Tarea 3 ALGORITMOS Y CONVERGENCIA

Pasquel Johann

Tabla de Contenidos

GITHUB

https://github.com/Vladimirjon/MetodosNumericos_PasquelJohann

CONJUNTO DE EJERCICIOS 1.3

2. La Serie de Maclaurin para la función arcotangente converge para $-1 < x <= 1$ 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 $\tan \frac{\pi}{4} = 1$ para determinar el número n de términos de la serie que se necesita sumar para garantizar que $|4P_n(1) - \pi| < 10^{-3}$

$4 \cdot \arctan(1) = \pi$

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

$\arctan(1) = \sum_{i=1}^{n} (-1)^{i+1} \frac{1}{2i-1}$

$\pi \approx 4 \sum_{i=1}^{n} (-1)^{i+1} \frac{1}{2i-1}$

```python
import math

pi_exact = math.pi

n = 1
approximation = 0
tolerance = 10**-3
error = float('inf')

while error >= tolerance:
    term = (-1)**(n+1) / (2 * n - 1)
    approximation += term
    pi_approx = 4 * approximation
    error = abs(pi_approx - pi_exact)
    n += 1

print("Number of terms (n):", n - 1)
print("Approximation of pi:", pi_approx)
print("Actual value of pi:", pi_exact)
print("Absolute error:", error)
```

```
Number of terms (n): 1000
Approximation of pi: 3.140592653839794
Actual value of pi: 3.141592653589793
```

```
Absolute error: 0.000999999749998981
```

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?

$4 \cdot \arctan(1) = \pi$

$\arctan(1) = \sum_{i=1}^{n} (-1)^{i+1} \frac{1}{2i-1}$

$\pi \approx 4 \sum_{i=1}^{n} (-1)^{i+1} \frac{1}{2i-1}$

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

```python
import math

pi_exact = math.pi

n = 1
approximation = 0
tolerance = 10**-10
error = float('inf')
max_iterations = 100000

while error >= tolerance and n <= max_iterations:
    term = (-1)**(n+1) / (2 * n - 1)
    approximation += term
    pi_approx = 4 * approximation
    error = abs(pi_approx - pi_exact)
    n += 1

if n > max_iterations:
    print("Reached maximum iterations before meeting tolerance.")
else:
    print("Tolerance met.")

print("Number of terms (n):", n - 1)
print("Approximation of pi:", pi_approx)
print("Actual value of pi:", pi_exact)
print("Absolute error:", error)
```

```
Reached maximum iterations before meeting tolerance.
Number of terms (n): 100000
```

```
Approximation of pi: 3.1415826535897198
Actual value of pi: 3.141592653589793
Absolute error: 1.0000000073340232e-05
```

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

$$\pi \approx 16\sum_{i=1}^{n}(-1)^{i+1}\frac{\left(\frac{1}{5}\right)^{2i-1}}{2i-1} - 4\sum_{i=1}^{n}(-1)^{i+1}\frac{\left(\frac{1}{239}\right)^{2i-1}}{2i-1}$$

```python
import math

pi_exact = math.pi

tolerance = 10**-3
error = float('inf')
n = 1
arctan_1_5_approx = 0
arctan_1_239_approx = 0

while error >= tolerance:
    term_1_5 = (-1)**(n+1) * (1/5)**(2 * n - 1) / (2 * n - 1)
    arctan_1_5_approx += term_1_5

    term_1_239 = (-1)**(n+1) * (1/239)**(2 * n - 1) / (2 * n - 1)
    arctan_1_239_approx += term_1_239

    pi_approx = 16 * arctan_1_5_approx - 4 * arctan_1_239_approx

    error = abs(pi_approx - pi_exact)

    n += 1

print("Number of terms (n):", n - 1)
print("Approximation of pi:", pi_approx)
print("Actual value of pi:", pi_exact)
print("Absolute error:", error)
```

```
Number of terms (n): 2
Approximation of pi: 3.1405970293260603
```

```
Actual value of pi: 3.141592653589793
Absolute error: 0.0009956242637327861
```

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

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

- Multiplicaciones

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

- Sumas

$$\sum_{i=1}^{n}(i-1) = \sum_{i=1}^{n} i - \sum_{i=1}^{n} 1 = \frac{n(n+1)}{2} - n = \frac{n(n-1)}{2}$$

Total Multiplicaciones: $\frac{n(n+1)}{2}$

Total Sumas: $\frac{n(n-1)}{2}$

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

Se puede escribir:

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

Optimización

$\sum_{j=1}^{i} b_j$
Multiplicamos para cada a_i necesitando n multiplicaciones

Total de Operaciones

- **Multiplicaciones**: n
- **Sumas**: $2n-2$

DISCUSIONES

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 con conjugados complejos) mediante la mejor fórmula para cada raíz.

```python
import cmath

def calcular_raices(a, b, c):
    D = b**2 - 4 * a * c
    raiz_D = cmath.sqrt(D)

    if b >= 0:
        x1 = (-b - raiz_D) / (2 * a)
        x2 = (2 * c) / (-b - raiz_D)
    else:
        x1 = (-b + raiz_D) / (2 * a)
        x2 = (2 * c) / (-b + raiz_D)

    return x1, x2

a = float(input("Introduce a: "))
b = float(input("Introduce b: "))
c = float(input("Introduce c: "))

x1, x2 = calcular_raices(a, b, c)
print("Raíz x1:", x1)
print("Raíz x2:", x2)
```

```
Introduce a:  2
Introduce b:  4
Introduce c:  -9
```

```
Raíz x1: (-3.345207879911715+0j)
Raíz x2: (1.3452078799117146-0j)
```

ALGORITMOS

Algoritmo 01

Pseudocódigo

ENTRADA $N, x_1, x_2, ..., x_n$

SALIDA

Paso 1

Tome $SUM = 0$ (Inicialize el acumulador.)

Paso 2

Para $i = 1, 2, ...N$ hacer

Tome $SUM = SUM + x_i$ (Añadir el siguiente término.)

Paso 3

$SALIDA(SUM)$

$PARE$

```python
def suma(n, x):
    sumatoria = 0
    for i in range(n):
        sumatoria += x[i]
    return sumatoria

valores = [3, 5, 7, 10]
n = len(valores)
resultado = suma(n, valores)
print("La suma es:", resultado)
```

```
La suma es: 25
```

Algoritmo 02

- ARRAY (a[0..n])
- FOR (i = 0..n)
  - swapped = false
  - FOR (j = 1..n-i)
    - IF (a[j] < a[j-1])
      - swap(a, j, j-1)
      - swapped = true
    - END-IF
  - END-FOR
  - IF ($\neg$)swapped
    - break
  - END-IF
- END-FOR

```
def algoritmo2(array):
    n = len(array)
    for i in range(n):
        swapped = False
        for j in range (1,n-i):
            if array[j] < array[j-1]:
                array[j],array[j-1] = array[j-1],array[j]
                swapped = True
        if not swapped:
            break

ejemplo = [8,6,4,3,6,9]
algoritmo2(ejemplo)
ejemplo
```

```
[3, 4, 6, 6, 8, 9]
```

Algoritmo 03

Pseudocódigo:

````
procedure iterative (n: nonnegative integer)
    if n = 0 then
        return 0
    else
        x := 0
        y := 1
        for i := 1 to n - 1
            z := x + y
            x := y
            y := z
        return y

::: {.cell execution_count=12}
``` {.python .cell-code}
def iterativo(n):
    if n == 0:
        return 0
    else:
        x = 0
        y = 1
        i = 1
        while i <= n - 1:
            z = x + y
            x = y
            y = z
            i += 1
        return y

n = int(input("Introduce un número entero no negativo: "))
resultado = iterativo(n)
print("El número de Fibonacci es:", resultado)
````

```
Introduce un número entero no negativo:  15
```

```
El número de Fibonacci es: 610
```

:::

Algoritmo 04

¿A qué valor converge?
```

$$\sum_{n=1}^{\infty} \frac{1}{n}$$

```python
def serie_armonica(n):
    suma = 0
    for i in range(1, n + 1):
        suma += 1 / i
    return suma

n = int(input("Introduce el número de términos a sumar: "))
resultado = serie_armonica(n)
print(f"La suma de los primeros {n} términos es: {resultado}")
```

```
Introduce el número de términos a sumar:  5
```

```
La suma de los primeros 5 términos es: 2.283333333333333
```