📊 Métodos Numéricos

👨‍🏫 Profesor: Jonathan Zea

👤 Estudiante: Matthew Cedeño

🗓️ Fecha: 16 de mayo de 2025

✨ ¡Vamos a aprender y a resolver problemas con precisión! ✨

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

a

Suma de la serie:

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

Primero sumando en orden ascendente:

$$\frac{1}{1} + \frac{1}{4} + \cdots + \frac{1}{100}$$

Luego sumando en orden descendente:

$$\frac{1}{100} + \frac{1}{81} + \cdots + \frac{1}{1}$$

---

b.

Suma de la serie:

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

Primero sumando en orden ascendente:

$$\frac{1}{1} + \frac{1}{8} + \frac{1}{27} + \cdots + \frac{1}{1000}$$

Luego sumando en orden descendente:

$$\frac{1}{1000} + \frac{1}{729} + \cdots + \frac{1}{1}$$

Psudocodigo

In [ ]:
```python
import math

def truncar_significativas(x, cifras=3):
    if x == 0:
        return 0.0
    else:
        orden = int(math.floor(math.log10(abs(x))))
        factor = 10**(cifras - 1 - orden)
        return math.trunc(x * factor) / factor


# De mayor a menor: i = 1 a 10
suma_mayor_a_menor = 0.0
for i in range(1, 11):
    term = 1 / (i**2)
    suma_mayor_a_menor = truncar_significativas(suma_mayor_a_menor + truncar_signif

# De menor a mayor: i = 10 a 1
suma_menor_a_mayor = 0.0
for i in range(10, 0, -1):
    term = 1 / (i**2)
    suma_menor_a_mayor = truncar_significativas(suma_menor_a_mayor + truncar_signif

print("Parte a:")
print("  Suma de mayor a menor:", suma_mayor_a_menor)
print("  Suma de menor a mayor:", suma_menor_a_mayor)
```

```
Parte a:
  Suma de mayor a menor: 1.53
  Suma de menor a mayor: 1.54
```

In [14]:
```python
import math

def truncar_significativas(x, cifras=3):
    if x == 0:
        return 0.0
    else:
        orden = int(math.floor(math.log10(abs(x))))
        factor = 10**(cifras - 1 - orden)
        return math.trunc(x * factor) / factor
# De mayor a menor: i = 1 a 10
suma_mayor_a_menor = 0.0
for i in range(1, 11):
    term= 1 / (i**3)
    suma_mayor_a_menor = truncar_significativas(suma_mayor_a_menor + truncar_signif
# De menor a mayor: i = 10 a 1
suma_menor_a_mayor = 0.0
for i in range(10, 0, -1):
    term = 1 / (i**3)
```

```
    suma_menor_a_mayor = truncar_significativas(suma_menor_a_mayor + truncar_signif
print("  Suma de mayor a menor:", suma_mayor_a_menor)
print("  Suma de menor a mayor:", suma_menor_a_mayor)
```

```
Suma de mayor a menor: 1.16
Suma de menor a mayor: 1.19
```

1.3.2. Serie de Maclaurin para la función arcotangente

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

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

a. Determinación del número de términos para alcanzar una precisión

Utilice el hecho de que $\tan(\pi/4) = 1$ para determinar el número n de términos de la serie que se necesita sumar para garantizar que:

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

.

In [1]:
```
import math

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

n_terms = 0
precision = 1e-3
difference = 1

while difference >= precision:
    n_terms += 1
    current_approximation = 4 * maclaurin_arctan_approx(n_terms)
    difference = abs(current_approximation - math.pi)

print(f"Número de términos necesarios para alcanzar la precisión de 10^-3: {n_terms
```

```
Número de términos necesarios para alcanzar la precisión de 10^-3: 1000
```

b. El lenguaje de programación C++ requiere que el valor de (π) se encuentre dentro de (10^{-6}). ¿Cuántos términos de la serie se necesitarían sumar para obtener este grado de precisión?

In [3]:
```
import math

def calcular_pi(precision_deseada):
    tolerancia = 0.5 * 10 ** (-precision_deseada)
    suma = 0.0
```

```
    n = 0
    signo = 1
    while True:
        termino = signo / (2 * n + 1)
        suma += termino
        aproximacion = 4 * suma
        error = abs(aproximacion - math.pi)
        if error < tolerancia:
            break
        signo *= -1
        n += 1
    return n + 1, aproximacion

precision_deseada = 6
n_terminos, aproximacion = calcular_pi(precision_deseada)
print(f"Número de términos necesarios para alcanzar la precisión de 10^-6:{n_termin
```

```
Número de términos necesarios para alcanzar la precisión de 10^-6:2000001
```

3. Aproximación de π mediante otra identidad

Otra fórmula para calcular π se puede deducir a partir de la siguiente identidad:

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

Determine el número de términos que se deben sumar para garantizar una aproximación de π con una precisión de 10^{-3}.

In [4]:
```
def maclaurin_arctan(x, terms):
    return sum((-1)**i * (x**(2 * i + 1) / (2 * i + 1)) for i in range(terms))

def calculate_pi_from_identity(terms):
    return 4 * (4 * maclaurin_arctan(1 / 5, terms) - maclaurin_arctan(1 / 239, term

precision_threshold = 1e-3
terms_used = 0
pi_approximation_error = 1

# Cálculo iterativo
while pi_approximation_error >= precision_threshold:
    terms_used += 1
    current_pi_value = calculate_pi_from_identity(terms_used)
    pi_approximation_error = abs(current_pi_value - math.pi)

print(f"Número de términos necesarios para obtener precisión de 10^-3: {terms_used}
```

```
Número de términos necesarios para obtener precisión de 10^-3: 2
```

5. Complejidad de la suma doble

a. Número de operaciones necesarias

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

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

In [5]:
```
def count_multiplications(n):
    total_multiplicaciones = 0
    for i in range(1, n + 1):
        total_multiplicaciones += i
    return total_multiplicaciones

n = 5
multiplicaciones = count_multiplications(n)
print(f"Para n={n}, se realizan {multiplicaciones} multiplicaciones.")
```

```
Para n=5, se realizan 15 multiplicaciones.
```

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

In [6]:
```
def count_operations(n):
    total_sumas = 0
    total_multiplicaciones = 0

    for i in range(1, n + 1):
        total_multiplicaciones += i
        total_sumas += (i - 1)

    return total_multiplicaciones, total_sumas

n = 5
multiplicaciones, sumas = count_operations(n)
print(f"Para n={n}, se realizan {multiplicaciones} multiplicaciones y {sumas} sumas
```

```
Para n=5, se realizan 15 multiplicaciones y 10 sumas.
```

Escriba un algoritmo para sumar la serie finita:

$$\sum_{i=1}^{n} x_i$$

pero **en orden inverso**, es decir, desde (x_n) hasta (x_1).

In [1]:
```
x=[1, 2, 3, 4, 5]
suma=0
for i in range (len(x)-1,-1,-1):
    suma+=x[i]
    print("La suma en orden inverso es :",suma)
```

```
La suma en orden inverso es : 5
La suma en orden inverso es : 9
La suma en orden inverso es : 12
La suma en orden inverso es : 14
La suma en orden inverso es : 15
```

$ax2 + bx + c = 0$. Construya un algoritmo con entrada a, b, c c y salida $x1$, $x2$ que calcule las raíces $x1$ y $x2$

In [3]:
```
import math
a=float(input("Ingrese el valor de a: "))
b=float(input("Ingrese el valor de b: "))
c=float(input("Ingrese el valor de c: "))
if a==0:
    print("No es una ecuación cuadrática")
else:
    D=b**2-4*a*c
    if D<0:
        print("No tiene solución real")
    elif D==0:
        x=-b/(2*a)
        print("La solución es:",x)
    else:
        x1=(-b+math.sqrt(D))/(2*a)
        x2=(-b-math.sqrt(D))/(2*a)
        print("Las soluciones son:",x1,"y",x2)
```

```
Las soluciones son: 5.0 y 1.0
```

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 [4]:
```
x = 0.25
tolerancia = 1e-6

# Calcular lado derecho
lado_derecho = (1 + 2*x) / (1 + x + x**2)

suma = 0.0
n = 1

while abs(suma - lado_derecho) >= tolerancia:
    a = 2**(n - 1) * x**(2**(n - 1) - 1)
    b = 2**n * x**(2**n - 1)
    numerador = a - b
    denominador = 1 - x**(2**(n - 1)) + x**(2**n)
    termino = numerador / denominador

    suma += termino
    n += 1

print(f"Número de términos necesarios: {n - 1}")
print(f"Suma parcial: {suma}")
print(f"Valor lado derecho: {lado_derecho}")
```

```
Número de términos necesarios: 4
Suma parcial: 1.1428571279559818
Valor lado derecho: 1.1428571428571428
```

Link del repositorio : https://github.com/MatthewC-20/Deber2/blob/main/Deber2.ipynb