Resumen de Métodos Numéricos

Base lista para pegar fragmentos

Equipo de estudio

25 de septiembre de 2025

Índice

1. Bisección y Regla Falsa

1.1. 1) Objetivo y cuándo usarlo

Qué resuelven. Queremos encontrar una raíz de la ecuación $f(x) = 0$. Muchas veces es muy difícil o directamente imposible **despejar** x de forma exacta (ecuaciones no lineales, trigonométricas, exponenciales, etc.), así que usamos **métodos numéricos** para aproximarla.

Condición inicial indispensable (cambio de signo). Elegimos un intervalo cerrado $[a, b]$ tal que $\boxed{f(a) \cdot f(b) < 0}$: en los extremos, la función debe tener **signos opuestos** (una evaluación negativa y la otra positiva). Bajo continuidad, esto asegura que hay al menos una raíz en el tramo.

Cómo verificar el intervalo. Graficar en GeoGebra $f(x)$ y elegir $[a, b]$ donde se vea el cruce con el eje x y el cambio de signo.

Diferencia entre los métodos (cómo calculan c).

- **Bisección:** $c = \dfrac{a+b}{2}$.
- **Regla Falsa (Falsa Posición):** $c = \dfrac{a\,f(b) - b\,f(a)}{f(b) - f(a)}$.

Criterio de parada (único). Repetimos hasta que el **error elegido** (absoluto o porcentual estimado entre c consecutivos) sea $<$ **tolerancia**.

1.2. 2) Pasos del método (qué se arma y cómo)

Preparación (siempre):

1. **GeoGebra:** graficar $f(x)$ y **elegir** $[a, b]$ con cruce visible y $f(a)f(b) < 0$.
2. Fijar **tolerancia** y el **tipo de error** a controlar (absoluto o porcentual).
3. Inicializar c_{anterior} (p.ej., con a).

Iteración (misma lógica para ambos):

1. **Calcular c:**

$$c = \begin{cases} \dfrac{a+b}{2}, & \text{(Bisección)} \\ \dfrac{a\,f(b) - b\,f(a)}{f(b) - f(a)}, & \text{(Regla Falsa)} \end{cases}$$

2. **Elegir subintervalo por signo:** si $f(a)f(c) > 0 \Rightarrow a \leftarrow c$, si $f(a)f(c) < 0 \Rightarrow b \leftarrow c$. Si $f(c) = 0$ exacto, terminar.
3. **Error y corte:** calcular $|c_{\text{act}} - c_{\text{ant}}|$ (o su porcentaje). Si $<$ tolerancia, parar; si no, $c_{\text{ant}} \leftarrow c$ y repetir.

Cuándo elegir cada error:

- **Error absoluto estimado** $|c_{\text{act}} - c_{\text{ant}}|$: conviene cuando la **escala esperada de la raíz** es conocida y no cercana a 0 (piden "error menor a una cantidad fija").
- **Error porcentual estimado** $\dfrac{|c_{\text{act}} - c_{\text{ant}}|}{|c_{\text{act}}|} \times 100$: conviene cuando **comparás problemas de distinta escala** o la raíz **puede estar cerca de 0**; el criterio queda relativo al tamaño de la solución.

1.3. 3) Ejemplo corto "a mano"

Función: $f(x) = x^2 - 3$. **Intervalo (desde GeoGebra):** $[a, b] = [1, 2]$ con $f(1) = -2$, $f(2) = +1 \Rightarrow f(a)f(b) < 0$.

3.A) Bisección Iteración 1: $c = \frac{1+2}{2} = 1{,}5$. $f(c) = -0{,}75$ (negativo). Como $f(a)f(c) > 0$, $a \leftarrow 1{,}5$.
Nuevo intervalo: [1,5, 2].
Iteración 2: $c = \frac{1{,}5+2}{2} = 1{,}75$. $f(c) = +0{,}0625$. Ahora $f(a)f(c) < 0$, $b \leftarrow 1{,}75$.
Nuevo intervalo: [1,5, 1,75].
(Seguir hasta que el error baje de la tolerancia.)

3.B) Regla Falsa Iteración 1:

$$c = \frac{1 \cdot (+1) - 2 \cdot (-2)}{(+1) - (-2)} = \frac{5}{3} \approx 1{,}6667.$$

$f(c) \approx -0{,}2222 \Rightarrow a \leftarrow 1{,}6667$.
Nuevo intervalo: [1,6667, 2].
Iteración 2: con [1,6667, 2]: $c \approx 1{,}72727$. $f(c) \approx -0{,}01653 \Rightarrow a \leftarrow 1{,}72727$.
(Seguir hasta cumplir tolerancia; suele converger más rápido si la recta "apunta" bien.)

1.4. 4) Mapa del código en C (cómo está dividido y qué hace cada parte)

Archivos: `biseccion.c` y `regla_falsa.c`. Cada uno permite elegir **tipo de error** (absoluto/-porcentual), ingresar **tolerancia** e **intervalo** $[a, b]$, verificar $f(a)f(b) < 0$ y ejecutar el método mostrando tabla de iteraciones.

4.1 Dónde cambiar la función $f(x)$

Listing 1: Función objetivo

```c
double Funcion(double x) {
    /* Ejemplo: cambiar aquí para otra f(x) */
    return x*x*x - 7.0*x + 1.0;
}
```

4.2 Chequeo del cambio de signo

Listing 2: Verificación f(a)·f(b)<0

```c
static int hay_cambio_de_signo(double a, double b) {
    return (Funcion(a) * Funcion(b) < 0.0);
}
```

4.3 Cálculo del punto interior c en cada método **Bisección (punto medio):**

Listing 3: Bisección: c = (a+b)/2

```c
static inline double punto_medio_biseccion(double a, double b) {
    return 0.5*(a + b);
}
```

Regla Falsa (falsa posición):

Listing 4: Regla Falsa: c = (a f(b) - b f(a)) / (f(b)-f(a))

```c
static inline double interseccion_recta(double a, double b) {
    double fa = Funcion(a);
    double fb = Funcion(b);
    return (a*fb - b*fa) / (fb - fa);
}
```

Listing 5: Elegir subintervalo por signo

4.4 Actualización del intervalo (idéntica en ambos)

```c
static void actualizar_intervalo(double *a, double *b, double c) {
    double fa = Funcion(*a);
    double fc = Funcion(c);
    if (fa * fc > 0.0) {
        *a = c; /* la raiz queda en [c, b] */
    } else {
        *b = c; /* la raiz queda en [a, c] */
    }
}
```

4.5 Criterios de parada (tal como se usan) **Error absoluto** $|c_{\text{act}} - c_{\text{ant}}| \leq$ `tolerancia`:

Listing 6: Corte por error absoluto

```c
static int corte_error_absoluto(double c_act, double c_ant, double tol) {
    return fabs(c_act - c_ant) <= tol;
}
```

Error porcentual $\dfrac{|c_{\text{act}} - c_{\text{ant}}|}{|c_{\text{act}}|} \times 100 \leq$ `tolerancia`:

Listing 7: Corte por error porcentual

```c
static int corte_error_porcentual(double c_act, double c_ant, double tol) {
    if (c_act == 0.0) return 0;
    double errp = fabs((c_act - c_ant) / c_act) * 100.0;
    return errp <= tol;
}
```

4.6 Esqueleto típico de iteración en cada ejecutable **Bisección** (punto medio):

Listing 8: Iteración de bisección

```c
double biseccion(double a, double b, double tol, int usa_porcentual) {
    double c_ant = a, c_act;
    for (;;) {
        c_act = punto_medio_biseccion(a, b);
        actualizar_intervalo(&a, &b, c_act);
        if (usa_porcentual ?
            corte_error_porcentual(c_act, c_ant, tol) :
```

```
            corte_error_absoluto(c_act, c_ant, tol)) break;
        c_ant = c_act;
    }
    return c_act;
}
```

Regla Falsa (intersección de la recta):

Listing 9: Iteración de regla falsa

```
double regla_falsa(double a, double b, double tol, int usa_porcentual) {
    double c_ant = a, c_act;
    for (;;) {
        c_act = interseccion_recta(a, b);
        actualizar_intervalo(&a, &b, c_act);
        if (usa_porcentual ?
            corte_error_porcentual(c_act, c_ant, tol) :
            corte_error_absoluto(c_act, c_ant, tol)) break;
        c_ant = c_act;
    }
    return c_act;
}
```

4.7 Menú y salida El `main` de cada archivo permite:

1. Elegir el **tipo de error** (absoluto/porcentual).
2. Ingresar **tolerancia** e **intervalo** $[a, b]$.
3. Validar $f(a)f(b) < 0$ y ejecutar el método elegido, imprimiendo por iteración $(a, b, c, f(c))$ hasta cumplir la tolerancia.

2. Punto Fijo (método abierto)

2.1. 1) Objetivo y cuándo usarlo

Qué resuelve. Buscamos una raíz de $f(x) = 0$. Muchas veces es difícil *despejar* x (aparece en trigonométricas, potencias, logs, etc.). Entonces lo convertimos en *punto fijo*: queremos x tal que $x = g(x)$ e iteramos

$$x_{n+1} = g(x_n).$$

Cómo construir $g(x)$ (dos caminos; probá primero **A** y, si no funciona, usá **B**):

- **Camino A (despeje directo).** Reescribí $f(x) = 0$ para dejar a x solo: $x = g(x)$.
  Ej.: $f(x) = x - \cos x = 0 \;\Rightarrow\; x = \cos x \;\Rightarrow\; g_A(x) = \cos x$.
- **Camino B (sumar x a ambos lados).**

$$f(x) = 0 \;\Rightarrow\; f(x) + x = x \;\Rightarrow\; \boxed{x = g_B(x) = x + f(x)}.$$

(Segunda opción de los apuntes: "sumar x y simplificar".)

Nota. En $f(x) = x - \cos x$, $g_A(x) = \cos x$ y $g_B(x) = x + f(x) = 2x - \cos x$. La ecuación fija $x = g_B(x)$ implica $x = \cos x$ (misma raíz).

Regla CLAVE de convergencia:

$$\boxed{|g'(x)| < 1}$$

cerca de la raíz. Si $|g'(x)| \geq 1$, *no converge* (con esa g y/o esa x_0) y el programa *termina*.

Cuándo conviene. Cuando podés armar una g razonable por A o por B y tenés una semilla x_0 cerca de la raíz (mirando $y = f(x)$ en GeoGebra).

2.2. 2) Pasos del método (con variantes si las hay)

Pasos del método

1. Elegí $g(x)$ y x_0. Empezá por **A (despeje)**; si no es estable, pasá a **B** con $g(x) = x + f(x)$.
2. **Evaluación:** con el valor actual x_n, calculá $g(x_n)$.
3. **Siguiente aproximación:** $x_{n+1} = g(x_n)$.
4. **Error y parada (elegís uno):**

$$\text{absoluto: } |x_n - x_{n-1}| \leq \text{tolerancia}, \qquad \text{porcentual: } \frac{|x_n - x_{n-1}|}{|x_n|} \times 100 \leq \text{tolerancia}.$$

5. **Chequeo de convergencia en ejecución:** estimamos $|g'(x_n)|$. Si $|g'(x_n)| < 1$ seguimos; si $|g'(x_n)| \geq 1$, el programa *corta*.

2.3. 3) Ejemplo corto "a mano" y cosas a tener en cuenta

Problema: $f(x) = x - \cos x = 0$.

Construcción de g (dos caminos):

$$\text{A: } x = \cos x \Rightarrow g_A(x) = \cos x, \qquad \text{B: } g_B(x) = x + f(x) = x + (x - \cos x) = 2x - \cos x.$$

Verificación rápida de convergencia (cerca de la raíz). Para g_A o g_B (coinciden acá): $g'(x) = -\sin x$. En la raíz $x^{\approx 0{,}739}$,

$$|g'(x^)| = |\sin x^| < 1 \Rightarrow \text{converge.}$$

Iteraciones (usamos $g(x) = \cos x$ y $x_0 = 0{,}5$):

$$x_1 = \cos(0{,}5) \approx 0{,}877582562, \quad x_2 = \cos(x_1) \approx 0{,}639012494, \quad x_3 = \cos(x_2) \approx 0{,}802685100, \ \ldots$$

Parar cuando el error elegido $\leq$ tolerancia.

Mini-ejemplo de Camino B (como en tus apuntes). Si $f(x) = 2 - \dfrac{x}{3}$:

$$f(x) + x = x \ \Rightarrow \ x = g(x) = x + \left(2 - \frac{x}{3}\right) = 2 + \frac{2x}{3},$$

y se itera con esa forma.

2.4. 4) Mapa del código en C (cómo está dividido y qué hace cada parte)

Mapa del código en C

- `g(x)`: definís la g del problema (A: despeje; B: $g = x + f(x)$).
- `gp(x)`: derivada aproximada de g para verificar $|g'(x)| < 1$.
- `punto_fijo_error_absoluto(...)` / `punto_fijo_error_porcentual(...)`: hacen $x_{n+1} = g(x_n)$, imprimen tabla y paran con el criterio elegido.
- **Chequeo en código:** si en cualquier paso $|g'(x_n)| \geq 1 \Rightarrow$ *termina*.

3. Newton–Raphson

3.1. 1) Objetivo y cuándo usarlo

Qué resuelve en el curso. Encontrar una **raíz** de $f(x) = 0$.
Qué busca obtener. Un x^* tal que el **error** sea menor que la **tolerancia**.
Cuándo conviene.

- Cuando hay una **buena aproximación inicial** y f es **suave** cerca de la raíz.
- Si la derivada **no se anula** cerca de la raíz.

Cuándo cuidar/no conviene.

- Si $f'(x)$ está **muy cerca de 0** en las iteraciones, puede divergir o "saltar".
- Con puntos iniciales muy malos puede alejarse de la raíz.

3.2. 2) Pasos del método (con variantes si las hay)

Chequeos previos.

1. Elegir x_0 (`valor_inicial`).
2. Fijar `tolerancia` e `iteraciones_maximas`.
3. Confirmar que la derivada **no sea** ~ 0 en el punto actual.

Iteración (Newton clásico).

1. Calcular $f(x_n)$ y $f'(x_n)$.
2. Si $|f'(x_n)|$ es muy pequeño, **detener** con aviso.
3. Actualizar:
$$x_{n+1} \;=\; x_n - \frac{f(x_n)}{f'(x_n)}.$$
4. Calcular el **error** según el criterio activo:
$$\text{error\_absoluto} = |x_{n+1} - x_n|, \qquad \text{error\_porcentual} = \frac{|x_{n+1} - x_n|}{|x_{n+1}|} \cdot 100.$$
5. **Corte** si el error $<$ `tolerancia` o si se alcanzan `iteraciones_maximas`.
6. Si no corta, poner $x_n \leftarrow x_{n+1}$ y volver al paso 1.

Criterios de parada (los que usa tu código).

- `error_absoluto`: $|x_n - x_{n-1}|$.
- `error_porcentual`: $|x_n - x_{n-1}|/|x_n| \cdot 100$.

Política en tu código: corta cuando el **único** criterio activo es $<$ `tolerancia` **o** cuando `it_maximas` se alcanza.

3.3. 3) Ejemplo corto "a mano" y cosas a tener en cuenta

Función del código: $f(x) = x^3 - 7x + 1$.
Derivada numérica (diferencia hacia adelante):

$$f'(x) \;\approx\; \frac{f\big(x + h_{\text{derivada}}\big) - f(x)}{h_{\text{derivada}}}, \qquad h_{\text{derivada}} = 10^{-4}.$$

Parámetros para ilustrar (reemplazables por los tuyos):

$$\texttt{valor\_inicial} = 0{,}5, \quad \texttt{tolerancia} = 10^{-6}, \quad \texttt{iteraciones\_maximas} = 10000.$$

Fórmulas (variables con nombres del código).

$$\texttt{valor\_actual} = \texttt{valor\_anterior} - \frac{f\big(\texttt{valor\_anterior}\big)}{\texttt{derivada\_en\_punto}},$$

$$\texttt{derivada\_en\_punto} = \frac{f\big(\texttt{valor\_anterior} + h_{\text{derivada}}\big) - f\big(\texttt{valor\_anterior}\big)}{h_{\text{derivada}}},$$

$$\texttt{error\_absoluto} = \big|\texttt{valor\_actual} - \texttt{valor\_anterior}\big|,$$

$$\texttt{error\_porcentual} = \frac{\big|\texttt{valor\_actual} - \texttt{valor\_anterior}\big|}{\big|\texttt{valor\_actual}\big|} \cdot 100.$$

Iteración 1.

$$\begin{aligned}
\texttt{valor_anterior} &= 0{,}500000000000,\\
f\big(\texttt{valor_anterior}\big) &= f(0{,}5) = 0{,}5^3 - 7 \cdot 0{,}5 + 1 = -2{,}375000000000,\\
\texttt{derivada_en_punto} &= \frac{f(0{,}5001) - f(0{,}5)}{10^{-4}} = -6{,}249849990003,\\
\texttt{valor_actual} &= 0{,}5 - \frac{-2{,}375000000000}{-6{,}249849990003} = 0{,}119990879173,\\
\texttt{error_absoluto} &= \big|0{,}119990879173 - 0{,}5\big| = 3{,}800091208267 \times 10^{-1},\\
\texttt{error_porcentual} &= \frac{3{,}800091208267 \times 10^{-1}}{0{,}119990879173} \cdot 100\\
&= 3{,}166983386112 \times 10^{2}\,\%.
\end{aligned}$$

Iteración 2.

$$\begin{aligned}
\texttt{valor_anterior} &= 0{,}119990879173,\\
f\big(\texttt{valor_anterior}\big) &= 1{,}617914517973 \times 10^{-1},\\
\texttt{derivada_en_punto} &= -6{,}956770559483,\\
\texttt{valor_actual} &= 0{,}119990879173 - \frac{1{,}617914517973 \times 10^{-1}}{-6{,}956770559483}\\
&= 0{,}143247568526,\\
\texttt{error_absoluto} &= \big|0{,}143247568526 - 0{,}119990879173\big| = 2{,}325668935232 \times 10^{-2},\\
\texttt{error_porcentual} &= \frac{2{,}325668935232 \times 10^{-2}}{0{,}143247568526} \cdot 100\\
&= 1{,}623531176947 \times 10^{1}\,\%.
\end{aligned}$$

Iteración 3.

$$\begin{aligned}
\texttt{valor_anterior} &= 0{,}143247568526,\\
f\big(\texttt{valor_anterior}\big) &= 2{,}064412157885 \times 10^{-4},\\
\texttt{derivada_en_punto} &= -6{,}938397418064,\\
\texttt{valor_actual} &= 0{,}143247568526 - \frac{2{,}064412157885 \times 10^{-4}}{-6{,}938397418064}\\
&= 0{,}143277321969,\\
\texttt{error_absoluto} &= \big|0{,}143277321969 - 0{,}143247568526\big| = 2{,}975344353309 \times 10^{-5},\\
\texttt{error_porcentual} &= \frac{2{,}975344353309 \times 10^{-5}}{0{,}143277321969} \cdot 100\\
&= 2{,}076633142229 \times 10^{-2}\,\%.
\end{aligned}$$

Iteración 4.

$$\begin{aligned}
\texttt{valor_anterior} &= 0{,}143277321969,\\
f(\texttt{valor_anterior}) &= -8{,}984664123801 \times 10^{-10},\\
\texttt{derivada_en_punto} &= -6{,}938371833831,\\
\texttt{valor_actual} &= 0{,}143277321969 - \frac{-8{,}984664123801 \times 10^{-10}}{-6{,}938371833831}\\
&= 0{,}143277321840,\\
\texttt{error_absoluto} &= \left|0{,}143277321840 - 0{,}143277321969\right| = 1{,}294923890338 \times 10^{-10}\\
&< \texttt{tolerancia} \Rightarrow \textbf{corta por } \texttt{error_absoluto},\\
\texttt{error_porcentual} &= \frac{1{,}294923890338 \times 10^{-10}}{0{,}143277321840} \cdot 100\\
&= 9{,}037884528491 \times 10^{-8}\,\% \Rightarrow \text{también cortaría si el criterio fuera porcentual.}
\end{aligned}$$

Cierre del ejemplo. Con estos parámetros, el método converge en **4 iteraciones** a `aproximacion_final` $\approx 0{,}143277321840$, con $f($`aproximacion_final`$)$ cercano a cero en precisión de máquina.

3.4. 4) Mapa del código en C (cómo está dividido y qué hace cada parte)

Archivo: `newton_raphson.c` (tal cual lo pasaste).

Firma y parámetros (main).

- Menú: 1 = error absoluto, 2 = error porcentual.
- Pide `tolerancia` y `x_inicial`, y llama a la variante correspondiente.

Funciones principales.

- `double Funcion(double x)`: define $f(x)$.
- `double Derivada(double x)`: derivada numérica con $h = 10^{-4}$. (Si piden "derivada a mano", reemplazar por la fórmula analítica).

Bucle de Newton (ambas variantes).

- Inicializa: `x_prev = x_inicial`, `it = 0`, `it_maximas = 10000`.
- En cada iteración:
  - `d = Derivada(x_prev)` y **corte** si `fabs(d)` es muy chico (umbral 10^{-4} en absoluto, 10^{-10} en porcentual; igualalos si querés el mismo umbral).
  - `x_actual = x_prev - Funcion(x_prev) / d;`
  - Calcular error según el modo, imprimir y actualizar.
  - Corte por `tolerancia` o por `it_maximas`.

Puntos editables en examen (30 s).

- **Función**: `Funcion`.
- **Criterio de parada**: elegís 1) o 2) en el menú.
- **`tolerancia`**, **`x_inicial`**: por `scanf`.
- **`iteraciones_maximas`**: `it_maximas`.
- **Umbral derivada** ~ 0: los `if (fabs(d) < ...)` (alinealos si querés el mismo umbral).
- **Columnas**: `printf` de cada función (agregar/quitar nombres en español).

4. Secante

4.1. 1) Objetivo y cuándo usarlo

Qué resuelve: aproxima una raíz de $f(x) = 0$ **sin derivada**.

Qué buscamos: un x tal que el **error** elegido sea $<$ **tolerancia**.

Cuándo conviene: cuando no tenemos $f'(x)$ (o calcularla es caro) y contamos con **dos semillas** razonables.

Cuidado: si $f_1 - f_0 \approx 0$ (con $f_0 = \texttt{Funcion}(x\_\texttt{prev})$, $f_1 = \texttt{Funcion}(x\_\texttt{actual})$), el paso **explota**.

4.2. 2) Pasos del método (con variantes si las hay)

Fórmula de actualización (Secante):

$$\boxed{x_{n+1} \;=\; x_n \;-\; f(x_n)\,\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}}$$

Chequeos por iteración (definidos a nivel método):

1. **Evaluar** $f(x_{n-1})$ y $f(x_n)$.
2. Si $|f(x_n) - f(x_{n-1})|$ es **muy pequeño** $\Rightarrow$ **detener** (denominador ≈ 0) o **cambiar semillas**.
3. **Calcular** la nueva aproximación con la fórmula y obtener x_{n+1}.
4. **Error** según el criterio activo:

   $\texttt{error\_absoluto} = |x_{n+1} - x_n|$, $\qquad \texttt{error\_porcentual} = \dfrac{|x_{n+1} - x_n|}{|x_{n+1}|} \cdot 100$ (si $x_{n+1} \approx 0$, usar la variante sin
5. **Corte del bucle:** detener si error $<$ `tolerancia` o si se alcanzó el **máximo de iteraciones**.
6. **Actualización:** $x_{n-1} \leftarrow x_n$, $x_n \leftarrow x_{n+1}$ y repetir.

4.3. 3) Ejemplo corto "a mano" y cosas a tener en cuenta

Función del código: $f(x) = x^3 - 7x + 1$.

Semillas para ilustrar (podés cambiarlas):
$\texttt{x\_prev} = 0{,}000000000000$, $\texttt{x\_actual} = 0{,}500000000000$,
$\texttt{tolerancia} = 10^{-6}$.
(Solo para mostrar el procedimiento. Con estas semillas converge a $\sim 0{,}1432773$*.)*

Fórmula que se usa cada vez:

$$x_{\text{nuevo}} = x_{\text{actual}} - f(x_{\text{actual}})\,\frac{x_{\text{actual}} - x_{\text{prev}}}{f(x_{\text{actual}}) - f(x_{\text{prev}})}.$$

Iteración 1

- $\texttt{x\_prev} = 0{,}000000000000$, $\texttt{x\_actual} = 0{,}500000000000$.
- $f(\texttt{x\_prev}) = f(0{,}0) = 1{,}000000000000$.
- $f(\texttt{x\_actual}) = f(0{,}5) = -2{,}375000000000$.
- $$x_{\text{nuevo}} = 0{,}5 - \frac{(-2{,}375)\,(0{,}5 - 0{,}0)}{-2{,}375 - 1{,}0} = 0{,}148148148148.$$
- **Errores:** $\texttt{err} = |0{,}148148148148 - 0{,}500000000000| = 3{,}518519 \times 10^{-1}$, $\texttt{errp} = 2{,}375000 \times 10^{2}\,\%$.
- **Actualizar:** $\texttt{x\_prev} \leftarrow 0{,}500000000000$, $\texttt{x\_actual} \leftarrow 0{,}148148148148$.

Iteración 2

- $f(0{,}5) = -2{,}375000000000$, $\quad f(0{,}148148148148) = -0{,}033785500178$.
- $$\begin{aligned} x_{\text{nuevo}} &= 0{,}148148148148 - \frac{(-0{,}033785500178)\,(0{,}148148148148 - 0{,}5)}{-0{,}033785500178 - (-2{,}375)} \\ &= 0{,}143070659176. \end{aligned}$$

- **Errores:** $\texttt{err} = 5{,}077489 \times 10^{-3}$, $\texttt{errp} = 3{,}548938 \times 10^{0}\,\%$.
- **Actualizar:** $\texttt{x\_prev} \leftarrow 0{,}148148148148$, $\texttt{x\_actual} \leftarrow 0{,}143070659176$.

Iteración 3

- $f(0{,}148148148148) = -0{,}033785500178$, $f(0{,}143070659176) = 1{,}433929636 \times 10^{-3}$.
- $x_{\text{nuevo}} = 0{,}143277384894$.
- **Errores:** $\texttt{err} = 2{,}067257 \times 10^{-4}$, $\texttt{errp} = 1{,}442836 \times 10^{-1}\,\%$.
- **Actualizar:** $\texttt{x\_prev} \leftarrow 0{,}143070659176$, $\texttt{x\_actual} \leftarrow 0{,}143277384894$.

Iteración 4

- $f(0{,}143070659176) = 1{,}433929636 \times 10^{-3}$, $f(0{,}143277384894) = -4{,}37499 \times 10^{-7}$.
- $x_{\text{nuevo}} = 0{,}143277321840$.
- **Errores:** $\texttt{err} = 6{,}305386 \times 10^{-8}$, $\texttt{errp} = 4{,}400826 \times 10^{-5}\,\%$.
- Con $\texttt{tolerancia} = 10^{-6}$, **corta en 4 iteraciones** (por error absoluto o porcentual).

4.4. 4) Mapa del código en C (cómo está dividido y qué hace cada parte)

Archivo: `secante.c`
Flujo general del programa (sin pegar código):

1. **Menú:** elegir **criterio de parada** (error absoluto o porcentual).
2. **Entradas:** leer **tolerancia**, **x0 (primera aproximación)** y **x1 (segunda aproximación)**.
3. **Inicialización:** preparar las dos aproximaciones de trabajo, el **contador** y el **límite** de iteraciones.
4. **Bucle de iteración (lógica de Secante):**
   - Evaluar f en las dos últimas aproximaciones.
   - Verificar que el **denominador** $f(x_n) - f(x_{n-1})$ **no sea** ≈ 0; si lo es, detener con mensaje.
   - Calcular la **nueva aproximación** con la fórmula de la secante.
   - Calcular el **error** según el modo (absoluto o porcentual).
   - **Imprimir** la línea de iteración; **actualizar** y continuar salvo que error $<$ `tolerancia` o se alcance el **máximo** de iteraciones.
5. **Salida final:** mostrar la **aproximación** obtenida, $f(x)$, el **error final** y la **cantidad de iteraciones**.

Puntos editables típicos (sin código): la **función** $f(x)$; el **criterio de parada**; **tolerancia**, **x0**, **x1** y el **máximo de iteraciones**; **formato** de impresión por iteración.

Notas de robustez (alto nivel): tratar $f(x_n) - f(x_{n-1}) \approx 0$ con un **umbral** pequeño; asegurar que el **límite de iteraciones** esté definido en ambas variantes.

5. Eliminación Gaussiana con pivoteo parcial

5.1. 1) Objetivo y cuándo usarlo

Qué resuelve en el curso. Sistemas lineales $A\,x = b$ de tamaño $n \times n$.
Qué busca obtener. El **vector incógnita** x.
Cómo se ve el sistema (expandido):

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1, \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2, \\ \quad\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n. \end{cases}$$

Qué se obtiene durante el proceso. Al terminar la "eliminación hacia adelante" queda una **matriz triangular superior** U (ceros debajo de la diagonal). Con esa U se resuelve x por **sustitución hacia atrás**. Además, el **determinante** se calcula como **producto de los elementos de la diagonal** de U.

Por qué pivoteo parcial (en palabras simples). Si el número del pivote (el de la diagonal en la columna que estoy trabajando) es **muy chico**, dividir por él amplifica errores. Entonces:

- **Regla práctica:** si el valor del pivote (en esa columna) es $<$ **0,01** en valor absoluto, busco en esa **misma columna** la **fila** con el **mayor valor absoluto** y **intercambio** filas para usar ese número como pivote.
- Con eso, las cuentas son más estables y la eliminación funciona mejor.

Cuándo cuidar (dicho simple). Si hay ecuaciones que casi repiten información, el sistema puede **no tener solución única** o ser **muy sensible** al redondeo. El pivoteo ayuda, pero no hace milagros.

5.2. 2) Pasos del método (con variantes si las hay)

Eliminación hacia adelante con pivoteo parcial (paso a paso):
Para cada **columna** $i = 1, 2, \ldots, n-1$:

1. **Elegir pivote**
   Mirá el número de la **diagonal** en esa columna (posición (i, i)). **Si su valor absoluto** $< 0{,}01$, en esa **misma columna** (de la fila i hacia abajo) buscá el número con **mayor valor absoluto** y **intercambiá** esa fila con la fila i para dejar ese número como **pivote**.

2. **Anular debajo del pivote (dejar ceros)**
   Para cada fila $j > i$, calculá el **multiplicador**

$$\text{multiplicador} = \frac{a_{j,i}}{a_{i,i}}$$

   y actualizá la fila j restando ese multiplicador por la fila i en **todas sus columnas** y también en b_j:

$$\text{Fila } j \leftarrow \text{Fila } j - \text{multiplicador} \times \text{Fila } i.$$

   Con eso, **debajo del pivote quedan ceros**.

Sustitución hacia atrás (resolver x con la triangular):

$$x_n = \frac{b_n}{u_{nn}},$$
$$x_i = \frac{b_i - \sum_{k=i+1}^{n} u_{ik}\, x_k}{u_{ii}}, \quad i = n-1, \ldots, 1.$$

Determinante (cómo se calcula acá). Tomás los números de la **diagonal** de la triangular y **los multiplicás**:

$$\det(A) = u_{11} \times u_{22} \times \cdots \times u_{nn}.$$

5.3. 3) Ejemplo corto "a mano" y cosas a tener en cuenta

Ejemplo que activa pivoteo en la primera columna:

$$A = \begin{bmatrix} 0 & 2 & 1 \\ 1 & -2 & -3 \\ 2 & 3 & 1 \end{bmatrix}, \qquad b = \begin{bmatrix} 1 \\ -4 \\ 7 \end{bmatrix}.$$

Paso 1 (columna 1, $i = 0$)
Pivoteo parcial: en la col. 1, $|0|, |1|, |2| \rightarrow$ el mayor es **2** (fila 3). Intercambio $R_1 \leftrightarrow R_3$:

$$\left[\begin{array}{ccc|c} 2 & 3 & 1 & 7 \\ 1 & -2 & -3 & -4 \\ 0 & 2 & 1 & 1 \end{array}\right]$$

Eliminar debajo:
Fila 2: $m_{21} = 1/2 = 0{,}5 \Rightarrow R_2 \leftarrow R_2 - 0{,}5\,R_1 = [0,\ -3{,}5,\ -3{,}5 \mid\ -7{,}5]$.
Fila 3: $m_{31} = 0$ (ya es cero).

Tras el paso 1:

$$\left[\begin{array}{ccc|c} 2 & 3 & 1 & 7 \\ 0 & -3{,}5 & -3{,}5 & -7{,}5 \\ 0 & 2 & 1 & 1 \end{array}\right]$$

Paso 2 (columna 2, $i = 1$)
Pivoteo en subcolumna: $|-3{,}5|, |2| \rightarrow$ queda $-3{,}5$ (sin intercambio).
Eliminar debajo: Fila 3: $m_{32} = 2/(-3{,}5) = -0{,}5714285714 \Rightarrow R_3 \leftarrow R_3 - (-0{,}5714)\,R_2 = [0,\ 0,\ -1 \mid\ -3{,}28571429]$.

Triangular superior y término independiente:

$$U = \begin{bmatrix} 2 & 3 & 1 \\ 0 & -3{,}5 & -3{,}5 \\ 0 & 0 & -1 \end{bmatrix}, \qquad \tilde{b} = \begin{bmatrix} 7 \\ -7{,}5 \\ -3{,}28571429 \end{bmatrix}.$$

Sustitución hacia atrás:
$x_3 = -3{,}28571429/-1 = 3{,}28571429$.
$x_2 = \dfrac{-7{,}5 - (-3{,}5)\,x_3}{-3{,}5} = \dfrac{-7{,}5 + 11{,}5}{-3{,}5} = -1{,}14285714$.
$x_1 = \dfrac{7 - 3\,x_2 - 1\,x_3}{2} = \dfrac{7 - 3(-1{,}14285714) - 3{,}28571429}{2} = 3{,}57142857$.

Solución: $x \approx (\,3{,}57142857,\ -1{,}14285714,\ 3{,}28571429\,)^{\top}$.

Determinante (producto de la diagonal de U): $2 \times (-3{,}5) \times (-1) = \mathbf{7}$.

5.4. 4) Mapa del código en C (cómo está dividido y qué hace cada parte)

Archivo principal: `eliminacion_gaussiana_pivoteo_parcial.c`
Flujo general del programa (sin pegar código):

1. **Elegir entrada de datos:**
   - **Teclado:** pedir n, luego cargar A y b.
   - **Archivo `datos.txt`:** leer n; luego A fila por fila; al final b.
2. **Mostrar lo leído:** imprimir la **matriz** A y el **vector** b.
3. **(Opcional) Norma de A:** calcular $\|A\|_F = \sqrt{\sum a_{ij}^2}$ si el usuario lo pide.
4. **Eliminación con pivoteo parcial:**
   - En cada columna, si el pivote $|a_{ii}| < 0{,}01$, buscar en la columna el **mayor valor absoluto** y **permuta** filas para usarlo de pivote.
   - **Anular** todo lo que esté **debajo** del pivote con los **multiplicadores**, dejando **ceros** debajo.
5. **Impresión intermedia:** mostrar la **matriz aumentada** $[A|b]$ ya **triangular**.
6. **Determinante:** calcular el **producto de la diagonal** de la triangular.
7. **Chequeo de unicidad:** si ese producto es **muy cercano a 0**, informar que **no hay solución única** y terminar.
8. **Sustitución hacia atrás:** calcular y mostrar el **vector incógnita** x.

Puntos editables rápidos (modo examen): umbral del pivoteo (0,01); tolerancia para "determinante ≈ 0" (10^{-12}); formato de impresiones; mostrar multiplicadores si lo piden.

6. Jacobi (método iterativo para $A\,x = b$)

6.1. 1) Objetivo y cuándo usarlo

Qué resuelve en el curso. Sistemas lineales $A\,x = b$.
Qué busca obtener. El **vector incógnita** x.

Cómo se ve el sistema (expandido).

$$\begin{aligned} &a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1,\\ &a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2,\\ &\cdots\\ &a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n. \end{aligned}$$

Muy importante (propio de Jacobi):

- En **Jacobi no se hace pivoteo ni se permutan filas durante el método**.
- Si la matriz **no es diagonalmente dominante**, Jacobi **puede no converger**. Por eso el código **solo muestra una advertencia** (no reordena).
- **Diagonalmente dominante (por filas)** significa: para cada fila i,

$$|a_{ii}| \;>\; \sum_{j\neq i} |a_{ij}|$$

(equivalente a: "la suma de todos los valores de la fila, excepto el pivote, debe ser menor al pivote").

Cuándo conviene. Cuando A es (idealmente) diagonalmente dominante o cuando probás unas vueltas y **ves que el error baja**.

6.2. 2) Pasos del método (con variantes si las hay)

Chequeos previos.

- **Diagonal no nula:** $a_{ii} \neq 0$ para todo i (si no, no se puede dividir).
- **Advertencia de dominancia:** si no es dominante, se avisa (el código no permuta filas).
- Fijar: `tolerancia`, `iteraciones_maximas` y `tipoError` (1 = absoluto, 2 = relativo porcentual).

Despejes de Jacobi (forma general, "con valores anteriores").

$$x_{i,\text{ nuevo}} \;=\; \frac{b_i \;-\; \sum_{j\neq i} a_{ij}\, x_{j,\text{ anterior}}}{a_{ii}}, \qquad i = 1, \ldots, n.$$

(En Jacobi, *todos* los $x_{i,\text{nuevo}}$ se calculan con los $x_{j,\text{anterior}}$.)

Criterios de parada (sin residuo).

$$\texttt{error\_absoluto} = \max_i |x_{i,\text{nuevo}} - x_{i,\text{anterior}}|, \qquad \texttt{error\_relativo} = \max_i \frac{|x_{i,\text{nuevo}} - x_{i,\text{anterior}}|}{|x_{i,\text{nuevo}}|} \times 100.$$

Política de corte.

- Cortar cuando el criterio elegido $<$ `tolerancia`, o al llegar a `iteraciones_maximas`.
- Además, **verificar** que el **error vaya disminuyendo**; si **aumenta** respecto de la vuelta anterior, **terminar** (como hace el código).

6.3. 3) Ejemplo corto "a mano" y cosas a tener en cuenta

Sistema (2×2):

$$\begin{cases} 4x + y = 9,\\ x + 3y = 8. \end{cases}$$

Despejes de Jacobi (con valores anteriores).

$$x_{\text{nuevo}} = \frac{9 - y_{\text{anterior}}}{4}, \qquad y_{\text{nuevo}} = \frac{8 - x_{\text{anterior}}}{3}.$$

Parámetros para ilustrar (podés cambiarlos). `x_anterior` $= 0$, `y_anterior` $= 0$, `tolerancia` $= 10^{-3}$. **Error usado:** `error_absoluto` $= \max(|\Delta x|, |\Delta y|)$.

Iteración 1

$$x_{\text{nuevo}} = \frac{9-0}{4} = 2{,}25, \qquad y_{\text{nuevo}} = \frac{8-0}{3} = 2{,}6666667.$$

$$\texttt{error\_absoluto} = \max\big(|2{,}25-0|,\ |2{,}6667-0|\big) = \mathbf{2{,}6667}.$$

Actualizar: `x_anterior` $\leftarrow 2{,}25$, `y_anterior` $\leftarrow 2{,}6666667$.

Iteración 2

$$x_{\text{nuevo}} = \frac{9-2{,}6666667}{4} = 1{,}5833333, \qquad y_{\text{nuevo}} = \frac{8-2{,}25}{3} = 1{,}9166667.$$

$$\texttt{error\_absoluto} = \max\big(|1{,}5833-2{,}25|,\ |1{,}9167-2{,}6667|\big) = \mathbf{0{,}75}.$$

Actualizar: `x_anterior` $\leftarrow 1{,}5833333$, `y_anterior` $\leftarrow 1{,}9166667$.

Iteración 3

$$x_{\text{nuevo}} = \frac{9-1{,}9166667}{4} = 1{,}7708333, \qquad y_{\text{nuevo}} = \frac{8-1{,}5833333}{3} = 2{,}1388889.$$

$$\texttt{error\_absoluto} = \max\big(|1{,}7708-1{,}5833|,\ |2{,}1389-1{,}9167|\big) = \mathbf{0{,}2222222}.$$

Actualizar: `x_anterior` $\leftarrow 1{,}7708333$, `y_anterior` $\leftarrow 2{,}1388889$.

Iteración 4

$$x_{\text{nuevo}} = \frac{9-2{,}1388889}{4} = 1{,}7152778, \qquad y_{\text{nuevo}} = \frac{8-1{,}7708333}{3} = 2{,}0763889.$$

$$\texttt{error\_absoluto} = \max\big(|1{,}7153-1{,}7708|,\ |2{,}0764-2{,}1389|\big) = \mathbf{0{,}0625}.$$

Cierre del ejemplo. El **error baja** $2{,}6667 \to 0{,}75 \to 0{,}2222 \to 0{,}0625$. Con `tolerancia` $= 10^{-3}$, seguir hasta que el error quede por debajo de 0,001. (La solución exacta es $x = 1{,}8$, $y \approx 2{,}0667$.)

6.4. 4) Mapa del código en C (cómo está dividido y qué hace cada parte)

Archivo: un único `.c` con las funciones indicadas.
Funciones y flujo:

- `leer_n_desde_archivo`, `leer_sistema_desde_archivo`: leen n, luego A y b de `datos.txt`.
- `imprimirMatriz(A,b,n)`: muestra la aumentada $[A|b]$.
- `verificarDiagonalDominante(A,n)`: para cada fila i, computa $\sum_{j\neq i} |A[i][j]|$ y lo compara con $|A[i][i]|$. **Solo imprime advertencia**; **no** permuta filas (**no** hay pivoteo en Jacobi).
- `jacobi(A,b,n,tol,tipoError)` (núcleo):
  1. Estado inicial: `x[MAX] = {0}`.
  2. Para cada i: $xn[i] \leftarrow \dfrac{b[i] - \sum_{j\neq i} A[i][j]\cdot x[j]}{A[i][i]}$ usando **solo** los `x` **anteriores**.
  3. Error (según `tipoError`): absoluto $= |xn[i]-x[i]|$; relativo $= \dfrac{|xn[i]-x[i]|}{|xn[i]|} \times 100$ si $|xn[i]| > 10^{-12}$. Toma el **máximo** como `error`.
  4. Monitoreo: si `error > error_viejo`, **corta** ("no converge").
  5. Corte normal: cuando `error <= tol` o si `iter` llega a `maxIter = 10000`.
  6. Salida: imprime `x[i]` (vector incógnita), `error` final y `iter`.

Puntos editables (modo examen, 30 s).

- `tolerancia`, `tipoError` (1 o 2), `maxIter`.
- Mensajes e impresiones por iteración (agregar/quitar columnas y decimales).
- Umbral de división segura en la diagonal: `1e-14`.

7. Gauss–Seidel (con SOR)

1) Objetivo y cuándo usarlo

Qué resuelve. Sistemas lineales $A\,x = b$.
Qué buscamos. El **vector incógnita** x.
Sistema (expandido).

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n \end{aligned}$$

Puntos clave.

- En **Gauss–Seidel no se hace pivoteo** (no se cambian filas).
- Si la matriz **no es diagonalmente dominante**, el método **puede no converger** ⇒ se **muestra advertencia**.
- **Diagonalmente dominante (por filas):** para cada fila i, $|a_{ii}| > \sum_{j\neq i} |a_{ij}|$.

  (En palabras: *la suma de todos los valores de la fila excepto el pivote debe ser menor que el pivote.*)

2) Pasos del método (con criterios de corte)

Chequeos previos.

- **Ningún valor de la diagonal puede ser 0:** para todo i, $a_{ii} \neq 0$ (si alguno vale 0, no se puede dividir en la actualización).
- Advertir si **no** hay dominancia diagonal.
- Fijar **tolerancia** y **tipo de error** (1 = absoluto, 2 = relativo porcentual).

Despeje general (vuelta k+1). Para cada ecuación $i = 1..n$:

$$x_i^{(k+1)} = \frac{b_i - \sum_{j<i} a_{ij}\, x_j^{(k+1)} - \sum_{j>i} a_{ij}\, x_j^{(k)}}{a_{ii}}$$

*Nota: no hace falta memorizar esta fórmula; es más fácil entenderla con el **ejemplo** de abajo.*

Criterios de parada (sin residuo).

$$\text{error absoluto} = \max_i \left|x_i^{(k+1)} - x_i^{(k)}\right| \qquad \text{error relativo } (\%) = \max_i \left(\frac{\left|x_i^{(k+1)} - x_i^{(k)}\right|}{\left|x_i^{(k+1)}\right|}\right) \cdot 100$$

Política de corte.

- **Cortar cuando error < tolerancia**.
- **Además, verificar que el error disminuya**; si **aumenta** respecto de la vuelta anterior, **detener** (no converge).

3) Ejemplo corto "a mano" (fracciones y "anteriores")

Sistema 2×2

$$\begin{cases} 4x + y = 9, \\ x + 3y = 8. \end{cases}$$

Despejes (para usar en Gauss–Seidel).

$$x_{\text{nuevo}} = \frac{9 - y_{\text{disponible}}}{4}, \qquad y_{\text{nuevo}} = \frac{8 - x_{\text{disponible}}}{3}.$$

En la **misma vuelta**: primero x_{nuevo} con y_{anterior}; luego y_{nuevo} usando x_{nuevo}.

Parámetros para ilustrar. $x_{\text{anterior}} = 0$, $y_{\text{anterior}} = 0$, tolerancia $= 10^{-3}$.

Error usado aquí (absoluto "común" por variable):

$$|x_{\text{nuevo}} - x_{\text{anterior}}| \quad \text{y} \quad |y_{\text{nuevo}} - y_{\text{anterior}}|.$$

Corte en el ejemplo: cuando **ambos** son $<$ tolerancia.

Iteración 1

$$x_{\text{nuevo}} = \frac{9 - 0}{4} = 2{,}25, \qquad y_{\text{nuevo}} = \frac{8 - 2{,}25}{3} = 1{,}9166667.$$

Actualizar: $x_{\text{ant}} \leftarrow 2{,}25$, $y_{\text{ant}} \leftarrow 1{,}9166667$.

Iteración 2

$$x_{\text{nuevo}} = \frac{9 - 1{,}9166667}{4} = 1{,}7708333, \qquad y_{\text{nuevo}} = \frac{8 - 1{,}7708333}{3} = 2{,}0763889.$$

Actualizar: $x_{\text{ant}} \leftarrow 1{,}7708333$, $y_{\text{ant}} \leftarrow 2{,}0763889$.

Iteración 3

$$x_{\text{nuevo}} = \frac{9 - 2{,}0763889}{4} = 1{,}7309028, \qquad y_{\text{nuevo}} = \frac{8 - 1{,}7309028}{3} = 2{,}0896991.$$

Actualizar: $x_{\text{ant}} \leftarrow 1{,}7309028$, $y_{\text{ant}} \leftarrow 2{,}0896991$.

Iteración 4

$$x_{\text{nuevo}} = \frac{9 - 2{,}0896991}{4} = 1{,}7275752, \qquad y_{\text{nuevo}} = \frac{8 - 1{,}7275752}{3} = 2{,}0908083.$$

Actualizar: $x_{\text{ant}} \leftarrow 1{,}7275752$, $y_{\text{ant}} \leftarrow 2{,}0908083$.

Iteración 5

$$x_{\text{nuevo}} = \frac{9 - 2{,}0908083}{4} = 1{,}7272979, \qquad y_{\text{nuevo}} = \frac{8 - 1{,}7272979}{3} = 2{,}0909007.$$

Aquí $|x_{\text{nuevo}} - x_{\text{ant}}| = 2{,}773 \times 10^{-4}$ y $|y_{\text{nuevo}} - y_{\text{ant}}| = 9{,}24 \times 10^{-5}$, ambos $< 10^{-3} \Rightarrow$ **corta**.

4) Mapa del código en C (flujo general y dónde tocar)

Archivo: `gauss_seidel.c` (incluye opción **SOR**).

1. **Entrada de datos:** teclado o archivo (`leer_n_desde_archivo`, `leer_sistema_desde_archivo`).
2. **Mostrar** $[A|b]$ (`imprimirMatriz`).
3. **Chequeo** de **diagonal dominante** (`verificarDiagonalDominante`): si no lo es, **solo advierte** (no cambia filas).
4. **Elegir parámetros:** `tolerancia`, `tipoError` (1 absoluto / 2 relativo porcentual), y **modo**:
   - **Sin relajación:** $w = 1$ (Gauss–Seidel puro).
   - **Con relajación (SOR):** ingresar w con $0 < w < 2$.

5. **Iteración** (`gaussSeidel`) por $i = 1..n$ en cada vuelta:
   - Calcular $x_i^{\text{nuevo}} = \dfrac{b_i - \sum_{j \neq i} a_{ij}\, x_j}{a_{ii}}$ usando ya los **actualizados** de GS.
   - Unificar GS y SOR con: $x_i \leftarrow (1-w)\, x_i^{\text{viejo}} + w\, x_i^{\text{nuevo}}$.
   - **Error** (según modo) y **corte** cuando error $<$ tolerancia; si el **error aumenta**, **detener**.
6. **Salida:** imprime $x[i]$, **error final**, **cantidad de iteraciones**.

Anexo: Relajación (SOR) para Gauss–Seidel

Idea. Se introduce un **factor** w que "acorta" o "alarga" el paso de GS:

$$x_i^{(k+1)} = (1-w)\, x_i^{(k)} + w \underbrace{\frac{b_i - \sum_{j<i} a_{ij}\, x_j^{(k+1)} - \sum_{j>i} a_{ij}\, x_j^{(k)}}{a_{ii}}}_{\text{valor de Gauss–Seidel}}$$

Casos.

- $w = 1 \Rightarrow$ **Gauss–Seidel clásico**.
- $0 < w < 1 \Rightarrow$ **sub–relajación** (pasos más cortos), útil si oscila.
- $1 < w < 2 \Rightarrow$ **sobre–relajación** (pasos más largos), suele **acelerar** si ya converge.

Cómo elegir w (práctico).

- Probar primero $w = 1$. Si converge pero **lento**, subir a $w \in [1{,}1, 1{,}5]$.
- Si **oscila** o el error **sube**, bajar a $w \in [0{,}8, 0{,}95]$.

Nota de uso en ejercicios. *El valor de w nos lo van a dar en el ejercicio*; con ese dato se ve claramente cuándo usar SOR (si piden GS puro, $w = 1$; si dan otro w, aplicar la fórmula anterior).

Criterio de corte. No cambia: **cortar** cuando error $<$ tolerancia, y **detener** si el **error aumenta** entre vueltas.

Interpolación (Ajuste de curvas)

1) Objetivo y cuándo usarlo

- **Qué nos dan:** una gran cantidad de puntos conocidos (x, y).
- **Qué buscamos:** una función polinómica de grado n que pase **exactamente por todos esos puntos**.
- **Cuándo conviene:** cuando queremos reconstruir la función entre los puntos que ya tenemos, sin perder información.

2) Idea general

Dado un conjunto de puntos conocidos (expresados verticalmente):

$$(x_0, y_0), \quad (x_1, y_1), \quad (x_2, y_2), \ \ldots, \ (x_n, y_n),$$

buscamos un polinomio $P_n(x)$ de grado n tal que

$$P_n(x_i) = y_i \quad \text{para todo } i = 0, 1, \ldots, n.$$

3) ¿Qué nos pueden pedir en un ejercicio?

En general, hay dos tipos de preguntas con el método de Lagrange:

1. **Encontrar el polinomio interpolador:** escribir $P_n(x)$, la función que pasa por todos los puntos dados.
2. **Encontrar un valor desconocido:** nos dan un valor de X_{copete} (algún x no listado en los puntos), y pedimos $P_n(X_{\text{copete}})$, es decir, calcular la y asociada en la curva interpoladora.

⇒ En los ejercicios puede pedirse *una cosa o la otra*, o incluso ambas.

4) Método de Lagrange paso a paso

A) Cuando nos piden un valor de la función en X_{copete}. Fórmula general de los polinomios base:

$$L_i(x) = \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{x - x_j}{x_i - x_j},$$

donde cada x_j son los valores de x **de los demás puntos en los que no estoy parado**.

△ **Nota:** no es necesario memorizar la fórmula, porque se entiende mucho más fácil con el ejemplo.

Fórmula del polinomio interpolador:

$$P_n(x) = \sum_{i=0}^{n} y_i\, L_i(x).$$

Y si lo que pedimos es un valor puntual:

$$P_n\big(X_{\text{copete}}\big) = \sum_{i=0}^{n} y_i\, L_i\big(X_{\text{copete}}\big).$$

B) Cuando nos piden el polinomio interpolador completo. Se construye combinando los $L_i(x)$:

$$P_n(x) = y_0\, L_0(x) + y_1\, L_1(x) + \cdots + y_n\, L_n(x),$$

y de ahí **ya sale el polinomio funcional** que representa la curva.

Ejemplo con 2 puntos. Puntos: $(1, 2)$ y $(2, 5)$. Polinomio lineal:

$$P(x) = a_0 + a_1 x.$$

Al plantear el sistema:

$$\begin{cases} a_0 + a_1(1) = 2, \\ a_0 + a_1(2) = 5, \end{cases} \qquad \Rightarrow \quad a_0 = -1,\ a_1 = 3,$$

entonces $P(x) = -1 + 3x$.

5) Ejemplo usando Lagrange

A) Calcular un valor en X_{copete}. Puntos: $(1, 2),\ (2, 3),\ (4, 6)$. Queremos calcular $P_2(X_{\text{copete}} = 3)$.

$$L_0(3) = \frac{(3-2)(3-4)}{(1-2)(1-4)} = \frac{(1)(-1)}{(-1)(-3)} = \tfrac{1}{3},$$
$$L_1(3) = \frac{(3-1)(3-4)}{(2-1)(2-4)} = \frac{(2)(-1)}{(1)(-2)} = 1,$$
$$L_2(3) = \frac{(3-1)(3-2)}{(4-1)(4-2)} = \frac{(2)(1)}{(3)(2)} = \tfrac{1}{3}.$$

Entonces:

$$P_2(3) = 2 \cdot \tfrac{1}{3} + 3 \cdot 1 + 6 \cdot \tfrac{1}{3} = \tfrac{2}{3} + 3 + 2 = 5{,}67.$$

Nota: en el numerador siempre restamos los valores de x de **los otros puntos**; esos x son los x_j de la fórmula.

B) Construir el polinomio interpolador con los mismos puntos.

$$P_2(x) = 2 \cdot \frac{(x-2)(x-4)}{(1-2)(1-4)} \; + \; 3 \cdot \frac{(x-1)(x-4)}{(2-1)(2-4)} \; + \; 6 \cdot \frac{(x-1)(x-2)}{(4-1)(4-2)}$$

$$= \frac{2}{3}(x-2)(x-4) \; - \; \frac{3}{2}(x-1)(x-4) \; + \; 1 \cdot (x-1)(x-2).$$

Desarrollos auxiliares:

$$(x-2)(x-4) = x^2 - 6x + 8, \quad (x-1)(x-4) = x^2 - 5x + 4, \quad (x-1)(x-2) = x^2 - 3x + 2.$$

Sustituyendo y simplificando:

$$P_2(x) = \boxed{\tfrac{1}{6}x^2 \; + \; \tfrac{1}{2}x \; + \; \tfrac{4}{3}}.$$

C) (Opcional) Error absoluto cuando nos piden un valor en X_{copete}. Si conocemos la función "real" f,

$$\boxed{\text{error\_absoluto} \; = \; \left| P_n(X_{\text{copete}}) \; - \; f(X_{\text{copete}}) \right|}.$$

(Análogo si comparamos el valor hallado para un x dado contra el valor "original" provisto por una función de referencia.)

8. Regresión por mínimos cuadrados (lineal y polinómica)

8.1. 1) Objetivo y cuándo usarlo

Contexto (mediciones): los puntos (x_i, y_i) vienen de mediciones con error (instrumentos/entorno), por eso no caen exactamente sobre una función perfecta.
Qué buscamos: la *función* que, dibujada, pasa lo más cerca posible de *todos* los puntos, minimizando $S_r = \sum (f(x_i) - y_i)^2$.
Calidad (lo que queremos calcular): coeficiente de correlación $r = \sqrt{(S_t - S_r)/S_t}$ con $S_t = \sum (\bar{y} - y_i)^2$ y $\bar{y} = \frac{1}{n+1} \sum y_i$. Si $r \approx 1$, lo no explicado suele ser ruido de medición; si es bajo, revisar modelo o cálculos.
Cuándo usar: muchos puntos con ruido; queremos una *tendencia* o un *promedio suavizado* de todos los puntos (recta o polinomio).

8.2. 2) Pasos del método (con variantes si las hay)

Chequeos previos: elegir tipo de función (lineal o polinómica) y verificar puntos suficientes (para grado p, al menos p+1 puntos).
Ecuaciones normales: $A_{\ell m} = \sum x_i^{\ell+m}$, $b_\ell = \sum y_i x_i^\ell$; resolver $A\,a = b$ para $a = (a_0, \ldots, a_p)^\top$.
Caso lineal ($p = 1$) con incógnitas $[a_0, a_1]$: $\begin{bmatrix} n+1 & \sum x_i \\ \sum x_i & \sum x_i^2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} = \begin{bmatrix} \sum y_i \\ \sum x_i y_i \end{bmatrix}$.
Calidad (r): $\bar{y} = \frac{1}{n+1} \sum y_i$, $S_t = \sum (\bar{y} - y_i)^2$, $S_r = \sum (f(x_i) - y_i)^2$, r *(coeficiente de correlación, calidad)* $= \sqrt{(S_t - S_r)/S_t}$; más cerca de 1 $\Rightarrow$ mejor.
Resultado: al resolver $A\,a = b$ obtenemos los coeficientes y por lo tanto la *función* $P_p(x)$ que aproxima a todos los puntos.

8.3. 3) Ejemplo corto "a mano" y cosas a tener en cuenta

Dos puntos: $(2,3)$ y $(4,5)$, recta $(p=1)$.
$\sum x = 6$, $\sum y = 8$, $\sum x^2 = 20$, $\sum x\,y = 26$, $n{+}1 = 2$.
$\begin{bmatrix} 2 & 6 \\ 6 & 20 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} = \begin{bmatrix} 8 \\ 26 \end{bmatrix} \Rightarrow a_0 = 1,\ a_1 = 1,\ P_1(x) = x + 1.$
$\bar{y} = 4$, $S_t = 2$, $S_r = 0$, $r = 1$.

8.4. 4) Mapa del código en C (cómo está dividido y qué hace cada parte)

Archivo: `Regresion_Minimos.c`. Flujo: leer puntos → elegir grado p → armar y resolver $A\,a = b$ → mostrar polinomio → calcular calidad (S_r, S_t, r).

Partes:

- **Entrada:** `leer_puntos_desde_archivo`, `leer_puntos_desde_teclado`.
- **Ajuste:** `ajustar_polinomio_minimos_cuadrados` arma $A_{\ell m} = \sum x_i^{\ell+m}$ y $b_\ell = \sum y_i x_i^\ell$, y llama a `gauss_pivoteo_parcial`.
- **Resolvente:** `gauss_pivoteo_parcial` (pivoteo parcial + eliminación + sustitución).
- **Calidad:** `coeficiente_correlacion` calcula S_r, S_t y $r = \sqrt{(S_t - S_r)/S_t}$, e imprime todo con nombres en español.
- **Utilidades:** `mostrar_puntos`, `mostrar_polinomio`, `evaluar_polinomio`, `verificar_puntos_suficientes`.

Qué puede pedir la profe (dónde tocar):

- Cambiar grado p (entrada).
- Mostrar A y b (agregar `printf` al final de `ajustar_polinomio_minimos_cuadrados`).
- Ajustar impresiones (columnas) en `mostrar_puntos`/`mostrar_polinomio`/`coeficiente_correlacion`.
- Mensajes si faltan puntos: `verificar_puntos_suficientes`.

9. Interpolación por splines (lineal y cúbica natural)

9.1. 1) Objetivo y cuándo usarlo

Qué resuelve: Queremos una curva que pase por todos los puntos (x_i, y_i), pero evitando los problemas de un polinomio global de grado alto (que puede oscilar mucho).

Idea: Dividimos el eje x en subintervalos entre nodos consecutivos $[x_k, x_{k+1}]$ y, en cada tramo, usamos un polinomio de bajo grado. A esa unión de "tramos" se la llama *spline* o *trazador*.

Tipos básicos:

- **Spline lineal:** rectas por tramos. Simple, pero no es suave (la derivada "salta" en los nodos).
- **Spline cuadrática:** cuadráticas por tramo. Más suave que la lineal.
- **Spline cúbica (la más usada):** cúbicas por tramo con continuidad de valor, 1ª y 2ª derivada en los nodos. Si además pedimos $S''(x_0) = 0$ y $S''(x_n) = 0$, es **cúbica natural** (muy habitual).

Cuándo conviene: cuando hay muchos datos y queremos una curva suave que interpola (pasa por los puntos) sin oscilaciones fuertes.

9.2. 2) Pasos del método (con variantes si las hay)

Preparación (común)

- Ordenar los puntos por x: $x_0 < x_1 < \cdots < x_n$.
- **Hallar el tramo k:** dado un valor x cualquiera, "hallar el tramo k" significa ver entre qué dos nodos consecutivos cae ese x. Ej.: si $x_1 \le x < x_2$, entonces estás en el tramo $k = 1$ (el intervalo $[x_1, x_2]$).

Spline lineal (recta por tramo)

- En cada tramo $[x_k, x_{k+1}]$ usamos la recta que une los dos puntos.
- **Pendiente entre dos puntos:** $m_k = \dfrac{y_{k+1} - y_k}{x_{k+1} - x_k}$.
- **Función del tramo (la que vamos a evaluar):** $F(x) = y_k + m_k(x - x_k)$, válida solo para $x \in [x_k, x_{k+1}]$.
- **Para evaluar $F(x)$:** (1) hallás el tramo k; (2) usás esa fórmula con ese k.

Spline cúbica natural (el estándar)

- En cada tramo $[x_k, x_{k+1}]$ definimos una función cúbica con la forma: $F(x) = a_k x^3 + b_k x^2 + c_k x + d_k$ (una por cada tramo).
- Para encontrar todos los coeficientes a_k, b_k, c_k, d_k armamos un sistema lineal del tipo $[A][?] = [B]$, donde:
  - $[?]$ contiene todas las incógnitas (los coeficientes que multiplican a las potencias de x en cada tramo),
  - $[A]$ y $[B]$ se completan con ecuaciones que salen de las condiciones que debe cumplir el spline.

Qué significan las condiciones (en palabras) y qué ecuaciones aportan:

- **Pasa por los puntos (valor):** en cada tramo, la función coincide con los datos en los extremos: $F(x_k) = y_k$ y $F(x_{k+1}) = y_{k+1}$. → **2 ecuaciones por tramo** (una en cada extremo).
- **Suavidad en nodos internos:** en los **nodos internos** (son todos los nodos salvo los extremos x_0 y x_n), al "pegar" dos tramos, la pendiente y la curvatura no saltan: F' izquierda $= F'$ derecha, y F'' izquierda $= F''$ derecha. → **2 ecuaciones por nodo interno** (una por F', otra por F'').
- **Borde natural:** en los **bordes** (los extremos x_0 y x_n) imponemos curvatura nula: $F''(x_0) = 0$ y $F''(x_n) = 0$. → **2 ecuaciones** (una en cada extremo).

Con esas ecuaciones llenamos $[A]$ y $[B]$. Al resolver $[A][?] = [B]$ obtenemos los coeficientes de cada tramo. *(No hace falta meter más formulación aquí: lo importante es saber qué ecuaciones van a $[A]$ y $[B]$ para poder resolver las incógnitas).*

9.3. 3) Ejemplo corto "a mano" y cosas a tener en cuenta

Datos: $(0,0)$, $(1,1)$, $(2,0)$, $(3,1)$.

Paso A (lineal, idea rápida)

- Queremos estimar en $x = 1{,}5$. Como $1 \le 1{,}5 < 2$, estamos en el tramo $k = 1$ (entre $(1,1)$ y $(2,0)$).
- **Pendiente:** $m_1 = (0-1)/(2-1) = -1$.
- **Función del tramo:** $F(x) = 1 + (-1)(x-1) = 2 - x$.
- **Evaluación:** $F(1{,}5) = 2 - 1{,}5 = 0{,}5$.

Paso B (cúbica natural, desarrollo numérico breve)

- **B.1. Mallas:** $h_0 = h_1 = h_2 = 1$.
- **B.2. Sistema para segundas derivadas** (nodos internos 1 y 2, natural $M_0 = M_3 = 0$):

$$\begin{bmatrix} 4 & 1 \\ 1 & 4 \end{bmatrix} \begin{bmatrix} M_1 \\ M_2 \end{bmatrix} = \begin{bmatrix} -12 \\ 12 \end{bmatrix} \Rightarrow M_1 = -4,\ M_2 = 4.$$

- **B.3. Evaluación en el tramo $[1,2]$ con la fórmula estándar:**

$$F(x) = \frac{M_1(2-x)^3}{6h_1} + \frac{M_2(x-1)^3}{6h_1} + \left(y_1 - \frac{M_1 h_1^2}{6}\right)\frac{2-x}{h_1} + \left(y_2 - \frac{M_2 h_1^2}{6}\right)\frac{x-1}{h_1}.$$

Con $M_1 = -4,\ M_2 = 4,\ y_1 = 1,\ y_2 = 0,\ h_1 = 1$ y $x = 1{,}5$: $(2-x) = (x-1) = 0{,}5 \Rightarrow$ los términos cúbicos se cancelan y queda:

$$F(1{,}5) = \tfrac{5}{3}\cdot 0{,}5 - \tfrac{2}{3}\cdot 0{,}5 = \tfrac{3}{3}\cdot 0{,}5 = 0{,}5.$$

Conclusión: el spline cúbico natural también da $F(1{,}5) = 0{,}5$, pero la curva completa es **suave** al unir todos los tramos (no saltan pendiente ni curvatura en los nodos).

9.4. 4) Mapa del código en C (cómo está dividido y qué hace cada parte)

Archivo: `Spline_Cubico.c`

Idea general del programa (flujo): cargar puntos → validar que x sea estrictamente creciente → menú:

1. ubicar X_{COPETE} en su tramo,
2. construir y resolver el sistema del spline cúbico natural,
3. evaluar el spline en X_{COPETE}.

4.1 Entradas / utilidades

- `leer_puntos_desde_archivo(...)` y `leer_puntos_desde_teclado(...)`: cargan (x_i, y_i).
- `imprimir_puntos(...)`: muestra la tabla de puntos.
- `x_estrictamente_crecientes(...)`: verifica que $x_0 < x_1 < \cdots < x_n$. Si no, corta (spline necesita nodos ordenados y sin repetidos).
- `buscar_subintervalo(...)`: dado X_{COPETE}, devuelve el índice i tal que $x_i \le X_{\text{COPETE}} \le x_{i+1}$ (sirve para "hallar el tramo k").

4.2 Memoria para el sistema lineal

- `crear_matriz(n)` / `liberar_matriz(...)`: reserva/libera una matriz cuadrada $n \times n$ contigua (útil para Gauss).
- **Tamaño del sistema:** si hay n tramos ($n =$ `cantidad_puntos` -1), el vector de incógnitas tiene $4n$ coeficientes $\Rightarrow [A]$ es $(4n) \times (4n)$; $[b]$ y el vector de coeficientes tienen tamaño $4n$.

4.3 Construcción del sistema $[A][?] = [B]$

- **Función:** `construir_sistema_spline_cubico_natural(cantidad_puntos, x, y, A, b)`
- **Incógnitas (orden por tramo k):** $[a_k, b_k, c_k, d_k]$ apiladas así: $[a_0, b_0, c_0, d_0,\ a_1, b_1, c_1, d_1,\ \ldots,\ a_{n-1}, b_{n-1}, c_{n-}$
- **Estructura de filas (en ese orden):**
  1. **2n — paso por extremos (valor):** $F_k(x_k) = y_k$ y $F_k(x_{k+1}) = y_{k+1}$. (Potencias $x^3, x^2, x, 1$ multiplicando a $[a_k, b_k, c_k, d_k]$.)
  2. **(n1) — continuidad de 1ª derivada:** en cada nodo interno x_{k+1}: $F_k'(x_{k+1}) = F_{k+1}'(x_{k+1})$. (Coeficientes de $3ax^2 + 2bx + c$ y su negativo para el tramo siguiente.)
  3. **(n1) — continuidad de 2ª derivada:** en cada nodo interno x_{k+1}: $F_k''(x_{k+1}) = F_{k+1}''(x_{k+1})$. (Forma compacta $3ax + b$, restando el tramo $k+1$.)
  4. **2 finales — condición natural (bordes):** $F_0''(x_0) = 0$ y $F_{n-1}''(x_n) = 0 \Rightarrow 6a \cdot x + 2b = 0$ en cada borde.

4.4 Resolución del sistema

- `gauss_pivoteo_parcial(n, A, b, x)`: eliminación gaussiana con pivoteo parcial (controla pivotes ≈ 0 con `EPS_PIVOTE`). Si resuelve, en `x` devuelve todas las incógnitas (los coeficientes $[a_k, b_k, c_k, d_k]$ de cada tramo).

4.5 Mostrar coeficientes por tramo

- En el `case 2` del menú, luego de resolver:
  `Tramo k [x[k], x[k+1]]: P_k(x) = a_k x^3 + b_k x^2 + c_k x + d_k`
  (con valores numéricos para a_k, b_k, c_k, d_k).

4.6 Evaluación del spline

- `evaluar_spline(cantidad_puntos, x, coeficientes, X_COPETE)`:
  1. Llama a `buscar_subintervalo(...)` para hallar el **tramo** i.
  2. Extrae (a_i, b_i, c_i, d_i) del vector de coeficientes.
  3. Evalúa con **Horner**: $((ax + b)x + c)x + d$.
  4. Si X_{COPETE} está fuera de $[x_0, x_n]$, devuelve `NaN`.

Qué te puede pedir la profe (y dónde tocar):

- **Imprimir** $[A]$ **y** $[B]$ para auditar: después de `construir_sistema_spline_cubico_natural(...)`.
- **Cambiar condición de borde** (p. ej., *clamped*): reemplazar las 2 últimas filas del bloque (4).
- **Formateo** (tabla/decimales): `printf` del `case 2` y `case 3`.
- **Validaciones extra** (duplicados, NaN): cargadores y `x_estrictamente_crecientes(...)`.