Ejercicios: Algoritmos y convergencia

David Morales

2. La serie de Macalurin para la función arcotangente converge para $-1 < x \leq 1$ y está dada por $\arctan x = \lim_{n\to\infty} P_n(x) = \lim_{n\to\infty} \sum_{i=1}^{n} (-1)^{i+1} \frac{x^{2i-1}}{2i-1}$

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

Partiendo de $\tan \frac{\pi}{4} = 1$, obtenemos: $\arctan(1) = \frac{pi}{4}$

Por Maclaurin:

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

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

Se da valores a n para garantizar

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

```python
import math

def calcular_n():
    n = 0
    suma = 0.0
    error = float('inf')
    pi_real = math.pi
    while error >= 1e-3:
        n += 1
        termino = (-1)**(n+1) / (2*n - 1)
        suma += termino
        pi_aproximado = 4 * suma
        error = abs(pi_real - pi_aproximado)
    return n, pi_aproximado, error

n, pi_aproximado, error = calcular_n()
print(f"Número mínimo de términos necesarios: {n}")
```

```
print(f"Valor aproximado de  : {pi_aproximado}")
print(f"Error absoluto: {error}")
```

```
Número mínimo de términos necesarios: 1000
Valor aproximado de  : 3.140592653839794
Error absoluto: 0.000999999749998981
```

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

A partir del enunciado anterior | $4 * \sum_{i=1}^{n} (-1)^{i+1} \frac{1}{2i-1} = \pi$

```
import math

def calcular_pi(n):
    suma = 0.0
    for i in range(n):
        termino = (-1) ** i * (1.0 / (2 * i + 1))
        suma += termino
    return 4.0 * suma

def contar_decimales_correctos(aproximacion, pi_exacto):
    str_aprox = f"{aproximacion:.15f}"
    str_pi = f"{pi_exacto:.15f}"
    contador = 0
    for a, b in zip(str_aprox[2:], str_pi[2:]):
        if a == b:
            contador += 1
        else:
            break
    return contador

precision_deseada = 6
pi_exacto = math.pi
n = 1

while True:
    aproximacion = calcular_pi(n)
    decimales_correctos = contar_decimales_correctos(aproximacion, pi_exacto)
    if decimales_correctos >= precision_deseada:
        break
    n += 1
```

```
print(f"Se necesitaron {n} términos para obtener {precision_deseada} decimales correctos.")
print(f"Valor aproximado de  : {aproximacion:.15f}")
print(f"Valor exacto de  :    {pi_exacto:.15f}")
```

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

Despejando π -> $\pi = 4 * (4 * \arctan\frac{1}{5} - \arctan\frac{1}{239})$

El valor de la arcotangente se obtiene a partir de Maclaurin

```
import math

def calcular_pi_aproximado(terminos):
    suma1 = sum((-1)**k / (5**(2*k + 1) * (2*k + 1)) for k in range(terminos))
    suma2 = sum((-1)**k / (239**(2*k + 1) * (2*k + 1)) for k in range(terminos))
    return 4 * (4 * suma1 - suma2)

def contar_decimales_correctos(aprox, valor_exacto, decimales):
    aprox_str = f"{aprox:.{decimales + 1}f}"
    exacto_str = f"{valor_exacto:.{decimales + 1}f}"
    coincidencias = sum(1 for i, j in zip(aprox_str, exacto_str) if i == j)
    return coincidencias - 1  # Para ignorar el punto decimal

pi_real = math.pi
decimales_deseados = 3
iteracion = 1

while True:
    pi_aprox = calcular_pi_aproximado(iteracion)
    coincidencias = contar_decimales_correctos(pi_aprox, pi_real, decimales_deseados)
    if coincidencias >= decimales_deseados:
        break
    iteracion += 1

print(f"Se necesitaron {iteracion} términos para obtener {decimales_deseados} decimales corre
```

```
Se necesitaron 2 términos para obtener 3 decimales correctos.
```

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

Para un valor dado de i, j toma valores de 1 hasta i, por lo que se realizan i multiplicaciones, dando que en total se realizarán $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$, siendo esta la cantidad de veces que se realiza la multiplicación.

```
n = 3
multiplicaciones = 0
for i in range(1, n+1):
    for j in range(1, i+1):
        multiplicacion = i * j
        multiplicaciones += 1

print(f'La multiplicación se ha realizado {multiplicaciones} veces')
```

```
La multiplicación se ha realizado 6 veces
```

Discuciones

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

```
import cmath

def calcular_raices_convergencia(a, b, c, tolerancia=1e-12):
    D = b**2 - 4 * a * c
    sqrt_D = cmath.sqrt(D)

    if abs(D) < tolerancia:
        x1 = -b / (2 * a)
        x2 = x1
    elif b >= 0:
        x1 = (-b - sqrt_D) / (2 * a)
        x2 = (2 * c) / (-b - sqrt_D)
    else:
        x1 = (-b + sqrt_D) / (2 * a)
        x2 = (2 * c) / (-b + sqrt_D)

    return x1, x2

a = 2
```

```
b = 3
c = 1
x1, x2 = calcular_raices_convergencia(a, b, c)
print("Las raíces son:")
print("x1 =", x1)
print("x2 =", x2)
```

```
Las raíces son:
x1 = (-1+0j)
x2 = (-0.5-0j)
```

GitHub: git@github.com:DavidME1604/MetodosNumericos2024B_MoralesDavid.git