Deber Metodos Numericos Parte 2

Jose Armando

Tabla de Contenidos

Discuciones

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

```
import math
```

```
def sumar_serie_decreciente(x):
    suma = 0

    # Iterar desde x hasta 1
    for i in range(x, 0, -1):
        suma += i

    return suma

# Ejemplo
x = 7
resultado = sumar_serie_decreciente(x)
print("La suma de la serie desde", x, "hasta 1 es:", resultado)
```

```
La suma de la serie desde 7 hasta 1 es: 28
```

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, 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.

```python
import cmath
def calcular_raices(a, b, c):

    # Calcular el discriminante
    discriminante = b**2 - 4*a*c

    # Caso 1: Raíces reales distintas
    if discriminante > 0:
        x1 = (-b + cmath.sqrt(discriminante)) / (2*a)
        x2 = (-b - cmath.sqrt(discriminante)) / (2*a)

    # Caso 2: Raíces reales iguales
    elif discriminante == 0:
        x1 = x2 = -b / (2*a)

    # Caso 3: Raíces complejas conjugadas
    else:
        discriminante_sqrt = cmath.sqrt(-discriminante)
        x1 = (-b + discriminante_sqrt) / (2*a)
        x2 = (-b - discriminante_sqrt) / (2*a)

    return x1, x2

# Ejemplo de uso
a = 1
b = -5
c = 6

x1, x2 = calcular_raices(a, b, c)
print(f"Las raíces son: x1 = {x1}, x2 = {x2}")
```

```
Las raíces son: x1 = (3+0j), x2 = (2+0j)
```

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}).

```python
def term(x, n):
    try:
        numerator = float(2**(n-1) * x**(2**(n-1)) - 2**n * x**(2**n))
```

```
        denominator = float(1 - x**(2**n) + x**(2**(n+1)))
        return numerator / denominator
    except OverflowError:
        return 0  # Consideramos que términos muy grandes son insignificantes

def left_side_sum(x, tolerance):
    n = 1
    left_sum = 0
    right_value = (1 + 2 * x) / (1 + x + x**2)

    while True:
        current_term = term(x, n)
        if current_term == 0:
            break
        left_sum += current_term
        if abs(left_sum - right_value) < tolerance:
            break
        n += 1

    return n, left_sum

x = 0.25
tolerance = 10**(-6)
n_terms, left_sum = left_side_sum(x, tolerance)

print(f"Numero de terminos necesarios: {n_terms}")
print(f"Suma del lado izquierdo con{n_terms} terminos: {left_sum}")
print(f"Valor del lado derecho: {(1 + 2 * x) / (1 + x + x**2)}")
```

```
Numero de terminos necesarios: 11
Suma del lado izquierdo con11 terminos: 0.2582075590925407
Valor del lado derecho: 1.1428571428571428
```