NOMBRE: Francis Bravo

[Deber 03] Algoritmos y convergencia

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} \left(\frac{1}{i^2}\right))
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} \left(\frac{1}{i^3}\right))
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})

```
def truncate_to_three_digits(value):
    return float(f"{value:.3g}")

def sum_ascending():
    terms = [1 / i**2 for i in range(1, 11)]
    truncated_sum = 0
    for term in terms:
        truncated_sum = truncate_to_three_digits(truncated_sum +
truncate_to_three_digits(term))
    return truncated_sum

def sum_descending():
    terms = [1 / i**2 for i in range(1, 11)][::-1]
    truncated_sum = 0
    for term in terms:
        truncated_sum = truncate_to_three_digits(truncated_sum +
truncate_to_three_digits(term))
    return truncated_sum

ascending_sum = sum_ascending()
descending_sum = sum_descending()

print(f"Suma en orden ascendente (tres dígitos): {ascending_sum}")
print(f"Suma en orden descendente (tres dígitos): {descending_sum}")
```

```
Suma en orden ascendente (tres dígitos): 1.55
Suma en orden descendente (tres dígitos): 1.55
```

```
def sum_ascending_cubic():
    terms = [1 / i**3 for i in range(1, 11)]
    return sum(terms)

def sum_descending_cubic():
    terms = [1 / i**3 for i in range(1, 11)][::-1]
    return sum(terms)
```

```
ascending_sum_cubic = sum_ascending_cubic()
descending_sum_cubic = sum_descending_cubic()

print(f"Suma en orden ascendente: {ascending_sum_cubic}")
print(f"Suma en orden descendente: {descending_sum_cubic}")
```

```
Suma en orden ascendente: 1.1975319856741933
Suma en orden descendente: 1.1975319856741933
```

1. Serie de Maclaurin para la función arcotangente

La serie de Maclaurin para la función arcotangente converge para (-1 < x \leq 1) y está dada por:

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

a. Determinación del número de términos para alcanzar una precisión deseada

Utilice el hecho de que (\tan\left(\frac{\pi}{4} \right) = 1) para determinar el número (n) de términos de la serie que se necesita sumar para garantizar que:

$$\left| 4\, P_n(1) - \pi \right| < 10^{-3}$$

```
import math

def maclaurin_arctan_approx(n):
    suma = 0
    for i in range(1, n + 1):
        suma += (-1)**(i + 1) / (2 * i - 1)
    return suma

n_terms = 0
precision = 0.001
difference = 1

while difference >= precision:
    n_terms += 1
    current_approximation = 4 * maclaurin_arctan_approx(n_terms)
    difference = abs(current_approximation - math.pi)

print(f"Número de términos necesarios para alcanzar la precisión de
10^-3: {n_terms}")
```

```
Número de términos necesarios para alcanzar la precisión de 10^-3:
1000
```

b. El lenguaje de programación C++ requiere que el valor de (π) se encuentre dentro de (10^{-6}). ¿Cuántos términos de la serie se necesitarían sumar para obtener este grado de precisión?

```
import math

def calcular_pi(precision_deseada):
    tolerancia = 0.5 * 10 ** (-precision_deseada)
    suma = 0.0
    n = 0
    while True:
        termino = (-1)**n / (2 * n + 1)
        suma += termino
        aproximacion = 4 * suma
        error = abs(aproximacion - math.pi)
        if error < tolerancia:
            break
        n += 1
    return n + 1, aproximacion

precision_deseada = 6
n_terminos, aproximacion = calcular_pi(precision_deseada)
print(f"Número de términos necesarios para alcanzar la precisión de
10^-6: {n_terminos}")
```

```
Número de términos necesarios para alcanzar la precisión de 10^-6:
2000001
```

3. Aproximación de π mediante otra identidad

Otra fórmula para calcular π se puede deducir a partir de la siguiente identidad:

$$\frac{\pi}{4} = 4\arctan\left(\frac{1}{5}\right) - \arctan\left(\frac{1}{239}\right)$$

Determine el número de términos que se deben sumar para garantizar una aproximación de π con una precisión de 10^{-3}.

```
def maclaurin_arctan(x, terms):
    suma = 0
    for i in range(terms):
        termino = (-1)**i * (x**(2 * i + 1) / (2 * i + 1))
        suma += termino
    return suma

def calculate_pi_from_identity(terms):
    arctan_1_5 = maclaurin_arctan(1 / 5, terms)
    arctan_1_239 = maclaurin_arctan(1 / 239, terms)
    return 4 * (4 * arctan_1_5 - arctan_1_239)

precision_threshold = 0.001
```

```
terms_used = 0
pi_approximation_error = 1

while pi_approximation_error >= precision_threshold:
    terms_used += 1
    current_pi_value = calculate_pi_from_identity(terms_used)
    pi_approximation_error = abs(current_pi_value - math.pi)

print("Número de términos necesarios para obtener precisión de 10^-
3:", terms_used)
```

```
Número de términos necesarios para obtener precisión de 10^-3: 2
```

5. Complejidad de la suma doble

a. Número de operaciones necesarias

¿Cuántas multiplicaciones y sumas se requieren para determinar una suma de la forma:

$$\sum_{i=1}^{n}\sum_{j=1}^{i}\left(a_i b_j\right)$$

```
def count_multiplications(n):
    total_multiplicaciones = 0
    for i in range(1, n + 1):
        total_multiplicaciones += i  # Sumar el número de
multiplicaciones para cada i
    return total_multiplicaciones

# Usar la variable 'n' ya definida en el notebook
multiplicaciones = count_multiplications(n)
print("Para n={}, se realizan {} multiplicaciones.".format(n,
multiplicaciones))
```

```
Para n=5, se realizan 15 multiplicaciones.
```

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

```
def count_operations(n):
    total_multiplicaciones = sum(range(1, n + 1))  # Suma directa de 1
a n
    total_sumas = sum(range(n))  # Suma directa de 0 a n-1
    return total_multiplicaciones, total_sumas

multiplicaciones, sumas = count_operations(n)
print(f"Para n={n}, se realizan {multiplicaciones} multiplicaciones y
{sumas} sumas.")
```

```
Para n=5, se realizan 15 multiplicaciones y 10 sumas.
```

DISCUSIONES

Algoritmo para el cálculo de las raíces de una ecuación cuadrática

Las ecuaciones (1.2) y (1.3) en la sección 1.2 proporcionan formas alternativas para las raíces x_1 y x_2 de la ecuación:

$$a\,x^2+b\,x+c=0$$

Construya un algoritmo con entrada $a\,,b\,,c$ y salida x_1, x_2 que calcule las raíces x_1 y x_2 (que pueden ser reales, iguales, o conjugados complejos) utilizando la mejor fórmula para cada raíz.

```
import cmath

def calcular_raices(a, b, c):
    """
    Calcula las raíces de una ecuación cuadrática de la forma ax^2 +
bx + c = 0.
    Maneja raíces reales y complejas.

    Args:
        a (float): Coeficiente cuadrático.
        b (float): Coeficiente lineal.
        c (float): Término independiente.

    Returns:
        tuple: Raíces de la ecuación (x1, x2).
    """
    discriminante = b**2 - 4 * a * c

    if discriminante > 0:
        # Raíces reales y distintas
        raiz_discriminante = discriminante**0.5
        if b > 0:
            x1 = (-b - raiz_discriminante) / (2 * a)
        else:
            x1 = (-b + raiz_discriminante) / (2 * a)
        x2 = c / (a * x1)
    elif discriminante == 0:
        # Raíces reales e iguales
        x1 = x2 = -b / (2 * a)
    else:
        # Raíces complejas
        raiz_discriminante = cmath.sqrt(discriminante)
        x1 = (-b + raiz_discriminante) / (2 * a)
        x2 = (-b - raiz_discriminante) / (2 * a)

    return x1, x2

# Las variables 'a', 'b', y 'c' ya están definidas en el notebook
```

```
raiz1, raiz2 = calcular_raices(a, b, c)
print(f"Las raíces de la ecuación cuadrática son: x1 = {raiz1}, x2 = {raiz2}")
```

```
Las raíces de la ecuación cuadrática son: x1 = 2.0, x2 = 1.0
```

3. Determinación del número de términos para una aproximación precisa

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

```
def calcular_lado_derecho(x):
    """
    Calcula el valor del lado derecho de la ecuación.

    Args:
        x (float): Valor de x.

    Returns:
        float: Resultado del lado derecho.
    """
    return (1 + 2 * x) / (1 + x + x**2)

def calcular_serie(x, tolerancia):
    """
    Calcula el número de términos necesarios para que la suma de la serie
    en el lado izquierdo se aproxime al valor del lado derecho con una
    tolerancia dada.

    Args:
        x (float): Valor de x.
        tolerancia (float): Tolerancia deseada para la aproximación.

    Returns:
        tuple: Número de términos, suma del lado izquierdo y valor del lado derecho.
    """
    suma = 0.0
    n = 1
    valor_derecho = calcular_lado_derecho(x)
```

```
    # Variables auxiliares para calcular los términos de la serie
    A = 1  # Coeficiente inicial
    y = x
    x_inv = 1 / x

    while True:
        # Cálculo del término actual de la serie
        numerador = A * (y * x_inv) * (1 - 2 * y)
        denominador = 1 - y + y**2
        termino = numerador / denominador

        # Actualización de la suma y cálculo del error
        suma += termino
        error = abs(suma - valor_derecho)

        # Verificación de criterios de parada
        if error < tolerancia or abs(termino) < tolerancia:
            break

        # Actualización de variables para el siguiente término
        A *= 2
        y **= 2
        n += 1

    return n, suma, valor_derecho

# Parámetros de entrada
tolerancia = 1e-6

# Cálculo de la serie
n_terminos, suma_izquierda, valor_derecho = calcular_serie(x,
tolerancia)

# Resultados
print(f"Número de términos necesarios: {n_terminos}")
print(f"Suma de la serie (lado izquierdo) con {n_terminos} términos:
{suma_izquierda}")
print(f"Valor del lado derecho: {valor_derecho}")
print(f"Diferencia: {abs(suma_izquierda - valor_derecho)}")
```

```
Número de términos necesarios: 4
Suma de la serie (lado izquierdo) con 4 términos: 1.1428571279559818
Valor del lado derecho: 1.1428571428571428
Diferencia: 1.4901160971803051e-08
```

Link de repositorio de GitHub

https://github.com/Francis1918/DeberesMetodosNumericos.git