ESCUELA POLITÉCNICA NACIONAL	Tarea No.
Metodos Numericos – Computación	**3**
NOMBRE: Ivonne Carolina Ayala	

Algoritmos y convergencia

1.3.2. 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:

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

a. Determinación del número de términos para alcanzar una precisión

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:

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

```
import math

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

n_terms = 0
precision = 1e-3
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
    signo = 1
    while True:
        termino = signo / (2 * n + 1)
        suma += termino
        aproximacion = 4 * suma
        error = abs(aproximacion - math.pi)
        if error < tolerancia:
            break
        signo *= -1
        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):
    return sum((-1)**i * (x**(2 * i + 1) / (2 * i + 1)) for i in range(terms))

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

precision_threshold = 1e-3
terms_used = 0
pi_approximation_error = 1

# Cálculo iterativo
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(f"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:

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

```
def count_multiplications(n):
    total_multiplicaciones = 0
    for i in range(1, n + 1):
        total_multiplicaciones += i
    return total_multiplicaciones

n = 5
multiplicaciones = count_multiplications(n)
print(f"Para n={n}, se realizan {multiplicaciones} 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_sumas = 0
    total_multiplicaciones = 0

    for i in range(1, n + 1):
        total_multiplicaciones += i
        total_sumas += (i - 1)

    return total_multiplicaciones, total_sumas

n = 5
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:

$$ax^2 + bx + c = 0$$

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):
    discriminante = b**2 - 4 * a * c

    if discriminante > 0:
        if b > 0:
            x1 = (-b - discriminante**0.5) / (2 * a)
        else:
            x1 = (-b + discriminante**0.5) / (2 * a)
        x2 = c / (a * x1)
    elif discriminante == 0:
        x1 = x2 = -b / (2 * a)
    else:
        raiz_discriminante = cmath.sqrt(discriminante)
        x1 = (-b + raiz_discriminante) / (2 * a)
        x2 = (-b - raiz_discriminante) / (2 * a)

    return x1, x2

a, b, c = 1, -3, 2
raiz1, raiz2 = calcular_raices(a, b, c)
print(f"Las raíces son: x1 = {raiz1}, x2 = {raiz2}")
```

```
Las raíces 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):
    return (1 + 2 * x) / (1 + x + x**2)

def calcular_serie(x, tolerancia):
    suma = 0.0
    n = 1
    valor_derecho = calcular_lado_derecho(x)

    A_prev = 1
    y_prev = x
    x_inv = 1 / x

    while True:
        if n == 1:
            A = A_prev
            y = y_prev
        else:
            A = 2 * A_prev
            y = y_prev ** 2
            A_prev = A
            y_prev = y

        numerador = A * (y * x_inv) * (1 - 2 * y)
        denominador = 1 - y + y * y
        termino = numerador / denominador

        suma += termino

        error = abs(suma - valor_derecho)
        if error < tolerancia:
            break
        n += 1
        if abs(termino) < tolerancia:
            break
    return n, suma, valor_derecho

x = 0.25
tolerancia = 1e-6

n_terminos, suma_izquierda, valor_derecho = calcular_serie(x, tolerancia)
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
```

Declaración de Uso de Inteligencia Artificial

Este documento fue creado con la ayuda de ChatGPT, diseñada para facilitar la comprensión de código y mejorar la eficiencia de desarrollo. La IA fue utilizada para:

1. **Entender el código**
2. **Optimización del formato**
3. **Sugerencias para mejoras**

Link al repositorio

https://github.com/carol230/MetodosNumericos