ESCUELA POLITÉCNICA NACIONAL
FACULTAD DE INGENIERÍA DE SISTEMAS
MÉTODOS NUMÉRICOS
INGENIERÍA DE SISTEMAS INFORMÁTICOS Y DE COMPUTACIÓN

Nombre: Sebastián Alexander Morales Cedeño

Curso: GR1CC

Fecha: 12/05/2025

[Tarea 03] Ejercicios Unidad 01-B

Repositorio:

https://github.com/SebastianMoralesEpn/Github1.0/tree/61086fa24427f6add7054523b126b5f9a73fe251/Tareas/%5BTarea%2003%5D%20Ejercicios%20Unidad%2001-B

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

Pseudocódigo:

```
// Función para truncar un número a un número específico de dígitos decimales
FUNCION Truncar(numero, digitos):
   factor = 10 elevado a la potencia de digitos
   resultado = piso(numero * factor) / factor
   RETORNAR resultado

// Función para calcular la suma con aritmética de corte usando la función Truncar
FUNCION SumaConCorteMath(terminos, digitos_corte):
   suma_acumulada = 0.0
   pasos_intermedios = una lista vacía
   PARA CADA termino EN terminos:
      termino_truncado = Truncar(termino, digitos_corte)
      suma_parcial_truncada = Truncar(suma_acumulada + termino_truncado, digitos_corte)
      AGREGAR (termino_truncado, suma_parcial_truncada) a pasos_intermedios
      suma_acumulada = suma_parcial_truncada
   FIN PARA
   RETORNAR suma_acumulada, pasos_intermedios

// Calcular los términos de la serie
terminos = una lista vacía
PARA i DESDE 1 HASTA 10:
   AGREGAR 1.0 / (i * i) a terminos
```

```
FIN PARA

// Método 1: Suma ascendente
IMPRIMIR "Suma ascendente:"
resultado_adelante, pasos_adelante = SumaConCorteMath(terminos, 3)
IMPRIMIR "Resultado de la suma hacia adelante con corte a 3 dígitos (math):", formatear resultado_adelante con 3 decimales, "\n"

// Método 2: Suma descendente
IMPRIMIR "Suma descendente:"
terminos_invertidos = invertir el orden de la lista terminos
resultado_atras, pasos_atras = SumaConCorteMath(terminos_invertidos, 3)
IMPRIMIR "Resultado de la suma hacia atrás con corte a 3 dígitos:", formatear resultado_atras con 3 decimales

// CALCULAR suma_precision
IMPRIMIR "\nSuma con mayor precisión para comparación:", formatear suma_precision con 6 decimales
error_adelante = valor absoluto de (resultado_adelante - suma_precision)
error_atras = valor absoluto de (resultado_atras - suma_precision)
IMPRIMIR "Error absoluto de la suma ascendente:", formatear error_adelante con 6 decimales
IMPRIMIR "Error absoluto de la suma descendente:", formatear error_atras con 6 decimales

SI error_atras < error_adelante ENTONCES
   IMPRIMIR "\nLa suma descendente es más precisa con aritmética de corte de tres dígitos."
SINO
   IMPRIMIR "\nLa suma ascendente es más precisa con aritmética de corte de tres dígitos."
FIN SI
```

Respuestas:

Resultado de la suma ascendente con truncamiento: 1.547
Resultado de la suma descendente con truncamiento: 1.546
...
Error absoluto de la suma ascendente: 0.002768
Error absoluto de la suma descendente: 0.003768

Conclusión: Como podemos observar la suma ascendente tiene un menor error absoluto, esto evidencia que la suma ascendente es más precisa.

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

Pseudocódigo:

```
// Función para truncar un número a un número específico de dígitos decimales
FUNCION Truncar(numero, digitos):
   factor = 10 elevado a la potencia de digitos
   resultado = piso(numero * factor) / factor
   RETORNAR resultado

// Función para calcular la suma con aritmética de corte usando la función Truncar
FUNCION SumaConCorte(terminos, digitos_corte):
   suma_acumulada = 0.0
   pasos_intermedios = una lista vacía
   PARA CADA termino EN terminos:
      termino_truncado = Truncar(termino, digitos_corte)
      suma_parcial_truncada = Truncar(suma_acumulada + termino_truncado,
digitos_corte)
      AGREGAR (termino_truncado, suma_parcial_truncada) a pasos_intermedios
      suma_acumulada = suma_parcial_truncada
   FIN PARA
   RETORNAR suma_acumulada, pasos_intermedios

// Calcular los términos de la serie para el literal b
terminos_b = una lista vacía
PARA i DESDE 1 HASTA 10:
   AGREGAR 1.0 / (i * i * i) a terminos_b
FIN PARA

// Método 1: Suma ascendente
IMPRIMIR "Literal b: Suma ascendente:"
resultado_ascendente_b, pasos_ascendente_b = SumaConCorte(terminos_b, 3)
IMPRIMIR "Resultado de la suma ascendente con corte a 3 dígitos (literal b):",
formatear resultado_ascendente_b con 3 decimales, "\n"

// Método 2: Suma descendente
IMPRIMIR "Literal b: Suma descendente:"
terminos_descendentes_b = invertir el orden de la lista terminos_b
resultado_descendente_b, pasos_descendente_b =
SumaConCorte(terminos_descendentes_b, 3)
```

IMPRIMIR "Resultado de la suma descendente con corte a 3 dígitos (literal b):", formatear resultado_descendente_b con 3 decimales

// CALCULAR suma_precision_b

IMPRIMIR "\nSuma con mayor precisión para comparación (literal b):", formatear suma_precision_b con 6 decimales
error_ascendente_b = valor absoluto de (resultado_ascendente_b - suma_precision_b)
error_descendente_b = valor absoluto de (resultado_descendente_b - suma_precision_b)
IMPRIMIR "Error absoluto de la suma ascendente (literal b):", formatear error_ascendente_b con 6 decimales
IMPRIMIR "Error absoluto de la suma descendente (literal b):", formatear error_descendente_b con 6 decimales

SI error_descendente_b < error_ascendente_b ENTONCES
  IMPRIMIR "\nPara el literal b, la suma descendente es más precisa con aritmética de corte de tres dígitos."
SINO
  IMPRIMIR "\nPara el literal b, la suma ascendente es más precisa con aritmética de corte de tres dígitos."
FIN SI

Respuestas:
Resultado de la suma ascendente con corte a 3 dígitos: 1.190
Resultado de la suma descendente con corte a 3 dígitos: 1.194
...
Error absoluto de la suma ascendente: 0.007532
Error absoluto de la suma descendente: 0.003532

Conclusión: Como podemos observar la suma descendente tiene un menor error absoluto, esto evidencia que la suma descendente es más precisa.

2. **La serie de Maclaurin para la función arcotangente converge para $-1 < x \leqslant 1$ 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 $\pi/4$ =1 para determinar el número n de términos de la serie que se necesita sumar para garantizar que $|4P_n(1) - \pi| < 10^{-3}$**

Pseudocódigo:

```
FUNCIÓN calculo_terminos_a():
    // Calcula términos necesarios para |4P_n(1) - π| < 10^-3

    tolerancia ← 1e-3
    suma ← 0.0
    n ← 1
    pi_real ← valor conocido de π

    MIENTRAS VERDADERO HACER:
        // Calcular término n-ésimo de la serie
        termino ← (-1)^(n + 1) / (2 * n - 1)
        suma ← suma + termino
        pi_aproximado ← 4 * suma

        // Calcular error absoluto
        error ← |pi_aproximado - pi_real|

        // Verificar precisión alcanzada
        SI error < tolerancia ENTONCES:
            IMPRIMIR "Términos necesarios: " + n
            IMPRIMIR "π aproximado: " + pi_aproximado
            IMPRIMIR "Error absoluto: " + error
            IMPRIMIR "π real: " + pi_real
            TERMINAR
        FIN SI

        n ← n + 1
    FIN MIENTRAS
FIN FUNCIÓN
```

Resultado:

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

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?

Pseudocódigo:

```
FUNCION Calculo_termino_B(tolerancia, max_terminos):
  suma = 0.0
  n = 1
  MIENTRAS n <= max_terminos HACER
    termino = (-1) elevado a la potencia de (n + 1) dividido por (2 * n - 1)
    suma = suma + termino
    pi_aproximado = 4 * suma
    error = valor absoluto de (pi_aproximado - PI)

    SI error < tolerancia ENTONCES
      IMPRIMIR "Se necesitan", n, "términos para alcanzar |4P_n(1) - π| <", tolerancia
      IMPRIMIR "Valor aproximado de π:", formatear pi_aproximado con 12 decimales
      IMPRIMIR "Error absoluto:", formatear error en notación científica con 1 dígito significativo
      IMPRIMIR "Valor real de π:", formatear PI con 12 decimales
      RETORNAR
    FIN_SI
    n = n + 1
  FIN_MIENTRAS
  IMPRIMIR "No se alcanzó la precisión deseada (", tolerancia, ") después de", max_terminos, "términos."

tolerancia_requerida = 10 elevado a la potencia de -10
maximo_terminos = 10 elevado a la potencia de 7
LLAMAR Calculo_terminos_B(tolerancia_requerida, maximo_terminos)
```

Resultado:

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

3. **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}.

Pseudocódigo:

```
FUNCION arctan(valor_x, cantidad_terminos):
  resultado = 0.0
```

```
  PARA indice DESDE 0 HASTA cantidad_terminos - 1 HACER
    numerador = ((-1) elevado a la potencia de indice) * (valor_x elevado a la potencia de (2 * indice + 1))
    denominador = 2 * indice + 1
    resultado = resultado + (numerador / denominador)
  FIN_PARA
  RETORNAR resultado

FUNCION aproximado_pi(margen_error):
  num_terminos = 1
  pi_estimado = 4 * (4 * arctan(1/5, num_terminos) - arctan(1/239, num_terminos))

  MIENTRAS valor absoluto de (pi_estimado - PI) > margen_error HACER
    num_terminos = num_terminos + 1
    pi_estimado = 4 * (4 * arctan(1/5, num_terminos) - arctan(1/239, num_terminos))
  FIN_MIENTRAS

  RETORNAR num_terminos, pi_estimado
LLAMAR n_terminos_necesarios, pi_calculado_nuevo = aproximado_pi(error_max)

IMPRIMIR "Número de términos requeridos:", n_terminos_necesarios
IMPRIMIR "Valor de pi estimado:", pi_calculado_nuevo
IMPRIMIR "Error absoluto:", valor absoluto de (pi_calculado_nuevo - PI)
```

Resultado:
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?**

a. ENTRADA $n, x1, x2, \cdots, xn$.
SALIDA PRODUCT.
Paso 1 Determine PRODUCT = 0.
Paso 2 Para $i = 1,2,\cdots,n$ haga
Determine PRODUCT = PRODUCT$*xi$.
Paso 3 SALIDA PRODUCT;
PARE.

b. ENTRADA $n, x1, x2, \cdots, xn$.
SALIDA PRODUCT.
Paso 1 Determine PRODUCT = 1.

Paso 2 Para i = 1,2,⋯,n haga
Set PRODUCT = PRODUCT$*xi$.
Paso 3 SALIDA PRODUCT;
PARE.

c. ENTRADA n,$x1$,$x2$,⋯,xn.
SALIDA PRODUCT.
Paso 1 Determine PRODUCT = 1.
Paso 2 Para i = 1,2,⋯,n haga
si xi=0 entonces determine PRODUCT = 0;
SALIDA PRODUCT;
PARE
Determine PRODUCT = PRODUCT$*xi$.
Paso 3 SALIDA PRODUCT;
PARE.

Respuesta: El Algoritmo a solo sería correcto en un escenario muy específico y probablemente no intencionado:

- Si n=0. En este caso, el bucle Para i = 1,2,...,n haga no se ejecutará en absoluto. El algoritmo pasará directamente al Paso 3 y mostrará el valor inicial de PRODUCTO, que es 0. En algunos contextos matemáticos, el producto de un conjunto vacío de números se define como la identidad multiplicativa, que es 1, no 0. Sin embargo, dependiendo de la definición específica que se esté utilizando, si el producto de ningún número se considera 0, entonces el Algoritmo a sería correcto para n=0.

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

Se piensa en cómo se evalúa esta doble suma:

1. **Multiplicaciones:** La cantidad total de multiplicaciones es la suma de los primeros n números naturales:

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

2. **Sumas:** La cantidad total de sumas es entonces la suma de los primeros $n-1$ números naturales:

$$\sum_{i=1}^{n} i = \frac{(n-1)\,n}{2}$$

Sin embargo, es importante notar que, al formar la suma final de todas estas sumas intermedias, añadimos una suma más por cada i excepción de la primera multiplicación para cada i.

Por lo tanto, el conteo final de sumas requeridas será una menos que el número de términos:

$$\frac{n(n+1)}{2} - 1$$

Pseudocódigo:

```
FUNCIÓN calcular_conteo(limite):
    INICIALIZAR producto_total = 0
    INICIALIZAR adiciones_totales = 0
    INICIALIZAR indice_externo = 1

    MIENTRAS indice_externo <= limite:
        INICIALIZAR indice_interno = 1
        MIENTRAS indice_interno <= indice_externo:
            INCREMENTAR producto_total EN 1
            SI indice_externo > 1 O indice_interno > 1 ENTONCES
                    INCREMENTAR adiciones_totales EN 1

            FIN_SI
            INCREMENTAR indice_interno EN 1
        FIN_MIENTRAS
        INCREMENTAR indice_externo EN 1
    FIN_MIENTRAS

    DEVOLVER producto_total, adiciones_totales

    // Ejemplo
    ESTABLECER valor_n = 5
    LLAMAR calcular_conteo CON valor_n Y ASIGNAR el resultado a productos, adiciones
    IMPRIMIR "Para un límite de", valor_n, ":"
    IMPRIMIR "Cantidad total de multiplicaciones:", productos
    IMPRIMIR "Número total de sumas:", adiciones
```

Resultado:

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.

Para optimizar el código y reducir el número de cálculos, podemos observar que el número total de multiplicaciones y sumas se puede calcular de manera más eficiente sin necesidad de realizar todos los bucles anidados.

Pseudocódigo:

```
FUNCION 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

   RETORNAR producto_total, adiciones_totales
FIN FUNCION

// Ejemplo
valor_n <- 5
productos, adiciones <- calcular_conteo(valor_n)
IMPRIMIR "Para un límite de ", valor_n, ":"
IMPRIMIR "Cantidad total de multiplicaciones: ", productos
IMPRIMIR "Número total de sumas: ", adiciones
```

Resultado:

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

DISCUSIONES

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

Pseudocódigo:

```
FUNCION suma_inversa(x)
   suma_total <- 0
   // Recorrer la lista en orden inverso
   PARA i DESDE longitud(x) - 1 HASTA 0 HACER
      suma_total <- suma_total + x[i]
```

```
    FIN PARA
    RETORNAR suma_total
FIN FUNCION

// Ejemplo
x <- [3, 5.2, 4, 8.6, 2.5]  // Ejemplo de serie
resultado <- suma_inversa(x)
IMPRIMIR "La suma de la serie en orden inverso es: ", resultado
```

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 a, b, c 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.**

Pseudocódigo:

```
FUNCION obtener_raices(a, b, c)
    // Calcular el discriminante
    discriminante <- b^2 - 4*a*c

    // Evaluar el discriminante para determinar el tipo de raíces
    SI discriminante > 0 ENTONCES
        // Dos raíces reales distintas
        SI b > 0 ENTONCES
            raiz1 <- (-b - raiz(discriminante)) / (2*a)
            raiz2 <- (2*c) / (-b - raiz(discriminante))
        SINO
            raiz1 <- (-b + raiz(discriminante)) / (2*a)
            raiz2 <- (2*c) / (-b + raiz(discriminante))
        FIN SI
        RETORNAR "Raíces reales distintas: x1 = ", raiz1, ", x2 = ", raiz2

    SINO SI discriminante = 0 ENTONCES
        // Una raíz real doble
        raiz <- -b / (2*a)
        RETORNAR "Raíz doble y real: x = ", raiz

    SINO
        // Raíces complejas
```

```
        parte_real <- -b / (2*a)
        parte_imaginaria <- (raiz(abs(discriminante))) / (2*a)
        RETORNAR "Raíces complejas: x1 = ", parte_real, " + ", parte_imaginaria, "i, x2 = ", parte_real, " - ", parte_imaginaria, "i"
    FIN SI
FIN FUNCION

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

Resultado:

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 $x < 1$ 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}.

Para esto, debemos:

- Implementar el cálculo del término general de la serie izquierda para un índice n.
- Sumar términos de n=0 hacia adelante hasta que la diferencia entre suma parcial y valor derecho sea menor a 10^-6.
- Retornar el número de términos que fueron necesarios y el valor de la suma parcial.

Pseudocódigo:

```
FUNCION calcular_termino(x, n)
    // Calcular el numerador
    pow_n <- 2^n
    pow_n1 <- 2^(n + 1)
    numerador <- (2^n) * (x^(pow_n - 1)) - (2^(n + 1)) * (x^(pow_n1 - 1))

    // Calcular el denominador
```

```
    denominador <- 1 - x^pow_n + x^pow_n1

    RETORNAR numerador / denominador
FIN FUNCION

FUNCION calcular_suma(x, tolerancia, max_iter)
    suma <- 0.0
    n <- 0
    valor_derecho <- (1 + 2*x) / (1 + x + x^2)

    MIENTRAS VERDADERO HACER
        termino <- calcular_termino(x, n)
        suma <- suma + termino
        diferencia <- ABS(suma - valor_derecho)

        SI diferencia < tolerancia ENTONCES
            SALIR
        FIN SI

        n <- n + 1

        SI n > max_iter ENTONCES
            IMPRIMIR "No se alcanzó la tolerancia en el máximo número de iteraciones."
            SALIR
        FIN SI
    FIN MIENTRAS

    RETORNAR n + 1, suma  // n + 1 términos sumados
FIN FUNCION

// Ejemplo
SI __nombre_del_archivo__ == "__main__" ENTONCES
    x <- 0.25
    tolerancia <- 1e-6
    terminos_usados, suma_aproximada <- calcular_suma(x, tolerancia)
    valor_derecho <- (1 + 2*x) / (1 + x + x^2)

    IMPRIMIR "Número de términos necesarios: ", terminos_usados
    IMPRIMIR "Suma aproximada de la serie: ", suma_aproximada
    IMPRIMIR "Valor del lado derecho exacto: ", valor_derecho
FIN SI
```

Resultado:

Número de términos necesarios: 4

Suma aproximada de la serie: 1.1428571280
Valor del lado derecho exacto: 1.1428571429