Tarea 3

Cristina Moreno

- 1._ Cuál es mas preciso?

a) $\sum_{i=1}^{10} \frac{1}{i^2} = \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \frac{1}{5^2} + \frac{1}{6^2} + \frac{1}{7^2} + \frac{1}{8^2} + \frac{1}{9^2} + \frac{1}{10^2}$

En forma Ascendente y Descendente

In [25]:
```python
import math
def cortartres(x):
    if x == 0:
        return 0
    exp=math.floor(math.log10(abs(x)))
    factor = 10 ** (exp - 2)
    return math.floor(x / factor) * factor

def suma_ascendente():
    suma=0
    for i in range (1,11):
        total=(cortartres(1/i**2))
        suma=cortartres(suma+total)
    return suma

def suma_descendente():
    sumades=0
    for i in range (10,0,-1):
        total=cortartres(1/i**2)
        sumades=cortartres(sumades+total)
    return sumades


def error_relativ(n):
    valorReal=0
    for i in range (1,11):
        valorReal=valorReal+(1/i**2)

    errorRer=(abs(valorReal-n)/valorReal)*100
    print(f"el error relativo es {errorRer}%")


suma=suma_ascendente()
print(f"con suma ascdendente {suma}")
error_relativ(suma)

sumades=suma_descendente()
print(f"son suma descendente {sumades}")
error_relativ(sumades)
```

```
con suma ascdendente 1.53
el error relativo es 1.2755286336786207%
son suma descendente 1.54
el error relativo es 0.6302706508922058%
```

Se puede decir que la suma descendente tiene error relativo menor por lo tanto es mas preciso

b) $\sum_{i=1}^{10} \frac{1}{i^3} = \frac{1}{1^3} + \frac{1}{2^3} + \frac{1}{3^3} + \frac{1}{4^3} + \frac{1}{5^3} + \frac{1}{6^3} + \frac{1}{7^3} + \frac{1}{8^3} + \frac{1}{9^3} + \frac{1}{10^3}$

En forma Ascendente y Descendente

In [26]:
```python
import math
def cortartres(x):
    if x == 0:
        return 0
    exp=math.floor(math.log10(abs(x)))
    factor = 10 ** (exp - 2)
    return math.floor(x / factor) * factor

def suma_ascendente():
    suma=0
    for i in range (1,11):
        total=(cortartres(1/i**3))
        suma=cortartres(suma+total)
    return suma

def suma_descendente():
    sumades=0
    for i in range (10,0,-1):
        total=cortartres(1/i**3)
        sumades=cortartres(sumades+total)
    return sumades


def error_relativ(n):
    valorReal=0
    for i in range (1,11):
        valorReal=valorReal+(1/i**3)

    errorRer=(abs(valorReal-n)/valorReal)*100
    print(f"el error relativo es {errorRer}%")


suma=suma_ascendente()
print(f"con suma ascdendente {suma}")
error_relativ(suma)

sumades=suma_descendente()
print(f"son suma descendente {sumades}")
error_relativ(sumades)
```

```
con suma ascdendente 1.16
el error relativo es 3.134111332572313%
son suma descendente 1.19
el error relativo es 0.628959039449181%
```

Por lo tanto la suma descendente es mas precisa

Al sumar primero los términos pequeños, se reduce la pérdida de cifras significativas causada por el corte

- 2. $\arctan(x) = \lim_{n\to\infty} P_n(x) = \lim_{n\to\infty} \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i+1}}{2i+1}$

a) $|4Pn(1) - \pi| < 10^{-3}$ cuando x=1

In [21]:
```python
import math

def arctan(n):
    return sum((-1)**(i + 1) / (2 * i - 1) for i in range(1, n + 1))

precision = 10**-3
diff = 1
n= 0

while diff>= precision:
    n+= 1
    approx= 4 * arctan(n)
    diff= abs(approx- math.pi)

print(f"N mínimo: {n}")
```

```
N mínimo: 1000
```

b) y con 10^-10

In [27]:
```python
def calcularn(tolerancia):
    nmin = (4 / tolerancia - 1) / 2
    return int(nmin) + 1


tolerancia = 1e-10
n= calcularn(tolerancia)
print(f"n mínimo = {n}")
```

```
n mínimo = 20000000000
```

- 3._ $\frac{\pi}{4} = 4\arctan\left(\frac{1}{5}\right) - \arctan\left(\frac{1}{239}\right)$

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

In [ ]:
```python
def arctan(x, n):
    suma = 0.0
    signo = 1
    for i in range(1, n+1):
        suma = suma+ signo * (x**(2*i-1)) / (2*i-1)
        signo =signo*-1
    return suma

tolerancia = 1e-3

n = 1

while True:
    pi_aprox = 16 * arctan(1/5, n) - 4 * arctan(1/239, n)

    error = abs(pi_aprox - 3.141592653589793)

    if error < tolerancia:
        break
    n =n+1

print(f"Número de términos necesarios: {n}")
print(f"Error: {error}")
```

```
Número de términos necesarios: 2
Error: 0.0009956242637327861
```

- 4._

Imagen1

El primer algortimo nunca modifica simepre multiplica por cero

El segundo algritmo es el corecto

El tercero es correcto pero tiene la condicion de cuando ingrese x=0 Tanto el segundo como el tercero estan correctos pero el tercero es mas óptimo

- 5._ $\sum_{i=1}^{n} \sum_{j=1}^{n} a_i b_j$

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

Número de multiplicaciones Cada vez que calculamos ai * bj hacemos una multiplicación
Para cada (i) hay (n) valores de (j), entonces:

$n + n + \cdots + n = n \cdot n = n^2$

Por eso Multiplicaciones = n^2

Número de sumas

Tenemos n^2 productos: a1 b1, a1 b2, ..., an bn Para sumar m números se necesitan m - 1 sumas, porque el primer número se asigna a la variable acumuladora.

Total de sumas = n^2 - 1

Ejemplo

In [28]:
```python
n = 4


a = [1, 2, 3, 4]
b = [5, 6, 7, 8]

total = 0
multiplicaciones = 0
sumas = 0

for i in range(n):
    for j in range(n):
        producto = a[i] * b[j]
        multiplicaciones += 1

        if i == 0 and j == 0:
            total = producto
        else:
            total += producto
            sumas += 1

print(f"Suma total: {total}")
print(f"Multiplicaciones realizadas: {multiplicaciones} (debería ser n^2 = {n**2})")
print(f"Sumas realizadas: {sumas} (debería ser n^2 - 1 = {n**2 - 1})")
```

```
Suma total: 260
Multiplicaciones realizadas: 16 (debería ser n^2 = 16)
Sumas realizadas: 15 (debería ser n^2 - 1 = 15)
```

DISCUSIONES

1. haga un alrogitmo para la suma inversa

Algoritmo Suma_Inversa Entrada: arreglo X[1..n] con n números Salida: suma total S

1. Inicializar S = 0
2. Para i desde n hasta 1 hacer: S = S + X[i]
3. Fin Para
4. Retornar S

2. Las ecuaciones (1.2) y (1.3) en la sección 1.2 proporcionan formas alternativas para las raíces $x1$ y $x2$ de ax 2 + bx + c = 0. Construya un algoritmo con entrada a, b, c c y salida $x1$, $x2$ que calcule las raíces $x1$ y $x2$ (que pueden ser iguales con conjugados complejos) mediante la mejor fórmula para cada raíz.

Algoritmo RaicesCuadratica Entrada: a, b, c Salida: x1, x2

1. Calcular discriminante: D = b^2 - 4ac

2. Si D > 0 entonces

   ```
   Si b >= 0 entonces

       x1 = (-b - sqrt(D)) / (2*a)

   Sino

       x1 = (-b + sqrt(D)) / (2*a)

   Fin Si

   x2 = c / (a * x1)
   ```
3. Sino si D = 0 entonces

   ```
   x1 = -b / (2*a)

   x2 = x1
   ```
4. Sino # D < 0, raíces complejas

   ```
   real = -b / (2*a)

   imaginario = sqrt(-D) / (2*a)

   x1 = real + i * imaginario

   x2 = real - i * imaginario
   ```

Fin Si

1. Retornar x1, x2

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

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 .

In [ ]:
```python
x = 0.25
tolerancia = 1e-6

suma = 0.0
n = 0

while True:
    pow1 = 2**n
    pow2 = 2**(n+1)

    numerador = pow1 * x**(pow1 - 1) - pow2 * x**(pow2 - 1)
    denominador = 1 - x**pow1 + x**pow2

    termino = numerador / denominador
    suma = suma+ termino
    n = n+1

    lado_derecho = (1 + 2*x)/(1 + x + x**2)

    if abs(suma - lado_derecho) < tolerancia:
        break

print(f"Número de términos necesarios: {n}")
print(f"Suma aproximada: {suma}")
print(f"Lado derecho: {lado_derecho}")
print(f"Diferencia: {abs(suma - lado_derecho)}")
```

```
Número de términos necesarios: 4
Suma aproximada: 1.1428571279559818
Lado derecho: 1.1428571428571428
Diferencia: 1.4901160971803051e-08
```