Escuela Politécnica Nacional

Asignatura: Métodos Numéricos

Nombre: Francisco Ulloa

Fecha: 5 de noviembre de 2025

Repositorio de GitHub

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

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

Usaremos corte a 3 cifras significativas en cada suma parcial.

Orden directo (i=1..10):

Términos:
$\frac{1}{1^2}=1,\ \frac{1}{2^2}=0.25,\ \frac{1}{3^2}=0.111111\ldots,\ \frac{1}{4^2}=0.0625,\ \frac{1}{5^2}=0.04,$
$\frac{1}{6^2}=0.027777\ldots,\ \frac{1}{7^2}=0.020408\ldots,\ \frac{1}{8^2}=0.015625,\ \frac{1}{9^2}=0.0123456\ldots,\ \frac{1}{10^2}=0.01$

Suma parcial (con corte a 3 cifras significativas después de cada adición):
$S_1 = 1$
$S_2 = \text{chop}_3(1+0.25) = 1.25$
$S_3 = \text{chop}_3(1.25+0.111111\ldots) = 1.36$
$S_4 = \text{chop}_3(1.36+0.0625) = 1.42$
$S_5 = \text{chop}_3(1.42+0.04) = 1.46$
$S_6 = \text{chop}_3(1.46+0.027777\ldots) = 1.48$
$S_7 = \text{chop}_3(1.48+0.020408\ldots) = 1.50$
$S_8 = \text{chop}_3(1.50+0.015625) = 1.51$
$S_9 = \text{chop}_3(1.51+0.0123456\ldots) = 1.52$
$S_{10} = \text{chop}_3(1.52+0.01) = \boxed{1.53}$

Orden inverso (i=10..1):

Términos en orden inverso: $0.01,\ 0.0123456\ldots,\ 0.015625,\ \ldots,\ 0.25,\ 1$

Suma parcial (corte 3 cifras después de cada adición):
$S'_1 = 0.01$
$S'_2 = \text{chop}_3(0.01+0.0123456\ldots) = 0.022$
$S'_3 = \text{chop}_3(0.022+0.015625) = 0.037$
$S'_4 = \text{chop}_3(0.037+0.020408\ldots) = 0.057$
$S'_5 = \text{chop}_3(0.057+0.027777\ldots) = 0.084$
$S'_6 = \text{chop}_3(0.084+0.04) = 0.124$
$S'_7 = \text{chop}_3(0.124+0.0625) = 0.186$
$S'_8 = \text{chop}_3(0.186+0.111111\ldots) = 0.297$
$S'_9 = \text{chop}_3(0.297+0.25) = 0.547$
$S'_{10} = \text{chop}_3(0.547+1) = \boxed{1.54}$

Valores de referencia (alta precisión):

$$\sum_{i=1}^{10} \frac{1}{i^2} = 1.54976773\ldots$$

Errores absolutos:
Orden directo: $|1.54976773 - 1.53| \approx 0.01977$
Orden inverso: $|1.54976773 - 1.54| \approx 0.00977$

Conclusión a): la **orden inverso** (sumar primero los términos más pequeños) es más preciso. Porque la aritmética de corte provoca pérdida de precisión cuando se suman números de tamaños muy distintos. Sumar de menor a mayor reduce esa pérdida acumulada. **b.** $\sum_{i=1}^{10}(\frac{1}{i^3})$ **primero por** $\frac{1}{1} + \frac{1}{8} + \frac{1}{27} + ... + \frac{1}{1000}$

y luego por $\frac{1}{1000} + \frac{1}{729} + ... + \frac{1}{1}$

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

Orden directo (i=1..10): términos:
$1,\ \frac{1}{2^3} = 0.125,\ \frac{1}{3^3} = 0.037037\ldots,\ \frac{1}{4^3} = 0.015625,\ \ldots,\ \frac{1}{10^3} = 0.001$

Suma parcial (corte 3 cifras):
$S_1 = 1$
$S_2 = \text{chop}_3(1 + 0.125) = 1.12$
$S_3 = \text{chop}_3(1.12 + 0.037037\ldots) = 1.15$
$S_4 = \text{chop}_3(1.15 + 0.015625) = 1.16$
$S_5 = \text{chop}_3(1.16 + 0.008) = 1.16$
$S_6 = \text{chop}_3(1.16 + 0.0046296\ldots) = 1.16$
$S_7 = \text{chop}_3(1.16 + 0.00291545\ldots) = 1.16$
$S_8 = \text{chop}_3(1.16 + 0.001953125) = 1.16$
$S_9 = \text{chop}_3(1.16 + 0.0013717\ldots) = 1.16$
$S_{10} = \text{chop}_3(1.16 + 0.001) = \boxed{1.16}$

Orden inverso (i=10..1): términos pequeños primero
Suma parcial (corte 3 cifras):
$S'_1 = 0.001$
$S'_2 = \text{chop}_3(0.001 + 0.0013717\ldots) = 0.002$
$S'_3 = \text{chop}_3(0.002 + 0.001953125) = 0.003$
$S'_4 = \text{chop}_3(0.003 + 0.00291545\ldots) = 0.005$
$S'_5 = \text{chop}_3(0.005 + 0.0046296\ldots) = 0.009$
$S'_6 = \text{chop}_3(0.009 + 0.008) = 0.017$
$S'_7 = \text{chop}_3(0.017 + 0.015625) = 0.032$
$S'_8 = \text{chop}_3(0.032 + 0.037037\ldots) = 0.069$
$S'_9 = \text{chop}_3(0.069 + 0.125) = 0.194$
$S'_{10} = \text{chop}_3(0.194 + 1) = \boxed{1.19}$

Valor real:

$$\sum_{i=1}^{10} \frac{1}{i^3} = 1.19753199\ldots$$

Errores absolutos:
Orden directo: $|1.19753199 - 1.16| \approx 0.03753$
Orden inverso: $|1.19753199 - 1.19| \approx 0.00753$

Conclusión b): nuevamente **sumar de menor a mayor** (orden inverso) es más preciso por la misma razón: minimiza la pérdida de información del término pequeño al sumarlo a acumulados grandes cuando se aplica corte (truncamiento).

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

Serie:
La serie de Maclaurin para $\arctan(x)$ es

$$\arctan(x) = \sum_{i=1}^{\infty} (-1)^{i+1} \frac{x^{2i-1}}{2i-1}.$$

Para $x = 1$ obtenemos

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

Resto por criterio de serie alternante:
Si $P_n(1) = \sum_{i=1}^{n} (-1)^{i+1} \frac{1}{2i-1}$ es el polinomio truncado de orden n, entonces el error de truncamiento satisface

$$|\arctan(1) - P_n(1)| \leq \frac{1}{2n+1},$$

porque el siguiente término (en magnitud) es $\frac{1}{2n+1}$ y la serie es alternante con términos decrecientes en magnitud.

Multiplicando por 4 (ya que usamos $4P_n(1)$ para aproximar π) obtenemos la cota:

$$|\pi - 4P_n(1)| \leq \frac{4}{2n+1}.$$

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

Usamos la cota anterior y exigimos

$$\frac{4}{2n+1} < 10^{-3}.$$

Resolviendo:

$$2n + 1 > \frac{4}{10^{-3}} = 4000 \quad \Longrightarrow \quad n > \frac{4000 - 1}{2} = 1999.5.$$

Por lo tanto el menor entero n que cumple la desigualdad es

$$\boxed{n = 2000.}$$

b. El elnguaje de programación C++ requiere que el valor de π **se encuentre dentro de** 10^{-10}**. ¿Cuántos términos de la serie se necessitan sumar para obtener este grado de presición?**
b) Condición $|4P_n(1) - \pi| < 10^{-10}$:
Usamos la misma cota:

$$\frac{4}{2n+1} < 10^{-10}.$$

Resolviendo:

$$2n+1 > \frac{4}{10^{-10}} = 4\times 10^{10} \quad \Longrightarrow \quad n > \frac{4\cdot 10^{10}-1}{2}.$$

Por lo tanto el menor entero n que satisface la condición es

$$\boxed{n = 20\,000\,000\,000.}$$

Observación: sumar $20\,000\,000\,000$ términos es prácticamente inviable en una implementación directa (tiempo y memoria), pero la fórmula anterior nos da el número teórico de términos requeridos por el criterio alternante.

Pseudocódigo (usar la cota alternante para calcular n):

```
Entrada: tol  # tolerancia deseada para |4 P_n(1) - pi|
Salida: n_min # número mínimo de términos necesarios

# Usando la cota |pi - 4 P_n(1)| <= 4/(2n+1)
1. Calcular N_real = (4 / tol - 1) / 2
2. n_min = ceil(N_real)
3. Retornar n_min
```

Pseudocódigo (si además se desea sumar realmente hasta n):

```
Entrada: n  # número de términos a sumar
Salida: Pn = sum_{i=1}^n (-1)^{i+1} / (2i-1)

1. Pn = 0
2. for i = 1 to n do
3.     term = ((-1)^(i+1)) / (2*i - 1)
4.     Pn = Pn + term
5. end for
6. Retornar Pn
```

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 de π dentro de 10^{-3}.

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

a.

ENTRADA $n, x_1, x_2, ..., x_n$.

SALIDA PRODUCT.
Paso 1 Determine PRODUCT $= 0$
Paso 2 Para $i = 1, 2, ..., n$ haga

Determine PRODUCT $=$ PRODUCT $*x_i$

Paso 3 SALIDA PRODUCT

PARE

b.
ENTRADA $n, x_1, x_2, ..., x_n$.

SALIDA PRODUCT.
Paso 1 Determine PRODUCT = 1
Paso 2 Para $i = 1, 2, ..., n$ haga

Set PRODUCT = PRODUCT $*x_i$

Paso 3 SALIDA PRODUCT

PARE

c.
ENTRADA $n, x_1, x_2, ..., x_n$.

SALIDA PRODUCT.
Paso 1 Determine PRODUCT = 1
Paso 2 Para $i = 1, 2, ..., n$ haga

si $x_i = 0$ entonces determine PRODUCT = 0
SALIDA PRODUCT;
PARE
Determine PRODUCT = PRODUCT $*x_i$

Paso 3 SALIDA PRODUCT

PARE

Al comparar los tres algoritmos tenemos que:
-El algoritmo a en todos los casos da como resultado 0 sin importar los valores de x_i.
-El segundo y tercer algoritmo obtinenen resultados iguales, pero este último mejora al algoritmo b porque si alguno de los x_i es 0, el producto total será 0 y no es necesario seguir multiplicando.
-Finalmente, el algoritmo a dará un resultado correcto solo y solo si todos los valores de x_i son iguales a 0 o si se espera obtener un 0 por respuesta.

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

$$\textbf{?}$$

Ejercicio 5.
a) Cálculo del número de operaciones

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

Para cada i fijo hay i multiplicaciones y $(i-1)$ sumas. Al final se suman los n resultados parciales:

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

$$\text{Sumas} = \sum_{i=1}^{n} (i-1) + (n-1) = \frac{n^2+n-2}{2}.$$

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

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

Definiendo las sumas prefijo $B_i = \sum_{j=1}^{i} b_j$, se obtiene:

$$S = \sum_{i=1}^{n} a_i B_i.$$

$$\text{Multiplicaciones} = n, \qquad \text{Sumas} = 2n - 2.$$

Otra forma equivalente es:

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

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

Algoritmo 6: Suma de una serie en orden inverso

Entrada: $n, x_1, x_2, ..., x_n$
Salida: SUMA

Paso 1: Determine SUMA $= 0$
Paso 2: Para $i = n, n-1, ..., 1$ haga

$$\text{SUMA} = \text{SUMA} + x_i$$

Paso 3: SALIDA SUMA
Paso 4: PARE
7. 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, x_2 que calcule las raíces x_1, x_2 (que pueden ser iguales con conjugados complejos) mediante fórmula para cada raíz

Algoritmo 7: Determinación de las raíces de un polinomio cuadrático

Entrada: a, b, c
Salida: x_1, x_2

Paso 1: Calcular el discriminante $D = b^2 - 4ac$

Paso 2: Si $D \geq 0$ entonces

a) Si $b \geq 0$ entonces

$$x_1 = \frac{2c}{-b - \sqrt{D}} \qquad \textit{(raíz más pequeña, fórmula (1.3))}$$
$$x_2 = \frac{-b - \sqrt{D}}{2a} \qquad \textit{(raíz más grande, fórmula (1.2))}$$

b) Sino

$$x_1 = \frac{-b + \sqrt{D}}{2a} \qquad \textit{(raíz más grande, fórmula (1.2))}$$
$$x_2 = \frac{2c}{-b + \sqrt{D}} \qquad \textit{(raíz más pequeña, fórmula (1.3))}$$

Paso 3: Si $D < 0$ entonces

$$\text{parte\_real} = \frac{-b}{2a}$$
$$\text{parte\_imaginaria} = \frac{\sqrt{|D|}}{2a}$$
$$x_1 = \text{parte\_real} + \text{parte\_imaginaria}i$$
$$x_2 = \text{parte\_real} - \text{parte\_imaginaria}i$$

Paso 4: SALIDA x_1, x_2
Paso 5: PARE

8. Suponga que

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

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}.

Algoritmo 8: Determinación del número de términos necesarios en una serie

Entrada: x, Diferencia
Salida: n

Paso 1: Asignar $x = 0.25$, Diferencia $= 10^{-6}$
Paso 2: Calcular lado derecho:

$$\text{right\_side} = \frac{1+2x}{1+x+x^2}$$

Paso 3: Inicializar suma_izquierda $= 0$, $n = 0$

Paso 4: Mientras sea verdadero, hacer:

a) Calcular numerador:

$$\text{numerador} = 2^n x^{2^n - 1} - 2^{n+1} x^{2^{n+1} - 1}$$

b) Calcular denominador:

$$\text{denominador} = 1 - x^{2^n} + x^{2^{n+1}}$$

c) Calcular término:

$$\text{término} = \frac{\text{numerador}}{\text{denominador}}$$

d) Actualizar suma:

$$\text{suma\_izquierda} = \text{suma\_izquierda} + \text{término}$$

e) Si $|\text{suma\_izquierda} - \text{right\_side}| < \text{Diferencia}$, entonces

SALIR del ciclo

f) Incrementar $n = n + 1$

Paso 5: Mostrar en pantalla:

"Se necesitan $n + 1$ términos para que la suma difiera del lado derecho en menos de 10^{-6}."

Paso 6: PARE