Tarea 2

Danny Iñaguazo

Tabla de Contenidos

Ejercicios

1. Aritmética de corte

Utilice aritmética de corte de tres dígitos para calcular las siguientes sumas. Para cada parte, ¿qué método es más preciso y por qué?

Literal a)

El siguiente código en python produce la suma exacta de esta función.

$$\sum_{i=1}^{10} \frac{1}{i^2}$$

```
sum = 0
for i in range (1, 11):
    sum += 1 / (i ** 2)

print(sum)
```

```
1.5497677311665408
```

Ahora, el código siguiente se encarga de redondear a 3 dígitos cada vez que se itera:

```
sum = 0
for i in range (1, 11):
    sum += (1 * 100) // (i ** 2) / 100

print(sum)
```

```
1.5300000000000002
```

Literal b)

El siguiente código en python produce la suma exacta de esta función.

$$\sum_{i=1}^{10} \frac{1}{i^3}$$

```
sum = 0
for i in range (1, 11):
    sum += 1 / (i ** 3)

print(sum)
```

```
1.197531985674193
```

Ahora, el código siguiente se encarga de redondear a 3 dígitos cada vez que se itera:

```
sum = 0
for i in range (1, 11):
    sum += (1 * 100) // (i ** 3) / 100

print(sum)
```

```
1.1600000000000001
```

Conclusiones

En el literal a) claramente se puede observar la mínima distancia que tiene el resultado real con el redondeado, por lo que el primer metodo es el más efectivo.

2. La serie de Maclaurin

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

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

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

Entonces, dado que $\arctan 1 = \frac{\pi}{4}$ con $x = 1$, reemplazando:

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

```
# Valor al que la sumatoria debe acercarse
import math
x = math.pi / 4
print(x)
```

```
0.7853981633974483
```

Ahora, se presenta el código que calcula la sumatoria para cierto número n:

```
sum = 0
n = 1000
for i in range(1, n + 1):
    sum += ((-1) ** (i + 1)) * (1 ** (2*i - 1)/(2*i - 1))
print(sum)
```

```
0.7851481634599485
```

Mediante prueba y acercamiento, se encontró el valor $n = 1000$ para el cual, la función se acerca al valor $\frac{\pi}{4}$.

Por tanto, $P_{1000}(1) = 0.78514....$

Comprobemos si cumple la condición:

$$|4P_{22}(1) - \pi| < 10^{-3}$$

```
total = abs(4*sum - math.pi)
print(total)
print(total < 10**(-3))
```

```
0.000999999749998981
True
```

En la función print() directamente se hizo la comprobación sobre la inecuación con los valores dados.

Literal 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?

Respuesta: Para este caso, se necesitaría hacer una suma infinita de la función $P_n(\pi)$. Esto se debe a que la tangente de π no está definida.

3. Pi

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 π dentro de 10^{-3}.

Para resolver este ejercicio, despejamos el número 4 de la izquierda, quedándonos:

$$\pi = \frac{4\arctan\frac{1}{5} - \arctan\frac{1}{239}}{4}$$

Ahora, el siguiente código cumple con la suma de esta funcion con un parámetro n

```
from numpy import arctan
sum = 0
n = 16
for i in range (1, n + 1):
    sum += (4*arctan(1 / 5) - arctan(1/239)) / 4
print(sum)
print(abs(math.pi - sum))
```

```
3.1415926535897944
1.3322676295501878e-15
```

Encontramos a $n = 16$, sin embargo la aproximación de π tiene una distancia muy pequeña con el valor real de 10^{-15}.

4. Algoritmos

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

```
a. ENTRADA  , 1, 2, ,  .
SALIDA PRODUCT.
Paso 1 Determine PRODUCT = 0.
Paso 2 Para   = 1,2, ,  haga
Determine PRODUCT = PRODUCT* .
Paso 3 SALIDA PRODUCT;
PARE.

b. ENTRADA  , 1, 2, ,  .
SALIDA PRODUCT.
Paso 1 Determine PRODUCT = 1.
```

```
Paso 2 Para   = 1,2, ,  haga
Set PRODUCT = PRODUCT* .
Paso 3 SALIDA PRODUCT;
PARE.

c. ENTRADA  , 1, 2, ,  .
SALIDA PRODUCT.
Paso 1 Determine PRODUCT = 1.
Paso 2 Para   = 1,2, ,  haga
si  =0 entonces determine PRODUCT = 0;
SALIDA PRODUCT;
PARE
Determine PRODUCT = PRODUCT* .
Paso 3 SALIDA PRODUCT;
PARE.
```

Respuesta: En realidad no sería muy correcto, ya que el producto tiene como valor cero. El algoritmo calcula el producto de todos los componentes de un vector pero esta función siempre retornará cero.

5. Sumas y Multiplicaciones

Literal a)

¿Cuántas multiplicaciones y sumas se requieren para determinar una suma de la forma:

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

Respuesta: Se requieren $\sum_{i=1}^{n} i$ multiplicaciones y $\frac{n(n+1)}{2}$ sumas en total.

Literal b)

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

1. Linealidad de sumatoria: $ = *{i=1}^{n} (a_i *{j=1}^{i} b_j)$
2. Suma de una progresión aritmética: $\sum_{i=1}^{n} a_i \cdot \frac{i(i+1)}{2}$
3. Reagrupación: $\sum_{i=1}^{n} \frac{i(i+1)}{2} a_i$

Discusiones

1. Algoritmo II

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

El siguiente algoritmo cumple con lo solicitado:

```
x = [1, 2, 3] # Lista a sumar de manera inversa
sum = 0
n = len(x)
for i in range (1, n + 1):
    sum += x[(n - i)]

print(sum)
```

```
8
```

2. Raíces

Construya un algoritmo con entrada a, b y 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.

```
def raices(a : float, b : float, c : float) -> tuple[float, float] | float | tuple[complex, c
    discriminante = b**2 - 4*a*c

    if (discriminante == 0):
        raiz = (-b + (discriminante)**0.5) / 2*a
        return raiz

    elif (discriminante > 0):
        raiz1 = (-b + (discriminante)**0.5) / 2*a
        raiz2 = (-b - (discriminante)**0.5) / 2*a
        return raiz1, raiz2

    else:
        raiz1 = complex(-b / 2*a, (abs(discriminante))**0.5 * 100 // 2*a / 100)
        raiz2 = complex(-b / 2*a, -(abs(discriminante))**0.5 * 100 // 2*a / 100)
        return raiz1, raiz2
```

```
resultado = raices(1.5, -2, 1)
print('La/s raiz/ices de la ecuacion cuadratica es/son: ', resultado)
```

```
La/s raiz/ices de la ecuacion cuadratica es/son:  ((1.5+1.05j), (1.5-1.065j))
```

Las celdas anteriores muestran el código que se utilizó para la creación de una calculadora de raíces en el que se usan tuplas.

3. Serie

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}.

$$\frac{1-2x}{1-x+x^2}+\frac{2x-4x^3}{1-x^2+x^4}+\frac{4x^3-8x^7}{1-x^4+x^8}..=\frac{1+2x}{1+x+x^2}$$

para $x<1$ y si $x=0.25$.

```
def calcular_lado_izquierdo(x):
    suma_lado_izquierdo = 0.0
    termino = 1
    denominador = 1.0
    while True:
        suma_lado_izquierdo += (2.0**(termino - 1) * x**(2*termino - 1)) / denominador
        if abs(suma_lado_izquierdo - (1 + 2*x) / (1 + x + x**2)) < 1e-6:
            break
        termino += 1
        denominador *= 1 - x**(2*termino)
    return termino
```