# Tarea_03_Lema_Luis

November 4, 2025

## 1 Escuela Politécnica Nacional

### 1.1 [Tarea 03] Ejercicios Unidad 01-B

#### 1.1.1 Nombre: Luis Alexander Lema Delgado

#### 1.1.2 Fecha: 02/11/2025

#### 1.1.3 Curso: GR1CC

#### 1.1.4 Repositorio:

https://github.com/LuisALema/Metodos_Numericos_2025B/tree/main/Deberes/Tarea05

**CONJUNTO DE EJERCICIOS 1.3**

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

a. $\sum_{i=1}^{10} \left(1/i^2\right)$ primero por $\frac{1}{1} + \frac{1}{4} + \cdots + \frac{1}{100}$ y luego por $\frac{1}{100} + \frac{1}{81} + \cdots + \frac{1}{1}$

[1]:
```
import math
import decimal

def truncar(numero, digitos):

    factor = 10.0 ** digitos
    return math.floor(numero * factor) / factor

def suma_corte(terminos, digitos_corte=3):

    suma_acumulada = 0.0
    pasos_intermedios = []
    for termino in terminos:
        termino_truncado = truncar(termino, digitos_corte)
        suma_parcial_truncada = truncar(suma_acumulada + termino_truncado,␣
 ↪digitos_corte)
        pasos_intermedios.append((termino_truncado, suma_parcial_truncada))
        suma_acumulada = suma_parcial_truncada
    return suma_acumulada, pasos_intermedios
```

```python
# Calcular los términos de la serie
terminos = [1.0 / (i**2) for i in range(1, 11)]

# Método 1: Suma hacia adelante
print("Suma ascendente:")
error_ascendente, pasos_a = suma_corte(terminos)
for termino, suma_parcial in pasos_a:
    print(f"Término truncado: {termino:.3f}, Suma parcial truncada: {suma_parcial:.3f}")
print(f"Resultado de la suma ascendente con truncamiento: {error_ascendente:.3f}\n")

# Método 2: Suma hacia atrás
print("Suma descendente:")
error_descendente, pasos_descendente = suma_corte(list(reversed(terminos)))
for termino, suma_parcial in pasos_descendente:
    print(f"Término truncado: {termino:.3f}, Suma parcial truncada: {suma_parcial:.3f}")
print(f"Resultado de la suma descendente con truncamiento: {error_descendente:.3f}")

# Determinar qué método es más preciso
decimal.getcontext().prec = 10
suma = sum(decimal.Decimal(1) / decimal.Decimal(i**2) for i in range(1, 11))

print(f"\nSuma con mayor precisión para comparación: {float(suma):.6f}")
print(f"Error absoluto de la suma ascedente: {abs(error_ascendente - float(suma)):.6f}")
print(f"Error absoluto de la suma descendente: {abs(error_descendente - float(suma)):.6f}")

if abs(error_descendente - float(suma)) < abs(error_ascendente - float(suma)):
    print("\nLa suma descendente es más precisa con aritmética de corte de tres dígitos.")
else:
    print("\nLa suma ascendente es más precisa con aritmética de corte de tres dígitos.")
```

```
Suma ascendente:
Término truncado: 1.000, Suma parcial truncada: 1.000
Término truncado: 0.250, Suma parcial truncada: 1.250
Término truncado: 0.111, Suma parcial truncada: 1.361
Término truncado: 0.062, Suma parcial truncada: 1.423
Término truncado: 0.040, Suma parcial truncada: 1.463
Término truncado: 0.027, Suma parcial truncada: 1.490
```

```
Término truncado: 0.020, Suma parcial truncada: 1.510
Término truncado: 0.015, Suma parcial truncada: 1.525
Término truncado: 0.012, Suma parcial truncada: 1.537
Término truncado: 0.010, Suma parcial truncada: 1.547
Resultado de la suma ascendente con truncamiento: 1.547

Suma descendente:
Término truncado: 0.010, Suma parcial truncada: 0.010
Término truncado: 0.012, Suma parcial truncada: 0.022
Término truncado: 0.015, Suma parcial truncada: 0.037
Término truncado: 0.020, Suma parcial truncada: 0.056
Término truncado: 0.027, Suma parcial truncada: 0.083
Término truncado: 0.040, Suma parcial truncada: 0.123
Término truncado: 0.062, Suma parcial truncada: 0.185
Término truncado: 0.111, Suma parcial truncada: 0.296
Término truncado: 0.250, Suma parcial truncada: 0.546
Término truncado: 1.000, Suma parcial truncada: 1.546
Resultado de la suma descendente con truncamiento: 1.546

Suma con mayor precisión para comparación: 1.549768
Error absoluto de la suma ascedente: 0.002768
Error absoluto de la suma descendente: 0.003768

La suma ascendente es más precisa con aritmética de corte de tres dígitos.
```

b. $\sum_{i=1}^{10} \left(1/i^3\right)$ primero por $\frac{1}{1}+\frac{1}{8}+\frac{1}{27}+\cdots+\frac{1}{1000}$ y luego por $\frac{1}{1000}+\frac{1}{729}+\cdots+\frac{1}{1}$

[2]:
```
# Calcular los términos de la serie para el literal b
terminos_b = [1.0 / (i**3) for i in range(1, 11)]

# Método 1: Suma hacia adelante para el literal b
print("Literal b: Suma ascendente:")
resultado_ascendente_b, pasos_b = suma_corte(terminos_b)
for termino, suma_parcial in pasos_b:
    print(f"Término truncado: {termino:.3f}, Suma parcial truncada:␣
 ↪{suma_parcial:.3f}")
print(f"Resultado de la suma ascendente con corte a 3 dígitos:␣
 ↪{resultado_ascendente_b:.3f}\n")

# Método 2: Suma descendete para el literal b
print("Literal b: Suma descendente:")
resultado_descendente_b, pasos_b_descendente =␣
 ↪suma_corte(list(reversed(terminos_b)))
for termino, suma_parcial in pasos_b_descendente:
```

```
    print(f"Término truncado: {termino:.3f}, Suma parcial truncada:␣
 ↪{suma_parcial:.3f}")
print(f"Resultado de la suma descendente con corte a 3 dígitos:␣
 ↪{resultado_descendente_b:.3f}")

# Determinar qué método es más preciso para el literal b
decimal.getcontext().prec = 10
suma_b = sum(decimal.Decimal(1) / decimal.Decimal(i**3) for i in range(1, 11))

print(f"\nSuma con mayor precisión para comparación: {float(suma_b):.6f}")
error_b = abs(resultado_ascendente_b - float(suma_b))
error_b_descendente = abs(resultado_descendente_b - float(suma_b))
print(f"Error absoluto de la suma ascendente: {error_b:.6f}")
print(f"Error absoluto de la suma descendente: {error_b_descendente:.6f}")

if error_b_descendente < error_b:
    print("\nPara el literal b, la suma descendente es más precisa con␣
 ↪aritmética de corte de tres dígitos.")
else:
    print("\nPara el literal b, la suma ascendente es más precisa con␣
 ↪aritmética de corte de tres dígitos.")
```

```
Literal b: Suma ascendente:
Término truncado: 1.000, Suma parcial truncada: 1.000
Término truncado: 0.125, Suma parcial truncada: 1.125
Término truncado: 0.037, Suma parcial truncada: 1.162
Término truncado: 0.015, Suma parcial truncada: 1.176
Término truncado: 0.008, Suma parcial truncada: 1.184
Término truncado: 0.004, Suma parcial truncada: 1.188
Término truncado: 0.002, Suma parcial truncada: 1.190
Término truncado: 0.001, Suma parcial truncada: 1.190
Término truncado: 0.001, Suma parcial truncada: 1.190
Término truncado: 0.001, Suma parcial truncada: 1.190
Resultado de la suma ascendente con corte a 3 dígitos: 1.190

Literal b: Suma descendente:
Término truncado: 0.001, Suma parcial truncada: 0.001
Término truncado: 0.001, Suma parcial truncada: 0.002
Término truncado: 0.001, Suma parcial truncada: 0.003
Término truncado: 0.002, Suma parcial truncada: 0.005
Término truncado: 0.004, Suma parcial truncada: 0.009
Término truncado: 0.008, Suma parcial truncada: 0.017
Término truncado: 0.015, Suma parcial truncada: 0.032
Término truncado: 0.037, Suma parcial truncada: 0.069
Término truncado: 0.125, Suma parcial truncada: 0.194
Término truncado: 1.000, Suma parcial truncada: 1.194
Resultado de la suma descendente con corte a 3 dígitos: 1.194
```

```
Suma con mayor precisión para comparación: 1.197532
Error absoluto de la suma ascendente: 0.007532
Error absoluto de la suma descendente: 0.003532

Para el literal b, la suma descendente es más precisa con aritmética de corte de
tres dígitos.
```

**2. La serie de Maclaurin para la función arcotangente converge para $-1< 1$ 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 $4/=1$ para determinar el número n de términos de la serie que se necesita sumar para garantizar que**

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

[3]:
```python
def calculo_terminos_a():

    tolerancia = 1e-3
    suma = 0.0
    n = 1

    while True:
        # Calcula el término n-ésimo de la serie
        termino = (-1)**(n + 1) / (2 * n - 1)
        suma += termino
        pi_aproximado = 4 * suma

        # Calcula el error absoluto
        error = abs(pi_aproximado - math.pi)

        # Verifica si se alcanzó la precisión deseada
        if error < tolerancia:
            print(f"Se necesitan {n} términos para alcanzar |4P_n(1) -  | <␣
 ↪10^-3")
            print(f"Valor aproximado de  : {pi_aproximado}")
            print(f"Error absoluto: {error:.6f}")
            print(f"Valor real de  : {math.pi:.6f}")
            return
        n += 1
```

```
calculo_terminos_a()
```

```
Se necesitan 1000 términos para alcanzar |4P_n(1) -  | < 10^-3
Valor aproximado de  : 3.140592653839794
Error absoluto: 0.001000
Valor real de  : 3.141593
```

**b. El lenguaje de programación C++ requiere que el valor de  se encuentre dentro de 10^−10. ¿Cuántos términos de la serie se necesitarían sumar para obtener este grado de precisión?**

[18]:
```
def calcular_terminos_B(tolerancia=1e-10, max_terminos=10**7):

    suma = 0.0
    n = 1
    while n <= max_terminos:
        termino = (-1)**(n + 1) / (2 * n - 1)
        suma += termino
        pi_aproximado = 4 * suma
        error = abs(pi_aproximado - math.pi)

        if error < tolerancia:
            print(f"Se necesitan {n} términos para alcanzar |4P_n(1) -  | <␣
 ↪{tolerancia}")
            print(f"Valor aproximado de  : {pi_aproximado:.12f}")
            print(f"Error absoluto: {error:.1e}")
            print(f"Valor real de  : {math.pi:.12f}")
            return
        n += 1
    print(f"No se alcanzó la precisión deseada ({tolerancia}) después de␣
 ↪{max_terminos} términos.")

# Calcular el número de términos necesarios para una tolerancia de 1e-10
calcular_terminos_B(tolerancia=1e-10)
```

```
No se alcanzó la precisión deseada (1e-10) después de 10000000 términos.
```

**3. Otra fórmula para calcular  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  dentro de 10^−3.**

[20]:
```
# definir la función arctan que calcula el valor de arctan(x) usando la serie␣
 ↪de Taylor
```

```
def arctan(valor_x, cantidad_terminos):

  resultado = 0.0
  for indice in range(cantidad_terminos):
    numerador = ((-1) ** indice) * (valor_x ** (2 * indice + 1))
    denominador = 2 * indice + 1
    resultado += numerador / denominador
  return resultado

# Calcular el número de términos necesarios para aproximar pi con un error␣
 ↪menor a 1e-3
def aproximado_pi(margen_error):

  num_terminos = 1
  pi_estimado = 4 * (4 * arctan(1/5, num_terminos) - arctan(1/239,␣
 ↪num_terminos))

  while abs(pi_estimado - math.pi) > margen_error:
    num_terminos += 1
    pi_estimado = 4 * (4 * arctan(1/5, num_terminos) - arctan(1/239,␣
 ↪num_terminos))

  return num_terminos, pi_estimado

# Se usa la variable error_max ya definida en el notebook
error_max = 1e-3
n_terminos_necesarios, pi_calculado_nuevo = aproximado_pi(error_max)

print(f"Número de términos requeridos: {n_terminos_necesarios}")
print(f"Valor de pi estimado: {pi_calculado_nuevo}")
print(f"Error absoluto: {abs(pi_calculado_nuevo - math.pi)}")
```

```
Número de términos requeridos: 2
Valor de pi estimado: 3.1405970293260603
Error absoluto: 0.0009956242637327861
```

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

**Algoritmo A**

[7]:
```
def algoritmo_a(n, x):
    producto = 0
    for i in range(n):
        producto *= x[i]
    return producto
```

**Algoritmo B**

[8]:
```python
def algoritmo_b(n, x):
    producto = 1
    for i in range(n):
        producto *= x[i]
    return producto
```

**Algoritmo C**

[9]:
```python
def algoritmo_c(n, x):
    producto = 1
    for i in range(n):
        if x[i] == 0:
            producto = 0
            return producto  # Terminación temprana si se encuentra un 0
        producto *= x[i]
    return producto
```

[10]:
```python
# Casos de prueba
casos_prueba = [
    (3, [2, 3, 4]),       # Producto normal
    (5, [-1, 2, -3, 4, -5]), # Números negativos
    (2, [2.5, 4.0])      # Números de punto flotante
]
```

[12]:
```python
print("Resultados de los algoritmos:")
for n, x in casos_prueba:
    print(f"\nEntrada: n={n}, x={x}")
    resultado_a = algoritmo_a(n, list(x))
    resultado_b = algoritmo_b(n, list(x))
    resultado_c = algoritmo_c(n, list(x))
    resultado_correcto = math.prod(x) if x else 1 # Producto correcto

    print(f"  Algoritmo a: {resultado_a}")
    print(f"  Algoritmo b: {resultado_b}")
    print(f"  Algoritmo c: {resultado_c}")
    print(f"  Resultado Correcto: {resultado_correcto}")
```

```
Resultados de los algoritmos:

Entrada: n=3, x=[2, 3, 4]
  Algoritmo a: 0
  Algoritmo b: 24
  Algoritmo c: 24
  Resultado Correcto: 24

Entrada: n=5, x=[-1, 2, -3, 4, -5]
  Algoritmo a: 0
  Algoritmo b: -120
```

```
  Algoritmo c: -120
  Resultado Correcto: -120

Entrada: n=2, x=[2.5, 4.0]
  Algoritmo a: 0.0
  Algoritmo b: 10.0
  Algoritmo c: 10.0
  Resultado Correcto: 10.0
```

**5.a. ¿Cuántas multiplicaciones y sumas se requieren para determinar una suma de la forma**

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

[13]:
```python
# definir la función que calcula el número de multiplicaciones y sumas
def calcular_conteo(limite):
    producto_total = 0
    adiciones_totales = 0
    indice_externo = 1
    while indice_externo <= limite:
        indice_interno = 1
        while indice_interno <= indice_externo:
            producto_total += 1
            if indice_externo > 1 or indice_interno > 1:
                adiciones_totales += 1
            indice_interno += 1
        indice_externo += 1
    return producto_total, adiciones_totales

# Ejemplo
valor_n = 5
productos, adiciones = calcular_conteo(valor_n)
print(f"Para un límite de {valor_n}:")
print(f"Cantidad total de multiplicaciones: {productos}")
print(f"Número total de sumas: {adiciones}")
```

```
Para un límite de 5:
Cantidad total de multiplicaciones: 15
Número total de sumas: 14
```

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

[14]:
```python
# Definir la función que calcula el número de multiplicaciones y sumas de␣
 ↪manera optimizada
def calcular_conteo(limite):
    # Calcular el total de multiplicaciones usando la fórmula
    producto_total = (limite * (limite + 1)) // 2
```

```
    # Calcular el total de sumas usando la fórmula
    adiciones_totales = ((limite - 1) * limite) // 2
    return producto_total, adiciones_totales

# Ejemplo
valor_n = 5
productos, adiciones = calcular_conteo(valor_n)
print(f"Para un límite de {valor_n}:")
print(f"Cantidad total de multiplicaciones: {productos}")
print(f"Número total de sumas: {adiciones}")
```

```
Para un límite de 5:
Cantidad total de multiplicaciones: 15
Número total de sumas: 10
```

### 1.1.5 Discusiones

**1. Escriba un algoritmo para sumar la serie finita**

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

**en orden inverso**

[15]:
```
# definir la suma de la serie en orden inverso
def suma_inversa(x):
    suma_total = 0
    # Recorrer la lista en orden inverso
    for i in range(len(x)-1, -1, -1):
        suma_total += x[i]
    return suma_total

# Ejemplo
x = [3, 5.2, 4, 8.6, 2.5]  # Ejemplo de serie
resultado = suma_inversa(x)
print(f"La suma de la serie en orden inverso es: {resultado}")
```

```
La suma de la serie en orden inverso es: 23.3
```

**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 , , y salida x_1 y 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.**

[16]:
```
def obtener_raices(a, b, c):
    # Calcular el discriminante
    discriminante = b**2 - 4*a*c

    # Evaluar el discriminante para determinar el tipo de raíces
    if discriminante > 0:
        # Dos raíces reales distintas
        if b > 0:
            raiz1 = (-b - discriminante**0.5) / (2*a)
            raiz2 = (2*c) / (-b - discriminante**0.5)
        else:
            raiz1 = (-b + discriminante**0.5) / (2*a)
            raiz2 = (2*c) / (-b + discriminante**0.5)
        return f"Raíces reales distintas: x1 = {raiz1}, x2 = {raiz2}"

    elif discriminante == 0:
        # Una raíz real doble
        raiz = -b / (2*a)
        return f"Raíz doble y real: x = {raiz}"

    else:
        # Raíces complejas
        parte_real = -b / (2*a)
        parte_imaginaria = (abs(discriminante)**0.5) / (2*a)
        return f"Raíces complejas: x1 = {parte_real} + {parte_imaginaria}i, x2␣
 ↪= {parte_real} - {parte_imaginaria}i"

# Ejemplo
print(obtener_raices(1, -5, 3))  # Raíces reales distintas
print(obtener_raices(1, -4, 2))  # Raíz doble
print(obtener_raices(1, 3, 5))   # Raíces complejas
```

```
Raíces reales distintas: x1 = 4.302775637731995, x2 = 0.6972243622680053
Raíces reales distintas: x1 = 3.414213562373095, x2 = 0.585786437626905
Raíces complejas: x1 = -1.5 + 1.6583123951777i, x2 = -1.5 - 1.6583123951777i
```

**3. Suponga que**

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

**para <1 y si =0.25. Escriba y ejecute un algoritmo que determine el número de tér-**

**minos necesarios en el lado izquierdo de la ecuación de tal forma que el lado izquierdo difiera del lado derecho en menos de 10^−6.**

[17]:
```python
# calcular el número de términos necesarios para aproximar la suma de la serie
def calcular_termino(x, n):
    # Numerador
    pow_n = 2**n
    pow_n1 = 2**(n+1)
    numerador = (2**n) * (x**(pow_n - 1)) - (2**(n+1)) * (x**(pow_n1 - 1))
    # Denominador
    denominador = 1 - x**pow_n + x**pow_n1
    return numerador / denominador

# calcular la suma de la serie usando el número de términos necesarios
def calcular_suma(x, tolerancia=1e-6, max_iter=1000):
    suma = 0.0
    n = 0
    valor_derecho = (1 + 2*x) / (1 + x + x**2)
    while True:
        termino = calcular_termino(x, n)
        suma += termino
        diferencia = abs(suma - valor_derecho)
        if diferencia < tolerancia:
            break
        n += 1
        if n > max_iter:
            print("No se alcanzó la tolerancia en el máximo número de␣
 ↪iteraciones.")
            break
    return n+1, suma  # n+1 términos sumados

# Ejemplo
if __name__ == "__main__":
    x = 0.25
    tolerancia = 1e-6
    terminos_usados, suma_aproximada = calcular_suma(x, tolerancia)
    valor_derecho = (1 + 2*x) / (1 + x + x**2)
    print(f"Número de términos necesarios: {terminos_usados}")
    print(f"Suma aproximada de la serie: {suma_aproximada:.10f}")
    print(f"Valor del lado derecho exacto: {valor_derecho:.10f}")
```

```
Número de términos necesarios: 4
Suma aproximada de la serie: 1.1428571280
Valor del lado derecho exacto: 1.1428571429
```

[ ]: