Instituto Tecnológico Superior del Oriente del Estado de Hidalgo

ITESA

Ingeniería en Sistemas Computacionales

Métodos Numéricos

Problemario

MTI. Efrén Rolando Romero León

Integrantes:

América Hernández Mendoza

Aneth Reyes Herrera

Jessica Jiuhtzal Muñoz Muñoz

Carlos Martinez Gonzalez

04 de Febrero de 2025

INTRODUCCIÓN

En este problemario veremos los errores más comunes en los métodos numéricos mediante la resolución de diversos ejercicios prácticos que resolvimos en el lenguaje de Python . Estos errores incluyen el error de redondeo, error absoluto, error relativo, error de truncamiento y exactitud, todos ellos fundamentales en el estudio de cálculos numéricos.

Para comprender la importancia de estos errores, resolvimos problemas aplicados a situaciones reales. Calculamos áreas, velocidades y volúmenes utilizando valores exactos y aproximados, comparando los resultados para identificar el impacto de cada tipo de error.

A lo largo de estos ejercicios, evaluamos la exactitud de los cálculos, determinando la magnitud de los errores mediante las fórmulas de error absoluto y relativo. Esto nos permitió comprender cómo minimizar los errores y mejorar la fiabilidad de los resultados en aplicaciones numéricas.

Finalmente nos ayudó a visualizar la importancia de los métodos numéricos en la vida cotidiana y en diversas áreas de la ingeniería, donde incluso pequeñas variaciones pueden generar grandes impactos en los cálculos y decisiones.

PROBLEMAS

Errores de Redondeo

Ejemplo 1: Redondeo de un número decimal

```python
num = 3.1415926535  # Número original
aprox = round(num, 2)  # Redondeado a 2 decimales
error_redondeo = num - aprox
print(f'Error de redondeo: {error_redondeo}')
```

- **Explicación:**

  - Se toma el número `3.1415926535` y se redondea a 2 decimales (`3.14`).
  - El **error de redondeo** se calcula como la diferencia entre el número original y su aproximación: 3.1415926535 − 3.14=0.0015926535
  - Salida:

```
PS C:\Users\aneth\Downloads\VISUALSTUDIO\METODOS NUMERICOS\Metodos_Numericos> & C:/
/Users/aneth/Downloads/VISUALSTUDIO/METODOS NUMERICOS/Metodos_Numericos/PROBLEMARIO
Error de redondeo: 0.0015926534999999298
PS C:\Users\aneth\Downloads\VISUALSTUDIO\METODOS NUMERICOS\Metodos_Numericos>
```

Ejemplo 2: Redondeo de una división

```python
num = 10 / 3  # Resultado exacto
aprox = round(num, 3)  # Redondeado a 3 decimales
error_redondeo = num - aprox
print(f'Error de redondeo: {error_redondeo}')
```

- **Explicación:**

  - `10 / 3` genera un número **periódico**: `3.333333...`
  - Se redondea a 3 decimales (`3.333`).
  - El error de redondeo es: 3.3333333... − 3.333 = 0.0003333...
  - Salida

```
PS C:\Users\aneth\Downloads\VISUALSTUDIO\METODOS NUMERICOS\Metodos_Numericos> &
/Users/aneth/Downloads/VISUALSTUDIO/METODOS NUMERICOS/Metodos_Numericos/PROBLEM
Error de redondeo: 0.0003333333333332966
PS C:\Users\aneth\Downloads\VISUALSTUDIO\METODOS NUMERICOS\Metodos_Numericos>
```

Ejemplo 3: Uso de flotantes en Python

```
PROBLEMARIO_T1 > Problema_3.py > ...
1   num = 0.1 + 0.2  # Suma de flotantes en Python
2   aprox = round(num, 1)  # Redondeo a 1 decimal
3   error_redondeo = num - aprox
4   print(f'Error de redondeo: {error_redondeo}')
```

- **Explicación:**

  - En teoría, `0.1 + 0.2 = 0.3`, pero en Python ocurre un error debido a cómo se representan los números flotantes en binario.
  - `num` no es exactamente `0.3`, sino algo como `0.30000000000000004`.
  - Se redondea a `1` decimal (`0.3`).
  - El error de redondeo es: 0.30000000000000004 – 0.3 = 0.00000000000000004
  - Salida: (Número muy pequeño, pero distinto de cero).

```
PS C:\Users\aneth\Downloads\VISUALSTUDIO\METODOS NUMERICOS\Metodos_Numericos> &
/Users/aneth/Downloads/VISUALSTUDIO/METODOS NUMERICOS/Metodos_Numericos/PROBLEMA
Error de redondeo: 5.551115123125783e-17
PS C:\Users\aneth\Downloads\VISUALSTUDIO\METODOS NUMERICOS\Metodos_Numericos>
```

Ejemplo 4: Raíz cuadrada aproximada

```
PROBLEMARIO_T1 > Problema_4.py > ...
1   import math
2   num = math.sqrt(2)  # Raíz cuadrada exacta
3   aprox = round(num, 4)  # Redondeado a 4 decimales
4   error_redondeo = num - aprox
5   print(f'Error de redondeo: {error_redondeo}')
```

- **Explicación:**

  - `math.sqrt(2)` genera un número irracional: `1.414213562...`
  - Se redondea a 4 decimales (`1.4142`).
  - El error de redondeo es: 1.414213562 – 1.4142 = 0.000013562
  - Salida:

```
PS C:\Users\aneth\Downloads\VISUALSTUDIO\METODOS NUMERICOS\Metodos_Numericos> & C
/Users/aneth/Downloads/VISUALSTUDIO/METODOS NUMERICOS/Metodos_Numericos/PROBLEMA
Error de redondeo: 1.3562373095243885e-05
PS C:\Users\aneth\Downloads\VISUALSTUDIO\METODOS NUMERICOS\Metodos_Numericos>
```

Ejemplo 5: Representación en binario

```
PROBLEMARIO_T1 > Problema_5.py > ...
1   num = 0.1  # Número original
2   aprox = format(num, '.17f')  # Representación con 17 decimales
3   print(f'Número almacenado en Python: {aprox}')
```

🔹 **Explicación:**

- Los números decimales no siempre tienen una representación exacta en binario.
- `0.1` en binario se convierte en un número periódico, causando errores mínimos de precisión.
- La salida nos muestra la representación interna con 17 decimales.
- Salida: (Aquí se ve que `0.1` realmente se almacena como `0.10000000000000001` en memoria).

```
PS C:\Users\aneth\Downloads\VISUALSTUDIO\METODOS NUMERICOS\Metodos_Numericos> &
/Users/aneth/Downloads/VISUALSTUDIO/METODOS NUMERICOS/Metodos_Numericos/PROBLEMA
Número almacenado en Python: 0.10000000000000001
PS C:\Users\aneth\Downloads\VISUALSTUDIO\METODOS NUMERICOS\Metodos_Numericos>
```

Error Absoluto.

Ejemplo 1:

Imagina que tienes una regla y mides un lápiz. Si el lápiz mide 10 cm y tú dijiste que mide 9.5 cm, te equivocaste por 0.5 cm. Eso es un error absoluto.

Código Python:

```
# Valores reales y aproximados
valores_reales = [10, 25, 50, 100, 200]
valores_aproximados = [9.5, 24.8, 49.9, 99, 201]

# Calcular error absoluto
for real, aprox in zip(valores_reales, valores_aproximados):
    error_absoluto = abs(real - aprox)
    print(f"Valor Real: {real}, Valor Aproximado: {aprox}, Error Absoluto: {error_absoluto}")
```

Corrida del programa:

```
Valor Real: 10, Valor Aproximado: 9.5, Error Absoluto: 0.5
Valor Real: 25, Valor Aproximado: 24.8, Error Absoluto: 0.2
Valor Real: 50, Valor Aproximado: 49.9, Error Absoluto: 0.1
Valor Real: 100, Valor Aproximado: 99, Error Absoluto: 1
Valor Real: 200, Valor Aproximado: 201, Error Absoluto: 1
```

Ejemplo 2:

Medir un lápiz:

Problema: Mides un lápiz y obtienes 15 cm. La medida real es 15.2 cm.

Error absoluto: $|15 - 15.2| = 0.2$ cm

Código Python:

```
medida_tu = 15
medida_real = 15.2
error_absoluto = abs(medida_tu - medida_real)
print("Error absoluto:", error_absoluto, "cm")
```

Corrida del programa:

```
Error absoluto: 0.2 cm
```

Ejemplo 3:

Contar caramelos:

Problema: Cuentas caramelos y obtienes 25. La cantidad real es 24.

Error absoluto: |25 - 24| = 1 caramelo

Código Python:

```
cantidad_tu = 25
cantidad_real = 24
error_absoluto = abs(cantidad_tu - cantidad_real)
print("Error absoluto:", error_absoluto, "caramelos")
```

Corrida del programa:

Error absoluto: 1 caramelos

Ejemplo 4:

Medir la temperatura:

Problema: Mides la temperatura y obtienes 25 grados Celsius. La temperatura real es 24.5 grados Celsius.

Error absoluto: |25 - 24.5| = 0.5 grados Celsius

Código Python:

```
temperatura_tu = 25
temperatura_real = 24.5
error_absoluto = abs(temperatura_tu - temperatura_real)
print("Error absoluto:", error_absoluto, "grados Celsius")
```

Corrida del programa:

Error absoluto: 0.5 grados Celsius

Ejemplo 5:

Calcular el área de un cuadrado:

Problema: Calcular el área de un cuadrado y obtienes 100 cm^2. El área real es 99 cm^2.

Error absoluto: |100 - 99| = 1 cm^2

Código Python:

```
area_tu = 100
area_real = 99
error_absoluto = abs(area_tu - area_real)
```

```
print("Error absoluto:", error_absoluto, "cm²")
```

Corrida del programa:

Error absoluto: 1 cm²

Error Relativo

Ejemplo 1:

Ahora, piensa que si el lápiz fuera más grande (por ejemplo, 100 cm) y te equivocas por 1 cm, ese error es más pequeño comparado con el tamaño del lápiz. Eso es un error relativo.

Código Python:

```
# Valores reales y aproximados
valores_reales = [10, 25, 50, 100, 200]
valores_aproximados = [9.5, 24.8, 49.9, 99, 201]

# Calcular error relativo
for real, aprox in zip(valores_reales, valores_aproximados):
 error_relativo = abs(real - aprox) / abs(real)
  print(f"Valor Real: {real}, Valor Aproximado: {aprox}, Error Relativo: {error_relativo}")
```

Corrida del programa:

Valor Real: 10, Valor Aproximado: 9.5, Error Relativo: 0.05
Valor Real: 25, Valor Aproximado: 24.8, Error Relativo: 0.008
Valor Real: 50, Valor Aproximado: 49.9, Error Relativo: 0.002
Valor Real: 100, Valor Aproximado: 99, Error Relativo: 0.01
Valor Real: 200, Valor Aproximado: 201, Error Relativo: 0.005

Ejemplo 2:

Medir una cuerda:

Problema: Mides una cuerda y obtienes 2.5 metros. La medida real es 2.55 metros.
Error relativo: (0.05 / 2.55) * 100 = 1.96% (redondeado a dos decimales).

Código Python:

```
medida_tu = 2.5
medida_real = 2.55
error_relativo = round((abs(medida_tu - medida_real) / medida_real) * 100, 2)
print("Error relativo:", error_relativo, "%")
```

Corrida del programa:

Error relativo: 1.96

Ejemplo 3:

Pesar una manzana:

Problema: Pesas una manzana y obtienes 150 gramos. El peso real es 153 gramos.

Error relativo: (3 / 153) * 100 = 1.96% (redondeado a dos decimales)

Código Python:

```
peso_tu = 150
peso_real = 153
error_relativo = round((abs(peso_tu - peso_real) / peso_real) * 100, 2)
print("Error relativo:", error_relativo, "%")
```

Corrida del programa:

Error relativo: 1.96

Ejemplo 4:

Calcular el precio de un libro:

Problema: Calcular el precio de un libro y obtienes 15.50 soles. El precio real es 15.75 soles.

Error relativo: (0.25 / 15.75) * 100 = 1.59% (redondeado a dos decimales)

Código Python:

```
precio_tu = 15.50
precio_real = 15.75
error_relativo = round((abs(precio_tu - precio_real) / precio_real) * 100, 2)
print("Error relativo:", error_relativo, "%")
```

Corrida del programa:

Error relativo: 1.59

Ejemplo 5:

Estimar la distancia a una ciudad:

Problema: Estimar la distancia a una ciudad y obtienes 120 km. La distancia real es 125 km.

Error relativo: (5 / 125) * 100 = 4%

Código Python:

```
distancia_tu = 120
distancia_real = 125
error_relativo = round((abs(distancia_tu - distancia_real) / distancia_real) * 100, 2)
print("Error relativo:", error_relativo, "%")
```

Corrida del programa:

Error relativo: 4.0

Error truncamiento

Ejemplo 1:

Se desea calcular el área de un círculo con un radio=3 unidades utilizando la fórmula A=πr(^2). Sin embargo, en lugar de utilizar el valor exacto de π, se decide aproximar π como 3.14.

Código Python:

```
import math

# Datos

radio = 3

pi_aprox = 3.14

pi_exacto = math.pi

# Cálculo del área con el valor truncado de π
```

```
area_aprox = pi_aprox * (radio ** 2)

# Cálculo del área con el valor exacto de π

area_exacta = pi_exacto * (radio ** 2)

# Cálculo del error de truncamiento

error_trunc_abs = abs(area_exacta - area_aprox)

error_trunc_rel = error_trunc_abs / area_exacta

print(f"Área aproximada (π ≈ 3.14): {area_aprox}")

print(f"Área exacta (π = {pi_exacto}): {area_exacta}")

print(f"Error absoluto: {error_trunc_abs}")

print(f"Error relativo: {error_trunc_rel:.5f}")
```

Corrida del programa:

```
Área aproximada (π ≈ 3.14): 28.26
Área exacta (π = 3.141592653589793): 28.274333882308138
 Error absoluto: 0.014333882308137984
Error relativo: 0.00051
```

Ejemplo 2:

Se desea calcular el área de un triángulo rectángulo con base b=6 unidades y altura h=4 unidades. Para hacerlo, se utiliza la fórmula del área de un triángulo: A= ½bh. Sin embargo, debido a un error en la medición, la altura se estima como h=3.9 unidades.

Código Python:

```
# Datos

base = 6

altura_exacto = 4.0
```

```
altura_aprox = 3.9

# Cálculo del área con la altura real

area_exacta = (1/2) * base * altura_exacto

# Cálculo del área con la altura aproximada

area_aprox = (1/2) * base * altura_aprox

# Cálculo del error de truncamiento

error_trunc_abs = abs(area_exacta - area_aprox)

error_trunc_rel = error_trunc_abs / area_exacta

print(f"Área exacta (h = {altura_exacto}): {area_exacta}")

print(f"Área aproximada (h ≈ {altura_aprox}): {area_aprox}")

print(f"Error absoluto: {error_trunc_abs}")

print(f"Error relativo: {error_trunc_rel:.5f}")
```

Corrida del programa:

```
Área exacta (h = 4.0): 12.0

Área aproximada (h ≈ 3.9): 11.7

Error absoluto: 0.3

Error relativo: 0.02500
```

Ejemplo 3:
Se desea calcular la velocidad media de un auto que recorre una distancia de 150 km en un tiempo de 2.5 horas. Sin embargo, debido a un error en la medición, el tiempo se aproxima a 2.4 horas.

Código Python:

```
# Datos

distancia = 150

tiempo_exacto = 2.5

tiempo_aprox = 2.4

# Cálculo de la velocidad exacta

velocidad_exacta = distancia / tiempo_exacto

# Cálculo de la velocidad aproximada

velocidad_aprox = distancia / tiempo_aprox

# Cálculo del error de truncamiento

error_trunc_abs = abs(velocidad_exacta - velocidad_aprox)

error_trunc_rel = error_trunc_abs / velocidad_exacta

print(f"Velocidad exacta (t = {tiempo_exacto} h): {velocidad_exacta} km/h")

print(f"Velocidad aproximada (t ≈ {tiempo_aprox} h): {velocidad_aprox} km/h")

print(f"Error absoluto: {error_trunc_abs}")

print(f"Error relativo: {error_trunc_rel:.5f}")
```

Corrida del programa:

```
Velocidad exacta (t = 2.5 h): 60.0 km/h
Velocidad aproximada (t ≈ 2.4 h): 62.5 km/h
Error absoluto: 2.5
Error relativo: 0.04167
```

Ejemplo 4:

Se desea calcular la energía potencial de un objeto con una masa de 10 kg y una altura de 5 metros, usando la aceleración de la gravedad g = 9.81 m/s^2. Sin embargo, debido a un error, la masa se aproxima a 9.8 kg.

Código Python:

```
# Datos

masa_exacto = 10.0  # Masa  en kg

masa_aprox = 9.8

altura = 5.0

gravedad = 9.81

# Cálculo de la energía potencial exacta

energia_exacta = masa_exacto * gravedad * altura

# Cálculo de la energía potencial aproximada

energia_aprox = masa_aprox * gravedad * altura

# Cálculo del error de truncamiento

error_trunc_abs = abs(energia_exacta - energia_aprox)

error_trunc_rel = error_trunc_abs / energia_exacta

print(f"Energía exacta (m = {masa_exacto} kg): {energia_exacta} J")

print(f"Energía aproximada (m ≈ {masa_aprox} kg): {energia_aprox} J")

print(f"Error absoluto: {error_trunc_abs}")

print(f"Error relativo: {error_trunc_rel:.5f}")
```

Corrida del programa:

```
Energía exacta (m = 10.0 kg): 490.5 J

Energía aproximada (m ≈ 9.8 kg): 480.69 J
```

Error absoluto: 9.81

Error relativo: 0.02000

Ejemplo 5:
Se desea calcular el volumen de una esfera con un radio de 4.5 cm, debido a un error en la medición, el radio se aproxima a 4.4 cm.

Código Python:

```
import math
# Datos
radio_exacto = 4.5
radio_aprox = 4.4

# Cálculo del volumen exacto
volumen_exacto = (4/3) * math.pi * (radio_exacto ** 3)

# Cálculo del volumen aproximado
volumen_aprox = (4/3) * math.pi * (radio_aprox ** 3)

# Cálculo del error de truncamiento
error_trunc_abs = abs(volumen_exacto - volumen_aprox)
error_trunc_rel = error_trunc_abs / volumen_exacto

print(f"Volumen exacto (r = {radio_exacto} cm): {volumen_exacto} cm³")
print(f"Volumen aproximado (r ≈ {radio_aprox} cm): {volumen_aprox} cm³")
print(f"Error absoluto: {error_trunc_abs}")
print(f"Error relativo: {error_trunc_rel:.5f}")
```

Corrida del programa:

Volumen exacto (r = 4.5 cm): 381.70 cm^3

Volumen aproximado (r ≈ 4.4 cm): 356.82 cm^3

Error absoluto: 24.88

Error relativo: 0.06521

Exactitud

Ejercicio 1:

Se desea aproximar la derivada de la función $f(x) = e^x$ en x=1 usando la fórmula de la diferencia progresiva para diferentes valores "h" y comparar la exactitud del resultado con la derivada exacta $f'(1) = e^1$.

Se pide calcular el error absoluto y error relativo para $h = 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}$.

main.py

```python
import numpy as np

# Definir la función f(x) = e^x
def f(x):
    return np.exp(x)

# Derivada exacta en x = 1
exact_derivative = np.exp(1)

# Valores de h a evaluar
h_values = [10**-i for i in range(1, 6)]

# Calcular la derivada aproximada y errores
print(f"{'h':<10}{'Aprox. f\'(1)':<20}{'Error Abs':<20}{'Error Rel':<20}")
for h in h_values:
    approx_derivative = (f(1 + h) - f(1)) / h  # Diferencia progresiva
    error_abs = abs(exact_derivative - approx_derivative)
    error_rel = error_abs / abs(exact_derivative)

    print(f"{h:<10.1e}{approx_derivative:<20.10f}{error_abs:<20.10f}{error_rel:<20.10f}")


```

Explicación del código

1. Se define la función $f(x) = e^x$.
2. Se establece la derivada exacta en x=1, que es e^1.
3. Se calcula la derivada aproximada usando la fórmula de diferencia progresiva.
4. Se determinan los errores absoluto y relativo.
5. Se imprime una tabla con los resultados.

Salida esperada

h	Aprox. f'(1)	Error Abs	Error Rel
1.0e-01	2.859016006826472	0.141618841308268	0.049239389239065
1.0e-02	2.731918730728194	0.014521565210010	0.005044927258409
1.0e-03	2.718145926824925	0.000748761306741	0.000275769188575
1.0e-04	2.718034184187073	0.000637018056407	0.000234919433612
1.0e-05	2.718280469095753	0.000001266852272	0.000000466280078

A medida que “h” se reduce, la aproximación mejora, pero si “h” es demasiado pequeño, los errores de redondeo pueden afectar la precisión.

Ejercicio 2:

Se desea aproximar la integral de la función $f(x) = x^2$ en el intervalo [0, 1] usando la regla del trapecio.

Compara los resultados para n=1, 2, 4, 8 divisiones del intervalo y calcula los errores absolutos y relativos.

main.py

```python
import numpy as np

# Función f(x) = x^2
def f(x):
    return x**2

# Integral exacta de x^2 en [0, 1]
exact_integral = 1/3

# Valores de n para diferentes divisiones
n_values = [1, 2, 4, 8]

# Calcular la integral aproximada y errores
print(f"{'n':<10}{'Aprox. Integral':<20}{'Error Abs':<20}{'Error Rel':<20}")
for n in n_values:
    h = 1 / n
    x = np.linspace(0, 1, n+1)
    integral_approx = h * (np.sum(f(x)) - f(0) / 2 - f(1) / 2)
    error_abs = abs(exact_integral - integral_approx)
    error_rel = error_abs / abs(exact_integral)

    print(f"{n:<10}{integral_approx:<20.10f}{error_abs:<20.10f}{error_rel:<20.10f}")

```

Salida esperada

```
n         Aprox. Integral     Error Abs           Error Rel
1         0.5000000000        0.1666666667        0.5000000000
2         0.3750000000        0.0416666667        0.1250000000
4         0.3437500000        0.0104166667        0.0312500000
8         0.3359375000        0.0026041667        0.0078125000


** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

Ejercicio 3:

Usando el método de bisección, encuentra la raíz de la ecuación $f(x) = x^2 - 2$ en el intervalo [1,2], con una tolerancia de 10^{-5}. Calcula el error absoluto y relativo en cada iteración.

main.py

```python
def f(x):
    return x**2 - 2

# Método de bisección
def bisection(a, b, tol=1e-5):
    iter_count = 0
    while (b - a) / 2 > tol:
        c = (a + b) / 2
        if f(c) == 0:
            break
        elif f(a) * f(c) < 0:
            b = c
        else:
            a = c
        iter_count += 1
    return c, iter_count

# Intervalo [1, 2]
root, iterations = bisection(1, 2)
exact_root = np.sqrt(2)
error_abs = abs(exact_root - root)
error_rel = error_abs / abs(exact_root)

print(f"Raíz aproximada: {root}")
print(f"Error Absoluto: {error_abs}")
print(f"Error Relativo: {error_rel}")
```

Salida esperada

```
Raíz aproximada: 1.414215087890625
Error Absoluto: 1.0728836059570312e-05
Error Relativo: 7.5870468988592e-06
```

Ejercicio 4:

Resuelve la ecuación diferencial $y' = -2y$, con la condición inicial $y(0) = 1$, usando el **método de Euler** para aproximar la solución en $t = 1$ con $h = 0.1$. Calcula el **error absoluto** y **relativo** comparando con la solución exacta $y(t) = e^{-2t}$.

main.py +

```python
# Solución exacta y(t) = e^(-2t)
def exact_solution(t):
    return np.exp(-2 * t)

# Método de Euler
def euler_method(f, t0, y0, h, t_end):
    t_values = np.arange(t0, t_end + h, h)
    y_values = [y0]
    for t in t_values[:-1]:
        y_values.append(y_values[-1] + h * f(t, y_values[-1]))
    return t_values, y_values

# Resolución con h = 0.1
t_values, y_values = euler_method(f, 0, 1, 0.1, 1)
approx_solution = y_values[-1]
exact_sol = exact_solution(1)
error_abs = abs(exact_sol - approx_solution)
error_rel = error_abs / abs(exact_sol)

print(f"Solución aproximada: {approx_solution}")
print(f"Solución exacta: {exact_sol}")
print(f"Error Absoluto: {error_abs}")
print(f"Error Relativo: {error_rel}")

```

Salida esperada

```
Solución aproximada: 0.13508517176729922
Solución exacta: 0.1353352832366127
Error Absoluto: 0.0002501114693134915
Error Relativo: 0.001848177997393319
```

Ejercicio 5:

Aproxima la raíz cuadrada de 5 usando el **método de Newton-Raphson** con una tolerancia de 10^{-5}. La función es $f(x) = x^2 - 5$, y su derivada es $f'(x) = 2x$. Calcula los **errores absolutos** y **relativos** en cada iteración.

```python
# Función f(x) = x^2 - 5
def f(x):
    return x**2 - 5

# Derivada de f(x)
def f_prime(x):
    return 2 * x

# Método de Newton-Raphson
def newton_method(f, f_prime, x0, tol=1e-5):
    iter_count = 0
    while abs(f(x0)) > tol:
        x0 = x0 - f(x0) / f_prime(x0)
        iter_count += 1
    return x0, iter_count

# Aproximación inicial x0 = 2
root, iterations = newton_method(f, f_prime, 2)
exact_root = np.sqrt(5)
error_abs = abs(exact_root - root)
error_rel = error_abs / abs(exact_root)

print(f"Raíz aproximada: {root}")
print(f"Error Absoluto: {error_abs}")
print(f"Error Relativo: {error_rel}")
```

Salida esperada

```
Raíz aproximada: 2.236067977499978
Error Absoluto: 4.440892098500626e-15
Error Relativo: 1.9852338666698312e-15
```

Precisión

Ejercicio 1:

Dado un valor real V_{real} y una aproximación $V_{\text{aproximado}}$, calcula el **error absoluto** y el **error relativo**.

main.py

```python
def calcular_errores(valor_real, valor_aproximado):
    error_absoluto = abs(valor_real - valor_aproximado)
    error_relativo = error_absoluto / abs(valor_real) if valor_real != 0 else float('inf')
    return error_absoluto, error_relativo

# Ejemplo con e ≈ 2.71828 y una aproximación 2.5
valor_real = 2.71828
valor_aproximado = 2.5
ea, er = calcular_errores(valor_real, valor_aproximado)

print(f"Error Absoluto: {ea}")
print(f"Error Relativo: {er}")
```

Salida esperada

```
Error Absoluto: 0.21828000000000003
Error Relativo: 0.08030077843342115


** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

Ejercicio 2:

Aproximar e^x usando los primeros **n** términos de la serie de Taylor:

$$e^x \approx 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

main.py

```python
import math

def aproximar_ex(x, n):
    suma = 0
    for i in range(n):
        suma += (x**i) / math.factorial(i)
    return suma

x = 1  # Queremos calcular e^1
n = 5  # Número de términos en la serie

aprox = aproximar_ex(x, n)
real = math.exp(x)
ea, er = calcular_errores(real, aprox)

print(f"Aproximación con {n} términos: {aprox}")
print(f"Valor real: {real}")
print(f"Error Absoluto: {ea}")
print(f"Error Relativo: {er}")
```

Salida esperada

Aproximación: `2.708333333333333`

Valor real: `2.718281828459045`

Error Absoluto: `0.009948495125712054`

Error Relativo: `0.003659846827343768`

Ejercicio 3:

Suma 10^6 veces el valor 10^{-6} y compárala con el valor teórico esperado.

main.py +

```python
import numpy as np

def suma_numeros_pequenos(n, valor):
    suma = 0.0
    for _ in range(n):
        suma += valor
    return suma

n = 10**6
valor = 10**-6
suma = suma_numeros_pequenos(n, valor)
esperado = n * valor
ea, er = calcular_errores(esperado, suma)

print(f"Suma calculada: {suma}")
print(f"Valor esperado: {esperado}")
print(f"Error Absoluto: {ea}")
print(f"Error Relativo: {er}")

```

Salida esperada

Suma calculada: `1.000000000007918`

Valor esperado: `1.0`

Error Absoluto: `7.918110611626616e-12`

Error Relativo: `7.918110611626616e-12`

Ejercicio 4:

Evaluar la expresión:

$$f(x) = \frac{1 - \cos(x)}{x^2}$$

para valores de x muy pequeños.

main.py

```python
def funcion_inestable(x):
    return (1 - math.cos(x)) / x**2 if x != 0 else float('inf')

x_valores = [10**-1, 10**-2, 10**-5, 10**-10]
for x in x_valores:
    print(f"x = {x}, f(x) = {funcion_inestable(x)}")

```

Salida esperada

x	$f(x)$
`0.1`	`0.49958347219741783`
`0.01`	`0.4999958333473664`
`1e-5`	`0.5000000413701854`
`1e-10`	`0.0`

Ejercicio 5:

Encontrar x tal que $x = \cos(x)$ iterando hasta alcanzar una tolerancia de 10^{-6}.

main.py

```python
def punto_fijo(func, x0, tol=1e-6, max_iter=100):
    x = x0
    for i in range(max_iter):
        x_new = func(x)
        if abs(x_new - x) < tol:
            return x_new, i + 1
        x = x_new
    return x, max_iter

x_sol, iteraciones = punto_fijo(math.cos, 1.0)
print(f"Solución encontrada: {x_sol} en {iteraciones} iteraciones")

```

Salida esperada

Solución encontrada: `0.7390855263619245`

Número de iteraciones: `34`

CONCLUSIÓN:

Cuando hacemos cálculos, a veces los números no son exactos y eso genera errores. Estos errores pueden ser pequeños o grandes, pero siempre afectan el resultado.

Es imposible evitar los errores por completo, pero si los conocemos bien, podemos hacer cálculos más precisos y evitar problemas en cosas importantes como la construcción, la programación y la ciencia. ¡Por eso es tan importante aprender sobre ellos!, y mejorar cada día , pero sobre todo aprender de nuestros errores.