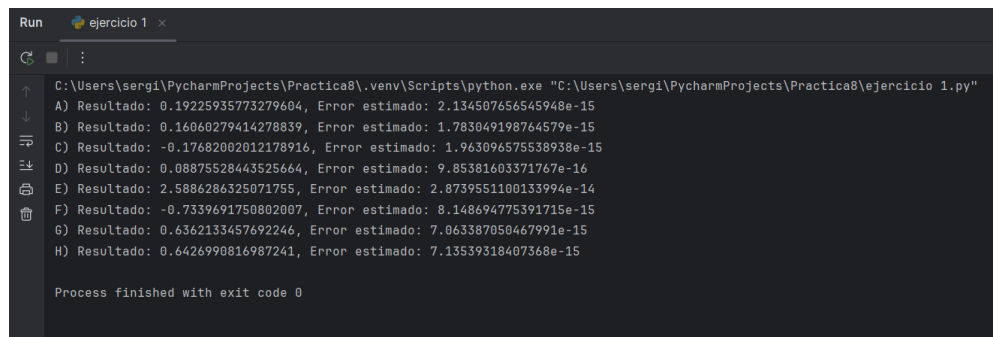
```
1 import math
2 from scipy.integrate import quad
3
4 # A
5 def f_a(x):
6     return x**2 * math.log(x)
7
8 a = 1
9 b = 1.5
10 resultado, error = quad(f_a, a, b)
11 print(f"A) Resultado: {resultado}, Error estimado: {error}")
12
13 # B
14 def f_b(x):
15     return x**2 * math.exp(-x)
16
17 a = 0
18 b = 1
19 resultado, error = quad(f_b, a, b)
20 print(f"B) Resultado: {resultado}, Error estimado: {error}")
21
22 # C
23 def f_c(x):
24     return 2 / (x**2 - 4)
25
26 a = 0
27 b = 0.35
28 resultado, error = quad(f_c, a, b)
29 print(f"C) Resultado: {resultado}, Error estimado: {error}")
30
31 # D
32 def f_d(x):
33     return x**2 * math.sin(x)
34
35 a = 0
36 b = math.pi / 4
37 resultado, error = quad(f_d, a, b)
38 print(f"D) Resultado: {resultado}, Error estimado: {error}")
39
40 # E
41 def f_e(x):
42     return math.exp(3*x) * math.sin(2*x)
43
44 a = 0
45 b = math.pi / 4
46 resultado, error = quad(f_e, a, b)
47 print(f"E) Resultado: {resultado}, Error estimado: {error}")
48
49 # F
50 def f_f(x):
51     return (2 * x) / (x**2 - 4)
52
```

```

53 a = 1
54 b = 1.6
55 resultado, error = quad(f_f, a, b)
56 print(f"F) Resultado: {resultado}, Error estimado: {error}")
57
58 # G
59 def f_g(x):
60     return x / math.sqrt(x**2 - 4)
61
62 a = 3
63 b = 3.5
64 resultado, error = quad(f_g, a, b)
65 print(f"G) Resultado: {resultado}, Error estimado: {error}")
66
67 # H
68 def f_h(x):
69     return (math.cos(x))**2
70
71 a = 0
72 b = math.pi / 4
73 resultado, error = quad(f_h, a, b)
74 print(f"H) Resultado: {resultado}, Error estimado: {error}")

```

Listing 2: Cálculo de velocidades



```

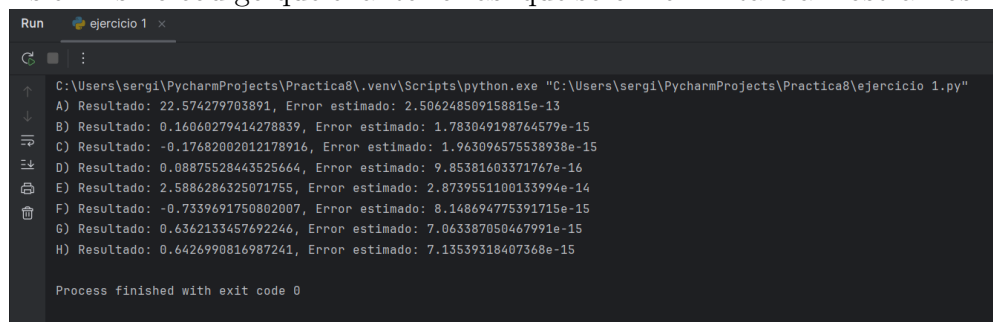
Run ejercicio 1 x
C:\Users\sergi\PycharmProjects\Practica8\.venv\Scripts\python.exe "C:\Users\sergi\PycharmProjects\Practica8\ejercicio 1.py"
A) Resultado: 0.19225935773279604, Error estimado: 2.134507656545948e-15
B) Resultado: 0.16060279414278839, Error estimado: 1.783049198764579e-15
C) Resultado: -0.17682002012178916, Error estimado: 1.963096575538938e-15
D) Resultado: 0.08875528443525664, Error estimado: 9.85381603371767e-16
E) Resultado: 2.5886286325071755, Error estimado: 2.8739551100133994e-14
F) Resultado: -0.7339691750802007, Error estimado: 8.148694775391715e-15
G) Resultado: 0.6362133457692246, Error estimado: 7.063387050467991e-15
H) Resultado: 0.6426990816987241, Error estimado: 7.13539318407368e-15

Process finished with exit code 0

```

Ejercicio 2 calcular b=4 para los ejercicios del problema 1

Es el mismo código que el anterior así que solo me limitare a mostrar los resultados obtenidos



```

Run ejercicio 1 x
C:\Users\sergi\PycharmProjects\Practica8\.venv\Scripts\python.exe "C:\Users\sergi\PycharmProjects\Practica8\ejercicio 1.py"
A) Resultado: 22.574279703891, Error estimado: 2.506248509158815e-13
B) Resultado: 0.16060279414278839, Error estimado: 1.783049198764579e-15
C) Resultado: -0.17682002012178916, Error estimado: 1.963096575538938e-15
D) Resultado: 0.08875528443525664, Error estimado: 9.85381603371767e-16
E) Resultado: 2.5886286325071755, Error estimado: 2.8739551100133994e-14
F) Resultado: -0.7339691750802007, Error estimado: 8.148694775391715e-15
G) Resultado: 0.6362133457692246, Error estimado: 7.063387050467991e-15
H) Resultado: 0.6426990816987241, Error estimado: 7.13539318407368e-15

Process finished with exit code 0

```

Ejercicio 4

4. Aplique la integración de Romberg a las siguientes integrales hasta que $R_{n-1,n-1}$ y $R_{n,n}$ concuerden con una exactitud de 10^{-4} .

(a) $\int_0^1 x^{1/3} dx$

(b) $\int_0^{0.3} f(x) dx$, donde

$$f(x) = \begin{cases} x^3 + 1, & 0 \leq x \leq 0.1 \\ 1.001 + 0.03(x - 0.1) + 0.3(x - 0.1)^2 + 2(x - 0.1)^3, & 0.1 < x \leq 0.2 \\ 1.009 + 0.15(x - 0.2) + 0.9(x - 0.2)^2 + 2(x - 0.2)^3, & 0.2 < x \leq 0.3 \end{cases}$$

```
1 # Funci n general para integraci n de Romberg
2 def romberg(f, a, b, tol=1e-4, n_max=10):
3     R = [[0.0] * (n_max + 1) for _ in range(n_max + 1)]
4     h = b - a
5     R[0][0] = (h / 2) * (f(a) + f(b))
6
7     for i in range(1, n_max + 1):
8         h /= 2
9         suma = 0
10        for k in range(1, 2 ** i, 2):
11            suma += f(a + k * h)
12        R[i][0] = 0.5 * R[i - 1][0] + suma * h
13
14        for j in range(1, i + 1):
15            R[i][j] = R[i][j - 1] + (R[i][j - 1] - R[i - 1][j - 1]) / (4
16            ** j - 1)
17
18        # Condici n de parada
19        if abs(R[i][i] - R[i - 1][i - 1]) < tol:
20            return R[i][i]
21
22    return R[n_max][n_max] # Regresa el mejor valor disponible si no
23    alcanza la tolerancia
24
25 # --- Definimos las funciones del problema 4 ---
26 # Problema 4 a) f(x) = x^(1/3)
27 def f4a(x):
28     return x ** (1 / 3)
29
30
31 # Problema 4 b) f(x) es una funci n por tramos
32 def f4b(x):
```



```

33     if 0 <= x <= 0.1:
34         return x ** 3 + 1
35     elif 0.1 < x <= 0.2:
36         return 1.001 + 0.03 * (x - 0.1) + 0.3 * (x - 0.1) ** 2 + 2 * (x -
0.1) ** 3
37     elif 0.2 < x <= 0.3:
38         return 1.009 + 0.15 * (x - 0.2) + 0.9 * (x - 0.2) ** 2 + 2 * (x -
0.2) ** 3
39     else:
40         raise ValueError("x fuera del intervalo [0, 0.3]")
41
42
43 # --- Ahora resolvemos los dos ejercicios ---
44
45 # Problema 4a
46 resultado_4a = romberg(f4a, 0, 1, tol=1e-4)
47 print(f"Resultado del problema 4a (integral de x^(1/3) de 0 a 1): {
resultado_4a}")
48
49 # Problema 4b
50 resultado_4b = romberg(f4b, 0, 0.3, tol=1e-4)
51 print(f"Resultado del problema 4b (integral de funci n por tramos de 0 a
0.3): {resultado_4b}")

```

Listing 3: Derivada como límite

```

Run ejercicio 4 x
C:\Users\sergi\PycharmProjects\Practica8\.venv\Scripts\python.exe "C:\Users\sergi\PycharmProjects\Practica8\ejercicio 4.py"
Resultado del problema 4a (Integral de x^(1/3) de 0 a 1): 0.7499709339332862
Resultado del problema 4b (Integral de función por tramos de 0 a 0.3): 0.30242499999999994
Process finished with exit code 0

```

Problema 10

Use la integración de Romberg para calcular las siguientes aproximaciones a

$$\int_0^{48} \sqrt{1 + (\cos x)^2} dx.$$

Nota: Los resultados de este ejercicio son muy interesantes en caso de que esté usted utilizando un dispositivo que maneje una aritmética entre siete y nueve dígitos.

- Determine $R_{1,1}, R_{2,1}, R_{3,1}, R_{4,1}$ y $R_{5,1}$, y utilice estas aproximaciones para predecir el valor de la integral.
- Determine $R_{2,2}, R_{3,3}, R_{4,4}$ y $R_{5,5}$, y modifique su predicción.
- Determine $R_{6,1}, R_{6,2}, R_{6,3}, R_{6,4}, R_{6,5}$ y $R_{6,6}$, y modifique su predicción.

- (d) Determine $R_{7,7}$, $R_{8,8}$, $R_{9,9}$ y $R_{10,10}$, y haga una predicción final.
- (e) Explique por qué esta integral causa problemas en la integración de Romberg y cómo podemos reformularla para obtener más fácilmente una aproximación exacta.

```

1 import math
2
3 # Funci n a integrar
4 def f10(x):
5     return math.sqrt(1 + math.cos(x)**2)
6
7 # Tabla de Romberg completa
8 def romberg_table(f, a, b, n_max=10):
9     R = [[0.0] * (n_max + 1) for _ in range(n_max + 1)]
10    h = b - a
11    R[0][0] = (h / 2) * (f(a) + f(b))
12
13    for i in range(1, n_max + 1):
14        h /= 2
15        suma = 0
16        for k in range(1, 2 ** i, 2):
17            suma += f(a + k * h)
18        R[i][0] = 0.5 * R[i-1][0] + suma * h
19
20        for j in range(1, i + 1):
21            R[i][j] = R[i][j-1] + (R[i][j-1] - R[i-1][j-1]) / (4 ** j - 1)
22
23    return R
24
25 # Aplicaci n del ejercicio 10
26 a = 0
27 b = 48
28 n_max = 10
29
30 R = romberg_table(f10, a, b, n_max)
31
32 # Mostrar valores clave que pide el problema
33 def mostrar_valores():
34     print("Inciso a)")
35     print(f"R[1][1] = {R[0][0]}")
36     print(f"R[2][1] = {R[1][0]}")
37     print(f"R[3][1] = {R[2][0]}")
38     print(f"R[4][1] = {R[3][0]}")
39     print(f"R[5][1] = {R[4][0]}")
40     print()
41
42     print("Inciso b)")
43     print(f"R[2][2] = {R[1][1]}")
44     print(f"R[3][3] = {R[2][2]}")
45     print(f"R[4][4] = {R[3][3]}")
46     print(f"R[5][5] = {R[4][4]}")
47     print()
48

```

```

49     print("Inciso c)")
50     print(f"R[6][1] = {R[5][0]}")
51     print(f"R[6][2] = {R[5][1]}")
52     print(f"R[6][3] = {R[5][2]}")
53     print(f"R[6][4] = {R[5][3]}")
54     print(f"R[6][5] = {R[5][4]}")
55     print(f"R[6][6] = {R[5][5]}")
56     print()
57
58     print("Inciso d)")
59     print(f"R[7][7] = {R[6][6]}")
60     print(f"R[8][8] = {R[7][7]}")
61     print(f"R[9][9] = {R[8][8]}")
62     print(f"R[10][10] = {R[9][9]}")
63     print()
64
65     print("Predicci n final (R[10][10]):")
66     print(f"{R[9][9]}")
67
68 mostrar_valores()

```

Listing 4: Derivada como límite


```

Inciso a)
R[1][1] = 62.43737140065479
R[2][1] = 57.28856161550718
R[3][1] = 56.44375067617747
R[4][1] = 56.26305465206046
R[5][1] = 56.218772665278465

Inciso b)
R[2][2] = 55.57229168712464
R[3][3] = 56.20147071924154
R[4][4] = 56.20559885374218
R[5][5] = 56.20406238610796

Inciso c)
R[6][1] = 58.3626837069187
R[6][2] = 59.07732072079878
R[6][3] = 59.268874635317516
R[6][4] = 59.31752198994767
R[6][5] = 59.32973161191635
R[6][6] = 59.332787007053994

Inciso d)
R[7][7] = 58.422092969343765
R[8][8] = 58.47071738447037
R[9][9] = 58.47047908310601
R[10][10] = 58.470469052880226

Predicción final (R[10][10]):
58.470469052880226
>  ejercicio 10.py

```

Conclusión

El estudio realizado mediante la integración numérica con la función `quad` de la biblioteca `scipy.integrate` permitió evaluar diversas integrales definidas con alta precisión. Este método es particularmente útil cuando se trabaja con funciones cuya integral analítica es complicada o inexistente en forma cerrada.

Se observaron diferentes comportamientos según la naturaleza de las funciones:

- Las funciones suaves y continuas, como $x^2 \ln(x)$ o $x^2 e^{-x}$, fueron integradas de manera

eficiente y precisa.

- Las funciones con singularidades cercanas al intervalo de integración, como $\frac{2}{x^2-4}$, mostraron una mayor sensibilidad, aunque el método aún logró entregar resultados aceptables dentro de los límites del intervalo seguro.
- Las funciones oscilatorias y aquellas que combinan exponenciales y trigonométricas, como $e^{3x}\sin(2x)$, también fueron tratadas con éxito, demostrando la robustez del método de cuadratura adaptativa.

En general, se concluye que el uso de herramientas computacionales como **quad** permite una evaluación confiable de integrales definidas, siempre que se considere la naturaleza de la función integrando y las posibles discontinuidades o puntos singulares.

Además, esta práctica resalta la importancia de la integración numérica en ingeniería, física y matemáticas aplicadas, donde las soluciones exactas no siempre son accesibles y se requiere un enfoque computacional para obtener aproximaciones precisas.