Escuela Politécnica Nacional

Metodos Numericos - Preparación visita 01

Quilumba Morocho Joel Patricio

2024-05-21

Tabla de contenidos

1 Conjunto De Ejercicios 1.3

1.1 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. $\sum_{i=1}^{10}\left(\frac{1}{i^2}\right)$ primero por $\left(\frac{1}{1}+\frac{1}{4}+....\frac{1}{100}\right)$ y luego por $\left(\frac{1}{100}+\frac{1}{81}+....\frac{1}{1}\right)$

```
def SumatoriaAcendente(num):
    res = 0
    for i in range(10):
        res += 1 / ((i+1)**num)
    return round(res,5)
```

```
def SumatoriaDecendente(num):
    res = 0
    for i in range(10, 0, -1):
        res += 1 / (i**num)
    return round(res,5)
```

```
## literal A
print("Resultado 1: ",SumatoriaAcendente(2))
print("Resultado 2: ",SumatoriaDecendente(2))
```

```
Resultado 1:  1.54977
Resultado 2:  1.54977
```

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

```
## literal B
print("Resultado 1: ",SumatoriaAcendente(3))
print("Resultado 2: ",SumatoriaDecendente(3))
```

```
Resultado 1:  1.19753
Resultado 2:  1.19753
```

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

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

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

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

```
import math
pi = math.pi
def arctan_series(x, n):
    result = 0
    for i in range(1, n+1):
        result += ((-1)**(i+1) * (x**(2*i-1))) / (2*i-1)
    return result
n = 1
while True:
    approx_pi = 4 * arctan_series(1, n)
    if abs(approx_pi - pi) < 10**(-3):
        break
    n += 1
print("Número de términos necesarios para garantizar |4P_n(1) -  | < 10^-3:", n)
```

```
Número de términos necesarios para garantizar |4P_n(1) -  | < 10^-3: 1000
```

1.3 3. Otra fórmula para calcular π se puede deducir a partir de la identidad $\frac{\pi}{4} = 4arctan\frac{1}{5} - arctan\frac{1}{239}$

Determine el número de términos que se deben sumar para garantizar una aproximación dentro de 10^-3

```
import math
def calculate_pi_epsilon(epsilon):
    n = 1
    pi_approx = 0
    term_1 = 1 / 5
    term_2 = 1 / 239

    while abs(pi_approx - math.pi) >= epsilon:
        pi_approx = 4 * (4 * sum_series_maclaurin(1 / 5, n) - sum_series_maclaurin(1 / 239, n
        n += 1

    return n - 1

def sum_series_maclaurin(x, terms):
    total_sum = 0
    for i in range(1, terms + 1):
        total_sum += (-1)**(i + 1) * x**(2 * i - 1) / (2 * i - 1)
    return total_sum

n_terms = calculate_pi_epsilon(1e-3)
print(f"El numero de terminos es : {n_terms}")
```

```
El numero de terminos es : 2
```

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

- a. ENTRADA $n, x1, x2, \cdots, xn$.
  SALIDA PRODUCT.
  Paso 1 Determine PRODUCT = 0.
  Paso 2 Para $i = 1, 2, \cdots, n$ haga
  Determine PRODUCT = PRODUCT $* xi$.
  Paso 3 SALIDA PRODUCT;
  PARE.

```
def algoritmo_a (num):
    resultado=0
    for i in range(num,1):
        resultado*=i
    return resultado

print("Resultado :",algoritmo_a(5))
```

```
Resultado : 0
```

- b. ENTRADA $n, x1, x2, \cdots, xn$.
  SALIDA PRODUCT.
  Paso 1 Determine PRODUCT = 1.
  Paso 2 Para $i = 1, 2, \cdots, n$ haga
  Set PRODUCT = PRODUCT $*xi$.
  Paso 3 SALIDA PRODUCT;
  PARE.

```
def algoritmo_b(num):
    resultado=1
    for i in range(num,1):
        resultado*=i
    return resultado
print("Resultado :",algoritmo_b(5))
```

```
Resultado : 1
```

- c. ENTRADA $n, x1, x2, \cdots, xn$.
  SALIDA PRODUCT.
  Paso 1 Determine PRODUCT = 1.
  Paso 2 Para $i = 1, 2, \cdots, n$ haga

si $xi = 0$ entonces determine PRODUCT = 0;
SALIDA PRODUCT;
PARE
Determine PRODUCT = PRODUCT $*xi$.
Paso 3 SALIDA PRODUCT;
PARE.

```
def algoritmo_c(num):
    resultado=1
    if num==0:
        return 0
    else:
        for i in range(num,1):
            resultado*=i
    return resultado

print("Resultado :",algoritmo_c(5))
```

```
Resultado : 1
```

Algoritmo a es incorrecto para cualquier propósito práctico de calcular un producto denúmeros. Algoritmo b es correcto y calcula el producto de una lista de números correctamente. Algoritmo c es también correcto y más eficiente en presencia de ceros, ya que termina temprano si encuentra un 0.

1.5 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 cálculo

```
def operaciones_para_suma_original(n):
    multiplicaciones = n * (n + 1) // 2
    sumas = n * (n + 1) // 2
    return multiplicaciones, sumas

def operaciones_para_suma_modificada(n):
    multiplicaciones = n * (n + 1) // 2
    sumas = n
```

```python
    return multiplicaciones, sumas

n = 5

# Literal a
mult_orig, sumas_orig = operaciones_para_suma_original(n)
print("Para la suma original:")
print("Total de multiplicaciones necesarias:", mult_orig)
print("Total de sumas necesarias:", sumas_orig)

# Literal b
mult_modif, sumas_modif = operaciones_para_suma_modificada(n)
print("\nPara la suma modificada:")
print("Total de multiplicaciones necesarias:", mult_modif)
print("Total de sumas necesarias:", sumas_modif)
```

```
Para la suma original:
Total de multiplicaciones necesarias: 15
Total de sumas necesarias: 15

Para la suma modificada:
Total de multiplicaciones necesarias: 15
Total de sumas necesarias: 5
```

Es decir que - Para las multiplicaciones:
Para i=1, tenemos 1 multiplicaciones : $a_1 b_1$
Para i=2, tenemos 2 multiplicaciones: $a_2 b_1 a_2 b_2$
Para i=n, tenemos n multiplicaciones: $a_n b_1, a_n b_2 ... a_n b_n$
Por tanto, se requieren $\frac{n(n+1)}{2}$ multiplicaciones

- Para las Sumas:
  Para i=1, no hay simas adicionales porque solo hay un término.
  Para i=2, se suma dos términos lo que implica 1 suma
  Para i=n, se suma tres términos lo que implica 2 sumas
  Por tanto, se requieren $\frac{n(n-1)}{2}$ multiplicaciones

2 Discusiones

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

```
def Algo1(num):
    res=0
    for i in range(num,0,-1):
        res+=i
    return res
```

2.2 2. Las ecuaciones (1.2) y (1.3) en la sección 1.2 proporcionan formas alternativas para las raíces 1 y 2 de ^2 + + = 0. Construya un algoritmo con entrada , , c y salida 1, 2 que calcule las raíces 1 y 2 que pueden ser iguales con conjugados complejos) mediante la mejor fórmula para cada raíz.

```
import cmath

def calcular_raices(a, b, c):
    # Calcula el discriminante
    discriminante = b**2 - 4*a*c

    # Calcula las raíces
    if discriminante > 0:
        raiz1 = (-b + discriminante**0.5) / (2*a)
        raiz2 = (-b - discriminante**0.5) / (2*a)
    elif discriminante == 0:
        raiz1 = raiz2 = -b / (2*a)
    else:
        parte_real = -b / (2*a)
        parte_imaginaria = cmath.sqrt(abs(discriminante)) / (2*a)
        raiz1 = parte_real + parte_imaginaria * 1j
        raiz2 = parte_real - parte_imaginaria * 1j

    return raiz1, raiz2
```

```
raiz1, raiz2 = calcular_raices(1, -3, 2)
print("Raíz 1:", raiz1)
print("Raíz 2:", raiz2)
```

```
Raíz 1: 2.0
Raíz 2: 1.0
```

2.3 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 < 1 y si $= 0.25$. 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 izquierd difiera del lado derecho en menos de 10−. .

```python
def series_difference(x, epsilon):
    n = 1
    left_side = (1 - 2 * x) / (1 - x + x**2)
    right_side = (1 + 2 * x) / (1 + x + x**2)
    term = (2 * x) / (1 - x + x**2)
    difference = abs(left_side - right_side)

    while difference >= epsilon:
        left_side += term
        term *= 2 * x
        n += 1
        difference = abs(left_side - right_side)

    return n

x = 0.25
epsilon = 1e-6
n_terms = series_difference(x, epsilon)
print(f"Number of terms needed: {n_terms}")
```