[Tarea 03] Ejercicios Unidad 01-B

Richard Tipantiza

Tabla de Contenidos

REPOSITORIO

1. Utilice aritmética de corte de tres dígitos para calcular las siguientes sumas. Para cada parte, ¿qué método es más preciso y por qué?

a. $\sum_{i=1}^{10}(1/i^2)$ primero por $\frac{1}{1}+\frac{1}{4}+\cdots+\frac{1}{100}$ y luego por $\frac{1}{100}+\frac{1}{81}+\cdots+\frac{1}{1}$

b. $\sum_{i=1}^{10}(1/i^3)$ primero por $\frac{1}{1}+\frac{1}{8}+\frac{1}{27}+\cdots+\frac{1}{1000}$ y luego por $\frac{1}{1000}+\frac{1}{729}+\cdots+\frac{1}{1}$

Pseudocódigo

INICIO

Para el literal a y b: 1. Definir la secuencia de términos en orden ascendente y descendente 2. Implementar función de suma con aritmética de 3 dígitos 3. Calcular las sumas 4. Comparar con el valor exacto 5. Determinar qué método es más preciso

FIN

Código

```
def chop_tres_digitos(n):
    if n == 0:
        return 0.0

    s_numero = "{:.8e}".format(n)
    partes = s_numero.split('e')
    mantisa_str = partes[0]
    exponente_str = partes[1]
```

```python
    mantisa_cortada = ""
    if mantisa_str.startswith('-'):
        mantisa_cortada = mantisa_str[0:5]
    else:
        mantisa_cortada = mantisa_str[0:4]

    numero_final_str = mantisa_cortada + "e" + exponente_str
    return float(numero_final_str)


print("--- Parte A: Suma de 1/i^2 ---")

suma_a1 = 0.0
for i in range(1, 11):
    termino = 1 / (i*i)
    termino_cortado = chop_tres_digitos(termino)
    suma_a1 = suma_a1 + termino_cortado
    suma_a1 = chop_tres_digitos(suma_a1)

print("Metodo 1 (grande a pequeno):", suma_a1)

suma_a2 = 0.0
for i in range(10, 0, -1):
    termino = 1 / (i*i)
    termino_cortado = chop_tres_digitos(termino)
    suma_a2 = suma_a2 + termino_cortado
    suma_a2 = chop_tres_digitos(suma_a2)

print("Metodo 2 (pequeno a grande):", suma_a2)


print("\n--- Parte B: Suma de 1/i^3 ---")

suma_b1 = 0.0
for i in range(1, 11):
    termino = 1 / (i*i*i)
    termino_cortado = chop_tres_digitos(termino)
    suma_b1 = suma_b1 + termino_cortado
    suma_b1 = chop_tres_digitos(suma_b1)

print("Metodo 1 (grande a pequeno):", suma_b1)
```

```python
suma_b2 = 0.0
for i in range(10, 0, -1):
    termino = 1 / (i*i*i)
    termino_cortado = chop_tres_digitos(termino)
    suma_b2 = suma_b2 + termino_cortado
    suma_b2 = chop_tres_digitos(suma_b2)

print("Metodo 2 (pequeno a grande):", suma_b2)
```

```
--- Parte A: Suma de 1/i^2 ---
Metodo 1 (grande a pequeno): 1.53
Metodo 2 (pequeno a grande): 1.54

--- Parte B: Suma de 1/i^3 ---
Metodo 1 (grande a pequeno): 1.16
Metodo 2 (pequeno a grande): 1.19
```

¿qué método es más preciso y por qué?

En ambos casos, el método de sumar en orden descendente es más preciso porque minimiza el error por truncamiento al sumar primero los términos más pequeños.

2. La serie de Maclaurin para la función arcotangente converge para $-1 \leq x \leq 1$ y está dada por

$$\arctan x = \lim_{n\to\infty} P_n(x) = \lim_{n\to\infty} \sum_{i=1}^{n} (-1)^{(i+1)} \frac{x^{2i-1}}{2i-1}$$

a. Utilice el hecho de que $\tan \pi/4 = 1$ para determinar el **número n de términos** de la serie que se necesita sumar para garantizar que $|4P_n(1) - \pi| < 10^{-3}$.

b. El lenguaje de programación C++ requiere que el valor de π se encuentre dentro de 10^{-10}. ¿Cuántos términos de la serie se necesitarían sumar para obtener este grado de precisión?

Pseudocódigo

INICIO

Parte a:

1. Definir función $P_n(x)$ que calcula la suma de la serie hasta n términos.
2. Inicializar $n = 1$.
3. **Mientras** $|4 \cdot P_n(1) - \pi| \geq 10^{-3}$:
    - Incrementar n.
    - Calcular nuevo $P_n(1)$.
4. Devolver n.

Parte b:

1. El mismo proceso pero con **tolerancia** 10^{-10}.
2. Encontrar n donde $|4 \cdot P_n(1) - \pi| < 10^{-10}$.

FIN

Código

```
import math
def calcular_terminos_a():
    def P_n(n):
        return sum((-1)**(i+1) / (2*i-1) for i in range(1, n+1))
    n = 1
    while True:
        pi_approx = 4 * P_n(n)
        error = abs(pi_approx - math.pi)
        if error < 1e-3:
            return n
        n += 1
        if n > 10000:
            break
    return n

n_a = calcular_terminos_a()
pi_approx_a = 4 * sum((-1)**(i+1) / (2*i-1) for i in range(1, n_a+1))
error_a = abs(pi_approx_a - math.pi)
print("Literal a:")
```

```
print(f"Número de términos requeridos: {n_a}")
print(f"Aproximación de : {pi_approx_a}")
print(f"Error absoluto: {error_a}")

print(f"¿Cumple con 10^-3? {'Sí' if error_a < 1e-3 else 'No'}")
```

```
Literal a:
Número de términos requeridos: 1000
Aproximación de : 3.140592653839792
Error absoluto: 0.0009999997499998692
¿Cumple con 10^-3? Sí
```

```
def expresion(n):
    return (-1)**(n + 1) / (2*n - 1)
def error_Absoluto(valor_aproximado):
    import math
    return abs(math.pi - valor_aproximado)

def sumatoriaConLimitacion(error_maximo):
    sumatoria = 0
    contador = 1
    mensaje = ""
    while True: # Ejecutar hasta alcanzar el error deseado
        sumatoria += expresion(contador)
        if error_Absoluto(4 * sumatoria) < error_maximo:

            mensaje = f"Número de términos necesarios: {contador}"
            break
        contador += 1
        if contador >= 1000000:
            mensaje = f"Número de iteraciones máximo alcanzado: {contador}"
            break
    return sumatoria, mensaje

error_maxi = 1e-10
suma_aproxi, num_termi = sumatoriaConLimitacion(error_maxi)
error_b = error_Absoluto(4 * suma_aproxi)
print("\nLiteral b:")
print(num_termi)
print(f"Valor aproximado de : {4 * suma_aproxi}")
print(f"Error absoluto: {error_b}")
print(f"¿Cumple con 10^-10? {'Sí' if error_b < 1e-10 else 'No'}")
```

```
Literal b:
Número de iteraciones máximo alcanzado: 1000000
Valor aproximado de : 3.1415936535907742
Error absoluto: 1.0000009811328425e-06
¿Cumple con 10^-10? No
```

3. Otra fórmula para calcular π se puede deducir a partir de la identidad $\pi/4 = 4\arctan\frac{1}{5} - \arctan\frac{1}{239}$. Determine el número de términos que se deben sumar para garantizar una aproximación π dentro de 10^{-3}.

Pseudocódigo

INICIO

1. Definir función $arctan\_series(x, n)$ que calcula $\arctan(x)$ usando n términos.
2. Definir función $calcular\_pi(n1, n2)$ usando la fórmula dada.
3. Inicializar $n = 1$.
4. **Mientras** $error > 10^{-3}$:
   - Calcular π aproximado con n términos.
   - Calcular error absoluto.
   - Incrementar n.
5. Devolver n requerido.

FIN

Código

```
tolerancia = 0.001
n = 1

denominador = (2.0 * n) + 1.0
error_1 = 16.0 / (denominador * (5.0 ** denominador))
error_2 = 4.0 / (denominador * (239.0 ** denominador))
error_total = error_1 + error_2
```

```python
while error_total >= tolerancia:
    n = n + 1

    denominador = (2.0 * n) + 1.0
    error_1 = 16.0 / (denominador * (5.0 ** denominador))
    error_2 = 4.0 / (denominador * (239.0 ** denominador))
    error_total = error_1 + error_2

print("Numero de terminos (n) para error < 10^-3:")
print(n)
```

```
Numero de terminos (n) para error < 10^-3:
3
```

4. Compare los siguientes tres algoritmos. ¿Cuándo es correcto el algoritmo de la parte 1a?

a. **ENTRADA** $n, x_1, x_2, \ldots, x_n$. **SALIDA** PRODUCT. *Paso 1* Determine $PRODUCT = 0$. *Paso 2* Para $i = 1, 2, \ldots, n$ haga Determine $PRODUCT = PRODUCT \cdot x_i$. *Paso 3* SALIDA PRODUCT; PARE.

b. **ENTRADA** $n, x_1, x_2, \ldots, x_n$. **SALIDA** PRODUCT. *Paso 1* Determine $PRODUCT = 1$. *Paso 2* Para $i = 1, 2, \ldots, n$ haga Set $PRODUCT = PRODUCT \cdot x_i$. *Paso 3* SALIDA PRODUCT; PARE.

c. **ENTRADA** $n, x_1, x_2, \ldots, x_n$. **SALIDA** PRODUCT. *Paso 1* Determine $PRODUCT = 1$. *Paso 2* Para $i = 1, 2, \ldots, n$ haga si $x_i = 0$ entonces determine $PRODUCT = 0$; SALIDA PRODUCT; PARE. Determine $PRODUCT = PRODUCT \cdot x_i$. *Paso 3* SALIDA PRODUCT; PARE.

Código

```python
x1 = [4, 5, 6]
n1 = 3

print("--- Prueba 1 (sin 0) ---")

print("Algoritmo A")
producto_a = 0
```

```python
for i in range(n1):
    producto_a = producto_a * x1[i]
print(producto_a)

print("Algoritmo B")
producto_b = 1
for i in range(n1):
    producto_b = producto_b * x1[i]
print(producto_b)

print("Algoritmo C")
producto_c = 1
for i in range(n1):
    if x1[i] == 0:
        producto_c = 0
        break
    producto_c = producto_c * x1[i]
print(producto_c)
```

```
--- Prueba 1 (sin 0) ---
Algoritmo A
0
Algoritmo B
120
Algoritmo C
120
```

```python
print("--- Prueba 2 (con 0) ---")

x2 = [4, 0, 6]
n2 = 3

print("Algoritmo A")
producto_a = 0
for i in range(n2):
    producto_a = producto_a * x2[i]
print(producto_a)

print("Algoritmo B")
producto_b = 1
for i in range(n2):
```

```python
    producto_b = producto_b * x2[i]
print(producto_b)

print("Algoritmo C")
producto_c = 1
for i in range(n2):
    if x2[i] == 0:
        producto_c = 0
        break
    producto_c = producto_c * x2[i]
print(producto_c)
```

```
--- Prueba 2 (con 0) ---
Algoritmo A
0
Algoritmo B
0
Algoritmo C
0
```

```python
print("\n--- Conclusion ---")
print("El Algoritmo A (inicia en 0) siempre da 0.")
print("El Algoritmo B (inicia en 1) es el metodo matematicamente correcto.")
print("El Algoritmo C es una optimizacion del B (para si encuentra un 0).")

print("\nRespuesta a la pregunta:")
print("El Algoritmo A solo es 'correcto' si el producto verdadero es 0.")
print("Esto solo pasa si al menos uno de los numeros en la lista es 0.")
```

```
--- Conclusion ---
El Algoritmo A (inicia en 0) siempre da 0.
El Algoritmo B (inicia en 1) es el metodo matematicamente correcto.
El Algoritmo C es una optimizacion del B (para si encuentra un 0).

Respuesta a la pregunta:
El Algoritmo A solo es 'correcto' si el producto verdadero es 0.
Esto solo pasa si al menos uno de los numeros en la lista es 0.
```

5.

a. ¿Cuántas **multiplicaciones y sumas** se requieren para determinar una suma de la forma $\sum_{i=1}^{n} \sum_{j=1}^{i} a_i b_j$?

b. Modifique la suma en la parte a) a un **formato equivalente** que reduzca el número de cálculos.

Pseudocódigo

INICIO

1. Para cada par (i, j) se realiza 1 multiplicación: $a_i \cdot b_j$.
2. Se requieren $(n \cdot l - 1)$ sumas para agregar todos los términos.
3. **Total**: $n \cdot l$ multiplicaciones $+(n \cdot l - 1)$ sumas.

FIN

```
n = 4
a = [1.0, 2.0, 3.0, 4.0]
b = [5.0, 6.0, 7.0, 8.0]

print("--- Parte A: Algoritmo Original (n=4) ---")

multiplicaciones_a = 0
sumas_a = 0
suma_total_a = 0.0

for i in range(n):
    suma_interna = 0.0

    for j in range(i + 1):
        termino = a[i] * b[j]
        multiplicaciones_a = multiplicaciones_a + 1

        suma_interna = suma_interna + termino
        if j > 0:
            sumas_a = sumas_a + 1

    suma_total_a = suma_total_a + suma_interna
    if i > 0:
        sumas_a = sumas_a + 1
```

```python
print("Multiplicaciones (A):", multiplicaciones_a)
print("Sumas (A):", sumas_a)
```

```
--- Parte A: Algoritmo Original (n=4) ---
Multiplicaciones (A): 10
Sumas (A): 9
```

```python
print("--- Parte B: Algoritmo Modificado (n=4) ---")

multiplicaciones_b = 0
sumas_b = 0
suma_total_b = 0.0
suma_parcial_b = 0.0

for i in range(n):

    suma_parcial_b = suma_parcial_b + b[i]
    if i > 0:
        sumas_b = sumas_b + 1

    termino = a[i] * suma_parcial_b
    multiplicaciones_b = multiplicaciones_b + 1

    suma_total_b = suma_total_b + termino
    if i > 0:
        sumas_b = sumas_b + 1

print("Multiplicaciones (B):", multiplicaciones_b)
print("Sumas (B):", sumas_b)
```

```
--- Parte B: Algoritmo Modificado (n=4) ---
Multiplicaciones (B): 4
Sumas (B): 6
```

DISCUSIONES

1. Escriba un algoritmo para sumar la serie finita $\sum_{i=1}^{n} x_i$ en orden inverso.

Pseudocódigo

INICIO

1. Leer lista de números $x = [x_1, x_2, \dots, x_n]$.
2. Inicializar $suma = 0$.
3. Para cada elemento en **orden inverso** (de x_n a x_1):

    a. Sumar elemento al acumulador.

4. Devolver resultado.

FIN

Código

```python
n = 5
x = [100.0, 20.0, 5.0, 1.0, 0.5]

suma_inversa = 0.0

for i in range(n, 0, -1):

    indice = i - 1

    termino = x[indice]

    suma_inversa = suma_inversa + termino

print("Datos:", x)
print("La suma en orden inverso es:")
print(suma_inversa)
```

```
Datos: [100.0, 20.0, 5.0, 1.0, 0.5]
La suma en orden inverso es:
126.5
```

2. Las ecuaciones (1.2) y (1.3) en la sección 1.2 proporcionan formas alternativas para las raíces x_1 y x_2 de $ax^2 + bx + c = 0$. Construya un algoritmo con entrada a, b y c y salida x_1, x_2 que calcule las raíces x_1 y x_2 (que pueden ser iguales o conjugados complejos) mediante la mejor fórmula para cada raíz.

Pseudocódigo

INICIO

// Definir los datos de entrada // n es el número total de elementos // x es la lista de números $[x_1, x_2, \ldots, x_n]$ $n = 5$ $x = [100.0, 20.0, 5.0, 1.0, 0.5]$

// Inicializar la variable para la suma $SUMA = 0.0$

// Iterar en orden inverso, desde $i = n$ hasta $i = 1$ PARA i DESDE n HASTA 1 (bajando) // En una lista 0-indexada, el término x_i está en el índice $(i-1)$ $termino\_actual = x[i-1]$

```
// Añadir el término a la suma total
$SUMA = SUMA + termino\_actual$
```

FIN PARA

MOSTRAR "La lista es:", x MOSTRAR "La suma en orden inverso es:", $SUMA$

FIN

Código

```
import math

a_str = input("Ingrese a: ")
b_str = input("Ingrese b: ")
c_str = input("Ingrese c: ")

a = float(a_str)
b = float(b_str)
c = float(c_str)

discriminante = (b * b) - (4 * a * c)

if discriminante < 0:
```

```
    real = -b / (2 * a)
    imag = math.sqrt(-discriminante) / (2 * a)

    print("Las raices son complejas:")
    print("x1:", real, "+", imag, "i")
    print("x2:", real, "-", imag, "i")

else:

    v = 0.0

    if b >= 0:
        v = -b - math.sqrt(discriminante)
    else:
        v = -b + math.sqrt(discriminante)

    x1 = v / (2 * a)
    x2 = (2 * c) / v

    print("Las raices son reales:")
    print("x1:", x1)
    print("x2:", x2)
```

```
Las raices son reales:
x1: -3.0
x2: -2.0
```

- 3. Suponga que

$$\frac{1-2x}{1-x+x^2}+\frac{2x-4x^3}{1-x^2+x^4}+\frac{4x^3-8x^7}{1-x^4+x^8}+\cdots=\frac{1+2x}{1+x+x^2}$$

para $x < 1$ y si $x = 0.25$. Escriba y ejecute un algoritmo que determine el **número de términos** necesarios en el lado izquierdo de la ecuación de tal forma que el lado izquierdo difiera del lado derecho en menos de 10^{-6}.

Código

```python
x = 0.25
tolerancia = 1e-6

numerador_rhs = 1.0 + (2.0 * x)
denominador_rhs = 1.0 + x + (x * x)
valor_rhs = numerador_rhs / denominador_rhs

suma_lhs = 0.0
n = 0

diferencia = valor_rhs - suma_lhs

num_a = 1.0
num_b = 2.0 * x
den_b = x
den_c = x * x

while abs(diferencia) >= tolerancia:

    numerador_termino = num_a - num_b
    denominador_termino = 1.0 - den_b + den_c
    termino = numerador_termino / denominador_termino

    suma_lhs = suma_lhs + termino

    n = n + 1

    diferencia = valor_rhs - suma_lhs

    num_a = num_b
    num_b = 2.0 * num_b * den_c
    den_b = den_c
    den_c = den_c * den_c

print("Numero de terminos (n) para error < 10^-6:")
print(n)
print("Suma izquierda (LHS):")
print(suma_lhs)
print("Valor derecho (RHS):")
print(valor_rhs)
```

```
Numero de terminos (n) para error < 10^-6:
```

```
4
Suma izquierda (LHS):
1.1428571279559818
Valor derecho (RHS):
1.1428571428571428
```